Learning from Ground Penetrating Radar data to identify ancient buried structures

By

Merope Manataki

A thesis submitted to the School of Mineral Resources Engineering and the committee on graduate studies Geotechnology and the Environment in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

Technical University of Crete

Chania, June 2021

To my parents

Anastasia & Konstantinos,

and my grandmother

Aikaterini

"Ancora Imparo"

– Michelangelo

Examination Committee

Antonios Vafidis (Supervisor)

Professor, Technical University of Crete

Apostolos Sarris

Professor, University of Cyprus

Michalis Zervakis

Professor, Technical University of Crete

Giorgos Apostolopoulos

Professor, National Technical University of Athens

Michalis Galetakis

Professor, Technical University of Crete

Panagiotis Partsinevelos

Associate Professor, Technical University of Crete

Nikos Papadopoulos

Principal Researcher, Foundation of Research and Technology Hellas

ABSTRACT

GPR data interpretation from archaeological prospection is a tedious and time-consuming process that requires skills and experience. The interpretation process is prone to mistakes, even by the more experienced users. The subsurface can create non-intuitive patterns, making the identification of the buried targets uncertain, requiring additional information from other methods and technologies. Archaeological remains may be bypassed or mistaken for other types of features. Further, residual noise can easily be mistaken as structural remains when in a stripe form that is quite common when surveying in rough terrains. Hence, a system capable of detecting archaeological remains from GPR data could be employed as a guide to assist their interpretation, saving time and reducing mistakes.

Recent developments of Deep Learning (DL) and, in particular, Convolutional Neural Networks (CNN) have shown impressive results for similar tasks in other scientific domains like computer vision and medical image analysis. When it comes to GPR data, these methods and approaches have not yet been used to the same extent. The studies dealing with the automatic detection of buried antiquities using CNNs are very few, leaving an ample margin for investigation, and this research contributes towards this direction.

In this study, AlexNet architecture is used to train CNN models for classifying GPR C-scans. The latter are 2D images derived from slicing pseudo-3D volumes that can be constructed when collecting data using survey grids. The data used have been collected from 52 archaeological sites located in Greece, Cyprus, and Sicily using a Noggin GPR system equipped with a 250MHz antenna. Data collection was conducted under the framework of research projects of the Laboratory of Geophysical - Satellite Remote Sensing and Archaeo-environment (GeoSat ReSeArch Lab) of the Foundation for Research and Technology Hellas (FORTH). The collected data were processed in MATLAB to export the C-scans. A preprocessing step is followed by applying an overlapping sliding window to crop square subregions of selected C-scans to increase the number of images used for training. Three classes were defined based on dominant features observed in the data: unidentified geophysical anomalies, structures, and noise of stripe form. In total, 18375 examples were selected, 6125 per class. Two datasets were constructed following two different splitting approaches to examine the generalization: a random and non-random one. The CNN implementation and training were performed in Python using the Tensorflow library and Keras API. Two optimizers were tested for each dataset and compared: The Stochastic Gradient Descent (SGD) with momentum and Adam. Tests to improve performance were also made by applying Batch Normalization (BN), Dropout, Image Augmentation, and tuning the learning rate and batch size using the Keras tuner library. Two final models were obtained, one for each dataset approach. The models were evaluated using 100 examples from two archaeological sites that were excluded from the training process.

The results showed that the model obtained from the dataset with the random split performed better on the evaluation set, reaching a classification accuracy of 92% over 85%. However, it was observed that the predictions were lacking robustness on similar images. Hence more data and further improvements are required. Further, SGD with momentum performed better but required BN in all five convolutional layers to achieve learning. Dropout improved the results further, but not drastically. Against the expectations, Image Augmentation was not beneficial in any case. While Adam did not require BN for the models to learn, it performed poorer due to overfitting and showed no improvements when BN and dropout were used. The obtained results and good classification performance showed that this is a very promising direction, and the automatic detection of buried structures is a feasible task.

Περιληψη

Η μέθοδος του *γεωραντάρ* είναι μη καταστροφική και εφαρμόζεται επιτυχώς σε αρχαιολογικές γεωφυσικές διασκοπήσεις για την χαρτογράφηση θαμμένων θεμελίων. Η αρχή λειτουργίας του στηρίζεται στην εκπομπή Η/Μ κυμάτων από κεραία-πομπό τα οποία διαδίδονται στο υπέδαφος με ταχύτητα η οποία επηρεάζεται κυρίως από τις ηλεκτρικές ιδιότητες του μέσου. Όταν εισέλθουν σε μέσο διαφορετικών ηλεκτρικών ιδιοτήτων ένα μέρος ανακλάται προς την επιφάνεια όπου ανιχνεύεται από την κεραία-δέκτη ενώ το υπόλοιπο συνεχίζει τη διάδοση στο νέο μέσο. Στις αρχαιολογικές διασκοπήσεις οι κεραίες πομπός-δέκτης κινούνται ταυτόχρονα έχοντας σταθερή απόσταση μεταξύ τους κατά μήκος μιας γραμμής μελέτης πάνω στην επιφάνεια του εδάφους συλλέγοντας καταγραφές που ονομάζονται *ίχνη* (traces). Κατά αυτό τον τρόπο προκύπτουν τομογραφικές εικόνες του υπεδάφους. Σε αυτού του είδους εικόνες, τα αρχαία θεμέλια συνήθως αποτυπώνονται με πρότυπα τα οποία έχουν την μορφή πολλαπλών υπερβολών και περιθλάσεων που αναφέρονται ως *ανακλάσεις*.

Η ερμηνεία τέτοιων δεδομένων είναι μία ιδιαίτερα χρονοβόρα και απαιτητική διαδικασία, η επιτυχία της οποίας στηρίζεται κυρίως στην εμπειρία. Ο λόγος είναι ότι τα πρότυπα με τα οποία απεικονίζονται στα δεδομένα οι καταγραφές από το υπέδαφος, δεν αποδίδουν ξεκάθαρα τη φύση του ανακλαστήρα που τα προκάλεσε. Έτσι ανακλάσεις θαμμένων αρχαιοτήτων μπορεί είτε να παραβλεφθούν ή να παρερμηνευτούν. Τα δεδομένα γεωραντάρ είναι επίσης ευαίσθητα σε θόρυβο ο οποίος συνήθως δεν μπορεί να απομακρυνθεί κατά την επεξεργασία αφήνοντας κατάλοιπα. Όταν έχει γραμμική μορφή, καταγράφεται με πρότυπα που μοιάζουν αρκετά με αυτά των θαμμένων κτιρίων. Σε τέτοια δεδομένα η αβεβαιότητα της ερμηνείας και η πιθανότητα λάθους είναι υψηλές καθιστώντας αναγκαστική τη λήψη πληροφορία από άλλες μεθόδους. Η ανάπτυξη ενός συστήματος αυτόματης αναγνώρισης προτύπων που αποδίδονται σε αρχαία αρχιτεκτονικά κατάλοιπα θα αποτελούσε ένα ιδιαίτερα χρήσιμο εργαλείο που θα διευκόλυνε την διαδικασία της ερμηνείας και θα βελτίωνε την ακρίβεια της περιορίζοντας τα λάθη. Πάνω σε αυτό το πλαίσιο, η παρούσα διδακτορική διατριβή εξετάζει τα *Συνελικτικά Νευρωνικά Δίκτυα* (ΣΝΔ) ως μέσο προς την υλοποίηση ενός τέτοιου συστήματος.

Τα ΣΝΔ είναι ευρέως γνωστά λόγω των ραγδαίων εξελίξεων που έχουν γνωρίσει τα τελευταία χρόνια σε σχέση με την αυτόματη αναγνώριση προτύπων και σε θέματα Βαθιάς Μαθήσεως. Είναι μία κατηγορία *Τεχνικών Νευρωνικών Δικτύων* (ΤΝΔ) *Πρόσθιας Τροφοδοσίας* (Feedforward) με *πλήρως συνδεδεμένα επίπεδα* (fully connected layers), στα οποία έχει ενσωματωθεί η λειτουργία της *συνέλιξης*. Η τελευταία επιτρέπει δισδιάστατες και τρισδιάστατες εισόδους. Μία ενδεικτική αρχιτεκτονική ΣΝΔ περιλαμβάνει το *επίπεδο εισόδου*, το *συνελικτικό επίπεδο*, το *επίπεδο αποκοπής ReLU* (Rectified Linear Unit), το *συγκεντρωτικό επίπεδο* (pooling layer), τα πλήρως συνδεδεμένα επίπεδα, και το *επίπεδο εξόδου*. Το συνελικτικό επίπεδο, το επίπεδο ReLU, και το επίπεδο συγκέντρωσης είναι υπεύθυνα για την εξαγωγή των σημαντικών *γνωρισμάτων* (features) της εισόδου, ενώ μέσω των πλήρως συνδεδεμένων επιπέδων, που είναι ουσιαστικά ένα ΤΝΔ πρόσθιας τροφοδοσίας, εκτελείται η προσέγγιση του προβλήματος μαθήσεως, όπως λ.χ. *ταξινόμηση* ή *παλινδρόμηση*. Τα βάρη των *συνάψεων* στα συνελικτικά επίπεδα στηρίζονται στα *δεκτικά πεδία* (receptive fields), όπου ο νευρώνας ενός επιπέδου συνδέεται με μία περιοχή νευρώνων του επόμενου επιπέδου.

Η εκπαίδευση των ΣΝΔ γίνεται με τον ίδιο τρόπο όπως στην περίπτωση των ΤΝΔ Πρόσθιας Τροφοδοσίας, χρησιμοποιώντας τον αλγόριθμο *οπισθοδιάδοσης* (backpropagation) για τον υπολογισμό του σφάλματος που προκύπτει από τα βάρη που έχουν αποδοθεί στις συνάψεις των νευρώνων ενός επιπέδου, και έναν αλγόριθμο βελτιστοποίησης της *κλίσης* (gradient). Τα βάρη αναπροσαρμόζονται έτσι ώστε να ελαχιστοποιείται η επιλεγμένη *συνάρτηση κόστους* (cost function). Για την εκπαίδευση χρησιμοποιείται σύνολο δεδομένων που έχει χωριστεί σε ένα *σετ εκπαίδευσης* (training set) και ένα *σετ δοκιμών γενίκευσης* (test set). Ένα σύνηθες πρόβλημα εκπαίδευσης είναι αυτό της *υπερπροσαρμογής* (overfitting) όπου τα βάρη του μοντέλου μαθήσεως έχουν προσαρμοστεί τόσο καλά στα δεδομένα του σετ εκπαιδεύσεως με αποτέλεσμα προβλέψεις με χρήση διαφορετικών δεδομένων να είναι ανακριβείς. Ορισμένες γνωστές τεχνικές που αντιμετωπίζουν το πρόβλημα της υπερπροσαρμογής και βελτιώνουν την γενίκευση είναι αναφορικά η *εφαρμογή μετασχηματισμών για αύξηση των εικόνων* (Image Augmentation), η *παράβλεψη νευρώνων* (Dropout) και η *κανονικοποίηση κατά σύνολα* (Batch Normalization).

Τα ΣΝΔ παρουσιάζουν ένα ευρύ πεδίο εφαρμογών κυρίως σε προβλήματα επιβλεπόμενης μάθησης που χρησιμοποιούν προσεγγίσεις όπως την *ταξινόμησης εικόνας*, την *κατάτμησης εικόνας* και τον *εντοπισμό αντικειμένων*. Έτσι είναι ιδιαιτέρως διαδεδομένα σε θέματα μηχανικής όρασης που σχετίζονται με ανάλυση εικόνας και βίντεο όπως π.χ. ανάλυση ιατρικών εικόνων, αναγνώριση προσώπων, αναγνώριση κειμένων, ανάλυση δορυφορικών εικόνων κ.α.. Η εφαρμογή τους σε δεδομένα γεωραντάρ δεν είναι το ίδιο διαδεδομένη, ενώ οι μελέτες που αφορούν συγκεκριμένα αρχαιολογικά δεδομένα είναι ελάχιστες. Παρόλα αυτά τα αποτελέσματα που παρουσιάζουν είναι αρκετά καλά, ενθαρρύνοντας την περαιτέρω έρευνα.

Στη παρούσα διατριβή χρησιμοποιείται η αρχιτεκτονική ΣΝΔ βαθιάς μαθήσεως *AlexNet* για την ταξινόμηση *οριζόντιων τομών βάθους* (C-scans) της μεθόδου γεωραντάρ. Η συγκεκριμένη αρχιτεκτονική ήταν αυτή που έκανε τα ΣΝΔ ευρέως γνωστά για θέματα ταξινόμησης εικόνας λόγω των πολύ καλών αποτελεσμάτων. Παράλληλα είναι απλή παρέχοντας τα οφέλη μιας βαθιάς αρχιτεκτονικής που αφορούν την αυτόματη αναγνώριση προτύπων. Αποτελείται από πέντε συνελικτικά επίπεδα, τρία συγκεντρωτικά επίπεδα, και τρία πλήρως συνδεδεμένα. Η συνάρτηση ενεργοποίησης που χρησιμοποιείται είναι η ReLU με εξαίρεση το τελευταίο πλήρης συνδεδεμένο επίπεδο στο οποίο χρησιμοποιείται η *Softmax*. Επίσης γίνεται χρήση των τεχνικών γενίκευσης Dropout και μίας τεχνικής κανονικοποίησης που εφαρμόζεται στα βάρη των συνελικτικών επιπέδων που αναφέρεται ως *Κανονικοποίηση Τοπικής Απόκρισης* (Local Response Normalization).

Τα δεδομένα που χρησιμοποιούνται έχουν συλλεχθεί με το σύστημα Noggin και κεραία κεντρικής συχνότητας 250MHz από 52 αναγνωρισμένες αρχαιολογικές θέσεις στην Ελλάδα, Κύπρο και Σικελία. Τα δεδομένα συλλέχθηκαν στα πλαίσια ερευνητικών προγραμμάτων του *Εργαστηρίου Γεωφυσικής-Δορυφορικής Τηλεπισκόπησης και Αρχαιοπεριβάλλοντος του Ιδρύματος Τεχνολογίας και Έρευνας*. Αρχικά πραγματοποιήθηκε η επεξεργασία των δεδομένων σε περιβάλλον MATLAB που αποσκοπούσε την αποθορυβοποίηση των δεδομένων, την ανάδειξη των ανακλάσεων από το υπέδαφος και η εξαγωγή των εικόνων τομών βάθους. Οι τεχνικές και τα φίλτρα που εφαρμόσθηκαν είναι οι: *δειγματοληψία ιχνών* (trace resampling), *διόρθωση μηδενικού χρόνου* (time-zero correction), *διόρθωση Dewow*, *ενίσχυση inverse amplitude decay*, *αφαίρεση μέσου σήματος υποβάθρου* (Average Background Removal), εφαρμογή *ζωνωπερατών φίλτρων* (Bandpass filtering), και ο υπολογισμός του *στιγμιαίου πλάτους μετασχηματισμού Hilbert* (Instantaneous Envelope). Εν συνεχεία δημιουργήθηκαν τρισδιάστατοι όγκοι του υπεδάφους και ακολούθησε η εξαγωγή των οριζόντιων τομών (C-scans).

Σε επόμενο βήμα ακολούθησε ένα στάδιο προετοιμασίας στο οποίο εφαρμόζεται κυλιόμενο παράθυρο αποκοπής με αλληλεπικαλυπτόμενο βήμα με σκοπό την αύξηση του αριθμού των εικόνων που θα χρησιμοποιηθούν στο σετ δεδομένων για την εκπαίδευση των ΣΝΔ. Το μέγεθος του παραθύρου προσαρμόσθηκε ώστε να αντιστοιχεί σε διαστάσεις 10x10m της κάθε τομής, ενώ η επικάλυψη ορίσθηκε στα δύο μέτρα. Τα συγκεκριμένα διαστήματα κρίθηκαν κατάλληλα καθώς επιτρέπουν την επαρκή απεικόνιση των αρχαίων κτιρίων ενώ παράλληλα αυξάνουν σημαντικά των αριθμό των εικόνων που μπορούν να χρησιμοποιηθούν για εκπαίδευση.

Όσο αφορά το σετ δεδομένων, ορίστηκαν τρεις τάξεις βάση των κυρίαρχων *γνωρισμάτων* (feature) που παρατηρήθηκαν στα δεδομένα και είναι: *απροσδιόριστες γεωφυσικές ανωμαλίες*, *κτίρια* και *γραμμικός θόρυβος*. Συνολικά επιλέχθηκαν 18375 παραδείγματα, με 6125 ανά τάξη και ακολούθησε διαμερισμός τους σε σετ εκπαίδευσης και σε αξιολόγησης. Σε αυτό σημείο εξετάζονται δύο προσεγγίσεις, του αυτόματου και μη αυτόματου διαχωρισμού ώστε να εξεταστεί ποια μπορεί να οδηγήσει σε καλύτερη γενίκευση. Στη πρώτη προσέγγιση τα δεδομένα του σετ γενίκευσης προέρχονται εξ' ολοκλήρου από την περιοχή μελέτης της *Ελάτειας* ενώ στη δεύτερη προσέγγιση ο διαχωρισμός είναι τυχαίος από όλο το σύνολο των επιλεγμένων εικόνων για κάθε τάξη.

Τα ΣΝΔ υλοποιήθηκαν και εκπαιδευτήκαν σε Python χρησιμοποιώντας την βιβλιοθήκη *Tensorflow* με το *Keras API*. Για την εκπαίδευση εξετάστηκαν δύο αλγόριθμοι βελτιστοποίησης ο *Stochastic Gradient Descent* (SGD) με χρήση *ροπής* (momentum) και ο *Adam* (Adaptive Moments). Για την βελτίωση των αποτελεσμάτων και απόδοσης εξετάστηκαν οι τεχνικές κανονικοποίησης συνόλου (Batch Normalization), παράλειψης νευρώνα (Dropout), και εφαρμογή μετασχηματισμών αύξησης εικόνων (Image Augmentation). Επιπλέον πραγματοποιήθηκε *συντονισμός* (tuning) των υπερπαραμέτρων *ρυθμού μάθησης* (learning rate) και *μέγεθος συνόλου* (batch size) των δύο αλγόριθμων βελτιστοποίησης που εξετάζονται, με σκοπό την περαιτέρω βελτίωση των αποτελεσμάτων. Ο συντονισμός πραγματοποιήθηκε με την βιβλιοθήκη *Keras Tuner*.

Μέσα από μία σειρά συγκρίσεων και δοκιμών προέκυψαν δύο τελικά μοντέλα, ένα για κάθε την κάθε προσέγγιση διαχωρισμού δεδομένων. Το μοντέλο A προέκυψε από τον μη αυτόματο διαχωρισμό ενώ το μοντέλο B προέκυψε από τον αυτόματο διαχωρισμό. Η γενίκευση των δύο μοντέλων εξετάζεται σε ένα νέο σετ δεδομένων που ονομάστηκε σετ αξιολόγησης (evaluation set). Σε αυτό επιλέχθηκαν 32 παραδείγματα γεωφυσικών ανωμαλιών, 32 θορύβου και 36 αρχαίων κτιρίων από τις αρχαιολογικές θέσεις της *Άλου* Θεσσαλίας και της *Σίσσι* Ηρακλείου που είχαν εξαιρεθεί της διαδικασίας εκπαίδευσης.

Συνοψίζοντας τα αποτελέσματα, καλύτερος αλγόριθμος βελτιστοποίησης αποδείχθηκε ο SGD με απαραίτητη όμως την χρήση κανονικοποίησης κατά σύνολα (Batch Normalization), ενώ η χρήση της παράβλεψη νευρώνων (dropout) βελτίωσε περαιτέρω τα αποτελέσματα. Σε αντίθεση η εφαρμογή μετασχηματισμών για αύξηση των εικόνων (Image Augmentation) είχε αρνητική επίδραση στα αποτελέσματα και κρίθηκε η αναγκαία η περαιτέρω έρευνα ώστε να βρεθούν οι κατάλληλοι μετασχηματισμοί που θα οδηγήσουν σε βελτίωση των αποτελεσμάτων. Όσο αφορά τις δοκιμές στο σετ αξιολόγησης, καλύτερη γενίκευση παρουσιάζει το μοντέλο Β (αυτόματου διαχωρισμού) πετυχαίνοντας ακρίβεια 92% έναντι 85%. Παρόλα αυτά, η ακρίβεια των προβλέψεων δεν ήταν σταθερή καθώς υπήρχαν περιπτώσεις όπου παρόμοιες εικόνες δεν ταξινομήθηκαν καλά. Αυτό υποδηλώνει την ανάγκη αύξησης του αριθμού των εικόνων εκπαίδευσης είτε με τεχνικές μετασχηματισμών, είτε με νέα δεδομένα ή πιθανόν με χρήση γενετικού δικτύου για παραγωγή εικόνων (Generative Adversarial Network). Εν κατακλείδι τα αποτελέσματα ταξινόμησης κρίνονται ιδιαιτέρως καλά, με περιθώρια βελτίωσης. Έτσι σηματοδοτείται μια νέα πορεία έρευνας για την εξέλιξη της διαδικασίας της ερμηνείας των δεδομένων γεωραντάρ.

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my supervisor Professor Antonios Vafidis for his invaluable guidance and encouragement during this Ph.D. program. His immense knowledge and suggestions have been genuinely inspiring, which helped me develop and become a better researcher. Further, I would like to extend my sincere appreciation to Professor Apostolos Sarris for everything he taught me, especially during fieldwork. His support, guidance, and advice were key factors in fulfilling this Ph.D. dissertation. Likewise, I would like to thank Professor Michalis Zervakis for his kindness and for offering his expertise and knowledge on Deep Learning, a subject with which I was unfamiliar. Furthermore, I would like to thank all the members of my examination committee for their participation and their help in completing this thesis.

I would also like to express my sincere gratitude to Dr. Nikos Papadopoulos and the GeoSat Research Lab of IMS - FORTH for hosting me as a Ph.D. scholar and supporting me financially through the POLITEIA research project Action KRIPIS, project No MIS-448300 (2013SE01380035) (http://politeia.ims.forth.gr/en/index.php). Further, the access they granted me to equipment for data collection, as well as additional data, were of great importance to realize and complete this Ph.D. dissertation. I would also like to express my appreciation to Stavros Niarchos Foundation for partially funding this research through the project "Advancing Young Researchers' Human Capital in Cutting Edge Technologies in the Preservation of Cultural Heritage and the Tackling of Societal Challenges – ARCHERS (https://archers.iesl.forth.gr).

Further, I would like to thank my colleagues and the people I met while staying at the GeoSat Research Lab for sharing their knowledge and expertise with me, the great collaboration we had, and the pleasant memories I made during fieldwork. Also, I would like to thank all the colleagues and students who participated and helped in the various geophysical surveys from which the collected data were used in this Ph.D. thesis.

My sincere gratitude goes to my friends and colleagues Dr. Tuna Kalayci and Dr. Ian Moffat, for the excellent collaboration and the time they gave reviewing this manuscript. Their feedback and corrections helped me improve the overall quality of this dissertation.

Special thanks also go to Pantelis Vafidis. His suggestions and clarifications on Machine and Deep learning topics were constructive and pointed out the direction that would work the best for the data used in this thesis.

Further, I would like to thank my dear cousins and friends for their encouragement and constant support. Finally, I would like to express my sincere gratitude to my family, especially my parents, for their continuous and unconditional support, encouragement, and love. They made everything possible.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS & ACRONYMS

GPR	Ground Penetrating Radar
CNN	Convolutional Neural Networks
DL	Deep Learning
ML	Machine Learning
GD	Gradient Descent
SGD	Stochastic Gradient Descent
PLA	Perceptron Learning algorithm
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
FNN	Feedforward Neural Networks
MLP	Multi-Layer Perceptron
BN	Batch Normalization
LRN	Local Response Normalization
FC	Fully Connected

NOTATIONS

Scalars, Vectors & Matrices

$\boldsymbol{a}$	Scalar (real or integer)
$\boldsymbol{a}$	Vector (column vector)
$\boldsymbol{a_i}$	The i-th element of vector $\boldsymbol{a}$
$\boldsymbol{a^T}$	Transpose of vector $\boldsymbol{a}$(row vector)
$\widehat{\boldsymbol{a}}$	Vector of prediction
$\widehat{\boldsymbol{a}_{\imath}}$	The i-th predicted element of vector $\widehat{\boldsymbol{a}}$
$\boldsymbol{A}$	Matrix

Spaces and Numbers

$\mathbb{Z}$	Integer
$\mathbb{H}$	Hypothesis set
$\mathbb{X}$	Input space
$\mathbb{Y}$	Output space or targeted space
$\mathbb{R}$	Real number

Machine and Deep Learning

$\boldsymbol{f}(\boldsymbol{x})$	Function of $\boldsymbol{x}$ where $\boldsymbol{x}$ can be scalar $\boldsymbol{a}$, vector $\boldsymbol{a}$ or matrix $\boldsymbol{A}$
$\boldsymbol{h}(\boldsymbol{w}, \boldsymbol{x})$	Hypothesis of training process and function for predictions
$\boldsymbol{cost}$	Cost function or loss function or error function
$\boldsymbol{\varphi}$	Activation function
$\boldsymbol{E}_{\boldsymbol{train}}$	Training error
$\equiv$	Equivalent
$\boldsymbol{a}^{(\boldsymbol{t})}$	The number of iterations $t = 1,2, \dots$ an algorithm performs on a vector
$\boldsymbol{P}(\boldsymbol{a}\|\boldsymbol{b})$	Probability of a given b
$\boldsymbol{P}(\boldsymbol{y} = \mathbf{1}\|\boldsymbol{x}; \boldsymbol{w})$	Probability of $y = 1$ given $\boldsymbol{x}$ parametrized by $\boldsymbol{w}$
$\boldsymbol{\mathcal{D}}_{\boldsymbol{train}}$	Training dataset
$\boldsymbol{\mathcal{D}}$	Dataset
$\boldsymbol{\varepsilon}$	Cost function or error function or loss function over the predicted or training quantities
$\boldsymbol{\varepsilon}_{\boldsymbol{n}}$	Cost function or error function or loss function for the sample point n
$\boldsymbol{\ell}$	The layer in a neural network where $\ell = 0,1, \dots, L$
$\boldsymbol{z}^{(\boldsymbol{\ell})}$	The activated input signal of layer ℓ that is a vector
$\boldsymbol{x}_{\boldsymbol{i}}^{(\boldsymbol{\ell})}$	The $i - th$ example of a dataset that is an input of layer ℓ

1. INTRODUCTION

This Ph.D. thesis examines the automatic detection of ancient buried structures from GPR data using Convolutional Neural Networks (CNNs). The main motive is the challenge met during GPR data interpretation to discriminate patterns in the complex data derived from archaeological prospection surveys. Considering the rapid advances of CNNs and the notable performance improvements they have brough in applications related to image classification, object detection, and image segmentation, they were chosen to investigate whether they can be used to aid and improve GPR data interpretation. In this introductory chapter, the interpretation challenges that triggered this research are firstly discussed, followed by the aim, scope, and objectives. The research's contribution is also mentioned, along with published material. Last, the structure of the thesis is given.

1.1 Motivation

GPR has become a useful tool to use in geophysical surveys applied for archaeological prospection. The latter is a process that can last for months and can be divided into four equally important phases. It begins with the *geophysical survey design*, moves on to *data collection*, then *data processing*, and last, *data interpretation*:

- *Survey design* is the phase where decisions regarding data collection are taken. This mainly includes choices on which GPR antennas to employ to meet the investigation needs and select the areas to cover with survey grids.
- The *data collection* and fieldwork phase can last from a few days to weeks, depending on the size of the area under investigation. In archaeological surveys, data collection is traditionally conducted using survey grids, as this approach gives most of the information GPR can provide.
- The *data processing* phase may begin alongside data collection and continues after. It may last for months after data collection is completed, depending on the data's processing needs, size, and the research questions placed. This is because GPR records are sensitive to changes in ground conditions like the water content and prone to contain noise derived from different sources, making them site-dependent. Hence, their processing requires trial and error to find an efficient workflow. Data processing aims to reduce noise from the data and to highlight the information related to the subsurface. Further, images are exported to be used for interpretation. A single survey grid can produce up to a few hundred images to examine. The two main representations used in archaeological investigations are tomographic images called *radargrams* or *B-scans* and 2D *depth slices,* also called *C-scans*. The two representations are used in an integrated manner as the former provide stratigraphic information while the latter provides spatial information about the distribution of the reflectors at different depth levels. C-scans can provide the most useful information in an archaeological interpretation and are usually preferred, employing the B-scans only when validation of a feature is required. Reprocessing is usually performed at least once to improve the GPR data and produce the final images used in data interpretation.
- The *data interpretation* phase may begin during data processing and continues after the latter is completed and requires experience and skills. This last phase involves browsing and studying the resulting GPR images from the early stages of data processing to understand the subsurface conditions and identify the reflectors. Data reprocessing is usually performed during this step to aid interpretation. Representative images are then selected to illustrate

the conditions and extract the shapes and other geometrical characteristics of the important features identified. This step is beneficial to be performed using a Geographic Information System (GIS) software like ArcGIS or the opensource QGIS. The selected GPR images can be georeferenced and vectorized using shapefiles. This allows to directly compare and validate the extracted GPR information with the one obtained by other geophysical methods, as well as aerial photos, drone images, and others. Further useful representation in map forms can be produced where the spatial relationship of the various identified features can be studied to reveal the past further.

While many challenges are faced during each phase mentioned above, this thesis is mainly motivated by those met during the data interpretation. As it was mentioned earlier, a valid interpretation requires experience, skills, and a deeper understanding of the GPR data. Very often, the nature of recorded reflections remains unknown while mistakes are easy to make, even by the more experienced users. A few common mistakes observed from experience are: a) oversee a reflection related to the target, b) interpret residual noise as a target, and c) misinterpret the target's true nature due to very similar patterns or personal bias.

These mistakes can be limited to a degree when used a *multi-sensor* approach or a *manifold* approach. In the former, multiple arrays of GPR antennas are used to collect data, which leads to faster data acquisition and denser data collection that makes the subsurface imaging more accurate. However, features not detectable from the GPR method due to their physical properties will remain invisible in the recorded data. On the other hand, the manifold approach uses other geophysical methods over the same area to measure different physical quantities. Although it is a more time-demanding approach, the additional information can be extremely useful as it can be used to validate the GPR interpretation or reveal a buried feature that GPR could not detect. The latter justifies the empirical rule to "*always use at least two geophysical techniques*." However, mistakes are still possible, as the results from other geophysical methods are prone to the same mistakes. Another way is to provide a GPR processing workflow to eliminate noise more efficiently, which requires time and effort while risking removing information related to the targets, which is often inevitable. Another approach is to use both B and C-scans to validate whether a feature under investigation exists in both representations and is not a processing artifact or noise. Still, similar patterns from different targets remain a challenge to discriminate. Also, this approach requires studying and cross-checking information from many images. When data from large-scale surveys are employed, it is practically difficult to apply this for all the under-question records. Hence, it requires much more time and effort to exploit all the available information.

For the abovementioned reasons, automatic feature detection seems appealing. A system capable of detecting and recognizing the recorded antiquities from GPR data could be proven valuable with many useful archaeological research applications. For example, it could be used as a guide in data interpretation, reducing the commonly made mistakes, or as a navigator that finds the images in a survey grid that have buried structures, speeding up their examination process significantly. Further, it could be used to develop a representation of the results that is more straightforward to interpret. When considering the recent remarkable advancements in Deep Learning, but in particular the Convolutional Neural Networks known for the automatic feature extraction capability, training such a system seems possible and worth investigating. Therefore, CNNs performance is examined using GPR data collected from several geophysical campaigns in various archaeological sites settlements.

1.2 Research Aim, Scope & Objectives

This research aims to develop a methodology capable of identifying buried structures from GPR data in an automatic way. Towards this aim, the study's scope includes the implementation and training of CNNs to classify GPR C-scans that exhibit buried structures while examines methods and techniques to improve the results. Under this scope, the following objectives are performed:

Objective 1: GPR data collection from archaeological surveys

Data collection was performed on several archaeological sites during the collaboration with the laboratory of Geophysical – Satellite Remote Sensing and Archaeo-Environment (GeoSat ReSeArch Lab) of the Institute for Mediterranean Studies – Foundation of Research and Technology Hellas (IMS - FORTH). Aside from the data collected, the GeoSat ReSeArch Lab provided additional data to cover the needs of the study.

Objective 2: Data processing to highlight buried structures

The gathered data were processed in MATLAB to extract C-scans for training. Tests were performed to establish a processing workflow that highlights structures by combining standard processing techniques and corrections.

Objective 3: Dataset construction for training CNN

The classification labels are set upon the three dominant features observed in the collected GPR data. These are patterns identified as structures, patterns of striping noise, and patterns of irregular geophysical anomalies related to the subsurface. Hence, the classes of Structure, Noise, and Anomaly were defined. A sliding window approach is used to increase the image number, and representative examples of each class are selected. Eventually, 6125 examples per class were gathered, split into the training set and test set. Two approaches are tested to examine the generalization. One uses test set examples derived solely from the same archaeological site while keeping them from the training set. The other shuffles all the examples and does the test set and training set split randomly.

Objective 4: Investigate and choose a CNN architecture to train

Several well-known CNNs architectures were examined, and AlexNet was selected to begin with this investigation before moving into deeper and more complex architectures. AlexNet was implemented using Python with Tensorflow library and Keras API.

Objective 5: Training process and performance improvements

Several tests and trials are performed during training to find the more beneficial settings to the constructed datasets. Two optimizers are considered and compared: The Stochastic Gradient Descent with momentums and the Adam optimizer. Further, techniques known to improve training performance and increase classification accuracy by limiting overfitting are examined. The ones tested are *Batch Normalization*, *Dropout*, and *Image augmentation*. Additionally, tuning of the learning rate and batch size hyperparameters is performed using the Keras tuner library. These tests are performed for both datasets. The training performance and generalization are evaluated using learning curves created by the Keras library metrics loss and classification accuracy calculated on the training and test set. Confusion matrices are also used on the test set to evaluate the classification performance.

Objective 6: Evaluation of the trained models' generalization using GPR images.

The best models obtained from each dataset are evaluated using GPR images from archaeological sites that were entirely excluded from the training process. Confusion matrices are used to compare the classification performance. The misclassified examples are presented and discussed.

1.3 Statement & Contributions

This thesis shows that CNN with AlexNet architecture can be trained to classify remarkably well patterns from structures, geophysical anomalies, and noise in GPR C-scans. With the lack of similar works in the related literature, this work can be considered a starting base that points to further improvements. The main contributions are:

- Guidelines for the dataset construction. This regards the selection of the examples, splitting them into the test set and training set and using a sliding window to increase the image number.
- Sharing trained weights. The weights of the models that yielded the best performance will be publicly available to be used either for predictions or to train new models with transfer learning. The latter is a recommended direction to exploit that could make training with smaller datasets possible and improve the training performance when larger datasets are available.
- Guidelines for the training process. Excessive tests and comparisons were made using the popular optimizers of *Adam* and *SGD* with momentum and Batch Normalization, Dropout, and Image augmentation techniques. Further, the tuning process is used to study the effect of learning rate and batch size in training performance. The results provide useful insights in designing the training process for similar tasks related to GPR and archaeological surveys.

1.4 PUBLICATIONS

This work is heavily based on the data collected, their processing, and their interpretation that was published in:

- Manataki, M., A. Vafidis, and A. Sarris (2014). "Application of empirical mode decomposition methods to ground penetrating radar data." First Break No.32. vol.8, pp. 67-71
- Manataki, M., Sarris, A., Donati, J.C., Cuenca Garcia, C., Kalayci, T., (2015). GPR: Theory and Practice in Archaeological Prospection, in Best Practices of Geoinformatics Technologies for the Mapping of Archaeolandscapes. Archaeopress Archaeology, pp. 13–24
- Manataki, M., Sarris, A., and Vafidis, A., 2015, July. Combining CEEMD and predictive deconvolution for the suppression of multiple reflections and coherent noise in GPR signals. In 2015 8th International Workshop on Advanced Ground Penetrating Radar (IWAGPR) (pp. 1-4). IEEE.
- Manataki, M., Sarris, A., and Vafidis, A., 2015, October. Employing CEEMD for Improving GPR Images-A Case Study from a Neolithic Settlement in Thessaly, Greece. In 8th Congress of the Balkan Geophysical Society (Vol. 2015, No. 1, pp. 1-5). European Association of Geoscientists & Engineers.
- Manataki, M., Sarris, A., Kalayci, T., Simon, F.-X., Cuenca-Garcia, C., Donati, J.C., Papadopoulos, N., 2015. Studying the Variation of Geophysical Signals of the Architectural Attributes of the Neolithic Tells and Landscape. In the International Conference on Computer Applications in Archaeology, CAA 2015, University of Sienna, Sienna, Italy.

1.5 Thesis overview

The layout of this thesis is as follows:

- In Chapter 2, the literature survey conducted is given and is divided into four parts: a) review on the GPR contribution in the archeological investigation, b) review of the DL algorithms and architectures that perform image classification, object detection, and image segmentation along with the recent advances, c) a general review of ML and DL algorithms applied in GPR along with the state-of-the-art and d) review on the related studies that uses DL techniques with GPR data derived from archaeological prospection.
- Chapter 3 focuses on the application of GPR in archaeological surveys. It includes describing the GPR principle, the data collection, the obtained data description, and their processing. Additionally, three case studies are presented to show its capabilities and limitations better.
- In Chapter 4, the theoretical framework behind CNNs is given. It begins with the concept of learning from data, moves on to the neuron description and feedforward neural networks, and ends with describing essential operations found in a typical CNN architecture. The training process is also described, along with challenges commonly met and techniques developed to overcome them.
- Chapter 5 is a methodological one that summarizes and describes all the steps and actions taken during this thesis to reach the final outcome. Details are given on the methods and tools used, data collection, the data processing applied, the construction of the dataset, the implementation of AlexNet architecture, and the testing process settings that were tested.
- In Chapter 6, the obtained results are presented and discussed.
- Chapter 7 is the concluding chapter, where future work and suggestions are also given.

Additionally, three appendices accompanied this dissertation:

- Appendix A includes additional details on data collection and the MATLAB processing script used as described in Chapter 5.
- Appendix B includes python examples and scripts used for AlexNet implementation and training, as described in Chapter 5.
- Appendix C includes complimentary results to Chapter 6.

2. LITERATURE REVIEW

In this chapter, the related literature to this research is reviewed and discussed. That includes a brief review of studies showing how useful GPR is in detecting ancient buildings and foundations and how it contributes to archaeological prospection. Next, a review of deep learning algorithms and their recent advances is conducted, emphasizing those based on CNNs and are used to perform image classification, object detection, and image segmentation. Further, a review of recent studies related to the automatic interpretation of GPR data using ML and DL algorithms is given. Last, the related work of this study is presented and discussed.

2.1 GPR IN ARCHAEOLOGICAL PROSPECTION

The first attempts to use and study the GPR application in archaeological surveys are reported in the mid-70s, where GPR was considered an experimental technology [1], [2]. Since then, it has become a popular geophysical technique in the archaeological investigation to map the subsurface, and with the latest hardware advances, its popularity is growing. The recorded data quality has improved, having less noise, while data collection is faster, which allows a denser acquisition with improved resolution. This led to the *multi-channel* or *multi-offset systems* that allow three-dimensional data collection that increases mapping details and even faster data collection, as is discussed and shown by Booth et al. [3], by Goodman et al. [4], and by Goodman and Piro [5].

GPR has been proven very efficient to map buried foundations and structural remains with detail in the near-surface when proper conditions are met, like soils of low conductivity and contrast in the buried target's electrical properties with the surrounding medium. Numerous studies in the related literature have proven that fact. A few examples, including recent studies, are the successful mapping of Roman structures [6]–[10], [11, p. 15], [12], ancient Greek structures [13]–[20], and houses in settlements of Viking age[11], [21], Iron age[22], Bronze age [23], [24], as well as Neolithic[25]. Aside from houses, GPR has been used to map and investigate graves, tombs and tumuli [11], [26]–[30], or more complex structures like roman baths [31], cisterns [32], [33], and ancient theaters[34]–[36]. It is also possible to detect pit-dwellings, surface-dwellings, farm plots, hedges, and paths, as reported by Tohge et al., for the case study of Komochi-mura village in Japan [37]. GPR usage has been expanded in the latest year to study the archaeological landscapes [38]. GPR can also be used to detect buried antiquities in modern urban environments. An example is given by Papadopoulos et al. in [39]. These are among the reasons GPR has become among the most popular geophysical techniques employed in integrated geophysical surveys designed for archaeological explorations.

GPR can contribute to archaeological research in many ways, especially when combined with other methods and technologies. These can be other geophysical methods (i.e., Magnetics, Electromagnetic Induction, Electrical tomography, Electrical resistivity), satellite and aerial images, drones, soil sampling, Geographical Integrated Systems (GIS), and Global Position System (GPS). The advantages of this approach are discussed in [40] by Moffat et al. A few representative examples include the case studies of Vészto-Mágor tell in Hungary[41], the Iron age settlement at Vesterager in Denmark [22], and several others in Turkey presented and discussed in [42]. The GPR data can fill with information regarding the subsurface to help validate the interpretation of information derived from other types of data and vice versa. When used with GIS and GPS, spatial relations can be studied, assisting in better understanding the past. The main contributions are the high-resolution data when compared with other geophysical techniques, including depth information that can help navigate and plan the archaeological excavations. Through advances in processing algorithms, 3D models of the subsurface can be created, assisting in the detection of the buried antiquities by producing more accurate representations. Examples are given in [8], [12].

2.2 Convolutional Neural Networks

Deep Learning (DL) is a type of Machine Learning (ML) that has met rapid development over the past decade due to faster computing innovations and the increment in the available digital data. This development, which is currently ongoing, includes algorithms, methods, and techniques to learn from the data. Goodfellow et al., in their book [43], showed that a common characteristic of DL algorithms is that they follow the *representation learning* [44] approach where the important piece of information for the learning task, called *features*, is learned in an automatic way directly from the data. The latter is the distinguishing characteristic over the ML algorithms, where features are hand-designed. DL algorithms are mostly applied in computer vision tasks such as image classification, object detection, image segmentation, image retrieval. For this reason, they are applicable in any domain that makes use of these tasks. A few examples are medical image analysis, natural language processing, speech recognition, and text recognition. Several studies exist reviewing the DL algorithms applied to each domain. DL algorithms reported in the review articles [45], [46] for computer vision tasks are the *Convolutional Neural Networks* (CNN), *Recurrent Neural Networks* (RNN), *Deep Boltzmann Machines* (DBMs), *Deep Belief Networks* (DBNs), *Stacked Denoising Autoencoders* (SDAE), and *Extreme Learning*. Among them, CNNs were developed to be the most prominent, especially for supervised learning tasks like classification, and soon expanded for any computer vision task dealing with detection and recognition [47]. Among the reasons they stood out compared to other DL algorithms, as LeCun et al. describes in [41], is the provision of automatic feature detection that is performed in a hierarchical manner. Also, they are easy to train using backpropagation and generalize better. Further, they are easily implemented due to the available open-source libraries like Tensorflow, Theano and PyTorch, and the Keras API, which offer GPU computations' advantages. Other are the availability of data along with hardware improvements that allow the implementation of deeper architectures[48].

Despite the concept proposed during the 80s under *Neocognitron* [49], it became popular in 2012 with the first Deep Convolutional Neural Network (DCNN) architecture named *AlexNet* proposed by

Krizhevsky et al. [50]. AlexNet architecture was based on *LeNet-5*, a CNN architecture proposed in 1998 by LeCun et al. for classifying handwritten digits [51]. Besides the structural innovations of a deeper architecture, AlexNet combined a few key research advances that were found to deal with drawbacks observed in CNN performance like *vanishing gradients* and *overfitting*. These were the usage of Rectified Linear Unit (*ReLU*) activation functions instead of tanh, weight initialization, the usage of a newly proposed generalization technique called dropout, image augmentation, and training using *Stochastic Gradient Descent* (SGD) with momentum. Further, it was the first architecture that was trained using a parallel GPU arrangement. The developments and research trends that followed are shown in the comprehensive review on CNN developments for image classification conducted by Rawat and Wang's in [52]. Briefly, these include various advances in architecture designs (i.e., increments of layers' number, the concept of Network in a Network, Inception module, different types of pooling), various modifications and improvements of ReLU activations and cost functions activation, suggestions for improvements of the regularization mechanisms (i.e., *DropConnect*), and various optimization techniques of the training process (i.e., different types of weight initialization like transfer learning, and batch normalization). More details on the developments on the CNN architectures for image classification are given in the recent survey conducted by Khan et al. in [48]. The authors classify the existing CNN architectures proposed over the period 1998-2018, according to the innovations made into seven categories: *Spatial Exploitation*, *Depth*, *Multi-Path*, *Width based Multi-Connection*, *Feature-Map Exploitation*, *Channel Boosting*, and *Attention*. Further they discuss the strengths and weaknesses of each architecture listed. Reportedly, most of the innovations have been made in relation to depth and spatial exploitation that has improved the CNNs learning capacity. The former includes the architecture of LeNet, AlexNet, ZfNet, VGG and GoogleNet, while the latter has the architecture of Highway Nets, ResNet, Inception-V3 and V4, and Inception-ResNet. According to their findings, the latest advances on CNN architectures replace the conventional layers with blocks acting as auxiliary learners improving further the performance.

Despite the excellent performance reported, there are many drawbacks and open challenges reported by many authors in the related literature. Some of them are discussed in [52] and [48]. Among them are:

- the lack of theoretical background that makes CNN acting like a black box difficult to interpret
- the computational cost remains high, and training demands powerful hardware
- a lack of robustness in classification accuracy is observed when adversarial examples are present

- the need for an unsupervised approach as supervised learning and the requirement of annotation can be restricted for large datasets
- the selection of hyperparameters highly affects the performance, and optimizing them requires much time, skill, and effort due to the large number and the inevitable trial and error.
- the trained models are exhibiting dataset bias since the training examples were gathered for a particular task [53].

The CNN architectures and developments mentioned above were made for image classification. Due to the significant performance improvements the CNN brought, various modifications were made on popular CNN architectures, like AlexNet, VGG, and Inception, to perform *object detection* and *image segmentation* tasks. A recent survey on object detection CNN-based architecture is conducted by Sultana et al. in [54]. The various algorithms used are divided into the one-stage approach and the two-stage approach. Some popular examples of the one-stage approach are the *Region-based CNN* (R-CNN), *Fast R-CNN*, *Faster R-CNN*, and *Mask R-CNN*. The two-stage approach includes the *You Only Look Once* (YOLO), *YOLO9000*, the *RetinaNet*, the *Single Shot Multibox Detector* (SSD), and *RefineNet*. Reportedly, the one-stage approaches perform better but are computationally more demanding. Image segmentation is divided into *semantic* and *instance*. A review of popular CNN-based approaches proposed and used for image segmentation is given by Garcia-Garcia et al. in [55]. For semantic segmentation, the Fully Convolutional Network proposed by Long et al. in [56] is a popular approach that modifies classification-intended architecture to produce spatial heat maps. This is achieved by replacing the hidden layers with convolutional ones and introducing a deconvolutional layer for upsampling. A popular architecture built with FCN and used in medical image analysis is the *U-Net* [57]. Other approaches are the *encoder-decoders*, with SegNet [58] architecture being one example, and the *Conditional Random Fields* (CRF) with *DeepLab* being an example [59]. On the other hand, instance segmentation is more challenging as it also includes the object detection problem. Hence modifications are proposed on architectures used for object detection tasks like R-CNN, Mask CNN, and faster CNN, with the *Simultaneous Detection and Segmentation* method (SDS) [60] and *DeepMask* [61] model being two examples.

Several future directions have been proposed as well in [47], [48], [52]. One of them is the combination of CNN, RNN, and *reinforcement learning* to achieve a deeper level of image understanding, while another is the use of ensemble learning that combines multiple and diverse architectures to improve generalization and improve accuracy and robustness.

2.3 DEEP & MACHINE LEARNING APPLICATIONS IN GPR

The automatic interpretation of GPR data is among the open research topics in the GPR community. The motive is the non-intuitive nature of the GPR data with complex patterns, which makes their interpretation difficult and prone to mistakes regardless of the field of application, i.e., archaeological prospection, civil engineering, etc. The first attempts are reported in the late 90s and aimed to detect the hyperbolas in the B-scans using pattern recognition and image analysis trends at the given time. A couple of examples are Capineri et al.'s work [62] based on Hough transform, and the work by Al-Nuaimy et al. in [63] that took a step further and combined neural networks and Hough transform to detect and classify hyperbolas associated with buried targets.

The research interest was grown as ML algorithms and techniques were advanced, which led to their use to approach more complex GPR problems than hyperbola detection from B-scans. These included attempts to identifying the material of the buried object using synthetic data [64], [65], as well as more application-driven tasks like to assess the condition of railway-ballast [66]. These approaches have as common the use of a pattern recognition technique to extract features and then use SVM to classify them. Other attempts used ML algorithms performing object detection like the Viola-Jones used in [67] to detect and fit hyperbolas in real-time from B-scans. At the same time, Artificial Neural Networks (ANNs) are also becoming a popular method to approach GPR problems related to automatic interpretation. Travassos et al. conducted a review in [68] on various approaches based on ML algorithms and ANNs combined with their GPR data application. Most of the studies mentioned use B-scans for rebar detection and landmine detection, and the problem has expanded from detecting hyperbolas to detect and localize targets. Regardless of the application, Travassos et al. identified three typical stages, with the first one being a preprocessing stage. An image segmentation stage was performed with ANN to classify areas of interest with hyperbolas, followed by a diagnosis stage that uses ML techniques to identify patterns. The latter involves SVM, Hidden Markov Models (HMM), Decision Trees (DT), and K-Nearest Neighbor, to name a few. All the above-mentioned attempts mention positive and promising results; however, several issues are reported. One of the challenges commonly addressed in the previously mentioned studies is the limited GPR data availability that prevents achieving a good generalization on a larger scale. Another is the complexity of extracting representative and effective features, especially when used in real-world data. Last is the need to expand these approaches on 3D data since GPR systems have developed to collect them and are used in several applications.

Several GPR studies are exploiting ML algorithms. In a recent one conducted by Giannakis et al. [69], a framework has been developed to model GPR data acting as a solver for the EM forward problem. Their work shows an alternative to the computationally demanding approach of the Finite-Difference Time-Domain (FDTD) used by gprMax software [70], [71]. The same authors have also proposed a new automated data interpretation scheme based on the regression task that identifies the diameter and depth of reinforcement rebars by receiving a single A-scan as input [72]. Despite training used entirely synthetic data, it generalized well on tests with real data. Another application is the automatic rebar detection and deterioration of the bridge deck concrete made by Asadi et al., [73], where they combine *HOG feature extraction*, *Adaptive Boosting* algorithm (Adaboost), and *adaptive polynomial filters* using real GPR B-scans for both training and testing, which returned very good results. The authors have made the dataset they used for training, called DECKGPRHv1.0, publicly available. HOG feature extraction is also used by Skartados et al. [74] in a proposed methodology to detect hyperbolas from real GPR data. They combined it with a preprocessing step that performs segmentation on the A-scans and SVM to classify the extracted features.

However, most of the latest studies related to automatic interpretation focus on DL algorithms and CNN that perform image classification, object detection, and segmentation tasks. This is an outcome of CNN's automatic feature extraction ability from raw inputs and the development and improvement of open-source DL libraries that make implementation easy while elaborating GPU's computation advantages without any further effort. Hence problems approached with ML and pattern recognition are re-approached from a DL perspective using mainly CNN-based algorithms. While some studies focused on hyperbolas detection from GPR B-scans using either image classification or object detection DL algorithms, others are more application-centered, with the majority being related to Civil Engineering problems and landmine detection. However, their small number up to date shows that DL research on GPR data is still new and in an early and experimental stage, with growing interest. A summary of the studies that employ DL using GPR data is given in the following paragraphs.

Regarding the hyperbolas detection studies, the implemented CNNs are either trained using synthetic B-scan images and validated with real B-scan images [75] or trained with real B-scan training datasets that were built from scratch [76], [77]. A practical guide for designing, training, and optimizing CNNs for classifying B-scans is given by Reichmann et al. in [77], where they mention several methods and techniques known to improve CNN performance. Among the mentioned ones are *weight initialization*, *regularization techniques* (L2 and dropout), *data augmentation*, and the use of *pre-trained networks*. The authors of [77] also compare the performance of three different CNN configurations with the HOG feature and *Random Forest* classifier, reporting that all three CNNs performed better. Pham and Lefevre performed a similar comparison in [78] between the performances of two object detection algorithms, the Faster R-CNN and the *Cascade Object Detection* (COD), based on HOG and Haar features. They reported that R-CNN outperformed COD in detecting hyperbolas from B-scans. In a different study, Ozkaya et al.[79] introduce the *Convolutional Support Vector Machine* (CSVM) network to detect hyperbolas in Bscans and classify the reflectors' shape and material and the soil type. Their proposed approach was trained and tested on simulated data, and the performance was compared with the popular CNN architectures of AlexNet, VGG16, GoogleNet, ResNet, and SqueezeNet, returning better results.

DL algorithms are applied and studied in several Civil Engineering applications that use GPR data and aim to detect and interpret the features of interest automatically. A few examples are using object detection with the SSD algorithm to detect rebars in concrete [80], detecting pavement distress using R-CNN, and detecting internal defects in tunnel lining through segmentation [81]. The latter study's authors, Yang et al., propose a new segmentation scheme, named *defect segmentation*, and compare its performance with SegNet and U-net. Additionally, a series of studies are conducted on different DL approaches and their ability to detect cavities, pipes, manholes, and subsoils in the urban streets in South Korea. Kim et al. [82] use AlexNet with B-scans to classify hyperbolas patterns attributed to the targets mentioned before, while Kim et al. [83] and Kang et al. [84] use both C-scans and B-scans to train AlexNet for the same purposes. Further, CNN architectures are proposed that are trained using tri-planar GPR data and 3D voxel data by Kim et al. [85] and Khudoyarov et al. [86], respectively. Last, a survey on the advances of deep learning application in GPR and Civil Engineering has been conducted by Tong et al. [87]. They review and compare the various methods used in the recent literature. They conclude that the approaches where A-scan were used as training data tend performed better.

As for the recent studies focusing on landmine automatic detection, they follow a similar concept, using an object detection approach with faster R-CNN [88], an image classification approach using CNN with a LeNet inspired architecture [89], and an *Autoencoder* approach to analyze volumetric data [90]. Other notable applications involve water content classification using A-scans with a regression approach using 1-D CNN [91], and DL approaches to the inversion problems. In the latter, Alvarez and Kodagoda in [92] propose a framework to transform GPR B-scans into permittivity maps that are easier to interpret. The approach they follow solves the migration inversion problem using semantic segmentation architectures that are trained with synthetic data. The architectures they explore are the U-Net, Encoder-Decoder, and *cGan*. In a similar study, Liu et al. introduce *GPRInvNet*

[93], an architecture based on Encoder-Decoder and was used to reconstruct permittivity maps of tunnel lining from GPR B-scans. Last, Rice et al. [94] explore the *Generative Adversarial Networks* (GANs) to produce realistic GPR B-scans.

The studies mentioned above, which are among the first attempts of using DL with GPR data, show the latest trends in the literature exploring automatic interpretation, and all of them report promising and good results. This shows that DL can work well with GPR data. However, this is judged on small-scale experiments based on the dataset used in each study and GPR application. To test each approach's generalization, it requires larger-scale experiments, and hence, bigger datasets that are currently lacking. Besides, DL and automatic feature detection may still not be the best approach for all GPR problems related to automatic interpretation. This is mentioned in a recent study conducted by Malof et al. in [95] to detect buried threats. They find that traditional classification ML approaches with handcrafted feature extraction still perform better. Another drawback that was mentioned in several studies is the incapability to discriminate similar hyperbolic patterns in B-scan derived from the different buried targets. Some recommended solutions to this, either than sharing data, are to use transfer learning and GANs to produce realistic GPR images and increase the training number.

2.4 Related work

Deep learning applications in GPR data derived from archaeological prospection are currently unexplored. The recent study conducted by Küçükdemirci and Sarris in [96] is the closest and only one found towards the direction that this Ph.D. research is heading. Küçükdemirci and Sarris applied semantic segmentation using U-Net to identify buried structures in C-scans. Their results were particularly good, which encourages further research and points a future direction in the analysis of GPR data derived from archaeological investigations. Hopefully, the current study will add to this direction by exploring image classification using AlexNet to classify C-scans that contain buried structures. Similar to the guide of [77], an emphasis is given to the training process, as well as applying techniques known to improve the performance. Additionally, different approaches to construct the training dataset are examined, aiming for a better training process and generalization.

3. GPR FOR MAPPING ANCIENT STRUCTURES

Ground Penetrating Radar is a near-surface geophysical method proven to be an appropriate tool in archaeological prospection. There are many studies in literature where authors manage to map buried antiquities like roads, paths, public and residential buildings, graves, etc. The operation principle of GPR lies in the interaction of E.M. energy with the matter. An antenna is transmitting and receiving E.M. pulses that are injected into the ground. The method is strongly affected by the medium's electrical properties, especially the conductivity, and the permittivity, which affect the transmitting signal’s penetration and velocity. Besides the electrical properties, the penetration depth is affected by the central frequency, which is decreasing as the frequency increases. The acquired data are called radargramms, and under certain processing, depth slices of the subsurface can be extracted. These are actual images of the subsurface showing areas with high contrast in electrical properties. When the conditions are ideal (low conductivity environments, flat surface, lack of vegetation), GPR can provide highly detailed and accurate results. Otherwise, it can still perform well as a complementary method if proper processing is done.

This chapter's material has been published in Manataki, M., Sarris, A., Donati, J.C., Cuenca Garcia, C., Kalayci, T., (2015). GPR: Theory and Practice in Archaeological Prospection, in Best Practices of Geoinformatics Technologies for the Mapping of Archaeolandscapes. Archaeopress Archaeology, pp. 13–24

3.1 GPR OVERVIEW AND APPLICATIONS

Ground Penetrating Radar (GPR) is a non-destructive electromagnetic (EM) geophysical technique that uses radio waves, in the frequency range of 10MHz to 2GHz, to map the subsurface. The first reported attempt of using radio wave signals to measure subsurface features was by El-said in 1956 [97], who tried to map the depth of a water table. The development of the method accelerated considerably after 1970 due to the tremendous progress in electronics and computer technology. However, it was not until after 1985 where the major advancement of GPR occurred [98]. During this period, GPR technology draws attention worldwide, as it has become more affordable, expanding its applications. Consequently, GPR's strengths and weaknesses were better examined and understood, which opened new ways into hardware development and improvements. Recent hardware advancements include multichannel systems that have greatly improved the speed, area coverage, and spatial resolution, as discussed by Goodman and Piro in [5].

GPR can be used in several geosciences' applications like to map the bedrock's depth [99], to determine the stratum thickness and the aquifer depth [100], to locate physical and artificial cavities in the subsurface [101], as well as fracture zones [102], [103]. It is widely used in archaeological prospection capable of detecting a variety of archaeological features, as was discussed in Chapter 2. In the following sections, GPR's operational principle is given along with all the important parameters that affect its performance in archaeological surveys. Additionally, a few practical data processing steps are briefly described using real-world data. Finally, three case studies are presented, and the capabilities of the method are discussed.

3.2 GPR OPERATIONAL PRINCIPLE

GPR is an electromagnetic technique (EM), and its operation principle has similarities with the seismic reflection method. A typical GPR system consists of the antenna, the timing unit, and a portable computer (*Figure 3.1a*). The antenna is responsible for emitting and detecting EM energy (10~2000MHZ) through a transmitter (Tx) and a receiver (Rx). The timing unit is an essential part since it controls the radar signal generation and converts the received signals as a function of time. The processing unit is used for storing the data and displays them in real-time.

The operation principle of GPR is simple. The transmitter emits high-frequency pulses of short duration into the ground that "travel" through the subsurface until they meet a different material boundary. At this point, part of the energy is reflected towards the surface and recorded by the receiver antenna (black arrows in *Figure 3.1b*), while the remaining energy is diffused deeper (red arrow in *Figure 3.1b*) until it hits another boundary, where it will be reflected and diffused again. This process reaches an end when all the energy is absorbed by the ground. This boundary or reflector is defined by differences in subsoil materials' electrical properties, such as *conductivity* and *permittivity*. Both affect the EM waves' propagation and are of significant importance. Conductivity affects the energy absorption, thus the signal penetration and the depth of the investigation, while permittivity affects the wave velocity. In general, GPR is most useful in low-electrical-loss materials (i.e., very low conductivity values). Clay-rich environments or areas of saline water will negatively affect the method's effectiveness, as explained by Cassidy in[104].

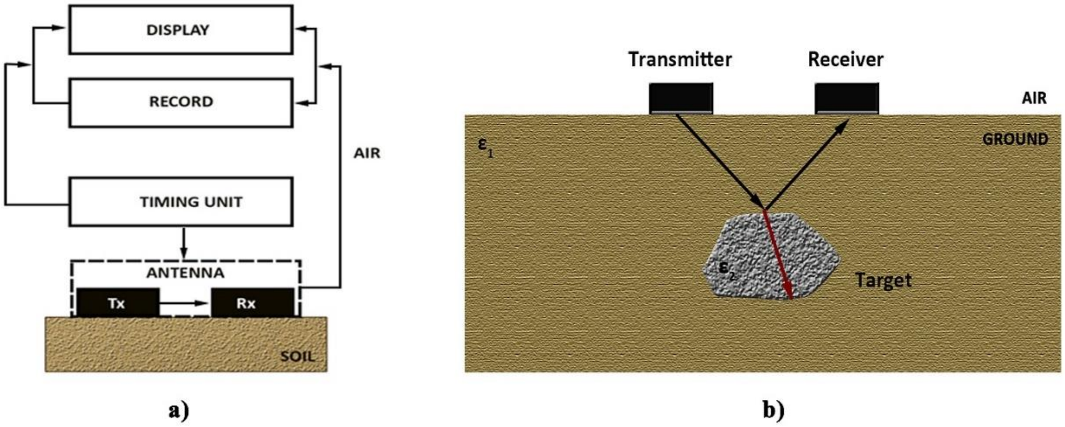


Figure 3.1: Illustration of GPR operation principle. In a) the system components are described, while in b) the behavior of the EM waves when they meet a boundary with different electrical properties from the soil (ε1>ε2). Part of the energy is reflected towards the surface, and another one (red arrow) is diffused at deeper levels (concept re-created from Daniels in [105]).

Unlike magnetic or electrical methods, GPR does not directly measure the properties of the ground. The receiver records the amplitude of the reflected pulses with respect to their *travel time*. This time series is called *trace* or *A-scan*. The travel time, also known as *double travel time* or *two-way time*, is the time that a signal needs to make the route transmitter-reflector-receiver and depends on the propagation velocity of the EM pulses. When an EM pulse leaves the transmitter, its energy is spread at different paths that are illustrated in *Figure 3.2*. In this example, the subsoil consists of two homogeneous layers of different electrical properties. The first records are the direct airwaves and ground waves since EM waves reach their maximum velocity when traveling in the air (0.3x10^9 m/s). Additionally, they exhibit the highest amplitude values since the energy loss during this path is minimum. Critically refracted and reflected waves exhibit slower velocities that depend on the medium's electrical properties, and their recordings appeared later than the direct waves. If the velocity is known, then the depth of the reflector can be determined.

To better explain the GPR records or traces, an earthly homogeneous layer (without contrast in electrical properties) is used as an example. Suppose a trace is recorded at a fixed position on that layer's surface. In that case, it will bear information only from the direct waves with no other reflections due to the homogeneity in electrical properties. Suppose a second homogeneous layer of different electrical properties exists below. In that case, the resulted trace will exhibit amplitudes from the direct waves, plus amplitudes corresponding to waves that were reflected at the boundary between the two layers (like in *Figure 3.2*), with no further information. For a non-homogeneous case scenario, the recorded traces are much more complicated, bearing multiple reflections. Hence, GPR A-scans can reveal stratigraphic information.

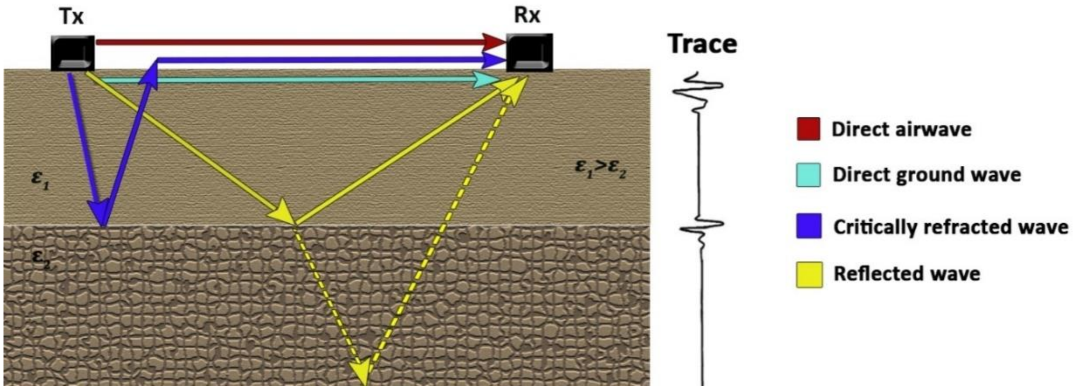


Figure 3.2: Illustration of the paths that an EM pulse can follow between the transmitter and the receiver. The direct airwaves and ground waves are the earliest records and are located at the top of the trace, while objects and regions in the subsurface called reflectors cause the refracted and reflected waves. In this illustrated case, the reflector is the boundary between the two layers. (Concept is redrawn from Annan in [106]).

3.3 Data collection & survey parameters

The most used GPR system in archaeological investigations is the common-offset reflection. In a common offset array, the transmitter and the receiver have a fixed spacing and orientation. The A-scans are collected by moving this fixed offset along the surface along a line called the *survey line* using a regular distance interval. The line’s direction is referred to as the *scan-axis* as well as *Y-axis* or X-axis when following a cartesian reference system. The concept is illustrated in *Figure 3.3a*. Additionally, in the common-offset reflection systems, antennas come with a fixed central frequency.

Frequency is a crucial parameter that affects both the investigation depth and data resolution. The higher the central frequency, the lower the pulse penetration depth, and the higher the resolution in both vertical and horizontal directions. This is due to EM wavelength and pulse width being inversely proportional to the frequency as explained in [104], [106]. Thus, prior to the survey with GPR, one must know the target(s) expected depth and select a frequency that allows reaching that depth. The antenna frequency selection is based mainly on experience taking into account the ground conditions. For example, a range of 200-300MHz can penetrate up to 2-3 m if the ground conditions (i.e., conductivity and permittivity) are appropriate. Less is expected otherwise, mainly due to the EM signals' attenuation.

The data in archaeological surveys are usually collected by employing grids that cover an area under investigation. The grids are set using a local cartesian system, mainly where several lines are collected and are usually directed along the Y-axis. An important parameter is the line spacing, which is the distance between the GPR parallel profiles (denoted as Δx in *Figure 3.3b*) and affects the spatial coverage and how well the target will be resolved in the obtained data. Line spacing is a fixed value and is set according to the antenna's central frequency, and size used, as well as the targets' expected size and geometry. For detecting buried structures and foundations, a line spacing of 0.5m usually suffices when combined when frequencies in the range of 200-300MHz are used. However, lower or smaller sizes are also used depending on the expected target size and the survey needs. The architectural remains can be mapped without losing spatial information, while unnecessary overlapping is avoided. The higher frequency antennas have a smaller size, and hence, smaller spacing is used (i.e., 0.25 for a 500MHz) to cover the same distance without leaving gaps that will lead to spatial information loss.

While GPR transmits signals continuously, the records are discrete signals. Hence, caution should be given to the sampling intervals in time and distance. The spatial sampling interval, Δy, defines how often the traces are recorded along the survey line (denoted as Δy in *Figure 3.3a*). In other words, it is the distance between two consecutive collected A-scans. Thus, it affects the total number of traces that the line will include. A small value will result in a high detail coverage, but caution is needed to avoid oversampling that slows down data collection speed without adding any further information. Usually, this value is selected with respect to the central frequency. The time sampling interval is the time lapse between two records in the same trace (spacing between the black dots denoted as Δt in *Figure 3.3a*), and it affects the resolution on the vertical axis. This value is set according to Nyquist criteria and the antenna's central frequency, as Annan explains it in [106]. In the recent GPR system, the time interval is set automatically. As an example, the Noggin system by Sensors and Software equipped with a 250MHz antenna uses the value Δt=0.4ns.

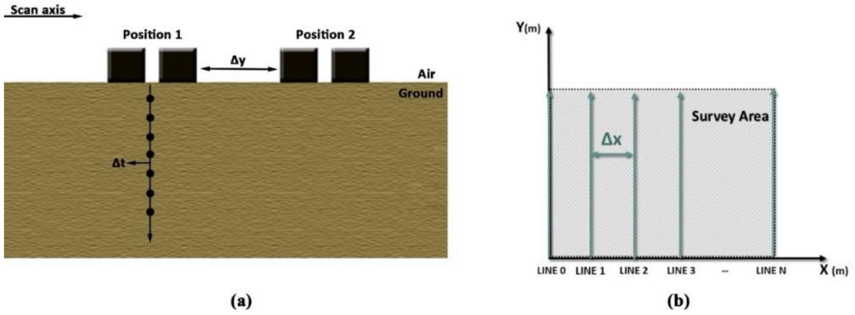


Figure 3.3: Survey parameters of common offset reflection GPR systems. (a) As the antenna is moving along the surface, traces are recorded with a step that is defined by Δy along the scan axis. Additionally, each trace includes a finite number of records (black dots) that are obtained with a step of Δt. (b) The survey grids consist of parallel lines that are separated by a constant distance defined by Δx.

As the antenna moves along the scan axis, traces are collected, forming the image of *Figure 3.4a*. When a colormap is applied to this image, the outcome is called *B-scan*, *radargram*, or *section* and can be viewed in real-time while surveying (*Figure 3.4a*). B-scans reveal stratigraphic information along the survey line. The reflections from targets may appear in hyperbolic forms or as linear reflections depending on the antenna's orientation with respect to the target geometry. Assuming a buried wall, as showing in *Figure 3.5a*, if the survey line is oriented perpendicular to the wall ("point" target), the latter will form a hyperbola on the B-scan. If the scan-axis is oriented parallel to the wall's longest dimension, then a linear anomaly will be formed (*Figure 3.5b*). Also, if the velocity is known, the time axis can be converted to distance, indicating the depth of each reflector. In case the velocity is unknown, it can be estimated from the hyperbolas appearing on the sections. This operation is included in every GPR processing software package and is usually carried out by fitting a curve on a hyperbola in a B-scan.

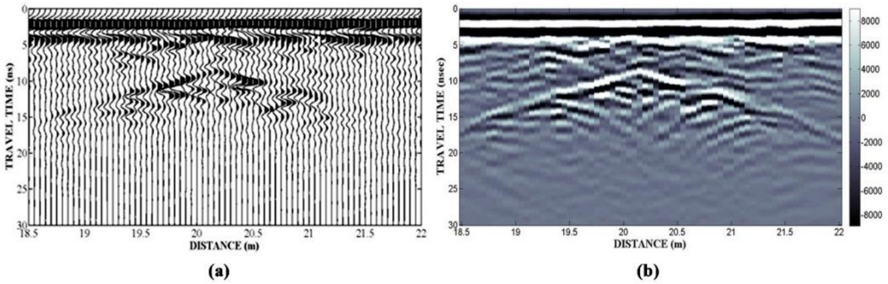


Figure 3.4: Description of a GPR image acquired from a survey line. (a) shows the collected traces plotted together (a), while (b) shows the resulting image after applying a colormap, known as B-scan, radargram, or a section.

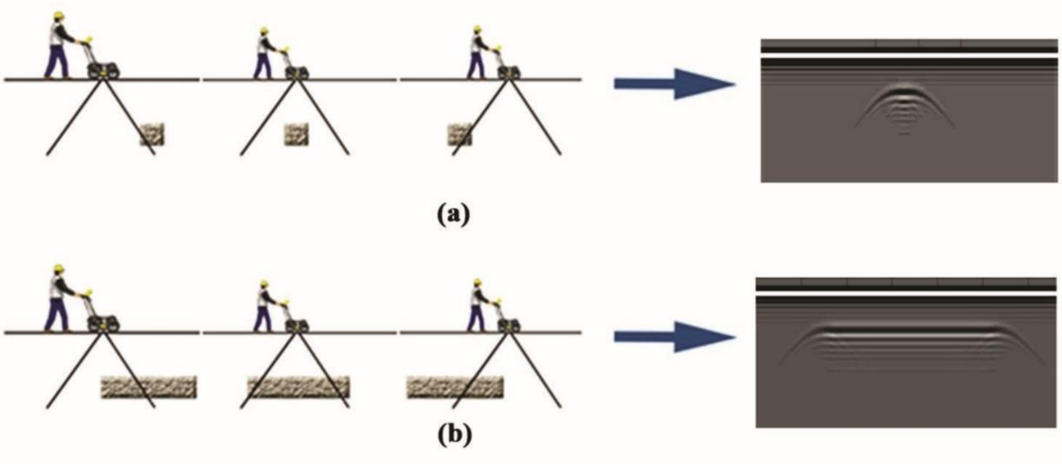


Figure 3.5: Description of hyperbolas and linear anomalies shown in GPR B-scans. (a) Hyperbolas are formed from small targets or walls that the antenna passes perpendicular to their longest dimension. (b) Linear anomalies characterize linear reflectors and are formed when the antenna is moving along its longest dimension. The B-scans presented here are synthetic and were made using the gprMax software [70], [71].

3.4 GPR DATA PROCESSING

Processing is an essential and time-consuming procedure aiming to highlight reflections related to the target(s) while removing unwanted information, i.e., noise. Various types of noise are present in GPR data. The most common are *white* or *random noise* and *coherent noise*. The former usually appears at deeper levels, overshadowing reflections from targets if they exist. Coherent noise can be caused by external sources (cell phones, TV antennas, etc.) or by the EM energy that escapes towards the air and is reflected by objects on the surface (trees, modern buildings, cars, rocks, electrical cables, etc.) back to the transmitter. Coherent noise appears as echoes similar to the ones caused by targets, and caution should be exercised during interpretation.

As described previously, the B-scans are created by moving a transmitter-receiver array along a survey line on the surface and are 2D images of the subsurface. The horizontal axis is the distance vector (m), while the vertical axis is the double travel time (ns) or can be converted to depth (m) if the EM travel velocity is known. When working in grids, 3D images of the subsurface can be constructed from the B-scans where *depth slices* or *C-scans* are extracted. Depth slices are also 2D images where the two axes are the distance vectors X and Y of the survey grid and provide information about the subsurface reflections at a specific time or depth. In other words, they are the resulting images when slicing the 3D volume in time.

Data processing of GPR data can be divided into two major stages: (1) the processing of the radargrams where signal processing techniques are used, and slices are extracted, and (2) the processing of slices where image processing corrections are applied. The emphasis is given to the first stage that aims to filter out the noise from the data and enhance the reflections from the subsurface. Some standard processes that are usually applied regardless of the field of application are[107]:

- Traces reposition corrects the position of GPR traces in a survey line. This correction is helpful to eliminate systematic offsets in survey lines' starting and ending position, which usually occurs in rough terrains.

- Time-zero correction estimates the first pulse's correct vertical position that left the antenna and entered the subsurface. The effect of time zero correction is shown in *Figure 3.6b*, where *Figure 3.6a* is the raw section.

- Dewow filter removes low-frequency noise derived by low-frequency energy near the transmitter and is associated with electrostatic and inductive fields. The dewow filter output is presented in *Figure 3.6c*.

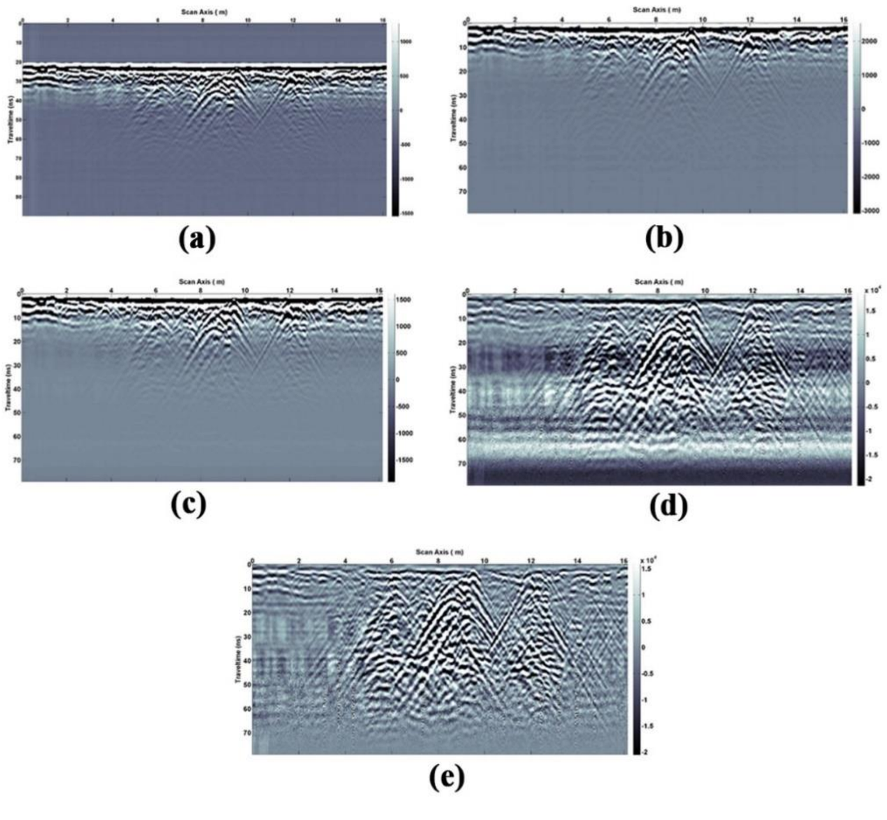
- Gain operation corrects the attenuation effect. There are a few different types used, namely the Automatic Gain Control (AGC), the constant gain, and the Spreading & Exponential Compensation (SEC) gain, and Inverse Amplitude Decay (IAD). However, when applied gain regardless of the type, the existing noise is also highlighted. The effect of gain is presented in *Figure 3.6d,* where IAD has been applied.

- Background subtraction filter reduces the random noise from the data and removes the direct waves and background noise. For archeological prospection GPR data, it was found more efficient to apply it after gain as it removes the noise enhanced by the latter while retaining the wanted information. This is shown in *Figure 3.6e*.

- Frequency domain filtering includes low-pass and high-pass 1D filters that remove high or low-frequency noise, respectively. These filters can be combined to retain frequencies at a specific range and are called *bandpass filters*. The effect of a bandpass filter is presented on the C-scans of *Figure 3.7*. The stripping noise caused by plowing lines is effectively removed by selecting an appropriate frequency range, as shown in *Figure 3.7b,* where barely visible structures are highlighted. However, finding a good frequency range is a challenging and tedious task.

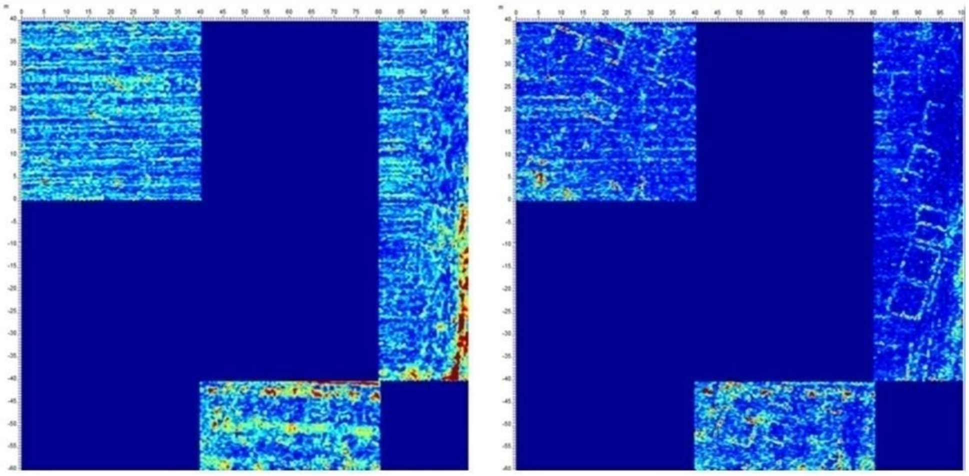
- Migration is an inverse process that removes distortion due to diffraction by reconstructing the reflections to fall at their correct position. When applied successfully, a hyperbola will be reduced to an isolated point target that is more accurate and easier to interpret representation. However, the migration requires velocity information that is usually not available and cannot be accurately estimated from common offset data. For GPR data, the Kirchhoff Migration is usually applied that requires a fixed velocity estimate rather than a velocities model.

The abovementioned filters and corrections are considered basic but are not standard, apart from time-zero correction and dewow, meaning that they may not always improve the B-scans. GPR data are known to be site-dependent, resulting in trial and error to find an effective processing workflow. By the time the processing on the B-scan is complete, Hilbert Transform (HT) is applied to calculate the Instantaneous Amplitude [108] and to extract depth slices (*Figure 3.7*). The slices indicate the changes in the instantaneous amplitude at a certain time, as mentioned earlier. High values designate big changes in electrical properties and high reflectivity.



(a) (b) (c) (d) (e)

Figure 3.6: A processing example of a GPR B-scan. In (a) is the raw image, in (b) the time zero correction is applied, in (c) the outcome of dewow filter is shown, in (d) the gain correction is applied, while in (e) is the outcome of average background noise removal. The GPR profiles were derived from the geophysical survey conducted by the GeoSat ReSeArch lab IMS-FORTH Institute at the monument of Yperia Krini in Thessaly.



(a) (b)

Figure 3.7: Bandpass filtering effect that is shown on a C-scan. In (a) is the C-scan after the basic processing, while (b) is the resulted C-scan of bandpass filtering. Structures that were not visible before are highlighted. The presented depth slices were derived from the geophysical survey conducted at Magoula Almyriotiki, Thessaly, by the GeoSat ResArch lab IMS-FORTH.

3.5 Case Studies examples

3.5.1 Ancient Demetrias

The ancient city of Demetrias is located south of the modern city of Volos in Thessaly. The city was established by the Macedonian military leader and eventual king, Demetrius Poliorcetes (337-283 BCE) in 294 BCE [109]. The city became the royal residence of the Antigonid dynasty of Macedonian kings and flourished as an international and political center. The city permanently fell to the Romans following the battle of Pydna in 168 BCE. The Antigonid dynasty was immediately dissolved, and the Roman province of Macedonia, of which Demetrias was a part, was officially established a few decades later in 146 BCE. During the Roman Imperial period, many central areas of the city, including the Hellenistic palace area, were used for burials. Demetrias experienced a brief recovery beginning in the 4th century CE when the Roman emperor Constantine the Great made the city an episcopal sit[109]. Early Christian churches attest to this change in fortune. The city was finally abandoned during the 6th century CE, and it was never reoccupied again.

The geophysical survey at the ancient Greek city of Demetriada was conducted during March 2014, where two GPR systems were used. The first was a single channel Sensors & Software NOGGIN Plus-Smart Cart system equipped with a 250 MHz shielded antenna frequency (*Figure 3.8a*), and the second was a multi-channel MALÅ Imaging Radar Array (M.I.R.A.) with 400 MHz antennas (*Figure*

3.8b). Both radars were employed to survey the soccer field area located east of the city agora and southeast of the Hellenistic palace. The area was ideal for using GPR due to the flat surface and the lack of vegetation. The total area covered is 120mx60m, where the line spacing was 0.5m. The results for both radars are very detailed and presented in Figure 3.9.

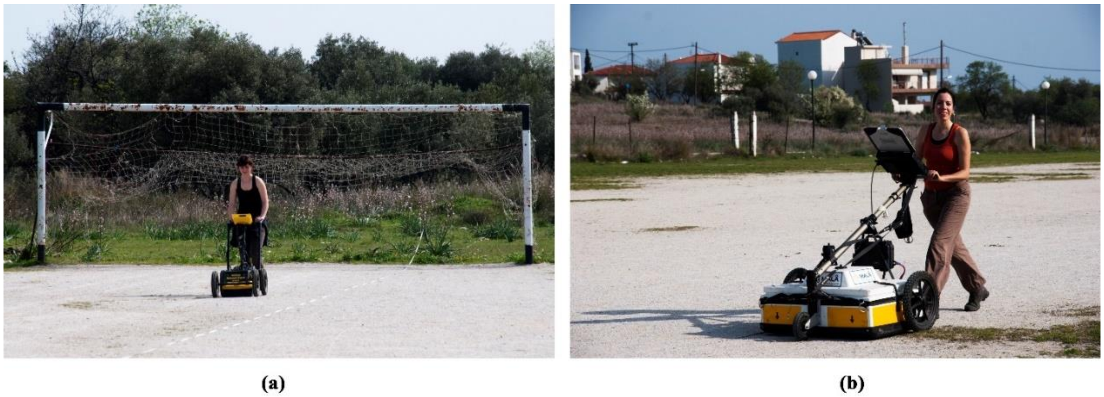


Figure 3.8: GPR survey at Demetria's soccer field. (a) Single-channel Noggin GPR equipped with a 250MHz antenna operated by Merope Manataki and (b) the multichannel Mala Mira GPR equipped with 400MHz antennas operated by Dr. Carmen Cuenca-Garcia.

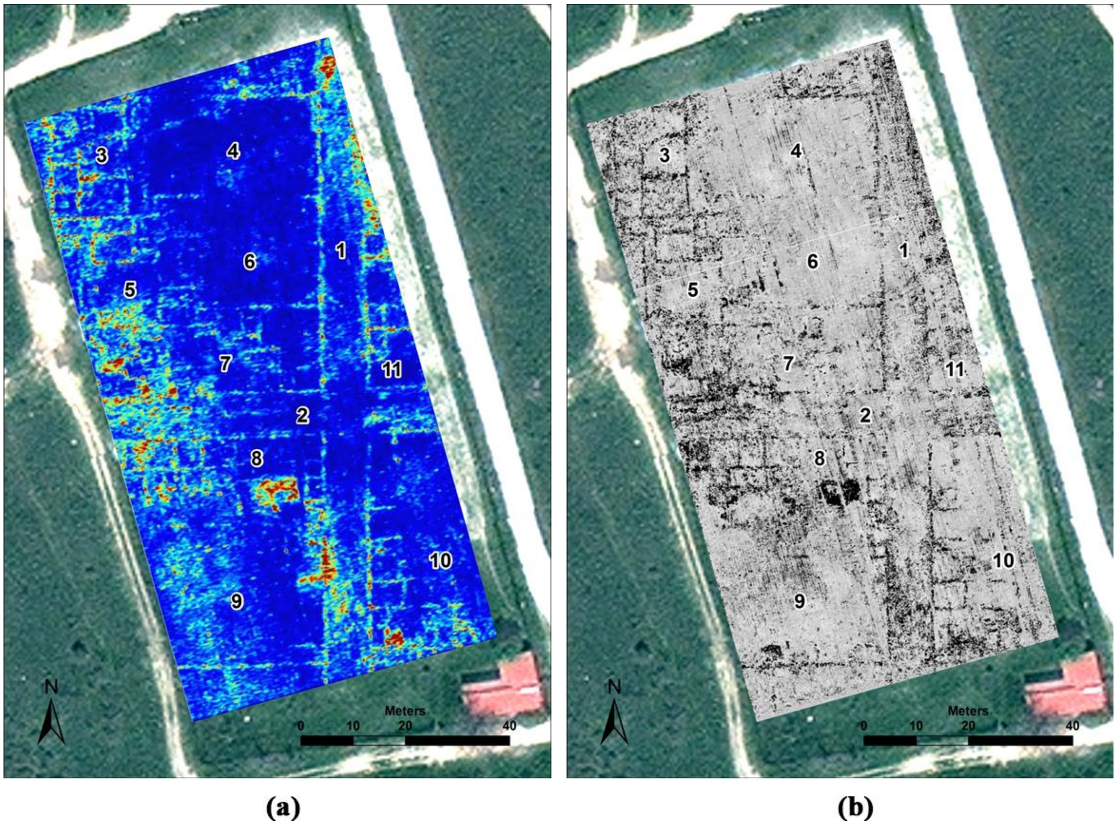


Figure 3.9:GPR results from Demetria's area at the soccer field. (a) The slice at 0.4-0.5m depth derived from Noggin GPR and (b) the slice at 0.47m derived from MALA Mira.

The architectural features' overall arrangement beneath the soccer field recalls Hellenistic and Roman urban houses with courtyards or gardens in the back and shared partition walls between houses [110], [111]. Two main roads are clearly distinguished (Features 1 and 2 in *Figure 3.9*), with a dense collection of buildings in the rectilinear city blocks in between. Features 3 and 4 seem to be a single structure with at least seven rooms located at the west and a large free zone at the east with few walls that functioned as a backyard courtyard or garden. A similar arrangement is noted with Features 5 and 6. Features 7 reveals another collection of rooms that take up the city block's whole width, but there is no clear evidence for an open courtyard. The survey found another cluster of the room described by Feature 8, while Feature 9 appears to be a large open area that is not clear if it relates to Feature 8. More than a dozen rooms were mapped from Feature 10 and at least four from Feature 11.

3.5.2 Ancient Mantinea, Peloponnese, Greece

Mantinea was established within a level flood basin of northeastern Arcadia in the Peloponnese before the middle of the 5th century B.C.E. Due to the lack of archaeological and literary evidence up to the present, its date of foundation remains unknown. In 385 BCE, the city was destroyed by a Spartan invasion, and its citizens were forced to depopulate. For 15 years, Mantinea was abandoned until it was reestablished in 370 BCE after Sparta's defeat in the Battle of Leuctra. The city played a prominent role in the activities of the newly established Arcadian League during the 4th century B.C.E., and, along with Megalopolis and Tegea, continued to be an influential regional presence in Arcadia and the Peloponnese for several centuries.

The known archaeological features at Mantinea include the well-preserved elliptical fortification walls, approximately 4 km in circumference, and the agora and theater at the center [112]–[114] but very little of the remaining urban area inside the fortification walls (~120 hectares) has been explored. A geophysical survey through the use of soil resistivity and magnetic methods was conducted by the University of Patras (Greece) from 1988-91 northwest of the theater [115]. The target area was limited to 1 hectare, but the survey revealed evidence for subsurface streets arranged at right angles together with various buildings, possibly domestic in nature.

A geophysical survey was conducted to explore the structure and urban development of the classical Greek city of Mantinea in the Peloponnese through an intensive geophysical fieldwork campaign carried out by the Laboratory of Geophysical, Satellite Remote Sensing and Archaeo-environment of the Institute for Mediterranean Studies (FORTH). For this task, the GPR system Noggin smart cart of Sensors & Software (*Figure 3.8a*) was also employed using a 250MHz antenna. An area of 2.45ha was covered in total with GPR using 0.5m spacing between each transect. The data acquired with Noggin GPR were noisy, but they exhibit anomalies related to buried structures. To enhance the anomalies related to buried antiquities the data were processed using the following corrections order: trace reposition, time zero correction, dewow filter, S.E.C. gain, background removal, bandpass filtering, and migration. Here, the results obtained from the eastern side of the agora are presented (*Figure 3.10a*).

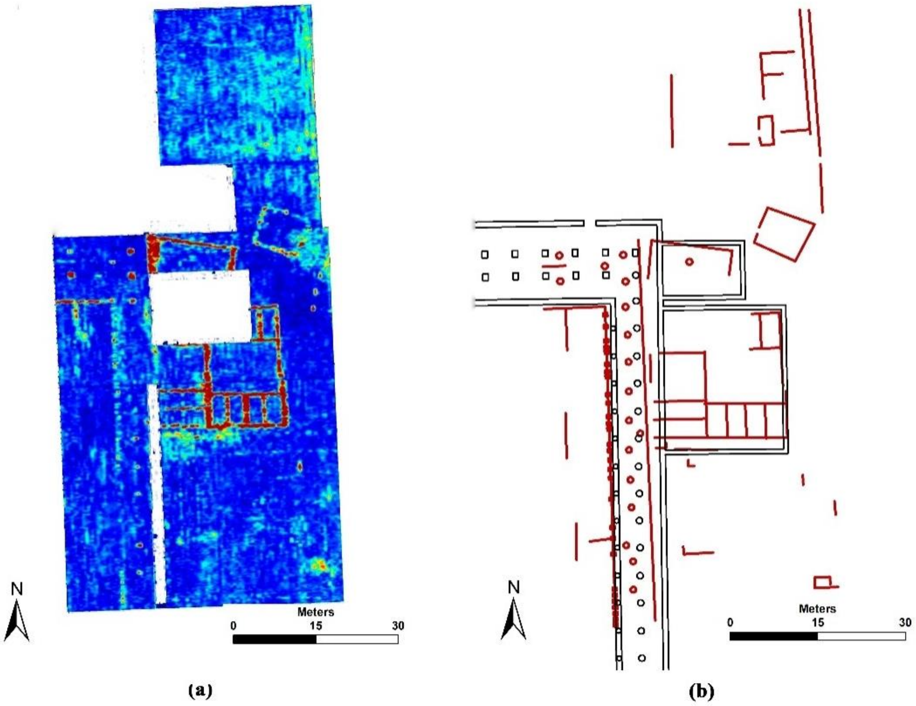


Figure 3.10: GPR results on the eastern side of the agora at Mantinea. (a) GPR slice illustrating the anomalies at 1.0-1.1 m depth. (b) Comparison of the French plans (black color) and interpretation (red color) occurred from selected GPR depth-slices using ArcGIS.

Figure 3.10a illustrates the slice at 1.0-1.1m estimated depth, where public buildings, including a long "L" shape stoa with columns, appear as strong anomalies with detail. Many of them seemed to match the public buildings excavated by the French in the 19th century and were reburied [116]. The similarities and differences with the French plans are shown in Figure 10b, where black lines are the findings of the excavation while red lines are the interpretation of the GPR data as occurred from all the depth slices. There is an inclination in the orientation of the whole settlement. The "L" shape stoa has double internal rows colonnades in the northeast direction and a single row along the northsouth direction. Two adjoining structures also appear as strong linear anomalies behind the west of the stoa. Those structures also appear in the French plan. The southern one is almost a perfect square and is subdivided into smaller rectilinear rooms on either side of larger central rooms. The French plan of the agora did not show the internal subdivision of space. Only a part of the other structure could be surveyed because of a large tree, but it is clear that it is smaller than its neighbor and likely had no interior rooms. The building is also oriented at a diagonal angle (unlike the southern building), which is a distinct characteristic that is not present on the French plan. Another notable anomaly to the northeast is a small structure and is also set at a diagonal angle. There does not appear to be an internal subdivision of space.

3.5.3 Neolithic Thessaly, Greece

A large-scale geophysical exploration was conducted by the Laboratory of Geophysical, Satellite Remote Sensing and Archaeo-environment of the Institute for Mediterranean Studies (FORTH) during 2014 at numerous Neolithic tell sites, known as *Magoules*, in Thessaly. The purpose of this study was the identification of intra- and inter-spatial patterns of Neolithic settlements in regions under study through the comparative study of both archaeological and geophysical data. Non-destructive geophysical methods like electrical resistivity, magnetics, E.M., and GPR were used on selected sites that were either partially excavated or identified by survey expeditions. The most remarkable results were derived from magnetics and included various Neolithic features like ditches, enclosures, paleochannels, burnt structures, daub, and stone structures, etc.

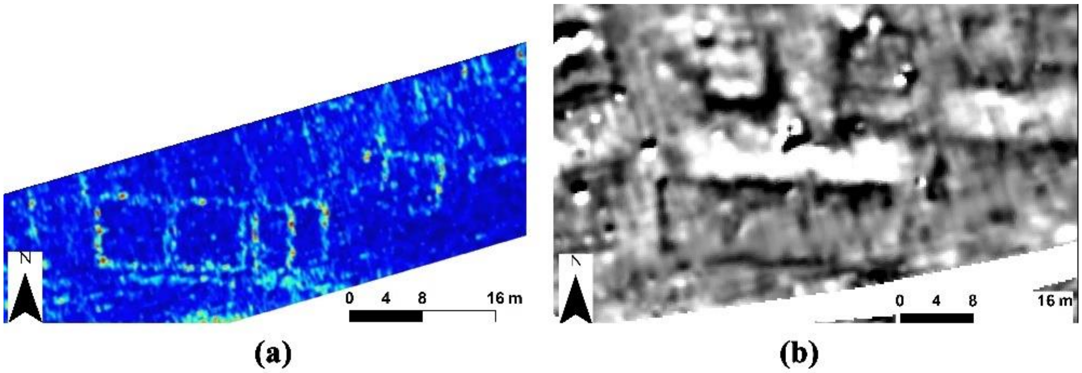


Figure 3.11: GPR and magnetic results from Magoula Almiriotiki. (a) GPR slice at 0.7-0.8m and (b) magnetic results. GPR exhibits better resolution revealing that the large structure on the magnetic results is a cluster of buildings.

Survey with GPR on those environments was not an easy task due to the rough terrain and the modern cultivations. The geomorphology of the area and the soil condition (clay-rich environments) resulted in noisy data with minimal signals penetration (up to 1.0m). Thus, GPR was used mostly as a supplementary method to enrich the information obtained from other geophysical methods. This case study aims to show the contribution of GPR, even though the data were problematic. One example is presented in *Figure 3.11*. The left image is the depth slice derived from GPR with a 250MHz antenna at Magoula Almyriotiki, while the right image is the magnetic results from the same position. Even though GPR could not map all the structures as the magnetics did, it shows better resolution to the mapped ones revealing that the big structure appearing on the magnetic data is, in reality, a cluster of individual smaller houses.

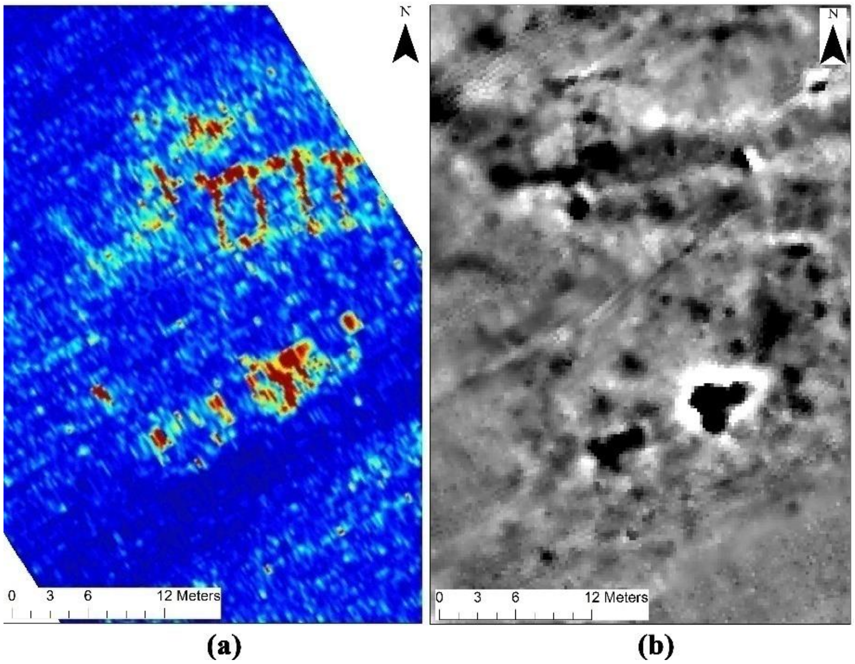


Figure 3.12: GPR and Magnetics obtained results from Magoula Perdika 2. (a) GPR slice at 0.7-0.8m and (b) magnetics results from the same position. GPR revealed a structure, probably stone-made, that is barely visible with white shades on the magnetics data. Both images are georeferenced in ArcGIS.

A different example comes from Magoula Perdika 2 (Figure 3.12). In this case, GPR managed to map a structure with great detail (*Figure 3.12a*) that is barely visible on the corresponding magnetic results (*Figure 3.12b*). This contrast between the two methods gives additional information about the structure's building materials that is most probably made of stone. This is because magnetics can better map burned clay structures as they exhibit greater contrast in magnetics properties, and hence, a stone structure will appear more faded than a burned one. On the other hand, GPR is relying mostly on the contrast of the electrical properties, and stone structures are expected to cause stronger reflections.

3.6 Discussion

In this chapter, an overview of GPR's theoretical background was given, including data collection and processing, emphasizing mapping structures where three case studies were presented. The aim here was to show the capability and limitation of GPR to detect structures. Overall, GPR is proven to be a great tool for archaeological investigations. When soil conditions and geomorphology are appropriate, GPR can provide detailed data of the subsurface and successfully map numerous features, including buried structures, roads, and city blocks. The results were also proven accurate when compared with ground truth information. Even if the survey conditions are not ideal and high noise levels exist in the data, it is impossible for them to be improved with a proper data processing workflow. For such cases, GPR can still perform nicely as a complementary method providing additional information to understand the subsurface's real conditions better.

4. Theoretical Framework of CNNs

Convolutional Neural Networks (CNNs or ConvNets) are part of the *Artificial Neural Networks* (ANNs) evolution that started with the introduction of the perceptron back in 1958. The use of convolutional layers in neural network architecture was firstly introduced in the model of *Neocognitron* by Fukushima in 1980 [49]. Based on this concept, along with developments in ANNs gradient-based training algorithms, LeCun et al. introduced a CNN architecture named LeNet-5 for handwritten digits classification in 1998 [51]. LeNet-5 is the foundation of every CNN architecture used nowadays. CNNs follow the same framework as other Neural Networks and ML algorithms in order to learn from data. In this chapter, this framework is described, emphasizing image classification. It begins with the basic Machine Learning (ML) concept to learn from data, proceeds with training a neuron using gradient-based algorithms, continues with training the Feed Forward Neural Networks (FFNs), and expands in applied these to a CNN architecture. Further, methods and techniques to improve training are also discussed as they play an important role in more recent CNNs developments and state-of-the-art performances.

4.1 The Concept of Learning from Data

Learning from data is among the primary purposes of Machine Learning. As a notion, “learning” is abstract and may expand to different domains and applications. Even in ML, there is no strict definition of it, and, as Goodfellow et al. mention in [43], many authors describe its concept intuitively. However, there are attempts for a formal definition. Mitchell in [117] defines learning from data in terms of *experience*, *tasks*, and *performance measures*. For the scope of this research and simplicity reasons, a more intuitive description will be given that is mainly adopted from Abu-Mostafa et al. in [118].

4.1.1 The learning problem

According to Abu Mostafa et al. in [118], the starting point of learning from data is the *learning problem*. It usually involves finding an unknown formula that will take some inputs and will produce a specific outcome. The produced outcome is the *learning target*. Expressing the learning problem using symbols, if $\mathcal{X}$denotes the space that all the possible inputs belong to, and $\mathcal{Y}$ denotes the space of the targeted outputs, a solution to the learning problem will be to find the target function, f_{target}, that will take any inputs derived from $\mathcal{X}$ and will produce outputs that will belong in space $\mathcal{Y}$:

$$f_{target}: \mathcal{X} \rightarrow \mathcal{Y} \tag{4.1}$$

Thus, the goal of the learning process is to find an approach to the unknown target function: $g \sim f_{target}$. This approach will be the solution to the learning problem, and the means to reach that solution is the *learning algorithm*.

4.1.2 The learning algorithm

Since an analytic solution of g is not feasible, the way to finally find g will be empirical using datasets and learning algorithms. The datasets are pairs of inputs-outputs *examples* or *data points*, $\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$, that is known to belong in the spaces $\mathcal{X}$ and $\mathcal{Y}$ so that, $y_i = f_{target}(x_i)$ with $i = 1, \dots, N$. The learning algorithm takes as input the examples in $\mathcal{D}$ and tries to find a formula that best approximates f_{target}. In other words, a learning algorithm is a sequence of steps and procedures capable of learning from the data by approaching f_{target}. To do so, the algorithm examines some candidate formulas, h_k, that derived from a set called the *hypothesis set*, denoted by $\mathcal{H}$, and chooses the one *that best fits* the *data*. This formula, let denote it as h_{out}, will be the output of the algorithm and a possible candidate of g. Whether $h_{out} = g$ is a crucial step in the learning process and is discussed in the following paragraphs.

The hypothesis set $\mathcal{H}$ depends on the *task* that the learning algorithm performs (also referred to as *a machine learning task* or *learning task*). According to [43], the task is the way a machine learning system should process an example and, more specifically, its *features*. Features are quantitative measures found in the dataset, $\mathcal{D}$, and describe the subject of the learning problem. The notion of the task is better understood when giving practical examples. Goodfellow et al., in [43], address a few that are commonly performed in learning problems. Namely, these are *classification*, *classification with missing outputs*, *regression*, *transcription*, *machine translation*, *structured output*, *anomaly detection*, *synthesis and sampling*, *imputation of missing values*, *denoising*, and *probability mass function estimation*. The list of examples is, of course, much more extensive as it expands to the whole range of machine and deep learning applications. Thrunn and Prut in [119] give a more conceptual insight into the learning tasks and their contribution to an algorithm's ability to learn.

4.1.3 Learning types

Up to this point, a generic and abstract concept of the learning process is described. When it comes to practice, significant variations of the learning process exist. One can guess that every learning problem's solution depends on the available data and can be acquired using different tasks and thus different learning algorithms. Despite the complexity, there are some common patterns observed in the algorithms' learning behavior concerning the data that allows the categorization of learning. The most common types of learning mentioned in the literature are *supervised, unsupervised*, and *reinforcement* [43], [120]–[122].

In supervised learning (also called *predictive*), the user acts as a "supervisor" by providing the learning algorithm sets of inputs along with their associated outputs. The learning algorithm's purpose is to find a function that will lead from input to output given a task. Tasks mostly performed using supervised learning algorithms are classification and regression. If the outputs are categorical, the former is used, while if the outputs are real-valued, the latter is chosen. Some examples of popular supervised learning algorithms are *Linear regression*, *Perceptron*, *Logistic regression*, and *Support Vector Machine*.

Unsupervised or descriptive learning makes use of data that solely consists of inputs. The purpose usually is to discover similarities or patterns within the data. Some tasks that are usually performed under this category are clustering, density estimation, and dimensionality reduction [120]. Among the well-known unsupervised algorithms are the *Principal Components Analysis* (PCA) and the *k-means clustering*.

Reinforcement learning is a particular category that focuses on learning through a trial and error system without any guidance from the user. In this case, the data are observations of an environment, and the algorithm learns how to act in this environment, given a particular observation. The learning comes through rewarding each correct action is taken [123].

At this point, it should be mentioned that this categorization of learning is not absolute nor has strict limits. Authors like Ayodele in [120] mention more types of learning like *semi-supervised*, *transduction*, and *learning to learn*. These are mainly subbranches of supervised and unsupervised learning types, but the further description of this topic goes beyond the scope of this dissertation. Since the main subject of this research is classification using supervised learning algorithms, the emphasis is given to concepts related to the supervised approach of learning.

4.2 Training process

The *training* "teaches" the learning algorithm to perform the task using the *training data* or *examples*. On an abstract level, training is the process during which the learning algorithm searches for the function h_{out} inside the hypothesis set $\mathcal{H}$, so that $h_{out} = g$ to best approximate f_{target}. Finding h_{out} is performed empirically using training data. In supervised learning, the training data or datasets are pairs of examples drawn from the input space $\mathcal{X}$ and the corresponding output from the target space $\mathcal{Y}$. In this case, finding h_{out} means *to model* the relations between inputs-outputs, usually using *weights*, and measure how well the model performs. The hypothesis set and the learning algorithm that are both chosen by the user will make a *learning model* [118]. The training description might seem vague on this abstract level, as it is a concept better described using practical examples. Here, training for classification will be described using linear models as the latter are used to model the *artificial neuron*.

4.2.1 Training for classification

So far, it has only been mentioned that classification is a task that a learning algorithm performs. What classification does is to assign labels $y = \{1, .., K\}$, that is the output space $\mathcal{Y}$, to some input data x ∈ $\mathcal{X}$. In other words, it is the process that categorizes the inputs into the K classes. In machine learning applications $K \in \mathbb{Z}$ should be at least 2. When $K = 2$, is the case of *binary* classification, and y takes the values of 0 and 1. When $K > 2$, then it is called *multiclass* or *categorical* classification. Regardless of the number of the classes, the classification problem can be formalized using the function approximation described earlier in subsection 4.1:

$$y = f_{class}(\mathrm{x}) \tag{4.2}$$

The unknown function f_{class}, takes the inputs x and classifies them into y. The learning algorithm's goal is to produce a function g_{class} that will classify the inputs in the same manner the unknown function would, i.e., $g_{class} \sim f_{class}$. Thus, finding g_{class} is what training a learning algorithm is about. The latter will pick functions, h, from the hypothesis set and measure their performance on the given classification task. This is performed on a training data set. The training dataset consists of examples drawn from the input space $x_i \in \mathcal{X}$, along with a label,$y_i \in \{1, \dots, K\}$ which indicates the category they belong to. In other words, there are pairs of inputs-labels that is known to satisfy $y_i = f_{class}(x_i)$. To approximate the unknown f_{class} , a known function $h \in \mathcal{H}$ is picked by the learning algorithm, assuming it fits the training data set. Then, the algorithm examines this assumption by using h to make predictions of the label that the input x_i belongs to:

$$\hat{y_i} = h(x_i) \tag{4.3}$$

Equations (4.2) and (4.3) describe the functional approach that the algorithm learns from the data. A probabilistic approach is the one to be preferred in most classification learning problems as it introduces uncertainty that makes it more realistic. Instead of the unknown classification function, the learning algorithm tries to find the unknown distribution that generates the output label y given the training data $\boldsymbol{x}$**.** So, f_{class} is now expressed in terms of $P(y|\boldsymbol{x})$ and the training data are samples derived from that distribution. The hypothesis $h(\boldsymbol{x})$ will now use the training data to approach the targeted distribution, and the way to approach it is through the notion of *likelihood*. That is, the probability of hypothesis $h(\boldsymbol{x})$ parametrized by $\boldsymbol{w}$ to have modeled the unknown distribution correctly. If the two classes are a binary case with labels being +1 and -1, i.e., $y = \pm 1$, the hypothesis $h(\boldsymbol{x})$ is now expressed as [121]:

$$P(y = 1|\boldsymbol{x}; \boldsymbol{w}) = h(\boldsymbol{x}) \tag{4.4}$$

Thus,

$$P(y = -1|\boldsymbol{x}; \boldsymbol{w}) = 1 - h(\boldsymbol{x}) \tag{4.5}$$

The parameter $\boldsymbol{w}$ is a vector of weights, which shows the contribution of an input value to the classification label under question. So, the functions in the hypothesis set are expressed in terms of the training input values, $h(x_i, w_i)$. The training process for classification is about updating the weights to find an h that best fits the training data. This is performed using error measures and then updating the weights using an optimization technique to minimize the cost function.

4.2.2 Training error & cost functions

The error measures serve two purposes in the training process. The first is to evaluate the produced hypothesis $h(\boldsymbol{x}, \boldsymbol{w})$, and the second is to update the weights. For the evaluation of the training results, the notion *training error*, E_{train}, is used. It is usually defined by a *loss* or *cost* function that quantifies the offset between the predicted value $\widehat{y_n}$ is and the value y_n of the training dataset. For classification where the y's are classification labels, E_{train} can be expressed as the average error of the individual data samples, N, defined by a cost function [118], [121]:

$$E_{train} = \frac{1}{N}\sum_{n=1}^{N} cost(\widehat{y_n}, y_n) \tag{4.6}$$

The type of cost function is chosen by the user and is an essential component of the learning process. Different error functions will lead to different results that affect any decision made regarding the final hypothesis $h(\boldsymbol{x}, \boldsymbol{w})$. It also affects the process of updating the weights, as will be described in the following paragraph. The choice of $cost(\widehat{y_n}, y_n)$ is usually done empirically while taking into account the training data, the learning, and the learning task.

An example is *the 0-1 cost function* that quantifies the misclassified points for the case of binary classification [120]:

$$cost(\widehat{y_n}, y_n) = \mathbb{I}(\widehat{y_n} \neq y_n) = \begin{cases} 0, & \widehat{y_n} = y_n \\ 1, & \widehat{y_n} \neq y_n \end{cases} \tag{4.7}$$

where $\mathbb{I}(\cdot)$ is the *indicator* function.

Another example is the *squared error* [43]:

$$cost(\widehat{y_n}, y_n) = \varepsilon_n(\widehat{y_n}, y_n) = (\widehat{y_n} - y_n)^2 \tag{4.8}$$

that is used in Mean Squared Error (MSE), a very popular approach in machine learning, especially for the task of regression.

However, the most popular choice for classification is using cost functions based on the *cross-entropy*. Cross entropy quantifies the dissimilarities between the distribution of the training set and the predictions of the model produced by the hypothesis $h(\boldsymbol{x}, \boldsymbol{w})$. The definition of cross-entropy is based on the expected value, $\mathbb{E}[\cdot]$, of a function $f(x)$ with respect to the probability distribution $P(x)$ [43]:

$$\mathbb{E}[f(x)] = \sum_{x} P(x) f(x) \tag{4.9}$$

If $P_{train} = P(y|\boldsymbol{x})$ is the empirical distribution that the training data are following, and $p_{model} = P(y|\boldsymbol{x}; \boldsymbol{w})$ is the distribution of the predictions parametrized by the weights $\boldsymbol{w}$, the cross-entropy is the expected value of the negative log-likelihood of p_{model}[43], [120]:

$$J(\boldsymbol{w}) = -\mathbb{E}\log p_{model} = -\sum_{x} P_{train}\log p_{model} \tag{4.10}$$

In other words, it expresses the cost of using the p_{model} with respect to the empirical distribution, P_{train}, that the algorithm aims to learn. Hence:

$$cost(\widehat{y_n}, y_n) \equiv J(\boldsymbol{w}) = -\mathbb{E}\log p_{model} \tag{4.11}$$

Equations (4.10) and (4.11) express a category of cost functions that are based on the cross-entropy. The final functional form of these equations depends on p_{model} and thus, varies from model to model. For example, it can be the negative log-likelihood for Gaussian or Bernoulli distribution. In this dissertation, the emphasis is given on cost functions used in *feedforward neural networks* (FNNs) (see section *4.3*). The chosen cost function is then used to improve the accuracy of the training. This is done using optimization techniques. In this dissertation, gradient-based techniques are used.

4.2.3 Gradient-based optimization

Gradient-based optimization techniques are the core of training in supervised learning. They are iterative algorithms that calculate the gradient of the cost function with respect to the weights and update them to minimize it. The principles of *gradient descent* (GD) and *stochastic gradient descent* (SGD) are given below.

- **Gradient Descent**

Gradient Descent or *Steepest Descent* is a general iterative algorithm used in finding a local minimum of a differentiable function or the solution of any linear or non-linear system. For training learning algorithms, it is used to find the values in the weight vector $\boldsymbol{w}$ that will minimize the training error E_{train} defined with a cost function (equation (4.6)) with respect to the weights. The minimization will occur at a point in the weight space that the gradient of the error function becomes:

$$\nabla E_{train}(\boldsymbol{w}) = 0 \tag{4.12}$$

This point can be either a *local* or a *global minimum* of the error function. Gradient descent will find a numerical solution to equation (4.12) through an iterative procedure of updating weights. For a number of iterations $t = 1,2, \dots, N$, and an initial value for weights $\boldsymbol{w}^{(0)}$, GD will update the weights using small steps in the direction of $-\nabla E_{train}(\boldsymbol{w})$ as [121]:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\nabla E_{train}\left(\boldsymbol{w}^{(t)}\right) \tag{4.13}$$

The parameter $\eta < 0$ is a step size, also known as the *learning rate,* and affects how quickly GD will approach the minimum (local or global). It has a high impact on GD performance. Ideally, η should be small enough when the algorithm is close to a local minimum and big when it is far from approaching it faster. One option to achieve such behavior, according to [118], is to set η in proportion to the norm of the gradient, $\eta_t = \eta \|\nabla E_{train}\|$ using a fix value for η. The empirical value of around 0.1 is suggested.

- **Stochastic Gradient Descent**

Stochastic Gradient Descent (SGD) is a randomized version of GD. The main difference is that the gradient, $\nabla E_{train}(\boldsymbol{w})$, is calculated for a single point of the training data set instead of the whole N points. For this reason, GD is also called *batch gradient descent*. In SGD, a random point (x_n, y_n) is picked from $\mathcal{D}_{train}$ and $cost(\widehat{y_n}, y_n)$ is calculated for this data point using the hypothesis $\widehat{y_n} = h(\boldsymbol{w}, x_n)$. The weights are then updated in the same way as GD to minimize the cost error gradient, using the equation (4.13). The justification on why SGD works lies in the expected value of the weight change with respect to a randomly picked point [118]:

$$\mathbb{E}[-\nabla cost(\widehat{y_n}, y_n)] = \frac{1}{N}\sum_{n=1}^{N} cost(h(\boldsymbol{w}, x_n), y_n) = -\nabla E_{train} \tag{4.14}$$

Equation (4.14) expresses that the expected weight change is the same as the batch GD. This is because, over several randomly selected points, possible fluctuations in the gradient direction will be canceled out, leading to an “average” direction that will be the same as the deterministic GD. In practice, SGD is a simple algorithm that works well with lower computational cost over GD. The introduced randomness of picking one data point at a time helps in preventing getting stuck on a shallow local minimum point, which leads to better optimization [43], [118].

4.3 Feedforward Neural Networks

Feedforward Neural Networks (FNN) are Artificial Neural Networks (ANN) that perform classification. ANNs are generally inspired by the biological neural networks [124], which are conceptually described as *neuron circuits* [125], i.e., a population of neurons interconnected by *synapses* to pass signals when activated. The synapses are modeled by weights that point to whether the connection with a different neuron is excitatory or inhibitory. The weights' magnitude also establishes whether a connection is strong (high values) or weak (values close to zero). The FNN architecture is based upon this concept, where the artificial neurons (also called *units* or *nodes*) are organized into layers. An example of a typical FNN architecture is presented in *Figure 4.1*. It consists of an *input layer*, an *output layer*, and *hidden layers* in between. The input layer feeds the network with information drawn from input space, $\mathcal{X}$. Then is processed in the hidden layers through the application of transformation or activation functions, φ, and is finally categorized into labels defined by the output layer. The output labels are generated by a hypothesis $h(\boldsymbol{x})$ that the FNN has learned. The indices $\ell = 0,1, \dots, L$ are used to number the layers of an FNN, where $\ell = 0$ is the input layer, $\ell = L$ is the output layer, and any value $0 < \ell < L$ is used for the hidden layers. When the number of the intermediate hidden layers increases, the FNN is characterized as *deep* and hence the name deep learning.

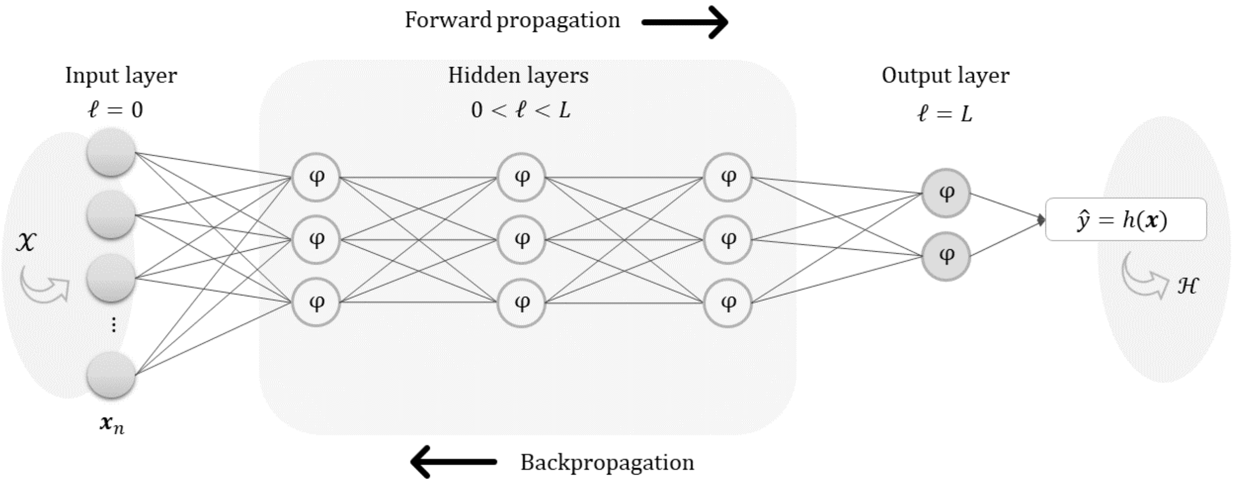


Figure 4.1: Illustration of a Feed-Forward Network. The network consists of the input layer that uses examples drawn from input space $\mathcal{X}$, the hidden layers where activation functions, φ, are applied and the output layer. The produced output is the classification labels of a learned hypothesis $h(\boldsymbol{x})$ picked from the hypothesis space $\mathcal{H}$. The black arrows denote the directions that forward and backpropagation are performed.

To train such a network requires two processes. The first is to make a prediction following a sequence of steps known as the *forward propagation,* and the second is to update the weights using the *backpropagation algorithm*. To better describe these two processes, the artificial neuron model will be first introduced as it is the fundamental unit of any ANN.

4.3.1 The neuron model

The artificial neuron is the mathematical model of the biological neuron. For convenience, the artificial neuron will now be referred to only as a neuron. The neuron performs *linear classification* through three main processes that are multiplication, summation, and the application of an *activation* or *transfer* function. In more detail, the inputs x_n that enter the neuron are weighted by multiplying them with a corresponding w_n. The products are summed up, and a bias term, b, is added. The last step is the application of the activation function, φ to produce the output [126]:

$$h(\boldsymbol{x}) = \varphi\left(\sum_{n=1}^{N} w_n x_n + b\right) \tag{4.15}$$

where $\boldsymbol{x}$ is a real-valued vector containing the inputs x_n, and $h \in \mathcal{H}$ is the produced hypothesis of linear form that uses a threshold based on the mathematical function of φ. Traditionally φ is the *signum function* that is known as the *perceptron,* the *sigmoid function* that is known as the *logistic regression*.

- **Perceptron**

Frank Rosenblatt introduced the perceptron in 1958 [127] as a hypothetical nervous system or a machine with the ability to learn. Nowadays, it is best known as a learning algorithm used solely for the binary classification of data. The perceptron can perform properly only when using linearly separable data. That means a line exists that can divide the data into two classes. Using the signum function, equation (4.15) becomes:

$$h(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{n=1}^{N} w_n x_n + b\right) \tag{4.16}$$

that uses a threshold:

$$h(\boldsymbol{x}) = \begin{cases} 1, & \sum_{i=1}^{N} w_n x_n + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{4.17}$$

The output values $y = \{-1, 1\}$ represent the two categories of the binary classification. The dot product of equation (4.16) is usually expressed in vector form:

$$h(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^T \boldsymbol{x}) \tag{4.18}$$

where $\boldsymbol{x}$ is a real-valued vector and $\boldsymbol{w}$ is the vector containing the weighs w_i and the bias term b as $w_0 = b$. The concept of the perceptron is illustrated in *Figure 4.2*. Training the perceptron is about finding the optimal $\boldsymbol{w}$ that produces a line described by final hypothesis $h(\boldsymbol{x})$ that assigns the correct labels in $\boldsymbol{x}$. The training process is performed iteratively and is known as the Perceptron Learning Algorithm (PLA).

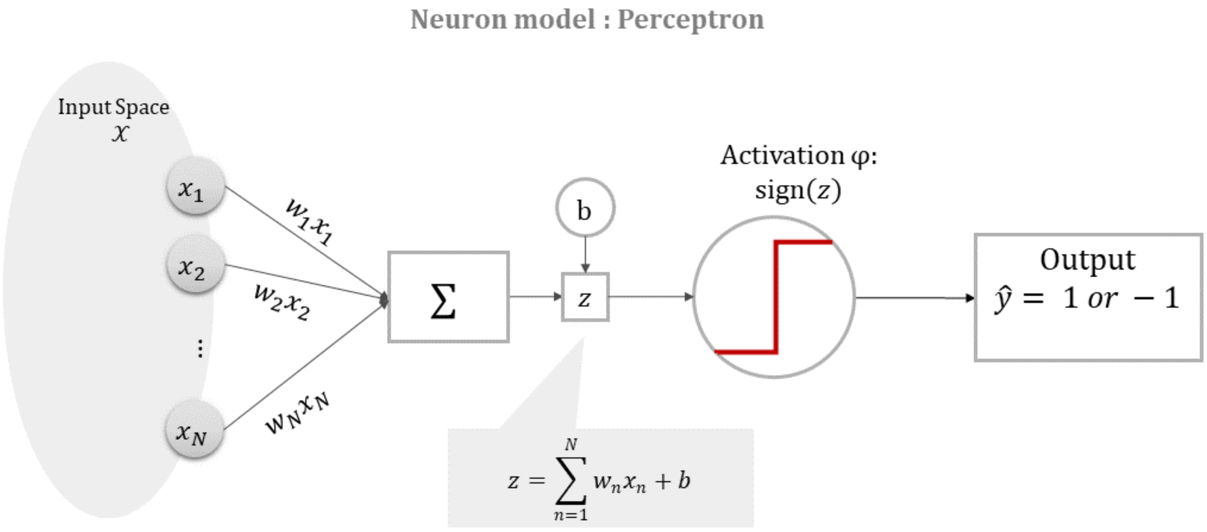


Figure 4.2: The Perceptron model of an artificial neuron.

PLA uses a dataset that contains pairs of inputs-labels, $D_{PLA} = \{(\boldsymbol{x}_n, y_n)\}$, with $n = 1, .., N$ being the number of examples, $\boldsymbol{x}_n \in \mathcal{X}: \mathbb{R}$ and $y_n \in \mathcal{Y}: \{-1, 1\}$. If t denotes the number of iterations, the starting point is to solve equation (4.18) using the inputs $\boldsymbol{x}_n$ and an initial value of $\boldsymbol{w}^{(\mathbf{0})}$ to produce labels $h(\boldsymbol{x}_n)$. Following the training process, the next step is to calculate the training error, E_{train} (equation (4.6)), using a cost function to find the misclassified points, $y_n \neq h(\boldsymbol{x}_n)$(equation (4.7)). The algorithm will focus on these points and, picking one misclassified point at a time and, it will iteratively update the corresponding weights that will lead to the correct labels. For each iteration, $t > 0$, PLA picks the current weight, $\boldsymbol{w}^{(t)}$ of the misclassified point , and updates it using the following rule [118]:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + y^{(t)} x^{(t)} \tag{4.19}$$

The algorithm terminates when there are no other misclassified examples in the data set. One worth mentioned modification of PLA is the *pocket algorithm*, which behaves better when training data are not linearly separable. Several learning algorithms that were developed in the years to follow were based on this concept, including Neural Networks. Gallant compares some of them in [128].

- **Logistic Regression**

Logistic regression performs binary classification using a line or a plane as in PLA. In logistic regression, the non-linear sigmoid function is used that is defined as:

$$\sigma(z) = \frac{1}{1+e^{-z}} \in (0,1) \tag{4.20}$$

If z is the neuron's input signal:

$$z = \sum_{n=1}^{N} w_n x_n = \boldsymbol{w}^T \boldsymbol{x} \tag{4.21}$$

the hypothesis space of equation (4.15) becomes:

$$h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) \tag{4.22}$$

The equation (4.22) is the vectorized form of the logistic regression and outputs values that range from 0 to 1. The model of logistic regression is presented in *Figure 4.3*.

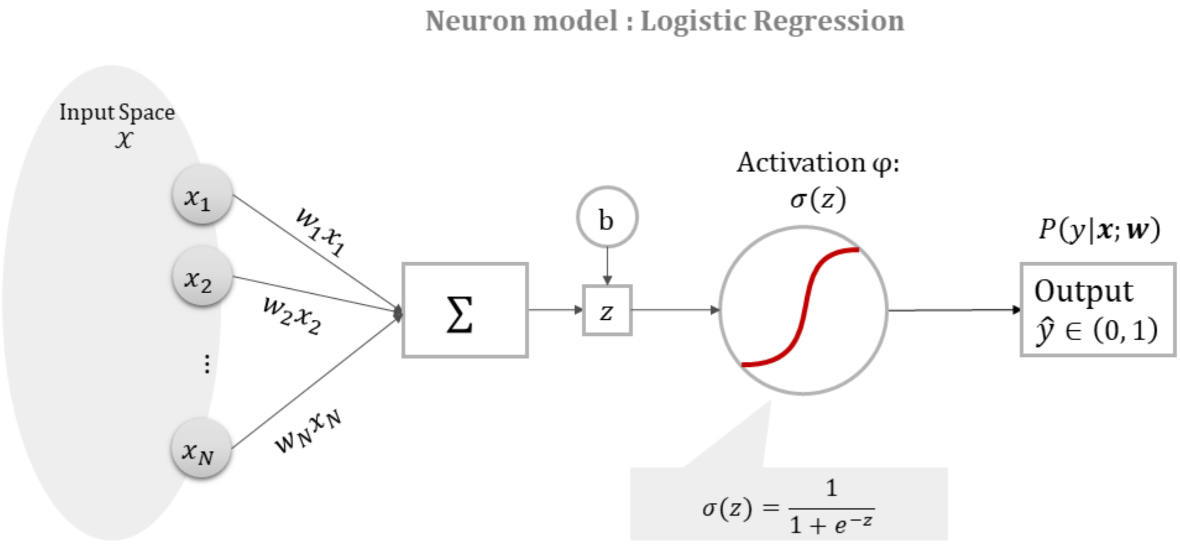


Figure 4.3: Logistic regression for modeling an artificial neuron.

The output of logistic regression is interpreted as the probability of the input belonging to a class, allowing this way a level of uncertainty. When compared to perceptron is like a softer threshold [118]. Further, logistic regression will try to learn the unknown distribution that generates the output labely given $\boldsymbol{x}$, $P(y|\boldsymbol{x})$ using the likelihood as was described earlier. Hence, the equation (4.4) becomes:

$$P(y = 1|\boldsymbol{x};\boldsymbol{w}) = h(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) \tag{4.23}$$

With,

$$P(y = -1|\boldsymbol{x};\boldsymbol{w}) = 1 - h(\boldsymbol{x}) \tag{4.24}$$

The cost function used to define the training error in logistic regression is based on the cross-entropy:

$$E_{train} = \frac{1}{N}\sum_{i=1}^{N} \ln\left(1 + e^{-y_i \boldsymbol{w}^T \boldsymbol{x}_i}\right) \tag{4.25}$$

The equation (4.25) is the result of maximizing the likelihood of $h(\boldsymbol{x})$ (which means closer to the target distribution) that is equivalent to minimizing the negative log-likelihood (equation (4.10)). The mathematical proof is given in [118]. Thus, to train the logistic regression means to update the weights of the sigmoid function $h(\boldsymbol{x})$ to minimize E_{train} of equation (4.25). This is performed with gradient-based optimization methods (SG or SGD) described previously.

The single neuron model has the limitation of classifying only linear separable data. This limitation was overcome by using multiple neurons arranged in layers, which led to feedforward networks. Modern FFNs are mainly based on logistic regression, while learning is implemented through two main processes, the feedforward, and backpropagation, which are discussed in the subsections below.

4.3.2 Feedforward propagation

Feedforward is the process of making a classification prediction using the weights assigned in the connections of the neurons of each layer. FNN was named after feedforward propagation to differentiate them from a different ANN type, the *Recurrent Neural Networks* (RNNs), where the signals are allowed to move back and forth using loops that form directed cycles [129].

Feedforward propagation resembles a "flow" of information inside the network that goes from the input layer to the output layer. The same processes described for the single neuron are performed, following this flow, by all the neurons of the FNN. Briefly, these are multiplication with the weights, summation, bias addition, and application of an activation function to produce the output. The key concept of this layered network is that an activated output of one neuron will be the input of the neurons resided in the next layer. To better describe how this mechanism works, *Figure 4.4* will be used that illustrates a part of a deep feedforward network. This part consists of three hidden layers index as $(\ell - 1)$, (ℓ), and $(\ell + 1)$. The index of each layer is expressed using the superscripts. Additionally, the subscripts i and j are used to index the neurons of two consecutive layers. The former denotes the neuron that provides the feed while the latter the neuron under computations. Also, the weights assigned in the connections are expressed as $\boldsymbol{w}_{ij}$, which is interpreted as the weight vector from the connection of neuron i to neuron j.

In the example of *Figure 4.4*, the neuron i, from the layer $(\ell - 1)$ has produced an output of $\boldsymbol{a}_i^{(\ell-1)}$ that will be the input to a neuron, j, of the next layer ℓ. The weight vector $\boldsymbol{w}_{ij}^{(\ell)}$ is assigned and multiplied with $\boldsymbol{a}_i^{(\ell-1)}$ and,the dot product is calculated to produce the signal that will feed neuron j of layer (ℓ):

$$\boldsymbol{z}_j^{(\ell)} = \boldsymbol{b}_j^{(\ell)} + \sum_{i=1}^{d^{(\ell-1)}} \boldsymbol{w}_{ij}^{(\ell)} \boldsymbol{a}_i^{(\ell-1)} \tag{4.26}$$

where $d^{(\ell-1)}$ expresses the dimensionality of the layer that provides the input, and $b_j^{(\ell)}$ is the bias term that is added to the signal. The bias term is usually incorporated in the weight vector by adding a zero-indexed unit input on the layer that provides the feed, i.e.,$\boldsymbol{a}_{i=0}^{(\ell-1)} = 1$, and modify equation (4.26) as:

$$\boldsymbol{z}_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} \boldsymbol{w}_{ij}^{(\ell)} \boldsymbol{a}_i^{(\ell-1)} \tag{4.27}$$

where $b_j^{(\ell)} = \boldsymbol{w}_{0j}^{(\ell)}$.

Next, a non-linear activation function, φ, of choice is applied on the weighted signal $\boldsymbol{z}_j^{(\ell)}$ to produce the output value of the current neuron and current layer $\boldsymbol{a}_j^{(\ell)}$:

$$\boldsymbol{a}_j^{(\ell)} = \varphi\left(\boldsymbol{z}_j^{(\ell)}\right) \tag{4.28}$$

When all the activations for all the neurons of the layer (ℓ) are computed, the algorithm of forward propagation proceeds to the next consecutive layers that, in the case of figure, are (ℓ) and $(\ell + 1)$. The activated output $\boldsymbol{a}_j^{(\ell)}$ will now feed the layer $(\ell + 1)$, hence it will become the input $\boldsymbol{a}_i^{(\ell)}$ and equations (4.26) to (4.28) will be applied to produce the corresponding values of $\boldsymbol{z}_j^{(\ell+1)}$ and $\boldsymbol{a}_j^{(\ell+1)}$. This process is performed by all the neurons of the hidden layers.

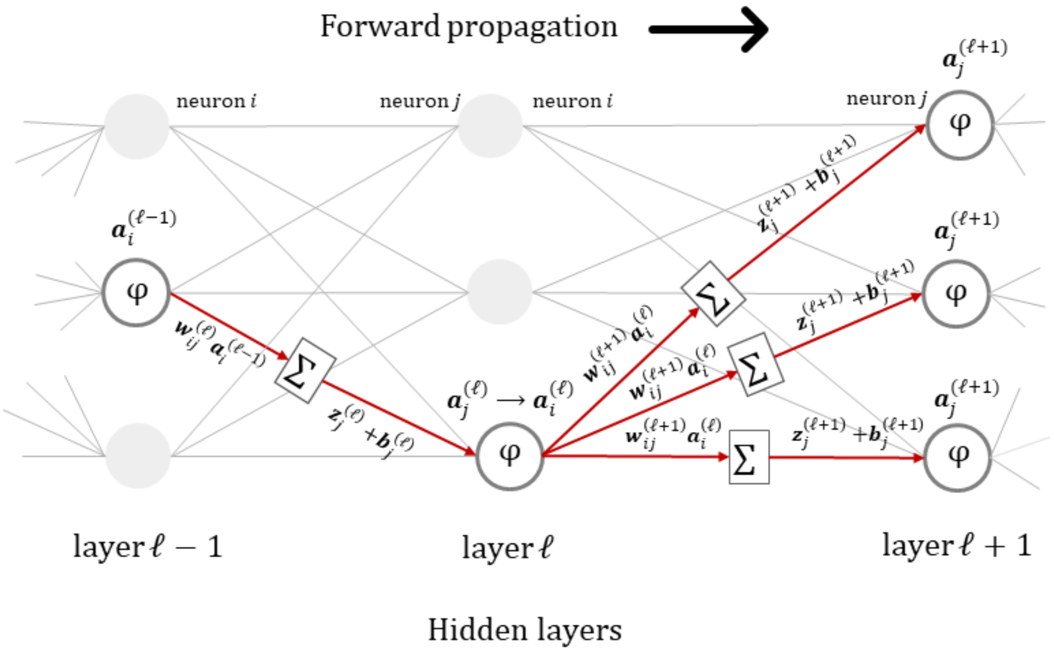


Figure 4.4: An example showing the computations performed in feedforward propagation for a neuron that is resided at the hidden layer ℓ. The neuron receives as input the activated output of a neuron of the previous layer, $\boldsymbol{a}_i^{(\ell-1)}$, and produces the current activated output $\boldsymbol{a}_j^{(\ell)}$. This output will be the input for the neurons of the next hidden layer $\ell + 1$ to produce the corresponding activated outputs using the same process.

For a more compact representation, the matrix form of (4.27) is used where $\boldsymbol{W}^{(\ell)}$ gathers all the weights of a hidden layer:

$$\boldsymbol{z}^{(\ell)} = \left(\boldsymbol{W}^{(\ell)}\right)^T \boldsymbol{a}^{(l-1)} \tag{4.29}$$

Hence, the forward propagation is a recursive process that begins when the network is fed with the training examples of the input layer $\boldsymbol{a}^{(\ell=0)}$ and ends when it reaches the output layer $\boldsymbol{a}^{(L)}$that are the predicted values of the produced hypothesis for the given $\boldsymbol{W}$. So, $\boldsymbol{a}^{(L)} = h(\boldsymbol{x}; \boldsymbol{w}) = \hat{\boldsymbol{y}}$, where $\boldsymbol{w} = \{\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{W}^{(L)}\}$. When reaching the point of obtaining the hypothesis and the predictions, the training error, E_{train}, is calculated using an efficient cost function, as was described earlier in this chapter (equations (4.6)- (4.11)). The cost function depends on the activation function that is chosen. More details on activation functions will be discussed later in this subsection.

4.3.3 Backpropagation Algorithm

The steps to train a feedforward neural network are no different from those used to train a single neuron (as described in perceptron and logistic regression). These are to use a cost function to calculate the training error $E_{train}(\boldsymbol{w})$ of the prediction made by the hypothesis $h(\boldsymbol{x}; \boldsymbol{w})$, and then use a gradient-based optimization method to update the weights by minimizing the cost function. The backpropagation algorithm performs the first step, which calculates the gradient of the cost function for a fully connected network of neurons. The gradient is calculated using partial derivatives of the chosen cost function with respect to the weights, $\partial E_{train}/\partial \boldsymbol{w}$. Because the resulted weight vector (i.e., from feedforward propagation) is a vector of matrices, $\boldsymbol{w} = \{\boldsymbol{W}^{(1)}, \dots, \boldsymbol{W}^{(L)}\}$, the partial derivative is calculated with respect to each weight matrix at layer (ℓ). For this reason, cost functions that can be expressed as sums of cost functions over the N individual data samples are used [130]:

$$\frac{\partial E_{train}}{\partial \boldsymbol{W}^{(\ell)}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial \varepsilon_n}{\partial \boldsymbol{W}^{(\ell)}} \tag{4.30}$$

where $\varepsilon_n = cost(\widehat{y_n}, y_n)$. Such cost functions can be the MSE or the cross-entropy-based cost functions. The term $\partial\varepsilon_n/\partial \boldsymbol{W}^{(\ell)}$ expresses that in a single layer (ℓ) of FNN, any change in the weights $\boldsymbol{W}^{(\ell)}$ will change the cost error of that layer. Due to the way the forward propagation works (*Figure 4.4*), the errors introduced with the assigned weights of a layer (ℓ) are affected by the errors introduced by the weights of all the previous layers through the input signal $\boldsymbol{z}^{(\ell)}$. In turn, they will be passed on the activated signal $\boldsymbol{a}^{(\ell)}$ that will affect the next layers, as equation (4.29) indicates, and will eventually be accumulated in the output layer (L). The way these errors are distributed is described by a quantity known as the *sensitivity* [130] or *delta error*:

$$\delta^{(\ell)} = \frac{\partial \varepsilon}{\partial z^{(\ell)}} \tag{4.31}$$

The error $\delta^{(\ell)}$ applies to a single layer of the FNN and expresses how the cost ε changes in that layer with respect to the input signal $z^{(\ell)}$ (equation (4.29)). Backpropagation make use of the *chain rule* of calculus and $\delta^{(\ell)}$ as an intermediate quantity to calculate the $\partial\varepsilon_n/\partial \boldsymbol{W}^{(\ell)}$of that layer as:

$$\frac{\partial \varepsilon}{\partial \boldsymbol{W}^{(\ell)}} = a^{(\ell-1)}\left(\delta^{(\ell)}\right)^T \tag{4.32}$$

The mathematical proof of the equation (4.32) can be found in [130] as well as in [121]. Equation (4.32) describes the basic concept of how the backpropagation algorithm works. It relies on how the weights of the previous layer (through the activated output $a^{(\ell-1)}$) and the $\delta^{(\ell)}$ of the weights of the current layer (through the input signal $z^{(\ell)}$) affect the cost function of layer(ℓ). The activated outputs, $a^{(\ell)}$, are known quantities that are calculated during the forward propagation, using the output of the previous layer $a^{(\ell-1)}$ (equations (4.28) and (4.29)). What remains for the backpropagation algorithm is to compute the delta errors. These calculations are made moving in the opposite direction of feedforward propagation (hence the name backpropagation). So, to calculate $\delta^{(\ell)}$, $\delta^{(\ell+1)}$is used [130]:

$$\delta^{(\ell)} = \varphi'\big(z^{(\ell)}\big)\odot\big(\boldsymbol{W}^{(\ell+1)}\boldsymbol{\delta}^{(\ell+1)}\big) \tag{4.33}$$

The error $\delta^{(\ell+1)}$ is fed backward towards layer (ℓ), multiplied with the weights, $\boldsymbol{W}^{(\ell+1)}$ and summed up. This is the input for the neurons of layer (ℓ). In forward propagation, when the input signal reaches a neuron, an activation function φ is applied. In backpropagation, the element-wise multiplication $\odot$, or *Hadamard product*, by $\varphi'\big(z^{(\ell)}\big)$ is applied to get $\delta^{(\ell)}$. The term $\varphi'\big(z^{(\ell)}\big)$ is the derivative of the activated input signal in layer (ℓ). The mathematical proof of equation (4.33) can be found in [130]. This equation, which is generic and applies to any hidden layer of an FNN, expresses the chain of dependencies between layers that will cause changes in the cost function. Since the direction of the computation is backward, the input and starting point of the backpropagation algorithm will be the error in the output layer $\delta^{(L)}$. Using equation (4.31) and the chain rule with the dependency of $\boldsymbol{a}^{(L)} = \sigma(z^L)$ as an intermediate value, the error of output layer can be computed as:

$$\delta^{(L)} = \frac{\partial E_{train}}{\partial z^{(L)}} = \frac{\partial E_{train}}{\partial a^{(L)}}\frac{\partial a^{(L)}}{\partial z^{(L)}} = \nabla_{\boldsymbol{a}} E_{train}\sigma'(z^L) \tag{4.34}$$

The functional form of the equations above depends on the activation functions and the cost function that are used. Once the backpropagation algorithm has computed the delta errors and hence $\nabla E_{train}(\boldsymbol{w})$, a gradient-based optimization algorithm like SG or SGD is implemented to update the weights, as was described earlier in this chapter.

Thus, a training algorithm for FFN has three main blocks (*Figure 4.5*); the first is forward propagation to make a prediction; the second is backpropagation to calculate the gradient of the error, and the third is to perform SG or SGD to update the weights. A complete run of these three blocks defines an epoch. Training algorithms are executed over many epochs until minimum error is achieved.

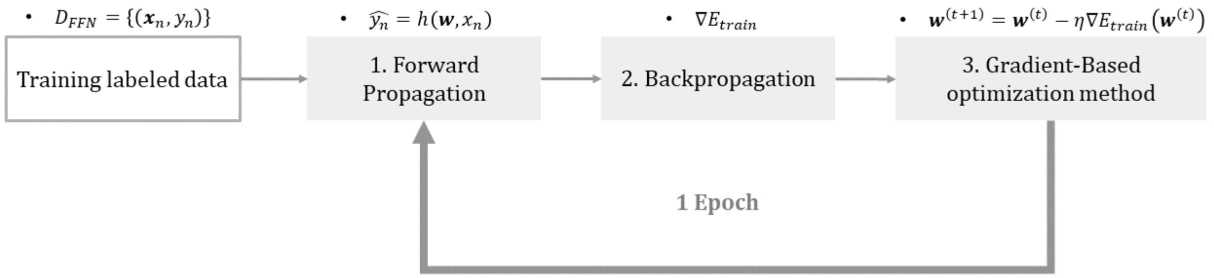

• $D_{FFN} = \{(\boldsymbol{x}_n, y_n)\}$
Training labeled data
• $\widehat{y_n} = h(\boldsymbol{w}, x_n)$
1. Forward Propagation
• ∇E_{train}
2. Backpropagation
• $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta\nabla E_{train}\big(\boldsymbol{w}^{(t)}\big)$
3. Gradient-Based optimization method
1 Epoch

Figure 4.5: Basic blocks of an FNN training algorithm. One complete circle defines an epoch.

4.3.4 Activation functions

Activation functions, φ, mentioned under the Neuron model in subsection 4.3.1, play an essential role in an FNN architecture as they introduce non-linearity in the network. Nonlinearity is a desirable addition to the networks as linearly non-separable data can be classified. Further, the choice of an appropriate nonlinear activation function is important to the training performance as its derivative will be used by the backpropagation and optimization algorithms. As was previously described, nonlinear activations are applied to the inputs of the hidden layers and the output layer of an FNN. Usually, the activation function used in a hidden layer differs from the activation function used in the output layer.

In the case of the hidden layers, the activation function is chosen accordingly to improve the neurons' training performance. A popular choice is the "s" shaped functions like the *hyperbolic tangent*, tanh [131][132]:

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.35}$$

The hyperbolic tangent was the most popular choice for FFN and multi-perceptron networks until the *Rectified Linear Unit* (ReLU) were introduced as an activation function [133] and become the default recommendation to use. ReLU is a piecewise linear function that outputs zero if the input is negative while retaining the input otherwise:

$$\varphi(z) = \max\{0, z\} \tag{4.36}$$

ReLU function is shown to have better performance, and it is easier to train using gradient-based optimization techniques [43]. It also appears to overcome a limitation when using a sigmoid function or the tanh in deep architectures, known as the *vanishing gradient problem*. Vanishingly small gradients will prevent efficient changes in the weight, making it difficult to find the correct direction to move so that the cost function is optimized [43]. The gains of using ReLU have led to the development of variations like the *Leaky ReLU* [134] or the *Exponential Linear Units* (ELU) [135], aiming for further training performance improvements. Figure 4.6 illustrates the most popular activations used in FFNs mentioned above. However, finding the optimum activation function for a hidden layer is a tedious process that relies on experimentation. While ReLU is a good choice for the hidden layers in deep architectures in most cases, Leaky ReLU is preferred when inactive neurons exist in the layer. A common practice is to start with ReLU, evaluate the results, and move on to other activations if performance is not satisfactory [136].

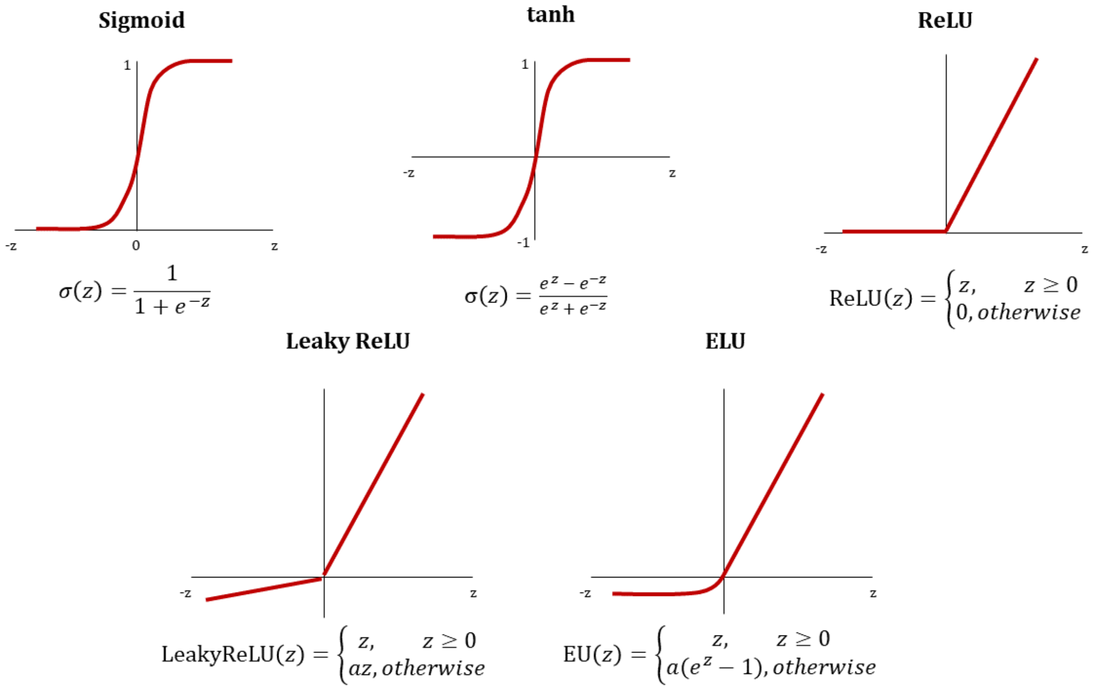


Figure 4.6: Popular activation functions used in FFN hidden layers.

For the output layer case, the activation function is chosen according to the classification type, i.e., categorical or binary, to filter the obtained results. The sigmoid function (equation (4.20)) can be applied for binary classification only, while in the case of two or more classes, *softmax* is used to represent the probability distribution over k different classes [43]:

$$\mathrm{softmax}(z)_i = \frac{\exp{(z_i)}}{\sum_{j=1}^{n}\exp{(z_j)}} \tag{4.37}$$

Since softmax is more generic and applies to binary classification as well, it is most preferred.

4.3.5 Weights' initialization

Weights' initialization is the process followed to set the values of the initial weights that will be used by forward propagation prior to training. It can be viewed as a preprocessing step that defines the starting point of training regarding the weights parameter. This starting point may determine whether the training algorithm converges and how quickly it will converge [43] and further prevent vanishing gradient problems from happening in deep architectures. In general, it is preferable to initialize the weights randomly rather than setting them to a fixed value or zero when using stochastic training algorithms like SGD [43]. Several publications show effective ways to initialize weights like in [22] and [23], and the choice of the proper method depends on the activation function used in the hidden layers. For example, the *Xavier (Glorot) Normal Initialization* [139] is a popular choice when using hyperbolic tangent, while for ReLU is the *He Normal Initialization* [140]. An overview of weight initialization methods for FFN is given in [141].

4.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are FNNs that involve the convolution operation in their architecture. Predictions are made following a forward propagation algorithm that performs convolution instead of multiplication in some layers, while training is performed using a backpropagation algorithm. CNNs overcome an important restriction of FNNs that is the 1D inputs. In CNNs, the inputs can be 1D, 2D, or 3D arrays. This allows the use of images that made CNNs very popular in various machine vision applications and other domains that use images as data (e.g., Biology, Remote sensing, Medical images. CNNs were the reason behind the boost of artificial neural network developments that is now known as Deep Learning. In this chapter, the way CNNs learn from data is discussed.

4.4.1 Convolution

Convolution is a mathematical operation between two functions that expresses how one modifies the other. Denoted with $*$, the convolution between two continuous functions, $f(t)$ and $g(t)$, is defined as the integral of their product, having one of them reversed and shifted [142]:

$$f(t) * g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \qquad (4.38)$$

For discrete signals with samples $n_i \in \boldsymbol{n}$, the convolution between two signals will produce a third signal that is defined as the Cauchy product [143]:

$$s(n_i) = (x * h)(n_i) = \sum_{k=-\infty}^{+\infty} x(k)h(n_i - k) \qquad (4.39)$$

where $x[n]$ is the input signal, $h[n]$ is the *impulse response* and $s[n]$ is the convolved output signal. The impulse response depends on the *system* which receives the input signal $x(n)$, and is defined as the output of that system when used the Dirac function, $\delta(n)$, as input. If the system is a filter, the impulse response will be the *filter kernel*, and the output will be the filtered input signal [144]. There are four steps followed in computing equation (4.39) and are: 1) *fold* $h(k)$ to obtain $h(-k)$, 2) *shift* $h(-k)$ according to n_i to obtain $h(n_i - k)$, 3) *multiply* $h(n_i - k)$ by $x(k)$ and 4) sum all the values of the product sequence to obtain the output for $\boldsymbol{n}$.

For computer vision, the discrete convolution of equation (4.39) is modified accordingly for 2D or 3D arrays that are used to represent images. If I is a 2D input image, and f_{kernel} is also a 2D filter kernel then the convolved image, S, is [43]:

$$S(i,j) = (I * f_{kernel})(i,j) = \sum_{m}\sum_{n} I(m,n) f_{kernel}(i-m, j-n) \tag{4.40}$$

or

$$S(i,j) = (f_{kernel} * I)(i,j) = \sum_{m}\sum_{n} I(i-m, j-n) f_{kernel}(m,n)$$

due to the commutative property. The type of the filter, f_{kernel}, in the convolution equation, modifies the input image. The filters are designed for a particular processing purpose. It can be either to eliminate or to enhance elements of the input image. For example, linear smoothing filters are used to remove random noise, while *Fourier filtering* removes noise at certain frequencies. Other filters are designed to identify certain *image features*. Image features are the important attributes of the image's regions (i.e., texture) and their boundaries (i.e., edges or lines), which define an object [145]. Thus, convolution can be viewed as a way to apply feature transforms to identify them through the usage of filters. By identifying the important features, tasks like *classification*, *segmentation*, or *object detection* can be performed either in a single image or among a group of images.

In Deep Learning applications, the central idea is to use the convolution operation to transform a neural network's input into a useful representation for the learning task. In practice, what is actually implemented is not convolution but the *cross-correlation function*, as explained in [43], with the cross-correlation function being the same as convolution but without the reversal of the filter:

$$S(i,j) = (\mathrm{I} * f_{kernel})(i,j) = \sum_{m}\sum_{n} I(i+m, j+n) f_{kernel}(m,n) \tag{4.41}$$

The flipping step turns out to be redundant for the learning process and the training of the algorithm, as explained in [43]. However, the process is referred to as convolution in most ML and DL libraries. In this work, this convention is adopted as well. Convolution is introduced in some CNNs layers in place of matrix multiplication and is called *convolution layers*. This allows the use of varying size inputs and not only 1D arrays. Further, the filter kernels that are applied to the input are of much smaller size, resulting in identifying the important features while having dimensionality reduction for more efficient computations and memory handling. The typical architecture of CNNs is described in the following paragraph.

4.4.2 CNN architecture

CNNs can receive as input images represented by arrays. If the image is black and white, then the array is 2D, with dimensions defined by the height and the width of the image (HxW). For colored images, the array is 3D (also called tensor) with dimensions (HxWxC), where C is the color model, e.g., the three RGB channels. The array is passed through a sequence of layers that perform a particular processing operation; each of them is depicted schematically in *Figure 4.7***.** There are four main layers found in a typical CNN architecture. These are the *convolutional layer*, the *pooling layer*, the *flattening layer*, and the *fully connected* or *dense layer*. The convolutional and pooling layers are responsible for feature detection, while the dense layers are responsible for the classification of the identifying features.

- **Convolutional layer**

The convolutional layer is usually the first layer in a CNN architecture and receives as an input a colored image. Color images are three-dimensional, having the size HxWx3, where number three indicates the three RGB channels. Then convolution is applied (equation (4.41)) for a number of different filters that act as feature detectors. Each convolution act will produce a filtered version of the input image called the *feature map.* The convolution operation is performed by sliding the filter on the input image using a fixed pixel step called the *stride*. Hence, the dimensions of the output feature map depend on the size of the filter and the stride. Bigger filter sizes and strides will result in a more drastic dimensionality reduction. So, a feature map can be viewed as a shrink version of the image that retains the features with a higher correlation to the ones indicated by the convolutional filter. This process is applied to all the filters designed for the given layer. So, the output of the convolutional layer is a 3D array that stacks all the feature maps obtained from all the filters used. The filters are designed automatically by the training algorithm during the optimization process to improve the classification results [43], [146]. There can be more than one convolutional layer in a CNN architecture, as will be discussed later in this chapter.

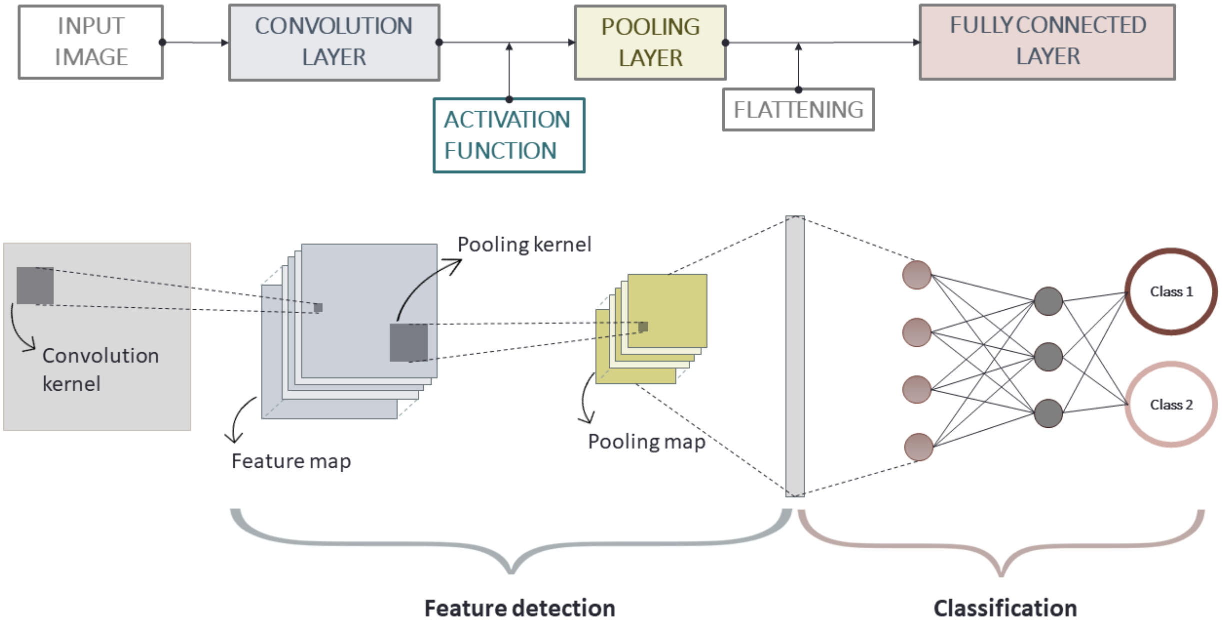


Figure 4.7: Main components of a typical CNN architecture (Top) with their graphical description (bottom). The overall architecture can be divided into two stages, the feature detection stage, and the feature classification stage. The feature detection stage usually has an arrangement of convolutional and pooling layers along with the application activation functions to increase non-linearity. The classification stage is an FNN. Between the two stages, a flattening step is implemented that vectorizes the input for the FNN.

An additional step that is implemented in most CNN architectures is the application of a non-linear activation. It is usually applied to the convolutional output to ensure non-linearity. Non-linearity increases classification performance as more complex and non-linear features can be identified. In most CNN architectures, the ReLU function is used [43]. Goodfellow et al., in [43], mention this step as the *detector stage* and usually is applied for every convolutional layer that exists in a CNN architecture. However, studies like the one in [142] debate the efficiency in applying ReLU in every convolutional layer for deep architectures. Other studies examine performance improvements using alternative activations of ReLU like the concatenated ReLU (CReLU)[147] or the Average Biased ReLU (AB-ReLU) [148]. In any case, these modifications are task-oriented, which indicates that experimentation is necessary when it comes to performance improvements.

- **Pooling layer**

Pooling is a down-sampling operation. It usually is placed after the convolutional layer, hence, it receives as inputs the feature maps. Pooling operations are performed by using kernels that slide on the input for a fixed stride. Two types are mostly used, the *max-pooling* and the *average pooling*. In max pooling, the max value of the region covered by the kernel is only kept. In average pooling, the average of the values of the region placed within the kernel is calculated and kept. Between the two, max pooling is the most popular choice [43]. The comparative study in [149] also suggests better performance when max pooling is used in CNN architectures for object detection. The output of a pooling operation is the *pooling maps*. Thus, a pooling map retains the important features or part of them while further reducing the dimensions for a more efficient computational cost. The pooling operation emphasizes whether a feature exists rather than where it appears in the image. This is a property called *invariance to local translation* [43] and is beneficial for feature classification as it adds more flexibility in their detection. Typically, in CNN architectures, a pooling layer succeeded a convolutional layer. For deeper architectures with many convolutional and pooling layers pairs, the features of the image are learned in hierarchical order according to the space they occupied in the input image [43]. An alternative to the pooling layer is to use a convolutional layer of larger stride.

- **Flattening layer**

Flattening is an intermediate step between the feature detection stage, that is, the convolutional and pooling layers and the fully connected layers that perform classification. The latter requires the input to be a vector. Thus, the flattening step converts the 3D array, resulted from the last layer of the feature detection stage, into a 1D array (vector) that will be the input of the FNN that follows.

- **Fully connected layers**

Fully connected layers describe the layers of an FNN design to perform feature classification, which is the last processing stage in a CNN architecture. A fully connected layer is also called the *dense layer*. The FNN receives as input the 1D array that includes all the important features identified during CNN's feature detection stage. It performs feature classification by following the forward propagation process described earlier in this chapter, i.e., using shared weights and biases and applying non-linear activation functions (equations (4.26)-(4.29)).

4.4.3 Training CNNs

The training of CNNs is performed within the same context as the FNN that was described earlier. That is, using the forward propagation algorithm to make a prediction for some initiated weight values and then update these weights to minimize the cost function. The weights are updated by using the backpropagation algorithm, which calculates the gradients of the errors and then a gradient-based optimization technique to minimize them. The particularity of CNNs lies in the layers of the feature detection stage that receive 3D arrays instead of vectors and are using operations like convolution and pooling instead of matrix multiplication. Here, a neuron or unit in a layer is a pixel that is connected with a previous layer using the concept of *receptive fields* or *field of view*.

Receptive fields, like the neurons, are models that are also biologically inspired by the visual system of animals and are regions that can trigger neuronal responses when stimulated [150], [151]. In deep learning, a receptive field is a block of neurons in the input that affects a neuron in the next layer [152], [153]. LeCun et al. in [43] named these regions as *filter banks*. Compare to the typical FFNs, the neurons of a layer in CNNs' feature detection stage are not fully connected. The convolutional filter or pooling filter defines the receptive field's size, and the weights are introduced by the type of filter used. The training process will update these values. The equations described in the feedforward and backpropagation algorithm can be applied. A step-by-step derivation using backpropagation in a CNN with two convolutional layers, two pooling layers, and a fully connected layer is given in [154]. To update the weights, SGD is among the most popular choices of optimizers. More recent algorithms embed the *momentum method* to accelerate learning of SGD or use newer developed *adaptive learning rate optimization algorithms*. In the last category, many algorithms have been proposed to overcome issues related to the learning rate of SGD, like slow convergence. An overview of the most popular ones for the field of DL is given in [155]. For image classification with CNNs, the most preferred optimizers are the SGD algorithm using momentum and the Adaptive Momentum estimation (Adam).

- **Momentum**

Momentum, introduced by Polyak in [156], is a method used in DL to accelerate learning when using gradient-based optimizers. It is particularly efficient in cases of gradients that a local minimum is harder to approach either due to noise, high curvatures, or when gradients are small but consistent [43]. Momentum introduces velocity, v, in the gradient descent updating process (equation (4.13)) that helps in accelerating towards the direction of the local minima. If $J(\boldsymbol{w})$ is a cost function to be minimized with respect to the weights $\boldsymbol{w}$ and t the iteration, the momentum is given by [157]:

$$\boldsymbol{v}^{(t+1)} = \mu \boldsymbol{v}^{t} - \eta \nabla C(\boldsymbol{w}^{(t)}) \tag{4.42}$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \boldsymbol{v}^{(t+1)} \tag{4.43}$$

where $\eta > 0$ is the learning rate, $\mu \in [0,1]$ is the momentum coefficient (usually set to 0.9), and $\nabla C(\boldsymbol{w}^{(t)})$ is the gradient at $\boldsymbol{w}^{(t)}$. The velocity vector accumulates the gradient elements of the previous steps that point to the same direction. When μ and η are close, the gradients of the previous steps have a greater effect on the current direction [43]. The weights are updated using equation (4.43).

- **Adam**

Kingma and Ba introduced adaptive moments or "Adam" in [158] as an adaptive learning rate optimization algorithm that is based on SGD. According to the authors, Adam combines beneficial features of the *Adaptive Gradient Algorithm* (AdaGrad) [159] and the *Root Means Square Propagation* (RMSProp) [160]. These are the efficiency of issues related to sparse gradients (like in Adagrad) and the efficiency with online and non-stationary problems (like in RMSProp). In Adam, the adaptive learning rates are computed by storing an exponentially decaying average of both the past gradients, $m^{(t)}$, and past squared gradients, $v^{(t)}$ [158]:

$$\boldsymbol{m}^{(t)} = \beta_1 \boldsymbol{m}^{t-1} + (1 - \beta_1)\nabla C(\boldsymbol{w}^{(t)}) \tag{4.44}$$

$$\boldsymbol{v}^{(t)} = \beta_2 \boldsymbol{v}^{t-1} + (1 - \beta_2)(\nabla C(\boldsymbol{w}^{(t)}))^2 \tag{4.45}$$

The $m^{(t)}$ and $v^{(t)}$ are estimates of the first moment and the second moment of the gradient, respectively, and β_1 and β_2 are the corresponding decay rates. The authors in [158] observed that the moments of equations (4.44) and (4.45) are biased to 0 for low decay rates. This led them to include bias-corrected estimations of the first and second-moment:

$$\widehat{\boldsymbol{m}} = \frac{\boldsymbol{m}^{(t+1)}}{1 - {\beta_1}^{(t+1)}} \tag{4.46}$$

$$\widehat{\boldsymbol{v}} = \frac{\boldsymbol{v}^{(t+1)}}{1 - {\beta_2}^{(t+1)}} \tag{4.47}$$

The weights are updated according to [158]:

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \frac{\widehat{\boldsymbol{m}}}{\sqrt{\widehat{\boldsymbol{v}}} + \epsilon} \tag{4.48}$$

where ϵ is a small scalar (i.e., 10^{-8}) to prevent division by 0. The recommended values for β_1 and β_2 are 0.9 and 0.999 according to [158].

The choice of the best optimizer is mostly based on experimentation. Several studies exist in the DL literature, comparing the performance of various optimizers providing empirical guides. For classification with CNN, many of these studies conclude that Adam performs better than similar algorithms like RMSProp and Adagram [161],[158]. However, there are also studies, like the one conducted by Wilson et al. [162], which conclude that SGD is a better choice than adaptive rate algorithms as a better generalization is achieved.

4.5 GENERALIZATION

The essence of learning from data in every ML and DL applications is that the final hypothesis, $h(\boldsymbol{x})$ resulted from the training process will be able to make correct predictions when using entirely new inputs from $\mathcal{X}$. That is the meaning of *generalization*. Thus, an important part of every ML and DL training session is to examine whether $h(\boldsymbol{x})$ generalizes well. If this is true, then the output, $h(\boldsymbol{x})$, is the best approximate of the unknown target function or distribution and the solution to the learning problem. Else, training should be repeated using different $\mathcal{H}$ (i.e., find different weights).

To evaluate generalization, $h(\boldsymbol{x})$ is used to make predictions using a *test set* and then measure the error between the real and the predicted value, E_{test} [118]:

$$E_{test} = \mathcal{E}\big(h(\boldsymbol{x}^{\boldsymbol{test}}) \neq f(\boldsymbol{x}^{\boldsymbol{test}})\big) = \mathcal{E}(\hat{y}^{test} - y^{test}) \tag{4.49}$$

This error is an estimation of the error that will occur when using any data drawn from $\mathcal{X}$, or any data that were generated by the unknown distribution $P(y|\boldsymbol{x})$ [118]. An indication that the picked $h(\boldsymbol{x})$ generalizes well, is when the gap between E_{test} and E_{train} is minimum. Ideally, it would be the same. Hence, setting an appropriate test set is a major step towards the evaluation of how well a model has learned, and further, how much the model's predictions using new data can be trusted. Usually, the test set is defined by splitting the data that will be used for training. The size of the test set, i.e., the number of the examples used, must be big enough for the error to be representative for the entire $\mathcal{X}$, but at the same time, not too big as it will reduce the number of examples available for training. An empirical rule is to split training and test sets using the ratio of 80% -20%, so having enough data is essential for learning. Additionally, the portion of data kept for the test set must be entirely intact from the training process.

An important property of $\mathcal{H}$ that is used to evaluate generalization is the *capacity* [43]. The capacity describes the size and complexity of $\mathcal{H}$. It is quantified using the *Vapnik-Chervonenkis dimension* (VC), denoted as d_{VC}. The VC dimension is used to apply an upper boundary on E_{test} that shows the effective number of parameters to keep the distance close to E_{train}. This boundary is called the VC generalization boundary [118]. Briefly, it expresses that good generalization, from E_{train} to E_{test}, is possible for an infinite $\mathcal{H}$ and finite d_{VC} when using enough training data N. A rule of thumb is $N = 10 \times d_{VC}$. On the other hand, the gap between the two errors grows as the model capacity grows. The latter agrees with the *Occam's razor* principle stating that among competing hypotheses that explain a known observation equally well, the simplest one should be chosen [43]. Thus, to have chances for good generalization, the following terms must be considered [43], [130]:

1. An efficient amount of training data, N, is required
2. E_{train} be as close to zero as possible
3. E_{test} and E_{train} gap be minimum
4. The capacity of the model be simple enough to keep the gap small
5. The capacity of the model be complex enough to keep E_{train} small

Another approach that may give insights on the generalization is the *bias-variance tradeoff*. It mainly applies to real-valued functions and is based on minimizing the square error of E_{test} and E_{train}. This is translated as a tradeoff of minimizing either the bias or the variance. Since this analysis is not practical for classification, as discussed in [113] and [115], it is not further discussed as it exceeds this dissertation's scope.

There are two behaviors observed during the training process of a learning algorithm, known as *overfitting* and *underfitting*, leading to poor generalization.

4.5.1 Overfitting

Overfitting means that the hypothesis, $h(\boldsymbol{x})$, fits the training data so well that it results in poor generalization. In other words, the error during training is very small, but the test error is very large. So, the gap between E_{test} and E_{train} is also large. Abu Mostafa et al. in [118] describe overfitting as the process of picking those hypotheses from $\mathcal{H}$ that keep reducing E_{train} while keep increasing E_{test}. There are a few obvious reasons that can cause overfitting. One is the noise level in training data. High noise level leads to overfitting, as the learning algorithm will try to reduce the error by picking a hypothesis to fit the noise or outliers as well. Another reason is that the hypothesis space is very complex, or the capacity is very high. Under this condition, overfitting is more likely to happen as it increases the chances of a hypothesis that is not a good approximate of f, that is $h \neq g$, to better fit the data. A graphical example of overfitting is presented in *Figure 4.8*.

Overfitting is a challenging issue, and an open research subject commonly met in several Machine and Deep Learning problems. Overfitting can be limited or even prevented when using regularization techniques (will be discussed in the following subsection) and when increasing the size and quality of training data.

4.5.2 Underfitting

Underfitting is the opposite of overfitting. A sign of underfitting is when E_{train} is not sufficiently low. That means the hypothesis picked by the learning algorithm does not adequately fit the data. Underfitting happens when the hypothesis set is not complex enough to capture the underlying structure of the data. Thus, the capacity is low. For example, using a simple linear $h(\boldsymbol{x})$ to describe points of a polynomial of a very high order. A graphical example is presented in *Figure 4.8*. As a result, h will not generalize well.

Thus, the real challenge is to find a hypothesis complex enough to avoid overfitting, underfitting, and at the same time to achieve a good generalization. One way to find this optimal capacity is graphical by observing the curves of the errors E_{test} and E_{train} when plotted with respect to the capacity (*Figure 4.9*).

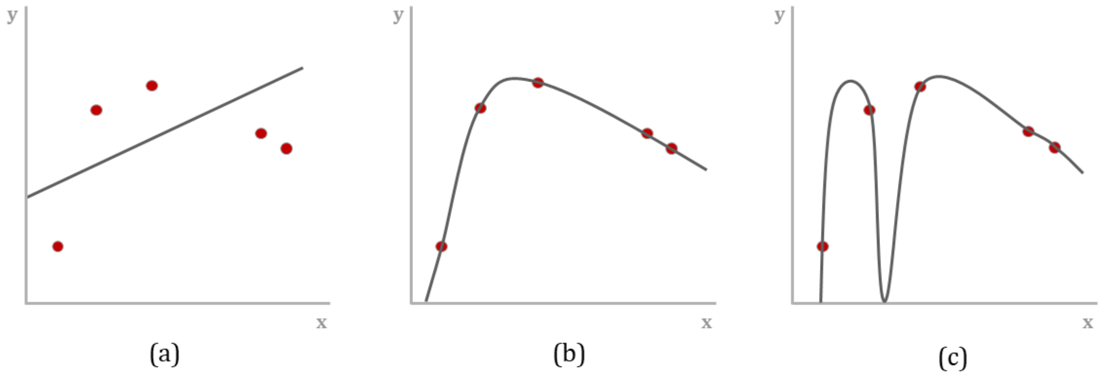


Figure 4.8: Example of a data set and three hypotheses where (a) is the underfitting scenario, (b) is the best fit scenario, and (c) is overfitting. (Concept recreated from [43])

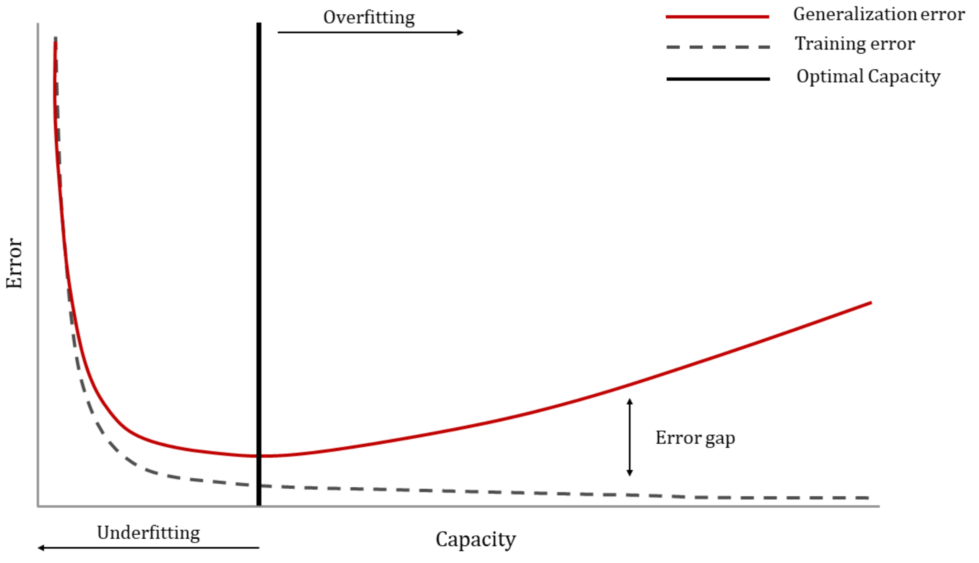


Figure 4.9: Behaviors of the training and test error with the capacity that can lead to overfitting and underfitting. The Underfitting zone lies in lower capacities. In the overfitting zone, the training error decreases as capacity increases while the test error increases. The optimal capacity lies in where both errors are low, and their gap is the minimum. (Concept recreated from [43]).

4.6 Regularization

The purpose of regularization is to prevent overfitting and achieve better generalization. In a more general sense, Goodfellow et al. in [43] describe regularization as any modification made to a learning algorithm that is intended to reduce its test error but not its training error. Kukačka et al. [163] categorize the various regularization methods that are used the most in the machine and deep learning problems. It becomes clear that regularization is a combination of different techniques that are being embedded in the different stages and components of the learning process. According to their taxonomy, regularization can be achieved: 1) by applying transformations on the training data, 2) by choosing a proper network architecture, 3) through the error function chosen during training, 4) through the addition of an explicit regularizer, and 5) via the optimization during training. It is also very common to combine different regularization techniques to achieve as good a generalization as possible. For this, Abu-Mostafa et al. in [118] characterize regularization *as much an art as it is a science*. It can make a difference, but that would depend on skills and experience. For the purpose of this study, regularization techniques used with CNN are examined. The most popular ones are *data augmentation*, *weight decay*, *dropout,* and *batch normalization*.

4.6.1 Data Augmentation

A way of achieving good generalization is using an adequate number of training data that represent the input space. Data augmentation assists in that direction by creating new data and add them to the training set. This is done by applying various transformations in the training set. Data augmentation is efficient for classification tasks like image classification for object recognition, particularly, when the available data are insufficient to train accurate and robust classifiers.

The transformations applied in these cases are label-preserving like small affine transformations [164]. The authors in [165] mentioned seven families of transformations proper for image classification. These are:

- *flip* that mirrors an image along the horizontal or vertical axis
- *crop* that produces sub-windows of the input image
- *homography* that changes the viewpoint of the input image
- *scale*, that changes the scale of the input image
- *colorimetric transformations* that create variations of the RGB color scheme of the input image using the covariance matrix
- *JPEG compression* that creates variations on the input image encoding
- *rotation* that rotates the input image around its center

- *Order-K Transformations* that refer to the number K of combinations of the previously mentioned transforms.

The purpose of applying these transformations, either standalone or in a combination, is to ensure that the target will be detectable despite differences related, for example, to the angle that the image was taken and the lens distortion, or the orientation and the size of the feature of interest, the contrast in colors or tones, etc. In other words, there are target-preserving transformations to mimic natural transformations [163]. Another data augmentation approach is to inject random noise on the inputs, for example, to improve the robustness of neural networks. Augmentation techniques vary significantly and usually are chosen manually according to the learning needs. Data augmentation is known to have contributed to achieving state-of-the-art results on the various machine and deep learning tasks related to computer vision. However, bad choices in the data augmentation scheme can lead to a detrimental impact on the accuracy and the robustness of the classifier. Thus, it is a strategy that requires good planning. Two algorithms that are developed to find the best possible applicable transformations for image classification are the adaptive data augmentation in [164] and the Image Transformation Pursuit in [165].

4.6.2 Weight Decay

Weight decay is a traditional technique that falls under the category of adding an explicit regularization term when following the taxonomy in [163]. This term assigns a penalty denoted by, $\mathit{\Omega}$, and is called a *regularizer*, denoted by $\mathscr{r}$. If N is the to total number of the data points, the regularizer has the general form of [118]:

$$\mathscr{r} = \frac{\lambda}{N}\mathit{\Omega} \tag{4.50}$$

where $\lambda \geq 0$ is a value that controls the amount of regularization. The regularizer is independent of the inputs and the targeted values, while it depends on parameters related to the hypothesis space and the number of the data points. For less training data, more amount of regularization is needed.

In weight decay, the penalty is the squared norm, L^2, of the weights: $\mathit{\Omega}(\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$. So, equation (4.50) becomes:

$$\mathscr{r}(\boldsymbol{w}) = \lambda\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} \tag{4.51}$$

The regularizer, $\mathscr{r}(\boldsymbol{w})$, is added to the loss function of equation (4.6), defining the *augmented error* [118]:

$$E_{aug} = E_{train}(\boldsymbol{w}) + \lambda \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} \tag{4.52}$$

Minimizing the augmented error becomes the new subject of training when using weight decay. To do so, both terms in equation (4.52) must be minimum. Thus, it enforces E_{train} to become small when weights in $\boldsymbol{w}$ are small. In other words, weight decay penalizes large weights as λ becomes larger. In a way it reduces the size or the capacity of the hypothesis space, and this may prevent overfitting. Weight decay is also known as L^2 regularization, *ridge regression,* and *Tikhonov regularization*.

4.6.3 Dropout

Dropout is one of the most popular methods and has many variants that are proposed over the years for improvements [163]. Srivastava et al. initially introduced dropout in [166] as a technique to prevent neural networks from overfitting with a low computational cost. Deep neural networks are capable of learning complicated relationships between their inputs and outputs due to their multiple non-linear hidden layers. This ability tends to cause overfitting as noise usually exists in the training data that is mistreated as data drawn from input space. Models of these relationships will fail to generalize when applied to the test set. A solution to this issue would be to combine different models with all possible parameters and average their predictions. However, this is not applicable as it would require unlimited computations, while dropout is a feasible approach.

The concept of dropout is to exclude units in both hidden and input layers and their connections while training a neural network. The choice of the units is random, using a probability to be “dropped out,” p. This probability is independent among the units. Srivastava et al. [166] recommend a value $p = 0.5$ for units inside the hidden layer, and p closer to 1 for units inside the input layers. Thus, the outcome of dropout will be a number of “thinned” versions of the original network. The concept of dropout is illustrated in *Figure 4.10*. If the original neural network consists of n number of units, then 2^n possible thinned neural networks may result from it. So, dropout is about sampling the 2^n thinned neural networks and then proceeding with their training using weight sharing and an approximate averaging method. This is performed by dividing the process into training time that uses dropout and test time without dropout. During training time, a unit of the network might be present with some probability p and will produce some weights $\boldsymbol{w}$. At test time, the same unit is always present since no dropout is used, and its produced weights $\boldsymbol{w}$ are multiplied by the probability p.

The dropout neural network model is described in comparative *Figure 4.11*. On the left is a neural network where L are the hidden layers with index $\ell \in \{1, \dots, L\}$, $z^{(\ell)}$ is the vector of inputs passed to layer l, $y^{(\ell)}$ is the vector of outputs from layer l ($y^{(0)}$ will be the input x). Following the feedforward propagation, with weights $\boldsymbol{w}^{(\ell)}$ and biases $\boldsymbol{b}^{(\ell)}$ of each layer ℓ, the input of each layer $z^{(\ell)}$ and corresponding output $y^{(\ell)}$ will be:

$$z_i^{(\ell+1)} = \boldsymbol{w}_i^{(\ell+1)}\boldsymbol{y}^{(\ell)} + b_{\boldsymbol{i}}^{(\ell+1)} \tag{4.53}$$

$$y_i^{(\ell+1)} = \varphi\left(z_i^{(\ell+1)}\right) \tag{4.54}$$

where φ is an activation function.

With dropout, the forward propagation will be [166]:

$$r_j^{(\ell)} \sim Bernoulli(p) \tag{4.55}$$

$$\widetilde{\boldsymbol{y}}^{(\ell)} = \boldsymbol{r}^{(\ell)} * \boldsymbol{y}^{(\ell)} \tag{4.56}$$

$$z_i^{(\ell+1)} = \boldsymbol{w}_i^{(\ell+1)}\widetilde{\boldsymbol{y}}^{(\ell)} + b_i^{(\ell+1)} \tag{4.57}$$

$$y_i^{(\ell+1)} = \varphi\left(z_i^{(\ell+1)}\right) \tag{4.58}$$

The equation (4.55) expresses a vector of independent Bernoulli random variables that correspond to each layer ℓ, and has a probability p of being 1. This vector is element-wise multiplied with the corresponding outputs of layer ℓ. In this way, the thinned outputs $\widetilde{\boldsymbol{y}}^{(\ell)}$ are created (equation (4.56)) and will be the inputs to the next layer, $z_{\boldsymbol{i}}^{(\ell+\mathbf{1})}$**,** as describes equation (4.57). This process is applied to each layer.

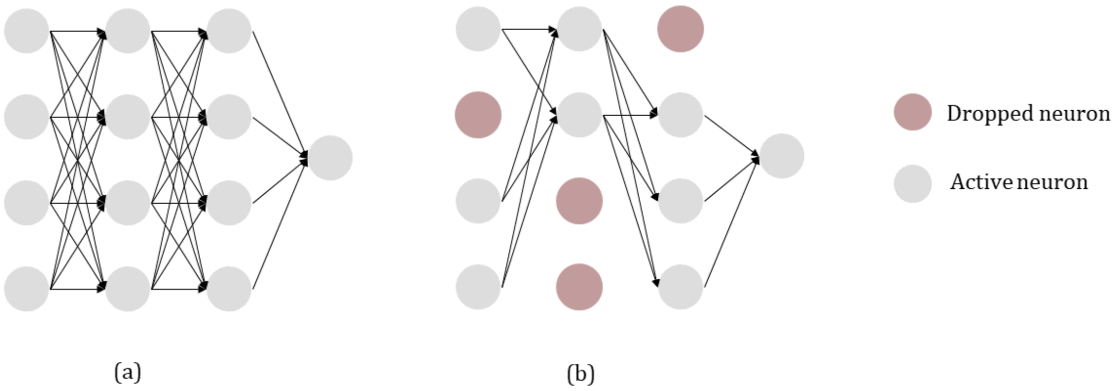


Figure 4.10: The concept of Dropout where (a) is the original network training and (b) is training with dropout applied (Concept redrawn from [166]).

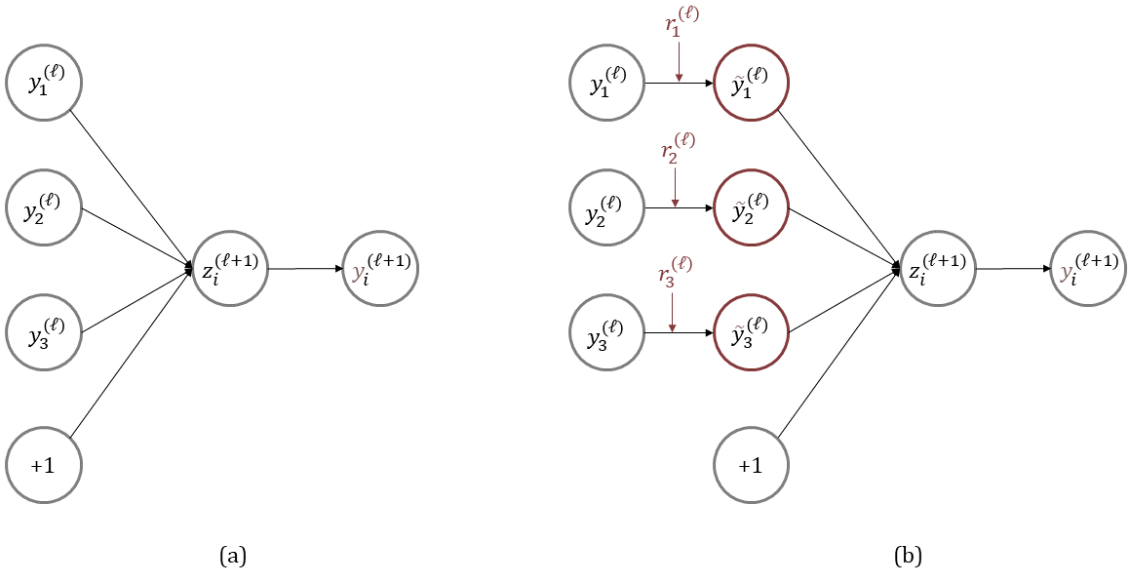


Figure 4.11: Dropout model where (a) is the original neural network while (b) is the network with dropout applied (Concept redrawn from [166]).

4.6.4 Batch normalization

Batch normalization (BN) is a technique proposed by Ioffe and Szegedy in [167], and while its primary purpose is to accelerate training in deep neural networks, it acts as a regularizer as well [168]. When using SGD, in deep neural network architectures, training times are very long as the distribution of each layer's inputs is sensitive to changes in the parameters of the previous layer (i.e., the learning rate of a batch). Moreover, any small change is amplified as the layers of the deep network proceed. Ioffe and Szegedy defined these changes in distribution at a network layer level as the *Internal Covariate Shift* [167]. Acceleration in training is then feasible if the internal covariate shift is reduced, which can be achieved by normalizing each layer's inputs. Normalizing a neural network's input is a common practice in many ML and DL applications as the non-linear activation functions (i.e., tanh or the sigmoid) are less saturated for this distribution range. As a result, the optimizer converges faster, which leads to faster training times. Batch normalization is a mechanism that embeds normalization in the architecture of a neural network and is applied for each activation layer's input.

Like dropout, BN operates differently for training and test time. For training time, BN is applied to the batches of data used by SGD. If $\mathcal{B} = \{x_{1,} \dots, x_m\}$ are the input values of the batch, the batch-mean, $\mu_{\mathcal{B}}$, and batch-variance, $\sigma_{\mathcal{B}}^2$, are calculated as[167]:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{4.59}$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \tag{4.60}$$

and the batch inputs are normalized, $\langle x_i \rangle$, as:

$$\langle x_i \rangle = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{4.61}$$

where ϵ is a small value to prevent division by zero.

Then, a scale, γ, and a shift, β, are applied to the normalized value $\langle x_i \rangle$:

$$y_i = \gamma\langle x_i \rangle + \beta \ \equiv BN_{\gamma,\beta}(x_i) \tag{4.62}$$

where y_i is the input to the activation at a single layer for the current batch. The pair of parameters γ and β are trainable and are updated during the training process, along with the rest of the parameters like the weights, using backpropagation, and SGD or any other extension of it (e.g., Adam). Equations (4.59)-(4.62) are referred to as the Batch Normalizing Transform [167].

During test time, where no batches are used, predictions are made for some new inputs using the trained model. If x is the input for the layer's activation, it is normalized as:

$$\langle x \rangle = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} \tag{4.63}$$

where the mean, $\mathrm{E}[x]$, and variance, $\mathrm{Var}[x]$ are the expected values collected from the batch training:

$$\mathrm{E}[x] = \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}] \tag{4.64}$$

$$Var[x] = \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}\left[\sigma_{\mathcal{B}}^2\right] \tag{4.65}$$

BN algorithm can be implemented prior to layers that use activation functions. For example, in FNNs, BN is applicable prior to any hidden layer. For CNNs, BN can be implemented between convolutional layers, using as inputs the feature maps that the nonlinear activation function will be applied (i.e., the ReLU step) or before a dense layer in the same manner as an FNN. However, it is more common for BN to be applied between convolutional layers rather than before dense layers. Aside from the gains in training time, BN has other benefits like acting as a regularization technique and, in some cases, eliminates the need for dropout[167], [168]. It also eliminates the need for a bias term in the activation input.

4.7 Validation

Throughout the various stages of the learning process discussed in previous subsections, one can realize that achieving good performance relies on experimentation upon the choices made for parameters and other settings during these stages, from designing an architecture to training and generalization. These settings and parameters, which are not learned using a learning algorithm but have a great impact on learning, are referred to as the *hyperparameters*. For example, in neural network architectures, hyperparameters can be the number of hidden layers, the number of units in a layer, and the choice of activation and cost function. Hyperparameters are also parameters of the training algorithm, like the learning rate or the momentum, as well as parameters related to the applied generalization methods like the dropout value.

Validation is an additional step in learning from data that aims to optimize or tune the hyperparameters that control the learning process for better performance. For this purpose, a *validation set* is used. Similar to the test set, the validation set is a subset of the training set that is excluded from the training process. The difference to the test set is that the validation set is used to make choices while training. After a hypothesis, h, is generated from the learning algorithm, using the initial choices of the hyperparameters, it is then evaluated on the validation set by calculating the *validation error* [118]:

$$E_{val} = \frac{1}{K}\sum_{i=1}^{K} cost\left(\widehat{y_i}^{val}, y_i\right) = I\left(\widehat{y_i}^{val} \neq y_i\right) \tag{4.66}$$

where $cost$ is expressed as the binary error for classification and $\widehat{y_i}^{val}$ is a prediction made using the hypothesis $h(\boldsymbol{w})$ with the samples in the validation set. The model's hyperparameters are then tuned manually or by using *meta-algorithms* and techniques, and training is repeated using the new choices [43]. For the new model E_{val} is calculated. Having repeated this process producing several models, the one with the lowest E_{val} is selected. The final evaluation of the model and its generalization is performed on the test set. Because the validation set includes samples withdrawn from the training set, it has to be small enough to refrain from significantly reducing the training examples and big enough to be representative for a good evaluation. A rule of thumb is to set 20% of the training data for validation [118]. A popular validation approach for splitting the data is *cross-validation*.

While the validation set can have various uses in the ML and DL application range, in neural networks, it is mostly used for applying *early stopping,* which helps to avoid overfitting by terminating training. Further tuning is mainly performed manually, focusing on certain hyperparameters. Automated hyperparameter tunning is challenging due to the complexity and the large number of parameters that FNNs and CNNs can have, especially when generalization techniques are also involved. When the hyperparameters under tuning are three or fewer, methods like the *grid search* or *random search* are used [43].

4.7.1 Cross-Validation

Splitting the available training data into a test set and validation set requires enough data for the learning process to be successful. In cases where the data are not enough, even if the rule of 80%-20% is followed, the test set will end up being small introducing statistical uncertainty in the test error estimation. This case scenario is met very often in real-world problems, and approaches are developed to reduce this uncertainty in the test error estimation. The concept behind this approach is to estimate the test error on different randomly chosen subsets of the original dataset. One of the most popular techniques is the k-fold cross-validation. The k-fold cross-validation algorithm splits the dataset in k non-overlapping subset; each one corresponds to a trial. For each trial, the given subset is used as a test set and the rest for the training set. The E_{test} is estimated as the average of k-trials.

Cross-validation can also be used to choose a model or the proper hypothesis set, among others [118]. The test error is estimated for each model, and the one with the smallest cross-validation error is chosen for the training process.

4.7.2 Early stopping

Early stopping is a simple, yet effective technique applied to ML and DL problems to prevent overfitting. As was described in generalization, during the training of the model, the curve of the training error decreases over time (epochs) as the model tends to overfit the training data. A way to understand when the model starts to overfit (*Figure 4.9*) is to observe the curve of generalization error as well. An indication of overfitting is when the generalization error starts to increase. Early stopping aims to find the time or epoch that the model starts to overfit and terminate training. In other words, it aims to determine the right number of epochs to train the model.

Early stopping can be seen as both a generalization technique and a hyperparameter selection meta-algorithm [43]. For the latter, the hyperparameter under question is the number of epochs that are tuned using the validation set. In this case, the curve of the validation error is observed. A common practice is to evaluate the validation error periodically during training while keeping copies of the best model. A benefit of early stopping is that after finding the best training time, training can be repeated, including the samples in the validation set as well. It can also be used alongside other regularization methods.

4.8 Closing remarks

In this chapter, the fundamental concepts of learning from data using CNN were described revealing their depth and complexity. Since a recipe that guarantees good results does not exist, the trial and error approach is inevitable. A starting base to produce reliable predictive models requires:

- a well-defined learning problem,
- enough and qualitative data to support it,
- a well-designed CNN architecture for the task (i.e., number of convolutional layers, pooling layers, dense layers, units in a layer, size of kernels, activation functions, etc.),
- an efficient training algorithm for the given architecture (i.e., SGD, employing momentum, Adam, etc.).

Choosing an appropriate performance measure (i.e., loss or cost functions) is important as it indicates the quality of training and how well the produced model generalizes when used new data. For cases where performance is poor or requires improvements, regularization techniques can be used to limit or prevent overfitting, a cause for bad generalization. Several options exist (i.e., weight decay, dropout, batch normalization), and more than one can be combined. This leads to many hyperparameters that are mostly defined empirically and might as well require tuning. Hence, validation approaches can be used to tune the hyperparameters, which can further improve performance and lower the learning process's computational cost. The successful application of generalization and validation techniques depends on how representative the samples included in the test and validation set are, highlighting the fact that in ML and DL, the more the data, the better.

5. RESEARCH METHODOLOGY

This chapter describes the methods and tools employed in this research that aims to identify in an automatic way ancient buried structures in GPR data. Toward this goal, Convolutional Neural Network (CNN) and their potential to classify GPR images exhibiting structures are examined. Since this research is among the first steps in using CNNs for interpreting GPR depth slices from archaeological prospection, it is designed to serve as a base, providing insights for further developments and improvements. This includes building datasets from scratch due to the lack of available ones for this research's needs and testing CNNs on a more fundamental level, rather than using more advanced and recent architectures, to understand what works better. For this reason, AlexNet was used as it is simple and has the benefits of a deep architecture.

The conducted research study can be divided into four major stages; data collection, data processing, dataset construction, and training using CNNs for the classification of selected features. Each stage is judged equally important in achieving learning from GPR depth slices. Learning would not be possible without GPR data featuring structures, while data processing helps create the right representations for dataset constructions. The latter are the inputs of CNNs and affect the training process. If the training dataset does not represent the input and output spaces well enough, then the produced models would be unreliable, and hence any prediction made using them. This was one of the biggest challenges faced while conducting this research. Last training CNNs is a tedious and time-consuming process ruled by trial and error. CNNs are quite complicated, involving many hyperparameters to set and different methods and techniques to choose from that could lead, or not, to improve learning from data. Here, some of the most popular approaches, known to work well in most cases, are tested. This includes the type of optimizer used for training, batch normalization, and dropout methods.

5.1 Equipment, Tools & Software

For the data collection phase, the GPR system NOGGIN smart cart by Sensors & Software equipped with a 250MHz antenna was used (*Figure 5.1*). The antenna is attached to a cart along with a wheel odometer. The odometer is responsible for triggering EM pulses and records traces while rotating, using a fixed distance interval (i.e., the trace sampling dy). The records are stored in memory card attached to the Digital Video Logger (DVL). Further, the DVL can display a GPR profile under scanning in real-time. At the same time, it also performs other data-collection-related processes like the odometer calibration or setting various parameters like sampling steps, survey grids dimensions, estimation of EM velocity, and stacking. More details can be found in the user guide in [169].

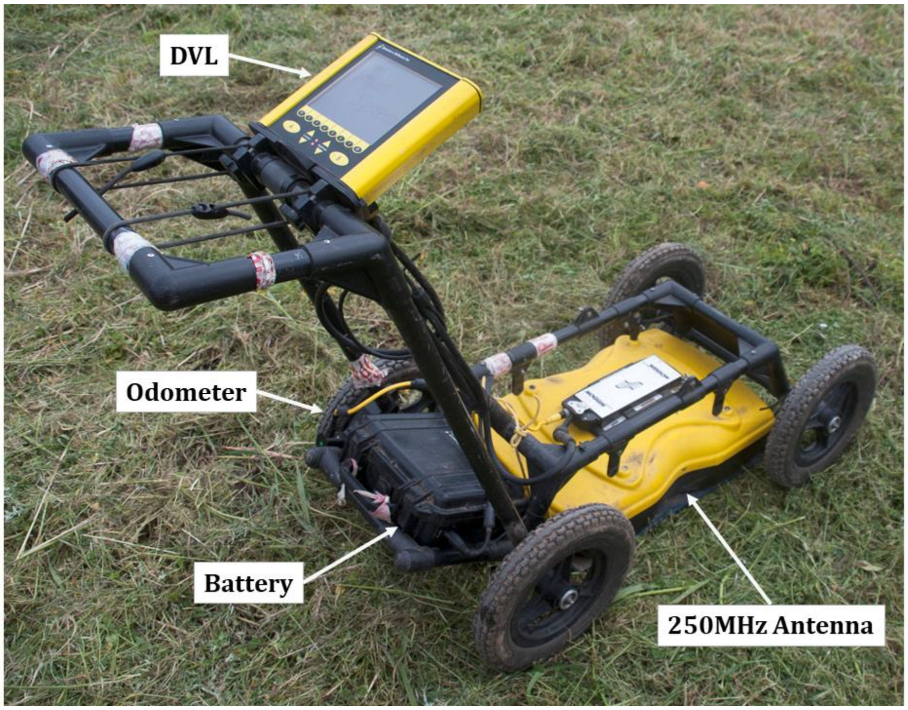


Figure 5.1: The components of the NOGGIN smart cart plus GPR system that was used for the data collection of this research. The photo was taken during the fieldwork at Naxos, Sicily.

The collected data were processed in MATLAB R2017b. Scripts were made to import and process the collected data, mostly using a compilation of functions from toolboxes developed for GPR processing like GPR-Pro V1.4 [170], matGPR R2 [171], as well as from the seismic toolboxes CREWES [172] and

SeismicLab [173]. The data preparation phase for CNNs was also performed in MATLAB R2017b, developing scripts to produce the images of the dataset.

The last phase of the research related to training CNNs for GPR feature classification was performed in Python v.3.6. For the implementation of CNN, the `Tensorflow` library and `Keras` API were employed using GPU support. NVIDIA CUDA development 10.1 was used. Other libraries used for various computations and producing plots and figures are `NumPy`, `matplotlib`, `PIL`, and `sklearn`. The latter methods and functions were used to evaluate classification performance.

Both data processing and CNN training were performed on a PC with the following specifications: CPU Intel Core i7-4790K @ 4.00GHz, RAM 16 GB, OS Windows 10 64-bit, GPU NVIDIA GeForce GTX 970. Google Colaboratory, or "Colab" for short, was also used in training and tuning CNNs. Google Colab is a hosted Jupyter notebook service by Google Research that allows running python scripts through a browser. At the same time, it provides access to computer resources, including GPUs, for a limited amount of time.

5.2 Archaeological Sites & GPR data collection

The data used for this research were collected from 52 different archaeological sites located in Greece, Cyprus, and Sicily, through integrated geophysical surveys. Most of them took place during the period 2013-2019. These surveys were organized and guided by the laboratory of Geophysical – Satellite Remote Sensing and Archaeo-Environment of the Institute for Mediterranean Studies - Foundation and Research and Technology Hellas (IMS - FORTH). The selected sites exhibit traces of civilizations from different historical periods, with most of them dating from the Neolithic to Ottoman years.

The purpose of GPR surveys in all cases was to map structural archaeological remains in the near-surface. The GPR system used was NOGGIN GPR equipped with a 250MHz antenna (*Figure 5.1*). This particular GPR system was efficient in mapping buried foundations up to 2m below the surface in various archaeological sites that were considered for this research, where different environmental conditions were met, including from flat and urban areas to rural and more rocky ones with rough terrains and conductive soils. The 250MHz antenna provided a good balance between the spatial resolution and investigation depth to map structural remains. A 500MHz antenna was also tested in a few sites but exhibited higher attenuation in the recorded signals, which decreased the investigation depth.

The data were collected using survey grids defined by a local cartesian coordinate system described in Chapter 3. Points along X-axis indicate the start and endpoints of the scan lines, while points along the Y-axis are the collected traces. The former is often referred to as the baseline, while the latter is the scan axis. When the area of interest was large, the mosaic approach was followed. According to it, a broader area is covered using subgrids. The size of each subgrid is decided upon by the terrain's morphology and the need for coverage. There is also the NOGGIN system's limitation that a survey grid cannot exceed the number of 100 lines. This led to various grid sizes and geometries, with 2x4m being the smallest and 50x120m being the largest measured among the areas this research concerns. The average grid dimension was 20x30m. The total number of survey grids measured with GPR is 470, covering a sum area of 321,918m^2.

All the GPR profiles were collected in parallel lines using a fixed 0.25-0.5m spacing between them, which is adequate for mapping foundation structures. The traces sampling was set to 0.05m or 0.025m along a scan line, which was decided upon the terrain's morphology. The smaller sampling might provide a better resolution but lowers the data collection speed, which can be challenging to handle in rough terrains resulting in skipping records. The lines orientation was either the same (i.e., parallel mode) or alternate (i.e., zig-zag mode). For cases that rectangular grids were defined, the zig-zag mode was preferred as it speeds up the data collection. The parallel mode was used when a rectangular survey grid was not possible to be defined, but the need for coverage was high (i.e., following a natural boundary). Last, stacking was used to amplify the transmitted signal and improve data quality (i.e., fixing the signal attenuation). The number of stacking was decided on the field according to factors like the signal's attenuation and terrain morphology on each survey grid. The more stacks, the longer it takes to record a trace; thus, it reduces the data collection speed.

Details on the 52 archaeological sites like the location, the number of grids per site measured, and published material on the collected GPR data are summarized in *Table 5.1* to *Table 5.3*. In the same tables, data collection parameters adapted to each site's conditions like trace sampling and stacking are also presented. The sites in *Table 5.1* were surveyed under the IGEAN research project. The surveys conducted for the sites presented in *Table 5.2* are organized per research program, and last, the sites presented in *Table 5.3*, the surveys were carried out in collaboration with the corresponding Ephorates of Antiquities responsible for the site. In these sites where human traces from different chronological periods have been found, dating from the Neolithic to Ottoman years, GPR could detect structural features from various chronological periods, as was discussed in Chapter 3. Its performance in detecting these traces varied, and this relied on the prevailing soil conditions, the contrast in physical properties, and the preservation state of the buried structures.

Table 5.1: Information on the Neolithic settlements of Thessaly, Greece, and GPR data collection that was carried out for the research program IGEAN. Data collection parameters include the total number of survey grids measured on each settlement, the trace sampling dy *and the number of pulse stacking. The line spacing for all the grids was* $dx = 0.5m$*. For the survey parameters, cells with more than one value indicate different settings for measuring the grids in the same area.*

Innovative Geophysical Approaches for the Study of Early Agricultural Villages of Neolithic Thessaly-(IGEAN)[174]

#	Site name	Location	Grids (#)	dy(m)	Stacks (#)	Published material
1	Agios Dimitrios	Agios Dimitrios, Magnesia, Thessaly	4	0.025	8	[175]
2	Agios Nikolaos	Kanalia, Magnesia, Thessaly	2	0.025	8	[175]
3	Almyriotiki	Almyros, Magnesia, Thessaly	8	0.025-0.05	4 & 8	[175]–[179]
4	Almyros 2	Almyros, Magnesia, Thessaly	5	0.05	8	[63], [69]
5	Bakalis	Velestino, Magnesia, Thessaly	2	0.025	16	[16], [18], [175], [182]
6	Belitsi	Anchialos, Magnesia, Thessaly	4	0.025 & 0.05	4 & 8	[175], [183]
7	Deksameni	Almyros -Velestino, Magnesia, Thessaly	1	0.025	8	[175]
8	Eleftherochori	Eleftherochori, Magnesia, Thessaly	4	0.025	8	[175]
9	Kamara	Almyros, Magnesia, Thessaly	4	0.05	8	[175], [181]
10	Karatsantagli	Almyros, Magnesia, Thessaly	1	0.025	4	[63], [69]
11	Karatsantagliou	Magnesia, Thessaly	1	0.025	4	[175]
12	Kastraki 2 /Perivlepto	Magnesia, Thessaly	8	0.025	8	[175], [181]
13	Kastro Kokkinas	Kokkina, Magnesia, Thessaly	1	0.025	8	[175]
14	Mylos Baitsi	Almyros, Magnesia, Thessaly	8	0.025	8	[175]
15	Perdika 1	Almyros, Magnesia, Thessaly	6	0.05	8	[176]
16	Perdika 2	Almyros, Magnesia, Thessaly	3	0.05	8	[175], [179], [181]
17	Rizomilos 2	Magnesia, Thessaly	3	0.025	8	[175], [184]
18	Velestino 3 - Mati	Velestino, Magnesia, Thessaly	9	0.05	8	[175], [176]
19	Velestino 4 - Visviki	Velestino, Magnesia, Thessaly	1	0.025	8	[175], [182]
20	Zerelia	Almyros, Magnesia, Thessaly	2	0.025	16	[175]

Table 5.2: Information on various archaeological sites and the GPR data collection that was carried out for various research programs. Data collection parameters include the total number of survey grids measured on each settlement, the trace sampling dy and the number of pulse stacking. The line spacing for all the grids where $dx = 0.5m$ with the only exception a few grids in Sissi where $dx = 0.25m$. For the survey parameters, cells with more than one value indicate different settings for measuring the grids in the same area.

<table>
<tr><th colspan="7">POLITEIA research project, Action KRIPIS [185] (Code: 2013SE0138003)</th></tr>
<tr><th>#</th><th>Site name</th><th>Location</th><th>Grids (#)</th><th>dy(m)</th><th>Stacks (#)</th><th>Published material</th></tr>
<tr><td>**21**</td><td>Demetriada</td><td>Magnesia, Thessaly</td><td>7</td><td>0.05</td><td>8</td><td>[16], [18], [186]–[188]</td></tr>
<tr><td>**22**</td><td>Gortyna</td><td>Heraklion, Crete</td><td>6</td><td>0.05</td><td>8</td><td>-</td></tr>
<tr><td>**23**</td><td>Lefkes</td><td>Livadeia, Boeotia, Central Greece</td><td>2</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**24**</td><td>Mantineia</td><td>Arcadia, Peloponnese</td><td>17</td><td>0.05</td><td>8</td><td>[186], [189], [190]</td></tr>
<tr><td>**25**</td><td>Magoula Balomenou</td><td>Chaeronea, Boeotia, Central Greece</td><td>3</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**26**</td><td>Psilomata 2</td><td>Livadeia, Boeotia, Central Greece</td><td>3</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**27**</td><td>Psilomata 3</td><td>Livadeia, Boeotia, Central Greece</td><td>3</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**28**</td><td>Orchomenos</td><td>Boeotia, Central Greece</td><td>2</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**29**</td><td>Tegyra</td><td>Boeotia, Central Greece</td><td>6</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><td>**30**</td><td>Voulokaliva-Site 35</td><td>Voulokaliva, Magnesia, Thessaly</td><td>3</td><td>0.025</td><td>8</td><td>-</td></tr>
<tr><th colspan="7">Ancient City project, ARISTEIA II Action (Code:2013SE0138004)</th></tr>
<tr><td>**31**</td><td>Halos</td><td>Magnesia, Thessaly</td><td>7</td><td>0.025</td><td>8</td><td>[186]</td></tr>
<tr><td>**32**</td><td>Heraia</td><td>Arcadia, Peloponnese</td><td>12</td><td>0.025</td><td>4</td><td>[186], [187]</td></tr>
<tr><th colspan="7">ArchaeoLandscapes Europe" (ArcLand) [191]</th></tr>
<tr><td>**33**</td><td>Hyettos</td><td>Boeotia, Central Greece</td><td>34</td><td>0.025 & 0.05</td><td>8</td><td>[192], [193]</td></tr>
<tr><th colspan="7">The Sissi Archaeological Project (Sarpedon) [194]</th></tr>
<tr><td>**34**</td><td>Sissi</td><td>Heraklion, Crete</td><td>17</td><td>0.05</td><td>8 & 16</td><td>[195]–[197]</td></tr>
<tr><th colspan="7">The Greek colony of Naxos in Sicily: mapping the town plan and geophysical survey [20]</th></tr>
<tr><td>**35**</td><td>Naxos</td><td>Giardini-Naxos, Sicily</td><td>89</td><td>0.025 & 0.05</td><td>8</td><td>[198]</td></tr>
<tr><th colspan="7">Salamis Urban Landscape Project 2016–2020 [199]</th></tr>
<tr><td>**36**</td><td>Salamis</td><td>Salamis island, Attica, Central Greece</td><td>9</td><td>0.05</td><td>8</td><td>-</td></tr>
</table>

Table 5.3: Information on archaeological sites and the GPR data collection that was carried out for various individual surveys in collaboration with the corresponding Ephorates of Antiquities. Data collection parameters include the total number of survey grids measured on each settlement, the trace sampling dy and the number of pulse stacking. The line spacing for all the grids where $dx = 0.5m$. For the survey parameters, cells with more than one value indicate different settings for measuring the grids in the same area.

Individual Geophysical surveys						
#	Site name	Location	Grids (#)	dy(m)	Stacks (#)	Published material
37	Amathounta	Limassol, Cyprus	5	0.025	8	-
38	Bentenaki	Heraklion, Crete	7	0.025	16	[39]
39	Delphi	Phocis, Central Greece	17	0.05	8	[200]
40	Elateia	Phthiotis, Central Greece	15	0.025-0.05	8 , 22 & 25	-
41	Idomeni	Kilkis, Macedonia	6	0.05	8	[201]
42	Ierapytna	Lasithi, Crete	6	0.025	16	[34]
43	Koumasa	Heraklion, Crete	13	0.05	8	-
44	Lechaion	Corinthia, Peloponnese	13	0.05	8	-
45	Palamari	Skyros island, Sporades, Central Greece	10	0.05	8	[202]
46	Paralia Avlidas	Euboea, Central Greece	25	0.025	8	-
47	Pella	Pella, Macedonia	5	0.05	8	-
48	Plataies	Boeotia, Central Greece	19	0.05	8	-
49	Sikyon	Corinthia, Peloponnese	19	0.025	16	[19], [203], [204]
50	Turkish school	Rethymno, Crete	6	0.05	8	[39], [205]
51	Yperia Krini spring	Velestino, Thessaly	2	0.025	8	-
52	Zominthos	Rethymno, Crete	5	0.05	4	-

In *Figure 5.2,* a compilation of photographs taken in a few of the abovementioned sites during data collection using NOGGIN GPR is presented. These are a few examples that show the variety of the different conditions met. These include different weather conditions, from rains to heatwaves that affect the soils' water content. Others are the different terrain conditions that include flat and easily access areas, more rocky areas, cultivated areas, high vegetation areas, and different slopes.

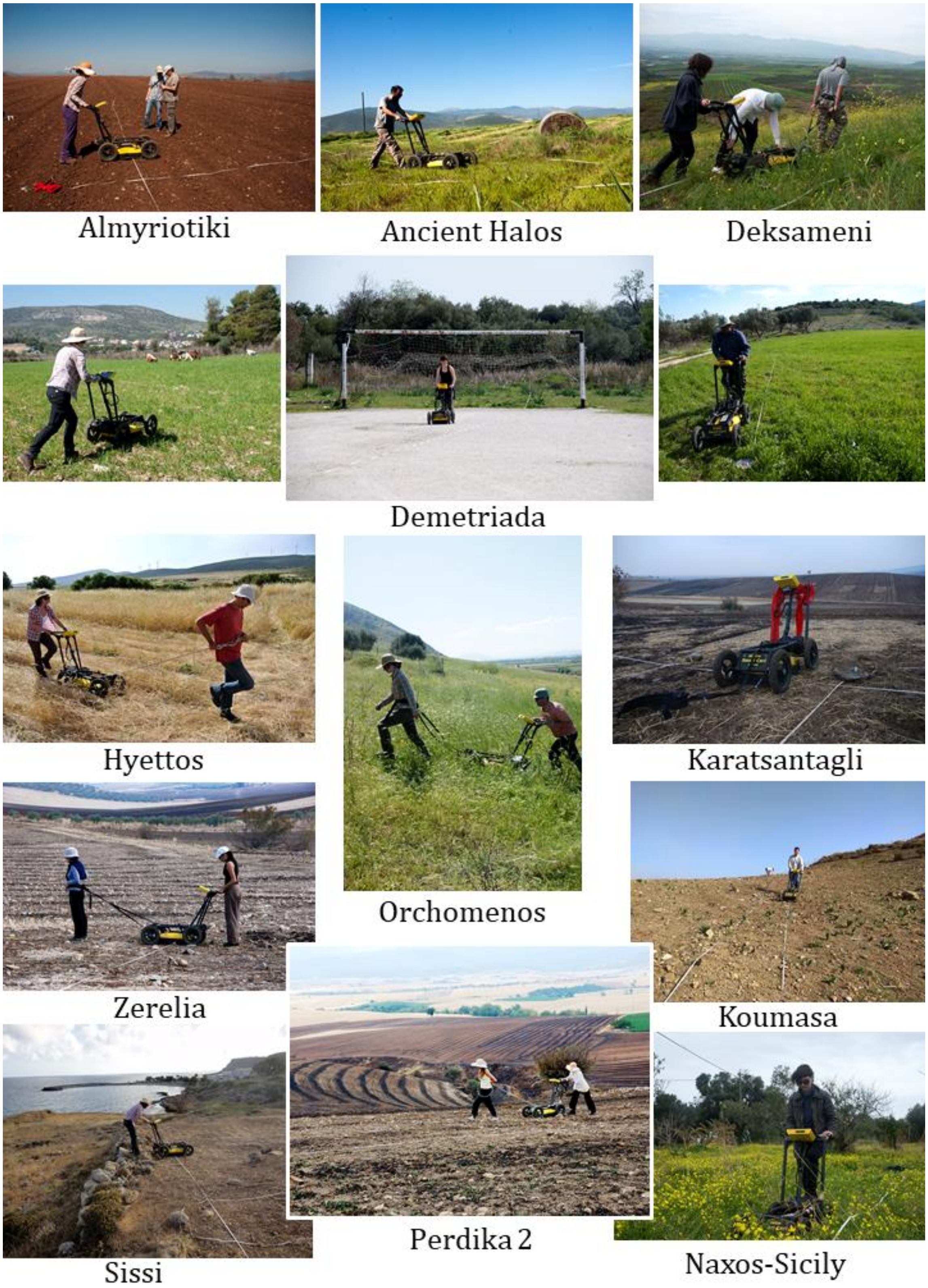


Figure 5.2: Compilation of photographs taken in various survey sites using NOGGIN GPR showing the different conditions met.

5.3 Data Processing

GPR data processing is known to be a tedious process characterized by trial and error as the quality of the data is *site-dependent*, and the processing needs may differ significantly not only among different areas but also for profiles collected in different grids in the same area. However, due to the amount of data used in this research, an empirical workflow that was found to work satisfactorily for the collected data was followed. Processing was held in MATLAB and is divided into three stages. The first is to import the data, the second to process the GPR profiles to reduce noise and highlight reflection from the subsurface, and the last to produce and export depth slices.

5.3.1 GPR data import

The first step of processing is to import survey grid data in the MATLAB environment. A NOGGIN GPR stored profile consists of two files; a header file (.HD) and a binary data file (.DT1). The header file is an ASCII file accessible by any word processor and text editor software. The stored information includes system details and data collection settings. Examples of header files can be found in *Table A.1* of Appendix A. The data file format is Digital Terrain Elevation Data (DTED) of level 1. It contains as many records as the collected traces, and each record is a set of a header section and a data section. The former serves as an annotation that links the information of the HD file to each trace. The latter is an array of 2-byte integers, and each value corresponds to a data point of a trace. An example of the data file structure can be found in appendix A of the NOGGIN user guide in [169]. The stored profiles used the prefix LINEY or LINEX, where Y or X indicate whether the scan-axis orientation is along the Y or X-axis of the survey grid and is followed by the line's number. Numbering starts from 0, so LINEY0 will be the first line of the grid with the Y-axis direction.

A MATLAB script was made to import the stored profiles and is given in *Script A*.*1* of Appendix A. The script is a modified version of the one found in the toolbox GPR-Pro [170] and is adapted to work with NOGGIN data. When running the script, navigation windows appear, prompting the user to select the .HD and their corresponding .DT1 files. The filenames and paths are stored along with selected parameters from the header file, useful for processing, which are assigned as variables. The data points are loaded using the built-in functions `fopen`, `fread,` and `fclose`. Since all the loaded traces are flattened in a single vector, the next step is to arrange them accordingly in a 2D array to create a B-scan where each column is a trace. Further, the scan-axis distance vector, the time sampling, and the double travel time vector are also computed and stored.

The script keeps the number of the selected lines, and if a single line is selected, then the 2D array is named `B-scan`. If multiple lines are selected, the B-scans are stored under the cell array named

Lines, as happens in a survey grid. A cell array has indexed data containers called cells containing any type of data like vectors or arrays of strings and numbers, or mixed and of different sizes and lengths. This data type is convenient in handling the different dimensions of the B-scans imported from a survey grid and makes possible the application of processing workflows through loops. After running the script, all the imported variables and GPR profiles were saved manually in a .mat file treated as the raw data.

5.3.2 Processing the imported B-scans

The processing workflow applies in the raw Bscans using the saved variables after running the `NOGGIN_data_import.m` script. The processing workflow presented here applies to the B-scans of a survey grid that are stored in a cell array. The purpose is to remove noise, highlight reflections from the subsurface, and geometrically correct the Bscans so that they can be inserted in a 3D array. Each processing step was applied to all the lines of the cell array using for loops. The output of each step was the input of the next one. A representative example of the processing workflow followed is given in Appendix A for a survey grid at Demetria's archaeological site. In summary, the workflow includes in order:

1. Geometrical corrections to fix offsets in the collected traces' position. This correction resamples the distance vector to meet the line length indicated by the survey grid geometry. A script was made to perform this task and is given in the processing example of Appendix A (*Script A.3*).
2. Time zero to correct the vertical position of the first pulse. A script was made that uses a user-defined time value and crops out earlier records and given in the processing example of Appendix A (*Script A.4*).
3. Dewow filter that is applied for the removal of low-frequency noise. The `dewow` function from the toolbox matGPR[171] was used.
4. Gain functions to compensate for the attenuation effect like SEC2 gain or AGC gain. In this research, the adaptive gain Inverse Amplitude Decay (IAD) of matGPR was used that is implemented with the function `gaininvdecay`. This particular implementation of gain was preferred due to its adaptivity that allows the user to select the attenuation model that better suits a survey line.
5. Filters and corrections that are applied to remove noise enhanced by the gain. This includes the Average Background Removal (ABR) and bandpass filtering. For ABR, the function `rmbackgr` of matGPR was used. As for bandpass filtering, the function `bp_filter` from the

SeismicLab toolbox [173] was used. The frequency window of 100MHz – 500MHz was found to retain useful information and remove most noise that was enhanced by the gain filter. This frequency band was determined by observing the average spectrum of the lines in several grids. The bandpass filtering effect is presented in *Figure 5.3*. The average spectrum was calculated for the bandpass filter input using a modified version of the `estimatefw` function made by Dr. Nikos Economou from the Laboratory of Applied Geophysics, School of Mineral Resources Engineering – TUC.

In *Figure 5.3,* representative examples are given that show the effect of bandpass filtering when using the frequency window of 100-500MHz. The presented examples are derived from two survey grids, one measured at the Ancient Halos's site and the other at Demetrias' site. For both cases, a B-scan and a C-scan are presented before and after bandpass filtering is applied. The corresponding average spectrum plots for all the traces in selected B-scans before and after bandpass filtering are also given where the frequencies cut-off is shown. For Ancient Halos, the processing workflow that was followed and described earlier left noise in the B-scans, which was enhanced with gain, and ABR could not remove. The noise that is visible in the B-scan at X=0.6m of the survey grid in *Figure 5.3* is low-frequency and is efficiently removed by cutting off frequencies below 100MHz. Additionally, the frequencies above 500MHz do not seem to be related to any useful information, and hence they are removed. In the filtered B-scan, the noise is removed while information related to the subsurface's reflections is retained. This greatly impacts the produced C-scans where bandpass filtering that retains the 100-500MHz frequency range has overly improved the depth slice at t=24.5ns, and structures are better visible.

Bandpass filtering at 100-500MHz after applying IAD gain and ABR is equally effective to the representative B-scan was taken at a survey grid from Demetria's site. The reflections identified as structures are better shown in the filtered image. By removing strong amplitudes that correspond to noise, the architectural remains are shown with greater detail in the resulting C-scan at t=25.8ns and without the loss of useful information. This was also the case for all the survey grids studied and processed in this thesis. Although this processing workflow was not always optimal for all the collected data, it did not worsen or distort the results.

A few exceptions were made during the first step of the abovementioned workflow. The trace resampling was only applied in case studies where a rectangular grid was defined, as the same fixed value can be used for all the lines' lengths. There were a few cases in the survey lines that were not starting or ending in a predefined position. A few examples are cases where natural boundaries were followed or when natural obstacles existed on the surface that prevented defining a rectangular grid. For those, trace resampling could not be applied and was excluded from the workflow.

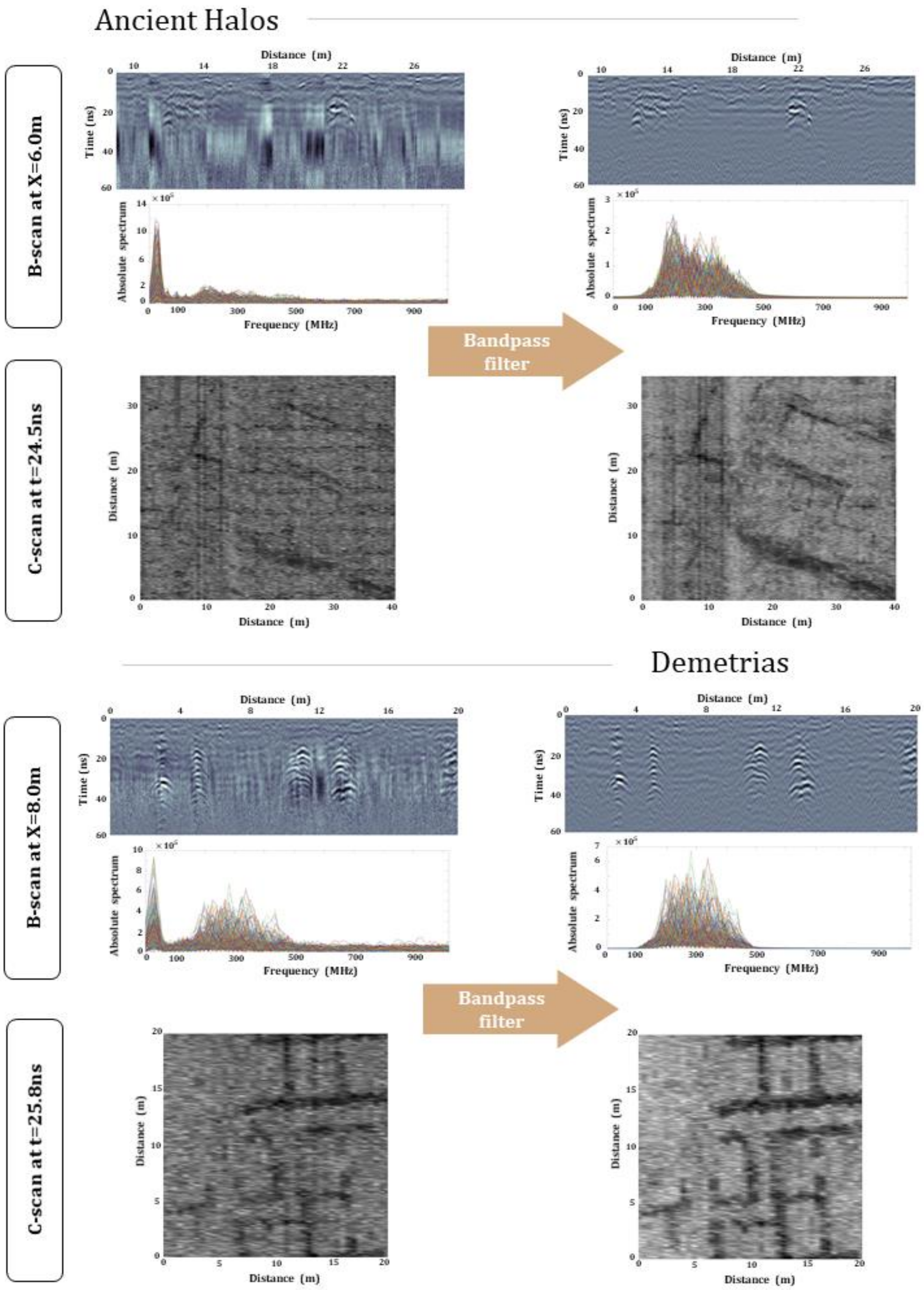


Figure 5.3: Representative examples that show the bandpass filtering effect on two B-scans and C-scans when using the frequency window of 100-500MHz.

5.3.3 Export depth slices

The last step is to produce depth slices or C-scans. This includes the application of *Hilbert Transform* (HT) on each profile to calculate the *Instantaneous Envelope* or *Amplitude* and then put the B-scans of the cell arrays in 3D arrays designed after the survey grid dimensions. The function of GPR-Pro, `dBInstantaneousEnvelope`, was used to calculate the instantaneous envelope. This step is implemented in *Script A.9* of Appendix A. If the zig-zag mode was used during data collection, then the orientation of the corresponding lines requires fixing to get the correct geometry. These lines correspond to the odd-numbered lines of the survey grid that go under the even indices of the 3D array, were reversed using MATLAB’s built-in function `fliplr(`*Script A.10*`)`. An alternative and more generic approach is given in *Script A.11* that while creating the volume, it searches for lines with a negative sampling step, and if found any, it reverses only those. For the case where the survey grid was not rectangular, the lines were padded with NaN to the longest line’s dimension collected to create a 3D volume and extract slices (*Script A.12*).

The next step is to extract the slices. The 3D volume has a dimension of MxNxK where M is the number per trace in time (ns), N is the number of the collected traces along a line (m), and M is the number of lines in the survey grid (m). In order to create the C-scans, arrays with size NxK are extracted by taking samples in the vertical axis. These arrays are like snapshots of the area outlined by the survey grid at different times. One characteristic of the GPR data is the difference in the size of N and K with N>>K, as the sampling step used is of a different order of magnitude. The line spacing is about ten times bigger than the sampling step that the traces are collected. Thus, to create images proportional to the grids' dimension, the scan-axis is usually downsampled, while the baseline-axis is upsampled. Further, an interpolation method and colormap of choice are applied to create pseudocolor images. This process was performed using MATLAB’s pseudocolor plot, `pcolor`. An implementation is given in *Script A.13* of Appendix A, which plots and saves the produced C-scans in the active root directory using the time sample's name. The script was used to produce the depth slices presented in this thesis using the reverse grayscale as colormap.

Last, for case studies, the X-axis was chosen as the scan-axis, the scripts were modified accordingly to get the correct geometry, and the same processes were applied. A schematic overview of the data processing that was followed in this research is presented in *Figure 5.4*. The B-scan presented is a line collected in a survey grid at Demetria's archaeological site that corresponds at x=10.0 m. Reflections related to buried structures are visible in the output Bscan produced by the processing workflow followed. The same processes were applied for all the B-scans in the survey grid, and 3D volume was created to export C-scans as described previously. The presented C-scan is sampled at the time of 28ns and exhibits anomalies identified as well-preserved structural remains of the Hellenistic period.

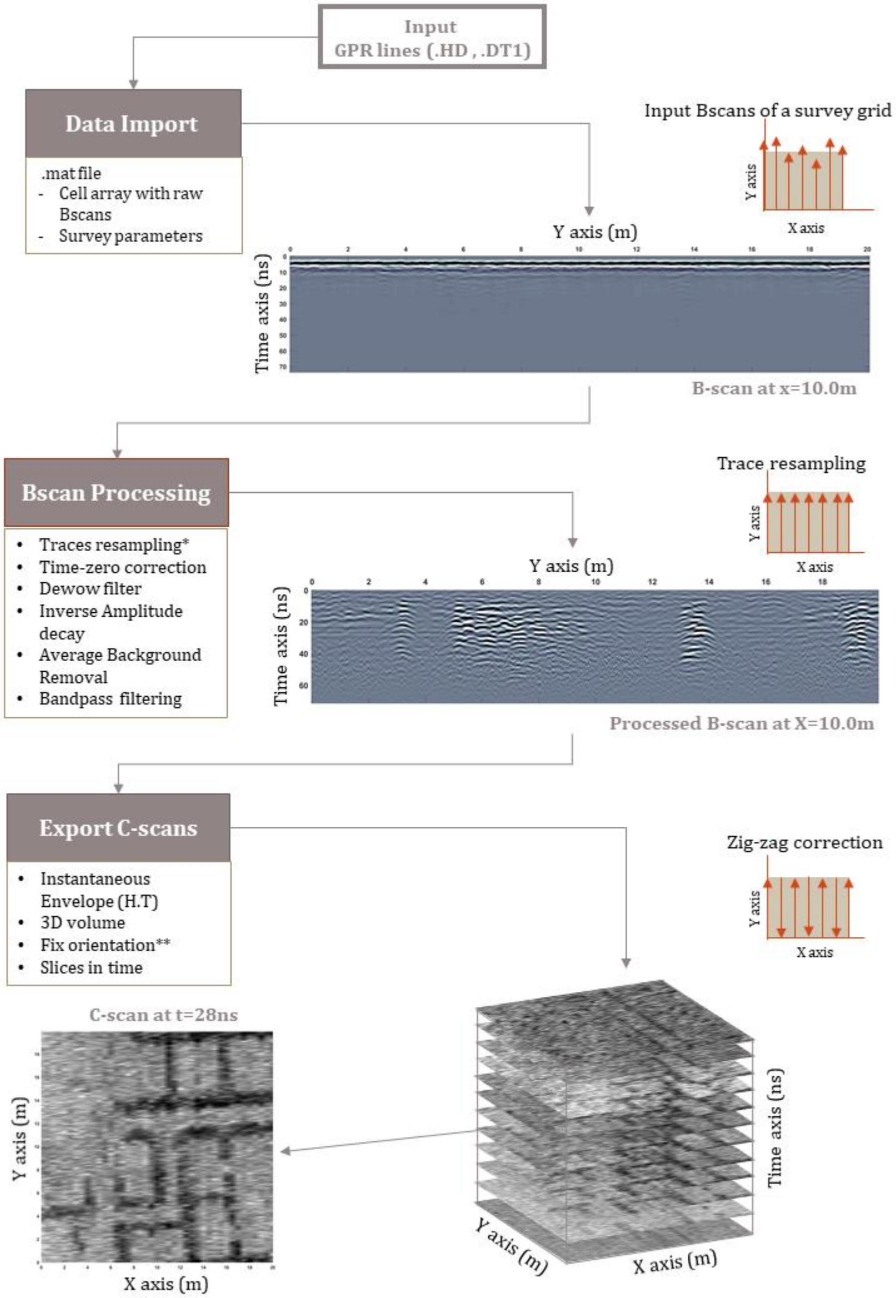


Figure 5.4: Data processing overview to produce C-scans. The presented examples were collected from Demetria's archaeological site. Trace reposition and zig-zag correction were applied only for survey grids that required it.

5.4 Building Datasets

The produced images from each survey grid were gathered and examined, looking for anomalies attributed to buried archaeological remains, and could be used to construct the training datasets. During this process, three main categories of features were observed in the images:

- Geophysical anomalies attributed to structural remains. These are linear features that imprint the geometries of buried walls, individual structures, or even more complex architectures like city blocks (*Figure 5.5*).
- Geophysical anomalies related to the subsurface. These are areas of strong amplitudes that usually have an irregular shape but can also exhibit a linear trend. For most of them, the reflector's nature is uncertain. Usually, assumptions are made based on their shape, their depth, and in-situ observations during data collections. These anomalies can be related to archaeological remains, but their poor preservation condition prevents their identification as they appear as fuzzy areas in the data. They can also be entirely irrelevant to archaeological contexts, like geological layers and voids, bedrock, buried debris, pipes, or other modern constructions. The knowledge of their existence is essential in evaluating the subsurface's overall condition within the investigation depth (*Figure 5.6*).
- Noise. Since different noise types exist, the noise here describes the residual noise that was not removed from the applied processing workflow and usually appears in stripes form. The cause that created this type of noise varies, and, at the same time, it is difficult to be determined with certainty. It can be the gap between the antenna and the surface when collecting data due to the terrain’s roughness, like plowing lines that were imprinted in the data or faulty traces due to the antenna bumping. Another noise type might be related to the low battery level, creating broad stripes of different amplitude intensity (*Figure 5.7*).

Thus, the labels of *structure*, *anomaly*, and *noise* were defined for the classification task to describe the abovementioned categories' features. Even though identifying the buried structure is the main interest of this research, it was deemed beneficial to identify patterns of other dominant features observed in the GPR depth slices towards a complete and more accurate classification. The idea behind it is that the chances of mistaking a non-structural feature for a structure are less when the classifier is trained to identify the other two classes as well. Especially for a common case met that striping noise co-exists with structures, it can be difficult to discriminate them, increasing this way the uncertainty of the data interpretation made even by the more experienced users. Thus, examining the capability of CNNs to identify and classify these features correctly was included in this dissertation's objectives.

On the next step, manual classification of images exhibiting features belonging to the three classes was held by organizing them accordingly into folders to gather the material for constructing the learning datasets.

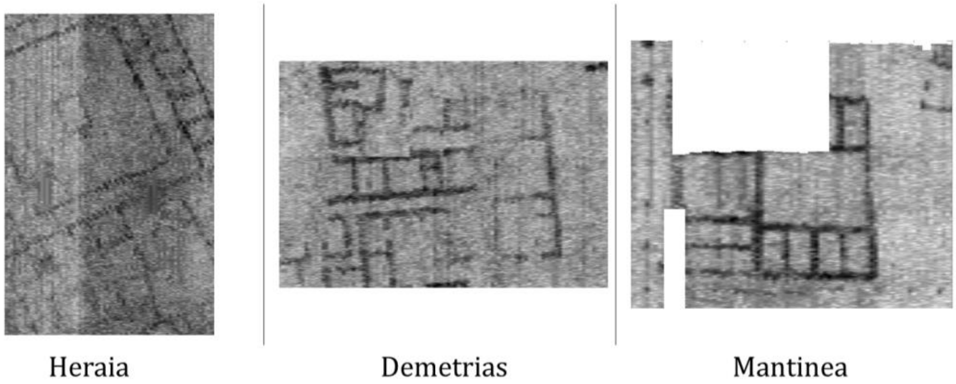


Figure 5.5: Representative examples of well-preserved buried structures imprinted in GPR depth slices. The examples are derived from survey grids at Heraia, Demetrias, and Mantinea archaeological sites.

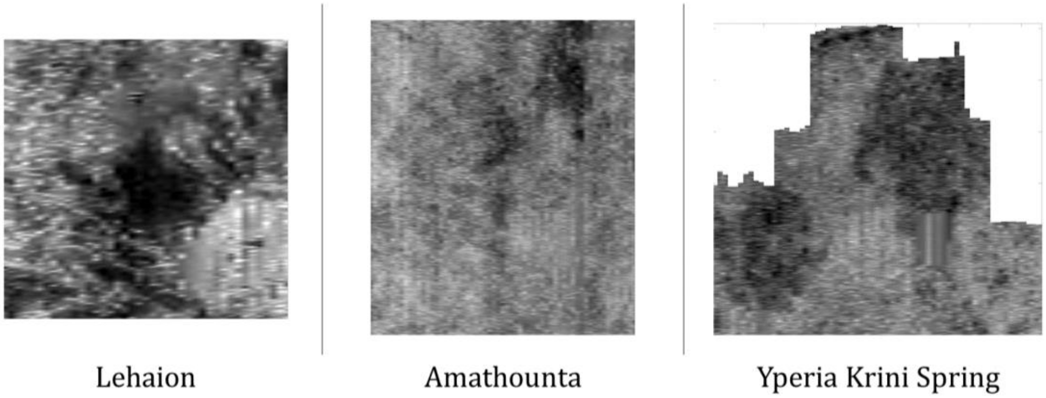


Figure 5.6: Representative examples of geophysical anomalies found in GPR data. For the case of Lehaion, the assumed reflector is the geological layer's boundaries, for Amathounta is debris, and for the monument of Yperia Krini is scattered structural material.

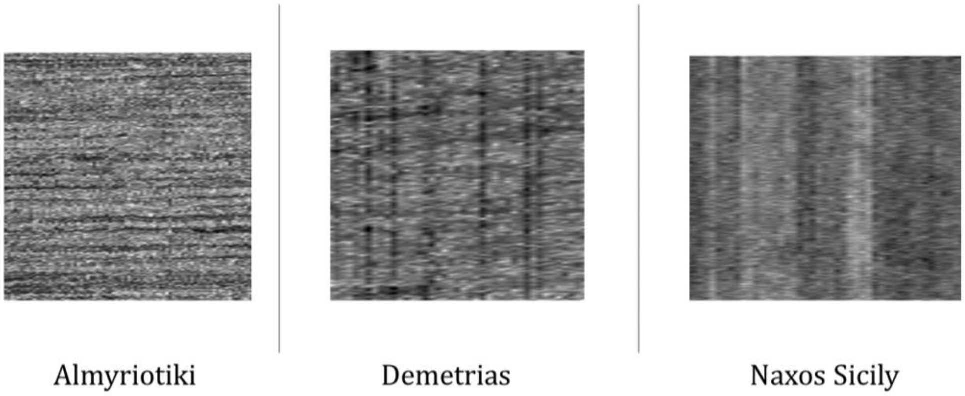


Figure 5.7: Representative examples of striping noise found in GPR data. For the case of Almyriotiki, the horizontal stripes were caused by plowing lines on the surface. For Demetrias, the vertical stripes were probably caused by faulty traces, and for Naxos, Sicily, the stripes are related to low battery levels.

5.4.1 Increasing image number per class

A challenge met during the dataset construction for training CNNs was finding enough images per class to achieve a good generalization and avoid underfitting. As it was described in *subsection 4.5*, a sufficient number of training data examples is required to describe the input space well enough, but practical guidelines on the smallest efficient number of images required are of lacking. For this reason, the size of the datasets used in similar GPR studies (like in [76] and [96]), as well as popular datasets used for training CNNs (e.g., MNIST [206] and CIFAR-10 [207]) were used as guides in filling a sufficient number of images. This has led to a minimum threshold of 5000 images per class as a starting point.

This number was proven difficult to reach. Features belonging to the three classes usually coexist in the depth slices, which prevents from using them as a training classification example for a class. A typical example is given in the depth slice of *Figure 5.8* from the GPR survey at the ancient Halos site. This particular C-scan is sampled at t=26.47ns and presents numerous features, including structural remains of two different constructional phases, residual striping noise from processing, and a strong linear reflector of uncertain nature. In this image, the residual noise has the same orientation as one of the structural phases, making their interpretation difficult and prompting to mistake the noise for structure or overlooking a structure.

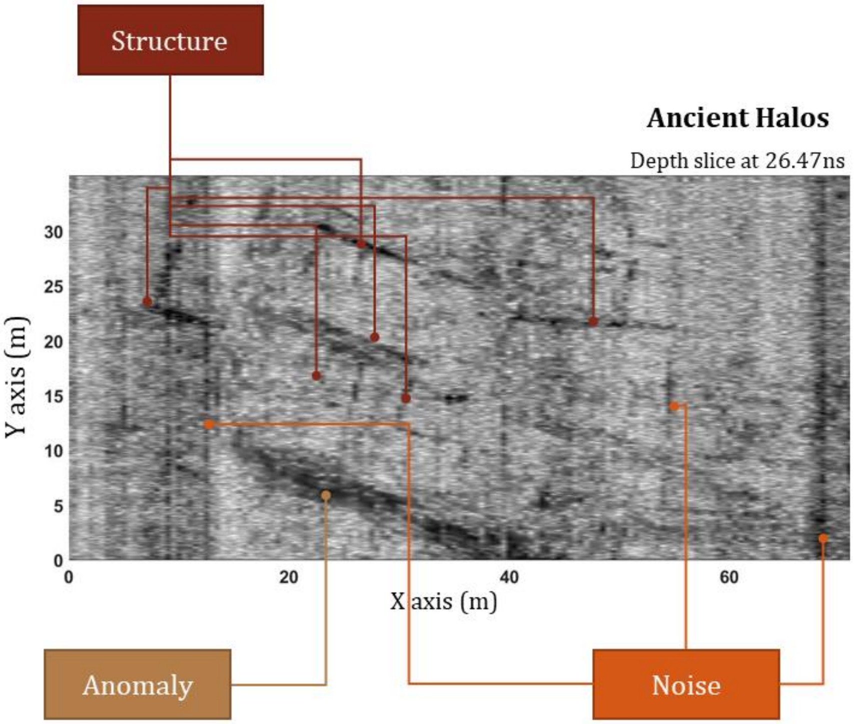


Figure 5.8: C-scan from ancient Halos exhibiting features that belong to the three classes.

Thus, many images could not be assigned to a single class, and in the end, the images that were solely exhibiting features belonging to a class were not that many. An idea to overcome this issue was to isolate and save as individual images subregions of the C-scans of unique features. To further increase the number of images, the subregions are extracted using a square sliding window. This process is implemented in the function `slidewcrop_y` given in *Script A.14* of appendix A. The function takes as input a volume, the sample in time corresponds to a C-scan, the sliding window length, and the stride in the X and Y directions in meters. For the given window's size and stride, the function applies the pseudocolor plot and plots the corresponding subregion of a C-scan as an image. The images are stored in folders on a given path, named after the window size, and the time in ns the C-scan is sampled. The function also applies to multiple time samples in a 3D volume. An example of applying `slidewcrop_y` is given in *Script A.15* of Appendix A. The resulted images were manually browsed and organized under folders according to the features.

The sliding window function was applied in selected C-scans from the various sites and grids using a 10x10m size window. This window size was found adequate to produce as many sub-images as possible and describe the patterns of interest well. For this reason, the survey grids that are bigger than 10x10m were only used to export images. As for the stride, a 2.0m distance was chosen, which results in overlapping sliding windows. This approach further increased the produced images' number. Krizhevsky et al. used a similar approach in [50], where they took five crops of an input image aligned at its center and the four corners. The sliding windows approach is preferred here as it can produce more images of useful features and can be adapted better to GPR images' different dimensions. An example is presented in *Figure 5.9*, where a region of the C-scan featuring structures is cropped using sliding windows. The starting point of the window in this example is X=0m, Y=84m. All the produced images were saved in .jpeg format using a pixel size of 256x256.

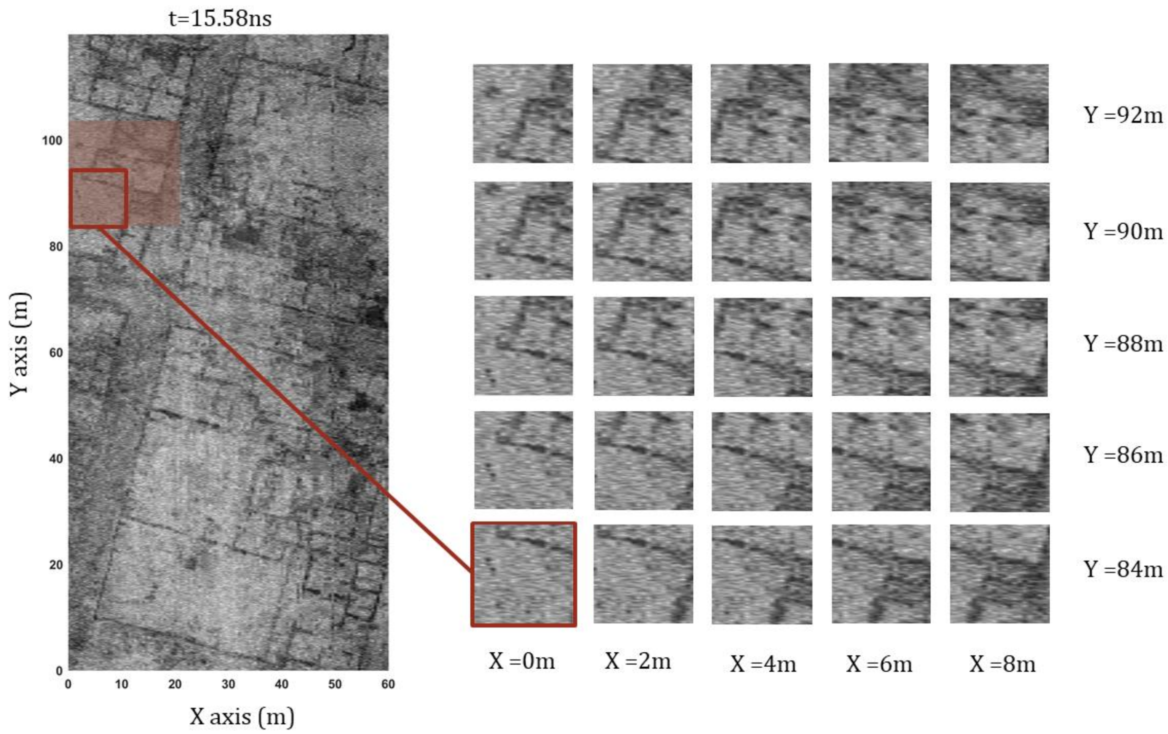


Figure 5.9: Image cropping with overlapping sliding windows using a C-scan from Demetria's GPR survey as an example. The window used has a length of 10mx10m while the stride is 2.0m in both directions X and Y.

The abovementioned process was applied to numerous C-scans picked from the survey grids studied in this research. Their selection was made while considering the following:

- to be representative of the subsurface conditions and summarize the most important features.
- to highlight the different ways the same feature appears in different depths.
- to avoid repetitive patterns and overly similar images

An example of C-scan selection for cropping is presented in *Figure 5.10* using images from a survey grid at Demetria's site, where city blocks, including several structural elements, are mapped. Each C-scan reveals a different level of detail of the same targets.

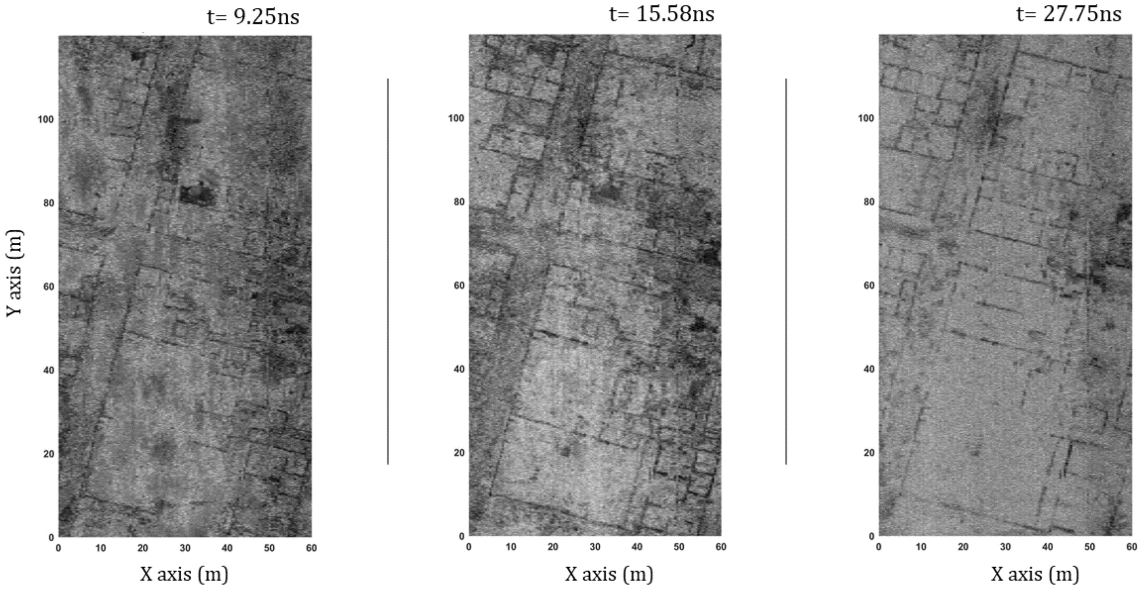


Figure 5.10: Example of C-scans selection in a survey grid measured at the ancient Demetrias site used for cropping. Shorter times are closer to the surface.

5.4.2 Constructing the test and training sets

The cropped images produced by the previous step were browsed manually, choosing representative examples for each class while avoiding the ones that exhibit features for more than one class or were almost identical. Based on this approach, a total of 18750 images were gathered, having 6250 images per class. The images were split using the 80%-20% rule into the training set and test set. This led to using 5000 images per class for training models and 1250 for testing their generalization. The images were organized into the directory structure identified by the *Keras API* and *Tensorflow* library to load the data and classification labels (*Figure 5.11*).

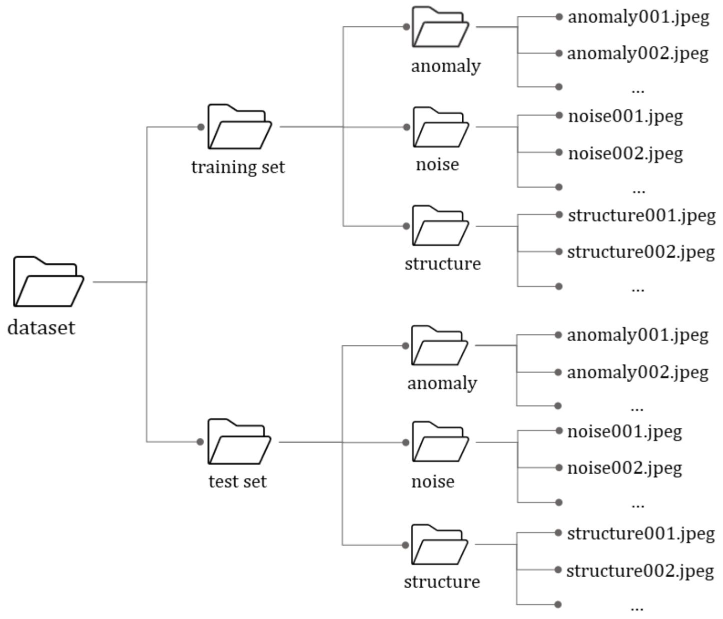


Figure 5.11: Directory structure of the dataset followed.

A dilemma that emerged during dataset construction concerns selecting the test set's examples that would lead to a better generalization. One idea was to exclude images derived from a site entirely from the training set and use them solely as a test set. The other was to follow a random selection on all the images gathered and split them into a test and training set. Hence, two datasets were built, named *dataset-A* and *dataset-B*. For dataset A the images used as a test set were exclusively selected from survey grids measured at Elateia archaeological site. This site was chosen as it exhibited enough examples to reach 1250 images per class. The training set includes 5000 images selected from the rest archaeological sites. The second approach was followed for dataset-B, i.e., splitting the 6250 images per class randomly into a training and test set. In *Figure 5.12* and *Figure 5.13,* a small sample from the two dataset's images is presented per class for comparison reasons. The presented samples were chosen randomly, and considering the small size, they do not fully represent the entire dataset. Both datasets have pros and cons. Dataset-A is closer to the real-case scenario to use the model for predictions in unseen data. However, the test set might not be representative enough, which may cause unstable training performance. The test set of dataset-B is representative of the corresponding training set, but due to the overlapping window crops, it might fail to predict correctly unseen data.

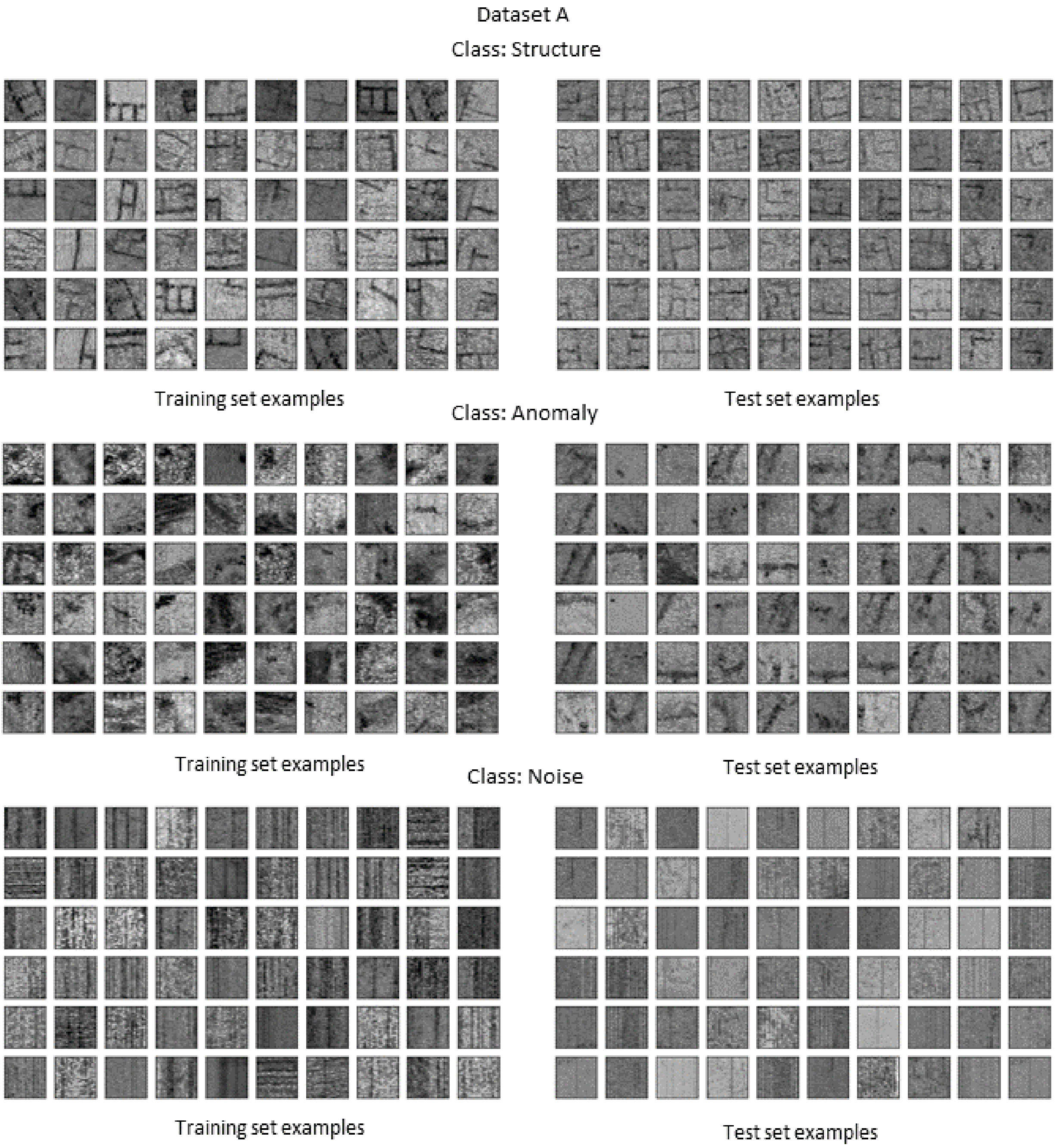


Figure 5.12: Training and test set examples from dataset-A. Test set images are sampled from Elateia archaeological site.

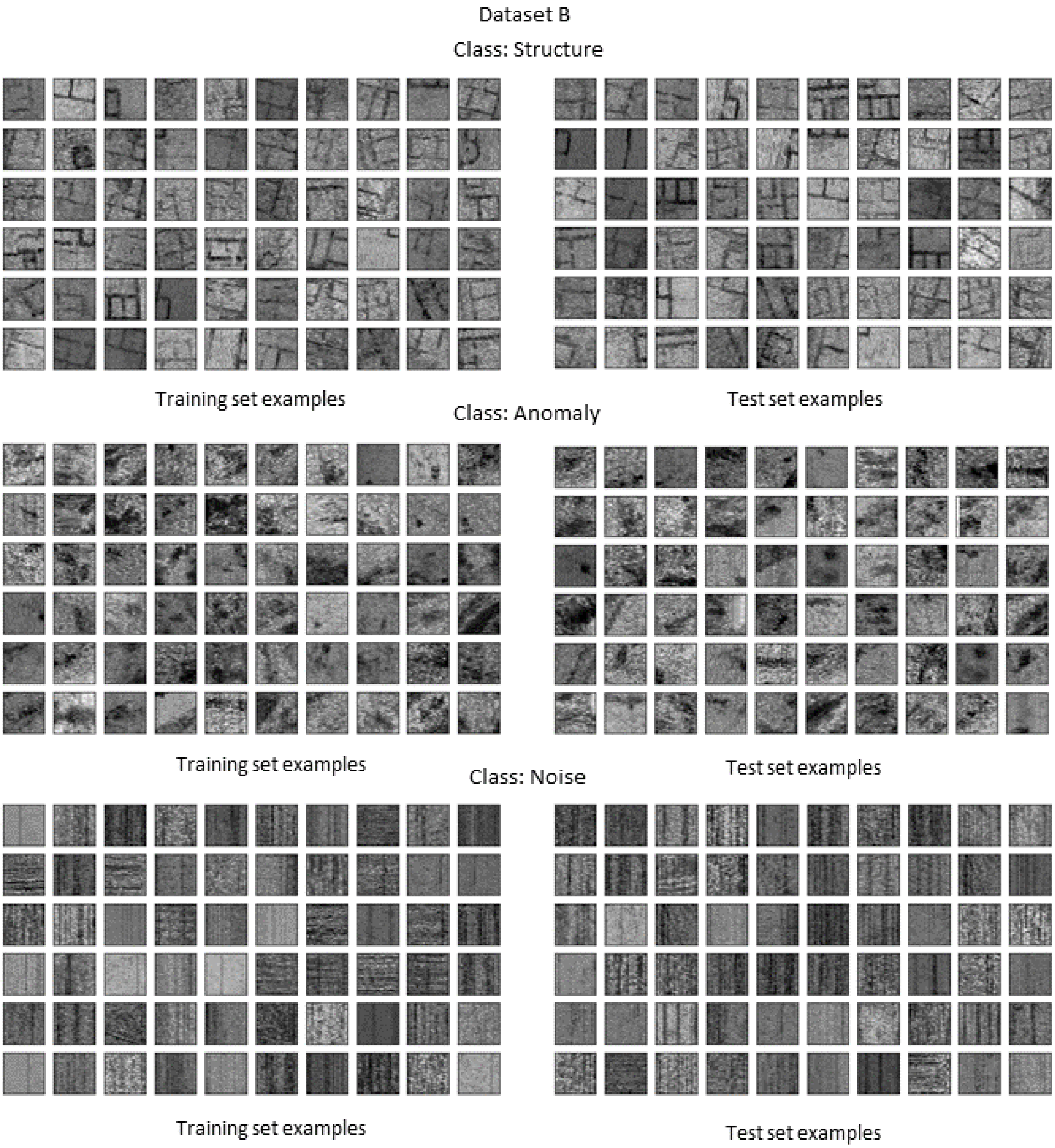


Figure 5.13: Training and test set examples from dataset-B. The split of test and train set is random following the 80%-20% rule.

5.5 CNN EXPERIMENTS

The two datasets, A and B, were used in a series of experiments to train CNN models. An overview is given in *Figure 5.14*. Following the evolution of popular CNNs architectures used in image classification, the *AlexNet* was chosen as the starting base of this investigation before moving into more complicated or deeper ones. In this series of experiments, two optimizers, the SGD and Adam, are tested and compared for both datasets.

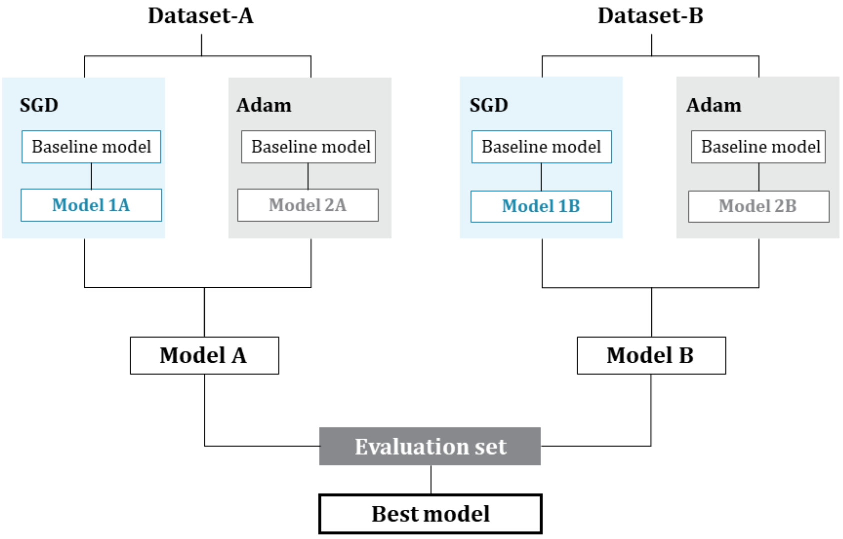


Figure 5.14: Overview of experiments and trials performed on the two datasets.

The trials begin on a model defined as the baseline, which is an implementation of AlexNet architecture. Further, techniques known to improve performance, such as batch normalization (BN), dropout, and image augmentation, are tested as well. The two optimizers are compared using the test set of the dataset used. The setup that returns the best results for each dataset is used to train the last two models, named Model A and Model B. These two models are used to evaluate the two approaches followed when constructed the datasets and examine which one generalizes better. This is performed on an evaluation set that has a small number of entirely new examples for each feature class. Details of the whole process are given in the following paragraphs.

5.5.1 AlexNet overview

AlexNet, proposed by Krizhevsky et al.[50], was the winning architecture of the ImageNet Large Scale Visual Recognition Challenge (LSVRC) 2012 competition, with significantly better performance over the runner-up (Top-5 error rate 15.3% over 26.2%). The input of AlexNet is a color image of 227x227x3 and has eight layers, five being convolutional (Conv) and three fully connected (FC). Among the key characteristics is that it uses ReLU activation functions (instead of $tanh$ that used to be the standard) after every Conv layer and after the two first FC layers. For the last FC layer, which feeds the output layer, the softmax activation function is applied, which produces a distribution over the 1000 class labels defined by the CIFAR-10 dataset used for training. CIFAR-10 dataset is a well-known dataset containing colored images used to train and test DL algorithms. Further, three overlapping max-pooling layers succeed the first, second, and fifth Conv layers. The authors used augmentation techniques on the input images, Local Response Normalization (LRN), and dropout regularization to improve training performance and reduce overfitting. LRN was applied after ReLU nonlinearities of the first and second Conv layers, while dropout, with probability 0.5, was applied at the first two FC layers. A schematic description of the architecture is presented in *Figure 5.15*.

As for training, the authors used Stochastic Gradient Descent (SGD) with a batch size of 128 examples, a momentum of 0.9, and a weight decay of 0.0005. The learning rate was initialized by 0.01 and was divided by 10 when the validation error stopped improving. Additionally, each layer's weights were initialized randomly from Gaussian distribution with zero mean and standard deviation 0.01. As for the biases, the second, fourth, and fifth Conv layers, and all FC layers, were initialized with 1’s, while for the remaining layers with 0’s.

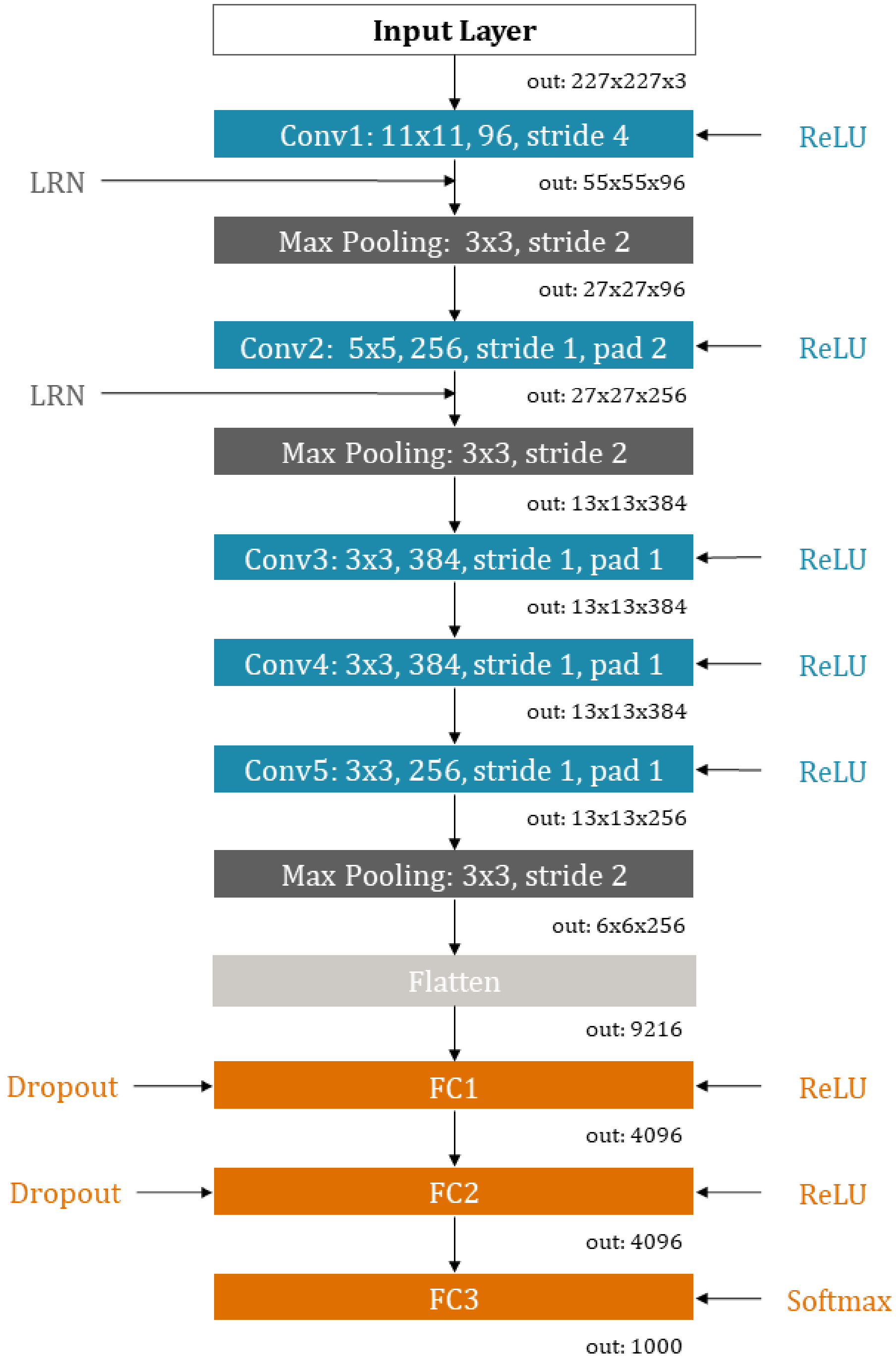


Figure 5.15: AlexNet architecture described in [50], including the activation functions applied on specific layers and training improving methods such as the Local Response Normalization and dropout. On each convolutional layer, information such as the kernels' size, kernels number, stride, and padding, when used, is shown. The shape of each layers' output is also denoted.

5.5.2 Baseline model implementation

In this research, the implementation of AlexNet was made using the Keras functional API. A model that served as a baseline for experimentation was constructed following the architecture presented in *Figure 5.15*, with the last FC layer adapted for the three classes. Weight and bias initializations were set in the same way as described in the original paper. Dropout and normalization techniques were excluded from the baseline model and applied to later trials to test their effectiveness and what works the best for the data used in this research. Details on the model parameters and hyperparameters are presented in *Table 5.4.* The Python script for this implementation is given in *Script B.1* of Appendix B.

Table 5.4: Chosen parameters and hyperparameters for the AlexNet baseline model. The number after each layer indicates its order in the architecture.

AlexNet baseline parameters and hyperparameters	
Input size	227x227x3
Conv1	Kernel size: 11x11, Number of filters: 96, Strides: 4, Padding = 0, Bias initializer: zeros, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$ Activation: ReLU
Conv2	Kernel size: 5x5, Number of filters: 256, Strides: 1, Padding = 2, Bias initializer: ones, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$ Activation: ReLU
Conv3	Kernel size: 3x3, Number of filters: 384, Strides: 1, Padding = 1, Bias initializer: zeros, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$ Activation: ReLU
Conv4	Kernel size: 3x3, Number of filters: 384, Strides: 1, Padding = 1, Bias initializer: ones, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$ Activation: ReLU
Conv5	Kernel size: 3x3, Number of filters: 256, Strides: 1, Padding = 1, Bias initializer: ones, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$ Activation: ReLU
FC1, FC2	Units: 4096, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$, Bias initializer: ones, Activation: ReLU
FC3	Units: 3, Weight initializer: Gaussian $\mu = 0, \sigma = 0.01$, Bias initializer: ones, Activation: softmax
Max Pooling	Kernel size: 3x3, Stride: 2

The initial settings for training the baseline were also taken from [50]. These are SGD as an optimizer, batch size of 128 examples, learning rate of 0.01, and momentum of 0.9. Weight decay and dividing learning rate by ten were avoided to test their effectiveness in later stages. As for the cost function, the categorical cross-entropy was chosen. The evaluation of the training performance was made per epoch, using the Keras accuracy class metrics, which calculates the predicted values' frequency that matches the correct label. The number of epochs was set to 100, using early stopping to prevent training the model when it overfits. The implementation of training the baseline is given in *Script B.4* of Appendix B. The optimizer is set through the Keras `optimizer` class, compiling the optimizer is done with the `.compile` method, while training is performed with the `.fit` method. Early stopping is implemented using the Keras `callbacks` API, setting an empirical patience value at 30 epochs while monitoring the validation loss. This means the training will be terminated if the validation loss will not improve over the next 30 epochs. *Table 5.5* summarizes the training settings of the baseline model.

Table 5.5: Training hyperparameters used for the baseline model.

Baseline training hyperparameters	
Optimizer SGD	learning rate: 0.01, momentum: 0.9,
Cost function	Categorical cross-entropy
Epochs	100
Early stopping	Patience: 30 epochs, monitor: validation loss

5.5.3 Improving performance & tuning hyperparameters

Several tests were performed aiming to find a combination of settings that leads to better classification for both datasets. The settings examined are methods and techniques used in various deep learning applications known to improve performance by reducing overfitting or tuning the model's hyperparameters for optimal performance. The tested settings used in this study are:

1. Data augmentation techniques
2. Dropout regularization and batch normalization methods applied to specific layers of the chosen architecture.
3. Settings, methods, and techniques that affect the optimizer and the training process. These are the optimizer type with the corresponding learning rate and batch size, early stopping, and reducing the learning rate in predetermined schedules.

The first step was to examine performance improvements using dropout and Batch Normalization (BN). BN was chosen instead of LRN, as it is known to perform better while it acts as a regularizer that prevents overfitting [43]. Following the baseline's training results, adjustments were made to the learning rate value, and tests of different setup combinations of BN and dropout applications were performed. For this step, training was held using 50 epochs, a number that was judged sufficient to understand from the learning curves the learning behavior and the need for improvements. For the setup with the best performance, the effectiveness of applying augmentation techniques was examined next.

Image augmentation was implemented using the `ImageDataGenerator` class and `flow_from_directory` method from Keras dataset preprocessing utilities. Several transformations are available, and tests were performed to find those that can produce real-like results. *Figure 5.16* shows the available transforms and an example of the images produced by their random combination. An example of importing the data and applying augmentation is given in *Script B.2* of Appendix B. Dropout and BN were implemented using the classes `Dropout` and `BatchNormalization` of the Keras layers API, respectively. Dropout layers were placed after the dense layers, FC1 and FC2, and prior to the ReLU layers. In addition, the BN layers were placed after the first two convolutional layers, Conv1 and Conv2, and prior to the ReLU layers. The implementation of adding both BN and dropout layers in the baseline model's architecture is given in *Script B.3* of appendix B.

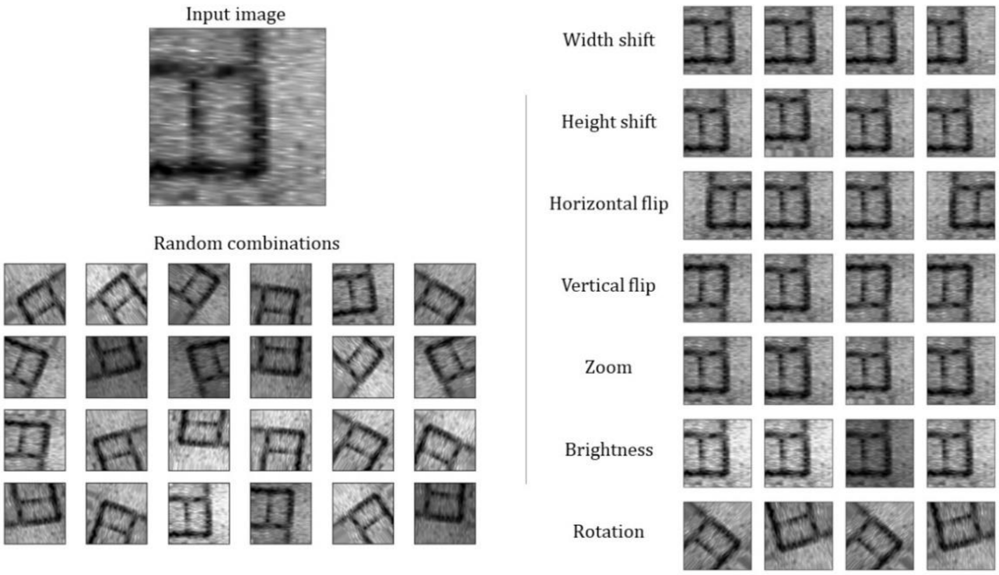


Figure 5.16: Examples of various image augmentation transforms. The input image will be replaced by a randomly generated one produced by the selected transforms.

Further improvements to the training setup were examined, including tuning the learning rate and batch size. Two optimizers were tested and compared, the SGD with momentum and Adam [158]. Adam is a popular optimizer based on adaptive estimates of lower-order moments and is often compared to SGD with momentum. Several studies exist reporting Adam performs better, like in [158] or in [208], while others are showing that SGD with momentum is a better choice, like in [162] for binary classification. In a different study, the authors state that even though Adam or similar adaptive optimizers tend to perform better at the initial training stages, SGD is the one that generalizes better [209]. For the abovementioned reasons, both optimizers were considered and tested. For Adam option's, the Keras default values were used, with a learning rate of 0.001.

The tuning process was performed to find an efficient combination of learning rate and batch size for each optimizer. These two hyperparameters significantly affect the training's performance and stability, and are crucial in the resulting model reliability, as discussed in [210]. For this task, the `RandomSearch` of Keras tuner library was chosen as an alternative to the grid search algorithm as it is faster and computationally less demanding. This process requires defining a hyper-model using the `model builder function` and set possible values, or a values' range for the hyperparameters under tuning. The random search algorithm receives the function's output, and the tuning process begins by calling the `.search` method. The learning rate values tested were chosen using a log scale. For SGD are [10^{-2}(default), 10^{-3},10^{-4}, 10^{-5},10^{-6}] since it was known from training the baseline that the default learning rate did not perform well when used a batch of size 128 but performed better when was reduced by 10 in the log scale. For Adam the values tested are [10^{-1} ,10^{-2}, 10^{-3}(default), 10^{-4}, 10^{-5}]. The batch sizes tested are [16, 32, 64, 128, 256], where 256 was the limit as memory issues were experienced with a batch size of 512. Tuning was performed for one batch size value at a time due to hardware limitations and memory issues. The training set was split into a validation set and a new training set using the 80%-20% rule. A full example of the tuning process for the case of SGD and a batch size of 32 is given in *Script B.5* of Appendix B. In this example, early stopping is used, setting the patience to 10 epochs to terminate training when the model starts overfitting.

The combinations with the highest score were used to train new models using the full dataset. During this training optimization step, early stopping was applied to refrain from training when overfitting occurs. Last, for the model with the best results, the potential for further improvements like reducing learning while training was also examined. This can be implemented using the `LearningRateScheduler` class of the Keras callbacks API.

5.6 MODELS METRICS & EVALUATION

After every epoch, training performance is evaluated using the validation accuracy and validation loss of `accuracyclass` by Keras `metrics`. The resulted weights for a completed epoch are used to make predictions on the test set, and the frequency in which the prediction matches the correct label is calculated. This frequency is eventually returned as binary accuracy by dividing the total amount of labels by the amount of those predicted correctly, while the validation loss is calculated based on the misclassified labels. The validation loss and accuracy were plotted with respect to the epochs creating learning curves that were used to evaluate the training process and decide on further action for improvements. Aside from the learning curves, a *confusion matrix* was also used to evaluate the classification performance of a model. The confusion matrices were proven useful as they could give information on the misclassified labels per class, thus finding which images' predictions were wrong and understanding the model's limits. Last, selected models (Model A and B) were used to make predictions on the evaluation set that has 100 GPR images that were intentionally excluded from the two datasets. These images derive from survey grids at Halos and Sissi archaeological sites, and examples are presented along with the results in the next chapter.

5.7 CLOSING REMARKS

This chapter described the methods and tools used to identify buried structures from GPR data. These were applied in four research stages: the GPR data collection, the data processing, the dataset preparation, and the training of CNNs for image classification. Data collection was conducted over a considerable time window period under various geophysical campaigns in different archaeological sites. All the collected data were reprocessed using MATLAB for the needs of this study. A workflow was established that works well for most data that suffer from signal attenuation. Even though it is not optimal for all case studies and their survey grids, as noise residuals exist, it highlights the buried structures, when present, satisfactorily. The central concept is to apply gain earlier in processing and then apply filters to remove the enhanced noise. Following this workflow resulted in producing depth slices that were interpreted and categorized according to the dominant features. Three main categories were observed: buried structures, geophysical anomalies of unknown targets, and residual noise in stripping forms. Identifying GPR signals under these three categories would guide the interpretation process quite well, reducing mistakes like interpreting linear residual noise as a structural remain. For this reason, datasets with the three classes were constructed.

Dataset construction was a tedious and challenging task as the gathered images were limited in number and not enough for training a CNN. A solution to this issue was to crop the chosen depth slices using a sliding window corresponding to 10x10m, which drastically increased the image’s number. The produced sub-images were manually organized under the correct label. The next issue faced was the way that the images should split into a training and test set. Since the test set should contain data unseen by the training set, a dataset was constructed using images from one case study solely for the test set, which resembles the real case scenario of adding new data in a model. An issue with this approach might be the sliding window approach, making the resulting test set not representative enough of the train set. The latter could lead to unstable training and, further, poor generalization. To test this scenario, a second dataset was constructed that randomly splits all the collected images into a training set and test set. This dataset might be more prone to overfitting due to the images' similarities because of the overlapping window approach. The training was performed in both datasets, following by comparing the best-resulted models to see which approach leads to better generalization.

For the training process on each dataset, a series of experiments and trials were made. AlexNet architecture was used for all cases, and the experiments focused on finding the best setup for the combined application of batch normalization, dropout, and image augmentation. For these setups, two popular optimizers were used and compared—the SGD with momentum and Adam to examine which one is more proper for the constructed dataset. Tuning was also performed on each optimizer for the learning rate and the size of the batches. A final model was trained for each dataset using the best combinations of setups and learning hyperparameters acquired from the previous steps. Early stopping was used to refrain from training when overfitting. The two models were compared and evaluated on images derived by two survey sites intentionally excluded from the learning process. The results and discussion on them are given in the following chapter.

6. TRAINING RESULTS & DISCUSSION

In this chapter, the results obtained during a series of training experiments are presented and discussed. These results provide useful insights into some research questions regarding the dataset construction and make it possible to evaluate the efficiency of some decisions made like the sliding windows crop approach to increase the image number. Further, through the various tests and comparisons made, some intuitions are provided regarding the training process for classifying GPR depth slices from archaeological prospection. These concern decisions upon the training optimizer, the impact of the learning rate and the batch size used, and how much performance improvements are expected when using image augmentation, batch normalization, and dropout techniques. First, the results of the training process using dataset-A are presented, followed then by dataset-B. The best-obtained models, named after each dataset as Model A and Model B, are then compared on an evaluation set, made with examples excluded from both datasets. The results suggested that Model B, trained with SGD and the dataset that the random split for training and test sets, performs better.

6.1 DATASET-A RESULTS

Dataset-A was constructed by using examples solely from the survey site of Elateia as a test set, as was described in Chapter 5. Several trials and experiments were performed, aiming to find the setup that can lead to better generalization. The trials started on a baseline, that is, the AlexNet architecture without normalization and regularization techniques. On this baseline, two optimizers, the SGD and Adam, are tested. Techniques like batch normalization and dropout were gradually applied using different setups and combinations. Additionally, the tuning of the batch size and learning rate was performed with the `randomsearch` algorithm. Image augmentation techniques were also applied during these training experiments to test their impact on the obtained results. The two optimizers showed different training behavior and, for this, were treated differently. All the choices made on the applied improvement techniques, their implantation in the AlexNet, and the tuning process were based on the learning curves of classification accuracy and loss function. These led into two models; Model 1, which was trained using the setups that gave better performance when used SGD, and Model 2, which was trained using the settings that were better for Adam optimizer. These choices and their outcome are presented and discussed in the following paragraphs.

6.1.1 Training with SGD

Training the baseline with SGD was challenging. Learning was not feasible when using the optimizer's initial settings, namely a learning rate of 0.01, a batch size of 128, and a momentum of 0.9. The learning curves of *Figure 6.1* present this behavior. The top chart presents the optimization learning curve that shows the categorical cross-entropy loss function, as resulted after an epoch was completed on both the training and test set. Loss is high and constant for both cases when, ideally, it should have dropped and stabilized at zero as epochs increase. This means that no optimization was achieved for training, and hence the model could not learn from data. Further, the test error is constant, meaning that no changes were made in the classification predictions. The bottom chart of *Figure 6.1* shows the learning curves of prediction accuracies calculated on the training and test set after an epoch was completed. The training accuracy fluctuates around 0.33, while the test set accuracy is constant on that number. This behavior is explained as having predicted only 1/3 of the total labels' count correctly, with no further improvements since the weights were not optimized during training. This is better shown in the corresponding classification matrix (*Figure 6.2*), which shows that all the images were classified as structures.

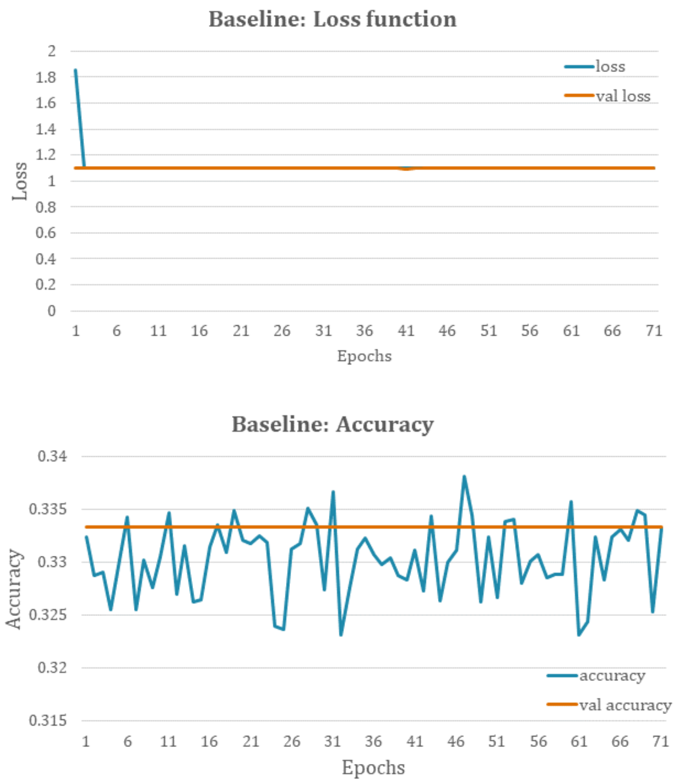


Figure 6.1: Learning curves produced by baseline training metrics showing that zero learning was achieved. On top is the categorical cross-entropy loss function as calculated on the training set (blue line) and test set (orange line). On the bottom is the accuracy of the training and test sets' classification predictions after an epoch was completed.

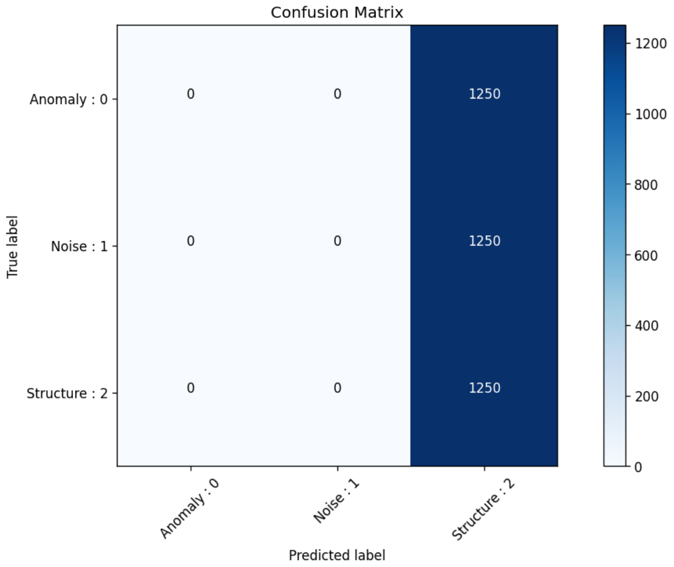
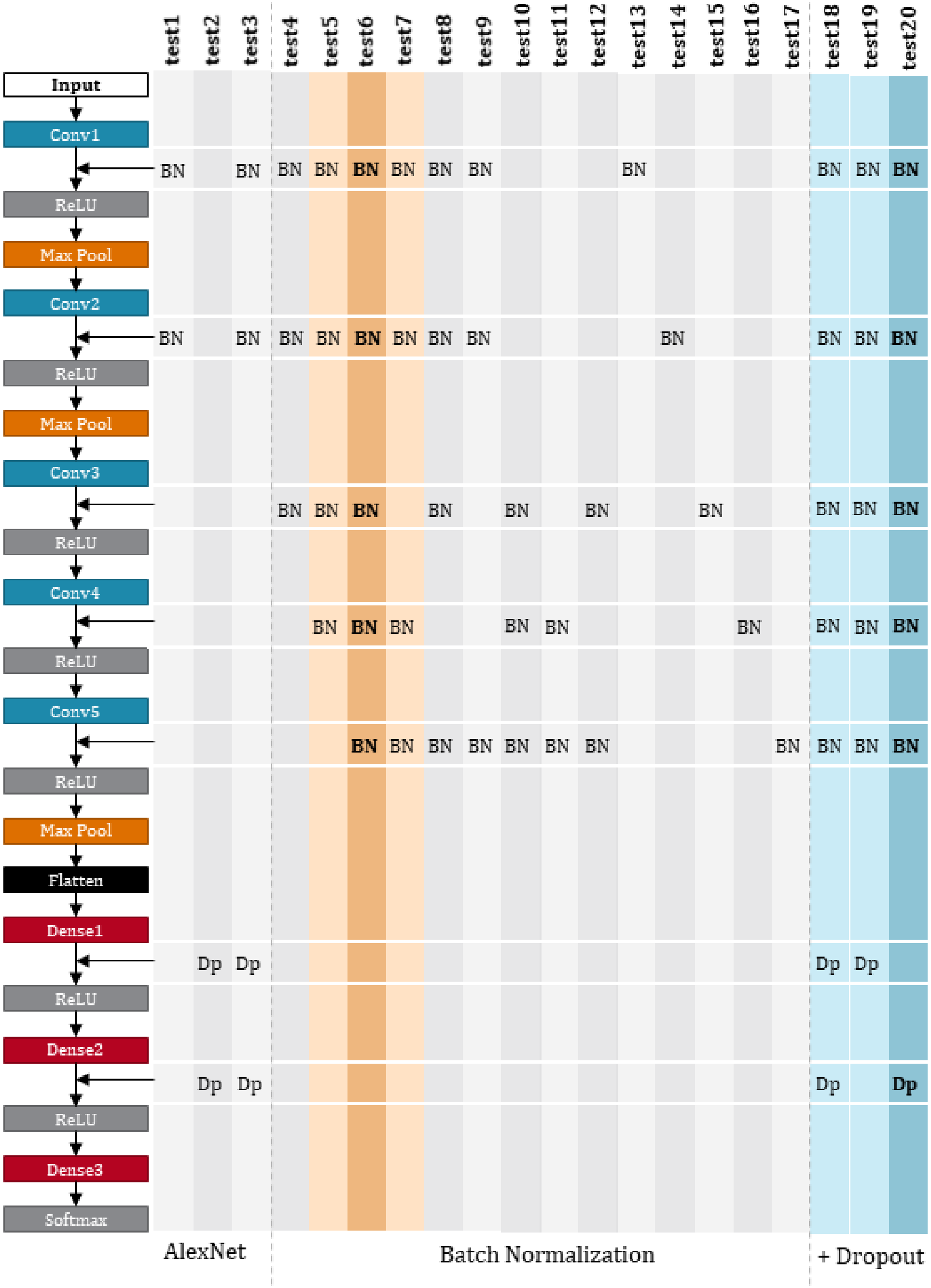


Figure 6.2: Confusion matrix for classification predictions on the test set using the baseline model. All the labels were classified as structures indicating that no learning was achieved during training. The classification accuracy is 0.33.

A possible reason for not achieving optimization during training, and hence learning, is the choice of inappropriate settings for the optimizer. Further, the network could also experience an internal covariate shift phenomenon due to the random weight initialization. The generalization and whether underfitting or overfitting exists cannot be evaluated at this point without trained weights. In the next step, different learning rates were used to train the baseline. The tested values were 0.001 and 0.0001, from the initial 0.01, returning the same negative result. Thus, several experiments were designed to test different setups of batch normalization, dropout, and learning rates for the optimizer aiming to achieve learning. The experiment's overview is presented in *Figure 6.3*. A total of 20 tests were performed. For all tests, batch normalization is implemented after the convolutional layers and prior to the activation layer, while dropout is implemented after the hidden layers and prior to the activation layer.

The first three tests were used to examine the effectiveness of BN and dropout individually, and then all together like they were implemented in the original article of AlexNet in [50]. The training was still not feasible under these setups using learning rates of 0.01 and 0.001, and further tests were conducted focusing on different setups of BN, as described by tests 4 to 17 of *Figure 6.3*. Training using the learning rate of 0.01 was still not feasible. It was eventually achieved by using the learning rate of 0.001 and applying batch normalization in most convolutional layers as described in tests 5, 6, and 7. The corresponding learning curves are presented in *Figure 6.4*. The charts of training loss and training accuracy show that test 6, which is applying batch normalization in all five convolutional layers, has the best training performance, followed by test 5 that is applying BN in the first four convolutional layers. With test 6 setup, the learning is faster. The resulting learning curves of test 7 (yellow lines) show that applying BN in the third convolutional layer significantly affects training accuracy, training loss, and learning speed. As for generalization, the validation loss curves show overall poor results, with test 6 performing better. It is less noisy than test 5 and exhibits less overfitting. Overfitting begins after 18 epochs, with a loss increment that stabilizes around one. The constant big gap between the validation loss and corresponding training of *Figure 6.4a* (1 vs. 0) is evidence of overfitting. There are no further improvements in classification predictions calculated on the test set, while the weights during training continue to fit the training set well. Thus, dropout was used as the next step in an attempt to improve the results of test 6 further. The testing setups include the application of dropout with a rate of 0.5 in the first two hidden layers (test 18), the first hidden (test 19), and the second hidden layer (test 20) (*Figure 6.3*), while the resulted learning curves are presented in *Figure 6.5*.



	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12	test13	test14	test15	test16	test17	test18	test19	test20
Input																				
Conv1																				
	BN		BN	BN	BN	**BN**	BN	BN	BN				BN					BN	BN	**BN**
ReLU																				
Max Pool																				
Conv2																				
	BN		BN	BN	BN	**BN**	BN	BN	BN					BN				BN	BN	**BN**
ReLU																				
Max Pool																				
Conv3																				
				BN	BN	**BN**		BN		BN		BN			BN			BN	BN	**BN**
ReLU																				
Conv4																				
					BN	**BN**	BN			BN	BN					BN		BN	BN	**BN**
ReLU																				
Conv5																				
						BN	BN	BN	BN	BN	BN	BN					BN	BN	BN	**BN**
ReLU																				
Max Pool																				
Flatten																				
Dense1																				
		Dp	Dp															Dp	Dp	
ReLU																				
Dense2																				
		Dp	Dp															Dp		**Dp**
ReLU																				
Dense3																				
Softmax																				
	AlexNet			Batch Normalization														+ Dropout		

Figure 6.3: Experiments and different setups of Batch Normalization and Dropout tested on the AlexNet architecture using SGD with a learning rate of 0.001, batch size of 128, and a momentum of 0.9. With orange is highlighted the setup of BN with the best performance, and blue is the best setup of adding dropout to chosen BN setup.

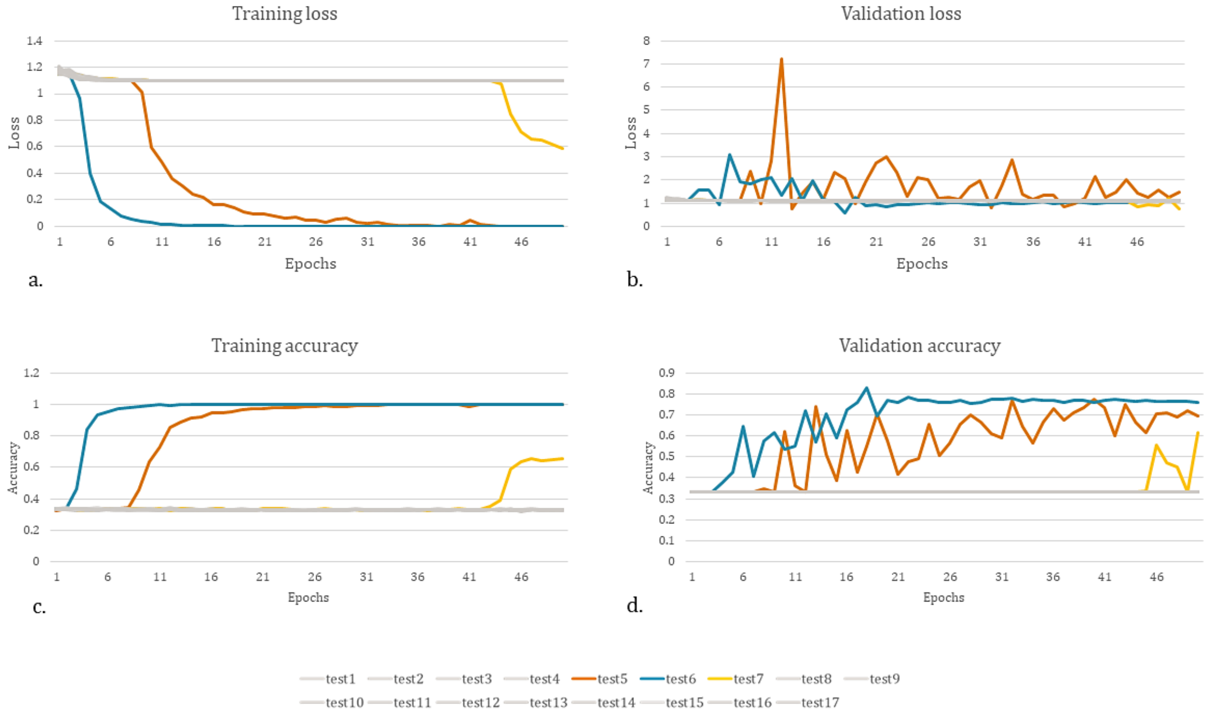


Figure 6.4: Learning curves that show improvements in learning. In a. is the loss calculated on the training set, b. is the loss calculated on the test set, c. is the classification accuracy calculated on the train set, and d. is the classification accuracy calculated on the test set. Charts a. and c. describe the training performance while charts b. and d. describe the generalization. The most efficient set is the one of test5 (blue line), where BN is applied in all five convolutional layers, followed by test 5 (orange), where BN is applied on the first four convolutional layers. Some degree of learning was achieved by the setup of test 7 (yellow) during the last epochs. Here BN was applied on the first two and last two convolutional layers. No learning was achieved for the rest setups (gray), resulting in the same flat learning curves as the baseline model.

Comparison of the learning curves of validation loss and accuracy of *Figure 6.5* shows that the three dropout setups (blue, grey, and red lines) perform similarly with performance improvements over the baseline with BN to all the convolutional layers (orange line). Adding dropout in the hidden layers seems to reduce overfitting as the validation loss decreases and validation accuracy increases for all three setups. The latter is improved from ~0.78 to ~0.84 after 20 training epochs. Before that, the curves appear noisy. For this reason, the setup of having dropout in the second hidden was chosen for being the less noisy one and exhibiting slightly higher accuracy in the classification predictions made on the test set. However, a gap between training loss and validation loss still exists (~0.75 vs. 0) without any signs of reduction, showing generalization issues still exist despite the improvements made. A possible reason is that the test set used is not representative enough of the training set. Whether this can be further improved remains to be answered while tuning the batch size and the learning rate.

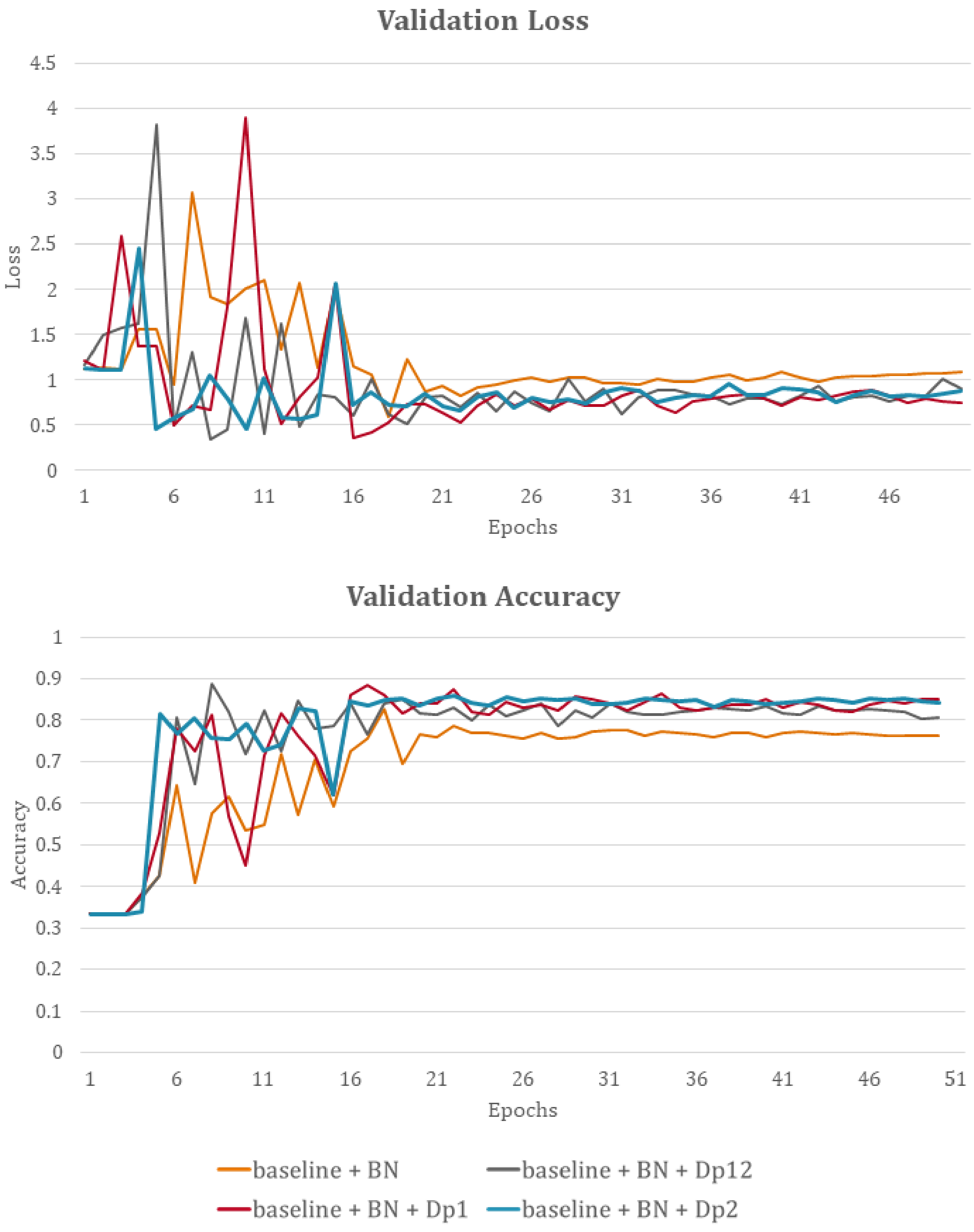


Figure 6.5: Learning curves showing the effect of adding dropout with a rate of 0.5. The orange curves are the ones that resulted from the chosen setup of BN (test 6) and are displayed for comparison reasons. The dark gray curves resulted from adding dropout to the first and second hidden layers (test 18), with red to the first hidden layer (test 19) and blue to the second hidden layer (test 20). The performance is overall improved, with loss decreasing and accuracy increasing for all three implementations. Adding dropout in the second hidden layer only produced less noisy curves and was chosen as the most efficient one.

On the next step to improve performance, the batch size and learning rate used by the SGD optimizer are tested for the chosen setup described previously. For simplicity, it will be referred to as Model 1. The algorithm of `randomsearch` by Keras `tuner` library was used to find the right combination of batch size and learning rate to retrain Model 1, as was described in Chapter 5. The results presented in *Table 6.1* show the final accuracy calculated on a validation set that was split randomly from the training set following the 80-20 rule. Thus, offset in the calculated accuracy is expected when used these settings of the dataset-A test set, where the images are entirely new. These tests were used as a guide for the next set of experiments.

Table 6.1:SGD - Classification accuracy calculated on a randomly split validation set using different batch sizes and learning rates. The highest accuracies per batch are highlighted with bold letters.

		Learning rates				
		0.01	0.001	0.0001	0.00001	0.000001
Batch sizes	16	0.975778	**0.988889**	0.984556	0.962556	0.9702222
	32	0.549778	**0.988556**	0.983444	0.971222	0.9426667
	64	0.346111	**0.989556**	0.978667	0.97	0.8914444
	128	0.723333	**0.982111**	0.979222	0.962556	0.8018889
	256	0.866778	**0.983667**	0.677222	0.393333	0.4531111

What can be seen from *Table 6.1* is that the learning rate of 0.001 returns the highest accuracies for all batch sizes, followed by the learning rate of 0.0001. Among them, the batch sizes of 16, 32, and 64 are the highest. The learning rate of 0.01 seems unstable as it returns very high accuracy for the smallest and then the largest batch size used. The lower learning rates exhibit overall very high accuracies for the batch sizes 16, 32, 64. Therefore, three new experiments are performed testing these batch sizes. For the training, 50 epochs were used, a learning rate of 0.001 and a momentum of 0.9. The resulting learning curves are presented in Figure 6.6, showing the learning curves resulted from batch size 128 for comparison reasons. The curves appear to be noisier as batch size decreases, and for the case of size 16, the performance becomes worse, increasing overfitting, while batch size 64 performs similarly to 128. Batch size 32 appears to be a better choice as it exhibits slightly higher classification accuracy. The loss shows no further improvements. In the next step, data augmentation is used with the image transforms described in Chapter 5, showing no further improvements (*Figure 6.7*). A possible explanation for this is that the applied image transforms are not representative of the test set's examples, and more tests are required.

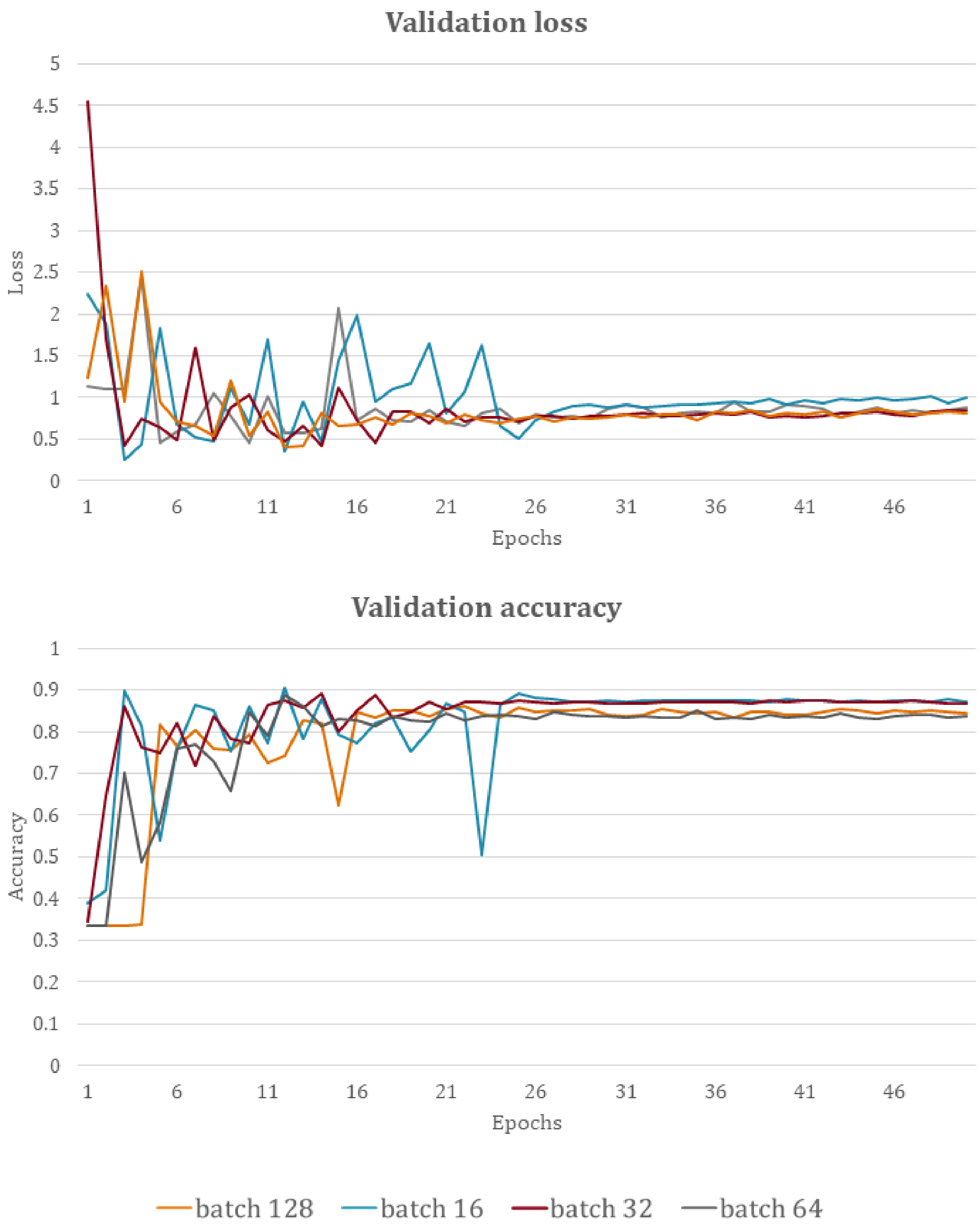


Figure 6.6: Learning curves obtained by training Model 1 using SGD with a learning rate of 0.001 and different batch sizes. Training with a batch size 32 performs slightly better.

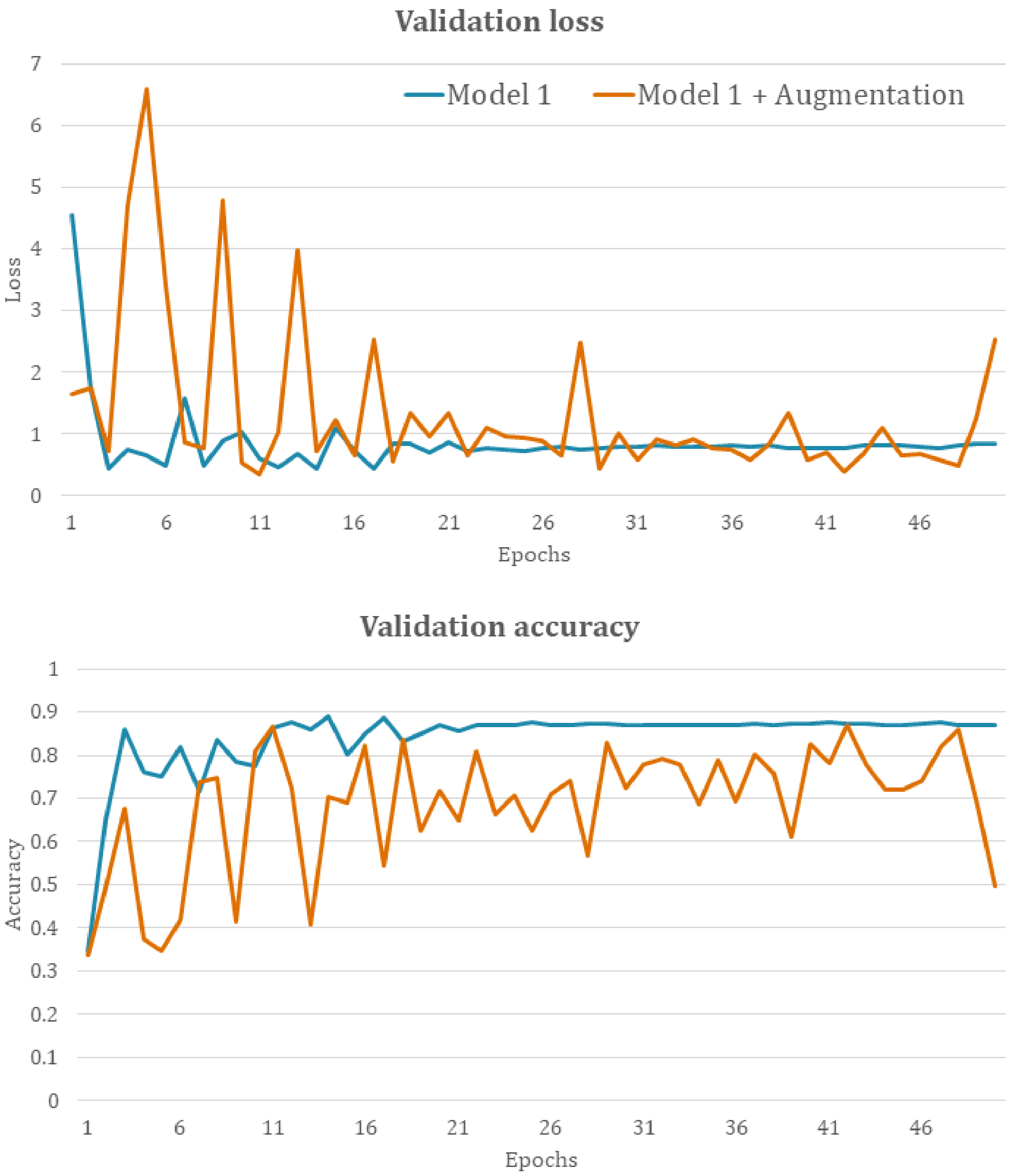


Figure 6.7: Comparative charts showing the obtained learning curves of Model 1 (blue line) and the application of image augmentation (orange line).

The setup that returned the best results for the tests and trials performed in this research is described by Model 1, i.e., using BN to all the convolutional layers and dropout with a rate of 0.5 to the second fully connected layers. The training settings for SGD are momentum 0.9, the learning rate of 0.001, and a batch size of 32. Despite there is room for improvements, the results were judged satisfactorily at this stage of this research. This setup will be compared to Adam's results described in the following subsection.

6.1.2 Training with Adam

The baseline's learning curves obtained with the Adam optimizer are presented in *Figure 6.8.* Without using any regularization or normalization technique, Adam's training achieved a maximum classification accuracy of ~0.8. The loss function appears noisy and exhibits signs of overfitting.

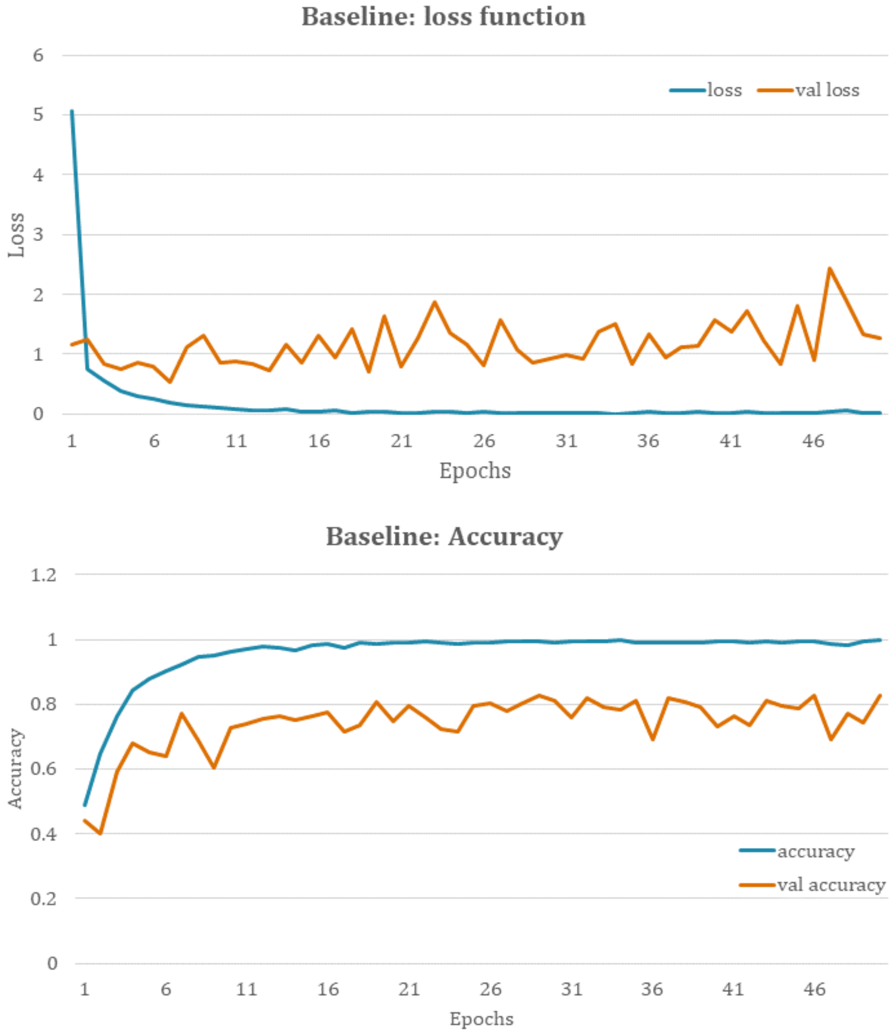


Figure 6.8: The resulted learning curves from training the baseline with Adam using a batch size of 128 and a learning rate of 0.001.

In the next step to improve learning, the BN and dropout application is examined, using the setups from test1 to test17 described in *Figure 6.3*. Learning was achieved using the setups of test1, test8, test9, and test10. The corresponding learning curves are presented in *Figure 6.9*, showing poor performance. Both validation accuracy and loss are very noisy with poor generalization, indicating that applying BN is not beneficial in this case. The learning curves produced by the test8 setup show a better accuracy curve after training for 30 epochs but might be due to overfitting as the validation loss remains constant.

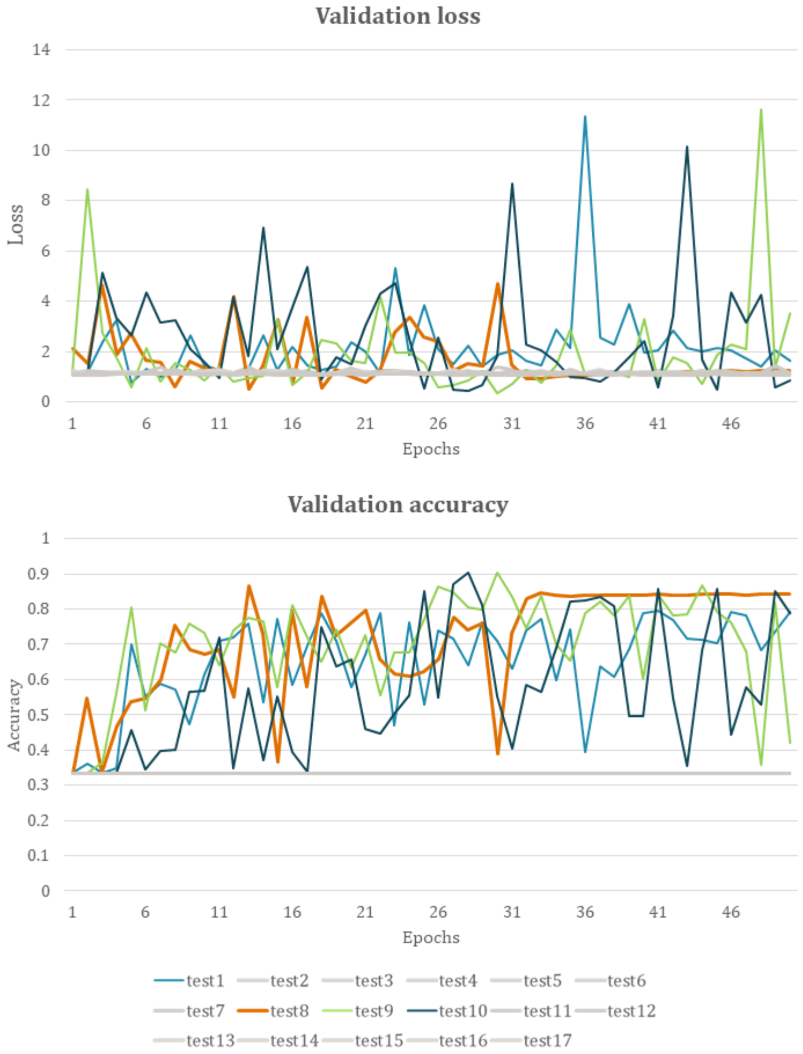


Figure 6.9: The effect of applying BN in different layers in AlexNet. The gray colors indicate that no learning was achieved using this particular setup. For test1, BN is applied to convolutional layers 1 and 2 (blue line). For test8 to convolutional layers 1,2,3 and 5 (orange line). For the test9 to convolutional layers 1,2, and 5 (green line). Last for test10 (dark blue line) to convolutional layers 3,4 and 5.

Dropout was next applied to the fully connected layers of test8 to check whether the generalization can be improved. Same as the SGD case, three different dropout setups were tested, to hidden layers 1 and 2, only to hidden layer one, and then only to hidden layer 2. The comparative charts in *Figure 6.10* show no improvement in the generalization, but on the contrary, the learning curves of both validation loss and accuracy appear noisier with large fluctuations. Further, for the case of applying dropout in the second hidden layer only, it led to achieving no learning.

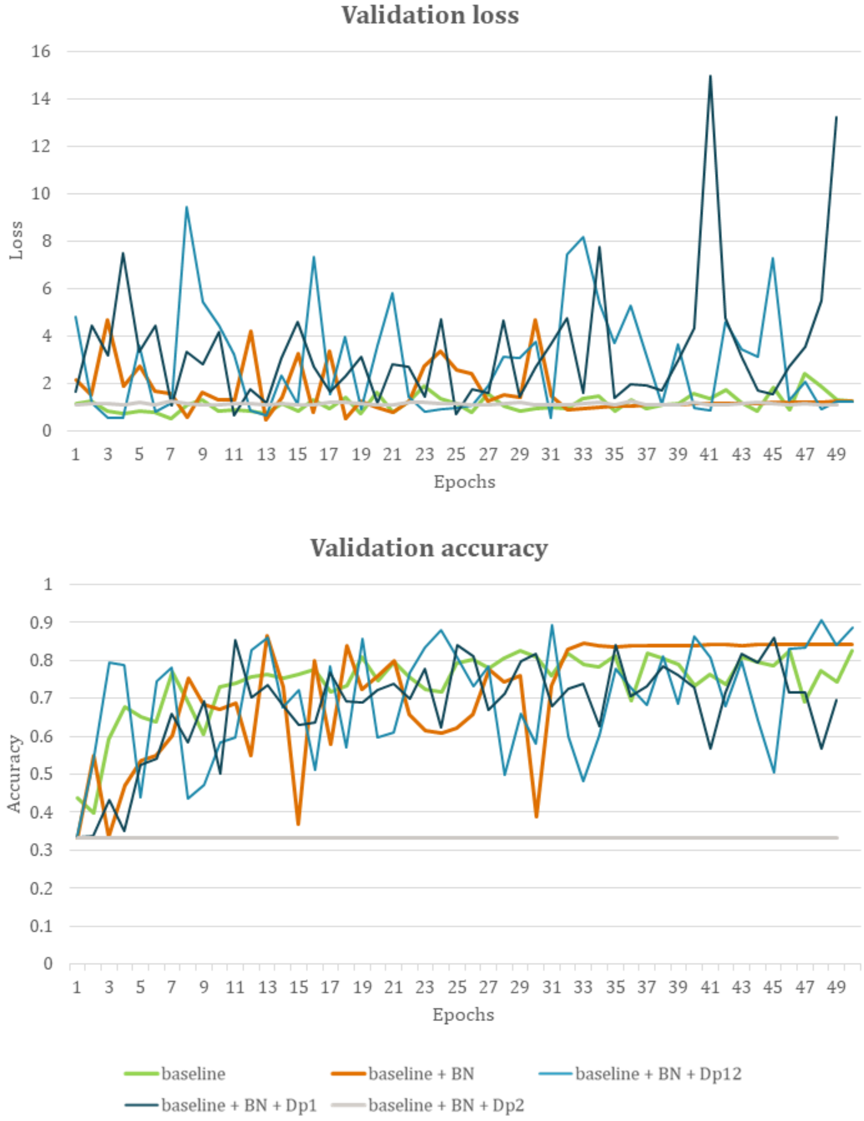


Figure 6.10: Comparative charts showing the effect of adding dropout with a rate of 0.5 in different fully connected layers setup. The BN setup used applies to convolutional layers 1, 2, 3, and 5. Dropout is tested on the 1st and second fully connected layers (light blue line), the 1st fully connected layer (dark blue line), and the 2nd fully connected layer (gray line). The latter resulted in the zero-learning case.

Up to this point, the baseline is the model that performs the best as the application of BN and dropout is not beneficial and more research is required. In an effort to improve the performance of the baseline trained with Adam, different learning rates and batch sizes were tested using the random search algorithm. The resulted accuracies are summarized in *Table 6.2*. The highest validation accuracy was achieved using the batch size of 64 and a learning rate of 0.001. High accuracies are also observed for larger batch sizes and lower learning rates. Additionally, the lowest learning rate returned very high accuracies for all the batch sizes.

Table 6.2: Adam - Classification accuracy calculated on a randomly split validation set using different batch sizes and learning rates. The highest accuracies per batch are highlighted in bold.

		Learning rates				
		0.1	0.01	0.001	0.0001	0.00001
Batch sizes	16	0.333333	0.334778	0.333333	0.333333	**0.9335556**
	32	0.333333	0.401889	**0.927222**	0.544778	0.9218889
	64	0.333333	0.333333	**0.946444**	0.737778	0.9324445
	128	0.333444	0.333333	0.910111	**0.943778**	0.9388889
	256	0.333333	0.516444	0.914111	**0.933222**	0.919

For the next step, the baseline line is retrained using batch size 64 with the learning rate of 0.001, and 128 using the learning rate of 0.0001 and then the lowest one of 0.00001. The resulted validation learning curves are presented in the charts of *Figure 6.11*. Compared to the baseline line model that was trained using batch size 128 and the learning rate of 0.001, reducing the learning rate to 0.0001 performed slightly better (blue line), resulting in higher classification accuracy. However, overfitting is still observed. Using a batch size of 256 with a learning rate of 0.0001 returned similar results (black line) but has lower classification accuracy. Last, the resulted learning curves of using a learning rate of 0.00001 and batch size 128 indicate that more training is required. Despite being noisy, the validation accuracy has an increasing trend, while the validation loss has a decreasing one. Here, the training settings of the 0.0001 learning rate and a batch size of 128 were selected due to the higher classification accuracy. This setup, named Model2, was retrained using augmentation to test whether the generalization improves. From the resulting learning curves, the overfitting and the classification accuracy are both reduced. However, no significant improvement is observed as the learning curves became noisier (*Figure 6.11*).

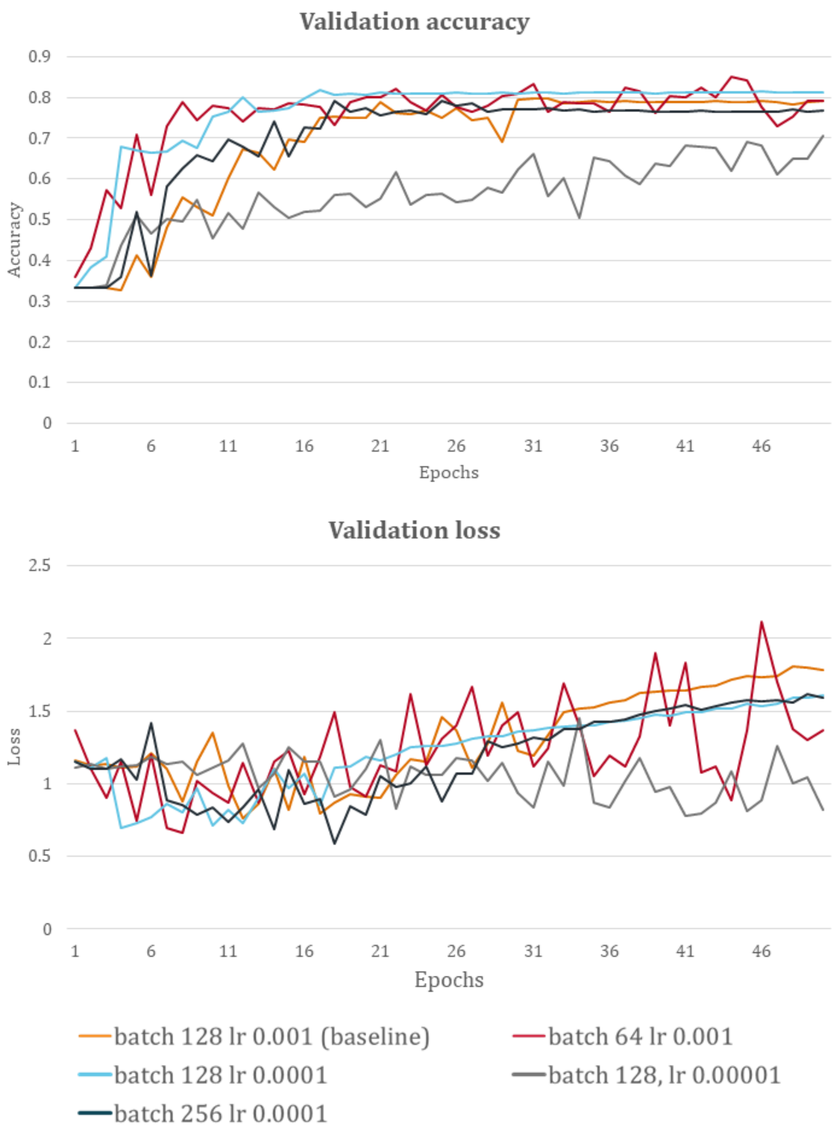


Figure 6.11: Comparative charts showing the effect of different batch sizes and learning rates when training the baseline model with Adam.

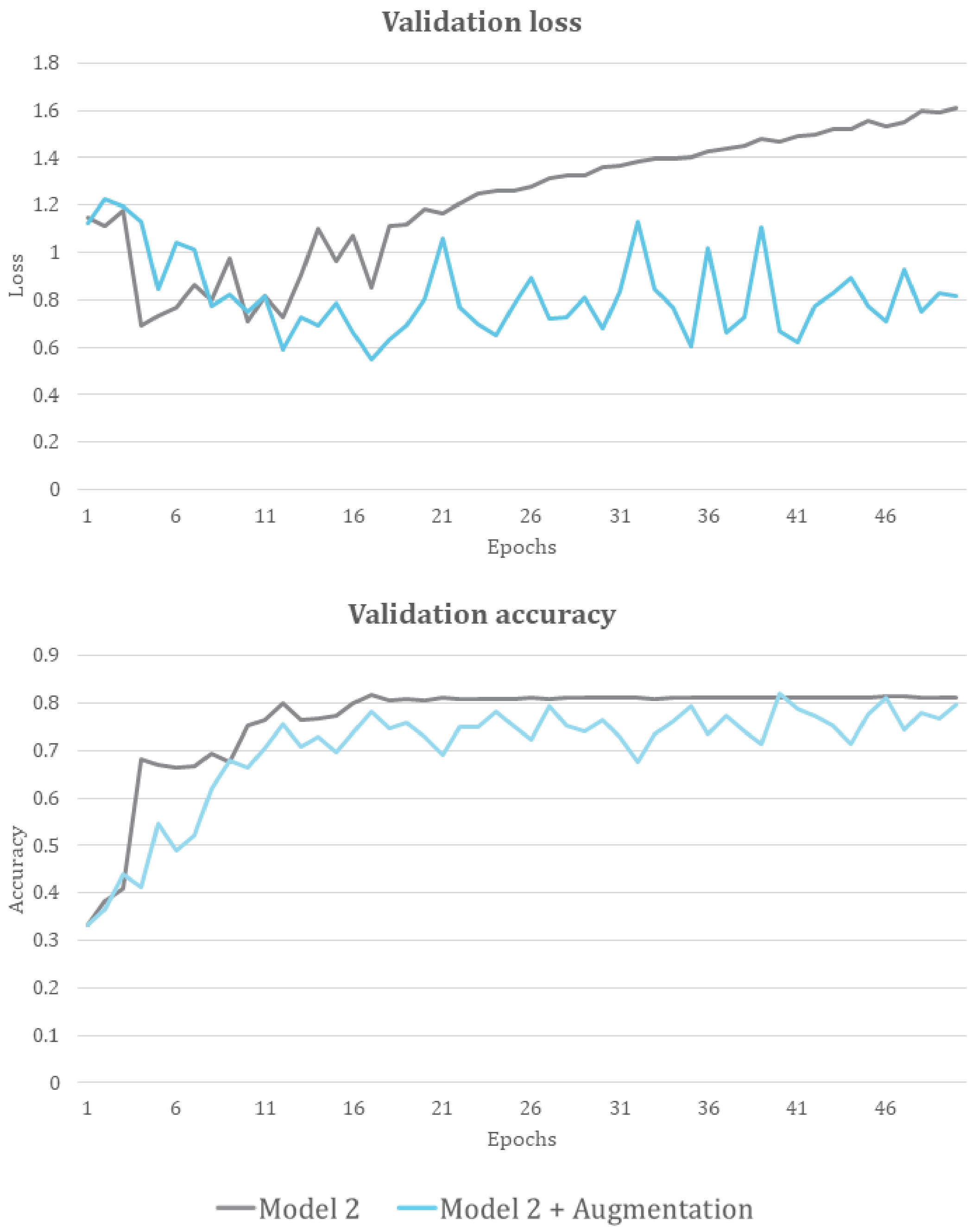


Figure 6.12: The effect of augmentation techniques on Model 2A using Adam optimizer. The validation loss chart shows that overfitting is reduced, but the curve is noisy. The validation accuracy chart shows a drop in classification accuracy when used the augmentation techniques.

6.1.3 SGD and Adam comparison

The training process of AlexNet with dataset-A using the SGD and Adam optimizers presented entirely different learning behaviors. On the one hand, SGD required applying BN and dropout techniques to achieve learning and classification performance, which improved with learning rate and batch size hyperparameters tuning, reaching validation accuracy around ~0.85. Image augmentation was not beneficial in reducing overfitting or increasing validation accuracy further. On the other hand, with Adam optimizer, learning was possible without the need to use BN and dropout. However, the results could not further be improved using the methods tested here, and further work is required on this matter. The validation accuracy was further improved, reaching ~0.8 after reducing the learning rate. Image augmentation did reduce overfitting but decreased the classification accuracy on the test set. Due to the differences mentioned above, the best-obtained models using each optimizer are chosen and compared. The comparison is made on the learning curves for classification accuracy and loss calculated on the test set. For SGD, the AlexNet model with the best performance resulted by applying BN to all five convolutional layers, a dropout rate of 0.5 to the second fully connected layer, while training options were a learning rate of 0.001, the momentum of 0.9, and a batch size of 32. For Adam, the AlexNet model with the best performance was obtained without any BN and dropout application using a learning rate of 0.0001 and a batch size of 128. The corresponding learning curves are presented in *Figure 6.13*. The SGD approach performs the best, given the setups tested in this dissertation, with higher accuracy on the test set and less overfitting. Indications that the SGD approach is more accurate than Adam's are also shown when comparing the tuning results of *Table 6.1* and *Table 6.2* for SGD and Adam, respectively.

This is also shown in the confusion matrices presented in *Figure 6.14* for the two models. Adam's model resulted in a classification accuracy of 81.20%, while the SGD model has 86.91%. Both models classified most of the labels correctly under the structure and noise classes, with SGD being slightly more accurate. The main difference is observed for the anomaly class, where most mistakes were made from both models, with SGD being better. These mistakes mainly concern examples of anomalies that were misinterpreted as structures and then examples of noise that were misinterpreted as structures.

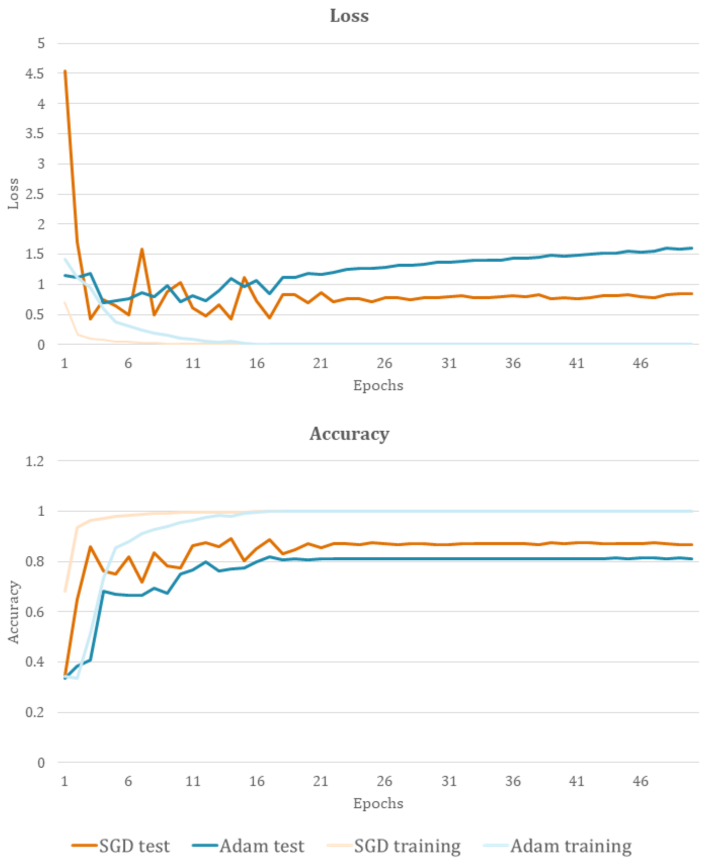


Figure 6.13: Comparative charts of loss and accuracy learning curves of the best-obtained model that were trained with SGD (orange) and Adam (blue). With faded orange and blue lines are the corresponding accuracy and loss curves on the training set.

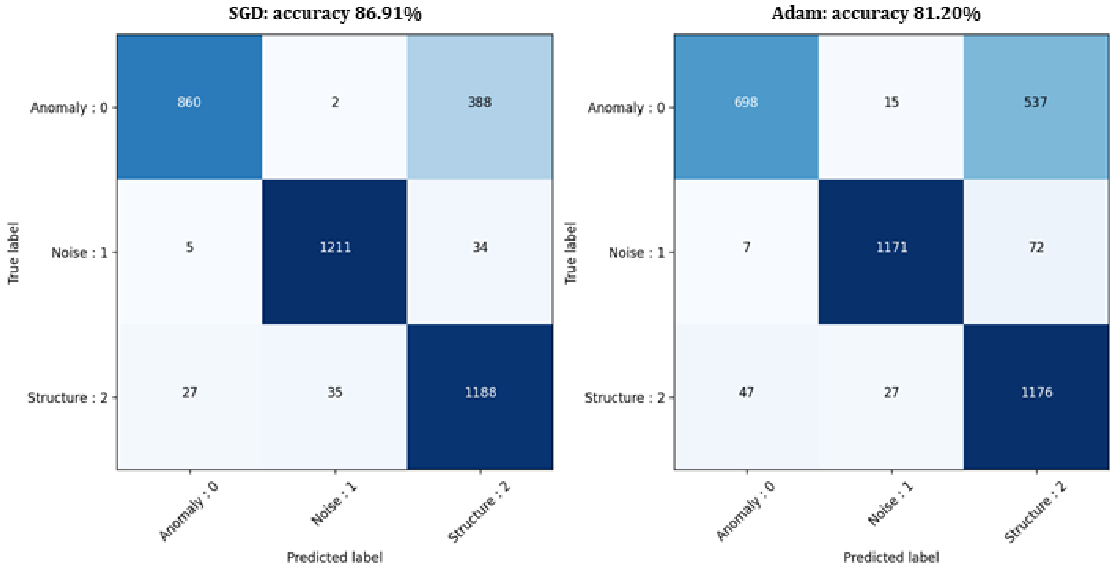


Figure 6.14: Confusion matrices resulted from training with SGD (left) and Adam (right).

6.1.4 Model A

The SGD approach that uses BN and dropout is selected as the most efficient one for dataset-A. The final model summary, named Model A, is presented on *Table C.1* of Appendix C. The SGD optimizer is used for training, with momentum 0.9, a learning rate of 0.001, and a batch size of 32, as discussed previously. Additionally, a stepped learning rate scheduler was set that reduces the learning rate by half every ten epochs. The training was performed for 100 epochs using early stopping, applied on the validation loss, and 20 epochs patience to ensure no further improvements exist before stopping. The resulted learning curves are presented in *Figure 6.15*. The training was terminated after 30 epochs due to overfitting that begins after the 11th epoch. This is also the point that the best performance is observed, with classification accuracy that reaches 91% and validation loss that drops to 0.4.

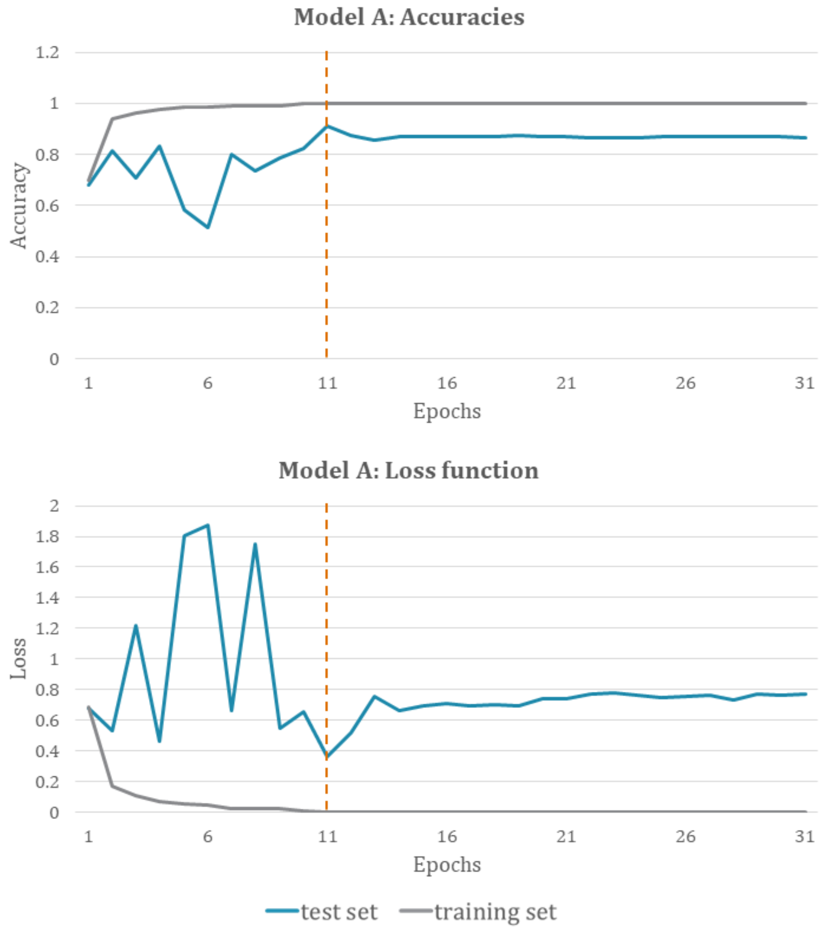


Figure 6.15: Learning curves of Model A. The highest validation accuracy and lowest validation loss are observed in epoch 11 (orange dashed line).

Next, the weights at the 11th epoch were used to calculate the test set's confusion matrix to evaluate classification (*Figure 6.16*). For the anomaly class, 1006 out of 1250 labels were correctly predicted. For the rest, most of them were misclassified as structures. This negatively affects the accuracy of the structure class and indicates the dataset requires reworking. The noise class predictions were the most accurate, having 1231 out of 1250 labels classified correctly. For the structure class, 1182 out of the 1250 labels were correctly predicted. Thirty of them were misclassified as anomalies and another 38 as noise. A few examples of misclassified labels are presented in *Figure 6.17*. Most of the mistaken anomalies as structures present some linearity or have corner-like shapes. The misclassified structures as anomalies are mainly intermittent linearities characterized by strong amplitudes, while the ones mistaken as noise have mainly weaker amplitudes derived from deeper levels. As for noise, the misclassified images are somewhat fuzzy and perhaps not representative enough of the corresponding ones in the training set. This information is found particularly useful for future improvements on the dataset used for training.

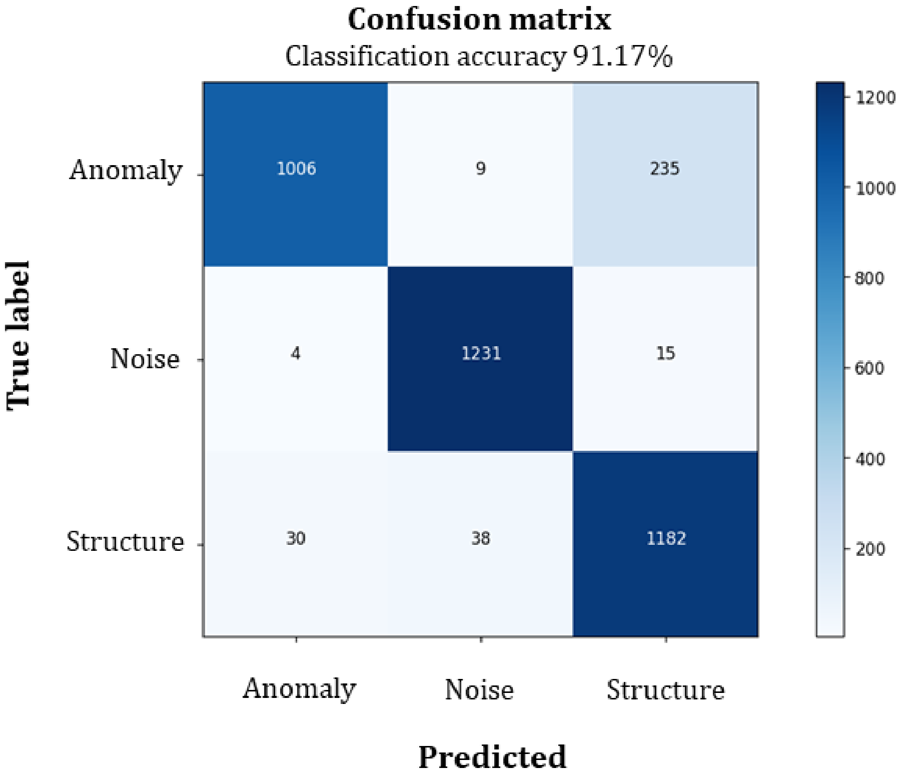


Figure 6.16: Confusion matrix calculated using the weights of Model A that returns 91.17% classification accuracy.

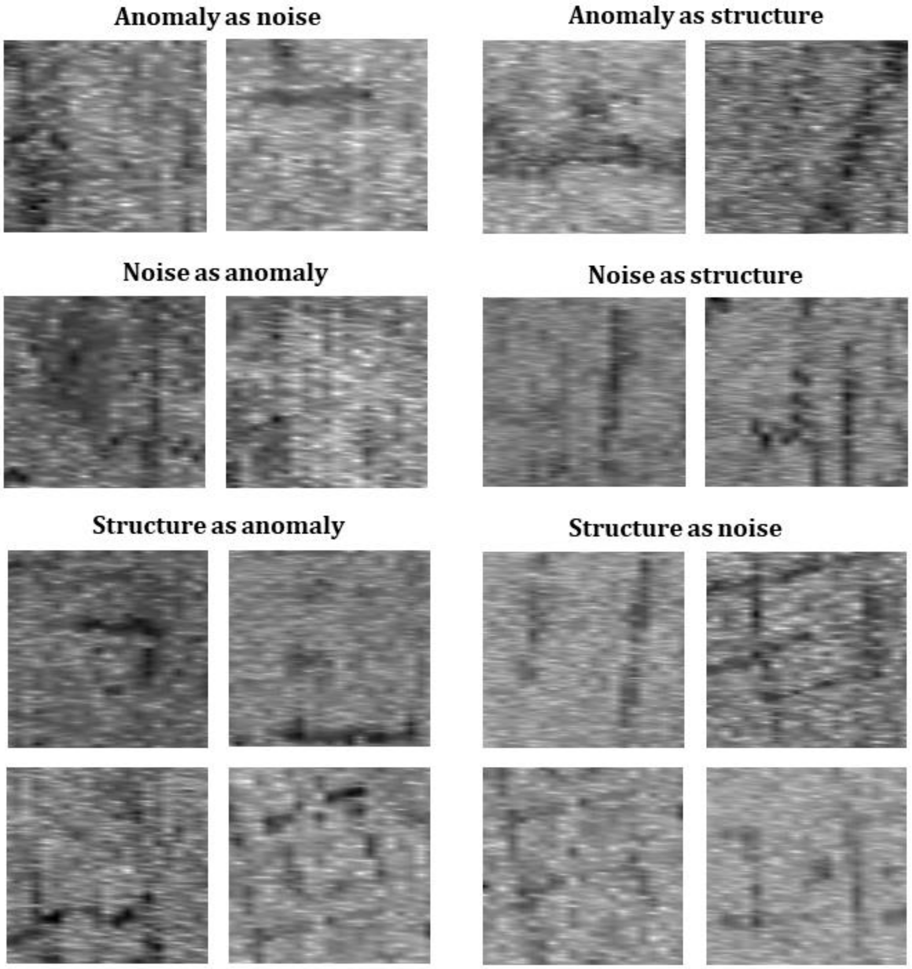


Figure 6.17:Examples of misclassified labels using Model A.

6.2 Dataset-B results

Dataset-B was constructed by using all the selected GPR c-scans and splitting them randomly into a training set and test set, as described in Chapter 5. Due to this study's comparative nature, the random split was performed once, and the resulted sets were used in common for all the tests performed to find and study the misclassified examples. The training process using dataset-B was performed in the same way as with dataset-A. That includes training the baseline with both optimizers and then applied BN and dropout to find the setup that yields better performance for each one of them. Tuning the batch size and learning rates was performed next for the chosen setups, and the results were used to train Model 1 for the SGD approach and Model 2 for Adam's approach. The two models were compared on the dataset-B test set, and the training settings, which gave the best results, were used to train the final model named Model-B.

6.2.1 Training with SGD

The training process using SGD on dataset-B had the same behavior as the corresponding one of dataset-A; namely, learning was not possible without BN. The resulted learning curves obtained from training the baseline model exhibiting this behavior are presented in *Figure 6.18*. Hence, BN and dropout were applied in different setup combinations, shown in *Figure 6.19*, to find the one that makes learning feasible. All the setup tested were trained for 50 epochs, using a learning rate of 0.001 and a momentum of 0.9. Same as dataset-A case, no learning was feasible for the initial learning rate of 0.01. Learning was achieved for the setup describe by test5 and test6. The former applies BN to the first four convolutional layers, while the latter to all of them. The corresponding learning curves (*Figure 6.20*) were advised in order to choose between the two setups.

Best performance is achieved when BN is applied in all the convolutional layers. Learning is faster, as both the training accuracy and loss function curves are showing in *Figure 6.20a* and *Figure 6.20c*, respectively. Further, the validation curves have shown that generalization is also better when applied BN to all the convolutional layers. They appear less noisy, with less overfitting and faster convergence to zero (*Figure 6.20b*), while the highest accuracy is also achieved faster (*Figure 6.20d*).

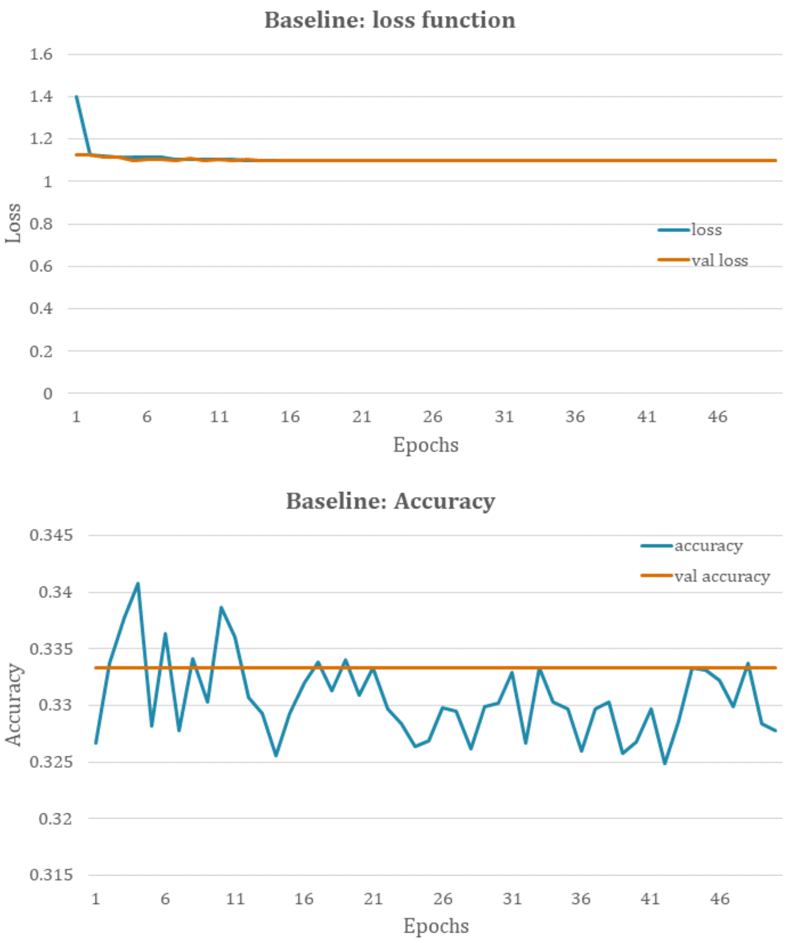
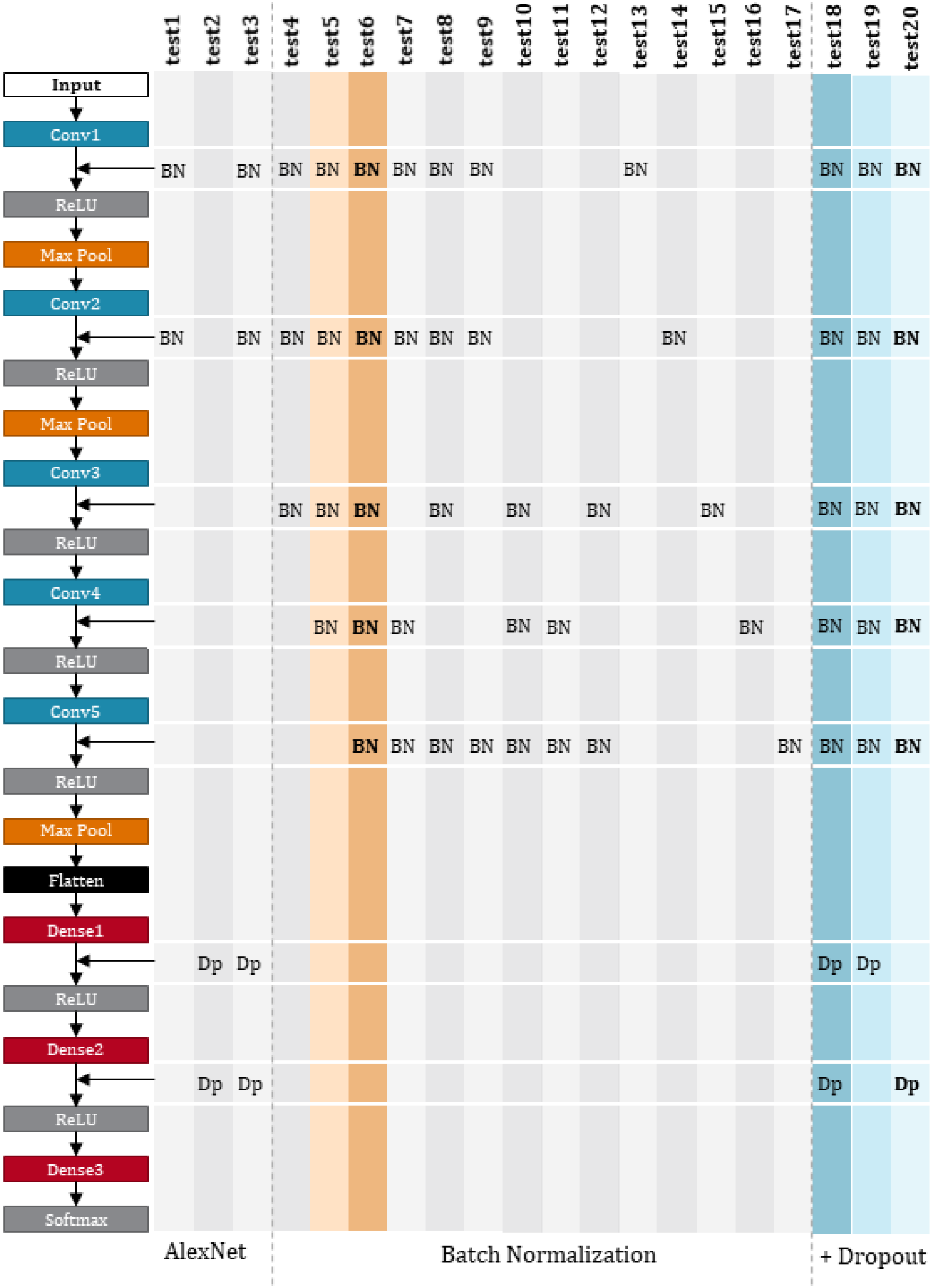


Figure 6.18: Learning curves of training the baseline showing that no learning was achieved. SGD was used with a learning rate of 0.01 and a batch size of 123. On top is the categorical cross-entropy loss function as calculated on the training set (blue line) and test set (orange line) after an epoch was completed. On the bottom is the accuracy of the training and test sets' classification predictions after an epoch was completed.



	test1	test2	test3	test4	test5	test6	test7	test8	test9	test10	test11	test12	test13	test14	test15	test16	test17	test18	test19	test20
Conv1	BN		BN	BN	BN	**BN**	BN	BN	BN				BN					BN	BN	**BN**
Conv2	BN		BN	BN	BN	**BN**	BN	BN	BN					BN				BN	BN	**BN**
Conv3				BN	BN	**BN**		BN		BN		BN			BN			BN	BN	**BN**
Conv4					BN	**BN**	BN			BN	BN					BN		BN	BN	**BN**
Conv5						**BN**	BN	BN	BN	BN	BN	BN					BN	BN	BN	**BN**
Dense1		Dp	Dp															Dp	Dp	
Dense2		Dp	Dp															Dp		**Dp**

Figure 6.19: Various setups and tests for training dataset-B with SGD with a learning rate of 0.001 and a momentum of 0.9. Gray colors indicate that no learning was achieved when used the current setup. The BN setup that returned the best performance is highlighted with intense orange color, while intense blue color highlights the final setup that includes dropout regularization.

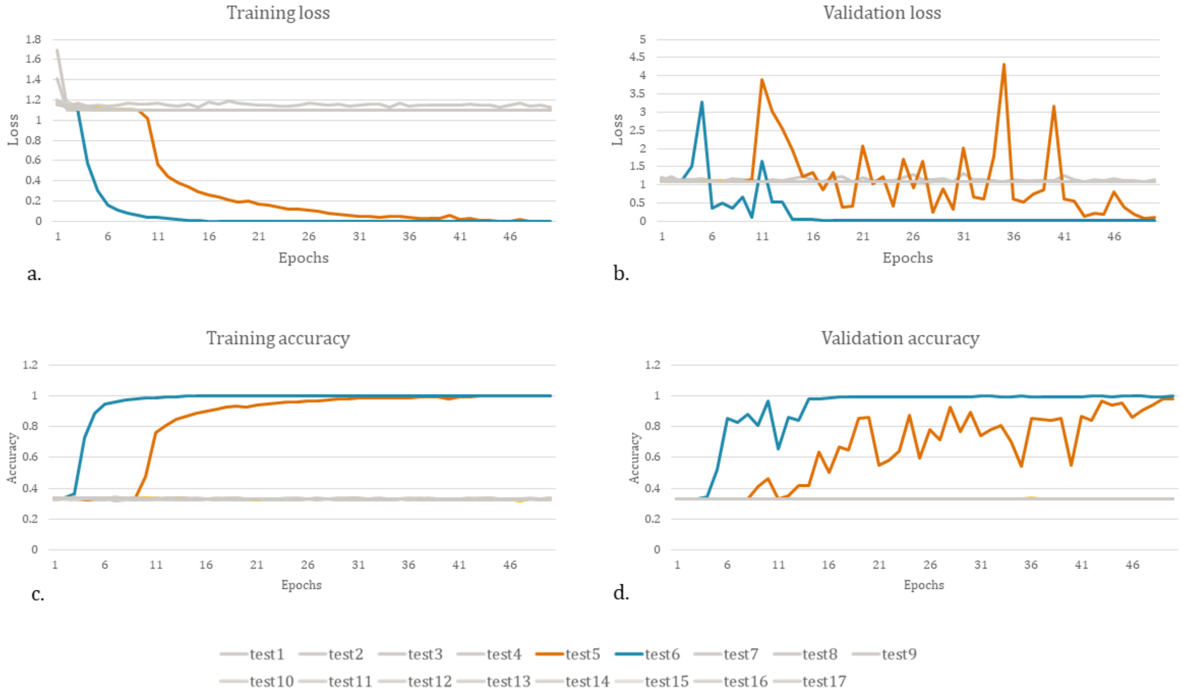


Figure 6.20: Learning curves that show improvements in learning where a. is the loss calculated on the training set, b. is the loss calculated on the test set, c. is the classification accuracy calculated on the train set, and d. is the classification accuracy calculated on the test set. Plots a. and c. describe the training performance while charts b. and d. describe the generalization. The most efficient set is the one of test6 (blue line), where BN is applied in all five convolutional layers, followed by test 5 (orange), where BN is applied on the first four convolutional layers. No learning was achieved for the rest setups (gray), resulting in the same flat learning curves as the baseline model.

The BN setup of test6 was used for the next series of tests that examine the effect of applying dropout in the first two hidden layers of AlexNet. Same as dataset-A, three setups are tested, applying dropout in both hidden layers, only the first and only the second, as described by tests 18 to 20 in *Figure 6.19,* respectively. Comparing the resulting learning curves with those of the best performed BN setup (orange colors in *Figure 6.21*) show that the performance is slightly improved when applied dropout in both hidden layers (dark gray colors in *Figure 6.21*). Applying dropout in the first hidden layer only appears to worsen performance, while applying in the second hidden layer only does not show any significant improvement. Hence the setup described by test18, which is applying BN to all convolutional layers and dropout to the first and second hidden layers, was used in the next step, which tunes the training hyperparameters.

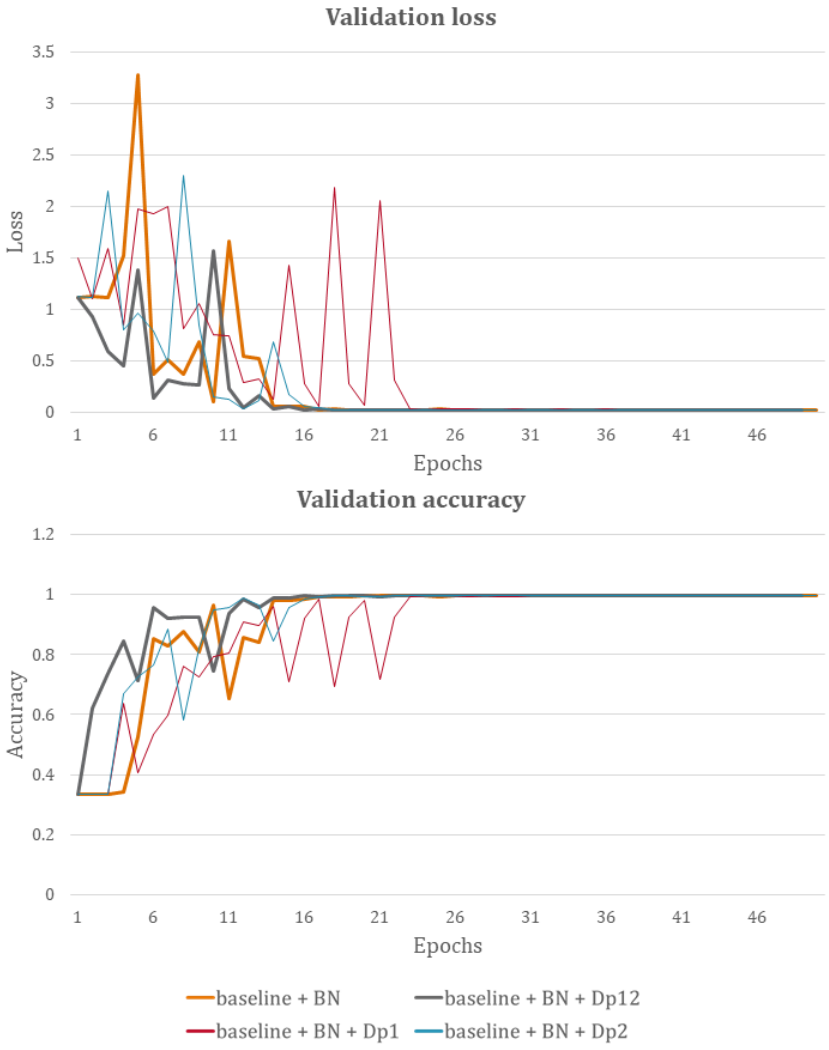
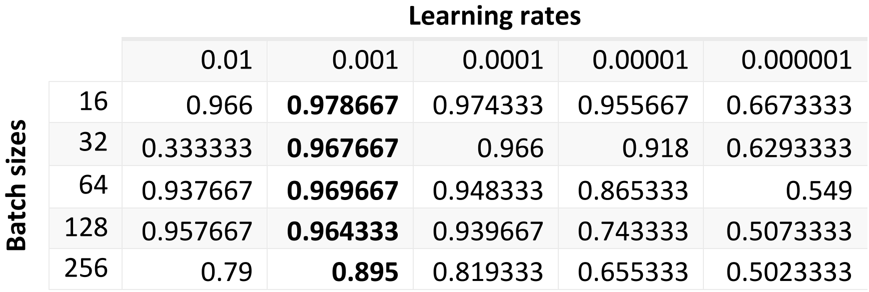


Figure 6.21: The effect of applying dropout with a rate of 0.5 in the best-performed BN setup. With orange color is the BN setup's resulted learning curves shown for comparison reasons. Dark gray color corresponds to applying dropout to the first and second hidden layers, red color to the first hidden layer, and blue color to the second hidden layer. Applying dropout in both hidden layers improves the results slightly.

For tuning the learning rate and batch size of Model 1, the `randomsearch` algorithm of the Keras tuner library is used. Following the same steps as the dataset-A case, the training set was randomly split into new training and validation sets using the 80%-20% rule. Hence, the results are expected to be close to those obtained for dataset-A and were presented in *Table 6.1*. The tuning here was performed using one trial per test instead of three to speed up the process. According to the previous tests' results, the number of 20 epochs was chosen as it seemed sufficient to identify whether the performance has improved or not. The batch sizes tested were 16, 32, 64, 128, and 256, for the five learning rates, from 10^{-2} to 10^{-6}, using the log sample step of 10. The resulted accuracies are gathered in *Table 6.3*. The learning rate of 0.001 has the highest accuracies for all the batch sizes, followed by the learning rate of 0.0001. The learning rate of 0.01 appears inconsistent. Accuracy drops as the learning rate decreases in the log scale. Following the tuning results, the batch sizes of 16, 32, 64, and 128 were used with the learning rate of 0.001 to retrain Model 1 for 50 epochs, and the resulted learning curves are presented in the comparative charts of *Figure 6.22*. The batch sizes that show better performance are those of 64 and 16. They both have fewer fluctuations than the batch sizes of 128 and 32 in the resulted validation loss and accuracy curves. Between the batch sizes of 64 and 16, the former performs better after ten training epochs, while the latter in the first ten epochs. Since the highest accuracies and lower losses are observed after the 10th epoch, the batch size of 64 is chosen as the best one. The setup of using BN in all the convolutional layers, dropout of rate 0.5 in the first two hidden layers, while training using SGD with the momentum of 0.9, the learning rate of 0.001, and batch size of 64 is named as Model 1 and will be compared with the corresponding one obtained with Adam. Augmentation was also applied in the training setup of Model 1 but did not improve the results (*Figure C.1* in Appendix C); instead, the resulted curves appeared much noisier, same as dataset A. Thus, augmentation techniques were excluded at this stage of this study.

Table 6.3: SGD - Classification accuracy calculated on a randomly split validation set using different batch sizes and learning rates. The highest accuracies per batch are highlighted with bold letters.



		Learning rates				
		0.01	0.001	0.0001	0.00001	0.000001
Batch sizes	16	0.966	**0.978667**	0.974333	0.955667	0.6673333
	32	0.333333	**0.967667**	0.966	0.918	0.6293333
	64	0.937667	**0.969667**	0.948333	0.865333	0.549
	128	0.957667	**0.964333**	0.939667	0.743333	0.5073333
	256	0.79	**0.895**	0.819333	0.655333	0.5023333

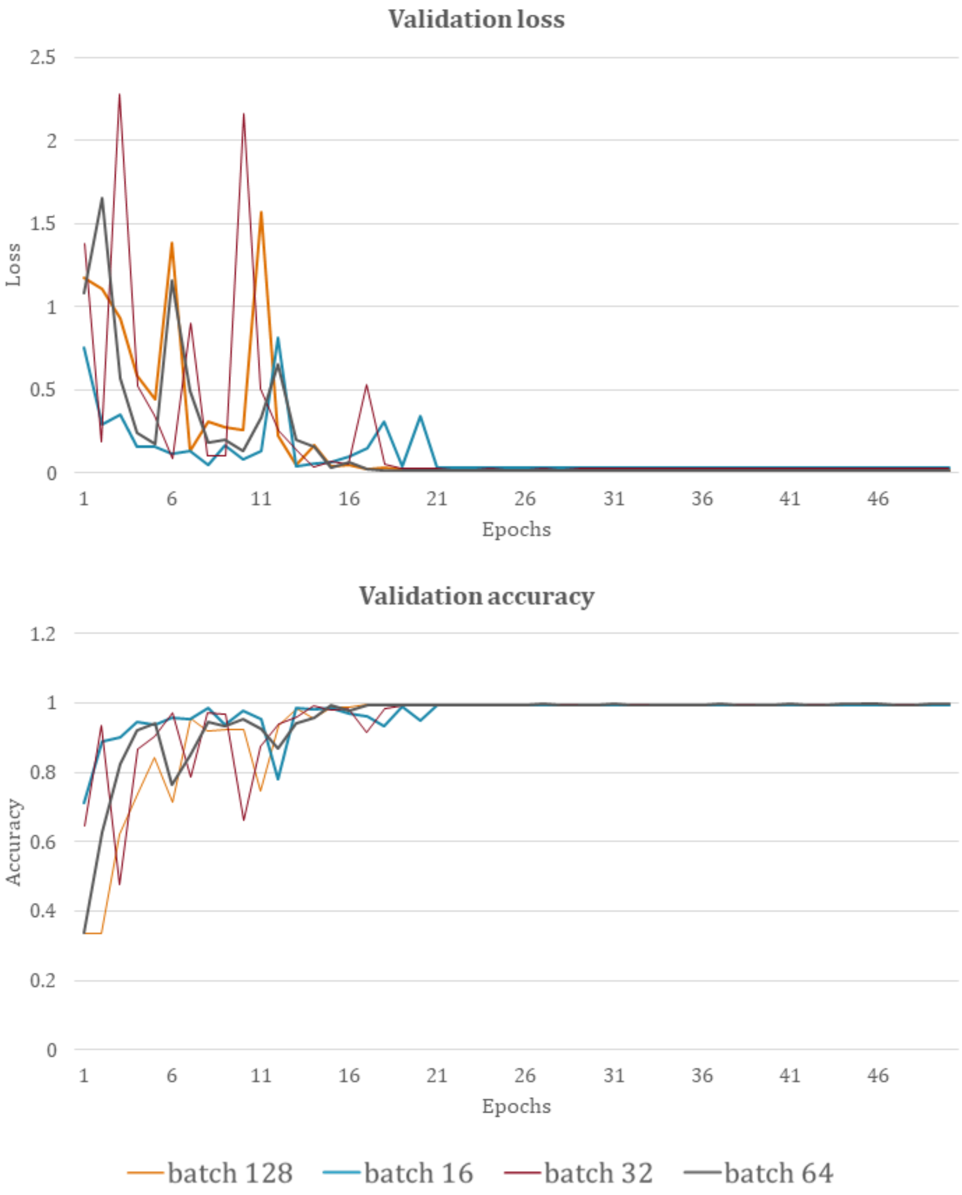


Figure 6.22:Comparative plots that show the resulting learning curves obtained from training using SGD with a learning rate of 0.001, a momentum of 0.9, and four different batch sizes that yielded high accuracy. The batch size 64 (gray line) appears to perform slightly better after ten training epochs.

6.2.2 Training with Adam

The resulting learning curves of training the baseline with Adam optimizer are presented in *Figure 6.23*. Compared with the corresponding ones obtained for dataset-A (*Figure 6.8*), they appear less noisy, achieving higher validation accuracy and less overfitting. However, learning was faster for dataset-A as the training accuracy curve converges faster to 1. Learning was also possible without the need to use BN.

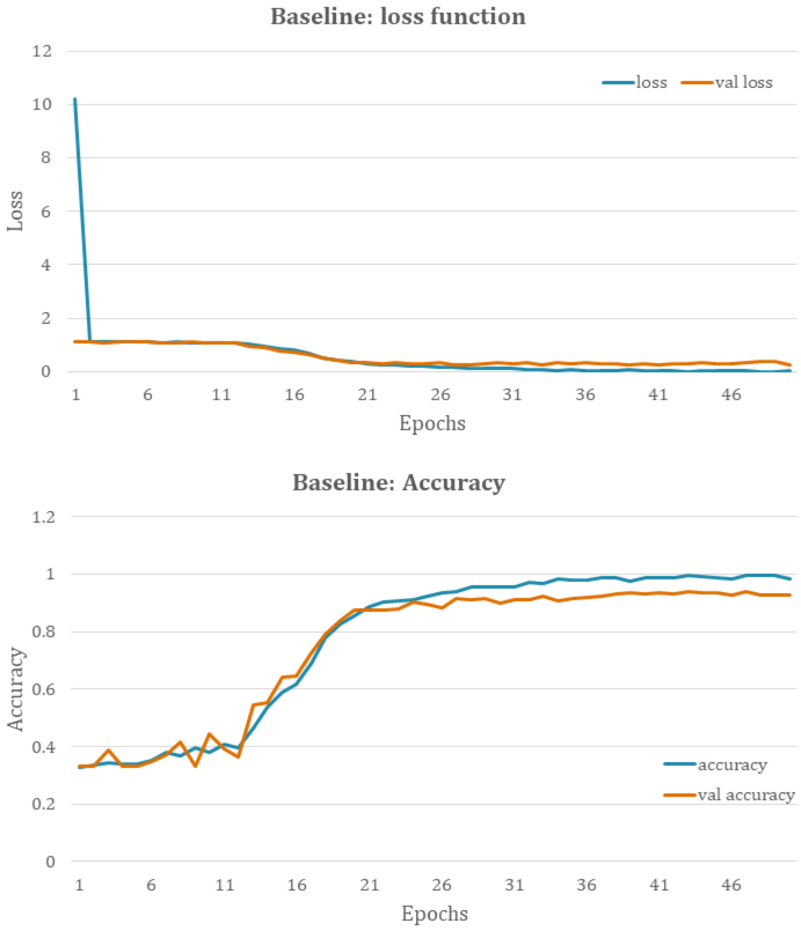


Figure 6.23: The learning curves from training the baseline using the dataset-B and Adam optimizer with a batch size of 128 and a learning rate of 0.001.

Next, the various BN and dropout setup described in Figure 6.19 were tested using Adam as an optimizer seeking performance improvements of the baseline results. The resulting learning curves of the setups described by test1 to test17 are presented in *Figure 6.24*. Learning is feasible for the setups described by test5, test6, test8, test9, and test17. However, the curves appear noisy with high fluctuations compared to the ones obtained from the baseline model. The setup described by test8, where BN is applied to the first three and the last convolutional layers, performs better after 40 epochs. After this point, the performance is improved when compared to the baseline model. The validation loss converges to zero and the validation accuracy to one. Dropout was applied next for the setups described by test18, 19, and 20 of *Figure 6.24,* but no further improvements were observed from the resulting learning curves; instead, the performance is worse. The corresponding graphs can be found in *Figure C.2* of Appendix C. For this reason, dropout was excluded, and the setup of BN described by test8 was chosen for the next step, where training hyperparameters are tuned to examine whether further performance improvements are possible.

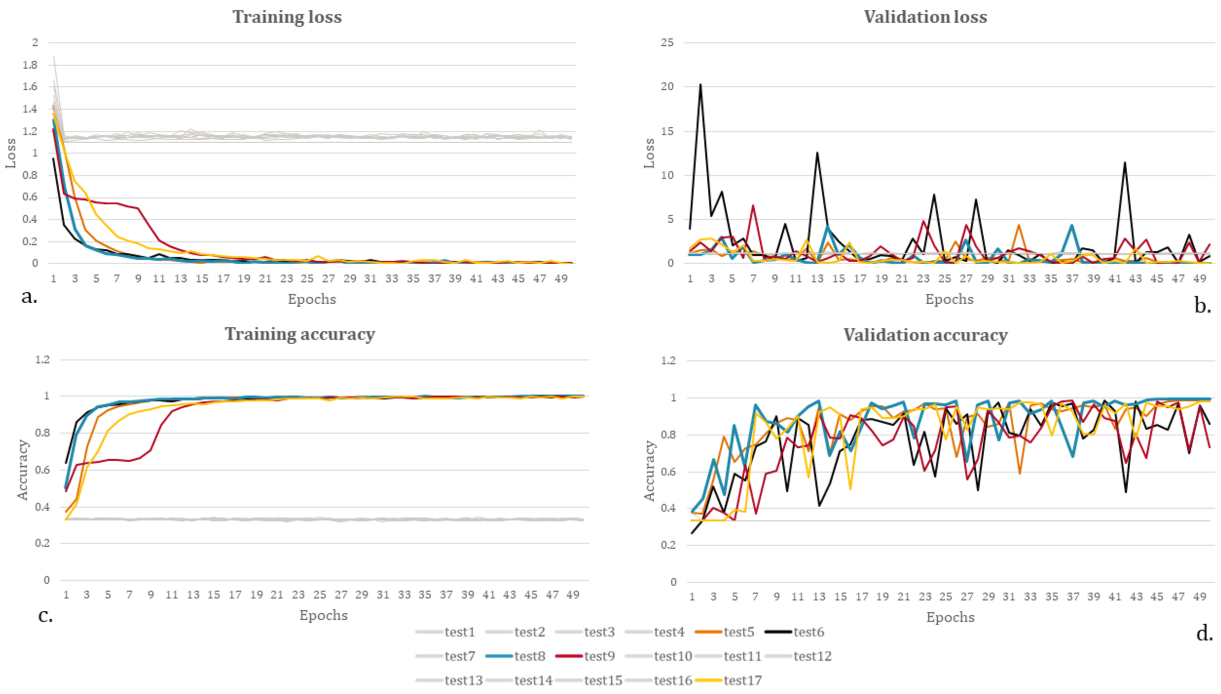


Figure 6.24: Comparative graphs showing the resulting learning curves from tests performed on different BN setups for dataset-B and Adam optimizer. a. is the training loss, b. is the validation loss, c. is the training accuracy, and d. is the validation accuracy. With gray colors are the models that resulted in no learning.

The tuning process was like the dataset-A case, with the exception that the learning rate of 0.1 was excluded, and the learning rate of 10^{-6} was included. The former was excluded as no training was feasible when using it. Further, the trials executed per experiment with `randomsearch` algorithm were also reduced to one from three to speed up the process. The results are summarized in *Table 6.4*. Remarkably high accuracies were achieved for the learning rate of 0.001. Hence, the learning curves were used to pick the batch size that yields better performance, and the results are presented in *Figure 6.25*. The batch size of 32 resulted in improvements in both validation accuracy and loss after 38 training epochs. For this reason, it was selected as the final training setup, named Model 2, despite producing noisier learning curves than the baseline for approximately the first 40 epochs. An observed inconsistency with the tuning results concerns batch 16, where non-learning is observed (gray lines in *Figure 6.25*). This might indicate a non-robust training setup, and further tests are required. Same as with the SGD case, augmentation techniques were not beneficial on the performance, as shown in *Figure C.3* Appendix C.

Table 6.4: Adam - Classification accuracy calculated on a randomly split validation set using different batch sizes and learning rates. The highest accuracies per batch are highlighted with bold letters.

		Learning rates				
		0.01	0.001	0.0001	0.00001	0.000001
Batch sizes	16	0.333333	**0.988**	0.987667	0.979	0.882
	32	0.333333	**0.988667**	0.981333	0.961333	0.7883334
	64	0.979333	**0.986667**	0.971	0.944333	0.6493334
	128	0.980667	**0.985**	0.969333	0.907	0.569
	256	0.976333	**0.974**	0.961333	0.822667	0.5236667

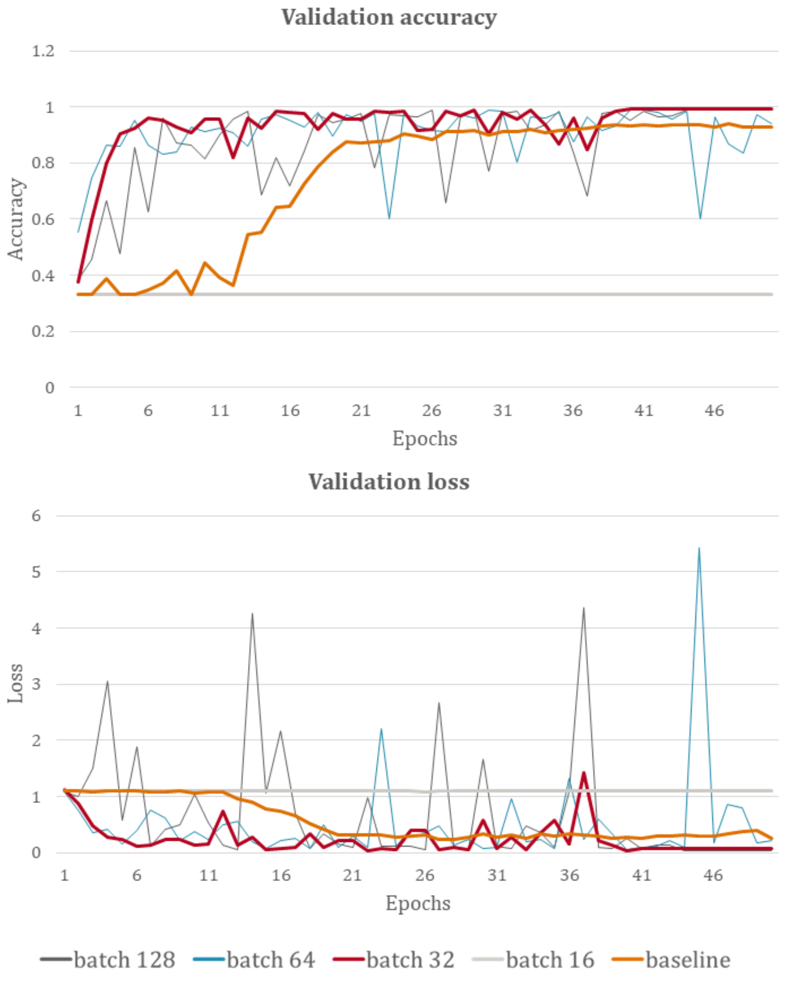


Figure 6.25: Comparative charts showing the resulting learning curves for each batch size tested using Adam optimizer with a learning rate of 0.001 on dataset-B. The baseline learning curves are presented for comparison reasons. Batch 32 is selected as the training setup for Model 2.

6.2.3 SGD and Adam comparison

Training the AlexNet architecture with dataset-B using SGD and Adam optimizers was like the case of dataset-A. The two training approaches required different BN and dropout setups to improve performance. For the SGD optimizer case applying BN was also essential in achieving learning. The setup that returned the best results was applying it in all the convolutional layers, and performance was further improved by applying dropout with a rate of 0.5 in the first two hidden layers. As for training, the learning rate of 0.001 with a batch size of 64 resulted in the best performance given the settings tested. This model is named Model 1B, and the learning curves for 50 training epochs are presented in *Figure 6.26* with orange color. On the other, training with Adam did not require BN to achieve learning, and only with the baseline setup, a validation accuracy of 92% was achieved. BN improved performance when applied to the convolutional layers 1, 2, 3, and 5, while dropout was excluded as it had a negative impact on the results. This setup that was trained with Adam returned the best results when used a learning rate of 0.001 and a batch size of 32, and the obtained model is named Model 2B. The corresponding learning curves are presented in the comparative charts of *Figure 6.26*. Between the two, Model 1B performs better. The validation loss curve is less noisy and converges to zero faster. The same goes for the validation accuracy curve, which converges to 1 faster. Both approaches eventually reach exceedingly high classification accuracy within the period of 50 epochs, as is shown in confusion matrices of *Figure 6.27*. SGD Model 1B has 99.65% classification accuracy on the test set, and Adam's Model 2Bhas 99.44%. The former has fewer misclassified labels, especially for the noisy class, where predictions are 100% correct. The settings of Model 1B were chosen to train the final model using dataset-B named Model B that is described in the following paragraph.

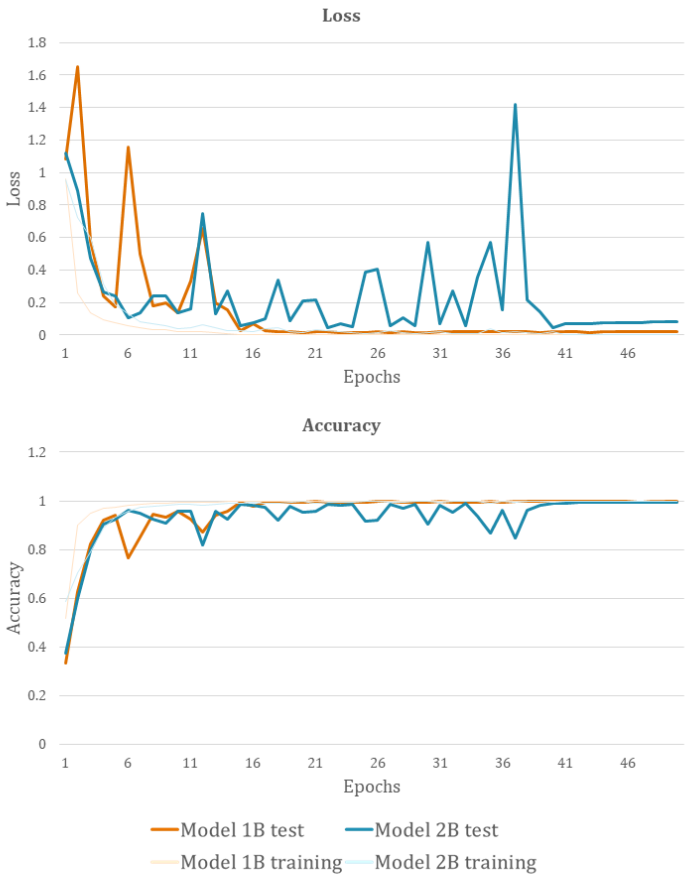


Figure 6.26: Comparative graphs showing the loss and accuracy learning curves of the best-obtained models that were trained with Model 1B (SGD) and Model 2B (Adam) using dataset-B. Faded orange and blue lines correspond to the accuracy and loss curves on the training set.

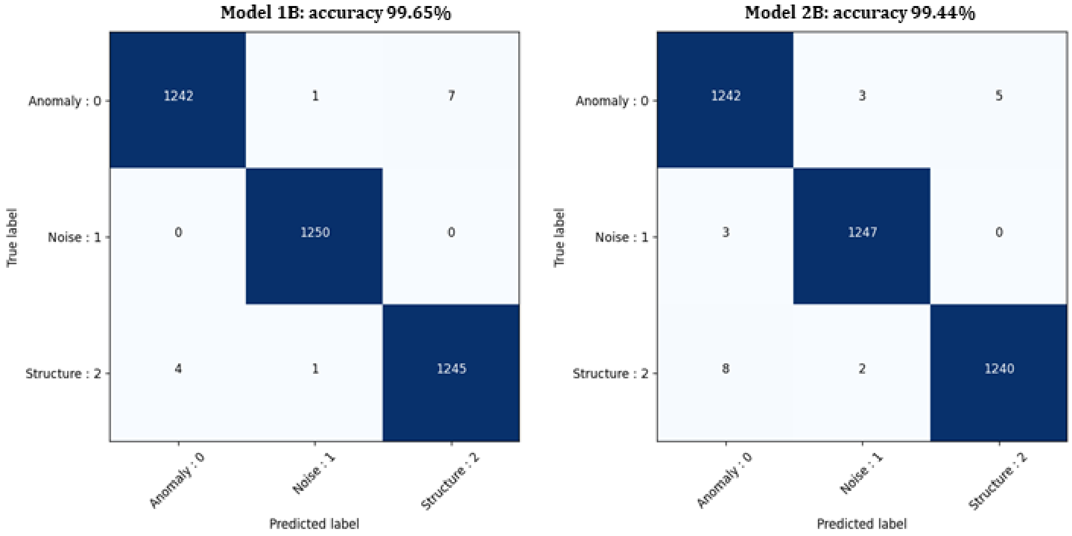


Figure 6.27: Confusion matrices resulted from training with Model 1B (left) and Model 2B (right) using SGD and Adam, respectively, for dataset-B.

6.2.4 Model B

Following the previous test results, the SGD approach was used to train the final model using dataset B, named Model B. The final model summary with details is presented in *Table C.2* of Appendix C. The training used a learning rate of 0.001, a batch size of 64, and a momentum of 0.9. Unlike the case of Model A, no further training tests were performed, like applying early stopping or reducing the learning rate with a Keras scheduler, as the performance was found very good. The training epochs were set to 30, as the highest performance was reached around 15 epochs in the previous tests, and the weights were saved when the validation accuracy was improved using the Keras library callbacks API. The obtained learning curves are presented in the charts of *Figure 6.28*. The best performance over the period of 30 epochs occurred at 19, with validation accuracy reaching 99.52% and loss function being at 0.0191.

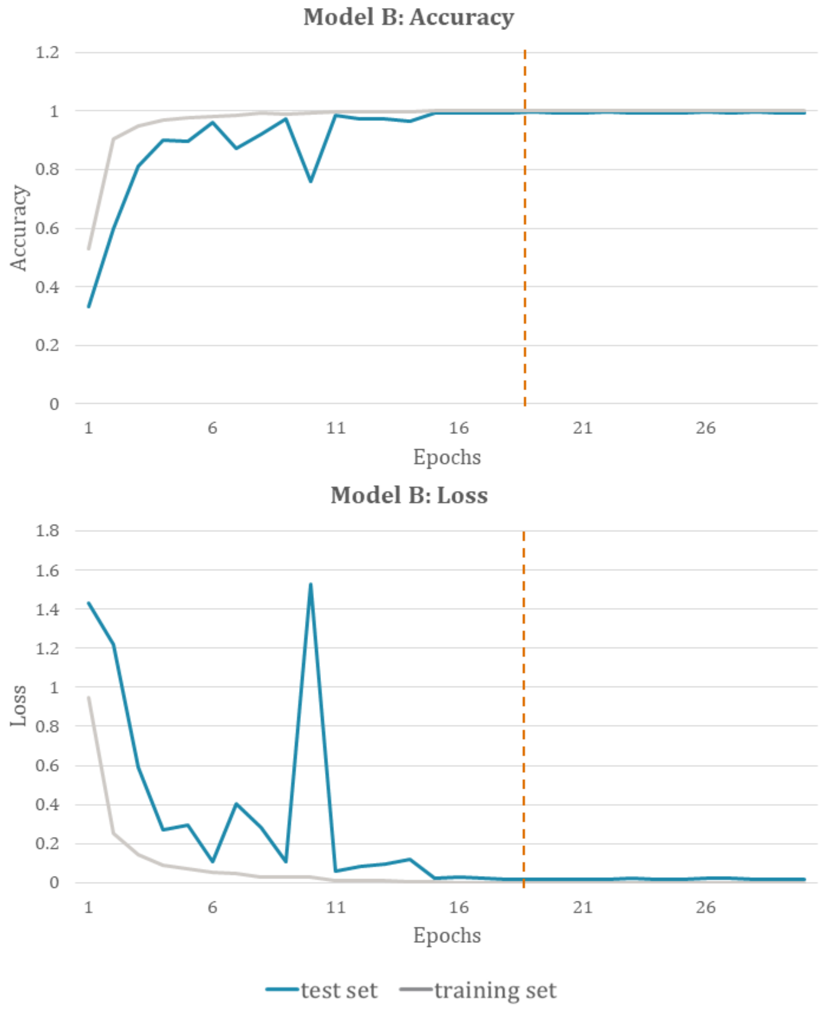


Figure 6.28: Learning curves of Model B. The highest validation accuracy and lowest validation loss were obtained after 19 training epochs (noted with orange dashed line).

The confusion matrix of *Figure 6.28* was computed using the weights of the 19th epoch on the test set of dataset-B. There are six misclassified labels under anomaly class, three under noise class, and nine under structure class. A few representative examples of the misclassified labels are presented in *Figure 6.30*. These provide some insights on further improvements for the training dataset, for example, including more linear features like pipes, under the anomaly class to better distinguish these patterns from the ancient buried structures. Horizontal noise appears to be challenging as well, as it can easily be mistaken for structural remains. Further, there seems to be the tendency to classify strong amplitudes as anomalies, which requires further research.

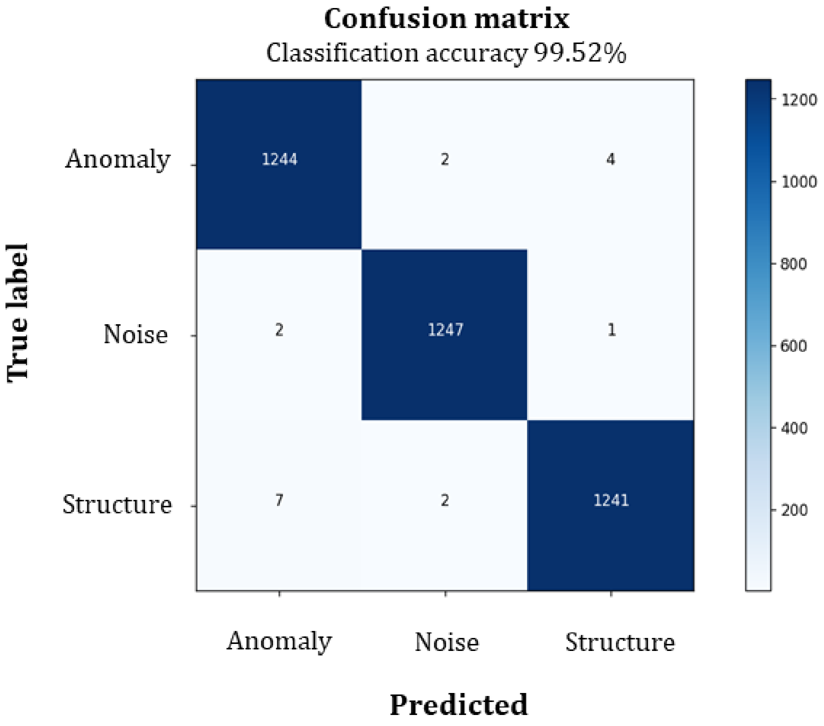


Figure 6.29: Confusion matrix calculated using the weights of Model B that returns 99.52% classification accuracy.

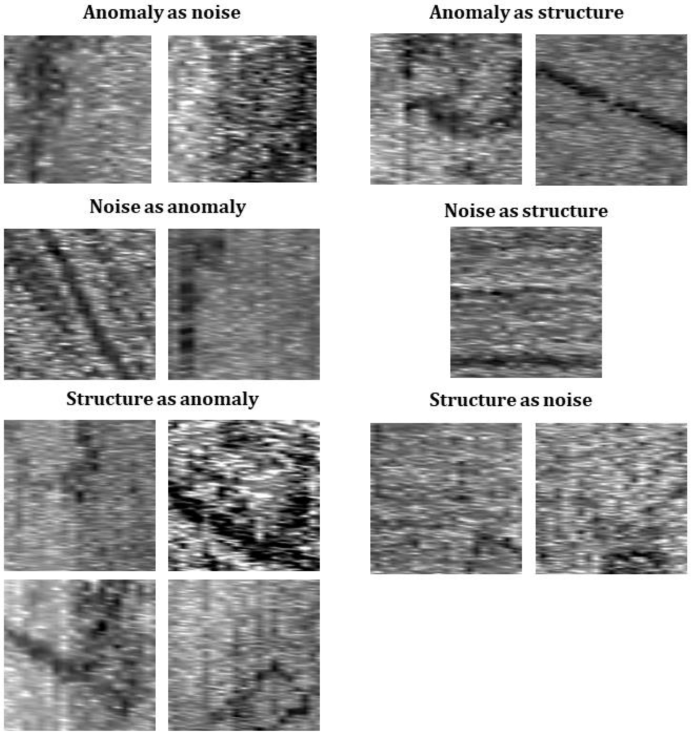


Figure 6.30: Examples of misclassified labels of each class using Model B.

6.3 Comparison & Final Evaluation

In this concluded section, the resulted models that yielded better performance when trained with dataset-A and dataset-B are compared to find what is the best approach for training CNN models with GPR depth slices from archaeological prospection surveys. For both approaches, the SGD optimizer was found to perform better than Adam, and both required BN application to all the convolutional layers for the models to learn. The learning curves resulted from Model A and B are presented in the comparative charts of *Figure 6.31*.

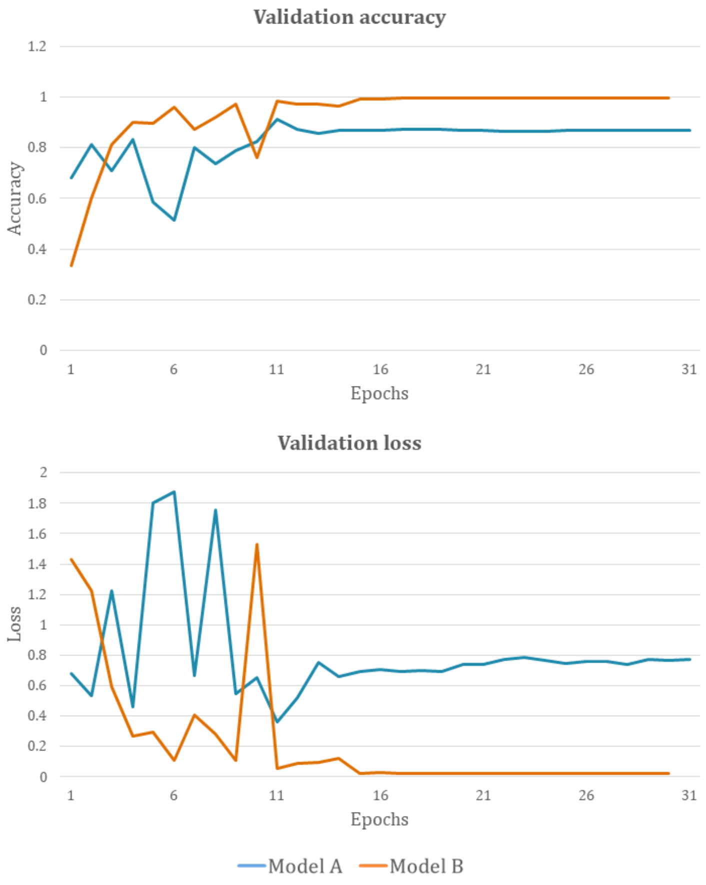


Figure 6.31: Comparative charts of the resulting learning curves from Model A and Model B.

Overall, Model B has better generalization performance as the test set, which was split randomly from the training set, better represents the latter. The result is smoother curves, with less overfitting and higher validation accuracy. This made the training process easier. However, the main question is which approach generalizes the best for entirely new data. Thus, the weights that yielded the best results from each model, namely the 11th epoch for Model A and the 20th epoch for Model B, are used on an evaluation set with 32 geophysical anomaly examples (*Figure 6.32*), 32 noise examples (*Figure 6.33*), and 36 structure examples (*Figure 6.34*).

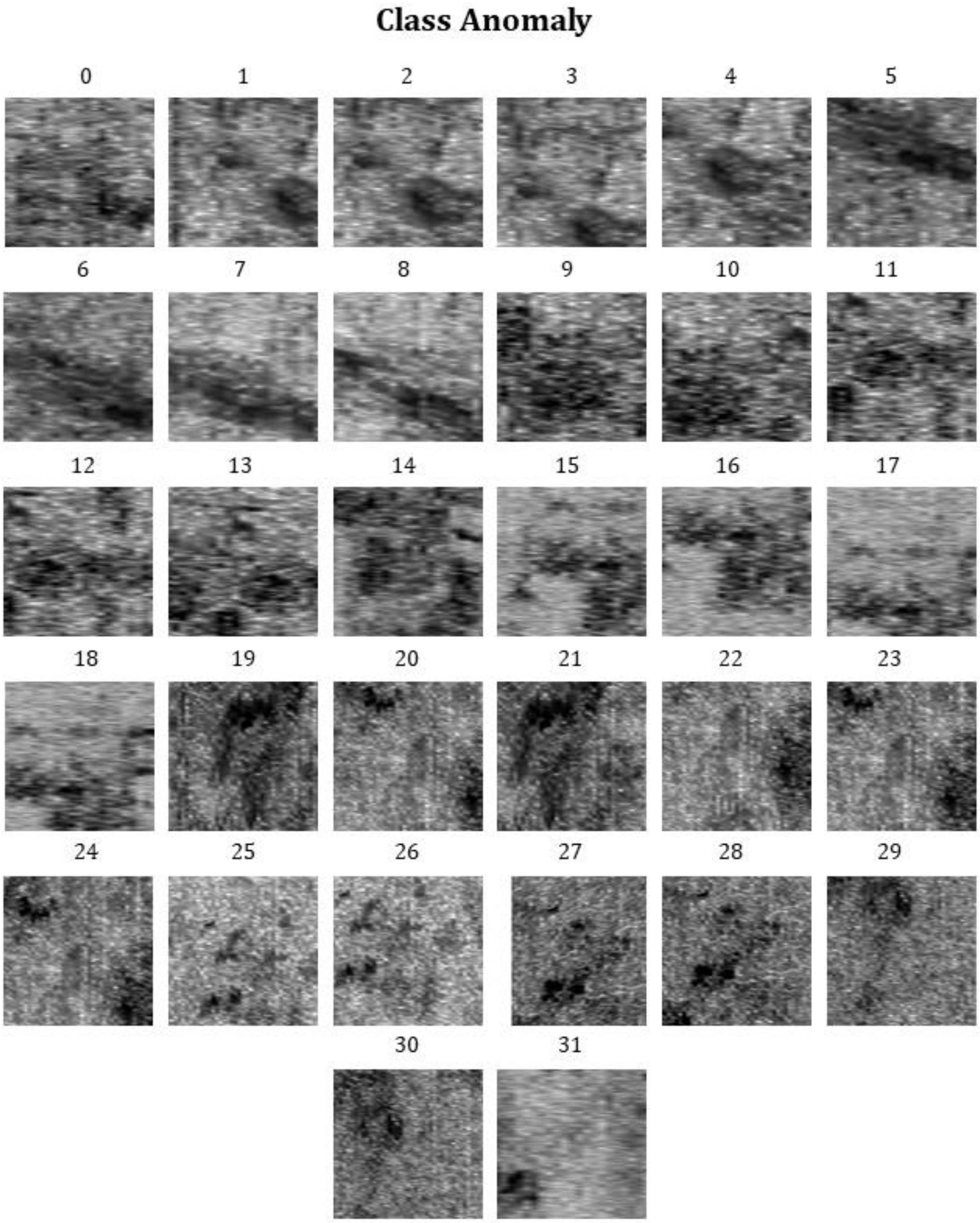


Figure 6.32: The 32 examples selected for the Anomaly class of the Evaluation set. Examples 0 to 18 derived from Halos' archaeological sites, while examples 19 to 31 from the archaeological site of Sissi.

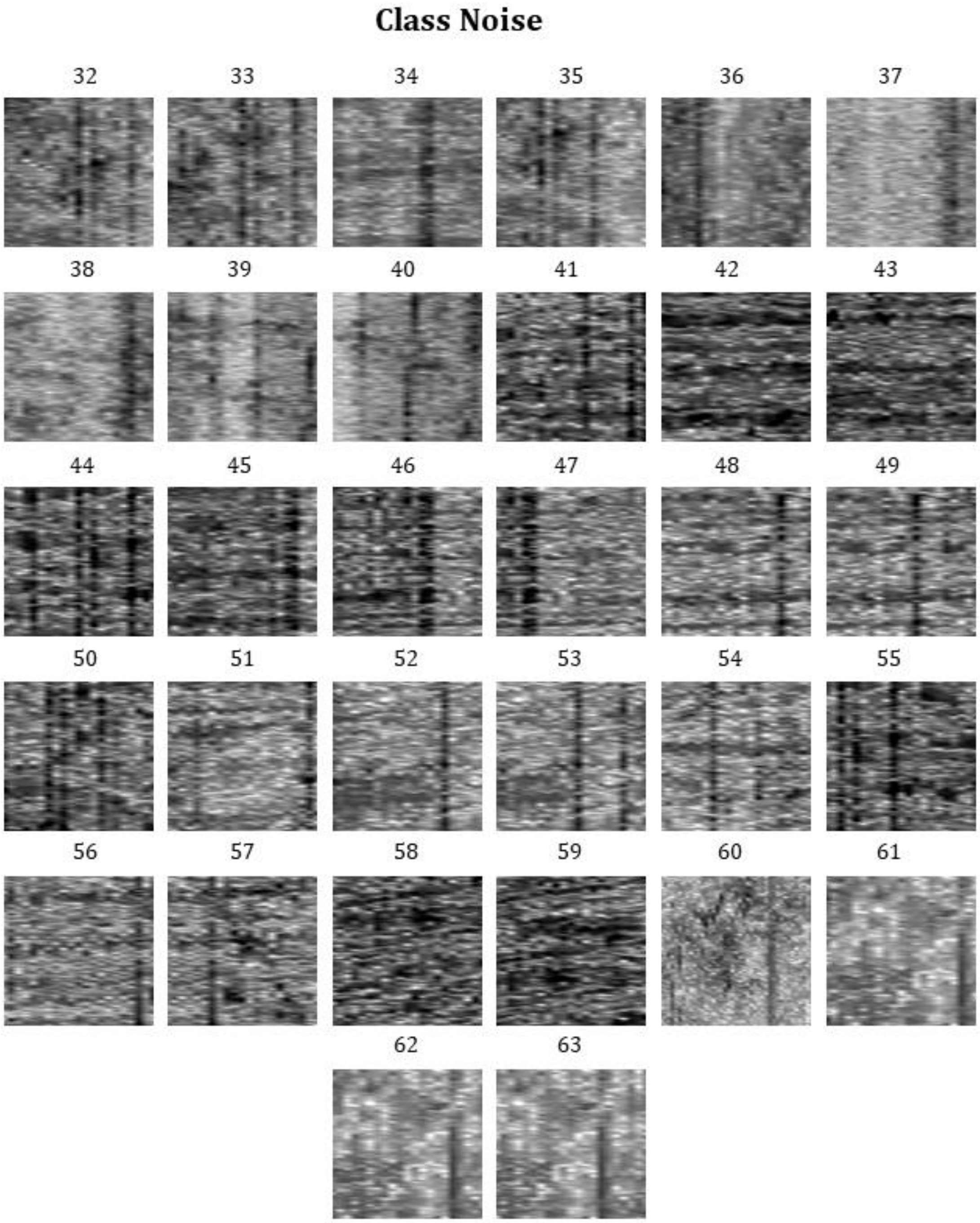


Figure 6.33: The 32 examples selected for the Noise class of the Evaluation set. Examples 32 to 63 derived from the Halos' archaeological site, while the examples 60 to 63 from the archaeological site of Sissi.

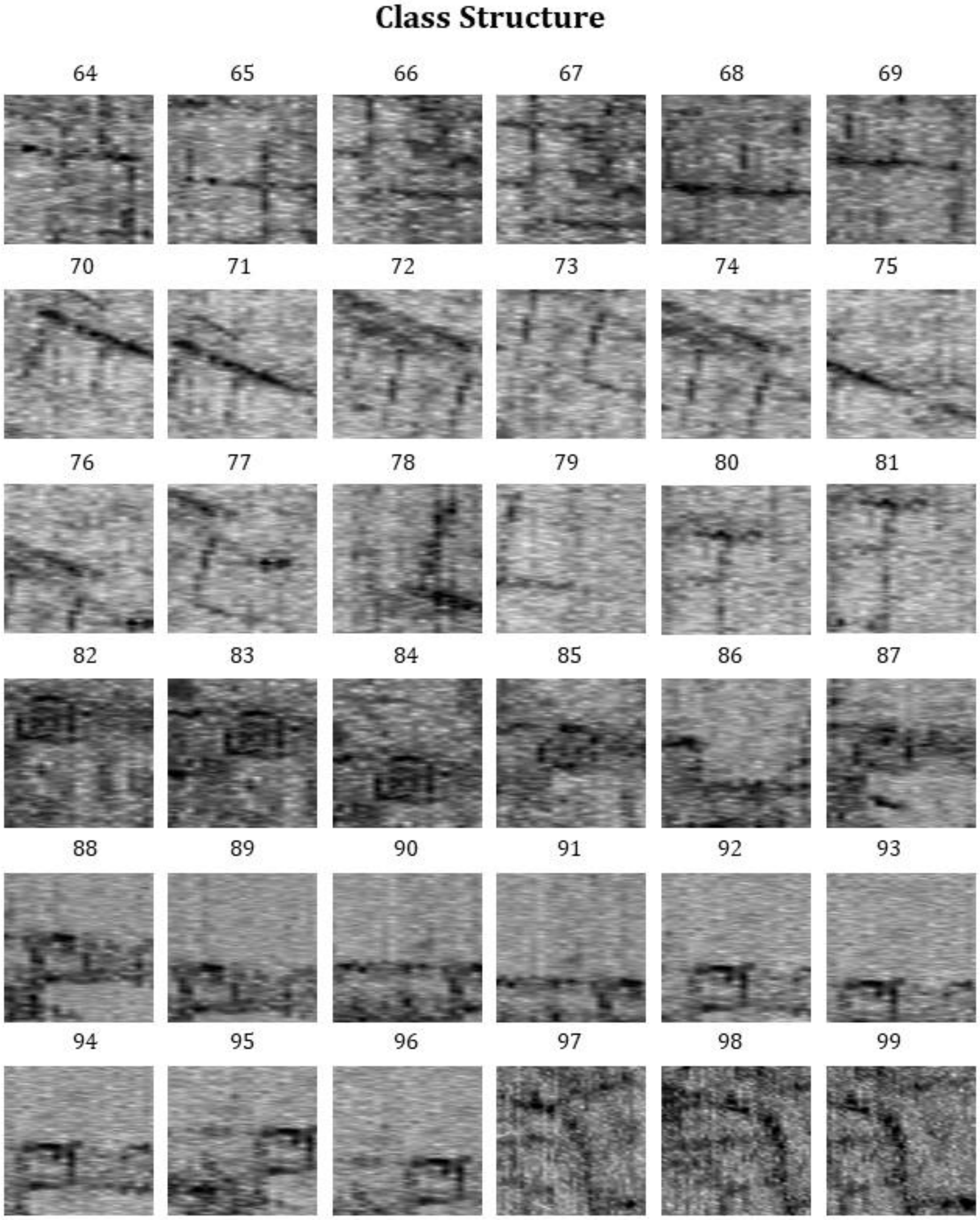


Figure 6.34: The 36 examples selected for the Structures class of the Evaluation set. Examples 64 to 96 derived from Halos' archaeological site, while examples 97 to 99 from the archaeological site of Sissi.

The models were evaluated using the `predict` and `evaluate` functions of Keras library, as described in Chapter 5, on the evaluation set. Further, the confusion matrices were computed to visualize the classification performance. The results are presented in *Figure 6.35.* Overall, Model B generalizes better than Model A, reaching a classification accuracy of 92% over 85%. The confusion matrices show that Model A performed better in predicting all the Anomaly class examples correctly, with zero mistakes over two made with Model B. However, Model A performed poorer than model A for the other two classes. Model A made seven wrong predictions for the Noise class, where model B made three, while for the Structure class, model A made eight mistakes where model B made three.

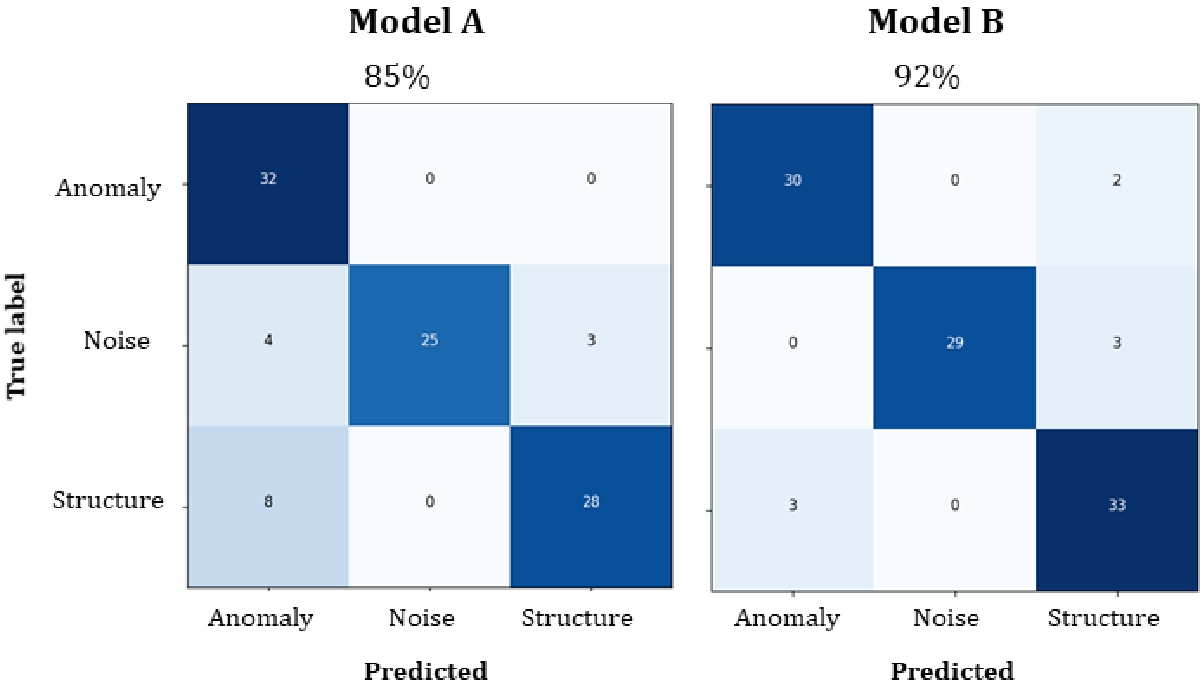


Figure 6.35: Confusion matrices calculated on the evaluation set for Model A and Model B, reaching the classification accuracies of 85% and 92%, respectively.

The two models' analytical predictions on the examples presented in *Figure 6.32*, *Figure 6.33*, and *Figure 6.34* are presented in *Table 6.5* for the Anomaly class, in *Table 6.6* for the Noise class, and in *Table 6.7* for the Structure class, respectively. All the misclassified labels are highlighted with orange color, while with yellow color are the marginally correct predictions, with accuracy less than 60%.

Table 6.5: Prediction results from Model A and Model B for the Anomaly class of the evaluation set occurred. With orange are highlighted the misclassified labels, while with yellow are the marginally correct predictions (<60%).

ID			**Model A: 85%**			**Model B: 92%**
	Anomaly(%)	Noise(%)	Structure (%)	Anomaly(%)	Noise(%)	Structure (%)
0	100.0	0.0	0.0	100.0	0.0	0.0
1	100.0	0.0	0.0	100.0	0.0	0.0
2	100.0	0.0	0.0	100.0	0.0	0.0
3	100.0	0.0	0.0	100.0	0.0	0.0
4	100.0	0.0	0.0	100.0	0.0	0.0
5	100.0	0.0	0.0	100.0	0.0	0.0
6	100.0	0.0	0.0	100.0	0.0	0.0
7	100.0	0.0	0.0	99.2	0.0	0.8
8	100.0	0.0	0.0	100.0	0.0	0.0
9	99.9	0.0	0.1	100.0	0.0	0.0
10	100.0	0.0	0.0	98.6	0.0	1.4
11	100.0	0.0	0.0	100.0	0.0	0.0
12	100.0	0.0	0.0	100.0	0.0	0.0
13	100.0	0.0	0.0	100.0	0.0	0.0
14	100.0	0.0	0.0	100.0	0.0	0.0
15	100.0	0.0	0.0	100.0	0.0	0.0
16	100.0	0.0	0.0	100.0	0.0	0.0
17	100.0	0.0	0.0	100.0	0.0	0.0
18	100.0	0.0	0.0	100.0	0.0	0.0
19	99.8	0.0	0.2	100.0	0.0	0.0
20	100.0	0.0	0.0	7.2	0.0	92.8
21	100.0	0.0	0.0	100.0	0.0	0.0
22	100.0	0.0	0.0	100.0	0.0	0.0
23	52.7	0.0	47.3	15.2	0.0	84.8
24	100.0	0.0	0.0	100.0	0.0	0.0
25	100.0	0.0	0.0	100.0	0.0	0.0
26	100.0	0.0	0.0	100.0	0.0	0.0
27	100.0	0.0	0.0	100.0	0.0	0.0
28	100.0	0.0	0.0	100.0	0.0	0.0
29	100.0	0.0	0.0	100.0	0.0	0.0
30	100.0	0.0	0.0	100.0	0.0	0.0
31	100.0	0.0	0.0	100.0	0.0	0.0

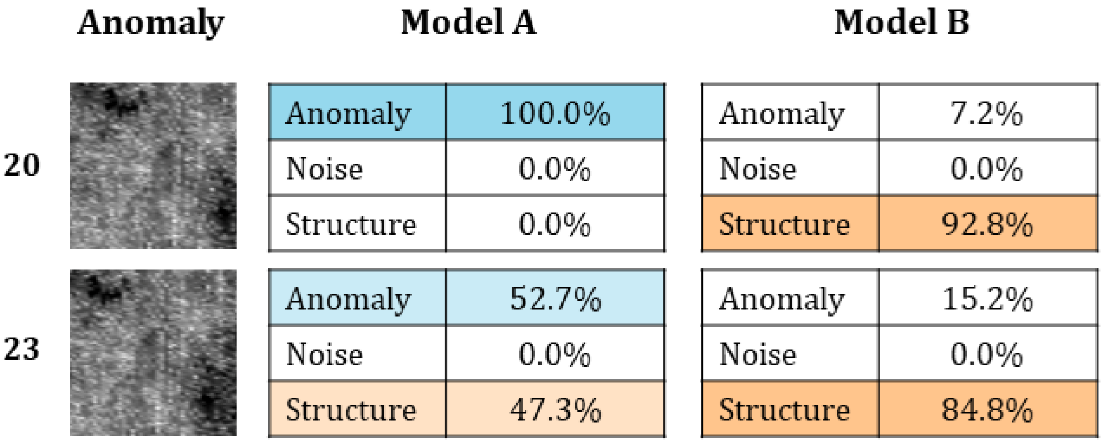
Table 6.6: Prediction results from Model A and Model B for the Noise class of the evaluation set occurred. With orange are highlighted the misclassified labels, while yellow shows the marginally correct predictions (<60%).

ID	Model A: 85%			Model B: 92%		
	Anomaly(%)	Noise(%)	Structure (%)	Anomaly(%)	Noise(%)	Structure (%)
32	2.3	96.8	1.0	0.0	100.0	0.0
33	0.0	100.0	0.0	0.0	100.0	0.0
34	0.0	100.0	0.0	0.0	100.0	0.0
35	0.0	99.8	0.1	0.0	100.0	0.0
36	0.0	99.9	0.1	0.0	100.0	0.0
37	0.0	99.9	0.1	0.0	99.9	0.1
38	0.0	86.1	13.9	0.0	4.7	95.3
39	0.0	1.1	98.9	0.0	0.1	99.9
40	0.0	43.3	56.7	0.0	5.0	95.0
41	0.0	100.0	0.0	0.0	100.0	0.0
42	60.6	38.3	1.1	0.0	100.0	0.0
43	0.0	0.0	100.0	0.0	100.0	0.0
44	0.0	100.0	0.0	0.0	100.0	0.0
45	0.0	100.0	0.0	0.0	100.0	0.0
46	0.0	100.0	0.0	0.0	100.0	0.0
47	0.0	100.0	0.0	0.0	100.0	0.0
48	0.0	100.0	0.0	0.0	100.0	0.0
49	0.0	99.9	0.1	0.0	100.0	0.0
50	0.0	100.0	0.0	0.0	100.0	0.0
51	0.1	98.7	1.2	0.0	100.0	0.0
52	0.0	100.0	0.0	0.0	100.0	0.0
53	0.0	100.0	0.0	0.0	100.0	0.0
54	0.0	100.0	0.0	0.0	100.0	0.0
55	0.0	100.0	0.0	0.0	100.0	0.0
56	0.0	92.0	8.0	0.0	100.0	0.0
57	53.3	28.6	18.1	0.0	100.0	0.0
58	99.1	0.9	0.0	0.0	100.0	0.0
59	99.3	0.7	0.0	2.5	97.5	0.0
60	0.2	98.7	1.1	0.0	83.3	16.7
62	0.0	100.0	0.0	0.0	100.0	0.0
62	0.0	100.0	0.0	0.0	100.0	0.0
63	0.0	100.0	0.0	0.0	100.0	0.0

Table 6.7: Prediction results from Model A and Model B for the Structure class of the evaluation set occurred. With orange are highlighted the misclassified labels, while yellow shows the marginally positive predictions (<60%).

ID			**Model A: 85%**			**Model B: 92%**
	Anomaly(%)	Noise(%)	Structure (%)	Anomaly(%)	Noise(%)	Structure (%)
64	0.0	0.0	100.0	0.0	0.0	100.0
65	0.0	0.0	100.0	0.0	0.0	100.0
66	0.0	0.0	100.0	0.0	0.0	100.0
67	0.0	0.0	100.0	0.0	0.0	100.0
68	0.1	0.0	99.9	0.0	0.0	100.0
69	0.0	0.0	100.0	0.0	0.0	100.0
70	0.0	0.0	100.0	0.0	0.0	100.0
71	0.0	0.0	100.0	0.0	0.0	100.0
72	0.0	0.0	100.0	0.0	0.0	100.0
73	0.0	0.0	100.0	0.0	0.0	100.0
74	0.0	0.0	100.0	0.0	0.0	100.0
75	0.0	0.0	100.0	0.0	0.0	100.0
76	0.0	0.0	100.0	0.0	0.0	100.0
77	0.0	0.0	100.0	0.0	0.0	100.0
78	0.7	0.0	99.3	22.4	0.0	77.6
79	0.0	0.0	100.0	0.0	0.0	100.0
80	0.0	0.0	100.0	0.0	0.0	100.0
81	0.0	0.0	100.0	0.0	0.0	100.0
82	100.0	0.0	0.0	99.3	0.0	0.6
83	100.0	0.0	0.0	0.4	0.0	99.6
84	100.0	0.0	0.0	98.7	0.0	1.3
85	100.0	0.0	0.0	44.0	0.0	56.0
86	90.4	0.0	9.6	33.3	0.0	66.7
87	95.9	0.0	4.1	92.1	0.0	7.9
88	1.4	0.0	98.6	0.0	0.0	100.0
89	75.7	0.0	24.3	31.3	0.0	68.7
90	0.0	0.0	100.0	0.0	0.0	100.0
91	1.2	0.0	98.8	0.0	0.0	100.0
92	10.4	0.0	89.6	0.0	0.0	100.0
93	1.2	0.0	98.8	0.0	0.0	100.0
94	0.0	0.0	100.0	0.0	0.0	100.0
95	98.5	0.0	1.5	0.2	0.0	99.8
96	1.5	0.0	98.5	0.1	0.0	99.9
97	0.0	0.0	100.0	0.0	0.0	100.0
98	0.0	0.0	100.0	0.0	0.0	100.0
99	0.0	0.0	100.0	0.0	0.0	100.0

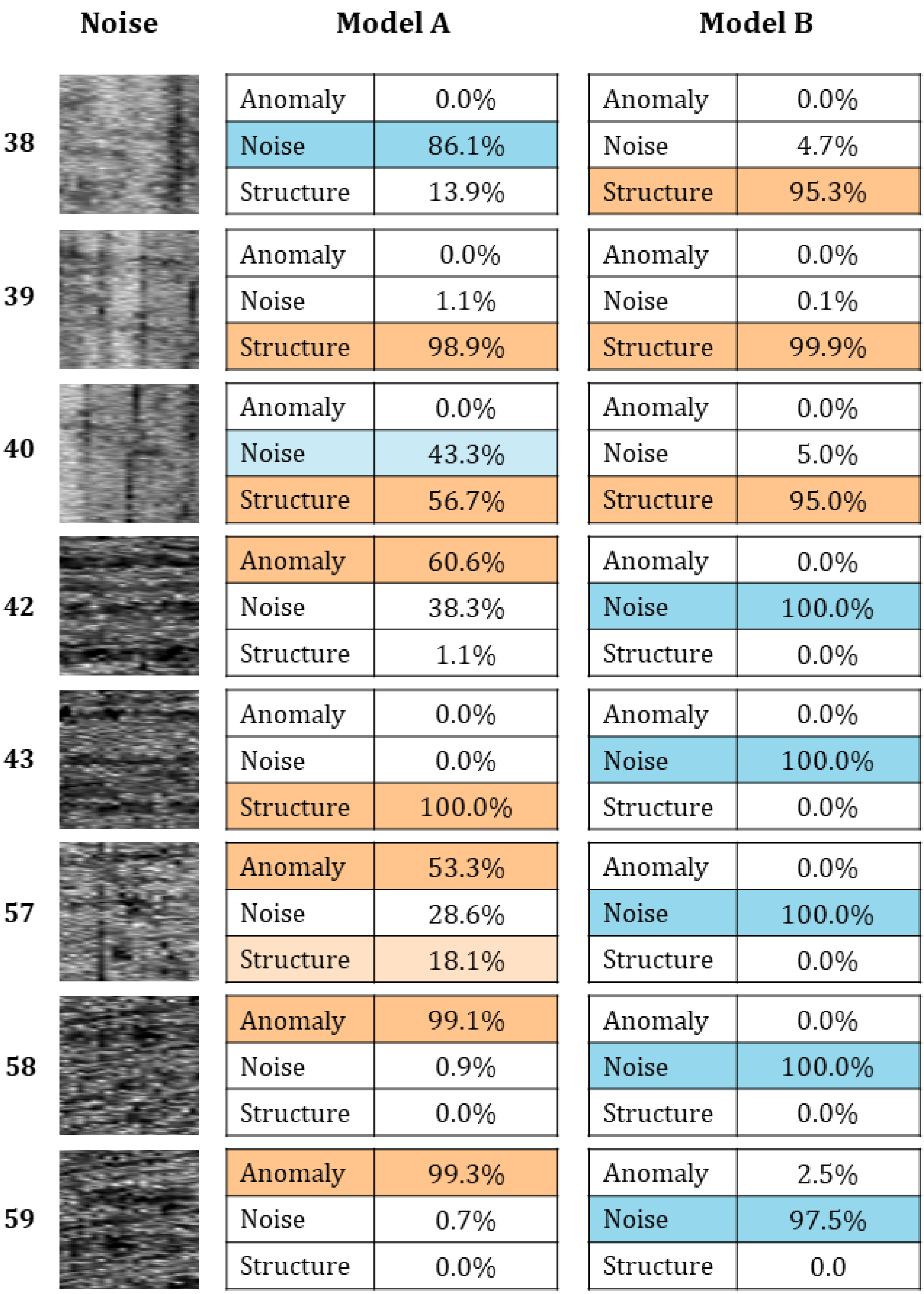
The incorrect predictions made by Model B for the Anomaly class (*Table 6.5*) were on the examples 21 and 23 that were classified as structures. The misclassified examples are shown in *Figure 6.36*. These are different instances of the same anomalies as occurred from the overlapping sliding window approach described in Chapter 5. Examples 23 is also a marginal case for Model A, which was found 52.7% as an anomaly and 47.3% as a structure. A notable observation is that example 24 (*Figure 6.32*), which is another instance of the same feature, is correctly classified by both models with 100% accuracy (*Table 6.5*). This shows a non-robust behavior to consider for future improvements. Further, more examples are required to examine which approach generalizes the best.


Anomaly
Model A
Model B
20
23

Anomaly	Model A		Model B	
20	Anomaly	100.0%	Anomaly	7.2%
	Noise	0.0%	Noise	0.0%
	Structure	0.0%	Structure	92.8%
23	Anomaly	52.7%	Anomaly	15.2%
	Noise	0.0%	Noise	0.0%
	Structure	47.3%	Structure	84.8%

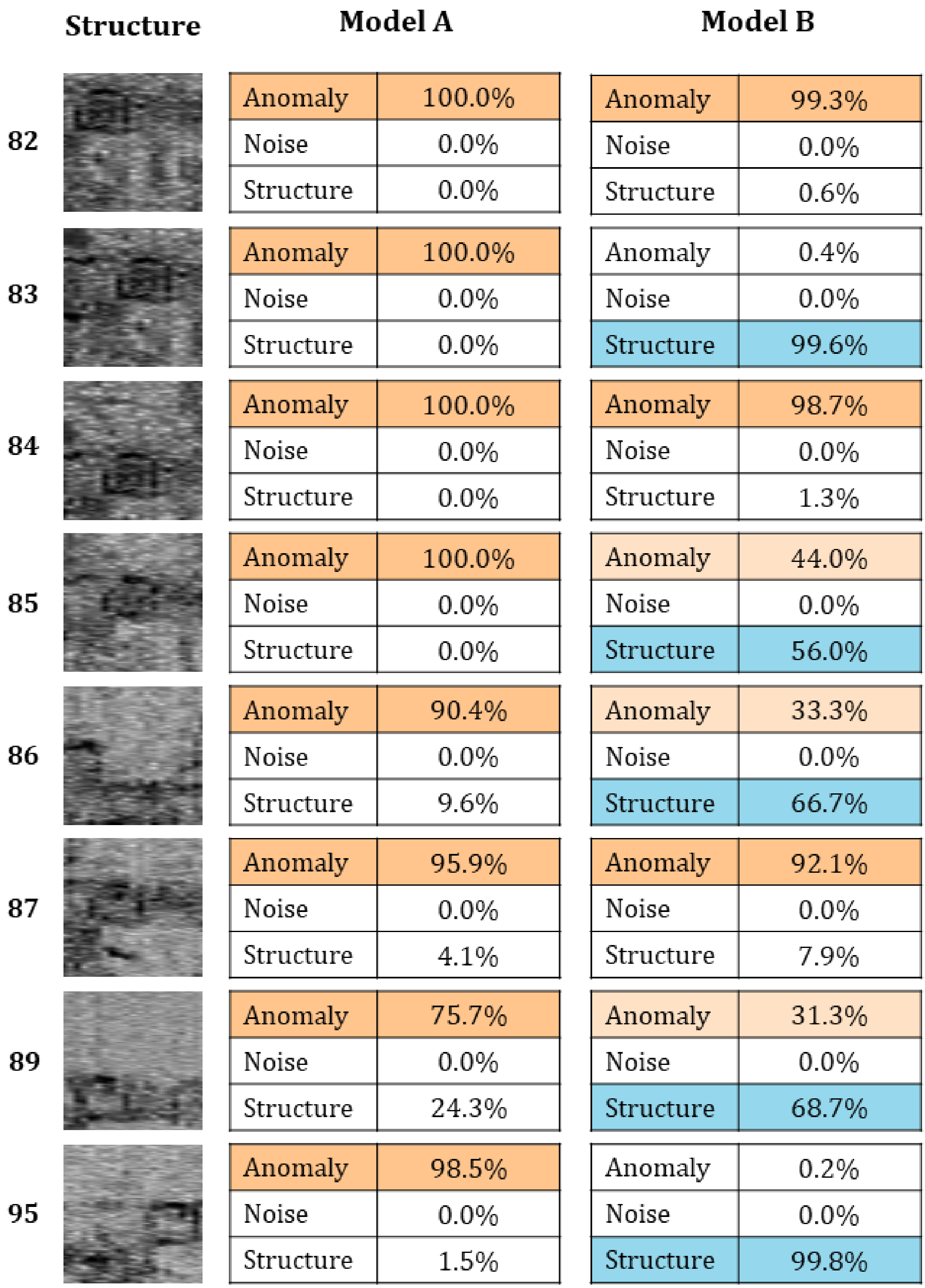
Figure 6.36: The misclassified examples for the Anomaly class. Orange color indicates a misclassified label, and blue a correctly classified label.

The Noise class's misclassified examples (*Table 6.6*) are 39, 40, 42, 43, 57, 58, and 59 for model A, while for model B are the examples 38, 39, and 40. All the misclassified examples are gathered and presented in *Figure 6.37*. The noise example 38 was classified as a structure by Model B, and more likely, was mistaken as a wall. In a similar way, examples 39 and 40 were mistaken as structures by both models. A notable observation regards example 39, which is a marginally negative prediction for Model A, having found 43.3% as noise and 56.7% as structure. However, Model B performed better in identified correctly noise derived from the surface like plowing lines, as presented in examples 42, 43, 58, and 59. Example 57 has three different noise types and appears in three different orientations: vertical, horizontal, and diagonal. Model B prediction was accurate, while Model A gave mixed results, having classified it 53.3% as an anomaly, 28.6% as noise, and 18.1% as a structure. The obtained results indicate that the random approach followed while training Model B leads to a better generalization for noise in GPR data.



Noise	Model A		Model B	
38	Anomaly	0.0%	Anomaly	0.0%
	Noise	86.1%	Noise	4.7%
	Structure	13.9%	Structure	95.3%
39	Anomaly	0.0%	Anomaly	0.0%
	Noise	1.1%	Noise	0.1%
	Structure	98.9%	Structure	99.9%
40	Anomaly	0.0%	Anomaly	0.0%
	Noise	43.3%	Noise	5.0%
	Structure	56.7%	Structure	95.0%
42	Anomaly	60.6%	Anomaly	0.0%
	Noise	38.3%	Noise	100.0%
	Structure	1.1%	Structure	0.0%
43	Anomaly	0.0%	Anomaly	0.0%
	Noise	0.0%	Noise	100.0%
	Structure	100.0%	Structure	0.0%
57	Anomaly	53.3%	Anomaly	0.0%
	Noise	28.6%	Noise	100.0%
	Structure	18.1%	Structure	0.0%
58	Anomaly	99.1%	Anomaly	0.0%
	Noise	0.9%	Noise	100.0%
	Structure	0.0%	Structure	0.0%
59	Anomaly	99.3%	Anomaly	2.5%
	Noise	0.7%	Noise	97.5%
	Structure	0.0%	Structure	0.0

Figure 6.37: The misclassified examples for the Noise class. Orange color indicates a misclassified label, and blue a correctly classified label.



Structure	Model A		Model B	
82	Anomaly	100.0%	Anomaly	99.3%
	Noise	0.0%	Noise	0.0%
	Structure	0.0%	Structure	0.6%
83	Anomaly	100.0%	Anomaly	0.4%
	Noise	0.0%	Noise	0.0%
	Structure	0.0%	Structure	99.6%
84	Anomaly	100.0%	Anomaly	98.7%
	Noise	0.0%	Noise	0.0%
	Structure	0.0%	Structure	1.3%
85	Anomaly	100.0%	Anomaly	44.0%
	Noise	0.0%	Noise	0.0%
	Structure	0.0%	Structure	56.0%
86	Anomaly	90.4%	Anomaly	33.3%
	Noise	0.0%	Noise	0.0%
	Structure	9.6%	Structure	66.7%
87	Anomaly	95.9%	Anomaly	92.1%
	Noise	0.0%	Noise	0.0%
	Structure	4.1%	Structure	7.9%
89	Anomaly	75.7%	Anomaly	31.3%
	Noise	0.0%	Noise	0.0%
	Structure	24.3%	Structure	68.7%
95	Anomaly	98.5%	Anomaly	0.2%
	Noise	0.0%	Noise	0.0%
	Structure	1.5%	Structure	99.8%

Figure 6.38: The misclassified examples for the Structure class. Orange color indicates a misclassified label, and blue a correctly classified label.

The Structure class's incorrect predictions (*Table 6.7*) are the examples 82, 83, 84, 85, 86, 87, 89, and 95 for model A, while for Model B are 82, 84, and 87. The examples mentioned were classified as anomalies instead of structures and are presented in *Figure 6.38*. This includes different images of the same structural feature of the examples 82, 83, 84, and 85. This particular feature is a unique pattern that was not included in the training data that is probably attributed to a collapsed roof. Model B was more accurate than Model A. However, the results are not robust. Examples 82 and 84 are misclassified, 83 is correctly classified, while 85 is a marginally correct prediction. The latter derived from a greater depth where the structure is not that well preserved, having a distorted image. The rest of the examples were derived from a structural complex that is not very well preserved, and Model B was overall more accurate than Model A. The obtained results indicate that Model B generalizes better for the Structure class than Model A. However, further improvements are required to improve the robustness of the predictions.

6.4 Closing remarks

This chapter presents and discusses the two CNN training approaches' obtained results using dataset-A and dataset-B. For both cases, AlexNet architecture was used. The performance of two optimizers, the SGD and Adam, was examined to find which one is better for learning for the two constructed datasets. For each optimizer, different batch sizes and learning rates were tested. The application of BN and dropout to different layers setup of AlexNet was also examined through a series of trials to select the one that yields better performance. The training optimizers were compared and evaluated according to their best-resulted models' performance on the test set of the current dataset used in the training process. The best settings that include the optimizer with the tuned learning rate and batch size and the corresponding BN-dropout setup were used to train the final models, one for each dataset named Model A and Model B, respectively.

Training with SGD performed better than Adam but required applying BN in most of the convolutional layers for the models to learn. Applying BN to all the convolutional layers had the best performance for both datasets A and B. Dropout applied to the hidden layers slightly improved the results by reducing overfitting. It was found better for dataset-A to apply it on the second hidden layer only, while for dataset-B, it was better to apply it on the first two hidden layers of AlexNet. The learning optimum for both cases was 0.001. As for the batch size, 32 was better for dataset-A, while 64 was better for dataset-B. This difference is more likely related to the non-random and random approach that was followed in the two datasets. Larger batch sizes tend to be more effective when selecting the examples randomly as they become more representative of the whole datasets, leading to more stable training. Last, the image augmentation techniques applied did not improve performance and were excluded from this study's training process. Training with Adam optimizer showed different and notable behavior. Learning was feasible without the need to use BN and dropout. For the case of dataset-A, the performance worsens when applied BN and dropout. On the contrary, it becomes better for dataset-B when applied BN to convolutional layers 1, 2, 3, & 5 and after ~40 training epochs. However, it was still performing worse than SGD. Dropout did not show any performance improvements, and neither did image augmentation.

The final Model A was trained with SGD using a batch size of 32, a momentum of 0.9, and a starting learning rate at 0.001. Due to the noisy validation curves and overfitting, an extra step was made where the learning rate is reduced by half every ten epochs after the first ten epochs. BN was applied to all five convolutional layers, while dropout was applied with a rate of 0.5 to the second hidden layer. This step slightly improved its performance. The model reaches a pick classification accuracy of 91.17% on the dataset-A test set before it started to overfit. The final Model B was also trained with SGD using a batch size of 64, a momentum of 0.9, and a learning rate of 0.001. Due to the exceedingly high performance (~99%) using a fixed learning rate, further tests were not required. BN was also applied to all five convolutional layers while, dropout, of rate 0.5, was found better to use in the first two hidden layers. This model reached a pick classification accuracy of 99.65% calculated on the test-set of dataset B. When comparing the two models' training process, Model B was easier to train as the random split followed for the test set makes the latter more representative of the training set. For this reason, the produced learning curves are less noisy and easier to interpret. Further, the validation accuracy was much higher, and it also has less overfitting than Model A.

The last verdict on which model generalizes the best was made on an evaluation set with examples that were entirely excluded from the training process. The results showed that Model B generalizes better, reaching a classification accuracy of 92% over 85% resulted from Model A. The obtained results provided some useful insights into constructing a dataset and training AlexNet CNN to achieve high classification performance. Further, it leaves space for future improvements that will be discussed in the next chapters.

7. CONCLUSIONS & FUTURE WORK

This research investigated the automatic detection of ancient buried structures from GPR data using a deep learning approach. Specifically, CNNs and AlexNet architecture that performs image classification is examined. This was realized through a series of actions and steps that included the data selection and preparation, dataset construction, the implementation of AlexNet, training of AlexNet, applying methods and techniques to improve performance, and last, evaluating learning using unseen from the training process data. The results proved that automatic detection of archaeological remains from GPR data is feasible, and DL algorithms and CNNs constitute the right approach to achieve it. In this conclusory chapter, a summary of the research objectives and the performed task is given along with the most important findings. The chapter closes with the planned future work and suggestions for further research.

7.1 SUMMARY & HIGHLIGHTS

In the following paragraphs, a review of the research objectives and the most important findings are summarized.

7.1.1 On data selection & preparation

Data selection and preparation were the first steps in creating a proper dataset so that CNNs can learn from it for this study purposes. Real GPR data were only considered while the chosen representation was C-scans. In C-scans, the buried structures have more distinct patterns than the ones that appear in B-scans. In the former, they are characterized by linear segments, while in the latter, by multiple reflections that can be similar to different buried objects, other than structures.

Synthetic data were not used, as it was found challenging and computationally demanding to simulate with the desired realism such data.

The data used were collected from several archeological sites located in Greece, Cyprus, and Sicily. The Noggin GPR system was used in all cases equipped with a 250MHz antenna, while the data was performed using survey grids. The data were processed in MATLAB using standard GPR processing methods and techniques, and C-scans were extracted from 470 grids. Several ancient structures are identified in the extracted images of different archaeological periods, including small houses, more complex and bigger structures, roads, fortifications, even entire city blocks. These are excellent examples that reflect the GPR's capability to map the past's architectural remains in detail. The main drawbacks were the signal attenuation and the increased noise level caused mainly by the rough terrain, which caused air gaps between the antenna and ground surface. The latter causes artifacts that appear almost in the whole recording time range. The main issue was the stripping noise that sometimes matched the structure's orientation, making the interpretation uncertain. Standard processing techniques can remove that noise up to a degree, having to choose between useful information removal or having noise residues.

All the images that exhibit structures were gathered only to find out that less than one hundred showed unique and clear structural patterns, which is not enough to train a CNN from scratch. The majority of C-scans featuring structures were a mixture of noise in stripping form and unidentified geophysical anomalies with irregular shapes. For this reason, an overlapping sliding window approach was used to crop square sub-regions of the selected C-scans and, at the same time, to increase the image number that can be used for training. This approach was beneficial, judging from the final results of this research and the predictions made on the evaluation set, where descent generalization was achieved. More importantly, learning would not be feasible without increasing the number of images employed in this method.

7.1.2 On dataset construction

With the candidate images prepared, the dataset construction followed. The classification labels were set, and the selected examples were split into a training set and a test set. Three classification labels were chosen based on the dominant features observed in the processed C-scans and are structures, noise, and geophysical anomalies. Representative examples were carefully selected for each label to trying to avoid identical or mixed examples. The images gathered for each class were 6125, making a total of 18375.

For splitting, the 80-20 rule was used for the number of the examples in the training set and test set. A dilemma faced was whether to split the data in a random way or use for a test set examples solely from one archaeological site. The former would be ideal if more data from more archaeological sites were available and no sliding window approach was used. However, in this case, with limited data, it might result in a non-representative test set that would negatively impact the training process. The random approach with the sliding window method was expected to have an adequate training performance, but the main question was how well could generalize when used entirely new data. Hence, two datasets were constructed to test which one generalizes better. Dataset-A describes the non-random split approach, and Dataset-B the random split approach.

The obtained results show that models trained with Dataset-B and the random approach were overall better in both training performance and generalization that was tested in the final evaluation set. However, it was found that more data are required to improve the prediction's robustness.

7.1.3 On implementing and training models with AlexNet

AlexNet was implemented in Python using Tensorflow library with Kera API. The implementation was straightforward as proposed in the original article, using a 227x227 input size, same size convolutional and pooling filters, and same biases and weight initialization. For training, two popular optimizers were tested and compared, the SGD with momentum and Adam. For each optimizer, different setups of BN and dropout were tested to find the one the yields better performance. For those best-performed setups, tuning of the learning rate and batch size was performed to study their impact on the training process. Image augmentation was also tested, choosing image transforms that produce real-like images without distortion of the feature. Additionally, early stopping was considered useful when a model started to overfit. Last, a learning rate scheduler, which reduces the learning rate by half every ten epochs, was also tested. The abovementioned process was applied twice, once for each dataset.

For the training performance and evaluation, the learning curves of the loss function and accuracy for both the training and test set with respect to the training epochs were employed. Confusion matrices were also used to evaluate the classification performance and find the examples that were either correctly classified or misclassified to study them. These two tools were proven handy in understanding and improving the learning process.

The above-mentioned comparison results and tests showed some useful insights on how to train CNNs using AlexNet architecture effectively. SGD with momentum required the applications of BN in at least four convolutional layers for the model to learn, while Adam required none. However, the performance and classification accuracy achieved with SGD and momentum was better than Adam, where BN and dropout did not significantly improve the performance. Hence, the Adam optimizer exhibits more overfitting. BN had a more significant impact on performance than dropout did. As mentioned, it was essential for learning when SGD with momentum was used. For that case, the best results were obtained by applying BN to all five convolutional layers for both datasets. Dropout with a rate of 0.5 further improved the results when applied either in the first two fully-connected layers for Dataset B or only in the second for Dataset A.

Tuning of the learning rate and batch size was performed using Keras tuner library for both optimizers and datasets. It appeared that the learning rate had a higher impact than the batch size. The former seems to affect more the learning capability and how fast it can be achieved, while the latter affects the training stability and the fluctuations in the accuracy scores. The highest accuracy of 91.17% was scored for dataset-A using SGD with momentum 0.9, learning rate 0.0001, and a batch size of 32. Respectively, for dataset-B, the highest of 99.65% accuracy was scored from SGD with momentum 0.9, learning 0.001, and batch size of 64.

Unexpectedly data augmentation did not help in improving the generalization. On the contrary, it worsened the results for both dataset and optimizers. The reasons remain unknown, and it can only be assumed that the chosen transformations were not representative, and a handcrafted augmentation approach is required. Further research is required. Last, the learning rate scheduler tested on dataset-A showed no further gains. The scheduler was not applied on Dataset-B as its performance was already high.

7.1.4 Final Evaluation

The final evaluation was made on a much smaller dataset with examples derived from Ancient Halos' archaeological sites in Thessaly and Sissi in Heraklion. The dataset includes 32 examples of geophysical anomalies, 32 examples of stripping noise, and 36 examples of structures. The images were used as inputs to make predictions using the best-obtained models, named model-A and model-B, resulting from training with dataset-A and dataset-B. Confusion matrices were used to evaluate the results and find the misclassified examples.

The results indicated that model-B performs overall better than model-A, reaching a classification score of 92% over 85%. This showed that the random split approach regarding dataset construction should be preferred as it is easier to train and generalizes better. Model A was 100% accurate in predicting the geophysical anomaly class examples, while Model B predicted 30 out of 32 correctly, where the two examples were mistaken as structures. Further, Model B misclassified three noise examples as a structure where model-A made seven mistakes, classifying three examples as structure and four as a geophysical anomaly. Model-B was also more accurate than Model-A in predicting the structure class examples. Model-B made three mistakes where the structure examples were classified as anomalies, while Model-A made mistaken eight of the structure examples as anomalies. However, it was found that more data are required to improve the predictions' accuracy.

To summarize, the best results of this study were obtained with:

- dataset-B following the random split approach,
- BN applied to all the convolutional layers,
- dropout of 0.5 applied to the first two hidden layers
- training using SGD with momentum 0.9, a constant learning rate at 0.001, and a batch size of 64.

The obtained results triggered many ideas for improvements and some future directions that are discussed in the following paragraphs.

7.2 FUTURE WORK

The obtained results and good performance showed that this is a very promising direction and automatic detection of buried structures is an achievable task. However, this is considered only the beginning, with many things left to improve, investigate, and try.

First, one thing that needs improvement is the training dataset. This involves increasing the image number, perform more tests to split the examples for training effectively, and enriching the variety of examples as much as possible. This can be achieved either through new data collection and data sharing (more case studies, different GPR systems, and antennas) or through synthetic data. For the latter, one idea is to use the transformed images by image augmentation for training. Another idea is to create synthetic images from the real ones using the Generative Adversarial Networks (GANs).

An interesting method to be implemented and tested in the near future is the *Class Activation Mapping* (CAM) that visualizes the parts of the image which CNN employs to decide upon the assigned label class using heatmaps. This information could be useful to improve the datasets and to understand what CNN has learned.

A couple of techniques recommended by many authors and were not tested in this thesis are ensemble learning and transfer learning. In the former, weights obtained from different models are averaged, while in the latter, the weights are initialized from a trained model. Both methods are reported to improve classification performance and the training process, even when the data are limited and small in numbers.

Next is to implement and try more recent and deeper architectures than AlexNet. VGG is next on the list, as it uses convolutional filters of smaller size and successive convolutional layers, which have reportedly improved classification performance. One suggestion for future work is to combine CNNs with Recurrent Neural Network (RNN) to add meaningful full text in the classification results. This could open a new perspective in the GPR data interpretation.

Aside from classification tested here, image segmentation is a very promising direction for GPR images, as shown in the recent study by Küçükdemirci and Sarris [96]. A future direction could be the application of segmentation in 3D GPR volumes to extract 3D of the subsurface, a GPR representation that is currently lacking. Last, CNNs could be adopted in data processing and denoising. One suggestion is to train CNN to detect noise through segmentation and then filter out the noise.

BIBLIOGRAPHY

[1] F. Rainey, “Archaeology: New tools for an old art: Thermoluminescence, magnetometers, satellite photography, and other techniques and instruments aid the archaeologist,” *IEEE Spectr.*, vol. 13, no. 9, pp. 39–43, 1976.

[2] B. Bevan and J. Kenyon, “Ground-penetrating radar for historical archaeology,” *MASCA Newsl.*, vol. 11, no. 12, pp. 2–17.

[3] A. D. Booth, N. T. Linford, R. A. Clark, and T. Murray, “Three-dimensional, multi-offset ground-penetrating radar imaging of archaeological targets,” *Archaeol. Prospect.*, vol. 15, no. 2, pp. 93–112, 2008.

[4] D. Goodman, A. Novo, G. Morelli, S. Piro, D. Kutrubes, and H. Lorenzo, “Advances in GPR imaging with multi-channel radar systems from engineering to archaeology,” in *Symposium on the Application of Geophysics to Engineering and Environmental Problems 2011*, 2011, pp. 416–422.

[5] D. Goodman and S. Piro, “Multi-channel GPR,” in *GPR Remote Sensing in Archaeology*, Springer Berlin Heidelberg, 2013, pp. 175–185. Accessed: Jan. 23, 2015. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-31857-3_9

[6] N. Linford, “From hypocaust to hyperbola: ground-penetrating radar surveys over mainly Roman remains in the UK,” *Archaeol. Prospect.*, vol. 11, no. 4, pp. 237–246, 2004, doi: https://doi.org/10.1002/arp.238.

[7] B. A. Berard and J. M. Maillol, “Common- and multi-offset ground-penetrating radar study of a Roman villa, Tourega, Portugal,” *Archaeol. Prospect.*, vol. 15, no. 1, pp. 32–46, 2008, doi: https://doi.org/10.1002/arp.319.

[8] L. Verdonck, “Detection of Buried Roman Wall Remains in Ground-penetrating Radar Data using Template Matching,” *Archaeol. Prospect.*, vol. 23, no. 4, pp. 257–272, 2016, doi: https://doi.org/10.1002/arp.1540.

[9] N. T. Linford and P. K. Linford, “Ground penetrating radar survey over a Roman building at Groundwell Ridge, Blunsdon St Andrew, Swindon, UK,” *Archaeol. Prospect.*, vol. 11, no. 1, pp. 49–55, 2004, doi: https://doi.org/10.1002/arp.220.

[10] L. Verdonck and F. Vermeulen, “GPR survey at the Roman town of Mariana (Corsica),” *ArcheoSciences Rev. Archéom.*, no. 33 (suppl.), Art. no. 33 (suppl.), Oct. 2009, doi: 10.4000/archeosciences.1635.

[11] D. Goodman *et al.*, “Chapter 15 - GPR Archaeometry,” in *Ground Penetrating Radar Theory and Applications*, Harry M. Jol, Ed. Amsterdam: Elsevier, 2009, pp. 479–508. Accessed: Jul. 22, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444533487000156

[12] L. B. Conyers and J. Leckebusch, “Geophysical archaeology research agendas for the future: Some ground-penetrating radar examples,” *Archaeol. Prospect.*, vol. 17, no. 2, pp. 117–123, 2010.

[13] A. Vafidis *et al.*, “Integrated geophysical studies at ancient Itanos (Greece),” *J. Archaeol. Sci.*, vol. 32, no. 7, pp. 1023–1036, Jul. 2005, doi: 10.1016/j.jas.2005.02.007.

[14] F. Yiğit, G. Tucker, and S. Özcelik, “GPR (Ground Penetrating Radar) Survey at Notion (June 2017),” in *2018 17th International Conference on Ground Penetrating Radar (GPR)*, Jun. 2018, pp. 1–4. doi: 10.1109/ICGPR.2018.8441675.

[15] S. Imposa, S. Grassi, G. Patti, and D. Boso, “New data on buried archaeological ruins in Messina area (Sicily-Italy) from a ground penetrating radar survey,” *J. Archaeol. Sci. Rep.*, vol. 17, pp. 358–365, Feb. 2018, doi: 10.1016/j.jasrep.2017.11.031.

[16] J. C. Donati *et al.*, “New Insights into the Urban Plans of Demetrias and Pherai from Integrated Geophysics and Satellite Remote Sensing”.

[17] J. C. Donati, A. Sarris, C. Cuenca-García, T. Kalayci, N. Papadopoulos, and F.-X. Simon, “The Urban Plan of Mantinea in the Peloponnese: An Intergrated Geophysical and Satellite Remote Sensing Fieldwork Campaign,” presented at the Archaeological Institute of America 2015 Annual Meeting, New Orleans, LA, 2015.

[18] A. Sarris *et al.*, “Amalgamation of Satellite Remote Sensing and Geophysical Prospection for the Investigation of Ancient Cities: Two Case Studies from Demetrias and Pherai at the Region of Magnesia, Thessaly, Greece,” in *8th Congress of the Balkan Geophysical Society*, 2015, vol. 2015, no. 1, pp. 1–5.

[19] A. Sarris, N. Papadopoulos, and V. Trigkas, “Recovering the urban network of ancient Sikyon through multi-component geophysical approaches,” in *Layers of Perception*, Berlin, 2008, pp. 11–16. doi: 10.11588/propylaeumdok.00000486.

[20] “The Greek colony of Naxos in Sicily: mapping the town plan and geophysical survey | FIA.” http://www.finninstitute.gr/naksoksen-kreikkalainen-siirtokunta-sisiliassa-kaupunkikuvan-kartoitus-ja-geofyysinen-tutkimus/ (accessed Sep. 21, 2020).

[21] I. Trinks, J. Gustafsson, J. Emilsson, C. Gustafsson, B. Johansson, and J. Nissen, “Efficient, large-scale archaeological prospection using a true 3D GPR array system,” *ArchéoSciences Rev. Archéom.*, no. 33 (suppl.), pp. 367–370, Oct. 2009.

[22] R. Filzwieser *et al.*, “Integration of Complementary Archaeological Prospection Data from a Late Iron Age Settlement at Vesterager—Denmark,” *J. Archaeol. Method Theory*, vol. 25, no. 2, pp. 313–333, Jun. 2018, doi: 10.1007/s10816-017-9338-y.

[23] T. M. Urban, J. F. Leon, S. W. Manning, and K. D. Fisher, “High resolution GPR mapping of Late Bronze Age architecture at Kalavasos-Ayios Dhimitrios, Cyprus,” *J. Appl. Geophys.*, vol. 107, pp. 129–136, Aug. 2014, doi: 10.1016/j.jappgeo.2014.05.020.

[24] A. Novo, M. Solla, J.-L. M. Fenollós, and H. Lorenzo, “Searching for the remains of an Early Bronze Age city at Tell Qubr Abu al-’Atiq (Syria) through archaeological investigations

and GPR imaging," *J. Cult. Herit.*, vol. 15, no. 5, pp. 575–579, Sep. 2014, doi: 10.1016/j.culher.2013.10.006.

[25] A. Gyucha *et al.*, "Settlement nucleation in the neolithic: a preliminary report of the Körös regional archaeological project's investigations at Szeghalom-Kovácshalom and Vésztő-Mágor," *Neolit. Copp. Age Carpathians Aegean Sea Chronol. Technol. 6th 4th Millenn. BCE*, pp. 129–142, 2012.

[26] R. Deiana, J. Bonetto, and A. Mazzariol, "Integrated Electrical Resistivity Tomography and Ground Penetrating Radar Measurements Applied to Tomb Detection," *Surv. Geophys.*, vol. 39, no. 6, pp. 1081–1105, Nov. 2018, doi: 10.1007/s10712-018-9495-x.

[27] S. Santos-Assunçao, K. Dimitriadis, Y. Konstantakis, V. Perez-Gracia, E. Anagnostopoulou, and R. Gonzalez-Drigo, "Ground-penetrating radar evaluation of the ancient Mycenaean monument Tholos Acharnon tomb," *Surf. Geophys.*, vol. 14, no. 2, pp. 197–205, Jun. 2016, doi: 10.3997/1873-0604.2015030.

[28] M. Pipan, L. Baradello, E. Forte, and I. Finetti, "Ground penetrating radar study of iron age tombs in Southeastern Kazakhstan," *Archaeol. Prospect.*, vol. 8, no. 3, pp. 141–155, 2001, doi: https://doi.org/10.1002/arp.162.

[29] J. T. Herrmann, J. L. King, and J. E. Buikstra, "Mapping the internal structure of Hopewell tumuli in the Lower Illinois River Valley through archaeological geophysics," *Adv. Archaeol. Pract.*, vol. 2, no. 3, pp. 164–179, 2014.

[30] C. Vaughan, "Ground-penetrating radar surveys used in archaeological investigations," *GEOPHYSICS*, vol. 51, no. 3, pp. 595–604, 1986, doi: 10.1190/1.1442114.

[31] L. B. Ciampoli, R. Santarelli, E. M. Loreti, A. Ten, and A. Benedetto, "Structural detailing of buried Roman baths through GPR inspection," *Archaeol. Prospect.*, vol. n/a, no. n/a, doi: https://doi.org/10.1002/arp.1776.

[32] N. S. Spanoudakis, M. Manataki, V. Niniou-Kindeli, and A. P. Vafidis, "GPR Imaging at Aptera Archaeological Site," presented at the 6th Congress of Balkan Geophysical Society, Oct. 2011. Accessed: Oct. 17, 2013. [Online]. Available: http://www.earthdoc.org/publication/publicationdetails/?publication=54679

[33] G. Dean, N. Yasushi, S. Kent, P. Salvadore, H. Hiromichi, and H. Noriaki, "Advances in Imaging of Subsurface Archaeology using GPR," 한국지구물리탐사학회:학술대회논문집, pp. 8–21, 2004.

[34] N. G. Papadopoulos, A. Sarris, M. C. Salvi, S. Dederix, P. Soupios, and U. Dikmen, "Rediscovering the small theatre and amphitheatre of ancient Ierapytna (SE Crete) by integrated geophysical methods," *J. Archaeol. Sci.*, vol. 39, no. 7, pp. 1960–1973, Jul. 2012, doi: 10.1016/j.jas.2012.01.044.

[35] N. S. Spanoudakis, A. Vafidis, A. Paganis, N. Andronikidis, N. Hatzidakis, and V. Niniou-Kindeli, "Geophysical Survey at the Area of the Ancient Theater of Aptera," in *8th Congress of the Balkan Geophysical Society*, 2015, vol. 2015, no. 1, pp. 1–5.

[36] A. Sarris, N. Papadopoulos, S. Déderix, and M.-C. Salvi, "Geophysical approaches applied in the ancient theatre of Demetriada, Volos," in *First International Conference on Remote Sensing and Geoinformation of Environment*, 2013, pp. 87950H-87950H–8.

[37] M. Tohge, F. Karube, M. Kobayashi, A. Tanaka, and K. Ishii, "The use of ground penetrating radar to map an ancient village buried by volcanic eruptions," *J. Appl. Geophys.*, vol. 40, no. 1–3, pp. 49–58, Oct. 1998, doi: 10.1016/S0926-9851(97)00039-6.

[38] L. B. Conyers, "Ground-penetrating radar for landscape archaeology: method and applications," *Seeing Unseen Geophys. Landsc. Archaeol.*, pp. 245–255, 2009.

[39] N. Papadopoulos, A. Sarris, M. Yi, and J. Kim, “Urban archaeological investigations using surface 3D Ground Penetrating Radar and Electrical Resistivity Tomography methods,” *Explor. Geophys.*, vol. 40, no. 1, pp. 56–68, 2009, doi: 10.1071/EG08107.

[40] A. Sarris, T. Kalayci, I. Moffat, and M. Manataki, “An introduction to geophysical and geochemical methods in digital geoarchaeology,” in *Digital Geoarchaeology*, Springer, 2018, pp. 215–236.

[41] A. Sarris *et al.*, “Integration of geophysical surveys, ground hyperspectral measurements, aerial and satellite imagery for archaeological prospection of prehistoric sites: the case study of Vésztő-Mágor Tell, Hungary,” *J. Archaeol. Sci.*, vol. 40, no. 3, pp. 1454–1470, Mar. 2013, doi: 10.1016/j.jas.2012.11.001.

[42] M. G. Drahor, “Integrated Geophysical Investigations in Archaeological Sites: Case Studies from Turkey,” in *Archaeogeophysics: State of the Art and Case Studies*, G. El-Qady and M. Metwaly, Eds. Cham: Springer International Publishing, 2019, pp. 27–68. doi: 10.1007/978-3-319-78861-6_2.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[44] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.

[45] R. K. Sinha, R. Pandey, and R. Pattnaik, “Deep learning for computer vision tasks: A review,” presented at the International Conference on Intelligent Computing and Control (ICICC), Madurai, India, 2017.

[46] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Comput. Intell. Neurosci.*, vol. 2018, 2018.

[47] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[48] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A survey of the recent architectures of deep convolutional neural networks,” *Artif. Intell. Rev.*, vol. 53, no. 8, pp. 5455–5516, 2020.

[49] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,” *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, Apr. 1980, doi: 10.1007/BF00344251.

[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[51] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[52] W. Rawat and Z. Wang, “Deep convolutional neural networks for image classification: A comprehensive review,” *Neural Comput.*, vol. 29, no. 9, pp. 2352–2449, 2017.

[53] T. Tommasi, N. Patricia, B. Caputo, and T. Tuytelaars, “A deeper look at dataset bias,” in *Domain adaptation in computer vision applications*, Springer, 2017, pp. 37–55.

[54] F. Sultana, A. Sufian, and P. Dutta, “A review of object detection models based on convolutional neural network,” in *Intelligent Computing: Image Processing Based Applications*, Springer, 2020, pp. 1–16.

[55] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, “A review on deep learning techniques applied to semantic segmentation,” *ArXiv Prepr. ArXiv170406857*, 2017.

[56] J. Long, E. Shelhamer, and T. Darrell, “Fully convolutional networks for semantic segmentation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[57] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical image computing and computer-assisted intervention*, 2015, pp. 234–241.

[58] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 12, pp. 2481–2495, 2017.

[59] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, “Semantic image segmentation with deep convolutional nets and fully connected crfs,” presented at the 3rd International Conference on Learning Representations, San Diego, California, 2015.

[60] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, “Simultaneous detection and segmentation,” in *European Conference on Computer Vision*, 2014, pp. 297–312.

[61] P. O. Pinheiro, R. Collobert, and P. Dollár, “Learning to segment object candidates,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1990–1998.

[62] L. Capineri, P. Grande, and J. a. G. Temple, “Advanced image-processing technique for real-time interpretation of ground-penetrating radar images,” *Int. J. Imaging Syst. Technol.*, vol. 9, no. 1, pp. 51–59, 1998, doi: https://doi.org/10.1002/(SICI)1098-1098(1998)9:1<51::AID-IMA7>3.0.CO;2-Q.

[63] W. Al-Nuaimy, Y. Huang, M. Nakhkash, M. T. C. Fang, V. T. Nguyen, and A. Eriksen, “Automatic detection of buried utilities and solid objects with GPR using neural networks and pattern recognition,” *J. Appl. Geophys.*, vol. 43, no. 2, pp. 157–165, Mar. 2000, doi: 10.1016/S0926-9851(99)00055-5.

[64] E. Pasolli, F. Melgani, and M. Donelli, “Automatic Analysis of GPR Images: A Pattern-Recognition Approach,” *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 7, pp. 2206–2217, Jul. 2009, doi: 10.1109/TGRS.2009.2012701.

[65] Q. Lu, J. Pu, and Z. Liu, “Feature Extraction and Automatic Material Classification of Underground Objects from Ground Penetrating Radar Data,” *JECE*, vol. 2014, p. 28:28-28:28, Jan. 2014, doi: 10.1155/2014/347307.

[66] W. Shao, A. Bouzerdoum, S. L. Phung, L. Su, B. Indraratna, and C. Rujikiatkamjorn, “Automatic Classification of Ground-Penetrating-Radar Signals for Railway-Ballast Assessment,” *IEEE Trans. Geosci. Remote Sens.*, vol. 49, no. 10, pp. 3961–3972, Oct. 2011, doi: 10.1109/TGRS.2011.2128328.

[67] C. Maas and J. Schmalzl, “Using pattern recognition to automatically localize reflection hyperbolas in data from ground penetrating radar,” *Comput. Geosci.*, vol. 58, pp. 116–125, Aug. 2013, doi: 10.1016/j.cageo.2013.04.012.

[68] X. L. Travassos, S. L. Avila, and N. Ida, “Artificial Neural Networks and Machine Learning techniques applied to Ground Penetrating Radar: A review,” *Appl. Comput. Inform.*, Oct. 2018, doi: 10.1016/j.aci.2018.10.001.

[69] I. Giannakis, A. Giannopoulos, and C. Warren, “A Machine Learning-Based Fast-Forward Solver for Ground Penetrating Radar With Application to Full-Waveform Inversion,” *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 7, pp. 4417–4426, Jul. 2019, doi: 10.1109/TGRS.2019.2891206.

[70] C. Warren, A. Giannopoulos, and I. Giannakis, “gprMax: Open source software to simulate electromagnetic wave propagation for Ground Penetrating Radar,” *Comput. Phys. Commun.*, 2016, doi: 10.1016/j.cpc.2016.08.020.

[71] A. Giannopoulos, “Modelling ground penetrating radar by GprMax,” *Constr. Build. Mater.*, vol. 19, no. 10, pp. 755–762, Dec. 2005, doi: 10.1016/j.conbuildmat.2005.06.007.

[72] I. Giannakis, A. Giannopoulos, and C. Warren, “A Machine Learning Scheme for Estimating the Diameter of Reinforcing Bars Using Ground Penetrating Radar,” *IEEE Geosci. Remote Sens. Lett.*, pp. 1–5, 2020, doi: 10.1109/LGRS.2020.2977505.

[73] P. Asadi, M. Gindy, and M. Alvarez, “A machine learning based approach for automatic rebar detection and quantification of deterioration in concrete bridge deck ground penetrating radar B-scan images,” *KSCE J. Civ. Eng.*, vol. 23, no. 6, pp. 2618–2627, 2019.

[74] E. Skartados *et al.*, “Ground penetrating radar image processing towards underground utilities detection for robotic applications,” in *2018 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO)*, 2018, pp. 27–31.

[75] J. Sonoda and T. Kimoto, “Object Identification form GPR Images by Deep Learning,” in *2018 Asia-Pacific Microwave Conference (APMC)*, Nov. 2018, pp. 1298–1300. doi: 10.23919/APMC.2018.8617556.

[76] K. Ishitsuka, S. Iso, K. Onishi, and T. Matsuoka, “Object Detection in Ground-Penetrating Radar Images Using a Deep Convolutional Neural Network and Image Set Preparation by Migration,” *Int. J. Geophys.*, vol. 2018, 2018.

[77] D. Reichman, L. M. Collins, and J. M. Malof, “Some good practices for applying convolutional neural networks to buried threat detection in Ground Penetrating Radar,” in *2017 9th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, Jun. 2017, pp. 1–5. doi: 10.1109/IWAGPR.2017.7996100.

[78] M. Pham and S. Lefèvre, “Buried Object Detection from B-Scan Ground Penetrating Radar Data Using Faster-RCNN,” in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, Jul. 2018, pp. 6804–6807. doi: 10.1109/IGARSS.2018.8517683.

[79] U. Ozkaya, F. Melgani, M. Belete Bejiga, L. Seyfi, and M. Donelli, “GPR B scan image analysis with deep learning methods,” *Measurement*, vol. 165, p. 107770, Dec. 2020, doi: 10.1016/j.measurement.2020.107770.

[80] H. Liu, C. Lin, J. Cui, L. Fan, X. Xie, and B. F. Spencer, “Detection and localization of rebar in concrete by deep learning using ground penetrating radar,” *Autom. Constr.*, vol. 118, p. 103279, 2020.

[81] S. Yang *et al.*, “Defect segmentation: Mapping tunnel lining internal defects with ground penetrating radar data using a convolutional neural network,” *ArXiv200313120 Phys. Prepring Submitt. Constr. Build. Mater.*, Mar. 2020, Accessed: Sep. 15, 2020. [Online]. Available: http://arxiv.org/abs/2003.13120

[82] N. Kim, K. Kim, Y.-K. An, H.-J. Lee, and J.-J. Lee, “Deep learning-based underground object detection for urban road pavement,” *Int. J. Pavement Eng.*, vol. 21, no. 13, pp. 1638–1650, 2018, doi: 10.1080/10298436.2018.1559317.

[83] N. Kim, S. Kim, Y.-K. An, and J.-J. Lee, “A novel 3D GPR image arrangement for deep learning-based underground object classification,” *Int. J. Pavement Eng.*, vol. 0, no. 0, pp. 1–12, Aug. 2019, doi: 10.1080/10298436.2019.1645846.

[84] M.-S. Kang, N. Kim, J. J. Lee, and Y.-K. An, “Deep learning-based automated underground cavity detection using three-dimensional ground penetrating radar,” *Struct. Health Monit.*, vol. 19, no. 1, pp. 173–185, 2020.

[85] N. Kim, S. Kim, Y.-K. An, and J.-J. Lee, “Triplanar Imaging of 3-D GPR Data for Deep-Learning-Based Underground Object Detection,” *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.*, vol. 12, no. 11, pp. 4446–4456, 2019.

[86] S. Khudoyarov, N. Kim, and J.-J. Lee, “Three-dimensional convolutional neural network–based underground object classification using three-dimensional ground penetrating radar data,” *Struct. Health Monit.*, p. 1475921720902700, 2020.

[87] Z. Tong, J. Gao, and D. Yuan, “Advances of deep learning applications in ground-penetrating radar: A survey,” *Constr. Build. Mater.*, vol. 258, p. 120371, 2020, doi: https://doi.org/10.1016/j.conbuildmat.2020.120371.

[88] V. Kafedziski, S. Pecov, and D. Tanevski, “Detection and Classification of Land Mines from Ground Penetrating Radar Data Using Faster R-CNN,” in *2018 26th Telecommunications Forum (TELFOR)*, Nov. 2018, pp. 1–4. doi: 10.1109/TELFOR.2018.8612117.

[89] S. Lameri, F. Lombardi, P. Bestagini, M. Lualdi, and S. Tubaro, “Landmine detection from GPR data using convolutional neural networks,” in *2017 25th European Signal Processing Conference (EUSIPCO)*, Aug. 2017, pp. 508–512. doi: 10.23919/EUSIPCO.2017.8081259.

[90] P. Bestagini, F. Lombardi, M. Lualdi, F. Picetti, and S. Tubaro, “Landmine Detection Using Autoencoders on Multipolarization GPR Volumetric Data,” *IEEE Trans. Geosci. Remote Sens.*, 2020.

[91] J. Zheng, X. Teng, J. Liu, and X. Qiao, “Convolutional Neural Networks for Water Content Classification and Prediction With Ground Penetrating Radar,” *IEEE Access*, vol. 7, pp. 185385–185392, 2019, doi: 10.1109/ACCESS.2019.2960768.

[92] J. K. Alvarez and S. Kodagoda, “Application of deep learning image-to-image transformation networks to GPR radargrams for sub-surface imaging in infrastructure monitoring,” in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, May 2018, pp. 611–616. doi: 10.1109/ICIEA.2018.8397788.

[93] B. Liu *et al.*, “GPRInvNet: Deep Learning-Based Ground Penetrating Radar Data Inversion for Tunnel Lining,” *ArXiv191205759 Phys.*, Dec. 2019, Accessed: Sep. 15, 2020. [Online]. Available: http://arxiv.org/abs/1912.05759

[94] W. Rice, M. Omwenga, D. Wu, and Y. Liang, “Enhanced Underground Object Detection with Conditional Adversarial Networks,” Aug. 2019, pp. 59–63. Accessed: Nov. 20, 2020. [Online]. Available: https://issatconferences.org/Abstracts/DSIS/Content_DSIS/content_DSIS_19/59.html

[95] J. M. Malof *et al.*, “A large-scale multi-institutional evaluation of advanced discrimination algorithms for buried threat detection in ground penetrating radar,” *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 9, pp. 6929–6945, 2019.

[96] M. Küçükdemirci and A. Sarris, “Deep learning based automated analysis of archaeo-geophysical images,” *Archaeol. Prospect.*, vol. 27, no. 2, pp. 107–118, 2020, doi: 10.1002/arp.1763.

[97] M. A. H. El-said, “Geophysical Prospection of Underground Water in the Desert by Means of Electromagnetic Interference Fringes,” *Proc. IRE*, vol. 44, no. 1, pp. 24–30, 1956, doi: 10.1109/JRPROC.1956.274846.

[98] A. P. Annan, “GPR—History, Trends, and Future Developments,” *Subsurf. Sens. Technol. Appl.*, vol. 3, no. 4, pp. 253–270, Oct. 2002, doi: 10.1023/A:1020657129590.

[99] J. L. Davis and A. P. Annan, “Ground-Penetrating Radar for High-Resolution Mapping of Soil and Rock Stratigraphy1,” *Geophys. Prospect.*, vol. 37, no. 5, pp. 531–551, 1989, doi: 10.1111/j.1365-2478.1989.tb02221.x.

[100] J. A. Doolittle and J. R. Butnor, “Chapter 6 - Soils, Peatlands, and Biomonitoring,” in *Ground Penetrating Radar Theory and Applications*, Harry M. Jol, Ed. Amsterdam: Elsevier, 2009, pp. 177–202. Accessed: Jul. 22, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444533487000065

[101] A. K. Benson, “Applications of ground penetrating radar in assessing some geological hazards: examples of groundwater contamination, faults, cavities,” *J. Appl. Geophys.*, vol. 33, no. 1–3, pp. 177–193, 1995, doi: 10.1016/0926-9851(95)90040-3.

[102] M. Grasmueck, “3-D ground-penetrating radar applied to fracture imaging in gneiss,” *GEOPHYSICS*, vol. 61, no. 4, pp. 1050–1064, 1996, doi: 10.1190/1.1444026.

[103] U. Theune, D. Rokosh, M. Sacchi, and D. Schmitt, “Mapping fractures with GPR: A case study from Turtle Mountain,” *GEOPHYSICS*, vol. 71, no. 5, pp. B139–B150, 2006, doi: 10.1190/1.2335515.

[104] N. J. Cassidy, “Chapter 2 - Electrical and Magnetic Properties of Rocks, Soils and Fluids,” in *Ground Penetrating Radar Theory and Applications*, Harry M. Jol, Ed. Amsterdam: Elsevier, 2009, pp. 41–72. Accessed: Jul. 22, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444533487000028

[105] J. J. Daniels, “Ground Penetrating Radar Fundamentals,” Ohio, Appendix to U.S.EPA, Region V, 2000. [Online]. Available: http://www.earthsciences.osu.edu/~jeff/Library/BASICS.PDF

[106] A. P. Annan, “Chapter 1 - Electromagnetic Principles of Ground Penetrating Radar,” in *Ground Penetrating Radar Theory and Applications*, Harry M. Jol, Ed. Amsterdam: Elsevier, 2009, pp. 1–40. Accessed: Jul. 22, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444533487000016

[107] N. J. Cassidy, “Chapter 5 - Ground Penetrating Radar Data Processing, Modelling and Analysis,” in *Ground Penetrating Radar Theory and Applications*, Harry M. Jol, Ed. Amsterdam: Elsevier, 2009, pp. 141–176. Accessed: Jul. 22, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780444533487000053

[108] P. Y. Ktonas and N. Papp, “Instantaneous envelope and phase extraction from real signals: Theory, implementation, and an application to EEG analysis,” *Signal Process.*, vol. 2, no. 4, pp. 373–385, Oct. 1980, doi: 10.1016/0165-1684(80)90079-1.

[109] A. Batziou-Efstathiou, “Δημητριάς,” *Athens Archaeol. Receipts Fund*, 2001.

[110] F. Rumscheid and W. Koenigs, *Priene: a guide to the" Pompeii of Asia Minor"*. Ege Yayinlar, 1998.

[111] P. Zanker, *Pompeii: public and private life*, vol. 11. Harvard University Press, 1998.

[112] S. Hodkinson and H. Hodkinson, “Mantineia and the Mantinike: Settlement and Society in a Greek Polis,” *Annu. Br. Sch. Athens*, vol. 76, pp. 239–296, Nov. 1981, doi: 10.1017/S0068245400019547.

[113] F. E. Winter, “Arkadian Notes I: Identification of the Agora Buildings at Orchomenos and Mantinea,” *EchCl*, vol. 31, pp. 235–46, 1987.

[114] F. E. Winter, “Arkadian Notes II: the Walls of Mantinea, Orchomenos and Kleitor,” *EchCl Ns*, vol. 8, pp. 192–96, 1989.

[115] A. Sarris, “Shallow Depth Geophysical Investigation Through the Application of Magnetic and Electric Resistance Techniques: AN Evaluation Study of the Responses of Magnetic and Electric Resistance Techniques to Archaeogeophysical Prospection Surveys in Greece and Cyprus,” *Thesis PHD-- Univ. Neb. - Linc. 1992Source Diss. Abstr. Int. Vol. 53-08 Sect. B Page*

4002, 1992, Accessed: Sep. 26, 2014. [Online]. Available: http://adsabs.harvard.edu/abs/1992PhDT.......248S

[116] G. Fougères, *Mantinée et l'Arcadie*. A. Fontemoing, 1898.

[117] T. M. Mitchell, *Machine learning*. McGraw-hill New York, 1997.

[118] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from data*, vol. 4. AMLBook New York, NY, USA:, 2012.

[119] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[120] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[121] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[122] T. O. Ayodele, "Types of machine learning algorithms," *New Adv. Mach. Learn.*, pp. 19–48, 2010.

[123] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," 2011.

[124] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Natl. Acad. Sci.*, vol. 79, no. 8, pp. 2554–2558, 1982.

[125] J. J. Hopfield and D. W. Tank, "Computing with neural circuits: A model," *Science*, vol. 233, no. 4764, pp. 625–633, 1986.

[126] A. Krenker, J. Bester, and A. Kos, "Introduction to the artificial neural networks," in *Artificial neural networks-methodological advances and biomedical applications*, IntechOpen, 2011.

[127] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.

[128] I. Stephen, "Perceptron-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 50, no. 2, p. 179, 1990.

[129] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, 2013, pp. 6645–6649.

[130] Y. Abu-Mostafa, M. Magdon-Ismail, and H. Lin, "E-Chapter 7: Neural Networks," *Learn. Data*, 2015.

[131] B. L. Kalman and S. C. Kwasny, "Why tanh: choosing a sigmoidal function," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, 1992, vol. 4, pp. 578–581.

[132] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized MLP architectures of neural networks," *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.

[133] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," 2010.

[134] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *ArXiv Prepr. ArXiv150500853*, 2015.

[135] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *ArXiv Prepr. ArXiv151107289*, 2015.

[136] S. Sharma, "Activation functions in neural networks," *Data Sci.*, vol. 6, 2017.

[137] S. Koturwar and S. Merchant, "Weight Initialization of Deep Neural Networks(DNNs) using Data Statistics," *ArXiv171010570 Cs Stat*, Mar. 2018, Accessed: Sep. 11, 2020. [Online]. Available: http://arxiv.org/abs/1710.10570

[138] S. K. Kumar, "On weight initialization in deep neural networks," *ArXiv Prepr. ArXiv170408863*, 2017.

[139] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[140] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[141] C. A. de Sousa, “An overview on weight initialization methods for feedforward neural networks,” in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 52–59.

[142] *Handbook of Formulas and Tables for Signal Processing | Alexander D. Poularikas | Springer*. Accessed: Feb. 09, 2017. [Online]. Available: http://www.springer.com/gp/book/9783540648345

[143] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications*, 3rd ed. Prentice Hall, 1995.

[144] S. W. Smith, “The scientist and engineer’s guide to digital signal processing,” 1997.

[145] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*, vol. 5. McGraw-hill New York, 1995.

[146] J. Wu, “Introduction to convolutional neural networks,” *Natl. Key Lab Nov. Softw. Technol. Nanjing Univ. China*, pp. 5–23, 2017.

[147] W. Shang, K. Sohn, D. Almeida, and H. Lee, “Understanding and improving convolutional neural networks via concatenated rectified linear units,” in *international conference on machine learning*, 2016, pp. 2217–2225.

[148] S. R. Dubey and S. Chakraborty, “Average biased ReLU based CNN descriptor for improved face retrieval,” *ArXiv Prepr. ArXiv180402051*, 2018.

[149] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” in *International conference on artificial neural networks*, 2010, pp. 92–101.

[150] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex,” *J. Physiol.*, vol. 195, no. 1, pp. 215–243, 1968.

[151] D. H. Hubel and T. N. Wiesel, “Receptive fields of single neurones in the cat’s striate cortex,” *J. Physiol.*, vol. 148, no. 3, p. 574, 1959.

[152] W. Luo, Y. Li, R. Urtasun, and R. Zemel, “Understanding the effective receptive field in deep convolutional neural networks,” in *Advances in neural information processing systems*, 2016, pp. 4898–4906.

[153] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, “Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach,” *Procedia Comput. Sci.*, vol. 132, pp. 679–688, Jan. 2018, doi: 10.1016/j.procs.2018.05.069.

[154] Z. Zhang, “Derivation of backpropagation in convolutional neural network (cnn),” *Univ. Tenn. Knoxv. TN*, 2016.

[155] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv160904747 Cs*, Jun. 2017, Accessed: Oct. 04, 2020. [Online]. Available: http://arxiv.org/abs/1609.04747

[156] B. T. Polyak, “Some methods of speeding up the convergence of iteration methods,” *USSR Comput. Math. Math. Phys.*, vol. 4, no. 5, pp. 1–17, 1964.

[157] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *International conference on machine learning*, 2013, pp. 1139–1147.

[158] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *ArXiv Prepr. ArXiv14126980*, 2014.

[159] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization.," *J. Mach. Learn. Res.*, vol. 12, no. 7, 2011.

[160] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited On*, vol. 14, no. 8, 2012.

[161] S. Bera and V. K. Shrivastava, "Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification," *Int. J. Remote Sens.*, vol. 41, no. 7, pp. 2664–2683, 2020.

[162] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in neural information processing systems*, 2017, pp. 4148–4158.

[163] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy," *ArXiv Prepr. ArXiv171010686*, 2017.

[164] A. Fawzi, H. Samulowitz, D. Turaga, and P. Frossard, "Adaptive data augmentation for image classification," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 3688–3692.

[165] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid, "Transformation pursuit for image classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 3646–3653.

[166] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[167] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv Prepr. ArXiv150203167*, 2015.

[168] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," *ArXiv Prepr. ArXiv180900846*, 2018.

[169] "Noggin - User's Guide." Sensors & Software Inc., 2011. Accessed: Sep. 19, 2020. [Online]. Available: https://www.sensoft.ca/products/noggin/overview/

[170] S. N. Spanoudakis and A. Vafidis, "GPR-PRO: A MATLAB module for GPR data processing," in *2010 13th International Conference on Ground Penetrating Radar (GPR)*, 2010, pp. 1–5. doi: 10.1109/ICGPR.2010.5550131.

[171] A. Tzanis, "Matgpr: A freeware matlab package for the analysis of common-offset GPR data," in *Geophysical Research Abstracts*, 2006, vol. 8.

[172] G. F. Margrave and K. W. Hall, "CREWES Matlab Toolbox," *Consortium for Research in Elastic Wave Exploration Seismology*. https://www.crewes.org/ResearchLinks/FreeSoftware/ (accessed Jan. 05, 2020).

[173] M. D. Sacchi, "Signal Analysis and Imaging Group - SeismicLab - Matlab Scripts for Seismic Data Processing." http://seismic-lab.physics.ualberta.ca/ (accessed Sep. 17, 2020).

[174] "IGEAN." https://igean.ims.forth.gr/?q=en (accessed Sep. 20, 2020).

[175] A. Sarris *et al.*, "Opening a New Frontier in the Neolithic Settlement Patterns of Eastern Thessaly, Greece.," in *Communities, Landscapes, and Interaction in Neolithic Greece*, Rethymno, 2017, vol. 20, pp. 27–48.

[176] T. Kalaycı and A. Sarris, "Multi-Sensor Geomagnetic Prospection: A Case Study from Neolithic Thessaly, Greece," *Remote Sens.*, vol. 8, no. 11, p. 966, 2016.

[177] M. Manataki, A. Sarris, and A. Vafidis, “Employing CEEMD for Improving GPR Images-A Case Study from a Neolithic Settlement in Thessaly, Greece,” 2015.

[178] C. Cuenca-García *et al.*, “Walking over Magoulas: Mapping Neolithic Tell Settlements in Thessaly (Greece) using Integrated Archaeo-geophysical Techniques.,” presented at the 17th IUPPS Congress, Burgos, Sep. 2014.

[179] M. Manataki *et al.*, “Studying the Variation of Geophysical Signals of the Architectural Attributes of the Neolithic Tells and Landscape,” presented at the In International Conference on Computer Applications in Archaeology, CAA 2015, University of Sienna, Sienna, Italy, Apr. 2015.

[181] T. Kalayci, F.-X. Simon, and A. Sarris, “A Manifold approach for the investigation of early and middle Neolithic settlements in Thessaly, Greece,” *Geosciences*, vol. 7, no. 3, p. 79, 2017.

[182] E. Alram-Stern *et al.*, “Magoula Visviki revisited: comparing past excavations’ data to recent geophysical research.”.

[183] K. Vouzaxakis, “A new Neolithic site in Thessaly (Greece): the Belitsi magoula,” *Antiquity*, vol. 75, no. 287, pp. 15–16, 2001.

[184] F.-X. Simon, T. Kalayci, J. C. Donati, C. C. Garcia, M. Manataki, and A. Sarris, “How efficient is an integrative approach in archaeological geophysics? Comparative case studies from Neolithic settlements in Thessaly (Central Greece),” *Surf. Geophys.*, vol. 13, no. 6, pp. 633–643, 2015.

[185] “POLITEIA.” https://www.ims.forth.gr/en/project/view?id=73 (accessed Sep. 21, 2020).

[186] J. C. Donati *et al.*, “A regional approach to ancient urban studies in Greece through multi-settlement geophysical survey,” *J. Field Archaeol.*, vol. 42, no. 5, pp. 450–467, 2017.

[187] J. C. Donati and A. Sarris, “Geophysical survey in Greece: recent developments, discoveries and future prospects,” *Archaeol. Rep.*, vol. 62, pp. 63–76, 2016.

[188] M. Manataki, A. Sarris, and A. Vafidis, “Combining CEEMD and predictive deconvolution for the suppression of multiple reflections and coherent noise in GPR signals,” in *2015 8th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, 2015, pp. 1–4.

[189] A. Sarris *et al.*, “Revealing the urban features of the ancient greek city of Mantineia through the employment of ground penetrating radar,” in *2015 8th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, 2015, pp. 1–4.

[190] M. Manataki, A. Sarris, J. C. Donati, C. Cuenca Garcia, and T. Kalayci, “GPR: Theory and Practice in Archaeological Prospection,” in *Best Practices of Geoinformatic Technologies for the Mapping of Archaeolandscapes*, Archaeopress Archaeology, 2015, pp. 13–24.

[191] “ArcLand - Home.” http://www.arcland.eu/ (accessed Sep. 21, 2020).

[192] A. Sarris and J. Bintliff, “Scouring the surface and peering beneath it at the ancient city of Hyettos, Boeotia, Greece,” presented at the Sensing the Past, Final ArcLand Conference, Frankfurt, Feb. 2015.

[193] A. Sarris, N. Papadopoulos, C. Cuenca-Garcia, D. Alexakis, M. Manatakia, and G. Cantoro, “Exposing the Urban Plan of the ancient city of Hyettos, Boeotia, Greece,” *Archaeol. Pol.*, vol. 53, pp. 364–367, 2015.

[194] “The Sissi Archaeological Project | Sarpedon.” https://sarpedon.be/ (accessed Sep. 21, 2020).

[195] J. Driessen, “A new ceremonial centre at Sissi (Nomos Lassithiou),” 2016.

[196] A. Sarris, M. Manataki, S. Dederix, and J. Driessen, “Revealing the structural details of the Minoan settlement of Sissi, Eastern Crete, through geophysical investigations,” in *12th International Conference of Archaeological Prospection*, 2017, pp. 206–208.

[197] S. Déderix, A. Sarris, and M. Manataki, “Geophysical Investigations at Sissi; 2015-2016,” in *Excavations at Sissi, IV. Preliminary Report on the 2015-2016 Campaigns (Aegis 13)*, Louvain-la-Neuve: Presses universitaires de Louvain, 2018.

[198] J. Pakkanen, M. C. Lentini, A. Sarris, E. Tikkala, and M. Manataki, “Recording and Reconstructing the Sacred Landscapes of Sicilian Naxos,” *Open Archaeol.*, vol. 5, no. 1, pp. 416–433, Nov. 2019, doi: 10.1515/opar-2019-0026.

[199] “Salamis Urban Landscape Project 2016–2020 | FIA.” http://www.finninstitute.gr/arkeologinen-yhteistyoprojekti-salamiin-saaren-ambelakiassa-2016-2020/ (accessed Sep. 21, 2020).

[200] A. Sarris *et al.*, “Uphill and downhill geophysical challenges in Delphi, Greece,” in *Archaeologia Polona*, Warsaw, Poland, 2015, vol. 53, pp. 343–347. Accessed: Sep. 21, 2020. [Online]. Available: https://www.researchgate.net/publication/317066409_Uphill_and_downhill_geophysical_challenges_in_Delphi_Greece

[201] S. Chatzitoulousis *et al.*, “Records and Transformations of Memories in the Cultural Landscape of Idomeni (Kilkis, Northern Greece),” *Open Archaeol.*, vol. 5, no. 1, pp. 563–585, Dec. 2019, doi: 10.1515/opar-2019-0035.

[202] A. Sarris, N. Papadopoulos, S. Déderix, M. C. Salvi, and E. Monahan, “Geophysical Mapping of the Prehistoric Settlement at Palamari of Skyros,” 2012.

[203] A. Sarris *et al.*, “Revealing the ancient city of Sikyon through the application of integrated geophysical approaches and 3D modelling,” *ArcheoSciences Rev. Archéom.*, no. 33 (suppl.), Art. no. 33 (suppl.), Oct. 2009, doi: 10.4000/archeosciences.1468.

[204] Y. A. Lolos *et al.*, “Surveying the Sikyonian plateau: integrated approach to the study of an ancient cityscape,” in *Proceedings of the 5th Congress of the Greek Archaeometric Society, Athens*, 2012, pp. 305–326. Accessed: Sep. 21, 2020. [Online]. Available: https://www.academia.edu/36028879/SURVEYING_THE_SIKYONIAN_PLATEAU_INTEGRATED_APPROACH_TO_THE_STUDY_OF_AN_ANCIENT_CITYSCAPE

[205] N. G. Papadopoulos, A. Sarris, and C. Giapitsoglou, “Mapping the buried archaeological remains in the area of the old Turkish School of Rethymno (Crete, Greece) through the application of surface ERT and GPR techniques,” in *Πρακτικα 5ού Σύμποσιού Ελλήνικής Αρχαιομετρικής Εταιρειας, Επιστ. Επιμελεια Ν. Ζαχαριας, Μ. Γεώργακοπούλού, Κ. Πολύκρετή, Γ. Φακορελλής, Θ. Βακούλής*, Αθήνα, 2012, pp. 101–116.

[206] “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges.” http://yann.lecun.com/exdb/mnist/ (accessed Sep. 26, 2020).

[207] “CIFAR-10 and CIFAR-100 datasets.” http://www.cs.utoronto.ca/%7Ekriz/cifar.html (accessed Sep. 26, 2020).

[208] J. Zhang *et al.*, “Why ADAM beats SGD for attention models,” *ArXiv Prepr. ArXiv191203194*, 2019.

[209] N. S. Keskar and R. Socher, “Improving generalization performance by switching from adam to sgd,” *ArXiv Prepr. ArXiv171207628*, 2017.

[210] F. He, T. Liu, and D. Tao, “Control batch size and learning rate to generalize well: Theoretical and empirical evidence,” in *Advances in Neural Information Processing Systems*, 2019, pp. 1143–1152.

APPENDIX A: CASE STUDIES & PROCESSING SCRIPTS

A.1 NOGGIN GPR HEADER FILES

Table A.1: Example of header files (.HD) acquired with the NOGGIN system showing the stored information. On the left is the header file of the survey line acquired in Sissi, Crete, while on the right is a survey line acquired in Demetrias, Thessaly. The header of the left was acquired with an upgraded antenna and DVL firmware. These two headers are representative of the data used for this research. If the zig-zag mode is followed, the value of the step size used is negative for opposite orientated lines.

1234	1234
Data Collected with Noggin Gold	Data Collected with Noggin Plus
2017-06-27	2014/03/16
NUMBER OF TRACES = 393	NUMBER OF TRACES = 2381
NUMBER OF PTS/TRC = 222	NUMBER OF PTS/TRC = 185
TIMEZERO AT POINT = 22	TIMEZERO AT POINT = 37
TOTAL TIME WINDOW = 88.800	TOTAL TIME WINDOW = 74
STARTING POSITION = 0.0000	STARTING POSITION = 0.0000
FINAL POSITION = 19.6000	FINAL POSITION = 119.0000
STEP SIZE USED = 0.0500	STEP SIZE USED = 0.0500
POSITION UNITS = m	POSITION UNITS = m
NOMINAL FREQUENCY = 250.00	NOMINAL FREQUENCY = 250.00
ANTENNA SEPARATION = 0.2500	ANTENNA SEPARATION = 0.2794
PULSER VOLTAGE (V) = 165	PULSER VOLTAGE (V) = 100
NUMBER OF STACKS = 8	NUMBER OF STACKS = 8
SURVEY MODE = Reflection	SURVEY MODE = Reflection
ODOMETER CAL (t/m) = 1132.550049	ODOMETER CAL (t/m) = 1144.300049
STACKING TYPE = F1, P8, DynaQ OFF	STACKING TYPE = F1
Site: Sissi	Site: Demetrias

A.2 MATLAB SCRIPTS USED FOR PROCESSING

A.2.1 Import Noggin data

Script A.1:MATLAB script to import NOGGIN GPR data that have been stored in DVL type III.

```
clear all; clc;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% A script made to import NOGGIN DATA using DVL type III.

% Outputs:
% Bscan: The GPR profile if a single line is selected (2D array)
% Lines: Cell array of Bscans if multiple lines are selected
% K: is the total number of the selected lines.
% x: scan axis vector (in selected distance unit)
% dt:sampling in time (ns)
% t: double travel time vector.

% Variables name assinged to selected Header information
% ntrace ---------> number of traces
% nsample --------> number of points per trace
% t0 -------------> Timezero at point (in ns)
% dx -------------> Step size used (i.e traces sampling)
%                                  (in selected position units)
% ant_sep --------> Antenna Separation (in selected position units)
% time_window ----> Total time window (in ns)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[filename_header, pathname_header] = uigetfile({'*.HD'},'Select NOGGIN header files
(.HD)', 'MultiSelect', 'on');
[filename_data, pathname_data] = uigetfile({'*.DT1'}, 'Select NOGGIN data files
(.HD)', 'MultiSelect', 'on');

pathname_header = fullfile(pathname_header, filename_header);
pathname_data = fullfile(pathname_data, filename_data);
tf=iscell(pathname_data);

if tf==1 %multiple GPR lines - Survey Grid
    K=length(pathname_data);
for k=1:K
        fileID{k} = fopen(pathname_header{k},'rt');
for i=1:39
                header_data{k,i} = fgetl(fileID{k});
end
        ntrace{k}=str2num(header_data{k,7}(22:length(header_data{k,7})));
        nsample{k}=str2num(header_data{k,9}(22:length(header_data{k,9})));
        dx{k}=str2num(header_data{k,19}(22:length(header_data{k,19})));
        antenna{k}=str2num(header_data{k,23}(22:length(header_data{k,23})));
        ant_sep{k}=str2num(header_data{k,13}(22:length(header_data{k,13})));
        time_window{k}= str2num(header_data{k,13}(22:length(header_data{k,13})));
        t0{k}=str2num(header_data{k,11}(22:length(header_data{k,11})));

%import the data
        fileID2{k} = fopen(pathname_data{k}, 'r');
        data{k} = fread(fileID2{k},inf,'int16');
        fclose(fileID2{k});

% Create the 2D Bscan
        [m,n] = size(data{k});
        L=m/ntrace{k};
        e=reshape(data{k},(L),ntrace{k});
        [m2,n2]=size(e);
        data1{k}=e((1):m2,:);
        [m,n]=size(data1{k});
        Lines{k}=data1{k}(60+t0{k}+1:m,:);

% Recalculate ntrace and nsamples
        ntrace{k}=size(Lines{k},2);
        nsamples{k}=size(Lines{k},1);

% Scan axis vector x, time sampling dt, and double travel time
% vector t
        x{k}=0:dx{k}:ntrace{k}*dx{k}-dx{k};
        dt{k}=time_window{k}/nsamples{k};
        t{k}=0:dt{k}:time_window{k}-dt{k};
end
else %Single GPR line
    fileID = fopen(pathname_header,'rt');
for i=1:39
    header_data{i} = fgetl(fileID);
end
    ntrace=str2num(header_data{7}(22:length(header_data{7})));
    nsample=str2num(header_data{9}(22:length(header_data{9})));
    dx = str2num(header_data{19}(22:length(header_data{19})));
    antenna=str2num(header_data{23}(22:length(header_data{23})));
    ant_sep = str2num(header_data{25}(22:length(header_data{25})));
    time_window = str2num(header_data{13}(22:length(header_data{13})));
    t0=str2num(header_data{11}(22:length(header_data{11})));

    fileID2 = fopen(pathname_data, 'r');
    data = fread(fileID2,inf,'int16');
    fclose(fileID2);
    [m,n] = size(data);
    L=m/ntrace;
    e=reshape(data,(L),ntrace);
    [m2,n2]=size(e);
    data1=e((1):m2,:);
    [m,n]=size(data1);
    Bscan=data1(60+t0+1:m,:);

    nsamples=size(Bscan,1);
    x=0:dx:ntrace*dx-dx;
    dt=time_window/nsamples;
    t=0:dt:time_window-dt;
end
```

A.2.2 Processing example with traces resampling and zig-zag mode and scan-axis in the Y direction.

The following scripts were used to process the data acquired from a survey grid at Demetrias archaeological site, Thessaly. The data were acquired using the zig-zag mode.

Script A.2: Load RAW lines and survey parameters saved in .mat file using a navigation window.

```
a=(uigetfile({'*.mat'},'Select GPR RAW mat file to plot(.mat)'));
load(a,'Lines','filename_data','ntrace','x','t','dt','dx','nsamples');

Lines_raw=Lines;
K=length(Lines_raw);
yr=x;
twindow=dt{1}*nsamples{1};
dy=dx;
```

Script A.3: Processing Step 1 - Script that implements trace resampling along the scan axis Y.

```
input=Lines_raw;
t_samples=size(input{1,1},1);
Y=120; %Set length of the survey grid along the scan axis Y in m
y_samples=round(Y/dy{1,1});
targetSize = [t_samples y_samples];
Lines_repos=input;
for k=1:K
    Lines_repos{k}=imresize(input{k},targetSize);
    y{k}=0:dy{k}:dy{k}*targetSize(2)-dy{k};
end
```

Script A.4: Processing Step 2 – Script that implements time zero correction.

```
%Plot raw Bscans to select time zero
nl=1:20; % set Bscans to plot
b=0.4; % set amplifier for Bscan contrast
for k=nl
        name{k}=strcat(filename_data{k}(1:end-4),'_RAW');
        name_fig{k}=strcat('Demetriada Stadium_a 250MHz',{' '},filename_data{k}(1:end-
4),{' '},'Raw');
        h{k}=figure('visible','on','Position',[1 1 1920 1080]);
        imagesc(yr{k},t{k},Lines_raw{k}), colorbar; colormap bone;
        set(gca,
'YDir','reverse','XaxisLocation','top','fontsize',16,'fontweight','bold');
        set(gcf, 'Color', 'w')
        Ctmp=max(max(abs(Lines_raw{k})));
        caxis([-Ctmp*b Ctmp.*b])
        xlabel('DISTANCE (m)','fontsize',16,'fontweight','bold')
        ylabel('TIME (ns)','fontsize',16,'fontweight','bold')
        title(name_fig{k},'fontsize',16,'Interpreter', 'none')
end

t0=6; % set index of time zero
for k=1:K
    Lines_tz{k}=Lines_repos{k}(t0:end,:);
    tz{k}=0:dt{k}:length(t{k}(t0:end))*dt{k}-dt{k};
end
```

Script A.0.5: Processing Step 3- Application of the Dewow function by matGPR

```
for k=1:K
    Lines_dewow{k}=dewow(Lines_tz{k});
end
```

Script A.6:Processing Step 4 - Application of the Inverse amplitude decay function by matGPR

```
%%STEP 5: GAIN - Inverse Amplitude Decay
for k=1:K
    Lines_gain{1,k}=gaininvdecay(Lines_dewow{1,k},tz{1,k});
end
```

Script A.7: Processing Step 5 – Application of Average Background Removal function by matGPR

```
%%STEP 6. Average Background Removal
for k=1:K
    Lines_bnr{k}=rmbackgr(Lines_gain{k});
end
```

Script A.8:Processing Step 6 – Calculation of the absolute spectrums and application of bandpass filtering function by SeismicLab.

```
%Absolute Spectrum
[M,N]=size(Lines_bnr{1});
for k=1:K
for n=1:N

[BNR_Spectr{k}(:,n),fw(:,n),BNR_CSpectr{k}(:,n),f(:,n)]=estimatefw(Lines_bnr{k}(:,n),d
t{k});
end
end

%% Bandpass filtering
f1=100e+6;% frequencies in hertz
```

```
f2=200e+6;
f3=400e+6;
f4=500e+6;

for k=1:K
    k
    [Lines_bp{k},Transfer{k}] =  bp_filter(Lines_bnr{k},dt{k}*1e-9,f1,f2,f3,f4);
end
```

Script A.9: Computation of the Instantaneous Envelope in DB using the function dBInstantaneousEnvelope by GPR-Pro and creation of the 3D volume to extract slices.

```
for k=1:K
V(:,:,k)=dBInstantaneousEnvelope(Lines_bp{k});
end
```

Script A.10: Script to correct the orientation of the even indexed lines when using zig-zag mode.

```
for k=2:2:K-1
V(:,:,k)=fliplr(V(:,:,k));
end
```

Script A.11: Instantaneous envelope calculation and volume creation that detects and reverses the lines with negative sampling step.

```
for k=1:K
V(:,:,k)=dBInstantaneousEnvelope(Lines_bp{k});
if dy{k}<0
V(:,:,k)=fliplr(V(:,:,k));
end
end
```

Script A.12: Instantaneous envelope calculation, nan-padding, and volume creation for non-rectangular grids that works for both parallel and zig-zag mode.

```
for k=1:K
    k
    Lines_DB{k} =  dBInstantaneousEnvelope(Lines_bp{k});
end

%Find maximum Y
for k=1:K
    [M, N]=size(Lines_DB{k});
    maxY(k)=N;
end
Ymx=max(maxY);
y_max=0:dy{1}:dy{1}*Ymx-dy{1};

for k=1:K
    k
if dy{k}>0
        [M, N]=size(Lines_DB{k});
        Lines_DB_pad{k}=padarray(Lines_DB{k},[0 Ymx-N], NaN, 'post' );
V(:,:,k)=Lines_DB_pad{k};
elseif dy{k}<0
```

```
        [M, N]=size(Lines_DB_bp{k});
        Lines_DB_pad{k}=padarray(Lines_DB{k},[0 Ymx-N], NaN, 'pre' );
V(:,:,k)=fliplr(Lines_DB_bp_pad{k});
end
end
```

Script A.13: Producing and saving C-scans (depth slice) from a GPR volume created from the previous steps. The script uses MATLAB CPU parallel processing.

```
% Variables used from previous steps: V, dt, dy, tz, nsamples, y, Y, K

% Setting the variables used for ploting and saving C-scans
lp=0.5; % line space (m)
x=[];
x=0:lp:(K*lp)-lp;
X=lp*(K-1);
y=y{1};
vel=0.1; %velocity estimation of EM wave (m/ns)
T=(size(tz{1},2))*dt{1};
D=(T*vel)/2; %Depth estimation
dd=D/nsamples{1}; %depth sampling estimation
dpth=0:dd:D-dd;
dpth_si=round(dpth,3);

volume_name=strcat('Site1_Grid1');%the volume name of choice
tsi_vec=round(tz{1},2);
for m=1:M
    tsi_tmp{m}=strcat(mat2str(tsi_vec(m)));
    tsi_chr{m}=strrep(tsi_tmp{m},'.','_');
    slicename{m}=char(strcat(volume_name,{'_'},tsi_chr{m},{'ns'}));
    slicename_wo{m}=char(strcat(volume_name,{'_'},tsi_chr{m},{'ns'},{'_'},{'wo'}));
end

% Plot and save C-scans
c1=1.6;  %scale for colormap min
c2=0.97; %scale for colormap max
M=1:134; %sampling of depth slices

V1=(permute(V,[2 3 1]));
parfor m=M
        name_fig{m}=strcat({'Depth='},{' '},mat2str(dpth_si(m)),{'m'},{'
'},{'t='},mat2str(tsi_vec(m)),{'ns'});

        h{m}=figure('visible','off','Position',[1 1 1920 1080]); %use to plot image on
screen resolution
        pcolor(x,y{1},V1(:,:,m)), colorbar; colormap gray;
colormap(flipud(colormap));shading interp
        set(gca,'fontsize',18,'fontweight','bold','LooseInset', get(gca,
'TightInset'));
        set(gcf, 'Color', 'w')
        Cmin=min(min(V1(:,:,m))); Cmax=max(max(V1(:,:,m)));
        caxis([(Cmin*c1) (Cmax*c2)])
        pbaspect([X Y 1])
        xlabel('DISTANCE (m)','fontsize',18,'fontweight','bold')
        ylabel('DISTANCE (m)','fontsize',18,'fontweight','bold')
        title(name_fig{m},'fontsize',18,'Interpreter', 'none');
        saveas(h{m},slicename{m},'jpeg')
%         close(h{m})
end
```

```
poolobj = gcp('nocreate');
delete(poolobj);
```

Script A.14: MATLAB function, `slidewcrop_y`, that was made to save as images subregions in selected C-scans using the pseudocolor plot function. The function applies to survey grids that use the Y direction as the scan axis.

```
function
slidewcrop_y(path_v,V,A,a,b,wx,wy,stridex,stridey,dpth_si,tsi_vec,dy,lp,dt,volume_name
)

%slidewcrop_y crops and saves the selected depth slices using sliding windows.
%   Detailed explanation goes here
[N, K, M]=size(V);
dy=abs(dy{1});dx=lp;dt=(dt{1});
y=0:dy:N*dy-dy; y=round(y,4);
Y=dy*N;
x=0:dx:K*dx-dx; x=round(x,4);
X=dx*K;

dest_dir1=char(strcat(path_v,num2str(wx),{'x'},num2str(wy),{'\'}));
mkdir(dest_dir1)

for i=1:X-1
    Xc(i,:)=(i-1)*stridex:dx:wx+(i-1)*stridex;
    [tfx,IDX(i,:)]=ismember(Xc(i,:),x);
end
IDX(any(~IDX,2),:)=[]; % nulls zero values if any

if stridex==0
    IDX=unique(IDX,'rows');
end

for j=1:Y-1
    Yc(j,:)=round((j-1)*stridey:dy:wy+(j-1)*stridey,4);
    [tfy,IDY(j,:)]=ismember(Yc(j,:),y);
end
IDY(any(~IDY,2),:)=[]; % nulls zero values if any

if stridey==0
    IDY=unique(IDY,'rows');
end

nwX=size(IDX,1); %length for
nwY=size(IDY,1);

for m=A
    tic
    name_fig{m}=strcat({'Depth='},{' '},mat2str(dpth_si(m)),{'m'},{'
'},{'t='},mat2str(tsi_vec(m)),{'ns'});
    tsi_tmp{m}=strcat(mat2str(tsi_vec(m)));
    tsi_chr{m}=strrep(tsi_tmp{m},'.','_');
    dest_dir2=char(strcat(dest_dir1,num2str(m),{'_'},tsi_chr{m}));
    mkdir(dest_dir2)
for i=1:nwX
for j=1:nwY
            m
            i
            j
            pos_name=sprintf('Xstr_%d_Ystr_%d',i,j);
            slicename{m}=char(strcat(volume_name,{'_'},tsi_chr{m},{'ns_'},pos_name));
            h{m}=figure('visible','off','Position',[1 1 1920 1080]);
            pcolor(x(IDX(i,:)),y(IDY(j,:)),V(IDY(j,:),IDX(i,:),m)),colormap gray;
colormap(flipud(colormap));shading interp
            set(gca,'fontsize',18,'fontweight','bold','LooseInset', get(gca,
'TightInset'));
            set(gcf, 'Color', 'w')
            Cmin=min(min(V(:,:,m))); Cmax=max(max(V(:,:,m)));
            caxis([(Cmin*a) (Cmax*b)])
            pbaspect([10 10 1])
            xlabel('DISTANCE (m)','fontsize',18,'fontweight','bold')
            ylabel('DISTANCE (m)','fontsize',18,'fontweight','bold')
            title(name_fig{m},'fontsize',18,'Interpreter', 'none');
            saveas(h{m},fullfile(dest_dir2,slicename{m}),'jpeg')
            close(h{m})
end
end
    toc
end

end
```

Script A.15: Example of applying slidewcrop_y. Windows of 10x10 and 20x20 are used for the selected C-scans in A. The windows are overlapping using a stride of 2m along Y and X directions.

```
clear variables; close all; clc
v='name_of_processed_volume.mat';
path_v='C:\path_to_volume_v \';
load(v,'V1','dpth_si','tsi_vec','dy','lp','dt','vel','volume_name');

V=V1;
A=[10, 27, 33, 37, 42, 50, 72, 150, 176]; %indices of selected C-scans
a=1.7; %scaler Cmin
b=0.98; %scale Cmax

wx=10;
wy=10;
stridex=2;
stridey=2;
slidewcrop_y(path_v,V,A,a,b,wx,wy,stridex,stridey,dpth_si,tsi_vec,dy,lp,dt,volume_name
)

clear wx wy stridex stridey
wx=20;
wy=20;
stridex=2;
stridey=2;
slidewcrop_y(path_v,V,A,a,b,wx,wy,stridex,stridey,dpth_si,tsi_vec,dy,lp,dt,volume_name
)
```

APPENDIX B: CNN IMPLEMENTATION& TRAINING

B.1 AlexNet implementations

Script B.1: Implementation of AlexNet baseline model in python.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

H=227
W=227

initializer = tf.keras.initializers.RandomNormal(mean=0, stddev=0.01)

inputs = keras.Input(shape=(H, W,3), name='input_layer')

x = layers.Conv2D(filters=96, kernel_size=11, strides=4, padding='valid',
                  kernel_initializer=initializer,
bias_initializer='zeros')(inputs)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=256, kernel_size=5, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer,
bias_initializer='zeros')(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=256, kernel_size=5, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Flatten()(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Activation('relu')(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Activation('relu')(x)

outputs = layers.Dense(units=3, kernel_initializer=initializer,
bias_initializer='ones')(x)
outputs = layers.Activation('softmax')(outputs)
```

```
model = keras.Model(inputs, outputs, name='AlexNet_baseline')

model.summary()
```

Table B.1: Implementation of AlexNet used as the baseline.

```
Model: "AlexNet_dropout_normalization"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input Layer (InputLayer)     [(None, 227, 227, 3)]     0
_________________________________________________________________
Conv1 (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
Batch Normalization (None, 55, 55, 96)        384
_________________________________________________________________
ReLU (Activation)       (None, 55, 55, 96)        0
_________________________________________________________________
Max Pooling   (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
Conv2 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
Batch Normalization (None, 27, 27, 256)       1024
_________________________________________________________________
ReLU (Activation)       (None, 27, 27, 256)       0
_________________________________________________________________
Max Pooling   (MaxPooling2D) (None, 13, 13, 256)       0
_________________________________________________________________
Conv3 (Conv2D)            (None, 13, 13, 384)       885120
_________________________________________________________________
ReLU (Activation)       (None, 13, 13, 384)       0
_________________________________________________________________
Conv4 (Conv2D)            (None, 13, 13, 384)       1327488
_________________________________________________________________
ReLU (Activation)       (None, 13, 13, 384)       0
_________________________________________________________________
Conv5 (Conv2D)            (None, 13, 13, 256)       2457856
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 256)       1024
_________________________________________________________________
ReLU (Activation)       (None, 13, 13, 256)       0
_________________________________________________________________
Max Pooling   (MaxPooling2D) (None, 6, 6, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
FC1 (Dense)                (None, 4096)               37752832
_________________________________________________________________
Dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
ReLU (Activation)       (None, 4096)              0
_________________________________________________________________
FC2 (Dense)              (None, 4096)               16781312
_________________________________________________________________
Dropout (Dropout)          (None, 4096)              0
_________________________________________________________________
ReLU (Activation)       (None, 4096)              0
_________________________________________________________________
FC3 (Dense)              (None, 3)                  12291
_________________________________________________________________
```

```
Softmax (Activation)      (None, 3)                 0
=================================================================
Total params: 59,868,931
Trainable params: 59,867,715
Non-trainable params: 1,216
```

Script B.2: Python script to import training set, test set, and apply image augmentations transforms on the training dataset.

```
from keras.preprocessing.image importImageDataGenerator

H=227
W=227

pth_tr ='dataset/training/'
pth_ts ='dataset/test/'
batch_size =128

train_datagen =ImageDataGenerator(rescale=1./255,
                             width_shift_range=[-5,5],
                             height_shift_range=[-5,5],
                             zoom_range=0.1,
                             horizontal_flip=True,
                             vertical_flip=True,
                             rotation_range=90,
                             brightness_range=[0.9,1.1])

train_set = train_datagen.flow_from_directory(directory=pth_tr,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              batch_size=batch_size)

test_datagen =ImageDataGenerator(rescale =1./255)

test_set = test_datagen.flow_from_directory(directory=pth_ts,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              batch_size=batch_size)
```

Script B.3: Adding dropout and batch normalization layers in the baseline model architecture.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

H=227
W=227

initializer = tf.keras.initializers.RandomNormal(mean=0, stddev=0.01)

inputs = keras.Input(shape=(H, W,3), name='input_layer')
```

```
x = layers.Conv2D(filters=96, kernel_size=11, strides=4, padding='valid',
                  kernel_initializer=initializer,
bias_initializer='zeros')(inputs)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=256, kernel_size=5, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer,
bias_initializer='zeros')(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=256, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Flatten()(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Dropout(0.5)(x)
x = layers.Activation('relu')(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Dropout(0.5)(x)
x = layers.Activation('relu')(x)

outputs = layers.Dense(units=3, kernel_initializer=initializer,
bias_initializer='ones')(x)
outputs = layers.Activation('softmax')(outputs)

model = keras.Model(inputs, outputs, name='AlexNet_baseline_BN_Dropout')

model.summary()
```

Script B.4: Setting the learning optimizer, compiling, and training the baseline model using early stopping to terminate training when 30 epochs have passed without improvement in validation loss.

```
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9,
nesterov=False)

model.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])

epochs =100

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=30)
csv_logs =
tf.keras.callbacks.CSVLogger('checkpoints/AlexNet_A_SGDm_baseline_training.lo
g', separator=',', append=True)

callbacks_list =[csv_logs, early_stop]

hist = model.fit(
    train_set,
    epochs=epochs,
    verbose=1,
    callbacks=callbacks_list,
    validation_data=test_set)
```

Script B.5: Implementation of splitting the training set into training and validation sets using the batch size of 32 examples, defining the hyper-model with possible learning rates values for SGD, setting the random search parameters, and then run the tuning process.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners importRandomSearch
from keras.preprocessing.image importImageDataGenerator


# Make the train and validation sets by spliting 80-20
pth_tr ='/content/datasetA/dataset/training/'
pth_ts ='/content/datasetA/dataset/test/'

batch_size=32
H =227
W =227
train_datagen =ImageDataGenerator(rescale=1./255,
                             validation_split=0.2)

train_set = train_datagen.flow_from_directory(directory=pth_tr,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              subset='training',
                                              shuffle=True,
                                              batch_size=batch_size)

validation_set = train_datagen.flow_from_directory(directory=pth_tr,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              subset='validation',
                                              shuffle=True,
                                              batch_size=batch_size)

# Define the hypermodel using the model builder function
```

```
def build_model(hp):
# set AlexNet weight initializer
    initializer = tf.keras.initializers.RandomNormal(mean=0, stddev=0.01)
# input layer
    inputs = keras.Input(shape=(H, W,3), name='input_layer')
# 1st convolutional layer
    x = layers.Conv2D(filters=96, kernel_size=11, strides=4, padding='valid',
                  kernel_initializer=initializer,
bias_initializer='zeros')(inputs)
    x = layers.BatchNormalization()(x)# It has Local Response Normalization
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
# 2nd convolutional layer
    x = layers.Conv2D(filters=256, kernel_size=5, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
# 3rd convolutional layer
    x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer,
bias_initializer='zeros')(x)
    x = layers.Activation('relu')(x)
# 4th convolutional layer
    x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
    x = layers.Activation('relu')(x)
# 5th convolutional layer
    x = layers.Conv2D(filters=256, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D(pool_size=3, strides=2)(x)
# Flattening layer
    x = layers.Flatten()(x)
# 1st Fully connected layer
    x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Activation('relu')(x)
# 2nd Fully connected layer

    x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Activation('relu')(x)
# 3rd Fully connected layer aka Output
    outputs = layers.Dense(units=3, kernel_initializer=initializer,
bias_initializer='ones')(x)
    outputs = layers.Activation('softmax')(outputs)

    model = keras.Model(inputs, outputs, name='AlexNet_SGD_tuning')

# Tune the learning rate for the optimizer
    hp_lr = hp.Choice('learning_rate', values =[1e0,1e-1,1e-2,1e-3,1e-
4],default=1e-2)
    opt = tf.keras.optimizers.SGD(learning_rate=hp_lr, momentum=0.9,
nesterov=False)
```

```
    model.compile(loss='categorical_crossentropy', optimizer=opt,
metrics=['accuracy'])

return model

# specify the tuner
tuner =RandomSearch(build_model, objective='val_accuracy', max_trials=5,
                     executions_per_trial=3,
project_name='AlexNet_A_SGD_tlr32')

# run the tunner with early stopping
callbacks=tf.keras.callbacks.EarlyStopping('val_loss', patience=10)

tuner.search(train_set, epochs=100, verbose=0,
validation_data=validation_set,
             callbacks=[callbacks])
```

Script B.6: Full script of implementing AlexNet and training Model B, which returned the best results during the study.

```
# Setup
import numpy as np

import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from keras.preprocessing.image importImageDataGenerator

# Get the data and prepare the datasets
H=227
W=227

pth_tr ='dataset/training/'
pth_ts ='dataset/test/'
batch_size =64

train_datagen =ImageDataGenerator(rescale=1./255)

train_set = train_datagen.flow_from_directory(directory=pth_tr,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              batch_size=batch_size)

test_datagen =ImageDataGenerator(rescale =1./255)
test_set = test_datagen.flow_from_directory(directory=pth_ts,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              batch_size=batch_size)

# AlexNet implementation with BN and dropout
initializer = tf.keras.initializers.RandomNormal(mean=0, stddev=0.001)

inputs = keras.Input(shape=(H, W,3), name='input_layer')
```

```
x = layers.Conv2D(filters=96, kernel_size=11, strides=4, padding='valid',
                  kernel_initializer=initializer,
bias_initializer='zeros')(inputs)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=256, kernel_size=5, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer,
bias_initializer='zeros')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=384, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)

x = layers.Conv2D(filters=256, kernel_size=3, strides=1, padding='same',
                  kernel_initializer=initializer, bias_initializer='ones')(x)
x = layers.BatchNormalization()(x)
x = layers.Activation('relu')(x)
x = layers.MaxPooling2D(pool_size=3, strides=2)(x)

x = layers.Flatten()(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Dropout(0.5)(x)
x = layers.Activation('relu')(x)

x = layers.Dense(units=4096, kernel_initializer=initializer,
bias_initializer='ones')(x)
x = layers.Dropout(0.5)(x)
x = layers.Activation('relu')(x)

outputs = layers.Dense(units=3, kernel_initializer=initializer,
bias_initializer='ones')(x)
outputs = layers.Activation('softmax')(outputs)

model = keras.Model(inputs, outputs, name='AlexNet_ModelB')

#Model Overview
model.summary()

keras.utils.plot_model(model,"AlexNet_ModelB_simple.png")

keras.utils.plot_model(model,"AlexNet_ModelB.png", show_shapes=True)

# setting optimizer - compile
```

```
opt = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9,
nesterov=False)

model.compile(loss='categorical_crossentropy',
              optimizer=opt, metrics=['accuracy'])

# Train - Fit the model
epochs =30

early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
csv_logs = tf.keras.callbacks.CSVLogger('checkpoints\AlexNet_ModelB.log',
separator=',', append=True)
pth_checkp ='checkpoints/AlexNet_ModelB_{epoch:02d}_{val_accuracy:.2f}.hdf5'

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=pth_checkp, monitor='val_accuracy', verbose=1,
    save_best_only=True, mode='auto', save_freq='epoch')

 callbacks_list =[checkpoint_callback, csv_logs]

hist = model.fit(
    train_set,
    epochs=epochs,
    verbose=1,
    callbacks=callbacks_list,
    validation_data=test_set)

# evaluate the model
scores = model.evaluate(test_set, verbose=0)
print("%s: %.2f%%"%(model.metrics_names[1], scores[1]*100))
# save model and architecture to single file
model.save('AlexNet_ModelB.h5')
print("Model saved in disk")


#%%
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.title('AlexNet_ModelB')
plt.plot(hist.history['loss'], label='loss')
plt.plot(hist.history['val_loss'], label='val_loss')
plt.legend()
plt.show()

# Plot accuracy per iteration
plt.title('AlexNet_ModelB')
plt.plot(hist.history['accuracy'], label='accuracy')
plt.plot(hist.history['val_accuracy'], label='val_accuracy')
plt.legend()
plt.show()
#%%
# cell to plot confusion matrix
from sklearn.metrics import confusion_matrix
import itertools

test_datagen =ImageDataGenerator(rescale =1./255)
test_set = test_datagen.flow_from_directory(directory=pth_ts,
                                              target_size=(H,W),
                                              class_mode='categorical',
                                              batch_size=batch_size,
                                              shuffle=False)

Im_predict= model.predict(test_set, verbose=1, callbacks=callbacks_list)
Im_test= np.argmax(Im_predict, axis=1)

cm = confusion_matrix(test_set.classes,Im_test)
print(cm)

def plot_confusion_matrix(cm, classes,
                        normalize=False,
                        title='Confusion matrix',
                        cmap=plt.cm.Blues):
"""
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

if normalize:
        cm = cm.astype('float')/ cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
else:
print('Confusion matrix, without normalization')

print(cm)

    thresh = cm.max()/2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white"if cm[i, j]> thresh else"black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


cm_plot_labels =['Anomaly : 0','Noise : 1','Structure : 2']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion
Matrix')

from sklearn.metrics import classification_report

class_report=classification_report(test_set.classes,Im_test,
target_names=cm_plot_labels)

print(classification_report(test_set.classes,Im_test,
target_names=cm_plot_labels))
```

Script B.7: Script used to load and evaluate a saved model using the evaluation set.

```
# Load model and calculate the confusion matrix
import numpy as np
import tensorflow as tf
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image importImageDataGenerator
from tensorflow.keras.models import load_model
import matplotlib.pyplot as plt

model = load_model('AlexNet_ModelB_best.hdf5')
model.summary()

H=227
W=227

pth_ts ='dataset/'
batch_size =1

test_datagen =ImageDataGenerator(rescale =1./255)
test_set = test_datagen.flow_from_directory(directory=pth_ts,
                                             target_size=(H,W),
                                             class_mode='categorical',
                                             batch_size=batch_size,
                                             shuffle=False)

score = model.evaluate(test_set, verbose=0)
print("%s: %.2f%%"%(model.metrics_names[1], score[1]*100))

#%% cell to plot the confusion matrix

from sklearn.metrics import confusion_matrix
import itertools


Im_predict= model.predict(test_set, verbose=1)
predictions = np.argmax(Im_predict, axis=1)
predictions100 =Im_predict*100


cm = confusion_matrix(test_set.classes, predictions)
print(cm)

def plot_confusion_matrix(cm, classes,
                        normalize=False,
                        title='Confusion matrix',
                        cmap=plt.cm.Blues):
"""
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

if normalize:
        cm = cm.astype('float')/ cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
else:
print('Confusion matrix, without normalization')

print(cm)

    thresh = cm.max()/2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white"if cm[i, j]> thresh else"black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')


cm_plot_labels =['Anomaly : 0','Noise : 1','Structure : 2']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion
Matrix')

from sklearn.metrics import classification_report

class_report=classification_report(test_set.classes, predictions,
target_names=cm_plot_labels)

print(classification_report(test_set.classes, predictions,
target_names=cm_plot_labels))

# Im_predict2 = np.rint(Im_predict)
# Im_predict3 = np.argmax(Im_predict2, axis=1)
labels=test_set.classes

# Show miscassified examples # to do
missclass_idx = np.where(predictions != labels)[0]
i = np.random.choice(missclass_idx)
#plt.imshow([i])
#plt.title('True label: %s Predicted label: %s' % (labels(Im_test[i]),
labels(Im_predict[i])))
```

Appendix C: Additional Results & Final models

Table C.1: Summary of Model A.

```
Model: "AlexNet_ModelA"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_layer (InputLayer)     [(None, 227, 227, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 55, 55, 96)        34944
_________________________________________________________________
batch_normalization (BatchNo (None, 55, 55, 96)        384
_________________________________________________________________
activation (Activation)      (None, 55, 55, 96)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 27, 27, 256)       614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 256)       1024
_________________________________________________________________
activation_1 (Activation)    (None, 27, 27, 256)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 384)       885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 384)       1536
_________________________________________________________________
activation_2 (Activation)    (None, 13, 13, 384)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 13, 13, 384)       1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 13, 384)       1536
_________________________________________________________________
activation_3 (Activation)    (None, 13, 13, 384)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 256)       884992
_________________________________________________________________
batch_normalization_4 (Batch (None, 13, 13, 256)       1024
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 256)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 4096)              37752832
_________________________________________________________________
activation_5 (Activation)    (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
activation_6 (Activation)    (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 12291
_________________________________________________________________
activation_7 (Activation)    (None, 3)                 0
=================================================================
Total params: 58,299,139
Trainable params: 58,296,387
Non-trainable params: 2,752

_________________________________________________________________
```

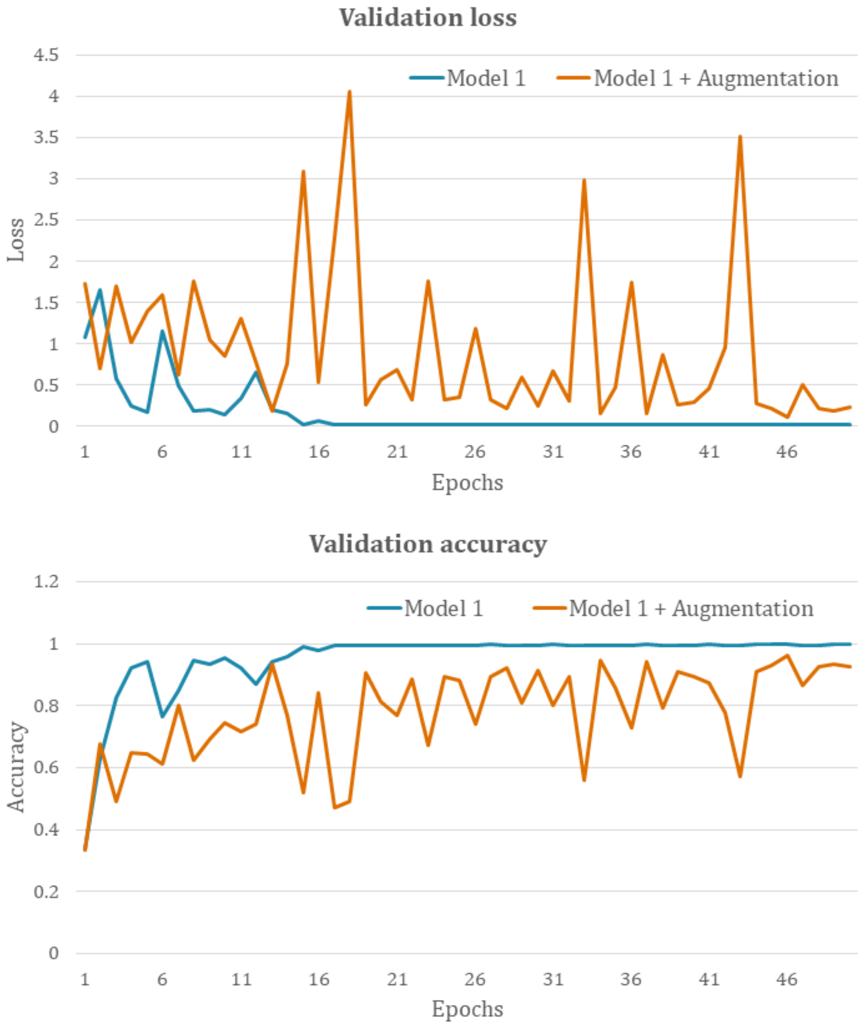


Figure C.1: Learning curves comparing the effect of applying augmentation in Model 1 training setup with SGD, a learning rate of 0.001, and a batch size of 64. The chosen augmentation techniques that were applied have a negative impact on the generalization.

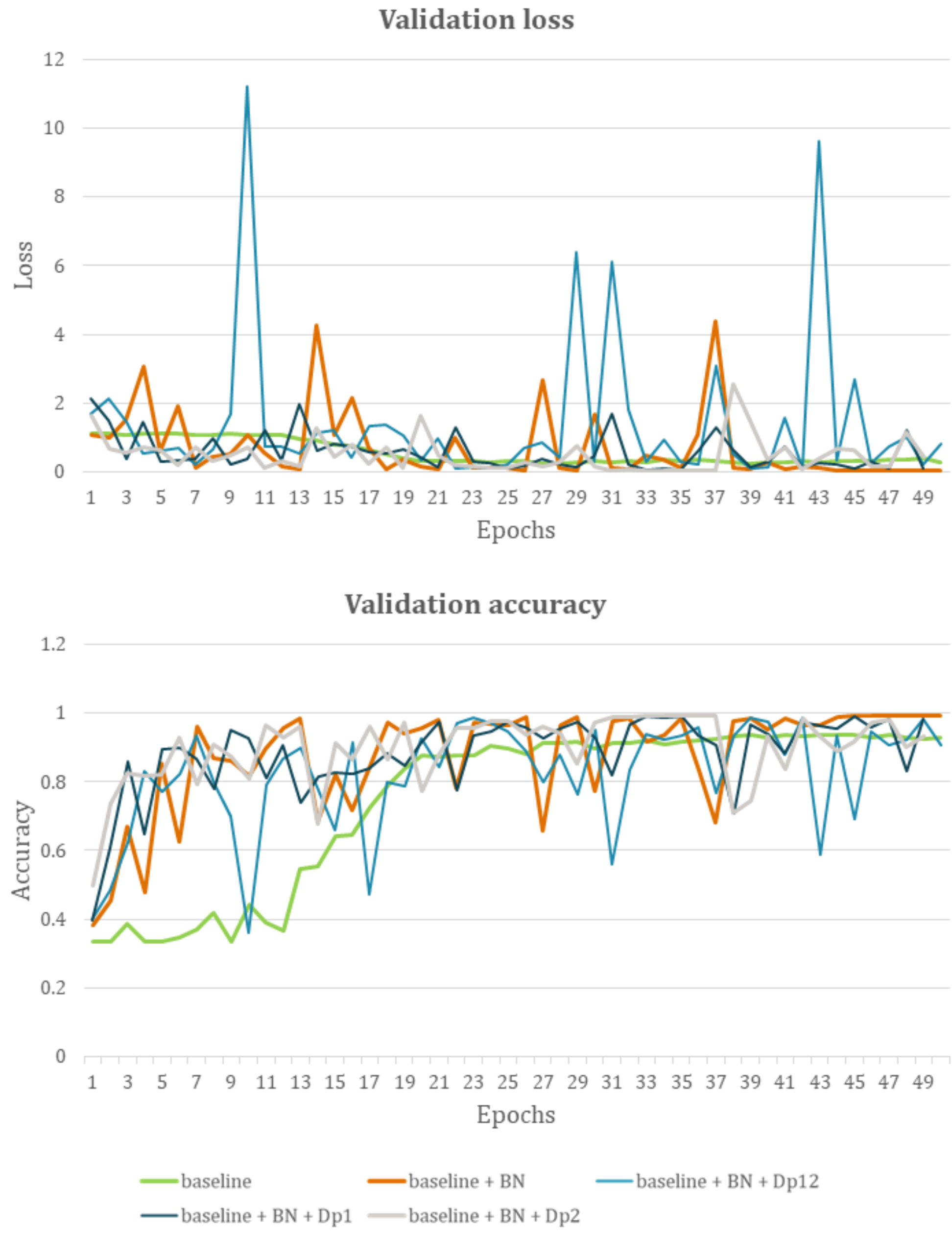


Figure C.2: The effect of applying dropout on the chosen BN setup for the case of dataset-B using Adam optimizer. Dropout was applied to three different setups, but none improves the performance. The baseline with BN in the first three and last convolutional layers gives the best results after 40 epochs (orange line).

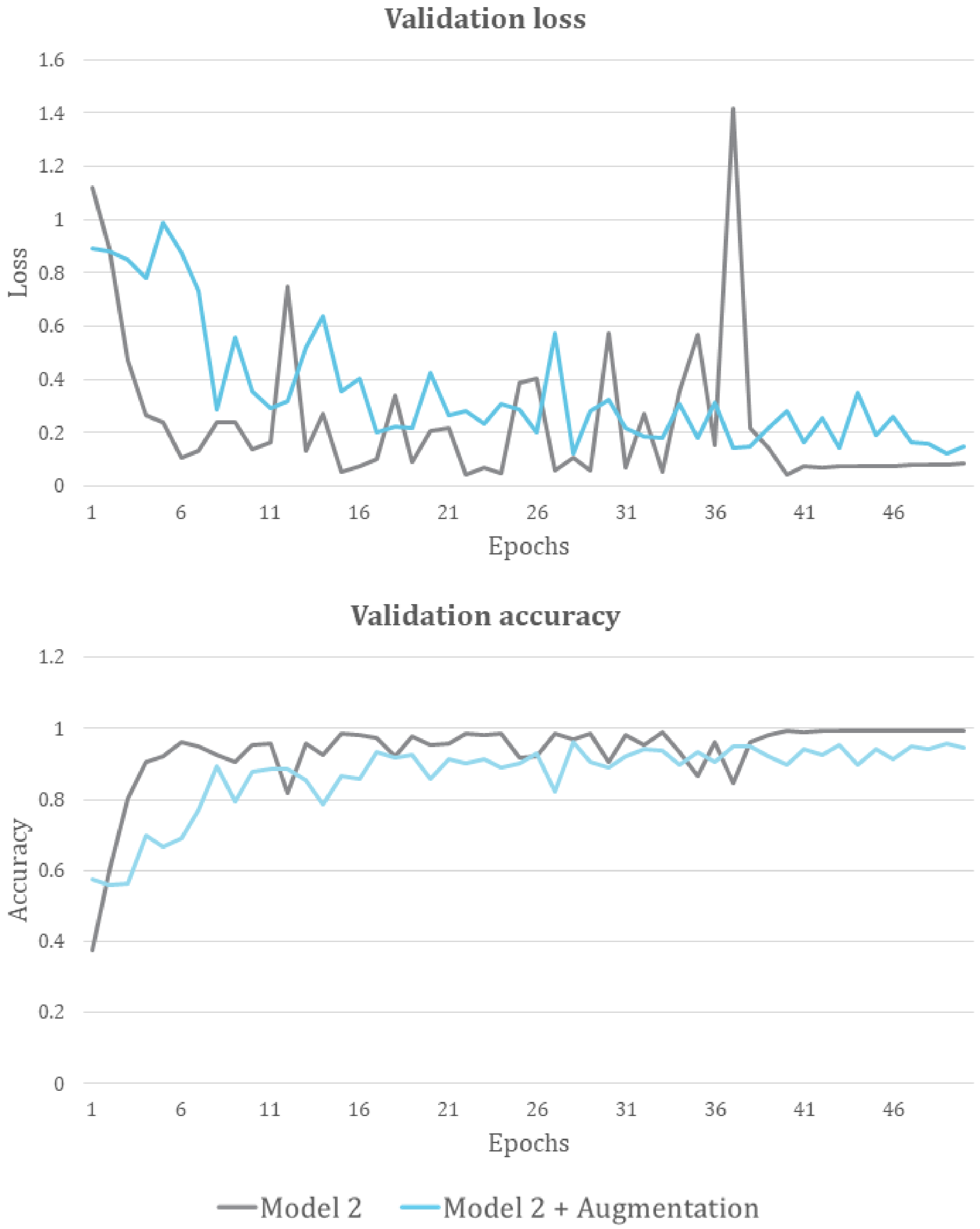


Figure C.3: Comparative chart showing the effect of applying image augmentation techniques using dataset-B and Adam optimizer with a learning rate of 0.001 and a batch size of 32.

Table C.2: Summary of Model B.

```
Model: "AlexNet_ModelB"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
input_layer (InputLayer) [(None, 227, 227, 3)] 0
_________________________________________________________________
conv2d (Conv2D) (None, 55, 55, 96) 34944
_________________________________________________________________
batch_normalization (BatchNo (None, 55, 55, 96) 384
_________________________________________________________________
activation (Activation) (None, 55, 55, 96) 0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96) 0
_________________________________________________________________
conv2d_1 (Conv2D) (None, 27, 27, 256) 614656
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 256) 1024
_________________________________________________________________
activation_1 (Activation) (None, 27, 27, 256) 0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 256) 0
_________________________________________________________________
conv2d_2 (Conv2D) (None, 13, 13, 384) 885120
_________________________________________________________________
batch_normalization_2 (Batch (None, 13, 13, 384) 1536
_________________________________________________________________
activation_2 (Activation) (None, 13, 13, 384) 0
_________________________________________________________________
conv2d_3 (Conv2D) (None, 13, 13, 384) 1327488
_________________________________________________________________
batch_normalization_3 (Batch (None, 13, 13, 384) 1536
_________________________________________________________________
activation_3 (Activation) (None, 13, 13, 384) 0
_________________________________________________________________
conv2d_4 (Conv2D) (None, 13, 13, 256) 884992
_________________________________________________________________
batch_normalization_4 (Batch (None, 13, 13, 256) 1024
_________________________________________________________________
activation_4 (Activation) (None, 13, 13, 256) 0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 256) 0
_________________________________________________________________
flatten (Flatten) (None, 9216) 0
_________________________________________________________________
dense (Dense) (None, 4096) 37752832
_________________________________________________________________
dropout (Dropout) (None, 4096) 0
_________________________________________________________________
activation_5 (Activation) (None, 4096) 0
_________________________________________________________________
dense_1 (Dense) (None, 4096) 16781312
_________________________________________________________________
dropout_1 (Dropout) (None, 4096) 0
_________________________________________________________________
activation_6 (Activation) (None, 4096) 0
_________________________________________________________________
dense_2 (Dense) (None, 3) 12291
_________________________________________________________________
activation_7 (Activation) (None, 3) 0
=================================================================
Total params: 58,299,139
Trainable params: 58,296,387
Non-trainable params: 2,752
_________________________________________________________________
```