Technical University of Crete

Department of Electrical and Computer Engineering

# Graph-Based Modeling of Cellular Hotspot Data Analysis

A Thesis by Konstantinos Zacharopoulos

Examination Committee

Professor Sotiris Ioannidis
Professor Apostolos Dollas
Professor Michalis Zervakis

December 2022

# Abstract

Cellular networks have gone through significant changes in infrastructure during the last few decades. The technological advance along with the massive increase of population keep pushing the mobile network's capabilities to their limits. As the network usage requirements become larger, so do the radio expert's needs for accurate and timely information regarding the status of the cellular network. With that information at hand, they have the ability to foresee and prevent unwanted circumstances, such as network failure due to unmanageable overload. In this thesis, I propose a Neural Network structure which aims to make both swift and precise forecasts of such undesirable events. I make use of well-known Neural Network architectures such as the Graph Neural Network(GNN) and the Recurrent Neural Network(RNN), a combination which allows for monitoring and learning both spatial and temporal patterns that the cellular network may exhibit. In addition, a Graph partition is introduced, which effectively splits the original graph into much smaller and manageable sub-graphs with the idea of further increasing the Neural Networks performance metrics while also significantly scaling down its time complexity. Furthermore, I propose the addition of an Hierarchical model in the original architecture, an addition which nearly maximizes the precision in most use-cases. The proposed architecture succeeds in increasing the precision of its predictions compared to other known implementations. Moreover, it has a steady performance across various sizes of historical data provided to the network and across different targeted prediction horizons.

# Acknowledgements

In this section, I would like express thanks and gratitude certain individuals, First and foremost, special thanks to Ioannis Arapakis for his technical support and guidance throughout the development of this project. Secondly, many thanks to my co-student and co-worker Georgios Koutroumpas with whom we had great collaboration and information exchange in order to achieve our goals. Additional thanks to my thesis supervisors Sotiris Ioannidis and Konstantinos Georgopoulos as well as lab assistant Ioannis Morianos from the Technical University of Crete.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

In recent years, our society has undergone an immense and unforeseen urbanization in nearly every country throughout the globe. In addition, ever since the first mobile network came out in 1983, the global population has seen a nearly 70% increase, meaning that the demand for mobile network usage has significantly gone up. The technological advance in this same time period has led to larger requirements from cellular networks, since the vast majority of the population is using at least one device in need of Internet access on a daily basis. These factors have contributed in the desideratum for better cellular infrastructure, one which could support all the requests made from its users. However, such infrastructure is not easy to maintain. Despite the upgrades that it has undergone, there are still cases in which the circumstances may lead to unwanted events such as network overload leading to user inconvenience or system failure. When a cellular antenna is overloaded, it is often referred to as a "hotspot" by radio experts. In order for the cellular companies to maintain a smooth operation for the mobile network, they require accurate and timely provided information regarding the status of antenna cells.

The status of a cellular network however, may depend on several factors which are not always linked to its design. As explained above, urbanization is one of the factors that may result in repeatedly occurring problems since the original design of the network in that specific location was not meant to support the needs of more people than the ones that it was built for. Another thing worth mentioning is that in our days, tourism activity is constant and most of the times unpredictable. The most common way of monitoring a network's performance in such cases, is by examining the system's Key-Performance-Indicators otherwise known as KPIs. These KPIs provide all the necessary information about an antenna cell's status including active users, bandwidth and many other metrics. They can also be utilized by Machine Learning (ML) algorithms in the attempt to adapt to the patterns that the data

may exhibit and learn to make accurate predictions for these so-called hotspot events.

One of the most efficient ML algorithms which is being applied for the purpose of solving forecasting problems, are the Artificial Neural Networks (ANNs). The reason why ANNs have been getting increased traction in such fields, is that they can absorb large amounts of information, while requiring less storage resources compared to other methods. They have the advantage of being capable to approximate non-linear functions and with the benefit of having hidden layers they can achieve much higher performance metrics. One of the companies which has shown interest in this field is Telefonica, and through project manager Ioannis Arapakis has provided the real-world data (KPIs) from a cellular network and has also introduced two target goals. The initial goal was to design an Neural Network architecture with the attribute of high sensitivity in the attempt to predict the maximum possible performance drops. This task was handled by co-student and co-worker Georgios Koutroumpas. The latter of the two objectives, which will be presented in this thesis, was to create an architecture capable of providing high quality predictions by minimizing the error rate of the classifier.

# 2 Problem Formulation

## 2.1 Thesis Objective and Motivation

In this thesis, I propose the design and development of a Neural Network which is capable of detecting possible hotspot cases of cellular antennas in the near future. The data input used for this project is part of real world data in a multivariate time series format with a time-step of one hour. In this time-series, all antenna cells' KPIs are included, which provide the information that each of the antenna's sensors have collected within that hour, as well as some static information regarding their geo-location and structural design.

The target goal is to create an architecture that will be able to cover various prediction horizons and maximize the precision of the Neural Network. The reason why it is essential to have as few false positive(FP) cases as possible, is due to the fact that every time the model predicts a hotspot case, a technician will be assigned to verify the result and take preemptive measures in the attempt to avoid network overload or failure. If the model provides false predictions, then its services will not only be inefficient in terms of time, since the radio experts will devote their time investigating a false case, but also in terms of bad cellular network services. Mobile networks have been so deeply integrated in almost everyone's daily routines [10], that in case of malfunctions, the problems will not just be about user inconvenience but perhaps an obstruction of professional work. Several organizations as well as certain individuals conduct their business which in many cases is almost entirely contingent on continuous and stable connection to the network.

In a worst case scenario where network downtime cannot be avoided, having an insight of when a system malfunction might occur, enables IT specialists and radio experts to inform their clients about imminent connection disruption, or scheduled maintenance. The purpose of this

project is to provide this option to the tech-teams dealing with such incidents and hopefully contribute in eliminating as many of these cases as possible. In collaboration with Telefonica, which is also the source of the real-world data, the desired output which is hoped to be achieved is having above 0.85% regarding the NNs precision metric.

## 2.2 Challenges

Handling large amounts of data when it comes to Deep Learning Neural Networks can be quite challenging. As mentioned above, the data is in a multivariate time series format, for each of the cellular antennas with an one hour step, meaning there are millions of samples which the NN has to process. This does not only impair the model's capacity of giving accurate results but it also means that it may suffer from quite large time complexity. Another concern regarding the nature of the data is the huge class imbalance that they exhibit, with only 0,25% of the samples containing positive cases(hotspots), and when it comes to antenna statistics, only 9,3% of the antenna's show at least one positive case throughout the dataset. Having this information, one should initially expect facing difficulties creating a pipeline that would be capable of adapting to patterns shown by positive cases. Last but not least, the proposed architecture is required to provide its results within certain time-constraints in order to be efficient in real-world applications.

## 2.3 Related Work

Lately, the topic of traffic or cellular hotspot forecasting has been getting increased traction and there have been several related works on this subject. When dealing with datasets in multi-variate time-series formats and being tasked to predict future states of certain variables, one of the first things that come in mind is the implementation of an RNN. In [11], there is an in-depth analysis on a number of well known Recurrent Neural Network(RNN) architectures, such as the vanilla-RNN, the Long-Short Term Memory (LSTM) and the Gated Recurrent Unit(GRU) regarding their performance on network traffic forecasting.

However, in many cases there might exist not only temporal dependencies in the dataset but also spatial dependencies. In such cases, there is the option to organize the data into a Graph structure which will allow for the implementation a Graph Neural Network in order to investigate these spatial dynamics and how they might affect the model. In [12], a GAT is being used in combination with a Gated Recurrent Unit to learn traffic patterns. Here, the GAT part of the model learns from the topology of the road network through graph convolution with the attention coefficient while the GRU unit learns from past information.

Besides the RNN-type models, there are also other ways of dealing with temporal dynamics. As proposed in [13], Temporal Convolutional Neural Networks (TCNs) present another approach to this problem. They are far superior in terms of training time and have the capability of accepting a much larger receptive field of historical data without causing memory or vanishing/exploding gradient issues. Combined with a Graph Neural Network architecture, they have been used in order to moderate urban traffic as shown in [14], by using Chebyshev Polynomials to track down spatial patterns.

Finally, regarding the research of cellular hotspot forecasting in particular, which is also the work which my model's evaluations and performance will be compared to, is the [15] which implements tree-based models. It takes a multi-variate time-series dataset as its input and makes use of Random Forest Regression trees in order to make short-sighted predictions. However this method disregards any spatial dependencies that the input data may exhibit. This is a drawback also present in [16]. In this work, a Long-Short Term Memory is applied, which is an RNN architecture, effectively learning from the temporal dynamics that exist in the data.

## 2.4 Contribution

My model consists of a Graph Convolutional Neural Network combined with a Recurrent Neural Network. With this structure, I can investigate patterns for each antenna individually in the arrow of time, as well as between the antennas which will be considered neighbors once the Graph structure is defined. In this implementation, I propose a

partition of the initial graph into smaller and more manageable groups, which significantly decreases time complexity and in the meantime it improves the model's capacity of achieving the desirable precision in its predictions.

The idea is that I to separate the cellular antennas into two groups, the first group that does not show any active case throughout the data, and the second one in which each antenna shows at least one active case. The second group is then split into K sub-graphs of the same size, on which a separate instance of the NN is trained. Finally, these K models will go through a process of cross validation, proving that a trained instance of the model is also highly transferable, meaning it can perform well on sub-graphs that are different than the one it was trained on. These classifiers will be defined in this thesis as "weak classifiers". In the end, the implementation of an Hierarchical model is proposed, which changes the nature of the pipeline into ensemble learning and will act as the thesis "strong classifier". The goal of the Hierarchical model is to combine the results of all the weak classifiers like a voting system in the attempt to maximize the model's precision.

The NN architecture and the design of this pipeline have some significant advantages over some of its counterparts which are not only limited to the performance metrics. The implementation of sub-graphs and the transfer-ability of the model make it possible to run the model locally, on targeted geographic areas without having the need to examine the entire graph. Furthermore, big organizations, such as cellular network providers, usually have problems in bypassing the latency caused while communicating information from one single device across the network to a centralized computing system or server. This latency may be small in the scale of milliseconds or much larger reaching several minutes. On top of that, there are some remote locations which can be far away from the central server, where connectivity might be very limited or even non existent from time to time. In such cases, most organizations would prefer edge-computing to cloud-computing, in order to significantly boost the speed of the decision making process of the Machine Learning Algorithms that they are using. The design of this project allows for the usage of very efficient processing hardware devices such as the Field Programmable Gate Arrays(FPGAs), that could notably decrease the time needed for the NN's inference phase.

# 3 Background

## 3.1 Data Imputation

Time series is a sequence of data points organized in a temporal manner. It is a sequence of discrete-time data which are drawn at successive equally spaced points in time and is a very common format for modern day datasets regarding forecasting. When dealing with difficult tasks and highly complex pipelines, it is crucial to ensure high quality of data. One of the main threats to data quality that frequently appears in machine learning applications, is the occurrence of missing values. The reasons why a dataset might include missing values could be attributed to equipment malfunctions, human error, inability of the system to record the measured data at the time, or network failure.

### 3.1.1 Types of Non-Response

There are two main categories of non-response when dealing with missing values, which are very common and in many cases unavoidable for most datasets. Considering a household survey during which the participants are required to fill a questionnaire:

- Item non-response occurs when one of the respondents manages to provide most of the required information but fails to do so for a small part of them. This type of non-response is considered to be the least threatening between the two and is usually treated separately.

- Unit non-response, poses a much larger threat to data quality, as it refers to complete absence of data from one or more households.

### 3.1.2 Techniques

In general, there exist several methods that help prevent the occurrences of missing data, however it is a phenomenon that cannot be fully

Item NonResponse

| Household ID | Question A | Question B | Question C | Question D | Question E |
|---|---|---|---|---|---|
| 193 | Answered | Answered | Answered | Answered | Answered |
| 2918 | Answered | NaN | Answered | Answered | NaN |
| 1290 | NaN | Answered | NaN | Answered | NaN |
| 785 | Answered | Answered | Answered | Answered | Answered |
| 498 | Answered | Answered | Answered | NaN | NaN |
| 1946 | NaN | Answered | Answered | Answered | Answered |

Unit NonResponse

| Household ID | Question A | Question B | Question C | Question D | Question E |
|---|---|---|---|---|---|
| 193 | Answered | Answered | Answered | Answered | Answered |
| 2918 | Answered | Answered | Answered | Answered | Answered |
| 1290 | NaN | NaN | NaN | NaN | NaN |
| 785 | Answered | Answered | Answered | Answered | Answered |
| 498 | Answered | Answered | Answered | Answered | Answered |
| 1946 | NaN | NaN | NaN | NaN | NaN |

**Figure 3.1:** The two different categories of non-response

avoided. Two of the most common ways of dealing with this problem are list-wise and pairwise deletion. On the one hand, list-wise deletion eliminates all samples containing missing values, while on the other hand, pairwise deletion only removes samples if the items missing are required for the analysis. Both cases have a high chance of decreasing the efficiency of the analysis by ruining data continuity, especially if the number of missing values is high and not randomly distributed.

Data imputation refers to the process of substituting missing fields with an estimated value, instead of removing the entire sample. Imputation techniques might be simple and fast, or more sophisticated and slow. Each dataset reacts differently to the various imputation techniques, according to the nature and distribution of their missing values. For the purpose of this thesis, the following algorithms have been implemented and tested:

- Zero Filling : Substitutes missing values with zero.
- Mean / Median / Most-Frequent Imputation : Calculates the mean / median / most-frequent of the recorded values for the target variable and uses it to impute missing ones.

- Hot / Cold deck Imputation : The hot deck imputation algorithm will randomly choose a value from another sample that is similar to the one under examination, but does not have a missing value in the variable under examination. It will then replace the missing value with the randomly selected one. On the other hand, cold deck imputation uses the exact same value for the imputation from the same sample.

- K-Nearest-Neighbors Imputation : This algorithm will search for K number of samples which are considered close to the sample which shows a missing value. Two samples are considered close to each other, if the non-missing features are close. This proximity is usually estimated by calculating the nan-euclidian distance between the sample elements. After finding the K closest samples, the algorithm calculates the mean value of those samples on the same variable as the one which the sample under question has the missing value, and use it for the imputation.

The aforementioned methods are applied in order to tackle the item non-response issues. In order to deal with unit non-response, a different approach will be used, one that estimates the mean values (for all items) between the sample that was lastly observed before the unit non-response and the item that is observed right after. In case of consecutive unit non-response cases, the algorithm will start estimating the innermost missing samples and then work its way to the outermost missing samples.

## 3.2 Feature Importance

Every dataset consists of a number of columns, which are the variables/features that a NN will use to calculate an output. If the NN is a classifier, the term feature importance is used to describe how important a certain feature is to the classification process. Each feature has its own unique contribution to this process which may vary depending on the type and nature of the model. In this thesis, I examine several feature analysis techniques which will allow us to rank the most important features to this classification problem, and perhaps discard the ones that are deemed unnecessary. This elimination process does not only reduce time complexity, but it might also help improve the

accuracy of the classifier. For this task I will investigate three different types of feature selection methods which are described as follows:

- Wrapper Methods : These methods use a ML algorithm (in this case a classification algorithm) to evaluate the importance of the features, each time using a subset. Iteratively, they evaluate the possible combinations of features through a greedy search algorithm and select the combination that yields the optimal results according to the evaluation criterion.
    1. Forward Selection : This algorithm starts with zero features and attempts to fit the model with every feature one at a time. The first feature that will be selected is the one that returns the minimum p-value. In every step, the algorithm will re-fit the model trying combinations of previously selected features with the remaining features. This process is repeated until it reaches a subset of features having an individual p-value that is less than a designated threshold.
    2. Backward Selection : It works in the opposite way that Forward Selection does. In this cases the model starts with all the features and in each step it removes the one with the highest p-value, repeating the process until a subset of features having an individual p-value that is not higher than the designated threshold has been reached.
    3. Step-wise Selection : A combination of Forward and Backward Selection methods. Starting with zero features, the algorithm first adds the feature with the lowest p-value and in the next step it will add the second most important feature into the set. After that, it goes into a loop, during which every time an important feature is added, the algorithm will also check the importance of already selected features and remove those that have an insignificant p-value. This algorithm is an improvement to the Forwards and Backwards Selection methods, where if a feature is added/removed, it will not be re-examined in later iterations. It also takes into account the interactions between features, saying that a feature might have a significant p-value on its own, but this value might change drastically by adding more features into the set.

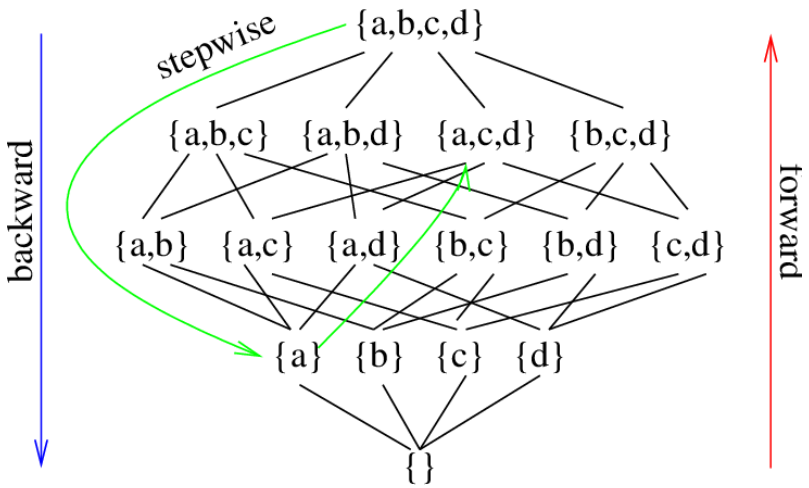


**Figure 3.2:** Forward, Backward and Step-wise selection methods [1].

It is important to note that even though wrapper methods never fail to reach a conclusion and provide a decent performing subset, they are characterized by extremely high time complexities since they have to evaluate on the given model during each iteration. Such techniques are usually not suggested for large datasets, or for datasets with many features.

- Filter Methods : Filter methods use a descriptive measure in order to rank each feature in the dataset, and then select the highest ranking ones for the final subset. They implement various statistical tests in order to find the correlation between every individual feature and the variable noted as ground truth.

    1. Variance Threshold : This method is a very simple and baseline approach for selecting features. The algorithm will eliminate all the features which have a variance that does not meet a specified threshold. This works under the assumption that features with little to zero variance have considerably less useful information to offer than the ones with high variance. Though extremely quick as a method, it is completely blind to relationships between features and is usually never adopted as the only feature selection method in a project, rather it is combined with one of the others.
    2. Correlation Coefficients : One way to check the relationships between features is calculating their correlation coefficients. This method operates under the theory that if two variables are highly correlated with each other, we can use one of

them to predict the other, therefore only one of them will be of significant value to our final model. The first step is to calculate a correlation map and see all the coefficients that the features have between each other as shown in 3.3. Then we define a coefficient threshold, effectively selecting all features that are highly correlated between each other, either with a positive or a negative correlation coefficient. Once we have these pairs, we check the correlations between each feature in a pair and the target variable and remove the feature that will have the largest coefficient. This process is repeated for all the highly correlated pairs that we can find.

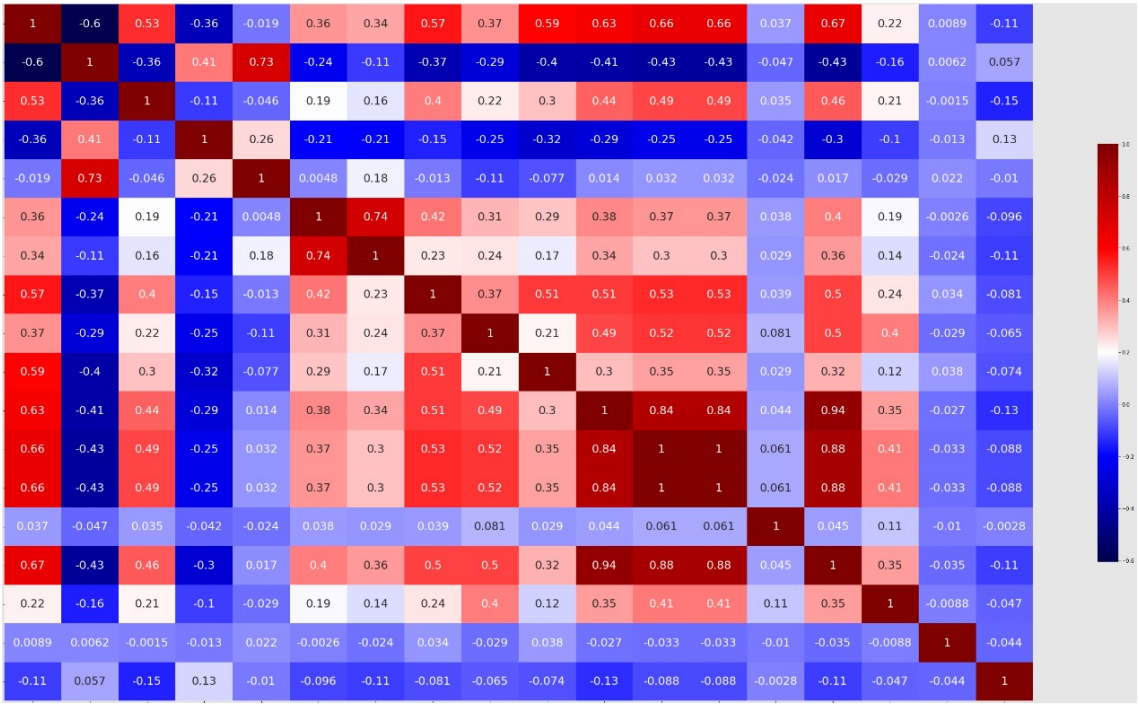


**Figure 3.3:** Correlation Map.

Generally, filter methods are much faster than any wrapper method especially when dealing with large datasets. However, the majority of them are fully blind to feature-to-feature relations and in some cases might even prove to be unsuccessful in reaching their goal.

- Embedded Methods : Embedded methods are known to have the benefits of both wrapper and filter methods and in the meantime minimize their disadvantages. They include feature interactions,

have considerably less computational cost than wrapper methods and are generally much better at reducing over-fitting potential. In these cases, the feature selection algorithm is integrated as part of the learning algorithm , which performs feature selection and in our case, classification, at the same time.

1. Ridge Regression : Ridge regression offers the option of keeping all the features in the final model while penalizing the beta coefficients of the model for being too large by using L2 Regularization penalty. The algorithm ensures that all the features will be kept in the final model by not letting their coefficients be shrank to zero. This technique is usually preferred when the data suffers from multicollinearity, as it scales down the strength of correlation of the variables that are as significant as others for the prediction process. The regression is accomplished by penalizing the betas with a lambda parameter that is tuned with cross-validation.
2. LASSO Regression : LASSO Regression works similarly to the Ridge Regression, but instead uses L1 Regularization penalty and enforces some restrictions on the sum of the model parameter values which effectively shrinks some of the coefficients to zero. All features with coefficients shrank to zero will be removed from the final model, meaning the model complexity will be lowered. LASSO Regression makes use of a lambda operator as well, which again needs to be tuned with cross-validation.

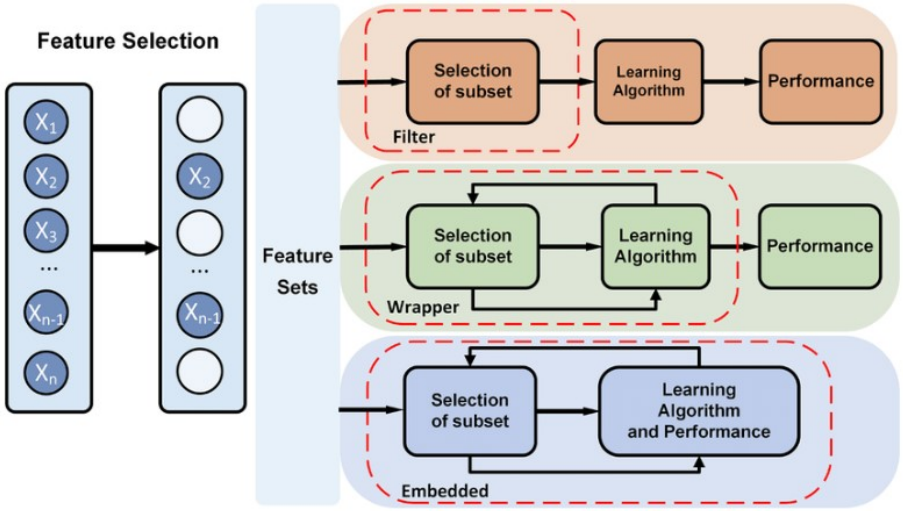


**Figure 3.4:** Process of each feature selection type method, as explained above [2].

## 3.3 Neural Networks

Neural networks, are a category of ML which falls under the category of deep learning algorithms. Their structure is much similar and inspired by biological neural networks, and their learning process mimics the way that human beings learn and adapt to patterns. Their architecture consists of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

The training process of NNs, is based on training, validation and evaluation sets of data. The training set is the one which the NNs use in order to improve their accuracy over time by iterating through a number of training epochs. The validation set is often used in order to fine-tune the model's hyper-parameters in order to maximize accuracy and efficiency, and the evaluation set is where we can apply the trained NN and get our classification, clustering, or regression results depending on what type of NN we are training.

In the beginning of the training process, an input layer is defined and the model's weights are assigned. The higher the weight of a single variable, the more significant it is to the process compared to other variables. Once the weights are assigned, all input features are multiplied by their respective weights and summed.

The output of a layer is determined by an activation function. Essentially, the activation function will "activate" the node if its output exceeds a certain threshold, so that the data may pass to the next layer, becoming the input of another node. There are several different activation functions and their usage depends on the type of problem at hand. Some of the most commonly used functions are presented in Figure 3.5.

The process of passing data from one node to another in further layers, defines the NN as a feed-forward network.

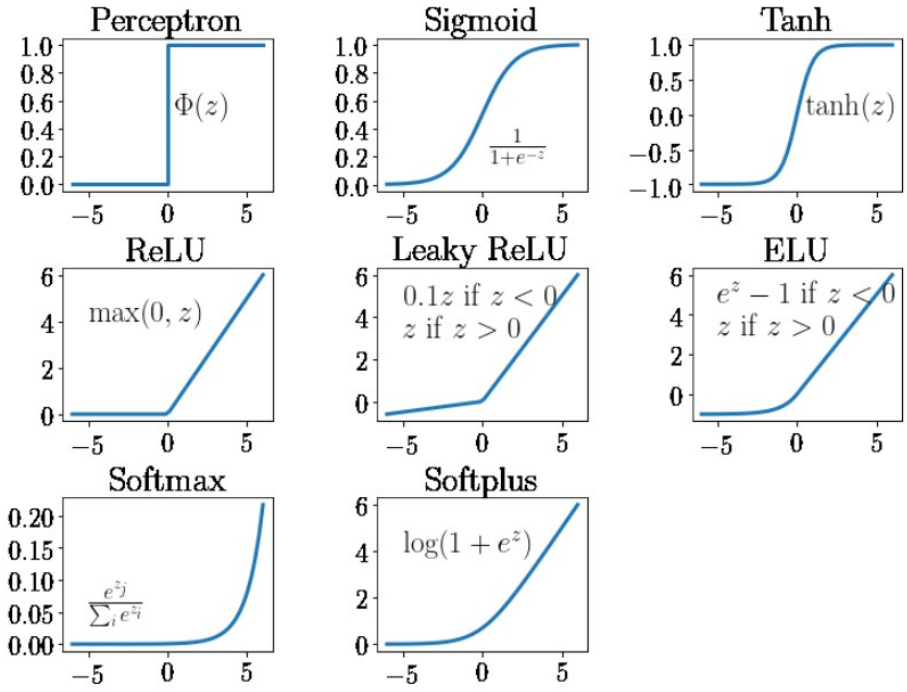


**Figure 3.5:** Well-known Activation Functions [3].

At the end of each training epoch, the model's performance is evaluated through a cost / loss function. This cost function basically tells our model how far away it is from its target and how much more it needs to be improved in order to reach it, thereby the objective is to minimize it. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence,

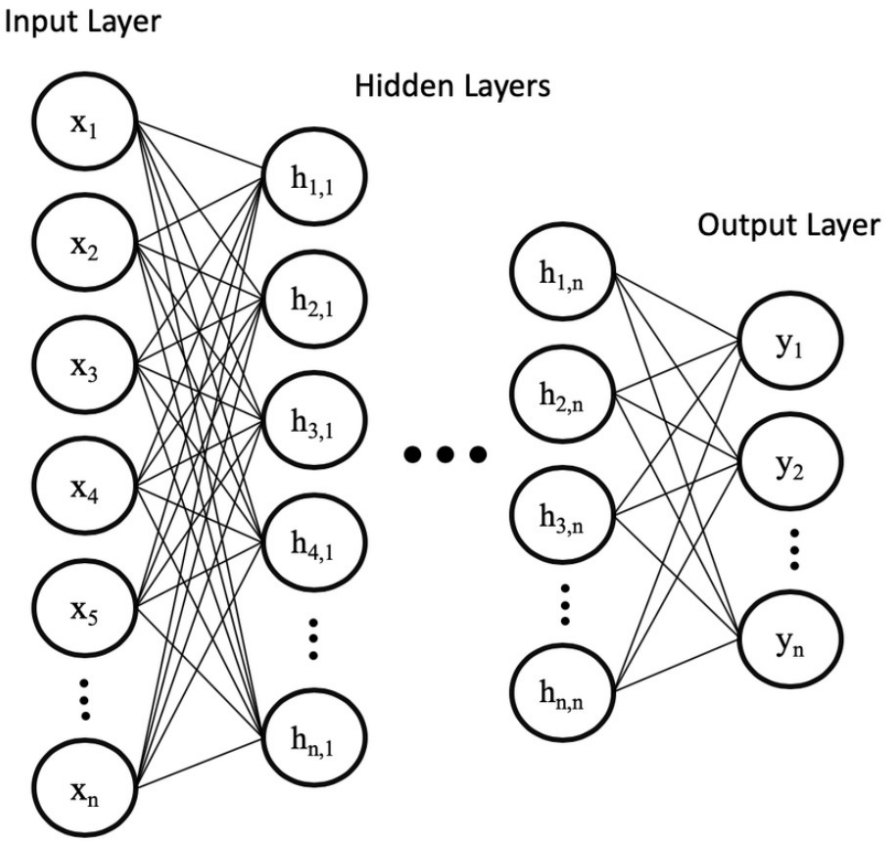


**Figure 3.6:** Feedforward Neural Network example [4].

or the local minimum. This weight adjustment is made with the use of gradient decent, a process during which the neural network determines which direction to take in order to reduce its errors. Some of the well-known loss functions are demonstrated in Figure 3.7. The choice of the function is contingent on the type of problem that the model is trying to solve. For classification tasks, two commonly used loss functions are the Binary Cross Entropy and the Hinge-Loss.

With each training example, the parameters of the model adjust to gradually converge at the minimum. Once a training epoch is concluded, back-propagation can be used, in order to feed the loss backwards in a way that the model's weights and parameters can be fine-tuned.

Three of the most basic and well known categories of NNs are:

- The Perceptron: Perceptrons consist of an input layer, hidden layer(s), and an output layer, being one of the simplest types of NNs. [17].

- Convolutional Neural Networks (CNNs), [18], which are most

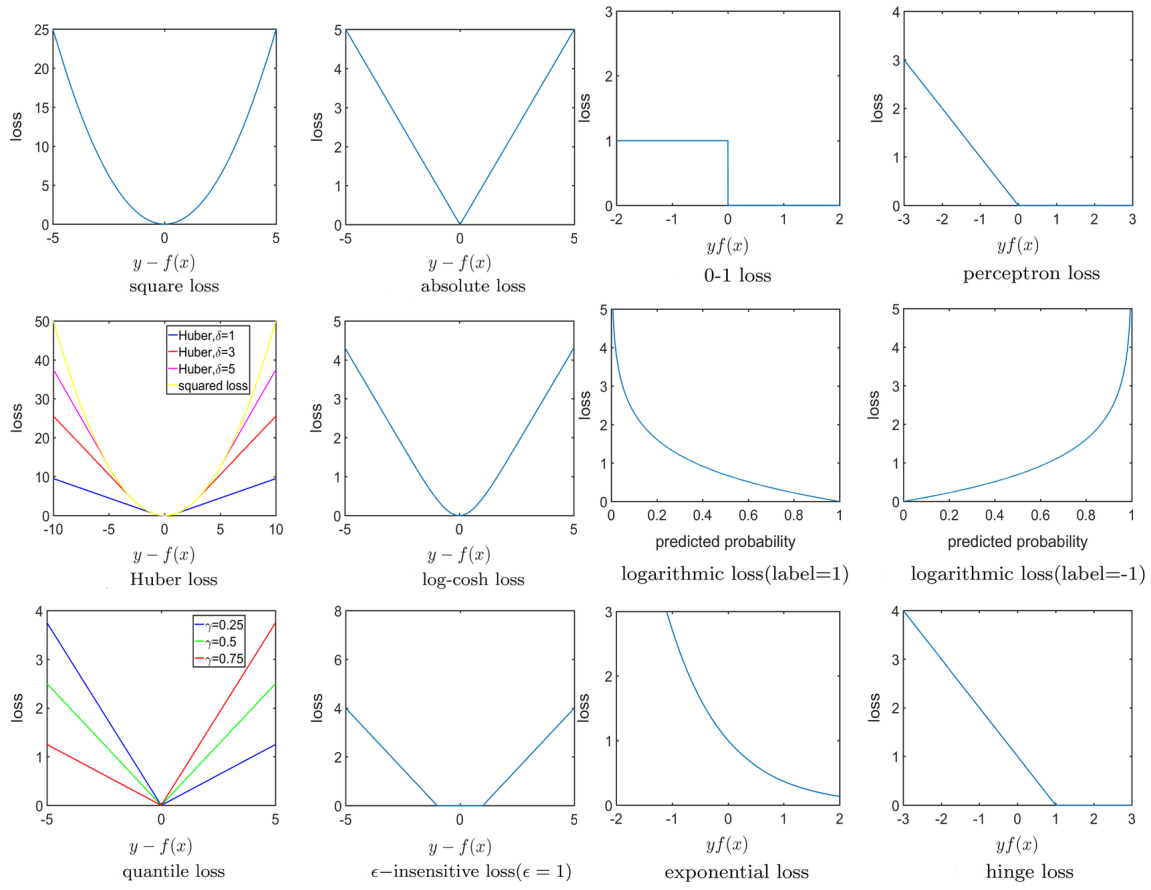


**Figure 3.7:** Well-Known Loss Functions [5]

of the times applied for image recognition, pattern recognition, and/or computer vision. They detect patterns by using matrix multiplication instead of the usual summation which occurs in other types of NNs.

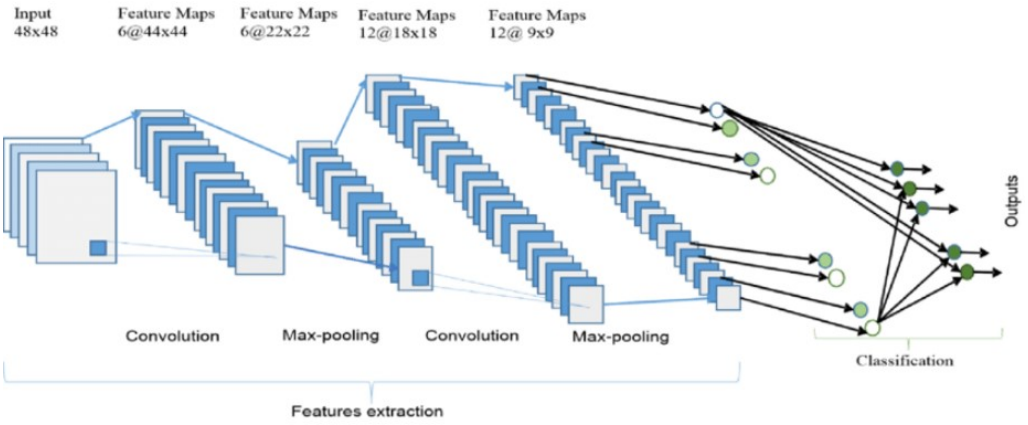


**Figure 3.8:** Convolutional Neural Network example [6].

- Recurrent neural networks (RNNs) which are mostly used against time-series formatted datasets, as they take advantage of past information in order to make predictions about future values Recurrent Neural Networks are based on the work made by [19].

Aside from the loss function, there are certain performance metrics which are used by analysts in order to evaluate a model. The most common performance metrics which will also be referred to in this thesis for the evaluation of the classifier are:

- Confusion Matrix : It is a $2 \times 2$ matrix containing the values of the True Positive(TP), False Positive(FP), False Negative(FN) and True Negative(TN) predictors in the output of the Neural Network

- Accuracy : The ratio of the correct predictions and the total number of predictions. In several cases, this metric might be misleading, especially in highly-imbalanced datasets, in which it is easy for the model to predict negative cases but not positive ones.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

- Precision : The precision of a model defines the ratio of the correct predictions and the total number of positive predictions. An important metric in cases where a high number of False Positive labels could be concerning

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

- Recall(Sensitivity) : Defines how many of the actual positive cases were successfully predicted by the model.

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

- F1-Score : A harmonic mean of the precision and recall of the model

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

- F-beta : F-beta is another version of F1-Score, which introduces a beta coefficient in the equation. It is mostly used in classification/regression problems in which the precision and the recall are not equivalent in terms of importance. This beta coefficient can be defined by the user, depending on the classification/regression needs.

$$F - beta = (1 + beta^2) * 2 * \frac{Precision * Recall}{beta^2 * Precision + Recall} \tag{3.5}$$

- AUC-ROC : The Area Under the Curve (AUC) measures the ability of the classifier to distinguish the classes. The Receiver Operator Characteristic (ROC) plots the True Positive Rate against the False Positive Rate.

### 3.3.1 Recurrent Neural Networks

When it comes to time series forecasting, it is quite possible that the input data will exhibit strong temporal dynamics. Recurrent Neural Networks(RNNs) are capable of recognizing sequential characteristics and detect temporal patterns which enable them to predict future values. Most RNNs share the same characteristics:

1. The input and output sizes may vary

2. There are three important vectors, the input vector, the hidden state vector and the output vector.

3. The hidden state vector, usually initialized to zero, represents the past knowledge. In each RNN layer, the hidden state vector from the previous layer, along with the current input vector, form the current hidden state vector which is used to provide the current output vector.

4. The vectors, the parameters and the activation functions of the RNN will remain the same throughout the entire process and only be updated in each layer.

5. There are four widely used RNN types which are categorized by the input/output vector sizes. There's one-to-one, one-to-many, many-to-one, and many-to-many.

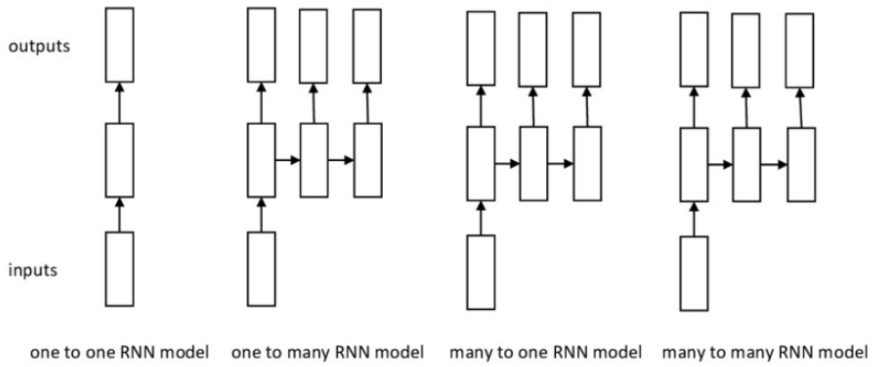


**Figure 3.9:** The four different RNN architectures as explained above [7]

The most popular RNN architecture is the Long-Short-Term-Memory (LSTM), originally proposed in [20]. Differing from the Vanilla RNN architecture, it introduces three types of selected memories known as Gates :

- The Input Gate : The input gate filters the information in the input, by not allowing non essential information to be included.
- Considering that the information changes while time passes, some of it might lose its value. This information is filtered out.
- The output gate chooses only the essential information to send in the output, the rest will be stored in the hidden state.

Furthermore, there's a Sigmoid activation function which allows the model to represent the percentage of information in all the three mentioned gates.

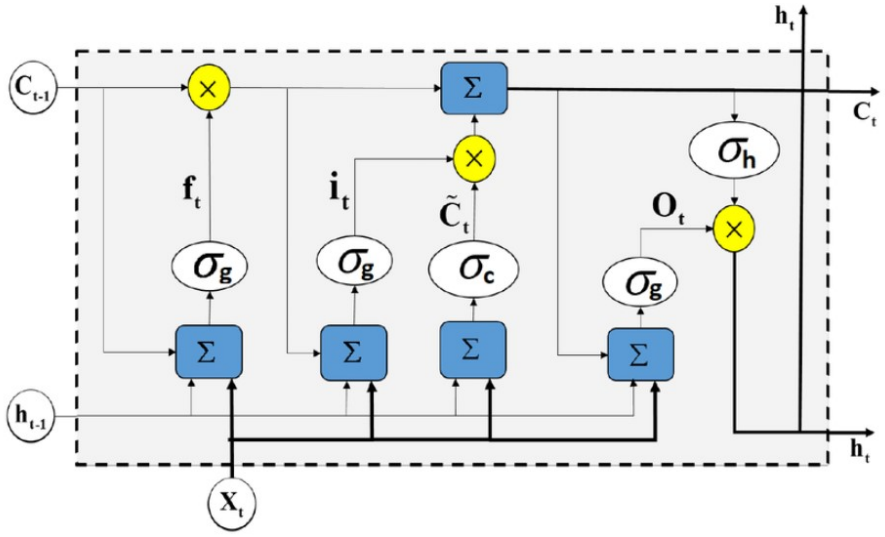


**Figure 3.10:** The architecture of a basic LSTM with an input gate $i_t$, an output gate $O_t$ and a hidden state $h_t$ [8].

Another RNN architecture which is very popular is the Gated-Recurrent-Unit(GRU), originally proposed by [21]. A similar approach to the LSTM, GRU also gives a solution to the vanishing gradient problem that Vanilla RNNs are struggling with. This time, the gates that are used are:

- The Update Gate : Enables the model to evaluate what percentage of past information should be passed down to future steps.
- The Reset Gate : Helps the model determine what percentage of past information should be forgotten.

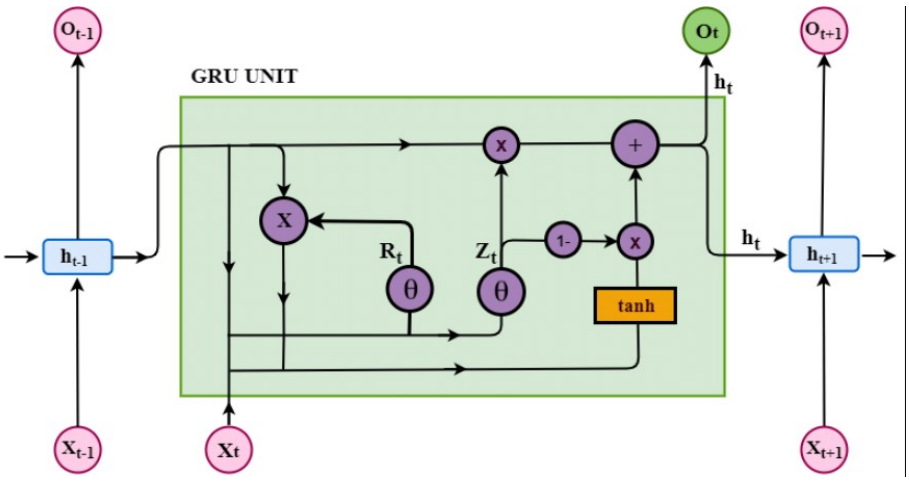


**Figure 3.11:** The architecture of a basic GRU with a reset gate $R_t$, and an update gate $Z_t$ [9].

### 3.3.2 Graph Neural Networks

The design and implementation of Graph Neural Networks (GNNs) has been getting increased traction over the last few years. An idea originally proposed by [22], GNNs can be applied to most types of Graphs and have proven to be very useful for forecasting models especially if the input data exhibit strong spatial dependencies. As we know, a graph consists of nodes and edges which can both have a unique set of features. The goal of a GNN is to make use of graph convolution and predict the state of the node in the next layer, by using a function based on the features of that node as well as its neighbors. The most widely known GNN architectures which were also reviewed for the project of these thesis belong in two major categories for GNN architectures which are:

- Spectral Methods : Spectral methods in [23] use Fourier transform in order to create a representation of the graph in the spectral domain, conduct the convolution operation with element-wise multiplication and then use inverse Fourier transform to transform the signal back to its original domain. The convolution operator is defined as:

$$g * x = F^{-1}[F(g) * F(x)] = U(U^T g * U^T x) \tag{3.6}$$

where $U$ is the matrix defined by the eigenvectors of the $L = U\Lambda U^T$, and $\Lambda$ being the diagonal matrix with the eigenvalues of the graph.

The spectral methods that were considered for this thesis are :

1. ChebNets : Proposed by [24], ChebNets make use of Chebyshev expansion of order K to define a K-Localized convolution which will be computed using Chebyshev polynomials.
2. Graph Convolutional Networks(GCNs) : Proposed by [25], GCNs simplify the K-Localized convolution proposed by ChebNets by setting K equal to 1. In addition, they introduce self-connections by adding the identity matrix I to the adjacency matrix A:

   $$\widetilde{A} = A + I \tag{3.7}$$

   They implemented the symmetric normalization of the Laplacian L:

   $$L_{norm} = D^{\frac{-1}{2}} * L * D^{\frac{-1}{2}} = I - D^{\frac{-1}{2}} * A * D^{\frac{-1}{2}} \tag{3.8}$$

   And finally, they created a renormalization to minimize exploding/vanishing gradient problems:

   $$I + D^{-\frac{1}{2}} * A * D^{-\frac{1}{2}} \rightarrow \widetilde{D}^{-\frac{1}{2}} * \widetilde{A} * \widetilde{D}^{-\frac{1}{2}} \tag{3.9}$$

   where $\widetilde{D}_{ij}$ is the degree matrix of the graph and it gives us information about the degree of each node which is basically the number of the node's neighbors plus one and is created through row-wise summation of its adjacency matrix

   $$\widetilde{D}_{ij} = \sum_{j=1}^{M} (\widetilde{A}_{ij}) \tag{3.10}$$

To summarize all of the above, this is the update rule of the GCNs:

$$H^{(l+1)} = \sigma(\widetilde{D}^{-\frac{1}{2}} * \widetilde{A} * \widetilde{D}^{-\frac{1}{2}} * H^{(l)} * W^{(l)}) \tag{3.11}$$

where $H$ is the feature matrix, $l$ is the layer, and $W$ the trainable weight.

- Spatial Methods : In Spatial Graph Convolutions, the neural network will attempt to learn a function $f$ that is invariant to permutations of node orderings (ex. sum,mean) which will generate a projection of the node's feature vector and update it based on itself as well as the aggregated neighborhood representation. The algorithms that were considered in this thesis are:

  1. Message Passing Neural Networks (MPNNs) : The MPNNs [26] have the ability to send messages across the edges of a graph which are computed using a multi-layer perceptron function $f_e$. The message between two nodes can be described as:

  $$m_{ij} = f_e(h_i, h_j, e_{ij},) \tag{3.12}$$

  In this case, the updated feature vector of the node is created by using the aggregated representation of all the inbound messages to the node via another multi-layer-perceptron function $f_v$. The update rule is:

  $$h_i = f_v(h_i, \sum_{j}^{N_i} m_{ij}) \tag{3.13}$$

  2. Proposed by [27], Graph Attention Networks (GATs) introduce an attention mechanism to the graph convolution. While in GCNs the attention coefficient is calculated explicitly from the graph structure, GATs consider this coefficient as a learnable parameter.

  $$\alpha_{ij} = attention(h_i, h_j) \tag{3.14}$$

Each neighbor's node features as well as edge features, if any, will be the parameters that will define the attention coefficient. An important note is that in GATs, the structure of the graph is not taken into consideration for the computation of the attention coefficient. Finally, the update rule of the GAT is:

$$h_i^{(l)} = \sigma(\sum_{i}^{N_j} \alpha_{ij} * W * h_j) \tag{3.15}$$

Comparing the aforementioned architectures simply from a theoretical point of view, we know that the GNNs are the most computationally efficient, while GATs have the capacity of giving different attention to each neighboring node which may lead to better results. Message Passing Neural Networks is a quite generic GNN architecture and has decent transfer-ability from graph to graph, however they lack scale-ability since they require additional processing and storing due to the messages.

# 4 Implementation

## 4.1 Graph Creation

The real world data consist of the key performance indicators (KPIs) of approximately ten thousand cellular antennas, as well as some meta-data features describing the structural specifications of the antenna and its geo-location. Three of these features will contribute in the creation of the Graph, two of which are the latitude and longitude of the cellular antenna, while the third being the identifier of the cell site in which that cellular antenna belongs to.

In order to perform a Graph Convolutional Layer, a graph needs to be defined. It is an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ denotes the vertices or nodes $v_i \in \mathcal{V}$ and $\mathcal{E}$ denotes the edges $(v_i, v_j \in \mathcal{E})$. The nodes $\mathcal{V}$ of the graph represent the antenna cells and the edges $\mathcal{E}$ represent their physical distance, in essence describing whether or not two antennas are considered neighbors. Using the features which define the antenna's geo-location through their latitude and longitude, the pairwise Geodesic Distance is calculated, by first finding the angle between two cells and then multiplying it by the circumference of the earth:

$$\begin{aligned} angle &= \arccos(cell_i * cell_j) \\ dist &= angle * pi * R \end{aligned} \tag{4.1}$$

where $cell_i, cell_j$ are the latitude and longitude values of the two cells and $R$ is the radius of the earth.

This process provides a strictly upper triangular distance matrix $D \in \mathbb{R}^{|M|\times|M|}$, with $M$ being the total number of antenna cells in the network. The adjacency matrix $A$ is defined by setting a distance threshold $t$. If $dist \leqq t$, cells $cell_i, cell_j$ are considered neighbors. This will result in a strictly upper triangular adjacency matrix $A \in \mathbb{R}^{|M|\times|M|}$

which essentially defines which of the antennas are considered to be neighbors.

It is important to remember that the dataset is characterized by class imbalance. In order for the model to be able to efficiently detect spatial patterns, the Graph needs to be defined with no more than the absolute necessary connections between the antenna cells. If the threshold $t$ is a relatively high number, the above process will provide a quite exuberant adjacency matrix. As a result, the training of the model would have increased time complexity, much larger memory requirements while also having significant noise in the calculations for the detection of spatial patterns.

## 4.2 Data Pre-processing

Aside from the KPIs which give information about an antenna's geolocation and structural information, there are others which provide information about the antenna's status. These features are the most important ones for the model since they contain the spatial and temporal patterns which the model will try to detect. The meta-features are not relative to this classification problem and are completely removed from the dataset after the graph is created. The next step is to deal with the missing values problem.

### 4.2.1 Imputation

In the dataset there exist about twenty eight million samples and seventy seven total features, which is roughly 2,15 billion values out of which 15% is missing. As for the unit non-response, it is estimated that about 2 million samples are missing from the dataset. Such cases can be detected by iterating through the samples of any antenna and checking the time-stamp difference between two subsequent samples. The cases of item non-response and unit-non-response will be dealt with separately. It is also worth mentioning that in this case, the values are missing completely at random (MCAR), meaning that no specific pattern can be detected in the abscence of data point.

The first category of missing values that will be handled, is the item non-response. The imputation techniques applied for this task are the

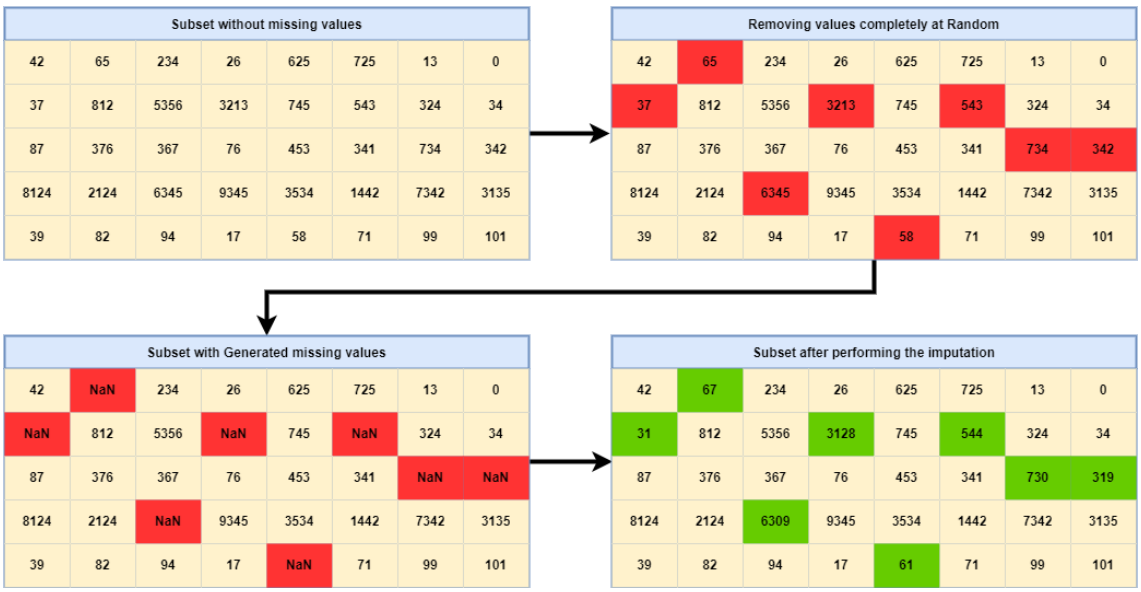


**Figure 4.1:** Imputation Validation process. Comparing the values removed from the original subset with the values that resulted from the imputation.

ones mentioned in the Data Imputation section 3.1 and compare each of the algorithm's time complexity and imputation accuracy. In order to validate the imputation results, I select a subset of the dataset in which there are no missing items in the samples and then randomly select some of these values to be set to *null* while also keeping the original values for the validation process.

The algorithms which applied for this task are the zero filling, mean / median / most-frequent imputation, hot and cold deck imputation and K-Nearest-Neighbors.

| | Zero Filling | Mean | Median | Most Frequent | Hot Deck | Cold Deck | KNN |
|---|---|---|---|---|---|---|---|
| Imputation Accuracy(%) | 0.39 | 0.61 | 0.57 | 0.48 | 0.60 | 0.58 | 0.64 |
| Run-Time(sec) | 5,732 | 10,621 | 9,105 | 9,009 | 16,318 | 15,758 | 25,122 |

**Table 4.1:** Performance of the different imputation methods.

Though the least time consuming algorithm is the Zero-Filling, which was to be expected since there are no extra computations to be made,

it is also the least accurate of the algorithms. The KNN algorithm is capable of producing the best results, however due to the nature of the data it has an extremely high time complexity which exceeds the time-constraints which are set for the project. The best all around option between the above algorithms is the Mean Imputation approach.

### 4.2.2 Prediction Horizons

Each of the antennas has a feature which provides the intelligence of whether or not an antenna is overloaded at a given time. It is a binary feature and it will act as the target variable for the classification problem. In order to create the prediction horizons, for each antenna cell in the time-series, some new features need to be created, one for each hotspot prediction horizon. The new features will have the same values as the original target variable but shifted to the right according to the $H$ factor which is the horizon time-step.

The creation of the prediction horizons also marks the point at which my data is ready to be processed by the Neural Network.

## 4.3 The Neural Network

In this section, I present all the necessary steps and decisions which were made for the creation of the Neural Network's architecture.

### 4.3.1 Capturing Spatial Dependencies

The first step is to determine which of the three GNN architectures that initially considered for this project in section 3.3.2 should be the final choice. To this end, I need to consider the following requirements:

- Efficiency : The primary target of the model is to maximize the precision of its forecasting. In my baseline models, the architecture that will provide the best precision metrics will be the most likely candidate for the final choice.

- Time Complexity : It is essential that the model can be trained and evaluated on real world data quickly. The goal of this application is to be both fast and accurate, since most of the forecasting horizons that I have created make predictions not further than a twenty-four hour window. Out of the aforementioned architectures, the GCN is by far the least time consuming, since the MPNN needs to calculate more variables due to the edge features while the GAT has more learning parameters.

- Memory : The size of the dataset is quite large, containing samples for every hour within four months for about ten thousand cellular antennas. In order to create a model applicable for most machines I need to consider the memory requirements. The MPNN architecture is a needy algorithm in terms of memory as it requires the storage and calculation of both node features and edge features. Taking into account that I will be dealing with a considerably large number of nodes in the graph, the idea of MPNNs was abandoned.

In order to evaluate the performance of the remaining two architectures, a very small subset of the dataset is selected, which will be the product of a down-sampling technique in order to take care of the class imbalance. In all these experiments, the exact same model parameters are used.

| | $hz = 12$ | | $hz = 24$ | | $hz = 48$ | | All Horizons |
|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Time per Epoch (sec) |
| GCN | **0.07** | 0.02 | **0.05** | 0.03 | 0.03 | **0.02** | **12,849** |
| GAT | 0.04 | 0.02 | 0.04 | **0.04** | 0.03 | 0.01 | 14,296 |

**Table 4.2:** Performance of the GCN and GAT architectures across prediction horizons `hz` $\in \{12, 24, 48\}$ hours.

Taking into consideration the results of the baseline, the final choice is the Graph Convolutional Neural Network. The goal here is to detect the patterns that exist between the behavioral status of cellular antennas. The depth of this search is determined by the number of the GCN layers that implemented in the model. In each layer, the model

predicts the next hidden state by aggregating the feature vectors of all neighboring antennas as explained in the theoretical section 3.3.2 and shown in Figure 4.2.

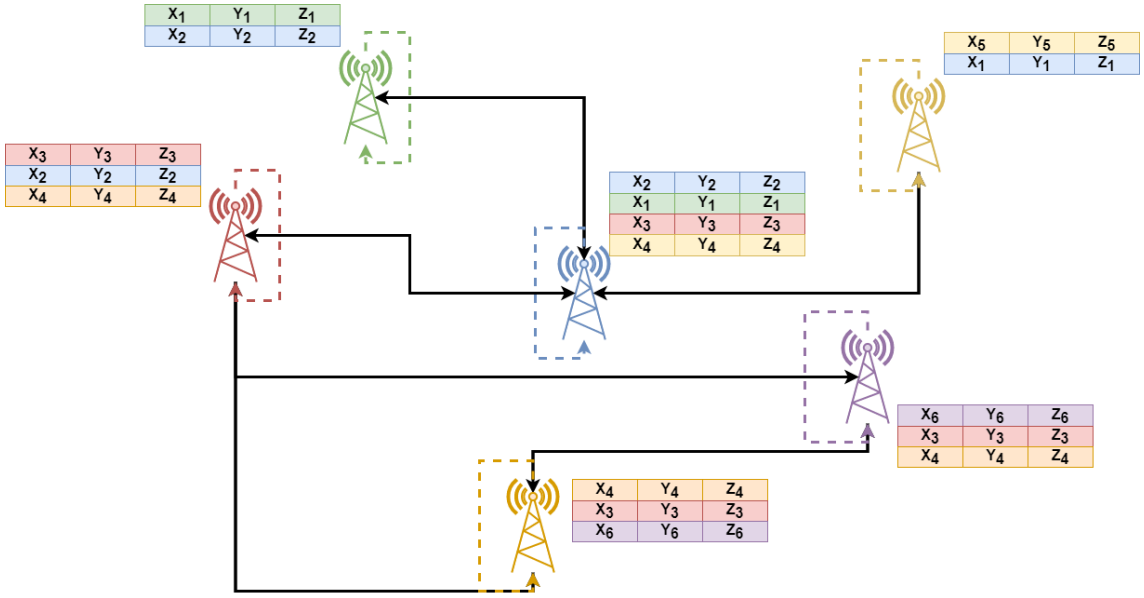


**Figure 4.2:** GCN Process in the Cellular Network.

Once again, I will make a reference to the class imbalance that the data suffers from, which in this particular case means that the more antennas I add into the aggregation, the more noise I will have in the output signal. This in turn means that the initial precision score will not be the desirable one.

Another parameter I need to consider is the ratio between input and output channels of each layer. In the initial approach they are set to be equal.

### 4.3.2 Capturing Temporal Dependencies

For the choice regarding the RNN architecture, both the LSTM and the GRU architectures will be evaluated in order to compare their efficiency, time complexity and also memory usage in the same baseline example with the exact same parameters.

| | | $hz = 12$ | | $hz = 24$ | | $hz = 48$ | | All Horizons |
|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Time per Epoch (sec) |
| mb = 12 | GRU | 0.19 | 0.13 | 0.18 | 0.11 | 0.15 | 0.09 | **15,108** |
| | LSTM | 0.18 | 0.14 | 0.13 | 0.12 | 0.13 | 0.09 | 15,894 |
| mb = 24 | GRU | **0.29** | 0.13 | 0.24 | 0.10 | **0.18** | 0.11 | **16,322** |
| | LSTM | 0.26 | 0.11 | 0.22 | **0.15** | 0.17 | **0.12** | 17,488 |
| mb = 48 | GRU | 0.21 | 0.10 | 0.17 | 0.11 | 0.16 | 0.09 | **18,292** |
| | LSTM | 0.19 | 0.08 | 0.13 | **0.15** | 0.14 | 0.10 | 20,094 |

**Table 4.3:** Performance of the GRU and LSTM architectures across prediction horizons hz $\in \{12, 24, 48\}$ hours with different layers mb $\in \{12, 24, 48\}$.

As shown in Table 4.3, these two algorithms yield almost the same results, however since the GRU is slightly better than the LSTM it will be my final choice for the Neural Network. In my initial approach I will use a default twenty-four-layer GRU which means I will be looking back twenty four time-steps to get the temporal information in order to achieve my forecast. As for the hidden state and the output state feature vectors I will set them to be the same size as the input vector.

### 4.3.3 Full Model Representation

Combining the aforementioned modules, we have the GCN that creates an aggregated feature vector, based on the information of each antenna combined with its neighbors, which I then want to pass as input to the GRU module. The final layer in my model will be a fully connected linear layer which will provide us with a single-shot prediction for my target variable. It's important to note that this architecture supports multiple consecutive GCN layers to be added depending on the size of the Graph that is being tested and the complexity of the problem, as well as variant sized stacked Gated Recurrent Unit layers which would change the historical depth of past data.

After each GCN layer, a Rectified Linear Unit (ReLU) activation

function is applied which eliminates all negative values in the output of the layer. The ReLU helps dealing with the vanishing gradient problem since its derivative for values larger than zero equals to one, therefore the product of the multiplication $1x1$ always gives one as many times as it is used. Furthermore it introduces sparsity in the Network with benefits such as information disentangling, efficient variable-size representation and linear separability [28]. After the fully connected layer, a Sigmoid activation function is applied, resulting in the probability of each antenna to become a hotspot at the specified time.

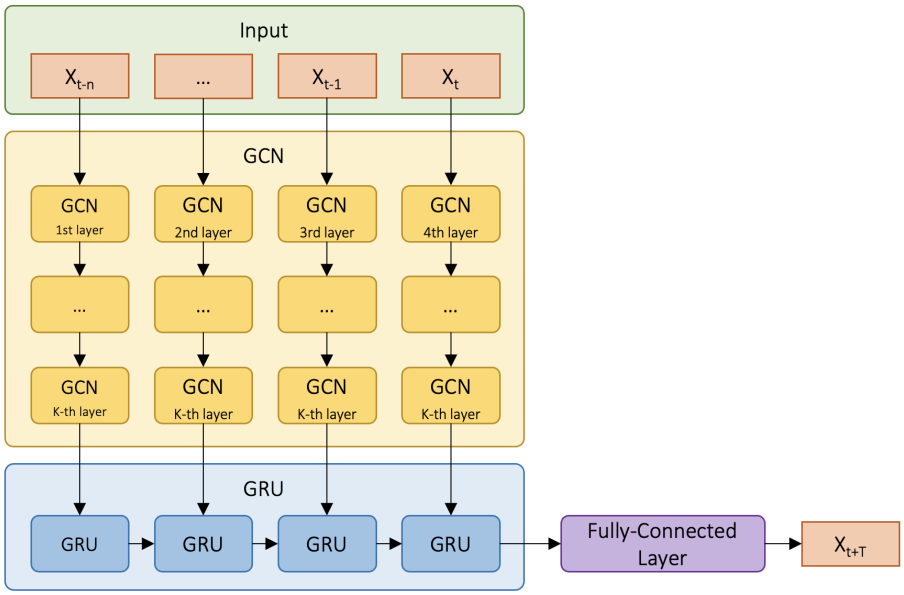


**Figure 4.3:** Model architecture shown in a block diagram.

The decision threshold is 0,5%, which means that every antenna having a probability higher than 0,5% will be predicted to be a hotspot at the target forecasting horizon.

The loss of the model after each iteration is calculated by the Binary Cross Entropy with Logits(BCE) function, since I am dealing with a binary classification problem.

$$\begin{aligned} L &= l_1, ..., l_N \\ l_n &= -w_{n,c}[p_c * y_{n,c} * \log \sigma(x_{n,c}) + (1 - y_{n,c}) * \log(1 - \sigma(x_{n,c}))] \end{aligned} \tag{4.2}$$

where $c$ is the class number(in my case $c = 1$ as I am dealing with a binary classification problem), $n$ is the number of samples in the batch, $w_{n,c}$ is a re-scaling weight for the loss function, and $p_c$ is the positive weight that I use to multiply the positive examples in my data.

In the end, a reduction rule is applied to the $L$ which can be either *mean* or *sum*.

### 4.3.4 Initial Design Results

The initial model design was trained on the entire graph using a default number of training epochs and neurons. The memory buffer used in this experiment is $mb = 24$ hours.

| | $hz = 12$ | | $hz = 24$ | | $hz = 48$ | | All Horizons |
|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Time per Epoch (sec) |
| GCN | 0.07 | 0.02 | 0.05 | 0.03 | 0.03 | 0.02 | 12,849 |
| GRU | 0.29 | 0.13 | 0.24 | 0.10 | 0.18 | 0.11 | 15,108 |
| GCNGRU | **0.39** | 0.08 | 0.36 | 0.05 | 0.30 | 0.06 | 20,575 |

**Table 4.4:** Performance of the GCN, GRU and GCN-GRU across prediction horizons `hz` $\in \{12, 24, 48\}$ hours.

As shown in Table 4.4, the initial goal of the project is far from being accomplished. One of the reasons behind the initial failure is the massive class imbalance of my dataset. The BCE loss function supports the application of class weights in order to help tackle this issue. In [equation], I can try a variety of combinations between the scaling factor and the $pos_{weight}$ in my attempt to reach a desirable outcome. From the documentation of the BCEWithLogitsLoss, I know that adding a $pos_{weight} < 1$ should improve the precision of the model.

Again, none of the above combinations is capable of improving the results. The issue can not be handled by simply adding class weights, so the solution has to be a down-sampling technique.

| | $hz = 12$ | | $hz = 24$ | | $hz = 48$ | | All Horizons |
|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Time per Epoch (sec) |
| GCN | 0.09 | 0.01 | 0.08 | 0.03 | 0.04 | 0.01 | 12,849 |
| GRU | 0.31 | 0.12 | 0.27 | 0.07 | 0.22 | 0.06 | 15,108 |
| GCNGRU | **0.42** | 0.08 | 0.38 | 0.04 | 0.35 | 0.03 | 20,575 |

**Table 4.5:** Performance of the GCN, GRU and GCNGRU across prediction horizons `hz` $\in \{12, 24, 48\}$ hours, with added class weights.

## 4.4 Hyper-parameter Tuning

### 4.4.1 Feature Selection

As explained in the theoretical background section, a crucial factor that could help improve the precision of the Neural Network's predictions is the application of a feature selection algorithm. Incorporating feature selection in the pipeline ensures that all of the features that will be used as input for the training process are important for the classification process, while the non-important features will be eliminated to avoid noise and unnecessary calculations thus also saving some time.

The first algorithm that will be used is the variance thresholding. A small number of the features are static values, which do not contribute at all in the forecasting task, but rather increase training time and consume memory. By applying the method VarianceThreshold from the scikit-learn library and setting the threshold to be equal to 0 all of the static features will be eliminated from the training set.

The rest of the features will need to be evaluated through more elaborate techniques and algorithms, therefore the the wrapper, filter and embedded methods will be applied as explained in the theoretical section. Once again, the algorithm that will be used will be selected based on its efficiency and time complexity. For the evaluation regarding wrapper and embedded methods, a subset of the original dataset will be used.

As expected, the wrapper methods are the most time consuming

| | Forward | Backward | Stepwise | CorMap | LASSO | RIDGE |
|---|---|---|---|---|---|---|
| Precision | 0.49 | 0.47 | 0.52 | 0.43 | 0.55 | 0.51 |
| Time(sec) | 4,027 | 3,918 | 9,878 | 193 | 2,519 | 2,384 |

**Table 4.6:** Performance of the different feature selection algorithms.

solutions and the filter methods the least efficient ones. As a result I will be using the Lasso Regression algorithm for the feature selection. Finally, I compare the results provided by using the original dataset with the results that came from the subset of the selected features. The feature analysis has improved the precision of the Neural Network's predictions and slightly decreased its training time.

### 4.4.2 Graph Partitioning

Since my data is presented in a time-series format, down-sampling can not happen on a sample level, so it has to be on a node level. The goal here is to help reduce the system requirements in terms of memory, speed-up the training and evaluation process and in the meantime deal with the class imbalance that I am facing. For my initial test, I will randomly select 100 antenna cells from the grid, and perform the calculations on that sub-graph, as shown in the figure below.

| | $hz = 12$ | | $hz = 24$ | | $hz = 48$ | | All Horizons |
|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Time per Epoch (sec) |
| GCN | 0.47 | 0.11 | 0.46 | 0.14 | 0.39 | 0.11 | 1,917 |
| GRU | 0.63 | 0.14 | 0.64 | 0.09 | 0.48 | 0.12 | 11,917 |
| GCNGRU | **0.75** | 0.09 | 0.71 | 0.13 | 0.070 | 0.14 | 13,028 |

**Table 4.7:** Performance of the GCN, GRU and GCNGRU across prediction horizons $\texttt{hz} \in \{12, 24, 48\}$ hours.

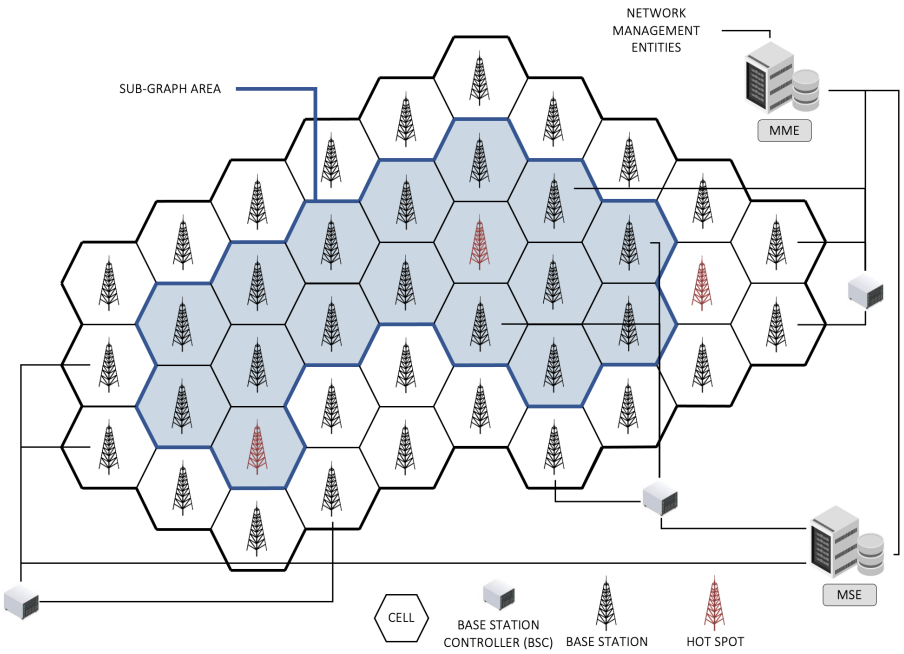


**Figure 4.4:** Overview of the selection of the sub-graph in a targeted area.

The results shown in Table 4.7 are quite close to the original goal. This sparked the idea and main contribution of this thesis, which is to detect all antennas with at least one active case in the training data and then split them into groups of 100 creating K different sub-graphs. Then, a different model is trained on each sub-graph, and afterwards it is evaluated it not only on the sub-graph that it was trained on, but also all the rest K-1 sub-graphs, as well as sub-graphs that contain antennas with zero active cases. In Table 4.8 the average precision and recall of the models were calculated using the aggregated confusion matrices.

This solution is not only better in terms of efficiency, but is also a lot less time consuming and does not have as large memory requirements as the initial approach. Furthermore, it gives us the option to train and evaluate the model locally, meaning it is not required to have the entire graph as the input if the research is only interested in the behavior of a specific group of antenna cells.

In addition, I can compare the transfer-ability of each model from one graph to another by also looking at the similarities between the graphs. The similarity grading algorithm that I will use calculates

| mb = 24 hours | | $SG_1$ | | $SG_2$ | | $SG_3$ | | $SG_4$ | | $SG_5$ | | $SG_6$ | | $SG_7$ | | $SG_{1-7}$ | $SG_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| | $SG_1$ | 0.70 | 0.51 | 0.53 | 0.35 | 0.67 | 0.40 | 0.65 | 0.38 | 0.52 | 0.22 | 0.81 | 0.44 | 0.59 | 0.41 | 0.66 | 0.39 |
| | $SG_2$ | 0.66 | 0.09 | 0.77 | 0.27 | 0.55 | 0.13 | 0.72 | 0.22 | 0.57 | 0.11 | 0.90 | 0.36 | 0.63 | 0.15 | 0.71 | 0.30 |
| | $SG_3$ | 0.63 | 0.41 | 0.57 | 0.27 | 0.63 | 0.46 | 0.64 | 0.26 | 0.45 | 0.18 | 0.75 | 0.48 | 0.54 | 0.32 | 0.61 | 0.43 |
| | $SG_4$ | 0.66 | 0.45 | 0.54 | 0.44 | 0.53 | 0.33 | 0.68 | 0.57 | 0.42 | 0.25 | 0.67 | 0.63 | 0.55 | 0.43 | 0.60 | 0.45 |
| | $SG_5$ | 0.61 | 0.41 | 0.53 | 0.37 | 0.68 | 0.32 | 0.65 | 0.41 | 0.48 | 0.02 | 0.64 | 0.29 | 0.59 | 0.39 | 0.60 | 0.38 |
| | $SG_6$ | 0.74 | 0.29 | 0.61 | 0.36 | 0.62 | 0.25 | 0.72 | 0.40 | 0.53 | 0.19 | 0.86 | 0.59 | 0.68 | 0.33 | 0.70 | 0.35 |
| | $SG_7$ | 0.68 | 0.33 | 0.62 | 0.10 | 0.72 | 0.18 | 0.58 | 0.38 | 0.44 | 0.13 | 0.58 | 0.16 | 0.76 | 0.40 | 0.63 | 0.27 |
| | GCN | 0.29 | 0.08 | 0.00 | 0.00 | 0.35 | 0.09 | 0.21 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.11 | 0.06 | | |
| | GRU | 0.61 | 0.39 | 0.30 | 0.28 | 0.25 | 0.28 | 0.51 | 0.41 | 0.38 | 0.33 | 0.59 | 0.29 | 0.41 | 0.47 | | |

**Table 4.8:** Performance of the GCNGRU, the GCN and the GRU across all sub-graphs ($SG_1$-$SG_7$), for forecasting horizon $\texttt{hz} = 12$ hrs. Average Precision and Recall were calculated using the aggregated confusion matrices.

the Laplacian Eigenvalues of each sub-graph's adjacency matrix as proposed in [29]. Then for each sub-graph, I find the smallest $k$ such that the sum of the $k$ largest eigenvalues constitutes at least 90% of the sum of all of the eigenvalues. Then if the values of $k$ are different between the two sub-graphs, I use the smaller one. Finally, the sum of the squared differences between the largest $k$ eigenvalues between the sub-graphs is the similarity metric and is defined as:

$$\begin{gathered} sim = \sum_{i=1}^{k} (\lambda_{1i} - \lambda_{2i})^2 \\ \text{where k is chosen so that,} \\ \min_j \Big\{ \frac{\sum_{i=1}^{k} (\lambda_{ji})}{\sum_{i=1}^{k} (\lambda_{ji})} > 0.9 \Big\} \end{gathered} \tag{4.3}$$

where $sim \in [0, +\infty)$ for the two sub-graphs $\mathbf{SG}_1$ and $\mathbf{SG}_2$ and

$$\lim_{sim \to 0} f(SG_1, SG_2) \tag{4.4}$$

suggests that the two sub-graphs are similar.

This algorithm can be applied in this classification problem to investigate the sub-graph similarity. The reason why it is necessary for such inquiry to be conducted, is due to the fact that the NN implementation that is being tested makes use of GCN-Layers. As explained in the theoretical section 3.3.2, the structural information of the graph is also

being used for the computations, therefore graph similarity is bound to influence the performance of the model depending on what sub-graph it is being evaluated on.

| | $SG_1$ | $SG_2$ | $SG_3$ | $SG_4$ | $SG_5$ | $SG_6$ | $SG_7$ |
|---|---|---|---|---|---|---|---|
| $SG_1$ | 0.00 | 422.81 | 366.03 | 347.03 | 790.57 | 136.38 | 478.85 |
| $SG_2$ | 422.81 | 0.00 | 367.33 | 248.12 | 492.47 | 137.24 | 480.34 |
| $SG_3$ | 366.03 | 367.33 | 0.00 | 247.54 | 791.29 | 136.68 | 479.51 |
| $SG_4$ | 247.03 | 248.12 | 247.54 | 0.00 | 793.26 | 137.57 | 480.91 |
| $SG_5$ | 790.57 | 792.47 | 791.29 | 793.26 | 0.00 | 795.67 | 475.56 |
| $SG_6$ | 136.38 | 137.24 | 136.68 | 137.57 | 795.67 | 0.00 | 482.97 |
| $SG_7$ | 478.85 | 480.34 | 479.51 | 480.91 | 475.56 | 482.97 | 0.00 |

**Table 4.9:** sub-graph Similarity matrix. The largest the value, the more dissimilar the sub-graphs are.

With the results provided from the similarity analysis which is demonstrated by Table 4.9, one can already detect some patterns. For example the $SG_5$ sub-graph is seemingly the most dissimilar one compared to the others and it has the lowest performance metrics. I will further investigate these results as well as their correlation with the transfer-ability later.

### 4.4.3 Neurons

Another parameter that can be tuned in order to provide more accurate results is the number of neurons that I am using in the hidden state between the GCN layer and the GRU module. In my initial approach I used 32 neurons which is about the same size as the input features of the data.

Generally, if I use too many neurons for the training then the model will tend to over-fit to the training data while also increasing the memory requirements. However, if I use fewer than necessary then the training time which is required to reach a desirable result will increase. For this reason, I conducted some experiments with various neurons in order to select the optimal number.

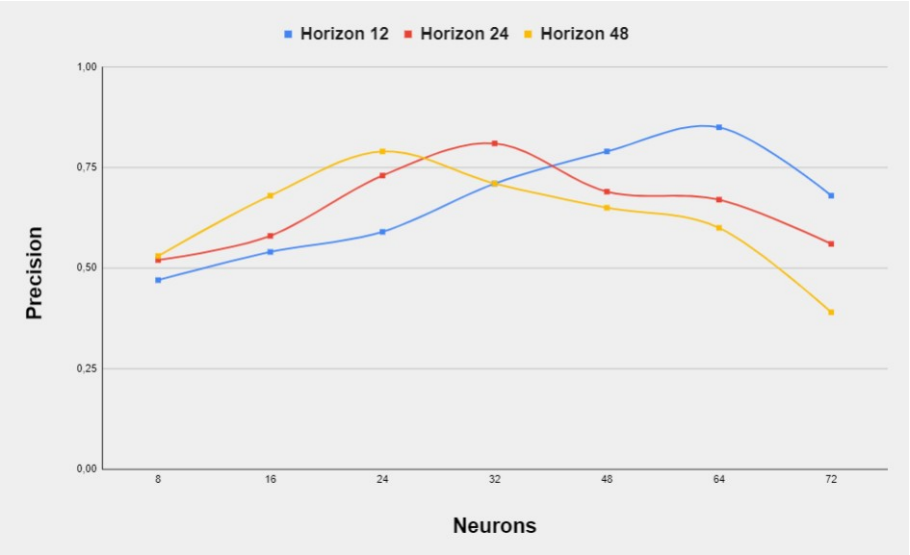


**Figure 4.5:** Precision-Neurons diagram for different hotspot horizons.

### 4.4.4 Optimizing training time

Up until this point, I have been using a default of twenty epochs for the training of each model. However, this is also a parameter that can be tuned and potentially increase the values of my target performance metrics, which are first the precision and second the recall of the model.

In order to optimize the number of epochs that each model will be trained for, I introduce two new hyper-parameters, Patience(P) and Delta(D). I will also be using a validation set aside from the train and test sets. The validation set will be the means through which I will decide the values of my two new hyper-parameters.

- The Patience hyper-parameter is a positive integer that determines the number of epochs that the model will keep training if a target goal is not being reached. Since my first priority is to maximize the precision, this will be the target metric. When this hyper-parameter reaches zero, the model will stop training.
- This hyper-parameter will act as a threshold. After an epoch of training is concluded, if the precision of the model has not improved by a specific margin defined by Delta, then the Patience hyper-parameter will be reduced by one.

| mb = 24 hours | SG$_1$ | | SG$_2$ | | SG$_3$ | | SG$_4$ | | SG$_5$ | | SG$_6$ | | SG$_7$ | | SG$_{1-7}$ | SG$_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| SG$_1$ | 0.86 | 0.25 | 0.60 | 0.05 | 0.81 | 0.16 | 0.92 | 0.15 | 0.68 | 0.04 | 0.86 | 0.20 | 0.96 | 0.19 | 0.85 | 0.15 |
| SG$_2$ | 0.67 | 0.00 | **1.00** | 0.04 | **0.83** | 0.03 | 0.83 | 0.01 | **1.00** | 0.01 | **1.00** | 0.02 | 0.00 | 0.00 | **0.91** | 0.01 |
| SG$_3$ | 0.83 | 0.23 | 0.50 | 0.06 | 0.77 | 0.22 | 0.86 | 0.21 | 0.61 | 0.09 | 0.83 | 0.25 | 0.91 | 0.25 | 0.80 | 0.19 |
| SG$_4$ | 0.74 | 0.24 | 0.63 | 0.21 | 0.71 | 0.16 | 0.83 | 0.34 | 0.66 | 0.10 | 0.79 | **0.43** | 0.69 | 0.16 | 0.75 | **0.24** |
| SG$_5$ | **1.00** | 0.00 | 0.57 | 0.03 | 0.00 | 0.00 | 0.73 | 0.03 | 0.88 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.67 | 0.01 |
| SG$_6$ | 0.62 | 0.11 | 0.88 | 0.07 | 0.80 | 0.10 | 0.87 | 0.13 | 0.49 | 0.04 | 0.95 | 0.33 | 0.98 | 0.19 | 0.83 | 0.14 |
| SG$_7$ | **1.00** | 0.03 | 0.00 | 0.00 | **1.00** | 0.03 | 1.00 | 0.05 | 0.00 | 0.00 | 0.83 | 0.04 | **1.00** | 0.07 | 0.96 | 0.03 |
| GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| GRU | 0.61 | **0.49** | 0.36 | **0.28** | 0.35 | **0.38** | 0.63 | **0.47** | 0.46 | **0.38** | 0.68 | 0.33 | 0.49 | **0.56** | | |

**Table 4.10:** Performance of the GCNGRU across all sub-graphs (SG$_1$-SG$_7$), for forecasting horizon `hz` = 12 hrs with optimized training time.

As I can see, the optimization of training time is another crucial factor which can drastically improve my performance metrics. Also, in most cases there is a trade-off between precision and recall as more and more training epochs pass. This is to be expected as in early training the model is only capable of predicting the strongest patterns, both temporal and spatial, while with increased training time it becomes more generalized, can detect more patterns, however it increases the chances of making FP predictions. Since minimizing these FP predictions is one of my target goals, the combinations of P and D that I will select for each model are the ones which accomplish this goal.

### 4.4.5 Optimizing Memory Buffer Size

The RNN module of my model can be optimized as well. In the previously shown experiments, the input vector had a default size of 24, meaning my historical input dated back as far as one day. Since I am training more than one model, the input vector needs to be optimized for each individual model separately. Though increasing the amount of input data results in larger training times, it does not necessarily provide better results.

The results shown in Table 4.11 provide some interesting information regarding the temporal patterns in my dataset. Clearly none of the

| | | $SG_1$ | | $SG_2$ | | $SG_3$ | | $SG_4$ | | $SG_5$ | | $SG_6$ | | $SG_7$ | | $SG_{1-7}$ | $SG_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| mb = 12 hours | $SG_1$ | 0.82 | **0.31** | 0.76 | 0.15 | 0.86 | 0.27 | 0.83 | 0.23 | **0.79** | 0.08 | 0.88 | **0.44** | 0.84 | 0.30 | 0.84 | **0.25** |
| | $SG_2$ | **0.94** | 0.03 | **0.82** | 0.08 | **0.88** | 0.02 | 0.83 | 0.04 | 0.78 | 0.02 | **0.96** | 0.05 | 0.74 | 0.08 | 0.83 | 0.04 |
| | $SG_3$ | 0.89 | 0.18 | 0.76 | 0.05 | 0.80 | 0.19 | 0.85 | 0.24 | 0.54 | 0.05 | 0.83 | 0.25 | 0.82 | 0.19 | 0.82 | 0.17 |
| | $SG_4$ | 0.84 | 0.15 | 0.73 | **0.17** | 0.65 | 0.13 | 0.83 | 0.09 | 0.55 | 0.07 | 0.92 | 0.36 | 0.60 | 0.11 | 0.78 | 0.19 |
| | $SG_5$ | 0.76 | 0.18 | 0.45 | 0.08 | 0.76 | 0.17 | 0.70 | 0.21 | 0.58 | 0.11 | 0.61 | 0.08 | 0.71 | 0.19 | 0.67 | **0.25** |
| | $SG_6$ | 0.62 | 0.17 | 0.48 | 0.06 | 0.52 | 0.05 | 0.78 | 0.19 | 0.43 | 0.06 | 0.94 | 0.36 | 0.75 | 0.32 | 0.72 | 0.16 |
| | $SG_7$ | 0.87 | 0.09 | 0.81 | 0.04 | 0.84 | 0.12 | **0.86** | 0.15 | 0.50 | 0.02 | 0.87 | 0.23 | **0.91** | 0.16 | **0.86** | 0.12 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.61 | 0.13 | 0.56 | 0.17 | 0.54 | **0.40** | 0.67 | **0.48** | 0.46 | **0.41** | 0.73 | 0.18 | 0.54 | **0.47** | | |
| mb = 24 hours | $SG_1$ | 0.86 | 0.25 | 0.60 | 0.05 | 0.81 | 0.16 | 0.92 | 0.15 | 0.68 | 0.04 | 0.86 | 0.20 | 0.96 | 0.19 | 0.85 | 0.15 |
| | $SG_2$ | 0.67 | 0.00 | **1.00** | 0.04 | **0.83** | 0.03 | 0.83 | 0.01 | **1.00** | 0.01 | **1.00** | 0.02 | 0.00 | 0.00 | **0.91** | 0.01 |
| | $SG_3$ | 0.83 | 0.23 | 0.50 | 0.06 | 0.77 | 0.22 | 0.86 | 0.21 | 0.61 | 0.09 | 0.83 | 0.25 | 0.91 | 0.25 | 0.80 | 0.19 |
| | $SG_4$ | 0.74 | 0.24 | 0.63 | 0.21 | 0.71 | 0.16 | 0.83 | 0.34 | 0.66 | 0.10 | 0.79 | **0.43** | 0.69 | 0.16 | 0.75 | **0.24** |
| | $SG_5$ | **1.00** | 0.00 | 0.57 | 0.03 | 0.00 | 0.00 | 0.73 | 0.03 | 0.88 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.67 | 0.01 |
| | $SG_6$ | 0.62 | 0.11 | 0.88 | 0.07 | 0.80 | 0.10 | 0.87 | 0.13 | 0.49 | 0.04 | 0.95 | 0.33 | 0.98 | 0.19 | 0.83 | 0.14 |
| | $SG_7$ | **1.00** | 0.03 | 0.00 | 0.00 | **1.00** | 0.03 | **1.00** | 0.05 | 0.00 | 0.00 | 0.83 | 0.04 | **1.00** | 0.07 | 0.96 | 0.03 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.63 | **0.51** | 0.38 | **0.29** | 0.33 | **0.41** | 0.62 | **0.48** | 0.41 | **0.39** | 0.62 | 0.36 | 0.48 | **0.59** | | |
| mb = 36 hours | $SG_1$ | 0.83 | 0.33 | 0.72 | 0.25 | 0.79 | 0.27 | 0.91 | 0.30 | 0.68 | 0.11 | 0.87 | **0.49** | 0.82 | 0.26 | 0.82 | 0.29 |
| | $SG_2$ | 0.81 | 0.03 | **1.00** | 0.04 | **1.00** | 0.02 | 0.74 | 0.05 | 1.00 | 0.01 | **0.94** | 0.07 | 0.73 | 0.03 | 0.85 | 0.04 |
| | $SG_3$ | 0.94 | 0.13 | 0.57 | 0.06 | 0.84 | 0.13 | 0.93 | 0.15 | 0.62 | 0.08 | 0.85 | 0.20 | 0.89 | 0.16 | 0.84 | 0.15 |
| | $SG_4$ | 0.87 | 0.23 | 0.71 | 0.14 | 0.82 | 0.15 | 0.85 | 0.37 | 0.60 | 0.08 | 0.87 | 0.44 | 0.91 | 0.21 | 0.84 | 0.24 |
| | $SG_5$ | **1.00** | 0.01 | 0.83 | 0.01 | 0.67 | 0.01 | 0.78 | 0.01 | **1.00** | 0.02 | 0.63 | 0.01 | 0.50 | 0.01 | 0.80 | 0.02 |
| | $SG_6$ | 0.69 | 0.44 | 0.58 | 0.24 | 0.68 | 0.20 | 0.77 | 0.33 | 0.64 | 0.11 | 0.85 | 0.46 | 0.70 | 0.33 | 0.72 | 0.31 |
| | $SG_7$ | 0.86 | 0.17 | 0.87 | 0.07 | 0.83 | 0.11 | **0.99** | 0.19 | 0.75 | 0.34 | 0.82 | 0.16 | **0.99** | 0.17 | 0.90 | 0.23 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.72 | **0.45** | 0.51 | **0.31** | 0.57 | **0.38** | 0.64 | **0.41** | 0.43 | **0.35** | 0.19 | 0.07 | 0.38 | **0.34** | | |
| mb = 48 hours | $SG_1$ | 0.82 | 0.27 | 0.71 | 0.12 | 0.83 | 0.19 | 0.94 | 0.22 | 0.71 | 0.09 | 0.84 | 0.29 | 0.90 | 0.19 | 0.84 | 0.20 |
| | $SG_2$ | **1.00** | 0.01 | 0.89 | 0.04 | 0.50 | 0.01 | **1.00** | 0.01 | 0.40 | 0.00 | 0.95 | 0.09 | 0.70 | 0.02 | 0.86 | 0.03 |
| | $SG_3$ | 0.89 | 0.22 | 0.65 | 0.06 | 0.85 | 0.17 | 0.93 | 0.21 | 0.74 | 0.08 | 0.87 | 0.29 | 0.98 | 0.23 | 0.86 | 0.18 |
| | $SG_4$ | 0.63 | 0.25 | 0.67 | 0.17 | 0.70 | 0.16 | 0.88 | **0.35** | 0.80 | 0.09 | 0.88 | 0.45 | 0.80 | 0.21 | 0.78 | 0.25 |
| | $SG_5$ | 0.82 | 0.02 | **1.00** | 0.15 | 0.90 | 0.03 | 0.91 | 0.03 | **1.00** | 0.02 | **1.00** | 0.01 | **1.00** | 0.02 | **0.93** | 0.02 |
| | $SG_6$ | 0.72 | 0.30 | 0.51 | **0.29** | 0.65 | 0.26 | 0.64 | 0.41 | 0.53 | 0.19 | 0.86 | **0.55** | 0.57 | **0.40** | 0.65 | 0.35 |
| | $SG_7$ | **1.00** | 0.00 | 0.00 | 0.00 | **1.00** | 0.03 | **1.00** | 0.04 | 0.00 | 0.00 | 0.86 | 0.01 | **1.00** | 0.05 | 0.98 | 0.04 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.69 | **0.438** | 0.71 | 0.23 | 0.66 | **0.43** | 0.73 | 0.11 | 0.57 | **0.38** | 0.58 | 0.21 | 0.77 | **0.41** | | |

**Table 4.11:** Performance of my Architecture across all sub-graphs ($SG_1$-$SG_7$), for forecasting horizon $\texttt{hz} = 12$ hrs and memory buffer $\texttt{mb} \in \{12, 24, 48\}$ hrs.

input buffer sizes can be considered as the best one globally, for all the sub-graphs and models. In addition, it is obvious that by modifying the number of the historical data could result in better results for each model not only on an individual level but also in their transfer-ability.

### 4.4.6 Hierarchical Model

The next step is to incorporate an ensemble learning module in the architecture. This Hierarchical module aims to combine the results of the previously trained and optimized model instances and essentially act as a voting system. The goal here is to create the final strong classifier which will benefit from the findings of all the weak classifiers, gaining an even better precision with some trade-off from recall. To this end, I aggregate the predictions produced by all models and validate them against one sub-graph at a time. Considering that each of the weak classifiers was trained on a different set of parameters as well as a different part of my original data, they will all have a significant contribution in the ensemble learning since they have adapted to learn and predict unique patterns.

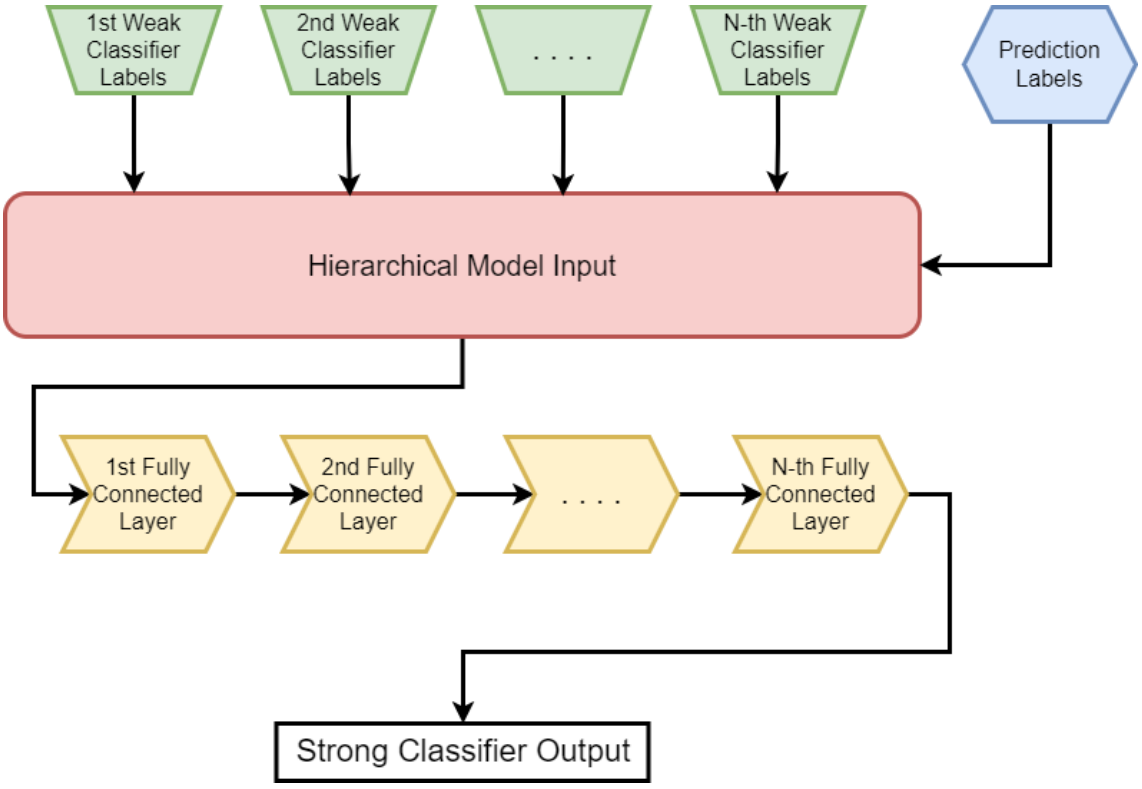


**Figure 4.6:** Hierarchical Model Architecture.

The architectural design of the Hierarchical model may vary depending on the forecasting horizon. The number of linear layers required in order to reach the desired outcome generally increases as the model tries to predict farther into the future. Once again, the Sigmoid activation function is applied at the end of the linear layers which provides each of the antennas' probability of becoming a hotspot.

The training for this module requires less epochs than the weaker classifiers, but also significantly less optimization. This time the optimization process is limited to simply selecting the proper learning rate for the model, optimize the training time and use the right amount of neurons on each linear layer. Even with the addition of this procedure, the entire pipeline is still well within the time constraints of the initial goal. The results of the Hierarchical model will be shown in the next section.

# 5 Results

## 5.1 Final Pipeline Representation

In the end, all the previously mentioned modules and steps are combined in order to form the final pipeline of the project as shown in Figures 5.1, 5.2. Initially, the information is drawn from each antenna individually depending on network traffic and is then passed through the mobile network's server to the Neural Network's pipeline. There, the data is processed. Once the data has been cleaned from any occurring duplicate data points and the missing values are imputed, the original graph is created followed by its partition into smaller sub-graphs. The data is then passed on to the Neural Network and the training process for each trainable sub-graph begins, optimizing the model's hyper-parameters until the results reach satisfying levels with primary target the precision and secondary the recall. After the complete training of all the weak classifiers, their output is passed down to the Hierarchical model (strong classifier) in order to combine the results. The trained and optimized strong classifier produces the final output for the cellular hotspot prediction.

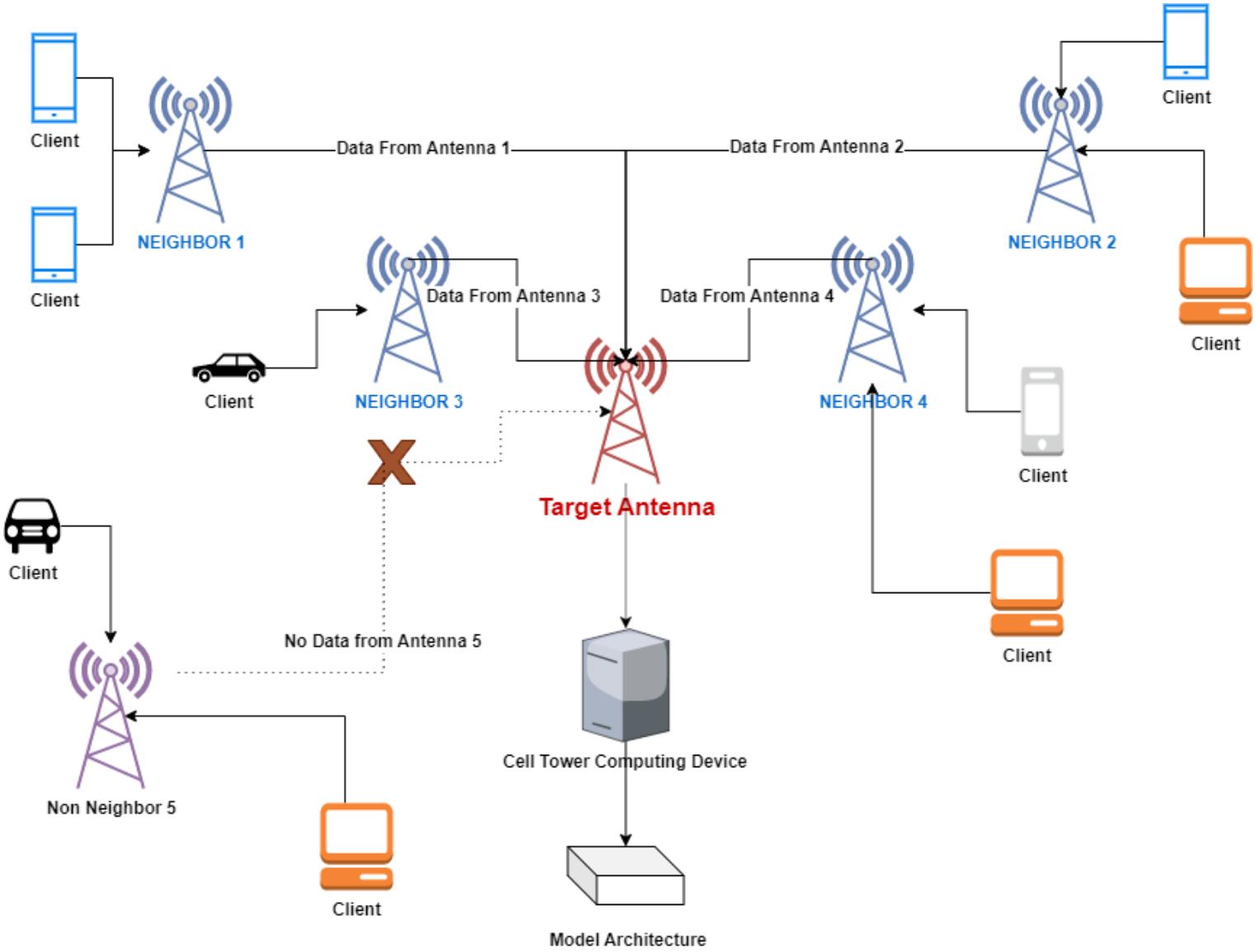


**Figure 5.1:** System Architecture.

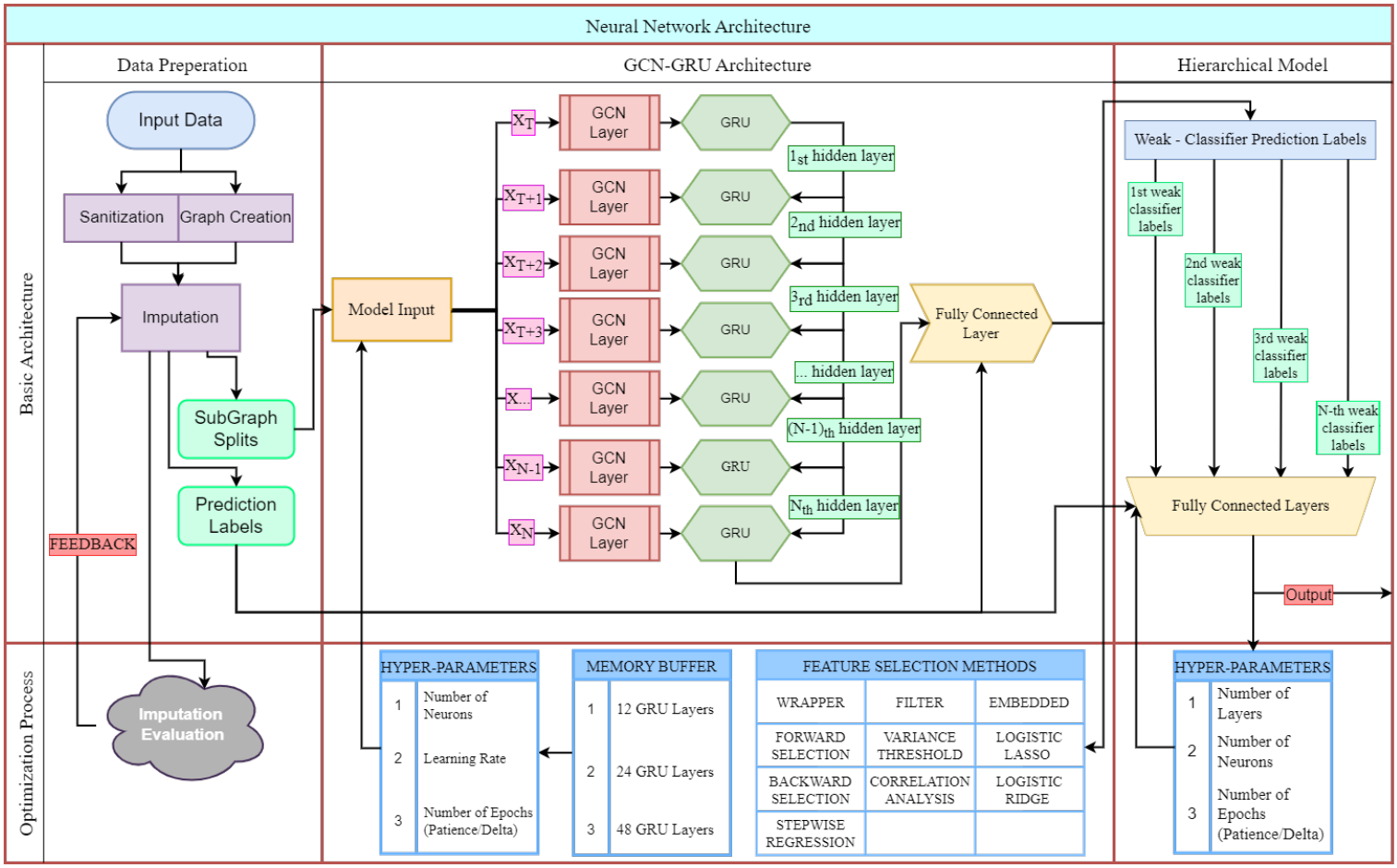


**Figure 5.2:** Pipeline Architecture.

## 5.2 Evaluation Results

In this section we will present the final results of the pipeline. First of all we present the weak classifier results for different forecasting horizons with various memory buffer sizes as shown in Table 5.1 , 5.2 and 5.3 as well as the performance of the baseline GCN and GRU architectures. The final output of the pipeline which is the result of the strong classifier provided by the Hierarchical Model is shown in Table 5.4.

| | | $SG_1$ | | $SG_2$ | | $SG_3$ | | $SG_4$ | | $SG_5$ | | $SG_6$ | | $SG_7$ | | $SG_{1-7}$ | $SG_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| mb = 12 hours | $SG_1$ | 0.82 | **0.31** | 0.76 | 0.15 | 0.86 | 0.27 | 0.83 | 0.23 | **0.79** | 0.08 | 0.88 | **0.44** | 0.84 | 0.30 | 0.84 | **0.25** |
| | $SG_2$ | **0.94** | 0.03 | **0.82** | 0.08 | **0.88** | 0.02 | 0.83 | 0.04 | 0.78 | 0.02 | **0.96** | 0.05 | 0.74 | 0.08 | 0.83 | 0.04 |
| | $SG_3$ | 0.89 | 0.18 | 0.76 | 0.05 | 0.80 | 0.19 | 0.85 | 0.24 | 0.54 | 0.05 | 0.83 | 0.25 | 0.82 | 0.19 | 0.82 | 0.17 |
| | $SG_4$ | 0.84 | 0.15 | 0.73 | **0.17** | 0.65 | 0.13 | 0.83 | 0.09 | 0.55 | 0.07 | 0.92 | 0.36 | 0.60 | 0.11 | 0.78 | 0.19 |
| | $SG_5$ | 0.76 | 0.18 | 0.45 | 0.08 | 0.76 | 0.17 | 0.70 | 0.21 | 0.58 | 0.11 | 0.61 | 0.08 | 0.71 | 0.19 | 0.67 | **0.25** |
| | $SG_6$ | 0.62 | 0.17 | 0.48 | 0.06 | 0.52 | 0.05 | 0.78 | 0.19 | 0.43 | 0.06 | 0.94 | 0.36 | 0.75 | 0.32 | 0.72 | 0.16 |
| | $SG_7$ | 0.87 | 0.09 | 0.81 | 0.04 | 0.84 | 0.12 | **0.86** | 0.15 | 0.50 | 0.02 | 0.87 | 0.23 | **0.91** | 0.16 | **0.86** | 0.12 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.61 | 0.13 | 0.56 | 0.17 | 0.54 | **0.40** | 0.67 | **0.48** | 0.46 | **0.41** | 0.73 | 0.18 | 0.54 | **0.47** | | |
| mb = 24 hours | $SG_1$ | 0.86 | 0.25 | 0.60 | 0.05 | 0.81 | 0.16 | 0.92 | 0.15 | 0.68 | 0.04 | 0.86 | 0.20 | 0.96 | 0.19 | 0.85 | 0.15 |
| | $SG_2$ | 0.67 | 0.00 | **1.00** | 0.04 | **0.83** | 0.03 | 0.83 | 0.01 | **1.00** | 0.01 | **1.00** | 0.02 | 0.00 | 0.00 | **0.91** | 0.01 |
| | $SG_3$ | 0.83 | 0.23 | 0.50 | 0.06 | 0.77 | 0.22 | 0.86 | 0.21 | 0.61 | 0.09 | 0.83 | 0.25 | 0.91 | 0.25 | 0.80 | 0.19 |
| | $SG_4$ | 0.74 | 0.24 | 0.63 | 0.21 | 0.71 | 0.16 | 0.83 | 0.34 | 0.66 | 0.10 | 0.79 | **0.43** | 0.69 | 0.16 | 0.75 | **0.24** |
| | $SG_5$ | **1.00** | 0.00 | 0.57 | 0.03 | 0.00 | 0.00 | 0.73 | 0.03 | 0.88 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.67 | 0.01 |
| | $SG_6$ | 0.62 | 0.11 | 0.88 | 0.07 | 0.80 | 0.10 | 0.87 | 0.13 | 0.49 | 0.04 | 0.95 | 0.33 | 0.98 | 0.19 | 0.83 | 0.14 |
| | $SG_7$ | **1.00** | 0.03 | 0.00 | 0.00 | **1.00** | 0.03 | 1.00 | 0.05 | 0.00 | 0.00 | 0.83 | 0.04 | **1.00** | 0.07 | 0.96 | 0.03 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.63 | **0.51** | 0.38 | **0.29** | 0.33 | **0.41** | 0.62 | **0.48** | 0.41 | **0.39** | 0.62 | 0.36 | 0.48 | **0.59** | | |
| mb = 36 hours | $SG_1$ | 0.83 | 0.33 | 0.72 | 0.25 | 0.79 | 0.27 | 0.91 | 0.30 | 0.68 | 0.11 | 0.87 | 0.49 | 0.82 | 0.26 | 0.82 | 0.29 |
| | $SG_2$ | 0.81 | 0.03 | **1.00** | 0.04 | **1.00** | 0.02 | 0.74 | 0.05 | 1.00 | 0.01 | **0.94** | 0.07 | 0.73 | 0.03 | 0.85 | 0.04 |
| | $SG_3$ | 0.94 | 0.13 | 0.57 | 0.06 | 0.84 | 0.13 | 0.93 | 0.15 | 0.62 | 0.08 | 0.85 | 0.20 | 0.89 | 0.16 | 0.84 | 0.15 |
| | $SG_4$ | 0.87 | 0.23 | 0.71 | 0.14 | 0.82 | 0.15 | 0.85 | 0.37 | 0.60 | 0.08 | 0.87 | 0.44 | 0.91 | 0.21 | 0.84 | 0.24 |
| | $SG_5$ | **1.00** | 0.01 | 0.83 | 0.01 | 0.67 | 0.01 | 0.78 | 0.01 | **1.00** | 0.02 | 0.63 | 0.01 | 0.50 | 0.01 | 0.80 | 0.02 |
| | $SG_6$ | 0.69 | 0.44 | 0.58 | 0.24 | 0.68 | 0.20 | 0.77 | 0.33 | 0.64 | 0.11 | 0.85 | **0.46** | 0.70 | 0.33 | 0.72 | **0.31** |
| | $SG_7$ | 0.86 | 0.17 | 0.87 | 0.07 | 0.83 | 0.11 | **0.99** | 0.19 | 0.75 | 0.34 | 0.82 | 0.16 | **0.99** | 0.17 | **0.90** | 0.23 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.72 | **0.45** | 0.51 | **0.31** | 0.57 | **0.38** | 0.64 | **0.41** | 0.43 | **0.35** | 0.19 | 0.07 | 0.38 | **0.34** | | |
| mb = 48 hours | $SG_1$ | 0.82 | 0.27 | 0.71 | 0.12 | 0.83 | 0.19 | 0.94 | 0.22 | 0.71 | 0.09 | 0.84 | 0.29 | 0.90 | 0.19 | 0.84 | 0.20 |
| | $SG_2$ | **1.00** | 0.01 | 0.89 | 0.04 | 0.50 | 0.01 | **1.00** | 0.01 | 0.40 | 0.00 | 0.95 | 0.09 | 0.70 | 0.02 | 0.86 | 0.03 |
| | $SG_3$ | 0.89 | 0.22 | 0.65 | 0.06 | 0.85 | 0.17 | 0.93 | 0.21 | 0.74 | 0.08 | 0.87 | 0.29 | 0.98 | 0.23 | 0.86 | 0.18 |
| | $SG_4$ | 0.63 | 0.25 | 0.67 | 0.17 | 0.70 | 0.16 | 0.88 | **0.35** | 0.80 | 0.09 | 0.88 | 0.45 | 0.80 | 0.21 | 0.78 | 0.25 |
| | $SG_5$ | 0.82 | 0.02 | **1.00** | 0.15 | 0.90 | 0.03 | 0.91 | 0.03 | **1.00** | 0.02 | **1.00** | 0.01 | **1.00** | 0.02 | **0.93** | 0.02 |
| | $SG_6$ | 0.72 | 0.30 | 0.51 | **0.29** | 0.65 | 0.26 | 0.64 | 0.41 | 0.53 | 0.19 | 0.86 | **0.55** | 0.57 | **0.40** | 0.65 | 0.35 |
| | $SG_7$ | **1.00** | 0.00 | 0.00 | 0.00 | **1.00** | 0.03 | **1.00** | 0.04 | 0.00 | 0.00 | 0.86 | 0.01 | **1.00** | 0.05 | 0.98 | 0.04 |
| | GCN | 0.37 | 0.10 | 0.00 | 0.00 | 0.42 | 0.12 | 0.26 | 0.02 | 0.00 | 0.00 | 0.00 | 0.00 | 0.19 | 0.08 | | |
| | GRU | 0.69 | **0.438** | 0.71 | 0.23 | 0.66 | **0.43** | 0.73 | 0.11 | 0.57 | **0.38** | 0.58 | 0.21 | 0.77 | **0.41** | | |

**Table 5.1:** Performance of my Architecture across all sub-graphs ($SG_1$-$SG_7$), for forecasting horizon $\texttt{hz} = 12$ hrs and memory buffer $\texttt{mb} \in \{12, 24, 36, 48\}$ hrs.

| | | $SG_1$ | | $SG_2$ | | $SG_3$ | | $SG_4$ | | $SG_5$ | | $SG_6$ | | $SG_7$ | | $SG_{1-7}$ | $SG_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| mb = 12 hours | $SG_1$ | 0.80 | 0.30 | 0.60 | 0.16 | 0.86 | 0.35 | 0.83 | 0.39 | 0.53 | 0.15 | 0.94 | 0.30 | 0.84 | 0.37 | 0.79 | **0.29** |
| | $SG_2$ | 0.00 | 0.00 | **0.77** | 0.02 | **1.00** | 0.01 | 0.84 | 0.06 | 0.27 | 0.01 | 0.00 | 0.00 | 0.64 | 0.02 | 0.70 | 0.02 |
| | $SG_3$ | **0.94** | 0.11 | 0.55 | 0.04 | 0.90 | 0.17 | 0.85 | 0.18 | 0.69 | 0.02 | 0.88 | 0.08 | 0.93 | 0.15 | **0.86** | 0.11 |
| | $SG_4$ | 0.83 | 0.28 | 0.58 | 0.11 | 0.92 | 0.22 | 0.87 | 0.33 | 0.50 | 0.04 | 0.95 | 0.21 | 0.89 | 0.28 | 0.83 | 0.22 |
| | $SG_5$ | 0.88 | 0.15 | 0.51 | 0.11 | 0.91 | 0.15 | 0.79 | 0.19 | **0.72** | 0.04 | 0.93 | 0.07 | **0.94** | 0.16 | 0.80 | 0.13 |
| | $SG_6$ | 0.78 | 0.18 | 0.58 | 0.16 | 0.82 | **0.36** | **0.88** | 0.33 | 0.61 | 0.19 | 0.89 | 0.32 | 0.84 | 0.29 | 0.79 | 0.26 |
| | $SG_7$ | 0.84 | 0.08 | 0.55 | 0.06 | 0.90 | 0.22 | 0.84 | 0.21 | 0.53 | 0.12 | 0.85 | 0.09 | 0.89 | 0.21 | 0.81 | 0.14 |
| | GCN | 0.67 | 0.26 | 0.66 | 0.05 | **1.00** | 0.00 | 0.79 | 0.36 | 0.00 | 0.00 | **1.00** | 0.03 | 0.76 | 0.07 | | |
| | GRU | 0.70 | **0.41** | 0.59 | **0.37** | 0.88 | 0.29 | 0.84 | **0.53** | 0.60 | **0.23** | 0.73 | **0.51** | 0.74 | **0.43** | | |
| mb = 24 hours | $SG_1$ | 0.83 | 0.33 | 0.70 | 0.10 | 0.83 | 0.34 | 0.86 | **0.36** | 0.60 | 0.10 | 0.94 | 0.25 | 0.89 | **0.37** | 0.84 | **0.27** |
| | $SG_2$ | 0.50 | 0.00 | **1.00** | 0.05 | 0.97 | 0.07 | 0.81 | 0.08 | 0.00 | 0.00 | 0.00 | 0.00 | **0.97** | 0.08 | **0.87** | 0.04 |
| | $SG_3$ | 0.86 | 0.27 | 0.68 | 0.07 | 0.85 | 0.33 | 0.83 | 0.32 | 0.63 | 0.11 | 0.85 | 0.20 | 0.92 | 0.29 | 0.83 | 0.24 |
| | $SG_4$ | 0.82 | 0.30 | 0.69 | 0.08 | 0.83 | 0.30 | **0.89** | 0.35 | 0.69 | 0.09 | 0.93 | 0.21 | 0.90 | 0.29 | 0.85 | 0.25 |
| | $SG_5$ | 0.85 | 0.24 | 0.60 | 0.09 | 0.91 | 0.21 | 0.86 | 0.24 | 0.60 | 0.04 | 0.93 | 0.07 | 0.93 | 0.20 | 0.83 | 0.17 |
| | $SG_6$ | **0.88** | 0.26 | 0.60 | 0.08 | 0.81 | 0.32 | 0.86 | 0.30 | **0.69** | 0.12 | 0.94 | 0.28 | 0.94 | 0.28 | 0.83 | 0.23 |
| | $SG_7$ | 0.87 | 0.27 | 0.69 | 0.09 | 0.85 | 0.32 | 0.85 | 0.33 | 0.58 | 0.11 | 0.90 | 0.24 | 0.88 | 0.34 | 0.83 | 0.25 |
| | GCN | 0.67 | 0.26 | 0.66 | 0.05 | **1.00** | 0.00 | 0.79 | **0.36** | 0.00 | 0.00 | **1.00** | 0.03 | 0.76 | 0.07 | | |
| | GRU | 0.70 | **0.59** | 0.58 | **0.36** | 0.86 | **0.35** | 0.79 | 0.22 | 0.65 | **0.18** | 0.74 | **0.53** | 0.69 | 0.24 | | |
| mb = 36 hours | $SG_1$ | 0.90 | 0.23 | 0.64 | 0.02 | 0.91 | 0.26 | 0.91 | 0.26 | 0.57 | 0.01 | 0.97 | 0.09 | **0.95** | 0.21 | **0.91** | 0.17 |
| | $SG_2$ | 0.58 | 0.02 | **0.86** | 0.10 | 0.96 | **0.05** | 0.70 | 0.12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.85 | 0.09 | 0.74 | 0.05 |
| | $SG_3$ | 0.87 | 0.01 | 0.67 | 0.01 | 0.82 | 0.22 | 0.82 | 0.22 | 0.33 | 0.00 | 0.67 | 0.06 | 0.91 | 0.18 | 0.84 | 0.15 |
| | $SG_4$ | 0.87 | 0.27 | 0.63 | 0.07 | 0.89 | 0.24 | **0.92** | **0.33** | 0.47 | 0.02 | 0.93 | 0.17 | 0.94 | 0.25 | 0.88 | 0.21 |
| | $SG_5$ | 0.80 | 0.10 | 0.51 | 0.10 | 0.92 | 0.16 | 0.80 | 0.18 | **0.73** | 0.04 | 0.92 | 0.06 | 0.93 | 0.15 | 0.79 | 0.13 |
| | $SG_6$ | 0.81 | 0.31 | 0.63 | 0.16 | 0.81 | 0.40 | 0.83 | 0.35 | 0.62 | 0.17 | 0.89 | **0.31** | 0.84 | 0.35 | 0.80 | **0.30** |
| | $SG_7$ | **0.93** | 0.15 | 0.55 | 0.04 | 0.90 | 0.25 | 0.89 | 0.25 | 0.57 | 0.04 | 0.83 | 0.08 | 0.89 | 0.20 | 0.86 | 0.15 |
| | GCN | 0.67 | 0.26 | 0.66 | 0.05 | **1.00** | 0.00 | 0.79 | 0.36 | 0.00 | 0.00 | **1.00** | 0.03 | 0.76 | 0.07 | | |
| | GRU | 0.80 | **0.39** | 0.71 | **0.13** | 0.84 | 0.19 | 0.79 | 0.54 | 0.63 | **0.20** | 0.77 | 0.43 | 0.80 | **0.39** | | |
| mb = 48 hours | $SG_1$ | 0.85 | 0.29 | 0.73 | 0.08 | 0.83 | 0.32 | 0.90 | 0.36 | 0.57 | 0.08 | 0.93 | 0.26 | 0.89 | 0.35 | 0.85 | 0.26 |
| | $SG_2$ | 0.50 | 0.01 | **1.00** | 0.03 | 0.96 | 0.06 | 0.78 | 0.05 | 0.00 | 0.00 | 0.00 | 0.00 | **0.93** | 0.07 | 0.84 | 0.03 |
| | $SG_3$ | 0.88 | 0.27 | 0.70 | 0.05 | 0.85 | 0.36 | 0.89 | 0.33 | 0.56 | 0.05 | 0.89 | 0.21 | 0.88 | 0.24 | 0.86 | 0.24 |
| | $SG_4$ | 0.85 | 0.28 | 0.70 | 0.08 | 0.84 | 0.29 | 0.91 | 0.35 | 0.60 | 0.06 | 0.95 | 0.25 | **0.93** | 0.33 | 0.87 | 0.24 |
| | $SG_5$ | 0.72 | 0.39 | 0.50 | 0.25 | 0.84 | 0.32 | 0.81 | 0.36 | 0.60 | 0.21 | 0.65 | 0.23 | 0.82 | 0.44 | 0.74 | 0.32 |
| | $SG_6$ | 0.73 | 0.50 | 0.56 | 0.32 | 0.76 | **0.50** | 0.79 | 0.44 | 0.61 | **0.27** | 0.86 | 0.41 | 0.81 | **0.56** | 0.76 | **0.43** |
| | $SG_7$ | **0.93** | 0.20 | 0.71 | 0.05 | 0.88 | 0.23 | **0.92** | 0.28 | 0.50 | 0.05 | 0.91 | 0.19 | 0.92 | 0.24 | **0.88** | 0.19 |
| | GCN | 0.67 | 0.26 | 0.66 | 0.05 | **1.00** | 0.00 | 0.79 | 0.36 | 0.00 | 0.00 | **1.00** | 0.03 | 0.76 | 0.07 | | |
| | GRU | 0.79 | **0.53** | 0.64 | **0.39** | 0.82 | 0.44 | 0.83 | **0.51** | **0.63** | 0.24 | 0.76 | **0.46** | 0.77 | 0.41 | | |

**Table 5.2:** Performance of my Architecture across all sub-graphs ($SG_1$-$SG_7$), for forecasting horizon $\texttt{hz} = 24$ hrs and memory buffer $\texttt{mb} \in \{12, 24, 36, 48\}$ hrs.

| | | $SG_1$ | | $SG_2$ | | $SG_3$ | | $SG_4$ | | $SG_5$ | | $SG_6$ | | $SG_7$ | | $SG_{1-7}$ | $SG_{1-7}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Avg Pre | Avg Rec |
| mb = 12 hours | $SG_1$ | 0.80 | **0.22** | 0.59 | **0.15** | 0.80 | 0.23 | 0.81 | **0.24** | 0.50 | 0.05 | 0.84 | 0.15 | 0.92 | 0.20 | 0.84 | **0.18** |
| | $SG_2$ | 0.00 | 0.00 | **1.00** | 0.09 | **1.00** | 0.04 | 0.81 | 0.03 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | **0.89** | 0.05 |
| | $SG_3$ | 0.93 | 0.13 | 0.54 | 0.03 | 0.89 | 0.13 | 0.85 | 0.16 | 0.61 | 0.01 | 0.81 | 0.02 | 0.93 | 0.10 | 0.85 | 0.09 |
| | $SG_4$ | 0.85 | 0.21 | 0.73 | 0.08 | 0.93 | 0.13 | 0.86 | 0.11 | 0.49 | 0.02 | **1.00** | 0.09 | 0.93 | 0.07 | 0.84 | 0.13 |
| | $SG_5$ | 0.85 | 0.06 | 0.74 | 0.04 | 0.84 | 0.05 | 0.70 | 0.05 | **0.71** | 0.06 | 0.75 | 0.03 | 0.91 | 0.06 | 0.77 | 0.06 |
| | $SG_6$ | **1.00** | 0.03 | 0.68 | 0.02 | 0.78 | 0.07 | **0.89** | 0.09 | 0.91 | 0.03 | **1.00** | 0.08 | 0.86 | 0.11 | 0.85 | 0.07 |
| | $SG_7$ | 0.80 | 0.03 | 0.54 | 0.02 | 0.86 | 0.13 | 0.80 | 0.19 | 0.69 | 0.05 | 0.84 | 0.08 | **0.96** | 0.17 | 0.84 | 0.08 |
| | GCN | 0.80 | 0.14 | 0.47 | 0.04 | 0.77 | 0.03 | 0.64 | 0.17 | 0.00 | 0.00 | 0.51 | 0.07 | 0.61 | 0.01 | | |
| | GRU | 0.80 | 0.19 | 0.38 | 0.08 | 0.71 | **0.15** | 0.70 | 0.17 | 0.53 | **0.14** | 0.66 | **0.20** | 0.59 | **0.23** | | |
| mb = 24 hours | $SG_1$ | 0.81 | 0.36 | 0.59 | 0.05 | 0.83 | 0.33 | 0.83 | **0.35** | 0.61 | 0.05 | 0.88 | 0.14 | 0.86 | **0.32** | 0.82 | **0.25** |
| | $SG_2$ | 0.71 | 0.02 | **1.00** | 0.04 | 0.87 | 0.11 | 0.77 | 0.03 | 0.72 | 0.03 | 0.82 | 0.01 | 0.92 | 0.06 | 0.81 | 0.06 |
| | $SG_3$ | 0.83 | 0.28 | 0.70 | 0.02 | 0.81 | **0.31** | 0.83 | 0.27 | 0.66 | 0.04 | **1.00** | 0.12 | 0.94 | 0.04 | 0.88 | 0.19 |
| | $SG_4$ | 0.81 | 0.21 | 0.84 | 0.05 | **0.92** | 0.14 | **0.94** | 0.27 | 0.64 | 0.03 | **1.00** | 0.12 | 0.95 | 0.04 | **0.92** | 0.13 |
| | $SG_5$ | 0.81 | 0.21 | 0.70 | **0.12** | 0.86 | 0.16 | 0.81 | 0.21 | **0.90** | 0.13 | 0.87 | 0.16 | 0.88 | 0.29 | 0.81 | 0.20 |
| | $SG_6$ | 0.83 | **0.31** | 0.71 | 0.06 | 0.88 | 0.29 | 0.80 | 0.29 | 0.74 | 0.08 | 0.96 | **0.22** | 0.95 | 0.24 | 0.85 | 0.22 |
| | $SG_7$ | **0.92** | 0.20 | 0.65 | 0.05 | 0.88 | 0.24 | 0.82 | 0.28 | 0.68 | **0.23** | 0.95 | 0.10 | **1.00** | 0.23 | 0.88 | 0.17 |
| | GCN | 0.81 | 0.11 | 0.40 | 0.05 | 0.70 | 0.00 | 0.60 | 0.22 | 0.00 | 0.00 | 0.00 | 0.00 | 0.63 | 0.02 | | |
| | GRU | 0.78 | 0.20 | 0.40 | 0.22 | 0.70 | 0.11 | 0.74 | 0.09 | 0.63 | 0.07 | 0.70 | 0.17 | 0.61 | 0.19 | | |
| mb = 36 hours | $SG_1$ | 0.85 | 0.18 | 0.65 | 0.01 | 0.89 | 0.19 | 0.88 | 0.17 | 0.77 | 0.01 | 0.90 | 0.08 | 0.90 | 0.18 | 0.88 | 0.14 |
| | $SG_2$ | 0.68 | 0.01 | **1.00** | 0.07 | 0.90 | **0.28** | 0.72 | 0.07 | 0.00 | 0.00 | 0.80 | 0.03 | 0.85 | 0.04 | 0.81 | 0.10 |
| | $SG_3$ | 0.85 | 0.02 | 0.77 | 0.03 | 0.80 | 0.17 | 0.79 | 0.16 | 0.77 | 0.01 | 0.80 | 0.05 | 0.92 | 0.15 | 0.80 | 0.11 |
| | $SG_4$ | **0.90** | 0.22 | 0.76 | 0.04 | **0.91** | 0.18 | **1.00** | 0.26 | 0.69 | 0.01 | **0.96** | 0.13 | 0.98 | 0.21 | **0.90** | 0.14 |
| | $SG_5$ | 0.80 | 0.21 | 0.60 | 0.11 | 0.88 | 0.16 | 0.84 | 0.21 | **0.91** | 0.04 | 0.86 | 0.15 | 0.80 | 0.21 | 0.81 | 0.16 |
| | $SG_6$ | 0.80 | **0.35** | 0.68 | 0.20 | 0.88 | 0.28 | 0.81 | **0.31** | 0.71 | **0.13** | 0.87 | **0.31** | 0.86 | 0.32 | 0.82 | **0.28** |
| | $SG_7$ | 0.84 | 0.05 | 0.70 | 0.06 | 0.89 | 0.18 | 0.81 | 0.16 | 0.75 | 0.08 | 0.88 | 0.04 | **0.99** | 0.25 | 0.84 | 0.11 |
| | GCN | 0.76 | 0.14 | 0.41 | 0.03 | 0.64 | 0.01 | 0.61 | 0.09 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | | |
| | GRU | 0.79 | 0.22 | 0.59 | **0.30** | 0.73 | 0.04 | 0.00 | 0.00 | 0.00 | 0.00 | 0.68 | 0.12 | 0.64 | 0.20 | | |
| mb = 48 hours | $SG_1$ | 0.79 | 0.16 | 0.70 | 0.02 | 0.84 | 0.21 | 0.93 | 0.20 | 0.68 | 0.02 | 0.89 | 0.18 | 0.83 | 0.14 | 0.80 | 0.11 |
| | $SG_2$ | 0.00 | 0.00 | **1.00** | 0.02 | **1.00** | 0.03 | 1.00 | 0.01 | 0.00 | 0.00 | 0.00 | 0.00 | 0.67 | 0.03 | 0.91 | 0.02 |
| | $SG_3$ | 0.84 | **0.23** | 0.73 | 0.01 | 0.81 | 0.32 | 0.84 | 0.27 | 0.66 | 0.03 | 0.90 | 0.14 | 0.89 | 0.17 | 0.81 | 0.16 |
| | $SG_4$ | 0.87 | 0.20 | 0.74 | **0.05** | 0.83 | 0.22 | 0.92 | **0.29** | 0.66 | 0.03 | **0.92** | **0.23** | 0.90 | **0.30** | 0.84 | **0.20** |
| | $SG_5$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.80 | 0.06 | 0.89 | 0.09 | **0.90** | 0.18 | 0.00 | 0.00 | 0.00 | 0.00 | 0.83 | 0.10 |
| | $SG_6$ | 0.80 | 0.14 | 0.79 | 0.05 | 0.80 | **0.29** | 0.81 | 0.25 | 0.71 | **0.20** | 0.90 | 0.19 | 0.87 | 0.24 | 0.81 | 0.19 |
| | $SG_7$ | **1.00** | 0.16 | 0.78 | 0.03 | 0.91 | 0.15 | **1.00** | 0.23 | 0.00 | 0.00 | 0.89 | 0.13 | **1.00** | 0.21 | **0.92** | 0.16 |
| | GCN | 0.62 | 0.03 | 0.00 | 0.00 | 0.50 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | | |
| | GRU | 0.77 | 0.14 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.65 | 0.15 | 0.69 | 0.17 | | |

**Table 5.3:** Performance of my Architecture across all sub-graphs ($SG_1$-$SG_7$), for forecasting horizon $\texttt{hz} = 48$ hrs and memory buffer $\texttt{mb} \in \{12, 24, 36, 48\}$ hrs.

| | mb = 12h | | mb = 24h | | mb = 36h | | mb = 48h | |
|---|---|---|---|---|---|---|---|---|
| | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec |
| $SG_1$ | 1.00 | 0.13 | 1.00 | 0.12 | 0.95 | 0.23 | 0.90 | 0.21 |
| $SG_2$ | 0.49 | 0.09 | 0.77 | 0.08 | 0.89 | 0.10 | 1.00 | 0.04 |
| $SG_3$ | 1.00 | 0.16 | 1.00 | 0.05 | 1.00 | 0.11 | 0.88 | 0.11 |
| $SG_4$ | 1.00 | 0.10 | 1.00 | 0.02 | 1.00 | 0.21 | 1.00 | 0.15 |
| $SG_5$ | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| $SG_6$ | 1.00 | 0.02 | 1.00 | 0.07 | 0.94 | 0.28 | 1.00 | 0.26 |
| $SG_7$ | 1.00 | 0.13 | 1.00 | 0.10 | 1.00 | 0.16 | 1.00 | 0.13 |

**Table 5.4:** Performance of the Hierarchical Model for $SG_{1-7}$, for forecasting horizon `hz`=12 hrs and memory buffer `mb` $\in \{12, 24, 36, 48\}$ hrs based on aggregated results across all seven sub-graphs.

The results from the weak classifiers, prove that the model's transferability is, for the most part, steady across all tested memory buffer sizes and forecasting horizons. However, no particular pattern can be detected in the behavior of the model. One conclusion to be made is that with a slight majority, most use-cases are performing better if the forecasting horizon does not exceed the memory buffer in its input. This was to be expected as the farther into the future the model is trying to predict, the more information it should require. Another observation is that in the forecasting horizon `hz = 48 hrs` there is a noticeable and steady drop in the classifier's recall metrics, as it is getting harder for the classifier to detect patterns. As demonstrated in Table 4.4, the RNN module has significantly larger contribution to the results than its GCN counterpart. Stretching the horizon means that the temporal dependencies will become scarcer and hence the less desirable outcome.

Regarding model transfer-ability, I can now investigate how much it can be influenced by the sub-graph similarity values shown in Table 4.9. For this research, two new variables are introduced, the Precision and Recall Ratios. These ratios will be calculated for each model instance, in order to estimate the difference of performance that the each model has as it's being applied to different sub-graphs. Then, the Spearman correlation can be applied for the Precision and Recall ratios against the similarity values in order to verify if this coefficient is statistically significant. The results used for this test are non-optimized

and the output will be expressed in the one-tailed p-value. The different instances of the model have been named alphabetically.

| A | | B | | C | | D | | E | | F | | G | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec | Pre | Rec |
| 0.14 | 0.01 | 0.11 | 0.13 | 0.03 | 0.07 | 0.01 | 0.09 | 0.17 | 0.19 | 0.12 | 0.18 | 0.11 | 0.15 |

**Table 5.5:** Spearman Correlation Results.

It is known theory that if a p-value is below 0.1 it is considered statistically significant and if it is below 0.05 it is highly statistically significant. Therefore, as shown in the results of Table 5.5, in several cases it is proven that having a low transfer-ability may have severe impact in the model's performance.

The conclusion drawn by the strong classifier results, is that in almost all cases the target performance metric which is the precision of the model is maximized. There are cases however where the strong classifier fails to produce satisfactory results. Specifically, in the case of the sub-graph $SG_5$ we can see that the results are all zeroes. This can be explained by the fact that in the results of the weak classifiers regarding this sub-graph, even though the model achieved a high precision, the recall of the model was significantly low. It was low enough that the input passed down to the Hierarchical Model was extremely biased towards the negative cases making it hard for the classifier to detect patterns. This can also be explained via the similarity table 4.9, where it is demonstrated that this particular sub-graph is the most dissimilar compared to the others.

In cases where the strong classifier fails, the pipeline will select the results of the weak classifier, in order to maintain stable performance across all sub-graphs without having inconsistencies.

# 6 Conclusions

## 6.1 Thesis Conclusion

This thesis presents a Neural Network architecture which is capable of early detection of cellular hotspot cases by taking advantage of the spatio-temporal dynamics exhibited by the the dataset consisting of the cellular KPIs. The proposed architecture with ensemble learning and the implementation of sub-graphs has the benefits of a timely executed training process with large amounts of data, and the target performance metrics of the model are higher than any other baseline method it has been compared to, or other existing solutions regarding this matter. The model has the capacity of forecasting on different prediction horizons while maintaining its high accuracy and precision, and in the meantime be almost invariant to the amount of input historical data that is fed in its input.

Furthermore, it is shown that the model's attribute of transfer-ability through different graphs is relatively high considering the fact that the implementation makes use of Graph Convolution Neural Networks. This has the benefit of training and evaluating the model locally in targeted areas of interest without having the need of using the entire set of data which would significantly rise the time complexity. With ensemble learning, incorporating an Hierarchical Model at the end of the pipeline, the Network has the potential of maximizing its precision metric with a small trade-off from recall by combining the results provided by all the weak classifiers.

Even in some presented cases where the strong classifier fails to produce a satisfying output in terms of precision, the pipeline will choose the weak classifier instead which yields results always withing the acceptable ranges. The final results provided show that regardless of the sub-graph, the memory buffer size, or the prediction horizon, the pipeline is capable of accomplishing the initial goal.

## 6.2 Future Work

In future work, the implementation of a better graph partitioning algorithm is suggested, one that could potentially create the sub-graphs in a way that their similarities could reach minimum values, meaning that the sub-graphs would be near identical. This is crucial since the Graph Convolution Layers are known to make calculations by using the structural information of the graph, which concludes that the less similar the sub-graphs are, the less transfer-ability the model shall have.

In addition, the model's performance could increase by increasing the quality of the input data. The mean imputation though efficient, is not the most accurate algorithm for data imputation. In the last few years, more advanced techniques have been developed such as the Edge-To-Edge Generative Adversarial Network (E2GAN) from [30] or the Bidirectional Recurrent Imputation for Time-Series (BRITS) proposed in [31].

There might also be ways of improving the Neural Network's performance with a few changes in its architecture. Instead of implementing an RNN to capture temporal dependencies, it is possible to apply temporal convolutions. Based on their theory, temporal convolutions have the ability to learn temporal patterns significantly faster than any RNN and with a much larger receptive field. If dilation is applied, that receptive field could become even larger, which counters not only exploding/vanishing gradient problems for large historical data, but also requires much less memory.

Finally, as discussed in previous sectors, this pipeline allows for the application of hardware processing devices which would allow the network companies to speed up the inference phase of the Neural Network while also running it on edge. The idea would be to incorporate a device

# Bibliography


[1] Stefan Rüping. Learning interpretable models. 10 2006.

[2] Liping Xie, Zilong Li, Yihan Zhou, Yiliu He, and Jiaxin Zhu. Computational diagnostic techniques for electrocardiogram signal analysis. *Sensors*, 11 2020.

[3] N. Johnson, P. Vulimiri, A. To, X. Zhang, C. Brice, Branden Kappes, and Aaron Stebner. Machine learning for materials developments in metals additive manufacturing. 05 2020.

[4] Divish Rengasamy, Mina Jafari, Benjmain Rothwell, Xin Chen, and Grazziela Figueredo. Deep learning with dynamically weighted loss function for sensor-based prognostics and health management. *Sensors*, 01 2020.

[5] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, 9, 04 2022.

[6] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Van Essen, Abdul Awwal, and Vijayan Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8:292, 03 2019.

[7] Yi Zhou. *Sentiment classifcation with deep neural networks*. PhD thesis, 07 2019.

[8] Hisham ElMoaqet, Mohammad Eid, Martin Glos, Mutaz Ryalat, and Thomas Penzel. Deep recurrent neural networks for automatic detection of sleep apnea from single channel respiration signals. *Sensors (Basel, Switzerland)*, 20, 09 2020.

[9] Iram Bibi, Adnan Akhunzada, Jahanzaib Malik, Javed Iqbal, Arslan Musaddiq, and Sung Kim. A dynamic dl-driven architecture

to combat sophisticated android malware. *IEEE Access*, PP:1–1, 07 2020.

[10] Max Roser, Hannah Ritchie, and Esteban Ortiz-Ospina. Internet. *Our World in Data*, 2015. https://ourworldindata.org/internet.

[11] Nipun Ramakrishnan and Tarun Soni. Network traffic prediction using recurrent neural networks. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 187–193, 2018.

[12] Jiandong Bai, Jiawei Zhu, Yujiao Song, Ling Zhao, Zhixiang Hou, Ronghua Du, and Haifeng Li. A3t-gcn: Attention temporal graph convolutional network for traffic forecasting. *ISPRS International Journal of Geo-Information*, 10(7), 2021.

[13] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 47–54, Cham, 2016. Springer International Publishing.

[14] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. pages 3634–3640, 07 2018.

[15] Joan Serrà, Ilias Leontiadis, Alexandros Karatzoglou, and Konstantina Papagiannaki. Hot or not? forecasting cellular network hot spots using sector performance indicators. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 259–270, 2017.

[16] Lixia Zhou, Xia Chen, Runsha Dong, and Shan Yang. Hotspots prediction based on lstm neural network for cellular networks. *Journal of Physics: Conference Series*, 1624:052016, 10 2020.

[17] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[18] Y. Bengio and Yann Lecun. Convolutional networks for images, speech, and time-series. 11 1997.

[19] David E. Rumelhart and James L. McClelland. *Learning Internal Representations by Error Propagation*, pages 318–362. 1987.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[21] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[22] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for earning in raph domains. volume 2, pages 729 – 734 vol. 2, 01 2005.

[23] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

[24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3844–3852, Red Hook, NY, USA, 2016. Curran Associates Inc.

[25] Bo Jiang, Ziyan Zhang, Doudou Lin, Jin Tang, and Bin Luo. Semi-supervised learning with graph learning-convolutional networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11305–11312, 2019.

[26] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[27] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *6th International Conference on Learning Representations*, 2017.

[28] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.

[29] Danai Koutra, Ankur P. Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. 2011.

[30] Yonghong Luo, Ying Zhang, Xiangrui Cai, and Xiaojie Yuan. E2gan: End-to-end generative adversarial network for multivariate time series imputation. IJCAI'19, page 3094–3100. AAAI Press, 2019.

[31] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. Brits: Bidirectional recurrent imputation for time series. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.