NTNU
Norwegian University of
Science and Technology

Neural Network Based Characterization and Feature Extraction from Analogue Radio Signals for Verification Purposes

Morten Olsen Lykkedrang

Master of Science in Electronics
Submission date: June 2018
Supervisor: Kjetil Svarstad, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Sammendrag

Mixed-signal applikasjoner er blant de raskest voksende markeds-segmentene innen elektronikk- og halvleder-industri, og har foresaket at mange silisium produsenter hare mixed-signal designs som en av sine primære fokusområder. Mange SoC design i dag er derfor mixed-signal. En eskalerende kompleksitet i elektroniske kretser har dermed ført til økte utfordringer når det kommer til mixed-signal SoC verifikasjon[1].

Dene avhandlingen utforsker bruken av kunstige nevrale nettverk til å klassifisere strukturelle trekk, samt position og signal-lengde av RVM radio signaler i støyfylte omgivelser. Trening og testing av disse nevrale nettverkene har blitt gjort ved help av MatLab R2018a, hvor de nevrale nettverkene har blitt utviklet ved hjelp av *network* verktøykassen som og er en del av MatLab R2018a. Nettverkene var testet både med å bruke SGDM, RMSProp og ADAM algoritmene under treningsprocessen, samt tatt i bruk inngangsdata både i tids-domenet og frekvens-domenet.

Inngangsdataene var utsatt for hvit Gaussian støy, hvor støynivået i dB er lagt i forhold til en signalkraft lik 25. Dette ga støy-topper rundt 50-55 for $0S_{ref}NR$, hvorav amplituden på radio signalene er satt til 50. Den strukturelle klassifikasjonen oppnådde en nøyaktighet på 0.9623 ved $0S_{ref}NR$, med en klassifiseringstid på $34 * 10^{-4}s$ ved bruk av CPU. En mindre versjon av CNN arkitekturen oppnådde en nøyaktighet på 0.9529 med en klassifiseringstid på $9.92 * 10^{-4}$. Klassifikasjonen av signalets posisjon var gjort gjennom en heirarkisk arkitektur som hadde en total klassifiseringstid på $13.99 * 10^{-4}s$. Denne tiden inkluderer ikke tiden som kreves for å flytte data mellom dem. Denne heirarkiske arkitekturen oppnådde en nøyaktighet på 0.982 for å klassifisere riktig 1bit område for start posisjon, med en 0.7104 nøyaktighet for å klassifisere riktig sample. Lengde-klassifiseringen oppnådde en nøyaktighet på 0.9359, med 0.9970 for å enten klassifisere riktig eller kun være 1 bit fra riktig. Klassifiseringstiden for denne CNN arkitekturen var å $4.49 * 10^{-4}s$ per iterasjon på en CPU. Disse resultatene er alle fra bruk av data fra tids-domenet, som viste en høyere nøyaktighet en bruk av frekvens-domenet for alle testene.

Abstract

Mixed-signal applications are among the fastest growing market segments in the electronics and semiconductor industry, and have caused many silicon manufacturers to have mixed-signal designs as one of their main focuses. Most SoC designs today are therefore mixed signal. With an escalation in circuitry complexity, there are increasing challenges when it comes to mixed-signal SoC verification[1].

This thesis explores the use of artificial neural networks for the classification of structural features, position, and signal length of RVM radio signals in a noisy environment. The training and testing data for the artificial neural networks where generated in MatLab R2018a, with the neural networks themselves being developed using MatLab's *network* toolbox. The networks were tested using the SGDM, RMSProp and ADAM solver algorithms, using both time-domain and frequency-domain input data.

The added white Gaussian noise was set based on a reference signal power of 25, which placed the noise peaks for the test-scenarios around 50-55. The amplitudes for the actual signals were set to 50 throughout all test scenarios. The structural classification achieved an accuracy of 0.9623, with a classification time of $34 * 10^{-4}s$ on a CPU. A smaller version of the CNN achieved an accuracy of 0.9529, with a classification time of $9.92 * 10^{-4}s$ per classification. The positional classification, in a similar environment, managed to pinpoint the position down to a 1 bit region with an accuracy of 0.982, with an accuracy of 0.7104 of classifying the exact sample number. This solution followed a hierarchical approach, using 4 CNNs and a total of $13.99 * 10^{-4}s$ per classification, not counting the time spent transferring data between the networks. The length classification achieved an accuracy of 0.9359, with a 0.9970 chance of being either accurate or 1 bit off. The classification time for this CNN was $4.49 * 10^{-4}s$ for each iteration. All of these results are taken using time-domain input data, which outperformed frequency-domain input-data throughout all test scenarios.

Preface

This report is the result of the Master's thesis conducted during the spring of 2018, concluding a Master of Science degree in Electronics, with a specialization in Embedded Systems. The report is submitted to the Department of Electronic Systems at the Norwegian University of Science and Technology (NTNU).

I would like to thank professor Kjetil Svarstad from the Department of Electronic Systems at NTNU, for valuable support, guidance, and feedback throughout the development process. I would also like to thank my friends and family for their support during both this project, as well as throughout the years spent on the degree.

Contents

List of Figures

List of Tables

Abbreviations

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
AMC	Automatic Modulation Classification
AMS	Analog Mixed Signal
ANN	Artificial Neural Network
ASK	Amplitude Shift Keying
BNN	Binary Neural network
CDV	Coverage Driven Verification
CPU	Central processing Unit
DBPSK	Differential Binary Phase Shift Keying
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
DQPSK	Differential Quadrature Phase Shift Keying
DUV	Device Under Verification
EoF	End of Frame
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
GFSK	Gaussian Frequency Shift Keying
GMSK	Gaussian Minimum Shift Keying
GPU	Graphics Processing Unit
HDL	Hardware Descriptive Language
IC	Integrated Circuit
IP	Intellectual Property
I/Q	In-phase and Quadrature
MFCC	Mel-Frequency Cepstrum Coefficients
OOK	On-Off Keying
PSK	Phase Shift Keying
ReLU	Rectified Linear Unit
RMSProp	Root Mean Square Propagation
RNM	Real Number Modeling
RVM	Real Value Modeling
SDR	Software Defined Radio
SGDM	Stochastic Gradient Descent with Momentum
SoC	System on Chip
SoF	Start of Frame
SSDA	Stacked Sparse Denoising Autoencoders

Chapter 1

Introduction

1.1 Background and motivation

Mixed-signal applications are among the fastest growing market segments in the electronics and semiconductor industry, and has caused many silicon manufacturers to have mixed-signal designs as one of their main focuses[1]. Most system-on-chip (SoC) designs today are therefore mixed-signal. With a continuous escalation in circuit complexity, there are increasing challenges when it comes to mixed-signal SoC verification. These include both incomplete SoC-level and system level verification, as well as uncertainties in coverage. A report [1] by *Cadence Design Systems* claims that, according to industry estimates, over 60% pf SoC design re-spins at 45nm and below are due to mixed-signal errors, with functional verification for digital ICs now taking up 70% of the logic design phase. Adding analog and mixed-signal IP makes this task even more complex, resulting in verification and simulation never being fast enough.

As described in the project assignment, the stimuli of the circuit is dynamically decided by a controller based on on line monitoring of the response from the Design Under Verification (DUV) in UVM based verification. While this works well for the digital domain, it is difficult to adapt for the analogue domain such as radio signals. Realistic in Spice and analog mixed signal (AMS) are typically too slow for this purpose, and difficult to integrate in UVM. Instead. real value modeling (RVM) may be a relevant alternative for verification of digital and analog mixed behavior. The challenge then becomes to extract relevant features from the real value signal which can be used for monitoring and controlling the testbed.

1.2 Contribution

This thesis aims to examine the use of artificial neural networks (ANNs) to extract relevant features from a RVM signal as a potential step in the UVM verification process. As ANNs have been utilized in areas, such as image classification and voice recognition, the general strategy has been to look at the solutions used for those domains and utilize these as a basis for a solution regarding RVM signals. The assessment of each solution will primarily be based upon the solution's accuracy, complexity in terms of memory size and operation count, as well as the execution time.

Relevant features to look for in mixed-signal radio circuitry may range from anywhere between correct signal power, frequency, rise and fall time, to more digitally leaned characteristics such as delay, and features contained in the signal's structure. In this thesis, it has been chosen to examine the extraction of structural features, delay in form of the sample position equating to the start of a signal, as well as signal length. The RVM signals will be inserted into the artificial neural networks as either raw time-domain data, or as frequency domain-data with or without the phase component. A number of fully connected and convolutional architectures will be explored in order to find a suitable neural network architecture.

In more detail, the features that will be looked at are:

- Signal structure - This part primarily concerns with a ANNs ability to recognize certain patterns or parts of a RVM signal. It concerns the existence of the correct header sequence and the existence of a data payload. It also concerns with determining the existence of the package's end point, or EoF, as well as the start of frame (SoF) delimiter placed between the header and the data payload. The checks for the SoF delimiter also serves as a test to see the ANNs ability to recognize the existence of identifiers and similar small changes within a RVM signal.
- Signal start positions - One aspect of the verification process that's often important to verify is the verification of correct delay at different points of the design. Since in this case the neural network will be unaware of the delay of the initial sampling data, it will instead give out the position of where the it believes the first data sample resides. This should allow the rest of the system to determine the actual delay. The distance between the start and end point of the signal may also give information regarding correct length or frequency.
- Signal length - In some ways connected to the classification of signal position, the length of the RVM signal may help determine whether or not all parts of the desired signal are present, or whether or not the design output is of the desired output frequency when compared to the sampling frequency.

All RVM signals will be expected to exist in a noisy environment, and will be subjected to white Gaussian noise before entering the artificial neural network.

1.3 Structure of thesis

Chapter 2: Background elaborates on the use analogue and mixed-signals in the verification process, including the use of RVM. The chapter also introduces key concepts for ANNs and its components. Chapter 3: Related work will look at earlier studies and publications within the field of ANNs and radio signal feature extraction. Chapter 4: Architecture and test development provides information regarding the different test cases, including how the systems were built and how data for them were generated. Chapter 5: Analysis contains the resulting data generated from the architectures discussed in Chapter 4, as well including comparisons for their performance. Chapter 6: Discussion discusses the the results and comparisons described in Chapter 5, as well as potential strengths and weaknesses for the different solutions, as well as for the development process. The thesis concludes with Chapter 7: Conclusion, which proposes some thoughts and possible routes for future development within the same field. Further details regarding the neural networks explored in this thesis can also be found in the appendices.

Chapter 2

Background

2.1 UVM based verification

The Universal Verification methodology (UVM) is a complete methodology that codifies the best practices for efficient and exhaustive verification [2], with the goal of helping developers find more bugs earlier in the design process. UVM aims to develop reusable verification components, and is targeted to both verify small designs and large-gate count IP-based system on chip (SoC) designs. UVM is an open sourced format primarily based on the Open Verification Methodology (OVM) library, and has been tested to work on all major commercial simulators[2].

UVM provides the ability to cleanly partition a verification environment into a set of specific components. It provides classes and infrastructure to enable fine-grain control for sequential data stimulus generation, both for the module and system level, and provides built-in stimulus generation, which can be customized. The UVM base classes are made to provide automation and help streamline usage, allowing the creation of hierarchical reusable environments[2].

UVM is based around providing coverage driven verification (CDV), which combines automatic test generation, self-checking testbenches, and coverage metrics to reduce time spent verifying a design. A full description of UVM and its classes can be found in *Universal Verification Methodology (UVM) 1.2 User's Guide*[3] and *Universal Verification Methodology (UVM) 1.2 Class Reference*[4] released by *accellera Systems Initiative.*

2.2 SPICE, Analog Mixed Signals and Real Value Modeling for verification purposes

Mixed-signal applications are today one of the fastest growing segments in the electronics and semiconductor industry. Electronic equipment is both expected to do more, and to have a wider specter of operation. Growth opportunities in a wide array of electronic equipment has probed many silicon vendors into refocusing their business on RF, high-performance analog, and mixed-signal designs. Due to this trend, most SoC designs today are mixed signal, with all, or close to all, being mixed signal at advanced process nodes in the near future[1]. [1] states that, according to industry estimates, over 60% of SoC design re-spins at 45nm and below are due to mixed-signal errors. Many of these re-spins are caused by commonplace, avoidable errors such as inverted or disconnected signals.

Mixed-signal verification is still primarily done by SPICE simulations, where [5], [6] and [1] all agree that SPICE based verification is too slow to the point of chip-level simulations being impractical. Because of this, Analog Mixed-Signal (AMS) and Real Value/Number Modeling (RVM/RNM) techniques have been introduced in order to enhance simulation speed. [5] illustrates that UVM is suited for both AMS and RVM simulations. While AMS and RVM models offer progressively higher performance, it is worth noting that they give an extra cost when it comes to developing them [5]. [6] offers a comparison for the different methods. As stated, the virtual prototypes based on purely digital models and model description give the highest verification speed but may not offer an efficient way to capture analog behavior, which is often an integral part of the embedded system. It is also worth noting that while simulations that are purely digital are faster than real value modeling (RVM), they can only represent an analog signal as a single logic value, which may only be sufficient for connectivity checks[1]. RVM gains an advantage over AMS as it does not require an analog solver [5]. Instead, RNVM utilizes discrete floating-point real numbers in order to enable the user to describe an analog block as a signal-flow model, which can be simulated in a digital solver at near-digital simulation speeds[1]. While this restricts RVM to a signal flow approach, it means the issue of analog convergence becomes less of a problem due to not requiring an analog solver.

2.3 Machine learning and neural networks

2.3.1 Machine learning

Artificial neural networks (ANNs) are one type of machine learning which has seen much practical value in the field of pattern recognition. Artificial intelligence (AI), and with it machine learning, is today a thriving field with numerous active research topics[7]. Early implementations primarily concerned solving problems that could be described by a list of formal, mathematical rules. In contrast, the challenges for AI and machine learning nowadays often entail problems that may be simple for a human to solve, yet difficult to describe formally. This includes problems like recognizing spoken words or recognizing images. While these tasks are often trivial for a human being to solve, yet hard to describe to a computer. AI aims to solve this by allowing computers to learn from previous experiences. As defined by Tom M. Mitchell in [8] *"The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience"*. ANNs are one rapidly growing subset of AIs, that aims to solve these problems through mimicking the human brain by creating neural structures to handle decision-making

2.3.2 Artificial neural networks

Perhaps the easiest way to describe how a ANN works if by describing a type of artificial neuron called a *perceptron*. Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts[9]. A perceptron takes a number of binary inputs x and produces a single binary output y, based on the values of said inputs. Each input has an associated weight w, which determines the importance each input has to to the output. The output value for a perceptron is determined by whether the weighted sum of the inputs are above a threshold value, as described in Equation 2.1.

$$output = \begin{cases} 0, & \text{if } 0 \sum_j w_j x_j \leq threshold \\ 1, & \text{if } 0 \sum_j w_j x_j > threshold \end{cases} \tag{2.1}$$

Modern ANNs are in many ways similar to Rosenblatt's perceptrons. However, instead of using threshold values, *biases* are introduced. A *bias* can be interpreted as a measure regarding how easily one can turn a specific perceptron's output to '1'. An activation function is also introduced, who's purpose is to make sure small changes made to weights and biases only cause a small change to the network's output. One additional change these have when compared to perceptrons is that outputs are no longer limited to the binary values '0' and '1', but can instead take any value between 0 and 1[9]. The form the activation function takes depends on the type of neuron, with a common type being the *sigmoid* function. The new definition for this can be seen in Equation 2.2, with weights w_i, inputs x_i and bias b. The activation function for the sigmoid neuron can be seen in Equation 2.3, where z in this case equates to Equation 2.2.

$$y = f \sum_i (w_i + x_i + b) \tag{2.2}$$

$$\sigma = \frac{1}{1 + e^{-z}} \tag{2.3}$$

The neurons that constitute to making a neural network are set up in *layers*. The first of these layers is commonly referred to as the *input* layer, with the last layer commonly being referred to as the *output* layer. The layers in between are called *hidden* layers. The number of hidden layers, the number of neurons they contain, as well as how these neurons are connected to both the previous and the following layers depends on the type of layer in question. These layers will be explained more in detail in Chapter 2.3.4. Neural networks where all layer inputs derive from the previous layer are labeled as *feedforward* networks, with networks that allow a cyclic form being labeled as *recurrent* neural networks. This thesis will only focus on the *feedforward* networks[9].

The performance of convolutional neural networks (CNNs) in particular have improved significantly in recent years, to the point where they now outperform other visual recognition algorithms, as well as human accuracy on certain problems[10].

2.3.3 Training a neural network

The training of a neural network involves an optimization process of the network's parameters, such as weights and biases, in a process that essentially trains the network to recognize the desired parameters through the correct output neuron.

As described in [11], in order to train a neural network with a set of input vectors x_n where $n = 1, ...N$, together with a corresponding set of target vectors t_n, one must minimize the error function shown in Equation 2.4

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} ||y(x_n, w) - t_n||^2 \tag{2.4}$$

An algorithm is employed in order to minimize the error function. The primary objective of this function is to alter the values of the network's parameters, particularly weights and biases, by updating them with small steps. Algorithms that handle this operation include Stochastic Gradient Descent (SGD), alternatively Stochastic Gradient Descent with Momentum (SGDM), root mean square propagation (RMSProp), and adaptive moment estimation (ADAM)[12] . The equations for these algorithms can be seen in Equation 2.5 (SGDM), 2.6 (RMSProp) and 2.7. (ADAM)[13].

$E(\theta)$ is for these cases the loss/error function, with $\nabla E(\theta)$ being its gradient. α acts as the learning rate, θ as the parameter vector, with ι being the iteration number. In addition, SGDM utilizes the γ parameter, which determines the contribution of the previous gradient, and is specified before training starts. RMSProp and ADAM utilizes decay rates, notated as β_1 and β_2, that are also specified before training starts. ϵ is a small constant added to avoid division by zero [13].

$$\theta_{\iota+1} = \theta_\iota - \alpha \nabla E(\theta_\iota) + \gamma(\theta_\iota - \theta_{iota-1}) \tag{2.5}$$

$$\theta_{\iota+1} = \theta_\iota \frac{\alpha \nabla E(\theta_\iota)}{\sqrt{v_\iota} + \epsilon} \tag{2.6}$$

$$\theta_{\iota+1} = \theta_\iota - \frac{\alpha m_\iota}{\sqrt{v_\iota} + \epsilon} \tag{2.7}$$

$$m_\iota = \beta_1 m_{\iota-1} + (1 - \beta_1) \nabla E(\theta_\iota) \tag{2.8}$$

$$v_\iota = \beta_2 v_{\iota-1} + (1 - \beta_2)[\nabla E(\theta_\iota)]^2 \tag{2.9}$$

An algorithm known as the *backpropagation algorithm* is responsible for computing the gradient of the cost/loss function. While this will not be explained in detail here, the essence of it entails first doing a forward pass using an input vector x_n to find the activations for all the neurons. The error values on the outputs are then propagated backwards through the network, which are used to calculate the gradients and perform updates[11][9]. The number of times the training vectors are used once to update the network's weights is referred to as an *epoch*.

2.3.4 Neural network layers

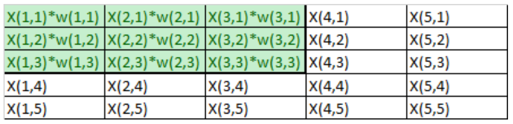
As previously mentioned, a neural network consists of a number of layers where both the method for how they connect to the previous layer, as well as how they use their input parameters, depends on the type of layer. This section will describe how some of these layers operate. Further information regarding each layer, and ANNs in general, can be found in [14] and [15].

The first of these layers is the *convolutional* layer, which is the core building block for a CNN. The convolutional layer operates by having one or more filters move along the layer's input data, handing one section of the data at the time. Each filter has a number of weights equal to the product of the filter's dimensions, as well as a single bias. These values used for the weights and the bias remains the same while the filter traverses the layer's inputs. The important parameters for a convolutional layer are the following:

- The number of filters, with each filter having its own weights and bias.

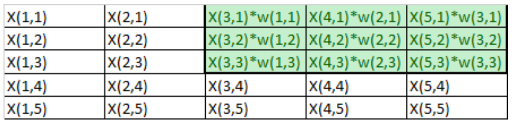
- The dimensions of these filters.
- The filters' stride, which refers to the step size moved between each activation.
- The padding, which is rows or columns of zeroes added to the borders of the inputs

Each filter operates independently from the others, and produces its own set of neurons to be used by the next layer in the neural network. The primary purpose of adding *padding* to the layer's input is to increase the significance of the input neurons located near the edges and corners of the previous layer. Convolutional filters are generally two-dimensional, meaning the number of weights scales with the number of channels (the 3rd-dimension) of the previous layer. As an example, should a convolutional filter have the dimensions [3 3], with the previous layer having 4 channels, the convolutional filter will in total have 3x3x4 weights, and create a single output for each activation in the height and width dimensions. A simple example of how the convolutional filter moves can be seen in Figure 4.1.



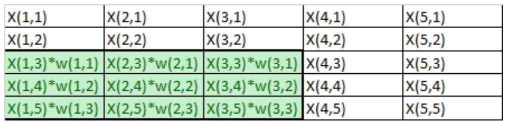
X(1,1)*w(1,1)	X(2,1)*w(2,1)	X(3,1)*w(3,1)	X(4,1)	X(5,1)
X(1,2)*w(1,2)	X(2,2)*w(2,2)	X(3,2)*w(3,2)	X(4,2)	X(5,2)
X(1,3)*w(1,3)	X(2,3)*w(2,3)	X(3,3)*w(3,3)	X(4,3)	X(5,3)
X(1,4)	X(2,4)	X(3,4)	X(4,4)	X(5,4)
X(1,5)	X(2,5)	X(3,5)	X(4,5)	X(5,5)

(a) Position 1



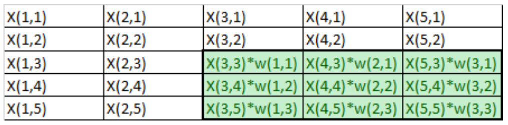
X(1,1)	X(2,1)	X(3,1)*w(1,1)	X(4,1)*w(2,1)	X(5,1)*w(3,1)
X(1,2)	X(2,2)	X(3,2)*w(1,2)	X(4,2)*w(2,2)	X(5,2)*w(3,2)
X(1,3)	X(2,3)	X(3,3)*w(1,3)	X(4,3)*w(2,3)	X(5,3)*w(3,3)
X(1,4)	X(2,4)	X(3,4)	X(4,4)	X(5,4)
X(1,5)	X(2,5)	X(3,5)	X(4,5)	X(5,5)

(b) Position 2



X(1,1)	X(2,1)	X(3,1)	X(4,1)	X(5,1)
X(1,2)	X(2,2)	X(3,2)	X(4,2)	X(5,2)
X(1,3)*w(1,1)	X(2,3)*w(2,1)	X(3,3)*w(3,1)	X(4,3)	X(5,3)
X(1,4)*w(1,2)	X(2,4)*w(2,2)	X(3,4)*w(3,2)	X(4,4)	X(5,4)
X(1,5)*w(1,3)	X(2,5)*w(2,3)	X(3,5)*w(3,3)	X(4,5)	X(5,5)

(c) Position 3



X(1,1)	X(2,1)	X(3,1)	X(4,1)	X(5,1)
X(1,2)	X(2,2)	X(3,2)	X(4,2)	X(5,2)
X(1,3)	X(2,3)	X(3,3)*w(1,1)	X(4,3)*w(2,1)	X(5,3)*w(3,1)
X(1,4)	X(2,4)	X(3,4)*w(1,2)	X(4,4)*w(2,2)	X(5,4)*w(3,2)
X(1,5)	X(2,5)	X(3,5)*w(1,3)	X(4,5)*w(2,3)	X(5,5)*w(3,3)

(d) Position 4

Figure 2.1: Example of convolutional filter with stride [2 2]

Another common layer type, and perhaps the easiest to understand, is the *fully connected* layer. As the name implies, each of the N neurons in a fully connected layer are directly connected to the M neurons in the previous layer. Each of these N neurons have a unique *weight* for each input in M, as well as having its own bias. This means for neural networks with large layers, a fully connected layer will require enormous amounts computations as well as physical memory.

The rectified linear unit are usually placed after either a convolutional or batch normalization layer. It performs a threshold operation to each neuron, setting any input value that is less than zero to zero, as can be seen in Equation 2.10[14].

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x >= 0 \end{cases} \tag{2.10}$$

The batch normalization layer is commonly placed between convolutional layers and rectified linear unit layers. The batch normalization layer aim to normalize the activations and gradients that propagate though a network, in order to make network training into an easier optimization problem [14][16]. The layer works by subtracting the mean μ_B of each mini-batch, and dividing it by the mini-batch's standard deviation σ_B^2. The inputs are then scaled by a factor γ and shifted by β. An ϵ parameter is also added for stability. The formula for the batch normalization layer can be seen in

Equation 2.11. γ and β are trained variables, with μ_B and σ_B^2 being replaced with values that apply the the entire training set, instead of just one mini-batch, after network training has completed.

$$x_i = \gamma \frac{\tilde{x_i} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \tag{2.11}$$

Chapter 3

Related Work

This chapter will present some recent works that are of interest when it comes to utilizing artificial neural networks to extract features from a RVM signal. While were no articles for using ANNs to extract features from a radio signal utilizing RVM for verification purposes, there were still papers that can be considered relevant. Section 3.1 entails different approaches for automatic modulation classification (AMC) of radio signals and software-defined radio (SDR). Section 3.2 touches briefly on the use of ANNs in voice recognition. While not directly related to radio signals, both instances entails using a stream of floating point values representing amplitude values. Section 3.3 briefly touches on the subject of binary neural networks (BNNs)

3.1 Modulation recognition in cognitive radio using artificial neural networks

3.1.1 Biologically Inspired Radio Signal Feature Extraction with Sparse Denoising Autoencoders

Automatic modulation classification (AMC) has become an important task for communication systems in the later years, with the challenge being when signal features and precise models for generating each modulation may be unknown[17]. [17] utilizes in-phase and quadrature (I/Q) signals acquired together with stacked sparse denoising autoencoders (SSDAs) to generate features, and then use those features to perform automatic modulation classification. Using this method, they managed to achieve $> 99\%$ correct classification at 7.5 dB signal-to-noise ratio (SNR), and > 92 at 0 SNR, with as few as 100 I/Q timepoints in a 6-way classification test. The six modulation schemes were:

- On-off Keying (OOK)
- Gaussian frequency-shift keying (GFSK)
- Gaussian minimum-shift keying (GMSK)
- Differential binary phase-shift keying (DBPSK)

- Differential quadrature phase-shift keying (DQPSK)
- Orthogonal frequency-division multiplexing (OFDM)

3.1.2 Automatic recognition of both inter and intra classes of digital mdoualted signals using artificial neural network

[18] developed an AMR classifier used for classification of five digital modulation formats: 2ASK, 4ASK, 2FSK, BPSK, and QPSK. As opposed to the method used in [17], this paper chose to look extract information contained in the instantaneous amplitude, phase, and frequency of the incoming radio signal. The four values extracted from those were:

- γ_{max} - the maximum value of the power spectral density of the normalized-centered instantaneous amplitude of the intercepted signal segment.
- σ_{ap} - the standard deviation of the absolute value of the centered non-linear component of the instantaneous phase t time instant t.
- σ_{dp} - the standard deviation of the direct value of the centered non-linear component of the direct instantaneous phase.
- σ_{aa} - the standard deviation of the absolute value of the normalized centered instantaneous amplitude

These extracted valued allows one to distinguish between the different modulation schemes by placing them into subsets based on which modulation schemes contain, and which doesn't contain, information in each of these parameters. As an example γ_{max} can be used to distinguish 2FSK from the other four modulation schemes in question, as 2FSK contains no amplitude information. Utilizing a multi-layer feed-forward neural network, they managed to achieve $> 99\%$ correct classification at an SNR above 5dB, and $> 98\%$ correct classification at SNR values as low as -5dB.

3.1.3 Algorithms for Automatic Modulation Recognition of Communication Signals

[19] encompasses a more expansive, yet similar, version of the method for AMR utilizing information contained in the instantaneous amplitude, phase, and frequency of the incoming radio signal. Utilizing a structure of three connected ANNs, and a total of 9 parameters derived from the instantaneous amplitude, phase, and frequency, they managed to achieve correct classification rate between 13 different modulation schemes of $> 96\%$ at the SNR of 15dB.

3.2 Voice recognition

Speech signal recognition commonly utilize Mel-frequency cepstrum coefficients (MFCC) to, which conveys vocal tract characteristics. The method extracts information from short time intervals using Discrete Fourier Transform (DFT)). The DFT results then go through a bank of Mel-spaced triangular filters, with the Mel-scale being linear below 1000Hz, and logarithmic above 1000Hz.

Lastly, as Duscrete Cosine Transform (DCT) is applied to the bank of the filter bank energies. [20] reports having successfully implemented this using a convolutional neural network (CNN).

3.3 FINN

FINN: A Framework for Fast, Scalable Binarized Neural Network Interface[10], compares the performance of binary neural networks (BNNs) to that of CNNs. As opposed to CNNs, that normally utilize 32bit floating point parameters, BNNs, as the name describes, instead utilize binary values in either a fully binarized network (full BNN), or partly binarized networks. FINN achieved an accuracy of 95.8% for the MNIST dataset with 12.3 million image classifications per second. The platform used was a ZC706 FPGA platform, which drew less than 25 W of total system power. They also achieved a 80.1% accuracy with the CIFAR-10 dataset, and a 94.9% accuracy with the SVHN dataset. Their prototypes maintained an accuracy within 3% of other low-precision works, which they state could have been improved by using larger BNNs[10].

Chapter 4

Architecture and Test Development

4.1 RVM features, parameters and test-cases

The neural networks focuses on verifying three primary features that may be of interest in when it comes to verification of a RVM signal:

- Structural signal components
- Signal position/delay/timing
- Signal length

As was mentioned in Chapter 3, no previous cases of extracting radio signal features utilizing ANNs and RVM were found. The papers regarding cognitive radio in Chapter 3.1 describes both utilizing the frequency domain, as well as higher order parameters derived from the instantaneous amplitude, phase, and amplitude. However, since the classifications here assumes only a single modulation scheme is used for each data-set, the higher order parameters may therefore not be useful. Voice recognition also includes the use of a DFT to translate the signal to the frequency-domain. However, as the signals used in this thesis will be mostly randomized, looking going through additional filters to find something akin to vocal tracts had been deemed unnecessary. It has therefore been chosen to look at the ANNs ability to extract information from both time-domain and frequency-domain data. For the case of frequency-domain, this will include versions both with and without including the phase parameter.

In order to keep the scenarios as realistic as possible, the signals has been subjected to a constrained randomization process. For all of the three classification scenarios mentioned, the position the signal has within the RVM samples has been randomized, including having the samples be taken at different points of the sinusoidal wave for each instance to signify the case where the receiver has not yet synchronized with the signal. There will also be a comparison between basic ASK (OOK), FSK, and PSK. The FSK will for these cases have a doubling in frequency to differentiate between logical '1' and '0', with PSK following a binary form where the phases are 180 degrees apart. The following cover points were made for each feature in an attempt to find a suitable neural network structure for each of them:

- Comparison between the SGM, RMSProp and ADAM solver algorithms
- Comparison between time-domain input data and FFT input data. In the case of using FFT, this included a version both with and without including the phase parameter (imaginary part).
- Comparison between the accuracy when utilizing ASK (OOK), FSK and PSK modulation schemes for the RVM signal.
- Comparison between CNN and FNN
- Comparison between different ratios of sampling frequency and signal frequency, here referred to as "samples per bit"
- Comparison of performance between differently sized CNNs, including:
  - Varying number of convolutional layers
  - Varying sizes of filters, as well as varying number of filters
  - Varying amounts of padding
  - Varying amounts of stride

4.2 Generation of training and classification data

As no existing databases containing relevant data were found, these had to be generated for the neural networks before training or validation could begin. The RVM signals used as input parameters were generated using MatLab R2018a[21]. Each feature that were to be examined utilized the same method for constructing the training and testing data-sets.

The data generation started out with a procedure that goes through each of the neural network's intended output neurons, and creates appropriate input data that is randomized within that output's valid ranges. The parameters chosen here include:

- The appropriate modulation scheme
- The total number of samples, measured in how many bits the area is supposed to cover.
- The RVM signal as a binary sequence.
- The starting position of the RVM signal within the sampled region.
- The amplitude of the signal
- The signal to noise ratio
- The ratio between the signal's frequency and the sampled frequency. In other words, how many samples are used to cover each binary value of the RVM signal.
- The number of periods in the sinusoidal wave for a single bit of the RVM signal.

- At which points of the sinusoidal wave the samples are to be taken. This is meant to represent the case where the receiver and the signal have not been synchronized.

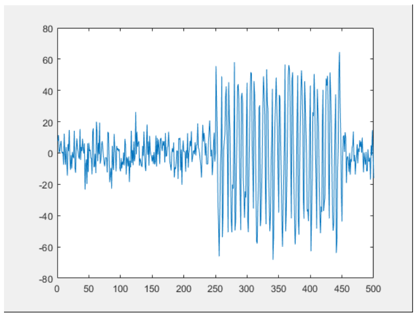
A separate function is called that uses the above-mentioned parameters to transform the binary sequence into amplitude values, as is required for a RVM signal. The formula used can be seen in Equation 4.1. A_m, f and ϕ are here parameters used to describe the ASK (OOK), FSK, and PSK modulation schemes. Each test scenario will only ever utilize one of these modulation schemes for a given data-set. $N_{PeriodsPerBit}$ is in this case an additional parameter with the same function as the FSK parameter f, but is separate for clarity. ϕ_{skew} is the parameter responsible for altering which points of the sinusoidal ware the samples are taken at. This value always remain within the region covered by a single sample, as higher values would equate to adding a n-sample delay, which would look identical to the system yet complicate the testing procedure due to the starting position no longer being at the sample specified. $M_{BitSampleNumber}$ and M_{Bit} are sequences holding a number of elements equal to the number of samples used to describe one bit of data. The values of $M_{BitSampleNumber}$ are equally spaced floating point values in the range [0 1].

$$M_{Bit} = A_m * cos(2\pi f N_{PeriodsPerBit} M_{BitSampleNumber} + \phi + \phi_{skew}) \tag{4.1}$$

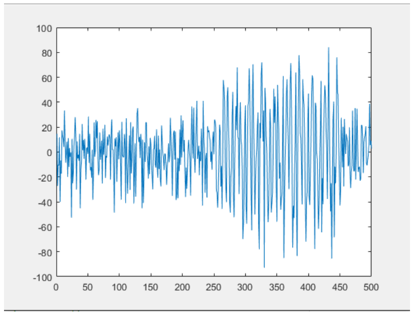
The RVM sequences for each of the signals were then concatenated together and placed at the appropriate located within the sampled region. Values outside the region covered by the signal were kept at the value 0. Lastly, the entire signal is subjected to white Gaussian noise. In order to keep the actual noise values constant for each iteration, the SNR were set when compared to a signal with a signal power of 25, with the actual signals commonly being given an amplitude of 50. As a comparison, with a signal power of 25, the values of the noise levels can be seen in the table below. Keep in mind the values are rough estimates with slight changes from case to case.

$S_{ref}NR$ (dB)	Average noise level (absolute value)	Peak noise level (absolute value)
6 dB	7.5	27
0 dB	15	55
-6 dB	27.5	110

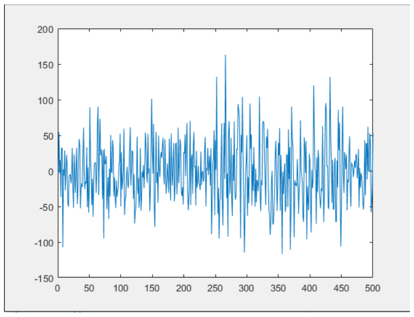
A visual representation can be seen in Figure 4.1. The case used covers an area of 50 bits, with 10 samples being used to cover the period of 1 bit. The signal included utilizes the PSK modulation scheme, with a 20 bit long sequence of alternating '1's and '0's starting at sample number 250.



(a) $S_{ref}NR$ of 6



(b) $S_{ref}NR$ of 0



(c) $S_{ref}NR$ of -6

Figure 4.1: Visual representation of a RVM signal with different levels of white Gaussian noise added

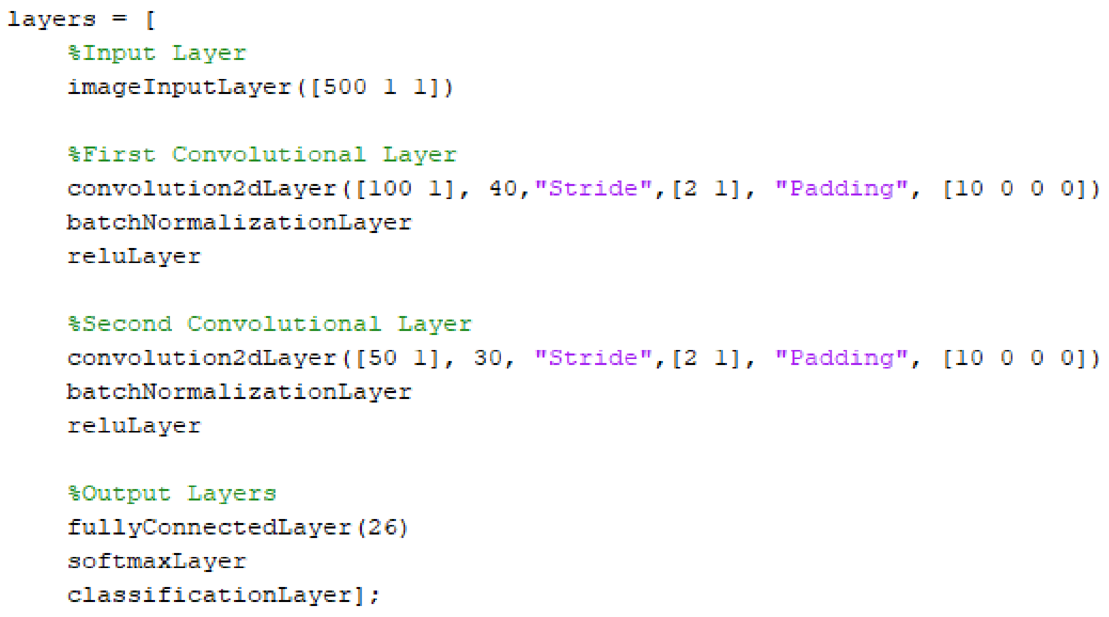
4.3 Neural network structure, training and validation

The ANNs developed primarily took the form of convolutional neural networks (CNNs), that utilized the layer sequence of convolutional layer, batch normaliation layer, and ReLu layer, repeated for each convolutional layer. A comparison where the convolutional layers were swapped out with fully connected layers were also made. The solutions were developed utilizing the *network* toolbox included in MatLab R2018a[15], which contains convenient tools for building, training and veryfying the operations of ANNs.

Matlab R2018a includes convenient tools and functions that simplifies the specifications of the neural network layers. An example for creating the structure of a CNN can be seen in Figure 4.2 The first convolutional layer here consists of 40 filters with dimensions [100 1], with a stride of [2 1] and padding of [10 0 0 0].

The same network as shown in Figure 4.2 can be seen in more detail in Figure 4.3, which utilizes the *neural network analyzer* extension for the *network* toolbox to extract information such as layer size and number of weights and biases for for each layer.

The *network* toolbox's also includes functions for training the neural network, as well as using it to classify a test-set. Both of these functions takes a 4-dimensional matrix as input, where iterations are stacked along the 4th dimension. The network training function also takes includes parameters for training and an output matrix corresponding to the 4-dimensional data-matrix as inputs in order to train the network. The networks are set to train for a maximum of 100 epochs.



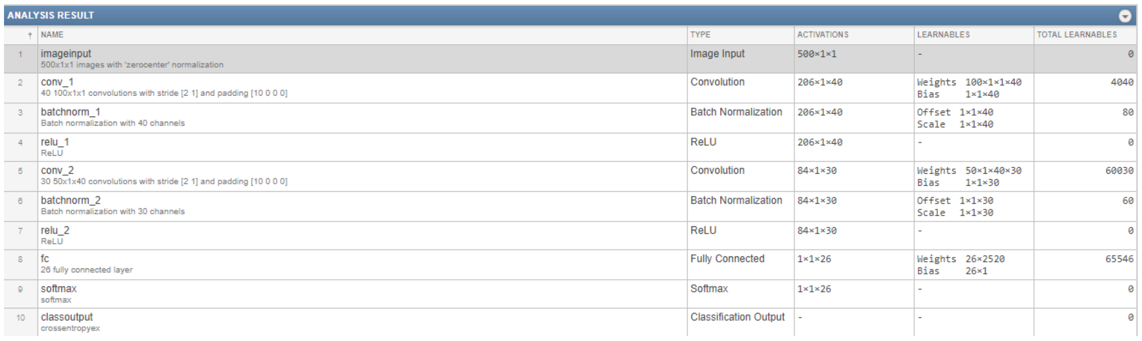
```
layers = [
    %Input Layer
    imageInputLayer([500 1 1])

    %First Convolutional Layer
    convolution2dLayer([100 1], 40,"Stride",[2 1], "Padding", [10 0 0 0])
    batchNormalizationLayer
    reluLayer

    %Second Convolutional Layer
    convolution2dLayer([50 1], 30, "Stride",[2 1], "Padding", [10 0 0 0])
    batchNormalizationLayer
    reluLayer

    %Output Layers
    fullyConnectedLayer(26)
    softmaxLayer
    classificationLayer];
```

Figure 4.2: MatLab2018 syntax for specifying neural network layers



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES	TOTAL LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-	0
2	conv_1 40 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×40	Weights 100×1×1×40 Bias 1×1×40	4040
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40	80
4	relu_1 ReLU	ReLU	206×1×40	-	0
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30	60030
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30	60
7	relu_2 ReLU	ReLU	84×1×30	-	0
8	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×2520 Bias 26×1	65546
9	softmax softmax	Softmax	1×1×26	-	0
10	classoutput crossentropyex	Classification Output	-	-	0

Figure 4.3: Example of CNN structure

4.4 RVM structural classification

As previously mentioned, the structural test-case had as its main objective to determine to what extent the CNN was capable of recognizing certain sequences and identifiers when the signal is placed at a random position with in a sampled range of a noisy environment. The specific points were the following:

- Determine if the preamble sequence consisting of alternating '1's and '0's is present in the signal
- Determine whether or not the start of frame (SoF) delimiter, here recognized as a '11' sequence at the end of the preamble, is present.
- Determine whether the signal contains any data bits (not including the preamble)

- Determine whether or not the entire signal is present in the sampled region. In other words, look for the end of frame (EoF).

The outputs were specified shown below, with the letters signaling the presence of the preamble sequence (P), SoF delimiter (S), data packet (D), and EoF (E). Some combinations of these have not been included, due to them not being possible. As an example, one cannot detect a SoF delimiter without having the preamble and a data sequence both be present.

- 0: *No signal detected*
- P: *Only preamble detected*
- D: *Only data detected. End-point **not** present within sampled data*
- DE: *Only data detected. End-point found within sampled data*
- PD: *Preamble and data detected. Correct SoF delimiter not found. End-point **not** found within sampled data*
- PDE *Preamble and data detected. Correct SoF delimiter **not** found. End-point found within sampled data*
- PSD: *Preamble and data detected. Correct SoF delimiter detected. End-point **not** found within sampled data*
- PSDE: *Preamble and data detected. Correct SoF delimiter detected. End-point found within sampled data.*

The input data generated for the neural network were done following the method described in Chapter 4.2. For this case, the parameters were set as follows:

- The modulation scheme were set to either ASK (OOK), FSK, or PSK. However, only one were ever used during the same set of data.
- The sampled area were set to cover 50 bits worth of data, with the actual number depending on the number of samples used to describe 1 bit of data (the ratio between signal frequency and sample frequency)
- The RVM signal was set as a binary sequence of length 0-25, not including the preamble. The binary sequence for the data part of the signal was set to be entirely randomized for the FSK and PSK modulation schemes. For the ASK (OOK) modulation scheme, the last bit was always set to a logical '1', as the network would be unable to see trailing zeroes for this scheme. The sequence was re-randomized should the data sequence turn out identical to the preamble.
- The starting position was randomized, with the constraint being that the last bit of the sequence had to have at least one sample in the sampled region. The start of the signal was always placed within the sampled region.
- The amplitude was chosen to a constant of 50 throughout all tests. The exact value of 50 holds no significant importance outside the ratio between it and the signal power used for the SNR.
- The $S_{ref}NR$ was varied between -6 dB and 6 dB, compared to a signal with a power of 25.

- The ratio between the sampled frequency and the frequency of the signal was kept at a rate of 4-20 samples for each bit of data.
- The number of periods on the sinusoidal wave used to represent a single bit was kept at a constant of 1, with the logic behind this being that systems where each binary value is translated into several periods equate to having the same binary value repeated for this test case. As an example, the binary sequence '11' will in this case seem identical to a binary '1' in a system that utilizes two periods on the sinusoidal wave to represent one binary value.
- The points of where the samples are taken along the sinusoidal wave were randomized within the region covered by each sample, and denote the case where the receiver has not yet synchronized to the received signal.

4.5 RVM positional classification

The positional test-case had as its function to help estimate timing and delays for the device under testing (DUT). Due to the neural network not knowing the actual sampling frequency, but rather just the ratio between signal and sampling frequency, as well as not knowing how much of a delay there's been before the first sample in the sampled region, the neural network instead focuses on determining at which sample the signal starts in a noisy environment.

The proposed solution for this test-case follows a hierarchical approach. This was done both to limit the total number of output neurons, as well as to allow the networks with larger amounts of input neurons to do more coarse grain classification. The hierarchy went as described below:

- The first neural network is set to take in a sampled region capable of holding at most 100 bytes worth of data, and is tasked with ascertain in which 10 byte region the starting point of the signal is located.
- The second neural network is set to take the 10 byte region ascertained by the first neural network, and go through the same process to ascertain the starting position down to a 1 byte region.
- The third neural network continues the process described above to ascertain the starting position down to a 1 bit region.
- The forth neural network will then lastly ascertain which of these samples acts as the starting point for the signal.

As a more concrete example, one can consider a 100 byte region where the sampling frequency is 10 times higher than the time-period used for each bit, meaning each bit is described within 10 samples. This causes the first neural network to have a total of 8000 input neurons, where it is tasked to ascertain in which region of 800 samples the starting position is located in. The second neural network further pinpoints this down to 80 samples, the third down to 10 samples, with the last attempting to pinpoint the exact sample.

The input data generated for the neural networks were done following the method described in Chapter 4.2, with each neural network having its training data generated independently of the other neural networks. The parameters for these neural networks were set as follows:

- The modulation schemes were set to either ASK (OOK), FSK, or PSK. However, only one scheme was utilized for any given set of training and testing data.
- The sampled region for each neural network were set as described above.
- The RVM signal was set as a binary sequence of length 1-80 bits. The content of this binary sequence was fully randomized for the FSK and PSK modulation schemes, while for the ASK (OOK) modulation scheme the first bit is always set as a logic '1' due to the network being unable to see leading zeroes for this modulation.
- The starting position was randomized within the region covered by each output neuron.
- The amplitude was chosen to a constant of 50 throughout all tests. As mentioned earlier, the exact amplitude value holds little significance, with the important factor being the ratio between the amplitude and the noise levels.
- The $S_{ref}NR$ was varied between -6 dB and 6 dB, compared to a signal power of 25.
- The ratio between the smapled frequency and the frequency of the signal was kept at a rate of 4-20 samples for each bit worth of data.
- The number of periods for each bit was kept at a constant value of 1.
- The points where the samples were taken along the sinusoidal wave were randomized within the region covered by a single sample to represent a lack of synchronization.

4.6 RVM length classification

Length of a RVM signal may be a relevant evaluation point to determine whether or not all parts of a sent message are present. As the size of data and the accuracy needed are highly dependent on the actual application, it has here been chosen to look for the differences in signal length for small signals of a length of 0-25 bits in a noisy environment. The input data generated for the neural networks were done following the method described in Chapter 4.2, utilizing the following parameters:

- The modulation schemes were set to either ASK (OOK), FSK, or PSK. However, only one modulation scheme was utilized during a single set of training and test data.
- The sampled region was set to cover 50 bits worth of data, with the actual number of neurons in the input layer depending on the number of samples used to describe one bit.
- The RVM signal was set as a binary sequence of length 0-25 bits. The actual sequence was fully randomized for the FSK and PSK modulation schemes. For the ASK (OOK) modulation scheme, the first and last bit of the signal were always set as a logical '1', with the bits between being randomized.
- The starting position was randomized within the sampled region, with the valid positions being any position where the entire signal would fit within the sampled region.
- The amplitude was chosen to a constant of 50 throughout all tests, for the same reasons as mentioned for the other test-cases.
- The $S_{ref}NR$ was varied between -6dB and 6dB, with the noise levels being compared to a signal with a power of 25.

- The ratio between the sampled frequency and the frequency of the signal was kept at a rate of 4-20, meaning each bit worth of data covered 4-20 neurons in the input layer.
- The points of where the samples were taken along the sinusoidal wave were randomized within the region covered by each sample/neuron.

Chapter 5

Analysis

5.1 Comparison between the SGDM, RMSProp and ADAM solvers

Test-case	Modulation Scheme	SGDM accuracy	RMSProp accuracy	ADAM accuracy
Structural	ASK	0.8057	0.8115	0.7889
Structural	FSK	0.9265	0.9300	0.9165
Structural	PSK	0.9384	0.9350	0.9185
Position (10 byte NN)	ASK	0.9129	0.9087	0.9102
Position (10 byte NN)	FSK	0.9492	0.9388	0.9490
Position (10 byte NN)	PSK	0.9469	0.9372	0.9450
Length	ASK	0.3067 / 0.7130	0.3281 / 0.7368	0.3979 / 0.7980
Length	FSK	0.6575 / 0.9775	0.6975 / 0.9792	0.7643 / 0.9898
Length	PSK	0.6273 / 0.9726	0.7399 / 0.9848	0.7168 / 0.9779

Table 5.1: Comparison between the SGDM, RMSProp and ADAM solvers

The results when testing the performance of the different solvers can be seen in Table 5.1, with the accuracy shown for the length test scenario being written as $P_{Correct}/P_{CorrectOrOneOff}$. All tests within the same test scenario were done using the same neural network structure. As denoted by the table, the differences in accuracy for both the structural and position test-cases, who were both done at an already high accuracy, were rather small. As the actual accuracy can vary slightly even when using the exact same parameters, training, and testing data for each iteration, the differences here are rather negligible. However, for the length test-case, which in this scenario was done at a lower overal accuracy, the differences between the SGDM solver and the more sophisticated RMSProp and ADAM solvers were more noticable. One of the highest being a difference in accuracy of over 10% for the FSK modulation scheme when using ADAM as opposed to SGDM. The ADAM solver also attained quite significantly better results than the RMSProp for the ASK and FSK modulation schemes for the length test-case, only performing slightly worse for the PSK modulation scheme.

Due to the RMSProp and ADAM solvers being rather close for the remaining instances shown in Figure 5.1, the remaining comparisons shown throughout this chapter all utilize the ADAM solver. More information regarding the neural networks and parameters utilized for each comparison can be seen in Appendix A.1, B.1, and C.1.

5.2 Comparison between time-domain and FFT with and without phase-component

Test-case	Modulation Scheme	Time	Frequency	Frequency and phase
Structural	PSK	0.9185	0.4960	0.6276
Position (10B NN)	PSK	0.9450	0.9134	0.9140
Length	PSK	0.7168 / 0.9779	0.2725 / 0.6592	0.2731 / 0.6601

Table 5.2: Comparison of time-domain input parameters versus frequency domain with and without phase component

The results from the comparison between utilizing time-domain input data and frequency-domain input data (with or without phase component) for the PSK modulation scheme can be seen in Table 5.2. The neural networks utilized remained the same for the time-domain version and the frequency-domain version that did not include the phase. For the frequency domain version that included both frequency and phase component, a single change was made to the neural network by introducing another layer along the 3rd dimension. More clearly, the network was altered from having (X,Y,1,N) input neurons to (X,Y,2,N) input neurons, essentially doubling the number of neurons in the input layer. More details regarding each the time-domain and frequency-domain comparison can be found in Appendix A.2, B.2, and C.2.

As can be seen from Table 5.2, the frequency domain versions yielded significantly poorer accuracy for both the structural and length test-cases, while only having a slight difference in the case of determining position. Due to the poor performance, frequency domain input appear to be unsuited for distinguishing structural differences in a signal, as well as for the signal length. On the other hand, the due to the positional results being just above 3% apart for the three versions of the input, there may exist neural network structures that utilize frequency domain input that can outperform the time-domain version. However for the sake of this thesis, only the time-domain was further looked into.

The time required to perform the FFTs remained small when compared to the execution time of the neural network. As would be expected, the difference scales with the complexity of the neural network structure, due to the FFT only being dependent on the number of neurons for the input layer. The execution time for performing the classification and for the FFT can be seen in Table 5.3.

Test-case	FFT time (per iteration)	Classification time (per iteration)
Structural	$0.38 * 10^{-4}s$	$0.96 * 10^{-4}s$
Position (10 byte NN)	$0.24 * 10^{-4}s$	$0.94 * 10^{-4}s$
Length	$0.19 * 10^{-4}s$	$3.71 * 10^{-4}s$

Table 5.3: Execution time of FFTs compared to neural network classification time

5.3 Comparison between the use of the ASK (OOK), FSK, and PSK modulation schemes

Test-case	Mod. scheme	$6S_{ref}NR$	$0S_{ref}NR$	$-6S_{ref}NR$
Structural	ASK	0.8820	0.7889	0.4706
Structural	FSK	0.9791	0.9165	0.6953
Structural	PSK	0.9653	0.9185	0.7317
Position	ASK	0.9897	0.9102	0.6580
Position	FSK	0.9886	0.9490	0.8127
Position	PSK	0.9871	0.9450	0.7981

Table 5.4: Comparison between ASK, FSK, and PSK modulation schemes for structural and positional classification

Test-case	Mod. scheme	$6S_{ref}NR$	$3.5S_{ref}NR$	$0S_{ref}NR$
Length(Accurate)	ASK	0.7073	0.5436	0.3979
Length(Accurate or 1 off)	ASK	0.9377	0.9092	0.7980
Length(Accurate)	FSK	0.9943	0.9632	0.7643
Length(Accurate or 1 off)	FSK	1.0	0.9992	0.9898
Length(Accurate)	PSK	0.9923	0.9881	0.7168
Length(Accurate or 1 off)	PSK	0.9999	1.0	0.9779

Table 5.5: Comparison between ASK, FSK, and PSK modulation schemes for length classification

The modulation schemes examined were the basic versions of ASK (OOK), FSK, and PSK, with a focus on how they behave at different levels of noise. As can be seen in Table 5.4 and Table 5.5, the accuracy for FSK and PSK remain relatively close to each other for all test-cases and noise levels. The ASK on the other hand performed considerably worse for all the cases. ASK also appeared to have a steeper drop in accuracy with increasing noise levels. As an example, the accuracy for ASK dropped more than 33.17% in accuracy between a $S_{ref}NR$ of 6 and -6, while the FSK and PSK modulation schemes dropped 17.59% and 18.90%. One possible explanation for this could be due to on-off keying version describing logical '0' as an absence of signal. This essentially means the neural network received no visible difference between a logical '0' and pure noise. For a randomly generated binary stream, this would equate to the neural network only having half the amount of actual signal data to work with, which could explain why it performs worse than the FSK and PSK modulation schemes.

An additional note to make from Table 5.4 and Table 5.5 would be that classifying the signal's length appear more subject to error when the noise levels increase than the other test-cases. However, if one looks at the percentage of classifications for the length analysis that are either correct or just 1 off at $0S_{ref}NR$, one can see that the values are comparable to the accuracy of the structural test-case. The likely cause could therefore be that since the distinction between the output neurons for the length analysis are rather small, a spike in the noise levels near the start- or end-point of the signal may cause a wrong classification. Another possible reason could be that the length analysis simply has more output neurons than the other cases.

5.4 Comparison between CNN and FNN

Test-case	CNN accuracy	CNN exe. time	FNN accuracy	FNN exe. time
Structural	0.9185	1.19ms	0.6440	10ms
Position	0.9450	0.83ms	0.7547	3.28ms
Length	0.7168 / 0.9779	3.94ms	0.2163 / 0.5585	62.18ms

Table 5.6: Comparison of accuracy and execution time for CNN and FNN

Layer	# Neurons	# Weights	# Biases	# Multiplications	# Additions
Struct conv1	6090	3000	30	618 000	618 000
Struct conv2	1680	30 000	20	2 520 000	2 520 000
Struct conv3	380	4000	10	152 000	152 000
Sum	8150	37 000	60	3 290 000	3 290 000
Struct full1	6180	3 090 000	6180	3 090 000	3 090 000
Struct full2	1680	10 382 400	1680	10 382 400	10 382 400
Struct full3	380	638 400	380	638 400	638 400
Sum	8240	11 026 980	8240	14 102 560	14 110 800
Pos10 conv1	3240	400	10	129 600	129 600
Pos10 conv2	1280	4000	10	256 000	256 000
Pos10 conv3	210	1000	5	42 000	42 000
Sum	4730	4610	25	427 600	427 600
Pos10 full1	3240	2 592 000	3240	2 592 000	2 592 000
Pos10 full2	1280	4 147 200	1280	4 147 200	4 147 200
Pos10 full3	210	268 800	210	268 800	268 800
Sum	4730	7 008 000	4730	7 008 000	7 008 000
Length conv1	8240	4000	40	824 000	824 000
Length conv2	2520	60 000	30	5 040 000	5 040 000
Length conv3	700	12 000	20	420 000	420 000
Length conv4	160	2000	10	64 000	64 000
Sum	11 620	78 000	100	6 348 000	6 348 000
Length full1	8240	4 120 000	8240	4 120 000	4 120 000
Length full2	2520	20 764 800	2520	20 764 800	20 764 800
Length full3	700	1 764 000	700	1 764 000	1 764 000
Length full4	160	112 000	160	112 000	112 000
Sum	11 620	26 760 800	11 620	26 760 800	26 760 800

Table 5.7: Size and operation comparison between CNNs and FNNs

One aspect of the testing procedure was to compare the results given from a neural network primarily made out of convolutional layers, and a neural network made out of primarily fully connected layers. The convolutional and fully connected layers in the CNN and FNN are made to have roughly matching number of neurons their corresponding layers. As shown in Table 5.6, the CNN vastly outperforms the FNN both in terms of execution time and accuracy, to the point where the FNN networks are likely not even close to accurate enough to be viable to take part in a verification process.

Furthermore, as shown in Table 5.7, the FNN demands 4.3 (structural), 16.4 (position 10 byte version), and 4.2 (length) times as many floating point operations in the form either multiplications or additions. The higher disparity, however, is in the amount of storage space required by the FNN compared to the CNN, the highest of which in this case is the FNN for length analysis with 20.76 million weights in total. Attempts using smaller sized FNNs also failed to challenge the accuracy yielded by the CNN versions.

Please note that all layers shown in Table 5.7 are followed by a batch normalization layer and a rectified linear unit (ReLU) layer. These are not included in the calculations for storage space or floating point operations required in this case, due to them being roughly equal due to the similar sized convolutional and fully connected layers. An interesting detail would be that, in terms of accuracy. both the CNNs and FNNs performed significantly worse without the batch normalization layers. For some FNN structures, the difference was to the point where the training process would break down entirely without the batch normalization layers, resulting in the network predicting the same output regardless the input values.

The number of floating point operations for the CNNs should also be slightly lower in an actual system than shown in Table 5.7, as the numbers include floating point operations on padded areas where the input values are 0.

5.5 Neural network for structural analysis of RVM signal

5.5.1 Comparison of sampling rates

Samples per bit	Accuracy	Classification time
4	0.8146	$0.92 * 10^-4$s
10	0.9185	$0.93 * 10^-4$s
20	0.9356	$1.08 * 10^-4$s

Table 5.8: Accuracy and classification time for different number of samples per bit

As can be seen in Table 5.8, there appears to be a strong correlation between the ratio of sampling frequency versus signal frequency (here referred to as "samples per bit"), and the accuracy the neural network is capable of yielding. An important note is that, since the sampled window here scales with the number of samples used to describe one bit, the input layer scales proportionally with the number of samples used to describe each bit. The neural networks used to yield the data given by Table 5.8 have had the filter sizes, padding and stride of their first convolutional layer scaled with the number of neurons in the input layer to make the networks as similar as possible. They are, however, not entirely equivalent, as not all parameters were allowed to be scaled to a strictly proportional value. The details regarding the networks used for each case can be found in Appendix A.5.

The networks in the following subsections will utilize the 20 samples per bit version, in an attempt to optimize the accuracy.

5.5.2 Comparison of differently sized CNNs

The CNN architectures for structural feature recognition appears, according to the results shown in Table 5.9, seems to peak around 4 layers when using a stride of [4 1] for the first convolutional layer, and [2 1] for the remaining ones. The fifth layer on the 5 convolutional layer version in this

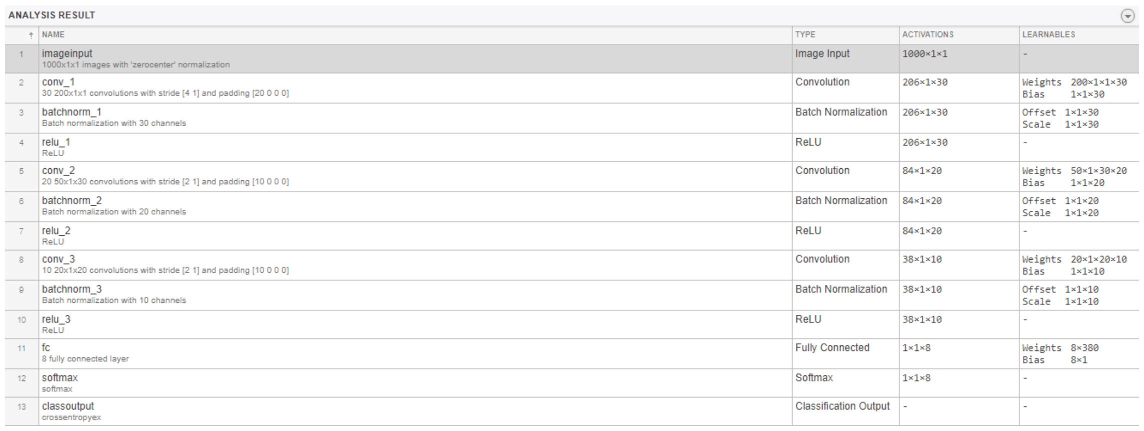
Layer	2CNN network	3CNN network	4CNN network	5CNN network
Conv1	30x[200 1 1] filters	30x[200 1 1] filters	30x[200 1 1] filters	30x[200 1 1] filters
Conv2	5x[20 1 30] filters	20x[50 1 30] filters	20x[50 1 20] filters	20x[50 1 20] filters
Conv3	-	10x[20 1 20] filters	20x[20 1 20] filters	20x[20 1 20] filters
Conv4	-	-	10x[20 1 20] filters	10x[20 1 20] filters
Conv5	-	-	-	5x[20 1 10] filters
Accuracy	0.9130	0.9356	0.9516	0.8601
Classification time	$1.11 * 10^{-4}s$	$1.08 * 10^{-4}s$	$1.12 * 10^{-4}s$	$1.84 * 10^{-4}s$

Table 5.9: Details for CNNs with 2-5 convolutional layers

case does appear to be too small to be beneficial to the network, only having the dimensions [3 1 5]. Reducing the stride for the layers may yield better accuracy, at the cost of increased computational and spacial needs.

Outside of the number of actual number of convolutional layers, the layers' stride, padding, filter size, and the number of filters used were also altered. The alterations that will be mentioned here will all be alterations of a single parameter from the CNN described in Figure 5.1, which is the same CNN as the 3CNN network from Figure 5.9. The details of these experiments can be found in Appendix A.6. Talking primarily about the first convolutional layer, since the remaining layers scales off of it's parameters, the accuracy does seem to have a peak somewhere around the [200 1 1] value, yielding an accuracy of 0.8131 with a filter size of [40 1 1], 0.8896 for [100 1 1], and 0.8431 for [500 1 1]. The padding also appear to yield the best results around the values shown in Figure 5.1, with an accuracy of 0.9213 without any padding. An increase in padding to almost match the filter size yielded an accuracy of 0.9386, which is within the normal deviation that can appear even when retraining a neural network with the exact same parameters.

Reducing the stride does increase accuracy, yielding 0.9519 when the stride of the first convolutional layer is reduced to [1 1]. However, this did more than double the time necessary to classify each iteration to 2.23ms. Increasing the number of filters at each convolutional layer also increases the accuracy, with a doubling of filters for all convolutional layers yielding an accuracy of 0.9559. A combination of low stride and more convolutional layers and7or filters may allow one to approach an accuracy of 1.0. However, the classification time will likely increase substantially as well. The accuracy drops to 0.8915 when the $S_{ref}NR$ is lowered to -6dB. This decrease is primarily caused by a further drop i accuracy for detecting the end-point of the signal, as well as an added confusion between instances of very short data packets ('D') and instances with no data ('0').

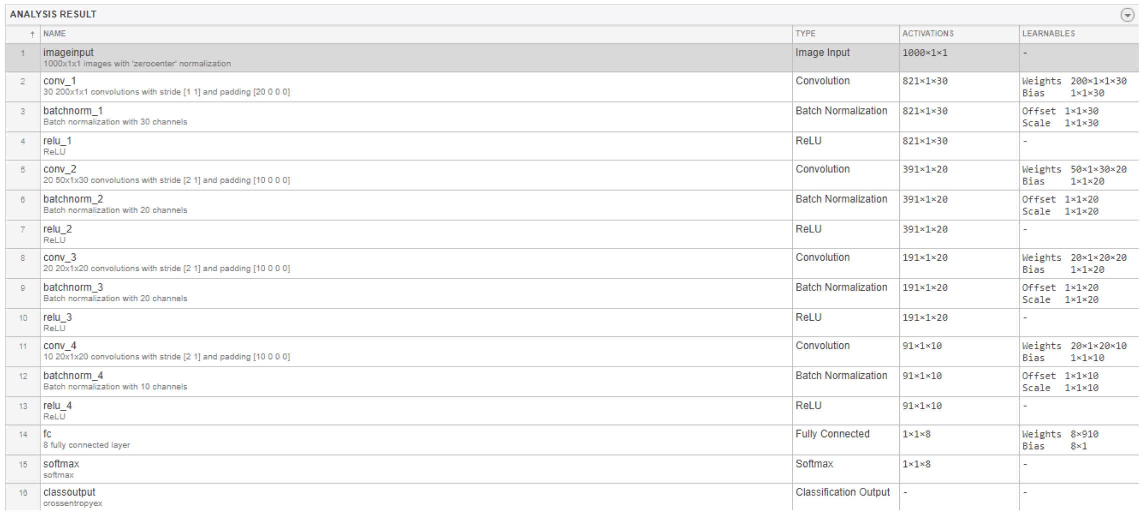


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Figure 5.1: CNN used as a basis for parameter alterations

5.5.3 CNN performance for structural classification



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [1 1] and padding [20 0 0 0]	Convolution	821×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	821×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	821×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	391×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	391×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	391×1×20	-
8	conv_3 20 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	191×1×20	Weights 20×1×20×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	191×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	191×1×20	-
11	conv_4 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	91×1×10	Weights 20×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	91×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	91×1×10	-
14	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×910 Bias 8×1
15	softmax softmax	Softmax	1×1×8	-
16	classoutput crossentropyex	Classification Output	-	-

Figure 5.2: High accuracy CNN for structural classification

Figure 5.2 shows structure for the CNN with the highest accuracy for structural classification, without going to overboard on the number of filters at each convolutional layer. As shown in Table 5.11, this neural network achieves an accuracy of 0.9623, requiring $45 * 10^{-4}$ seconds for each classification on average. As can be seen in Table 5.10, where the letters P, S, D, and E represent the presence of preamble, SoF delimiter, data, and EoF, the vast majority of the false classifications regards not detecting the EoF. Since the neural network seems capable of detecting the difference between a correct and incorrect SoF, which are differentiated by a single binary value, these wrong classifications regarding the EoF are likely caused when the EoF is placed extremely close to the end of the sampled region. The results of a second, smaller, neural network is also shown in Table 5.11. This neural network is identical to the one shown in Figure 5.2, except the number of filters have been reduced to [10 10 5 5] for the 4 convolutional layers. As can be seen by comparing the two, one can maintain an accuracy only 0.01 below the larger network while only having to use roughly 0.22 times the weights and biases, and 0.21 times the floating point operations. The numbers in the parenthesis in Table 5.11 are the numbers needed to connect the last convolutional layer to the output layers, with the other numbers just including the convolutional layers themselves.

Actual \Predicted	0	P	D	D E	P D	P D E	P S D	P S D E
0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
P	0.0	0.969	0.0	0.0	0.018	0.0	0.013	0.0
D	0.023	0.001	0.959	0.002	0.010	0.005	0.0	0.0
D E	0.008	0.004	0.057	0.919	0.0	0.007	0.0	0.005
P D	0.0	0.005	0.0	0.0	0.994	0.001	0.0	0.0
P D E	0.0	0.009	0.0	0.0	0.057	0.933	0.0	0.001
P S D	0.0	0.002	0.0	0.0	0.001	0.0	0.993	0.004
P S D E	0.0	0.002	0.0	0.0	0.0	0.001	0.066	0.931

Table 5.10: Accuracy for each output neuron for structural classification for neural networn in Figure 5.2

Type	Accuracy	Classification time	CNN weights+biases	CNN add+mult ops
Optimized for accuracy	0.9623	$45 * 10^{-4}s$	48080 (+7288)	37 096 000 (+14560)
Smaller version	0.9529	$9.92 * 10^{-4}$	8530 (+3648)	7 667 000 (+7280)

Table 5.11: Accuracy and classification time for structural classification at $0S_{ref}NR$ and 20 samples per bit

5.6 Neural network for positional analysis of RVM signal

The results shown in this chapter utilize the CNN responsible for narrowing down the position from a 10 byte region to a 1 byte region when making comparisons, with the results for the other parts being added in Chapter 5.6.3.

5.6.1 Comparison of sampling rates

Table 5.12 shows the comparison of accuracy and classification time for different number of samples used to describe one bit of data. As the table shows, there appears to be a strong correlation between accuracy and number of samples used to describe each bit. However, the accuracy remains above the 90-percentile for all three cases shown. The neural networks used to yield these results are made to be as similar as possible, having the filter sizes, stride, and padding scaled as proportionally as possible to make the networks as similar as possible. This is reflected in Table 5.12 with the networks having similar classification times, with the major difference being in the number of neurons in the input layer.

The details regarding the networks used for each case can be found in Appendix B.5. The networks in the following subsections will utilize the 20 samples per bit version, in an attempt to optimize the accuracy.

Samples per bit	Accuracy	Classification time
4	0.9214	$0.79 * 10^{-}4$s
10	0.9450	$0.83 * 10^{-}4$s
20	0.9708	$0.95 * 10^{-}4$s

Table 5.12: Accuracy and classification time for different number of samples per bit for positional classification at $0S_{ref}NR$

Layer	2CNN network	3CNN network	4CNN network
Conv1	10x[40 2 1] filters	10x[40 2 1] filters	10x[40 2 1] filters
Conv2	10x[10 2 10] filters	10x[10 2 10] filters	10x[10 2 10] filters
Conv3	-	5x[10 2 10] filters	10x[5 2 10] filters
Conv4	-	-	5x[5 2 10] filters
Accuracy	0.9787	0.9739	0.9847
Classification time	$0.83 * 10^{-4}s$	$0.90 * 10^{-4}s$	$0.97 * 10^{-4}s$

Table 5.13: Details for CNNs with CNNs with 2-4 convolutional layers at 0SNR

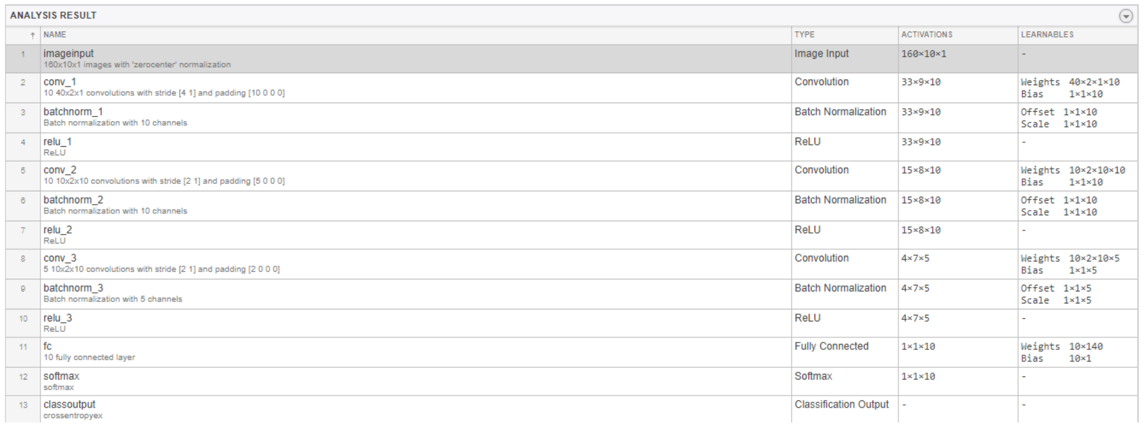
5.6.2 Comparison of differently sized CNNs

As can be seen in Table 5.13, the CNN architectures for positional recognition appear to stay extremely close to each other in terms of accuracy. All of these CNNs utilize a stride of [4 1] for their first convolutional layer, and a stride of [2 1] for the remaining convolutional layers. This does seem to point towards the neural network having little difficulty in classifying which position the start of the signal appears in, allowing for a high accuracy without a high complexity.

Utilizing the neural network described in Figure 5.3, which is the same network as the *3CNN network* in Table 5.13, the network appears to achieve a higher accuracy both when reducing and increasing the filter size for the first convolutional layer. A reduction down to [20 2 1] gives an accuracy of 0.9799, while increasing the filter size to [80 2 1] gives an accuracy of 0.9823. The highest accuracy, however, is achieved when reducing the stride along the 2nd dimension to 1, giving a filter size of [40 1 1]. Utilizing this filter, the network was capable of achieving an accuracy of 0.9912. Reducing the stride of the first convolutional layer, and adding additional padding to the convolutional layers also increased the network's accuracy. Still, these alterations did not manage to reach as high an accuracy as changing the filter sizes did. Further details regarding each of the test-cases described can be found in Appendix B.6.

5.6.3 CNN performance for positional classification

As mentioned earlier, the positional classification follows a hierarchical approach, where a sampled region capable of holding up to 100 byte worth of data is first reduced from 100 byte region to 10 byte region, from a 10 byte region to a 1 byte region, 1 byte region to 1 bit region, and then finally down to a single sample. The examples shown up until now have all been from the neural network responsible for reducing the area in question from a 10 byte region to a 1 byte region. The other CNNs have been modelled based on the performance this CNN had through the previous test-cases. The details, such as filter sizes and layer structure, regarding the structure of each CNN can be found in Appendix B.7. The results for each of the CNNs can be seen in Table 5.14, with the measurements being from the PSK modulation scheme at $0S_{ref}NR$, and 20 samples per bit. The number of weights, biases



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	4×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	4×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	4×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×140 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Figure 5.3: CNN used as a basis for parameter alterations for positional classification

and floating point operations listed are the ones utilized for the convolutional layers, with the ones needed for connecting to the output layer being placed in parenthesis.

The outermost CNN, which in Table 5.14 has been called CNN($100B \Rightarrow 10B$), showed a perfect accuracy for the test-set in question. While the size of CNN($100B \Rightarrow 10B$) may be reduced, the version here was included due to keeping an accuracy of 0.9994 when the $S_{ref}NR$ was reduced to -6. However, one should keep in mind that since the position is randomized within the valid range, and with that range in this case being rather large, the number of cases close to the borders between the different output neurons would statistically be rather few. In addition, this version of the neural network has a staggering 16000 input neurons in total.

As shown in Table 5.14, the accuracy remains extremely high down to classifying the correct 1 bit region of the input samples, maintaining an accuracy of 0.983 to the correct region. However, there is a rather huge drop in accuracy to find the correct sample within that 1 bit region. As was mentioned in Chapter 4.2, the noise peaks for $0S_{ref}NR$ can reach up to 55 in value. With a signal amplitude set to 50, it should be understandable why the neural network struggles to reach high accuracy at such a high precision. A spike in noise at the correct sample could either completely nullify the signal at that sample, or heighten another sample to make it look like the start position. A note to make is that CNN($1b \Rightarrow 1sample$) has a 0.88 chance of either predicting the correct sample or be 1 sample off, which makes the system as a whole have a 0.865 chance of being at most 1 sample off. Another note to make for CNN($1b \Rightarrow 1sample$), which utilizes a single hidden layer trio, is that it was the only case where a fully connected version got close in accuracy to its CNN counterpart. In this case, a FNN managed to obtain an accuracy of 0.70. For the case of an $S_{ref}NR$ of -6, the accuracy of CNN($10B \Rightarrow 1B$) drops to 0.9285 and CNN($1b \Rightarrow 1sample$) to 0.8084, resulting in a 0.75 chance of correct classification within a 1 bit region. CNN($1b \Rightarrow 1sample$) drops to an accuracy of 0.3342, being mostly unusable at this point.

Neural Network	Acc.($0S_{ref}NR$)	Class. time	#CNN weight+bias	CNN FLOPS
CNN($100B \Rightarrow 10B$)	1.0	$12.0 * 10^{-4}$s	1925(+48 010)	5 174 400(+96 000)
CNN($10B \Rightarrow 1B$)	0.9912	$0.90 * 10^{-4}$s	1925 (+3010)	624 000 (+6000)
CNN($1B \Rightarrow 10b$)	0.9918	$0.96 * 10^{-4}$s	8460 (+5768)	1 000 800 (+11 520)
CNN($1b \Rightarrow 1sample$)	0.7226	$0.13 * 10^{-4}$	510 (+17 020)	17 000 (+34 000)
Combined	0.7104	$13.99 * 10^{-4}$	12 820 (+73 808)	6 816 200 (+147 520)

Table 5.14: Performance and details for CNNs for positional classification.

Samples per bit	Accuracy	Accurate or 1 off	Classification time
4	0.3451	0.7955	$4.08 * 10^{-}4$s
10	0.7168	0.9779	$3.94 * 10^{-}4$s
20	0.9009	0.9985	$4.76 * 10^{-}4$s

Table 5.15: Accuracy and classification time for different number of samples per bit for length classification

5.7 Neural network for signal length analysis of RVM signal

5.7.1 Comparison of sampling rates

As should be apparent from Table 5.15, the number of samples used to describe one bit worth of data has a significant impact on the accuracy of the neural network. This does makes sens, considering the amount of samples directly correlate to how much of an actual difference there are between the output neurons. Much more for the structural or positional cases, the classification of a signal's length randomly positioned in a noisy environment appears highly dependent on the sampling rate of the receiver. As with structural and positional classifications, the CNNs utilized to yield the results in Table 5.15 had the parameters of their first convolutional layer scaled, in order to appear as similar as possible.

The networks in the following subsections will utilize the 20 samples per bit version, as this one appears to be the only one that may reach an acceptable accuracy.

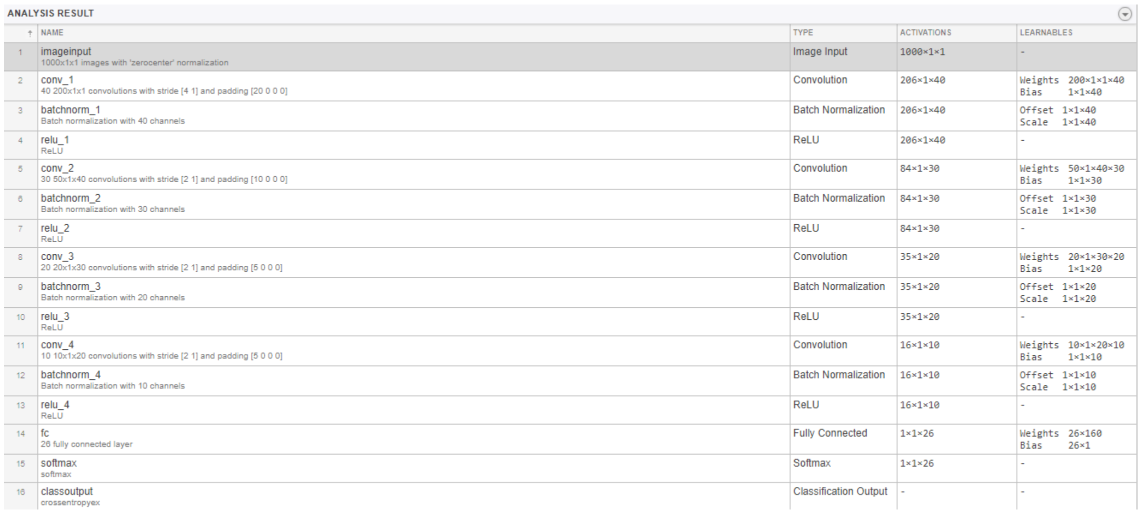
5.7.2 Comparison of differently sized CNNs for length classification

Layer	3CNN network	4CNN network	5CNN network
Conv1	30x[200 1 1] filters	40x[200 1 1] filters	50x[200 1 1] filters
Conv2	20x[50 1 30] filters	30x[50 1 40] filters	40x[50 1 50] filters
Conv3	10x[20 1 20] filters	20x[20 1 30] filters	30x[20 1 40] filters
Conv4	-	10x[10 1 20] filters	20x[10 1 30] filters
Conv5	-	-	10x[5 1 20] filters
Accuracy	0.6892	0.9009	0.9045
Accurate or 1 off	0.9727	0.9985	0.9887
Classification time	$2.74 * 10^{-4}s$	$3.94 * 10^{-4}s$	$7.57 * 10^{-4}s$

Table 5.16: Details for CNNs with 3-5 convolutional layers for length classification

Table 5.16 shows a comparison between CNNs with different amounts of convolutional layers. While the version utilizing 3 convolutional layers lacks behind, the 4 convolutional layers and 5 convolutional layers versions are extremely close when it comes to obtaining the correct classification. With a difference of only 0.0036, this remains within the area a neural network may deviate in accuracy after training with the exact same input parameters. The details regarding these CNNs can be found in Appendix C.5 and C.6.

using the 4CNN network from Table 5.16 as a basis, which can be seen in Figure 5.4, the neural network's stride, padding, and filters were altered in an attempt to achieve better accuracy. In opposition to the results given when attempting to optimize for structural or positional classification, reducing the stride of the first convolutional layer to the minimum amount ([1 1]) resulted in the accuracy dropping to 0.5751, or 0.8755 for either being accurate or 1 sample off. Furthermore, both increasing and decreasing the filter size of the first convolutional layer caused a loss in accuracy. For a filter size reduced to [40 1], the accuracy dropped to 0.7241, and to 0.8027 for a filter increased to [400 1] in size. Most results points to there being a middle-ground in complexity where the length classification network has it's peak in accuracy. Details regarding the neural networks used for each of the test-cases can be found in Appendix C.6.


ANALYSIS RESULT

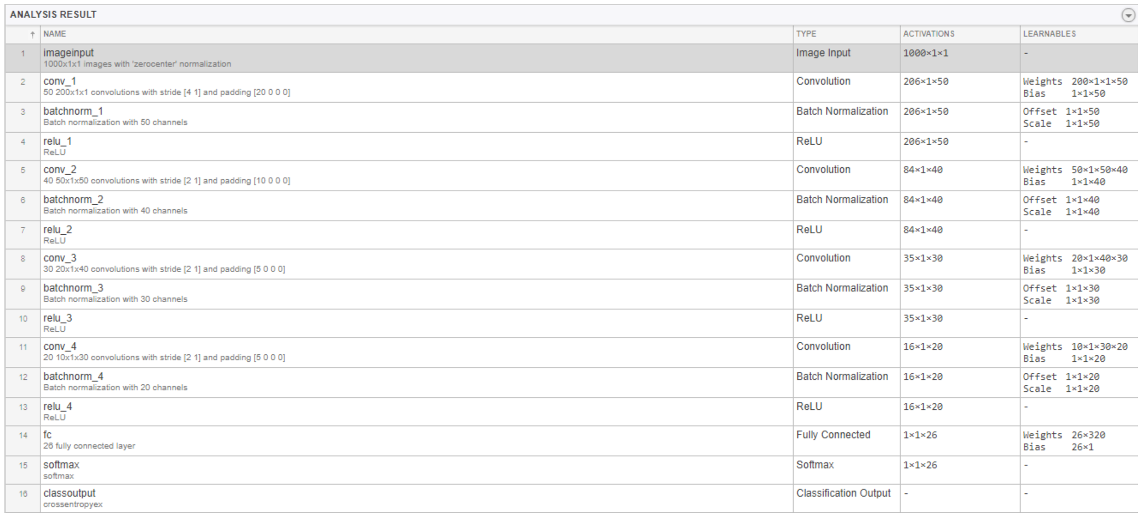
↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 40 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×40	Weights 200×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Figure 5.4: CNN used as basis for parameter alterations for length classification

5.7.3 CNN performance for length classification

Figure 5.5 shows the CNN structure that yielded the highest accuracy. As shown in Table 5.17, the accuracy this CNN achieved was 0.9359 for correct classification, and 0.9970 for being at most one bit off. This CNN is, however, extremely large to the point of being comparable to the FNNs discussed in Chapter 5.4. Out of the 26 output neurons, the only one which differentiated from the rest in form of accuracy is the '0' length neuron. This neuron had an accuracy of 0.998, having only a single false classification out of the 500 instances that went through the classification procedure for this output.

The accuracy of the CNN drops significantly when the $S_{ref}NR$ increases. With a $S_{ref}NR$ of -6, the accuracy drops to 0.3432, with the chance of it being either accurate of 1 bit off being 0.7868



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	206×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	206×1×50	-
5	conv_2 40 50x1x50 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×40	Weights 50×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	84×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	84×1×40	-
8	conv_3 30 20x1x40 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×30	Weights 20×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	35×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	35×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	16×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	16×1×20	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×320 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Figure 5.5: CNN optimized for accuracy for lengths classification

Accuracy	Accurate or 1 off	Classification time	CNN weights+biases	CNN add+mult ops
0.9359	0.9970	$4.49 * 10^{-4}s$	134140 (+8346)	22 792 000 (+16640)

Table 5.17: Accuracy and classification time for length classification at 0SNR and 20 samples per bit

Chapter 6

Discussion

This chapter will discuss the methodology chosen, as well as the results presented in the previous chapters.

6.1 Validity of generated data

As was mentioned during Chapter 4.2, the generation for the RVM training and testing data was done using manually using MatLab R2018a. This was done due to no existing databases with relevant data being found. One aspect that should be kept in mind with this, is that since since the data was made using programmed logic, there may be a risk of the ANNs managing to re-create that logic, which would be an unintended effect. Also, because the data has been generated and not sampled, some parameters have perfect replication, while in an actual sampled situation they would not. An example of this would be that in the generated data, the ratio between the sampling frequency and the signal frequency is a perfect integer with no deviation. Each bit is covered buy an exact number of samples, and each instance have the exact same signal frequency. In a real application, the sampled values may have small deviations in these aspects.

The case where the ratio between sampling frequency and signal frequency is not a perfect integer value could have an effect on the overall accuracy. While the sampling points on the sinusoidal wave have been skewed to represent a non-synchronized design, the sampling points remain the at the same points for each bit-period of that instance. This essentially means that each binary value '1' and '0' holds the same pattern, when not considering the added noise. For a non-integer ratio, the points on the sinusoidal wave the samples are taken at would change depending on where the bit is located in the sampled region, possibly causing an added complication for the classification process. This would naturally have a higher impact when the ratio between sampling and signal frequency is low, as for higher values, the difference would likely just act as a small addition of noise.

6.2 Accuracy and size of ANNs

The results shown throughout Chapter 5 usually points towards larger neural networks being more accurate, up to a certain point. As should be apparent from Table 5.6 in Chapter 5.4, the fully connected neural networks are far outmatched by the convolutional neural networks both in terms of speed and accuracy. As further noticed in Chapter 5.5.3, 5.6.3, and 5.7.3, the CNNs also appear to perform best when the filter sizes of medium size. The likely cause could be that, while too small may fail to gather sufficient parameters for a correct classification, a larger neural network may end up with too many parameters that has no importance for the classification process, but instead simply adds noise throughout the layers. As previously mentioned, the only case where a fully connected network had a comparable accuracy was for the smallest neural network in the positional classification. This neural network had a single convolutional or fully connected layer, followed by a batch normalization layer and ReLU layer. Adding additional layers in this case only caused the accuracy to decrease.

The amount of filters used in the convolutional layers probably have the largest impact when it comes to the comparison of both physical (storage) and computational size versus the accuracy achieved. While an increased number of filters does have a positive impact on the accuracy, they do have a massive impact on both the computational requirements, and the amount of storage space required. This increase in accuracy does seem to level out eventually. However, as exemplified in Table 5.11, one can gain a massive difference in terms of size and computational requirements at only a small loss in accuracy. As long as a slight loss in accuracy isn't a critical problem, it should be considered to utilize a smaller CNN, especially in the case of limited storage, such as on a FPGA. Of the other parameters, filter size padding appeared to have a window where they work best, meaning changing them either to become smaller or larger than this area would have a negative impact on the CNNs accuracy. With exception of the length classification, reducing the stride generally saw a minor increase in accuracy, at the cost often having a significantly larger computational and spatial footprint.

The amount of storage space required is also highly dependent on the amount of neurons in the last layer before the output layer. Since the connection to the output is fully connected, it required a total of $n * m$ weights and m biases, with m being the number of neurons in the output layer, and n the number of neurons in the layer connected to it. As is best seen in Table 5.14, this number may be far larger than that of the convolutional layers combined. An increase in the number of convolutional layers may therefore actually cause a net gain in both size and computational requirements. A comparison between the 5 convolutional layer networks shown in Table 5.9 and 5.16, the accuracy appears to be more linked to the number of neurons connected to the output layer than the actual number of convolutional layers.

A common trend that goes throughout the entire testing procedure would be that the accuracy is highly linked with how many input neurons are used to differentiate between two output neurons. In other words, the robustness is directly linked with how many samples the noise must distort before invalidating that section of the input. This should come as no surprise, as it essentially equates to increasing how distinct each output neuron appears from the others.

Network sizes influenced the training time for the networks. While the training times have not been included as part of this thesis, a general understanding in the difference in training time can be seen by looking at the classification times for each ANN. A thing to keep in mind would be that all of these classification times are taken using CPUs, meaning devices with more parallelism such as GPUs or FPGAs should require a much lower time for both training and classification.

6.3 Time-domain versus frequency-domain

One of the early decisions made during for this thesis was to look at the comparison between how time-domain input data and frequency-domain input data. While these two domains essentially contains the same data, the CNNs proved much better at recognizing differences in the time-domain for these cases than for the frequency domain. As the cases looked at could be described as primarily an evaluation of signal amplitude, this shouldn't be that much of a surprise. One would, however, still expect the frequency domain version to outperform the time-domain for cases more closely related to the signal's frequency, as those should have a greater distinction in the frequency domain.

Other parameters, such as the higher-order parameters utilized to classify modulation scheme in cognitive radio, as described in Chapter 3.1, may also be of use as input parameters, either as an addition or as a replacement to having the time-data inputs. An advantage that could be made if higher-order variables prove accurate enough would be that the neural network structures they utilize could potentially be of a significant smaller size than the ones the time-data CNNs utilize.

6.4 Results for the classification problems

The choice of solver algorithms used to minimize the loss/error function does appear to have quite the impact on how well the ANN is able to be trained. As shown in Table 5.1, this difference appears to be largest when the accuracy is low, with the ADAM solver pulling ahead of both RMSProp and SGDM for these cases.

The modulation scheme also seems to have an impact on the classification process. However, the difference appears to be more linked to the amount of data visible to the neural network, and not as much as to the actual modulation scheme. As an elaboration for this, the FSK and PSK has remained extremely close in terms of accuracy throughout all tests, commonly within 1% of each other, usually taking turns on who's most accurate. Even when using the same neural network and the same set of training data, the resulting accuracy tends to deviate up to around 0.3%. This is most probably caused by a combination of the randomized initial weights and biases in combination with the random order the training data. The difference between FSK and PSK is therefore basically negligible. ASK, on the other hand, had consistently worse results than the other two. Since it utilizes on-off-keying, the binary value '0' is described as an absence of signal. This results in the neural network essentially having half the amount of visible data to work with. The amount of visible data the modulation scheme gives the neural network therefore appears to be far more significant than how the visible data is structured.

As previously mentioned, the accuracy appears to correlate mostly with how many samples that differentiate the output neurons. As an be seen in Table 5.10, the false classifications are primarily a failure to detect the end-point of the data package. These cases are most probably the cases where the end point is located only a few samples from the end of the sampled region, making a few, or even a single, spike in the added noise enough to cause a wrong classification. This would also mean the accuracy is dependent on the size of the sampled region for this classification. Since the signal's position is randomized, a larger sampled region would make it less likely for a signal to be placed close to the edges. An added note would also be that the most accurate solution for the structural may prove too large to be implemented on an FPGA or a similar device. As the structural classification is trained to look for specific patterns, it must also be trained specifically for a set of structures to look for. CNNs does however appear to be capable of distinguishing between single binary differences, as the cases where their only difference was the binary value of the 1-bit SoF delimiter had an accuracy of 0.994 and 0.993 respectively for 0SNR.

Positional classification appears highly accurate down to the 1bit region, which for the case shown equated to a region of 20 samples. These neural networks, even combined, are significantly smaller than the ones used for both structural and length analysis. The largest CNN in the hierarchy could also probably be reduced in size without much loss in accuracy, due to how well it holds up even at a SNR of -6. However, one should keep in mind that much of the reason why the larger networks have such a high accuracy is due to how unlikely the randomized position in the training and test set are to be placed near the edges of each region. This CNN hierarchy may be the one most likely to be useful for more general cases, with the primary challenge for implementing it on a FPGA being funneling data to the 16000 input neurons. The network does by no means have to start out covering such a large region, as the size was mostly chosen to verify if the network would be able to pinpoint the position from such a large set of data.

Out of the three, the length classification is the worst performer both in terms of accuracy, and in terms of size and computational requirements. Considering the high accuracy of the positional classification hierarchy, it will probably be more beneficial to just run the region through this hierarchy, reversing the order of the neurons to find the end-point.

6.5 Possible improvements

While the CNNs have gone through an optimization process, this does not mean they cannot be improved further in terms of accuracy, size, or computational requirements. Out of the three, positional architecture is probable the only one that could have any use for a general case.

One optimization could be to look for the amount of convolutional filters that would yield the smallest amount of storage requirement while maintaining a near peak accuracy. Attempting to make the cases more generalized would also be a possible optimization. Also, as mention in Chapter 3.3, finding structures that work for binary neural networks would be a possible optimization option, as it could reduce both the number storage registers needed, as well as being able to utilize counters instead of arithmetic logic units (ALUs)

6.6 Additional features

This thesis has mostly focused on looking for structural characteristics contained in a RVM signal. More analogue values such as amplitude, frequency, rise times, and so on could also be of interest. The main challenge with more analogue values would be that, since a neural network's output has a binary form, these characteristics would most likely be classified within ranges instead of exact values.

Chapter 7

Conclusion

This thesis aimed to develop and test a series of artificial neural networks for classification of features in a radio signal described through real value modeling. The thesis chose to focus on three primary characteristics: the search for structural characteristic, the search for delay, by finding the starting position of a signal within a sampled region, and by finding the length of a RVM signal. The data generation and test environment were done using MatLab R2018a. All data was subjected to Gaussian white noise with a SNR compared to a reference signal power of 25. This resulted in noise peaks roughly equal to the signal amplitude at $0S_{ref}NR$.

Of the three different solvers tested, the ADAM solver proved to yield the highest accuracy when the resulting accuracy could be considered low, especially sub 50%. SGDM, RMSProp, and ADAM all having roughly equal performance for high accuracy systems, especially above 90%. Of the three modulation schemes utilized, the FSK and PSK performed roughly equal, while the ASK (OOK) modulation lacked behind for all tests completed. This seems to be more a result of FSK and PSK providing the neural network with more data, as ASK (OOK) does not provide data for binary '0' values, as opposed to the actual form the data takes. CNN architectures outperformed FNN architectures by 20-50% in accuracy, while maintaining a far smaller physical and computational footprint. Time-domain input data yielded higher accuracy results than frequency-domain input data. The closest comparison here was the positional classification, yielding 0.9450 in accuracy for time-domain input data, and 0.9134 for frequency domain input data.

The structural characteristics achieved an accuracy of 0.9623 at $0S_{ref}NR$ with a classification time of $34.0 * 10^{-4}$s on average for each classification. The sum of weights and biases for this network was 55360, requiring roughly 37.1 million floating point operations for its convolutional layers, plus the connections to the output layer. A smaller version which achieved an accuracy of 0.9529, with a classification time of $9.92 * 10^{-4}$s, 12178 weights and biases, and roughly 7.7 million floating point operations for its convolutional layers and the output layer. The network had an accuracy of 0.993 at recognizing a single-bit identifier, with the primary source of false classifications being detecting the signal end-point when placed close to the edge of the sampled region.

The positional characteristics was developed in a hierarchical fashion, where the first CNN took in a sampled region capable of holding 100 bytes worth of data, and determining in which 10 byte region the start of the signal was located in. The second CNN would further pinpoint the region down to a 1 byte region, the 3rd down to a 1 bit region, and the forth attempt to find the exact sample. The CNN hierarchy achieved an accuracy of 0.983 to classify the right bit region at $0S_{ref}NR$, and

0.7104 for classifying the correct sample. The total classification time for the entire hierarchy was $13.99 * 10^{-4}$s, not counting the time needed to move data from one network to the other. The sum of weights and biases for the entire network was 86628, whereas 73808 were between the last convolutional layer and the output layers in the CNNs. The total number of floating point operations for the convolutional layers and output layers were roughly 7.0 million. The hierarchy achieved a 0.75 accuracy for classifying the correct bit region at -60$S_{ref}NR$, with the last CNN being unable to reliably classify the correct sample at this $0S_{ref}NR$ value.

The length characteristics achieved an accuracy of 0.9359, with a 0.9970 chance of being either accurate or 1 bit off. The classification time for the network was $4.49 * 10^{-4}$s per classification. The sum of weights and biases for the convolutional layers plus the output layer 142486, with the number of floating point operations performed by these layers being roughly 22.8 million. The accuracy dropped to 0.3432 when the $S_{ref}NR$ was decreased to -6, with a 0.7868 chance of being either accurate or 1 bit off. Running through the positional CNN hierarchy twice to obtain the start and end position should generally outperform using the CNN for length characteristics.

7.1 Future work

A natural way forward would be to attempt to characterize other radio signal utilizing real value modeling and artificial neural networks. Examples of such features could be signal power, frequency, and rise/fall times. Further exploration into possible higher-order parameters to use as inputs for the ANNs could also be a viable next step.

Another path forward would be to implement the CNNs into an FPGA, analyzing the classification times and comparing them to that of SPICE or AMS. One may also attempt to include the CNN architectures into an actual verification procedure and measure their performances. One may also wish to generalize the structures to create a wider area of use.

A separate route, as mentioned during Chapter 6, would be to utilize fixed data types with a reduced precision. Binary neural networks are a valid candidate to explore, but so is reducing the data types down to 8 bit fixed point.

Bibliography

[1] Pete Hardee Sathishkumar Balasubramanian. Solutions for mixed-signal soc verification using real number models.

[2] Kathleen A. Meade Sharon Rosenberg. *A Practical Guide to Adopting the Universal Verification Methodology (UVM)*. Cadence Design Systems, Inc., 2010.

[3] Accellera Systems Initiative. *Universal Verification Methodology (UVM) 1.2 User's Guide*, October 2015.

[4] Accellera Systems Initiative. *Universal Verification Methodology (UVM) 1.2 Class Reference*, June 2014.

[5] Hao Fang Neyaz Khan, Yaron Kashai. Metric driven verification of mixed-signal designs.

[6] Martin Barnasconi. Systemc ams extensions: Solving the need for speed.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Tom M. Mitchell. *Machine Learning*. McGrac-Hill Science/Engineering/Math, March 1997.

[9] Michael Nielsen. *Neural Networks and Deep Learning*. Online Book, Dec 2017. `http://neuralnetworksanddeeplearning.com/index.html`.

[10] Giulio Gambardella Michaela Blott Philip Leong Magus Jahre Kees Vissers Yaman Umuroglu, Nicholas J. Fraser. Finn: A framework for fast, scalable binarized neural network interface. *FPGA '17 Proceedings of the 2017 ACM/SIGDA International Symposiom on Field-Programmable Gate Arrays*, pages 65–74, February 2017.

[11] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006. `http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20-%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf`.

[12] Jimmy Ba Diederik P Kingma. Adam: A method for stochastic optimization, Dec 2014. `arXiv:1412.6980v9`.

[13] Deep learning training from scratch: Options for training deep learning neural network. `https://se.mathworks.com/help/nnet/ref/trainingoptions.html`.

[14] Howard B. Demuth Mark Hudson Beale, Martin T. Hagan. Neural network toolbox user's guide, March 2018. `https://se.mathworks.com/help/pdf_doc/nnet/nnet_ug.pdf`.

[15] MathWorks Inc. Neural network toolbox. `https://se.mathworks.com/help/nnet/index.html`.

[16] Christian Szegedy Sergey Ioffe. Batch normalizaiton: Accelerating deep network training by reducing internal covariate shift, March 2015. `arXiv:1502.03167`.

[17] Daniel Grady Daniel Gebhardt Benjamin Migliori, Riley Zeller-Townson. Biologically inspired radio signal feature extraction with sparse denoising autoencoders, May 2016. `arXiv:1605.05239`.

[18] Jide Julius Popoola. Automathic recognition of both inter and intra classes of digital modulatated signals using artificial neural networks. *Journal of Engineering Science and Technology*, 9:273–285, 2014.

[19] E.E. Azzouz Asoke K. nandi. Algorithms for automatic modulation recognition of communication signals. *IEEE transactions on communications*, 46(10), April 1998. `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=664294`.

[20] Ole Martin Skafså. Fpga implementation of a convolutional neural network for "wake up word" detection. Master's thesis, June 2017.

[21] MathWorks Inc. Matlab r2018a. `https://se.mathworks.com/products/new_products/latest_features.html?s_tid=srchtitle`.

Appendix A

Artificial neural network results for structural test-cases

- 0: *No signal detected*
- P: *Only preamble detected*
- D: *Only data detected. End-point* ***not*** *present within sampled data*
- DE: *Only data detected. End-point found within sampled data*
- PD: *Preamble and data detected. Correct SoF delimiter not found. End-point* ***not*** *found within sampled data*
- PDE *Preamble and data detected. Correct SoF delimiter* ***not*** *found. End-point found within sampled data*
- PSD: *Preamble and data detected. Correct SoF delimiter detected. End-point* ***not*** *found within sampled data*
- PSDE: *Preamble and data detected. Correct SoF delimiter detected. End-point found within sampled data.*

A.1 Comparison between the different solvers for analog structural feature analysis

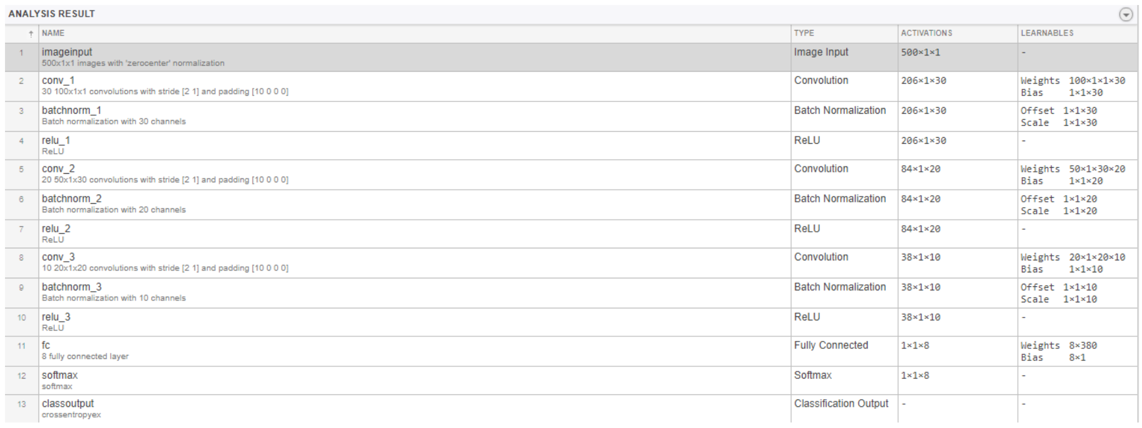
Signal information:

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	sgdm, rmsprop, adam
Epochs	100
Iterations per epoch	62
Learning rate	Constant 0.01 (sgdm), Constant 0.001 (rmsprop, adam)

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 30 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×30	Weights 100×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Analog signal structural feature analysis using SGDM:

ASK	Accuracy	0.8057
	Execution time	1.0465s (for 8000 iterations)
	Execution per iteration	$1.00 * 10^{-4}$s

FSK	Accuracy	0.9265
	Execution time	0.8373s (for 8000 iterations)
	Execution per iteration	$1.05 * 10^{-4}$s

PSK	Accuracy	0.9384
	Execution time	0.8085s (for 8000 iterations)
	Execution per iteration	$1.01 * 10^{-4}$s

Analog signal structural feature analysis using RMSProp:

ASK	Accuracy	0.8115
	Execution time	0.8565s (for 8000 iterations)
	Execution per iteration	$1.07 * 10^{-4}$s

FSK	Accuracy	0.9300
	Execution time	0.8159s (for 8000 iterations)
	Execution per iteration	$1.02 * 10^{-4}$s

PSK	Accuracy	0.9350
	Execution time	0.7630s (for 8000 iterations)
	Execution per iteration	$0.95 * 10^{-4}$s

Analog signal structural feature analysis using ADAM:

ASK	Accuracy	0.7889
	Execution time	1.2283s (for 8000 iterations)
	Execution per iteration	$1.54 * 10^{-4}$s

FSK	Accuracy	0.9165
	Execution time	0.8003s (for 8000 iterations)
	Execution per iteration	$1.00 * 10^{-4}$s

PSK	Accuracy	0.9185
	Execution time	0.9534s (for 8000 iterations)
	Execution per iteration	$1.19 * 10^{-4}$s

A.2 Structural comparison of time-data and FFT

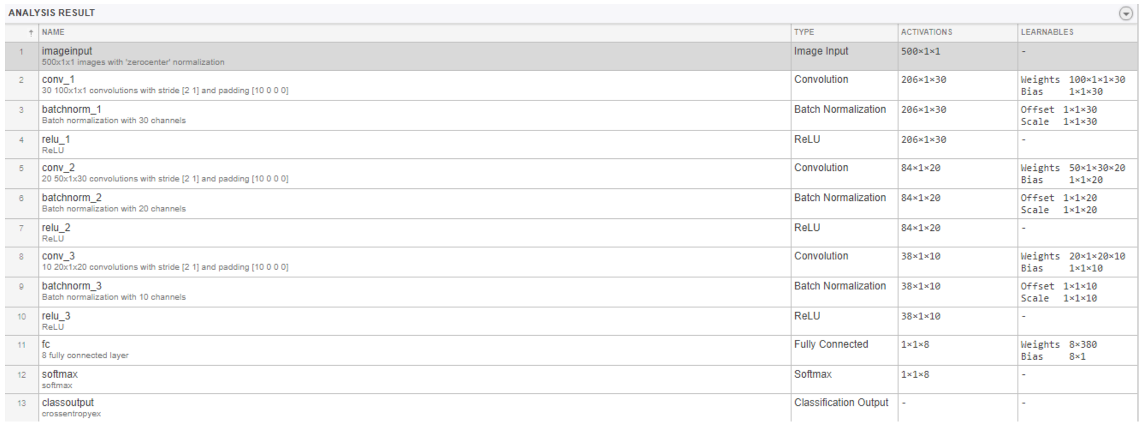
Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	62
Learning rate	Constant 0.001

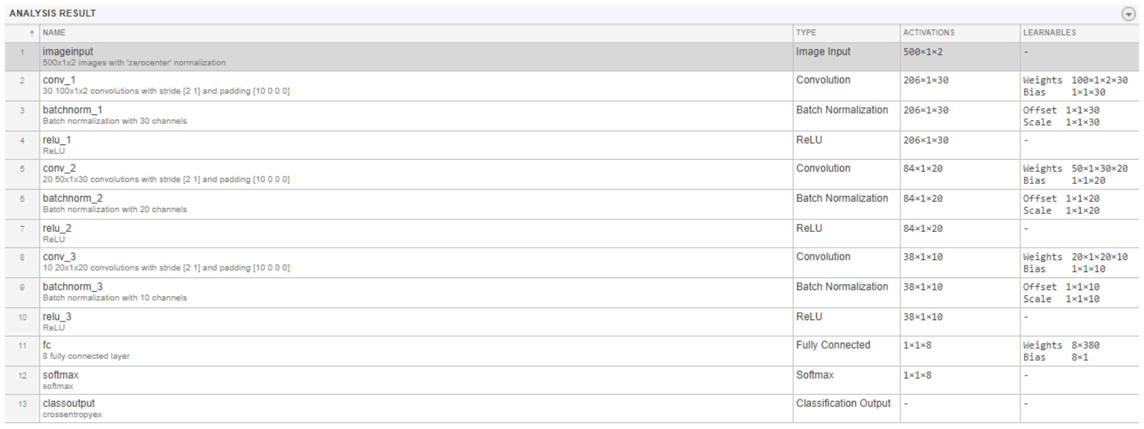
Network Structure for time data and FFT (amplitude only):



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 30 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×30	Weights 100×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Network Structure for FFT (amplitude and phase):



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x2 images with 'zerocenter' normalization	Image Input	500×1×2	-
2	conv_1 30 100x1x2 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×30	Weights 100×1×2×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

PSK time-domain classification results

Accuracy	0.9185
Execution time	0.9534s (for 8000 iterations)
Execution per iteration	$1.19 * 10^{-4}$s

PSK FFT classification results with amplitude and phase:

Accuracy	0.6276
FFT time	1.6516s
FFT time per iteration	$2.06 * 10^{-4}$s
Execution time	1.1462s (for 8000 iterations)
Execution per iteration	$1.43 * 10^{-4}$s

PSK FFT classification results with only amplitude:

Accuracy	0.4960
FFT time	0.3029s
FFT time per iteration	$0.38 * 10^{-4}$s
Execution time	0.7653s (for 8000 iterations)
Execution per iteration	$0.96 * 10^{-4}$s

A.3 ASK, FSK and PSK comparison at different SNR for structural analysis

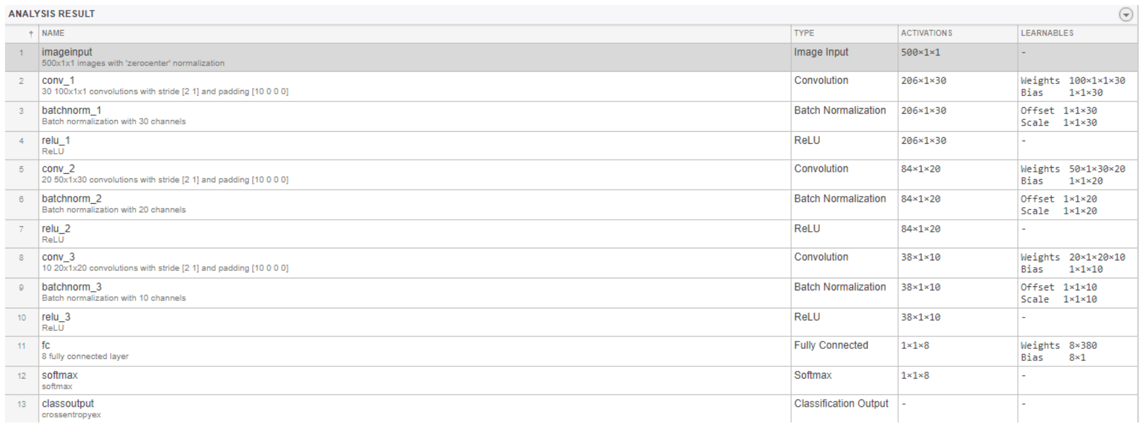
Signal information:

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	Varies
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	62
Learning rate	Constant 0.001

Network Structure:



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 30 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×30	Weights 100×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

6SNR

ASK	Accuracy	0.8820
	Execution time	0.8027s (for 8000 iterations)
	Execution per iteration	$1.00 * 10^{-4}$s

FSK	Accuracy	0.9781
	Execution time	0.8243s (for 8000 iterations)
	Execution per iteration	$1.03 * 10^{-4}$s

PSK	Accuracy	0.9653
	Execution time	0.8968s (for 8000 iterations)
	Execution per iteration	$1.12 * 10^{-4}$s

-6SNR

ASK	Accuracy	0.4706
	Execution time	0.7542s (for 8000 iterations)
	Execution per iteration	$0.94 * 10^{-4}$s

FSK	Accuracy	0.6953
	Execution time	0.7545s (for 8000 iterations)
	Execution per iteration	$0.94 * 10^{-4}$s

PSK	Accuracy	0.7318
	Execution time	0.8005s (for 8000 iterations)
	Execution per iteration	$1.00 * 10^{-4}$s

A.4 Comparison compared to fully connected neural networks

Neural numbers for each layer have been chosen to match the number or neurons each CNN network in Chapter A.2. Please refer to this, and and other chapter describing CNN results, for comparisons. The FNNs below all utilize the same set of training and test data.

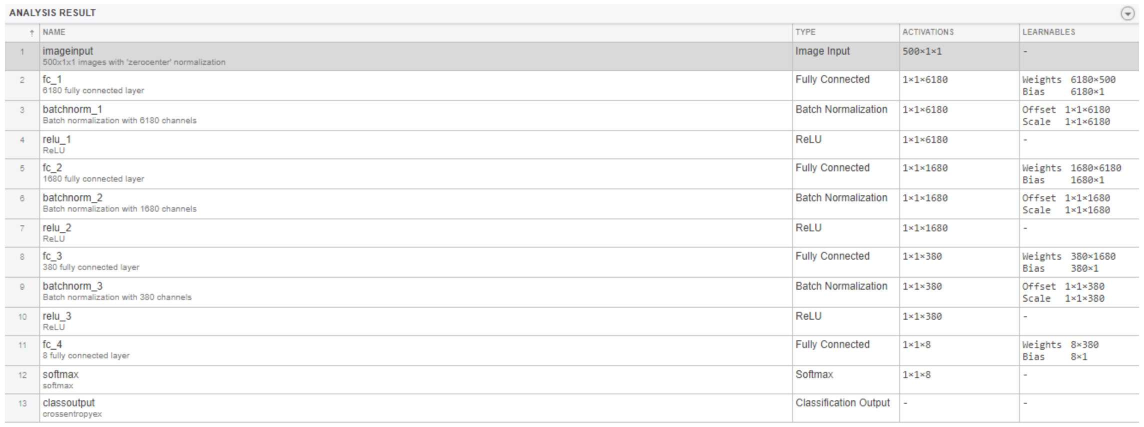
Signal information:

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	62
Learning	Constant 0.001

Network Structure: The network below is made to have equal number of nodes in each of the three fully connected layers as the standard CNN size used in Appendix A.3.



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	fc_1 6180 fully connected layer	Fully Connected	1×1×6180	Weights 6180×500 Bias 6180×1
3	batchnorm_1 Batch normalization with 6180 channels	Batch Normalization	1×1×6180	Offset 1×1×6180 Scale 1×1×6180
4	relu_1 ReLU	ReLU	1×1×6180	-
5	fc_2 1680 fully connected layer	Fully Connected	1×1×1680	Weights 1680×6180 Bias 1680×1
6	batchnorm_2 Batch normalization with 1680 channels	Batch Normalization	1×1×1680	Offset 1×1×1680 Scale 1×1×1680
7	relu_2 ReLU	ReLU	1×1×1680	-
8	fc_3 380 fully connected layer	Fully Connected	1×1×380	Weights 380×1680 Bias 380×1
9	batchnorm_3 Batch normalization with 380 channels	Batch Normalization	1×1×380	Offset 1×1×380 Scale 1×1×380
10	relu_3 ReLU	ReLU	1×1×380	-
11	fc_4 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Batch Normalization layers are include here, as otherwise the training reaches a certain point before the training accuracy breaks down, causing the network to give the same prediction regardless of input.

Classification results:

Accuracy	0.6440
Execution time	8.3247s (for 8000 iterations)
Execution per iteration	$10.0 * 10^{-4}$s

One aspect worth noting is that training this FNN required a total training time of 194 minutes, while the CNN equivalent can requires only around 10 minutes for the same case.

A.5 Structural analysis for different number of samples per bit

A comparison to 10 samples per bit can be found in Appendix A.1.

Signal information:

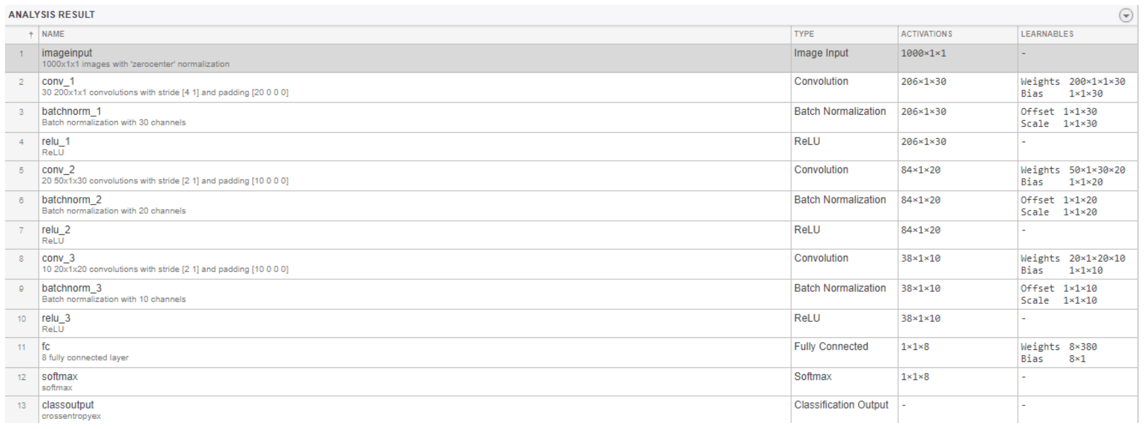
Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	varies
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	62
Learning rate	Constant 0.001

20 samples per bit

Network Structure:



ANALYSIS RESULT

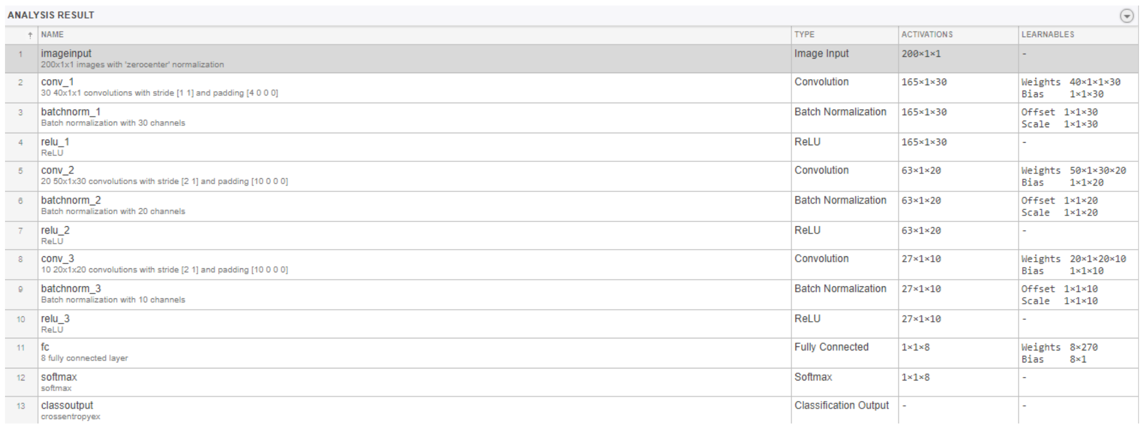
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9356
Execution time	0.8623s (for 10000 iterations)
Execution per iteration	$1.08 * 10^{-4}$s

4 samples per bit

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 200x1x1 images with 'zerocenter' normalization	Image Input	200×1×1	-
2	conv_1 30 40x1x1 convolutions with stride [1 1] and padding [4 0 0 0]	Convolution	165×1×30	Weights 40×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	165×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	165×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	63×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	63×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	63×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	27×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	27×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	27×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×270 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8146
Execution time	0.7340s (for 10000 iterations)
Execution per iteration	$0.92 * 10^{-4}$s

A.6 Structural comparison of differently sized CNNs

Signal information:

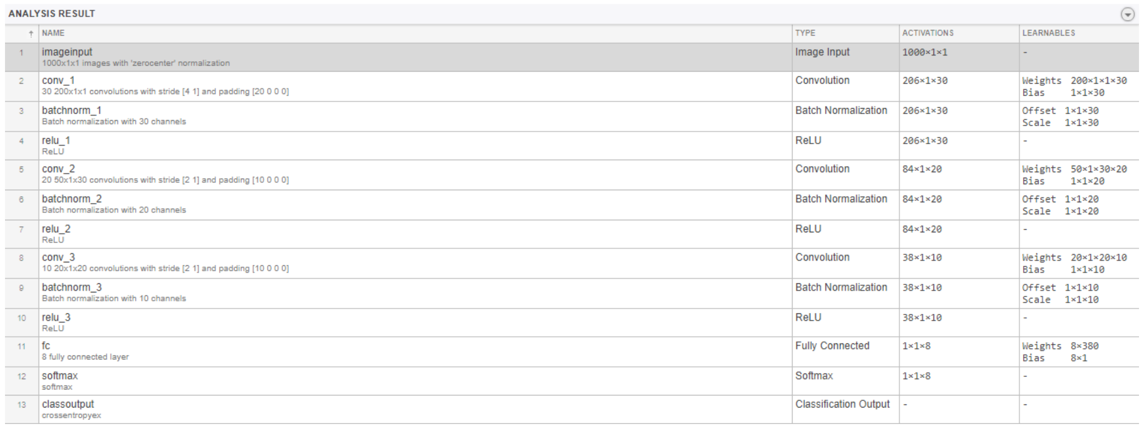
Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	20
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	62
Learning rate	Constant 0.001

Unmodified structure with results:

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	38×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	38×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×380 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

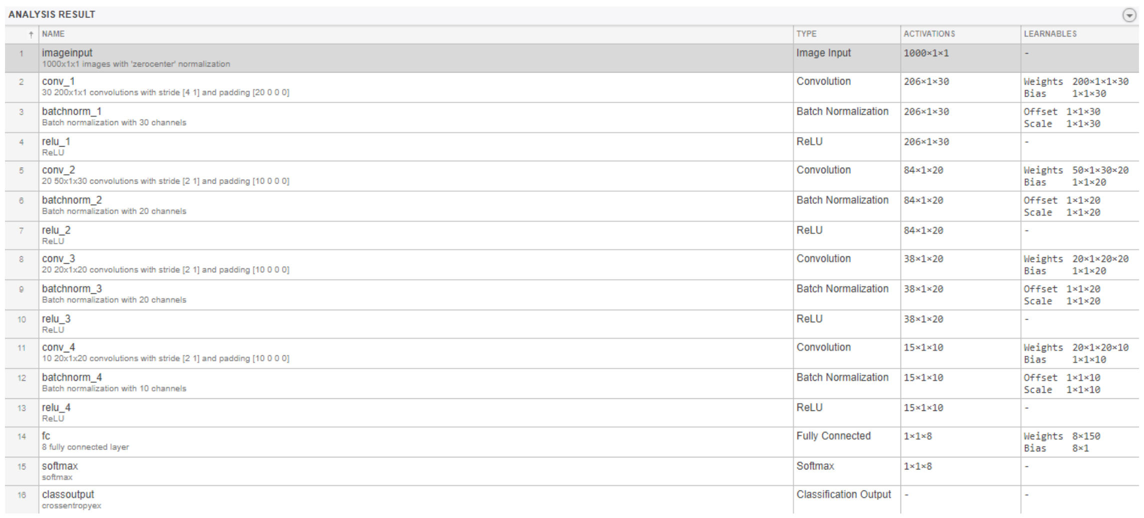
Results:

Accuracy	0.9356
Execution time	0.8623s (for 10000 iterations)
Execution per iteration	$1.08 * 10^{-4}$s

4 CNN layers

Network Structure:

Results:



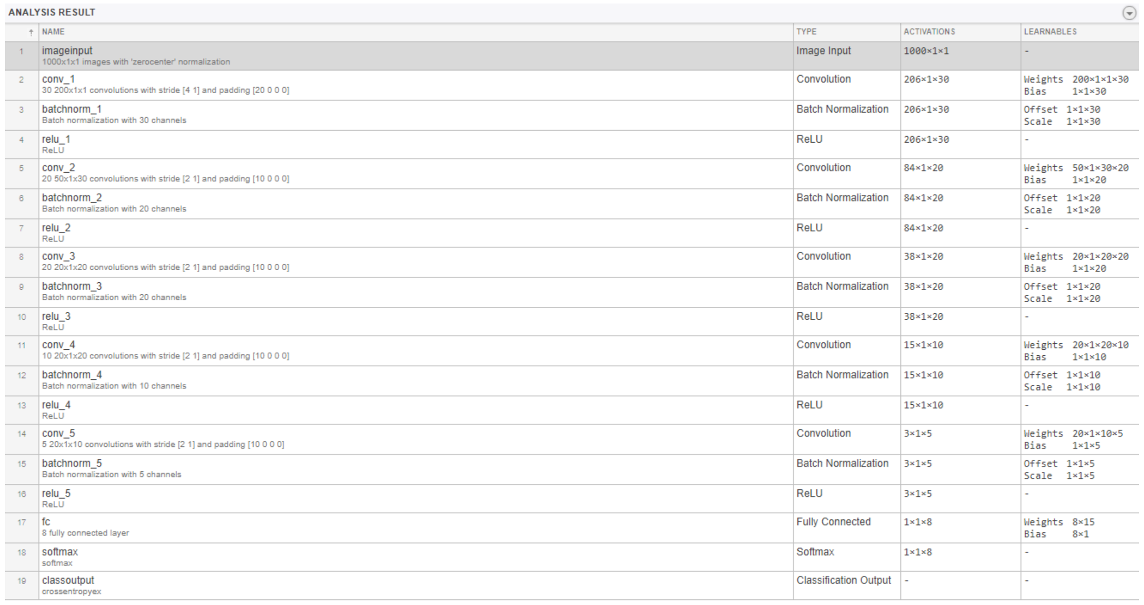
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 20 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×20	Weights 20×1×20×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	38×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	38×1×20	-
11	conv_4 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	15×1×10	Weights 20×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	15×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	15×1×10	-
14	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×150 Bias 8×1
15	softmax softmax	Softmax	1×1×8	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9516
Execution time	0.8994s (for 10000 iterations)
Execution per iteration	$1.12 * 10^{-4}$s

5 CNN layers

Network Structure:



ANALYSIS RESULT

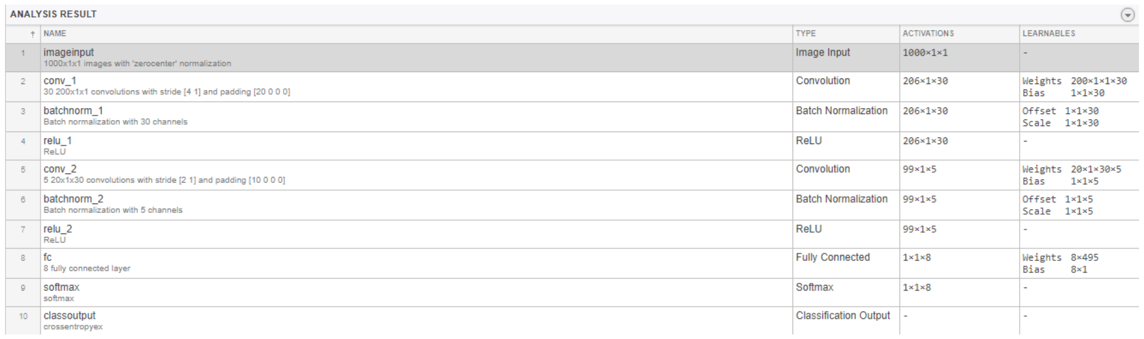
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 20 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×20	Weights 20×1×20×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	38×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	38×1×20	-
11	conv_4 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	15×1×10	Weights 20×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	15×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	15×1×10	-
14	conv_5 5 20x1x10 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	3×1×5	Weights 20×1×10×5 Bias 1×1×5
15	batchnorm_5 Batch normalization with 5 channels	Batch Normalization	3×1×5	Offset 1×1×5 Scale 1×1×5
16	relu_5 ReLU	ReLU	3×1×5	-
17	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×15 Bias 8×1
18	softmax softmax	Softmax	1×1×8	-
19	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8601
Execution time	1.4734s (for 10000 iterations)
Execution per iteration	$1.84 * 10^{-4}$s

2 CNN layers

Network Structure:



ANALYSIS RESULT

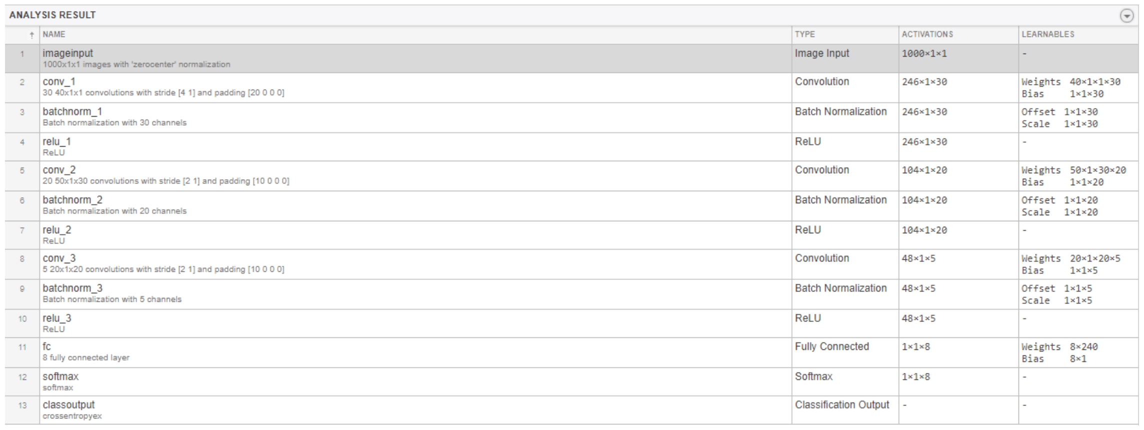
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 5 20x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	99×1×5	Weights 20×1×30×5 Bias 1×1×5
6	batchnorm_2 Batch normalization with 5 channels	Batch Normalization	99×1×5	Offset 1×1×5 Scale 1×1×5
7	relu_2 ReLU	ReLU	99×1×5	-
8	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×495 Bias 8×1
9	softmax softmax	Softmax	1×1×8	-
10	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9130
Execution time	0.8902s (for 10000 iterations)
Execution per iteration	$1.11 * 10^{-4}$s

Smaller sized convolutional filters

Network Structure:



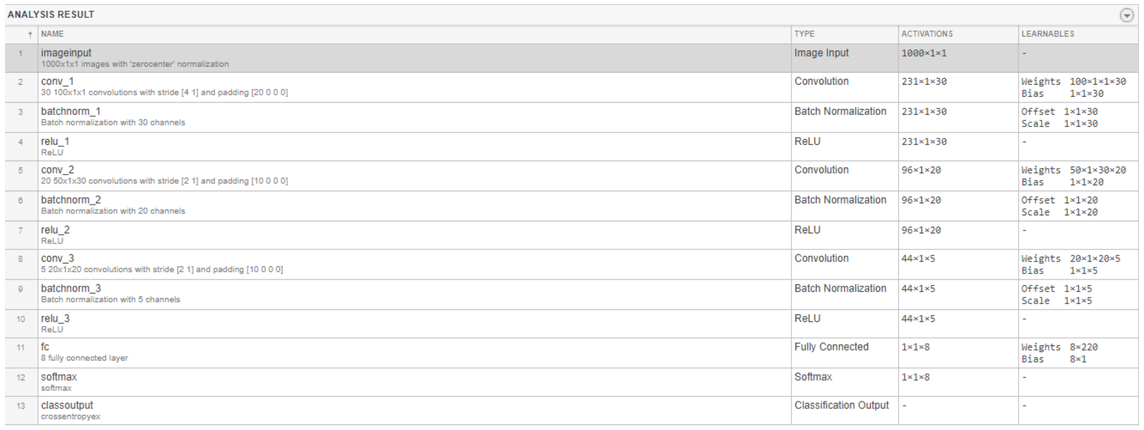
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 40x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	246×1×30	Weights 40×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	246×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	246×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	104×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	104×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	104×1×20	-
8	conv_3 5 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	48×1×5	Weights 20×1×20×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	48×1×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	48×1×5	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×240 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8131
Execution time	0.8660s (for 10000 iterations)
Execution per iteration	$1.08 * 10^{-4}$s

Network Structure:



ANALYSIS RESULT

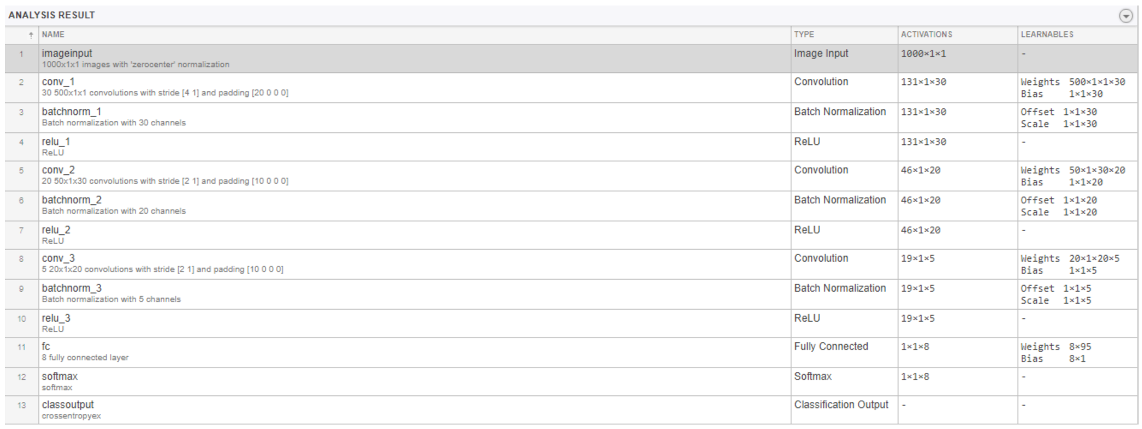
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 100x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	231×1×30	Weights 100×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	231×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	231×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	96×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	96×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	96×1×20	-
8	conv_3 5 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	44×1×5	Weights 20×1×20×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	44×1×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	44×1×5	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×220 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8896
Execution time	0.8362s (for 10000 iterations)
Execution per iteration	$1.05 * 10^{-4}$s

Larger sized convolutional filters

Network Structure:



ANALYSIS RESULT

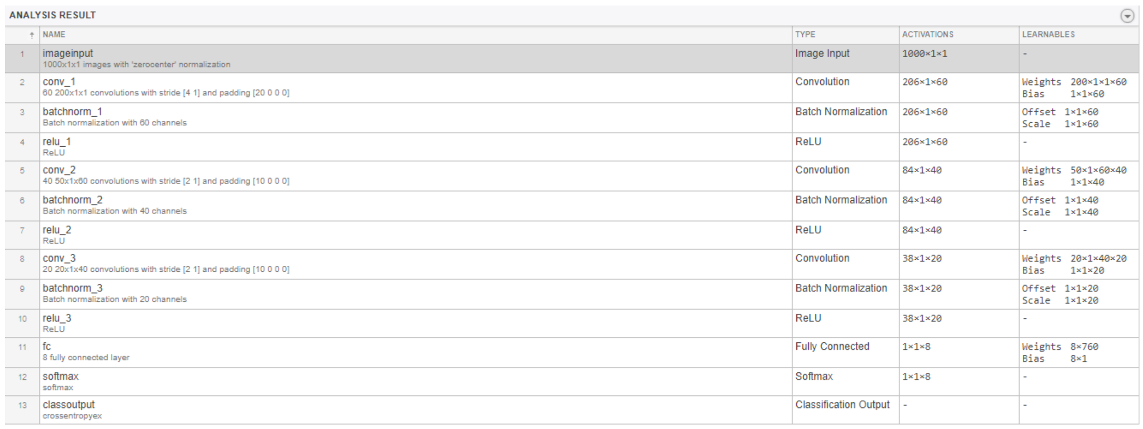
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 500x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	131×1×30	Weights 500×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	131×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	131×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	46×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	46×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	46×1×20	-
8	conv_3 5 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	19×1×5	Weights 20×1×20×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	19×1×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	19×1×5	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×95 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8431
Execution time	0.7471s (for 10000 iterations)
Execution per iteration	$0.934 * 10^{-4}$s

Doubled amount convolutional filters

Network Structure:



ANALYSIS RESULT

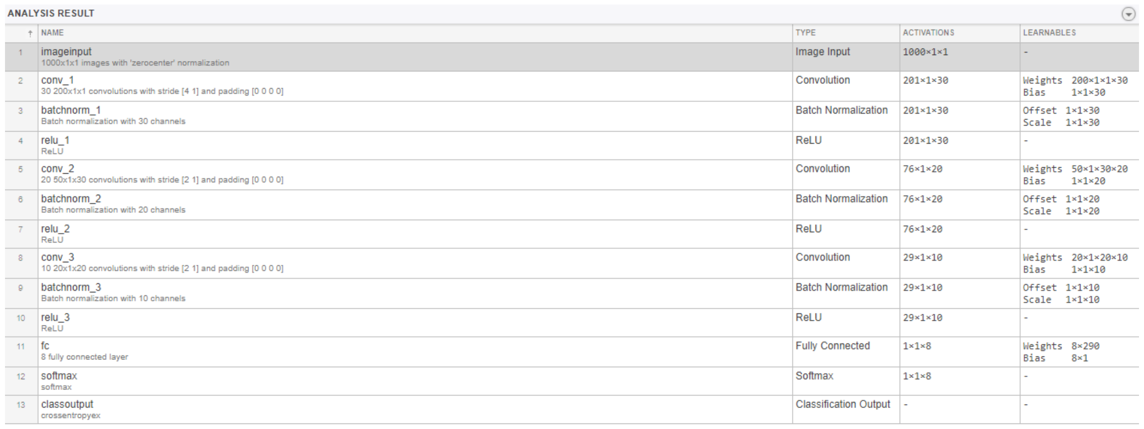
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 60 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×60	Weights 200×1×1×60 Bias 1×1×60
3	batchnorm_1 Batch normalization with 60 channels	Batch Normalization	206×1×60	Offset 1×1×60 Scale 1×1×60
4	relu_1 ReLU	ReLU	206×1×60	-
5	conv_2 40 50x1x60 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×40	Weights 50×1×60×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	84×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	84×1×40	-
8	conv_3 20 20x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	38×1×20	Weights 20×1×40×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	38×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	38×1×20	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×760 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9559
Execution time	1.3155s (for 10000 iterations)
Execution per iteration	$1.64 * 10^{-4}$s

No padding

Network Structure:



ANALYSIS RESULT

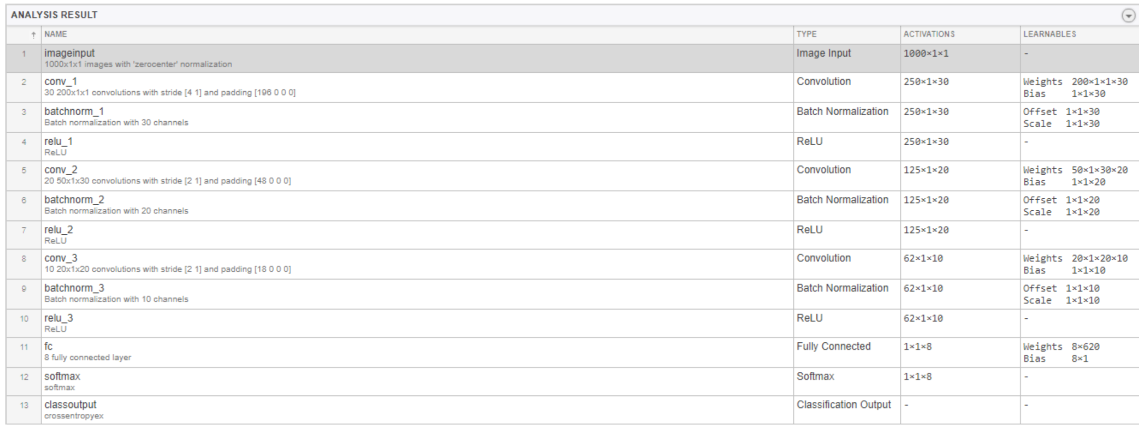
↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [0 0 0 0]	Convolution	201×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	201×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	201×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	76×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	76×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	76×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	29×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	29×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	29×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×290 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9213
Execution time	0.6473s (for 10000 iterations)
Execution per iteration	$0.81 * 10^{-4}$s

More padding

Network Structure:



ANALYSIS RESULT

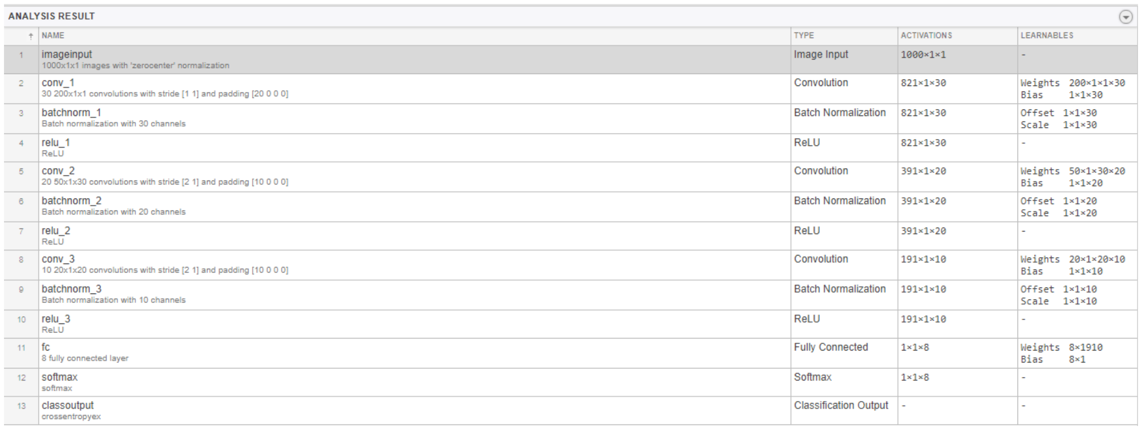
↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [196 0 0 0]	Convolution	250×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	250×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	250×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [48 0 0 0]	Convolution	125×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	125×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	125×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [18 0 0 0]	Convolution	62×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	62×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	62×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×620 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9386
Execution time	0.9785s (for 10000 iterations)
Execution per iteration	$1.22 * 10^{-4}$s

Less stride

Network Structure:



ANALYSIS RESULT

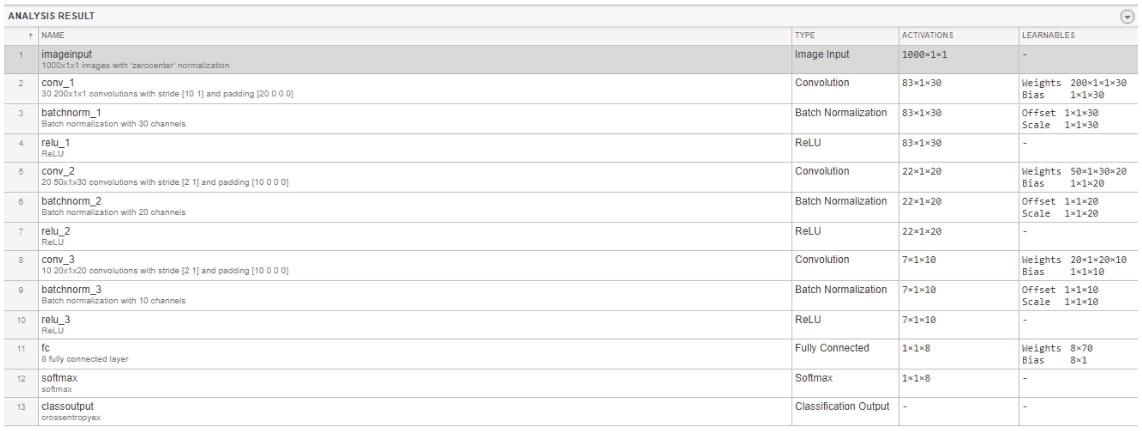
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [1 1] and padding [20 0 0 0]	Convolution	821×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	821×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	821×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	391×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	391×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	391×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	191×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	191×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	191×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×1910 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9519
Execution time	1.7823s (for 10000 iterations)
Execution per iteration	$2.23 * 10^{-4}$s

More stride

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [10 1] and padding [20 0 0 0]	Convolution	83×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	83×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	83×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	22×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	22×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	22×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	7×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	7×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	7×1×10	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×70 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.8769
Execution time	0.8616s (for 10000 iterations)
Execution per iteration	$1.08 * 10^{-4}$s

Appendix B

Artificial neural network results for positional test-cases

B.1 Comparison between the different solvers for analog positional analysis

Signal information:

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	80 bit area
Data bits	1-80
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

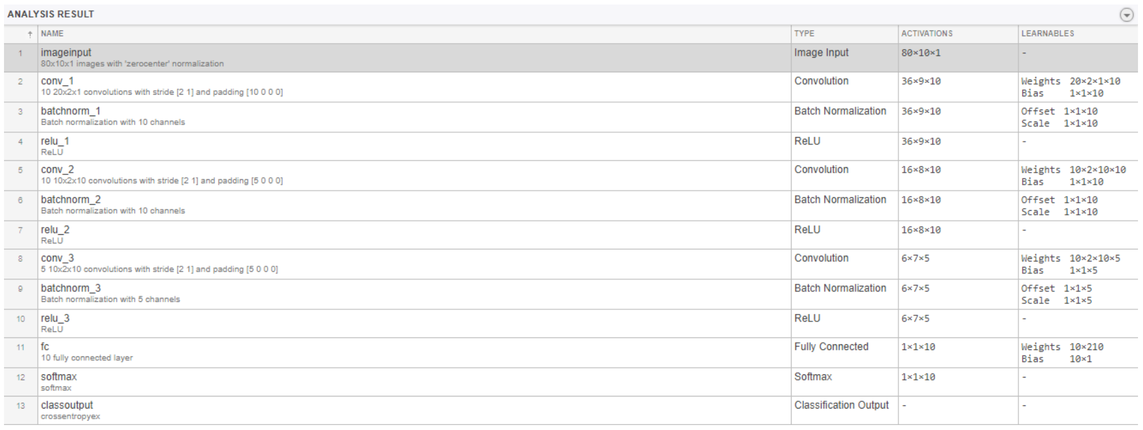
Training options:

Solver	sgdm, rmsprop, adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.01 (sgdm), Constant 0.001 (rmsprop, adam)

Network Structure:

Analog signal positional analysis using SGDM:

ASK	Accuracy	0.9129
	Execution time	0.9224s (for 10000 iterations)
	Execution per iteration	$0.92 * 10^{-4}$s


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 80x10x1 images with 'zerocenter' normalization	Image Input	80×10×1	-
2	conv_1 10 20x2x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	36×9×10	Weights 20×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	36×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	36×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	16×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	16×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

FSK	Accuracy	0.9492
	Execution time	0.9913s (for 10000 iterations)
	Execution per iteration	$0.99 * 10^{-4}$s

PSK	Accuracy	0.9469
	Execution time	1.1098s (for 10000 iterations)
	Execution per iteration	$1.11 * 10^{-4}$s

Analog signal positional analysis using RMSProp:

ASK	Accuracy	0.9087
	Execution time	0.8868s (for 10000 iterations)
	Execution per iteration	$0.89 * 10^{-4}$s

FSK	Accuracy	0.9388
	Execution time	0.8176s (for 10000 iterations)
	Execution per iteration	$0.81 * 10^{-4}$s

PSK	Accuracy	0.9372
	Execution time	0.8648s (for 10000 iterations)
	Execution per iteration	$0.86 * 10^{-4}$s

Analog signal positional analysis using ADAM:

ASK	Accuracy	0.9102
	Execution time	0.8351s (for 10000 iterations)
	Execution per iteration	$0.84 * 10^{-4}$s

FSK	Accuracy	0.9490
	Execution time	0.8440s (for 10000 iterations)
	Execution per iteration	$0.84 * 10^{-4}$s

PSK	Accuracy	0.9450
	Execution time	0.8267s (for 10000 iterations)
	Execution per iteration	$0.83 * 10^{-4}$s

B.2 Positional comparison of time-data and FFT

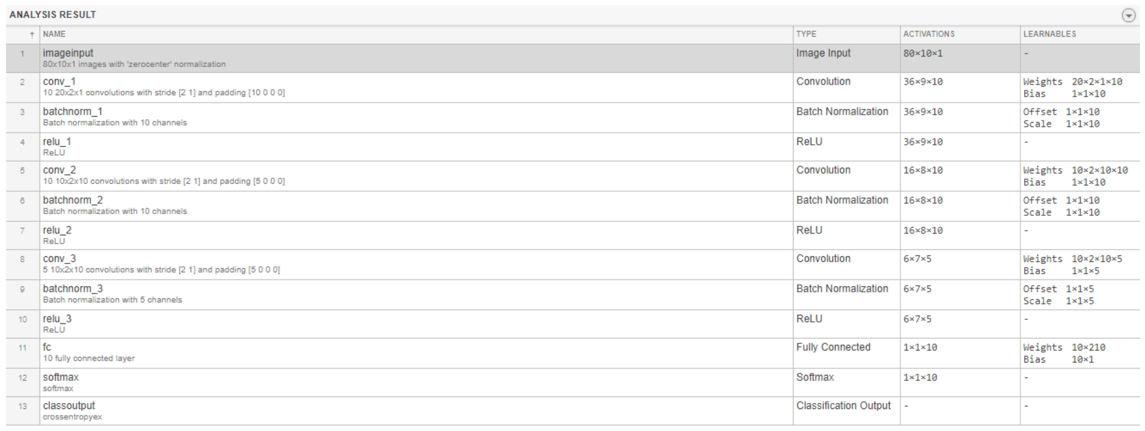
Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	80 bit area
Data bits	1-80
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.001

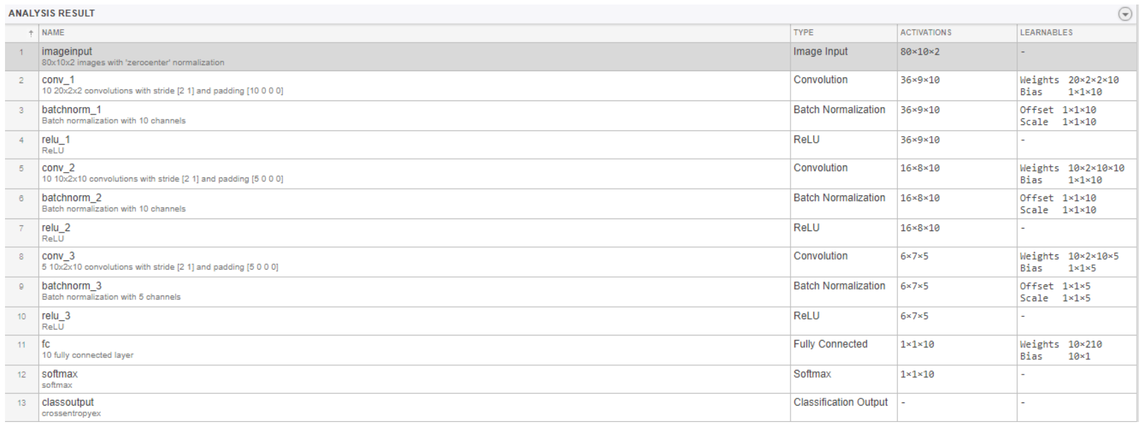
Network Structure for time data and FFT (amplitude only):



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 80x10x1 images with 'zerocenter' normalization	Image Input	80×10×1	-
2	conv_1 10 20x2x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	36×9×10	Weights 20×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	36×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	36×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	16×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	16×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Network Structure for FFT (amplitude and phase):



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 80x10x2 images with 'zerocenter' normalization	Image Input	80×10×2	-
2	conv_1 10 20x2x2 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	36×9×10	Weights 20×2×2×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	36×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	36×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	16×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	16×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

PSK time-domain classification results

Accuracy	0.9450
Execution time	0.8267s (for 10000 iterations)
Execution per iteration	$0.83 * 10^{-4}$s

PSK FFT classification results with amplitude and phase:

Accuracy	0.9140
FFT time	1.7156s
FFT time per iteration	$1.72 * 10^{-4}$s
Execution time	0.9364s (for 10000 iterations)
Execution per iteration	$0.94 * 10^{-4}$s

PSK FFT classification results with only amplitude:

Accuracy	0.9134
FFT time	0.2425s
FFT time per iteration	$0.24 * 10^{-4}$s
Execution time	0.9370s (for 10000 iterations)
Execution per iteration	$0.94 * 10^{-4}$s

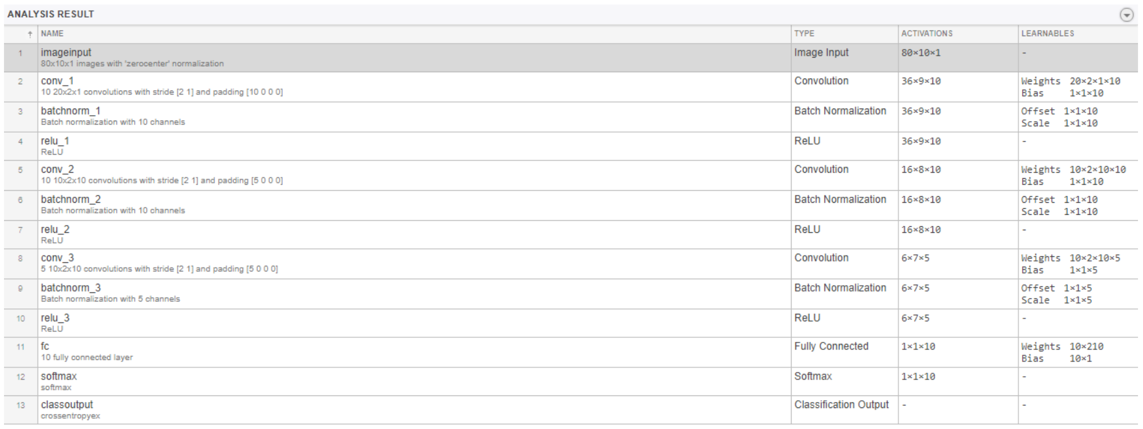
B.3 ASK, FSK and PSK comparison at different SNR for positional analysis

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	Varies
Input size	80 bit area
Data bits	1-80
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.001

Network Structure:



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 80x10x1 images with 'zerocenter' normalization	Image Input	80×10×1	-
2	conv_1 10 20x2x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	36×9×10	Weights 20×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	36×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	36×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	16×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	16×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

6SNR

	Accuracy	0.9897
ASK	Execution time	0.8672s (for 8000 iterations)
	Execution per iteration	$0.87 * 10^{-4}$s

	Accuracy	0.9886
FSK	Execution time	0.8285s (for 8000 iterations)
	Execution per iteration	$0.83 * 10^{-4}$s

	Accuracy	0.9871
PSK	Execution time	0.7933s (for 8000 iterations)
	Execution per iteration	$0.79 * 10^{-4}$s

-6SNR

ASK	Accuracy	0.6580
	Execution time	0.8736s (for 8000 iterations)
	Execution per iteration	$0.87 * 10^{-4}$s

FSK	Accuracy	0.8127
	Execution time	0.8517s (for 8000 iterations)
	Execution per iteration	$0.85 * 10^{-4}$s

PSK	Accuracy	0.7981
	Execution time	0.8182s (for 8000 iterations)
	Execution per iteration	$0.82 * 10^{-4}$s

B.4 Comparison between the CNN and FNN

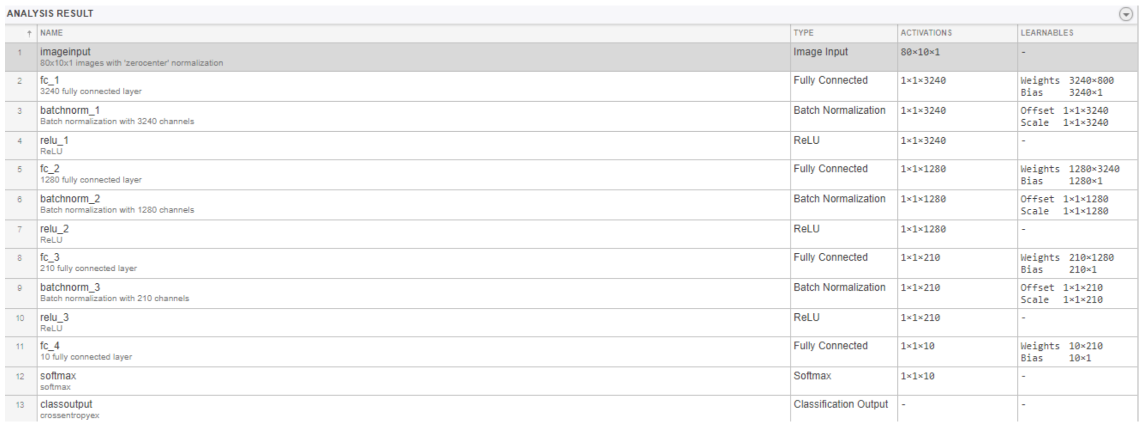
Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	80 bit area
Data bits	1-80
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.001

Network Structure:



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 80x10x1 images with 'zerocenter' normalization	Image Input	80×10×1	-
2	fc_1 3240 fully connected layer	Fully Connected	1×1×3240	Weights 3240×800 Bias 3240×1
3	batchnorm_1 Batch normalization with 3240 channels	Batch Normalization	1×1×3240	Offset 1×1×3240 Scale 1×1×3240
4	relu_1 ReLU	ReLU	1×1×3240	-
5	fc_2 1280 fully connected layer	Fully Connected	1×1×1280	Weights 1280×3240 Bias 1280×1
6	batchnorm_2 Batch normalization with 1280 channels	Batch Normalization	1×1×1280	Offset 1×1×1280 Scale 1×1×1280
7	relu_2 ReLU	ReLU	1×1×1280	-
8	fc_3 210 fully connected layer	Fully Connected	1×1×210	Weights 210×1280 Bias 210×1
9	batchnorm_3 Batch normalization with 210 channels	Batch Normalization	1×1×210	Offset 1×1×210 Scale 1×1×210
10	relu_3 ReLU	ReLU	1×1×210	-
11	fc_4 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.7547
Execution time	3.2751s (for 10000 iterations)
Execution per iteration	$3.28 * 10^{-4}$s

B.5 Positional analysis comparison of different number of samples per bit

A comparison to 10 samples per bit can be found in Appendix B.1.

Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	80 bit area
Data bits	1-80
Samples per bit	Varies
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

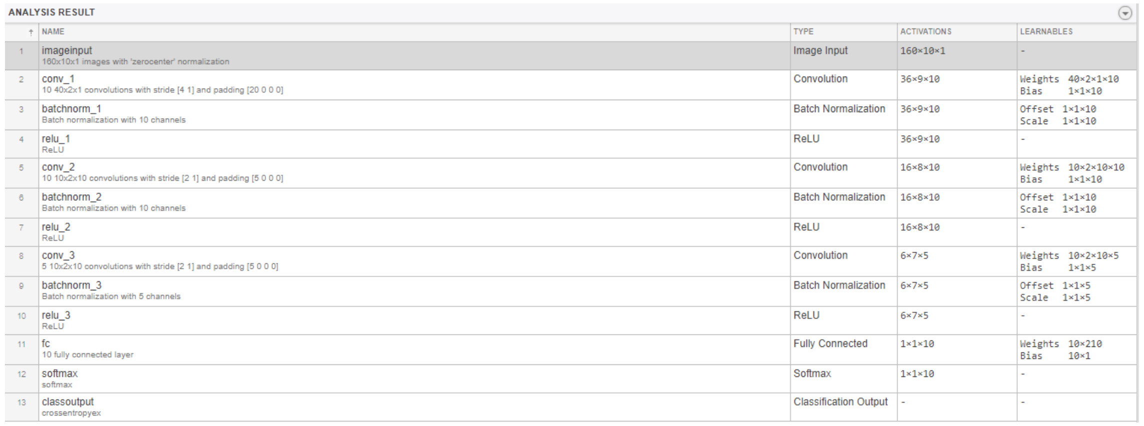
Training options:

Solver	adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.001

20 samples per bit

Filter and stride have here been doubled when compared to the 10 samples/bit version in order to make the two cases as similar as possible for the system. Padding has also been adjusted to be equal to half the filter size.

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	36×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	36×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	36×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	16×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	16×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×210 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9708
Execution time	0.9526s (for 10000 iterations)
Execution per iteration	$0.9526 * 10^{-4}$s

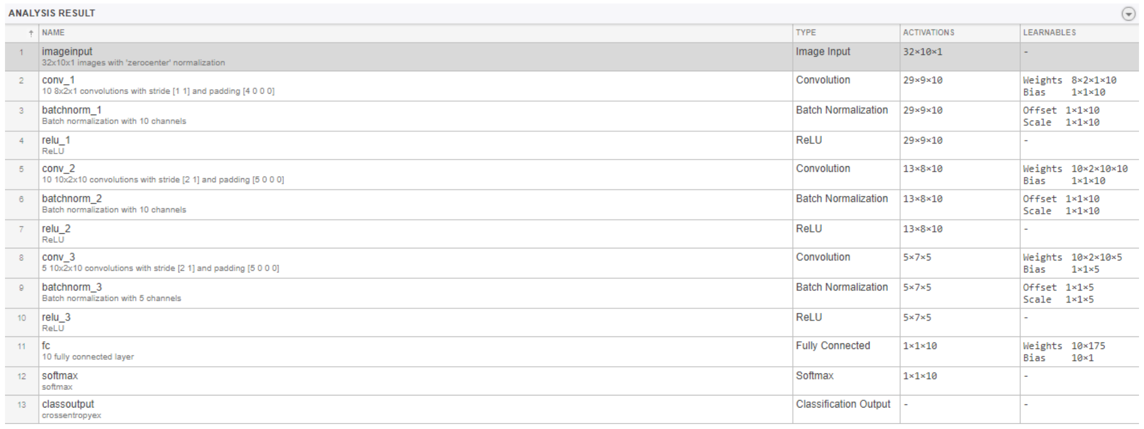
Same scenario except for the padding at the first convolutional layer being 10 instead of 20

Accuracy	0.9832
Execution time	0.9153s (for 10000 iterations)
Execution per iteration	$0.92 * 10^{-4}$s

4 samples per bit

Filter and stride have here been reduces when compared to the 10 samples/bit version in order to make the two cases as similar as possible for the system. Padding has also been adjusted to be equal to half the filter size.

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 32x10x1 images with 'zerocenter' normalization	Image Input	32×10×1	-
2	conv_1 10 8x2x1 convolutions with stride [1 1] and padding [4 0 0 0]	Convolution	29×9×10	Weights 8×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	29×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	29×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	13×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	13×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	13×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	5×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	5×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	5×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×175 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9214
Execution time	0.7916s (for 10000 iterations)
Execution per iteration	$0.79 * 10^{-4}$s

B.6 Signal position comparison of differently sized CNNs

Signal information:

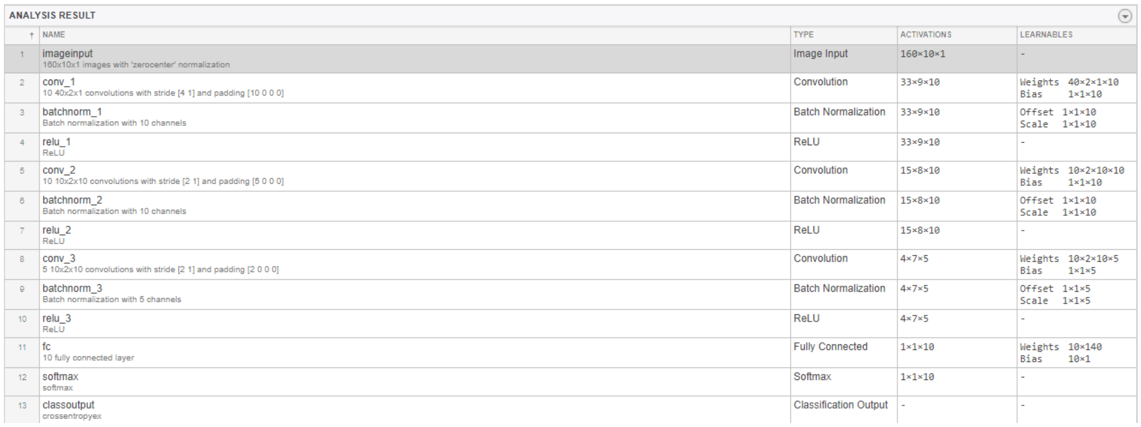
Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	80 bit area
Data bits	1-80
Samples per bit	20
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	78
Learning rate	Constant 0.001

Unmodified structure with results:

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	4×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	4×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	4×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×140 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

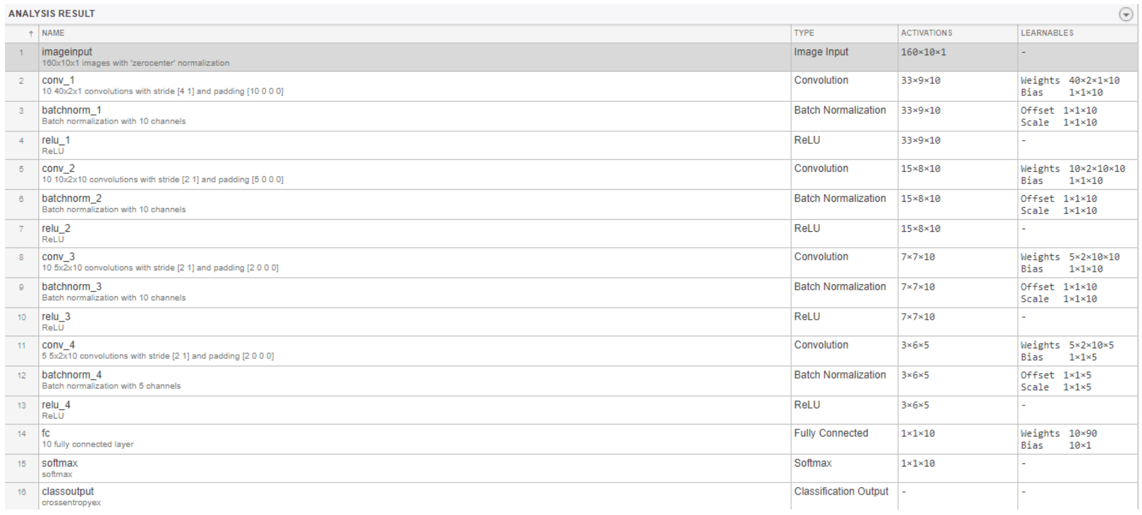
Results:

Accuracy	0.9739
Execution time Execution per iteration	0.8983s (for 10000 iterations) $0.90 * 10^{-4}$s

4 CNN layers

Network Structure:

Results:



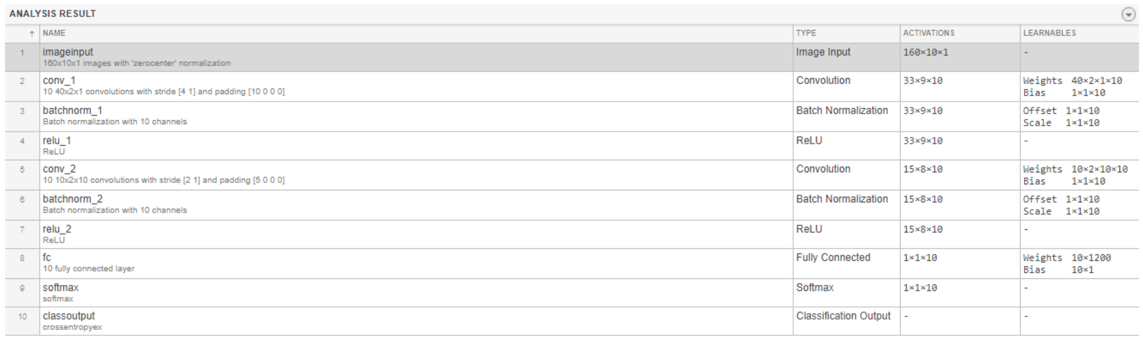
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×8×10	-
8	conv_3 10 5x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	7×7×10	Weights 5×2×10×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	7×7×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	7×7×10	-
11	conv_4 5 5x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	3×6×5	Weights 5×2×10×5 Bias 1×1×5
12	batchnorm_4 Batch normalization with 5 channels	Batch Normalization	3×6×5	Offset 1×1×5 Scale 1×1×5
13	relu_4 ReLU	ReLU	3×6×5	-
14	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×90 Bias 10×1
15	softmax softmax	Softmax	1×1×10	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9847
Execution time	0.9742s (for 10000 iterations)
Execution per iteration	$0.97 * 10^{-4}$s

2 CNN layers

Network Structure:



ANALYSIS RESULT

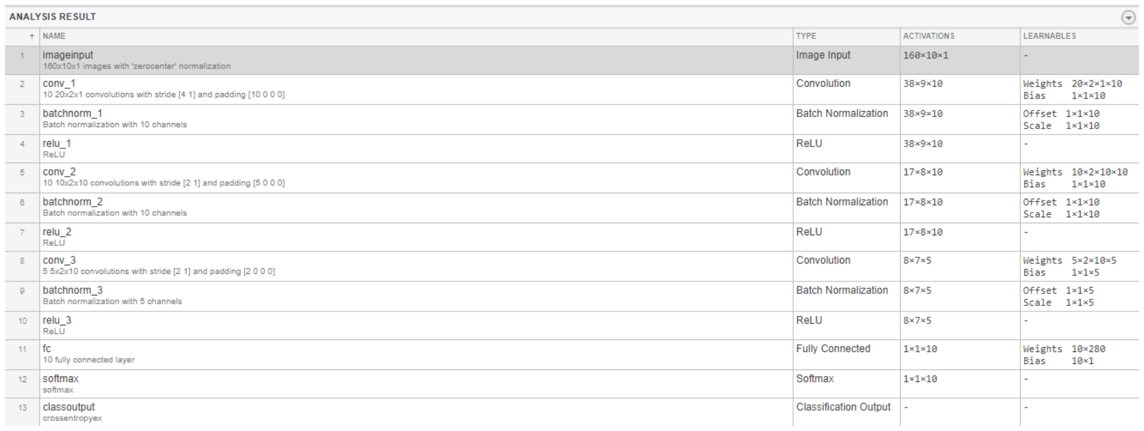
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×8×10	-
8	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×1200 Bias 10×1
9	softmax softmax	Softmax	1×1×10	-
10	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9787
Execution time	0.8338s (for 10000 iterations)
Execution per iteration	$0.83 * 10^{-4}$s

Smaller sized convolutional filters

Network Structure:



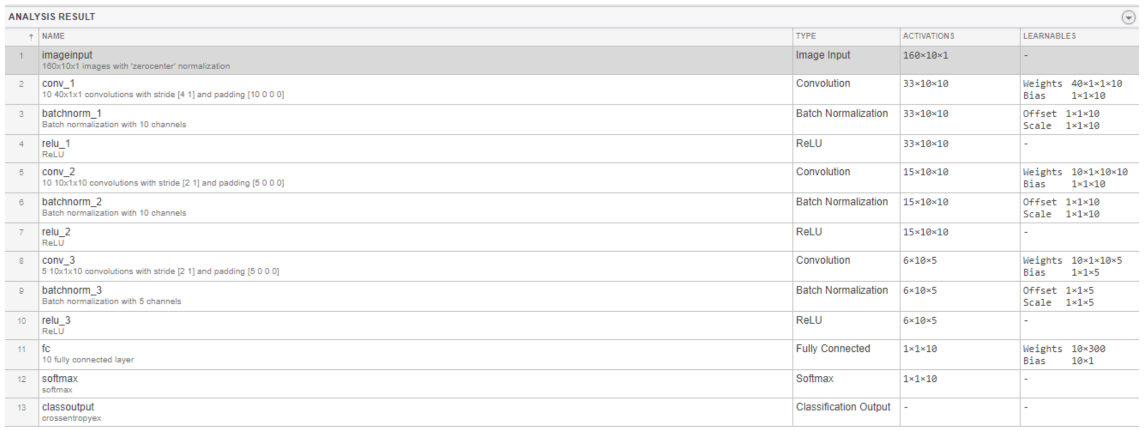
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 20x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	38×9×10	Weights 20×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	38×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	38×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	17×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	17×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	17×8×10	-
8	conv_3 5 5x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	8×7×5	Weights 5×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	8×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	8×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×280 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9799
Execution time	0.9236s (for 10000 iterations)
Execution per iteration	$0.92 * 10^{-4}$s

Network Structure:



ANALYSIS RESULT

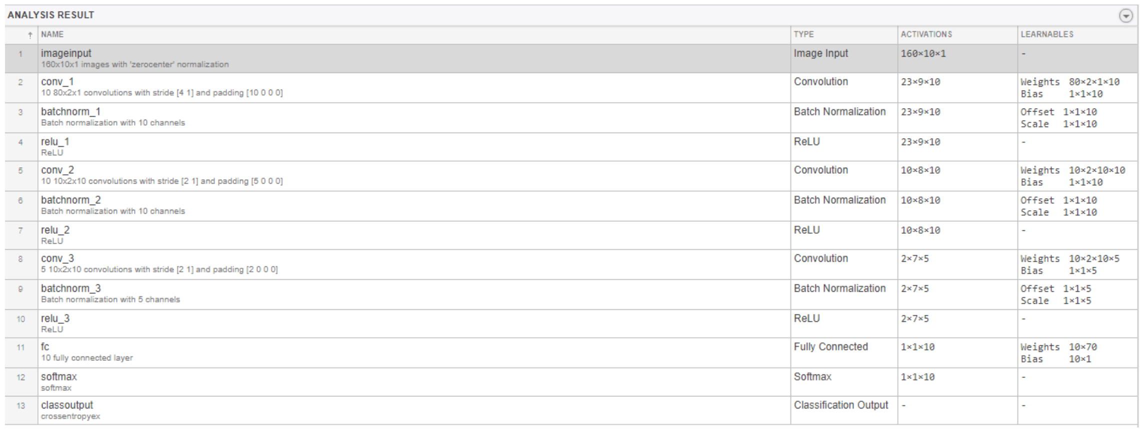
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x1x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×10×10	Weights 40×1×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×10×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×10×10	-
5	conv_2 10 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×10×10	Weights 10×1×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×10×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×10×10	-
8	conv_3 5 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×10×5	Weights 10×1×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×10×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×10×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×300 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results for filters with size 1 along 2nd dimension:

Accuracy	0.9912
Execution time	0.9007s (for 10000 iterations)
Execution per iteration	$0.90 * 10^{-4}$s

Larger sized convolutional filters

Network Structure:



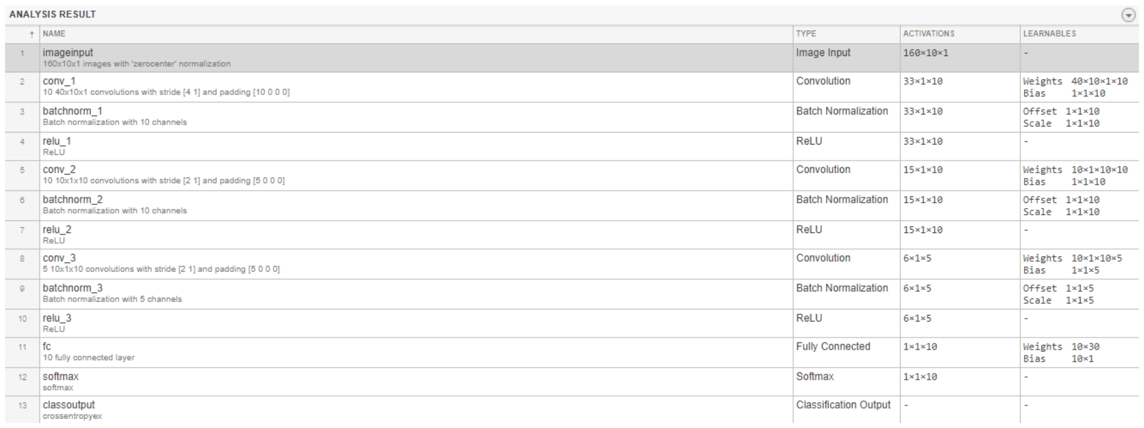
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 80x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	23×9×10	Weights 80×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	23×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	23×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	10×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	10×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	10×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	2×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	2×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	2×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×70 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9823
Execution time	0.7320s (for 10000 iterations)
Execution per iteration	$0.73 * 10^{-4}$s

Network Structure:



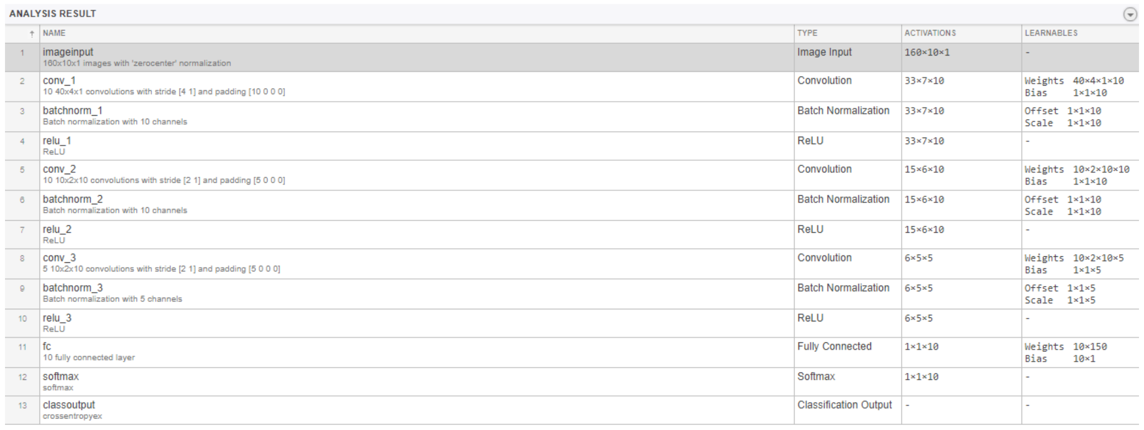
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x10x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×1×10	Weights 40×10×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×1×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×1×10	-
5	conv_2 10 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×1×10	Weights 10×1×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×1×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×1×10	-
8	conv_3 5 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×1×5	Weights 10×1×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×1×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×1×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×30 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results with filter size of 10 along 2nd dimension:

Accuracy	0.7769
Execution time	0.3225s (for 10000 iterations)
Execution per iteration	$0.32 * 10^{-4}$s

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x4x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×7×10	Weights 40×4×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×7×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×7×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×6×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×6×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×6×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×5×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×5×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×5×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×150 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

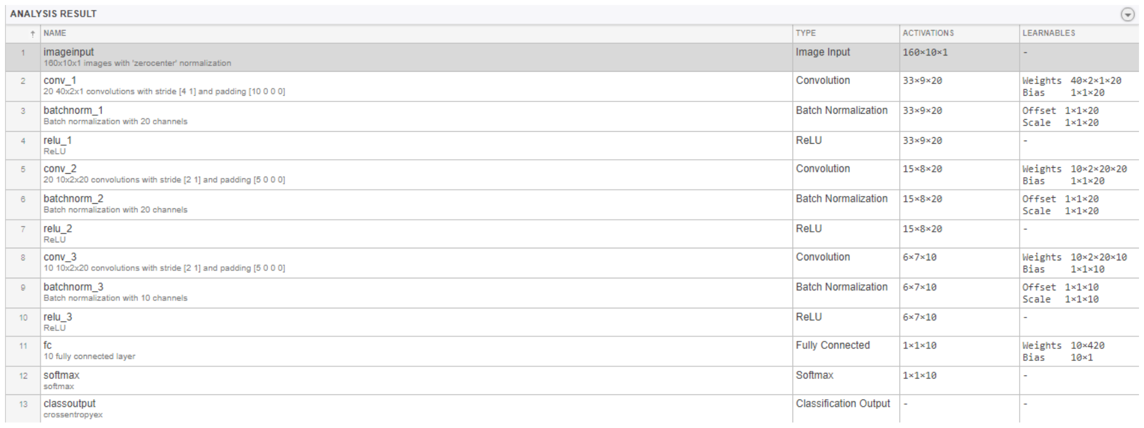
Results with filter size of 4 along 2nd dimension:

Accuracy	0.9626
Execution time	0.8116s (for 10000 iterations)
Execution per iteration	$0.81 * 10^{-4}$s

Increased number of convolutional filters

Number of filters for each CNN layer is here doubled compared to the unmodified version

Network Structure:



ANALYSIS RESULT

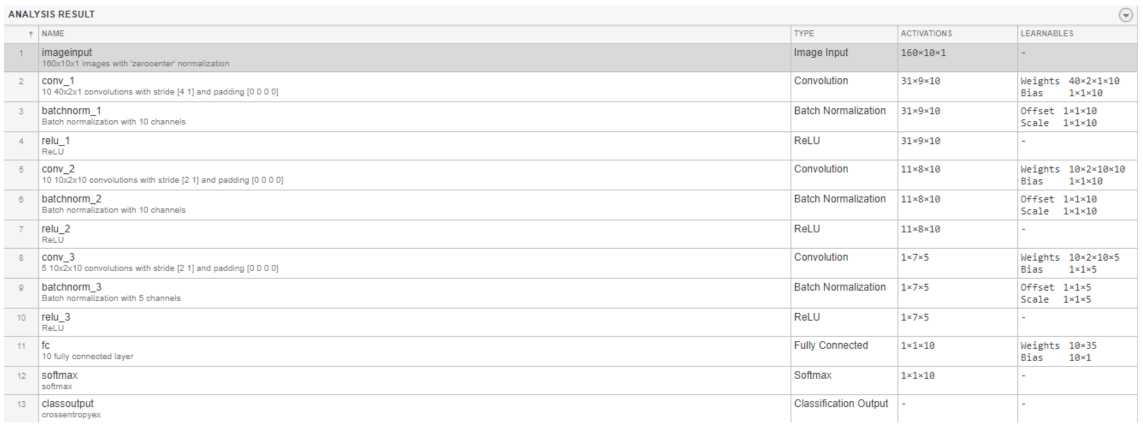
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 20 40x2x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×9×20	Weights 40×2×1×20 Bias 1×1×20
3	batchnorm_1 Batch normalization with 20 channels	Batch Normalization	33×9×20	Offset 1×1×20 Scale 1×1×20
4	relu_1 ReLU	ReLU	33×9×20	-
5	conv_2 20 10x2x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×8×20	Weights 10×2×20×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	15×8×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	15×8×20	-
8	conv_3 10 10x2x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×7×10	Weights 10×2×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	6×7×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	6×7×10	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×420 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9836
Execution time	1.8991s (for 10000 iterations)
Execution per iteration	$1.90 * 10^{-4}$s

No padding

Network Structure:



ANALYSIS RESULT

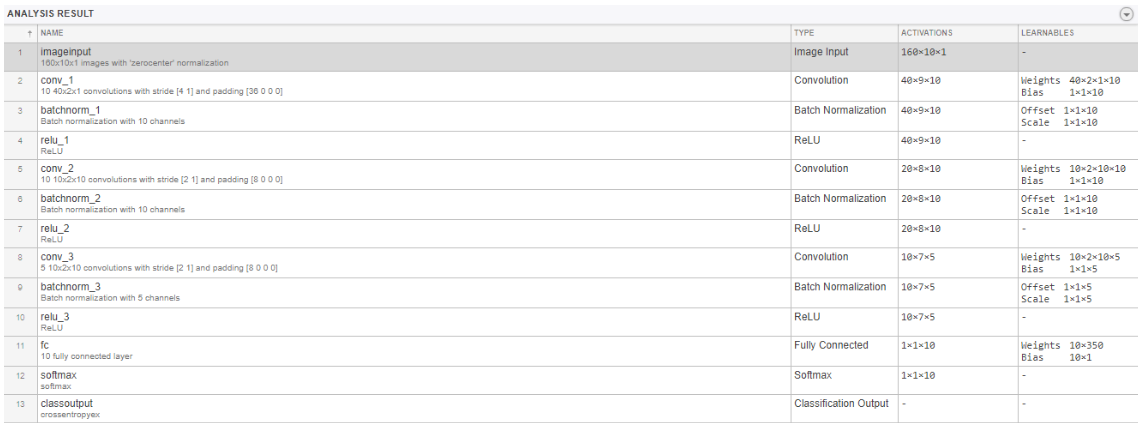
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [0 0 0 0]	Convolution	31×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	31×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	31×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	11×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	11×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	11×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	1×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	1×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	1×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×35 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9661
Execution time	0.7845s (for 10000 iterations)
Execution per iteration	$0.78 * 10^{-4}$s

Maximum padding

Maximum padding in this case means that the outermost activations only include a number of inputs (that are not part of the padding) equal to the size of the stride. With a stride of 2, this means only the 2 first samples are part of the first position of the filter, and equivalently for the last position.



ANALYSIS RESULT

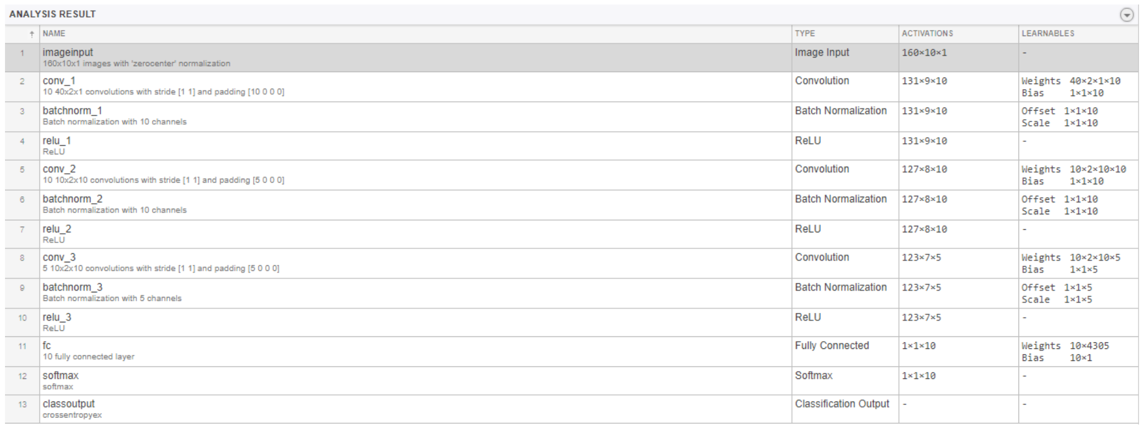
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 1] and padding [36 0 0 0]	Convolution	40×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	40×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	40×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [8 0 0 0]	Convolution	20×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	20×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	20×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [8 0 0 0]	Convolution	10×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	10×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	10×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×350 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9875
Execution time	1.0406s (for 10000 iterations)
Execution per iteration	$1.04 * 10^{-4}$s

Less stride

Network Structure:



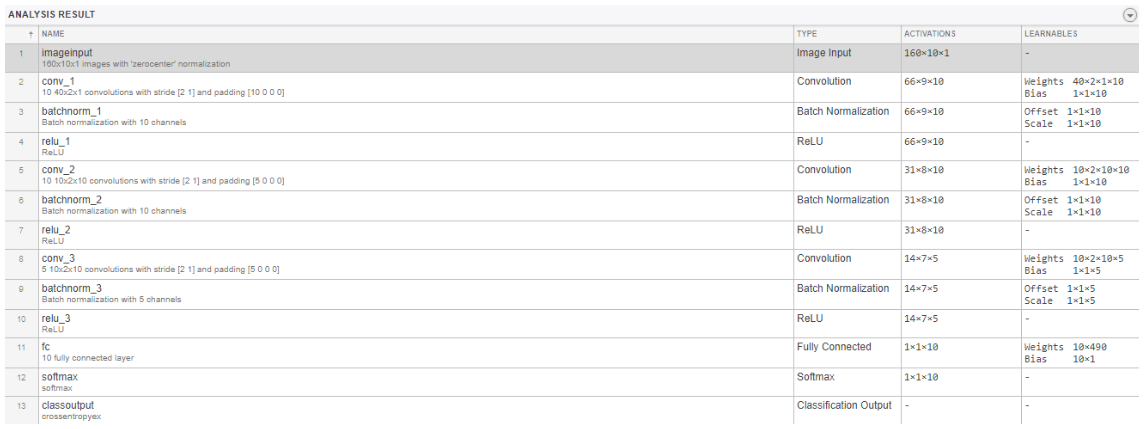
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [1 1] and padding [10 0 0 0]	Convolution	131×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	131×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	131×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [1 1] and padding [5 0 0 0]	Convolution	127×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	127×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	127×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [1 1] and padding [5 0 0 0]	Convolution	123×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	123×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	123×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×4305 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9799
Execution time	8.6617s (for 10000 iterations)
Execution per iteration	$8.66 * 10^{-4}$s

Network Structure:



ANALYSIS RESULT

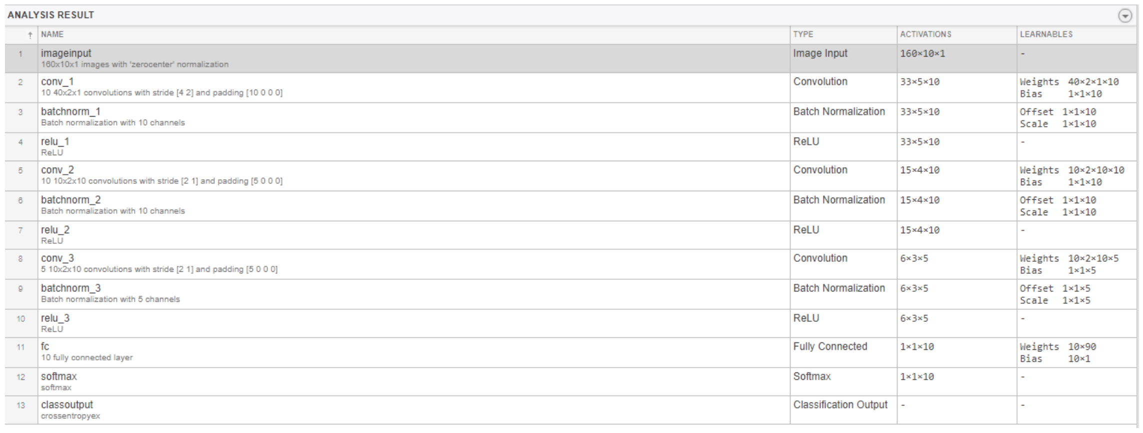
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	66×9×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	66×9×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	66×9×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	31×8×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	31×8×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	31×8×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	14×7×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	14×7×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	14×7×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×490 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results:

Accuracy	0.9824
Execution time	1.8013s (for 10000 iterations)
Execution per iteration	$1.80 * 10^{-4}$s

More stride

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x2x1 convolutions with stride [4 2] and padding [10 0 0 0]	Convolution	33×5×10	Weights 40×2×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×5×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×5×10	-
5	conv_2 10 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×4×10	Weights 10×2×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×4×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×4×10	-
8	conv_3 5 10x2x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×3×5	Weights 10×2×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×3×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×3×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×90 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

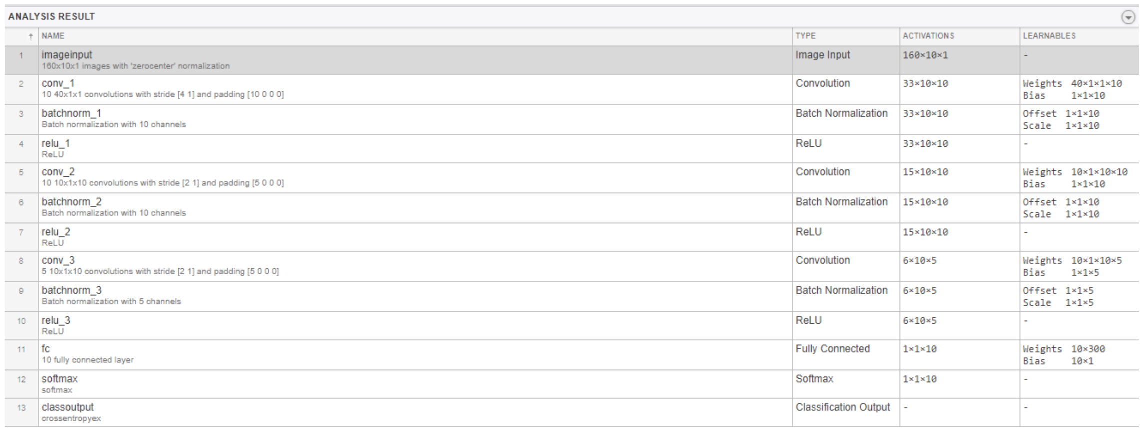
Results:

Accuracy	0.9557
Execution time	0.4877s (for 10000 iterations)
Execution per iteration	$0.49 * 10^{-4}$s

B.7 Final structure for positional analysis neural network

For 10 byte region

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x10x1 images with 'zerocenter' normalization	Image Input	160×10×1	-
2	conv_1 10 40x1x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	33×10×10	Weights 40×1×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	33×10×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	33×10×10	-
5	conv_2 10 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	15×10×10	Weights 10×1×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	15×10×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	15×10×10	-
8	conv_3 5 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	6×10×5	Weights 10×1×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	6×10×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	6×10×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×300 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

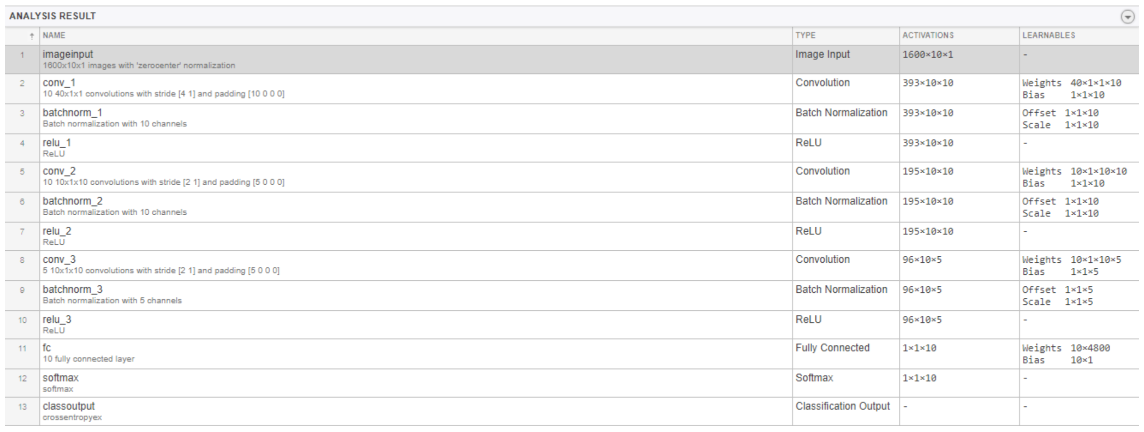
Results

Accuracy	0.9912
Execution time	0.9007s (for 10000 iterations)
Execution per iteration	$0.90 * 10^{-4}$s

A note to make is that this is identical to one of the test-cases shown earlier, using optimization for stride. Combining this optimization with optimization to padding or number of convolutional layers did not yield higher accuracy. Increasing the number of filters also did not increase accuracy

For 100 byte region

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1600x10x1 images with 'zerocenter' normalization	Image Input	1600×10×1	-
2	conv_1 10 40x1x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	393×10×10	Weights 40×1×1×10 Bias 1×1×10
3	batchnorm_1 Batch normalization with 10 channels	Batch Normalization	393×10×10	Offset 1×1×10 Scale 1×1×10
4	relu_1 ReLU	ReLU	393×10×10	-
5	conv_2 10 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	195×10×10	Weights 10×1×10×10 Bias 1×1×10
6	batchnorm_2 Batch normalization with 10 channels	Batch Normalization	195×10×10	Offset 1×1×10 Scale 1×1×10
7	relu_2 ReLU	ReLU	195×10×10	-
8	conv_3 5 10x1x10 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	96×10×5	Weights 10×1×10×5 Bias 1×1×5
9	batchnorm_3 Batch normalization with 5 channels	Batch Normalization	96×10×5	Offset 1×1×5 Scale 1×1×5
10	relu_3 ReLU	ReLU	96×10×5	-
11	fc 10 fully connected layer	Fully Connected	1×1×10	Weights 10×4800 Bias 10×1
12	softmax softmax	Softmax	1×1×10	-
13	classoutput crossentropyex	Classification Output	-	-

Results at 0 SNR:

Accuracy	1.0
Execution time	12.2042s (for 10000 iterations)
Execution per iteration	$12.0 * 10^{-4}$s

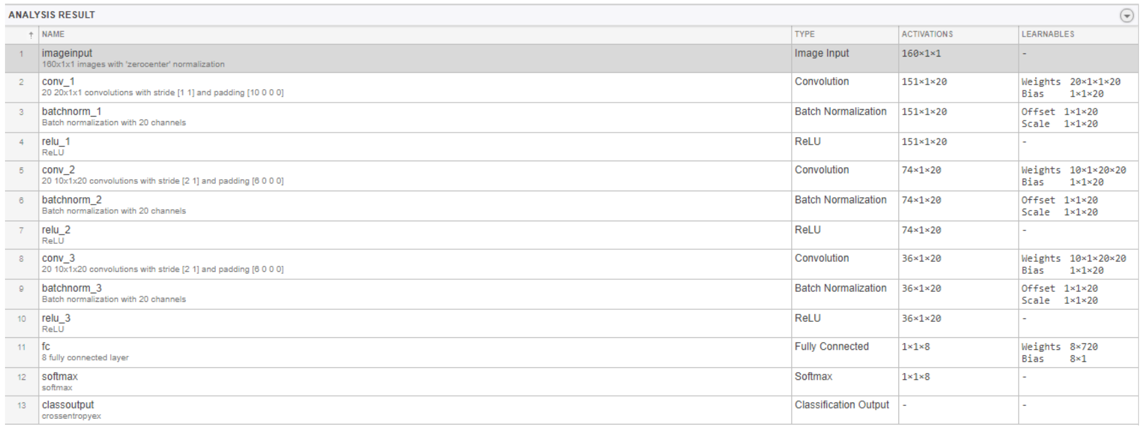
Results at -6 SNR:

Accuracy	0.9994
Execution time	12.16812s (for 10000 iterations)
Execution per iteration	$12.0 * 10^{-4}$s

A point to make here is that due to the data being randomly placed inside the area covered by each output neuron, the number of corner cases, referring to cases where the starting position is only one or a few samples from the border between the areas covered by two output neurons, are probably few in number.

For 1 byte region

Network Structure:



ANALYSIS RESULT

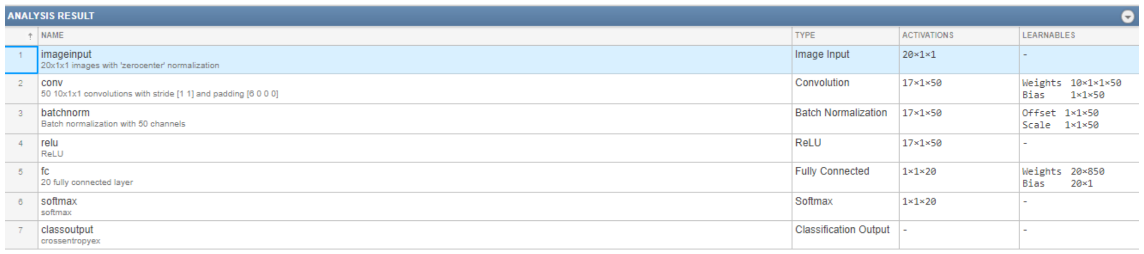
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 160x1x1 images with 'zerocenter' normalization	Image Input	160×1×1	-
2	conv_1 20 20x1x1 convolutions with stride [1 1] and padding [10 0 0 0]	Convolution	151×1×20	Weights 20×1×1×20 Bias 1×1×20
3	batchnorm_1 Batch normalization with 20 channels	Batch Normalization	151×1×20	Offset 1×1×20 Scale 1×1×20
4	relu_1 ReLU	ReLU	151×1×20	-
5	conv_2 20 10x1x20 convolutions with stride [2 1] and padding [6 0 0 0]	Convolution	74×1×20	Weights 10×1×20×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	74×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	74×1×20	-
8	conv_3 20 10x1x20 convolutions with stride [2 1] and padding [6 0 0 0]	Convolution	36×1×20	Weights 10×1×20×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	36×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	36×1×20	-
11	fc 8 fully connected layer	Fully Connected	1×1×8	Weights 8×720 Bias 8×1
12	softmax softmax	Softmax	1×1×8	-
13	classoutput crossentropyex	Classification Output	-	-

Results at 0 SNR:

Accuracy	0.9918
Execution time	0.7651s (for 10000 iterations)
Execution per iteration	$0.96 * 10^{-4}$s

For 1 bit region

Network Structure:



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 20x1x1 images with 'zerocenter' normalization	Image Input	20×1×1	-
2	conv 50 10x1x1 convolutions with stride [1 1] and padding [6 0 0 0]	Convolution	17×1×50	Weights 10×1×1×50 Bias 1×1×50
3	batchnorm Batch normalization with 50 channels	Batch Normalization	17×1×50	Offset 1×1×50 Scale 1×1×50
4	relu ReLU	ReLU	17×1×50	-
5	fc 20 fully connected layer	Fully Connected	1×1×20	Weights 20×850 Bias 20×1
6	softmax softmax	Softmax	1×1×20	-
7	classoutput crossentropyex	Classification Output	-	-

Results at 0 SNR:

Accuracy	0.7226
Execution time	0.2693s (for 10000 iterations)
Execution per iteration	$0.13 * 10^{-4}$s

Appendix C

Artificial neural network results for signal length test-cases

C.1 Comparison between the different solvers for analog signal length analysis

Signal information:

Modulation Scheme	ASK, FSK, PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

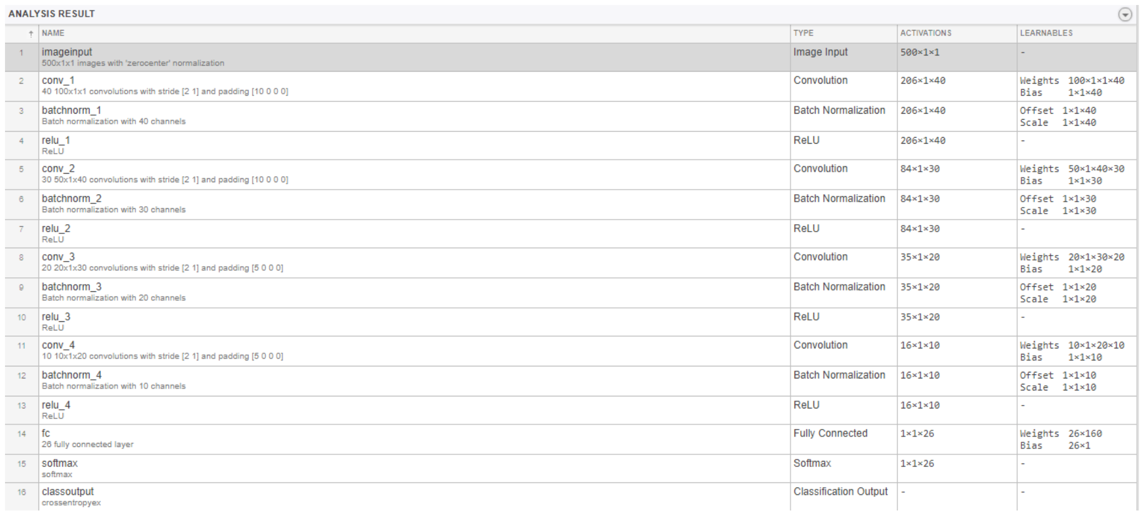
Training options:

Solver	sgdm, rmsprop, adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.01 (sgdm), Constant 0.001 (rmsprop, adam)

Network Structure:

Analog signal length analysis using SGDM:

ASK	Accuracy	0.3067
	Accuracy with max 1 bit wrong	0.7130
	Execution time	5.1360s (for 13000 iterations)
	Execution per iteration	$3.95 * 10^{-4}$s


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 40 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×40	Weights 100×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

FSK	Accuracy	0.6575
	Accuracy with max 1 bit wrong	0.9775
	Execution time	5.4057s (for 13000 iterations)
	Execution per iteration	$4.16 * 10^{-4}$s

PSK	Accuracy	0.6273
	Accuracy with max 1 bit wrong	0.9726
	Execution time	5.2802s (for 13000 iterations)
	Execution per iteration	$4.06 * 10^{-4}$s

Analog signal length analysis using RMSProp

ASK	Accuracy	0.3281
	Accuracy with max 1 bit wrong	0.7368
	Execution time	5.0345s (for 13000 iterations)
	Execution per iteration	$3.87 * 10^{-4}$s

FSK	Accuracy	0.6975
	Accuracy with max 1 bit wrong	0.9792
	Execution time	4.9742s (for 13000 iterations)
	Execution per iteration	$3.83 * 10^{-4}$s

PSK	Accuracy	0.7399
	Accuracy with max 1 bit wrong	0.9848
	Execution time	4.9100s (for 13000 iterations)
	Execution per iteration	$3.78 * 10^{-4}$s

Analog signal length comparison using ADAM

ASK	Accuracy	0.3979
	Accuracy with max 1 bit wrong	0.7980
	Execution time	4.7087s (for 13000 iterations)
	Execution per iteration	$3.62 * 10^{-4}$s

FSK	Accuracy	0.7643
	Accuracy with max 1 bit wrong	0.9898
	Execution time	4.9796s (for 13000 iterations)
	Execution per iteration	$3.83 * 10^{-4}$s

PSK	Accuracy	0.7168
	Accuracy with max 1 bit wrong	0.9779
	Execution time	5.1230s (for 13000 iterations)
	Execution per iteration	$3.94 * 10^{-4}$s

C.2 Signal length comparison of time-data and FFT

The matrix size for time-data: (500,1,1,13000), with 500 instances for each output.

The matrix size for FFT with amplitude and phase: (500,1,2,13000), where first index of 3rd dimension are absolute values, and 2nd index of 3rd dimension is phase.

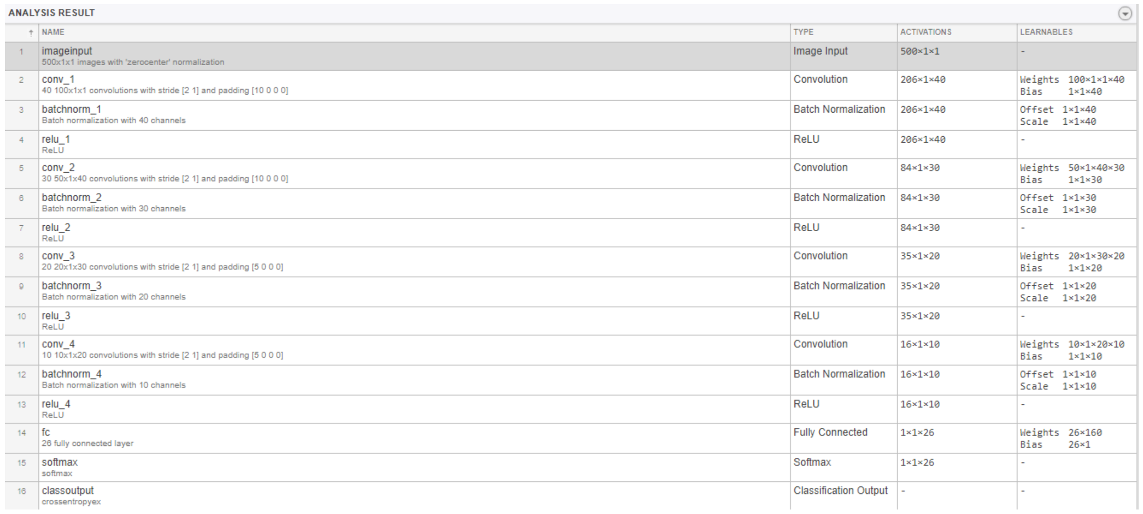
The matrix size for FFT with only amplitude: (500,1,1,13000), with 500 instances for each of the 26 outputs.

Training options:

Solver	adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.001

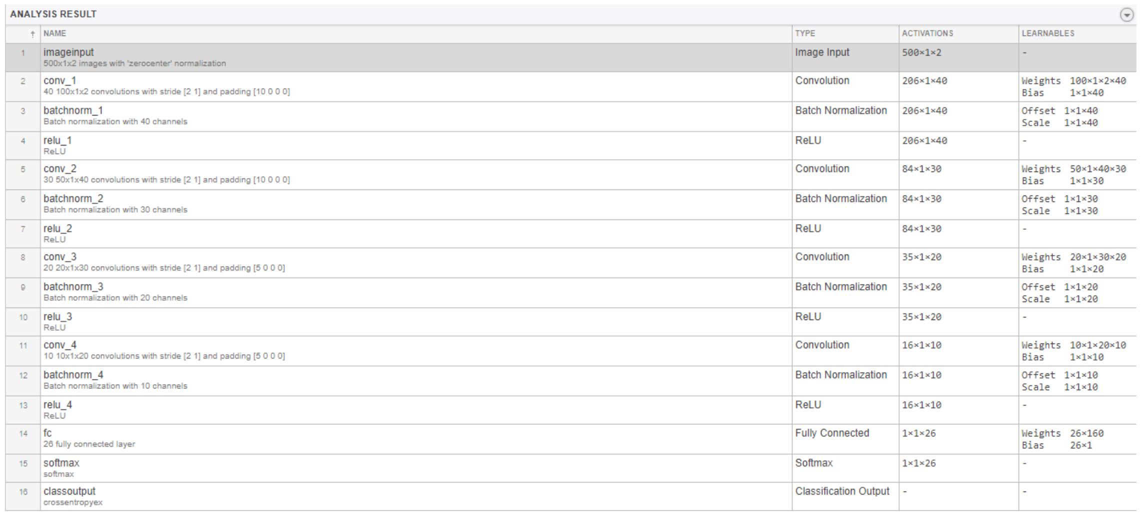
Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bit length	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Network Structure for time data and FFT (amplitude only):


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 40 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×40	Weights 100×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Network Structure for FFT (amplitude and phase):



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x2 images with 'zerocenter' normalization	Image Input	500×1×2	-
2	conv_1 40 100x1x2 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×40	Weights 100×1×2×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

PSK time-domain classification results

Accuracy Accuracy with max 1 bit wrong	0.7168 0.9779
Execution time Execution per iteration	5.1230s (for 13000 iterations) $3.94 * 10^{-4}$s

PSK FFT classification results with amplitude and phase:

Accuracy Accuracy with max 1 bit wrong	0.2731 0.6601
FFT time FFT time per iteration	1.2611s $0.97 * 10^{-4}$s
Execution time Execution per iteration	5.1012s (for 13000 iterations) $3.92 * 10^{-4}$s

PSK FFT classification results with only amplitude:

Accuracy Accuracy with max 1 bit wrong	0.2725 0.6592
FFT time FFT time per iteration	0.2485s $0.19 * 10^{-4}$s
Execution time Execution per iteration	4.8185s (for 13000 iterations) $3.71 * 10^{-4}$s

C.3 Signal length comparison beetween ASK, FSK, and PSK modulation scheme

Results for 0SNR can be seen in Appendix C.1.

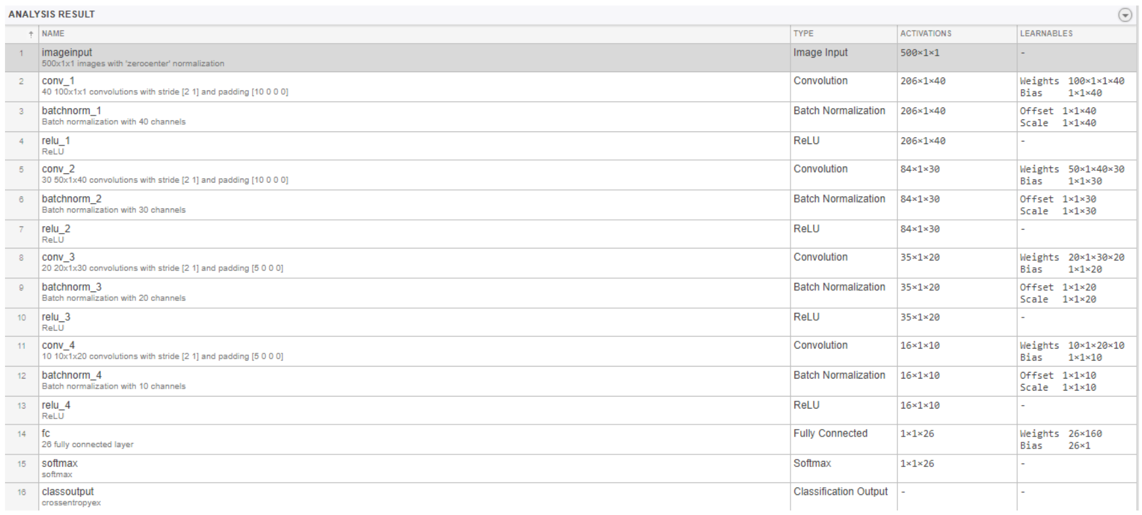
Signal information:

Modulation Scheme	ASK, FSK and PSK (separately)
Amplitude	50 peak
Noise Levels	varies
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.001

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	conv_1 40 100x1x1 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	206×1×40	Weights 100×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

6SNR

ASK classification results:

Accuracy	0.7073
Accuracy with max 1 bit wrong	0.9377
Execution time	5.2538s (for 13000 iterations)
Execution per iteration	$4.04 * 10^{-4}$s

FSK classification results:

Accuracy	0.9943
Accuracy with max 1 bit wrong	1.0
Execution time	4.9542s (for 13000 iterations)
Execution per iteration	$3.81 * 10^{-4}$s

PSK classification results:

Accuracy	0.9923
Accuracy with max 1 bit wrong	0.9999
Execution time	5.1648s (for 13000 iterations)
Execution per iteration	$3.97 * 10^{-4}$s

3.5NR

ASK classification results:

Accuracy	0.5436
Accuracy with max 1 bit wrong	0.9092
Execution time	4.7313s (for 13000 iterations)
Execution per iteration	$4.18 * 10^{-4}$s

FSK classification results:

Accuracy	0.9632
Accuracy with max 1 bit wrong	0.9992
Execution time	4.7454s (for 13000 iterations)
Execution per iteration	$3.65 * 10^{-4}$s

PSK classification results:

Accuracy	0.9771
Accuracy with max 1 bit wrong	1.0
Execution time	4.85233.73s (for 13000 iterations)
Execution per iteration	$*10^{-4}$s

C.4 Comparison between the CNN and FNN

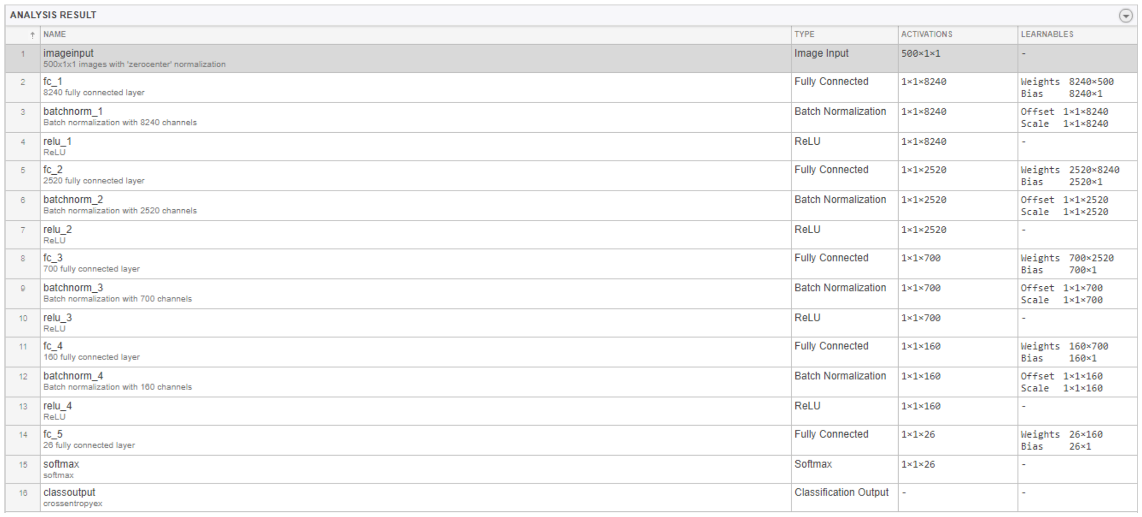
Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	3.5 SNR: approx 10.0 average amplitude , approx 37 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	10
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.001

Network Structure:



ANALYSIS RESULT

↑	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 500x1x1 images with 'zerocenter' normalization	Image Input	500×1×1	-
2	fc_1 8240 fully connected layer	Fully Connected	1×1×8240	Weights 8240×500 Bias 8240×1
3	batchnorm_1 Batch normalization with 8240 channels	Batch Normalization	1×1×8240	Offset 1×1×8240 Scale 1×1×8240
4	relu_1 ReLU	ReLU	1×1×8240	-
5	fc_2 2520 fully connected layer	Fully Connected	1×1×2520	Weights 2520×8240 Bias 2520×1
6	batchnorm_2 Batch normalization with 2520 channels	Batch Normalization	1×1×2520	Offset 1×1×2520 Scale 1×1×2520
7	relu_2 ReLU	ReLU	1×1×2520	-
8	fc_3 700 fully connected layer	Fully Connected	1×1×700	Weights 700×2520 Bias 700×1
9	batchnorm_3 Batch normalization with 700 channels	Batch Normalization	1×1×700	Offset 1×1×700 Scale 1×1×700
10	relu_3 ReLU	ReLU	1×1×700	-
11	fc_4 160 fully connected layer	Fully Connected	1×1×160	Weights 160×700 Bias 160×1
12	batchnorm_4 Batch normalization with 160 channels	Batch Normalization	1×1×160	Offset 1×1×160 Scale 1×1×160
13	relu_4 ReLU	ReLU	1×1×160	-
14	fc_5 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.2163
Accuracy with max 1 bit wrong	0.5585
Execution time	80.8351s (for 13000 iterations)
Execution per iteration	$62.18 * 10^{-4}$s

C.5 Signal length comparison of different number of samples per bit

A comparison to 10 samples per bit can be found in Appendix C.1.

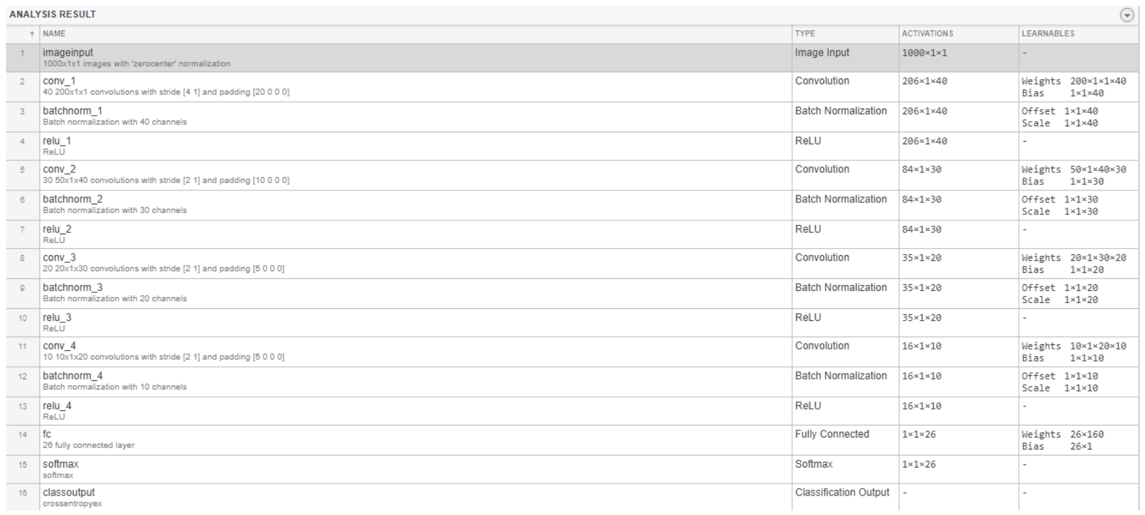
Signal information:

Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	varies
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.001

Network Structure:



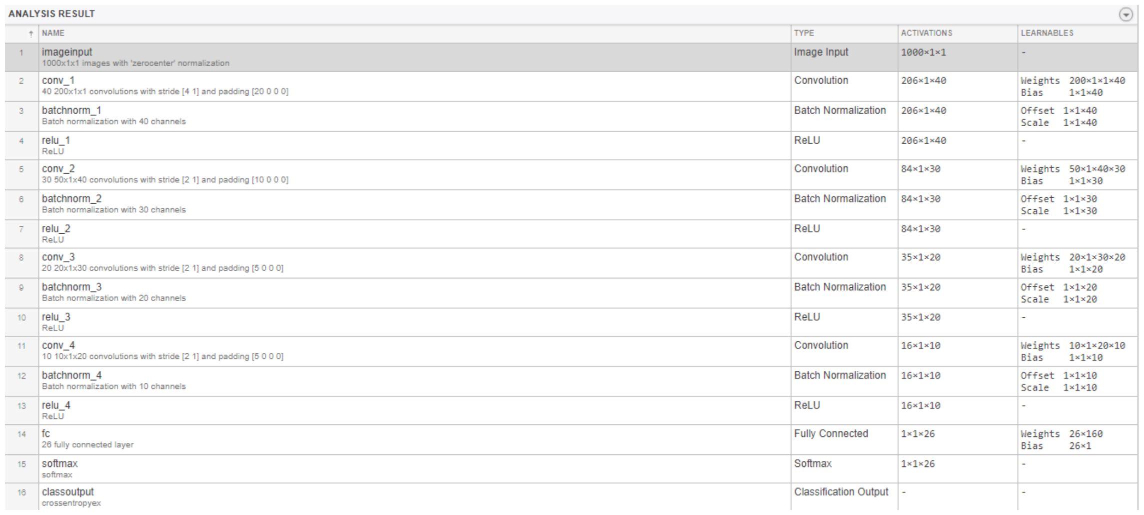
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 40 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×40	Weights 200×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Signal length with 20 samples per bit

Filter of the first convolutional layer has been doubled in length. The stride and padding for the first convolutional layers have also been doubled. This is done to create a neural network that should be as equivalent as possible to the 10 samples/bit version.

Network Structure:



ANALYSIS RESULT

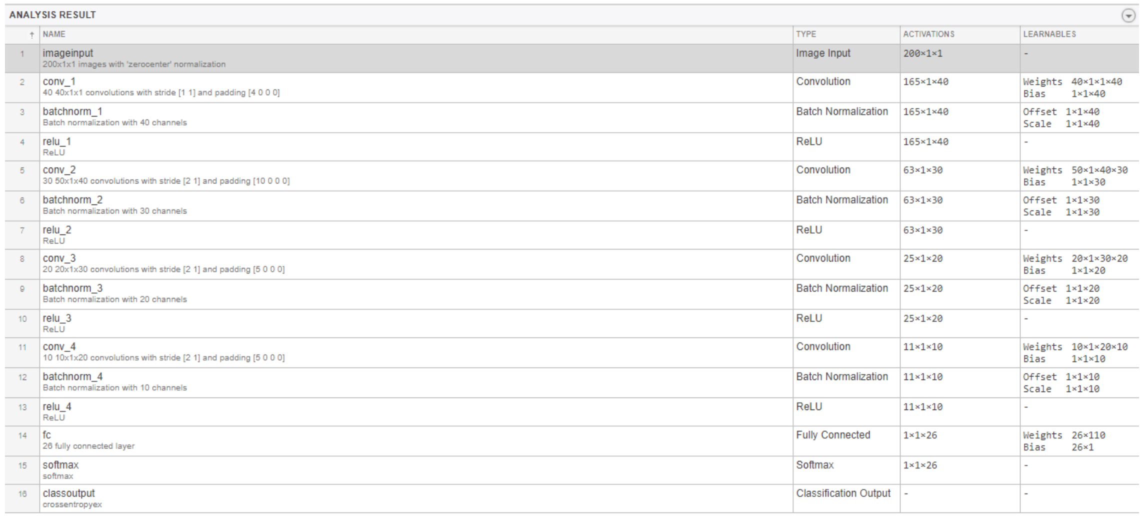
	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 40 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×40	Weights 200×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	206×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	206×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	84×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	84×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	35×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	35×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	16×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	16×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×160 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9009
Accuracy with max 1 bit wrong	0.9985
Execution time	6.1875s (for 13000 iterations)
Execution per iteration	$4.76 * 10^{-4}$s

Signal length with 4 samples per bit

The fiter for the first convolutional layer has been reduced to 40. The stride and padding for the first convolutional layer have also been reduced when compared to the 10 samples/bit version. This is in an attempt to create a network that is as similar as possible to the 10 samples/bit version.

Network Structure:



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 200x1x1 images with 'zerocenter' normalization	Image Input	200×1×1	-
2	conv_1 40 40x1x1 convolutions with stride [1 1] and padding [4 0 0 0]	Convolution	165×1×40	Weights 40×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	165×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	165×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	63×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	63×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	63×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	25×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	25×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	25×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	11×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	11×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	11×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×110 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.3451
Accuracy with max 1 bit wrong	0.7955
Execution time	5.2753s (for 13000 iterations)
Execution per iteration	$4.08 * 10^{-4}$s

C.6 Signal length comparison of differently sized CNNs

Signal information:

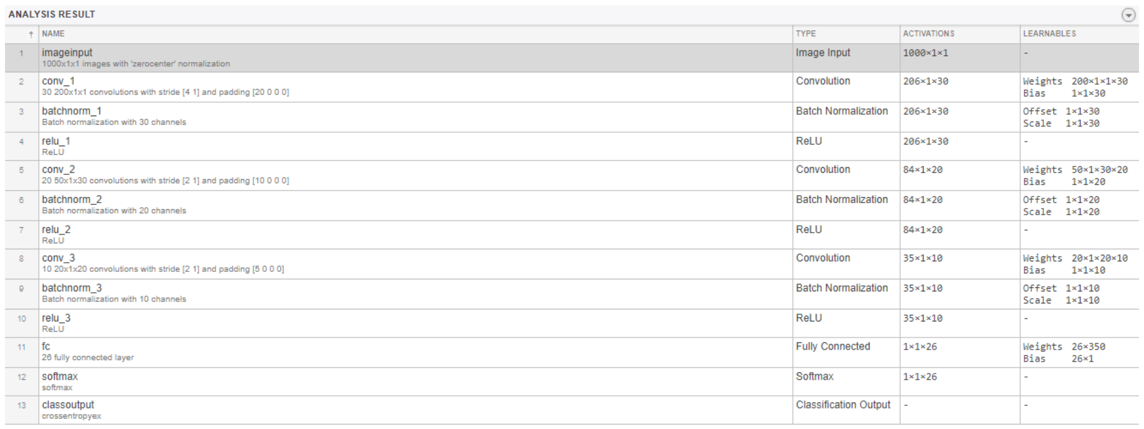
Modulation Scheme	PSK
Amplitude	50 peak
Noise Levels	0 SNR: approx 14.0 average amplitude , approx 50 peak
Input size	50 bit area
Data bits	0-25
Samples per bit	20
Periods per bit	1
Phase-scew	Randomized within timespace of 1 sample

Training options:

Solver	adam
Epochs	100
Iterations per epoch	101
Learning rate	Constant 0.001

CNN with 3 convolutional layers

Network Structure:



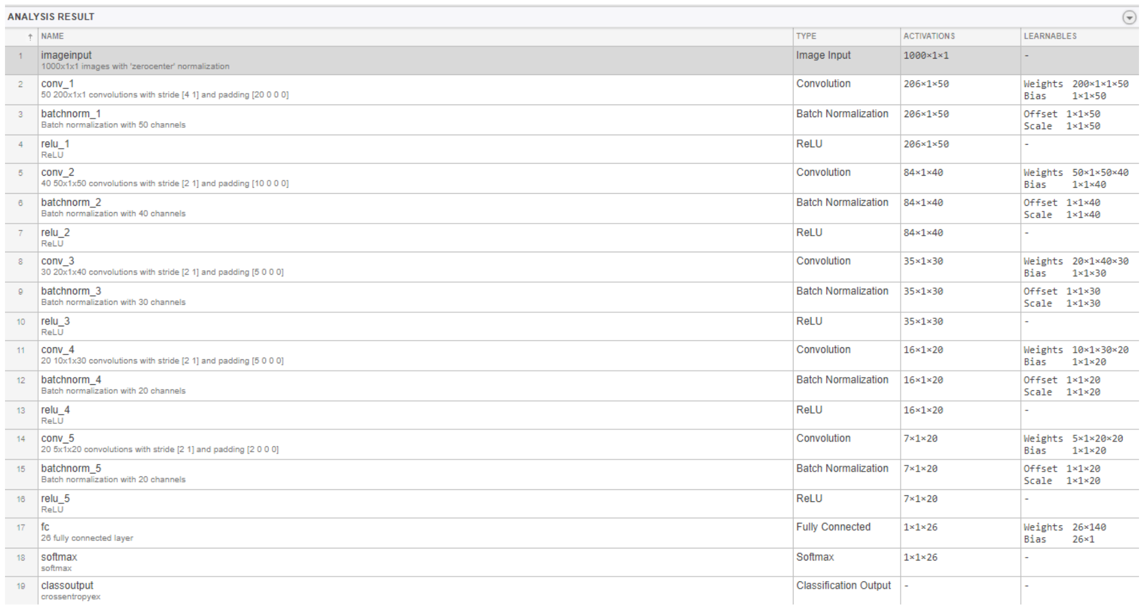
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 30 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×30	Weights 200×1×1×30 Bias 1×1×30
3	batchnorm_1 Batch normalization with 30 channels	Batch Normalization	206×1×30	Offset 1×1×30 Scale 1×1×30
4	relu_1 ReLU	ReLU	206×1×30	-
5	conv_2 20 50x1x30 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×20	Weights 50×1×30×20 Bias 1×1×20
6	batchnorm_2 Batch normalization with 20 channels	Batch Normalization	84×1×20	Offset 1×1×20 Scale 1×1×20
7	relu_2 ReLU	ReLU	84×1×20	-
8	conv_3 10 20x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×10	Weights 20×1×20×10 Bias 1×1×10
9	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	35×1×10	Offset 1×1×10 Scale 1×1×10
10	relu_3 ReLU	ReLU	35×1×10	-
11	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×350 Bias 26×1
12	softmax softmax	Softmax	1×1×26	-
13	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.6892
Accuracy with max 1 bit wrong	0.9727
Execution time	3.5636s (for 13000 iterations)
Execution per iteration	$2.74 * 10^{-4}$s

CNN with 5 convolutional layers

Network Structure:



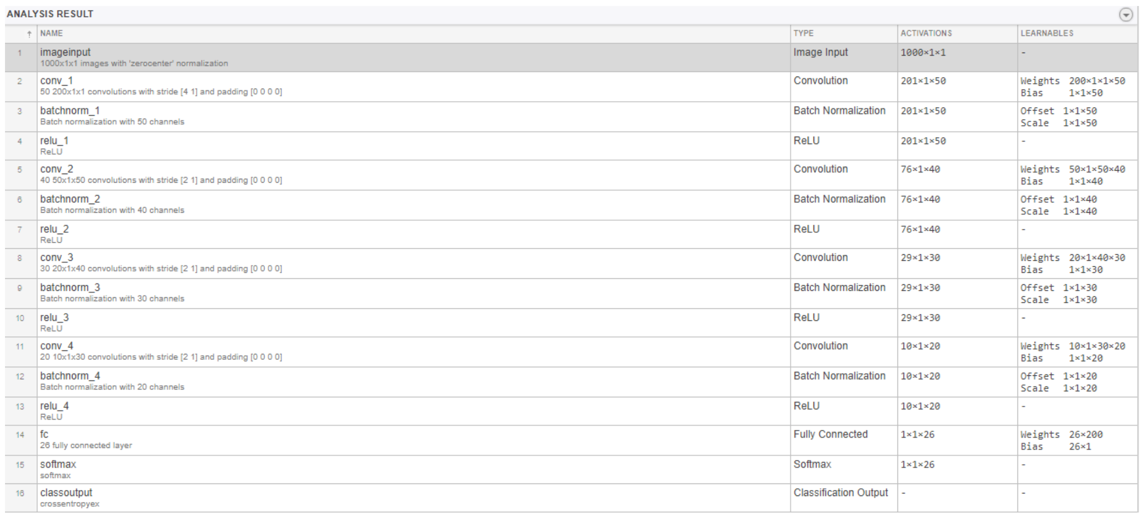
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [4 1] and padding [20 0 0 0]	Convolution	206×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	206×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	206×1×50	-
5	conv_2 40 50x1x50 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	84×1×40	Weights 50×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	84×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	84×1×40	-
8	conv_3 30 20x1x40 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	35×1×30	Weights 20×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	35×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	35×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	16×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	16×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	16×1×20	-
14	conv_5 20 5x1x20 convolutions with stride [2 1] and padding [2 0 0 0]	Convolution	7×1×20	Weights 5×1×20×20 Bias 1×1×20
15	batchnorm_5 Batch normalization with 20 channels	Batch Normalization	7×1×20	Offset 1×1×20 Scale 1×1×20
16	relu_5 ReLU	ReLU	7×1×20	-
17	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×140 Bias 26×1
18	softmax softmax	Softmax	1×1×26	-
19	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9045
Accuracy with max 1 bit wrong	0.9887
Execution time	9.8385s (for 13000 iterations)
Execution per iteration	$7.57 * 10^{-4}$s

CNN with different amount of padding

No padding



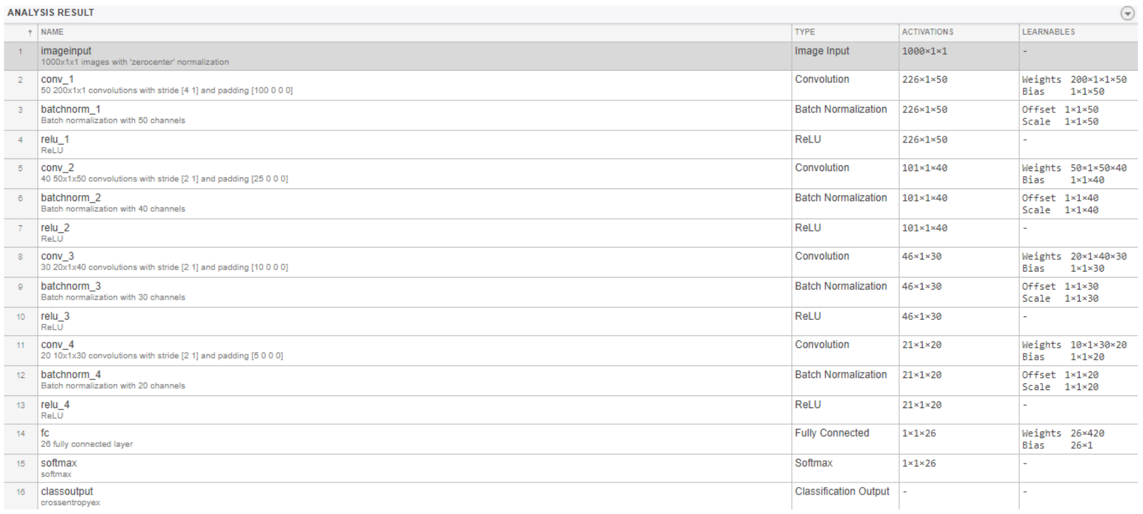
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [4 1] and padding [0 0 0 0]	Convolution	201×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	201×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	201×1×50	-
5	conv_2 40 50x1x50 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	76×1×40	Weights 50×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	76×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	76×1×40	-
8	conv_3 30 20x1x40 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	29×1×30	Weights 20×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	29×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	29×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [0 0 0 0]	Convolution	10×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	10×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	10×1×20	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×200 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9125
Accuracy with max 1 bit wrong	0.9961
Execution time	7.2252s (for 13000 iterations)
Execution per iteration	$5.56 * 10^{-4}$s

Large amounts of padding

The padding introduced in this iteration is equal to half the filter size at each convolutional layer.



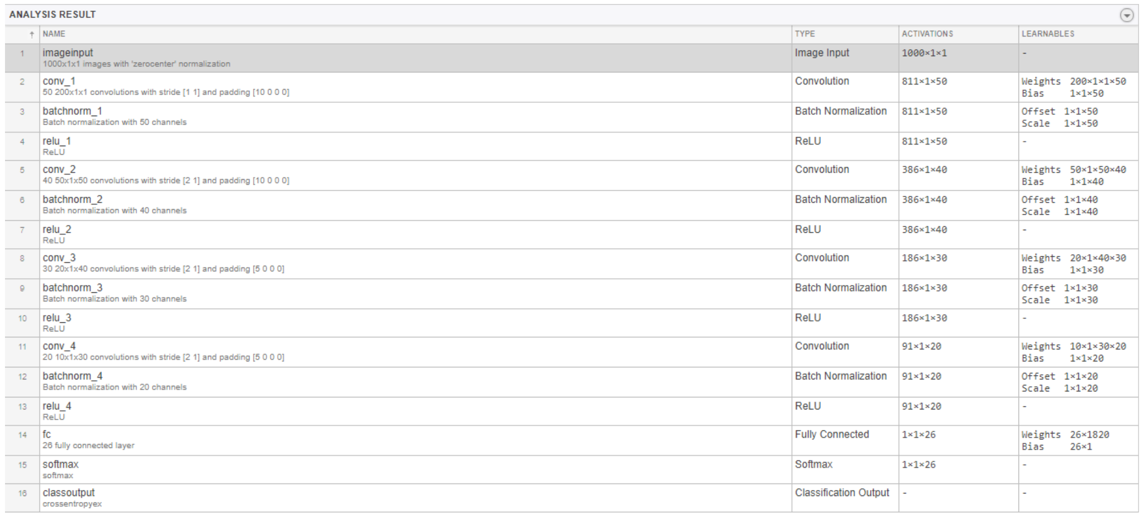
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [4 1] and padding [100 0 0 0]	Convolution	226×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	226×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	226×1×50	-
5	conv_2 40 50x1x50 convolutions with stride [2 1] and padding [25 0 0 0]	Convolution	101×1×40	Weights 50×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	101×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	101×1×40	-
8	conv_3 30 20x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	46×1×30	Weights 20×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	46×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	46×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	21×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	21×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	21×1×20	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×420 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.9142
Accuracy with max 1 bit wrong	0.9925
Execution time	8.6235s (for 13000 iterations)
Execution per iteration	$6.63 * 10^{-4}$s

CNN with different amounts of stride

CNN with stride of first convolutional layer minimized

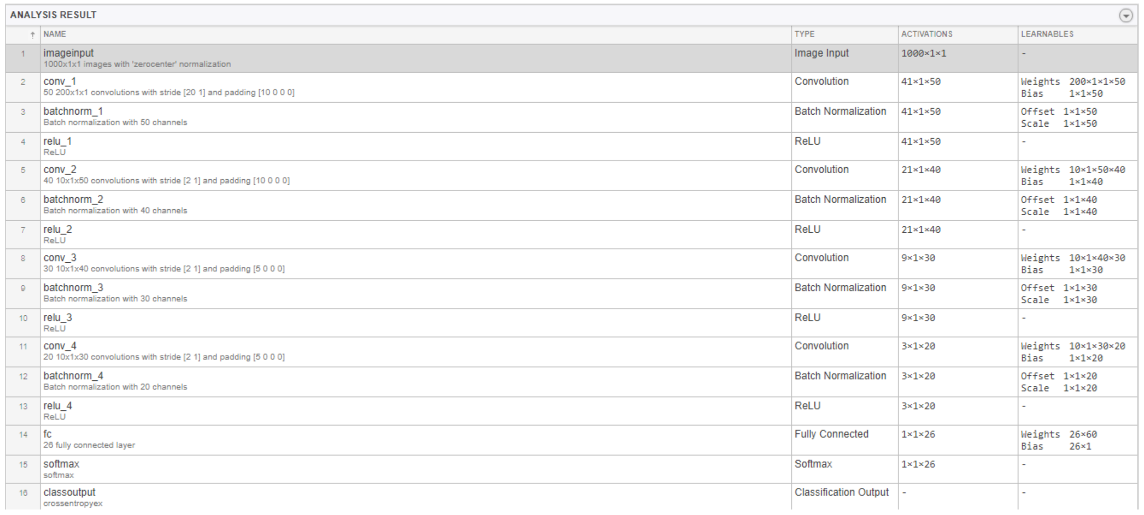


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [1 1] and padding [10 0 0 0]	Convolution	811×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	811×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	811×1×50	-
5	conv_2 40 50x1x50 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	386×1×40	Weights 50×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	386×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	386×1×40	-
8	conv_3 30 20x1x40 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	186×1×30	Weights 20×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	186×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	186×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	91×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	91×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	91×1×20	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×1820 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.5751
Accuracy with max 1 bit wrong	0.8755
Execution time	6.2462s (for 13000 iterations)
Execution per iteration	$4.81 * 10^{-4}$s

Stride increased

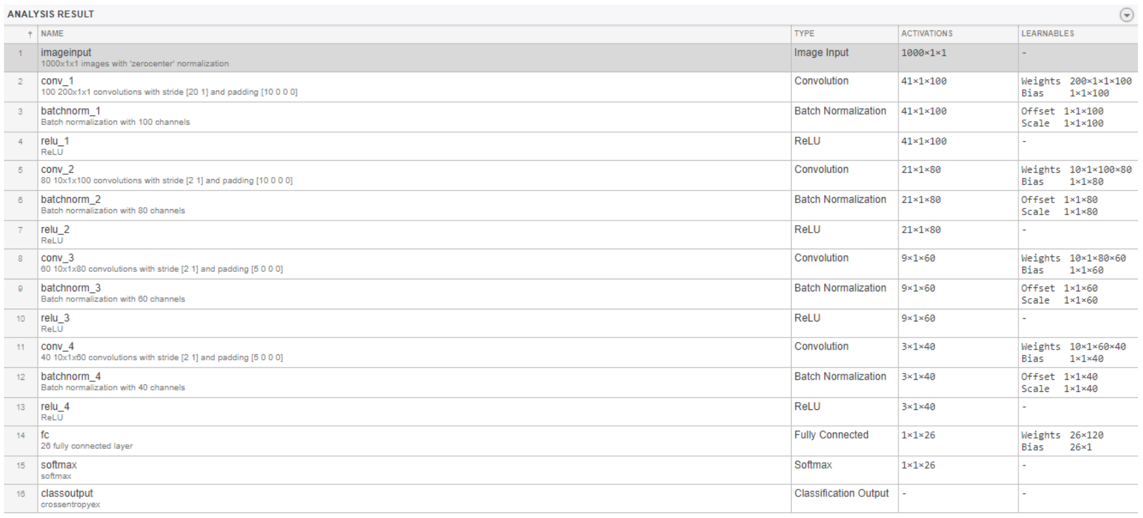


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 50 200x1x1 convolutions with stride [20 1] and padding [10 0 0 0]	Convolution	41×1×50	Weights 200×1×1×50 Bias 1×1×50
3	batchnorm_1 Batch normalization with 50 channels	Batch Normalization	41×1×50	Offset 1×1×50 Scale 1×1×50
4	relu_1 ReLU	ReLU	41×1×50	-
5	conv_2 40 10x1x50 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	21×1×40	Weights 10×1×50×40 Bias 1×1×40
6	batchnorm_2 Batch normalization with 40 channels	Batch Normalization	21×1×40	Offset 1×1×40 Scale 1×1×40
7	relu_2 ReLU	ReLU	21×1×40	-
8	conv_3 30 10x1x40 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	9×1×30	Weights 10×1×40×30 Bias 1×1×30
9	batchnorm_3 Batch normalization with 30 channels	Batch Normalization	9×1×30	Offset 1×1×30 Scale 1×1×30
10	relu_3 ReLU	ReLU	9×1×30	-
11	conv_4 20 10x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	3×1×20	Weights 10×1×30×20 Bias 1×1×20
12	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	3×1×20	Offset 1×1×20 Scale 1×1×20
13	relu_4 ReLU	ReLU	3×1×20	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×60 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.7018
Accuracy with max 1 bit wrong	0.9801
Execution time	1.6127s (for 13000 iterations)
Execution per iteration	$1.24 * 10^{-4}$s

Stride and number of filters increased



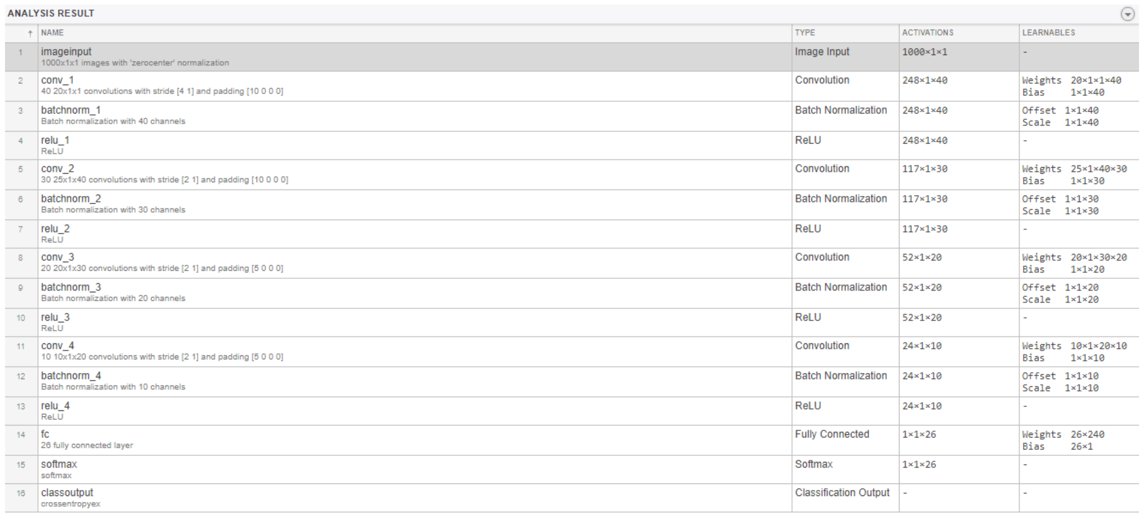
ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 100 200x1x1 convolutions with stride [20 1] and padding [10 0 0 0]	Convolution	41×1×100	Weights 200×1×1×100 Bias 1×1×100
3	batchnorm_1 Batch normalization with 100 channels	Batch Normalization	41×1×100	Offset 1×1×100 Scale 1×1×100
4	relu_1 ReLU	ReLU	41×1×100	-
5	conv_2 80 10x1x100 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	21×1×80	Weights 10×1×100×80 Bias 1×1×80
6	batchnorm_2 Batch normalization with 80 channels	Batch Normalization	21×1×80	Offset 1×1×80 Scale 1×1×80
7	relu_2 ReLU	ReLU	21×1×80	-
8	conv_3 60 10x1x80 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	9×1×60	Weights 10×1×80×60 Bias 1×1×60
9	batchnorm_3 Batch normalization with 60 channels	Batch Normalization	9×1×60	Offset 1×1×60 Scale 1×1×60
10	relu_3 ReLU	ReLU	9×1×60	-
11	conv_4 40 10x1x60 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	3×1×40	Weights 10×1×60×40 Bias 1×1×40
12	batchnorm_4 Batch normalization with 40 channels	Batch Normalization	3×1×40	Offset 1×1×40 Scale 1×1×40
13	relu_4 ReLU	ReLU	3×1×40	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×120 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.6583
Accuracy with max 1 bit wrong	0.9742
Execution time	2.2124s (for 13000 iterations)
Execution per iteration	$1.70 * 10^{-4}$s

CNN with different filter sizes

Filter size [20 1] for first convolutional layer, and 25 for second convolutional layer

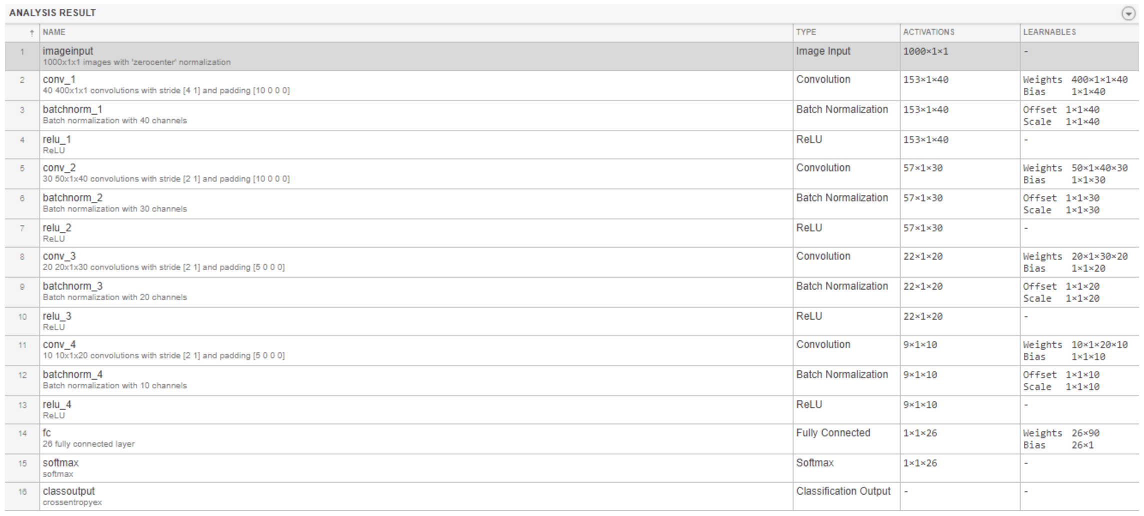


ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 40 20x1x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	248×1×40	Weights 20×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	248×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	248×1×40	-
5	conv_2 30 25x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	117×1×30	Weights 25×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	117×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	117×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	52×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	52×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	52×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	24×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	24×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	24×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×240 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.7241
Accuracy with max 1 bit wrong	0.9521
Execution time	5.0418s (for 13000 iterations)
Execution per iteration	$3.88 * 10^{-4}$s

Filter size [400 1] for first convolutional layer



ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES
1	imageinput 1000x1x1 images with 'zerocenter' normalization	Image Input	1000×1×1	-
2	conv_1 40 400x1x1 convolutions with stride [4 1] and padding [10 0 0 0]	Convolution	153×1×40	Weights 400×1×1×40 Bias 1×1×40
3	batchnorm_1 Batch normalization with 40 channels	Batch Normalization	153×1×40	Offset 1×1×40 Scale 1×1×40
4	relu_1 ReLU	ReLU	153×1×40	-
5	conv_2 30 50x1x40 convolutions with stride [2 1] and padding [10 0 0 0]	Convolution	57×1×30	Weights 50×1×40×30 Bias 1×1×30
6	batchnorm_2 Batch normalization with 30 channels	Batch Normalization	57×1×30	Offset 1×1×30 Scale 1×1×30
7	relu_2 ReLU	ReLU	57×1×30	-
8	conv_3 20 20x1x30 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	22×1×20	Weights 20×1×30×20 Bias 1×1×20
9	batchnorm_3 Batch normalization with 20 channels	Batch Normalization	22×1×20	Offset 1×1×20 Scale 1×1×20
10	relu_3 ReLU	ReLU	22×1×20	-
11	conv_4 10 10x1x20 convolutions with stride [2 1] and padding [5 0 0 0]	Convolution	9×1×10	Weights 10×1×20×10 Bias 1×1×10
12	batchnorm_4 Batch normalization with 10 channels	Batch Normalization	9×1×10	Offset 1×1×10 Scale 1×1×10
13	relu_4 ReLU	ReLU	9×1×10	-
14	fc 26 fully connected layer	Fully Connected	1×1×26	Weights 26×90 Bias 26×1
15	softmax softmax	Softmax	1×1×26	-
16	classoutput crossentropyex	Classification Output	-	-

Accuracy	0.8027
Accuracy with max 1 bit wrong	0.9890
Execution time	4.3073s (for 13000 iterations)
Execution per iteration	$3.31 * 10^{-4}$s