A COMPARATIVE ANALYSIS OF NEURAL-BASED VISUAL RECOGNISERS FOR SPEECH ACTIVITY DETECTION

UNIVERSITY OF LINCOLN

Sajjadali Raza

MSc by Research - Computer Science

December 2020

Acknowledgements

The first and biggest thanks go to the supervisors for this project: Dr Heriberto Cuayáhuitl and Dr Mark Doughty. The support and guidance from the supervisors since the beginning of the project have been exceptional. In particular, Dr Heriberto Cuayáhuitl, who has been encouraging, supportive and provided guidance throughout the project. Without him, the project would not have been a success.

Second appreciation goes to the friends and colleagues in the research offices at INB. They have provided support, and encouragement in difficult times, but also provided assurance and motivation in accomplishing this project.

Last but not least, my gratitude goes to my family who have been encouraging and supportive in every single way throughout my life.

Abstract

Recent advances in Neural network has offered great solutions to automation of various detections including speech activity detection (SAD). However, existing literature on SAD highlights different approaches within neural networks, but do not provide a comprehensive comparison of the approaches. This is important because such neural approaches often require hardware-intensive resources.

As a result, the project provides a comparative analysis of three different approaches: classification with still images (CNN), classification based on previous images (CRNN) and classification based on a sequence of images (Seq2Seq). The project aims to find a modest approach-one that provides the highest accuracy but yet does not require expensive computation whilst providing the quickest output prediction times. Such approach can then be adapted for real-time application such as activation of infotainment systems or interactive robots etc.

Results show that within the problem domain (dataset, resources etc.) the use of still images can achieve an accuracy of 97% for SAD. With the addition of RNN, the classification accuracy is increased further by 2%, as both architectures (classification based on previous images and classification of a sequence of images) achieve 99% classification accuracy.

These results show that the use of history/previous images improves accuracy compared to the use of still images. Furthermore, with the RNNs ability of memory, the network can be defined smaller which results in quicker training and prediction times. Experiments also showed that CRNN is almost as accurate as the Seq2Seq architecture (99.1% vs 99.6% classification accuracy, respectively) but faster to train (326s vs 761s per epoch) and 28% faster output predictions (3.7s vs 5.19s per prediction). These results indicate that the CRNN can be a suitable choice for real-time application such as activation of infotainment systems based on classification accuracy, training and prediction times.

Table of Contents

1 – Introduction

A recent study by RAC Limited (The RAC) suggests that an estimated 9.2 million drivers still use mobile phones whilst operating their vehicle, considering the higher penalties for such actions introduced in 2017. Even with bans and laws that state against such actions, the study showed that 39% confess to checking emails, text and social media, 29% confessed in writing a text and social media and 16% admitted in taking photos or videos (albeit whilst the car was stationary in traffic) (RoSPA, 2018). Another study (in the United States) showed that using a mobile device whilst driving is 6 times more likely to cause an accident compared to driving under the influence of alcohol, leading to 1.6 million crashes each year (Snyder & Associates LLC, 2019).

Various manufacturers have attempted to tackle this phenomenon over the decades such as introducing hands-free calling and more functionalities with touch interfaces for the head unit/infotainment system. However, this yielded not a great success because "consumers have become enamoured by the breadth, variety and timeliness of the information that they get on their phone" and are "expecting the same level of information in a vehicle". In some cases, "they want the same display and choices built into the car" (Greengard, 2015, 18). Furthermore, each manufacturer has different means of accessing and operating the infotainment systems. Moreover, they may not necessarily share the same features and operate differently, which requires the consumer to learn the infotainment system as a consequence. This causes frustration for the consumer and means that the consumer would have to get used to operating their infotainment systems. As a consequence, consumers either refrain from using the features from the infotainment systems or would use their mobile phones due to convenience and ease of use.

In 2015, Android Auto (Google, 2020) and Apple CarPlay (Apple, 2020) saw their software being released in vehicles and as time went on, more manufactures opted for this approach which would leave the consumer to choose which system they would prefer. The idea of this was simple, the manufacturer would manufacture vehicles with the head unit and hardware (buttons, touchscreen etc) with some basic functionality, while Apple and Google would focus more on the software aspect of the infotainment systems. These systems are simplified versions of Android and IOS smartphones but are also empowered with their popular assistants (Google Assistant and Siri). Furthermore, they allow consumers to call, text, read messages, navigate and control music from consumer's preferred choice (i.e.

Spotify, Deezer, iTunes etc) whilst controlling the notifications to reduce distractions. Google in particular also offers Android Auto as an app that can be downloaded and used in cars (Google, 2020).

Greengard (2015, 18) states “there is a huge challenge associated with providing a driver with the right amount of information at the right time. You don’t want to overwhelm a driver or have someone get to the point where they are distracted or tuning out crucial information”.

For the consumer, the availability of such a system is useful as it means using the infotainment system that is very much like their phone. Which also means that the consumer does not need to learn how to access or operate the infotainment system. As anyone comfortable with their phone should be comfortable with their interface (Greengard, 2015). These systems offer continuity of the mobile phone experience and enable the consumers to “carry smartphone into cars and integrate these systems and controls seamlessly” (Greengard, 2015, 18).

Research by Strayer et al. (2019) found that Android Auto and CarPlay offered more functionalities and resulted in lower workload (on operating these systems) compared to manufacturer’s infotainment system. A comprehensive and qualitative study by Oviedo-Trespalacios et al. (2020) found that using Android Auto and CarPlay did reduce the need to use mobile phones and consequently reduces distractions.

However, both of these studies found that these systems can be improved. In a study by Strayer et al. (2019) participants found that some tasks with Android Auto and CarPlay were more demanding (such as the use of touch interfaces). While Oviedo-Trespalacios et al., (2020) concluded that these systems can still be improved in particular with the activation of these systems.

To activate/access these systems, two options are offered: using the touch screen or by saying the keyword i.e. “Hey Google” or “Hey Siri”. However, touch screens have a proven higher risk compared to a button or keypad interface due to the visual feedback. It provides an uncertainty if the correct option or key has not been selected (Holstein et al., 2015). This is why both of these systems offer an assistant where the subject can speak, instead of the physical interaction with the infotainment system. However, the second activation (using speech) can be also affected by various aspects such as internet connection and

background noise (e.g. Noise via speech or music). Through the app, the distance between the mobile device and the user, and disturbances in or outside the vehicle can induce noise. All of these issues can affect the activation of accessing/operating the system.

Artificial intelligence (AI) particularly neural networks, have become increasingly popular. Predominantly because they are designed around the human brain. Similar to humans, neural networks can learn through experiences, and apply this knowledge to new experiences without any supervision. This type of technology is already being used for various tasks such as speech recognition, the voice assistant or even the search engine used by the assistant. Neural Networks are not limited just to speech or text but can also be applied to images or videos.

As a result, this thesis considers using neural networks for the activation of the infotainment system. In this proposed work, the activation of the infotainment system would be for the subject to move their lips (i.e. to speak) and the system would detect whether to be in assistance of the subject. Application of such a system allows for more natural/conversational-like assistance rather than the current activation of these systems. Thus, eliminating physical interaction(touch) or saying the keyword ("hey Google") each time the subject requires any assistant which in return could also reduce distractions. The proposed system could also be applied in different tasks or scenarios such as speech recognition in robotics, where the proposed system can aid a robot in communicating with a user, or in healthcare industries to aid those who have difficulties in speaking.

Aim and Objectives:

The aim of this project is to create an artefact using 2-D images to detect when someone is speaking or not. To achieve the aim, the following objectives are proposed.

1) Collect or gather a dataset that can be manipulated and utilised for speech detection.
2) Implement the artefact using different methods with efficient training, based on the recent literature.
3) Consider different modes of classification (when one is speaking or not) using still images, classification based on previous images and sequence of images.
4) Investigate which way performs better in terms of accuracy and prediction times – still images, or sequences of images.

The contribution of this thesis is an investigation of different neural-based methods in which visual-SAD is applied. The experimental results based on VidTIMIT show that classifications based on still images (CNN) can achieve an accuracy of 97%. The inclusion of RNN further increases the accuracy, as classification based on previous images (CRNN) and sequences of images (Seq2Seq/encoder-decoder) achieve 99% accuracy. Results show that the increase of sequence size has a positive impact on classification accuracy. The simpler architecture (CRNN) is marginally outperformed by Seq2Seq model (99.1% vs 99.6% accuracy). However, the CRNN model is 57% faster to train (326s vs 761s per epoch) and 28% faster in output predictions (3.7s vs 5.19s per prediction). Further results show that use of existing CNN models can not only affect the accuracy but the training and prediction times within the context of SAD.

The rest of the thesis is structured as follows. Chapter 2 – Background discusses the background of neural networks. Chapter 3 – Literature Review provides existing work in the field of SAD. Chapter 4 – Visual Speech Detection provides the experimental design/setting. Chapter 5 – CNN for Still Images and chapter 6 – CNN for Image History and Sequence compares the respective approaches for SAD. Chapter 7 – Conclusion and Future Work concludes with future work highlighted.

2 – Background

This chapter discusses the background knowledge into neural networks and how they can be applied to the problem in question; speech detection in the context of infotainment systems. Therefore, the chapter introduces various methods of speech detection to analyse and differentiate how speech can be recognised. This allows for an understanding of the most appropriate method to address the problem in question. Furthermore, the chapter explores what neural networks are, how they operate and the infrastructure required when implementing a network. This includes the various network types that can be considered and how they can be evaluated to assess their success in solving the problem. Lastly, popular techniques to improve the network are examined.

2.1 – Speech Detection

Speech detection is a method of detecting the absence or presence of human speech. Speech detections aim to differentiate speech and non-speech and thus is widely applied in various fields. In the context of infotainment systems, speech can be detected in various ways.

Physical Interaction

One of the most common way speech is identified (i.e. the user is speaking/saying a command) especially in vehicles is through the button/keypad or touch interface. In this case, the driver physically presses a button or touches a certain icon which then triggers the system to listen for a command/speech. This may be a button which triggers the manufacturer's built-in system to activate or an icon to touch if Android Auto or Apple CarPlay is utilised.

Keyword Detection

A common feature found with likes Android Auto or Apple CarPlay is the ability to obtain assistance after saying a hot word or keyword such as "Hey Google" or "Hey Siri". In this scenario, the system listens for the keyword and then activates the system to listen for the command from the driver. In contrast to the physical interaction, this would not require the driver to physically interact with the system. Instead, the user says the keyword to interact with the system which reduces distractions for the driver.

Speech Activity Detection

Speech Activity Detection (SAD) or also known as VAD (Voice Activity Detection) is the process of detecting speech though audio or sound. In this case, the audio or sound is converted to graphical images such as spectrograms, which are then analysed and studied to differentiate in speech and non-speech (i.e. noise). SAD is popularly used within the field of speech processing including speech coding, Automatic Speech Recognition (ASR) and speech enhancements. In SAD, there is no requirement for a keyword but rather speech is automatically analysed with noise being filtered to provide a smoother interaction. Whereas with keyword detection, each time one requires assistance the keyword needs to be said.

Video-Based Speech Activity Detection

Similar to SAD, video-based SAD is the process of detecting speech but through images or videos. In this scenario, facial images are studied to differentiate between speech and non-speech. These images are segmented into particular features which are then studied to understand the differences of those features between speech and non-speech. Depending on the resources and the functionality required, current literature highlights various approaches of utilising SAD either as using audio or video alone or as a combination of both. Arguably, with the use of audio or video alone, there are limitations for either approach but opting for a multimodal (audio and video) approach can also require extensive resources. Noise in the surrounding area or multiple voices can affect the performance SAD (audio) whereas V-SAD focuses on facial images and thus, surrounding noise or multiple voices do not affect it.

2.2 – Neural Networks

A neural network is a mechanism consisting of a series of algorithms whose functionality is loosely based on the human brain. These algorithms allow the network to recognise patterns in numerical form (i.e. vectors) into which various data such as images, sound, text and time series can be translated. Like the human brain, the network comprises of interconnected neurons who take in some information, apply a function and pass the output to the next neuron. As a result, these networks can learn from experiences and provide an output based on those experiences.

Figure 2.1 represents the algorithm behind one of the many neurons in a neural network. In this case, the neuron takes two inputs x_1 and x_2. The individual input is multiplied by its weight W which is then added together with bias b. This value is then passed through an activation function which outputs a value that is sent to the next neuron z.

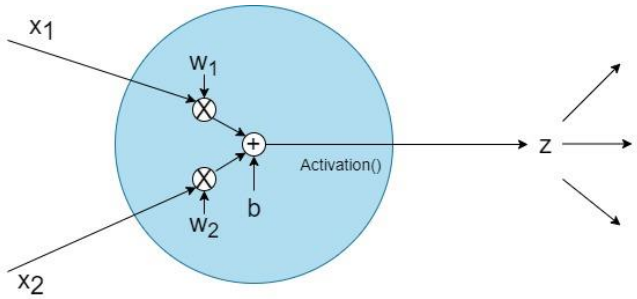


Figure 2.1: Design of the hidden unit

As demonstrated in Figure 2.1, information (in this case) is passed forwards which is known as a forward pass or forward propagation. This type of network is described as a feedforward network. A Feedforward network, as described in Figure 2.2, typically consists of 3 layers. The first layer is often referred to as an input layer which processes the input (such an image) and provides an input feature vector. The second layer computes the input feature and provides the computation for the next layer for a response. The last layer is typically an output layer which uses the information from the hidden layers to provide a meaningful response/output. Each layer may consist of single or multiple neurons depending on the problem domain and computational resources etc. The strength of the connection between each neuron is referred to as weight.

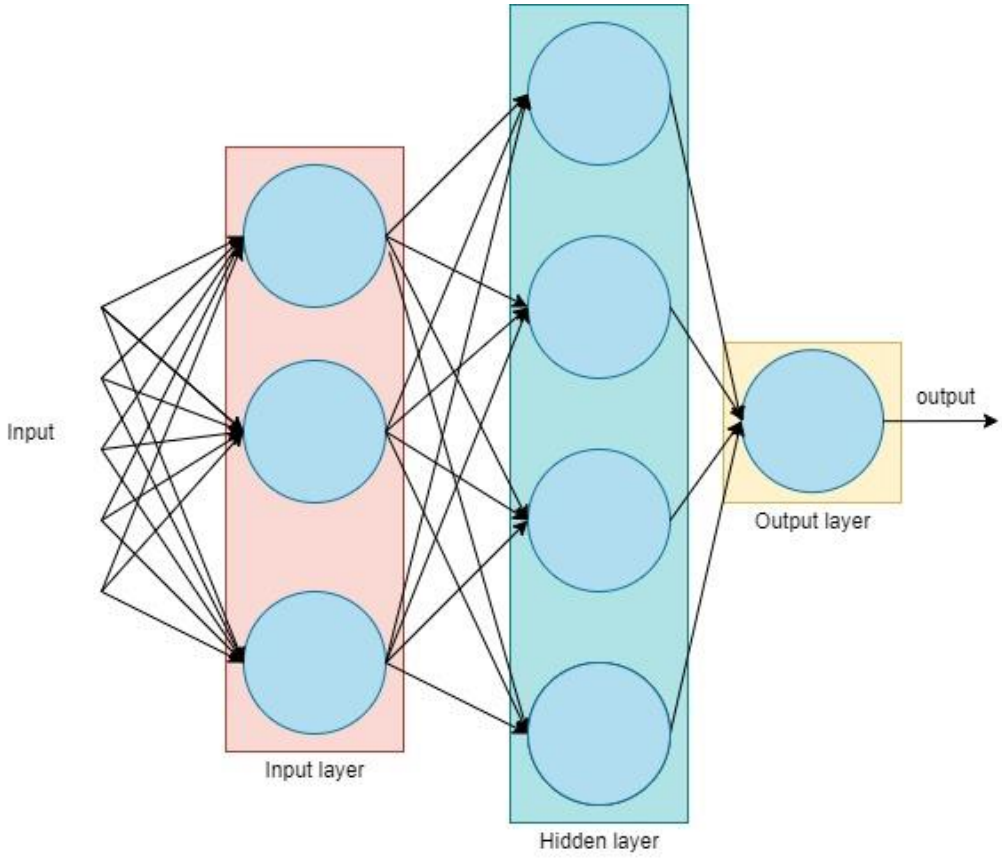


Figure 2.2: Architecture of Feedforward Network

The task of the feedforward network is to learn an approximation function that produces the target output based on the input provided. Thus, it can be denoted as $y = f(x; \theta)$ where the network learns the parameter θ and applies to the input x to result in the best function approximation (Goodfellow et al., 2016). To train the network, various training algorithms can be used. However, the most popular one is the backpropagation introduced by Rumelhart et al. (1986). During training, an error is calculated based on the target output and the output from the network. Backpropagation is a method that uses the error, goes backwards from the output to the input layers and updates the weights to reduce the error (Rojas, 1996). As a result, the network is trained by an application of forward propagation and backpropagation. Typically, a forward and back pass act as one iteration. The iteration is repeated until an epoch is completed, which is based on the batch size and training samples. The network is trained until set epochs or desires loss is achieved (Rojas, 1996).

2.3 – Activations

Activation functions are equations that determine the output of the neuron. Activation functions are attached to each neuron in a network and regulates whether it should be activated or not. The function calculates the weighted sum of the inputs and normalizes the output to range between 0 and 1 or -1 and 1.

Sigmoid

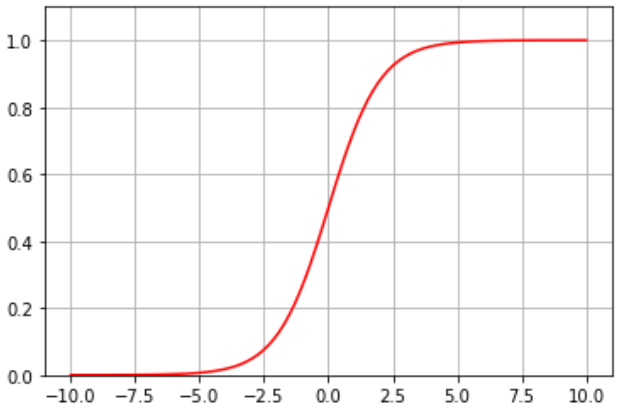


Figure 2.3: Graph of the sigmoid function

Sigmoid is a nonlinear activation function known as the logistic function which transforms the input so the output ranges between 0 and 1 and thus its denoted as:

$$\sigma(x) = \frac{1}{1 + exp^{-x}}$$

(1) Goodfellow et al. (2016, 67).

Use of sigmoid function can be used in a binary classification where the output must be between the two classes. Sigmoid can also be used to determine probabilities (as probabilities lie between 0 and 1), and for memory allocation in LSTM's or GRU (described in section 2.9 – RNN).

Softmax

Softmax is another logistic function similar to sigmoid, however, softmax is used to represent a probability distributed over discrete variables with n classes. Consequently, it is commonly used in the output layer for classification in cases where there are multiple target classes. Softmax combines the inputs so the sum of the output equates to 1 as well as each input is transformed so the value ranges from 0 to 1. Softmax is calculated as:

$$f(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

(2) Goodfellow et al. (2016, 81)

ReLU

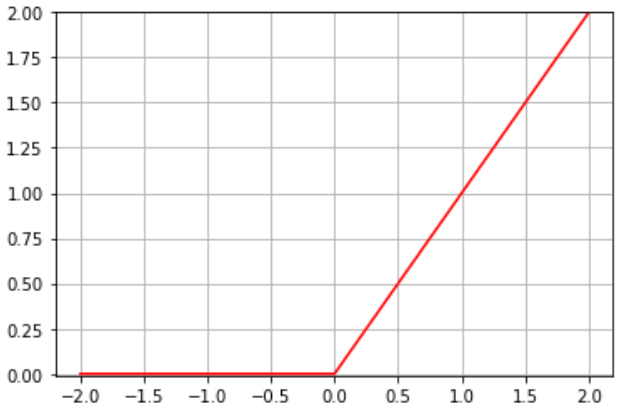


Figure 2.4: Graph of the Rectified Linear Function (ReLU)

The rectified linear activation function (ReLU) is a linear function that outputs the input if the input is positive, otherwise outputs zero. Thus, it is denoted as:

$$ReLU(x) = \max(x, 0)$$

(3) Goodfellow et al. (2016, 193).

ReLU is a piecewise linear activation function; however, it acts like a nonlinear function that allows complex relationships in data to be learned. ReLU is also computationally inexpensive as other functions such as tanh, which requires an exponential equation. As a result, deep convolutional neural networks with ReLU train several times faster than other activations such as tanh. Hence, it is the recommended default activation function recommended for most feedforward networks (Goodfellow et al., 2016).

2.4 – CNN

Convolutional Neural Network (CNN) is a type of Feedforward network which is specialized for processing data that is in a grid-like structure (images, sound, or timeseries-data). The name convolutional neural network derives from the mathematical term convolution. In a CNN, images are read in small squares(kernels), where parts of the image are studied. These convolutions allow detection of small meaningful features such as edges, shapes and texture with kernels that have only tens or hundreds of pixels. Thus, allowing the network to distinguish between images (classification), cluster images by similarity (photo search) or object recognition. As the kernel sizes for convolution are much smaller than other operations, convolution has much fewer parameters leading to less required memory. Convolutions have also proven to be efficient as it computes the output features with fewer operations (Goodfellow et al., 2016).

The operation of convolution can be described as:

$$A_j = f(\sum_{i=1}^{N} (I_i * K_{i,j} + B_j)$$

(4) Li et al. (2014, 845)

Where each input I_i is convoluted with the corresponding kernel $K_{i,j}$. The total of all convoluted matrices is computed and bias B_j is added to each element of the resulting matrix. Lastly, the non-linear activation (ReLU) is applied to each element of the previous matrix to produce one output matrix A_j.

CNN's use supervised learning algorithm in which the data provided is labelled based on the type of data such as images corresponding to speech or non-speech. Apart from providing the data and the network architecture (number of layers, kernel sizes, optimisers etc), the network can learn and distinguish without any further supervision. Therefore, making CNN's a very powerful tool within computer vision.

The popularity of CNN increased particularly with the ImageNet Large Scale Visual Recognition Challenge – ILSVRC (Russakovsky et al., 2015). The ILSVRC introduced various models such as AlexNet (Krizhevsky et al., 2012), VGG - Visual Geometry Group (Simonyan and Zisserman, 2015), ResNet (He et al., 2016) and Xception (Chollet, 2017a) which changed the way the CNN's a are designed. For example, AlexNet introduced the concept of applying max pooling after convolutional layer, a concept that is being followed to this

day. On the other hand, VGG introduced smaller filters for convolutions (3x3 with the stride of 1) with more depth (number of layers) to achieve better accuracy. These models achieved great accuracy in the ILSVRC and are being exploited in various ways. In some cases, these models are used to extract information(embeddings) regarding images (Ariav and Cohen (2019) using ResNet embeddings) whilst in other cases, networks are designed following the architecture of these models (at a smaller scale) yet achieving reasonable accuracy (Cornu and Milner, 2015) with the deployment of smaller VGG-like model).

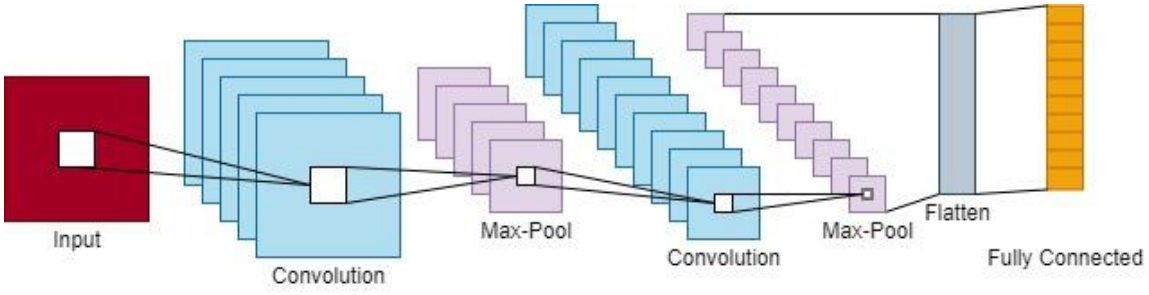


Figure 2.5: An example of CNN architecture

Figure 2.5 shows the architecture of CNN. The network consists of an input layer, convolutional layer, downsampling layer, and fully connected layer. The convolutional layer extracts local information from the input and adjusts the weights based on the activations. After every forward pass, the convolution layer generates a feature map. The features are trained to activate feature maps when patterns of interest are observed in the input. To make the network efficient and size of the network smaller, these features are often downsampled via max-pooling (choosing the maximum value of the feature in a region) and fed to the next layer. With the reduced features, they are fed to another convolutional layer to extract further information. Features from this layer are downsampled again before being fed to a fully connected layer, which combines the features and uses a non-linear output layer such as “softmax”. In this case, the classification is generated by associating probabilities for the relevant classes/labels, values of which lie between 0 and 1 (Sehgal and Kehtarnavaz, 2018).

2.5 – Pooling Layer

As discussed previously, convolutions learn the image and produce a representation (feature map) which summarises the presence of features. However, as they record every feature at every location, a small change/movement in the image (i.e. Cropping, rotation) can result in a different feature map. As a result, pooling is often used after a convolution layer to summarise the representation learnt by the convolution. Pooling replaces the output feature (obtained in convolution) with a summary of present features in patches of the feature map. Two popular pooling operations used in CNN's are average pooling and max pooling. In average pooling, the average value for each patch (window) of the feature map is calculated. Whereas in max pooling, the maximum value for each window is calculated.

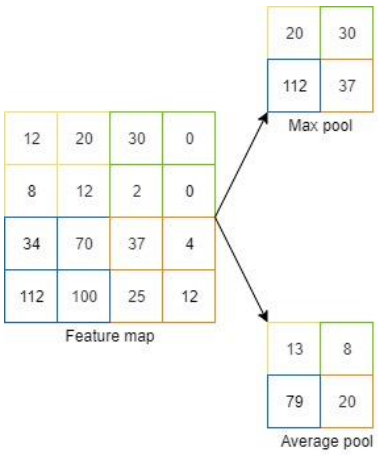


Figure 2.6: Comparison of average and max pooling.

Figure 2.6 shows the comparison of the output between the two pooling operations (max pooling and average pooling). In CNNs, the popular practice is to apply max pooling as opposed to average pooling, as it is more important to look at the maximum presence instead of the average presence of the feature (Chollet, 2017b). Similar to the convolution operation, filter size and stride also need to be defined for max pooling. Chollet (2017b) suggests the strategy for max-pooling is to apply 2x2 filter with a stride of 2 which allows the feature to be downsampled by the power of 2. This particular approach is mostly used particularly for CNNs.

2.6 – Fully Connected Layer

In a CNN fully connected (FC) layer is essentially a feedforward network as described in section 2.2 – Neural Networks. FC layer is responsible for classifying the input images into various classes. As features are passed on from subsampling (max pooling), the features are flattened to a one-dimensional vector (as the features from the max pool are 3D). This vector is then analysed and weights are applied to predict the correct label/class based on the training dataset. The output of this layer is typically a softmax for multiclass classification which provides the probability of the classes to the sum of 1. For binary classification, sigmoid activation is used which outputs 0 or 1 depending on the arrangement of the labels. The output layer also has a loss function such as categorical cross-entropy for multi-classification or binary cross-entropy for binary classification. The loss computes the prediction error. Once the forward pass is completed the backpropagation begins which updates the weights and biases for loss reduction. Thus, the application of the FC layer also makes the network end-to-end trainable (Goodfellow et al., 2016; Tummala, 2019).

As described in section 2.2 – Neural Networks, in the FC layer each neuron calculates the following:

$$y = g(Wx + b)$$

(5) Goodfellow et al. (2016, 196)

Where x is the input vector, W is the weight, b is the bias vector and g is the activation function such as ReLU. In the final layer, for classification, a squashing function (such as softmax) is applied to obtain probabilities for the labels/classes.

2.7 – Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is a technique used to optimize a neural network. Typically, in a neural network during training, the distribution of each layer's input is affected, as the parameters of previous layers change. The change in parameters can slow down the training of the network, which would require a lower learning rate and careful parameter initialization as a result. This makes it difficult to train models with saturating nonlinearities or squashing functions such as softmax. Batch normalization provides a way to reparametrize a network by performing normalization for each mini-batch in training (Goodfellow et al., 2016), thus, coordinating the update of multiple layers in the model. Batch Normalization can be denoted as:

$$H' = \frac{H - \mu}{\sigma}$$

(6) Goodfellow et al. (2016, 318)

Where H' replaces H – to normalize the minibatch of activations. μ is the mean of each unit and σ contains the standard deviation for each unit. Therefore, batch normalization can be deployed to any input or hidden layer in a network. For CNN, batch normalization allows the network to learn quicker, mainly due to the reduced problem of coordinating across the layers, which also reduces the number of epochs required. Ioffe and Szegedy (2015) add batch normalization before the activation to achieve stable distribution of activation values which resulted in accelerated training and better accuracy on the Inception architecture (Szegedy et al., 2015).

2.8 – Dropout

Dropout (Srivastava et al., 2014) is a method of regularising a network, particularly used to reduce overfitting. In simple terms, overfitting occurs when network performs better during training than with unseen data (test data). Dropout temporarily removes the neurons in a network along with incoming and outgoing connections. By setting each output of the hidden neuron to zero with probability p of occurrence, these neurons are dropped out during training and therefore do not participate in a forward pass or backpropagation. As these neurons are dropped at random, in every input, the network samples a different architecture (much like in Figure 2.7) whilst still sharing the weights. The change of sampled architecture reduces the reliance of a neuron on the presence of other neurons (complex adaptation), thus forcing the network to learn more robust features in conjunction with other random subsets of neurons. At test time, no dropout is applied and all neurons are used. However, the outputs from the neurons are multiplied by $1 - p$. Srivastava et al. (2014) suggest dropout of $p = 0.5$ for hidden layers. However, they further suggest that p should be closer to 1 for input units.

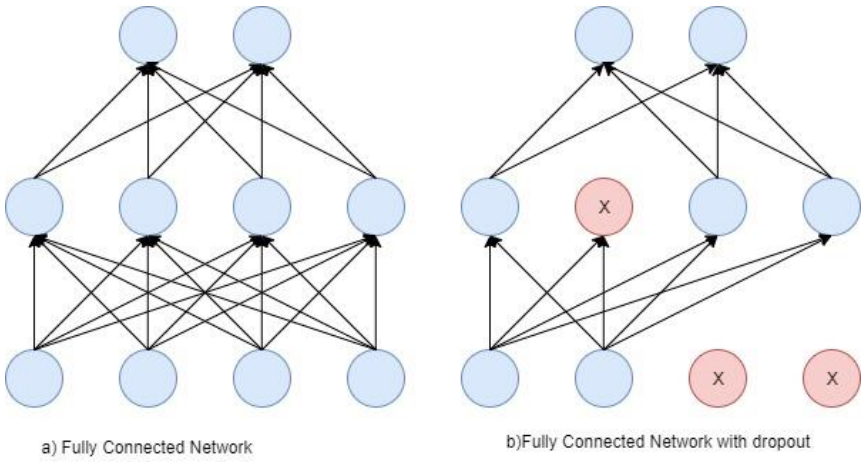


Figure 2.7: Difference in a network with dropout. a) A fully connected network. b) The same network but after dropout is applied.

2.9 – RNN

Unlike CNNs where information flows forward (to the next state/neuron), in RNNs the connection of neuron can be to itself and the next state, thus storing information that is sent to the next neuron. RNNs use the memory to store information on what has been computed. The input and stored information are combined to update their hidden units containing the history of previous timesteps (elements), whilst passing the information to the next state, thus allowing the network to learn from its past. As a result, an RNN is often used for processing sequential data as it's able to adjust itself based on its history. RNNs are popularly used in machine translation, text classification and speech recognition. They can also be used to study grid-like structures such as images in a time series, provided that the entire sequence is introduced to the network. As a consequence, they can also be used for handwriting generation and image captioning.

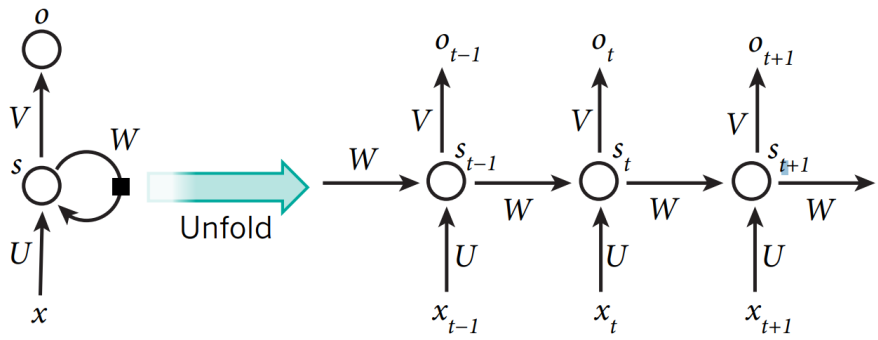


Figure 2.8: An RNN with the unfolding of the RNN during computation (LeCun et al., 2015, 442).

Figure 2.8 shows the unfolding of the RNN in computation. In this case, the hidden units under node s with values $s_t$at time t, get the input from other neurons at previous timesteps (represented with black square). In this way, the RNN can map input sequence with elements x_t into an output sequence with elements o_t. Value at o_t is dependent on $x_{t-1\ldots.t-n}$. Parameters (U, V, W) are shared across every time step and can be seen as a deep feedforward network (LeCun et al., 2015).

Figure 2.8 can also be defined as:

$$s_t = f(Ux_t + Ws_{t-1})$$

(7) Lim et al. (2016, 2)

Where x_t is the input at timestep t, s_t is the hidden state at timestep t, o_t is the output at timestep t. As noted, each element of the output $o_t$is a function of Vs_t. Each element of

the output is produced using the same output rule applied to previous outputs, thus the recurrent formulation results in sharing of parameters (Lim et al., 2016).

LSTM

In recurrent networks, as the timestep goes on information is stored. However, in deep recurrent networks, storing such information can be computationally expensive and also lead to "vanishing" or "exploding gradients". The exploding gradient occurs when the value of weights become increasingly large as the timestamps go on. In other cases, the value of the weights is too small and as they go through continuous matrix multiplications, the values decrease in size making it impossible for a network to learn (known as vanishing gradient). Consequently, they both result in an unstable network.

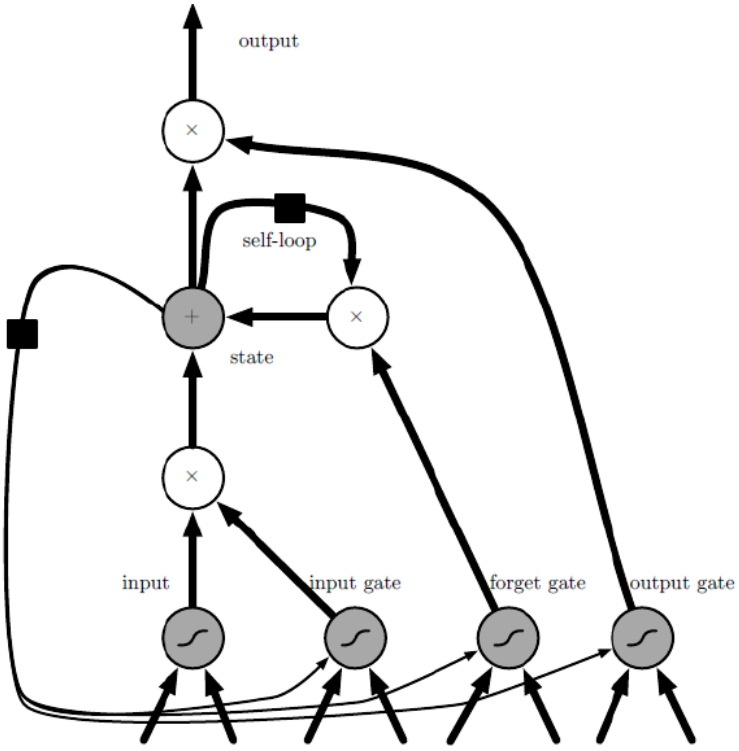


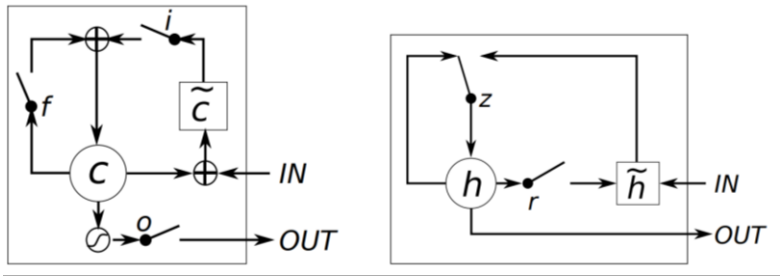
Figure 2.9: Architecture of LSTM (Goodfellow et al., 2016, 409)

LSTMs were created as a solution for the issues of vanishing and exploding gradients. LSTMs provide internal states called gates which control the flow of information. With these gates, information can either be allowed or removed depending on the gate. An input feature is computed on the regular neuron but the value can only be passed into the state if the input gate permits. The state unit has connections to the next time step and therefore copies its own value and accumulates the external signal. The weight of the state is controlled by the forget gate, which learns to decide when to clear the content of the memory. Similarly, the output gate controls the output of the cell. All gates have sigmoid functions whilst the

input gate can have squashing non-linearity such as sigmoid or tanh. The state can also provide input to the next time step, resulting in two sources for the input-the present, and the recent past. (Goodfellow et al., 2016)

GRU

Gated recurrent units (GRU) (Cho et al., 2014) are similar to LSTMs in the sense that they also have gates in which the flow of information is controlled. However, unlike LSTM they do not have a memory cell. The number of gates in a GRU is reduced as it only has an update and reset gate. These gates control the forgetting and updating of the state unit. The reset and update gates can individually "ignore" parts of the state vector. The update gate can choose to copy information if the values lie closer to 1 or ignore the information closer to 0 and replace it with new "target state". The reset gate controls which part of the state gets used to compute the next target state. The activation of the gates in GRU relies on information from current input and previous output. This results in fewer parameters which lead to quicker computation (Goodfellow et al.,2016; Tang et al., 2016). Goodfellow et al. (2016) argue that LSTM and GRU provide similar/comparable results. However, Tang et al. (2016) found the GRU to be quicker as it has fewer parameters and better in performance (accuracy) at detecting questions in conversational speech.



a) LSTM b) GRU

Figure 2.10: comparison of LSTM and GRU cells (Cho et al., 2014, 1726).

Figure 2.10 shows a comparison of LSTM and GRU cells. Where for (a) i, f, o are the input, forget and output gate respectively. c denotes the memory cell and $\tilde{c}$ denotes the content of new memory. For (b) r and z are the reset and update gate, whilst h and $\tilde{h}$ are the activation and candidate activations (Cho et al., 2014).

Encoder-Decoder (sequence to sequence)

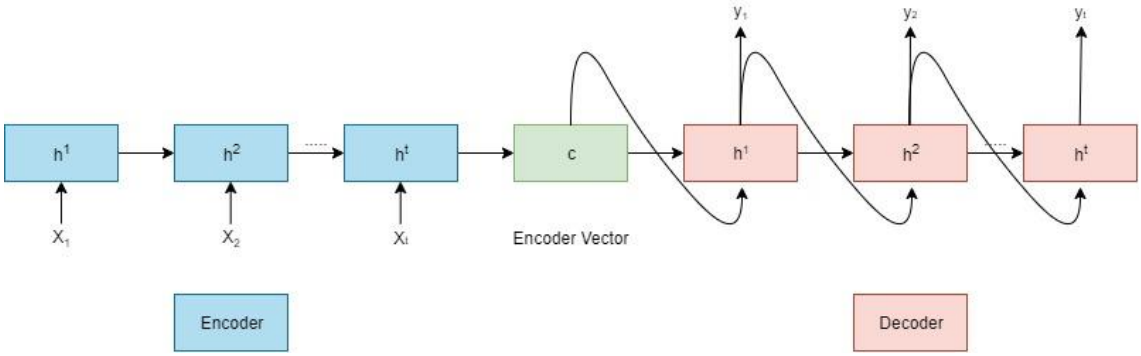


Figure 2.11: Encoder-decode architecture.

Figure 2.11 represents the encoder-decoder architecture. Encoder-decoder is a type of neural network which compromises of two RNNs. An encoder learns the input (x) one timestep (t) at a time and obtains a fixed-length vector representation (c). On the other hand, the decoder extracts the output (y) from the vector (c). The two networks are trained as a combination to maximise the conditional probability of the target sequence given the input sequence. The encoder-decoder architecture allows the network to learn sequences where the input is not the same length as the output. As a consequence, the encoder-decoder architecture is often used for machine translation, speech recognition and question-answer systems (Cho et al., 2014).

The encoder reads the input sequence x sequentially and updates its state based on:

$$h_{(t)} = f(h_{(t-1)}, x_t)$$

(8) Cho et al. (2014, 1725)

Where f is a non-linear activation function such as sigmoid or an LSTM, h represents the hidden state at timestep t with x denoting the variable-length sequence. After reading the entire sequence, the hidden state of the RNN is c (in Figure 2.11) which holds a representation of the entire sequence.

The decoder updates its state similar to the encoder but is conditioned on output y_{t-1} and the entire sequence representation c. Thus, the hidden state of the decoder is denoted as:

$$h_{(t)} = f(h_{(t-1)}, y_{t-1}, c)$$

(9) Cho et al. (2014, 1725)

The probability of next output (y) from the decoder is:

$$g(h_{(t)}, y_{t-1}, c)$$

(10) Cho et al. (2014, 1725)

Where g is the activation function such as softmax, as it produces valid probabilities.

The encoder-decoder architecture was introduced by Sutskever et al. (2014) as a sequence to sequence learning for machine translation. Sutskever et al. (2014) suggested that LSTMs could be used for machine translation where one LSTM can perform the encoding, and the other LSTM can perform the decoding. Their architecture compromised of four layers of LSTMs with each layer consisting of 1000 units. Their architecture particularly introduces "EOS" to sign as the end of the sentence during training.

Cho et al. (2014) introduced a scoring system in which the target words are scored – the highest scoring word being used as output. Cho et al. (2014) particularly introduced such architecture as "encoder-decoder". Their proposed architecture did not use LSTMs but rather proposed a simpler recurrent unit called GRU. The architecture consisted of 1 layer of GRU with 1000 units and 500 maxout units for pooling (Goodfellow et al., 2014). Sutskever et al. (2014) found that using deeper LSTM significantly increases the performance of the model as opposed using shallow LSTMs. Whereas Cho et al. (2014) used single GRU with 1000 units whilst still achieving similar results.

Although the encoder-decoder architecture was introduced for machine translation, Vinyals et al. (2015), Venugopalan et al. (2015), Soh, (2016) and Donahue et al. (2017) adapted the encoder-decoder architecture to caption images/videos. Instead of providing sequences of text, the authors provided embeddings from popular CNN architectures such Inception (Szegedy et al., 2015) and VGG (Simonyan and Zisserman, 2015), together with embeddings for words to train the network to provide captions/descriptions. The encoder-decoder architecture in these papers has been developed in two different ways.

Vinyals et al. (2015) introduced the encoder-decoder for images where the encoding was done by the CNN and decoding by the RNN. Venugopalan et al. (2015) and Soh (2016) used the LSTMs for encoding and decoding but with the input of features from the CNN-much like the initial encoder-decoder architectures suggested by Cho et al. (2014) and Sutskever et al. (2014). Donahue et al. (2017) compared both of these models with other variants and concluded these architectures provide similar/comparable results. However, they found

the CNN encoder and RNN decoder outperforms the RNN encoder-decoder by a smaller margin.

2.10 – CNN-RNN

Convolutional Neural Network – Recurrent Neural Network (CNN-RNN) is a network with two independent networks - CNN and RNN. In this network, CNN is utilised to study images and provide image representations while the RNN studies sequential data and builds the image-label relationship. As a result, one of the popular uses of this network is the multi-label generation. In this network, images are passed to a CNN (typically, an existing CNN model) to obtain the embeddings while the RNN takes the embedding of the predicted label at each timestep and models the occurrence of the label within the hidden state (Wang et al., 2016). The RNNs can also consider the label based on the previous timestep. The output from the RNN and image representation of the CNN are formatted/concatenated where these features are studied as shown in Figure 2.12.

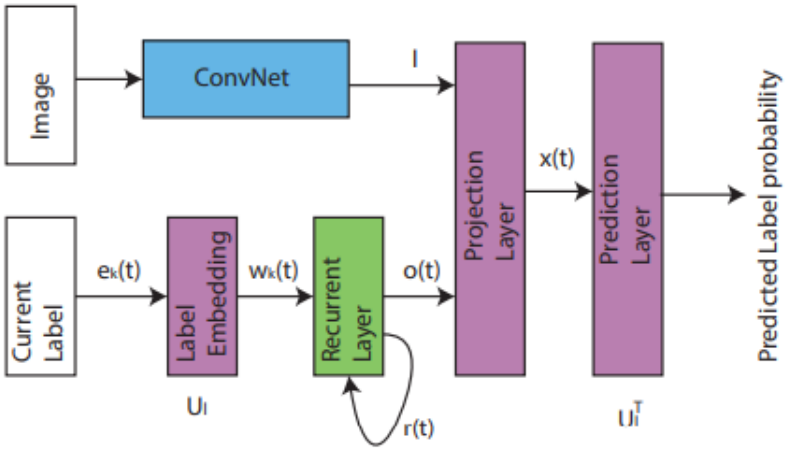


Figure 2.12: CNN-RNN architecture (Wang et al., 2016, 2288).

Often the CNN used within CNN-RNN architecture, derives from an existing CNN model (such as VGG or Inception) to provide the best representation for the image. The RNN in this case can be of various types. This can include a few layers of LSTMs/GRUs or the encoder-decoder model as discussed in the previous section. The utilization of the encoder-decoder for images can be described as a CNN-RNN architecture as the CNN and RNN are two independent networks.

2.11 – CRNN

Convolutional Recurrent Neural Network (CRNN) is a hybrid combination of the CNN and RNN architecture. In a CNN, convolution and pooling perform feature extraction between images and learn the representation of those features. However, they do not consider contextual dependencies between different images. These dependencies represent useful spatial structure information in images. RNNs on the other hand, are designed for learning contextual dependencies in sequential data by using recurrent connections. Therefore, in a CRNN, the CNN learns the images and provides a representation of the images (features), where the RNN learns to encode spatial dependencies between those image regions (Zuo et al., 2015).

In a CRNN, the outputs of a CNN are converted into spatial sequences given to RNN and the feature outputs of the RNN are treated as fully connected layers (as shown in Figure 2.13). Thus, the fully connected layers are replaced by RNN (LSTMs or GRUs). Regarding the architecture, CRNN still follows the layout of the CNN but the LSTM/GRU is replaced for the fully connected layers as shown in Figure 2.14.

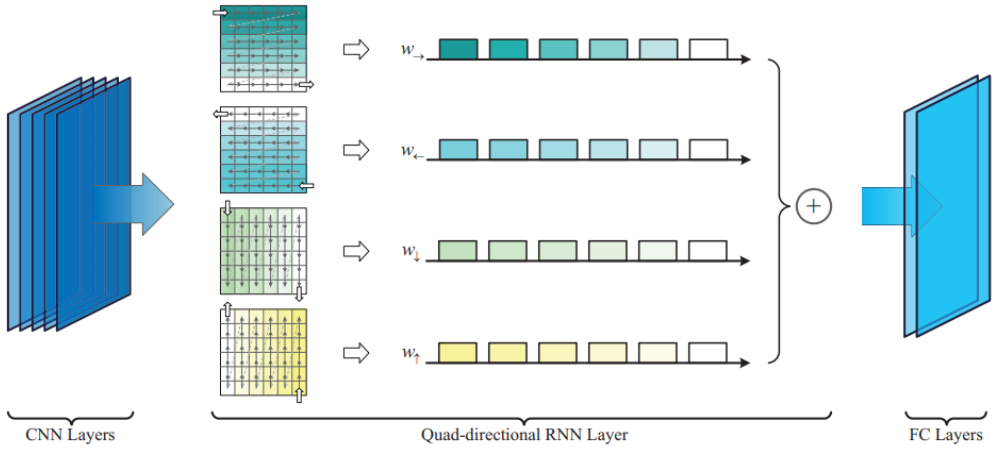


*Figure 2.13: the CRNN framework (*Zuo et al., 2015, 19).

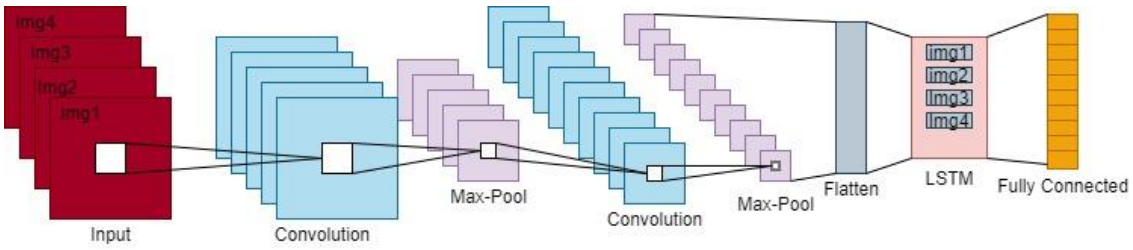


Figure 2.14: CRNN Architecture

CRNN architecture has been employed in various fields and tasks. Tao and Busso (2019), (Ariav and Cohen (2019) and Sharma et al. (2019) employed CRNN architecture for the task

of Visual Activity Detection (VAD). Similarly, the CRNN has been used for speech emotion (Lim et al., 2016), music classification (Choi et al., 2017), image-based sequence recognition (Shi et al., 2017) and for medical imaging (MRI) (Han and Kamdar, 2018).

Regarding the architecture, Shi et al. (2017) employed a smaller VGG model (using 3x3 filters similar to VGG11) with 2 bidirectional LSTM with 256 hidden units. Lim et al. (2016) and Choi et al. (2017) used 4-layered CNN with 3x3 filters with 2 layers of LSTM and GRU respectively, consisting of 1024 units. Han and Kamdar (2018) used 3 layers of CNN with 5x5 filters and single bidirectional GRU with 256 units.

Zuo et al. (2015), Lim et al. (2016), Choi et al. (2017) and Shi et al. (2017) in particular compared CNN and CRNN performances for their relevant tasks and found CRNN to be superior to CNN. Furthermore, Shi et al. (2017) suggest CRNN models can be suitable for mobile devices, as the CRNN architecture has fewer parameters compared to a CNN because CRNN models do not have fully connected layers. However, Choi et al. (2017) highlight that the CRNN can be more computationally expensive compared to CNN depending on the length of the sequence.

2.12 – Evaluation Metrics

There are various ways in which a model can be measured. While loss (error rate) can be used to measure the performance of a model, it is predominately used to adjust the network based on the error rate in predicting the right label/class. For classification, evaluation metrics are typically based on the confusion matrix.

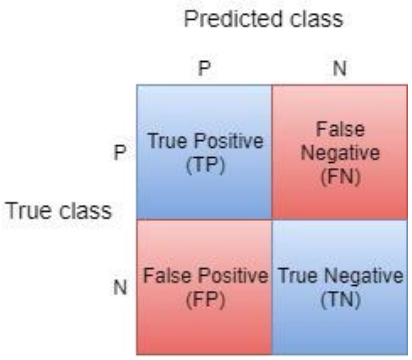


Figure 2.15: A confusion matrix summarising the predictions (assuming there are two classes)

Figure 2.15 shows a confusion matrix - a table that summarises the predictions of a model on unseen data (test data). A true positive (TP) is an outcome where the model predicts a positive class based on a positive input. Similarly, a true negative (TN) is an outcome where the network predicts a negative class based on a negative input. On the other hand, a false positive (FP) is an outcome where the network predicts a positive class for a negative input similar to a false negative (FN), which is an outcome where the network predicts a negative class for the positive input.

The popular and straightforward method to measure performance is the accuracy which considers the positive predictions based on total predictions and is denoted as:

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

(11) (Alpaydin, 2014, 562)

In some cases, accuracy may not necessarily show a true measure of a model. In cases with unbalanced data where the data for one class may be more significant than the other, this may automatically lead to a higher accuracy.

Precision and recall give a more in-depth measure of the network. Precision identifies the number of positive predictions that actually belong to the positive class. Precision can be represented as:

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

(12) Alpaydin (2014, 562)

Recall identifies the number of positive class predictions based on all positive inputs in the dataset. Thus, it is denoted as:

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

(13) Alpaydin (2014, 562)

To summarise precision and recall to one metric, F-score is often used which is useful for imbalanced data. F-Score is denoted as:

$$F = \frac{2pr}{p + r}$$

Where p and r represent precision and recall respectively.

(14) Goodfellow et al. (2016, 424)

3 – Literature Review

The task of a detecting speech is often referred to as Voice Activity Detection (VAD) or Speech Activity Detection (SAD) in the existing literature. Traditional approaches often used audio signals for SAD but in the current literature, there has arisen a choice of input used for SAD. Recent literature shows authors using video signals or a combination of both over audio signals alone.

In a SAD system via audio signals, the audio signals are often converted to graphical images (such as spectrograms) in which they are analysed and used to detect speech. However, the performance of the SAD is affected when background noise or the inclusion of multiple voices are involved.

In a noisy environment, non-speech can often be classified as speech due to an increase in noise (Cornu and Milner, 2015; Ariav and Cohen, 2019; Sharma et al., 2019). As a result, recent approaches regarding SAD involve use of video or combination of both (audio and video) to create a more robust SAD system

Furthermore, recent approaches involving SAD utilize deep neural networks (DNN) as opposed to a "static" approach or using algorithms. Due to their powerful abilities, DNN provides a system in which through feeding raw data, the system can learn differences between speech and non-speech. Non-speech activities such as cough, singing, whistling and yawn can often introduce noise within an image-based system but DNN's can learn to differentiate between such activities (Bairaju et al., 2017). In various cases, multimodal is used for better robustness of the system. Moreover, DNNs require less supervision and computation workload by utilising smaller networks (Tao and Busso, 2019; Ariav and Cohen, 2019; Sharma et al., 2019).

Table 3.1 provides an overview of the approaches for SAD in the literature in recent years. The table shows how modern DNNs (especially for SAD) rely on RNNs due to their ability to learn temporal dynamic behaviour (Ariav and Cohen, 2019; Sharma et al., 2019). Recent literature shows authors opting for a combination of CNN and RNN, as featured in Table 3.1. In this case, CNNs are used to learn image representation as they carry the ability to distinguish between images. But the addition of RNN allows the network to learn temporal information, as it introduces history and information regarding the sequence. The addition

of RNN to the network architecture also increases classification accuracy (Sharma et al., 2019).

Table 3.1: Summary of closely related works to the proposed research

Author	**Task**	**Dataset**	**Architecture**	**Accuracy**
Cornu and Milner (2015)	Voicing classification (classifying frame as speech/non-speech) in speaker-dependent and speaker-independent scenarios.	GRID	CONV (32x3x3) Max pool 2x2 Dropout 0.2 CONV (64x3x3) Max pool 2x2 Dropout 0.2 FC 512 Dropout 0.5	97.66% for speaker dependent. 74.68%. for speaker independent
Sharma et al. (2019)	Visual voice activity detection for detecting endpoint.	GRID, VidTIMIT, Indian-English (locally collected)	CONV (16x5x5), s=2 Max pool 2x2 Batch Normalization CONV 32x5x5, s=2 Max pool 2x2 Batch Normalization CONV 8x5x5, s=2 Max pool 2x2 Batch Normalization LSTM (64) x 2 CONV stride=2	96.5% for speaker dependent. 92.2% for speaker independent.
Wang and Wang (2019)	Speech activity detection using facial images via landmark pooling	Labelled Speech in the Wild (LSW) – locally collected.	CONV (64x7x7), s=2. Landmark pooling layer (20 landmarks). FC Layer. GRU (64). FC Layer.	79.9% for the proposed architecture vs 76.7% for CRNN.

Tao and Busso (2019)	Speech activity detection using multimodal recurrent neural networks	CRSS-4English-14 – locally collected.	Audio- Maxout Layers (512) x 2. LSTM x 2. Visual - 3 x CONV (64x5x5), s=2. LSTM (64) x 2. Multimodal - Feature from each modality is concatenated. LSTM (512) x 2 Maxout layer (512)	Audio: 90.3%, 92.7% Video: 60%, 65.5% Multimodal: 92.1%, 93.8%
Ariav and Cohen (2019)	VAD by incorporating audio and visual modalities.	Locally collected (Dov et al. 2015)	ResNet-18 (He et al., 2016) to extract visual features. WaveNet encoder for audio features. Features are fed to MCB for fusion. LSTM (1024) x 2. FC 1024. Dropout 0.2,0.5 -across the network. Batch Normalization - across the network.	Audio: AUC=0.92 Video: AUC=0.94 Multimodal: AUC=0.97 Multimodal Accuracy: 91.5%

3.1 – Closely Related Research Works

Cornu and Milner (2015) proposed a novel technique to extract visual speech features and use them to classify between speech and non-speech. The paper also compares CNNs and GMMs (Gaussian mixture model) to assess the performance of classification in visual-SAD. The paper argues how GMM systems have been outperformed by CNNs for automatic speech recognition and as a result, explores CNNs for SAD.

Their suggested architecture consists of 2 convolution layers (3x3x32,3x3x64), followed by max-pooling (2x2) at each convolutional layer with Dropout of 0.2 for convolutional layers. The architecture also includes L2 regularization with a value of 0.0001 and a dropout of 0.5 in the fully-connected layer with 512 neurons. For activation, they use ReLU.

GRID (Cooke et al., 2006), is the dataset used for their experiments in two different scenarios. In the first case, all data is split based on 80% for training and 20% for testing. In the second case, data is split based on speakers on the ratio of 80:20. For the first task (speaker-dependent), the CNN achieves an accuracy of 97.66% as opposed 94.34% by GMM. In the second task (speaker-independent), the CNN achieves an accuracy of 74.68% whereas the GMM achieves 70.50% accuracy.

The authors conducted further experiments by the inclusion of temporal information via CNNs whereby the first and last frames of the sequence are used for classification of speech. However, in this case, there was only a slight increase in accuracy compared to those mentioned previously. As a result, they conclude by suggesting how CNN can outperform GMM for visual-SAD and potentially other applications such as lip reading and automatic speech recognition.

The implementation of temporal information by Cornu and Milner (2015) suggest that there are better architectures/methods (such as use LSTMs or GRUs) that can be considered. They also advise for further exploration of different architectures regarding temporal information and increasing dataset, which could further improve the accuracy of the CNN. However, the architecture of the CNN, considering the accuracy, highlights that smaller networks can achieve reasonable accuracy for binary classification.

Sharma et al. (2019) extended the work of Cornu and Milner (2015). Their study involves visual SAD but is focused on the endpoint - when one stops speaking. The suggestion by Cornu and Milner (2015) of exploring different architectures for temporal dependency is

conducted by Sharma et al. (2019). As a result, they suggest an architecture for SAD by combining CNN and RNN (CRNN). Their architecture involves 3 layered-CNN with 5x5 filters with kernel sizes of 16, 32, and 8 respectively with a stride of 2. For every layer of CNN, max-pooling(2x2) is added as well as a batch normalization layer. The LSTMs are added to the CNNs to learn temporal dependence. In this case, the LSTMs are unidirectional, with a size of 64 and are 2 layered. The state is then passed to a dense layer followed by a SoftMax layer to determine the frame as speech or non-speech. Classification of the endpoint is based on the sequence, consisting of classification for each frame.

Their network is run on multiple datasets such as GRID, VidTIMIT (Sanderson and Lovell, 2009) and a personally collected dataset referred to as “Indian-English dataset”. Similar to Cornu and Milner (2015) the experiments are run on these datasets in two ways: speaker-dependent and speaker-independent.

In the case of speaker-dependent, the whole dataset is split across train, test and validation. For speaker-independent, the dataset is split based on speakers resulting in unseen data for each dataset (train, test and validation). The paper also compares performances of Cornu and Milner (2015) and their suggested architecture for the GRID dataset. Their results show that there is an increase in performance by adding RNNs as their architecture achieves an accuracy of 92.2% in the speaker-independent scenario as opposed to Cornu and Milner (2015) architecture achieving 74.68%. For speaker-dependent, the accuracy achieved is 96.5% which is a similar result achieved by Cornu and Milner (2015).

Sharma et al. (2019) applied the suggestion of considering different architecture for temporal information by Cornu and Milner (2015). As noted, this increased classification accuracy. Although the architecture lacks justification (such as the use of kernels), their paper provides evidence that the use of smaller CNN with RNN provides better accuracy than CNN (within the problem domain). Although the authors mention such architecture is suitable for real-time application, they do not consider or provide any evidence for real-time application. Furthermore, the authors do not apply the SAD for any purpose to validate the claim of real-time application.

Wang and Wang (2019) introduced Landmark Pooling Network (LPN) which acts as an attention guide scheme to help deep neural network only focus on the region of interest (ROI). In LPN, the network can be provided with full (raw) images with landmarks for the

network to focus on. This eliminates any pre-processing and computation to obtain ROI images, as well as reduces the number of parameters. Furthermore, (Wang and Wang, 2019) argue that in an LPN the network can assign higher values to weights in important locations.

The LPN focuses on facial images and can classify an image as speech or non-speech. The network utilises a CNN to learn image representation, landmark pooling for ROI of the image and uses RNN (GRU in particular) to learn sequential information and obtain a better understanding of the input signal.

The LPN uses a convolution layer (7x7x64), followed by a landmark pooling layer which uses landmark location to pull convolutional feature maps. The landmark pooling layer uses 20 landmarks around the mouth and thus has 20 - 64-dimensional vector which is passed to Fully Connected (FC) Layer.

The vector is then passed to a GRU with 64 hidden units and passed to another FC layer (at every timestamp), at which case classification is made in the softmax layer. The network uses AdaGrad (Duchi et al., 2011) with a learning rate of 0.0001.

The dataset for their network is personally collected and referred to as Labelled Speech in the Wild (LSW). The dataset consists of speech and non-speech sequences with 195 subjects and 8903 sequences. 171 subjects and 8002 sequences are used for training and 24 subjects with 901 sequences are used for testing. Data is augmented at random and includes random flipping, random cropping, random distraction and face movement speed is varied (by deletion or duplication of images) to reduce the network from overfitting.

The highest accuracy for their network is 79.9% which involves LPN and CNN - using CNN features of the image and landmarks as input. In their case, the LPN alone achieves 72.1% compared to a CNN (one convolutional layer, spatial max pooling layer, GRU) achieving 76.7%.

Although Wang and Wang (2019) propose the idea of using landmarks as opposed to hand-crafted features (or ROI images). Their architecture achieves better accuracy but by only a small margin.

Furthermore, it remains in question as to whether this requires further computation, especially for mobile devices. On a different note, as the images fed are full face images, this can affect the accuracy due to noise in the network. Lastly, their paper highlights that

use of single convolution layer does not necessarily lead to reasonable accuracy, as other architectures (with more layers) have achieved a higher accuracy.

3.2 – Other Research Works

Tao and Busso (2019) incorporate an audio-visual system which they suggest increases the robustness of the SAD. The authors present an end-to-end SAD with bimodal RNNs where acoustic features and visual features are learnt from raw data. Their proposed system models temporal information regarding audio-visual data and each modality (audio and video) is a sub-system of the proposed system. These sub-systems are implemented with LTSMs to capture temporal relationship within each modality and provide a representation for each modality. The features learnt are then concatenated and fed as input to another set of LSTM layers to capture temporal information across the modalities.

For the visual modality, they employ a CRNN architecture, in which 3 convolutional layers are stacked with each layer having 64 filters. The kernel sizes for each layer is 5x5 with a stride of 2.

They do not use a max-pooling layer as the feature map is reduced by the application of increased stride size. 2 LSTM layers are then stacked with 64 neurons, to further process the visual representation. A similar architecture is also deployed for audio modality. But instead, two maxout layers (Goodfellow et al., 2014) are used to learn acoustic features with 512 neurons instead of convolutional neurons to reduce the computational complexity in training. The features learnt are then fed to two LSTMs. For the audio-visual network, the features (audio and video) are concatenated and processed by two LSTM layers with 512 neurons which is then fed to a FC layer (implemented with maxout with 512 neurons). The output is then sent to a softmax layer for classification between speech or non-speech. The architecture uses ReLU for activation and Adam as the optimizer. With their implementations, their authors aim to provide a small and compact network to lower hardware requirements and computational resources which the authors argue is ideal for practical applications. For the same reason, the LSTMs used in their architectures are unidirectional LSTM as opposed to bidirectional LSTM and that is to reduce the latency of the model.

The authors created their dataset in which they record speakers reciting sentences in front of a camera using ideal and practical scenarios. In the ideal scenario, data is collected with a close-talking microphone and HD camera, whereas the practical scenario data is recorded using a tablet. In total, the data consisted of 55 females and 50 males totalling to 60 hours and 48 minutes of recordings. Data is split based on speakers with 70 subjects for training,

25 for test and 10 for validation. The authors use IntraFace (Xiong and De la Torre, 2013) to extract ROI from the full-face images recorded. These images are then down-sampled to 32x32 and converted to grayscale to reduce computational workload.

In their experiments, the multimodal achieves the highest accuracy of 93.8% in the ideal scenario and 92.1% in the practical scenario. The video modality achieves 60% accuracy in ideal scenario and 65.5% accuracy whereas the audio modality achieves 92.7% and 90.3% respectively. The authors argue with their proposed multimodal system, their suggested architecture achieves better performance than the state-of-the-art unimodal SAD systems.

The paper by Tao and Busso (2019) lacks justification for the same kernel sizes in video modality and resulting accuracy. However, the architecture does highlight that use of same size kernels and removal of subsampling can affect the accuracy of the network.

Furthermore, the proposed architecture relies on the visual and audio modality, which as stated, the failure of one component can affect the detection. Tao and Busso (2019) also suggest that further work involves researching where one component fails or there is missing information within the component. Nonetheless, their suggestion of using unidirectional LSTM as opposed to bidirectional LSTM can aid in the computation for real-time application, as they suggest the use of bidirectional LSTM can increase latency in the model.

Ariav and Cohen (2019) studied the performances between using audio, video and combination of both (audio and video) to assess the performances of the detection. For audio, they proposed a WaveNet encoder (Oord et al., 2016), whilst for video, they used a resNet-18 (He et al., 2016) to classify between speech and noise. Their multimodal is based on features extracted from WaveNet and ResNet-18 (with 512 embeddings) which are fused with Multimodal Compact Bilinear (MCB) pooling (Gao et al., 2016) module. This information is then passed to a 2-layer LSTM (with 1024 hidden neurons) to explore more temporal information in a supervised manner. Dropout at 0.2 and 0.5 is deployed across the network and batch normalization on the outputs of the audio and video networks and MCB modules output.

Their dataset derives from Dov et al. (2015) which involves 11 speakers reciting an article consisting of 33,000 sequences (each image sized at 90x110) of which 24000 were used to

train and 9000 for testing. To make the model more robust, data was also infused with noise.

Their results show the multimodal achieving AUC (area under the curve) of 0.97 whilst the video model achieved AUC of 0.94, with audio achieving AUC of 0.92. They found opting for smaller ResNet model provided better accuracy, as ResNet-50 and ResNet-101 caused a decrease in performance. They argue that the bigger models "are very deep models usually tasked with hundreds or even thousands of classes whereas voice activity detection is a binary classification" (Ariav and Cohen, 2019, 268). However, in their comparison between audio, video and multimodal, the unimodal (video) uses ResNet CNN whereas the multimodal uses combinations of CNN and RNN. As discussed previously adding RNN for SAD does increase the classification accuracy.

While Ariav and Cohen (2019) achieve reasonable accuracy, as the proposed architecture relies on multiple components, this can increase the prediction times.

Furthermore, it also means that the failure of one component can affect the overall accuracy. Nonetheless, due to their proposed architecture and the fact that the detection requires audio and video, this can be computationally expensive and not best suited for mobile devices or real-time application. Although Ariav and Cohen (2019) did not see any evidence (in terms of accuracy) over selecting different sequence lengths, it highlights that different sequence sizes need to be experimented with, to understand the difference for accuracy and prediction times.

3.3 – Summary

Various approaches in recent years for the task of SAD have been considered. As one may note, the literature does not provide a comprehensive comparison of the approaches/models. For example, the difference in the application of the RNN to CNN is mainly based on accuracy but not in terms of computation and prediction times. Secondly, the models proposed within the literature are trained on hardware-intensive resources and do not consider modest computational resources. Furthermore, existing literature shows classification based on each frame/image and do not consider the classification of sequences of images, which would suit the real-time application. This would also open to different architectures regarding temporal information.

Lastly, as the existing SAD do not consider particular purpose, the dataset can be labelled differently as to when one is speaking or not. Thus, leading the network to learn speech and non-speech differently.

On the other hand, the literature highlights that the task of SAD can affect the accuracy with a large network. Reasonable accuracy can be achieved with fewer convolutional layers and smaller filter sizes. Regularization techniques such as batch normalization and dropout are evident to be useful for a smaller network as well as reduce overfitting. The literature also shows the use of unidirectional LSTMs being better to decrease the latency of the model as well as feed each image independently.

4 – Visual Speech Detection

Existing literature in section shows that SAD is regarded as a classification or captioning of images and shows that neural networks can automate a system in identifying speech activity, whilst outperforming other systems. In particular, CNNs have proven to have the ability to be able to distinguish between speech and non-speech. In recent years, the addition of Recurrent Neural Network (RNN) can further increase the accuracy of the detection with its introduction of memory/history. However, the addition of RNN can also increase the computation resources required and affect prediction times. Nonetheless, existing literature highlights that detection of speech can be done using still images or sequences of images.

Therefore, detection of speech is examined in the following way:

1) Detection of speech using still images - to obtain a baseline performance and comparison to other models. CNNs are applied due to their powerful abilities as mentioned in section 2.4 – CNN and 3 – Literature Review.

2) Detection of speech using sequence images. In this case, the detection of speech is examined in two ways:

- classification of the image based on previous images (or history of images).
- classification of the sequence i.e. classifying images in a sequence (sequence to sequence).

Furthermore, for each type of classification mentioned in sequences of images, two types of architectures are deployed. In the first architecture, CNN and RNN are combined (CRNN) similar to work of Tao and Busso (2019), Sharma et al. (2019) and Wang and Wang (2019). The second architecture is the CNN-RNN, where embeddings from an existing CNN model are used together with an RNN similar to work of Donahue et al. (2017) and Ariav and Cohen (2019).

Accuracy is used to differentiate and compare other models. Although there are better metrics available as mentioned in section 2.12 – Evaluation Metrics, accuracy requires no further computation to assess a model. Other metrics may provide a better measure of performance of the model, but the application of such a metric can be complex due to the nature of models (such as sequence classification). It is also a popular metric used as mentioned in section 3 – Literature Review.

Furthermore, factors such as loss (error rate), training and prediction times are also considered as the network/system is intended for mobile use/infotainment system.

4.1 – Experimental Design

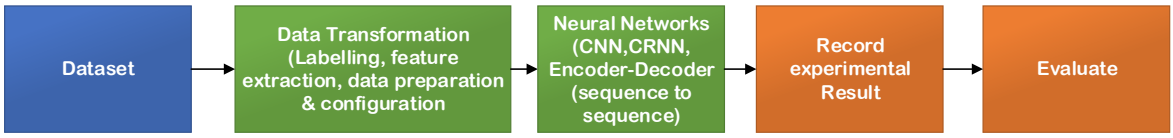


Figure 4.1:Design for approaching the problem in question.

Figure 4.1 shows the overall design for approaching the problem - identifying speech with facial images. Firstly, the dataset for the experiments is identified. Secondly, data transformation is highlighted with the configuration/pre-processing necessary. Thereafter, the different variations of speech detection (detecting speech with still and sequence of images) are explored. Lastly, these variations of speech detections are evaluated and compared to find the detection that provides the highest accuracy and lowest prediction times.

4.2 – Dataset

In approaching the problem, one of the most important aspects is to identify a suitable dataset. As without the right data, one cannot build any form of neural network. More importantly, the dataset needs to be appropriate and meet a set of requirements. The requirements in this problem domain are: data should be in a usable format (i.e. JPEG, PNG), the dataset should provide images of individuals speaking and not speaking, include a time series (in terms of the images) of when the user starts to speak and finishes speaking.

The data utilised in the networks is derived from the VidTIMIT dataset (Sanderson and Lovell, 2009), which is also the dataset used by Sharma et al. (2019). The dataset consists of video and audio recordings of 43 people saying short phrases. Using the VidTIMIT dataset, it allowed obtaining facial images, as well as images of individuals that can be categorised based on speech (speaking and not speaking). Additionally, the total images are in excess of 100,000, which provides enough data for a neural network to truly learn the differences between images. It also allows to test the network thoroughly and build an efficient system.

VidTIMIT dataset

The dataset consists of 43 individuals which include 19 females and 24 males. The individuals were tasked to speak 10 different sentences recorded over 3 sessions. There was a delay of 6-7 days between sessions so that different data of the same individual (such as different attire, hairstyle, beard etc) can be recorded. Each session started with a head rotation sequence whilst images were captured. After that, the individuals were asked to speak some sentences. Audio and video were recorded during the recitation. For each individual 1346 images were captured (on average) during the head rotation sequence, and 1061 images were captured (on average) during the recitation. Thus, the total of head rotation images equates to 57881 images and recitation images to 45661.

Figure 4.2: Sample of images recorded over 3 sessions

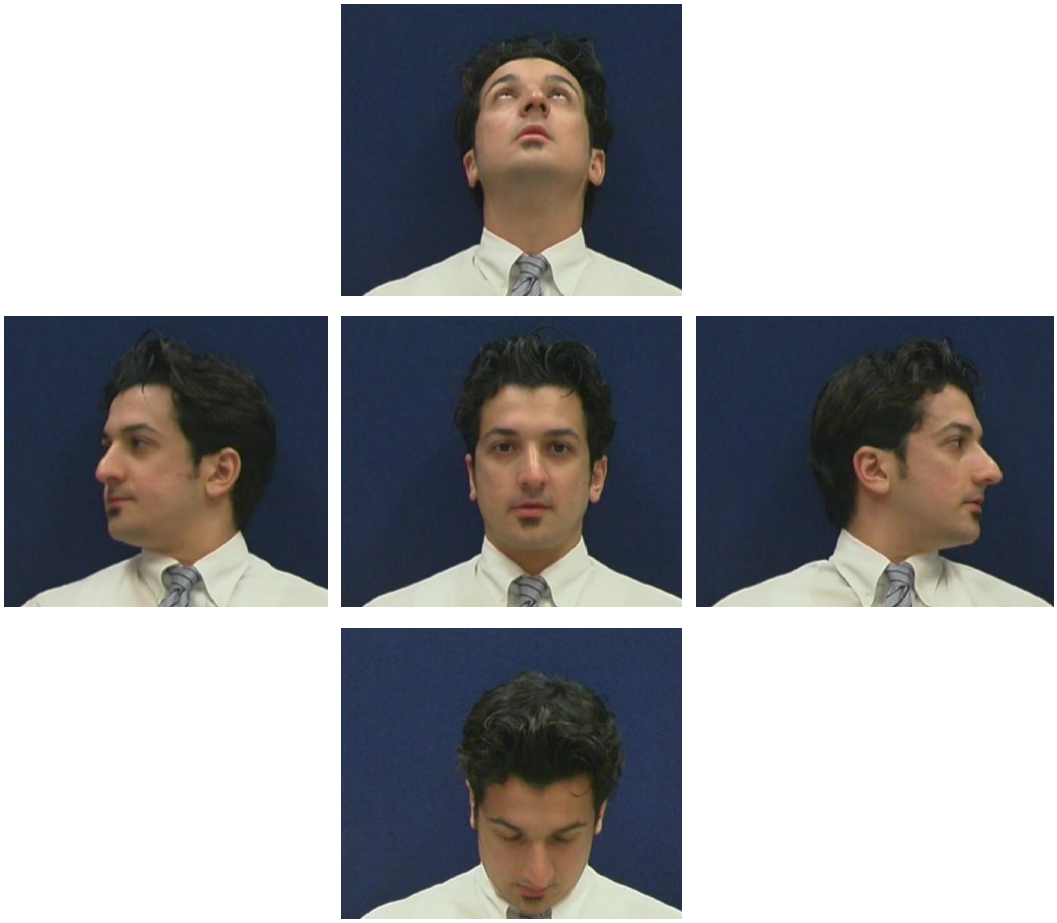


Figure 4.3: Head rotation sequence

4.3 – Data transformation

Data transformation is the process of converting the dataset to usable data for the neural networks. Data transformation occurs at three stages: image labelling, feature extraction and preparation of data for the neural network.

Data Labelling

Data labelling involves categorising the images into two categories - “speaking” and “not speaking”. Each image was labelled (“speaking”/ “not speaking”) manually after a review of the labels from two individuals. This process is initiated with splitting the dataset at the ratio of 50:50 where each individual labels the 50% of the dataset. Upon completion, the data is swapped where the individual now reviews the labelled data from the previous individual.

Each individual allocates each image (at a time) to the relevant category (“speaking” / “not speaking”). To clarify as to which image is speech or non-speech, two factors are considered: the state of the image at the given time (Figure 4.4), and the state of the current image based on previous images (Figure 4.5). Upon any disagreement in the label of the image, the individuals reconsider and discuss the label based on the two factors mentioned. Additionally, the individuals place the image on the timeline based on the audio file provided from the relevant dataset.

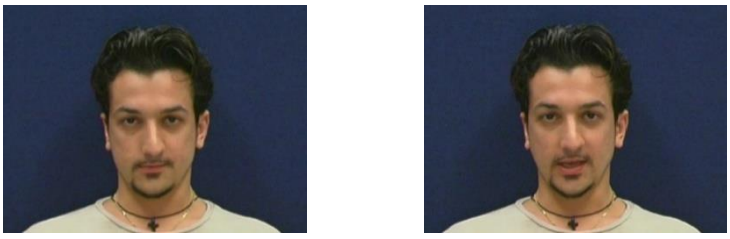
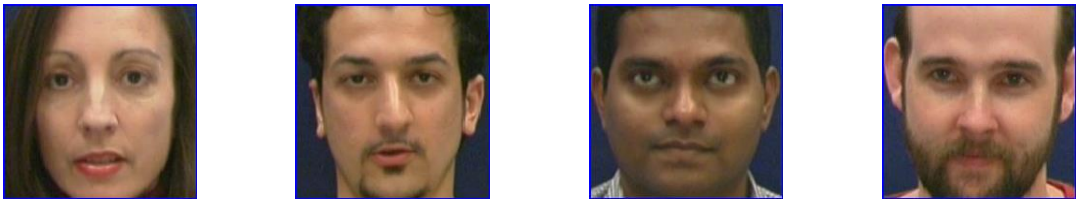


Figure 4.4: factor 1 – labelling image based on current image

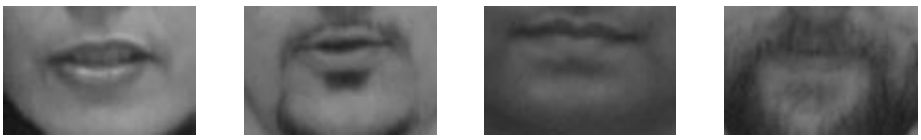
Figure 4.5: factor 2 - labelling image based on previous images

Feature Extraction

Image from Dataset



Applying Haar Cascade



Applying Heuristics

Applying Brightness + contrast

Figure 4.6: Transforming images from dataset to usable data for the network

Feature extraction involves altering the labelled images so that only mouth-region images are obtained. The dataset mentioned includes full-face images, together with background and clothing which can add noise as noted in Figure 4.6.

As the problem domain is based on detecting speech via facial images, only the mouth region of the images is necessary. Information regarding the background, clothing, and facial features (i.e. hair), is unnecessary and can potentially add noise to the network making it more difficult in distinguishing between speech and non-speech. As a result, within the process of feature extraction, unnecessary information regarding clothing and background etc is eliminated, and only relevant features from images are kept (mainly the mouth-region area).

Haar cascades were used to obtain facial images and using heuristics the mouth-region of the image is obtained. The frontal face Haar cascade by Lienhart and Maydt (2002) is used,

as it provides a mechanism in which images can be passed and only those images that have a face visible are kept. If the face is not clearly visible, then the image is discarded (e.g. images obtained within the head sequence – see Figure 4.7). Of the valid images, heuristics were applied (as noted in Figure 4.6).

Figure 4.7: discarded images during face detection

Figure 4.8 highlights the calculation in which the mouth-region is extracted from the facial image, where x and y represent the axes of the image. The w and h represent the width and height of the image and ex and ey are the starting points of the mouth-region as noted in Figure 4.9.

$$for\ (x, y, w, h)\ in\ faces:$$
$$ew = \left(\frac{w}{4}\right)$$
$$eh = \frac{h}{3}$$
$$ex = x + ew$$
$$ey = y + (eh * 2)$$
$$Mouth\_region = ey: ey + eh, ex: ex + (ew * 2)$$

Figure 4.8: pseudo code of obtaining a mouth image assuming the same face proportion for all images

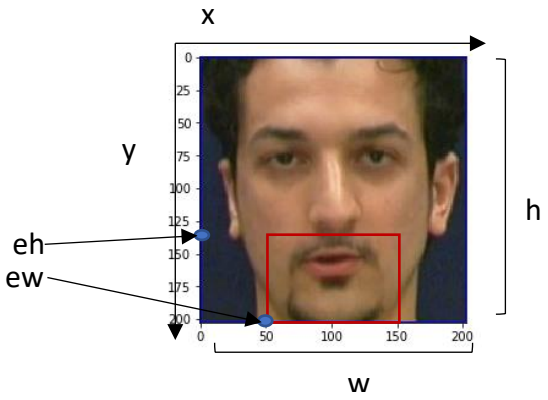


Figure 4.9: mouth-region extraction based on pseudo-code in Figure 4.8

Arguably, a mouth Haar Cascade (Castrillón et al., 2007) could be deployed to retrieve the mouth-region as opposed to the application of face Haar cascade and heuristics. However, Manoharan and Chandrakala (2015) found that mouth Haar cascade fails in detecting mouth-region images if there is any rotation of the face. Therefore, as the dataset involves head rotation, the mouth Haar cascade could ignore these images (see Figure 4.10) which can be useful to train the network on "no speaking"/non-speech element.

Figure 4.10: Images where the face may be visible but mouth may not be clearly visible.

As noted in Figure 4.6, images can be inconsistent after the application of Haar Cascades. Due to lighting conditions, facial/race colour, or facial aspects such as beards etc, some images can appear to be brighter or darker than others. These inconsistencies can cause fatal issues in a neural network. As the images are greyscale images, the brightness and contrast of the image are enhanced. Through using the median of the image, brightness and contrast levels are increased or decreased to ensure all images appear consistent.

Applying the algorithm in Figure 4.11 allows controlling the pixels of all images, thus allowing to maintain same/similar average pixels between all images in the dataset. Furthermore, the contrast is altered to enhance the images so images appear more clear and vivid (as noted in Figure 4.6).

Figure 4.11 utilises the built-in libraries by Clark and Contributors (2016) to obtain a median for the image. If the image is dark i.e. less than 170, 170 is divided by the median to provide a small positive number that is applied to increase the brightness through a library by Clark and Contributors (2020). This allows increasing the brightness of the darker images. If the image median falls between 171 and 180 the factor is set to 1 – which means that no brightness is added. If the median is greater than 181, the median is divided by 181 to provide a small negative number to enhance/darken the image through a library by Clark and Contributors (2020). This allows to darken a fairly bright image. In all cases the image is enhanced with contrast by a factor 2, allowing the colours to appear more vivid. This is particularly useful if the image is greyed out due to brightness adjustment. This process

allows to provide images that have the same brightness and contrast so all images appear consistent.

$$int\ Image\_Median = Image.median(\ )\ *$$
$$if\ Image\_Median \leq 170:$$
$$factor = \frac{170}{Image\_Median}$$
$$if\ Image\_Median \geq 171\ and \leq 180:$$
$$factor = 1$$
$$if\ Image\_Median \geq 181:$$
$$factor = \frac{181}{Image\_Median}$$
$$Enhance\ brightness(factor)\ **$$
$$Enhance\ constrast(2)\ **$$

* (Clark and Contributors, 2016)

** (Clark and Contributors, 2020)

Figure 4.11: pseudo-code for increasing/decreasing brightness/pixels of an image - allowing to keep images consistent.

In comparison to the pre-processing techniques (extraction of ROI) used by authors in chapter 3 – Literature Review, Sharma et al. (2019) and Tao and Busso (2019) used detectors "YOLO" and "IntraFace" respectively. Cornu and Milner (2015) and Wang and Wang (2019) obtained facial landmarks and extracted ROI images based on the calculations from the landmark data. Lastly, Ariav and Cohen (2019) only recorded mouth-region data when collecting their dataset. The pre-processing conducted in this thesis is similar to the work of Cornu and Milner (2015) and Wang and Wang (2019), where extraction of ROI is calculated based on the facial landmarks. This method was selected as it provided quicker extraction of ROI and proved to be less complex, which leads to less computational resources required. Furthermore, the methods used by Sharma et al. (2019) and Tao and Busso (2019) have stated to be complex and require consideration and work in the validity and speed of extraction.

Data Preparation and Configuration

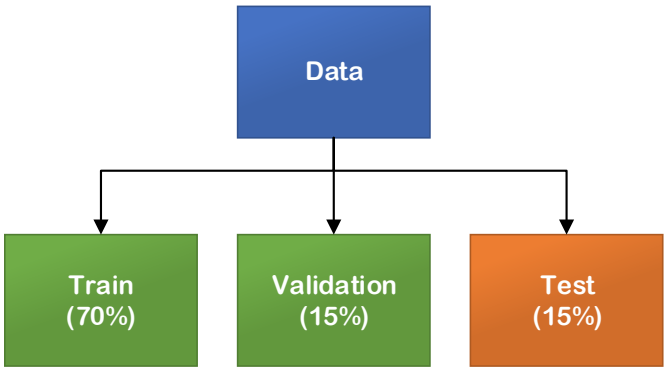


Figure 4.12: Allocation of the datasets

In this stage, the usable data is prepared for the neural networks, which involves creating different datasets: train, validation and test. For all neural networks, the data is split in the ratio of 70:15:15 for train, validation and test respectively. The data is chosen randomly.

As noted in Figure 4.12, the train and validation datasets are utilised during training, whilst test dataset is used to evaluate. The test set is the data that would be unseen during training phases of any experiment.

Some configurations for parameters are set as default, such as Adam as the optimiser which is used by various authors mentioned in section 3 – Literature Review. For similar reasons, ReLU and softmax are the activation functions used. Where applicable default values are used (as suggested within the Keras documentation) such as the learning rate for the optimiser (Keras, 2020c). All networks are run for 25 epochs which are then evaluated. As this thesis examines different architectures and networks, further configuration/ preparation can be found in the relevant section.

The accuracy set for the initial model is 90% at which case the model is adjusted depending on the accuracy and performance. Once the model achieves adequate results and provides reasonable performance, it is then used to compare against other models for different networks. The best model within the problem domain is selected based on accuracy, error rate (loss) and history of performance.

For each experiment, the model that achieves the highest accuracy is cross-validated where $k = 3$. The train and test data sets are randomised in each round, with data deriving from train and test datasets. Whilst the validation set is used to evaluate the model. This method allows for obtaining robust/consistent results.

5 – CNN for Still Images

Existing work shows CNNs have been used for various researches and fields where authors have boasted great results for recognition. As a result, the detection of speech with still images is first experimented on for two reasons. Firstly, due to its popularity for image classification and secondly, to obtain baseline performance and compare other models in this thesis.

5.1 – Pre-Processing and Configuration for CNN

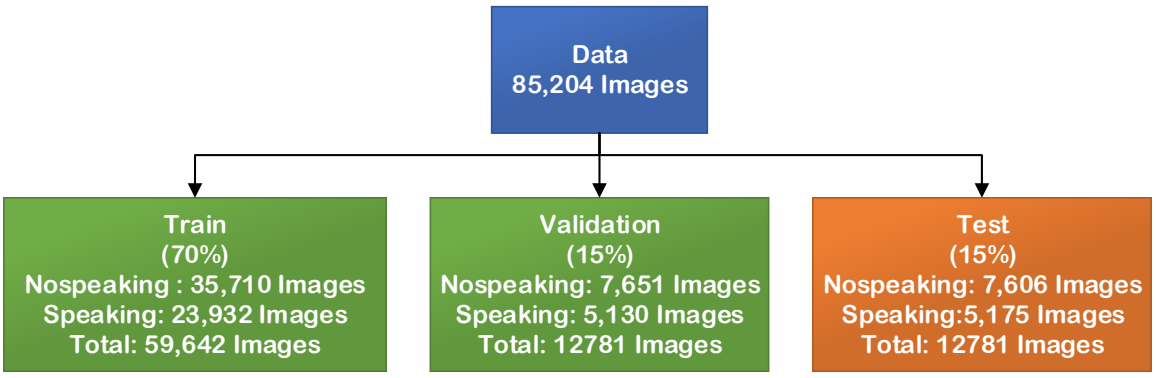


Figure 5.1: Data utilised in CNN networks

Preparing data for CNN models involved creating datasets (see Figure 5.1) manually-albeit via computation. The data distributed is selected randomly and is ensured to not be overlapping/repeating. Adapting the above process allows better control over images that are utilised in the network, as well as provides an easier and more accessible way of backtracking during errors/debugging (e.g. tracking invalid data such as mislabelled images).

Image data generator (Keras, 2020a) is utilised for generating images for the neural network. The image data generator allows a simple and computationally inexpensive way to generate images for the network. As images can be generated for each epoch rather than pre-processing all images within the dataset, which can be computationally expensive. Furthermore, the image data generator allows to perform any data augmentation if necessary and control how images are being read in a neural network.

In order to utilise the image data generator, it is required for data to be split based on the classes. Thus, after images were selected for each dataset, they were then categorised based on their class names.

5.2 – VGG-Like Model

As mentioned in section 2.4 – CNN and 3 – Literature Review, VGG is one of the popular CNN models that has achieved top performances but also changed the way architectures are designed for the networks. Due to its simplicity, the architecture is adapted for the problem in question. However, as there are various versions of VGG (VGG-16, VGG-19) which carry a substantial number of layers and parameters, the architecture requires not only time but powerful machines that can compute such a model.

From their publication, Simonyan and Zisserman (2015) introduced using smaller 3x3 filters for CNN models and utilising more Convolutional Layers in a convolutional block. In the paper, the authors compared the different depths and sizes of the VGG model. Figure 8.2 (in the Appendix section) shows different architectures of VGG with their parameters. As noted, even the smallest VGG model (VGG-11) has over 100 million parameters. As a result, the architecture was adapted by reducing the layers (as noted in Figure 5.2) to allow the architecture to be computed with the resources available.

VGG-11 – 133 Million Parameters, 10.4% Top-5 Error Rate

Adapted VGG model

Figure 5.2: VGG-11 vs VGG-Like model deployed.

Initial Architecture

As noted in Figure 5.2, the last convolution block of 512 was removed. Furthermore, the number of hidden units in FC layers are also altered as a large number of hidden units require more computational resources (i.e. memory). However, the value of the hidden units is only set by experimenting with a different number of hidden units.

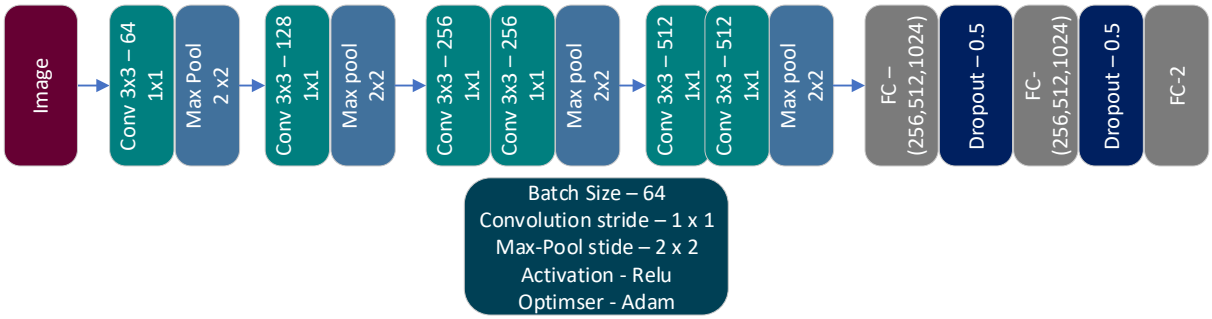


Figure 5.3: The VGG-9 model deployed.

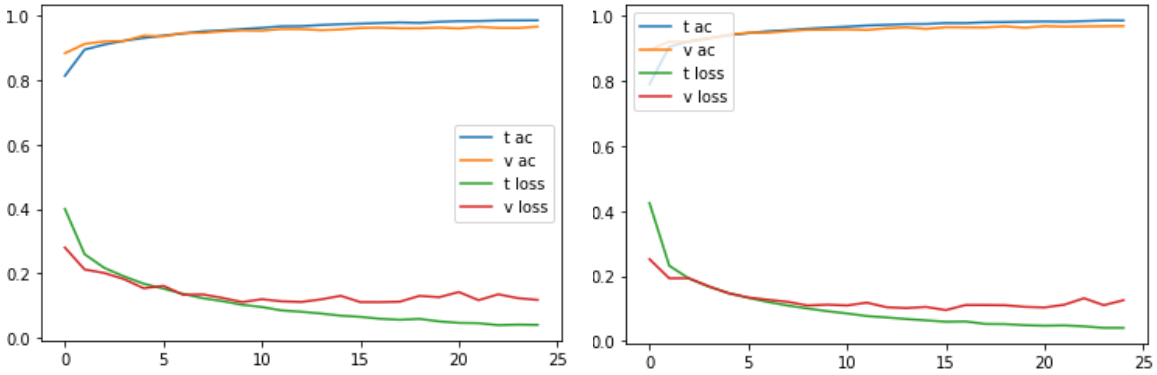
Initial Results

Table 5.1: Comparison of the VGG-9 architecture with a different number of hidden units in the FC layer.

Number of hidden units	**Validation (Loss, Acc)**	**Test (Loss, Acc)**
FC 256	0.1282, 0.9644	0.1409, 0.9639
FC 512	0.1208, 0.9651	0.1239,0.9615
FC 1024	***0.1244,0.9684***	***0.1217, 0.9687***

Table 5.1 shows the VGG-9 model's performance in the problem domain, with a different number of hidden units in the FC layers. The model achieved the best accuracy of 96% with the loss of 0.12. As one may note, 1024 hidden units in the FC layer provided the optimal result, although there is only a marginal difference on the metrics (accuracy and loss) between a different number of hidden units.

The graphs in Figure 5.4 also indicate that the model can be improved based on validation loss. As one may note, there is overfitting (albeit marginally) as the performance of validation loss ("v loss") is not in line with the training loss ("t loss").



FC 512 FC 1024

Figure 5.4: comparison of the performance of VGG-9 architecture between 512 and 1024 hidden units in the FC layer

5.3 – Further VGG-Like Experiments

With the initial experiments highlighting that the model can be improved, further experiments are conducted. Improvement of the network is done by adding batch normalization and altering the value and presence of dropout. As mentioned in sections 2.7 – Batch Normalization and 3 – Literature Review, batch normalization has been proven to not only aid the network in learning but also improve the overall performance of the network. As a result, batch normalization is added particularly before the activation function as recommend in the paper by Ioffe and Szegedy (2015).

Similarly, dropout is known to reduce overfitting as mentioned in section 2.8 – Dropout. Furthermore, as mentioned in the paper, Srivastava et al. (2014) suggest dropout can be used in various layers such as the input layers. Application of this is shown in the work of Cornu and Milner (2015) and Ariav and Cohen (2019) who used dropout of 0.2 in convolutional layers and 0.5 in FC layers.

Furthermore, Ariav and Cohen (2019) particularly, used a combination of batch normalization and dropout, Ioffe and Szegedy (2015) mention use of batch normalization reduces the value of dropout or in some cases eliminates the need of dropout.

With this information, further experiments are conducted. To keep the experiments fair and comparable, the same parameters are shared across all networks. These parameters include Adam as the optimizer, ReLU as the activation, 1024 hidden neurons in FC layers, same padding for convolutional layers, max pooling of 2x2 and batch size of 64.

Enlarging this architecture (VGG-9) in any shape albeit by adding batch normalization affects the resources required. That is due to the computational resources required for the convolutional operation and adding the batch normalization means more computational resources are required; as the network is reparametrized with each mini-batch. As a result, the stride of the first convolution is changed (stride=2) to facilitate the memory required for the batch normalization operation.

The Presence and Alteration of Dropout

Firstly, dropout is explored based on the work of Cornu and Milner (2015), who utilised dropout with a rate of 0.2 in convolutional layers and 0.5 in FC layers. Ariav and Cohen (2019) also follow the same principle of utilising dropout at 0.2 and 0.5. To get a better understanding and to compare the performance of dropout, the network is infused with

dropout at 0.2 (as suggested by Srivastava et al., (2014) in all layers and 0.2 in convolutional layers and 0.5 in FC layers.

Table 5.2: Comparison of two networks with different dropout rates. Left: dropout at 0.2 in all layers. Right: Dropout at 0.2 in convolutional layers and 0.5 in FC layers.

Experiment name	**VGG-9 – 20% dropout**	VGG-9 – 20% and 50% dropout
Architecture	CONV (64, stride = 2)>RELU>Max-Pool CONV (128)>ReLU>Max-Pool CONV (256)>ReLU>CONV (256)>ReLU>Max Pool CONV (512)>>ReLU>CONV (512)>ReLU>Max Pool FC (1024)>ReLU FC (1024)>ReLU FC (2)>SOFTMAX	
Dropout	**Dropout of 0.2 in all layers**	Dropout 0.2 in CONV, 0.5 in FC layers.
Test (loss, Acc)	**0.0968, 0.9641**	0.1028, 0.9603

Table 5.2 shows the results in the comparison between different dropout rates. The results show that network with a dropout of 20% provides a marginally better accuracy (96.41%) than that with a dropout of 20% in convolution with 50% in FC layers (96.03%). In comparison with the initial architecture, the addition of dropout improves the error rate (loss) of the network with a slight impact on accuracy.

The Addition of Batch Normalization

In this experiment, batch normalization is explored. As mentioned previously, batch normalization has proven to aid in performance regarding the speed of learning (training time) and classification accuracy. Ioffe and Szegedy (2015) suggest that the use of batch normalization can improve the network alone. Thus, use of batch normalization eliminates the presence or the amount of dropout applied. Contrary to Ioffe and Szegedy (2015) suggestion, the application of batch normalization and dropout can help in improving a network. Ariav and Cohen (2019) utilised batch normalization and dropout (at 0.2 and 0.5) which they suggest helped their network. As a result, in this experiment, the application of batch normalization and batch normalization with dropout is compared. These comparisons will help to understand the effect of applying batch normalization as well as whether applying batch normalization and dropout is useful for a network. Batch

normalization is added after each convolutional and fully connected layer but before activation, as recommended by Ioffe and Szegedy (2015). A dropout rate of 0.2 is utilised in each layer based on the performance from the previous experiment. Same architecture (VGG-9) is utilised as mentioned previously.

Table 5.3: Comparison between the application of batch normalization and batch normalization with dropout at 20%

Network name	VGG-9 – BN	**VGG-9 – BN + Dropout**
Technique	BN	**BN, Dropout 0.2 in CONV, 0.5 in FC layers**
Test (Loss, Acc)	0.1607, 0.9591	***0.0967, 0.9708***

Table 5.3 shows the results in the application of batch normalization and batch normalization with dropout. The results show batch normalization improving the speed of learning but had no positive impact on accuracy. The initial architecture (VGG-9) achieved 96.87% accuracy while applying batch normalization the accuracy obtained is 95.91%. On the other hand, the combination of batch normalization and dropout is evident to improve the training times and accuracy of the network. The results show that this combination provided 97.08% accuracy and 0.097 error rate which is a marginal improvement from the initial architecture.

Smaller is better

Table 5.4: Comparison of 3 significant smaller networks

Experiment Name	VGG-7 – BN + Dropout	7 Layer CNN	**5 Layer CNN**
CONV Layers	4	5	**3**
CONV filter size	64,128,256,512	32,64,128,256,512	**32,64,128,**
FC size	1024, 1024	1024	**1024**
Technique	BN + Dropout	Dropout	**Dropout**
Test (Loss, Acc)	0.1312, 0.9581	0.0998, 0.9654	***0.075, 0.9721***

Ariav and Cohen (2019) found that with lesser classes and depending on the dataset, smaller models can obtain better or similar accuracies than larger models. Furthermore, as noted in chapter 3 – Literature Review, various authors utilised a smaller network with 2 or 3 convolutional layers for SAD which could also improve training and prediction times. As a result, in this experiment, smaller networks are explored to find out whether they can

improve the accuracy previously obtained (97.08%). Table 5.4 simplifies the various number of experiments conducted and highlights three networks that provided significant results. The changes to the networks are conducted based on experimental results with parameters chosen based on these results and published work (literature review). The number of filters is kept the same (3x3 filters) with incremental number filters (i.e. 32, 64, 128) as popularly used within literature and to keep the experiments comparable.

Based on the comment by Ariav and Cohen (2019), VGG-7 (architecture on left) in Table 5.4 derives from the initial model (VGG-9) explored previously. In the attempt of going smaller, the VGG-9 model was altered by removing the extra convolution layers. With this change, the model was overfitting with dropout at 20%. As a result, the dropout was altered to 0.25 and 0.5 based on experimental results and the popular use found in chapter 3 – Literature Review. This architecture (VGG-7 in Table 5.4) showed an accuracy of 95.81% with error rate if 0.1312. However, the network was still overfitting albeit marginally.

In chapter 2 – Background and 3 – Literature Review, it is highlighted that various authors utilised smaller filter sizes within SAD and similar problem scenario (i.e. few classes, dataset etc.). As a result, the filter sizes within the convolution operation were altered as shown in Table 5.4. Comparisons on the effect of batch normalization are also conducted based on the overfitting and variation in the validation accuracy (as opposed to validation accuracy being in line with training accuracy). This will validate as to whether batch normalization causes this variation in validation accuracy. The result of this alteration in the network impacted positively – albeit marginally. The 7-layer CNN (architecture on the middle) in Table 5.4 achieved the accuracy of 96.54% with 0.0998 error rate. This validated the claim that batch normalization could have been causing the variation in the performance of the validation accuracy. Furthermore, the changes to the filter size also improved the accuracy from 95.81% with an error rate of 0.1312 to 96.54% with 0.0998 error rate.

The 5-layered CNN in Table 5.4 (architecture on the right) derives from the review in section in 3.1 – Closely Related Research Works. The review highlighted that for an online system, a CNN with 3 convolutional layers or less is adequate to provide reasonable accuracy and prediction times – best suited for online systems. As a result, CNN with 3 convolutional layers is explored. Based on the results thus far and the review, smaller filter sizes are used (as shown in Table 5.4). Within smaller CNNs, various authors (in chapter 2 – Background and 3 – Literature Review) have utilised dropout rates of 0.2/0.25 in convolutional layers

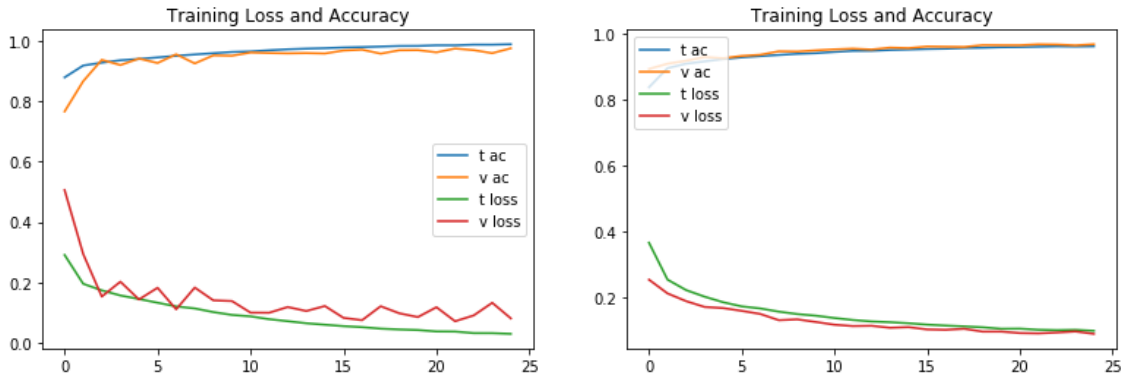
and 0.5 in FC layers. Based on this, a dropout rate of 0.25 and 0.5 is utilised. The 5-layered CNN (the architecture on the right) provided the highest accuracy so far. The addition of dropout at 0.25 in convolutional layers and 0.5 in FC layers had a positive effect with the network classifying at 97.21% accuracy with 0.075 error rate.

5.4 – Discussion

Table 5.5: Comparison of the two highest accuracy achieving architectures. Values for train and prediction times are average.

Experiment Name	VGG-9 – BN + Dropout (0.2)	**5 Layer CNN – Dropout (0.25, 0.5)**
CONV Layers	9	**3**
CONV filter size	64,128,256, 256, ,512, 512	**32,64,128**
FC size	1024, 1024	**1024**
Technique	BN + Dropout (0.2)	**Dropout (0.25, 0.5)**
Total Parameters	10,286,082	***Total Parameters: 18,970,114***
Test (Loss, Acc)	0.0967, 0.9708	***0.075, 0.9721***
Train Times	163.16s	***155.92s***
Prediction Time	9.69ms	***5.865ms***

In the previous section various architectures deriving from chapter 2 – Background and chapter 3 – Literature Review, were explored. As noted, numerous architectures and combinations are experimented including a different number of layers, sizes and techniques such as batch normalization and dropout. Table 5.5 shows two architectures provided the highest accuracy within CNNs. The 9-layered CNN (VGG-9) with batch normalization and a dropout rate of 0.2 provided the second-highest accuracy. The architecture achieved an accuracy of 97.08% with a loss of 0.0967 (as noted in Table 5.5). The average training time was 163.16s(per epoch) with an average prediction time of 9.69ms. With the addition of batch normalization and dropout, there is a slight increase in accuracy from the initial model mentioned in Table 5.1. However, there is a better decrease in loss which was at 0.1217 from the initial architecture in Table 5.1 which is now at 0.0967. Figure 5.5 compares the performance between the two architectures. As noted for the 9-layered CNN in Figure 5.5, the network is marginally overfitting as the validation accuracy and loss are not in line with the training accuracy and loss.



9 Layered CNN 5 Layered CNN

Figure 5.5: Comparison of the performance between the two architectures in Table 5.5.

Ultimately, a smaller 5 layered network with 3 convolutions (architecture on the right in Table 5.5) achieved the highest accuracy of 97.21% with a loss of 0.075. Furthermore, the performance of the validation accuracy and loss was smoother and in line with the training accuracy and loss as noted in Figure 5.5. This architecture achieved 5.865ms for prediction time and thus outperforms the previous model (architecture on the left in Table 5.5) in all aspects such as accuracy, loss and training and prediction times.

In this section, CNNs were considered for the task of SAD. Two types of architectures were initially experimented with. One deriving from the VGG architecture and the other from the existing literature which highlights the use of a smaller number of convolutional layers. However, as mentioned previously, the smaller architecture still followed the architecture of VGG, in the sense it used 3x3 filters and incremental kernel sizes of 32, 64 and 128, with a dropout of 0.25 and 0.5. This architecture was found to produce the highest accuracy of 97.21% with almost half of the prediction time compared to the VGG-like model (VGG-9). Therefore, this model/architecture is selected for detection of speech with still images and used as a comparison for further experiments in this thesis.

With the computational resources available only a selected set of configurations/models are experimented. This includes experiments being conducted in an offline setting. Online experimentation would provide a better assessment of these models. Furthermore, the models experimented had to be altered to facilitate the memory required such as changing the stride of the convolutional layer. This could potentially impact the accuracy of the network. However, the accuracy achieved with CNN (97.21% accuracy) is similar to those mentioned in 3 – Literature Review (CNN-based accuracy). One of the limitations of CNN is memory. CNN analyse each image and do not store the data which can be useful for future operations. This is why Cornu and Milner (2015) suggest exploring temporal

information which can further improve the accuracy of the CNN. Furthermore, with the addition of memory, the network can be designed smaller and require less computational resources. This would suit an online/mobile implementation for quicker training and prediction times. The impact of exploring temporal information with CNN is evident in the work of Sharma et al. (2019). Furthermore, Sharma et al. (2019) suggest that such architecture is evident to outperform other methods.

6 – CNN for Image History and Sequence

In the previous chapter, identification of speech via still images was discussed. That is, considering an image at a given time if the image belongs to the category of speech/non-speech. In this chapter, detection of speech is identified based on sequences of images. Much like Figure 4.5, speech is identified based on history/time-lapse of previous images. This chapter aims to find the effect of adding history/sequence of images on the detection of speech and compare its performance to the previous detection.

However, in this section sequences of images are utilised in two ways: -

- classification of the image based on previous images (or history of images).
- classification of the sequence i.e. classifying images in a sequence (sequence to sequence).

As explored previously, RNN is a type of network that allows a series of inputs and provides the ability to learn from the past. Moreover, existing work (in section 2.9 – RNN and 3 – Literature Review) shows various authors utilising RNNs in different ways. Particularly with images in mind, several authors have used a combination of CNNs and RNNs. The CNNs, in this case, are often used to learn information regarding the image whilst the RNNs are utilised to learn sequential data and spatial dependencies. However, within this method, there are various approaches in which this method can be implemented.

Soh (2016), Wang et al. (2016) and Ariav and Cohen (2019) implemented a network with two subsystems including CNN and RNN. Existing models such as VGG or Xception are used to obtain image representations (embeddings) which are then concatenated/formatted to smaller vectors. These vectors are then passed to LSTMS which are designed and attached to learn sequential information. Commonly, these types of models are categorized as CNN-RNN.

Figure 6.1: CNN-RNN - using existing CNN model.

On the other hand, Choi et al. (2017), Shi et al. (2017), Han and Kamdar (2018) and Sharma et al. (2019) implemented a network which combines a smaller CNN and RNN to form a unified model. In this case, the design of CNN may follow the architecture of an existing

model (such as VGG). LSTM's are then attached to learn sequential information passed from the convolutional layers. This type of model is known as CRNN.

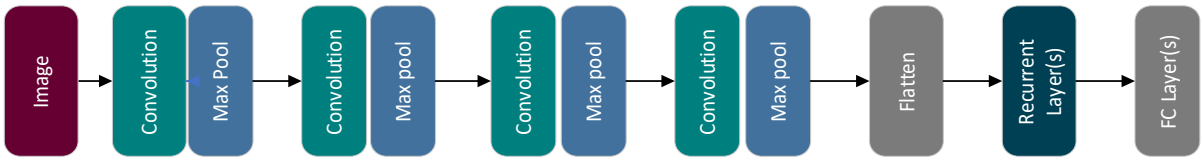


Figure 6.2: A CRNN model.

For the architectures mentioned above, the authors have claimed publishable results and concluded the performance to be superior to CNN models. However, regarding the Recurrent Layer(s) in the above models, one may find two popular ways in which authors have utilised the RNN.

Lim et al. (2016), Choi et al. (2017), Shi et al., 2017, Tao and Busso (2019) and Sharma et al. (2019) attached multiple recurrent layers to a CNN (CRNN) to encode the individual information to a sequence, thus, allowing for classification of the image based on the history of the image. This architecture in this thesis is referred to as encoder.

For captioning images (sequence to sequence), Vinyals et al. (2015), Venugopalan et al. (2015), Soh (2016) and Donahue et al. (2017) used a separate set of recurrent layers in which the first set of recurrent layer(s) act as an encoder, whilst the latter set of Recurrent Layer(s) pose as a decoder. As previously mentioned, this architecture is referred to as encoder-decoder.

As a result, this thesis considers the use of CRNN for classification of the image based on previous images. For sequence classification (sequence to sequence), the encoder-decoder architecture is used. However, as noted, the encoder-decoder architecture within captioning of images uses embeddings from an existing CNN such as VGG or Inception. However, Ariav and Cohen (2019) found the use of these architectures for embeddings for binary classification or multi-classification (with few classes) reduces accuracy as opposed to using embeddings from a smaller CNN, as these architectures (VGG and Inception) are large networks used for large datasets with 1000 classes (Ariav and Cohen, 2019). Taking the findings by Ariav and Cohen (2019) in consideration in this thesis the encoder and encoder-decoder are applied with CNN-RNN and CRNN architectures.

As mentioned previously, accuracy is considered to denote which architecture is better as well as considering the prediction times.

6.1 – CNN-RNN

Typically, the CNN-RNN is associated with caption generation and used with encoder-decoder architecture. In this network, CNN is an existing model utilised to obtain embeddings for the image. These embeddings are then passed to a problem-specific RNN. Vinyals et al. (2015) and Soh (2016) utilised the Inception model to obtain embeddings and created similar variations to the encoder-decoder with these embeddings as input. In this thesis, the embeddings derive from Xception (Chollet, 2017a). Xception is a more recent model based on Inception but with a smaller number of parameters and better performance(accuracy) than its predecessors. Opting for Xception would provide quicker prediction times as the Xception is one of the smallest models available (as noted in Table 6.1). Furthermore, Xception has a higher top-1 and top-5 accuracy than various popular CNN models (such as Inception, VGG and ResNet).

Table 6.1: Comparison of popular CNN models

Model	**Size**	**Top-1 Accuracy**	**Top-5 Accuracy**	**Parameters**	**Depth**
Xception	***88 MB***	***0.790***	***0.945***	***22,910,480***	***126***
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet201	80 MB	0.773	0.936	20,242,984	201

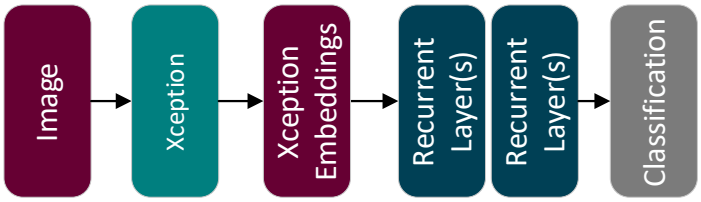


Figure 6.3: Overview of the CNN-RNN model.

Figure 6.3 shows an overview of the CNN-RNN model used for detecting speech. An image is predicted against Xception, with the embeddings saved and utilised as input to the relevant RNN. Regarding the recurrent layer(s)/RNN, the model will be deployed with an encoder and encoder-decoder architecture detailed in the relevant sections below.

6.2 – CRNN

Figure 6.4: The CRNN model.

Choi et al. (2017), Shi et al. (2017), Tao and Busso (2019) and Sharma et al. (2019) implemented a smaller CNN network where the fully connected layers are replaced with recurrent layers. The authors employed a smaller 3 layered convolutional architecture and added recurrent layers to learn sequential data. The CRNN proved to be better (accuracy-wise) than other networks including CNN (Choi et al., 2017; Shi et al., 2017; Sharma et al., 2019). For this thesis, the CNN derives from the CNN experiments conducted previously. In particular, the 5-layered VGG-Like model mentioned in Table 5.5 as it achieved the highest accuracy in the CNN experiments.

Additionally, by combining the 5-Layered VGG-Like model to an RNN, it allows to obtain a fair comparison, as well as examine the effect of combining an RNN to a CNN. Regarding the RNN in Figure 6.4, the Recurrent layer(s) will be deployed with encoder and encoder-decoder architectures.

6.2 – Encoder

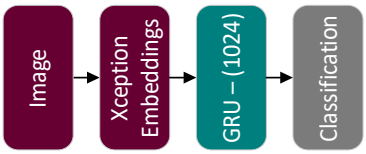


Figure 6.5: Encoder in the CNN-RNN model

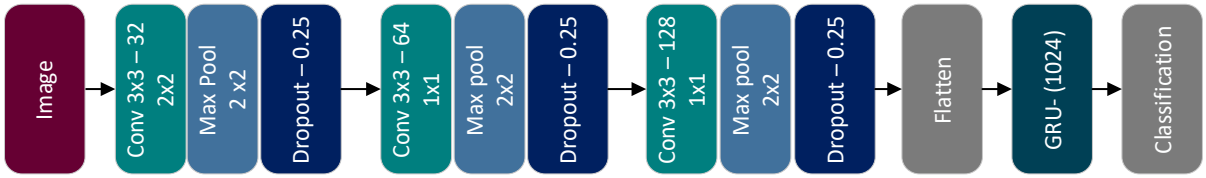


Figure 6.6: Encoder in a CRNN model

In their work, Lim et al. (2016) utilise 3 recurrent layers for their CNN-RNN architecture and utilise 2 layers of recurrent layers in their CRNN model both with 1024 nodes. On the other hand, Han and Kamdar (2018) deployed a single recurrent layer with 256 nodes. For both methods, the mentioned authors have highlighted publishable results especially compared to other methods in their fields.

In this thesis, a single layer of GRU is deployed with 1024 neurons. Various authors including Lim et al. (2016) use LSTM as the recurrent layers, but due to limitation on computational power, GRUs were employed as they require less computation (due to fewer parameters) and can achieve similar results compared to LSTMs (Jozefowicz et al., 2015). Regarding the nodes, 1024 nodes are often used for the recurrent layer as highlighted by work of Lim et al (Lim et al., 2016). Furthermore, increasing the size of the recurrent layer can have an impact on computational memory, which for larger models experimented at later stages, can cause issues regarding computational resources required. As noted in Figure 6.5 and Figure 6.6, the encoder is applied to the CRNN and CNN-RNN architectures.

Within these experiments, sequence length is variable. Different sequence lengths are considered to obtain an understanding of the accuracy and prediction times. The classification for the encoder is based on the last image of the sequence as highlighted in Figure 6.7.

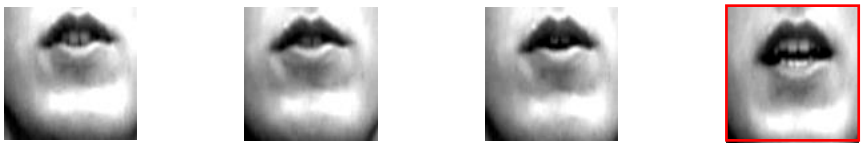


Figure 6.7: Classification based on the last image of the sequence.

6.3 – Encoder-Decoder

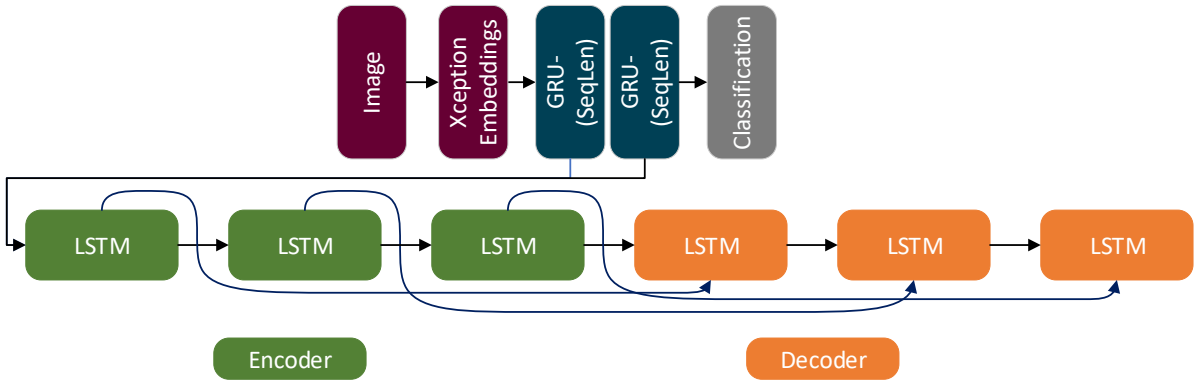


Figure 6.8: Encoder-Decoder in CNN-RNN

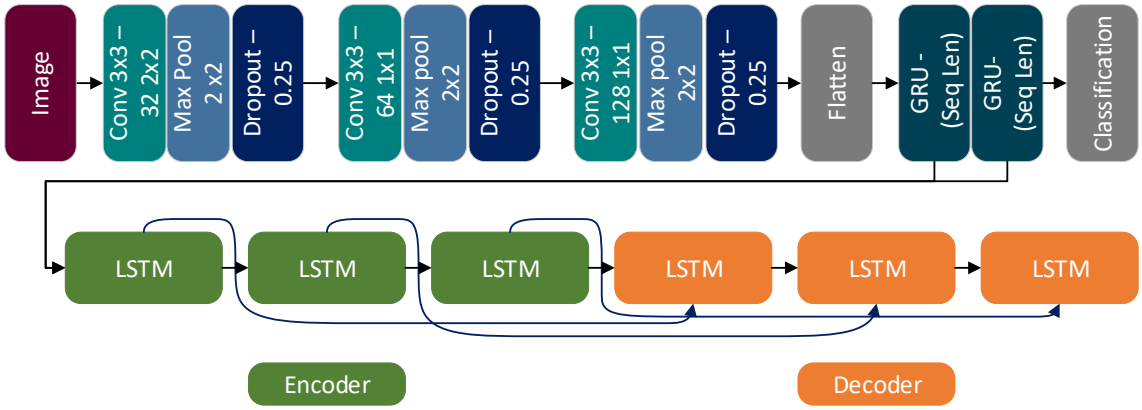


Figure 6.9: Encoder-Decoder in a CRNN

In this thesis, the encoder-decoder is deployed on CNN-RNN (Figure 6.8) and CRNN (Figure 6.9). Similar to work of Vinyals et al. (2015), Venugopalan et al. (2015), and Soh (2016) the thesis considers using an existing CNN model for image embeddings (Figure 6.8). In this case, embeddings from Xception are obtained for all images. The embeddings are then utilised as input to the encoder-decoder model. In the second case, the model is attached to the CNN model from the previous section.

For the encoder-decoder architecture, GRUs with 1024 hidden units are deployed due to reasons mentioned in the previous section (Encoder), but also to keep the models (Encoder and Encoder-Decoder) fair and comparable. In terms of the number of layers/GRUs, this is dependent on the length of the sequence. Similarly, the sequence length is variable - to understand the difference in sequence sizes. The classification in this case is for the whole sequence of images i.e. classifying each image in the sequence.

6.4 – Pre-Processing and Configuration

CNN-RNN

Figure 6.10:pre-processing images for CNN-RNN

To reduce the computation resources required, the CNN-RNN model is executed in two parts. Firstly, the images from the datasets mentioned in CNN (Figure 5.1: Data utilised in CNN networks) are predicted against Xception and the embeddings obtained are saved onto a CSV file. The embeddings are then used as input for the relevant models.

For obtaining the embeddings, each image is loaded and converted to an array to be pre-processed for the prediction against Xception. Regarding the pre-processing for the image, built-in libraries were utilised.

Once an embedding for the image is returned from the model, the value is then saved onto a CSV file along with its label ("speaking"/ "no speaking"). The whole process is repeated for all images in all datasets resulting in three CSV files (train, test and validation). The ordering of the embeddings is crucial and as a result, for every dataset, "no speaking" images were predicted first (with their sequences). Following that, "speaking" images were then predicted. As otherwise invalid sequences can be created with different person's image at a different timestamp from a different category ("speaking"/"no speaking").

The same process is then repeated for labels of the embeddings. Once the data has been allocated to sequences, the sequences are shuffled to avoid overfitting and keep the model robust.

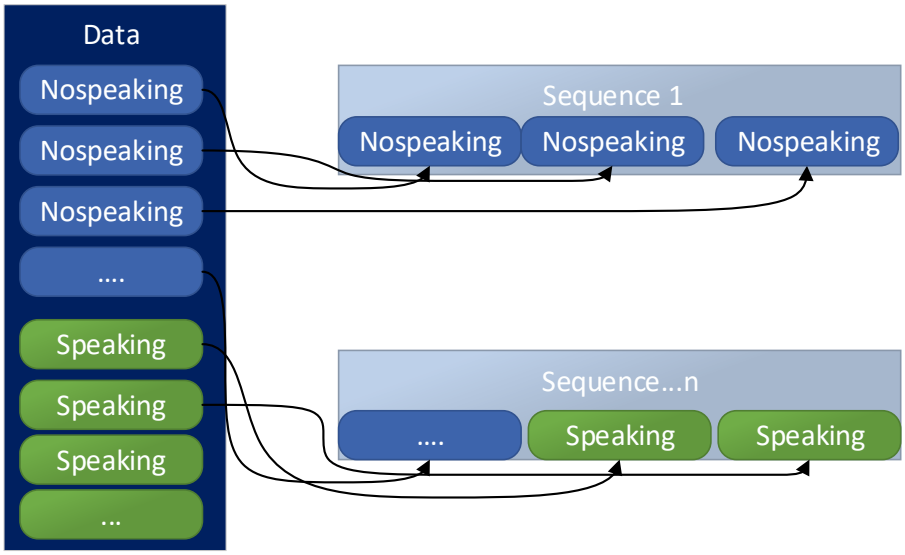


Figure 6.11: allocation of data for sequence creation (assuming sequence length=3)

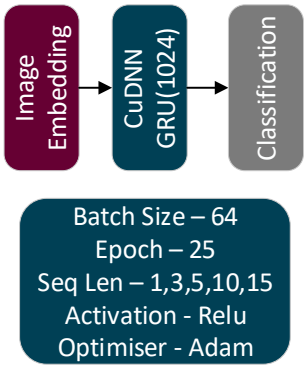


Figure 6.12: Architecture for CNN-RNN with an Encoder

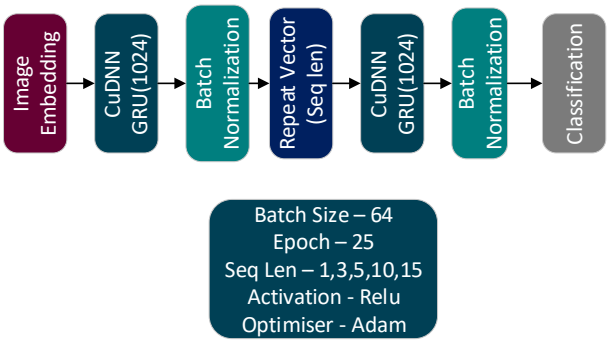


Figure 6.13: Architecture for CNN-RNN with an Encoder-Decoder

Figure 6.12 and Figure 6.13 show the architecture used for the CNN-RNN network. As it may be noted, CuDNNGRUs are utilised as opposed to standard GRU. That is because CuDNNGRUs provide a faster implementation of GRU, up to 7.2x faster than standard GRU. (Braun, 2018). In any case, the hidden neurons are kept the same for the experiments as

used for the CNN-1024. Furthermore, authors including Lim et al have deployed the same number of hidden units. (Lim et al., 2016).

The batch size for CNN-RNN is kept the same as for CNNs at 64, as increasing the batch size any further can cause memory-related issues especially for larger sequences. For the same reason, sequence length for CNN-RNN is up to sequence size of 15. For both networks, different sequence lengths are examined to find the optimal result and understand the difference in performance and accuracy. Batch normalization was particularly useful for the encoder-decoder as suggested by Li et al. (2019) and as a result batch normalization is applied to the encoder-decoder models.

CRNN

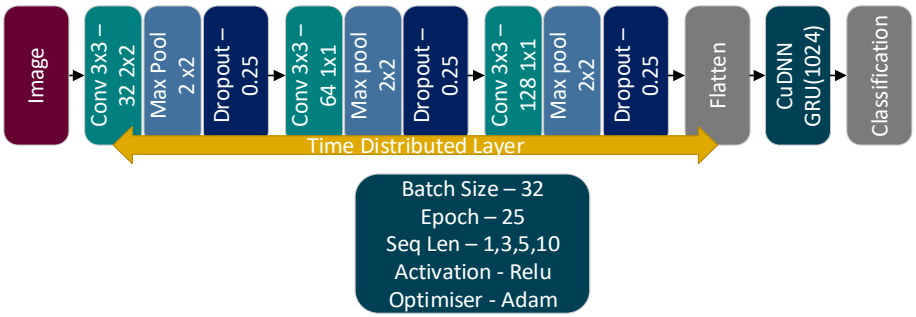


Figure 6.14: Encoder architecture for CRNN

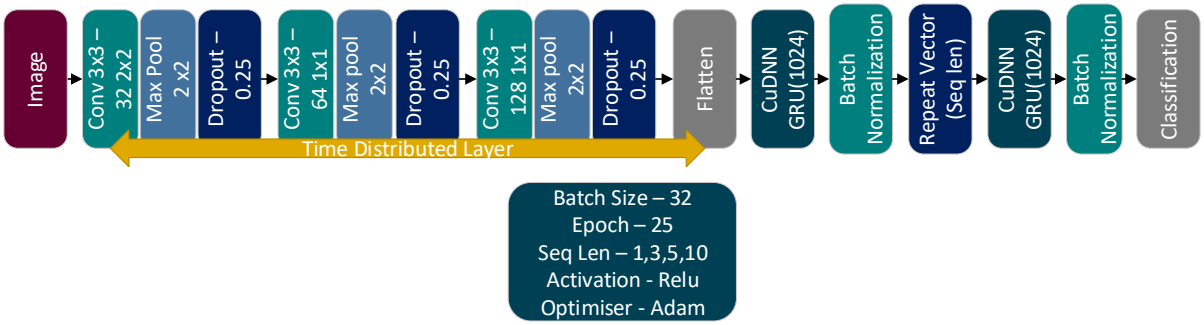


Figure 6.15: Encoder-Decoder in a CRNN architecture

From the data organised in Figure 5.1, the images are shaped and organised similarly to those mentioned in CNN-RNN. For CRNN, images from the datasets (Train, Test and Validation) are first read and converted to arrays before they are shaped and organised in sequences (Figure 6.11). Once the images are converted to arrays, the same pre-processing (shaping and organization of the data) takes place as mentioned for CNN-RNN.

As mentioned previously, the CNN model in the CRNN derives from the previous CNN experiment. The 5-Layered VGG-Like model from CNN in Table 5.5 is attached before the relevant RNN architecture (as noted in Figure 6.14 and Figure 6.15).

In terms of the RNN, the CRNN model is deployed with the encoder and encoder-decoder architectures as mentioned for the CNN-RNN. In the first case, the 5-layered VGG-like model is attached before a CuDNN GRU with 1024 hidden units. In the other case, the encoder-decoder is attached on top of the CNN with the same number of hidden units. As mentioned for the CNN-RNN same parameters are applied. Choosing the same parameters for both CNN-RNN and CRNN allows the results to be fair and comparable.

The CNN model in the above figures is wrapped in Time Distributed Layer (Keras, 2020b). As mentioned previously, Time Distributed Layer allows introducing a fifth dimension (time). Utilising Time Distributed Layer allows for each image to be used as a timestamp in the sequence.

For CRNN models mentioned in Figure 6.14 and Figure 6.15, the batch size is reduced to 32 as deploying batch size of 64 with larger sequences leads to more computational resources necessary. Similarly, the sequence size for the CRNN models is up to sequence size of 10. However, like the CNN-RNN the sequence size is variable.

6.5 – Results

Table 6.2: Overview of the four neural networks experimented with varying sequence lengths, highlighting evaluation scores.

	Encoder (Loss, Accuracy)	**Encoder-Decoder (Loss, Accuracy)**
CNN-RNN		
Sequence Length 1	0.1673,0.9465	0.1701,0.9391
Sequence Length 3	0.1081,0.9697	0.1231,0.9557
Sequence Length 5	0.0616,0.9810	0.0995,0.9642
Sequence Length 10	(0.0473±0.0138), (0.9857±0.0033)	(0.0985 ±0.0115), (0.9694 ± 0.00285)
Sequence Length 15	***(0.0545±0.0291), (0.9867±0.043)***	***(0.0218 ± 0.0013), (0.9929 ± 0.00019)***
CRNN		
Sequence Length 1	0.0935,0.9639	0.1029,0.9611
Sequence Length 3	0.0409,0.9880	0.0447,0.9846
Sequence Length 5	***(0.0302 ±0.0012), (0.9914 ± 0.00045)***	(0.027 ± 0.0027), (0.9915 ± 0.00070)
Sequence Length 10	(0.0555 ±0.00525, (0.9892 ±0.00019)	***(0.014 ± 0.0017), (0.9961±0.00059)***

Table 6.2 highlights the results of the four architectures mentioned. The architectures mentioned include:

1. Encoder - a single-layered RNN where the classification is based on previous/history of images (last image of the sequence).
2. Encoder-decoder - a sequence-based RNN used to classify each image in the sequence.
3. CNN-RNN – using an existing CNN model to obtain embeddings and input for the encoder/encoder-decoder architectures.
4. CRNN - using CNN from previous experiments (5-Layered VGG-Like model) and attaching the encoder/encoder-decoder architectures instead of fully connected layers.

The table shows the accuracies of different networks with varying sequence lengths. All four architectures achieve reasonable results with a marginal difference in accuracy between different networks. The results show that increasing the sequence size has a positive impact on the accuracy of the network, as noted in Table 6.2. However, for sequence-based classification, results show that sequence size needs to at least 3 or more to outperform the CNN classification (97.21% accuracy). At larger sequences (sequences ≥ 5) all architectures outperform the CNN.

Nonetheless, within the problem domain and with resources available, for classification of image based on previous images, the CRNN architecture achieved the highest accuracy of 99.14%. Similarly, for sequence classification (classifying each image in sequence), the CRNN with encoder-decoder achieved the highest accuracy of 99.61%. These results show that in this problem domain use of large CNN architectures for embeddings does reduce accuracy over using smaller CNN architectures.

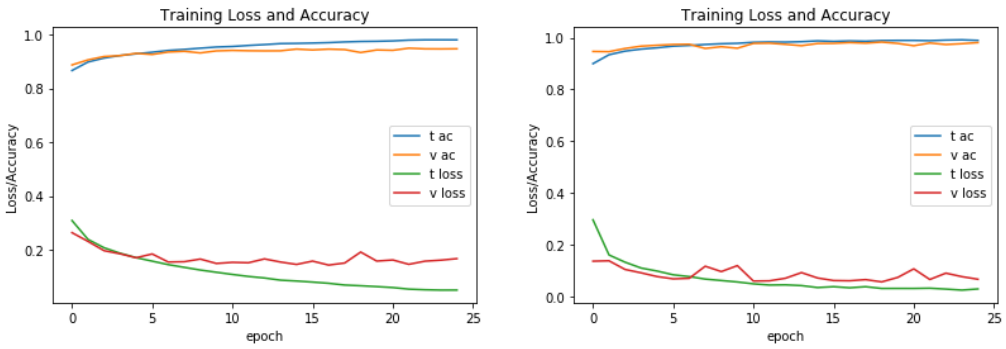
The encoder-decoder architecture does provide the highest network accuracy but requires an increase in history (sequence size). At lower sequence sizes, the encoder models achieve greater accuracy compared to the encoder-decoder models.

6.6 – Discussion

In this section, the use of sequences of images was suggested as a means of classification. In particular, the classification of image based on previous images and classification of sequences of images (sequence to sequence). Existing work showed various approaches in which these classifications can be done. As a result, CRNN (Encoder) was applied to the task of classification of image based on previous images and encoder-decoder for the task of sequence classification.

Existing work showed that use of existing CNNs to obtain embeddings can impact classification accuracy, as existing CNNs are primarily used for large datasets with at least 1000 classes. As a result, the architectures (encoder and encoder-decoder) were implemented with CNN-RNN and CRNN to clarify if the accuracy of the network can be affected by the use of existing CNNs.

One of the positives from these experiments was setting the sequence length as a variable. In doing so, it allowed highlighting that the increase of sequence length has a good effect on accuracy, as noted in Table 6.2. That is because the network has more data in memory (bigger history) for classification. However, in some cases increasing the sequence size can also lead to a decrease or no effect to the accuracy as noted in Table 6.2 (for the CRNN Encoder model for sequence length of 10). In their work, Zhang et al. (2017) argued that increasing the sequence size makes it more difficult for the right prediction of the input, which can be the reason for the slight decrease for CRNN Encoder models at larger sequence sizes.



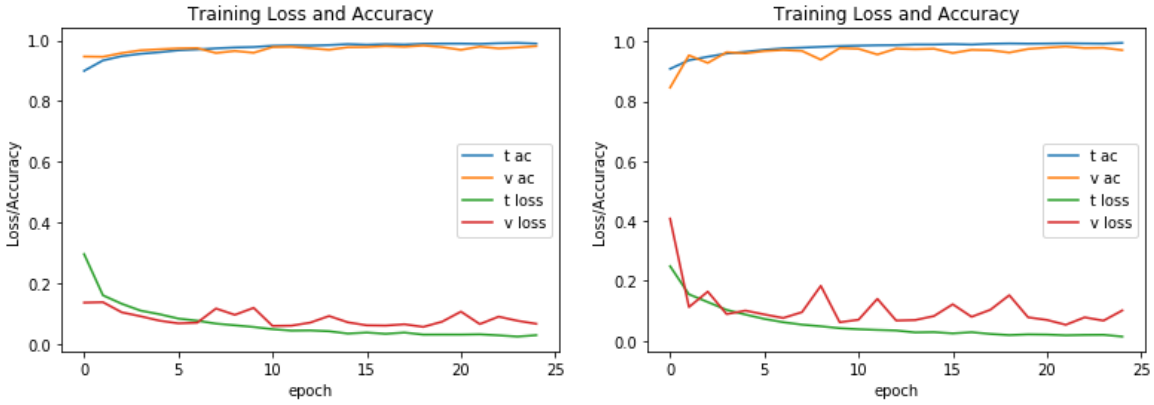
CNN-RNN Encoder-Decoder Sequence Length 1

CNN-RNN Encoder-Decoder Sequence Length 5

Figure 6.16: Comparison of the CNN-RNN encoder-decoder models at a sequence length of 1 and 5

In any case, increasing sequence size was proven to be less vulnerable to overfitting (arguably by a small margin in some cases). As the validation accuracies and loss for the models were more in line with training accuracies and loss (as evident in Figure 6.16).

Shi et al. (2017) suggested batch normalization is useful in overfitting and improving accuracy. As a result, batch normalization was deployed in both encoder and encoder-decoders models. In the case of encoder-decoder, batch normalization proved to not only reduce overfitting but also increase accuracy. However, in the encoder model, the addition of batch normalization did improve the training times but did not improve the accuracy as evident in Figure 6.17. Similar results were also found by Laurent et al. (2016). Thus, batch normalization for the encoder models was disregarded as to keep the CNN-RNN and CRNN architectures comparable, to understand the difference in performance (accuracy, training and predicting times).



without Batch Normalization

Accuracy:0.9810

with Batch Normalization

Accuracy : 0.9710

Figure 6.17: comparison of the performance in using Batch Normalization in encoder architecture for CNN-RNN.

Lim et al. (2016) and Han and Kamdar (2018) in their work deployed recurring dropout within the recurrent networks which they suggest helps to stabilize the network. However, as the models within these experiments were conducted using CuDNNGRU, adding recurrent dropout is not an available feature.

Further experiments were done on different sizes of strides for convolution to make the network smaller which would result in quicker prediction. Change of stride has also been applied by Sharma et al. (2019). (Sehgal and Kehtarnavaz, 2018) compared different strides and found that increasing stride by more than 2, can cause a network to be unstable and a noticeable reduction in classification accuracy. As a result, for CRNN encoder architecture,

all convolutional stride was increased to 2. The performance of this network did reduce training times by half but also caused a reduction of 2% in overall accuracy, in comparison to the architecture of the CRNN encoder in Table 6.3. Consequently, this architecture was disregarded.

Table 6.3: Comparison of the 4 models achieving the highest accuracy in Table 6.2.

Experiment Name	CNN-RNN Encoder – Seq Len:15	CNN-RNN Encoder-Decoder – Seq Len:15
No of Parameters	9,445,378	15,751,170
Test (loss, Acc)	(0.0545±0.0291), (0.9867±0.043)	(0.0218 ±0.0013), (0.9929 ± 0.00019)
Average Train Times (per epoch)	102.68s	173.24s
Prediction Time	31.14ms	31.7ms
Experiment Name	**CRNN Encoder– Seq Len:5**	**CRNN Encoder-Decoder – Seq Len:10**
No of Parameters	**17,402,370**	**23,708,162**
Test (Loss, Acc)	**(0.0302 ±0.0012), (0.9914 ± 0.00045)**	**(0.014 ± 0.0017), (0.9961±0.00059)**
Average Train Times (per epoch)	**325.72s**	**761s**
Prediction Time	**3.7ms**	**5.19ms**

Table 6.3 provides further differences between the four architectures that achieved the highest accuracies with the experiments conducted. One may note, the CRNN models (encoder and encoder-decoder) took twice as long (or more) to train than the CNN-RNN models. However, in the case of CNN-RNN, the embeddings from Xception were obtained beforehand and were used as input through a CSV file.

Nonetheless, the prediction times for the CNN-RNN models are around 30ms which is much longer than the CRNN models which take between 3-5ms to predict. The significant difference between the prediction times is merely down to the embeddings. For CNN-RNN, the model requires embeddings from Xception which a larger network with bigger kernel

sizes of convolution. On the contrary, the CRNN models utilise 3 layers of convolution with smaller kernel sizes. Thus, allowing for faster prediction.

Similarly, the training times between the encoder-decoder models for CNN-RNN and CRNN are significant due to the architectures. As the Encoder models have been deployed with a single GRU as opposed the encoder-decoder models with multiple sets of GRUs (i.e. one set of GRUs for encoding and the other for decoding -depending on sequence length).

Within this section, two widely used Convolution-Recurrent networks were experimented (CRNN with Encoder, and CNN-RNN Encoder-Decoder) with further two architectures (CRNN Encoder-Decoder and CNN-RNN Encoder). These architectures were used for the task of SAD. The classification was based on sequences of images which was categorised in two ways.

Classifying current image based on previous images in which the CRNN encoder architecture was applied and classifying sequence of images in which the encoder-decoder architecture was deployed.

Results (from Table 6.2 and Table 6.3) highlight that use of smaller networks over embeddings from existing CNN architectures, provide better accuracy. Nonetheless, for classification based on previous images the CRNN architecture achieves the accuracy of 99.14% with 3.7ms per prediction. On the other hand, classification of the sequence, the CRNN with encoder-decoder architecture achieves the highest accuracy of 99.61% with 5.19ms for prediction time. Both of these architectures tend to perform better than CNN architectures but require some history. In this problem domain, the experiments showed that a sequence of 3 or greater can outperform the CNN.

Both of these architectures can be used for real-time applications due to their accuracies and prediction times. Where the CRNN achieves an accuracy of 99.14% with 3.7ms per prediction, the encoder-decoder architecture takes almost twice as long for prediction but provides a classification of 10 images with an accuracy of 99.61%. Therefore, choosing between these architectures will be dependent on the requirements and computational resources of the real-time application.

6.7 – Comparisons to the State of the Art

The work presented in this thesis showed accuracies of 97.21% for CNN applied to the classification of still images, 99.14% for CRNN applied to classification based on previous images and 99.61% for CRNN with encoder-decoder architecture applied to the classification of a sequence of images (i.e. classifying all images in sequence).

Within the same dataset (VidTIMIT) Sharma et al. (2019) achieved accuracy 73% at classifying each frame/image with a sequence size of 21. However, the VidTIMIT dataset used by Sharma et al. (2019) was utilised as test dataset whilst the network was trained on other datasets. Within the work of Sharma et al. (2019), their CRNN architecture achieved the highest accuracy of 96.5% for the speaker-dependent scenario with GRID dataset. The work presented in this thesis achieves a higher accuracy (99.61% vs 96.5%) than of Sharma et al. (2019), arguably, with a different dataset and sequence size but within the same speaker-dependent scenario. Sharma et al. (2019) primarily focus on the speaker-independent scenario which their accuracy (92.2%) proves to be a significant improvement compared to similar/previous works.

Sharma et al. (2019) extended the work of Cornu and Milner (2015) who utilised a 3-layered CNN and applied it to the GRID dataset in a speaker-dependent and independent scenarios. Cornu and Milner (2015) achieved 97.66% accuracy for speaker-dependent scenario and 74.58% in the speaker-independent scenario at classifying a single frame/image. Within a similar setting, the work presented in thesis achieved 97.21% for the CNN at classifying a single image and 99.14% for CRNN at classifying image based on previous images. Arguably, the work presented in the thesis used VidTIMIT dataset in comparison to GRID used by Cornu and Milner (2015).

Wang and Wang (2019) introduced landmark pooling with a single layer of CNN followed by landmark pooling and GRU. This architecture was applied to a personally collected dataset. The network takes in 40 frames/images and classifies each frame in the sequence. The highest accuracy obtained by Wang and Wang (2019) is 79.99% with a network that utilised landmark pooling. In comparison to the work in thesis, the CRNN with encoder-decoder architecture achieves 99.61% accuracy at classifying each frame with sequence size of 10 and uses ROI images as input instead of full-face images used by Wang and Wang (2019) within a different dataset.

Tao and Busso (2019) and Ariav and Cohen (2019) incorporated multimodalities (audio and video) where each modality is studied and features from each modality are extracted and combined for classification. Tao and Busso (2019) utilised 3-layered CNN for visual modality and maxout layers to extract features for audio. These features are concatenated and passed to LSTMs. Similarly, Ariav and Cohen (2019) proposed ResNet-18 for video modality and WaveNet encoder for audio. Features from these systems are fused with MCB and passed to LSTMs. Both networks classify each frame within a set sequence size (Tao and Busso (2019) – 11 frames, Ariav and Cohen (2019) – 15 frames). Tao and Busso (2019) achieved the highest accuracy of 93.8% with their multimodal architecture. The video modality achieved 65.5% accuracy with audio modality at 92.7% accuracy. On the other hand, Ariav and Cohen (2019) achieved AUC of 0.97 with multimodal architecture, AUC of 0.94 for video and AUC of 0.92 for audio. In comparison to the work in thesis, the CRNN encoder-decoder achieved 99.61% accuracy at classifying each video frame with sequence size of 10 (within a different dataset).

In terms of prediction times, the comparisons made within the thesis were made to compare the different architectures and impact of certain features such as different classifications (classification of a single image, based on previous images and sequence of images). However, such comparisons are not made to the state of art. This is because the resources needed to compute are limited and less powerful than those used by state of the art (especially those mentioned in chapter 3 – Literature Review). Furthermore, as the performance of such models (in terms of time) is based on the hardware used, often there is little mention of such metric.

From chapter 3 – Literature Review, Sharma et al. (2019) suggest 200-300ms for 7-10 frames (20-42.9ms per frame) for the models to detect silence between words. On the other hand, Wang and Wang (2019) found their model predicting at 198 fps with Nvidia GTX 1080Ti GPU. In comparison, the CRNN with encoder-decoder predicts at 5.19ms with sequence size of 10 images (519 microseconds per image) at 99.61% accuracy. Alternatively, the CRNN encoder with sequence size of 5 images predicts with 3.7ms (740 microseconds per image) at 99.14% accuracy. Arguably, these models would perform better (in terms of time) with more powerful hardware - similar to those used within the state of the art.

7 – Conclusion and Future Work

This thesis highlighted the issues with infotainment systems of Android Auto and Apple CarPlay (i.e. activation of these systems using touch or saying the keyword each time one requires assistance). As a result, this thesis considered speech activity detection as a means of activation. The project aimed to create an artefact that can detect when someone is speaking or not. Existing work presented different approaches for implementing speech activity detection in neural networks but lacked in providing a comprehensive comparison of the models as well as require hardware-intensive resources. As a result, the thesis offered three types of classification: - using still images (CNN), use of previous images (CRNN) and classifying multiple images/sequences.

Experiments showed the use of a sequence of images help classification and provide better accuracy than still images (CNN). Furthermore, the experiments showed CNN takes almost twice as long for predictions. However, these networks do require multiple images in history to outperform a CNN. Moreover, depending on the problem domain, the use of existing popular architectures can affect not only the accuracy but prediction times of the network. With the proposed architectures achieving 99% accuracy, not only do they outperform CNN but provide quicker prediction times. A simpler RNN (encoder) may not achieve the best accuracy than a complex architecture (encoder-decoder) but provides quicker training and prediction times.

Therefore, the work presented in this thesis follows the general notion found in the literature. The key to improving the performance of the network requires utilising the history of images. Furthermore, increasing the sequence size has a positive impact on the performance of the classification. Complex architectures such as Seq2Seq (encoder-decoder) architectures provide better performance than simpler architectures (CRNN). However, the results suggest that the CRNN architecture is marginally outperformed but provides inexpensive computation requirements, quicker training and output predicting times. This is something that is not well documented in the literature.

Based on these findings, this meets the aim and objectives set within the thesis, which is one of the main positives of the work conducted. The goals of the project were to identify an approach that allows detecting speech via images, experimenting various approaches in the literature regarding detection of speech and consequently, finding an approach that provides reasonable accuracy and prediction time. The result of the sequence of images

particularly of the CRNN with Encoder is evident to be a suitable choice for detecting speech based on its accuracy and prediction times which is a key positive.

Another positive element of this project is the comparison of the various architectures and classifications, especially within SAD. These comparisons helped to identify the impact within accuracy, training and prediction times. For example, utilising existing CNN models for embeddings, using complex RNN architectures (Seq2Seq) which may provide the best accuracy but are evident to be more time consuming (training and prediction times) and lastly, the application and effect of Seq2Seq models for SAD. Such comparisons are not popularly found within the literature.

Unquestionably, there are some limitations or aspects that can be improved within the project. One of the limitations is the pre-processing of the images. Arguably, by employing more participants for allocating the dataset (allocating image based on “speech”/ “non-speech”) would be beneficial as it would allow this process to be more efficient and accurate. Alternatively, using a different dataset may be suitable especially one which has the images allocated beforehand based on the classes (“speech”/ “non-speech”). This would eliminate the process of sorting/labelling the data.

Secondly, the extraction of the ROI of the image could be improved by the application of other face detectors such as YOLO as utilised by Sharma et al. (2019). Application of such detection may provide more accurate and valid images from the dataset. However, issues regarding the computation, validity and speed of the extraction of ROI may rise and require consideration.

Lastly, the models/architectures have only been tested in a few cases and within one dataset. One of the common practises within this research area is to train and test the models with different datasets to ensure the validity and robustness of the model. Although due to the computational resources such practise was not applied, it would be better suited for the models to be tested against other datasets to obtain a better and accurate performance of the models. Furthermore, as the project is aimed towards the in-car environment, it would be beneficial if certain actions/scenarios are considered such as yawning or sneezing. The different cases of speech or non-speech data used are limited as data is only recorded for movement of face and start and end of the speech. Other cases such as yawning and sneezing would be a better addition to the dataset as well as for the network to learn such actions, especially considering the in-car environment. Moreover,

due to the resources and scope of the research, the models/architectures have are only applied in an offline setting. Conducting the experiments in the online setting would allow to gain a better assessment of the model and allow to test the model in a live setting.

Therefore, one area of future work regarding the project involves creating/improving the dataset and the processing of the dataset. As the dataset utilised in this project was from recordings conducted in an office, it would be better suited for the project if the dataset derives from the in-vehicle environment. This would allow obtaining data in the real scenarios with different lighting conditions and scenarios, where data is typically not recorded such as yawning or sneezing. Furthermore, it would also be beneficial for sound to be recorded which would aid in justifying the labelling of data and for multimodal classification. Enlarging the dataset would also allow training the network based on subjects/individuals. Therefore, allowing the network to detect speech regardless of the subject's images (speaker-independent) have been trained with the network.

Regarding the processing of the data, recruiting more participants would be beneficial for allocating and labelling the data, ensuring the data is labelled correctly. This would also allow to process the data accurately and efficiently.

The second area of future work involves the inclusion of audio, thus utilising both (video and audio) for classification. This would create a more robust system especially in changing environments (e.g. with good/poor illuminating conditions). Inclusion of audio would also aid in classification for noisy environments or in activities where non-speech can easily be classified as speech (such as yawn or sneeze).

Accomplishing the work above would allow obtaining real-life data than can be utilised to build and test a network that can detect speech within an in-car environment. With the work presented in this thesis, a suitable approach has been suggested. This can be used to develop the network without the need for further experimentation on other models/architectures. After completion of building the network, such a system can be integrated into mobile devices (client-server or as an app) that can be used as standalone or integration with the likes of Android Auto. The network can also be transformed and developed locally (within the app) which could train the network as one uses it. Thus, improving the performance of detection. In the ideal scenario, images would be sent to the network for the detection of speech. The device would obtain a result from the network

("speech"/ "non-speech") which would then either listen for a command or carry out an action. The app would also allow carrying out further tests in a real environment.

Ultimately, the project aimed to compare different approaches within SAD and identify an approach which provides a reasonable accuracy and prediction time. The results of the CRNN are evidence to suggest that this architecture may be a suitable choice for the development of activation of the infotainment system. This is because the architecture can achieve similar accuracy to a complex architecture such as Seq2Seq but is much quicker to train and predict and can be developed with modest hardware requirements. Arguably, such architecture requires to be developed and tested within the in-car environment before it can be suggested for the activation of infotainment systems. However, with the results in this thesis, the performance of the simpler RNN architecture (CRNN with encoder) seems to be a promising fit.

References

Alpaydin, E. (2014) *Introduction to Machine Learning*. 3rd edition. Cambridge, Massachusetts; London, England: The MIT Press.

Apple (2020) *iOS - CarPlay* Available from https://www.apple.com/uk/ios/carplay/ [accessed 3 February 2020].

Ariav, I. and Cohen, I. (2019) An End-to-End Multimodal Voice Activity Detection Using WaveNet Encoder and Residual Networks. *IEEE Journal on Selected Topics in Signal Processing*, 13(2) 265–274.

Bairaju, S.P.R., Sowmya, A. and Garimella, R.. (2017) *Facial Emotion Detection using Different CNN Architectures : Hybrid Vehicle Driving*. Available from https://pdfs.semanticscholar.org/26ee/3aa53b1063c08caecfaba474c12328ded796.pdf?_ga=2.52876141.471058476.1594154248-1859261091.1591983560 [accessed 7 July 2020].

Braun, S. (2018) *LSTM Benchmarks for Deep Learning Frameworks*. Available from http://arxiv.org/abs/1806.01818.

Castrillón, M., Déniz, O., Guerra, C. and Hernández, M. (2007) ENCARA2: Real-time detection of multiple faces at different resolutions in video streams. *Journal of Visual Communication and Image Representation*, 18(2) 130–140.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014) Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. October 2014 Stroudsburg, PA, USA: Association for Computational Linguistics, 1724–1734.

Choi, K., Fazekas, G., Sandler, M. and Cho, K. (2017) Convolutional recurrent neural networks for music classification. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. March 2017 New Orleans, LA, USA: IEEE, 2392–2396.

Chollet, F. (2017a) *Deep learning with Python*. 1st edition. Shelter Island, New York: Manning Publications.

Chollet, F. (2017b) Xception: Deep Learning with Depthwise Separable Convolutions. In:

2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). July 2017 Honolulu, HI, USA: IEEE, 1800–1807.

Clark, A. and Contributors (2020) *ImageEnhance Module - Pillow (PIL Fork) 7.1.2* Available from https://pillow.readthedocs.io/en/stable/reference/ImageEnhance.html [accessed 8 January 2020].

Clark, A. and Contributors (2016) *ImageStat Module - Pillow (PIL Fork) 3.1.x* Available from https://pillow.readthedocs.io/en/3.1.x/reference/ImageStat.html [accessed 8 January 2020].

Cooke, M., Barker, J., Cunningham, S. and Shao, X. (2006) An audio-visual corpus for speech perception and automatic speech recognition. *The Journal of the Acoustical Society of America*, 120(5) 2421–2424.

Cornu, T. Le and Milner, B. (2015) Voicing classification of visual speech using convolutional neural networks. In: *Facial Analysis, Animation, and Auditory-Visual Speech Processing (FAAVSP)*. December 2015 Vienna, Austria: International Speech and Communication Association (ISCA), 103–108.

Donahue, J., Hendricks, L.A., Rohrbach, M., Venugopalan, S., Guadarrama, S., Saenko, K. and Darrell, T. (2017) Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4) 677–691.

Dov, D., Talmon, R. and Cohen, I. (2015) Audio-Visual Voice Activity Detection Using Diffusion Maps. *IEEE Transactions on Audio, Speech and Language Processing*, 23(4) 732–745.

Duchi, J.C., Bartlett, P.L. and Wainwright, M.J. (2011) Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *The Journal of Machine Learning Research*, 12(61) 2121–2159.

Gao, Y., Beijbom, O., Zhang, N. and Darrell, T. (2016) Compact Bilinear Pooling. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 Las Vegas, NV, USA: IEEE, 317–326.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep learning*. 1st edition. The MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.

and Bengio, Y. (2014) Generative adversarial nets. In: *Advances in neural information processing systems 28*. December 2014 Montreal, Canada: Neural Information Processing Systems (NIPS), 2672--2680.

Google (2020) *Android Auto* Available from https://www.android.com/auto/ [accessed 3 February 2020].

Greengard, S. (2015) Automotive systems get smarter. *Communications of the ACM*, 58(10) 18–20. Available from https://dl.acm.org/doi/10.1145/2811286 [accessed 5 February 2020].

Han, L. and Kamdar, M. (2018) MRI to MGMT: predicting methylation status in glioblastoma patients using convolutional recurrent neural networks. *Pacific Symposium on Biocomputing.*, 23(3) 331–324.

He, K., Zhang, X., Ren, S. and Sun, J. (2016) Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. December 2016 Las Vegas, NV, USA: IEEE, 770–778.

Holstein, T., Wallmyr, M., Wietzke, J. and Land, R. (2015) Current Challenges in Compositing Heterogeneous User Interfaces for Automotive Purposes. In: *Human-Computer Interaction: Interaction Technologies*. August 2015 Los Angeles, CA, USA: , 531–542.

Ioffe, S. and Szegedy, C. (2015) Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: *32nd International Conference on Machine Learning, ICML 2015*. February 2015 Lille, France: , 448–456.

Jozefowicz, R., Zaremba, W. and Sutskever, I. (2015) An empirical exploration of Recurrent Network architectures. In: *32nd International Conference on Machine Learning, ICML 2015*. February 2015 Lille, France: , 2342–2350.

Keras (2020a) *Image Preprocessing - Keras Documentation* Available from https://keras.io/preprocessing/image/ [accessed 8 February 2020].

Keras (2020b) *Layer wrappers - Keras Documentation* Available from https://keras.io/layers/wrappers/ [accessed 8 February 2020].

Keras (2020c) *Optimizers - Keras Documentation* Available from https://keras.io/optimizers/ [accessed 8 February 2020].

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012) ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems 25*. May 2012 Nevada, USA: Neural Information Processing Systems (NIPS), 1097--1105.

Laurent, C., Pereyra, G., Brakel, P., Zhang, Y. and Bengio, Y. (2016) Batch normalized recurrent neural networks. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*. March 2016 Shanghai, China: IEEE, 2657–2661.

Li, A., Zheng, C. and Li, X. (2019) Convolutional Recurrent Neural Network Based Progressive Learning for Monaural Speech Enhancement. *arXiv*, Available from http://arxiv.org/abs/1908.10768 [accessed 3 December 2019].

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D.D. and Chen, M. (2014) Medical image classification with convolutional neural network. In: *13th International Conference on Control Automation Robotics and Vision, ICARCV 2014*. December 2014 Singapore, Singapore: IEEE, 844–848.

Lienhart, R. and Maydt, J. (2002) An extended set of Haar-like features for rapid object detection. In: *International Conference on Image Processing*. September 2002 Rochester, NY, USA: IEEE, 900–903.

Lim, W., Jang, D. and Lee, T. (2016) Speech emotion recognition using convolutional and Recurrent Neural Networks. In: *2016 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA)*. December 2016 Jeju, South Korea: IEEE, 1–4.

Manoharan, R. and Chandrakala, S. (2015) Android OpenCV based effective driver fatigue and distraction monitoring system. In: *Proceedings of the International Conference on Computing and Communications Technologies, ICCCT 2015*. February 2015 Chennai, India: IEEE, 262–266.

Oord, A. van den, Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. (2016) WaveNet: A Generative Model for Raw Audio. *ArXiv*, (arXiv:1609.03499v2) 1–15. Available from https://arxiv.org/abs/1609.03499 [accessed 28 January 2020].

Oviedo-Trespalacios, O., Truelove, V. and King, M. (2020) 'It is frustrating to not have control even though I know it's not legal!': A mixed-methods investigation on applications to prevent mobile phone use while driving. *Accident Analysis & Prevention*, 137(105412) 1–11. Available from https://doi.org/10.1016/j.aap.2019.105412 [accessed 3 February 2020].

RoSPA (2018) *Mobile Phones and Driving Factsheet*. Birmingham: The Royal Society for the Prevention of Accidents (RoSPA).

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986) Learning representations by back-propagating errors. *Nature*, 323(6088) 533–536. Available from http://www.nature.com/articles/323533a0 [accessed 6 February 2020].

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. (2015) ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3) 211–252.

Sanderson, C. and Lovell, B.C. (2009) Multi-Region Probabilistic Histograms for Robust and Scalable Identity Inference. In: *Advances in Biometrics, ICB 2009*. June 2009 Alghero, Italy: Springer, 199–208.

Sehgal, A. and Kehtarnavaz, N. (2018) A Convolutional Neural Network Smartphone App for Real-Time Voice Activity Detection. *IEEE Access*, 6 9017–9026. Available from http://ieeexplore.ieee.org/document/8278160/ [accessed 5 January 2020].

Sharma, T., Aralikatti, R.C., Margam, D.K., Thanda, A., Roy, S., Kandala, P.A. and Venkatesan, S.M. (2019) Real Time Online Visual End Point Detection Using Unidirectional LSTM. In: *Interspeech 2019*. 15 September 2019 Graz, Austria: International Speech Communication Association (IISCA), 2000–2004.

Shi, B., Bai, X. and Yao, C. (2017) An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11) 2298–2304. Available from http://ieeexplore.ieee.org/document/7801919/ [accessed 5 January 2020].

Simonyan, K. and Zisserman, A. (2015) Very deep convolutional networks for large-scale image recognition. In: *3rd International Conference on Learning Representations, ICLR 2015*. May 2015 San Diego, CA, USA: International Conference on Learning

Representations (ICLR), 1–14.

Snyder & Associates LLC (2019) *Texting and Driving Accident Statistics* Available from https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/cell-phone-statistics.html [accessed 3 February 2020].

Soh, M. (2016) *Learning CNN-LSTM Architectures for Image Caption Generation*. Available from https://cs224d.stanford.edu/reports/msoh.pdf [accessed 15 September 2020].

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56) 1929−1958.

Strayer, D.L., Cooper, J.M., McCarty, M.M., Getty, D.J., Wheatley, C.L., Motzkus, C.J., Goethe, R.M., Biondi, F. and Horrey, W.J. (2019) Visual and Cognitive Demands of CarPlay, Android Auto, and Five Native Infotainment Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 61(8) 1371–1386. Available from http://journals.sagepub.com/doi/10.1177/0018720819836575 [accessed 3 February 2020].

Sutskever, I., Vinyals, O. and Le, Q. V. (2014) Sequence to sequence learning with neural networks. In: *Advances in Neural Information Processing Systems 27*. December 2014 Montreal, Canada: Neural Information Processing Systems (NIPS), 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015) Going deeper with convolutions. In: *2015 IIEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015 Boston, MA, USA: IEEE, 1–9.

Tang, Y., Huang, Y., Wu, Z., Meng, H., Xu, M. and Cai, L. (2016) Question detection from acoustic features using recurrent neural network with gated recurrent unit. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. March 2016 Shanghai, China: IEEE, 6125–6129.

Tao, F. and Busso, C. (2019) End-to-end audiovisual speech activity detection with bimodal recurrent neural models. *Speech Communication*, 113 25–35. Available from https://www.sciencedirect.com/science/article/pii/S0167639318303121 [accessed 28 January 2020].

Tummala, M. (2019) Image Classification Using Convolutional Neural Networks. *International Journal of Scientific and Research Publications (IJSRP)*, 9(8) 382–385. Available from http://www.ijsrp.org/research-paper-0819.php?rp=P928963 [accessed 28 January 2020].

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T. and Saenko, K. (2015) Sequence to sequence - Video to text. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. December 2015 Santiago, Chile: IEEE, 4534–4542.

Vinyals, O., Toshev, A., Bengio, S. and Erhan, D. (2015) Show and tell: A neural image caption generator. In: *2015 IIEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015 Boston, MA, USA: IEEE, 3156–3164.

Wang, B. and Wang, X. (2019) Are You Speaking: Real-Time Speech Activity Detection via Landmark Pooling Network. In: *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*. May 2019 Lille, France: IEEE, 1–5.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C. and Xu, W. (2016) CNN-RNN: A Unified Framework for Multi-label Image Classification. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 Las Vegas, NV, USA: IEEE, 2285–2294.

Xiong, X. and De la Torre, F. (2013) Supervised Descent Method and Its Applications to Face Alignment. In: *2013 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*. June 2013 Portland, OR, USA: IEEE, 532–539.

Zhang, Y., Sun, X., Ma, S., Yang, Y. and Ren, X. (2017) Does Higher Order LSTM Have Better Accuracy for Segmenting and Labeling Sequence Data? *arXiv*, (arXiv:1711.08231v3). Available from http://arxiv.org/abs/1711.08231 [accessed 4 January 2020].

Zuo, Z., Shuai, B., Wang, G., Liu, X., Wang, X., Wang, B. and Chen, Y. (2015) Convolutional recurrent neural networks: Learning spatial dependencies for image representation. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. June 2015 Boston, MA, USA: IEEE, 18–26.

Appendix

$$Dell\ XPS\ 15\text{”}\ 9550$$

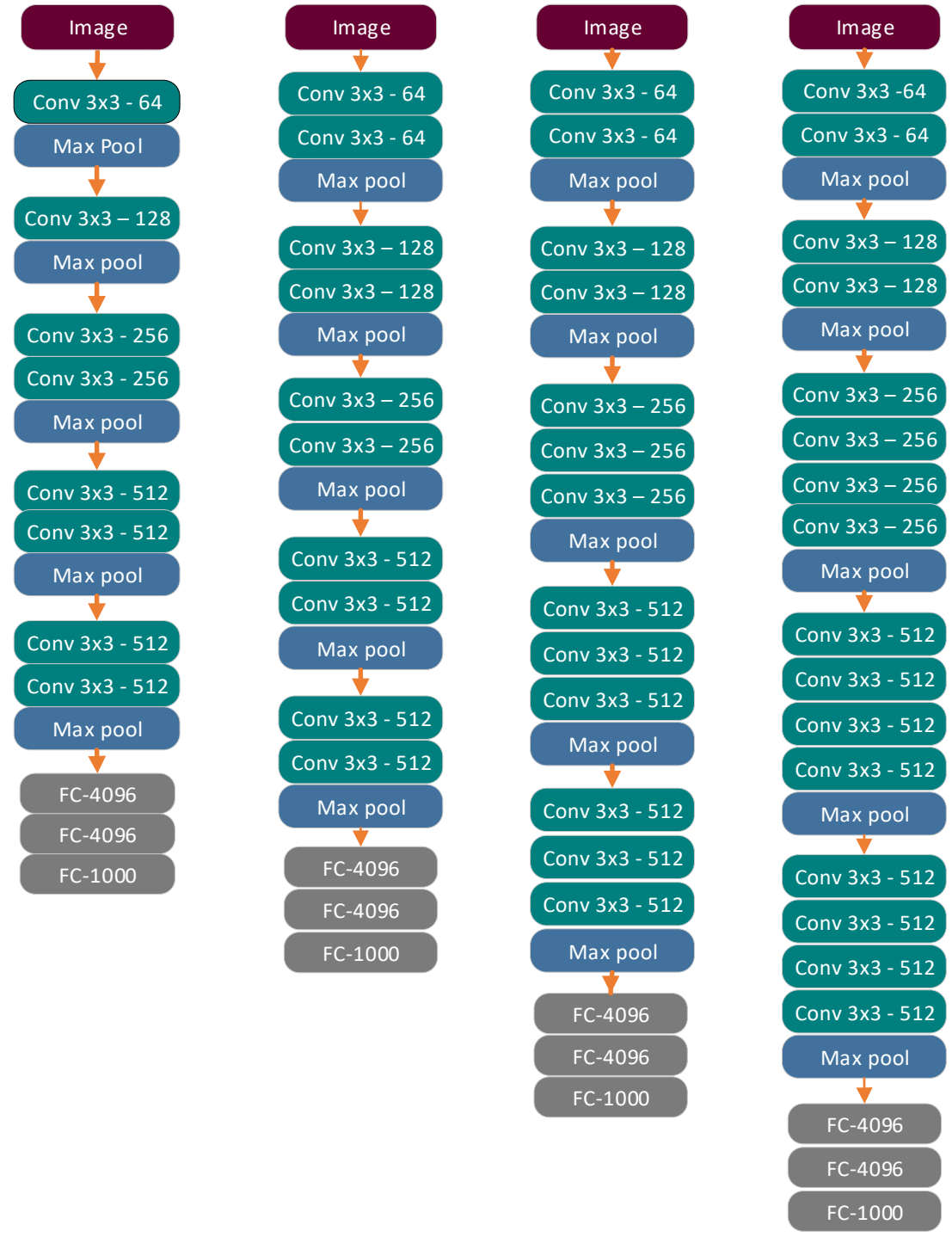
$$CPU: Intel\ Skylake\ i7-6700HQ$$

$$GPU: Nvida\ GTX\ 960M\ 2GB\ DDR5$$

$$RAM: 16GB\ DDR4\ 2133\ MHz$$

$$Hard\ drive: 512\ PCie\ NVMe\ SSD$$

Figure 8.1:Specification of the machine in which experiments were run on



VGG-11	VGG-13	VGG-16	VGG-19
133 Million	133 Million	138 Million	144 Million
Parameters, 10.4%	Parameters, 9.9%	Parameters, 8.8%	Parameters, 9.0%
Top - 5 Error Rate	Top-5 Error Rate	Top-5 Error Rate	Top-5 Error Rate

Figure 8.2: comparison of VGG architectures.