Acknowledgement

We would like to take this opportunity to express our sincere gratitude to our project guide, Dr. Kamal Captain and his continuous support, patience, motivation and constructive suggestions. The report is an outcome of his continuous guidance and support. We are extremely thankful to him for providing support and guidance, although he has a busy schedule managing the institute affairs. We would also like to thank our Head of Department, Dr. A.D. Darji for his continuous and outstanding support in completion of this task. We would also take this opportunity to thank all our lecturers who have directly or indirectly helped me by imparting the necessary knowledge of the field. Finally we would like to thank our classmate and friend Amrita Nair for her help at any time of day or night for which we are ever grateful.

Abstract

Project Title: Deep Learning for Automatic Modulation Classification

Deep neural networks have been pushing recent performance boundaries for a variety of machine learning tasks in fields such as computer vision, natural language processing, and speaker recognition. Recently researchers in the wireless communications field have started to apply deep neural networks to cognitive radio tasks with some success. In particular it has been shown that relatively simple convolutional neural networks outperform algorithms with decades of expert feature searches for radio modulation. This report provides an introduction to deep neural networks for the task of modulation recognition. The aim of this project is to find a computationally efficient algorithm that can be implemented practically to fulfill the vital requirement of modulation recognition for cognitive radio.

Contents

List of Figures

Chapter 1: Introduction

1.1 Introduction to Modulation Recognition

Our environment is inundated with communication signals travelling in space with different frequencies and different modulation schemes. The next evolution in wireless communications systems is the development of cognitive radio. A cognitive radio is a wireless system in which a receiver can intelligently detect which communication channels are in use and which are not, and intelligently move to communication channels which are vacant. This evolution will already make an already complex spectrum space much harder to govern, with a large amount of traffic flexibly using frequency bands.

Despite regulations by governments and regulating authorities, the signal space is a very noisy environment. A large number of applications require identification and monitoring of all these signals. These include both civilian and military applications. Civilian purposes include signal confirmation, interference identification and spectrum management, these authorities may wish to monitor transmissions in order to maintain control over these activities as well as detecting and monitoring non licensed transmitters. The applications for military purposes are electronic warfare applications, surveillance and threat analysis.

A central concept of Cognitive Radio is Dynamic Spectrum Access. The key sensing per- formed is that of providing awareness of nearby emitters to avoid radio interference and optimize spectrum allocation.

The radio spectrum is being used concurrently for a wide range of applications at all frequency ranges. Thus Dynamic Spectrum Access involves identifying and differentiating broadcast radio, local and wide area data and voice. A cognitive radio would require the ability to tell the state of the spectrum space to make a decision to shift bands.[4] [16]

1.2 Brief Overview

Modulation Recognition is the task of classifying the modulation type of a received radio signal as a step towards understanding what type of communications scheme and emitter is present.

Modulation recognition has traditionally been implemented using expert systems which have been designed specifically. These used to be operated manually by skilled operators, however technological advancements have made automatic modulation classification possible.

Automatic modulation recognition refers to a system that can automatically determine which modulation scheme is currently being used in the waveform being received.

Deep neural networks are seeing widespread implementation in domains such as video, speech and image processing. This success can be extended towards problems in communications theory where the existing mathematical models suffer from inaccuracies and sufficiently large datasets exist. Work in this field has begun using different networks to successfully identify modulation schemes. Modulation classification has been attempted by using a number of different Neural architectures.

The problem of modulation recognition, from the perspective of deep learning can be treated as an N class decision problem where we input a complex time series signal (with in- phase and quadrature components) and train a network on windowed sequences of the time series signal.

1.3 Motivation

Modulation classification is currently primarily used in military applications. The ability to detect incoming signals and their properties automatically is an important tool in the arsenal for modern militaries. The development of cognitive radio would greatly benefit from an efficient algorithm to automatically detect modulation schemes, as it would let a system flexibly and automatically decide which modulation scheme to utilize based on prevailing channel conditions.

In Dynamic Spectrum Access (DSA) one of the key sensing performed is that of providing awareness of nearby emitters to avoid radio interference and optimize spectrum allocation. This means identifying and differentiating broadcast radio, local and wide area data and voice radios, radar users, and other sources of potential radio interference in the vicinity which each have different behaviors and requirements.

Research in the area has shown that deep learning networks outperform older methods based upon expert designed systems. Additionally, these networks allow much greater flexibility, and bring in learning in variance which is essential for a system designed to perform analysis on data from our communications signal space. We can simply train a network again to introduce a new type of modulation, as compared to older methods. Utilizing deep learning networks makes this task more accessible, and moves it out of the field of pure communications research to that of Data Science where a large talent pool is available.

1.4 Modulation Recognition

1.4.1 Problem Analysis

This can be treated as an N-class decision problem where our input is a complex base-band time series representation of the received signal. That is, we sample in-phase and quadrature components of a radio signal at discrete time steps through an analog to digital converted with a carrier frequency roughly centered on the carrier of interest to obtain a $1 \times N$ complex valued vector. Classically, this is written as in equation 3.1 where $s(t)$ is a time series signal of either a continuous signal or a series of discrete bits modulated onto a sinusoid with either varying frequency, phase, amplitude, trajectory, or some permutation of multiple thereof. c is some path loss or constant gain term on the signal, and $n(t)$ is an additive Gaussian white noise process reflecting thermal noise.[4]

$$r(t) = s(t)c + n(t) \tag{1.1}$$

Analytically, this simplified expression is used widely in the development of expert features and decision statistics, but the real world relationship looks much more like that given in equation 2 in many systems.

$$r(t) = e^{jnL_0(t)} \int_{T_0}^{\tau=0} s(n_{Clk}(t-\tau))h(\tau) + n_{Add}(t) \tag{1.2}$$

This considers a number of real world effects which are non-trivial to model: modulation by a residual carrier random walk process, $n_{L_o}(t)$, resampling by a residual clock oscillator random walk, $n_{Clk}(t)$, convolution with a time varying rotating non-constant amplitude channel impulse response $h(t-\tau)$, and the addition of noise which may not be white, $n_{Add}(t)$. Each presents an unknown time varying source of error. Modeling expert features and decision metrics optimally analytically under each of these harsh realistic assumptions on propagation effects is non-trivial and often forces simplifying assumptions. In this paper we focus on empirical measurement of performance in harsh simulated propagation environments which include all of the above mentioned effects.[13]

1.4.2 Technical Approach

In a radio communication system, one class of receiver which is commonly considered is a "matched-filter" receiver. That is on the receive side of a communications link, expert designed filters matched with each transmitted symbol representation are convolved with the incoming time signal, and form peaks as the correct symbol slides over the correct symbol time in the received signal. By convolving, we average out the impulsive noise in the receiver in an attempt to optimize signal to noise. Typically, before this convolutional stage, symbol timing and carrier frequency is recovered using an expert envelope or moment based estimators derived analytically for a specific modulation and channel model. The intuition behind the use of a convolutional neural network in this application then is that they will learn to form matched filters for numerous temporal features, each of which will have some filter gain to operate at lower SNR, and which when taken together can form a robust basis for classification.[13]

1.4.3 Learning Invariance

Many of these recovery processes in radio communications systems can be thought of in terms of in-variance to linear mixing, rotation, time shifting, scaling, and convolution through random filters (with well characterized probabilistic envelopes and coherence times). These are analogous to similar learning in-variance which is heavily addressed in vision domain learning where matched filters for specific items or features in the image may undergo scaling, shifting, rotation, occlusion, lighting variation, and other forms of noise. We seek to leverage the shift-invariant properties of the convolutional neural network to be able to learn matched filters which may delineate symbol encoding features naively, without expert understanding or estimation of the underlying waveform.[2]

1.5 Organisation of report

- **Literature Survey**: Consists of a survey of research papers related to modulation classification.
- **Deep Learning**: This section explains various deep learning networks that will be used in this project.
- **Evaluation of Dataset & Implementation**: A brief description of the dataset used and details of the implemented models.
- **Results**: Final results in form of confusion matrix and SNR vs Accuracy line graph.
- **Conclusion and Future Work**: Conclusion of report and discussion of future work.

Chapter 2: Literature Survey

In the past, manual modulation recognition was implemented based on a number of measured parameters to provide a classification of different emitters. Generally, any automatic modulation classifier, based on the decision theoretic approach, comprises of three different stages:

- Pre-processing
- Key feature extraction
- Modulation Classification

In terms of preprocessing, for our specific problem, the main functions are signal isolation, and appropriate signal segmentation. [17]The advantage of using deep learning is that the computer will perform the essential step of Key feature extraction and development, and the preprocessing of data can be automated. This will lead to a robust and powerful system which is the need for a system like a cognitive radio, where the transceiver needs to intelligently make its own decisions. [5]

This section contains the survey of the task of modulation classification and various deep learning networks that have been used for the task of automatic modulation classification:

2.1 Study of Modulation Classification

2.1.1 Performance of Feature-Based Techniques for Automatic Digital Modulation Recognition and Classification

This study by Dhamyaa H. Al-Nuaimi, Ivan A. Hashim, Intan S. Zainal Abidin, Laith B. Salman, and Nor Ashidi Mat Isa provides a comprehensive survey of AMC techniques primarily based on FB approaches. While likelihood techniques provide satisfactory results, they are impractical due to their computational complexity. Furthermore, likelihood techniques have a hard time determining the best analytical solution for decision functions, particularly when there are a lot of unknowns.[14]

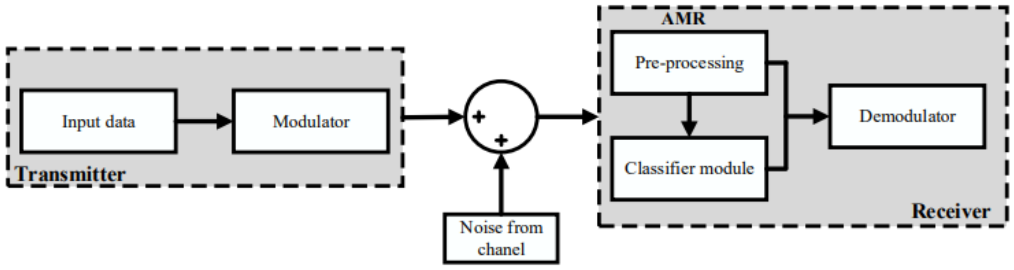


Figure 2.1: Communication system model that utilizes AMC[17]

When relevant features are extracted and then fed to the classifier for classification, this is referred to as the FB method. The authors looked at various features that are used to differentiate different modulation types and how they're being used and applied to particular modulation types. The computational complexity of instantaneous features is low, but they are susceptible to noise. Therefore, these features should be combined with other features to improve their performance for identifying the signals.[4]

In regard to feature extraction, the paper goes through the classifiers used for MT in depth. The classifiers presented include DTs, ANNs, SVMs, k-NNs, clustering algorithms, and hybrid algorithms based on these classifiers. DTs and k-NNs are considered simple classifiers, while ANNs and SVMs have better classification efficiency and are more noise-resistant, but they are more computationally complex.[8]

2.2 Study of Research Papers on Deep Learning Networks for Modulation Recognition

2.2.1 Convolutional Radio Modulation Recognition Networks

This paper[13] by Timothy J. O'Shea , Johnathan Corgan , and T. Charles Clancy introduces the concept of using Deep Networks for the task of automatic modulation classification. The authors of this paper are also the creators of the standard dataset used as a benchmark for this task. They set out to explain the dataset creation process and dataset parameters in this paper, before exploring the use of neural networks for modulation classification.

The authors demonstrate a number of feature learning methods in the paper, but focus primarily on the use of Convolutional Networks for Modulation Classification.

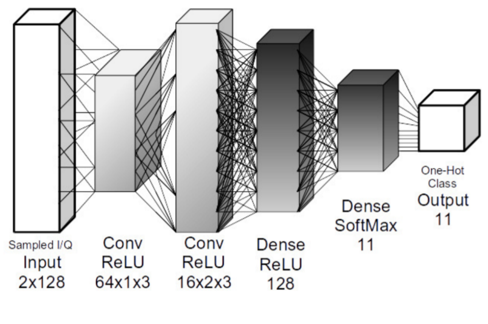


Figure 2.2: Block Diagram of Convolutional Neural Network[13]

The above diagram showcases the Convolutional Neural Network utilised in the paper. The network is a 4-layer network utilizing two convolutional layers and two dense fully connected layers (CNN and CNN2). Layers use rectified linear (ReLU) activation functions except for a Softmax activation on the one-hot output layer. The authors use this network depth as it is roughly equivalent to networks which work well on similar simple datasets in the vision domain such as MNIST.

The authors also implement two more deep networks: A DNN containing 4 dense layers of size 512, 256, 128, and n-classes neurons, and a second CNN which is identical to the first CNN but larger, containing 256 and 80 filters in layers 1 and 2, and 256 neurons in layer 3.

The authors also implement existing expert feature based classifiers, and provide us with a comparison to demonstrate the better performance of CNNs for this task.

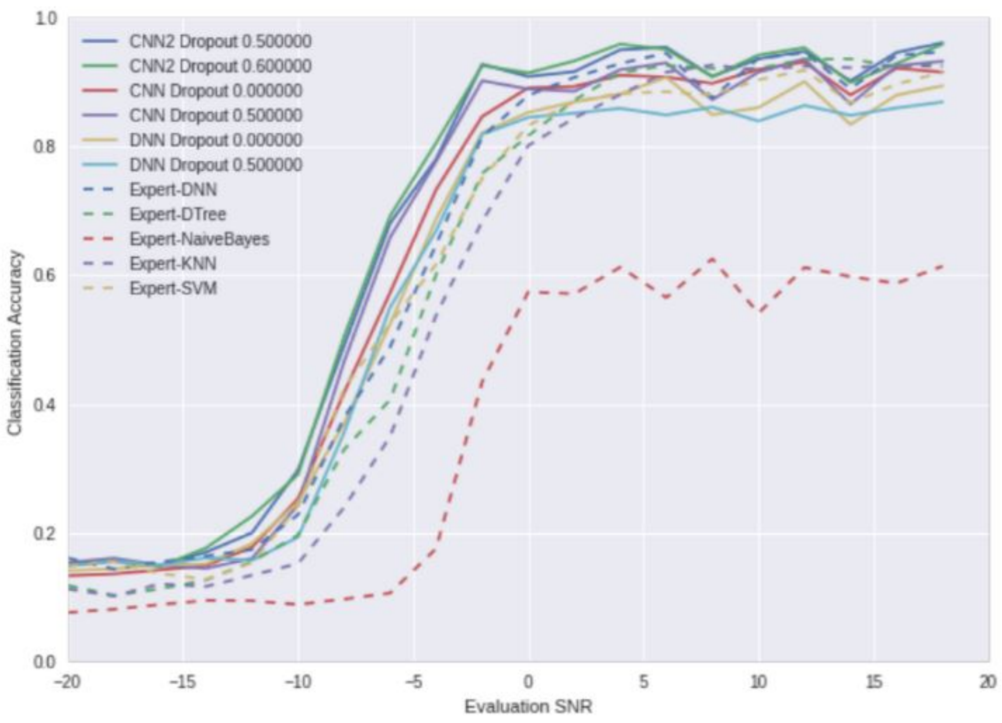


Figure 2.3: SNR vs Accuracy comparison for Convolutional Networks and classifiers using expert learned features [13]

2.2.2 Deep Architectures for Modulation Recognition

This paper[11] by Timothy J O' Shea and Nathan E West was published after the original paper by Timothy O' Shea. In this paper, they set out to explore novel deep architectures for the task of modulation classification. The authors explore methods where they will be able to increase the depth of the network.

To this end, they explore an Inception architecture used in Googlenet, which was demonstrated to be a successful approach to increasing network depth and ability to generalize to features of differing scales while still managing complexity. Another approach to increasing depth uses an architecture that forwards information untouched across layers: Residual networks. A Residual network adds one layer's output to the output of the layer two layers deeper. [2] Finally, the authors also implemented a model commonly used in voice processing that operates on raw time-domain waveforms rather than expert voice features such as log-mel cepstrums. The architecture uses two convolutional layers followed by two recurrent layers made up of Long Short-Term Memory (LSTM) cells.

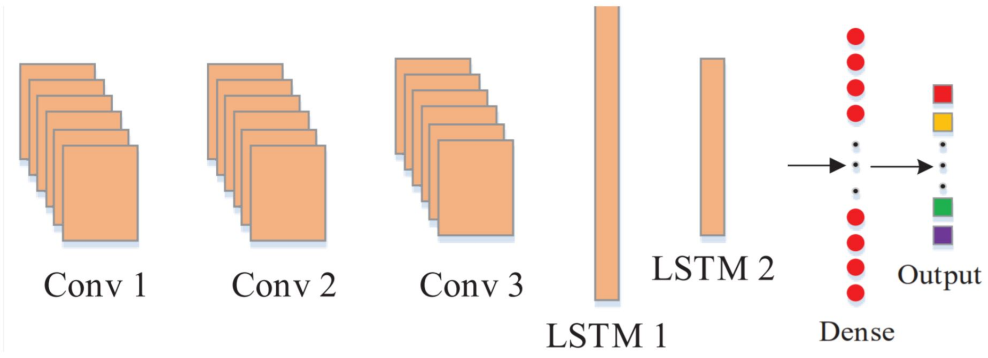


Figure 2.4: CNN-LSTM based dual-stream architecture[5]

The authors implemented all these different deep convolutional architectures, in their paper.

A key finding of this paper included experimentation with network depth using the ResNet, they found that results from ordinary CNN depth suggests we are not limited by network depth for radio learning tasks as much as we are limited by features purely CNN architectures can learn. The authors discovered that Inception modules also do not improve radio modulation classification. The main takeaway from this paper is that performance in deep neural networks in the radio domain does not seem to be limited by network depth the same way the image, natural language processing, and acoustic domains are.

2.2.3 Fast Deep Learning for Automatic Modulation Classification

In this paper[15], by Sharan Ramjee , Shengtai Ju, Diyu Yang, Xiaoyu Liu, Aly El Gamal,and Yonina C. Eldar the authors designed their own CNN, DenseNet, and CLDNN architectures for the modulation recognition task, and derived optimized versions of the ResNet architecture and the LSTM architecture of [7] , by tuning the number of residual stacks for ResNet and the hyperparameters for LSTM.

In this paper the amplitude and phase information of the time domain modulated signal are generated with some preprocessing. This Amplitude and Phase information fed to all cells of the LSTM model as a two-dimensional vector, at each time step for classification. The amplitude vector is L2 normalized and the phase, which is in radians is normalized between -1 and 1. The first two layers are comprised of 128 LSTM cells each. The final output from the second LSTM layer, a vector of dimension 128, after all time steps, is fed to the last dense layer of the model. The final layer is a dense softmax layer that maps the classified features to one of the 11 output classes representing the modulation.[8]

The intuition to use an LSTM model for classification is based on the fact that different modulation schemes exhibit different amplitude and phase characteristics, and the model can learn these temporal dependencies effectively. Even though fading and other real world effects may slightly hamper the characteristics of the signal, we expect the model to classify signals efficiently by learning good fading resistant representations. Since the proposed model can work on variable length input time domain samples, we expect the model to learn useful symbol rate independent representations for classification. In

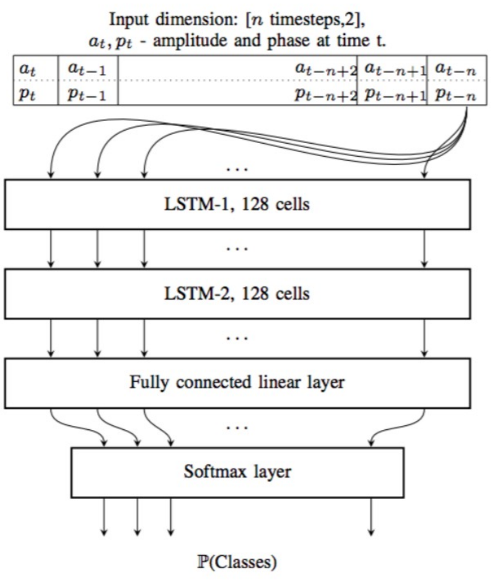

Input dimension: [n timesteps,2],
a_t, p_t - amplitude and phase at time t.
a_t a_{t-1} a_{t-n+2} a_{t-n+1} a_{t-n}
p_t p_{t-1} p_{t-n+2} p_{t-n+1} p_{t-n}
LSTM-1, 128 cells
LSTM-2, 128 cells
Fully connected linear layer
Softmax layer
$\mathbb{P}$(Classes)

Figure 2.5: LSTM Architecture[15]

addition, the importance of the number of LSTM cells and layer depth is further investigated by varying these parameters.[14]

2.2.4 Automatic Modulation Classification Using CNN-LSTM Based Dual-Stream Structure

This paper by Zufan Zhang, Hao Luo, Chun Wang, Chenquan Gan and Yong Xiang puts forward a novel architecture for modulation classification. The authors have developed a dual stream CNN-LSTM network, to combine the benefits of effective characteristics from each signal representation.

This facilitates the exploration of the spatial and temporal correlation of signals. The main contributions of their work are:

- Dual stream based structure, where one stream is to extract the local raw temporal features from raw signals, and the other stream is to learn the knowledge from amplitude and phase information.

- To learn spatial and temporal information from each signal representation, each stream consists of CNN and LSTM (denoted as CNN-LSTM), which combines the excellent performance of CNN

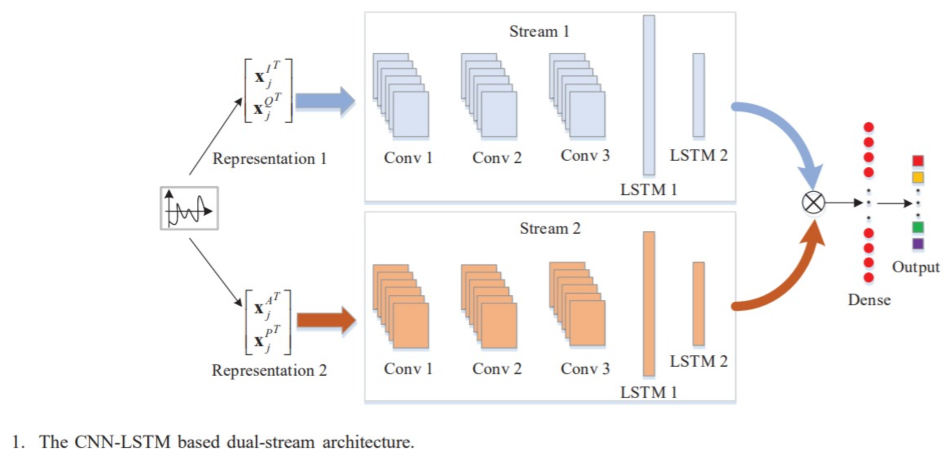


Figure 2.6: CNN-LSTM based dual-stream architecture[1]

in spatial feature extraction and the superior capacity of LSTM in processing time series data.

- Through an effective operation, the features trained from two streams interact in pairs, enriching the diversity of properties, and thereby further enhancing the performance.

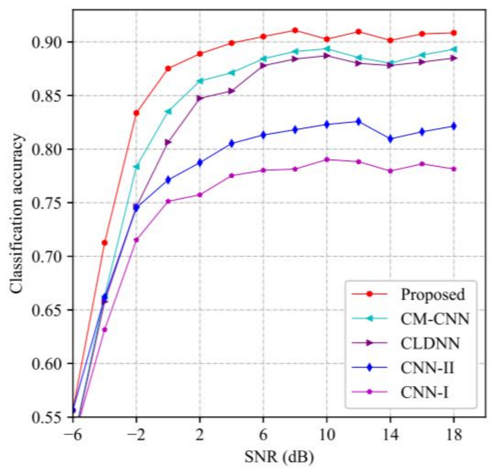


Figure 2.7: Comparision of Dual Stream CNN LSTM with other Deep Learning architectures[6]

Chapter 3: Deep Learning

Deep learning is an artificial intelligence (AI) function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence that has networks capable of learning unsupervised from data that is unstructured or unlabeled. [5]

3.1 Working principle

Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks.

The term "deep" usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction.In this chapter we will explore different kinds of classifiers used in Deep Learning in some detail. [12]

3.2 Convolutional Neural Networks

One of the most popular types of deep neural networks is known as convolutional neural networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

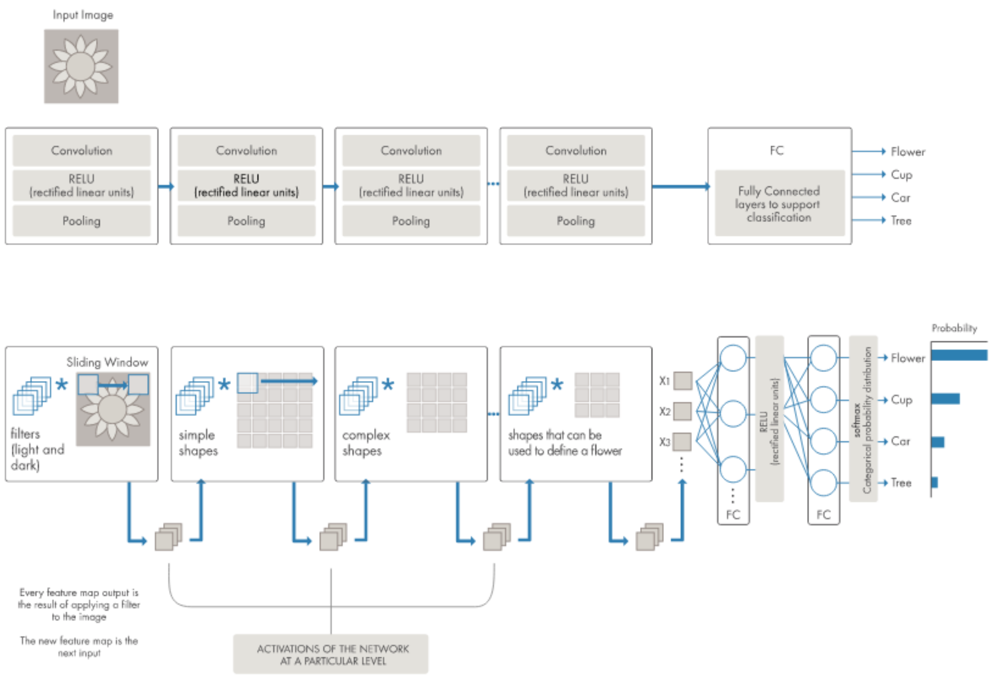

Input Image
Convolution
RELU (rectified linear units)
Pooling
FC
Fully Connected layers to support classification
Flower
Cup
Car
Tree
Sliding Window
filters (light and dark)
simple shapes
complex shapes
shapes that can be used to define a flower
X1
X2
X3
RELU (rectified linear units)
softmax Categorical probability distribution
Probability
Every feature map output is the result of applying a filter to the image
The new feature map is the next input
ACTIVATIONS OF THE NETWORK AT A PARTICULAR LEVEL

Figure 3.1: Neural networks with multiple convolutional layer[10]

CNN is a mathematical construct that is typically composed of three types of layers (or building blocks): convolution, pooling, and fully connected layers. As seen in figure 3.1 the first two convolution and pooling layers perform feature extraction, whereas the third a fully connected layer maps the extracted features into final output such as classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional (2D) grid, i.e., an array of numbers, and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. [20]

The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation and gradient descent, among others[16]

CNNs eliminate the need for manual feature extraction, so you do not need to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pretrained; they are learned while the network trains on a collection of images.[6] This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.[11]

CNNs learn to detect different features of an image using tens or hundreds of hidden layers. Every hidden layer increases the complexity of the learned image features. [6]

3.3 ResNet

According to the universal approximation theorem, given enough capacity, we know that a feedforward network with a single layer is sufficient to represent any function. However, the layer might be massive and the network is prone to overfitting the data. Therefore, there is a common trend in the research community that our network architecture needs to go deeper.

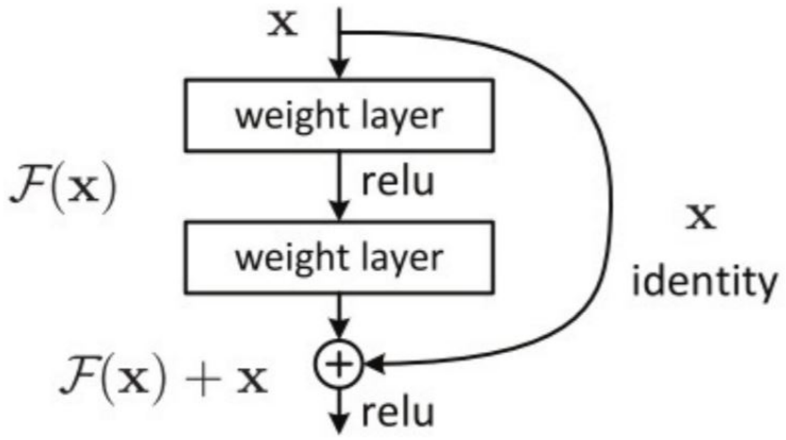

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3.2: ResNet Architecture[13]

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitively small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly. The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers, as shown in the figure.[5]

3.4 Long Short Term Memory(LSTM)

LSTM Networks are a modification of RNN that can learn long-term dependencies. They overcome the problem of vanishing gradients of Recurrent Neural Networks. Recurrent Neural Networks have a chain of repeating modules of neural network. LSTMs also possess this chain structure but the repeating modules are different from that of an RNN. In an LSTM module, three gates are present –input gate, output gate and forget gate. The cell state is the key to LSTMs.[18] It is the horizontal line at the top of the above diagram. It runs down the entire chain and information can flow easily along it without any change.[21]

The LSTM can remove or add information to the cell state using gates, which can optionally allow information to pass through them. They comprise of a sigmoid and pointwise multiplication operation. The sigmoid layer outputs a number between zero and one and describes how much of each input can be allowed to pass.[6]

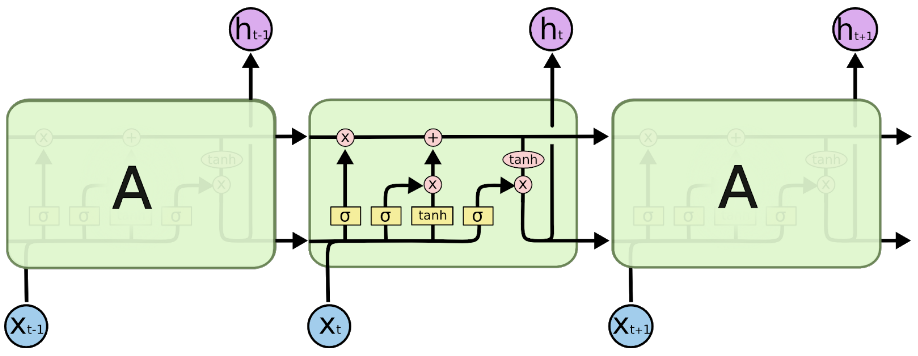
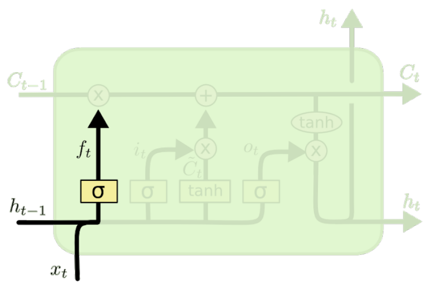


Figure 3.3: Modules in Long Short-Term Memory network [18]

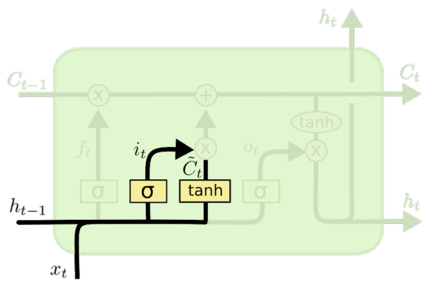
The first step is to decide what information is to be forgotten, which is decided by the forget gate.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \;+\; b_f\right)$$

Figure 3.4: Forget Gate of LSTM [18]

For handwriting recognition, the cell state might store the stroke width of a particular sample of handwriting. But for a new sample, that information needs to be forgotten and the stroke width of that sample is to be stored.Next, which information is to be stored will be decided by the input gate. In the example above, it would characterize the stroke width of the next sample.

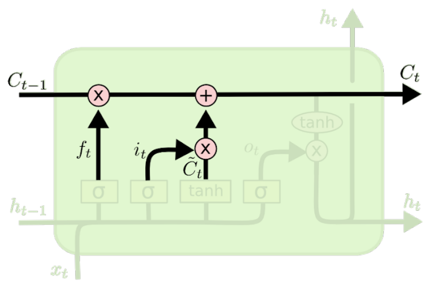


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \;+\; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

Figure 3.5: Input gate of LSTM [18]

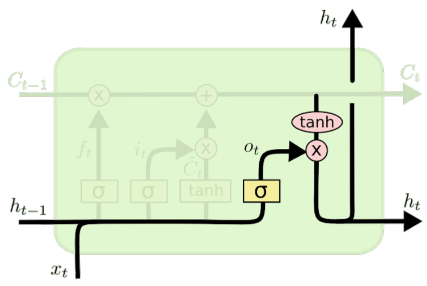
To update the old cell state Ct-1into Ct, it is multiplied with the old state to forget it. Then the new state it* Ct is added.

Finally, the output is to be decided which will be based on a filtered version of the cell state.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 3.6: Updating cell state in LSTM [18]



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 3.7: Output Gate in LSTM [18]

3.5 CNN-LSTM

CNN-LSTMs or CLDNNs are an approach for voice processing that operate on raw time-domain waveforms rather than expert voice features such as log-mel cepstrums.

Both Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) have shown improvements over Deep Neural Networks (DNNs) across a wide variety of speech recognition tasks. CNNs, LSTMs and DNNs are complementary in their modeling capabilities, as CNNs are good at reducing frequency variations, LSTMs are good at temporal modeling, and DNNs are appropriate for mapping features to a more separable space.[22] In our project, the CNN and the LSTM layers are entirely combined to collect effective characteristics from each signal representation, which facilitates the exploration of the spatial and temporal correlation of signals.

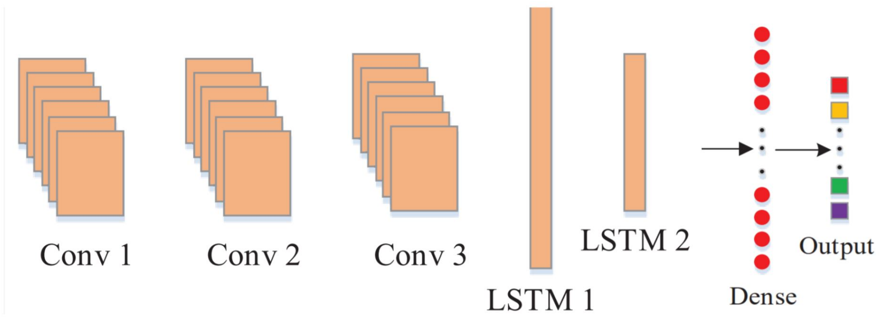


Figure 3.8: General architecture of CNN-LSTM [18]

3.6 Attention Mechanism

In broad terms, Attention is one component of a network's architecture, and is in charge of managing and quantifying the interdependence:

- Between the input and output elements (General Attention)
- Within the input elements (Self-Attention)

It was initially designed in the context of Neural Machine Translation using Seq2Seq Models. In the traditional Seq2Seq model[17], we discard all the intermediate states of the encoder and use only its final states (vector) to initialize the decoder. This technique works well for smaller sequences, however as the length of the sequence increases, a single vector becomes a bottleneck and it gets very difficult to summarize long sequences into a single vector. This observation was made empirically as it was noted that the performance of the system decreases drastically as the size of the sequence increases.[6]

The central idea behind Attention is not to throw away those intermediate encoder states but to utilize all the states in order to construct the context vectors required by the decoder to generate the output sequence.[19]

As human beings,[16] we are quickly able to understand these mappings between different parts of the input sequence and corresponding parts of the output sequence. However it's not that straight forward for artificial neural networks to automatically detect these mappings.The attention mechanism is developed to "learn" these mappings through Gradient Descent and Back-propagation

3.7 Inception Networks

Typical architecture of CNN comprises of convolution layer followed by sub-sampling (i.e. pooling layer) layer. The convolutional layers segregate the features of the image. Sampled features are classified by the fully connected layer followed by the output layer. In order to get a better accuracy network, the general approach is to increase depth of network which causes issues of vanishing gradients and demands more computational power. To deal with these problems inception layers were introduced. Using multiple windows to form an inception layer also aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously. [7]

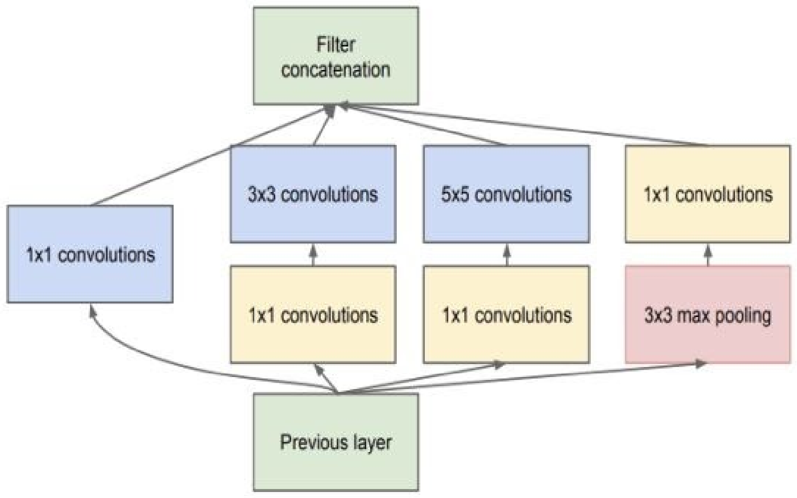


Figure 3.9: General architecture of Inception Network [18]

The advantages of inception networks over vanilla convolutional networks are :

- Avoid representational bottlenecking at the earlier layer to prevent information loss from the input images.
- Higher dimensional representation can easily be operated regionally within the network, also increasing receptive field will provide disentangled features.
- Spatial aggregation can be done by lower dimensional filter if strong correlation is present between adjacent activation.
- The optimal improvement with constant computational cost can be reached if width and depth are increased parallelly.

3.8 Accuracy of Deep Learning

Deep learning achieves recognition accuracy at higher levels than ever before. While deep learning was first theorized in the 1980s, there are two main reasons it has only recently become useful[12]:

- Deep learning requires large amounts of labeled data. For example, driverless car development requires millions of images and thousands of hours of video.
- Deep learning requires substantial computing power. High-performance GPUs have a parallel architecture that is efficient for deep learning. When combined with clusters or cloud computing, this enables development teams to reduce training time for a deep learning network from weeks to hours or less.[5]

Chapter 4: Evaluation of Dataset & Implementation

4.1 Evaluation of Dataset

While simulation and the use of synthetic data sets for learning are frowned upon in machine learning, radio communications present a special case. Training with real data is important and valuable but the results of the simulation are still meaningful. Radio communications signals are in reality synthetically generated, and we do so deterministically in a way identical to a real system, introducing modulation, pulse shaping, carried data, and other well characterized transmit parameters identical to a real world signal. We modulate real voice and text data sets onto the signal. In the case of digital modulation the data is whitened using a block randomizer to ensure bits are equiprobable.

Radio channel effects are relatively well characterized. The creators of the dataset employ robust models for time varying multi-path fading of the channel impulse response, random walk drifting of carrier frequency oscillator and sample time clocks, and additive Gaussian white noise. They pass the synthetic signal sets through harsh channel models which introduce unknown scale, translation, dilation, and impulsive noise onto the signal. The generation of this dataset is modelled in GNU Radio using the GNU Radio channel model blocks and each time series signal is sliced up into a test and training set using a 128 samples rectangular windowing process. The total dataset is roughly 500 MBytes stored as a python pickle file with complex 32 bit floating point samples.

4.1.1 Dataset Availability

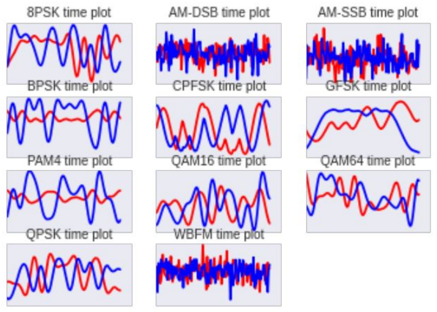
This data is of great use in the field and may serve as a benchmark for this domain. There are a number of papers using this dataset. This dataset is available in pickled python format at deepsig.ai, consisting of time-windowed examples and corresponding modulation class and SNR labels. The dataset is meant to grow in scope of modulations addressed and the channel realism as interest in this area grows.[3]

4.1.2 Dataset Parameters

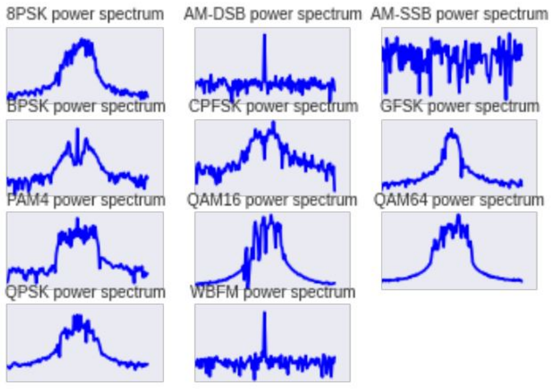
The dataset consists of 11 modulations: 8 digital and 3 analog modulation, all widely used in wireless communications systems all around us. These consist of BPSK, QPSK, 8PSK, 16QAM, 64QAM, BFSK, CPFSK, and PAM4 for digital modulations, and WB-FM, AM-SSB, and AM-DSB for analog modulations. Data is modulated at a rate of roughly 8 samples per symbol with a normalized average transmit power of 0dB.[11]

4.1.3 Dataset Visualization

Inspecting a single example from each class of modulation in the time (figure 4.1(a)) and frequency domain (figure 4.1(b)), we see a number of similarities and differences between modulations visually, but due to pulse shaping, distortion and other channel effects they are not all readily discernible by a human expert visually. In the frequency domain, each of the signals follows a similar band-limited power envelope by design whose shape provides some clues as to the modulation, but against poses a difficult noisy task for a human expert to judge visually.



(a) Time Domain of High-SNR Example Classes



(b) Power Spectrum of HighSNR Example Classes

Figure 4.1: Visualising Signals [13]

4.1.4 Modulated Information

In radio communications, signals are typically comprised of a number of modulated data bits on well defined and understood basis functions into discrete modes formed by these bases. Complex baseband representation of a signal decomposes a radio voltage level time-series into its projections onto the sine and cosine functions at a carrier frequency. By manipulating the frequency, amplitude, phase, or sum thereof data bits are then modulated into this space through discrete and separable modes for each distinct symbol period in time in the case of digital, or continuous location in the case of analog modulation. For the case of QPSK this phase-mapping is shown in 4.1.[3]

$$s(t_i) = e^{j2\pi f_c t + \pi \frac{2c_i+1}{4}}, c_i \epsilon 0, 1, 2, 3 \tag{4.1}$$

Pulse shaping filters such as root-raised cosine are then typically applied to band-limit the signal in frequency and remove sharp wide-band transients between these distinct modes, resulting in mixing of adjacent symbols' bases at the transmitter in a deterministic and invertible. In our simulated data set we use a root-raised cosine pulse shaping filter with an excess bandwidth of 0.35 for each digital signal.

4.1.5 Effects on the Modulated Signal

Channel effects in contrast are not deterministic and not completely invertible in a communications system. Real systems experience a number of effects on the transmitted signal, which make recovery and representation thereof challenging.[10] Thermal noise results in relatively flat white Gaussian noise at the receiver which forms a noise floor or sensitivity level and signal to noise ratio. Oscillator drift due to temperature and other semiconductor physics differing at the transmitter and receiver result in symbol timing offset, sample rate offset, carrier frequency offset and phase difference. These effects result in temporal shifting, scaling, linear mixing/rotating between channels, and spinning of the received signal based on unknown time varying processes. Last, real channels undergo random filtering based on the arriving modes of the transmitted signal at the receiver with varying amplitude, phase, Doppler, and delay. This is a phenomenon commonly known as multi-path fading or frequency selective fading, which occurs in any environment where signals may reflect off buildings, vehicles, or any form of reflector in the environment. This is typically removed at the receiver by the estimation of the instantaneous value of the time varying channel response and deconvolution of it from the received signal.[9]

4.1.6 Generating a dataset

Since this is a well characterized dataset, we have available a collection of modulations which are used widely in practice and operate on both discrete binary alphabets (digital modulations), and continuous alphabets (analog modulations). Known data is modulated over each modem and exposed to the channel effects described above using GNU Radio. The data is then segmented into millions of samples into a dataset consisting of numerous short-time windows in a fashion similar to how a continuous acoustic voice signal is typically windowed for voice recognition tasks. Steps of 128 samples with a shift of 64 samples are extracted to form the dataset. After segmentation, examples are roughly 128 µsec each assuming a sample rate of roughly 1 MSamp/sec. Each contains between 8 and 16 symbols with random time offset, scaling, rotation, phase, channel response, and noise. These examples represent information about the modulated data bits, information about how they were modulated, information about the channel effects the signal passed through during propagation, and information about the state of the transmitted and receiver device states and contained random processes. We focus specifically on recovering the information about how the signal was modulated and thus the dataset is labelled according to a discrete set of 11 class labels corresponding to the modulation scheme.

4.2 Backend and libraries

The model was implemented on **Google Colaboratory** using both CPU and GPU compute power separately. The dataset was downloaded, and then uploaded to Google Drive. Where it was mounted and depickled to access the data. The dataset available has been pickled by the authors. The dataset is depickled by importing the **pickle** library. To depickle the dataset, the encoding parameter had to be set to "latin1". **numpy** was imported to support mathematical operations on matrices. This model was implemented using **keras** library of python with **tensorflow** as backend We used **matplotlib** to plot the confusion matrix and the SNR vs accuracy table, matplot lib was also utilized to graph the data points. This helped visualize the data and get an insight.

4.3 Dataset and Data processing

A synthetic dataset, generated with GNU Radio, consisting of 11 modulations (8 digital and 3 analog) at varying signal-to-noise ratios provided by deepsig.ai RML2016.10a.pkl was used for this model implementation.This dataset had 220000 signals with different SNRs ranging from -18dB to 18dB with a step value of 2dB.

This represents a cleaner and more normalized version of the dataset. The file is formatted as a "pickle" file which was opened with the help of pickle library of python 3.6.The depickled data was in dictionary format and was changed to numpy array using np-array with a shape of (220000, 2, 128). This data was further split into two equal halves under training and testing data.

4.4 Preprocessing

In signal preprocessing, two simple data conversions were utilized. The dataset provided already has information in I/Q format.

We processed the raw complex data inspired by other papers to generate signal amplitude and phase information . The A/P vector consists of[**h**] two real valued vectors: x_j^A, x_j^P :

$$\mathbf{x}_j^{A/P} = \begin{bmatrix} \mathbf{x}_j^{A^T} \\ \\ \mathbf{x}_j^{P^T} \end{bmatrix},$$

We obtain x_j^A, x_j^P by the following elementary operations using numpy:

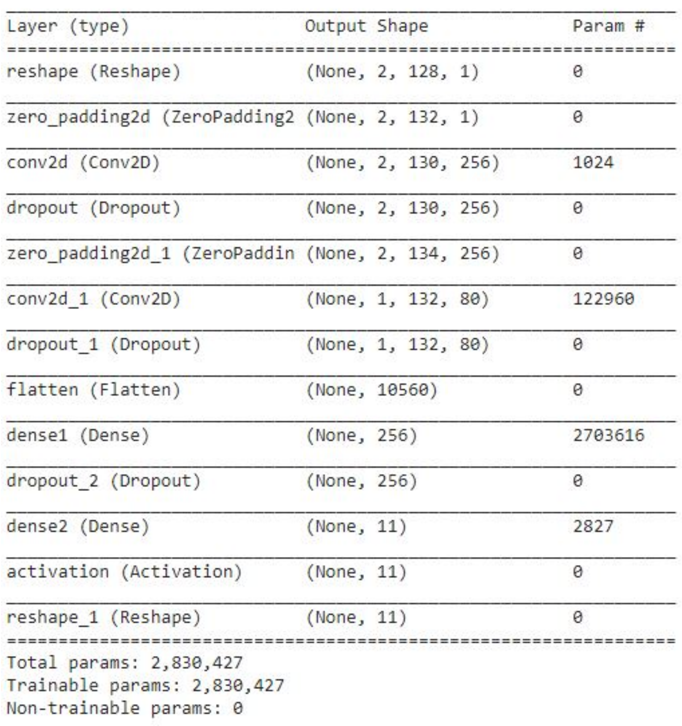
$$x_j^A[n] = \sqrt{\left(r_j^I[n]\right)^2 + \left(r_j^Q[n]\right)^2},$$

$$x_j^P[n] = \arctan\left(\frac{r_j^Q[n]}{r_j^I[n]}\right),$$

4.5 Convolutional Neural Networks

4.5.1 Sequential Model

Figure 4.2 shows the parameters for the implemented Convolutional Neural Network Model.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 2, 128, 1)         0
_________________________________________________________________
zero_padding2d (ZeroPadding2 (None, 2, 132, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 2, 130, 256)       1024
_________________________________________________________________
dropout (Dropout)            (None, 2, 130, 256)       0
_________________________________________________________________
zero_padding2d_1 (ZeroPaddin (None, 2, 134, 256)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 1, 132, 80)        122960
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 132, 80)        0
_________________________________________________________________
flatten (Flatten)            (None, 10560)             0
_________________________________________________________________
dense1 (Dense)               (None, 256)               2703616
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense2 (Dense)               (None, 11)                2827
_________________________________________________________________
activation (Activation)      (None, 11)                0
_________________________________________________________________
reshape_1 (Reshape)          (None, 11)                0
=================================================================
Total params: 2,830,427
Trainable params: 2,830,427
Non-trainable params: 0
```

Figure 4.2: CNN parameters

4.5.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 50. Google Colab CPU took 492 seconds to train one epoch the final validation accuracy came out to be 50.28%. The model weights were saved. Figure 4.3 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

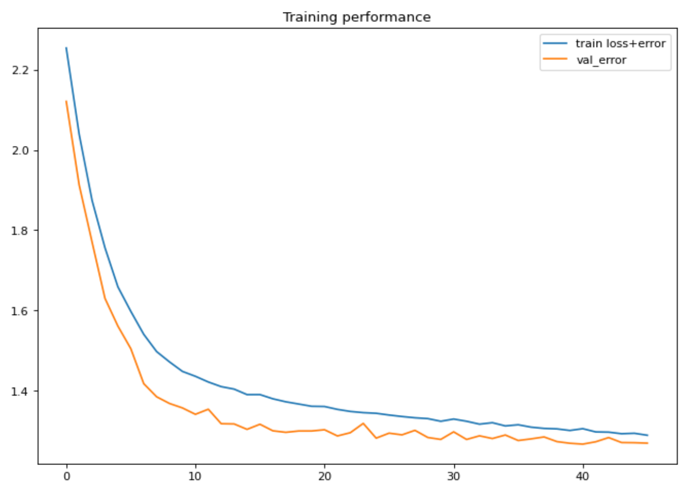
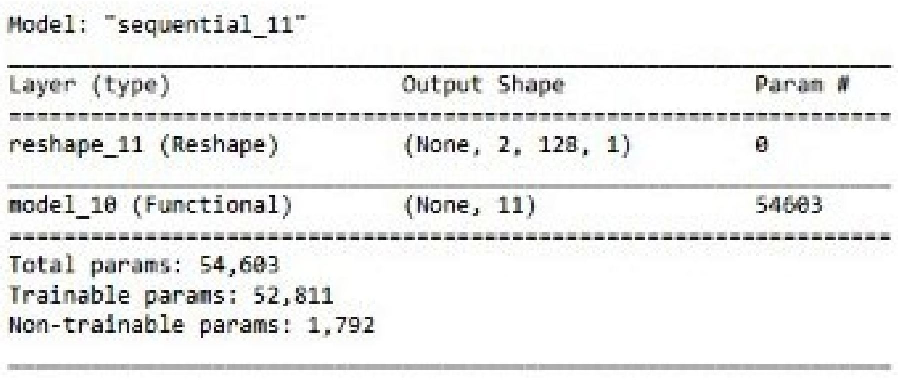


Figure 4.3: CNN Training Performance

4.6 ResNet

4.6.1 Sequential Model

Figure 4.4 shows the parameters for the implemented ResNet Model.



```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_11 (Reshape)         (None, 2, 128, 1)         0
_________________________________________________________________
model_10 (Functional)        (None, 11)                54603
=================================================================
Total params: 54,603
Trainable params: 52,811
Non-trainable params: 1,792
_________________________________________________________________
```

Figure 4.4: ResNet parameters

4.6.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. Google Colab CPU

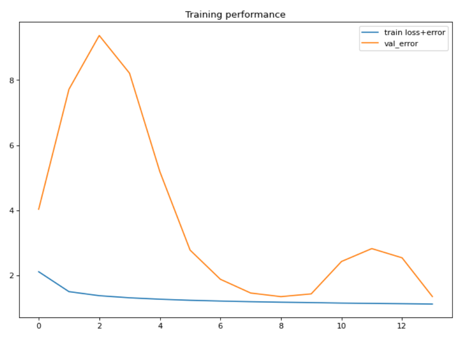


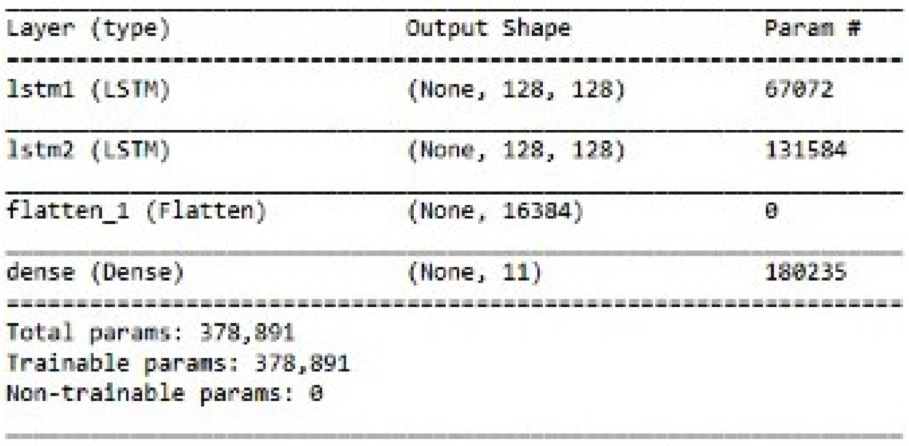
Figure 4.5: Training Performance of ResNet

took 916 seconds to train one epoch the final validation accuracy came out to be 43.53%. The model weights were saved.
Figure 4.5 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

4.7 Long Short Term Memory(LSTM)

4.7.1 Sequential Model

Figure 4.6 shows the parameters for the implemented Long Short Term Memory Model.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm1 (LSTM)                 (None, 128, 128)          67072
_________________________________________________________________
lstm2 (LSTM)                 (None, 128, 128)          131584
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense (Dense)                (None, 11)                180235
=================================================================
Total params: 378,891
Trainable params: 378,891
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.6: LSTM parameters

4.7.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. Google Colab CPU took 1254 seconds to train one epoch the final validation accuracy came out to be 42.38%. The model weights were saved. Figure 4.7 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

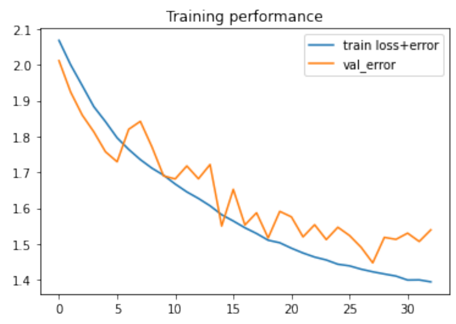
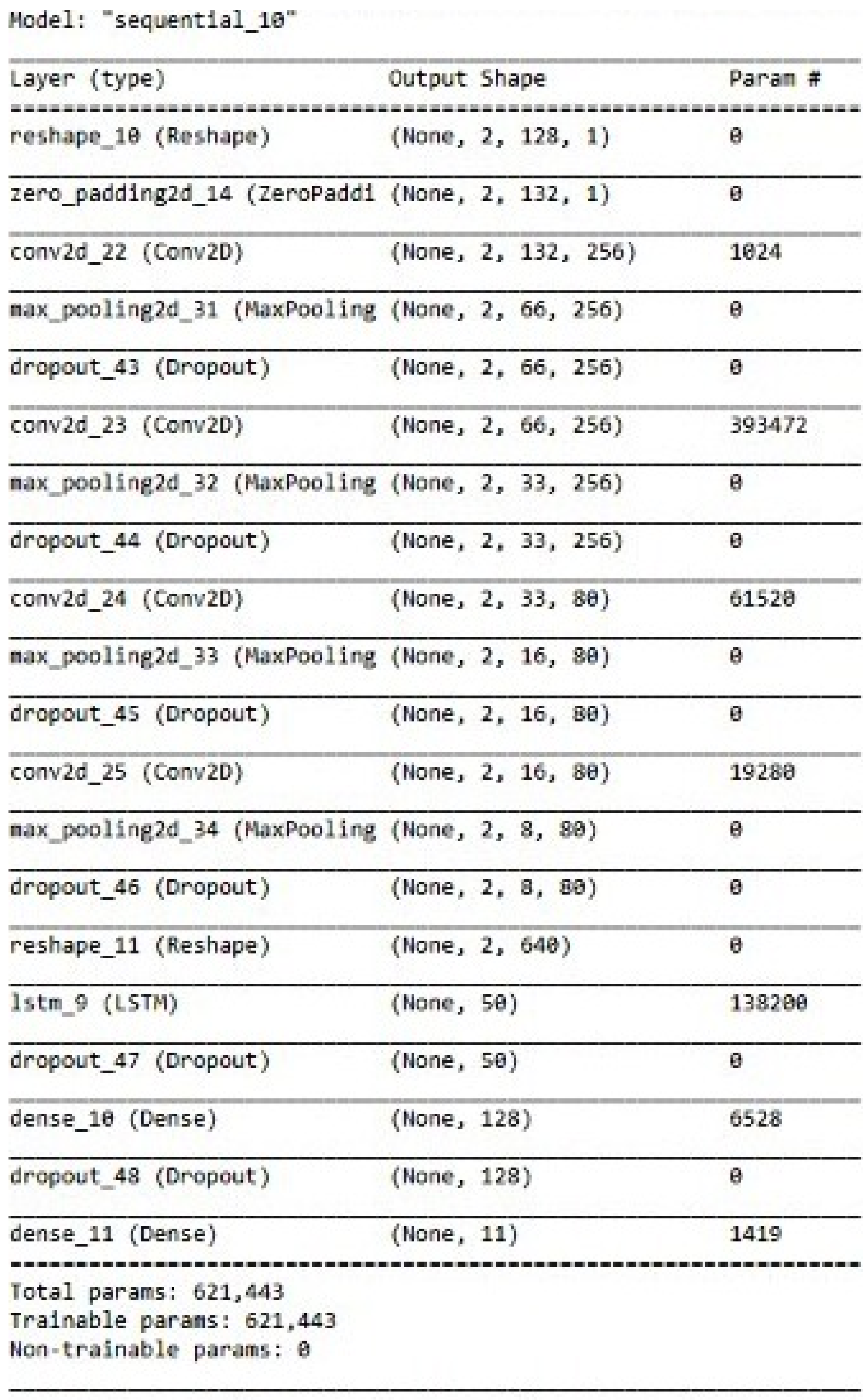


Figure 4.7: Training Performance of LSTM

4.8 CNN-LSTM

4.8.1 Sequential Model

Figure 4.8 shows the parameters for the implemented CNN-LSTM Model: In our model we zero pad our incoming data, and then pass it through 3 convolutional layers, which are attached to further dropout and zero padding layers. After that the dimensions of the data are reshaped to be input into the LSTM portion of the network . After passing through the LSTM network, the data is then sent through 3 densely connected layers, where we get out output through the softmax function.



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_10 (Reshape)         (None, 2, 128, 1)         0
_________________________________________________________________
zero_padding2d_14 (ZeroPaddi (None, 2, 132, 1)         0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 2, 132, 256)       1024
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 2, 66, 256)        0
_________________________________________________________________
dropout_43 (Dropout)         (None, 2, 66, 256)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 2, 66, 256)        393472
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 2, 33, 256)        0
_________________________________________________________________
dropout_44 (Dropout)         (None, 2, 33, 256)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 2, 33, 80)         61520
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 2, 16, 80)         0
_________________________________________________________________
dropout_45 (Dropout)         (None, 2, 16, 80)         0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 2, 16, 80)         19280
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 2, 8, 80)          0
_________________________________________________________________
dropout_46 (Dropout)         (None, 2, 8, 80)          0
_________________________________________________________________
reshape_11 (Reshape)         (None, 2, 640)            0
_________________________________________________________________
lstm_9 (LSTM)                (None, 50)                138200
_________________________________________________________________
dropout_47 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_10 (Dense)             (None, 128)               6528
_________________________________________________________________
dropout_48 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 11)                1419
=================================================================
Total params: 621,443
Trainable params: 621,443
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.8: CNN-LSTM Parameters

4.8.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 59.45%. The model weights were saved. We will see the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs in the next chapter.

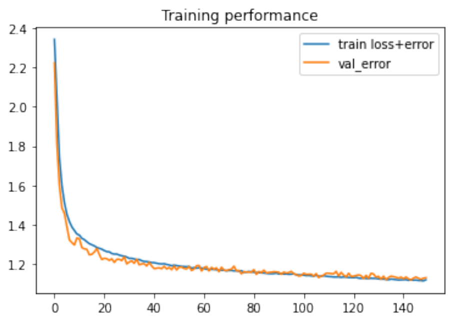
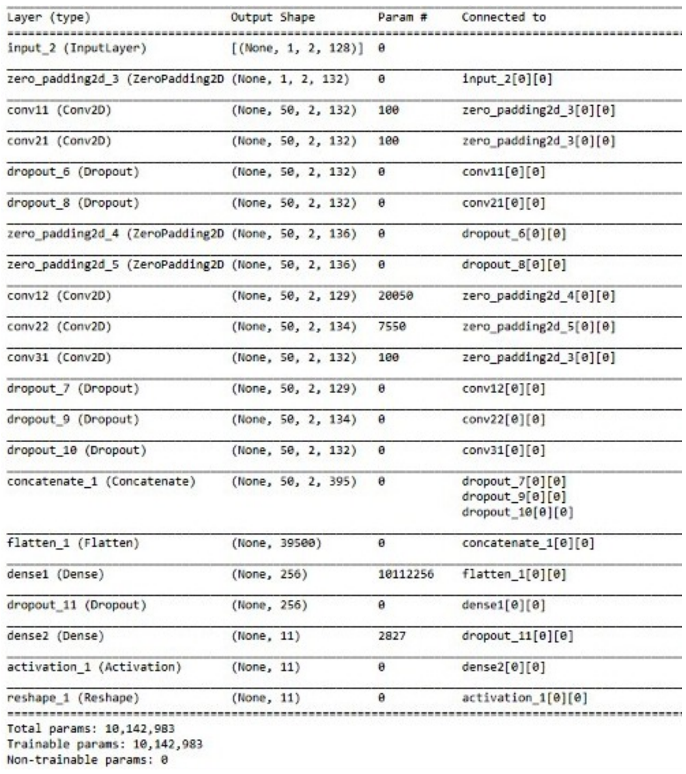


Figure 4.9: Training Performance of CNN-LSTM

4.9 Inception Network

4.9.1 Sequential Model

Figure 4.10 shows the parameters for the implemented Inception Network Model:



```
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_2 (InputLayer)            [(None, 1, 2, 128)]  0
________________________________________________________________________________________________
zero_padding2d_3 (ZeroPadding2D (None, 1, 2, 132)    0           input_2[0][0]
________________________________________________________________________________________________
conv11 (Conv2D)                 (None, 50, 2, 132)   100         zero_padding2d_3[0][0]
________________________________________________________________________________________________
conv21 (Conv2D)                 (None, 50, 2, 132)   100         zero_padding2d_3[0][0]
________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 50, 2, 132)   0           conv11[0][0]
________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 50, 2, 132)   0           conv21[0][0]
________________________________________________________________________________________________
zero_padding2d_4 (ZeroPadding2D (None, 50, 2, 136)   0           dropout_6[0][0]
________________________________________________________________________________________________
zero_padding2d_5 (ZeroPadding2D (None, 50, 2, 136)   0           dropout_8[0][0]
________________________________________________________________________________________________
conv12 (Conv2D)                 (None, 50, 2, 129)   20050       zero_padding2d_4[0][0]
________________________________________________________________________________________________
conv22 (Conv2D)                 (None, 50, 2, 134)   7550        zero_padding2d_5[0][0]
________________________________________________________________________________________________
conv31 (Conv2D)                 (None, 50, 2, 132)   100         zero_padding2d_3[0][0]
________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 50, 2, 129)   0           conv12[0][0]
________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 50, 2, 134)   0           conv22[0][0]
________________________________________________________________________________________________
dropout_10 (Dropout)            (None, 50, 2, 132)   0           conv31[0][0]
________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 50, 2, 395)   0           dropout_7[0][0]
                                                                 dropout_9[0][0]
                                                                 dropout_10[0][0]
________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 39500)        0           concatenate_1[0][0]
________________________________________________________________________________________________
dense1 (Dense)                  (None, 256)          10112256    flatten_1[0][0]
________________________________________________________________________________________________
dropout_11 (Dropout)            (None, 256)          0           dense1[0][0]
________________________________________________________________________________________________
dense2 (Dense)                  (None, 11)           2827        dropout_11[0][0]
________________________________________________________________________________________________
activation_1 (Activation)       (None, 11)           0           dense2[0][0]
________________________________________________________________________________________________
reshape_1 (Reshape)             (None, 11)           0           activation_1[0][0]
================================================================================================
Total params: 10,142,983
Trainable params: 10,142,983
Non-trainable params: 0
```

Figure 4.10: Inception Network Parameters

4.9.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 48.74%. The model weights were saved. Figure 4.11 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

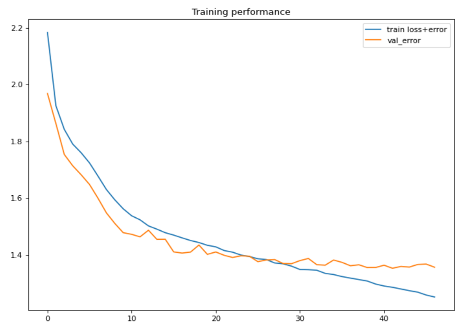
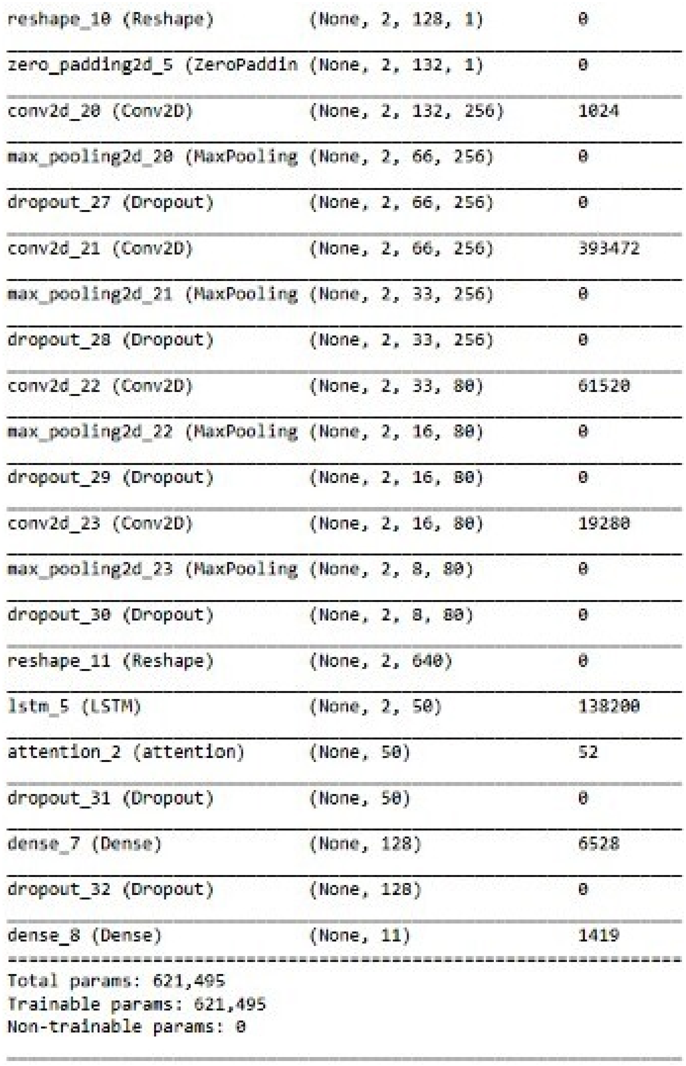


Figure 4.11: Inception Network Training

4.10 CNN-LSTM(Attention)

4.10.1 Sequential Model

Figure 4.12 shows the parameters for the implemented Inception Network Model:



```
reshape_10 (Reshape)         (None, 2, 128, 1)         0
_________________________________________________________________
zero_padding2d_5 (ZeroPaddin (None, 2, 132, 1)         0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 2, 132, 256)       1024
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 2, 66, 256)        0
_________________________________________________________________
dropout_27 (Dropout)         (None, 2, 66, 256)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 2, 66, 256)        393472
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 2, 33, 256)        0
_________________________________________________________________
dropout_28 (Dropout)         (None, 2, 33, 256)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 2, 33, 80)         61520
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 2, 16, 80)         0
_________________________________________________________________
dropout_29 (Dropout)         (None, 2, 16, 80)         0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 2, 16, 80)         19280
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 2, 8, 80)          0
_________________________________________________________________
dropout_30 (Dropout)         (None, 2, 8, 80)          0
_________________________________________________________________
reshape_11 (Reshape)         (None, 2, 640)            0
_________________________________________________________________
lstm_5 (LSTM)                (None, 2, 50)             138200
_________________________________________________________________
attention_2 (attention)      (None, 50)                52
_________________________________________________________________
dropout_31 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               6528
_________________________________________________________________
dropout_32 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 11)                1419
=================================================================
Total params: 621,495
Trainable params: 621,495
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.12: CNN-LSTM Network Parameters

4.10.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 59.11%. The model weights were saved. Figure 4.13 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

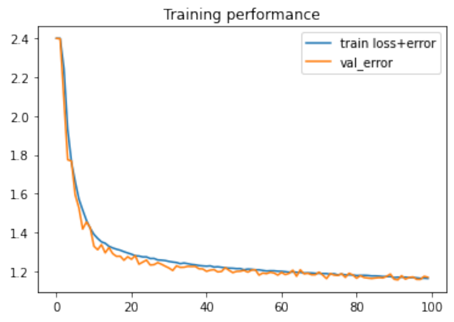
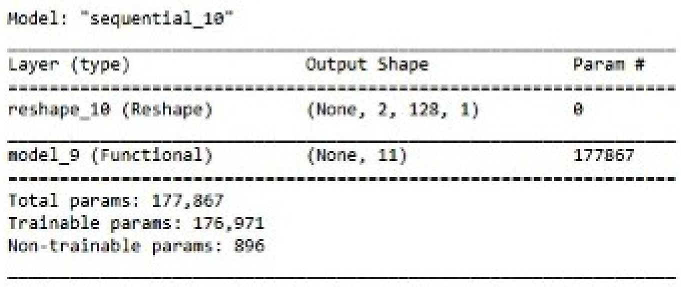


Figure 4.13: CNN-LSTM Training

4.11 VGG

4.11.1 Sequential Model

Figure 4.14 shows the parameters for the implemented Inception Network Model:



```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_10 (Reshape)         (None, 2, 128, 1)         0
_________________________________________________________________
model_9 (Functional)         (None, 11)                177867
=================================================================
Total params: 177,867
Trainable params: 176,971
Non-trainable params: 896
_________________________________________________________________
```

Figure 4.14: VGG Network Parameters

4.11.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 37.84%. The model weights were saved. Figure 4.15 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

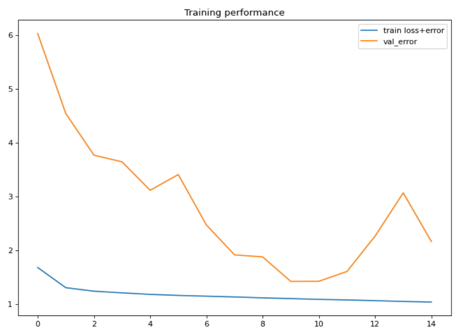
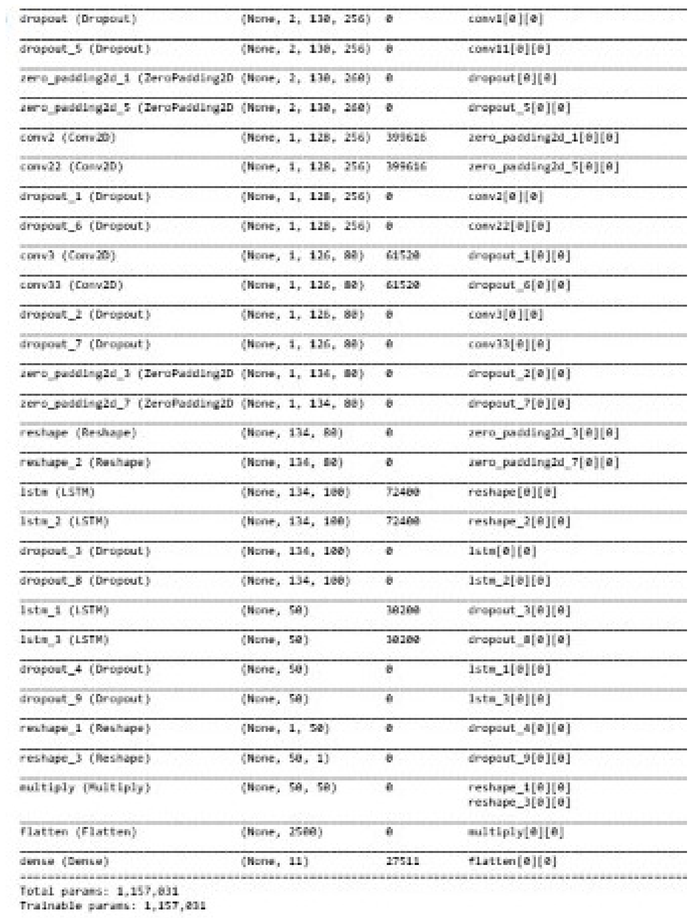


Figure 4.15: VGG Network Training

4.12 Dual Stream LSTM-CNN Network

4.12.1 Sequential Model

Figure 4.16 shows the parameters for the implemented Inception Network Model:



```
________________________________________________________________________________
dropout (Dropout)              (None, 2, 130, 256)  0           conv1[0][0]
________________________________________________________________________________
dropout_5 (Dropout)            (None, 2, 130, 256)  0           conv11[0][0]
________________________________________________________________________________
zero_padding2d_1 (ZeroPadding2D (None, 2, 130, 260)  0           dropout[0][0]
________________________________________________________________________________
zero_padding2d_5 (ZeroPadding2D (None, 2, 130, 260)  0           dropout_5[0][0]
________________________________________________________________________________
conv2 (Conv2D)                 (None, 1, 128, 256)  399616      zero_padding2d_1[0][0]
________________________________________________________________________________
conv22 (Conv2D)                (None, 1, 128, 256)  399616      zero_padding2d_5[0][0]
________________________________________________________________________________
dropout_1 (Dropout)            (None, 1, 128, 256)  0           conv2[0][0]
________________________________________________________________________________
dropout_6 (Dropout)            (None, 1, 128, 256)  0           conv22[0][0]
________________________________________________________________________________
conv3 (Conv2D)                 (None, 1, 126, 80)   61520       dropout_1[0][0]
________________________________________________________________________________
conv33 (Conv2D)                (None, 1, 126, 80)   61520       dropout_6[0][0]
________________________________________________________________________________
dropout_2 (Dropout)            (None, 1, 126, 80)   0           conv3[0][0]
________________________________________________________________________________
dropout_7 (Dropout)            (None, 1, 126, 80)   0           conv33[0][0]
________________________________________________________________________________
zero_padding2d_3 (ZeroPadding2D (None, 1, 134, 80)   0           dropout_2[0][0]
________________________________________________________________________________
zero_padding2d_7 (ZeroPadding2D (None, 1, 134, 80)   0           dropout_7[0][0]
________________________________________________________________________________
reshape (Reshape)              (None, 134, 80)      0           zero_padding2d_3[0][0]
________________________________________________________________________________
reshape_2 (Reshape)            (None, 134, 80)      0           zero_padding2d_7[0][0]
________________________________________________________________________________
lstm (LSTM)                    (None, 134, 100)     72400       reshape[0][0]
________________________________________________________________________________
lstm_2 (LSTM)                  (None, 134, 100)     72400       reshape_2[0][0]
________________________________________________________________________________
dropout_3 (Dropout)            (None, 134, 100)     0           lstm[0][0]
________________________________________________________________________________
dropout_8 (Dropout)            (None, 134, 100)     0           lstm_2[0][0]
________________________________________________________________________________
lstm_1 (LSTM)                  (None, 50)           30200       dropout_3[0][0]
________________________________________________________________________________
lstm_3 (LSTM)                  (None, 50)           30200       dropout_8[0][0]
________________________________________________________________________________
dropout_4 (Dropout)            (None, 50)           0           lstm_1[0][0]
________________________________________________________________________________
dropout_9 (Dropout)            (None, 50)           0           lstm_3[0][0]
________________________________________________________________________________
reshape_1 (Reshape)            (None, 1, 50)        0           dropout_4[0][0]
________________________________________________________________________________
reshape_3 (Reshape)            (None, 50, 1)        0           dropout_9[0][0]
________________________________________________________________________________
multiply (Multiply)            (None, 50, 50)       0           reshape_1[0][0]
                                                                reshape_3[0][0]
________________________________________________________________________________
flatten (Flatten)              (None, 2500)         0           multiply[0][0]
________________________________________________________________________________
dense (Dense)                  (None, 11)           27511       flatten[0][0]
================================================================================
Total params: 1,157,031
Trainable params: 1,157,031
```

Figure 4.16: Dual Stream LSTM-CNN Network Parameters

4.12.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 54.41%. The model weights were saved. Figure 4.17 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.

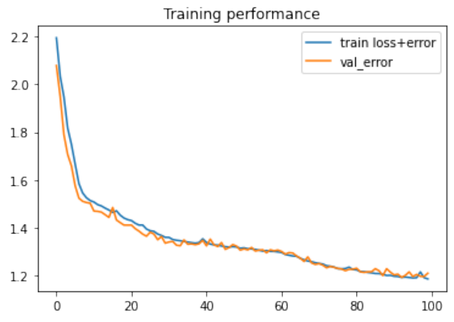


Figure 4.17: Dual Stream LSTM-CNN Network Training

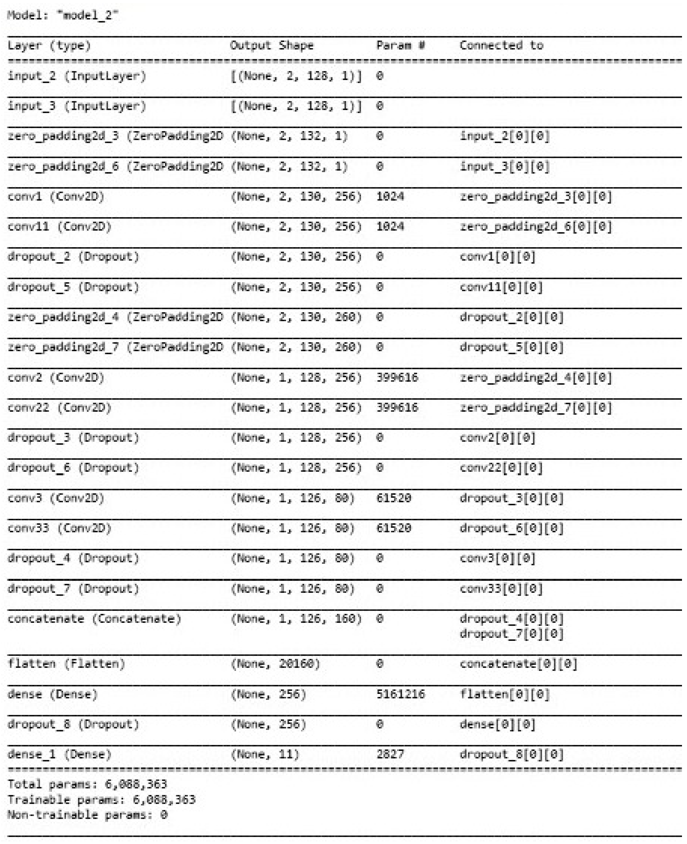
4.13 Dual Stream CNN Network

4.13.1 Sequential Model

Figure 4.18 shows the parameters for the implemented Inception Network Model:

4.13.2 Training the Model

For training the model dropout rate was chosen to be 0.5, the batch size was 1024 and the number of epochs were 100. The final validation accuracy came out to be 50.10%. The model weights were saved. Figure 4.19 shows the Training Performance and the variation of loss and value error as the model was being trained with respect to epochs.



```
Model: "model_2"
____________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
====================================================================================================
input_2 (InputLayer)            [(None, 2, 128, 1)]  0
____________________________________________________________________________________________________
input_3 (InputLayer)            [(None, 2, 128, 1)]  0
____________________________________________________________________________________________________
zero_padding2d_3 (ZeroPadding2D (None, 2, 132, 1)    0           input_2[0][0]
____________________________________________________________________________________________________
zero_padding2d_6 (ZeroPadding2D (None, 2, 132, 1)    0           input_3[0][0]
____________________________________________________________________________________________________
conv1 (Conv2D)                  (None, 2, 130, 256)  1024        zero_padding2d_3[0][0]
____________________________________________________________________________________________________
conv11 (Conv2D)                 (None, 2, 130, 256)  1024        zero_padding2d_6[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 2, 130, 256)  0           conv1[0][0]
____________________________________________________________________________________________________
dropout_5 (Dropout)             (None, 2, 130, 256)  0           conv11[0][0]
____________________________________________________________________________________________________
zero_padding2d_4 (ZeroPadding2D (None, 2, 130, 260)  0           dropout_2[0][0]
____________________________________________________________________________________________________
zero_padding2d_7 (ZeroPadding2D (None, 2, 130, 260)  0           dropout_5[0][0]
____________________________________________________________________________________________________
conv2 (Conv2D)                  (None, 1, 128, 256)  399616      zero_padding2d_4[0][0]
____________________________________________________________________________________________________
conv22 (Conv2D)                 (None, 1, 128, 256)  399616      zero_padding2d_7[0][0]
____________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 1, 128, 256)  0           conv2[0][0]
____________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 1, 128, 256)  0           conv22[0][0]
____________________________________________________________________________________________________
conv3 (Conv2D)                  (None, 1, 126, 80)   61520       dropout_3[0][0]
____________________________________________________________________________________________________
conv33 (Conv2D)                 (None, 1, 126, 80)   61520       dropout_6[0][0]
____________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 1, 126, 80)   0           conv3[0][0]
____________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 1, 126, 80)   0           conv33[0][0]
____________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1, 126, 160)  0           dropout_4[0][0]
                                                                 dropout_7[0][0]
____________________________________________________________________________________________________
flatten (Flatten)               (None, 20160)        0           concatenate[0][0]
____________________________________________________________________________________________________
dense (Dense)                   (None, 256)          5161216     flatten[0][0]
____________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 256)          0           dense[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                 (None, 11)           2827        dropout_8[0][0]
====================================================================================================
Total params: 6,088,363
Trainable params: 6,088,363
Non-trainable params: 0
____________________________________________________________________________________________________
```

Figure 4.18: Dual Stream CNN Network Parameters

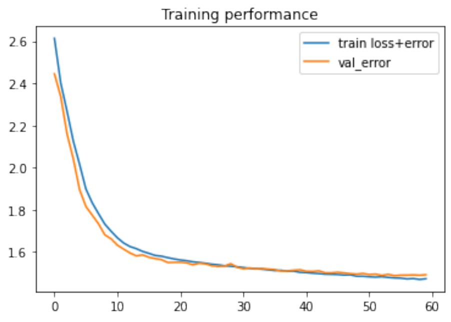


Figure 4.19: Dual Stream CNN Network Training

Chapter 5: Results

5.1 Convolutional Neural Network

The overall validation accuracy for this model came out to be 50.28% and the training accuracy to be 50.29%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.1, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 69% which is acceptable for a real world application. The average classification accuracies of different modulation techniques are given in figure 5.2 in the form of a confusion matrix.

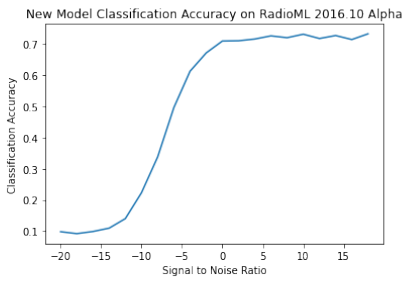


Figure 5.1: Accuracy vs SNR

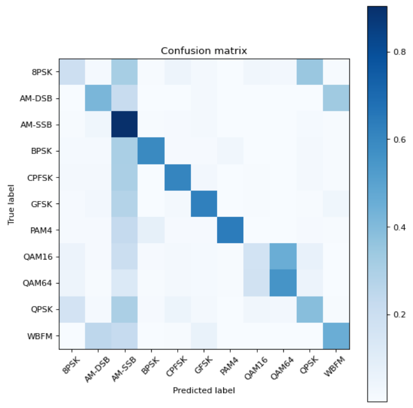


Figure 5.2: Confusion Matrix.

5.2 Long Short Term Memory

The overall validation accuracy for this model came out to be 42.38% and the training accuracy to be 47.59%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.3, for low Signal to Noise ratios the accuracy was very low , even for SNR = -2dB it was around 56.25% which is quite low. Furthermore,average classification accuracies of different modulation

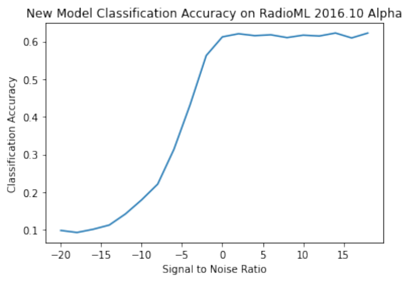


Figure 5.3: Accuracy vs SNR

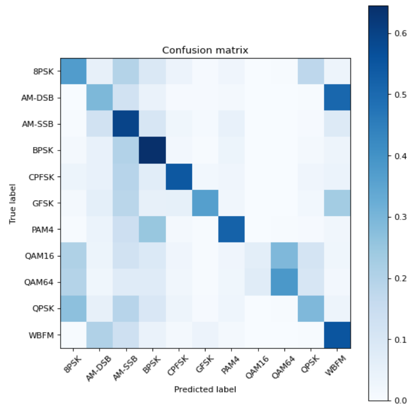


Figure 5.4: Confusion Matrix.

techniques are given in figure 5.4 in the form of a confusion matrix.

5.3 ResNet

The overall validation accuracy for this model came out to be 48.55% and the training accuracy to be 58.42%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.5, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 61% which is acceptable for a real world application.

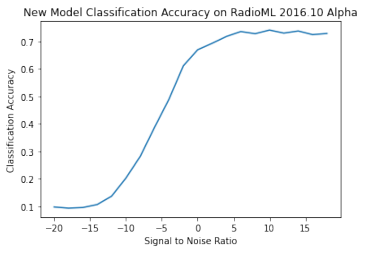


Figure 5.5: Accuracy vs SNR

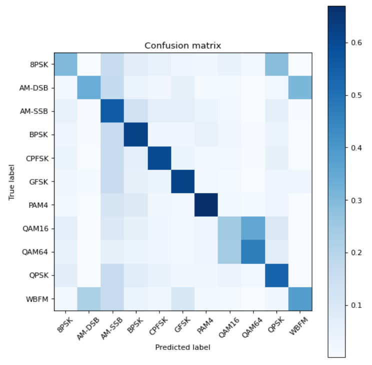


Figure 5.6: Confusion Matrix.

5.4 CNN-LSTM

The overall validation accuracy for this model came out to be 56.78% and the training accuracy to be 57.10%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.7, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 76.59% which is acceptable for a real world application. Furthermore, the average classification

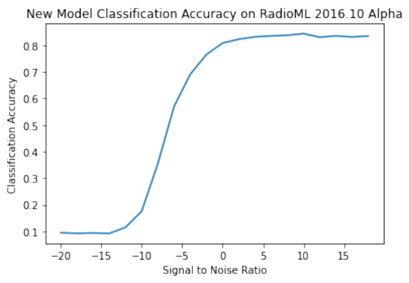


Figure 5.7: Accuracy vs SNR

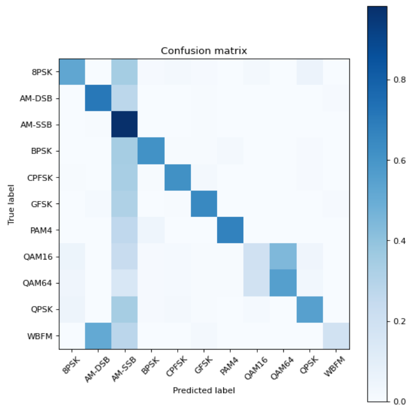


Figure 5.8: Confusion Matrix.

accuracies of different modulation techniques are given in the form of a confusion matrix figure 5.8 in the form of a confusion matrix.

5.5 Inception Network

The overall validation accuracy for this model came out to be 48.74% and the training accuracy to be 51.79%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.9, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 63% which is acceptable for a real world application.

Furthermore,average classification accuracies of different modulation techniques are given in 5.10 in the form of a confusion matrix

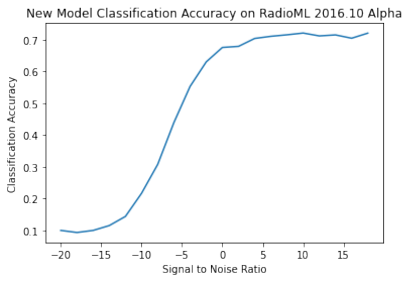


Figure 5.9: Accuracy vs SNR

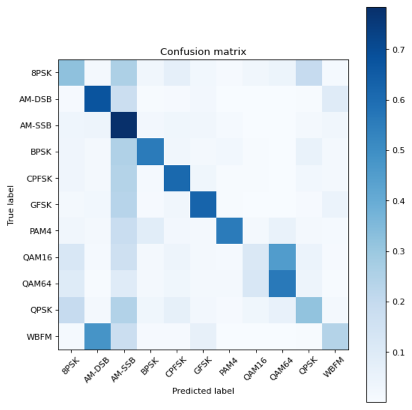


Figure 5.10: Confusion Matrix.

5.6 VGG

The overall validation accuracy for this model came out to be 37.84% and the training accuracy to be 60.84%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.11, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 49.72% which is not acceptable for a real world application.

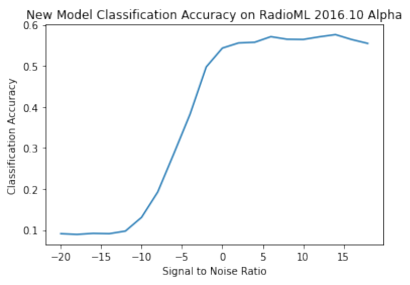


Figure 5.11: Accuracy vs SNR

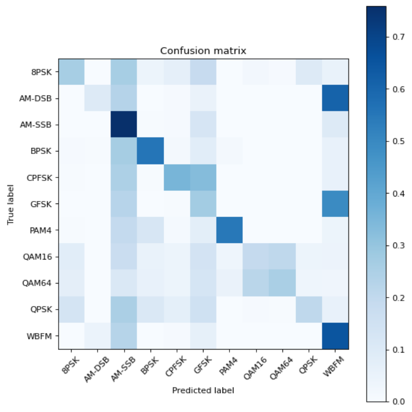


Figure 5.12: Confusion Matrix.

Furthermore,average classification accuracies of different modulation techniques are given in figure 5.12 in the form of a confusion matrix

5.7 CNN-LSTM(Attention)

The overall validation accuracy for this model came out to be 59.11% and the training accuracy to be 60.16%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.13, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 81.30% which is acceptable for a real world application. Furthermore,average classification accuracies

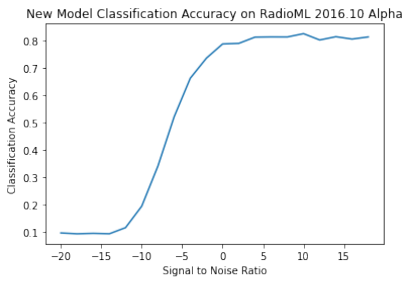


Figure 5.13: Accuracy vs SNR

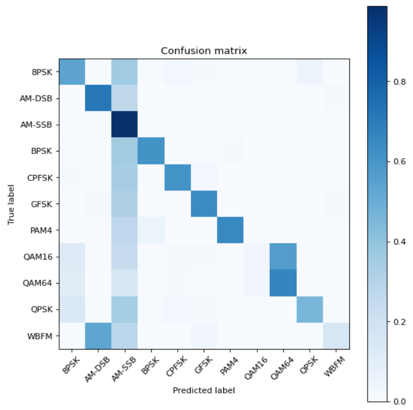


Figure 5.14: Confusion Matrix.

of different modulation techniques are given in figure 5.14 in the form of a confusion matrix

5.8 Dual Stream CNN-LSTM

The overall validation accuracy for this model came out to be 54.41% and the training accuracy to be 54.80%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.15, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 73% which is pretty good for a real world application.

Furthermore,average classification accuracies of different modulation techniques are given in 5.16 in the form of a confusion matrix

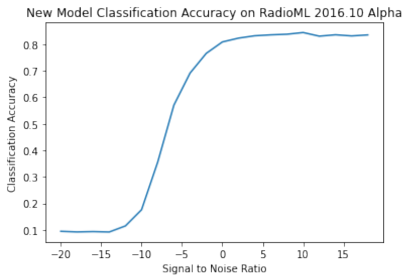


Figure 5.15: Accuracy vs SNR

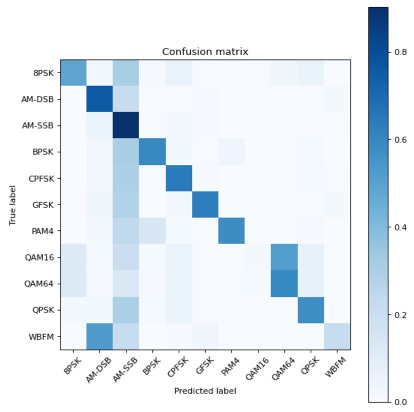


Figure 5.16: Confusion Matrix.

5.9 Dual Stream CNN

The overall validation accuracy for this model came out to be 50.10% and the training accuracy to be 50.16%. Classification accuracy vs SNR trend was found out to be as expected which can be seen in figure 5.17, for low Signal to Noise ratios the accuracy was very low but for SNR = -2dB it was around 55% which is not acceptable for a real world application.

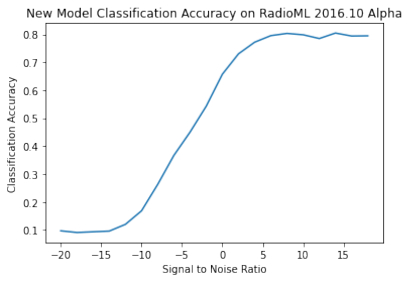


Figure 5.17: Accuracy vs SNR

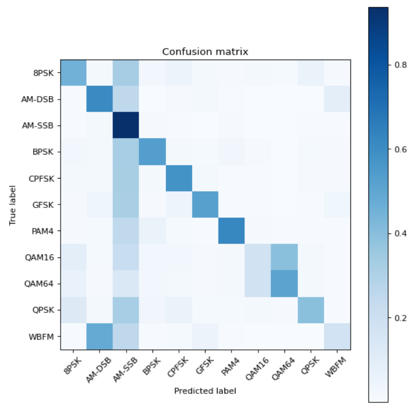


Figure 5.18: Confusion Matrix.

Furthermore, the average classification accuracies of different modulation techniques are given in figure 5.18 in the form of a confusion matrix.

5.10 Comparative Analysis

The following Figure 5.19 gives us a comparative analysis of all the models implemented, in a single graph. Here we can clearly see that the Dual Stream CNN LSTM network outperforms all other networks, and that the CNN-LSTM network is a close second.

It is also interesting to note that there is a gap of at least 5% between the top 2 performing CNN-LSTM networks and all other networks at higher SNRs. Even then, the network that performs third best is also a CNN-LSTM based model. This clearly depicts that the CNN-LSTM architecture is much better suited for this kind of classification problem, than any other network architecture.

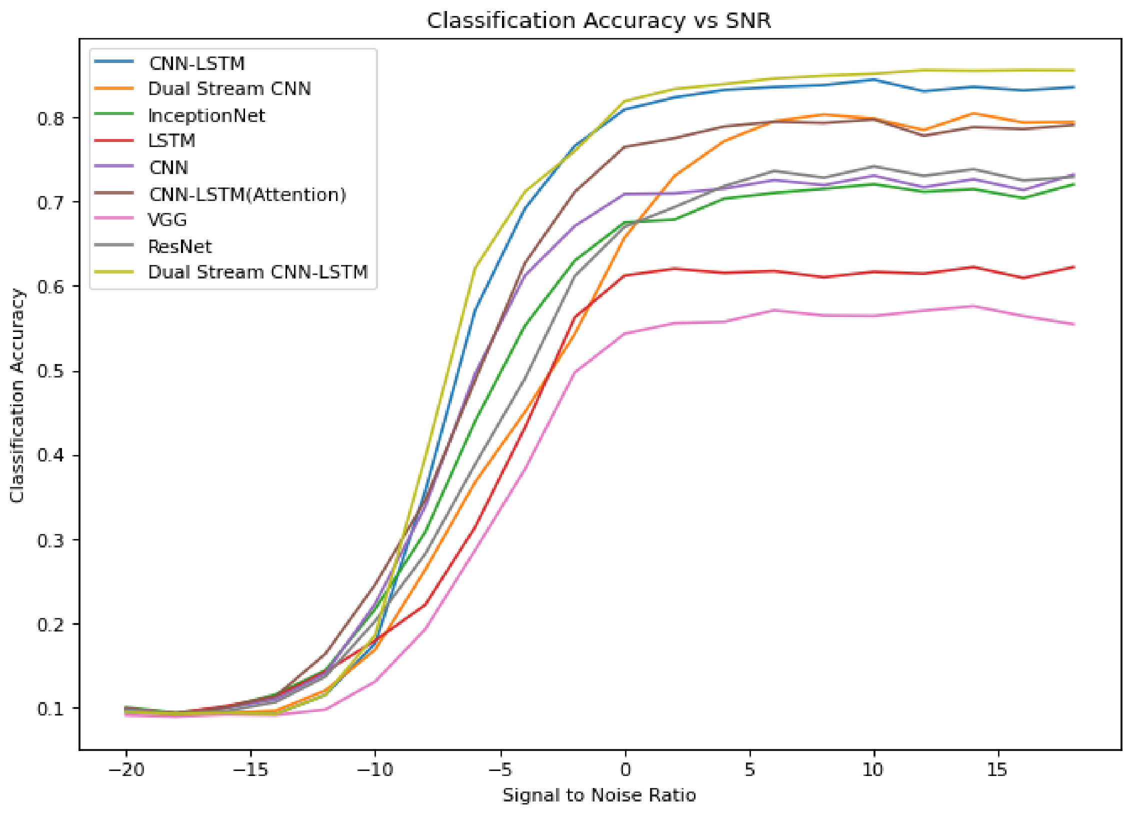
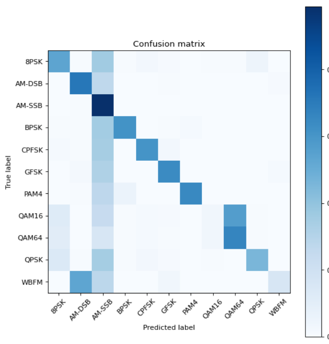


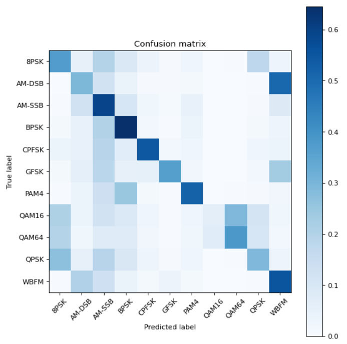
Figure 5.19: Dual Stream LSTM-CNN Network Training

The next page depicts the average classification accuracy of each confusion matrix for all the networks deployed. Here again, we can see a clear pattern emerge: The diagonal is much cleaner for all CNN-LSTM based architectures as compared to other architectures which have noisier graphs. This demonstrates the better classification accuracy of these networks.

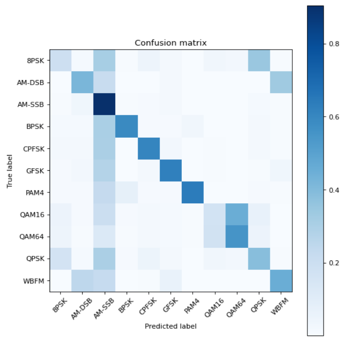
Additionally, we also note that the most misclassified modulation scheme is AM-SSB, as is evident from the column which is common to all networks. We are yet to ascertain the exact reason behind this.



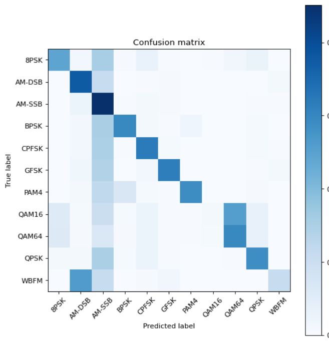
(a) CNN-LSTM(Att)



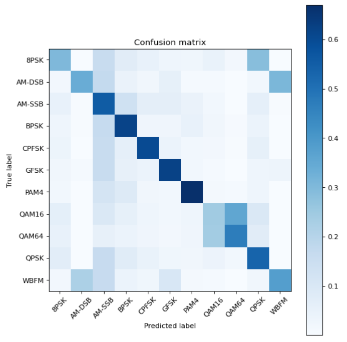
(b) LSTM



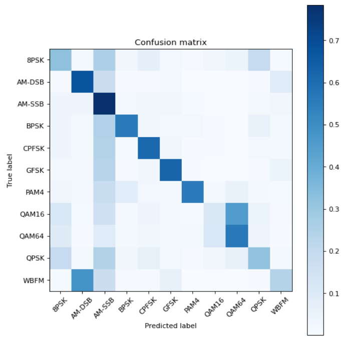
(c) CNN



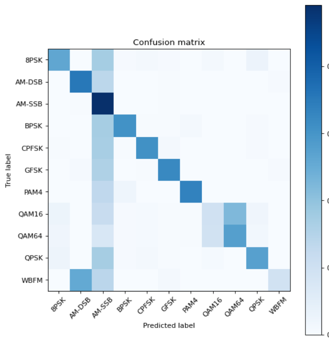
(d) DS CNN-LSTM



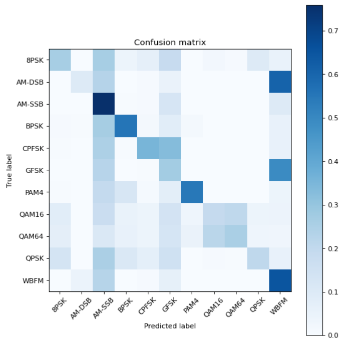
(e) ResNet



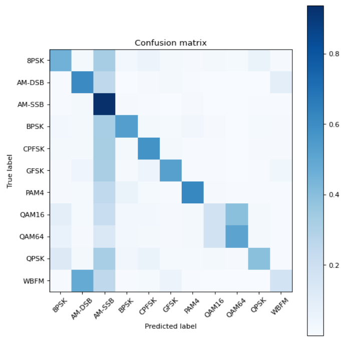
(f) InceptionNet



(g) CNN-LSTM



(h) VGG



(i) DS CNN

Figure 5.20: Confusion Matrices

Chapter 6: Conclusion and Future Work

Through the development of our project we have implemented a number of different networks for modulation recognition, including :

1. CNN
2. LSTM
3. CNN-LSTM
4. ResNet
5. Inception Network
6. VGG
7. Attention Module
8. Dual Stream CNN
9. Dual Stream CNN-LSTM

We developed all these different networks based on the recommendation of previous papers in the field that validation accuracy in the radio domain was not a simple function of the depth of the Neural Network but of the features learned by these layers. We focused on implementing as many architectures as possible, in a breadth first manner, in order to find an architecture which seemed most suitable for further development in depth.

The results of all these networks have been tabulated and plotted in the previous section.

Upon analysis of the result, the first important conclusion is that our results with similar architectures, were still not giving us classification accuracy as high as the published models. This might be because we have not exhaustively explored all hyperparameters. This is a clear future avenue for this project.

Upon starting the project, we implemented the CNN architecture in Timothy O' Shea's 2016 paper.We found that the CNN's overall validation accuracy for this model came out to be 50.28% and the training accuracy to be 50.29% after running 100 epochs. For low Signal to Noise ratios the classification accuracy was very low but for SNR = -2dB it was around 69% . The LSTM model demonstrated much better performance in classification accuracy. After training only 13 epochs the LSTM had a validation accuracy of 42.38% and a training accuracy of 45.50%. However each epoch takes close to 1000 seconds to run. This also causes the problem of inability to go through all our epochs, as Google Colab has a time limit on training time. The results for SNR classification vs accuracy were also lower than the CNN model at the SNR of -2 dB.

Based on these results we realized that implementation of either the vanilla CNN or vanilla LSTM in their current form will not succeed. We would have to modify our networks and use more complex architectures to perform our task. To that effect we have implemented the ResNet, Inception network, and VGG which use ideas in the field of vision and image recognition especially regarding layer depth, and applied them to our problem set. Further we have implemented the CNN- LSTM which looked like a promising candidate due to the nature of the data, as it varies both temporally and spatially.

The CNN LSTM performed much better than other networks that we had deployed especially at lower SNR's. The CNN LSTM outputted a classification accuracy close to 70% at -2 db SNR, which makes it possible for real time applications. This helped us focus our attention in this direction, and we decided to developed more architectures using CNN-LSTM. We embedded an Attention module in the CNN-LSTM which improved results, and then we decided to implement the dual stream CNN-LSTM network from Zhang's paper based on its promising results. The dual stream model we implemented while giving us promising results, did not perform as well as the published model.

The other implemented models did not provide us with the accuracy we were expecting, while being quite computationally expensive.

Based on our experimentation with different architectures we believe that future work in this domain should be focused on improving the CNN-LSTM single & dual stream architectures. This is due to the fact that this architecture can use both temporal and spatial information from our data, which are both meaningful in the radio context.

To this effect, we have a number of suggestions and ideas for future work. We are in the process of implementing Denoising autoencoders, which can be implemented as simple blocks or using stacked autoencoders. Papers using these blocks show promising results. We can use them to denoise our data both in the IQ and AP plane and use them for both dual and single stream CNN-LSTM networks. Since CNN-LSTM with Attention showed better results than the plain CNN-LSTM we can add this Attention module to our dual stream network as well. Additionally, we can try training our CNN- LSTM network solely on the AP data and see if that leads to an increase in accuracy. Our literature survey has also made us aware of other novel networks being implemented which correct the signal before passing them into a CNN. This correctional network can correct phase offset, carrier frequency, and noise, which all affect the accuracy of classification. We aim to implement this correctional network and then pass the corrected signal to our CNN-LSTM network(dual or single stream). Last we believe that we can still optimize hyperparamters for the CNN-LSTM networks, as we have not done an exhaustive search yet.

DeepSig has more than one dataset available for this problem. The 2018 DeepSig dataset is much bigger and has 24 different modulation schemes. It is a clear goal of ours to run our network on this larger dataset and see how our network performs.

This is not all, the current network architecture is not very robust, in terms of the overall communication system. It is impossible to implement it in a real life scenario based on its current status. Future networks used in real-world applications will need to learn to either resample signals to be bandwidth normalized or learn features for many bandwidths. Networks that can resample, synchronize, and remove non-linear channel distortions are all exciting future work for the field.

We believe that as radio environments become increasingly complex, combining varying temporal behaviors of modulations, multi-modulation protocols and combining many radio emitters interoperating within a single band, many of these notions of hierarchy within deep networks will become increasingly important in allowing our networks to scale to cope with the complexity effectively as has been similarly shown in the vision domain within complex multi-object scenes.

Finally, we would also like to briefly discuss issues faced by us during the execution of the project. The lack of computational resources or funding led us to use freely available cloud based computation. While Google Colab is a great resource, and we are immensely thankful for the ability to develop and train our models remotely, we faced a few issues. There is a cap on the amount of GPU power that can be used by free users, additionally, there is not much flexibility in choosing the available boards. Over and above that, there is also a time cap for how long a model can be trained, due to this we were not able to train a few models with a larger number of parameters for more than 20-30 epochs. We would like to offer a suggestion to our respected faculty, that from next year on, they should allow undergraduate students developing deep learning projects access to the university's computing resources.

Bibliography

[1] Elsayed Azzouz and Asoke Kumar Nandi. “Automatic modulation recognition of communication signals”. In: (2013).

[2] Nikhil Balwani et al. “Long Short-Term Memory based Spectrum Sensing Scheme for Cognitive Radio”. In: *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC).* IEEE. 2019, pp. 1–6.

[3] Eric Blossom. “GNU radio: tools for exploring the radio frequency spectrum”. In: *Linux journal* 2004.122 (2004), p. 4.

[4] Charles Clancy et al. “Applications of machine learning to cognitive radio networks”. In: *IEEE Wireless Communications* 14.4 (2007), pp. 47–52.

[5] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016). DOI: `10.1109/cvpr.2016.90`.

[6] Kaisheng Liao et al. “Sequential Convolutional Recurrent Neural Networks for Fast Automatic Modulation Classification”. In: *IEEE Access* 9 (2021), pp. 27182–27188. DOI: `10.1109/access.2021.3053427`.

[7] Xiaoyu Liu, Diyu Yang, and Aly El Gamal. “Deep neural network architectures for modulation classification”. In: *2017 51st Asilomar Conference on Signals, Systems, and Computers.* IEEE. 2017, pp. 915–919.

[8] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[9] Timothy O’Shea and Nathan West. “Arnold, Ludovic and Rebecchi, Sébastien and Chevallier, Sylvain and Paugam-Moisy, Hélène”. In: *Proceedings of the GNU Radio Conference* 1.1 (2016). URL: `https://pubs.gnuradio.org/index.php/grcon/article/view/11`.

[10] Timothy O’Shea and Nathan West. “How Deep Learning Can Help Prevent Financial Fraud”. In: *Proceedings of the GNU Radio Conference* 1.1 (2016). URL: `https://pubs.gnuradio.org/index.php/grcon/article/view/11`.

[11] Timothy O'Shea and Nathan West. "Radio Machine Learning Dataset Generation with GNU Radio". In: *Proceedings of the GNU Radio Conference* 1.1 (2016). URL: `https://pubs.gnuradio.org/index.php/grcon/article/view/11`.

[12] Timothy O'Shea and Nathan West. "What Is Deep Learning?: How It Works, Techniques Applications". In: *Proceedings of the GNU Radio Conference* 1.1 (2016). URL: `https://pubs.gnuradio.org/index.php/grcon/article/view/11`.

[13] Timothy J O'Shea, Johnathan Corgan, and T Charles Clancy. "Convolutional radio modulation recognition networks". In: *International conference on engineering applications of neural networks*. Springer. 2016, pp. 213–226.

[14] Sreeraj Rajendran et al. "Deep learning models for wireless signal classification with distributed low-cost spectrum sensors". In: *IEEE Transactions on Cognitive Communications and Networking* 4.3 (2018), pp. 433–445.

[15] Sharan Ramjee et al. "Fast deep learning for automatic modulation classification". In: *arXiv preprint arXiv:1901.05850* (2019).

[16] Thomas Warren Rondeau. "Application of artificial intelligence to wireless communications". PhD thesis. Virginia Tech, 2007.

[17] Tara N. Sainath et al. "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks". In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2015). DOI: `10.1109/icassp.2015.7178838`.

[18] *Understanding LSTM Networks*. URL: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[19] Nathan E West and Tim Oshea. "Deep architectures for modulation recognition". In: *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)* (2017). DOI: `10.1109/dyspan.2017.7920754`.

[20] Rikiya Yamashita et al. "Convolutional neural networks: an overview and application in radiology". In: *Insights into imaging* 9.4 (2018), pp. 611–629.

[21] Duona Zhang et al. "Automatic modulation classification based on deep learning for unmanned aerial vehicles". In: *Sensors* 18.3 (2018), p. 924.

[22] Zufan Zhang et al. "Automatic Modulation Classification Using CNN-LSTM Based Dual-Stream Structure". In: *IEEE Transactions on Vehicular Technology* 69.11 (2020), pp. 13521–13531.