5-2016

Image Description using Deep Neural Networks

Ram Manohar Oruganti
ro5963@rit.edu

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritsscholarworks@rit.edu.
Image Description using Deep Neural Networks

by

Ram Manohar Oruganti

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Raymond Ptucha
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
May 2016

Approved By:

_____________________________________________        ___________
   Dr. Raymond Ptucha
   Primary Advisor – R.I.T. Dept. of Computer Engineering

______________________________________________        _________
   Dr. Andreas Savakis
   Secondary Advisor – R.I.T. Dept. of Computer Engineering

______________________________________________
   Dr. Christopher Kanan
   Secondary Advisor – R.I.T. Carlson Center for Imaging Science
Dedication

Dedicated to the people working towards developing intelligent machines for a smarter and safer tomorrow and those who encourage and support them.
I would like to thank my advisor Dr. Raymond Ptucha, for entrusting me with this research and for his constant guidance, support and supervision. I would like to express my gratitude to Dr. Andreas Savakis and Dr. Christopher Kanan, whose valuable comments and reviews helped enhance the quality of my thesis work. I would also like to thank Shagan Sah and Suhas Pillai for their help with Caffe framework. I would like to appreciate the constant support of Dr. Shanchieh Yang and Dr. Sonia Lopez-Alarcon and thank them for standing by my decisions while I was trying to figure out my research interests. Also, this work wouldn’t have been possible without the support of Rick Tolleson, Emilio Del Plato and the Computer Engineering Department who have provided me with the resources and technical support for this thesis work.
Abstract

Current research in computer vision and machine learning has demonstrated some great abilities at detecting and recognizing objects in natural images. Current state-of-the-art results in object detection, classification and localization in ImageNet Challenges have the validation accuracy for top 5 predictions for classification to be at 3.08% while similar classification experiments run by trained humans report an accuracy of 5.1%. While some people might argue that human accuracy is a function of training time it can be said with great confidence that automated classification models are at least as good as trained humans in classification problems. The ability of these models to analyze and describe complex images, however, is still an active area of research.

Image description is a good starting point for imparting artificial intelligence to machines by allowing them to analyze and describe complex visual scenes. This thesis work introduces a generic end-to-end trainable Fusion-based Recurrent Multi-Modal (FRMM) architecture to address multi-modal applications. FRMM allows each input modality to be independent in terms of architecture, parameters and length of input sequences. FRMM image description models seamlessly blend convolutional neural network feature descriptors with sequential language data in a recurrent framework. In addition to introducing FRMMs, this work also analyzes the impact of varying activation functions and vocabulary size. For training and testing Flickr8k, Flickr30K and MSCOCO datasets have been used, demonstrating state-of-the-art description results.
# Table of Contents

Image Description using Deep Neural Networks ............................................. i

Dedication ........................................................................................................ ii

Acknowledgements .......................................................................................... iii

Abstract ........................................................................................................... iv

Table of Contents .............................................................................................. v

List of Figures ................................................................................................... viii

List of Tables ..................................................................................................... x

Chapter 1  Introduction .................................................................................... 1

  1.1. Contributions .......................................................................................... 3

  1.2. Overview of Thesis ................................................................................ 4

Chapter 2  Motivation from previous work ..................................................... 5

Chapter 3  Artificial Neural Networks ............................................................. 13

  3.1. Activation Functions ............................................................................ 14

  3.2. Cost Functions ...................................................................................... 19

  3.3. Determining the weights ..................................................................... 23

  3.3.1 Initializing the weights .................................................................... 23

  3.3.2 Updating the weights ...................................................................... 24

  3.4. Avoiding overfitting ........................................................................... 30

  Dropout ....................................................................................................... 32

Chapter 4  Feedforward Neural Networks ...................................................... 35
6.1.3 Vanishing and exploding gradients ........................................... 57
6.1.4 Inability to capture long term dependencies .............................. 58
6.2. Long Short Term Memories ....................................................... 60
  6.2.1 Forward propagation in LSTM .............................................. 62
  6.2.2 Avoiding vanishing gradient problem .................................... 63
Chapter 7 Activation and vocabulary based experiments ...................... 65
Chapter 8 FRMM architectures ....................................................... 71
  8.1. FRMM architectures ............................................................. 73
  8.1.1 Image Captioning through FRMMs ........................................ 73
  8.2. FRMM Results ................................................................. 75
  8.2.1 Datasets ............................................................................. 75
  8.2.2 Implementation details ......................................................... 76
  8.2.3 Experimental Results ........................................................... 76
  8.2.4 Sample Captions ............................................................... 80
Chapter 9 Conclusions and Future Work ......................................... 83
References ............................................................................... 84
List of Figures

Fig 1. An illustration of FRMM architecture using a video description example . . 2
Fig 2. Architecture of image description model proposed by Karpathy et al. [13] 7
Fig 3. RCNN/BRNN based word and image feature vector embedding. [13] ....... 8
Fig 4. Architecture of Neural Image Caption generator. [14] .......................... 10
Fig 5. LRCN variants: Unfactored and factored LRCN architectures. [9] ........... 11
Fig 6. Fusion models proposed by Karpathy et al. [15] ................................. 12
Fig 7. Basic architecture of an ANN with input, hidden and output layers .......... 14
Fig 8. Plots of various activation functions. ................................................. 18
Fig 9. Gradient descent illustration. [23] ...................................................... 25
Fig 10. A pictorial depiction of issues with small and large learning rates ......... 26
Fig 11. A scenario where the solution is stuck at a local minima .................. 28
Fig 12. Momentum helping weights from being stuck at a local minima. .......... 29
Fig 13. Illustration of dropout applied to a ANN (a), resulting in (b). .......... 33
Fig 14. Neuron's behavior during training and testing time with dropout .......... 34
Fig 15. A FNN with one input, one output and one hidden layer .................. 35
Fig 16. Comparison of FNN and CNN. [32] ................................................. 40
Fig 17. Illustration of the receptive field concept for a 32x32x3 image. [32] ...... 41
Fig 18. Output computation for CONV illustrated using a 5x5x3 input. [32]..... 45
Fig 19. Downsampling the output size through pooling. [32] .......................... 48
Fig 20. CNN architecture for a typical image classification problem. [32] ...... 50
Fig 21. RNN displaying the characteristic cyclical connections. ................. 52
Fig 22. SRN architecture with one hidden layer ........................................... 54

Fig 23. Illustration of forward propagation in SRN .................................... 55

Fig 24. Illustration of inability to capture long-term dependency using a SRN based activity classifier that uses video input .................................................................. 59

Fig 25. Preserving long term dependencies using input and output gates ........ 60

Fig 26. The internal architecture of a LSTM cell ........................................ 61

Fig 27. Illustration of BPTT for a LSTM cell ............................................. 64

Fig 28. Images and corresponding captions from Flickr8k dataset. [40] ....... 66

Fig 29. SRN predictions for RELU, tanh and sigmoid activations on Flickr8k .. 68

Fig 30. Illustrations of FRMM architectures with image captioning examples. .. 72

Fig 31. Some of the accurate image descriptions generated by AFRMM .......... 80

Fig 32. FRMM generated captions that are not very accurate ..................... 81
List of Tables

Table 1. Evaluation of the performance for various activation functions in SRN based architecture for Flickr8k. ................................................................. 66

Table 2. Comparison of performance between tanh and RELU in LSTM based architecture for Flickr8k. ................................................................. 68

Table 3. Comparison of performance between tanh and PL in LSTM based architecture for Flickr8k. ................................................................. 69

Table 4. Relation between word frequency threshold based vocabulary elimination and performance in LSTM based architecture for Flickr8k. ........................................... 70

Table 5. Evaluation of the performance of image description model with various FRMM architectures on the MSCOCO dataset. ................................................................. 77

Table 6. Evaluation of the performance of image description model when image feature descriptor is extracted from layers fc6, fc7 and fc8 of the CNN on the MSCOCO dataset. ...................................................................................................................... 78

Table 7. Performance of image description model with two-layered LSTM architectures in each stage. ...................................................................................................................... 78

Table 8. Comparison of our approach with other proposed methods using BLEU and METEOR scores for MSCOCO dataset................................................................. 79

Table 9. Comparison of our approach with other proposed methods using BLEU and METEOR scores for Flickr30K dataset. ................................................................. 79
Chapter 1   Introduction

Accurate text annotation of image and video content enables more efficient search and retrieval, can aid visual understanding in medical, security, and military applications, and can even be used to describe pictorial content to the visually impaired. Uncertainties about salient content, main subject detection, object recognition, action detection, and scene understanding make this a challenging problem. Despite the difficult nature of this task, computer vision and natural language processing researchers have made significant strides in this area. This thesis work builds upon these previous successes, and introduces a new framework that simultaneously addresses the diverse input modalities, while producing state-of-the-art results.
In the past five years, supervised convolutional models have forever changed the computer vision and machine learning landscape. Due to the recent introduction of large supervised datasets [1] and accelerated training models using Graphic Processing Units (GPUs) [2], the traditional pairing of hand crafted low level vision features with complimentary classifiers has been bested by Convolutional Neural Networks (CNNs) [3]. CNNs are deep feed forward networks based upon a hierarchy of abstract layers which simultaneously learn the low level features and classifier. These networks have been shown to equal the performance of neurons in the primate inferior temporal cortex, even under difficult conditions such as pose, scale, and occlusions [4]. CNNs have won competitions in traffic sign detection, house number detection, handwriting recognition, pedestrian detection, object recognition, speech recognition, breast cancer detection, and many more.

A number of works have extended the image recognition framework to video frames. Motivated by the need to learn temporal sequences, Recurrent Neural Networks (RNNs) enable solutions to basic problems such as activity recognition and

Fig 1. An illustration of FRMM architecture using a video description example
object trajectory prediction. In practice, RNNs suffer from vanishing and/or exploding gradients- a problem where the backpropagation of an error signal over several iterations will diminish quickly (e.g. several successive multiplies by a value < 1) or explode quickly (e.g. several successive multiplies by a value > 1). The Long Short Term Memory (LSTM) models [5] solved the gradient problem by replacing the traditional artificial neuron with a memory cell containing long and short term non-linear capabilities. The LSTM’s incredible power was first realized in the speech and natural language processing domains [6], and more recently to the annotation of image and videos [7-12]. LSTMs are a natural fit for temporal sequences of varying lengths and can be trained using standard back propagation.

1.1. Contributions

Our architecture takes advantage of these powerful CNN and LSTM architectures and uses a multi-modal shared representation for learning a combination of data sequences. In this thesis work a set of end-to-end trainable fusion based recurrent architectures for multi-modal learning, called FRMMs, are introduced. These FRMMs are used in the context of image captioning where image features and vocabulary are learned in independent stages and then mapped to a shared representation in the fusion stage. This allows for the fusion of multiple arbitrary length sequential streams, which to the best of my knowledge, is the first study of its kind. The independent stages also allow for doing away with the use of shared parameters for learning different modalities and allowing each modality to be learned using an architecture that best captures the
features of that modality. In addition, this thesis work analyzes the impact of various activation functions on the performance of RNNs and LSTMs and the impact of vocabulary size on the effectiveness of the image description model. Furthermore, the generic nature of the FRMM based architectures allow them to be deployed for a multi-modal learning problem. Fig 1 shows an illustrative example of an FRMM model that is deployed for video description.

1.2. Overview of Thesis

The rest of the thesis is organized as follows. Chapter 2 briefly describes some recent research works in this area that have motivated this work and how this work adapts and builds upon those techniques. Chapter 3 introduces Artificial Neural Networks (ANN). Chapter 4, Chapter 5 and Chapter 6 cover specific ANN architectures namely Feedforward Neural Networks, Convolutional Neural Networks and Recursive Neural Networks. Chapter 7 presents the findings for activation function and vocabulary size based experiments while Chapter 8 introduces the FRMM architectures and compares its performance with other state of the art image description works. The conclusions and ideas for future work are presented in Chapter 9.
Chapter 2  Motivation from previous work

Recent research [9, 13-15] has demonstrated state-of-the-art image captioning results using deep learning technique. These methods analyze visual information, recognize and classify objects and actions, and describe both still and video frames through captions. All these works use a supervised learning scheme where images with corresponding captions are used to train the network. Convolutional Neural Networks (CNNs) are deployed for visual feature extraction and recursive neural network based architectures, either a simple recursive network or a Long-Short Term Memory (LSTM) based architecture are used to learn the language model and then generate descriptions. This thesis work draws inspiration from their work, adapts some of the concepts used in the works and builds upon those techniques to help overcome their limitations in an attempt to improve results. This section briefly walks the readers through the approaches employed in the aforementioned researches and describes the concepts adapted.
The first work being described in this section is by Karpathy et al. [13]. The basic architecture for his model is shown in Fig 2. It uses a CNN that has been pretrained on ImageNet [16] and fine-tuned on the datasets in ImageNet challenge [1]. In addition a Simple Recurrent Network (SRN) is used to function as a caption generator. During training the SRN is fed the image feature descriptor from the CNN in addition with the keyword START at the first time instance, followed by each word in the ground truth image caption from the training data at every time instance, along with the hidden state from the previous time step. After training with enough exemplars, the SRN learns the language semantics and predict the next word with a good accuracy based on either the previous word or the image features through the weight updates. During testing, the image feature descriptor extracted from the CNN is used as the first input to the SRN along with the keyword START. The first word of the image caption is predicted based on the image feature descriptor. The next prediction is made based on the previous prediction as input along with the previous hidden state. The process continues until the end of sentence has been encountered. Fig 2 demonstrates how the proposed architecture makes a prediction on a test image that has a picture of a man wearing a straw hat.
Fig 2. Architecture of image description model proposed by Karpathy et al. [13]

Before the training and testing, Karpathy preprocessed the words by mapping them into the same vector space as the image feature vector extracted from the CNN such that the dot product of a word vector with its corresponding image vector is maximized. This has been achieved through an RCNN as proposed by Girshick et al in [17], which identifies the top nineteen regions/objects in an image and generates twenty image feature vectors by passing these nineteen regions along with the entire image through a CNN. A SRN architecture, called Bidirectional Recursive Neural Network (BRNN) [18] is used to map each word into the same vector space as the image feature vector based on the contextual information surrounding the word in both directions and the feature vector of the word’s corresponding image.
This is illustrated in Fig 3, where a picture of a dog catching a frisbee is passed to the RCNN. The example has three regions of interest: dog, frisbee and the entire image. The words *dog*, *catch* and *leaps* correlate well with the image feature vector of *dog*, maximizing the image-sentence score, which is the dot product of image feature vector and word vector. Similarly, the image feature vector *frisbee* has a high correlation with the word *frisbee*. Higher scores are indicated in the image with lighter shades, while darker shades indicate lower image-sentence scores. Because of this preprocessing step for word vectors and the freezing of all the layers in CNN, which is the image feature extracting stage, this model is not end-to-end trainable. Also, while multi-modal embedding is an important start as other researches show, learning it offline through a separate model is probably unnecessary.
The second research work that is very relevant to this thesis research is the Neural Image Caption Generator (NIC) by Vinyals et al. [14]. It is similar to [13] in that it uses a CNN, mapping the word vector into the same dimensional vector space as the image vector. The difference is that this model is end-to-end trainable and does the word vector \( (S_0 \text{ to } S_{N-1}) \) embedding into image feature vector space by learning the weight vector \( W_e \) within the NIC. It also uses a LSTM architecture in place of a SRN architecture to improve performance and a beam search algorithm that keeps track of \( k \) possible image captions and then picks the sentence with the least loss, which is the absolute value of sum of log probabilities of each word in the caption given the previous words. This has apparently helped them improve the BLEU scores of NIC by an average of two points. They also use a different CNN, a variant of GoogLeNet [19], the best performing model from the ILSRVC 2014 classification competition. Their results demonstrate advantages LSTM based networks and end-to-end training offer, over SRNs based caption generators and the superfluousness of having a separate BRNN for a word vector embedding into an image vector space. Fig 4 shows the architecture of CNN and LSTM based NIC generator used by [14].
The third research work in image description using deep learning is the Long-term Recurrent Convolutional Networks (LRCNs) proposed by Donahue et al. [9]. It shares the similar features such as end-to-end trainability and CNN/LSTM based caption generators. It is different from the previously discussed research in the sense that it doesn’t constrain the word vector to be in the same dimensional vector space as the image feature vector and it introduces interesting multi-layered LSTM architectures. It employs a pre-trained CNN architecture on ImageNet dataset [16] as proposed by Krizhevsky et al. in [2], and fine-tunes the fully-connected layers of the CNN using the end-to-end trainability of LRCN architecture. In addition to LRCNs that have just a single layer LSTM, similar to NIC, it has two layered LSTM architectures where either both layers are used to train on both image and word vector inputs (unfactored LRCN) or only the second layer is used for both
image and word vectors while the first layer is exclusively reserved for the word vector inputs (factored version). These LRCN variants are depicted in Fig 5.

![Diagram of LRCN variants: Unfactored and factored LRCN architectures.](image)

The results of LRCN indicate that factored LRCN yields better captions than unfactored LRCN. This has led to the idea of having independent learning stages for image feature vectors and word vectors in this research, in order to avoid sharing parameters for disparate modalities. On top of this independent learning stages for each modality, a LSTM based architecture has been used to learn image features in FRMMs.
The concept of *fusion* and combining learning from two independent streams has been borrowed from [15] where visual information from two distinct set of frames is fused temporally into one layer. This work extends that concept of fusion of disparate input modalities and uses the LSTM based fusion architecture as multi-modal applications often have a temporal dependency on output from previous time steps. Fig 6 shows the fusion models where the red, green, blue and yellow boxes indicate convolutional, normalization, pooling and softmax layers of CNNs respectively.
Chapter 3    Artificial Neural Networks

Artificial Neural Networks (ANNs) are a computer emulation of a simplified model of the billions of neurons in the human brain. Each node within an ANN mimics a neuron of the human brain and is also often referred to as a neuron. This chapter only talks about the functionality of ANNs in the context of supervised learning where the input and corresponding outputs from the training data are used to help the network understand the relationship between input and output. In order to recreate the functionality of the nervous system and capture the relationship between input and output, each node in an ANN is connected to multiple input and output nodes through weighted connections. These weights are learned over time, during backpropagation where the ANN modifies the weights based on how closely the predicted outputs match the ground truths of the testing data. This process will be elaborated in the following sections. The firing of the neurons in the brain is caused by sharp electrical spikes. This firing is recreated in ANNs through the use of non-linear activation functions like sigmoid, tanh and rectified linear units within each node. A collection of such nodes where each node is connected to a number of other nodes forms an ANN. The nodes that are connected to sensory and responsive parts of the system form the input and output layers of the network while the nodes that are only connected to
other nodes in the network are called hidden nodes and form the hidden layer. Fig 7 illustrates the basic architecture of an ANN. Based on how the nodes within each layer of an ANN interact with one other, ANNs can be broadly classified into Feedforward Neural Networks (FNN) and Recurrent Neural Networks (RNN). In the following discussion, the word neuron refers to a node in an ANN and not a neuron of the human brain and the terms node and neuron will be used interchangeably.

![Basic architecture of an ANN with input, hidden and output layers.](image)

3.1. **Activation Functions**

The previous sections talk about using activation functions to modulate inputs, but they don’t describe what these functions are. This section describes the most common activation functions used in ANNs.
**Linear**

A linear activation function, as the name suggests is a linear function of the form:

\[ f(t) = c \times t \]  \hspace{1cm} (1)

where \( c \) is a constant. Linear activation functions are seldom used in practice, and were used only during the infancy of neural computing, as an ANN using a linear activation can be reduced to a linear multi-variate regression model and thus can’t be used to generate non-linear boundaries between classes. The activation functions that follow are all non-linear.

**Binary Threshold**

The binary threshold is a step function and is helpful in classifying the inputs into two distinct classes. It is mathematically represented as shown in (2). The numbers \( p \) and \( n \) are real numbers usually with values 1 and -1/0 while \( \text{threshold} \) is usually a positive real value.

\[ f(t) = \begin{cases} 
  p & \text{for } t \geq \text{threshold} \\
  n & \text{for } t < \text{threshold} 
\end{cases} \]  \hspace{1cm} (2)

Binary Threshold is one of the first non-linear activation functions to be used in ANNs called perceptrons. However as it is a non-differentiable, it is no longer used in ANNs. The importance of a differentiable non-linear activation function will become more evident after the back propagation concept is introduced. To overcome this problem S-
shaped non-linear functions like logistic sigmoid and hyperbolic tangent are employed. They are non-linear and differentiable.

---

**Logistic Sigmoid**

The logistic sigmoid, simply referred to as sigmoid, is a non-linear S-shaped function defined by the mathematical equation:

$$ f(t) = \frac{1}{1 + e^{-t}} $$  \hspace{1cm} (3)

It is a very popular activation function and has been widely used in many ANN solutions. As it can be seen, with a value of 0.5 at t=0, this function is not zero centered and is always positive, leading to both positive and negative exemplars having positive outputs in classification paradigms. While this can be rectified with the right bias, it would be preferable for an activation function in classification models to have positive values for positive exemplars and negative values for negative exemplars. Hence as sigmoid, is not zero-centered and for other reasons discussed in the following sections, like the vanishing gradient problem, this activation function is no longer favored and is being replaced by hyperbolic tangent and rectified linear unit.

**Hyperbolic Tangent**

16
The hyperbolic tangent (tanh) is a non-linear S-shaped function like logistic sigmoid, which unlike sigmoid is zero-centered, with a steeper rise which can be especially helpful in classification models to reduce the number of misclassified samples. The tanh function is differentiable, unlike the binary threshold. Mathematically, the hyperbolic tangent is represented as:

\[
f(t) = \frac{e^{2t} - 1}{e^{2t} + 1}
\]

While tanh offers many benefits over the previously discussed activation functions, it is still susceptible to vanishing gradient problem, although much less likely than sigmoid.

**Rectified Linear Units (RELU)**

A rectified linear unit is a non-linear, differentiable activation function that is not as susceptible to the vanishing gradient problem, an issue that shall be discussed in the following sections. Mathematically a RELU is represented as

\[
f(t) = \max(0, t)
\]

RELU, is the most popular activation function in deep neural networks, an ANN with many hidden layers, due to its ability to circumvent the vanishing gradient problem.

**Piecewise linear**
A piecewise linear function is a non-linear, S-shaped, differentiable activation function which may not be as susceptible to the vanishing gradient problem. Mathematically a piecewise linear function is represented as

\[
f(t) = \begin{cases} 
1 & \text{if } t > 1 \\
t & \text{if } -1 \leq t \leq 1 \\
-1 & \text{if } t < -1
\end{cases}
\]  

(6)

Fig 8 shows all the activation functions discussed so far in the same order as discussed from top right to bottom left.
3.2. Cost Functions

In order to determine the accuracy and efficiency of an ANN, the predicted results need to be compared to the ground truth during the training process and the network needs to be penalized every time the predicted output is incorrect. This is achieved through employing a cost function, $c$, that determines the amount by which the network needs to be penalized. This section describes the most common cost functions used. Before various cost functions, also referred to as loss functions, are introduced, it is important to describe the properties a cost function must satisfy. Firstly, a cost function should always produce a non-negative result, as knowing whether the prediction is greater than or less than the target value doesn’t convey much information about the accuracy of the system. It’d be more beneficial to understand how close or far away our predicted output is from the desired output. Secondly, the cost should decrease and approach zero when the predicted output is close to the target output, i.e. ground truth for that particular input sample and increase when the predicted output moves away from the target output.

Mean Absolute Error

The mean absolute error is the mean of sum of absolute difference between target output and predicted output over all the training samples. The absolute difference, highlights the magnitude of difference between the prediction and the ground truth, satisfying both the conditions required to be a valid cost function. Mathematically it can be represented as:
where $c$ is the total cost incurred by the network through penalty, $S$ is the number of training samples and $y^i, t^i$ are the predicted output and target output for $i^{th}$ training sample. Mean Absolute Error is mainly used in ANN models designed for prediction and regression analysis.

**Least Squares**

The least squares method calculates the mean of the sum of the squared difference between target output and predicted output over all the training samples. Using the square of the difference ensures both positive and negative errors are all converted to positive values, and the squared difference exponentially increases the cost when the prediction moves further away from the target and vice versa. The least squares function is given by the equation:

$$
c = \frac{1}{2} \sum_{i=0}^{S} (y^i - t^i)^2
$$

(8)

where $c$ is the total cost incurred by the network through penalty, $S$ is the number of training samples and $y^i, t^i$ are the predicted output and target output for $i^{th}$ training sample. The least squares cost function, like the mean absolute error is for ANN models used in
predictive applications, and is very often favored over mean absolute error, due to its convex shape offering a single minima for linear regression models and thus guarantees a closed-form solution. Although there might be multiple minima for multivariable nonlinear models that use least squares, leading to the problem of the parameters being stuck at one of the local minima, this can usually be avoided by choosing the right values for training parameters like learning rate and momentum, which shall be discussed in further sections.

**Hinge loss**

Hinge loss is a cost function that is used almost exclusively for classification problems. In order to devise a cost function for classification models, it is first essential to understand how the output of a classifier is interpreted. In the context of ANNs it is conventional to assume that the node of the output layer, where each output node represents a particular class in the multi-class model, with the highest output value indicates the class the input sample belongs to. With this understanding, it can be said that whenever a node has a value greater than the actual output, the network needs to be penalized. Also, establishing a safety margin, that dictates the minimum difference between the actual output and the output of all the other nodes, and penalizing whenever this condition is not met, ensures that there wouldn’t be any nodes whose output despite being less than the actual output is too close it to be completely ignored. The loss function can be mathematically written as:

\[
c = \sum_{i \neq j} \max(0, y^i - y^j + \Delta)
\]  

(9)
where $\Delta$ is the safety margin, $y^j$ is the output for the ground truth node, and $y^i$ represents all other nodes other than node $j$. The loss is zero when node’s $i$ output are a distance of $\Delta$ or less than node $j$. The hinge loss function also goes by the name multiclass SVM loss, as this function was first used in multiclass SVM models.

**Cross entropy**

Cross entropy, also referred to as softmax loss, is a cost function used by classification problems. It is an improvement over the hinge loss function in that it generates scores that are more meaningful and easier to interpret by using probability of the input belonging to each class as the output. For this reason softmax is the most commonly encountered loss function for classification problems. Softmax converts the large +/- range class scores generated by a classifier into values between 0 and 1 such that the sum of the values is one. This can be achieved by using the softmax function for all output values. The softmax function is given by:

$$softmax(y^j) = \frac{e^{y^j}}{\sum_{i=1}^{S} e^{y^i}}$$

(10)

Now that the outputs are squashed to values between 0 and 1, the rules for penalizing the network can be established. Whenever the network is not very confident about the input sample belonging to the desired class, i.e. whenever the probability of the desired class is less than 1, the network needs to be penalized. This can be achieved by the following cost function:
where $S$ is the number of classes. As it is evident from the above equation, whenever the probability is equal to one the cost incurred becomes zero and when the probability is less than 1 there is a positive cost incurred. Cross-entropy satisfies both the properties of a cost function and provides more meaningful insight than hinge loss.

### 3.3. Determining the weights

#### 3.3.1 Initializing the weights

The best way to initialize weights in an ANN is an area of active research in itself. Thus this document shall neither comprehensively cover the topic nor compare and contrast one method with another. However, the most commonly used techniques are briefly discussed in this section. When training an ANN from scratch, it is very important to ensure that all weights are not initialized to zero, as that would lead the output of any input to remain unchanged. This problem can be avoided by using random weight initializations, which usually use samples from a zero mean unit standard deviation Gaussian function. One common practice is to use small weights, to ensure that the network isn’t initially highly biased towards any particular subset of inputs as this would slow down the process of convergence. It is also common to normalize variance for the output of each node to ensure all nodes initially have an equal impact on determining the output. This normalization has also been empirically proven to speed up the process of convergence. For a node with $n$
inputs and $m$ outputs, this can be achieved by either scaling the weights with \( \sqrt{\frac{1}{n}}, \sqrt{\frac{2}{n+m}} \) [20] or \( \sqrt{\frac{2}{n}} \) for ANNs that use a ReLU activation function [21]. To avoid delays in convergence due to bad weight vector initializations by using a batch normalization layer before non-linear activations are applied to the output as proposed in [22]. If a pre-existing ANN architecture with an optimal solution that has been trained on a similar dataset exists, then transfer learning should be used. Transfer learning uses a good weight initialization, ultimately enabling faster convergence and higher accuracies.

### 3.3.2 Updating the weights

Once the performance of the ANN model has been determined by using one of the aforementioned cost functions, the next step is to update the weights so as to reduce the penalty incurred from the cost function. The procedures mentioned in the following discussion describe the methods through which the weights can be updated so as to converge to the minimum penalty point quickly.

**Stochastic Gradient Descent**

Derivatives are used to analyze the change in a dependent variable, like cost incurred, with respect to a change in an independent variable, such as weights in an ANN. As a typical network has many weights, the effect of change in each weight on the cost needs to be considered using partial derivatives (gradients). When the gradient is positive, it implies the cost increases with an increase in that weight parameter; hence the weight parameter needs to be decreased proportionally to the gradient. When the gradient is negative, it
implies that the cost decreases with an increase in the weight parameter; hence the weight parameter needs to be increased proportionally to the gradient. This process of descending towards the optimal minimum by constantly taking steps proportional to the negative of the gradient is referred to as gradient descent. Fig 9 illustrates how gradient descent leads the model to the optimal solution.

Parameter update based on gradient descent for a weight $W_{ij}$ in the network can be represented as

$$\Delta W_{ij}(t) = \eta \frac{\partial c}{\partial W_{ij}}$$  \hspace{1cm} (12)$$

$$W_{ij}(t) = W_{ij}(t) - \Delta W_{ij}(t)$$  \hspace{1cm} (13)$$
where $c$ is the cost incurred and $\eta$ is the learning rate that decides the pace at which the weights are updated. If $\eta$ is too high then the gradient descent step might be too large, making the model miss the optimal solution. This can be identified by looking at the cost function plot. If the cost oscillates or increases it is an indication of the learning rate being too high. On the other hand if $\eta$ is too low, it takes a longer time for the model to reach the optimal solution. The reason for large learning rate resulting in divergence and small learning rate resulting in slow convergence is illustrated in Fig 10.

**Fig 10. A pictorial depiction of issues with small and large learning rates.**

### Mini-batch Gradient Descent

As opposed to stochastic gradient descent, where the weights are updated based on the gradient for each input, mini-batch gradient descent evaluates the cumulative loss and gradients of weights over a collection of input samples, often referred to as a mini-batch, and updates the weights once for each mini-batch. The cumulative gradient updates, just
involve adding up of gradients obtained from various inputs. This helps reduce the number of times the weights are updated before convergence. Although it might be counter-intuitive at first, this update method also helps stabilize and improve the accuracy of the predictions in the classification paradigm as the network is provided exemplars from various classes before each update. As the update decision is not solely based on an input that belongs one of the many classes involved, the updated weights aren’t just modified in favor of just one class, or worse one outlier data sample. Thus, very often an increase in the hyperparameter batch size, the number of samples involved in each mini-batch update, is recommended whenever there is a considerable variation in accuracy between consecutive iterations. When the batch size is equal to all the samples in the training set, the gradient update scheme is referred to as batch gradient descent and when the batch size is 1, the gradient descent method becomes stochastic gradient descent. On a side note, when all the training samples are used to update the weights exactly once, it is called an epoch. A typical ANN needs to be trained for multiple epochs before it converges.

**Momentum**

Until this point, the focus has been mostly on analyzing convex functions with one minimum for the entire function. However, it is not uncommon for the cost functions to be have multiple local minima along with the global minimum that provides the optimal solution. Fig 11 depicts this scenario. In such a case, based on the initialization of weights, the network might be stuck at one of the local minima during the learning and can’t converge to the optimum solution. This is due to the fact that gradient descent is a greedy algorithm. The issue of being stuck in a local minima is more probable when the learning
rate is too slow. One way to avoid converging to a local minima is by taking a look at the previous amount by which the weight was previously changed $\Delta W_{ij}(t - 1)$. [23] proposes a solution using this technique by modifying (12) to include the gradient of the parameter during the previous iteration. $\alpha$ is a hyperparameter called momentum.

$$
\Delta W_{ij}(t) = \eta \frac{\partial c}{\partial W_{ij}} + \alpha \Delta W_{ij}(t - 1)
$$

(14)

![Fig 11. A scenario where the solution is stuck at a local minima.](image)

However, when first employed in [23], the aim of adding the momentum term was to speed up the training thus leading to a faster convergence without leading to oscillations, which happens by increasing the learning rate. But the same has been shown to overcome converging to a local minima by choosing an optimal momentum parameter. This can happen in one of two ways. Adding the momentum term might cause the gradient to skip over the local minima thus avoiding it. If not, despite the partial derivative of the cost function with respect to $W_{ij}$ being zero at local minima, the presence of momentum and
historical weight update parameter \( \alpha \Delta W_{ij}(t - 1) \) makes \( \Delta W_{ij}(t) \) non-zero causing the weight to move away from the local minima. This is illustrated in Fig 12.

![Graph showing large momentum helps from being stuck at local minima](image)

**Fig 12.** Momentum helping weights from being stuck at a local minima.

Scenario (a) avoids local minima by moving over it, scenario (b) moves away from local minima despite landing there, due to the momentum.

**Adaptive learning rate**

The discussion so far has treated the weight update hyperparameter learning rate \( \eta \), as a constant. However, as illustrated in Fig 10 picking a small learning rate, while helps us reach the global minima, takes a long time to converge and large learning rates might lead to divergence. However, a larger learning rate could be used initially during the training to reach closer to the minima sooner and then the learning rate could be lowered over time to avoid divergence. Thus having an adaptive learning rate helps us reach the global minimum with a faster convergence time. The adaptive learning rate schemes can be further classified based on whether the learning rate is changed globally or individually for each parameter.
The most common global update schemes include reducing the learning rate by a scaling factor when validation accuracy stops improving or reducing the learning rate with an exponential decay for each iteration. Per-parameter update methods on the other hand, rely on the gradient to determining the learning rates. Weights that receive high gradients have their learning rates lowered more aggressively than weights that receive lower gradients. This is because large gradients indicate large learning rates as shown in Fig 10. RMSprop [24] and Adagrad [25] are two of the most widely used and effective adaptive learning rate schemes in the same order.

3.4. **Avoiding overfitting**

Overfitting is the phenomena in which an ANN learns the intricate input output relations that are specific only to the training data and thus ceases to capture the generic relation between inputs and outputs. In other words, the ANN starts to impose additional restrictions and constrains learned from a subset of data, that are not always satisfied all data members, which should at all costs be avoided. A good example of this would be an ANN that expects a car to have four doors and a roof as it has been trained mostly on images of cars with four doors and a roof. Overfitting can be avoided in more than one way. Three of the most popular techniques are discussed here. The first way is to constrain the weights from having higher values thus preventing any particular subset of inputs to have a high weight in determining the outputs through a technique called regularization. The second technique uses multiple network architectures to be trained on the same data and pick the final output through an averaging or voting scheme, using an ensemble or a dropout scheme. The third method employs artificially generating more training data using
data augmentation techniques. The idea is to show more variations of inputs to the network. Each of these methods are briefly discussed below.

**Regularization**

Regularization is a technique used to prevent networks from overfitting to their training data. While there isn’t much theoretical evidence to demonstrate how regularization prevents overfitting, it has been empirically proven that the networks which employ regularization techniques, perform better over their counterparts on unforeseen data. Regularization ensures that the network can’t learn the noise patterns from the training dataset. Regularization can be incorporated into the training with either L1 or L2 regularization schemes discussed below.

**L2 regularization**

L2 regularization scheme incurs a cost on each of the weights in the network, the further away a weight value is from 0, the greater the cost. In addition, for linear classification models like SVM, where it is usual for many possible solutions to exist, L2 regularization reduces the ambiguity and the number of possible solutions by imposing penalty on higher weights, forcing them to have weights whose magnitude is closer to zero. The penalizing is implementing by including a term, called the regularization term, in the cost functions described by (7) through (11). The regularization term adds the product of sum of all the squares of the weights in the network and a hyperparameter $\lambda$ is used to control the amount of regularization. Using regularization the cross entropy cost function can be rewritten as:
L1 regularization

L1 regularization also reduces overfitting through constraining the weights from becoming too large just like L2 regularization. However, instead of making the penalty proportional to the sum of squares, it makes the penalty proportional to the sum of absolute weights. The L1 is generally less susceptible to large outliers, but does not have an exact gradient. The cost function of a network that uses L1 regularization techniques can be written as:

\[ c = -\frac{1}{S} \sum_{j=1}^{S} \log(\text{softmax}(y_j)) + \lambda \sum_{w} |W|^2 \]  

(15)

Dropout

Dropout [26] is a technique inspired by neural network ensembles, a technique that uses multiple network architectures to be trained on the same data and pick the final output through an averaging or voting scheme as mentioned in [27]. Dropout has gained a great deal of popularity in the past few years in the field of deep neural networks ever since they have been introduced in [2]. The idea is to drop hidden nodes in the network randomly with a probability of ‘p’ in each iteration during the training. A common value of p is typically 0.5. The network learns with the help of the hidden nodes that remain. This procedure creates multiple “sub-networks” during the training phase. Thus the training procedure is similar to creating an ensemble of neural networks, without the overhead of creating
multiple architectures manually and training each of them separately. The lack of one pre-defined architecture for the entire training phase also eliminates the possibility of co-dependence of weight updates during backpropagation leading to unwanted weight convergences as not all weights and biases are updated during each iteration. Dropout also serves as an alternative to regularization by reducing the impact of any particular node on the output. This process is illustrated in Fig 13, where the network on the left side shows how the architecture of a typical ANN on the left changes when nodes are randomly dropped out on the right.

![Diagram](image)

(a) An Artificial Neural Network  (b) An ANN that uses dropout

Fig 13. Illustration of dropout applied to a ANN (a), resulting in (b).

During the testing phase, the weights need to be scaled by a factor of ‘\(p\)’ as all the hidden nodes will be used. This scaling gives the average consensus among the “ensemble of sub-networks” during the testing. Fig 14 shows how each neuron acts during the training and testing phases in a neural network when using dropout.
Fig 14. Neuron's behavior during training and testing time with dropout.

Data Augmentation

Data augmentation is a technique that has been heavily utilized in both computer vision and speech research to artificially increase the size of the training data [2], [9], [15], [28], [29]. This increases the network’s accuracy, as the network gets to see more variations of the inputs. The size of the training data is usually increased by introducing noise into the existing training data. For images and videos, the inputs are randomly cropped, flipped horizontally and vertically and the parameters like brightness, contrast are played with. For speech recognition works, random noise functions like Gaussian noise are introduced to the speech inputs. As more training data avoids overfitting, data augmentation has led to the development of better performing networks.
Chapter 4  Feedforward Neural Networks

A Feedforward Neural Network (FNN) is an ANN where the output of neurons in a given layer can be fed as inputs only to the neurons in the next layer. As forward connections are the only acceptable connections for the neurons, they are feed forward in nature. Very often, a FNN is also referred to as a Multi-layer Perceptron (MLP) due to their similarity to perceptrons. Fig 15 depicts a FNN with one input layer, one output layer and two hidden layers.

![Diagram of a FNN with one input, one output and one hidden layer.](image)

Fig 15. A FNN with one input, one output and one hidden layer.
4.1. **Forward propagation in FNN**

In forward propagation or forward pass, the input patterns are fed to the input layer. The outputs of the input layer are propagated to the next layer and the propagation of the outputs from one layer to the next is continued until the output layer where the output prediction is obtained.

Consider a FNN with the input layer containing $M$ nodes, one hidden layer containing $H$ nodes, an output layer containing $N$ nodes, and the usage of non-linear activation functions after each node. Let $W$ and $W'$ be the input to hidden and hidden to output weight vectors respectively such that $W_{mh}$ and $W'_{hn}$ represent the weights of connections from $m$th input node to $h$th hidden node and $h$th hidden node to $n$th output node respectively such that $0 \leq m \leq M$, $0 \leq h \leq H$ and $0 \leq n \leq N$. Let $x$ and $y$ be the input and output vectors of cardinality $M$ and $N$ respectively and $i_h$, $o_h$, and $a_n$ be the net input and net output of the hidden node $h$ and net input of output node $n$. Then the output at each node of the output layer can be calculated using the following set of equations.

\[
  \begin{align*}
    i_h &= \sum_{m=0}^{M} W_{mh} \ast x_m \\
    o_h &= f(i_h) \\
    a_n &= f'(i_h)
  \end{align*}
\]
\[ a_n = \sum_{h=0}^{H} W'_{hn} \cdot a_h \]  
\[ y_n = f(a_n) \]  

4.2. **Backpropagation in FNN**

Backpropagation is the process through which the penalty incurred from the cost function is distributed to the parameters of the network. It has been stated in the previous sections that the parameter updates are done by using gradient descent. However, the partial derivative of the cost function with respect to all the weights in the network need to be known to update each of the weight parameters. Since its introduction in [31], backpropagation has been widely used in artificial neural networks to update the parameters during training stage. The process is similar to gradient descent, but is more general in that it is applicable to multi-layered FNNs. Gradient descent is a special case of backpropagation for an ANN with a single layer. The cost function, depends on the output of the output layer, the output layer in turn depends on the hidden layer(s), which is in turn fed by the input layer. So, there exists an indirect relation between all the weights and the cost function and the partial derivative of the cost function with respect to the weights can be determined using the chain rule. The partial derivatives of the cost function with respect to some hidden to output and input to hidden weights say \( W'_{hn} \) and \( W_{mh} \) are given by

\[
\frac{\partial c}{\partial W'_{hn}} = \frac{\partial c}{\partial y} \cdot \frac{\partial y}{\partial f} \cdot \frac{\partial f}{\partial a_n} \cdot \frac{\partial a_n}{\partial W'_{hn}}
\]  

(21)
\[
\frac{\partial c}{\partial W_{mh}} = \frac{\partial c}{\partial y} \frac{\partial y}{\partial f} \frac{\partial f}{\partial a_n} \frac{\partial a_n}{\partial o_h} \frac{\partial o_h}{\partial f} \frac{\partial f}{\partial i_h} \frac{\partial i_h}{\partial W_{mh}}
\]  

(22)

As it can be noticed from the above mathematical equations, calculating the gradient descent with respect to individual weights involves backward propagation of error obtained from the gradient of cost with respect to all the intermediate parameters that might have contributions from the respective weights and hence the name back propagation. Once these gradients of the cost function are computed with respect to the weights all that is left is to plug the gradient values in (14) and (13) to calculate the weights for the new iteration.
Chapter 5  Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specific form of FNNs that explicitly assume the inputs to the network be structured samples, such as audio signals or image pixels which can be filtered. These architectures typically focus on solutions for computer vision applications, like classification, localization and segmentation of images and videos. So far it has been assumed that layers in FNN are fully-connected, thus making each input contribute to the output of all hidden layers. If a fully-connected FNN were to be used for an application that uses an input from a VGA camera, whose standard resolution would be 640x480x3, then each hidden neuron shall have 921,600 weights for the connections between the input and first hidden layer alone. An image of this dimension would require the first hidden layer to have thousands of neurons. The model would have a billion weight parameters just for the connections between input and hidden layer. This is unacceptable both in terms of the computational power and memory requirements.

5.1. Convolutional Layers

To prevent the networks from having too many parameters, the fully-connected layers are replaced by convolutional layers in a FNN, leading to CNN models. In convolutional layers (CONV), the hidden neurons are replaced with convolutional filters. Instead of solving for neuron weights, we solve for a family of filters, each filter having its own weights. The convolutional layers arrange the neurons in a 3D fashion using the height, width and depth for the signal being processed. Fig 16 shows a comparison of a fully-connected
conventional FNN and a CNN. Each layer in the depth dimension, aka depth slice, of the CONV layer is analogous to a filtered signal used for digital image processing, where each filtered signal came from a learned filter, whose weights shall be learned during the training process.

![Comparison of FNN and CNN](image.png)

**Fig 16.** Comparison of FNN and CNN. [32]

### 5.2. **Hyperparameters used in CNNs**

Understanding the following hyperparameters is necessary to design and fully comprehend the way CNNs function. As depicted in Fig 16 and stated in the above discussion the convolutional layers in CNN are three dimensional with some height, width and depth. The 3D volume of this convolutional network itself, which is just the number of neurons in the convolutional layer, is computed based on the following hyperparameters.
5.2.1 Receptive Field

The filters in CNNs traverse the entire image using typical convolution. Because the filter size is much smaller than the image, the number of weights we need to solve for is drastically reduced. The spatial extent of the filters is determined by the receptive field size. This process is inspired by the cognitive science behind the receptive fields in the cortex of animals like cats and monkeys, where it has been demonstrated that nearby image regions are mapped to the same or nearby neurons in their brains. This research has been outlined in works [33-35]. Fig 17 depicts how a local spatial field in an input image is mapped to a bunch of neurons. The reason as to why, it is being mapped to a bunch of neurons would be clear after learning about the depth. It is very important to bear in mind that, the receptive field cannot be selective about the depth of its input and has to operate on the entire input depth. The local selection is only allowed along the width and height spatial dimensions of the input.

Fig 17. Illustration of the receptive field concept for a 32x32x3 image. [32]
5.2.2 Depth

The depth of the convolutional layer, determines the number of different neurons that process the same receptive fields which is called the depth column, with a different set of weights. For example, in traditional grayscale image processing, a filter may be of size 5x5. If the image were a color RGB image, the filter would be extended to 5x5x3. The underlying idea is similar to connecting the same input node being processed by multiple hidden nodes in traditional FNN architectures. The objective of having multiple neurons processing the same receptive field is to identify and capture different features for the same input region. Each filter applied to the input image (regardless of the depth), outputs a single output plane. The number of filters, and thus the depth of the convolutional layers are increased as the network moves from input to output as the network switches from capturing simple features to more complex features within images. The depth of the convolutional layer should not be confused with the depth of the CNN which is the number of hidden layers in a CNN. Fig 16 and Fig 17 can also be used to illustrate the depth and depth column respectively.

5.2.3 Stride

While the depth is determined by the number of input planes to a filter, the stride determines the step value across and down the image as the convolution is performed. The filter width, height, depth, and stride are used to construct the 3D convolutional layer. A unit stride implies the need for introducing new depth columns for spatial regions of the image that are a unit distance apart. The stride should be chosen carefully as low stride values lead to a higher number of resolution per each filtered image, with a high overlap in
the receptive fields leading to an increased redundancy in weights. Contrarily, higher stride values yield lower resolution filtered images, at the cost of an increased risk in rapid loss of vital information due to many input parameters contributing to a relatively smaller set of parameters.

5.2.4 Zero padding

Zero padding involves padding zeros of mentioned size in all dimensions on either side of the borders. For a spatial image with 2 dimensions, a zero padding of 1 involves padding a row on top and bottom and a column on the left and right, thus increasing both the height and width of the image by 2. Padding with a value greater than zero is a helpful method to preserve the information on the borders of the image from vanishing through multiple convolutions. It also preserves the spatial dimensions of the output from the convolutional layers, often called the output volume. To understand and appreciate the need for preserving the spatial dimensions of the output, one must be familiar with the pooling layers of CNNs which shall be introduced in the following sections.

5.2.5 Output volume of CONV

The output volume of each CONV layer is the dimensions of the output of convolutional layer, and is calculated using (23), (24) and (25). Let $H_{in}, W_{in}, D_{in}$ and $H_{out}, W_{out}, D_{out}$ be the height, width and depth of input and output of a given convolutional layer. In addition, let it be assumed that the hyperparameters receptive field, depth, stride and zero padding size are given by $H_{rf} \times W_{rf}, K, S$ and $P$ respectively. Then the output volume parameters can be obtained by the following equations.
\[ H_{out} = \frac{H_{in} - H_{rf} + 2 \cdot P}{S} + 1 \]  
\[ W_{out} = \frac{W_{in} - W_{rf} + 2 \cdot P}{S} + 1 \]  
\[ D_{out} = K \]

The stride value \( S \) needs to be picked such that \( H_{out}, W_{out} \) are integral values.

5.3. **Parameter Sharing**

In practice, there are a very few applications that evaluate pixel values at different locations in an image with different filter values. Thus, a parameter sharing scheme would lead to a great improvement in terms of the computational power, training time and memory requirements. Now that there is only one set of weights per filter for all the pixel values, the output of the CONV layer can be computed as a 3D convolution between the input and the filter weights. This is actually the reason for naming this particular FNN architectures as Convolutional Neural Networks.

5.3.1 **Benefits**

Based on what has been discussed so far the number of neurons in the convolutional layer shall be \( H_{out} \cdot W_{out} \cdot D_{out} \) and each of these neurons has \( H_{rf} \cdot W_{rf} \cdot D_{in} + 1 \) weight parameters. Considering the previous VGA input image with dimensions 640x480x3 with
a stride of 5, a receptive field of 5x5, a filter size of 100, and a zero padding size of 0, the output volume becomes 127x95x100 and each of the neuron in the CONV has 5*5*3+1, i.e. 76 weights. Thus the convolutional layer shall have 91,694,000 weight parameters which is very high.

The number of these parameters can be reduced by using the same weights for all spatial co-ordinates within each filter. This leaves only the weight parameter computation for $D_{out}$ filters each having $H_{rf} \times W_{rf} \times D_{in} + 1$ weight parameters. This reduces the parameters of the illustrative model to 7600 from 91,694,000 which is a huge improvement. Fig 18 shows output computation for a convolutional layer with inputs of size 5x5x3, receptive field of 3x3, zero padding size of 1, depth 2 and stride 2.

Fig 18. Output computation for CONV illustrated using a 5x5x3 input. [32]
5.4. **CNN Architecture**

CNNs are made up of four kinds of layers. The main constituent is the convolutional layer, CONV. The focus in this section will shift to the other three layers that constitute the CNNs. They are RELU layers (RELU), Pooling layers (POOL) and Fully Connected layers (FC).

5.4.1 **RELU Layers**

CONV layers are a way of replacing the traditional fully connected layers in FNNs with digital filters. Hence, just like hidden layers in traditional FNNs, CONV layers in CNN need to introduce non-linearities to make enable the network to learn complex non-linear surfaces. Thus, explicitly adding the non-linear activation function RELU as a layer after each CONV layer is necessary. RELU activations have been chosen over other activation functions like logistic sigmoid and hyperbolic tangent as the RELU doesn’t saturate and kill gradients towards the end, is zero centered and doesn’t suffer from the vanishing gradient problem, which will be discussed in RNN discussion.

5.4.2 **Pooling Layers**

As shown in (23), (24) and (25) computing the output volume for the CONV layer requires a careful choice of architectural specifications such that the parameters of the output volume always yield integral outputs. Also, it is important to consider the fact that the aforementioned equations are used recursively over multiple CONV layers where the output of the first CONV layer becomes the input to the second and so on until the end. Instead of going through the painstaking process of solving these equations, it is much simpler to fix the stride to 1 and the receptive field to some constant for all the
convolutional layers and adjust the padding size such that the input and output always have the same spatial dimensions. [36] uses this technique to simplify the design process as opposed to [2] which does it the complicated way. However, now that more researchers are preferring the simpler approach; it is essential to have a mechanism through which the spatial features can be downsized when moving away from the input layer towards the output layer thus effectively moving away from more number of simpler feature to less number of complex features. This can be achieved by using pooling methods. The pooling layer reduces the spatial dimensions of the output volume and keeps the number of weight parameters in check. The pooling operation, works on each depth slice of the input and downsamples it. The pooling operation uses two parameters receptive field and stride.

Let $H_{ip}, W_{ip}, D_{ip}$ and $H_{op}, W_{op}, D_{op}$ be the height, width and depth of input and output of a given pooling layer. In addition, let it be assumed that the receptive field and stride are $H_{rf} \times W_{rf}$ and $S$ respectively. Then the output parameters of the POOL layer can be obtained by the following equations.

$$H_{op} = \frac{H_{ip} - H_{rf}}{S} + 1 \quad (26)$$

$$W_{op} = \frac{W_{ip} - W_{rf}}{S} + 1 \quad (27)$$

$$D_{op} = D_{ip} \quad (28)$$
Large receptive fields are generally not used as that would throw away a lot of data. The reduction in the number of parameters shouldn’t be at the cost of reduced accuracies of the CNNs. Some of the most common pooling techniques are discussed below.

**Max Pooling**

The max pooling technique replaces all the elements of the receptive field in the input with the maximum element in the receptive field for the output. Then it moves with the specified stride to the next receptive field in the input. The most common values are 3x3 receptive fields with a stride of 2 and 2x2 receptive fields with a stride of 2. The former is referred to as overlapping max pooling, while the latter goes by non-overlapping max pooling. The latter is the most commonly employed pooling technique. Fig 19(a) provides visualization for downsampling through pooling along with Fig 19(b), which illustrates non-overlapping max pooling with an example.

![Fig 19](image-url)

**Fig 19.** Downsampling the output size through pooling. [32]

(a) Visualization of downsampling of an image using non-overlapping max pooling. (b) Illustrative example of non-overlapping max pooling.
**Average Pooling**

The average pooling method replaces the receptive field with a single element whose value is equal to the mean of all the elements in the receptive field. This method has been used historically but is no longer favored as it has been empirically demonstrated that max pooling outperforms average pooling. This is most likely due to the fact that max pooling retains the most prominent information while averaging blurs out details during downsampling.

**L2 Pooling**

The L2 pooling method computes the L2 norm of all the elements in the receptive field and replaces the receptive field with this value. The L2 norm is just the square root of the sum of squares of all elements in the receptive field.

**5.4.3 Fully Connected layers**

As previously talked about, fully connected (FC) layers are hidden layers where all the input nodes connect and contribute to all the output nodes. A fully connected layer can thus be represented as a special case of a convolutional layer where the receptive field of the filters is equal to the spatial dimensions of the input, with a padding size of zero and no stride, thus producing an output volume of 1x1xK, where K is the number total number of neurons in the FC layer. This relation between the two helps in implementing both FC and CONV layers the same way for CNNs.
Now that all the layers involved in a CNN architecture have been discussed, it is time to evaluate the architecture of a typical CNN. A typical CNN architecture is shown in Fig 20.

![CNN architecture for a typical image classification problem.](image)

Typically, the POOL layer is not used after each CONV and RELU layers. This is because using multiple convolutions with smaller receptive field are usually preferred over one CONV layer with a larger receptive field. [36] has theoretically proven that using multiple CONV layers with smaller receptive field has the same effect as using one convolutional filter with large receptive field, with the added benefit of having a lower number of parameters overall. To demonstrate this they have replaced a 7x7 convolutional filter with a 3x3 convolutional filter and used the 3x3 filter thrice. Performing a 3x3 convolution thrice would cover the same area as a 7x7 filter would. However a 7x7 filter would have 49 parameters and all the three 3x3 filters combined would have 27 parameters.
Thus, smaller filters perform the same job with much fewer parameters. Furthermore using more number of CONV layers with smaller filters to do the same job, will increase the depth of the CNN architecture, and will increase the non-linearity introduced in the data leading to better classification results. Despite all these advantages, a CONV layer with large receptive field can be used in the first layer, if the spatial co-ordinates of input to the CNN is very high and needs to be reduced in the output volume.
Chapter 6  Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are ANNs wherein the neurons are allowed to form cyclical connections with themselves and are allowed to connect with other neurons within the same layer. A baseline RNN is depicted in Fig 21. Two specific RNN architectures include the Simple Recursive Networks (SRNs) and Long Short Term Memories (LSTMs), each of which is described and analyzed in the sections that follow.

Fig 21. RNN displaying the characteristic cyclical connections.
The cyclical connections are useful in capturing the sequential prediction of outputs, where the current output is not only dependent on the current input but also the previous outputs. Such Markov models can be captured much more elegantly in RNNs than in conventional FNNs.

6.1. **Simple Recurrent Networks**

A Simple Recurrent Network is a basic RNN with both cyclical and in layer connections. The architecture of a SRN can be depicted as shown in the Fig 22a and Fig 22b. Both these figures represent the same architecture. While the former depicts the conventional representation with the recursive connection, the latter gives an insight into the working of an RNN by depicting what happens during each time step and how the previous output of the hidden layer impacts the output of the current hidden output, along with the current input. As the output is depend on the previous hidden state(s), the output of the previous time step is impacting the current output.
6.1.1 Forward propagation in SRN

Let $f$ be the activation function, $x_t$, $h_t$, $h_{t-1}$ and $y_t$ be current input, hidden, previous hidden and current output values, $W_{xh}$, $W_{hh}$ and $W_{hy}$ be the weight matrices for input, hidden and output stages respectively and $in_{y0}$ and $in_{h0}$ be the inputs to activation functions in output and hidden layers. Then the output in the forward pass in a SRN can be given by the following equations.

\[ in_{h0} = (W_{xh}x_t + W_{hh}h_{t-1}) \]  \hspace{1cm} (29)

\[ h_t = f(in_{h0}) \]  \hspace{1cm} (30)

\[ in_{y0} = W_{hy}h_t \]  \hspace{1cm} (31)
\[ y_t = f(\text{in}_{y0}) \]  

If the SRN started at time step \( t \) and continued until time step \( t+3 \), with inputs \( x_t, x_{t+1}, x_{t+2}, x_{t+3} \) outputs \( y_t, y_{t+1}, y_{t+2}, y_{t+3} \) with intermediate hidden node outputs \( h_t, h_{t+1}, h_{t+2}, h_{t+3} \) the forward propagation can be illustrated as shown in Fig 23. and each of these values can be computed from (29) to (32). For the first time step, the previous hidden output is assumed to be zero. The same weights are reused for each iteration.

![Fig 23. Illustration of forward propagation in SRN.](image)

### 6.1.2 Backpropagation in SRN

In addition to following the rules mentioned for backpropagation in FNNs (3.2) the backpropagation also involves a time component, which is often referred to as
backpropagation through time (BPTT). The idea is that, unlike a hidden node in a FNN, a hidden node in a RNN at any time step is directly responsible for the cost incurred at the given time step and the next time step and also indirectly responsible for every error that occurs thereafter. Hence, based on the concept of BPTT where each hidden node is directly responsible for the cost incurred at the current and next time step and equations (21), (22), (29) and (30) the following BPTT equations can be written for Fig 23.

\[
\frac{\partial c}{\partial h_{t+3}} = \frac{\partial c_3}{\partial h_{t+3}} = W_{hy} \cdot df(in_{y3}) \cdot \frac{\partial c_3}{\partial y_{t+3}}
\]

(33)

\[
\frac{\partial c_2}{\partial h_{t+2}} = W_{hy} \cdot df(in_{y2}) \cdot \frac{\partial c_2}{\partial y_{t+2}}
\]

(34)

\[
\frac{\partial c_1}{\partial h_{t+1}} = W_{hy} \cdot df(in_{y3}) \cdot \frac{\partial c_3}{\partial y_{t+3}}
\]

(35)

\[
\frac{\partial c_0}{\partial h_{t}} = W_{hy} \cdot df(in_{y3}) \cdot \frac{\partial c_3}{\partial y_{t+3}}
\]

(36)

\[
\frac{\partial c}{\partial h_{t+2}} = \frac{\partial c_2}{\partial h_{t+2}} + \frac{\partial c_3}{\partial h_{t+2}} = \frac{\partial c_2}{\partial h_{t+2}} + W_{hh} \cdot df(in_{h3}) \cdot \frac{\partial c}{\partial h_{t+3}}
\]

(37)

\[
\frac{\partial c}{\partial h_{t+1}} = \frac{\partial c_1}{\partial h_{t+1}} + \frac{\partial c_2}{\partial h_{t+1}} = \frac{\partial c_1}{\partial h_{t+1}} + W_{hh} \cdot df(in_{h2}) \cdot \frac{\partial c}{\partial h_{t+2}}
\]

(38)

\[
\frac{\partial c}{\partial h_{t}} = \frac{\partial c_0}{\partial h_{t}} + \frac{\partial c_1}{\partial h_{t}} = \frac{\partial c_2}{\partial h_{t}} + W_{hh} \cdot df(in_{h1}) \cdot \frac{\partial c}{\partial h_{t+1}}
\]

(39)
\[
\frac{\partial c}{\partial W_{hh}} = h_t \cdot df(in_{h1}) \cdot \left( \frac{\partial c_1}{\partial h_{t+1}} + \frac{\partial c_2}{\partial h_{t+1}} \right) \tag{40}
\]
\[
\frac{\partial c}{\partial W_{xh}} = x_t \cdot df(in_{h0}) \cdot \left( \frac{\partial c_0}{\partial h_{t+1}} + \frac{\partial c_1}{\partial h_{t+1}} \right) \tag{41}
\]

where \( df(x) \) is the derivative of \( f \), the activation function, evaluated at the point \( x \), and \( c_0, c_1, c_2 \) and \( c_3 \) are the costs incurred from time instants \( t \) to \( t+3 \).

### 6.1.3 Vanishing and exploding gradients

The vanishing gradient problem occurs due to the fact that the gradients are back propagated through time in SRNs, making the derivative of the activation function term \( df \) multiply rapidly over time. This can be observed if equations (38) to (33) are recursively substituted in (39). Although the example uses a time step of four for illustrative purposes, the number of time steps in RNNs are typically much higher than four. Unfortunately, if the derivative of the activation function is less than one, the gradients vanish rapidly over time and if the derivative is greater than one, the gradients explode. The risk of vanishing gradient increases with time- the longer the time lag between the dependencies of output on previous inputs, the sooner the gradients shall vanish making it difficult for the model to capture dependencies over time, the very reason why RNNs were introduced. Because weights are typically initialized with a mean value of zero and standard deviation of 0.001, and as the derivatives of common activation functions like logistic sigmoid, hyperbolic
tangent also tend to have a value less than or equal to one, exploding gradients are seldom an issue. However, the vanishing gradient problem is a major issue. RELU and piecewise linear activations always have their derivatives as either one or zero and don’t suffer this vanishing gradient problem. Thus employing these activation functions mitigate the issue. This vanishing gradient problem is also observed in deep FNNs which is one reason CNNs perform better with RELU activations. Large exploding gradient values are prevented by using gradient clipping in practice when they exceed a certain threshold.

6.1.4 Inability to capture long term dependencies

SRNs have the inability to capture long-term dependencies due to the vanishing gradient problem explained in the previous section and also due to the fact that newer input values tend to overwrite the previous hidden states, thus reducing the overall effect of the previous hidden states. This is illustrated in Fig 24. The output of the video classifier gives more weight to current input frame by changing its prediction whenever the current input frame significantly differs from the previous input frames, thus discarding the impact of previous output predictions.
Fig 24. Illustration of inability to capture long-term dependency using a SRN based activity classifier that uses video input.

LSTMs avoid this problem by using input gates that control the impact of previous hidden state and current input state on current hidden state and using output gates to regulate the impact of current cell state on current output as shown in Fig 25. The input gates are shown in blue and prevent current input from overwriting the impact of previous hidden states, the output gates, shown in red, regulate the impact of cell state on output by waiting until the end of video frames to make a prediction of the activity.
Fig 25. Preserving long term dependencies using input and output gates.

6.2. **Long Short Term Memories**

Long Short Term Memories (LSTMs) are an alternative to SRN and often do a better job in capturing dependencies over time due to their many gates. LSTMs have outperformed RNNs in various natural language processing applications such as handwriting recognition [37], language translation [38], and image and video annotation [7-12]. A typical LSTM cell has three gates: input, forget and output. A simple LSTM cell architecture is depicted and is compared against a standard RNN (SRN) cell in Fig 26. These LSTM cells make up the hidden layers in an LSTM based architecture instead of using the standard neurons. The responsibilities of multiple gates within the LSTM units shall be explained in the discussion that follows.
The input gate provides LSTM the ability to preserve or overwrite the input from the previous hidden layer and the current input node by controlling the impact of current input and previous time-step output on the current cell state. Although LSTMs can satisfy time dependencies for more than 1000 time steps, sometimes it is essential for the network to discard historical data to avoid unwanted dependencies being introduced during learning. The forget gate provides an ability to help reduce or even discard the impact of previous cell state on current cell state. The output gate regulates the output flow from the current cell state to the current hidden state output.
6.2.1 Forward propagation in LSTM

Given a temporal input sequence \(<x_1, x_2, ..., x_{t-1}, x_t, ..., x_T>\), where \(x_{t-1}\) and \(x_t\) are consecutive inputs to the system, the LSTM updates for each time step \(x_t\), as described in [9, 39] are:

\[
i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad (42)
\]

\[
f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad (43)
\]

\[
o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (44)
\]

\[
g_t = \phi(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \quad (45)
\]

\[
c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (46)
\]

\[
h_t = o_t \odot \phi(c_t) \quad (47)
\]

where \(i_t\) is the output of the input gate; \(f_t\) is the output of forget gate; \(o_t\) is the output of output gate; \(g_t\) is the input node or input modulation gate that modulates the input to the LSTM cell; \(c_t\) is the memory cell that is a combination of the previous memory cell governed by the forget gate and the input node governed by the input gate; and \(h_t\) is the current hidden state - the activation of the current cell state governed by the output
gate. All the gates use the sigmoid ($\sigma$) activation function to compute the outputs $i_t$, $f_t$, $o_t$, and can be thought of governors, modulating how much of a signal can pass. $\phi$ is the tanh activation function, typically used in LSTMs. The weights, $W$ and bias values, $b$ are learned during backpropagation through time. Similar to the way an ANN can have many hidden layers, each with many hidden nodes, a LSTM network also can have multiple LSTM layers, where each LSTM layer typically has many hundreds (or thousands) of cells (nodes).

### 6.2.2 Avoiding vanishing gradient problem

The vanishing gradient problem in LSTMs is avoid through maintaining a Constant Error Carousel [5]. Based on equations (46) if the backpropagated gradient at cell state at time instant $t$ is $\frac{\partial c_t}{\partial t}$, then the backpropagated gradient at time instant $t-1$ is given by

$$
\frac{\partial c_{t-1}}{\partial (t - 1)} = f(t)\bigodot \frac{\partial c_t}{\partial t}
$$

(48)
Fig 27. Illustration of BPTT for a LSTM cell.

So as long as the forget gate output is close to 1, the gradient will not vanish. When the forget gate is zero, there is no dependency between previous time steps and current time steps. Hence it is okay for gradient to vanish. The computation of (48) is explained in Fig 27.
Chapter 7  Activation and vocabulary based experiments

As a part of this thesis work, experiments have been conducted to analyze the effects of various activation functions and vocabulary on the performance of RNN based architectures. To do so, the python based deep learning framework NeuralTalk, which has implementations of architectures proposed in [13] and [14], has been employed and modified to suit the experimental needs. The aim of these experiments is to analyze the impact of activation functions on the vanishing gradient problem in SRNs and performance; and to determine the relation between the vocabulary size used for training and performance by modifying the minimum word frequency threshold at which a word can be included in the training vocabulary. The Flickr8k dataset [40] has been used for training, validation and testing. A sample of images and corresponding captions in the dataset has been shown in Fig 28.
The activation functions used in SRN architectures are RELU, hyperbolic tangent (tanh) and sigmoid (sig). Table 1 shows the performance of each of these activation functions when used in a CNN/SRN based image caption generator as proposed by Karpathy et al. in [13].

<table>
<thead>
<tr>
<th>Activation</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELU</td>
<td>56.7</td>
<td>38.0</td>
<td>23.8</td>
<td>14.6</td>
</tr>
<tr>
<td>tanh</td>
<td>13.5</td>
<td>0.4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sigmoid</td>
<td>13.1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Evaluation of the performance for various activation functions in SRN based architecture for Flickr8k.

As previously seen in the vanishing gradient discussion, RELU performs well due to its ability to overcome this problem, while tanh and sigmoid can’t seem to learn anything due to the vanishing gradient problem even for a time dependency of the order of ten time
steps. This is illustrated through the captions generated for test data of Flickr8k by each model in Fig 29. From the figure it is clear that tanh and sigmoid haven’t learned any probability predictions based on the context around the words due to vanishing gradient problem and just generate random length image captions using the most commonly occurring word in vocabulary ‘a’. Also, RELU seems to converge much sooner in fewer epochs and the training time per epoch is also lower.

Now, that RELU has been established to be the best activation function for SRNs the next logical step is to experiment with the efficacy of RELU in LSTM based architectures. While LSTM handles vanishing gradient problem through constant error backpropogation, RELUs are new to the fields of Natural Language Processing and Computer Vision. Architectures that employ LSTMs haven’t done much experimentation with RELUs. The next experiment in this thesis research work thus aims to compare RELUs with hyperbolic tangents as activation functions in LSTMs. The non-linearity in the gates of LSTMs wasn’t changed from sigmoid to RELU as the gates are supposed to act as continuous analog switches, with their outputs ranging between 0 and 1. RELU can’t offer smooth non-linearity in this region. Table 2 compares the performance of RELU with hyperbolic tangent in LSTMs.
Fig 29. SRN predictions for RELU, tanh and sigmoid activations on Flickr8k.

(a) RELU based SRN predictions

(b) tanh and sigmoid based SRN predictions

<table>
<thead>
<tr>
<th>Activation</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>tanh</td>
<td>56.9</td>
<td>38.5</td>
<td>25.1</td>
<td>16.5</td>
</tr>
<tr>
<td>RELU</td>
<td>55.7</td>
<td>36.8</td>
<td>23.0</td>
<td>14.5</td>
</tr>
</tbody>
</table>

Table 2. Comparison of performance between tanh and RELU in LSTM based architecture for Flickr8k.

From Table 2, it can be seen that the performance of RELU while being comparable to tanh in LSTM is slightly less, even in comparison with the results of RELU in SRNs. The plausible reason for this degradation might be due to the fact that the RELU outputs
much higher positive values than tanh and doesn’t output any negative values. Thus, it is
biasing the gates to one side making it harder for the gates to regulate impact of input,
previous hidden state or current cell state on the output. Hence, having an activation
function that has similar shape as RELU, but has the same range as hyperbolic tangent
might improve the performance of the system. A piecewise linear function that has the
same slope as RELU and same range as tanh has been chosen for experimentation to see if
that helps boost the performance. Table 3 compares the performance of tanh and piecewise
linear (PL) activation functions.

<table>
<thead>
<tr>
<th>Activation</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>tanh</td>
<td>56.9</td>
<td>38.5</td>
<td>25.1</td>
<td>16.5</td>
</tr>
<tr>
<td>PL</td>
<td>56.3</td>
<td>37.6</td>
<td>23.7</td>
<td>15.1</td>
</tr>
</tbody>
</table>

Table 3. Comparison of performance between tanh and PL in LSTM based architecture for Flickr8k.

While the results of PL are certainly better than the results of RELU based LSTM,
they still don’t beat the performance of tanh based activation functions. Also PL based
LSTMs seem to have more compute time than tanh based LSTMs. Because of these
reasons, they aren’t recommended for usage.

This thesis research also tries to identify the relation between limiting the
vocabulary based on minimum word frequency and performance of the system. Previous
research eliminates words that occur less than five times in the entire training data and
replace those words with a special unknown keyword. Intuitively, increasing this threshold
to higher values seem detrimental to the performance. However it is unclear if increasing
the vocabulary size by reducing this threshold helps generate better image captions. After
all, increasing vocabulary size increases the parameters involved and words that are not
encountered very frequently. This could just confuse the system and make the performance
worse. In order to answer this question, an experiment has been conducted varying the
word count threshold from one to nine with increments of two. The results are presented
in Table 4. Clearly having too many less frequently used words seems to confuse the system
reducing the performance as there are not enough exemplars to train from. Rigorous
elimination of words beyond a point also seems to be detrimental.

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Word count</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7374</td>
<td>54.2</td>
<td>35.8</td>
<td>22.8</td>
<td>14.8</td>
</tr>
<tr>
<td>3</td>
<td>3476</td>
<td>55.5</td>
<td>37.2</td>
<td>23.9</td>
<td>15.7</td>
</tr>
<tr>
<td>5</td>
<td>2537</td>
<td>56.9</td>
<td>38.5</td>
<td>25.1</td>
<td>16.5</td>
</tr>
<tr>
<td>7</td>
<td>2072</td>
<td>54.1</td>
<td>35.1</td>
<td>21.8</td>
<td>14.1</td>
</tr>
<tr>
<td>9</td>
<td>1775</td>
<td>52.5</td>
<td>34.5</td>
<td>21.5</td>
<td>13.9</td>
</tr>
</tbody>
</table>

Table 4. Relation between word frequency threshold based vocabulary elimination and performance
in LSTM based architecture for Flickr8k.
Chapter 8  FRMM architectures

The proposed Fusion-based Recurrent Multi-modal architecture (FRMM) learns each input modality in separate stages, thus allowing each modality to be learned by an architecture that works best at capturing the features of that particular modality. The outputs from these separate stages are then mapped to a shared representation in preparation for association with one another in the fusion stage. The fusion stage learns to predict the output based on these associations, and all stages are learned end-to-end in a supervised fashion. Fig. 1 uses an illustrative example of a video description model to give the reader a high level understanding of how the FRMM model works. The number of time steps in the fusion stage is always equal to the length of the output sequence. This can be observed in both Fig 1 and Fig 30.

The separate learning stages provide the FRMM architecture increased flexibility over previous approaches [9, 10, 14, 41-43]. The main limitation that these approaches have is the inability to have different architectures for disparate inputs as they share parameters across modalities. These approaches also impose constraints on the length of the input sequences or size of inputs by requiring them to be equal across modalities. These
shortcomings of previous researches are addressed through our FRMM model. The length of the input sequences should not be confused with the size of input. To illustrate with Fig 1, the length of input sequences in video and language modalities is 3 and 5 respectively while the input size in each of these modalities is the dimensions of the input frame and the input word vector respectively. FRMM architecture does not require the input sizes of different modalities to be equal.

Fig 30. Illustrations of FRMM architectures with image captioning examples.
In a FRMM model, the architecture for each individual stage might vary greatly based on the modalities involved, but can be learned using neural network architectures like FNNs, CNNs, and RNNs. This ensures that the overall architecture highly adaptable. FRMMs are designed to work with multi-modal applications in which at least one of the input modalities has a temporal or sequential aspect to them. Some examples of such multi-modal applications include image description, sentiment analysis and language translation.

8.1. **FRMM architectures**

Based on the length of the sequences in each modality, FRMMs can be broadly categorized into aligned and unaligned FRMMs. Aligned FRMMs (AFRMM) have input sequences of same length across various input modalities either naturally or by forcing them to be equal through padding techniques [42]. Unaligned FRMMs (UFRMM), allow input sequences of different modalities to have different lengths, without any padding.

8.1.1 **Image Captioning through FRMMs**

For image description, we have two independent input stages for each modality namely the *image stage* and *language stage*, and a shared *fusion stage* that learns the association between these two modalities. The image stage uses either a fully-connected layer or an LSTM layer on top of a CNN, while the language stage and fusion stage use a LSTM network. During the training phase, the image stage learns to extract important image
features and generates an image feature descriptor while the language stage learns the sentence structure, semantics of the language and word association probabilities and generates word feature descriptors. The image features learned in the image stage and language features learned in the language stage are then mapped into a common vector space in the fusion stage which learns to associate image features with appropriate words from the ground truth descriptions. Learning is facilitated by an end-to-end training where the error is backpropagated to all stages. During the testing phase, the test image and a keyword START are fed to the image and language stages respectively. Based on the image features the FRMM model makes a word prediction, this predicted word and the image feature descriptor are then fed to the FRMM in the next time step to generate the next prediction. This process is continued until the end of the sentence is encountered.

The AFRMM architecture for image description consists of an image stage that has an LSTM layer on top of CNN and the language stage that has an LSTM layer. As the input sequences from both modalities to the fusion stage need to be of the same length, an image feature vector is generated every time the output is not the end of sentence by repeatedly feeding the same input to the image stage. Although the image stage receives the same input at each time step, the LSTM layer in it facilitates producing a different image feature descriptor at each time step. Thus in AFRMM image feature descriptors are tied to individual words due to the one-to-one correspondence of the outputs from image and language stages. The fusion stage makes predictions based on this correspondence. This process is illustrated in Fig 30(b).
A more intuitive model for image description is the UFRMM model as the image stage and language stage are allowed to have input sequences of different lengths. Such a UFRMM architecture gives a lot more flexibility in terms of the architecture of the image stage. The image stage can have either a fully-connected layer (UFRMM+FC) or a LSTM layer (UFRMM+LSTM) on top of a CNN. The UFRMM+FC and UFRMM+LSTM architectures broadcast the same image feature descriptor to the fusion stage along with each word feature descriptor from the language stage. Thus in UFRMM architectures the image feature is tied to an entire sentence description and the model learns to associate the image features with a set of words. The UFRMM+LSTM architecture needs to have at least two time steps for the temporal functionality of the LSTMs to kick-in. The UFRMM+LSTM architecture can be further classified based on how the aggregate image feature vector is obtained from the LSTM layer. The image feature descriptor could either concatenate outputs from both time steps (UFRMM+C) or just use the output from the last time step (UFRMM+L). All the three possible UFRMM models, namely UFRMM+FC, UFRMM+L and UFRMM+C are shown in Fig 30(a), Fig 30(c) and Fig 30(d) respectively.

8.2. *FRMM Results*

8.2.1 Datasets

The proposed architectures have been evaluated on two widely used datasets, namely Flickr30K [44] and MSCOCO[45]. The datasets contain 29,000 and 80,000 training images respectively, with five sentences describing each image.
8.2.2 Implementation details

*CAFFE* [46], a popular deep learning framework, has been employed for implementing the proposed FRMM architectures. During the training phase, words that occur less than five times in the entire corpus are filtered out in order to include only those words in the vocabulary that have enough exemplars for training and are replaced by the keyword *unknown*. This generated a vocabulary size of 7,414 and 8,800 words for Flickr30K and MSCOCO datasets, respectively. The CNN architecture employed follows a minor variation of the 16-layer VGGNet [36], that has been pre-trained on the ImageNet dataset [1]. To compute the similarity of the automatically generated image description with the ground truth, BLEU [47] and METEOR [48] scores have been used. BLEU compares $n$-gram (where $n$ is of order 1, 2, 3, and 4) matches where each $n$-gram is a phase of length $n$ present in the sentence. The BLEU scores are calculated by calculation the number of exact $n$-gram matches between the predicted sentence and the ground truth; more matches correlate with higher similarity. METEOR compares not only exact $n$-gram matches but also stemmed tokens, paraphrased matches and semantically similar matches.

8.2.3 Experimental Results

LSTMs intuitively make sense for variable sequences such as frames from a video or words in a sentence. However, their efficacy is yet to be established for a static input like an image. Our results show that some LSTM architectures improve performances with still images. These results are shown in Table 5.
<table>
<thead>
<tr>
<th>Model</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFRMM</td>
<td>70.2</td>
<td>52.8</td>
<td>38.3</td>
<td>27.6</td>
</tr>
<tr>
<td>UFRMM+FC</td>
<td>67.0</td>
<td>48.9</td>
<td>34.6</td>
<td>24.5</td>
</tr>
<tr>
<td>UFRMM+L</td>
<td>66.7</td>
<td>48.5</td>
<td>34.2</td>
<td>24.2</td>
</tr>
<tr>
<td>UFRMM+C</td>
<td>67.0</td>
<td>48.8</td>
<td>34.6</td>
<td>24.5</td>
</tr>
</tbody>
</table>

Table 5. Evaluation of the performance of image description model with various FRMM architectures on the MSCOCO dataset.

Table 5 contrasts various FRMM architectures that have been described in section 3.3. AFRMM is the preferred architecture for still image datasets like MSCOCO. The main advantage AFRMM has over UFRMM architectures is its ability to associate image features with individual words as opposed to tying the image feature descriptor to an entire sentence. This seems to allow the AFRMM architecture to recursively filter out image features that are tied to the current word prediction.

Optimization experiments have also been conducted, in order to determine the best feature extraction layer from the CNN. Image feature descriptors were extracted from fully connected layers fc6, fc7 and fc8 of VGGNet and fed to the LSTM layer of image stage in the AFRMM model. The results are shown in Table 6. It has been observed that image features from fc8 are better at summarizing the image features than other fully-connected layers of VGGNet.
<table>
<thead>
<tr>
<th>CNN layer</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFRMM+fc6</td>
<td>66.7</td>
<td>48.6</td>
<td>34.4</td>
<td>24.3</td>
</tr>
<tr>
<td>AFRMM+fc7</td>
<td>68.4</td>
<td>50.3</td>
<td>35.7</td>
<td>25.2</td>
</tr>
<tr>
<td>AFRMM+fc8</td>
<td>70.2</td>
<td>52.8</td>
<td>38.3</td>
<td>27.6</td>
</tr>
</tbody>
</table>

Table 6. Evaluation of the performance of image description model when image feature descriptor is extracted from layers fc6, fc7 and fc8 of the CNN on the MSCOCO dataset.

To verify if increasing the number of LSTM layers can improve the performance of our AFRMM model experiments, have been performed by using a LSTM network with two LSTM blocks stacked on top of one another on both the language stage and fusion stage. It has been observed that adding more layers is not beneficial and probably detrimental for this application. These results are presented in Table 7. More LSTM layers have not been added for image stage as it didn’t make sense to have more than one. The results are in agreement with Donahue et al. [9] findings. Adding more layers for learning words and images together (2 layer unfactored, 4 layer factored) reduced the BLEU scores of their LRCN models. Hence, it has been concluded that multi-layered LSTM architectures are not needed for each individual stage.

<table>
<thead>
<tr>
<th>2 layer stage</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>66.8</td>
<td>48.8</td>
<td>34.6</td>
<td>24.6</td>
</tr>
<tr>
<td>Fusion</td>
<td>67.1</td>
<td>49.0</td>
<td>34.8</td>
<td>24.6</td>
</tr>
</tbody>
</table>

Table 7. Performance of image description model with two-layered LSTM architectures in each stage.
Table 8 and Table 9 compare the performance of the proposed AFRMM model, with similar research works on the MSCOCO and Flickr30K datasets. AFRMM model is the best performer on MSCOCO dataset. However, it didn’t perform as well on Flickr30K dataset. This is due to the fact that our AFRMM model needs to learn more parameters than other models and Flickr30k doesn’t have enough training exemplars to facilitate this learning process. Our high B-4 and METEOR results, which account for the fluency of language in image descriptions, can be attributed to our dedicated language stage that learns the grammatical rules and semantics of the language.

<table>
<thead>
<tr>
<th>Model</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karpathy[13]</td>
<td>62.5</td>
<td>45.0</td>
<td>32.1</td>
<td>23.0</td>
<td>19.5</td>
</tr>
<tr>
<td>Donahue [9]</td>
<td>62.7</td>
<td>44.1</td>
<td>40.4</td>
<td>21.0</td>
<td>-</td>
</tr>
<tr>
<td>Vinyals [14]</td>
<td>66.6</td>
<td>46.1</td>
<td>32.9</td>
<td>24.6</td>
<td>-</td>
</tr>
<tr>
<td>Fang [49]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>21.1</td>
<td>20.7</td>
</tr>
<tr>
<td><strong>AFRMM</strong></td>
<td>70.2</td>
<td>52.8</td>
<td>38.3</td>
<td>27.6</td>
<td>22.5</td>
</tr>
</tbody>
</table>

Table 8. Comparison of our approach with other proposed methods using BLEU and METEOR scores for MSCOCO dataset

<table>
<thead>
<tr>
<th>Model</th>
<th>B-1</th>
<th>B-2</th>
<th>B-3</th>
<th>B-4</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karpathy[13]</td>
<td>57.3</td>
<td>36.9</td>
<td>24.0</td>
<td>15.7</td>
<td>-</td>
</tr>
<tr>
<td>Donahue[9]</td>
<td>58.7</td>
<td>39.0</td>
<td>25.1</td>
<td>16.4</td>
<td>-</td>
</tr>
<tr>
<td>Vinyals [14]</td>
<td>66.3</td>
<td>42.3</td>
<td>27.7</td>
<td>18.3</td>
<td>-</td>
</tr>
<tr>
<td>Mao [50]</td>
<td>55</td>
<td>24</td>
<td>20</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>AFRMM</strong></td>
<td>58.9</td>
<td>40.0</td>
<td>26.6</td>
<td>17.7</td>
<td>17.8</td>
</tr>
</tbody>
</table>

Table 9. Comparison of our approach with other proposed methods using BLEU and METEOR scores for Flickr30K dataset.
8.2.4 Sample Captions

This section provides the image descriptions generated for some of the test images using our AFRMM architecture. As it can be seen from Fig 31 and Fig 32 some of them are very accurate while the rest are not so accurate.

![Image descriptions](image1.png)

A man riding a wave on top of a surfboard. A vase filled with red and yellow flowers. A young boy holding a baseball bat in his hands.

![Image descriptions](image2.png)

A group of people riding motorcycles down a street. A group of people sitting around a table with wine glasses.

![Image descriptions](image3.png)

A black cat sitting on a window sill looking out a window. A group of cows grazing in a field.

Fig 31. Some of the accurate image descriptions generated by AFRMM.
Fig 32. FRMM generated captions that are not very accurate.

A giraffe is standing in a zoo enclosure.
A man and woman walking down a street with an umbrella.
A bathroom with a sink, toilet, mirror and a shower.
A display of oranges and apples in a market. "A pizza with cheese, cheese and cheese on a plate."
A dog with a hat on its head.
A woman standing in a kitchen with a microwave.
From the sample descriptions shown in Fig 32 it can be noticed that inaccuracies in the description vary from minor errors that can be ignored to image descriptions that don’t go very well with the test image. Most common mistakes include: singular versus plural mistakes in the descriptions; presence of words that are commonly associated with one another although they are not present in the image; inaccurate action recognition due to the absence of visual temporal information; and failure to identify unforeseen objects or unforeseen views of an object leading to inaccurate descriptions. Some of these inaccuracies like singular versus plural errors and presence of words not in the image could be reduced by employing an RCNN [17] based network to detect objects within an image.
Chapter 9  Conclusions and Future Work

FRMM enables the fusion of multiple arbitrary length sequential streams, whereby each respective recurrent network can be learned simultaneously without sharing parameters. The results presented in this thesis work show that FRMM yields state-of-the-art results on both the Flickr30K and MSCOCO datasets. It has also been shown that recurrent networks are beneficial for learning image features from static images.

Future work involves improving the performance of existing FRMM models for image description by employing RCNNs and using FRMM models for other multi-modal applications such as video description, speech to text conversion and language translation and determining their efficiency.
References


