TECHNICAL UNIVERSITY OF CRETE
DIPLOMA THESIS

Study of a Rotationally Invariant Hardware Implementable Convolutional Neural Network using CORDIC Arithmetic

---

Author:
Sotirios MICHAIL

Thesis Committee:
Prof. Apostolos DOLLAS
Assoc. Prof. Sotirios IOANNIDIS
Prof. Michail ZERVAKIS

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

March 1, 2024

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Study of a Rotationally Invariant Hardware Implementable Convolutional Neural Network using CORDIC Arithmetic

by Sotirios MICHAIL

Introduced in this thesis is an approach in enhancing the rotational invariance of Convolutional Neural Networks (CNNs), through integrating the novel Log-CORDIC algorithm for image pre-processing. This image pre-processing algorithm presents an advantage over existing cartesian-to-polar transform algorithms for images, through the computational advantages of the Coordinate Rotation Digital Computer (CORDIC) algorithm. The results of the novel algorithm are studied and compared with existing transform methods, along with its efficiency improvements, and its ability to enhance rotational invariance in a CNN is ascertained by integrating it into the pipeline of a customized SqueezeNet neural network. Focusing on the CIFAR-10 and MNIST datasets, experiments with this customized SqueezeNet neural network demonstrate an improvement in classification accuracy for images with varied orientations.

Acknowledgements

I would like to express my deepest gratitude to my professor and supervisor of this work, Professor Apostolos Dollas, for their invaluable guidance, patience, and insightful critiques throughout the course of this research. Their expertise and thoughtful advice have been instrumental in shaping both the direction and success of this work.

My heartfelt thanks also go to my family, who have provided me with unwavering support and encouragement throughout my academic journey. Their belief in me and constant encouragement have been a source of strength and motivation. I am deeply grateful for their sacrifices and for always inspiring me to pursue my passions.

I would also like to extend my appreciation to my colleagues and friends in the Technical University of Crete, who have contributed to this journey through stimulating discussions, constructive feedback, and by providing a supportive academic environment. Their perspectives and companionship have been invaluable.

I would like to especially thank the members of the Microprocessor and Hardware Lab, Mr. Kimionis Markos and Associate Professor Sotirios Ioannidis, for their immense support and guidance during my academic journey and beyond.

This thesis would not have been possible without the contributions and support of each one of these individuals. I am immensely grateful to all of you.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ALU	**A**rithmetic **L**ogic **U**nit
ASIC	**A**pplication **S**pecific **I**ntegrated **C**ircuit
BRAM	**B**lock **R**andom **A**ccess **M**emory
CPU	**C**entral **P**rocessor **U**nit
CS	**C**omputer **S**cience
DDR4	**D**ouble **D**ata **R**ate type texbf4 memory
DRAM	**D**ynamic **R**andom **A**ccess **M**emory
DSP	**D**igital **S**ignal **P**rocessor
FF	**F**lip **F**lops
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
GDDR6	**G**raphics **D**ouble **D**ata **R**ate type **6** memory
GPU	**G**raphic **P**rocessor **U**nit
HBM	**H**igh **B**andwidth **M**emory
HDL	**H**ardware **D**escription **L**anguage
HLS	**H**igh **L**evel **S**ynthesis
HPC	**H**ight **P**erformance **C**omputing
LUT	**L**ook **U**p **T**able
MPSoC	**M**ulti **P**rocessor **S**ystem **o**n **C**hip
PL	**P**rogrammable **L**ogic
PS	**P**rocessing **S**ystem
RAM	**R**andom **A**ccess **M**emory
SDK	**S**oftware **D**evelopment **K**it
SIMD	**S**ingle **I**nstruction **M**ultiple **D**ata
SSE	**S**treaming **S**IMD **E**xtensions
SSD	**S**olid **S**tate **D**rive
TDP	**T**hermal **D**esign **P**ower
URAM	**U**ltra **R**andom **A**ccess **M**emory
USD	**U**nited **S**tates **D**ollar

Dedicated to my family and friends…

Chapter 1

Introduction

1.1 Motivation

The quest for rotational invariance in image classification models, particularly Convolutional Neural Networks (CNNs), stems from a fundamental challenge in computer vision: the variability in object orientations in real-world images. Traditional CNNs, despite their impressive performance, often falter when confronted with images of rotated objects, a scenario frequently encountered in practical applications. This limitation is particularly pronounced in fields such as autonomous navigation, satellite imagery, and medical diagnostics, where the orientation of objects or features can be arbitrary and unpredictable. The inability to accurately classify rotated images can significantly hinder the reliability and applicability of automated systems in these critical areas.

Recognizing this challenge, the central motivation of this thesis is to enhance the rotational invariance of CNNs, enabling them to maintain high classification accuracy regardless of object orientation. The proposed solution involves a novel approach: transforming images into their polar coordinate representations using the CORDIC algorithm before feeding them into CNNs. This transformation effectively normalizes rotational variations, allowing the CNN to focus on the intrinsic features of the objects rather than their orientation in the Cartesian plane. The CORDIC algorithm, known for its computational efficiency, is particularly well-suited for this task, providing a method to perform the polar transformation rapidly and accurately.

The significance of training CNNs with polar-transformed images lies in its potential to improve the robustness of image classification models against rotational variations. By addressing the rotational invariance challenge, this

thesis aims to contribute to the development of more versatile and dependable computer vision systems, capable of operating effectively in diverse and uncontrolled environments.

1.2 Scientific Contributions

The primary contribution of this research is the development and implementation of an algorithm that significantly improves the transformation of images from cartesian to polar representation, with a goal to enhance the rotational invariance of CNNs. By integrating the CORDIC algorithm for polar transformation of input images, the thesis demonstrates a novel approach to preprocessing images for CNNs. This method normalizes rotational variations, enabling CNNs to focus on the inherent features of objects rather than their orientation in space. The result is an increase in the efficiency of the transformation of images from cartesian to polar representation. Another of this work is the empirical validation of the proposed methodology through experimental testing and benchmarking. By employing well-established datasets like CIFAR-10 and MNIST, modified to include rotated images, the research provides evidence of the improved accuracy achieved by CNNs trained with polar-transformed images.

1.3 Thesis Outline

- **Chapter 2 - Theoretical Background:** Described in detail is the theoretical background for the concepts and technologies this thesis pertains, namely Machine Learning, CNNs, polar coordinate systems, FPGAs and the CORDIC algorithm
- **Chapter 3 - Related Work:** Academic and scientific work related to image classification and image processing
- **Chapter 4 - The Log-CORDIC Transform:** The core part of this thesis and the means of achieving rotational invariance
- **Chapter 5 - System Model:** Modelling the rotationally invariant image classification system in software form
- **Chapter 6 - Proposed Architecture:** A proposed hardware implementation of the system

- **Chapter 7 - Conclusions and Future Work:** Discussing the overall results of the thesis and how it can be further advanced

Chapter 2

Theoretical Background

2.1 Machine Learning

The quest for creating intelligent systems that can interpret and understand visual information as humans and animals do has been a long-standing pursuit in the realm of artificial intelligence. The human visual system, an epitome of high-level image processing, possesses the ability to rapidly identify patterns, detect objects, and perceive depth from an array of complex visual stimuli. Similarly, a bird, soaring high in the sky, can discern a tiny fish in a vast ocean, embodying a perfect model of adaptive image processing. The ambition of the research presented in this thesis is to develop a system inspired by these remarkable biological systems that can likewise identify and track an object in the same real time fashion, with high accuracy and reliability, regardless of the situation and environment variables.

This theoretical background chapters lays the foundation upon which the rest of this work lies. It provides an in-depth exploration of the core principles and mechanisms underpinning convolutional neural networks (CNNs), a class of deep learning models that have achieved ground-breaking performance in the field of image processing. We delve into the intricate structure and functionality of CNNs, including convolutional layers, pooling layers, and activation functions, illustrating how they collectively enable the efficient extraction and interpretation of hierarchical features within images. This understanding is crucial, not only for appreciating the existing capabilities of CNNs in image processing but to also give the basis to understand the developments and research presented in this work.

2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have been instrumental in a myriad of applications that involve real-time data and image processing. CNNs' unique architecture, combining convolutional layers for local feature extraction, pooling layers for dimensionality reduction, and fully connected layers for high-level reasoning, enables them to handle large volumes of data efficiently. [1] CNNs' ability to learn complex patterns in data makes them highly effective for a wide range of tasks in various sectors, from healthcare and autonomous vehicles to entertainment and e-commerce.

In healthcare, CNNs have been extensively applied to analyze medical images in real-time, aiding in diagnostics and treatment planning. For instance, they have been used for detecting tumors in MRI scans, identifying anomalies in X-ray images, and classifying skin lesions in dermoscopic images. [2] In autonomous vehicles, CNNs contribute to real-time object detection and scene understanding, critical for safe navigation. They analyze the visual data captured by vehicle sensors to identify pedestrians, other vehicles, traffic signs, and the drivable path. [3]

In the entertainment industry, CNNs have been used for real-time image and video processing tasks, such as enhancing image quality, colorizing black and white images, and generating animations from sketches. [4] In the realm of e-commerce, they aid in real-time product recommendation systems by analyzing user behavior and product images, thereby improving user engagement and sales. [5]

The power and flexibility of CNNs have led to several variations designed to enhance performance and broaden their applicability. LeNet [6], proposed by Yann LeCun in 1998, was one of the first CNN architectures, predominantly used for handwritten and machine-printed character recognition. AlexNet [7], proposed by Alex Krizhevsky in 2012, was instrumental in advancing CNNs' application in large-scale image classification tasks. VGGNet [8], known for its homogeneous architecture with repeated blocks of convolutional and pooling layers, and ResNet [9], featuring skip connections or shortcut connections to tackle the problem of vanishing gradients in deep networks, are other notable variations. These different architectures have evolved to address specific challenges and application domains, showcasing the versatility and potential of CNNs.

Despite their remarkable success in various applications, face several challenges and limitations. One of the primary concerns is the computational complexity and resource consumption, especially in deep architectures that require substantial memory and processing power. This can hinder real-time applications and deployment on devices with limited resources. Another challenge is the susceptibility to adversarial attacks, where slight perturbations in the input can lead to incorrect predictions. CNNs also often require large amounts of labeled data for training, making them less suitable for tasks with scarce or imbalanced data. The interpretability of CNNs remains a significant issue, as understanding the reasoning behind their decisions is often complex and non-intuitive. This lack of transparency can be a barrier in critical applications such as healthcare, where interpretability is essential for trust and compliance. Furthermore, CNNs may exhibit bias based on the data they are trained on, leading to unfair or skewed predictions. Addressing these challenges requires ongoing research and innovation in areas such as model optimization, robustness, interpretability, and fairness, to fully realize the potential of CNNs across diverse domains.

2.3 Structure of a Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep learning model primarily used for processing structured grid data such as images. The fundamental structure of a CNN is typically composed of a sequence of three types of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layers, often considered the core building block of a CNN, perform a mathematical operation known as convolution that extracts high-level features from the input data. Following the convolutional layers, pooling or subsampling layers are used to reduce the spatial dimensions of the data, thereby controlling overfitting and reducing computational complexity. After several alternations of convolutional and pooling layers, the high-level reasoning in the neural network is done via fully connected layers. To introduce non-linearity into the network and enhance learning capabilities, an activation function is applied to the output of each layer. Common activation functions include Perceptron, Rectified Linear Unit (ReLU), and Leaky ReLU. This hierarchical model design allows CNNs to learn complex patterns in data, making them highly effective for image processing tasks.

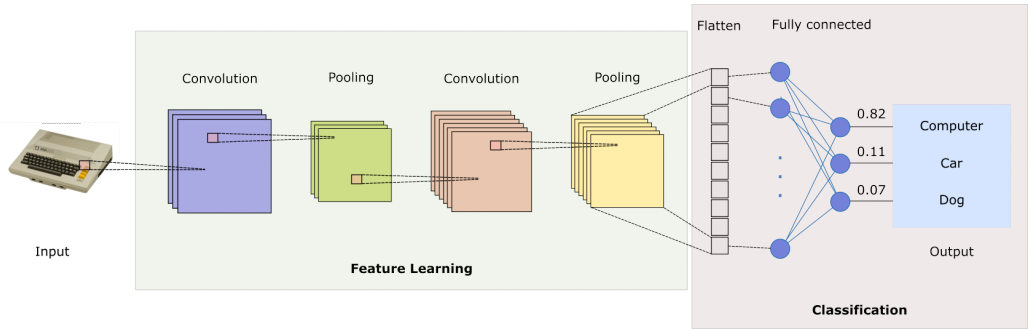

FIGURE 2.1: High-level Architecture of a CNN Source: [10]

2.3.1 Convolution Layers

Convolutional layers are an essential part of the convolutional neural network, and they play a crucial role in the CNN's ability to effectively process spatial and temporal data. [1] Each convolutional layer in a CNN comprises of numerous filters, also known as kernels. These kernels serve as the primary feature detectors within the network. The key operation in a convolutional layer is the convolution operation, a mathematical operation that involves two functions to generate a third function. [11] In the context of a CNN, the convolution operation is applied to the input data using the kernel.

The kernels move across the input data, usually in a sliding window fashion, and at each step, the kernel performs a dot product operation between its weights and the corresponding input values. This operation essentially multiplies the weights of the filter with the input data and then sums them to produce a single value. This value is then placed in a new matrix known as the feature map or convolved feature. The resulting feature map represents the locations and strength at which the filter's feature was detected in the input.

One key aspect of convolutional layers that distinguishes them from fully connected layers in other types of neural networks is the principle of local receptive fields. This principle means that each neuron in the convolutional layer is connected only to a small region of the input volume, as opposed to being connected to all the neurons in the previous layer. This concept is closely related to the idea of 'sparse connectivity', which suggests that by making connections only to a localized region of the input, the network can focus on low-level features such as edges or textures.

A vital advantage of convolutional layers is the concept of parameter sharing. [12] In this, the same filter (with the same weights) is used across the entire input, drastically reducing the number of parameters to be learned, enhancing computational efficiency, and making the model more scalable. This parameter sharing also allows convolutional networks to exhibit 'translation invariance', meaning they can recognize features irrespective of their location in the input, which is especially valuable in image and speech recognition tasks.

Furthermore, multiple filters in each convolutional layer enable the network to detect various features within the same layer, thereby allowing the network to learn a diverse set of features. [13] As the network gets deeper, these layers can capture more abstract and high-level features. This hierarchical feature learning is one of the key reasons behind the impressive performance of CNNs in various complex tasks.

2.3.2 Pooling Layers

Pooling layers, also known as subsampling or downsampling layers, serve a critical function in the architecture of a convolutional neural network. The primary purpose of these layers is to progressively reduce the spatial size of the input representation, thereby decreasing the amount of parameters and computations within the network. This reduction aids in controlling overfitting, which refers to the model's tendency to perform exceptionally well on training data but poorly on unseen or test data. [14] Pooling layers achieve this by summarizing the outputs of groups of neurons in the previous layer, essentially providing an abstracted form of the input.

Two common types of pooling operations are max pooling and average pooling. Max pooling operates by selecting the maximum value from each window of a given size on the input. This operation has been found to perform well in practice, likely due to its ability to preserve the strongest feature response within the window, which often corresponds to the presence of a particular feature. [15] Average pooling, on the other hand, calculates the average of all values within the window. This operation provides an overall summary of the feature responses in the window but may dilute the impact of strong feature responses. The selection between max pooling and average pooling often depends on the specific application and dataset at hand. Regardless of the type, pooling layers play a significant role in making the decision function more robust to variations in the input. [16]

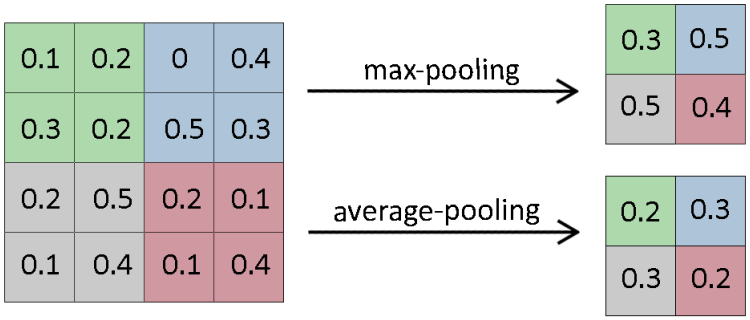

FIGURE 2.2: Pooling layer type comparison

2.3.3 Activation Functions

Activation functions are a pivotal aspect of convolutional neural networks, as they introduce non-linearity into the model. Without these functions, no matter how many layers the neural network has, it would still behave as a linear model, limiting its capability to learn from complex data and to perform complex tasks. These functions are applied to the output of each neuron in the network, thus determining whether and to what extent that neuron should be activated, i.e., transmit its signal to the subsequent layer. The choice of activation function can significantly influence the network's learning speed and performance.

The purpose of an activation function is to map the input onto a desired range, often introducing non-linearity in the process. This characteristic allows the network to model more complex relationships between its inputs and outputs, and to learn from errors through the gradient descent optimization process. The function's non-linearity is crucial for the learning process, as it enables the network to backpropagate errors, a procedure that adjusts the weights and biases to minimize the loss function.

Perceptron

The Perceptron is a type of artificial neuron that uses a step activation function. It takes a weighted sum of the input features and adds a bias. The result is then passed through the step function. If the resultant value exceeds a certain threshold, the perceptron fires, which in the context of a binary classification problem signifies classifying the input as the positive class. Otherwise, it remains inactive. The perceptron forms the foundation for more

complex neural networks, yet it has limitations in solving problems that are not linearly separable.

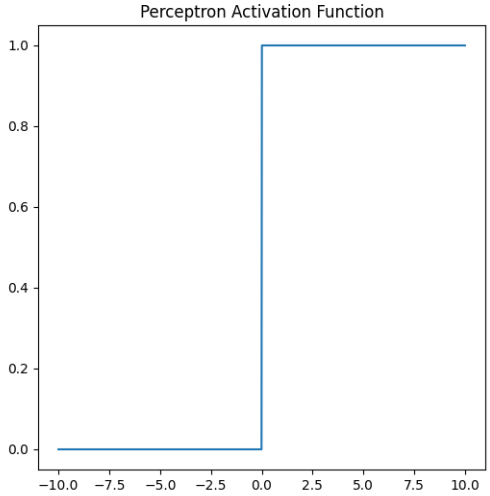

FIGURE 2.3: The Perceptron activation function

Rectified Linear Unit

ReLU, or Rectified Linear Unit, is currently one of the most widely used activation functions in CNNs and deep learning. It operates by returning the input directly if it is positive, else it returns zero. The simplicity of this function allows for faster computational time. Moreover, ReLU helps mitigate the vanishing gradient problem, a situation where the gradient becomes so small that the weights and biases of the network cease to learn effectively. However, a limitation of ReLU is that it can cause dead neurons, i.e., neurons that output zero for any input, thereby obstructing the learning process.

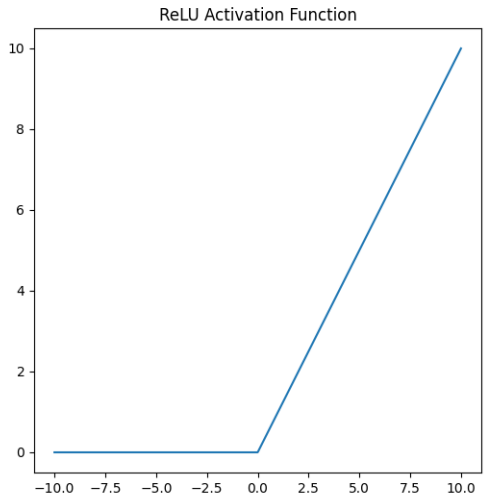

FIGURE 2.4: The ReLU activation function

"Leaky" Rectified Linear Unit

Leaky ReLU is a variant of ReLU that was proposed to fix the "dying ReLU" problem. Instead of having a zero gradient for negative input values as in the ReLU function, Leaky ReLU allows small negative values when the input is less than zero. By doing this, it keeps the gradient from becoming zero for negative input values, thus preserving the ability of these neurons to learn, even for inputs that induce a negative pre-activation value. Despite these benefits, choosing between ReLU and Leaky ReLU often depends on the specific problem and the dataset at hand.

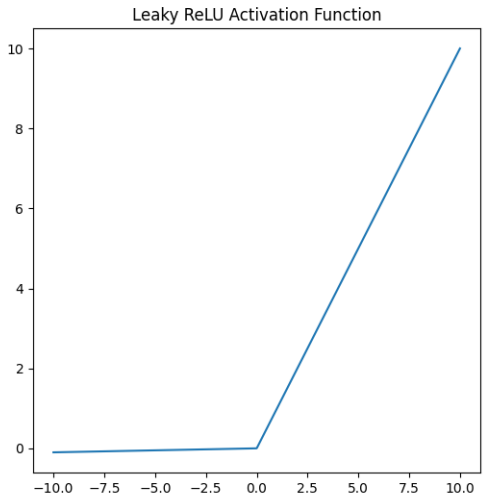

FIGURE 2.5: The Leaky ReLU activation function

2.4 Polar Coordinate Representation

Polar coordinate representation of images is an essential tool in image processing and computer vision, especially beneficial in applications where rotational or radial symmetry is present, such as biomedical imaging or in satellite imaging. This system provides a different perspective on the image data by representing the image coordinates not in terms of the traditional Cartesian coordinate system (x, y), but instead in terms of radial distance and angular position (r, θ).

The transformation from Cartesian to polar coordinates is achieved through the following mathematical relationships:

$$r = \sqrt{x^2 + y^2} \tag{2.1}$$

$$\theta = \arctan\left(\frac{y}{x}\right) \tag{2.2}$$

where (x, y) are the Cartesian coordinates of a point in the image, and (r, θ) are the corresponding polar coordinates. 'r' represents the distance from the origin (typically the center of the image) to the point, and 'θ' is the angle from the positive x-axis to the point in the counter-clockwise direction.

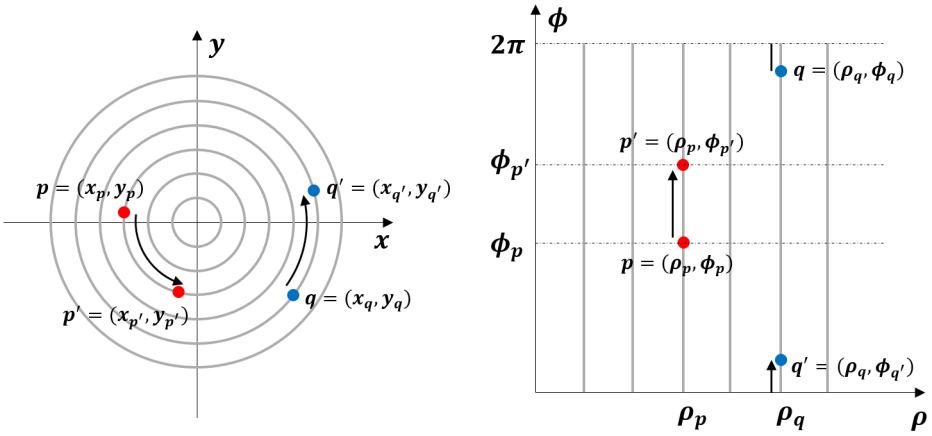

FIGURE 2.6: Converting an image from Cartesian to polar representation. The concentric circles in the Cartesian representation are mapped to vertical lines in the polar representation Source: [17]

In the context of digital images, which are discretely sampled, the conversion to polar coordinates involves a resampling process. This can be a simple nearest-neighbor resampling or involve more complex interpolation techniques, such as bilinear or bicubic interpolation, to better preserve the image details.

There are several benefits of using polar representation. For instance, rotation of the object in the image corresponds to a simple shift in the polar coordinate representation. Similarly, scaling corresponds to a stretching or compression along the radial axis. Therefore, geometric transformations become more intuitive in this representation.

However, care must be taken when using polar coordinate representation of images, as distortions may occur, particularly towards the outer regions of the image due to the larger spacing of polar coordinates compared to Cartesian coordinates. This can lead to a decrease in resolution at the edges of the image.

Despite these challenges, the polar coordinate system offers a powerful way to analyze and process images, especially in scenarios where the natural symmetry of the underlying scene or object can be exploited. It is a valuable tool in the arsenal of techniques for image processing and computer vision.

2.5 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are a class of integrated circuits (ICs) that can be programmed or reprogrammed to the desired functionality after manufacturing. They provide a unique blend of flexibility and performance that bridges the gap between general-purpose processors and application-specific integrated circuits (ASICs) [18].

FPGAs consist of an array of programmable logic blocks and a hierarchy of reconfigurable interconnects that allow these blocks to be wired together to perform complex combinational functions, or merely simple logic gates like AND, OR, and XOR. Logic blocks can be programmed to perform the function of basic logic gates, or they can be configured to perform complex combinational functions such as decoders or mathematical functions. In addition to the logic blocks, FPGAs also contain memory elements, which may include flip-flops and more complex memory blocks called embedded block RAM (EBRAM).

The design and programming of FPGAs involve the use of Hardware Description Languages (HDLs), such as VHDL or Verilog, to describe the desired digital logic in text form [19]. This code is then synthesized into a configuration file (bitstream) by specific software tools provided by the FPGA vendor. The bitstream is then loaded onto the FPGA, configuring the device to perform the specified logic functions.

One of the primary advantages of FPGAs is their parallel processing capability, which can lead to significantly higher performance than processors for certain types of applications. This attribute makes them particularly valuable for real-time image processing applications, which often require the processing of large amounts of data in parallel. Furthermore, FPGAs offer the ability to be reprogrammed to suit evolving needs, offering flexibility not found in traditional ASIC designs. However, designing with FPGAs requires a strong understanding of digital logic design principles, and the development of efficient designs can be a complex task.

This modular parallelism that FPGAs offer makes them highly suitable for an embedded image processing system that is based on CNNs[20]. The various parts and layers of a CNN that were described above, can be implemented in such manner in order to exploit the flexibility and parallelism that an FPGA can offer [21].

An image processing system such as the one implemented in the scope of this work, not only has to be effective in its task of identifying and tracking the target objects, but it has to do so in a constrained environment, where energy efficiency is of critical importance, such as an autonomous road vehicle, an unmanned aerial vehicle (UAV) or a surveillance system.

2.6 The CORDIC Algorithm

The COordinate Rotation DIgital Computer (CORDIC) algorithm, which was introduced by Jack E. Volder in 1959, is a computationally efficient method designed to calculate trigonometric and hyperbolic functions, among other mathematical operations [22]. At its core, CORDIC operates by iteratively rotating a vector to achieve the desired result, using only shifts and adds, which makes it particularly suitable for hardware implementations. The algorithm's iterative nature allows for a trade-off between the number of iterations and the desired accuracy, making it adaptable to various application requirements.

For Field-Programmable Gate Arrays (FPGAs), the CORDIC algorithm offers a compelling advantage due to its inherent parallelism and simplicity in hardware design [23]. Traditional methods for computing trigonometric functions often involve multipliers, which consume significant FPGA resources. In contrast, CORDIC's reliance on simpler operations like shifts and adds makes it resource-efficient, leading to faster computations and reduced power consumption. This efficiency is especially beneficial for real-time applications where rapid computations are crucial.

In the realm of Convolutional Neural Networks (CNNs), the potential integration of CORDIC can be seen in the context of optimizing certain operations, especially when CNNs are deployed on FPGAs [24]. Given that CNNs involve numerous mathematical computations, any optimization in these operations can lead to significant improvements in performance. Furthermore, the CORDIC algorithm finds applications in various domains, from digital signal processing and communication systems to computer graphics and robotics, showcasing its versatility and enduring relevance in the world of computational mathematics [25].

2.7 SqueezeNet: An Image Classification Model

Image classification is a fundamental task in the field of computer vision, where the goal is to categorize images into predefined classes. This task has been revolutionized by the advent of deep learning models, particularly Convolutional Neural Networks (CNNs). CNNs are specialized kinds of neural networks that are adept at processing data with a grid-like topology, such as images. They consist of one or more convolutional layers, often accompanied by pooling layers, fully connected layers, and normalization layers. These models have significantly improved the accuracy of image classification tasks by effectively learning hierarchical representations of image data, extracting features from low-level details such as edges and colors to high-level attributes like shapes and object identities.

2.7.1 Introduction to SqueezeNet

SqueezeNet is a distinctive CNN architecture that achieves high accuracy in image classification tasks with a significantly reduced model size. Developed by researchers at DeepScale, UC Berkeley, and Stanford University, SqueezeNet was designed to provide AlexNet-level accuracy with fewer parameters and a smaller memory footprint [26]. This aspect of SqueezeNet makes it particularly advantageous for deployment in environments with limited computational resources, such as embedded systems or mobile devices.

2.7.2 Architecture of SqueezeNet

The key to SqueezeNet's efficiency lies in its innovative architectural design, which centers around the 'Fire' module [26]. Each Fire module consists of a 'squeeze' layer, which uses 1x1 filters to reduce the depth of the input data, followed by an 'expand' layer that combines 1x1 and 3x3 filters. The squeeze layer significantly reduces the number of input channels (or depth) to the expand layer, decreasing the computational complexity and the number of parameters in the network. Furthermore, SqueezeNet employs delayed downsampling, meaning it reduces the spatial dimensions of the data at a later stage in the network. This approach allows the network to retain more fine-grained spatial information in the early layers, contributing to better feature extraction and overall accuracy.

2.7.3 Efficiency and Effectiveness of SqueezeNet

SqueezeNet's architectural innovations result in a model that is both computationally efficient and effective in performance. The reduction in model size and parameters directly translates to less memory usage and faster computation, which is crucial for real-time applications. Despite its compact size, SqueezeNet does not compromise on accuracy, maintaining performance levels comparable to larger models like AlexNet [26]. This balance between size, speed, and accuracy makes SqueezeNet an ideal choice for applications where resources are constrained but high performance is required.

2.7.4 SqueezeNet and FPGA Compatibility

The architecture of SqueezeNet is inherently well-suited for FPGA implementation, a factor that significantly influenced its selection in this project. FPGAs are known for their ability to efficiently handle parallel computations, making them ideal for deploying convolutional neural networks [27]. SqueezeNet's streamlined architecture, characterized by its small model size and reduced parameter count, aligns well with the resource constraints typical of FPGA devices. The low memory footprint of SqueezeNet mitigates the limited on-chip memory challenge often encountered in FPGA implementations. This compactness allows for the entire neural network model to be loaded onto the FPGA, minimizing the need for external memory access which can be a bottleneck in terms of speed.

To optimize SqueezeNet for FPGA deployment, certain adaptations are made to its architecture. The unique design of the Fire module, which combines 1x1 and 3x3 convolutions, lends itself to parallel processing, a strength of FPGAs. By restructuring these convolutional layers to exploit the parallelism capabilities of the FPGA, a significant increase in computational speed can be achieved. Additionally, quantization techniques are applied to the network weights and activations to fit the precision constraints of FPGA hardware, further enhancing the execution speed while maintaining model accuracy[21]. The flexibility of the FPGA also allows for custom hardware logic to be designed specifically for SqueezeNet's architecture, enabling more efficient data flow and processing than what is possible in general-purpose processors.

The adaptability of SqueezeNet to FPGA implementation is a key factor in its effectiveness for real-time image classification applications. Its small size,

combined with the ability to tailor its architecture to the parallel processing strengths of FPGAs, results in a powerful and efficient system. This compatibility is especially beneficial for edge computing applications where speed, accuracy, and efficiency are crucial.

Chapter 3

Related Work

3.1 Rotational invariance for CNNs

Worrall et al. [28] present a novel Convolutional Neural Network (CNN) architecture known as the Harmonic Network (HarmNet). This unique approach involves the utilization of circular harmonics, making the HarmNet equivariant to both rotation and translation. Their innovative idea was to represent images in polar coordinates, which, in turn, enabled the easier portrayal of rotations and translations in the harmonic domain. Their experimental results on the rotated MNIST dataset demonstrate that the HarmNet outperforms standard CNNs, showcasing the potential advantages of incorporating rotational invariance into CNN architectures. Kim et. al [17] introduce CyCNN, a CNN model designed to be invariant to rotational transformations in image classification tasks. The main novelty of CyCNN lies in the implementation of polar mapping of input images, which essentially translates rotational changes into translational ones. Furthermore, the model introduces cylindrical convolutional layers, which leverage a cylindrical sliding window mechanism, adapting to the unique cylindrical attributes of polar coordinates. The authors report that their model, when trained without data augmentation, significantly outperforms standard CNNs on rotated versions of the MNIST, CIFAR-10, and SVHN datasets. This work stands as an exemplar of algorithmic modifications enhancing rotational invariance in CNNs.

Veeling et. al [29] propose and implement a rotation-equivariant CNN specifically to enhance histopathology image classification tasks. The authors extend the functionality of CNNs through the incorporation of Group Equivariant Convolutional Networks (G-CNNs), thus instilling rotation equivariance into the model. The rotation equivariance significantly enhanced the model's

resilience against rotations. This study underscores the importance of rotational invariance in domains such as digital pathology, where orientation of microscopic tissue images can vary significantly. Further research and work into enhancing CNNs for medical applications, is explored in the work of Togo et al. [30]. The authors explore the use of deep convolutional neural network (DCNN)-based features to differentiate cardiac sarcoidosis (CS) from non-CS using polar maps. Analyzing 85 patients, the study constructs polar maps from PET/CT images of the left ventricle region and extracts high-level features through the Inception-v3 network. The study introduces the ReliefF algorithm and compares the DCNN-based classification method with SUVmax and CoV-based methods. The results demonstrate that DCNN-based high-level features extracted from polar maps are more effective than conventional quantitative analysis methods for CS classification. Another study in this field is the one by Sofian et al. [31] where the authors investigate the detection of calcification in intravascular ultrasound images using convolutional neural network architectures. The study compares three types of CNN architectures and seven types of classifiers using two types of images: Cartesian Coordinates images and polar reconstructed coordinate images. The results show that classifiers such as Support Vector Machine, Discriminant analysis, Ensembles, and Error-Correcting Output Codes obtained perfect results using polar reconstructed coordinate images for InceptionresNet-V2 architecture. The study emphasizes the effectiveness of polar coordinates in detecting calcification.

Cohen and Welling [32] propose a method to ensure CNNs are equivariant under rotations by proposing Group Equivariant Convolutional Networks (G-CNNs). This principled extension of CNNs retains a significantly larger group of symmetries, including rotation symmetry. This work laid the foundation for many subsequent studies in the field of rotationally equivariant or invariant neural networks. RotNet, by Johnson et al. [33], a novel approach that applies Convolutional Neural Networks (CNNs) to estimate stellar rotation periods from Kepler light curves. By converting the time-series data into image form and employing a ResNet-18 architecture through transfer learning, the authors have created a method that outperforms existing techniques, including the Auto-Correlation Function (ACF), which is a widely used standard. Notably, RotNet achieves greater accuracy with significantly less computational time, being 350 times quicker than ACF for the same data size and 10,000 times quicker for larger datasets. The success of RotNet underscores the potential of utilizing deep learning for stellar parameter estimation and

sets a new benchmark for efficiency and precision in determining stellar rotation periods.

3.2 Optimizing CNNs for FPGAs

Umuroglu et al. [34] present the FINN framework, a ground-breaking solution designed for creating high-performance accelerators for Binarized Neural Networks (BNNs) on Field-Programmable Gate Arrays (FPGAs), which are a specialized type of neural network where the weights and activations are binarized to -1 and +1. This binarization drastically simplifies the computational and memory requirements, thus making BNNs particularly suited to FPGA deployments that may have limited resources. The FINN framework proposes a flexible, FPGA-tailored design flow that can achieve high throughput even on low-cost devices, underlining the compelling potential of BNNs for FPGA-based acceleration.

An LSTM RNN implementation on FPGA is proposed by Han et al. [35] While the work does not strictly fall within the domain of Convolutional Neural Networks, it provides invaluable insights into the optimization of neural network architectures on FPGAs. The authors highlight the significance of network sparsity for augmenting computational and energy efficiency in FPGA-based neural network implementations. They achieve this sparsity by employing an Efficient Inference Engine (EIE) that prunes neurons with negligible effects on the final network accuracy. The study underscores how neural network sparsity can be leveraged for FPGA acceleration, offering wider implications for CNN architectures. Guan et al. [36] present another LSTM RNN accelerator design optimized for FPGAs. The authors propose various optimization techniques, including a novel partitioning method, to reduce the memory footprint and enhance computational efficiency. The FPGA-tailored design strategies adopted by Li and colleagues are evaluated using a Xilinx Zynq ZC706 Evaluation Kit, demonstrating significant performance gains over the baseline designs. While this study primarily focuses on LSTMs, the architectural optimizations and design principles can be informative for the design of FPGA-accelerated CNNs as well.

Suda et al. [37] introduce an OpenCL-based FPGA accelerator for large-scale CNNs. This work utilizes a roofline model to guide the design and optimization of their accelerator, aiming to achieve the highest throughput. They

provide a comprehensive performance evaluation showing how their accelerator design outperforms existing OpenCL-based FPGA designs for large-scale CNNs. The use of OpenCL in this context illustrates the potential of high-level synthesis tools in creating efficient FPGA designs for CNNs. Another efficient hardware implementation method for optical remote sensing object detection using CNNs on FPGAs is presented by Zhang et al. [38]. The authors optimize the CNN model for hardware implementation, providing a foundation for efficiently mapping the network on an FPGA. They propose a hardware architecture that includes a general processing engine to implement multiple types of convolutions using a uniform module. An efficient data storage and access scheme is also introduced, achieving low-latency calculations and high memory bandwidth utilization. The implementation on a Xilinx ZYNQ xc7z035 FPGA demonstrates competitive performance with GPUs in terms of mean average precision (mAP) and significant advantages in energy efficiency.

The research study of Zhuge et al. [39] explores different fast convolution algorithms, including Winograd and Fast Fourier Transform (FFT), to find an optimal strategy for applying them to different types of convolutions on FPGAs. The paper also proposes an optimization scheme to exploit parallelism in novel CNN architectures like Inception modules in GoogLeNet. The implementation on a Xilinx Ultrascale device achieves a significant latency speedup compared to high-end NVIDIA GPUs, surpassing previous FPGA results.

Han et al. [40] present and implement CNN-MERP, an FPGA-based processor designed to address the external memory bandwidth bottleneck in large-scale deep CNNs. The paper introduces an efficient memory hierarchy that significantly reduces bandwidth requirements through multiple optimizations, including on/off-chip data allocation, data flow optimization, and data reuse. The proposed 2-level reconfigurability enables fast and efficient reconfiguration, resulting in a 55 percent reduction in bandwidth requirements and a 5.48 times higher throughput compared to state-of-the-art FPGA implementations.

3.3 Using CORDIC on FPGAs

Bonabi et al. [41] explore techniques for implementing a Hodgkin-Huxley-based neural network on an FPGA. The complexity of the model poses challenges to network size and execution speed. The authors utilize the CORDIC algorithm, along with step-by-step integration, to implement arithmetic circuits. They also employ resource-sharing techniques to preserve model details while increasing network size and maintaining near real-time execution speed. The implementation provides a foundation for constructing large FPGA-based network models to study various neurophysiological mechanisms, making it suitable for neural control of cognitive robots and systems. S. D. Muñoz and J. Hormigo [42] present a hardware design for high throughput QR decomposition using the Givens rotation method on FPGAs. The design employs a 2-D systolic array architecture with pipelined processing elements based on the CORDIC algorithm. The approach allows continuous computation of QR factorizations with simple hardware, optimizing a fixed-point FPGA architecture for 4x4 matrices. The design achieves at least 50 percent more throughput and much less resource utilization compared to previous FPGA proposals.

A low-cost sequential and high-performance architecture for implementing the CORDIC algorithm on FPGAs for fingerprint recognition is presented by Neji et al. [43]. The design employs radix-2 arithmetic and is suited for serial operation, performing conversions between polar and rectangular coordinate systems. The paper presents a VHDL description of the CORDIC algorithm and introduces combinatory blocks to reduce iteration delay. The architecture is implemented and tested, demonstrating the design flow and accuracy of the CORDIC algorithm. In their work, the Francheschi et al. [44] present a case study of applying Inexact Speculative Adders (ISA) to the FPGA implementation of a CORDIC module within a tactile data processing system. The design focuses on low latency, low power consumption, and reduced hardware complexity for robotic and biomedical applications. The implementation on a Xilinx ZYNQ-7000 ZC702 device shows dynamic power reduction up to 40 percent and delay latency reduction up to 21 percent compared to a conventional CORDIC module, with a negligible average relative error for sine and cosine computations.

Pardo, Boluda and Sosa innovatively employ the CORDIC algorithm, primarily leveraging its vector mode, to facilitate a computationally efficient and precise transformation process. A specialized hardware architecture is

proposed, optimized to bolster the performance of the Log-CORDIC algorithm, ensuring enhanced computational throughput and reduced latency. Through rigorous empirical evaluations, the proposed approach demonstrates remarkable superiority over existing transformation methods in terms of accuracy and computational efficiency, thus offering significant advancements in the realms of image processing and computer vision, especially in applications necessitating rotational and scaling invariance.

3.4 The FPGA Perspective

The development of a rotationally invariant image detection system for Field Programmable Gate Arrays (FPGAs) is of significant interest for several reasons. Primarily, FPGAs offer a unique combination of high performance and flexibility, making them suitable for real-time image processing applications where latency is critical. These qualities would further be enhanced by a system capable of retaining its accuracy irrespective of the rotational orientation of the input images, broadening its application scope.

The inherent parallelism of FPGAs allows for simultaneous processing of multiple image segments, offering speed advantages over sequential processors. Implementing a rotationally invariant CNN on such a platform would enable rapid processing of image data, regardless of its orientation. This would be particularly beneficial in fields such as autonomous vehicles and drones, or satellite imagery where the orientation of captured images can be arbitrary and can vary dynamically.

Furthermore, the reprogrammable nature of FPGAs allows for continual system updates and modifications to adapt to evolving needs and advancements in neural network architectures. In a rapidly evolving field like machine learning, this adaptability is particularly beneficial. A rotationally invariant image detection system on FPGA would not only provide robust and fast image recognition capabilities but could also readily incorporate future improvements in rotational invariance techniques, thereby ensuring longevity and relevance in a rapidly progressing technological landscape.

Chapter 4

The Log-CORDIC Transform

4.1 Introduction

The Log-CORDIC transform is the cornerstone of our approach in achieving rotational invariance in a convolutional neural network. The CORDIC algorithm [22], [45], is traditionally used in signal processing systems in order to calculate trigonometric parameters efficiently. Utilizing it to transform an image from a cartesian to a polar representation has only really been explored once before [46], thus making this work a novel approach in the realm of image processing and neural networks.

4.2 The CORDIC Algorithm

The most basic form of the CORDIC algorithm, as described herein, suffices for our purposes. Let (x_0, y_0) denote the Cartesian coordinates of a point before rotation, and (x_n, y_n) represent the coordinates after rotation by an angle θ. The transformation equations are given by:

$$x_n = x_0 \cos\theta - y_0 \sin\theta$$
$$y_n = y_0 \cos\theta + x_0 \sin\theta$$

These equations can be rearranged into a more convenient form:

$$x_n = \cos\theta(x_0 - y_0 \tan\theta)$$
$$y_n = \cos\theta(y_0 + x_0 \tan\theta)$$

When the rotation angles are constrained such that $\tan\theta = \frac{1}{2^i}$ (where $i = 0, 1, \ldots, n$), the multiplication by the tangent term simplifies to a simple shift

operation. Utilizing this property, any rotation can be decomposed into a series of smaller rotations, each following this restriction. In this scenario, any angle rotation operation can be executed through a series of shift and addition operations, along with a final multiplication (due to the $\cos\theta$ factor). The i-th stage of this series follows the equations:

$$x_{i+1} = K_i(x_i - y_i d_i 2^{-i})$$
$$y_{i+1} = K_i(y_i + x_i d_i 2^{-i})$$

where $K_i = \cos\theta_i = \cos\left(\arctan\frac{1}{2^i}\right) = \sqrt{1+2^{-2i}}$, and $d_i = \pm 1$. All K_i factors are constants that can be applied at the conclusion of the rotation operation. The final product term can be viewed as a gain of the circuit. When this CORDIC operation is integrated into a larger circuit, it is possible to incorporate this multiplicative factor into subsequent operations for simplification. This holds particularly true for the LOG-CORDIC transformation circuit, where this factor has been entirely canceled out, as demonstrated later. The gain factor A_n is the product of the inverses of all K_i.

The factor d_i determines the direction of the corresponding elementary i rotation. This sign on each stage depends on the angle to be shifted and aims to approach the result to the global angle at each stage. A convenient method for safely converging to the final angle involves defining a difference variable that indicates the proximity to the target angle. This variable is also useful for determining the value for d_i:

$$z_{i+1} = z_i - d_i \arctan 2^{-i}$$

$$d_i = \begin{cases} -1 & \text{if } z_i < 0 \\ +1 & \text{if } z_i \geq 0 \end{cases}$$

The above equations, without the K_i constants, constitute the CORDIC equations in rotation mode. The only operations required for any calculation are additions and shifts, as the $\arctan 2^{-i}$ values are precalculated and stored in a table. These equations are useful for computing trigonometric functions. Given (x_0, y_0, z_0) as the initial values, the CORDIC rotator in rotation mode yields the following results:

$$
\begin{aligned}
x_n &= A_n(x_0 \cos z_0 - y_0 \sin z_0) \\
y_n &= A_n(y_0 \cos z_0 + x_0 \sin z_0) \\
z_n &= 0
\end{aligned}
$$

The CORDIC algorithm in rotation mode facilitates sine and cosine operations by providing the angle z_0 and appropriate values for x_0 and y_0. For instance, the sine can be calculated by setting $x_0 = 1$ and $y_0 = 0$, with the result obtained in y_n.

Another common mode for the CORDIC rotator is the vector mode. In this mode, the objective is to minimize the y_n term rather than the z_n angle. This simple modification results in a rotation until the rotated point aligns with the x-axis, with the rotated angle denoted as θ and the resulting x_n represented as r in polar coordinates:

$$
\begin{aligned}
x_n &= A_n \sqrt{x_0^2 + y_0^2} = r \\
y_n &= 0 \\
z_n &= z_0 + \arctan\left(\frac{y_0}{x_0}\right) = z_0 + \theta
\end{aligned}
$$

The definition for d_i sign in vector mode aims to minimize y_i:

$$
d_i = \begin{cases} -1 & \text{if } y_i < 0 \\ +1 & \text{if } y_i \geq 0 \end{cases}
$$

This vector mode of the CORDIC rotator has been implemented as part of the Cartesian to log-polar transformation algorithm. The image pixel coordinates (x, y) are entered into the CORDIC algorithm as (x_0, y_0), with $z_0 = 0$. Consequently, the results directly yield the polar coordinates (r, θ).

The CORDIC algorithm in vector mode operates by rotating a given vector to a predetermined angle, primarily utilizing elementary arithmetic operations. This mode is initiated with a vector $\mathbf{V} = (x_0, y_0)$ and involves iterative rotations based on a series of predefined angles α_i, which are arctangent values of inverse powers of two, specifically $\arctan(2^{-i})$. In each iteration, the vector is rotated by α_i, either clockwise or counterclockwise, depending on the desired final orientation. The iterative process involves updating the components of the vector using simple additions and subtractions, alongside binary shifts

Algorithm 1 CORDIC Vector Mode Algorithm

Constants:
  $N\_ITER \leftarrow 16$
  $cordic\_angles \leftarrow arctan(2.0^{-range(N\_ITER)})$
procedure CORDIC_VECTOR_MODE($x_0, y_0, iterations = N\_ITER$)
  $x_i \leftarrow x_0$
  $y_i \leftarrow y_0$
  $z_i \leftarrow 0$
  for i from 0 to $iterations - 1$ **do**
    $d_i \leftarrow -1$ if $y_i < 0$ else 1
    $x_i\_next \leftarrow x_i - y_i \cdot d_i \cdot (2^{-i})$
    $y_i\_next \leftarrow y_i + x_i \cdot d_i \cdot (2^{-i})$
    $z_i\_next \leftarrow z_i - d_i \cdot cordic\_angles[i]$
    $x_i \leftarrow x_i\_next$
    $y_i \leftarrow y_i\_next$
    $z_i \leftarrow z_i\_next$
  return x_i, z_i

that substitute for multiplications by 2^{-i}. This iterative rotation continues until the vector is aligned as closely as possible to the x-axis, thus allowing the determination of its magnitude and phase.

4.3 The Log-CORDIC Transform Algorithm

The Log-CORDIC Transform algorithm presented here is a novel approach to image transformation, specifically converting images from Cartesian to polar coordinates using the CORDIC algorithm. This procedure begins by loading and converting an image into an array format, where it then calculates the center of the image (c_x, c_y) to serve as a pivot for transformation. Iteratively, for each pixel in the image, the algorithm computes the relative Cartesian coordinates (x, y) of the pixel with respect to the center. Utilizing the CORDIC algorithm in vector mode, these Cartesian coordinates are then transformed into polar coordinates (ρ, θ). The radial component is further processed through a logarithmic transformation to enhance the radial variations in the image. These transformed coordinates are used to map the original pixel values into a new image array, effectively converting the image from a Cartesian plane to a polar logarithmic plane. This transformation is particularly useful in applications where radial and angular features are more prominent or informative than Cartesian representations.

Algorithm 2 Log-CORDIC Transform Algorithm

procedure LOG-CORDIC($Image, OutputDim, NIter$)
Input: $Image$ with dimensions $(Height, Width, Channels)$
Parameters: $OutputDim = (RadiusBins, AngleBins)$, $NIter$ for CORDIC iterations
Output: Transformed image in log-polar coordinates
Define $Centre$ as $(\frac{Width}{2}, \frac{Height}{2})$
Initialize $TransformedImage$ as a zero array with dimensions $(Channels, RadiusBins, AngleBins)$
Calculate $MaxRadius = \sqrt{(Centre_x)^2 + (Centre_y)^2}$
Determine $LogBase = \frac{\ln(MaxRadius)}{RadiusBins-1}$
for each $rBin$ from 0 to $RadiusBins - 1$ **do**
  for each $aBin$ from 0 to $AngleBins - 1$ **do**
    Compute R and Θ using $CORDIC\_VECTOR\_MODE$ for each pixel position:
    Given pixel position (x, y), calculate:
    $R_{temp} = e^{(rBin \cdot LogBase)}$
    $\Theta_{temp} = \frac{2\pi \cdot aBin}{AngleBins}$
    $(R, \Theta) = CORDIC\_VECTOR\_MODE(R_{temp}, \Theta_{temp}, NIter)$
    Convert to Cartesian coordinates using R and Θ:
    $X_{cartesian} = Centre_x + (R \cdot \cos(\Theta))$
    $Y_{cartesian} = Centre_y + (R \cdot \sin(\Theta))$
    Assign interpolated pixel value to $TransformedImage[:, rBin, aBin]$ by interpolating $Image$ at $(X_{cartesian}, Y_{cartesian})$

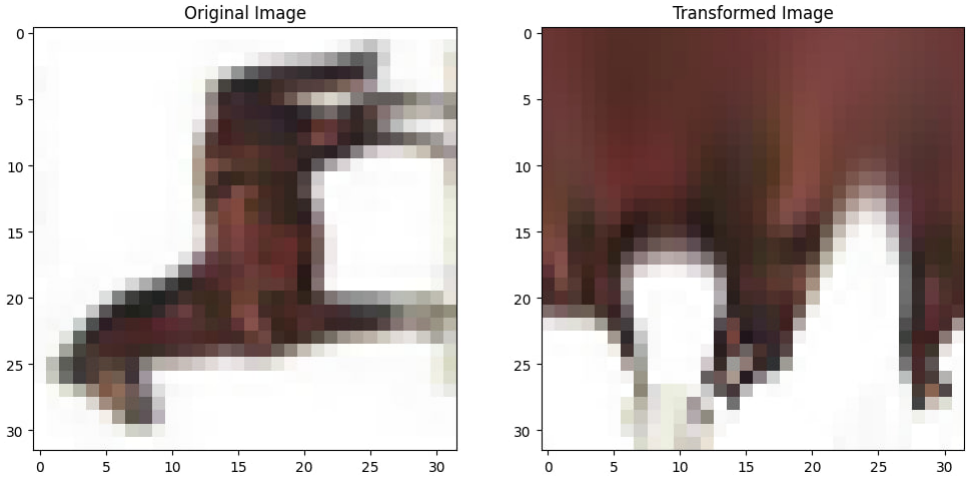

FIGURE 4.1: The result of an image's transformation with the Log-CORDIC algorithm

Enhancing the rotational invariance of a CNN through the Log-CORDIC transform is achieved, since rotated images vary little in their polar representation, compared to their cartesian representation. Below is the result of our Log-CORDIC transform applied to an image, after 90, 180 and 270 degree rotations have been applied to it.

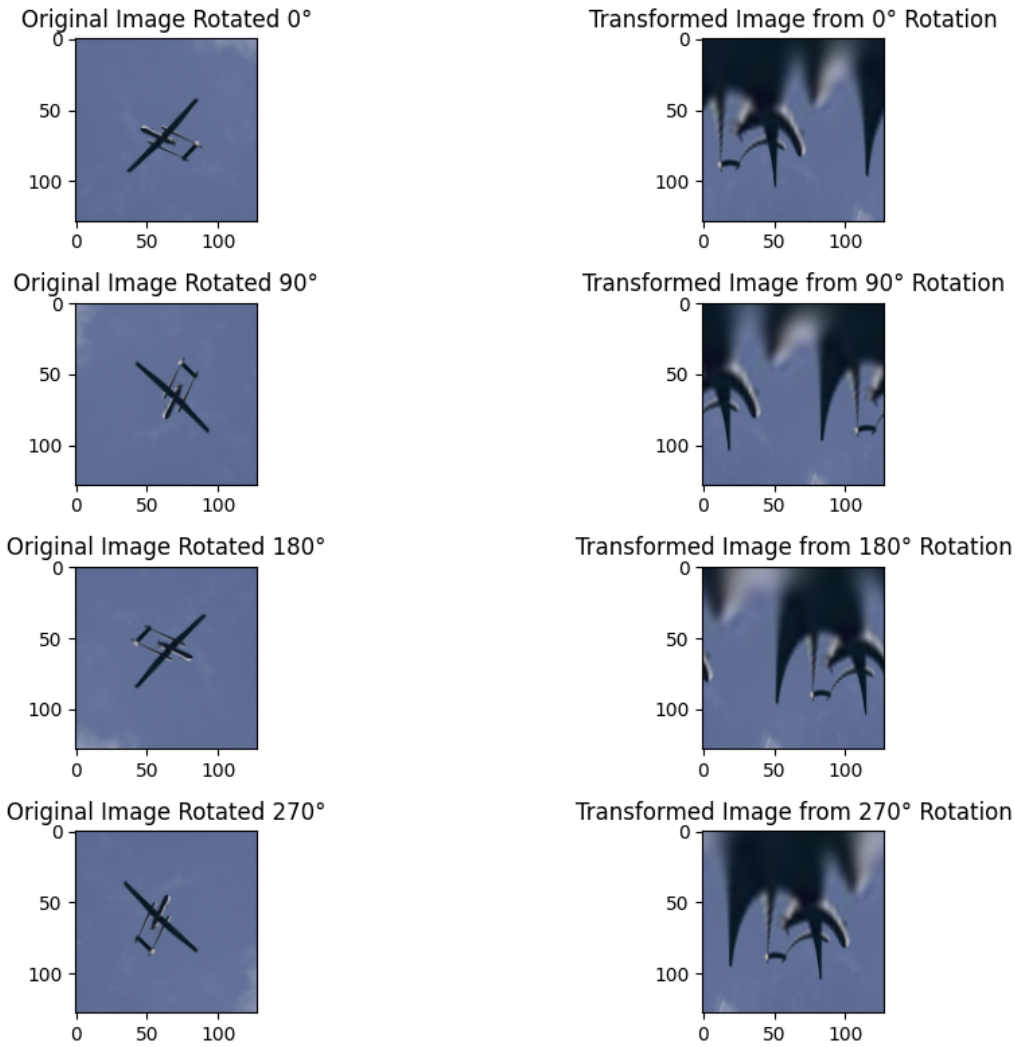

FIGURE 4.2: The result of an image's transformation with the Log-CORDIC algorithm

4.4 Logarithmic scaling

A particular area where a lot of development time for this algorithm was spent was the logarithmic scaling needed when using CORDIC to calculate the radii and angles in order to transform the image from a cartesian to polar representation. Incorrect logarithmic scaling can lead to a significant loss of detail from the resulting polar image, or even total distortion.

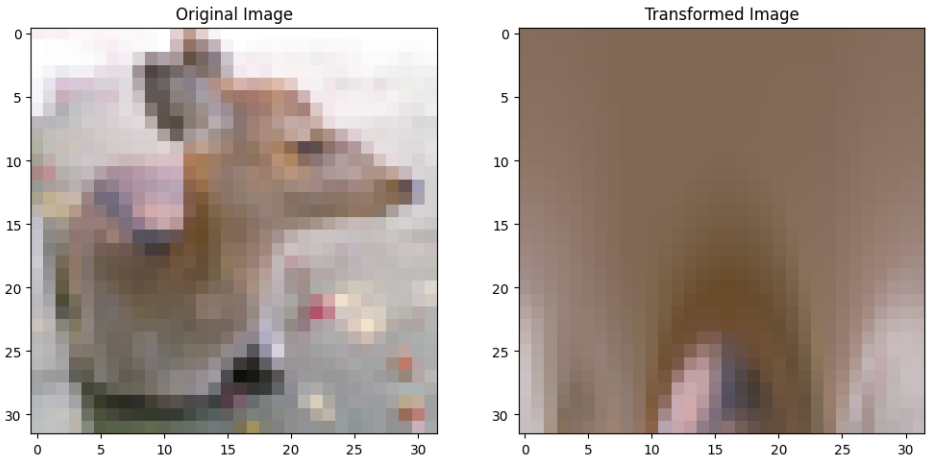

FIGURE 4.3: A distorted polar image due to incorrect logarithmic scaling

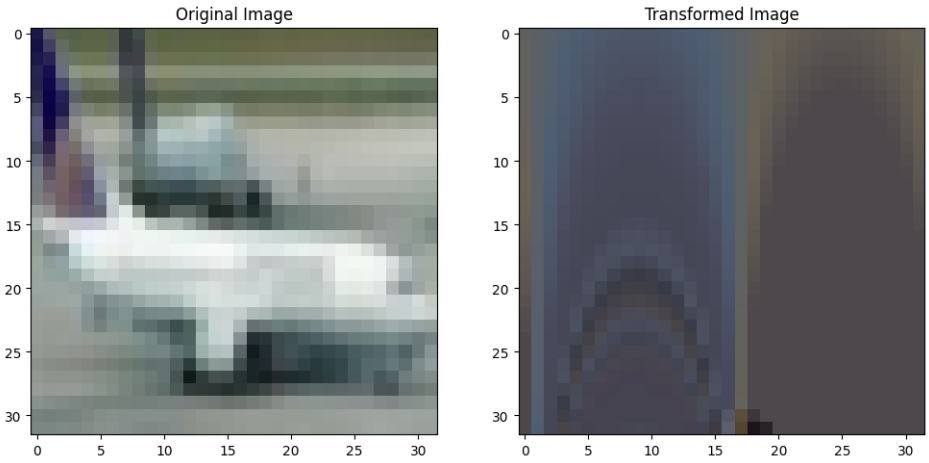

FIGURE 4.4: An example of further detail loss due to incorrect logarithmic scaling

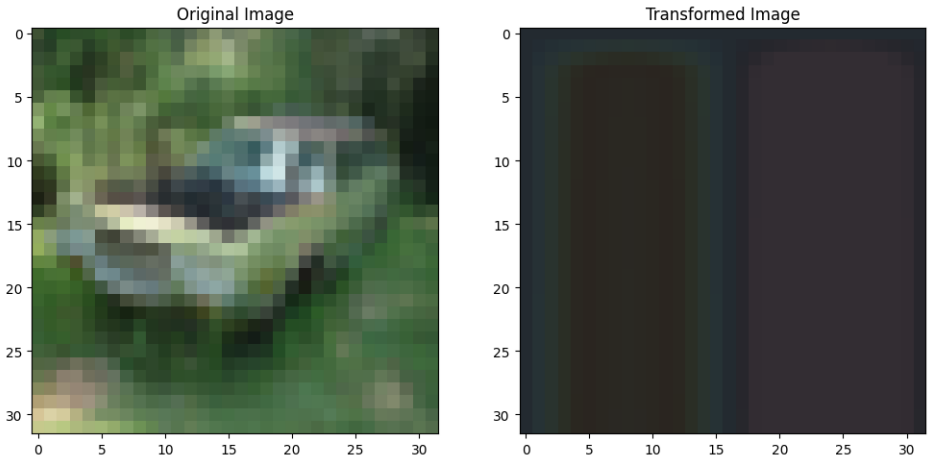

FIGURE 4.5: An even more extreme example

Incorrect logarithmic scaling can also cause loss of detail in specific ranges in the resulting image, where certain areas of the polar image will be mapped correctly on a cartesian plane in order to be displayed or input into a CNN, but others will not.

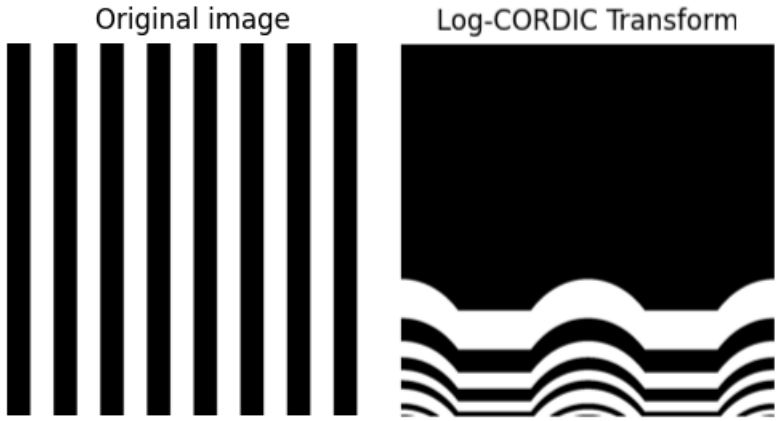

FIGURE 4.6: Example where specific areas of the polar image are affected

Therefore, in order to resolve these issues, a significant amount of research and development time was dedicated to this aspect of the algorithm. In this section, more details about the Log-CORDIC transform algorithm will be covered, specifically detailing the research about logarithmic scaling and what purpose it serves in our image transform.

This is due to the fact that the CORDIC algorithm relies on an iterative process to rotate and scale vectors to their target positions. The convergence of these iterations towards the correct logarithmic mapping is highly sensitive to the initial conditions and the step sizes used in each iteration. Small changes in these parameters can lead to significant differences in outcomes, making it challenging to find the optimal set that accurately represents the logarithmic mapping without distortion. The parameters governing the iterative scaling and rotation are interdependent. Changes to one parameter can affect the behavior of others, complicating the tuning process. For instance, adjusting the scaling factor requires corresponding adjustments in the rotation steps to maintain the integrity of the logarithmic mapping. The choice of output dimensions for the transformed image (i.e., the number of radii and angles) directly impacts the logarithmic mapping. Parameters must be chosen to ensure that the transformed image fits well within these dimensions without significant distortion or information loss, which becomes a balancing act between the theoretical aspects of the CORDIC algorithm and practical image processing needs.

Comparitively, the typical log-polar transform is mathematically straightforward, however, it becomes computationally expensive. The algorithm directly computes the logarithm of the maximum radius and divides it by the desired number of radii to determine the logarithmic base. With this, the logarithmic radius is then calculated, for each step in the output image, effectively mapping linear distances in the Cartesian plane to logarithmic distances in the log-polar plane. When it comes to mapping the output dimensions, the log-polar algorithm linearly spaces angles and logarithmically spaces radii. This approach is mathematically straightforward but computationally intensive due to the explicit use of logarithmic and exponential functions, as well as trigonometric functions for the conversion between coordinate systems.

While developing the solution to this issue, we constantly compared the output of the Log-CORDIC transform, with the result of OpenCV's log-polar image transform, which we shall consider the benchmark. Thus, we were able to solve the issue through continuous experimentation.

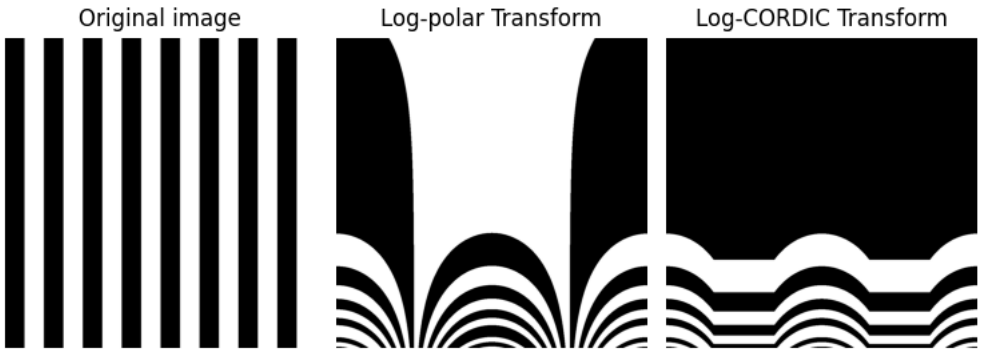

FIGURE 4.7: The output of the Log-CORDIC transform compared with a known good cartesian-to-polar transform algorithm

Logarithmic scaling is used to transform the linear scale of distances from the center of an image into a logarithmic scale. This means that as one moves radially outward from the center, each successive ring in the log-polar transformed image represents a geometrically increasing distance in the original image. This property is particularly useful for scale and rotation invariance in image processing tasks. The calculation of the logarithmic scale involves computing a base for the logarithm that maps the maximum radius of the image to the desired number of radial bins in the output image. The base is calculated as:

$$\text{log\_base} = \frac{\ln(\text{max\_radius})}{\text{radius} - 1}$$

This equation calculates the Euclidean distance from the center to the farthest corner of the image, essentially giving the maximum radius. The logarithmic base is then calculated to evenly distribute this maximum radius across the specified number of radial bins (radius). The subtraction by 1 ensures that the scale starts correctly from the center. This subtraction is crucial in the Log-CORDIC algorithm, because an incorrect center calculation will result in great loss of detail in the polar image, as shown above.

After calculating the logarithmic base, for each radial bin in the output image, the actual radius in the original image space is calculated with the formula:

$$r = e^{i \cdot \text{log\_base}}$$

This equation applies the logarithmic scaling by exponentiating the product of the bin index i and the calculated logarithmic base. This means that for each step outward in the radial direction, the corresponding radius in the original image increases exponentially. The first bin corresponds to the center of the image, and subsequent bins represent exponentially increasing radii.

In order to assess the validity of our Log-CORDIC transform algorithm, we also compared the Root Mean Square Error (RMSE) of the output of our transform compared to the output of the OpenCV log-polar image transform.

In the experimental benchmark designed to evaluate the accuracy of the Log-CORDIC transformation method relative to the established OpenCV log-polar transformation. The benchmark commenced with the selection of a standard test image, which was resized to a consistent dimension of 128 $\times$ 128 pixels to standardize the input across both transformation methods. This pre-processing step ensured that variations due to image size did not influence the comparison. Subsequently, the image underwent transformation by both the Log-CORDIC method, implementing a predefined set of parameters including the number of iterations for the CORDIC algorithm, and by OpenCV's log-polar transformation function, with both methods producing images of identical output dimensions for direct comparison.

The core of the benchmarking process involved the quantitative assessment of the two resulting images through the calculation of the Root Mean Square Error (RMSE). The Root Mean Square Error (RMSE) between two images is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(I_{1i} - I_{2i})^2}$$

Where:

- I_{1i} and I_{2i} are the pixel intensity values of the first and second image, respectively, at the i^{th} position.
- N is the total number of pixels in each image.

This metric provided a scalar value representing the average magnitude of difference between the two images, serving as an objective measure of the similarity and, by extension, the accuracy of the Log-CORDIC method in replicating the results of the well-established OpenCV transformation.

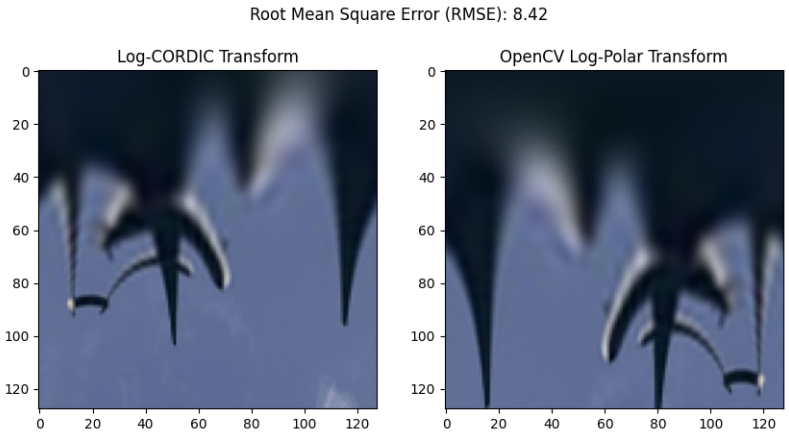

FIGURE 4.8: The result of the root mean square error calculation

As we can see from the results of the calculation benchmark, the root mean square error of the output image of the log-CORDIC algorithm is 8.42. While the value might not seem insignifacant, as we can see from the comparison of the output of the two different transforms, the error value stems from the difference in logarithmic mapping, instead of any transform errors or artifacts that can lead to loss of detail necessary for feature extraction in a neural network. Thus, the error, while being 8.42, does not lead to any impact when it comes to the application of our algorithm.

4.5 Advantages of using the log-CORDIC transform against a typical log-Polar transform

The employment of the CORDIC algorithm in this image transformation process offers significant advantages. Primarily, the CORDIC algorithm is renowned for its computational efficiency, as it relies solely on basic arithmetic operations and does not require complex multiplication or division operations. This efficiency is highly beneficial in image processing tasks, which often involve large data sets and require real-time performance. Furthermore, the iterative nature of the CORDIC algorithm allows for a fine-tuned balance between accuracy and computational load. By adjusting the number of iterations, the algorithm can achieve a desired level of precision in the calculation of polar coordinates. This adaptability makes it exceptionally suitable for embedded systems or hardware implementations where resources

are limited. In addition, the logarithmic transformation of the radial component, facilitated by the CORDIC algorithm's precision in calculating angles and magnitudes, allows for an enhanced visualization of details in images, particularly in regions with high radial variation. Logarithmic scaling is especially beneficial for preserving details in the center of the image while still capturing features at the periphery. It's particularly adept at handling scale variations, which makes it valuable in applications like pattern recognition and scale-invariant feature transformations.

The hallmark of the CORDIC algorithm's computational efficiency lies in its reliance on basic arithmetic operations, eschewing more resource-intensive operations like multiplication and division. This characteristic makes CORDIC particularly well-suited for hardware implementations where simplicity and speed are crucial. In digital systems, especially those with limited processing power or in embedded applications, the efficiency of CORDIC translates into significant performance benefits. Moreover, the algorithm's iterative nature allows for a pipelined approach in hardware, further enhancing its efficiency.

Another advantage of the log-CORDIC transform is the fact that its logarithmic mapping results in the local features within an image to be mapped to a reduced spatial extent in the transformed domain, compared to the log-polar transform. This concentration of local features can significantly enhance the efficiency of CNNs in detecting and analyzing these features. In technical terms, the transformation process effectively increases the density of informative pixels within certain regions of the image, thereby amplifying the signal-to-noise ratio with respect to the features of interest. When key features of an image occupy a more concentrated area in the transformed space, CNNs can potentially learn to discriminate between relevant features with greater precision. This concentration can help the network to focus on the most informative parts of the image, improving its ability to detect or classify objects even in the presence of scale and rotational variances. With important information condensed into a smaller area, the CNN may require fewer computational resources to process the same level of detail. This can lead to faster training and inference times, as the network focuses on the denser, more information-rich regions of the transformed image.

4.5.1 Performance Metrics

Number of Function Calls The metric quantifies the total number of function invocations executed throughout the algorithm's lifecycle. It serves as an

indirect indicator of the algorithm's computational complexity, reflecting the depth of subroutine dependencies and the potential for optimization. A minimal count suggests an efficient and streamlined algorithm, whereas a higher count may indicate areas prone to optimization, including redundant calculations or an inefficient algorithmic structure. From a technical standpoint, this metric is crucial for evaluating the overhead associated with recursive or iterative constructs, which can significantly impact memory usage and execution speed.

Execution Time Defined as the total duration required to complete an algorithmic process, execution time is measured in seconds and directly correlates with the algorithm's operational efficiency. It is a paramount metric for assessing the algorithm's practical applicability, particularly in scenarios demanding high throughput or low latency, such as real-time processing. Technical factors influencing execution time include the algorithm's inherent efficiency, data structure selection, and the exploitation of hardware capabilities (e.g., parallelism, cache optimization). Moreover, the choice of implementation language and its execution environment can substantially affect performance, with compiled languages typically offering superior speed relative to interpreted counterparts.

4.5.2 Measurement Techniques

Execution Time Measurement The process employs high-resolution timing functions, notably `time.time()` or `time.perf_counter()` from Python's `time` module, to capture precise execution durations. The technique involves recording timestamps immediately before and after the algorithm's execution, with their difference representing the net execution time. Accuracy considerations necessitate a controlled test environment to mitigate external influences and, when feasible, averaging results across multiple runs to accommodate variability in execution time due to system load or other transient factors.

Profiling for Computational Complexity Utilizing Python's `cProfile` module, this deterministic profiling approach measures function call frequencies, execution times, and the hierarchical structure of calls within the algorithm. Despite its detailed insights into the algorithm's execution profile, including identification of performance bottlenecks, `cProfile`'s instrumentation may

introduce overhead that potentially alters the algorithm's behavior. To minimize this effect, profiling is recommended to be confined to specific code segments integral to the algorithm's core functionality, thereby ensuring the relevance and accuracy of the performance data collected.

Isolation of Pre-computation In algorithms where performance can be enhanced through the pre-calculation of frequently used values (e.g., the CORDIC method's angles), isolating such pre-computations from the primary computational logic is crucial. By conducting these calculations outside the scope of performance measurement, the analysis more accurately reflects the efficiency of the algorithm's transformative process rather than the preparatory steps, thus ensuring a fair and focused assessment of the algorithm's computational demands.

4.5.3 Comparison results

Below are the results of the measurements between the Log-CORDIC transform implemented in this thesis, and a Log-Polar transform. In order to have a comparison of purely the computation effiency of the two transforms, the Log-Polar transform we are comparing our Log-CORDIC transform against, is implemented in Python, using solely the numpy library.

Testing setup The system that the neural network was trained and tested on was equipped with an Intel i7-12700H processor, with 6 performance cores and 8 efficiency cores, for a total of 14 cores and 20 total threads, with a maximum frequency of 4.7GHz. The total available system memory was 32 GB, and the system was also equipped with an nVidia Quadro T600 GPU, with 4 GBs of VRAM and 896 CUDA cores.

Metric	Log-Polar	Log-CORDIC	Improvement
Function Calls	73,400,393	1,728,517	x 42
Execution Time (seconds)	99.354	12.931	x 7.6

TABLE 4.1: Comparison of Log-Polar and Log-CORDIC Transformations

The results of the comparative analysis between the Log-Polar and Log-CORDIC transformation methods provide clear evidence of the superior efficiency of the Log-CORDIC approach in terms of both computational complexity and execution speed. The Log-CORDIC transformation method required significantly fewer function calls (1,728,517) compared to the Log-Polar method

(73,400,393). This stark reduction, approximately 97.65%, indicates a vastly lower computational complexity for the Log-CORDIC method. Fewer function calls suggest that the Log-CORDIC algorithm employs a more efficient computational strategy, possibly due to its iterative nature that leverages precomputed angles for rotation and vectoring, thus avoiding the computational overhead associated with the more complex pixel remapping and interpolation processes typical in the Log-Polar method.

In terms of execution time, the Log-CORDIC method completed the transformation in just 12.931 seconds, while the Log-Polar method took 99.354 seconds. This represents an 86.98% improvement in speed for the Log-CORDIC method. Such a significant decrease in execution time underscores the Log-CORDIC method's suitability for real-time applications or scenarios requiring rapid processing of large datasets. The efficiency gains in execution speed can be attributed to the method's streamlined computational process, which minimizes unnecessary calculations and exploits pre-computation to accelerate the transformation.

We can identify two main areas of inefficiency in the Log-Polar transform that lead to these results. Presented here is the algorithm of a typical Log-Polar transform.

Algorithm 3 Log-Polar Transform

```
image ← input image
output_dim ← (radius, theta)                          ▷ Output dimensions
centre ← find image centre
max_radius ← calculate maximum radius from centre
log_base ← calculate log base using max_radius and output_dim
for r in range(output_dim.radius) do
    for θ in range(output_dim.theta) do
        r_linear ← convert r from log space to linear space
        x, y ← convert polar coordinates (r_linear, θ) to Cartesian
        pixel_value ← interpolate pixel value at (x, y)
        store pixel_value in output image at (r, θ)
```

The typical log-polar transform requires interpolation for each pixel in the output image, based on Cartesian coordinates derived from the log-polar coordinates. This process is computationally expensive because it involves floating-point arithmetic and can require accessing non-contiguous memory locations, leading to cache inefficiencies. The conversion from log-polar coordinates back to Cartesian coordinates for each pixel in the output image is necessary to perform interpolation on the original image. This step involves

trigonometric operations (sine and cosine) for each pixel, which are more computationally intensive than the simpler arithmetic operations used in the CORDIC algorithm.

The Log-CORDIC algorithm circumvents the need for direct trigonometric function calls by employing an iterative approach to calculate vector rotations. The CORDIC technique relies on a series of predefined angles (for which the tangent values are powers of two) and executes a sequence of shift and add operations to approximate rotations. This method systematically rotates a vector towards a target angle using a series of iterative steps that either add or subtract these predefined angles from the current angle, based on the direction of rotation needed. The sine and cosine values of the target angle are implicitly calculated through this process, represented by the final coordinates of the rotated vector. This iterative rotation mechanism eliminates the need for expensive trigonometric function evaluations, replacing them with more efficient arithmetic operations. As a result, the computational load is significantly reduced, leading to faster execution times and lower power consumption.

Chapter 5

System Model

5.1 Introduction

This chapter details the systematic approach to modeling our real-time image classification system, with a particular emphasis on its implementation in Python. The process of system modeling is a critical step in the research and development phase, offering a detailed examination and understanding of the system's functionality and operational behavior. By engaging in modeling, we aim to construct a simplified yet comprehensive representation of the system, which facilitates the exploration, analysis, and validation of its design. This methodical approach is instrumental in identifying and addressing potential issues or inefficiencies at an early stage, contributing to the development of a robust and dependable system design.

The model is designed with precision to capture the key characteristics of the real-time image classification system, with a special focus on incorporating the CORDIC algorithm to achieve rotational invariance. The adoption of Python as the programming language for this model enhances its accessibility and adaptability, offering a flexible and dynamic development environment. Python's compatibility with a wide array of libraries and tools significantly simplifies the process of model implementation and testing, streamlining the development workflow.

In the following sections, we will explore in detail the components and functionalities that comprise our Python-based system model. Through comprehensive explanations, code examples, and illustrations, we aim to provide an in-depth view of the model's architecture, workflow, and the algorithms that underpin it. This detailed exposition is intended to equip the reader with a solid understanding of the model, setting a strong foundation for the later stages of system development and assessment.

5.2 Model Description

In the system's model equivalent, we encompass the various stages of the necessary image processing and transformation, all developed using Python and the numpy framework, in order to reduce dependence on libraries that obfuscate the algorithms that support them, and make the model clear in its functionality and efficiency. The model's architecture is delineated into several pivotal components, each playing a quintessential role in enhancing the robustness and efficacy of the real-time image classification system.

The initial phase involves image processing, where each image is subjected to padding. Padding is executed by appending white pixels around the image, the quantity of which is determined by predefined parameters. This process is instrumental in manipulating the spatial dimensions of the image, ensuring that essential features are not lost during subsequent convolution operations. The padding process is meticulously tailored, where k rows of pixels are added to all sides of the image, ensuring a symmetrical and balanced augmentation of the image dimensions.

Subsequent to the padding process, the images undergo a cylindrical wrapping transformation. In this approach, if k rows of pixels were added as padding, the bottom k rows of the image are replicated onto the top padding, and conversely, the top k rows are duplicated onto the bottom padding. This strategic manipulation fosters a cylindrical effect on the image, a crucial step that facilitates the seamless application of the CORDIC algorithm in later stages.

Following the cylindrical transformation, the images are then transmuted into a polar representation through the application of the CORDIC algorithm. This algorithm is pivotal in the computation of trigonometric functions, essential for the conversion of the image into its polar form with a Cartesian representation. The utilization of the CORDIC algorithm is instrumental in enhancing the rotational invariance of the image classification system, a cornerstone in achieving robust and accurate classification outcomes.

5.3 Image Classification Model

5.3.1 Overview of Image Classification Models

Image classification has been revolutionized by the advent of various deep learning models, each offering unique strengths. Among the prominent models are AlexNet, which ignited the deep learning revolution in image classification; VGGNet, known for its depth and architectural simplicity; ResNet, which introduced residual learning to facilitate training of very deep networks; and the Inception series, which implemented novel ideas to capture information at various scales. These models have continuously pushed the boundaries of accuracy in image classification tasks. However, their often considerable computational and memory requirements pose challenges for deployment on platforms with limited resources, such as FPGAs.

5.3.2 SqueezeNet: An optimized choice for edge computing

SqueezeNet emerges as an optimal choice in our system model due to its compact architecture and low memory footprint. This model achieves AlexNet-level accuracy with significantly fewer parameters, using an architecture that incorporates 'squeeze' and 'expand' layers to reduce parameter count while maintaining performance. The small model size of SqueezeNet is particularly advantageous for image classification systems targeted at embedded or mobile devices, as it allows the entire model to be loaded onto them without overburdening their limited resources. This characteristic makes SqueezeNet an ideal candidate for edge computing applications where both performance and efficiency are essential.

5.3.3 Adapting SqueezeNet for Rotational Invariance

The integration of SqueezeNet with the Log-CORDIC transform algorithm presents a powerful combination for enhancing rotational invariance in image classification. The Log-CORDIC transform, by converting images from Cartesian to polar coordinates, inherently improves the model's robustness to rotational variations in input images. SqueezeNet's architecture, with its streamlined yet effective feature extraction capabilities, complements this transform by efficiently processing the transformed images. This synergy allows for improved rotational invariance without a substantial increase in computational complexity.

In order to achieve the rotational invariance in the image classification model, we need to exploit the cylindrical properties of the polar coordinate system. In the Cartesian coordinate system, rotating an object translates to vertical movement in the polar coordinate system. Consider a point p with coordinates (x_p, y_p) in the Cartesian system, which is equivalent to a point with coordinates (ρ_p, φ_p) in the polar system. When we rotate the point p around the origin by an angle $\varphi_{p'} - \varphi_p$, it results in a new point p' with polar coordinates $(\rho_{p'}, \varphi_{p'})$. This action, due to the constancy of $\rho_{p'} = \rho_p$, is manifested as a shift along the φ axis by $\varphi_{p'} - \varphi_p$ radians in the polar system. It's important to recognize that in the polar coordinate framework, such vertical shifts can extend beyond the image's boundary limits. This is illustrated when rotating another point q with coordinates (x_q, y_q), where the rotation surpasses the Cartesian system's $(x > 0, y = 0)$ ray, leading to its polar translation exceeding the $\varphi = 2\pi$ limit.

The log-polar coordinate system shares its foundational principles with the polar coordinate system, with the primary difference being the logarithmic calculation of distance from the origin. In this system, the distance ρ is recalculated as $\rho = \log(\sqrt{x^2 + y^2})$. This log-polar representation draws inspiration from the human eye's structure and finds extensive applications across various visual tasks.

5.4 CyCNN and Cylindrically Sliding Windows

The CyCNN model by Kim et al. [17], processes images converted into polar coordinates. Imagine an original image and its variations rotated by 90°, 180°, and 270° within the polar coordinate framework. A particular region, denoted as the receptive field within a pooling layer, focuses on crucial aspects of the image, such as specific features on a plane.

When training the CNN with the original image, this receptive field is adept at identifying significant features, like distinctive marks on a plane's surface. However, the challenge arises when the image is significantly rotated, for example, by 270°. The model might struggle to recognize the plane if critical features, like markings or windows, appear too distant or if their spatial relationship is altered. This issue stems from the model's inability to accurately interpret the repositioned features within its receptive field.

To address this limitation, they introduced the concept of cylindrically sliding windows (CSW) for convolutional layers, thus creating a cylindrical convolutional layer (CyConv). This approach modifies how boundaries are treated: rather than adding zeros at the input's top and bottom edges, we replicate the pixels from the first and last rows at opposite boundaries. This replication, combined with standard zero padding on the sides, effectively rolls the image into a cylinder shape. The CyConv layer then cyclically scans this cylindrical input with its filter.

CSW fundamentally enhances the receptive field's vertical span on the original image, effectively wrapping the input and presenting it anew to each convolutional layer. Through integrating CSW with the layered structure of convolutional and pooling layers, the CyCNN is better equipped to discern complex feature relationships, thereby improving its ability to recognize and interpret rotated images or objects with shifted feature positions.

5.5 Introducing CSWs to SqueezeNet

While Kim et al. [17] modified the VGG19 and ResNet56 models, we chose to modify the SqueezeNet model due to its inherent efficiency. In order to do so, a custom cylindrical convolution layer was implemented for our model, that was utilised in SqueezeNet's Fire modules and the layers before and after the use of fire modules in the SqueezeNet CNN model.

To implement this cylindrical convolution, first, the input images have to be padded cylindrically. Instead of the usual zero or same padding in convolution layers of CNNs, we are mimicking a cylindrical wrapping along the vertical edges of the image while applying standard padding to the horizontal edges. For p padding pixels in our image, we are copying the top p pixel rows of the original image to the bottom p rows of the resulting padded image, and conversely, we are copying the bottom p pixel rows of the original image to the top p rows of the resulting padded image. The horizontal edges of the padding are padded with zeroes.

Algorithm 4 Cylindrical Padding Algorithm

procedure CYLINDRICALPADDING(X, P)
Input: Image tensor X with dimensions (N, C, H, W)
Input: Padding size P
Output: Padded image tensor
Step 1: Wrap top and bottom edges
$TopWrap \leftarrow$ Extract the last P rows from X
$BottomWrap \leftarrow$ Extract the first P rows from X
$X \leftarrow$ Concatenate $BottomWrap$, original X, and $TopWrap$ vertically
Step 2: Pad left and right edges
$LeftRightPadding \leftarrow$ Apply zero padding of width P to the left and right edges of X
return $LeftRightPadding$

This cylindrical padding is used to implement a Cylindrically Sliding Window convolution algorithm. Whereas Kim et al. [17] utilised the Winograd algorithm for their custom convolution layer, we took a simpler approach, where we overload the torchvision 2D convolution method with our custom padding method. This ensures that we are taking advantage of the enhancements in performance that an already mature image processing library like torchivision already offers, and development time is shortened to a degree.

Algorithm 5 Cylindrical Convolutional Operation

procedure CYLINDRICALCONVOLUTION(X, $InChannels$, $OutChannels$, $KernelSize$, $Stride$, $Padding$)
Input: Input tensor X with dimensions (N, C, H, W)
Parameters:
$InChannels$ - Number of input channels
$OutChannels$ - Number of output channels
$KernelSize$ - Size of the convolution kernel
$Stride$ - Stride of the convolution
$Padding$ - Padding size to be applied before convolution
Output: Output tensor after cylindrical convolution
Apply cylindrical padding to input tensor
$X_{padded} \leftarrow$ CYLINDRICALPADDING$(X, Padding)$
Apply 2D convolution over the padded tensor
Define convolution operation $Conv$ with parameters:
$InChannels$, $OutChannels$, $KernelSize$, $Stride$, and no internal padding
$X_{conv} \leftarrow Conv(X_{padded})$
return X_{conv}

With this custom cylindrical convolution layer, our SqueezeNet's Fire modules and its overall architecture are enhanced with rotational invariance.

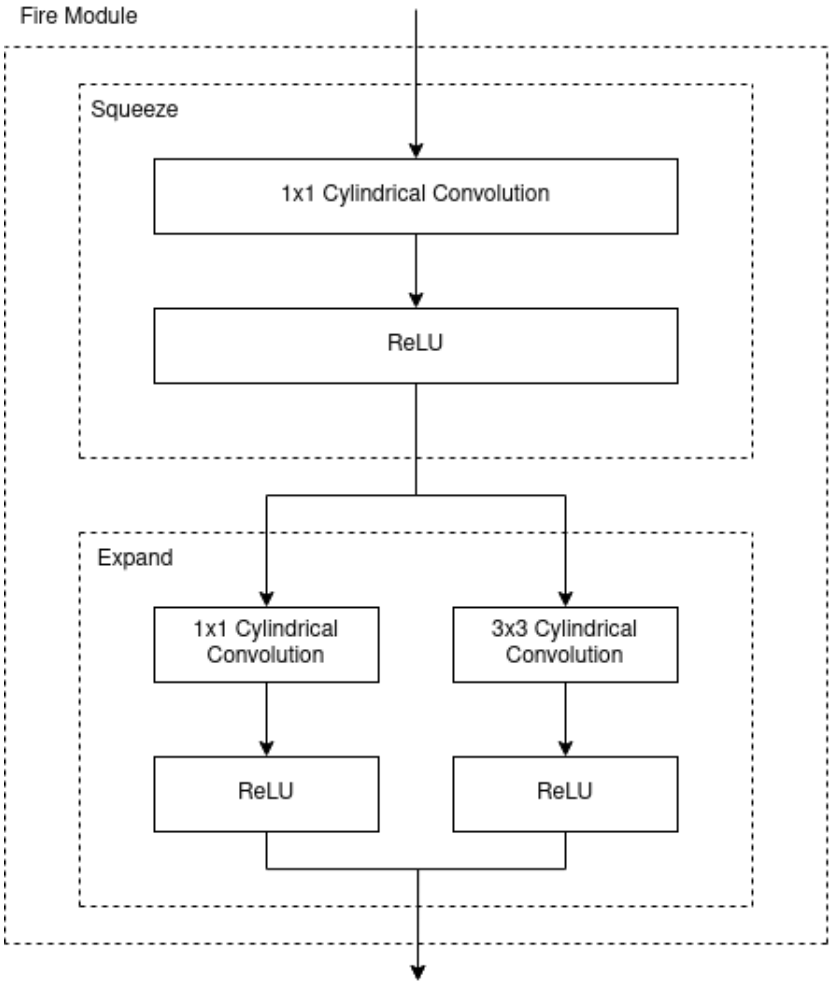

FIGURE 5.1: SqueezeNet Fire module with cylindrical convolution

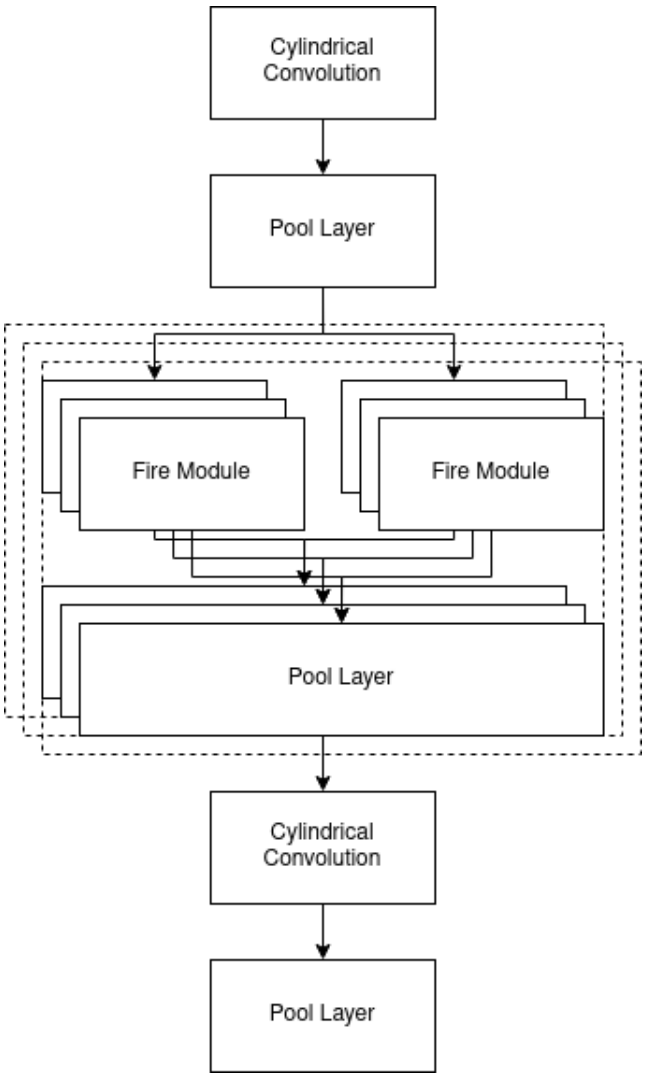

FIGURE 5.2: SqueezeNet architecture with cylindrical convolution

5.6 Training and validation

5.6.1 Exploring Potential Datasets for Image Classification

In the realm of image classification, the choice of dataset is critical for training and evaluating the performance of models. Several benchmark datasets have been established, each with its unique characteristics and challenges. ImageNet, one of the largest and most diverse datasets, has been instrumental in advancing deep learning models but requires significant computational resources. The CIFAR datasets, including CIFAR-10 and CIFAR-100, offer a balance between complexity and manageability with their smaller image sizes and limited classes. MNIST, a dataset of handwritten digits, is renowned for

its simplicity and is often used as an entry point for testing image classification models. Other datasets like SVHN (Street View House Numbers) and Fashion-MNIST provide alternative challenges in image recognition.

5.6.2 Criteria for Dataset Selection

The criteria for selecting datasets in this thesis centered around two key factors: the ability to effectively demonstrate the model's rotational invariance and the computational feasibility of processing the dataset on our FPGA-based system. It was essential to choose datasets that not only present a variety of image orientations but also allow for modifications to test the model's robustness against rotational variations. Additionally, given the hardware constraints and the focus on real-time processing, datasets that could be efficiently handled within the FPGA's resource limitations were preferred.

5.6.3 Choosing CIFAR-10 for Demonstrating Rotational Invariance

CIFAR-10 was chosen as one of the primary datasets for this project due to its moderate complexity and diversity. Consisting of 60,000 color images in 10 classes, CIFAR-10 presents a more realistic challenge compared to simpler datasets like MNIST. The variety of natural images in CIFAR-10, including animals and vehicles, provides a suitable testbed for demonstrating the enhanced rotational invariance of our modified SqueezeNet model. By applying rotations to the CIFAR-10 images and evaluating the model's performance, the effectiveness of the log-CORDIC transform in handling rotational variations could be effectively showcased.

5.6.4 Utilizing MNIST for Initial Testing and Validation

MNIST was selected as a complementary dataset to CIFAR-10 for its simplicity and uniformity. The dataset comprises 70,000 grayscale images of handwritten digits, making it less computationally intensive to process and ideal for initial testing and validation of our system. The simplicity of MNIST allows for a clear demonstration of the model's improved rotational invariance, particularly in the early stages of development. By augmenting MNIST images with various rotational angles, the impact of the log-CORDIC transform on the model's ability to recognize digits, irrespective of their orientation, could be directly observed and quantified.

5.6.5 Rationale Behind Combining CIFAR-10 and MNIST

The combination of CIFAR-10 and MNIST as test datasets provided a comprehensive approach to evaluating our system. While MNIST served as a straightforward platform for initial experimentation and proof-of-concept, CIFAR-10 introduced the complexities of real-world images. This dual-dataset strategy allowed for a nuanced assessment of the model's capabilities, ensuring that the improvements in rotational invariance were not just limited to simple or idealized images but were also effective in more challenging scenarios.

The selection of CIFAR-10 and MNIST as the datasets for demonstrating the enhanced rotational invariance of our FPGA-implemented image classification system was a strategic decision aligned with our objectives. CIFAR-10 provided the complexity and diversity necessary for a robust evaluation, while MNIST offered a simpler, computationally efficient platform for initial testing and validation. This approach ensured a thorough and nuanced demonstration of the system's capabilities, affirming the effectiveness of the log-CORDIC transform and the modified SqueezeNet model in achieving rotational invariance across different levels of dataset complexity.

5.7 System Robustness Analysis

5.7.1 Definition of Robustness

In computational systems and algorithms, robustness is defined as a system's ability to maintain consistent and effective operation under varied conditions, uncertainties, or potential adversities. This thesis examines the robustness of a real-time image classification system implemented on Field-Programmable Gate Arrays (FPGAs), enhanced with the CORDIC algorithm for rotational invariance. The system's robustness is demonstrated by its capacity to accurately classify images regardless of their rotational orientation, alongside resilience to image quality variations, hardware constraints, and external perturbations, ensuring not only theoretical soundness but also practical reliability and consistent performance in diverse real-world situations.

5.7.2 Perturbations and Uncertainties

The domain of real-time image classification on FPGAs, especially with the integration of the CORDIC algorithm for rotational invariance, encounters several challenges and uncertainties. Image quality variability, including resolution differences, contrast, brightness levels, and the presence of noise, artifacts, or blurring, can affect classification accuracy. Rotational ambiguities, particularly with extreme or uncommon angles, may challenge the system's design premise of rotational invariance. FPGA hardware limitations, such as memory capacity, processing power, and bandwidth, could restrict high-resolution image processing or complex model execution. Temporal variations and environmental changes add another layer of complexity, potentially influencing classification outcomes. Algorithmic limitations of the CORDIC algorithm may also arise, affecting computational precision. External factors like electromagnetic and thermal interference could impact FPGA operations, while challenges in model generalization, latency, data transmission anomalies, and scalability concerns necessitate careful navigation to fully leverage the system's capabilities in varied applications.

5.8 Experimental Setup and Methodology

5.8.1 Preparation of Datasets

The CIFAR-10 and MNIST datasets were selected for their relevance and popularity in image classification tasks. CIFAR-10 consists of 60,000 32x32 color images in 10 classes, offering a diverse array of subjects for classification. MNIST, on the other hand, includes 70,000 28x28 grayscale images of handwritten digits and is widely used for benchmarking classification algorithms. Prior to the experiments, these datasets underwent a standard preprocessing phase, which included normalization and shuffling, to ensure consistent input data quality and to prevent model overfitting.

5.8.2 Training Procedure

For the training phase, the models were initially trained with 40,000 non-rotated images from the CIFAR-10 and MNIST datasets respectively. This baseline training aimed to establish a reference performance level for the models under standard conditions. The training process involved 10 epochs, with batch processing and adaptive learning rate adjustments to enhance the

learning efficiency. The use of mini-batch gradient descent helped in achieving a balance between computational efficiency and convergence speed.

5.8.3 Testing and Evaluation Setup

Following the training phase, a distinct testing phase, over 10,000 images from each dataset, was initiated, where the models were exposed to a new set of images from the respective datasets. These images were pre-processed using the Log-CORDIC algorithm to introduce rotations in 90, 180 and 270 degrees, in a uniform distribution across the images. These rotations were selected in order to avoid an additional pre-processing step, where the 32x32 images would have gaps that would have needed to be filled in, were we to introduce rotations at random degrees. This testing setup was designed to evaluate the models' ability to maintain classification accuracy in the face of rotational variations, thereby assessing the improvement in rotational invariance.

The system that the neural network was trained and tested on was equipped with an Intel i7-12700H processor, with 6 performance cores and 8 efficiency cores, for a total of 14 cores and 20 total threads, with a maximum frequency of 4.7GHz. The total available system memory was 32 GB, and the system was also equipped with an nVidia Quadro T600 GPU, with 4 GBs of VRAM and 896 CUDA cores.

5.8.4 Evaluation Results

Presented are the outcomes of our investigation into the efficacy of an image classification model. The study was conducted by training the model with two distinct datasets: MNIST and CIFAR-10. The aim was to ascertain the effectiveness of the Log-CORDIC transform in enhancing the model's ability to classify rotated images accurately, when training our SqueezeNet implementation, which was enhanced with a custom cylindrical two-dimensional convolution layer.

The results indicate that incorporating the Log-CORDIC transform into the training process of image classification models leads to an extended training duration. However, this investment in training time is compensated by a marked improvement in model performance, specifically a 5% to 7% increase in accuracy for classifying rotated images. This enhancement underscores

TABLE 5.1: Experimental Results: Training and inference with the MNIST dataset

Training	**Without Log-CORDIC**	**With Log-CORDIC**
Training Time (seconds)	10.523	12.442
Testing Time (seconds)	0.337	0.342
Total Time (seconds)	12.816	13.101
Test Accuracy	0.344	0.410

TABLE 5.2: Experimental Results: Training and inference with the CIFAR-10 dataset

Training	**Without Log-CORDIC**	**With Log-CORDIC**
Training Time (seconds)	12.252	14.704
Testing Time (seconds)	0.572	0.590
Total Time (seconds)	20.302	23.001
Test Accuracy	0.359	0.425

the value of the Log-CORDIC transform in preparing models to handle geometric variations in input data, a common challenge in real-world image classification tasks.

Chapter 6

Proposed Architecture

6.1 A Proposed Hardware Architecture

In this chapter, we will explore the potential for an acceleration of the Log-CORDIC transform algorithm. Considering that CORDIC is particularly suited for embedded or edge computing devices, a hardware architecture in which the algorithm is utilised to transform images in order to enhance the rotational invariance of image classification or detection systems can prove to be a viable research area.

6.2 Selecting the Hardware Platform

6.2.1 The PYNQ-Z1 Development Board

The PYNQ-Z1 board, which integrates a dual-core ARM Cortex-A9 processor with an Artix-7 FPGA, could be used for its ability to balance processing power with programmable logic. This choice is based on the board's compatibility with the Python Productivity for Zynq (PYNQ) open-source framework, enabling Python-based development and simplifying the interface with the FPGA's programmable logic. The board's comprehensive I/O options, including HDMI, USB, Ethernet, Arduino, and Raspberry Pi expansion headers, along with a pre-installed Linux-based operating system, facilitate a wide range of applications, from signal processing to machine learning.

6.2.2 Accelerating Machine Learning Applications

The PYNQ-Z1 board's architecture is particularly advantageous for machine learning applications that require significant computational speed and efficiency. By offloading intensive tasks to the FPGA, the execution of machine

learning algorithms can be accelerated, enhancing performance for real-time processing and large dataset analysis. This capability is exemplified in the acceleration of algorithms that benefit from parallel computation, such as those common in machine learning.

6.3 Development Tools

6.3.1 Introduction to Development Tools

The development of FPGA-based machine learning applications requires tools that bridge the gap between high-level programming concepts and low-level hardware implementation. In conjuction with our proposed choice of the PYNQ platform, two primary tools to facilitate the translation of algorithms into optimized hardware designs can be used, namely the Xilinx Vitis HLS and the Vivado Design Suite toolkits.

6.3.2 Employing Xilinx Vitis HLS

Xilinx Vitis HLS can play a pivotal role in converting Python-based machine learning algorithms into optimized hardware descriptions. By allowing algorithmic descriptions in high-level languages, Vitis HLS enables a focus on algorithm functionality rather than intricate hardware details and design, simplifying the development process and accelerating the prototyping and testing phases of the project.

6.3.3 System Integration with Vivado Design Suite

The Vivado Design Suite provides a comprehensive environment for synthesizing and analyzing the hardware design, integrating HLS-generated code with other system components, and facilitating advanced debugging. Vivado's simulation capabilities can offer insights into the performance and interaction of system components, aiding in the optimization of the proposed architecture.

6.4 Advantages of CORDIC for FPGA-Based Machine Learning

Implementing the CORDIC algorithm on the FPGA can demonstrate considerable computational speed improvements. The CORDIC algorithm's suitability for FPGA architectures—owing to its iterative nature and ability to perform trigonometric, logarithmic, and hyperbolic functions using basic arithmetic operations can be leveraged to optimize machine learning tasks that include rotations, scaling, or transformations.

6.5 Transitioning from the model to High-Level Synthesis (HLS)

This section will present the proposed transition from high-level algorithmic descriptions and a Python system model to High-Level Synthesis code targeting FPGA platforms. Presented here is an in-depth analysis of the Log-CORDIC transform as a proposed architecture comprised of HLS modules.

Architecture and Design of Log-CORDIC HLS Module At the core of the Log-CORDIC HLS module lies an architecture tailored to leverage the iterative capabilities of the CORDIC algorithm for computing trigonometric and logarithmic functions efficiently. The design precomputes scaling constants (`K_values`) and arctangent values (`atan_values`) to reduce computational load during runtime. This precomputation is essential for minimizing the operations required for each iteration, directly impacting the FPGA's resource utilization.

Precision Handling through Fixed-Point Arithmetic Precision management is a critical aspect of the Log-CORDIC HLS module, where fixed-point arithmetic is utilized to balance computational accuracy with resource constraints. This proposition for a hardware architecture carefully selects data types and iteration counts to ensure that computations remain within the FPGA's capabilities while achieving the required precision. This approach underscores the importance of precision in hardware implementations, where resource limitations necessitate careful planning and execution.

First, fixed-point arithmetic is markedly less resource-intensive than its floating-point counterpart, a critical advantage in the resource-constrained environment of FPGAs. This efficiency stems from the simpler hardware logic required for fixed-point calculations, which does not necessitate the handling of floating-point exponents and mantissas. Additionally, fixed-point operations boast deterministic execution times, a property that guarantees consistent and predictable performance, essential for real-time processing applications where execution time variability can compromise system reliability.

Moreover, the precision afforded by a 16-bit fixed-point format, with an optimized allocation between integer and fractional parts, is generally sufficient for the demands of signal processing tasks, including those employing the CORDIC algorithm. The specific choice of 2 bits for the integer component intimates an expected operational range that does not necessitate a broad integer spectrum, thus maximizing the precision available for fractional components to enhance the accuracy of trigonometric calculations crucial to the CORDIC algorithm's efficacy.

This choice is also informed by the necessities of image processing applications. This allocation supports a [-2, 1.75] range for signed numbers, covering the normalized or scaled values common in image processing without unnecessary bit allocation. Furthermore, it optimizes FPGA resource use, enhancing precision where most needed and supporting the nuanced requirements of image transformations and analyses.

Optimization for Efficiency and Throughput To enhance efficiency and throughput, the Log-CORDIC HLS module can incorporate loop unrolling and pipelining optimizations. These techniques leverage the pipeline-compatible nature of the CORDIC algorithm into parallel operations that better utilize the FPGA architecture. The application of HLS directives for loop optimization demonstrates an effective utilization of FPGA resources, highlighting the module's design as focused on achieving high performance in hardware implementations.

In parallel, the pre-computation of CORDIC parameters, such as the iterative angles and the cumulative scaling factor (K), emerges as a strategy to diminish computational overhead during runtime. This approach alleviates the need for recalculating these constants for each operation, thereby streamlining computational processes. By embedding these pre-computed values

within the HLS module as a lookup table, direct access is facilitated, significantly accelerating computation by circumventing the execution of complex arithmetic operations repeatedly.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

7.1.1 Summary of Key Findings

This thesis presented a novel approach to enhancing the rotational invariance of Convolutional Neural Networks (CNNs) by integrating the Log-CORDIC algorithm into the image preprocessing pipeline. The key finding in this thesis is the comparative efficiency of the Log-CORDIC algorithm in transforming images from a cartesian to polar representation. Also, experiments were conducted to verify the increase in classification accuracy for rotated images. These results indicate that the Log-CORDIC algorithm efficiently and effectively normalizes rotational variations in images, enabling the CNNs to maintain high accuracy levels irrespective of image orientation.

7.1.2 Advantages of the Proposed Methodology

The methodology proposed in this thesis, involving the application of the Log-CORDIC algorithm for preprocessing, proposes the Log-CORDIC algorithm due to its computation effiency in transforming images from the a cartesian to polar representation. Unlike other methods that can be computationally expensive and may require extensive modifications to the CNN architecture or complex augmentation of training data, the Log-CORDIC algorithm provides a straightforward means to enhance rotational invariance. This simplicity makes the approach easily adaptable to existing CNN models and practical for various applications.

7.1.3 Practical Implications and Applications

Enhanced Reliability in Automated Systems The enhancement of rotational invariance in image classification models has profound implications for automated systems across various sectors. In domains such as autonomous driving and aerial drone navigation, the ability to accurately recognize objects regardless of their orientation is crucial for safe and efficient operation. The integration of the Log-CORDIC algorithm into the preprocessing pipeline of CNNs enhances the reliability of these systems in dynamically changing environments. For instance, in autonomous vehicles, the improved ability to recognize traffic signs, pedestrians, and other vehicles, irrespective of their angle or orientation, directly contributes to improved decision-making and safety protocols.

Advancements in Medical Imaging and Diagnostics In the field of medical imaging and diagnostics, the ability to accurately classify images regardless of rotation is of paramount importance. Medical scans such as MRIs and CT scans can vary in orientation, and the requirement for manual adjustments or re-scans due to rotational issues can be significantly reduced with the use of rotationally invariant CNNs. The application of the enhanced models can lead to more accurate and efficient diagnostic procedures, aiding in early detection and treatment planning. For example, the ability to detect anomalies or changes in tissue structure in scans, without the constraint of standard orientations, can streamline diagnostic workflows and potentially improve patient outcomes.

Impact on Surveillance and Security The implications of this research also extend to the fields of surveillance and security. In these areas, cameras often capture images and footage from varying angles and perspectives. The enhanced rotational invariance of CNNs ensures more accurate and reliable image classification, crucial for threat detection and situational awareness. In scenarios where rapid and accurate interpretation of visual data is essential, such as in public safety monitoring or border security, the implementation of rotationally robust CNNs can lead to more effective surveillance systems, contributing to overall safety and security measures.

7.1.4 Reflection on Research Objectives and Achievements

Reflecting on the initial objectives of this research, it is evident that the integration of the Log-CORDIC algorithm into CNN preprocessing has successfully addressed the challenge of rotational invariance in image classification. The achievement of this objective not only fulfills the primary goal of the thesis but also sets a precedent for future research in the field, highlighting the potential of algorithmic preprocessing in enhancing the capabilities of neural network models.

7.2 Future Work

7.2.1 Enhancing Algorithm Efficiency for Diverse Rotational Angles

Future research should focus on optimizing the Log-CORDIC algorithm to handle a wider array of rotational angles, particularly more extreme rotations. Enhancements in the algorithm's efficiency and precision would ensure consistent model performance across a broader spectrum of orientations. This optimization could involve developing adaptive or scalable versions of the algorithm that dynamically adjust based on the degree of rotation encountered, thereby maintaining high accuracy in classification across diverse scenarios.

7.2.2 Integration with Advanced Neural Architectures

Exploring the integration of the Log-CORDIC preprocessing technique with emerging and advanced CNN architectures offers a promising avenue for future work. Investigating architectures such as EfficientNet, which are known for their balance of accuracy and efficiency, or delving into Transformer-based models, could provide new insights into handling rotational invariance. This exploration may reveal synergies between advanced neural structures and the preprocessing technique, potentially leading to groundbreaking improvements in image classification.

7.2.3 Real-Time Processing and Edge Computing Applications

Given the increasing demand for real-time image processing, particularly in edge computing devices, future studies should investigate the deployment of

the current system in such environments. This involves optimizing both the neural network model and the Log-CORDIC preprocessing pipeline for low-latency operations. The aim would be to enable efficient processing on hardware with constrained computational resources, making the technology accessible for real-time applications like autonomous vehicles or mobile-based image analysis.

7.2.4 Application Across Diverse and Complex Datasets

Extending the validation of the model to more diverse and complex datasets is crucial for future research. By testing the enhanced model on datasets with varying complexities and image types, researchers can better understand the model's performance and limitations. This expansion could involve datasets with higher-resolution images, diverse object categories, or datasets from specialized fields like aerial or medical imaging, providing a comprehensive evaluation of the model's applicability.

7.2.5 Cross-Domain Application Studies

Further research could also explore the application of rotationally invariant CNNs in specific domains where rotational variation is a common challenge. Domains such as satellite imagery, underwater imaging, and medical diagnostics, where images are often captured at various angles and orientations, would benefit significantly from improved rotational invariance. Conducting domain-specific studies would not only demonstrate the practical utility of the research but also could lead to tailored solutions addressing unique challenges in these fields.

7.2.6 Addressing Other Forms of Image Variations

Lastly, there is an opportunity to extend the research to handle other forms of image variations, such as scale changes, translation, or lighting variations. Developing a more holistic solution that can adapt to a variety of image perturbations would significantly enhance the robustness and versatility of image classification models. This approach would move towards a more comprehensive solution, capable of tackling the multifaceted challenges encountered in real-world image classification scenarios.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. DOI: 10.1038/nature14539.

[2] G. Litjens, T. Kooi, B. E. Bejnordi, *et al.*, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, Dec. 2017. DOI: 10.1016/j.media.2017.07.005.

[3] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Dec. 2015. DOI: 10.1109/iccv.2015.312.

[4] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, pp. 649–666. DOI: 10.1007/978-3-319-46487-9_40.

[5] R. He and J. McAuley, "Ups and downs," in *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, Apr. 2016. DOI: 10.1145/2872427.2883037.

[6] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, May 2017. DOI: 10.1145/3065386.

[8] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016. DOI: 10.1109/cvpr.2016.90.

[10] N. Manchev, *Gpu-accelerated convolutional neural networks with pytorch*, Accessed: 2023-10-10, 2022. [Online]. Available: https://domino.ai/blog/gpu-accelerated-convolutional-neural-networks-with-pytorch.

[11] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," Mar. 23, 2016. arXiv: 1603.07285v2 [stat.ML].

[12] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, Springer International Publishing, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

[13] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning," in *2011 International Conference on Computer Vision*, IEEE, Nov. 2011. DOI: 10.1109/iccv.2011.6126474.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[16] Y. Boureau, J. Ponce, and Y. LeCun, "A theoretical analysis of feature pooling in visual recognition," in *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, J. Fürnkranz and T. Joachims, Eds., Omnipress, 2010, pp. 111–118. [Online]. Available: https://icml.cc/Conferences/2010/papers/638.pdf.

[17] J. Kim, W. Jung, H. Kim, and J. Lee, *CyCNN: A rotation invariant CNN using polar mapping and cylindrical convolution layers*, 2020. DOI: 10.48550/ARXIV.2007.10588.

[18] C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. Elsevier, 2004, ISBN: 978-0-7506-7604-5.

[19] K. Compton and S. Hauck, "Reconfigurable Computing: A Survey of Systems and Software," in *ACM Computing Surveys*, vol. 34, Jun. 2002, pp. 171–210. DOI: 10.1145/508352.508353.

[20] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "Large-scale FPGA-based Convolutional Networks," *Machine Learning on Very Large Data Sets*, vol. 1, no. 1, pp. 1–6, Nov. 2011.

[21] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, Feb. 2015. DOI: 10.1145/2684746.2689060.

[22] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959. DOI: 10.1109/tec.1959.5222693.

[23] P. K. Meher, J. Valls, T.-B. Juang, K. Sridharan, and K. Maharatna, "50 years of CORDIC: Algorithms, architectures, and applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 56, no. 9, pp. 1893–1907, Sep. 2009. DOI: 10.1109/tcsi.2009.2025803.

[24] J. Qiu, J. Wang, S. Yao, *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, Feb. 2016. DOI: 10.1145/2847263.2847265.

[25] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays - FPGA '98*, ACM Press, 1998. DOI: 10.1145/275107.275139.

[26] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," in *arXiv:1602.07360*, Feb. 2016. arXiv: 1602.07360 [cs.CV].

[27] S. Han, J. Kang, H. Mao, *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Feb. 2017. DOI: 10.1145/3020078.3021745.

[28] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow, "Harmonic networks: Deep translation and rotation equivariance," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jul. 2017. DOI: 10.1109/cvpr.2017.758.

[29] B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, and M. Welling, "Rotation equivariant CNNs for digital pathology," in *Medical Image Computing and Computer Assisted Intervention – MICCAI 2018*, Springer International Publishing, 2018, pp. 210–218. DOI: 10.1007/978-3-030-00934-2_24.

[30] R. Togo, K. Hirata, O. Manabe, *et al.*, "Cardiac sarcoidosis classification with deep convolutional neural network-based features using polar maps," *Computers in Biology and Medicine*, vol. 104, pp. 81–86, Jan. 2019. DOI: 10.1016/j.compbiomed.2018.11.008.

[31] H. Sofian, J. T. C. Ming, S. Muhammad, and N. M. Noor, "Calcification detection using convolutional neural network architectures in intravascular ultrasound images," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 17, no. 3, p. 1313, Mar. 2020. DOI: 10.11591/ijeecs.v17.i3.pp1313-1321.

[32] T. S. Cohen and M. Welling, "Group equivariant convolutional networks," *International Conference on Machine Learning (ICML), 2016*, 2016. DOI: 10.48550/ARXIV.1602.07576.

[33] J. E. Johnson, S. Sundaresan, T. Daylan, *et al.*, *Rotnet: Fast and scalable estimation of stellar rotation periods using convolutional neural networks*, 2020. DOI: 10.48550/ARXIV.2012.01985.

[34] Y. Umuroglu, N. J. Fraser, G. Gambardella, *et al.*, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, Feb. 2017. DOI: 10.1145/3020078.3021744.

[35] S. Han, J. Kang, H. Mao, *et al.*, "ESE: Efficient Speech Recognition Engine with Sparse LSTM on FPGA," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, Feb. 2017. DOI: 10.1145/3020078.3021745.

[36] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, Jan. 2017. DOI: 10.1109/aspdac.2017.7858394.

[37] N. Suda, V. Chandra, G. Dasika, *et al.*, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ACM, Feb. 2016. DOI: 10.1145/2847263.2847276.

[38] N. Zhang, X. Wei, H. Chen, and W. Liu, "FPGA implementation for CNN-based optical remote sensing object detection," *Electronics*, vol. 10, no. 3, p. 282, Jan. 2021. DOI: 10.3390/electronics10030282.

[39] C. Zhuge, X. Liu, X. Zhang, S. Gummadi, J. Xiong, and D. Chen, "Face recognition with hybrid efficient convolution algorithms on FPGAs," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, ACM, May 2018. DOI: 10.1145/3194554.3194597.

[40] X. Han, D. Zhou, S. Wang, and S. Kimura, "CNN-MERP: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks," in *2016 IEEE 34th*

International Conference on Computer Design (ICCD), IEEE, Oct. 2016. DOI: 10.1109/iccd.2016.7753296.

[41] S. Y. Bonabi, H. Asgharian, S. Safari, and M. N. Ahmadabadi, "FPGA implementation of a biological neural network based on the hodgkin-huxley neuron model," *Frontiers in Neuroscience*, vol. 8, Nov. 2014. DOI: 10.3389/fnins.2014.00379.

[42] S. D. Munoz and J. Hormigo, "High-throughput FPGA implementation of QR decomposition," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 9, pp. 861–865, Sep. 2015. DOI: 10.1109/tcsii.2015.2435753.

[43] N. Neji, A. Boudabous, W. Kharrat, and N. Masmoudi, "FPGA implementation of the CORDIC algorithm for fingerprints recognition systems," *International Journal of Computer Applications*, vol. 63, no. 6, pp. 39–45, Feb. 2013. DOI: 10.5120/10473-5204.

[44] M. Franceschi, V. Camus, A. Ibrahim, C. Enz, and M. Valle, "Approximate FPGA implementation of CORDIC for tactile data processing using speculative adders," in *2017 New Generation of CAS (NGCAS)*, IEEE, Sep. 2017. DOI: 10.1109/ngcas.2017.40.

[45] J. Walther, "A Unified Algorithm for Elementary Functions," in *Proceedings of the Joint Computer Conference*, 1971, pp. 379–385.

[46] J. Boluda and F. Pardo, "Synthesizing on a reconfigurable chip an autonomous robot image processing system," in *Field Programmable Logic and Application*, Y. K. Cheung and G. Constantinides, Eds., ser. Lecture Notes in Computer Science, vol. 2778, Berlin, Heidelberg: Springer, 2003. DOI: 10.1007/978-3-540-45234-8_45.

External Links

[10] N. Manchev, *Gpu-accelerated convolutional neural networks with pytorch*, Accessed: 2023-10-10, 2022. [Online]. Available: https://domino.ai/blog/gpu-accelerated-convolutional-neural-networks-with-pytorch.