Ensemble Learning on Deep Neural Networks for Image Caption Generation

by

Harshitha Katpally

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Ajay Bansal, Chair
Ruben Acuna
Javier Gonzalez Sanchez

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Capturing the information in an image into a natural language sentence is considered a difficult problem to be solved by computers. Image captioning involves not just detecting objects from images but understanding the interactions between the objects to be translated into relevant captions. So, expertise in the fields of computer vision paired with natural language processing are supposed to be crucial for this purpose. The sequence to sequence modelling strategy of deep neural networks is the traditional approach to generate a sequential list of words which are combined to represent the image. But these models suffer from the problem of high variance by not being able to generalize well on the training data.

The main focus of this thesis is to reduce the variance factor which will help in generating better captions. To achieve this, Ensemble Learning techniques have been explored, which have the reputation of solving the high variance problem that occurs in machine learning algorithms. Three different ensemble techniques namely, k-fold ensemble, bootstrap aggregation ensemble and boosting ensemble have been evaluated in this thesis. For each of these techniques, three output combination approaches have been analyzed. Extensive experiments have been conducted on the Flickr8k dataset which has a collection of 8000 images and 5 different captions for every image. The bleu score performance metric, which is considered to be the standard for evaluating natural language processing (NLP) problems, is used to evaluate the predictions. Based on this metric, the analysis shows that ensemble learning performs significantly better and generates more meaningful captions compared to any of the individual models used.

DEDICATION

This thesis is dedicated to my parents and family for being my constant support system, whose love and blessings towards me has led to the successful completion of my thesis.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude and indebtedness to my thesis mentor Dr. Ajay Bansal for his valuable suggestions, constant supervision, timely guidance, keen interest and encouragement throughout my thesis. I am thankful to Dr. Ruben Acuna and Dr. Javier Gonzalez-Sanchez for being supportive of my research and for willing to serve on my thesis committee.

I would also like to thank the Flickr team for their impressive free image datasets focused on the image captioning problem. Without that, it would have been very difficult to find a dataset with such a diverse collection of images.

Finally, I would like to thank Arizona State University for providing the amenities required in the successful completion of my thesis and the department of CIDSE for the constant guidance and assistance throughout my master's program and thesis.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

Table Page

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Overview

The emergence of the field of deep learning has helped in solving an enormous number of problems for some time now. One among these problems, is the problem of automatic image caption generation by computers, which has many applications in different domains. Some of the applications of image captioning include helping visually impaired people understand what the images contain by providing short descriptions, communicate to clinical experts regarding potential disease conditions found in medical images and to convert images to text, which can be used in certain applications where inferences are made only from textual data. With all the advantages of image captioning mentioned, it is crucial to develop state-of-the-art algorithms that can solve this problem and generate captions with acceptable accuracy.

As the problem deals with image processing, it can be classified as a computer vision problem, but including an additional step of generating textual sentences. So, a solution to this problem will have two phases, 1. A feature extraction phase, where features present in the image will be extracted and represented in the form of a feature vector, 2. A sentence generation phase, which takes as input the feature vector and a sequence of words are generated one after another which when combined together gives a meaningful description of the image. While the fundamental approach to solving this problem is established, there is still scope for research in this area, towards designing new models that can improve the accuracy of predictions.

The main focus of this thesis is to explore the effects of ensemble learning techniques on the problem of image captioning, which have long been proven to be very efficient in making better predictions when compared to single model settings.

1.2 Problem Statement

The main problem in image captioning is that of high variance, which occurs with the use of deep learning models as the models try to learn the specifics of training data. This means that the models used to solve image captioning have a lower tendency of generalizing on features from the training dataset and hence cannot give good predictions on new data. This thesis identifies the problem of high variance in image captioning and proposes an approach to solve it.

Though there are already existing sequence-to-sequence deep learning models for this purpose, this thesis proposes the use of ensemble learning techniques on these models. Deep learning neural networks can learn nonlinear complex relations in the data because of their deep structure and very large number of weights which get modified to represent these relations. But this nonlinearity gets reflected in the fact that the models tend to have a high variance, i.e. overfitting on the training data and not being able to generalize very well. One solution to reduce variance in deep models is to use the benefits of ensemble learning techniques which have been proven to solve the same.

These ensemble techniques explore the predictions of different good models and combine them to get one single better prediction. Not just implementing ensemble

learning on the problem but answering why and how the use of these techniques improves the performance over using a single model is the main goal. This thesis also explores the various ensemble techniques that are available out there and analyzes the best ones suitable for the image captioning problem.

CHAPTER 2

BACKGROUND LITERATURE

2.1 Image Captioning

It is common to write short textual descriptions for images to represent and understand the objects and actions in the images by humans. This task when performed automatically by a computer is known as image captioning. It is a significant problem which needs to be solved because of its potential applications in real life. The application can help visually impaired people understand the content in images through the descriptions [1]. These captions can also be saved for later use i.e. when there is a need to retrieve images solely based on this textual description. Image captioning in medical field can help doctors comprehend the disease or infection based on the descriptions of medical images. These benefits of using an image captioning system clearly provide a reason for further research in this domain to increase the accuracy of generating captions.

It is pretty easy for humans to produce a short caption while looking at the image as humans have the ability to easily understand the objects in the image, the interactions between these objects and combine them to form meaningful sentences. This ability of humans has been achieved over time with lots of experience by observing and learning. So, it is not very easy to replicate this behavior into machines and expect them to match human level accuracy. But it is not impossible either. A computer can also be trained to learn from different images, gain experience and understand the relations between objects to convert them into meaningful descriptions in a similar way that humans have gained experience. But the human brain is far more intelligent as it can learn quickly and draw

relations easily with little experience when compared to a machine's intelligence. Hence, there is a need to develop state-of-the art systems which will be capable of performing this task with at least close to human level intelligence. Figure 1 provides examples of captions describing two images.

"A brown dog is playing with football in a park"

"A boy in white shirt is playing basketball in the court.

Figure 1. Examples of Image Captioning. Source: https://www.analyticsvidhya.com/blog/ 20 18/04/ solving-an-image-captioning-task-using-deep-learning/

2.2 The Role of Deep Learning

Deep learning is a subfield of machine learning put forward in 2006, which is inspired from the functionality of neurons in the human brain. Another name this field has is deep neural networks which means it is an extension to the neural network architecture. The invention of neural networks has entirely changed the approach to problem solving. Neural networks are a replica of the neurons in the brain, the way they are connected and the way in which they transfer information among themselves. They learn from experience, experience here is the data that is fed into the network. They learn the trends or patterns in data, so that they can predict the same when they see similar or entirely new data as well. The greater the amount of data available to train the network, the greater experience the networks can gain, and the greater will be the accuracy of predictions. The smallest unit that makes up a neural network is a neuron in which most of the computation takes place i.e. neurons store within them different features that make up the data by changing weights on them to align to the patterns in the data. These features are combined together to predict the necessary. So, there is an input layer to the network, multiple hidden layers which comprise of these neurons stacked together and finally an output layer for predicting the output as shown in Figure 2. The greater the number of hidden layers the deeper the network.

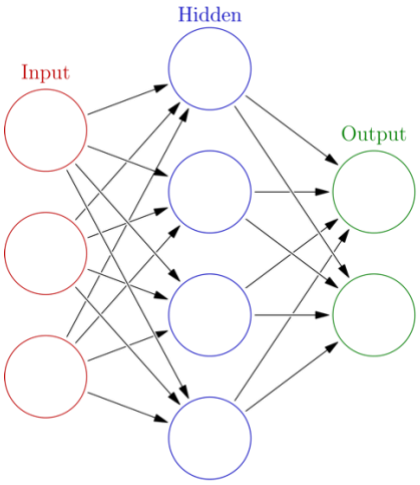

Figure 2. Standard Neural Network Architecture. Source: https://en.wikipedia.org/wiki/Artifi cial_neural_network

Deep learning involves very deep neural networks i.e. a large number of hidden layers and a greater number of neurons in each layer to learn deeper relations in data. The output from lower layers on which simple computations are performed is regarded as an input to the higher layers [2]. The increase in the network size demands an increase in processing speed and an increase in the size of data to take advantage of the deep structure. With the expansion in computational capacity and explosion of the amount of data available, neither of them is a problem. One main advantage of deep learning is that the models tend to get better and better by being trained on more and more data, unlike usual machine learning algorithms like decision trees, Bayesian networks, etc. as depicted in the graph in Figure 3. It can be clearly understood from the graph that the performance of traditional machine learning algorithms reaches a constant with increase in data size after a certain point. But the performance of deep learning algorithms keeps increasing as the amount of data increases.

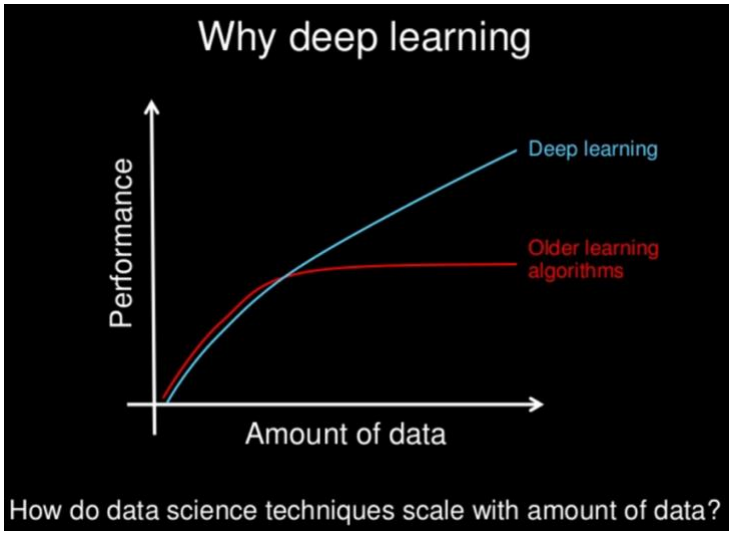

Figure 3. Performance Comparison of Deep Learning v/s Older Learning Algorithms with Increase in Data Size. Source: https://www.slideshare.net/ExtractConf

One problem that deep learning algorithms can solve efficiently is computer vision problems. Computer vision is a field that deals with computers handling and analyzing digital data such as images or videos. With the deep structure they have, deep networks can extract intricate features from images and videos to solve various problems such as image classification, object identification, etc. With the propagation into deeper layers of the network, it understands information from the low-level to high-level details of the digital content in that order, which is different from the usual neural networks. Figure 4 explains the differences in how different machine learning approaches solve the problem of feature extraction. A lot of research has been done in the deep learning domain and a lot of algorithms developed to solve distinct problems. Since, image captioning involves analyzing images to extract features from them, it can be classified as a computer vision problem and hence the use of deep neural networks.

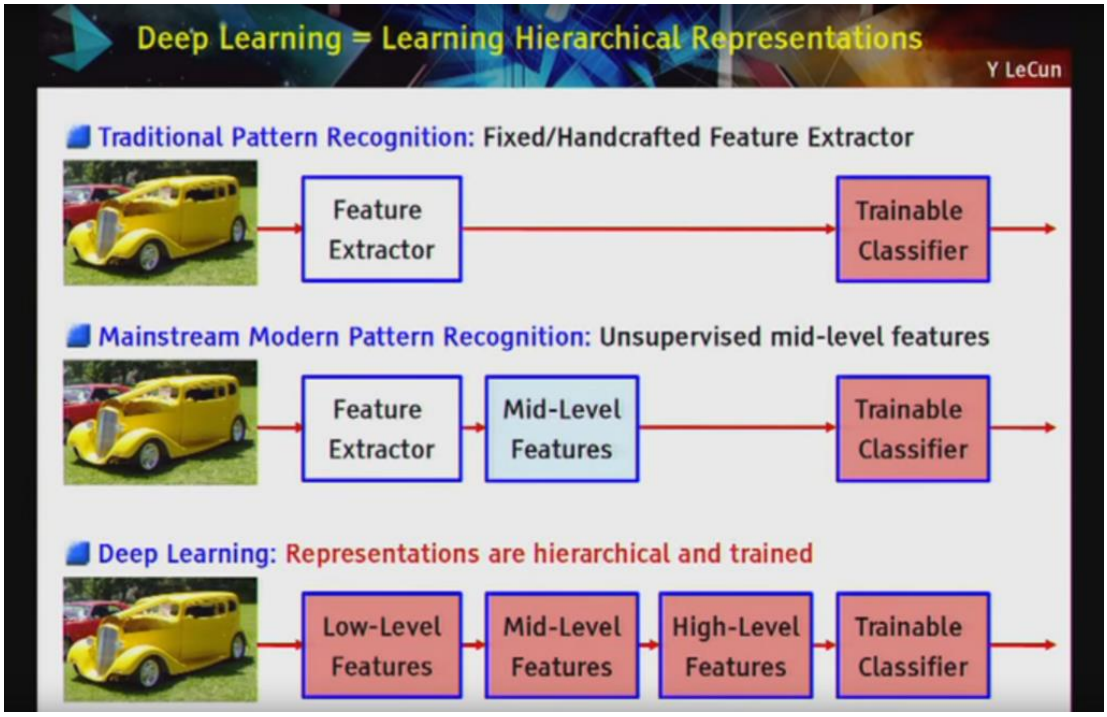

Figure 4. Feature Visualization of Deep Neural Networks. Source: https://slideplayer.com/slide/10202369/

Apart from computer vision, Natural Language Processing (NLP) also plays a very crucial role in the phase of generating captions from image features. NLP is the domain which deals with building computational algorithms that can automatically analyze and represent human language. The recent trends in the representation of natural language and the new algorithms that have been developed to generate natural language words and sentences have shown scope for solving problems like image captioning. The inception of Recurrent Neural Networks (RNNs) helped in handling situations where the next output in the sequence depends on the previous output. This is best suited for this thesis problem as there is a need to generate words in the caption in a sequential manner. The concept of distributed representation of data in deep learning, to understand the context of word

occurrences, has also been designed. All these advantages of deep learning combined give it an edge over other fields to be used to solve the problem at hand.

2.3 Basic Control Flow

The fundamental procedure to solve the problem of image captioning consists of two phases. The first involves extracting salient features of the objects and actions in the image and representing them in the form of a feature vector. The second phase includes understanding the relations between the features and generating a sequence of words one after another. The next word to be generated depends on the previous word that has been generated in the previous time step. Solving each of these phases requires special algorithms provided by the deep learning domain. It also requires the use of a special model known as the sequence-to-sequence model. The sequence-to-sequence model is a state-of-the-art single end-to-end model that is used to solve problems where sequence of inputs is to be converted into a sequence of outputs, instead of the tedious work of requiring a pipeline of models. It can be viewed as comprising of two individual steps. 1. The encoder, 2. The decoder [3], the architecture of which is represented in Figure 5. The encoder stage is where features from images that are input to the model will be encoded into a vector and in the decoder stage, the model decodes the vector to represent an understanding of the relation between the features in the form of a sentence. An interesting characteristic of the decoder is that it has the ability to understand the context and the effect of the previous word generated in the network on the next word.

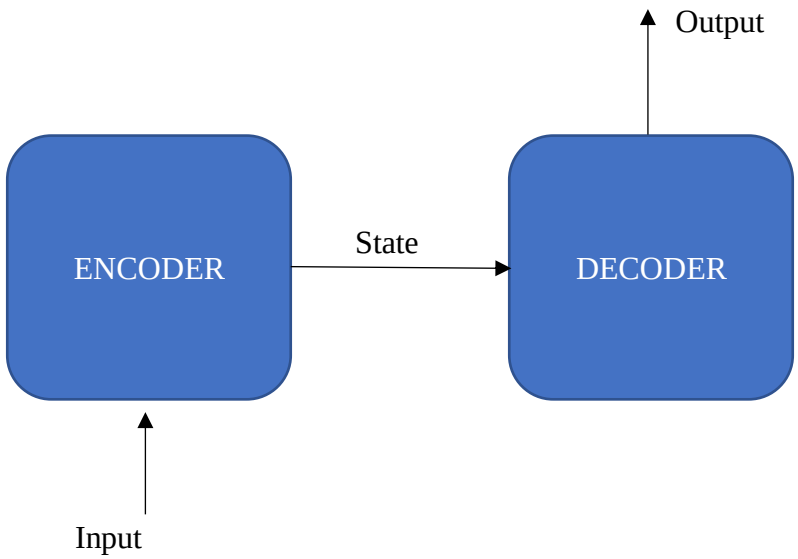

Figure 5. Sequence-to-Sequence Model Architecture.

2.3.1 Image Representation

The basic building block of an image is pixels. It is the color or light value that occupies a specific place in the image. In computer vision, images are usually represented as their RGB values or grayscale values. If represented as grayscale, each pixel will only have one value i.e. light value and the image can be represented as a 2-D matrix of pixel values which is the size of the image. Whereas, if the image is represented with its RGB values, then a 3-D matrix is required to represent the pixel values with one 2-D array for each color. If an image is of size 64 * 64, then it will be represented by a 3-D matrix of size 64 * 64 * 3 in RGB. One can imagine how computationally intensive it can be to process thousands of images of this size and hence Convolutional Neural Networks (CNNs) are used as they have the tendency to reduce the size of images by generating a compact representation which encodes the complex features in them. Figure 6 depicts the representation of a 4 * 4 pixel image in RGB.

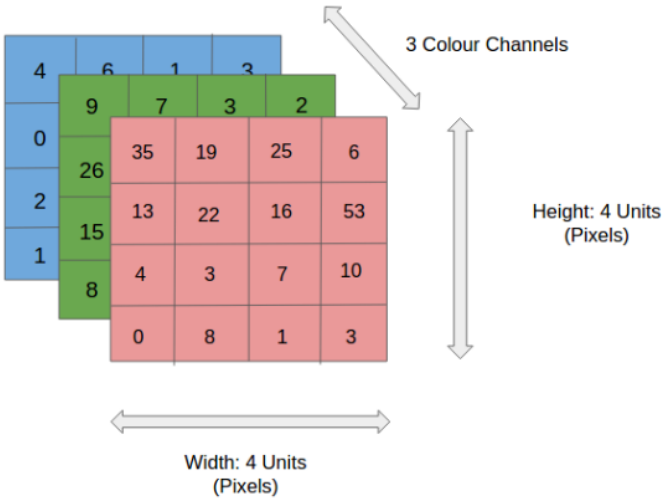

Figure 6. Representation of an Image with RGB Values. Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5way3bd2b1164a53

2.3.2 Convolutional Neural Networks (CNNs)

One particular algorithm that has helped bridge the gap between the capabilities of humans and machines is the CNN which is one of the deep learning algorithms. The main goal of CNNs is to work with images and understand the complex features that combine to form up the picture. CNN is designed in a way that they can take an input image, work with it, assign importance to objects in the image and differentiate one from another. There are three main layers that contribute to these extraordinary benefits of CNNs. 1. A convolution layer, 2. A Pooling layer and 3. A fully connected layer [4]. An entity called kernel/filter is used in the convolution layer to create a composed representation of the input image. Matrix multiplication is performed between the kernel and every portion of the image matrix of size same as the kernel, moving one step at a time as shown in Figure 7. If the image is represented as multiple channels, the depth of the kernel will be the

same as that of the image. This way the low-level features of the image such as horizontal and vertical edges are saved in a compact representation for faster processing. Initially, the convolution layers capture the low-level features of the image but with increasing number of layers, they capture the high-level features as well.

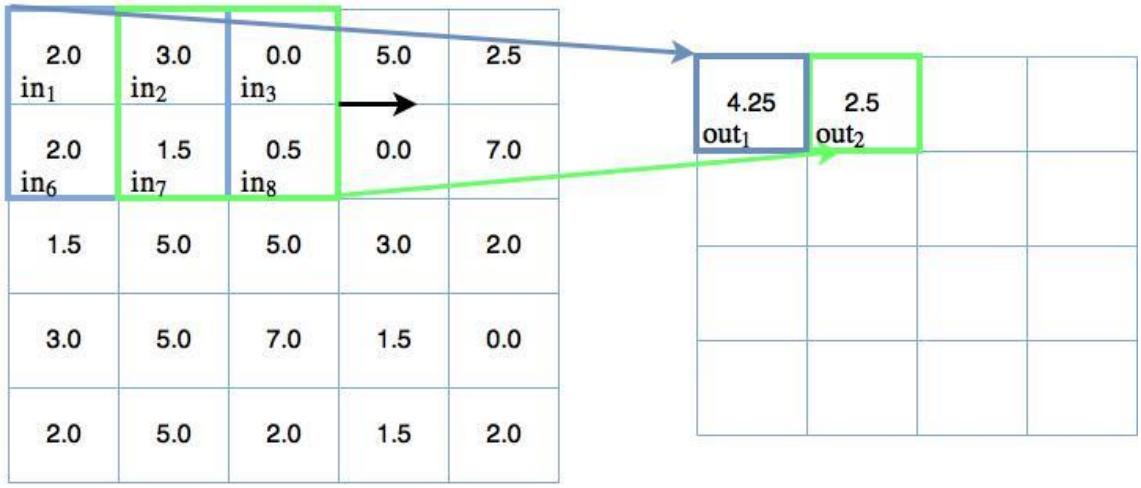

Figure 7. Initial Convolutional Operation on Input Image. Source: https://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/

Next in line is the pooling layer, serving the same purpose as the convolution layer i.e., reducing the size of the convolved feature. But it has the ability to extract more dominant features than the convolution layer. Max pooling and average pooling are the two kinds of pooling that can be performed on the convolved features to reduce the dimension even further. In max pooling, the maximum value from the portion covered by the kernel is returned whereas in average pooling the average of all the values covered by the kernel is returned. Max pooling is considered to be a better approach as it can act as a de-noising component and can discard noise activations completely. With a combination of

convolution layers and pooling layers, the features of the image can be clearly understood. The operation performed by a pooling layer can be observed in Figure 8.

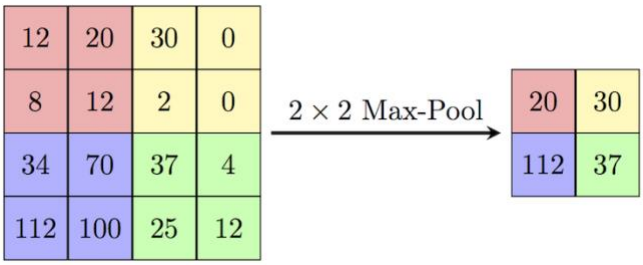

Figure 8. Max Pooling Layer Computation in CNN. Source: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

After the extraction of features, the final step is to flatten out all the features and feed it to a neural network for classification. So, the final layer is called a fully-connected layer which can learn the non-linear relations between the high-level features. This is finally fed to the usual fully connected neural network with backpropagation and to classify images using the softmax classification. Softmax classification is where an un-normalized input vector is normalized into a probability distribution, so the image is classified into the class with highest probability. This algorithm takes several epochs to differentiate between the salient and unimportant features in the images and finally perform a good classification. Figure 9 depicts the entire data flow that occurs in a CNN from input to classification.

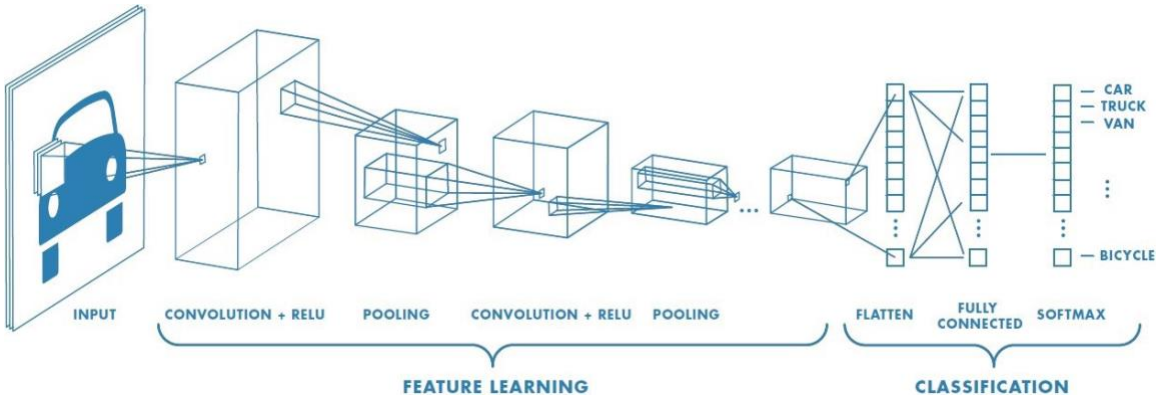

Figure 9. Convolutional Neural Network Architecture. Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5way3bd2b1164a53

Since the CNN is an efficient way to extract the features of an image, the first phase in image captioning takes advantage of this to extract features into a vector form. Although it takes considerable amount of time to train the CNN, with the high computational capacity available today, time taken can be reduced significantly. There are several pre-trained CNN models which have been trained on large datasets and the weights of these models are available as open-source. Using these models can help reduce the burden of weeks of training. These weights can directly be used to extract features from new images which can be very efficient and time saving. Some of the pre-trained models available are VGG, ResNet, InceptionV3, AlexNet, etc.

2.3.3 Recurrent Neural Networks (RNNs)

RNNs are a modified version of the traditional feedforward neural networks. In feedforward networks, at every timestep only the current input to the network is considered to classify it. It does not take into account the prediction on the previous input nor the next input in line, it is only concerned with the current input that is fed to the

network at that timestep. RNN is the exact opposite of this, which means in every step the input to the network will be the current input along with the output of the previous step as demonstrated in Figure 10. This special feature of RNNs is used to solve problems which the feedforward algorithms cannot. To facilitate this functionality, RNNs are said to have memory to save the previous output. This is often known as the hidden state. So, in each cell of the RNN network, the hidden state from the previous cell is combined with the input to the current cell over an activation function such as tanh. This will generate a new hidden state to be passed on to the next cell. So, RNNs can be used to solve the image captioning problem to generate a sequence of words describing the image using the vector of image features. But, the traditional RNN networks often suffer from short-term memory problems, i.e. while handling long sequences, they tend to forget the important information that has been acquired at the beginning of the network. This occurs because of the gradient vanishing problem present in RNNs. Gradient vanishing is a problem in which gradients in the network that are important to change the weights of the network tend to shrink as they backpropagate over time. This leads to gradients becoming so small that they do not contribute to the network at all and hence the short-term memory problem. Two variants of RNNs that have been proven to be better methods at solving sequence to sequence problems are Long Short-Term Memory model (LSTM) and Gated Recurrent Units model (GRU) which have a very similar structure as the RNN.

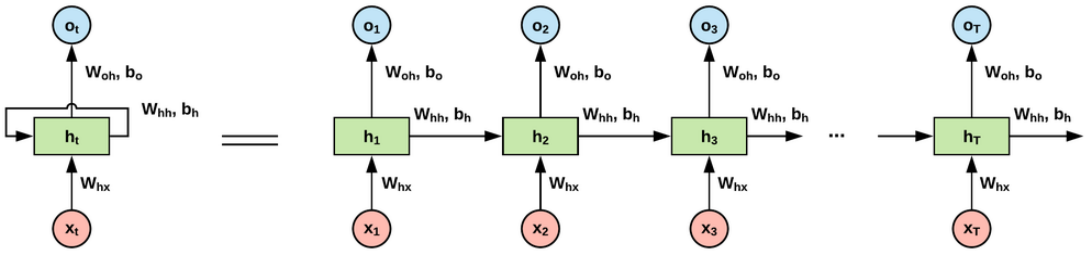

Figure 10. Architecture of a Simple RNN Model. Source: https://www.easytensorflow.com/tf-tutorials/install/cuda-cudnn?view=category&id=95

2.3.4 Long Short-Term Memory Model (LSTM)

Similar to an RNN, the LSTM stores the output from the previous cell in a hidden state and uses it to generate the next hidden state. In LSTM, information can be carried all the way from the beginning of the sequence to the current state and not only the previous state. This helps avoid the short-term memory problem. They also differ in the operations carried out inside each cell. LSTM comprises of two core components, i.e. the cell state and different gates [5]. A cell state is basically the memory of the network which can transfer information down to the present sequence state. This is how the short-term memory effect is reduced. Coming to the gates, they are also neural networks that decide whether information is to be added or removed in a current state. Three types of gates mainly form a cell in an LSTM [5]. First is the forget gate, which decides which information from the previous hidden state is to be kept and which is to be forgot. The information from hidden state and the current input is passed through a sigmoid function which restricts values to be between 0 and 1. Values which get closer to 0 are to be forgot and those which are closer to 1 are kept. Second is the input gate which is used to regulate the values from the hidden state along with the current input data. The regulation occurs by passing them through a tanh function which restricts the values to be between

-1 and 1 and multiplying these values with the values obtained from the forget gate. This way, only the information that is to be kept is finally generated. The final gate is the output gate where the values obtained from combining the outputs of forget and input gates comprise of the new hidden state which is carried forward to the next cell. Along with the hidden state, a new cell state is also sent to the next cell which is obtained by first multiplying the previous cell state with the forget vector so it can drop values in the cell state that are multiplied with values closer to 0. Pointwise adding values of the input gate with the cell state, thus generates new essential values of the cell state. The sequence of operations carried out by the gates in a single cell are described in Figure 11 and are formulated as:

$$i_t = \sigma(\, W_{xi}x_t + \, W_{hi}h_{t-1} + \, b_i)$$

$$f_t = \, \sigma\big(\, W_{xf}x_t + \, W_{hf}h_{t-1} + \, b_f\big)$$

$$o_t = \, \sigma(\, W_{xo}x_t + \, W_{ho}h_{t-1} + \, b_o)$$

$$c_t^{\sim} = \tanh(\, W_{xc}x_t + \, W_{hc}h_{t-1} + \, b_c)$$

$$c_t = \, f_t \odot c_{t-1} + \, i_t \odot c_t^{\sim}$$

$$h_t = \tanh(\, o_t \odot c_t)$$

Here, σ represents the sigmoid function and $\odot$ represents the Hadamard product. W_i, W_f, W_o, W_c represent the recurrent weight matrices of the network, b_i, b_f, b_o, b_c represent the bias vectors. h_t, i_t, f_t, o_t, $c_t^{\sim}$ and c_t are the hidden state, input gate, forget gate, output gate, input modulation gate and the cell state respectively.

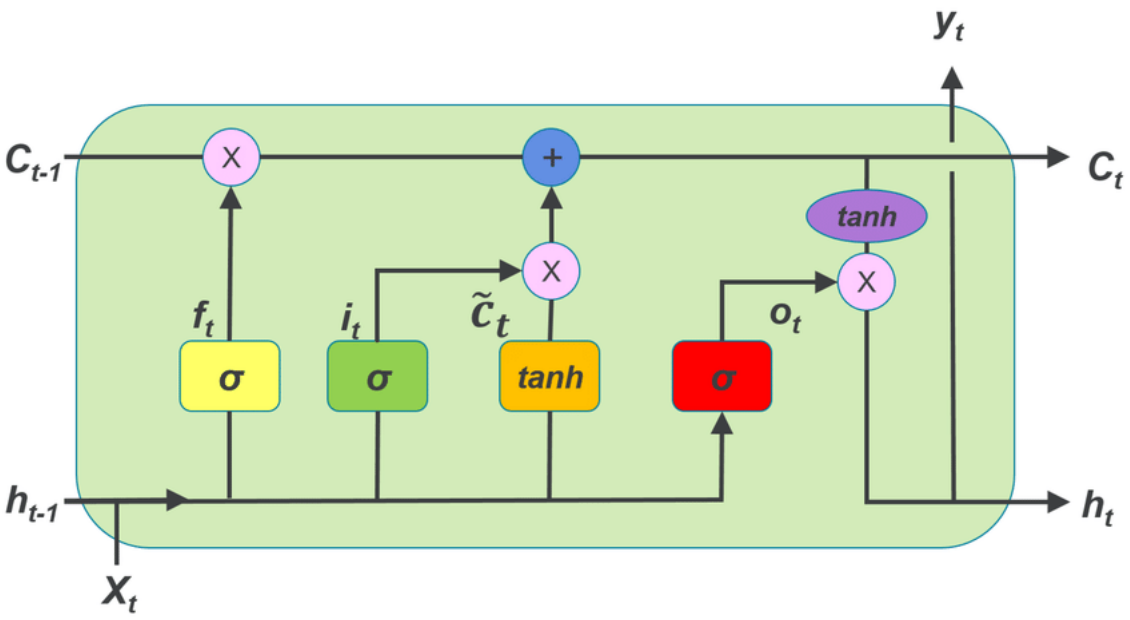

Figure 11. Operations Carried Out in a Single LSTM Cell. Source: [6]

2.3.5 Gated Recurrent Unit (GRU)

The GRU is very similar to an LSTM with the difference that the GRU does not have a cell state but instead uses a hidden state to transfer information. Also, it only has two gates namely, a reset gate and an update gate. The update gate has a similar functionality to the forget and input gates of an LSTM. It alone decides what information has to be kept for the next step and what information can be thrown away. The reset gate is where the decision of how much past information to be carried forward is decided. Since, GRUs have lesser operations they are comparatively faster than LSTMs. The sequence of operations carried out by the gates in a single cell of GRU are described in Figure 12 and are formulated as:

$$z_t = \sigma(W_z . [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r . [h_{t-1}, x_t])$$

$$h_t^{\sim} = \tanh(W . [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$$

In the equations, r represents the reset gate and z the update gate.

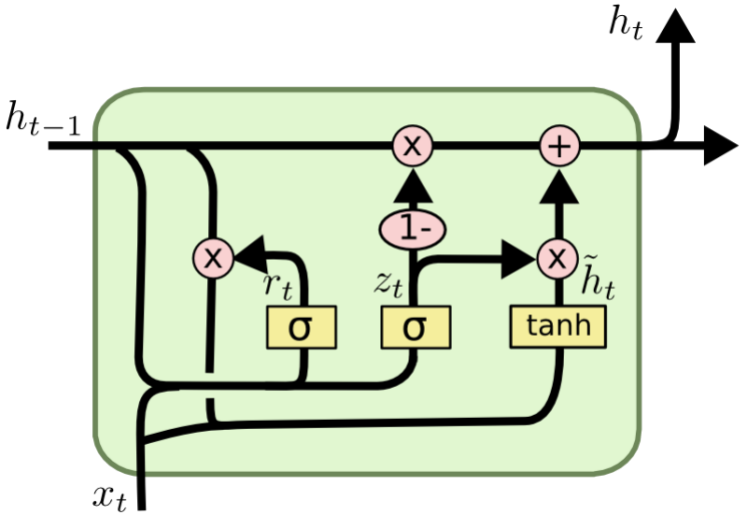

Figure 12. Operation Carried Out in a Single GRU Cell. Source: http://sqlml.azurewebsites.net/2017/08/12/recurrent-neural-network/

CHAPTER 3

RELATED WORK

There has been done a significant amount of research in generating captions for images using deep learning algorithms and this section discusses the same. Though the basic architecture of solving this problem is consistent, there have been attempts to modify the architecture in a way that gives better prediction accuracy. The scope for improvement lies in either developing better models to extract the significant features from images or in modifying the architecture of models that generate the sequence of words.

A series of CNN architectures have been designed to solve the image classification problem where objects in the images are classified into different classes. These CNNs are pre-trained on large image datasets which can be used directly without any further training for extracting features of images. But since these models are trained for classification, the outputs from the last second layer are used as feature vectors because the last layer gives a classification output, which is not necessary for this problem. Using these pre-trained models to extract image representations significantly reduces training time and are thus used widely. Several modifications have also been proposed to the sentence generation phase. Using the LSTM model with a copying mechanism for describing the novel objects in the captions is one approach which helps in selecting words from novel objects at correct places in the sentence. Another approach used to boost the prediction accuracy is using high-level image attributes in addition to the usual image representations. The relations between these attributes and image representations are explored to generate better captions. Attention mechanism is another important

technique where different regions of the image can be weighed differently and depending on these weights, captions are generated. This section gives a detailed description of each of the works mentioned above.

3.1 Feature Extraction Models

CNN is currently the state-of-the-art architecture for solving visual recognition problems. The core problem solved by them is the classification problem where objects in images are classified according to their class. A set of architectures have been trained on immensely large datasets of images which are the current top-notch architectures for classifying images. A very widely used dataset for solving computer vision problems is the ImageNet dataset containing over 14 million images and all of the pre-trained networks are trained on the same dataset to classify as many as 1000 objects. This section discusses three of the architectures that have been proposed and considered to be the best at what they do.

First is the VGG16 model proposed in [5] which achieves 92.7% top-5 test accuracy on ImageNet. The architecture of the model can be understood from Figure 13. The dimensions of the input image should be of a fixed size i.e. 224 x 224 RGB. The input is then passed to a series of convolution layers followed by max pool and fully-connected layers. The different configurations of the generic model with the number of total weights are clearly specified in Figure 14. The configurations from A to E vary in the number of weight layers in the network. VGG19 is also considered a potential candidate for feature extraction. Thus, features of the image captioning dataset images can be extracted using

the pre-trained VGG without the last softmax layer, so as to take advantage of the feature vector evaluated in the last second fully-connected layer. This model outperformed the GoogLeNet architecture existing at the time, crossing its error rate by a value of 0.9%. But there are two major drawbacks in the VGG network i.e. slow training process and the large values of the network weights which slow down the process are pertained in the network.

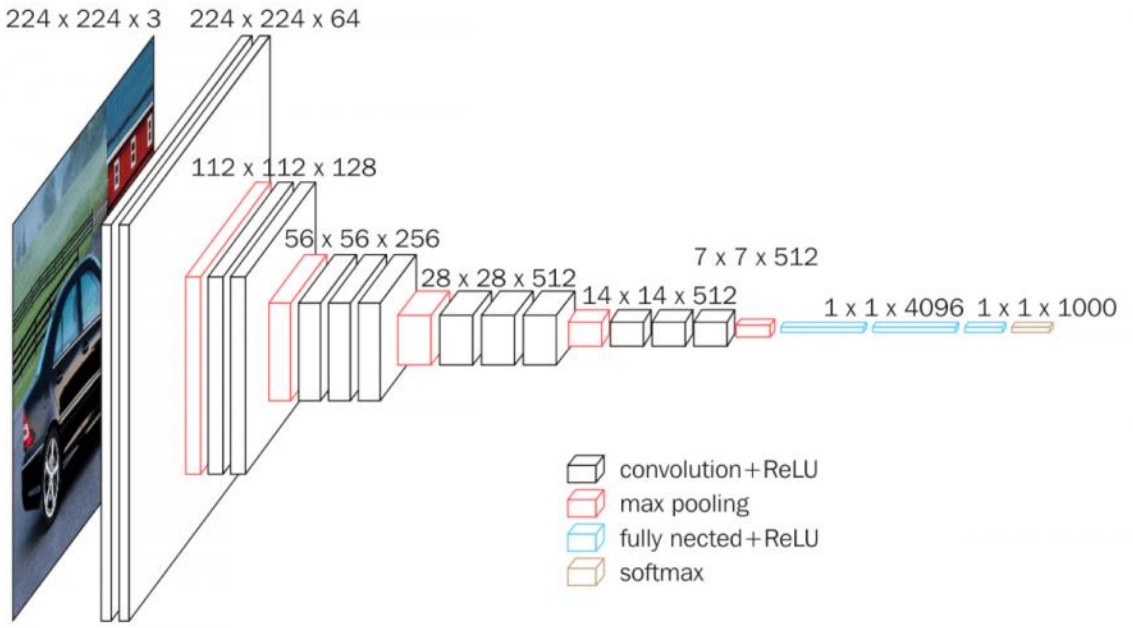

Figure 13. Architecture of the VGG16 Network. Source: https://neurohive.io/en/popular-networks/vgg16/

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 14. Configurations of the Different Layers in a VGG Network. Source: [7]

InceptionV3 is another model as proposed in [8] which has been proven to achieve an error rate of 4.2% which is less than the previously developed VGG networks. The architecture of InceptionV3 as shown in Figure 15 comprises of a combination of convolution layers, max pool layers and fully-connected layers. But, an advantage of InceptionV3 is that it performs a concatenation of multiple convolution outputs to incorporate a more precise representation of features in the network. This network also

has a lower computational cost as compared to VGG which allows it to be used to work with large amounts of data.

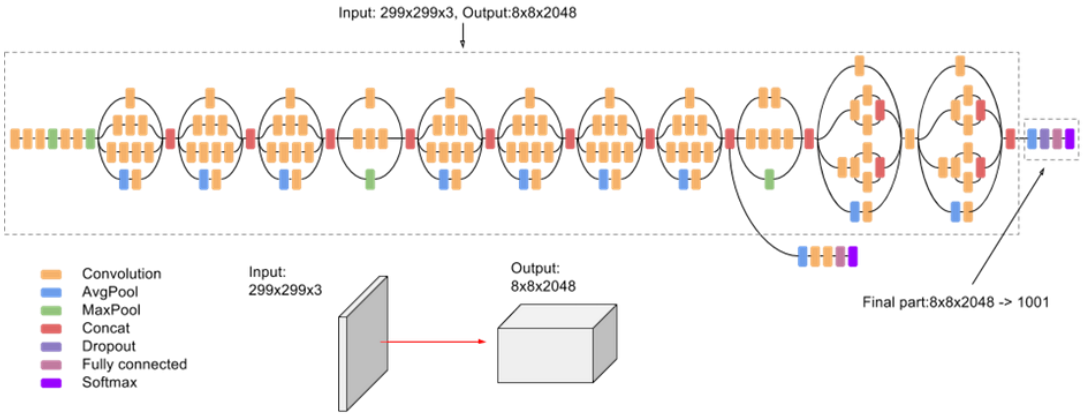

Figure 15. Architecture of the InceptionV3 Network. Source: [8]

The most recently developed pre-trained model is the ResNet50 model which allows for even better learning in deep networks when compared to InceptionV3 and VGG. Similar to both these architectures, a ResNet50 also is comprised of a series of convolution layers followed by fully-connected layers as shown in Figure 16. But it is different from them in a way that it has a special component called residual networks in the architecture. With architectures like VGG, it would result in a problem of vanishing gradients if the network is just extended in the number of layers because, the deeper the network the lesser the chance for gradients to get updated. So, this results in a vanishing gradient problem as the networks get deeper. To resolve this problem, a residual network has been introduced and used in ResNet50. A residual network typically stacks multiple layers into residual blocks and applies an identity function so that the gradient is preserved [18]. This way multiple layers can be stacked together so the images can be trained on much deeper networks.

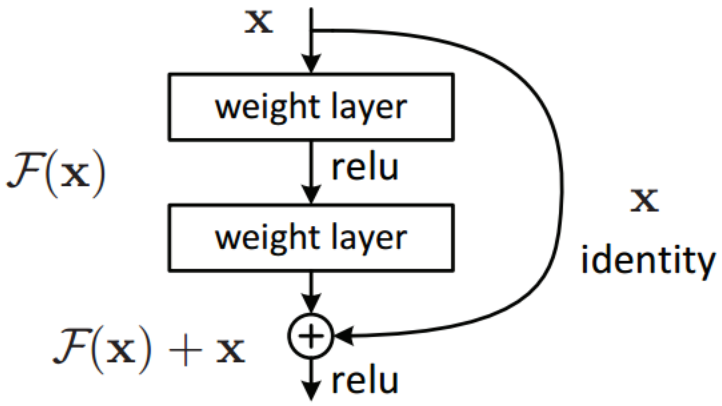

Figure 16. Architecture of a Single ResNet50 Block. Source: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec

3.2 Sentence Generation Architectures

Along with feature extraction, research has also been done in the way sentences can be generated from the extracted features. This section talks about different strategies that have been used to generate accurate captions.

A copy mechanism has been proposed in [1] where the architecture of the model is designed in a way to detect the novel objects in an image. The detected objects are then directly be introduced in the generated sentence. Initially, an image is input to a CNN which potentially extracts the features representing the image. These features will then be fed to an LSTM which generates a sequence of words. While LSTM is in the process of generating words, objects in the image are also detected using any one of the pre-trained models available out there. A copy layer is introduced on top of the entire architecture to combine the LSTM network with the copying mechanism so that novel objects detected from copying can be directly included in the sentence generated by the LSTM. This

approach towards generating captions helps in accommodating words in the sentence which closely relate to the objects in the images. In Figure 17(a), the vocabularies of the image-sentence dataset and the separate object detection dataset are represented, while Figure 17(b) depicts the architecture of copying the words generated from image recognition directly into the sentence being generated by the LSTM from the feature representations injected by CNN.

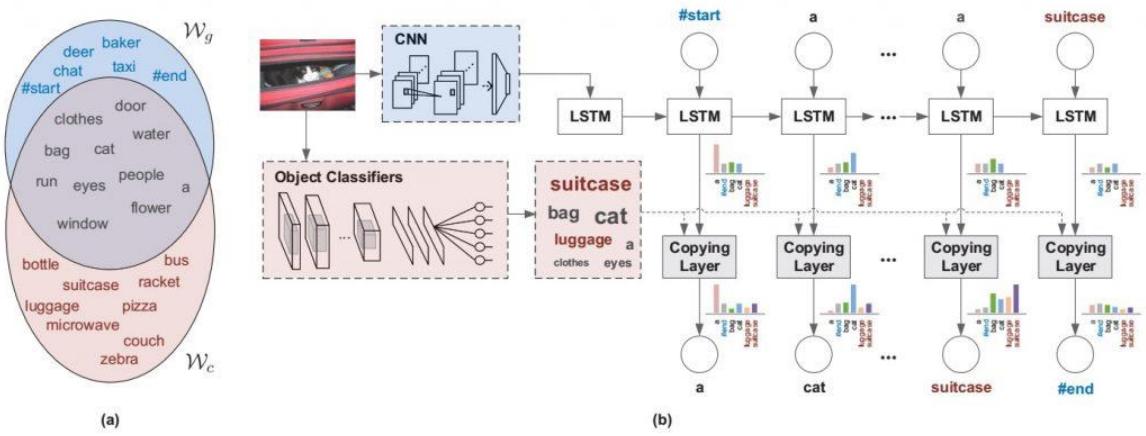

Figure 17. Architecture of Long Short-Term Memory Model with Copying Mechanism (LSTM-C). Source: [1]

Another model worth describing is image captioning with attributes proposed in [10]. In this approach, a series of variant CNN and RNN architectures are constructed by feeding the image representation along with attributes to the network in different ways to explore the relationships between them. Here, attributes are the properties seen in images which highly contribute to the salient objects in them. A variety of architectures are used to understand the impact of using both representation of images by feeding them to the networks at different times. Five variants of architectures have been proposed in the

paper as shown in Figure 18. In the first one of the architectures, only attributes are fed into the LSTM network. Both image representations and attributes are fed one after another in the second architecture. In the third, the order of feeding image representation and attributes is revered to that of network 2. In both fourth and fifth architectures, attributes and image representation are fed into the network together, but the orders are reversed.

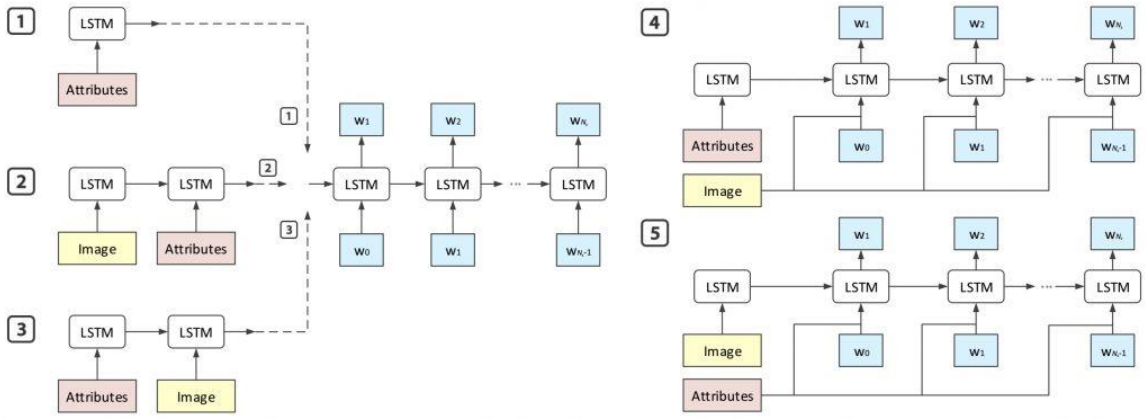

Figure 18. 5 Variants of the LSTM-A Architecture. Source: [10]

Another very important model that has been proposed to solve image captioning with greater accuracy is the use of attention mechanism in the model as proposed in [11]. The reason that humans can generate captions with the greatest accuracy is because their brains work by giving attention to the important things and less to the ones that are not very important. This way it can capture the important aspects of what the eyes see and thus will be able to generate sentences by focusing more on the important aspects. This capability when introduced into a machine is known as attention mechanism. The encoder-decoder models often tend to not perform very well as the length of the sequence

increases. Attention also helps in addressing this limitation of handling long sequences and also in speeding up the learning process. Two approaches towards using attention mechanism in image captioning have been proposed: 1. A soft deterministic mechanism that can be trained using standard back-propagation and 2. A hard stochastic model trained by maximizing an approximate variational lower bound. Using this mechanism has achieved state-of-the-art accuracy on the image captioning problem. Figure 19 represents how attention mechanism allows an additional computation of salient as well as non-salient regions of the image and include them in the final caption.

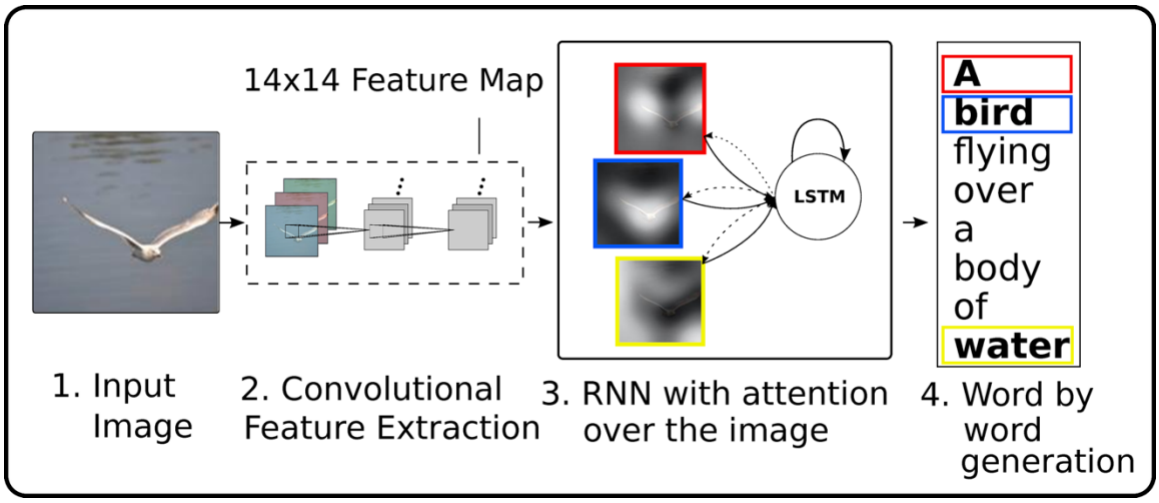

Figure 19. Architecture of Solving Image Captioning Problem with Attention. Source: [11]

CHAPTER 4

IMAGE CAPTIONING SYSTEM ARCHITECTURE

This section discusses about the structure into which input data has to be transformed to prepare it to train the model and defines the fundamental model to be used. First it is crucial to gather a good dataset to use for training. The Flickr8K dataset has been used in this thesis as it is relatively small compared to other available datasets and hence computations can be done faster. But it is also realistic and includes a good set of samples sufficient to learn good patterns from them. The dataset is available as open source. This dataset has a total of 8000 images with 6000 for training, 1000 for development and 1000 for testing. Every image in the dataset is associated with five different captions all of which can be used for training the model. One folder contains all of the 8000 images with unique identifiers as file names. Another folder contains four text files, where three of them have identifiers of the train, development and test images, and the fourth file has image identifiers associated with all five different captions for every image in the dataset.

4.1 Feature Extraction

These images and image descriptions have to be pre-processed and prepared before they can be used to train the models. Apart from preprocessing the images in the dataset, they have to also be converted into vector representations as machine learning models only accept vector inputs. These vector representations extract the critical features from images and represent them as float values. The size of the vector depends on the output layer size of the model being used for the purpose. As explained in section 2.3.2, CNNs are the best models available to extract feature vectors from images. It has also been

mentioned that a number of pre-trained models exist for image classification which can also be used to extract the features from images in the dataset. Since, highly tested and accurate pre-trained models already exist, the burden of training a new model to extract features from images has been avoided in this thesis. The Resnet50 pre-trained model has been used for this purpose as it is currently the most accurate model for image classification. Though this model is pre-trained for image classification, it can be used for extracting a feature vector for all of the images by removing the last layer of the model which performs the classification task. The layer before the last layer known as the convolutional base, outputs a vector representation of a particular image which can be used to train the image captioning model. This model can be used in combination with the image captioning model, but generating image features every time a new model is tried will be computationally expensive. Hence, feature vectors are extracted for all of the 8000 images in the dataset and saved to a pickle file. This file can later be loaded into the application and fed to the model whenever a new model is interpreted. The Resnet50 model requires the input image size to be a 3 channel 224 x 224-pixel image. So, before the images can be input to the model, they have to be reshaped to match the model specifications. Also, the model outputs a 1-dimensional vector of size 2048 for every image in the dataset. This concludes the process of generating a feature vector representation of images to be used by the problem model. Figure 20 represents how an image of a specific size is given as input to a pre-trained network and the feature vectors are generated, also of a pre-defined size.

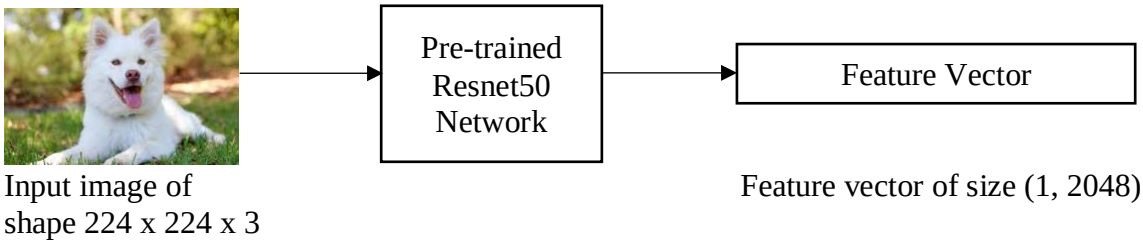

Figure 20. The Feature Extraction Process Using the Resnet50 Pre-trained Model.

4.2 Cleaning Description Text

The next phase would be to prepare a vocabulary and preprocess it from the set of descriptions in the dataset. First, the text from descriptions is extracted and it is cleaned to prepare a concise list of vocabulary. Cleaning of text includes converting all words into lower case, removing any alpha numeric content, removing all punctuations and remove words that are less than 2 characters in length. After performing cleaning on the text data, a set of vocabulary can be generated, and the descriptions can be stored in a separate text file so that it can be retrieved whenever required.

4.3 Defining the Model

The next phase is to define the structure of the fundamental model that is used in this thesis to generate captions. The model that will be defined has to generate a sequence of words those when combined form a meaningful sentence explaining something about the input image. So, first the previously generated feature vectors of all images and the saved descriptions are loaded into the model for the purpose of training. The caption generation phase is carried out by a recurrent neural network which generates words one after

another. So, to train the model, a list of previously generated words has to be input, and the next word should be the output. To do this there has to be a start word to start caption generation and an end word to notify that the sequence has reached end of the caption. To incorporate this, two tokens are appended to the beginning and end of every description present in the dataset, namely <startword> and <endword>. This is done when the descriptions are loaded into the model. As text cannot be directly input to a model for training, every word in the vocabulary is converted into an integer representation along with the start and end tokens. So, now every description is represented as a string of integer values where each integer uniquely identifies one word.

When the model is trained, every image is used to train the model for the number of times as the length of the image description. The model will first be provided with the image features and the first word in the sequence as input and the next word as the output. Then the first two words along with the image features go as input while the third word is the output. This is repeated until the <endword> is reached for that image description. An example of this combinations of input-output pairs is shown in Table 1:

For example, consider the input sequence “Dog is running on beach”

I1	I2	y
image_features	startword,	dog
image_features	startword, dog,	is
image_features	startword, dog, is,	running
image_features	startword, dog, is, running,	on
image_features	startword, dog, is, running, on,	beach
image_features	startword, dog, is, running, on, beach,	endword

Table 1. Input and Output Pairs for One Data Point in the Dataset.

I1, I2 represent the two inputs to the model and y represents the output word. A list of these input-output pairs will be generated prior to training the model. So, the input to the model will be in the form of two arrays, one is the image feature vector and the second is sequence of tokenized words, while the output is only one word that is next in line. The output of the model is not the exact next word, but a probability distribution over all the words in the vocabulary. This means the output is represented in the form of one-hot encoding which is how a word is represented in the model, i.e. the position of the word which is chosen as output is represented with a 1 and all the other words are represented with a 0. Also, the maximum sentence length of all the descriptions in the dataset is calculated as it is required to append 0 values to the sentences which are shorter than the longest sentence. This is essential to make sure that all sequence of words input to the recurrent model are of the same length. This means a series of sequence of words are

generated with pairs of image features and all sequence inputs along with single word outputs before training the model.

The final step is to design the model to incorporate the goal of generating a sequence of words given the image features and all previous words generated so far. To achieve this, it is necessary to merge the image features generated from the pre-trained Resnet50 model and the sequence of words that the recurrent neural networks handles. So, the outputs from these two layers which are of fixed length are merged together by the decoder to make the final prediction. But, before inputs can be passed to the recurrent layer, the words in the vocabulary are embedded. With a total of 40000 descriptions, i.e. 5 descriptions for every image, the vocabulary is very huge, up to 8000 words. Since every word is represented by one-hot encoding, every word will be a very sparse vector representation. The process of converting these sparse vector representations into a dense continuous representation of vectors is the role of word embeddings. This representation helps in identifying relations between different words [15]. For example, the words ‘hot’ and ‘oven’ often occur together though they are represented with different vectors. Converting the word representations into dense vectors provides capabilities to compare the correlations among words. So, if ‘hot’ and ‘oven’ appear in the same context, they are closer to each other and this is what is represented by the word embeddings layer. In Figure 21, the word embeddings for a vocabulary of 9 words is given, where the advantages of embeddings can be observed. It can be understood from the figure that similar words have similar vector values and the embeddings are dense. Hence, it is

crucial to convert the words in the vocabulary into a dense embedding before going to the recurrent layer.

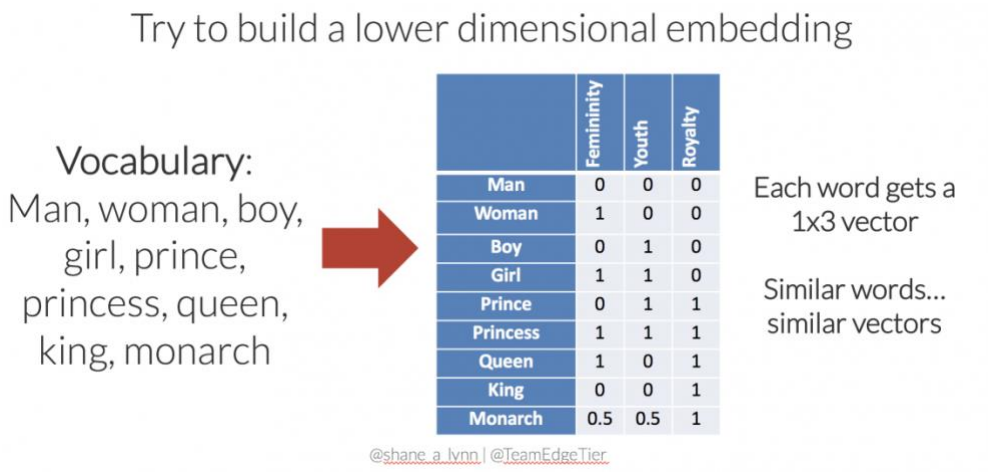

Figure 21. Word Embedding Representation for a Vocabulary of 9 words. Source: https://www.shanelynn.ie/get-busy-with-word-embeddings-introduction/

After a word embedding is generated for all words in the vocabulary, the output from this layer is input the recurrent neural network layer [17]. This layer can be a simple RNN layer, an LSTM or a GRU layer. In this thesis, all three networks have been explored and used to generate a distinct set of ensembles. Finally, both the input models are merged together, which is then connected to a dense layer with a softmax activation function to predict the next word in sequence. Figure 22 illustrates the control flow from when the image is input to a pre-trained CNN, the image feature vector is generated which is input to the recurrent layer along with word embeddings and the network outputs the sequence of words one after another. The weights of every model that is trained is saved to an HDF5 every time the loss on validation dataset decreases. This step is essential because

the same model can later be used to evaluate its performance or make predictions on new data. This ends the description of the primary model structure used in this thesis.

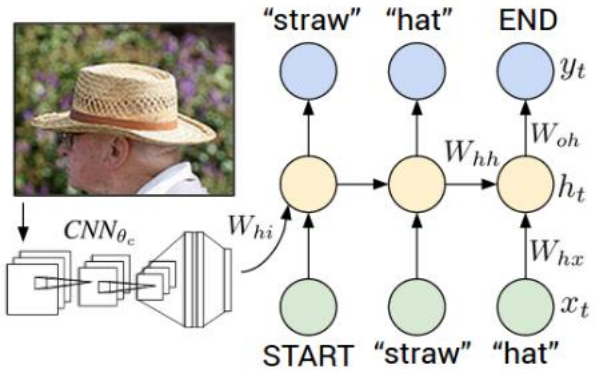

Figure 22. Model Architecture for Image Captioning. Source: https://towardsdatascience .com/image-captioning-in-deep-learning-9cd23fb4d8d2

4.4 Model Evaluation

Evaluating the model that has been trained is pretty straight forward. Since, the weight of all models at all epochs are saved, they can be used to evaluate the performance on the test dataset. There is a need for a special evaluation metric to calculate the performance of the model. Since the output of this problem is a sequence of words, evaluation metrics used in evaluated NLP problems has to be used. One such metric that is most widely used for evaluating image captioning is the Bleu score (Bilingual evaluation understudy) [16]. This score was initially put forward to evaluate the performance of machine translation systems, to understand how close the translated sentence is to a human translated sentence. Later, it found use in many potential applications. One of them is image captioning as the evaluation purpose is the same. The machine generated caption has to be compared to a human generated caption i.e. the refence caption available in the

dataset. Hence, in this thesis all model evaluations are performed using the Bleu score evaluation metric.

Blue scores range from 0.0 to 1.0 where 1.0 represents a perfect match but a 0.0 represents a perfect mismatch. Blue score works by calculating the n-grams in a predicted sentence against n-grams in the reference sentences. The higher the match count, the higher will be the bleu score. N-grams are the number of words that are chosen for comparison each time. An example of bleu score calculation is shown in Table 2:

One predicted sentence is compared with two references and the 1-gram, 2-gram, 3-gram and 4-gram bleu scores are explained.

Predicted Sentence: A boy playing in park football.

Reference 1: A boy is playing football in park.

Reference 2: A boy is playing in park with football.

N-grams	Reference-1	Reference-2
1-gram	6/6	5/6
2-gram	2/5	2/5
3-gram	0/4	1/4
4-gram	0/3	0/3

Table 2. Bleu Score Calculations for an Example Sentence.

It can be understood from the table that the predicted sentence has no 4-gram overlaps with either of the references, but has 1-gram, 2-gram and 3-gram overlaps with at least one of the references. Since the dataset has 5 reference captions for each of the images in the entire dataset, the bleu scores can be calculated for each predicted sentence with each of the reference captions. Generally, bleu scores up to 4-grams are calculated to understand the quality of the generated captions.

So, in order to evaluate every model, the model is loaded and used to generate a probability of occurrence for every word in the vocabulary. The word which has the highest of these probabilities is chosen as the next word in the sequence. This process continues until the <endword> character is reached. All words generated are then combined to generate a meaningful caption. Each of these predicted captions are evaluated with all 5 reference captions in the dataset to generate their respective bleu scores. A combined bleu score for all the predictions in the test set is generated using the corpus_bleu method available in the Natural Language Toolkit (NLTK).

CHAPTER 5

ENSEMBLE LEARNING ON IMAGE CAPTIONING

5.1 Introduction to Ensemble Learning

A lot of approaches have been adopted to increase the accuracy of predictions on the image captioning problem. In this thesis, ensemble learning, an approach which has been proven to be very useful towards solving machine learning problems is tested on the problem. Most of the machine learning algorithms suffer from the problem of high variance, which ensemble learning techniques are considered to be very good at solving. Every algorithm in the field of machine learning is associated with two very important factors called bias and variance. An algorithm can go wrong in two ways. It either makes false assumptions and misses to capture the relevant relations between features and outputs or mislead by noise and outliers in data, resulting in capturing the noisy patterns [12]. The former feature is bias while the latter is called variance of the algorithm. Models with high bias are said to underfit, whereas those with high variance are said to overfit the training data. In, Figure 23 three variations of an algorithm when it overfits, underfits and has a good balance over the dataset can be seen to understand the trade-off that bias, and variance should have. It is important to find a balance between bias and variance so that the model gives good predictions. A good algorithm will always try to achieve low bias and low variance to build a balance between predictions in future.

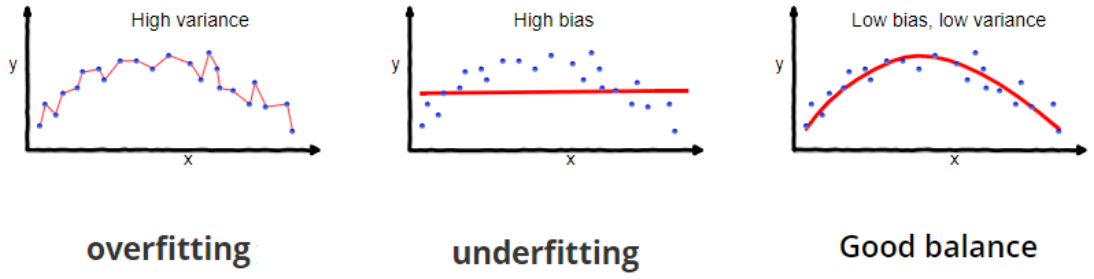

Figure 23. A Representation of Three Hypothesis that Overfit, Underfit and Have a Good Balanced Fitting over Data. Source: https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Deep neural networks are considered very flexible and to be able to scale to large amounts of data. But, a disadvantage of this flexibility is that they tend to get sensitive with every data point, i.e. they generate a new set of weights which produces different predictions every time they are trained and hence suffer from high variance [19]. Ensemble learning thus comes into picture to solve the problem of low variance faced by deep networks. Ensemble learning is the practice of training multiple good but different models instead of a single model and finally combining the predictions of all the models in a way suitable to the problem definition. This way, the predictions from ensemble learning can be better than the predictions from a single model itself [20]. In Figure 24, ensemble learning technique is applied on a classification problem, i.e. multiple classifiers are trained on the entire dataset or subsets of the dataset. A combined prediction of these classifiers is given as the final output which will be better than the output from a single classifier.

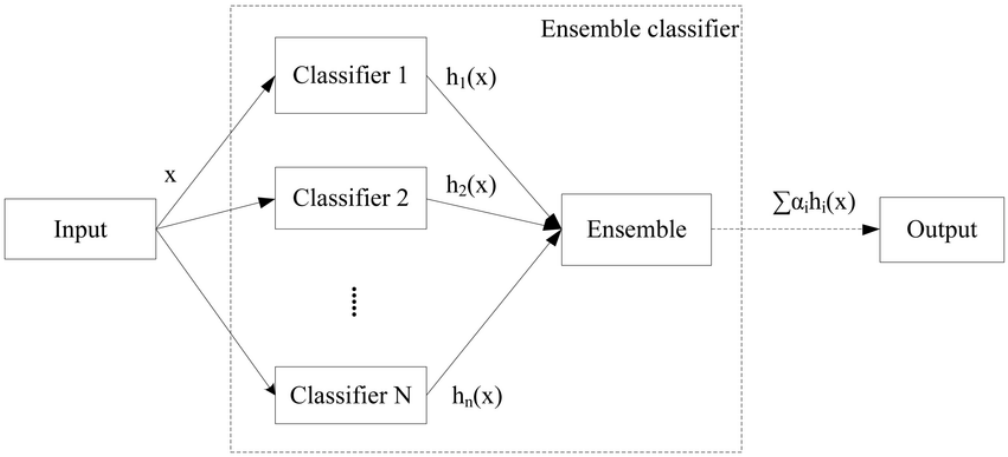

Figure 24. Basic Structure of Applying Ensemble Learning to a Classification Problem. Source: [13]

Since, image captioning is a problem which involves the use of deep learning techniques and involves dealing with long sequences of sentences, it also suffers from the problem of high variance. Every time a model is trained, it learns a different set of weights thus leading to differences in predictions. So, using ensemble learning on image captioning with multiple models and finally combining the predictions would result in better prediction accuracy over the problem. There are a variety of techniques that can be applied, and the results can be analyzed to find the right technique that can work well on the image captioning problem. Different elements of a deep network can be varied to result in different ensemble techniques. These variables include the training data, the model and the combination of predictions [14]. The number of models used in an ensemble on deep networks is often limited to a small number because of the increase in computational cost with increase in models and also drastic variations that can exist between a large set of models.

5.1.1 Varying Training Data

The training data used for training each model in the ensemble can be varied, i.e. varying the size of training data for every member. A very simple approach is the k-fold cross-validation where k different models are trained on k different subsets of the training data. The predictions from all the models combined can be used as the final prediction. Another very famous approach known as bootstrap aggregation or bagging which involves resampling the training dataset with replacement and using them to train different models. This approach allows the models to have a different density of training data which allows for the models to grow.

5.1.2 Varying Models

Using a single model on the same dataset with different initial conditions can itself result in good and different models, but the errors made by the model may still be corelated as they all learn from the same mapping function [14]. A good alternative to this would be to generate a set of differently configured models by varying the hyperparameters used in the model. Training these models with the given dataset would result in a better combination of models. Using different configurations would allow the models to learn differently and hence give different predictions which can be combined. Alternatively, for models that may take weeks to train, the best models called snapshots or checkpoints can be saved while training which can be used as ensemble members. This gives the advantage of using multiple models to be part of an ensemble along with the benefit of collecting them during a single training process. A variation of this snapshot ensemble is to save models from a series of epochs by reviewing the model performance on validation

dataset during training itself and using them as members of the ensemble. These set of models are known as horizontal ensembles. A vertical ensemble technique is also available which involves saving the outputs from the intermediate hidden layers as they contribute to the low level learned representations in data. The outputs from multiple hidden layers of different models can be combined to be used as input to a new model.

5.1.3 Varying Output Combinations

Generating a combined prediction from the predictions of multiple models used in an ensemble depends on the problem specification. Generally, an average of the predictions from different models is used as the final prediction. Different models in the ensemble can be assigned different weights if we know that one model gives a better prediction compared to the others. Based on these assigned weights, the prediction from every model is weighed accordingly and combined to give a weighted average prediction. A widely used approach to combine predictions is using a non-linear model to learn the best way to combine them, which takes into account the input data along with the predictions from each model in the ensemble. One of the best and sophisticated approaches to solve this is stacking, where a new classifier is developed that can take the outputs from each of the ensembled members as inputs and estimate the best output. Another approach to this is boosting, where every model tries to correct the mistakes of the previous models in generating predictions on the training data. Initially, a model is trained with all of the data having equal weights. This model is used for prediction on the training dataset and the data points which have been misclassified are given higher weight. So, while resampling the dataset the next time, more weight is given to these data points. The next model runs

on this newly resampled dataset and this process continues until there is no change in weights or all of the models present in the ensemble have been exhausted. One last method is to combine or average the weights generated for each of the models at the end of training and using the combined weights for performing predictions.

So, there are different combinations of ensembles available to be used on the image captioning dataset and this thesis aims at analyzing the consequences of using some of these methods and comparing the results with the ones obtained on using a single model.

5.2 Ensemble Learning Applied on Image Captioning

Based on the different types of ensemble learning techniques explained in section 5.1, a few of these techniques have been chosen to experiment with on the image captioning problem in this thesis. This section describes each of these techniques in detail and explains how they have been applied on the defined problem.

5.2.1 k-fold Cross Validation Ensemble

This type of ensemble is generated when using different sets of training data to train different members of the ensemble. K-fold cross validation is the process used to understand how the machine learning algorithm responds on new data. The entire dataset is split into k-folds where k can be any integer value. If k is chosen to be 5, then the entire dataset is divided into 5-folds. Now, one of the folds is chosen to be a hold-out or test-dataset and all the remaining folds are considered as training data. The model is trained on this data and tested on the holdout dataset. This process is repeated for k-times where

each fold gets a chance to act as a hold-out dataset. This approach can be used to generate an ensemble of models to achieve better results. As shown in Figure 25, the training dataset is split into k-folds and one of the folds is chosen as a validation fold. All the other folds are considered as training folds and a different model is used to train on each of these folds. The predictions from each of these folds are then combined for a final prediction. This process is repeated until each of the folds in the dataset act as a hold-out dataset. This gives the benefit of allowing different models to learn from different parts of the dataset so they can learn different patterns and finally the predictions combined to give a final more accurate prediction.

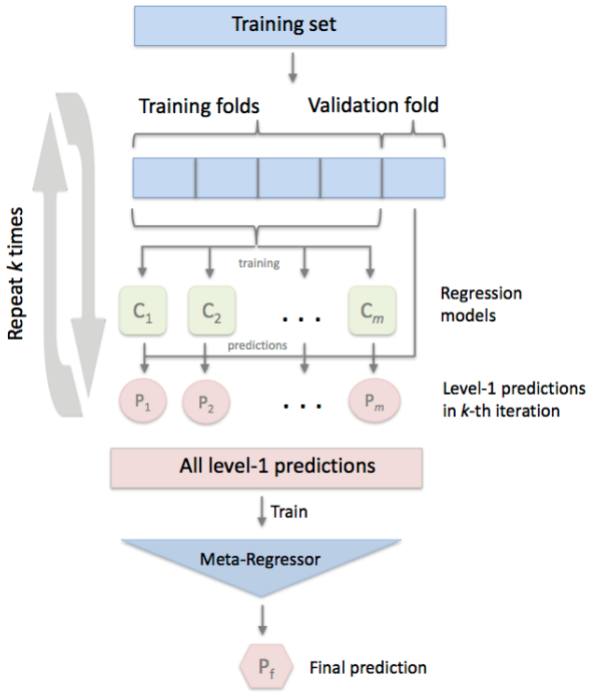

Figure 25. Representation of k-fold Cross Validation Ensemble Process. Source: http://rasbt.github.io/mlxtend/user_guide/regressor/StackingCVRegressor/

This approach has been applied on the image captioning problem. There are 7000 images available for training in the Flickr8k dataset. So, a k-fold cross validation dataset has

been generated on this set of 7000 images. Different sizes of cross validation datasets have been used to analyze the performance of all combinations of k-folds over image captioning.

5.2.2 Bootstrap Aggregation Ensemble

Bootstrap aggregation also comes under the category of ensemble learning by varying the training data size or composition for training different models. In this technique, a subset of the entire training dataset is chosen with replacement to train the model network. This approach is beneficial because it allows the models to expect a different density of samples in the training dataset when they are trained so they can reduce the generalization error. Figure 26 represents how the training data is resampled into m-subsets with replacement and used to train m different models to finally combine their predictions for a higher prediction accuracy.

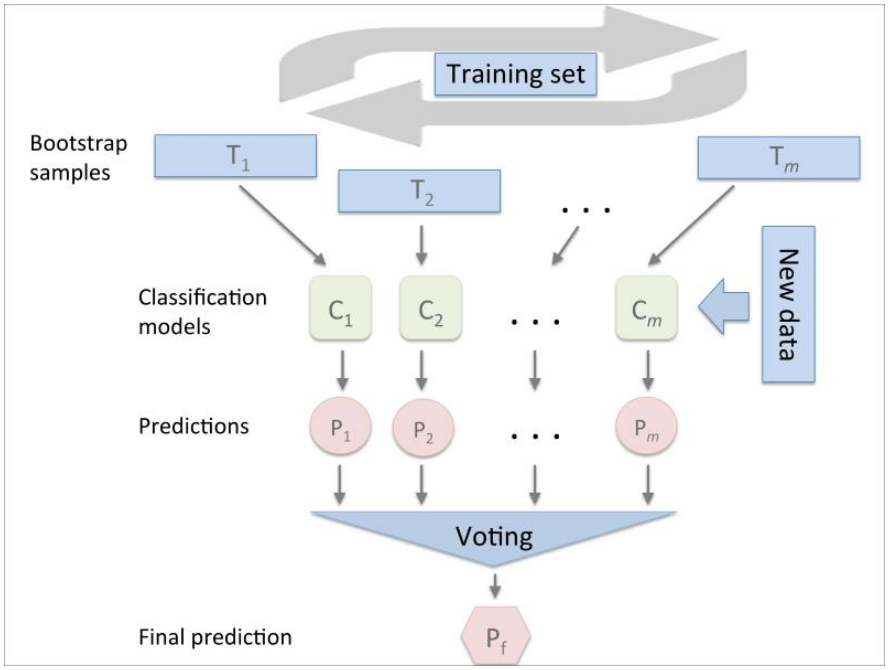

Figure 26. Representation of Bootstrap Aggregation Ensemble Process. Source: Python machine learning by Sebastian Raschaka

Implementing the bootstrap aggregation technique on the image captioning dataset has been pretty straight forward. Different number of samples have been generated from the training dataset with replacement and these samples are used to train different models based on the sample size. Finally, the predictions from each of the models have been combined to generate a final prediction.

5.2.3 Hyperparameter Tuning

This is not a separate ensemble technique but a way to generate various models that can be used with varying training data or whose predictions can be combined in different ways for better results. Hyperparameter tuning is a very important word used in the deep learning field. Every deep learning model involves the use of different parameters which

can be tuned to make it suitable to solve a particular problem. This can also be used to generate different models that will be part of the ensemble. So, by varying the different configurations of the different hyperparameters present in the model, a series of models that can perform well on the training dataset can be generated. Examples of these hyperparameters are number of layers in the network, number of neurons in each layer, activation function, learning rate, etc. This will generate a group of ensemble models which will have the capability of having minimal overlap of predictions among themselves.

There are a few hyperparameters present in the image captioning model. They include, the feature extraction model, the number of layers in caption generation model, the activation function, learning rate, normalization factor, dropout layers and dropout rate. Different combinations of these parameters and the effect they have on prediction accuracy are explained in chapter 6.

5.2.4 Boosting Ensemble

Boosting is a technique more complicated than stacking. In this approach, every data point in the training dataset is assigned a weight value and a subset of it is sampled based on the weights to be used to train a model. Initially all data points are given equal weights. The error in prediction on the training dataset is calculated after training every model. The weights of these wrongly predicted data points are increased based on the error rate, so that they have a higher chance of being sampled in the next subset as shown in Figure 27. This process is repeated until all the ensemble members are trained. All

these models are used to predict on a new image and these predictions are combined using one of the output combination techniques mentioned in the next section. This way data points that have not been learned correctly are learned by the next model. Hence a combination of predictions from all models can be guaranteed to give better predictions than a single model as every model learns from different data points and patterns.

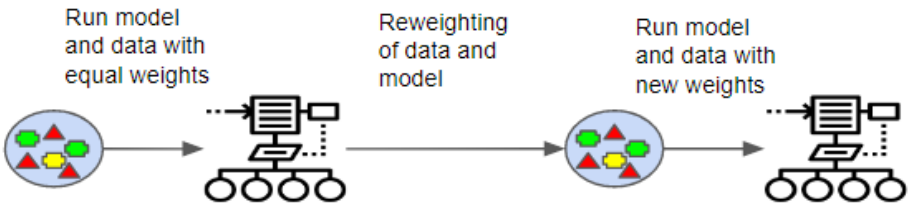

Figure 27. Representation of Boosting Ensemble Process. Source: https: //hacker.com/how-to-develop-a-robust-algorithm-c38e08f32201

5.2.5 Output Prediction Combination Variations

The process of making new predictions using the saved models and evaluating those predictions has been explained in section 4.4. This process is easy when there is only one model available to make predictions. But, when an ensemble of models is available, the process of prediction becomes complicated. The predictions of all ensemble model members have to be considered at every step so they can be combined in a form that can give better results. The ensembles on basic machine learning algorithms are combined using different techniques based on the problem being solved. This thesis explores three different approaches to combine predictions from each model.

It has been explained that every model predicts a probability of next occurrence values for every word in the vocabulary. One approach that has been used to combine these probabilities is to save the probabilities of the word which has the maximum probability according to every model. Then the word which has the highest probability among these chosen probabilities is selected as the final output. This combination metric name is abbreviated as MMP (Maximum of maximum probabilities) for the sake of easy representation. A code snippet of this method is shown in Figure 28. A second approach is to select one word from each of the predictions with highest probabilities and choose the word which has been selected by the largest number of models. This combination metric is abbreviated as MP (Maximum voting). A code snippet of this method is shown in Figure 29. The third approach is to average all probabilities generated by every ensemble member for a word and then pick the word with the highest averaged probability. This combination metric is abbreviated as AP (Average probability). A code snippet of this method is shown in Figure 30.

```
# generate a description for an image
def generate_desc(models, tokenizer, photo, max_length):
    word_seq = 'startword'
    for i in range(max_length):
        pred = []
        max_value = []
        for each_model in model:
            sequence = tokenizer.texts_to_sequences([word_seq])[0]
            sequence = pad_sequences([sequence], maxlen = max_length)
            yhat = each_model.predict([photo,sequence], verbose=0)
            max_value.append(amax(yhat))
            pred.append(argmax(yhat))
        yhat = max(max_value)
        max_index = max_value.index(yhat)
        yhat = pred[max_index]
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        word_seq += ' ' + word
        if word == 'endword':
            break
    return word_seq
```

Figure 28. Code Snippet of Maximum of Maximum Probabilities method.

```
# generate a description for an image
def generate_desc(models, tokenizer, photo, max_length):
    word_seq = 'startword'
    for i in range(max_length):
        pred = []
        max_value = []
        for each_model in model:
            sequence = tokenizer.texts_to_sequences([word_seq])[0]
            sequence = pad_sequences([sequence], maxlen = max_length)
            yhat = each_model.predict([photo,sequence], verbose=0)
            pred.append(argmax(yhat))
        yhat = max(pred, key = pred.count)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        word_seq += ' ' + word
        if word == 'endword':
            break
    return word_seq
```

Figure 29. Code Snippet of Maximum Voting Method

```
# generate a description for an image
def generate_desc(models, tokenizer, photo, max_length):
    word_seq = 'startword'
    for i in range(max_length):
        pred = []
        max_value = []
        sequence = tokenizer.texts_to_sequences([word_seq])[0]
        sequence = pad_sequences([sequence], maxlen = max_length)
        yhats = [model.predict([photo, sequence], verbose=0) for model in models]
        summed = np.sum(yhats, axis=0)
        yhat = argmax(summed, axis=1)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        word_seq += ' ' + word
        if word == 'endword':
            break
    return word_seq
```

Figure 30. Code Snippet of Average Probability Method

CHAPTER 6

ANALYSIS

This section gives a detailed analysis of the different ensemble learning techniques used on image captioning and the performance variations that each of these techniques have shown. It shows how an ensemble of models is better in making predictions when compared to a single model trained on the entire dataset. Not just their benefits over a single model, but a comparison among these models to define the best ensemble technique that can be used to make predictions for image caption generation.

During analysis, feature extraction for all types of ensemble learning has been performed using the Resnet50 pre-trained model as it has shown to provide the best prediction results during experimentation in this thesis. Every ensemble of models and their results are shown in a single table representing the parameter combinations used in the model and the prediction accuracies. Since the model predictions are combined in three different approaches in this thesis, those figures are also shown in the same table. These tables are shown for all of the ensemble models that have been tested out in this thesis and finally a conclusion is made based on these analytics so as to say which combination of ensemble model techniques are best suitable for image captioning.

6.1 Analysis of k-fold Cross Validation Ensemble

Different combinations of k-fold cross validation datasets have been chosen to experiment with different model architectures to understand the effects of ensemble learning on these datasets. The tables show the parameters of these combinations of

models used and their individual bleu score accuracies along with the bleu scores of the whole ensemble.

6.1.1 K-fold Ensemble Member Combination-1

For this ensemble, the dataset is divided into 4-folds which requires 4 ensemble models to be trained on the 4 samples of datasets. Different combinations of two hyperparameters have been used in building the ensemble and shown in Table 3. Other hyperparameters namely the number of layers, number of neurons, learning rate and batches remain constant with values 6, 256, 0.001 and 256 respectively.

Model	RNN type	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	Rmsprop	0.558	0.336	0.243	0.123
M-2	GRU	Adam	0.593	0.370	0.280	0.156
M-3	LSTM	Adam	0.607	0.382	0.286	0.156
M-4	GRU	Rmsprop	0.597	0.360	0.258	0.133
MMP	N/A	N/A	0.610	0.386	0.292	0.162
MV	N/A	N/A	0.611	0.388	0.292	0.160
AP	N/A	N/A	0.621	0.397	0.300	0.167

Table 3. Model Types and the Corresponding Bleu Scores for k-fold Cross Validation

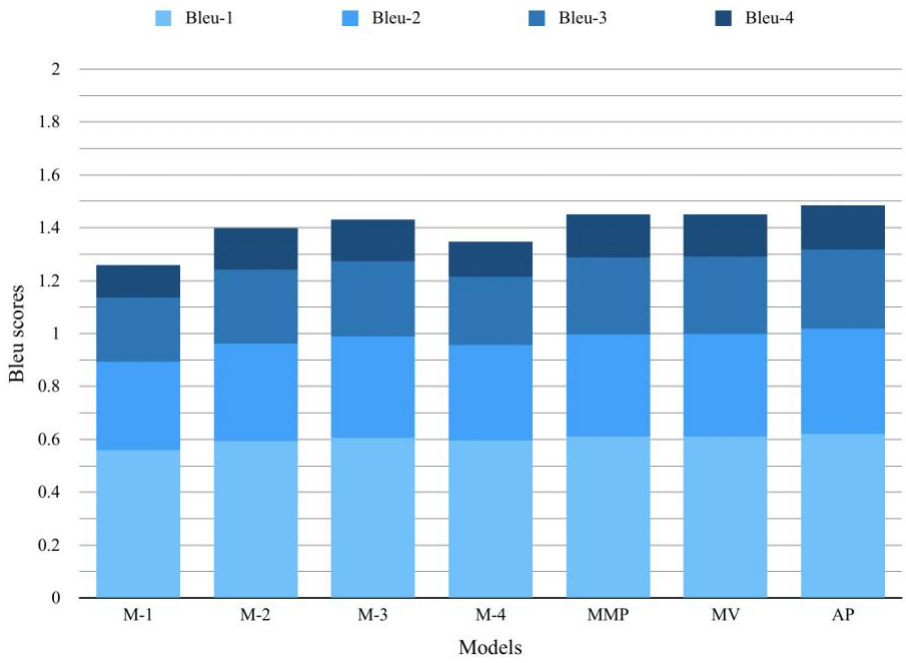

Figure 31. Bleu Score Comparision for k-fold Cross Validation Ensemble-1.

It can be observed from Figure 31 that using ensemble-1 showed a significant increase in prediction accuracies over any of the individual models with all three combination methods. This shows that using combinations of LSTM and GRU model with different activations gives good results on image captioning.

6.1.2 K-fold Ensemble Member Combination-2

For this ensemble, the dataset is divided into 4-folds which requires 4 ensemble models to be trained on the 4 samples of datasets. Different combinations of two hyperparameters have been used in building the ensemble and shown in Table 4. Other hyperparameters namely the number of neurons, activation function, learning rate and batches remain constant with values 256, Adam, 0.001 and 256 respectively.

Model	RNN type	No. of layers	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	6	0.590	0.374	0.280	0.153
M-2	GRU	6	0.600	0.375	0.280	0.150
M-3	SimpleRNN	6	0.603	0.378	0.282	0.157
M-4	LSTM	8	0.584	0.363	0.270	0.145
MMP	N/A	N/A	0.628	0.403	0.302	0.166
MV	N/A	N/A	0.614	0.391	0.290	0.158
AP	N/A	N/A	0.625	0.406	0.306	0.170

Table 4. Model Types and the Corresponding Bleu Scores for k-fold Cross Validation Ensemble-2

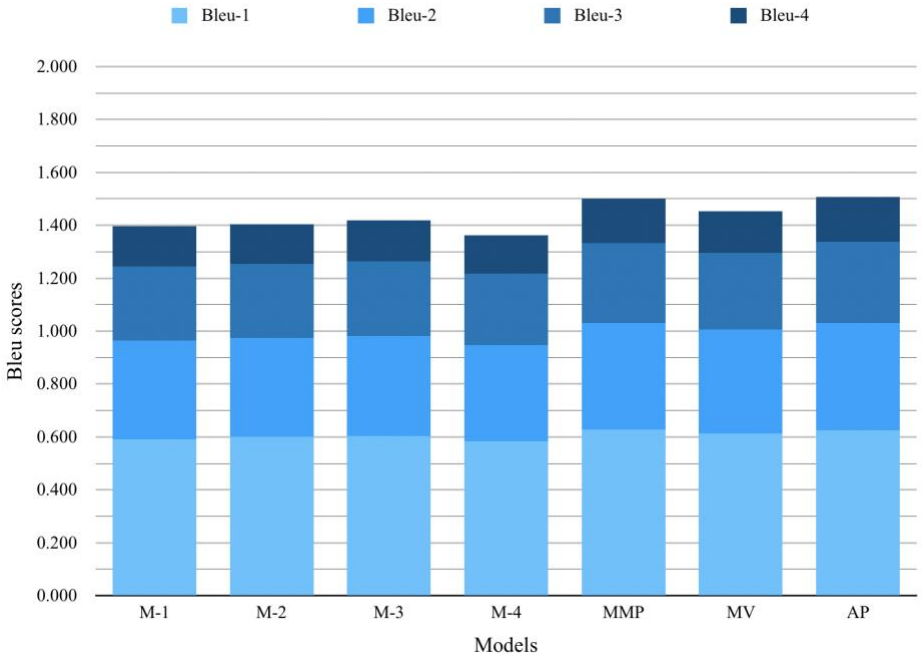

Figure 32. Bleu Score Comparision for k-fold Cross Validation Ensemble-2.

It can be observed from Figure 32 that using ensemble-2 also showed a significant increase in prediction accuracies over any of the individual models with all three

combination methods. This shows that using a combination of different RNN models gives better results as they can capture different trends in the data.

6.1.3 K-fold Ensemble Member Combination-3

For this ensemble, the dataset is divided into 3-folds which requires 3 ensemble models to be trained on the 3 samples of datasets. Different combinations of two hyperparameters have been used in building the ensemble and shown in Table 5. Other hyperparameters namely the RNN type, number of neurons, learning rate and batches remain constant with values LSTM, 256, 0.001 and 256 respectively.

Model	No. of layers	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	8	Adagrad	0.472	0.211	0.098	0.020
M-2	6	Adam	0.613	0.382	0.280	0.150
M-3	6	Adagrad	0.595	0.357	0.257	0.129
MMP	N/A	N/A	0.614	0.375	0.271	0.143
MV	N/A	N/A	0.624	0.388	0.284	0.154
AP	N/A	N/A	0.626	0.388	0.280	0.147

Table 5. Model Types and the Corresponding Bleu Scores for k-fold Cross Validation Ensemble-3

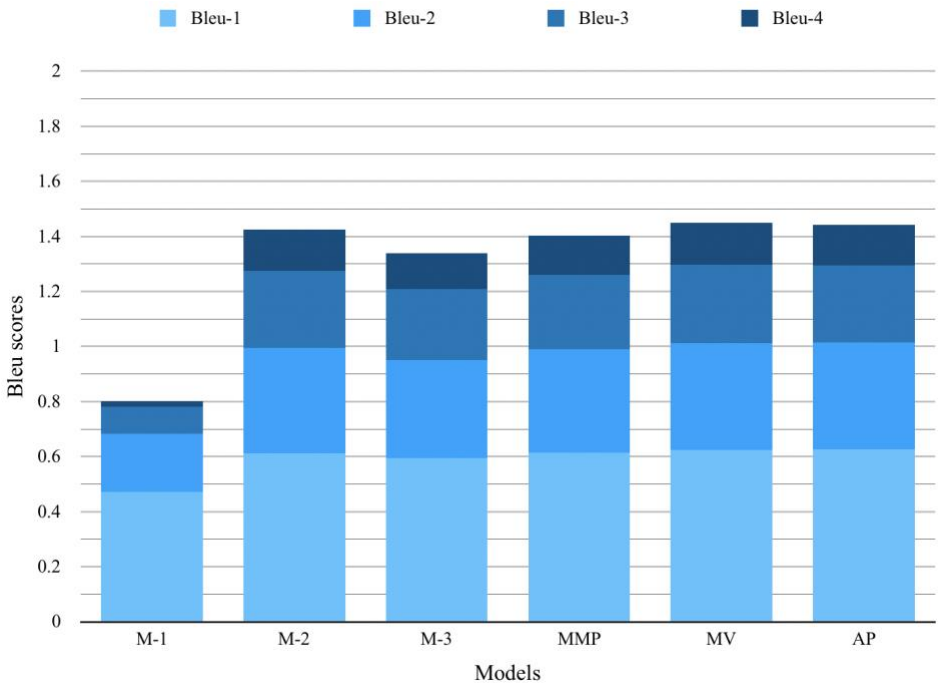

Figure 33. Bleu Score Comparision for k-fold Cross Validation Ensemble-3.

It can be observed from Figure 33 that using ensemble-3 showed very less increase in prediction accuracy with MV and AP methods over the individual models. But, using MMP in this case provided less accurate results in than Model-2. The reason for this could be the use of Adagrad activation function. This states that Adagrad might not be a good function to use for image captioning. Another reason could be the number of ensembles used. Using only 3 ensemble members leads to capturing less trend variations in data.

Performing ensemble learning on k-folds of datasets has shown considerable amount of increase in accuracies when compared to the individual models. It can be understood from the charts provided that the first two ensembles performed better than any of their individual members by an amount of 1.6% in bleu-1 scores on an average. But,

ensemble-3 did not show significant increase in accuracy. The reason for this is the number of k-folds chosen to perform ensemble learning. While the first 2 ensembles have datasets divided into 4-folds, the last ensemble has dataset split into 3-folds. This shows that small number of ensemble members will not be able to capture the different trends in the dataset. Hence, chosen a mediocre number of models to participate as ensemble members would provide better results for the problem as can be seen from the charts provided. Other ensemble member combination tested out have been explained in Appendix A.

6.2 Analysis of Bootstrap Aggregation Ensemble

Bootstrap aggregation technique has been used to create subset samples from the dataset with replacements and used to train different models. The experimental results of applying bootstrap aggregation on image captioning have been presented in the tables below. The bleu score of the individual models and those of the combinations of these models have also be presented.

6.2.1 Bootstrap Aggregation Ensemble Member Combination-1

For this ensemble, the dataset is sampled 8 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 6. Other hyperparameters namely the number of neurons, learning rate and batches remain constant with values 256, 0.001 and 256 respectively.

Model	RNN type	No. of layers	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	6	Adagrad	0.462	0.203	0.060	0.091
M-2	LSTM	6	Adam	0.576	0.346	0.244	0.116
M-3	GRU	6	Adam	0.594	0.360	0.260	0.131
M-4	SimpleRNN	6	Adam	0.589	0.357	0.261	0.137
M-5	LSTM	8	Adam	0.583	0.338	0.240	0.122
M-6	GRU	8	Adam	0.583	0.350	0.256	0.134
M-7	GRU	6	Rmsprop	0.603	0.362	0.259	0.134
M-8	LSTM	6	Rmsprop	0.580	0.342	0.236	0.113
MMP	N/A	N/A	N/A	0.616	0.380	0.277	0.147
MV	N/A	N/A	N/A	0.626	0.383	0.276	0.143
AP	N/A	N/A	N/A	0.626	0.387	0.280	0.147

Table 6. Model Types and the Corresponding Bleu Scores for Bootstrap Aggregation Ensemble-1

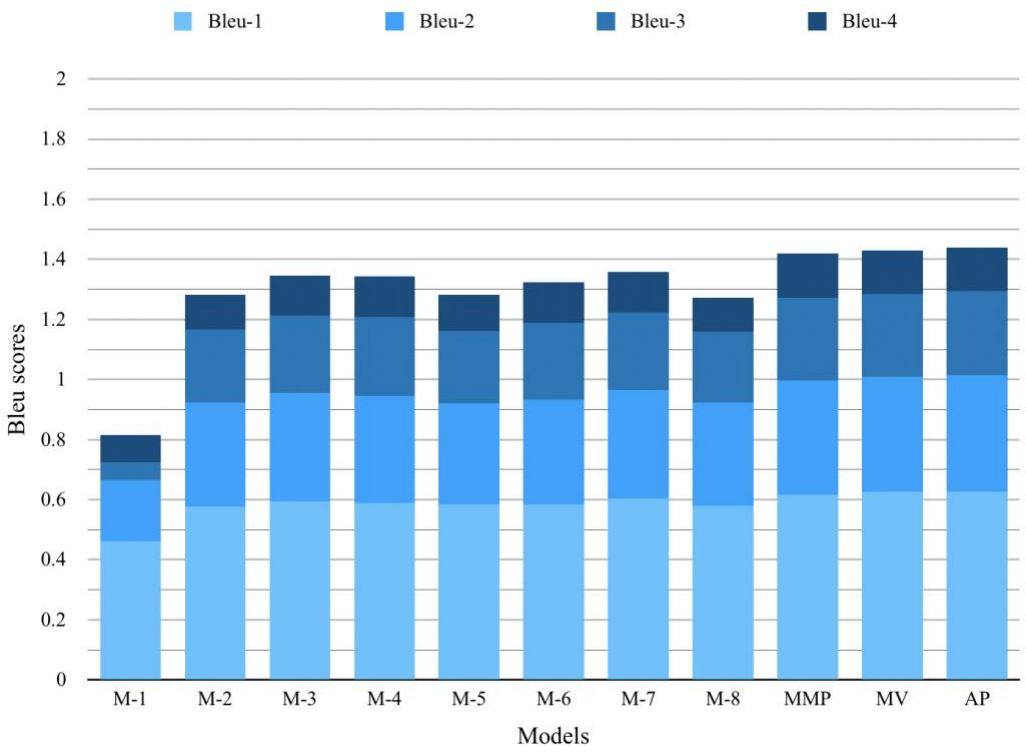

Figure 34. Bleu Score Comparision for Bootstrap Aggregation Ensemble-1.

It can be observed from Figure 34 that using ensemble-1 showed a good increase in prediction accuracies over all of the individual models using all combination methods. From this first ensemble, a lot cannot be inferred but it can definitely be considered that bootstrap aggregation works well with image captioning.

6.2.2 Bootstrap Aggregation Ensemble Member Combination-2

For this ensemble, the dataset is sampled 8 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 7. Other hyperparameters namely the number of neurons, learning rate and batches remain constant with values 256, 0.001 and 256 respectively.

Model	RNN type	No. of layers	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	8	Adam	0.465	0.212	0.073	0.016
M-2	LSTM	6	Adam	0.595	0.337	0.233	0.111
M-3	GRU	6	Adam	0.601	0.368	0.270	0.143
M-4	SimpleRNN	6	Adam	0.600	0.352	0.247	0.121
M-5	LSTM	8	Rmsprop	0.565	0.312	0.215	0.101
M-6	GRU	8	Rmsprop	0.587	0.334	0.232	0.111
M-7	GRU	6	Rmsprop	0.574	0.338	0.240	0.119
M-8	LSTM	6	Rmsprop	0.611	0.380	0.273	0.140
MMP	N/A	N/A	N/A	0.623	0.376	0.266	0.134
MV	N/A	N/A	N/A	0.623	0.373	0.267	0.139
AP	N/A	N/A	N/A	0.623	0.375	0.266	0.134

Table 7. Model Types and the Corresponding Bleu Scores for Bootstrap Aggregation Ensemble-2

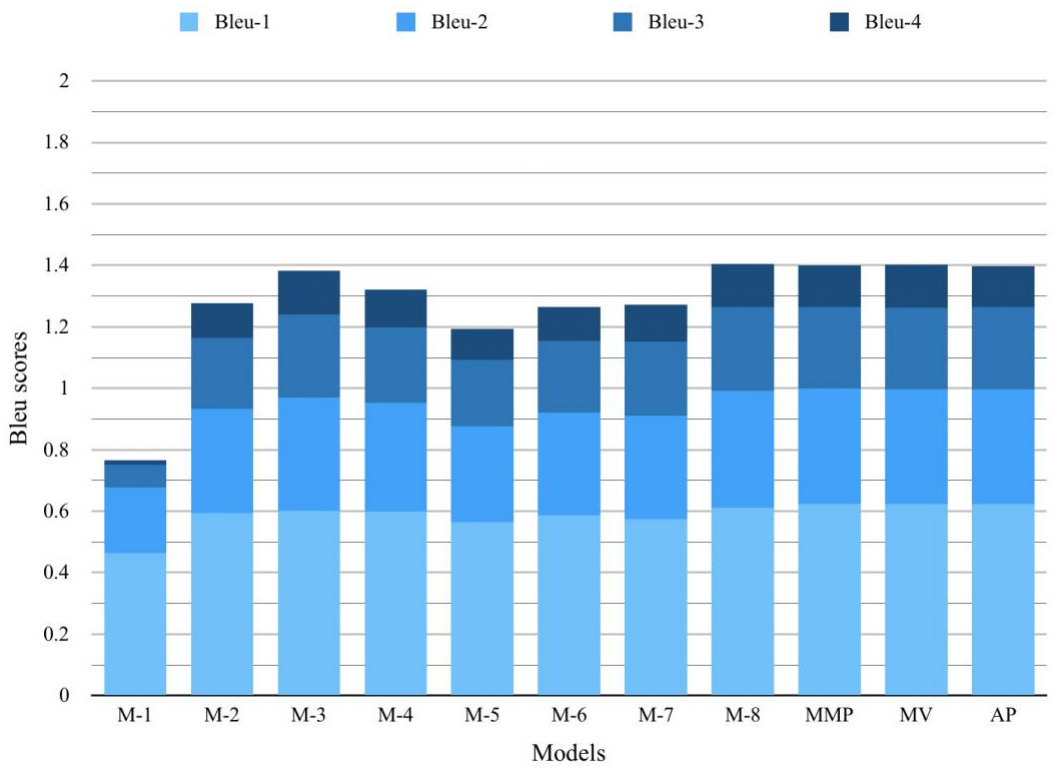

Figure 35. Bleu Score Comparision for Bootstrap Aggregation Ensemble-2.

It can be observed from Figure 35 that using ensemble-2 showed an increase in prediction accuracies over all of the individual models using all combination methods. From the hyperparameters used in this ensemble it can be understood that using greater number of layers is affecting the combination prediction.

6.2.3 Bootstrap Aggregation Ensemble Member Combination-3

For this ensemble, the dataset is sampled 6 times with each sample having a size of 4000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 8. Other hyperparameters namely the number of layers, number of neurons, learning rate and batches remain constant with values 6, 256, 0.001 and 256 respectively.

Model	RNN type	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	Adam	0.616	0.377	0.273	0.144
M-2	GRU	Adam	0.602	0.371	0.274	0.147
M-3	SimpleRNN	Adam	0.607	0.371	0.268	0.139
M-4	LSTM	Rmsprop	0.599	0.354	0.253	0.127
M-5	GRU	Rmsprop	0.596	0.361	0.260	0.135
M-6	SimpleRNN	Rmsprop	0.609	0.356	0.244	0.117
MMP	N/A	N/A	0.631	0.390	0.280	0.146
MV	N/A	N/A	0.620	0.384	0.282	0.151
AP	N/A	N/A	0.626	0.388	0.282	0.148

Table 8. Model Types and the Corresponding Bleu Scores for Bootstrap Aggregation Ensemble-3

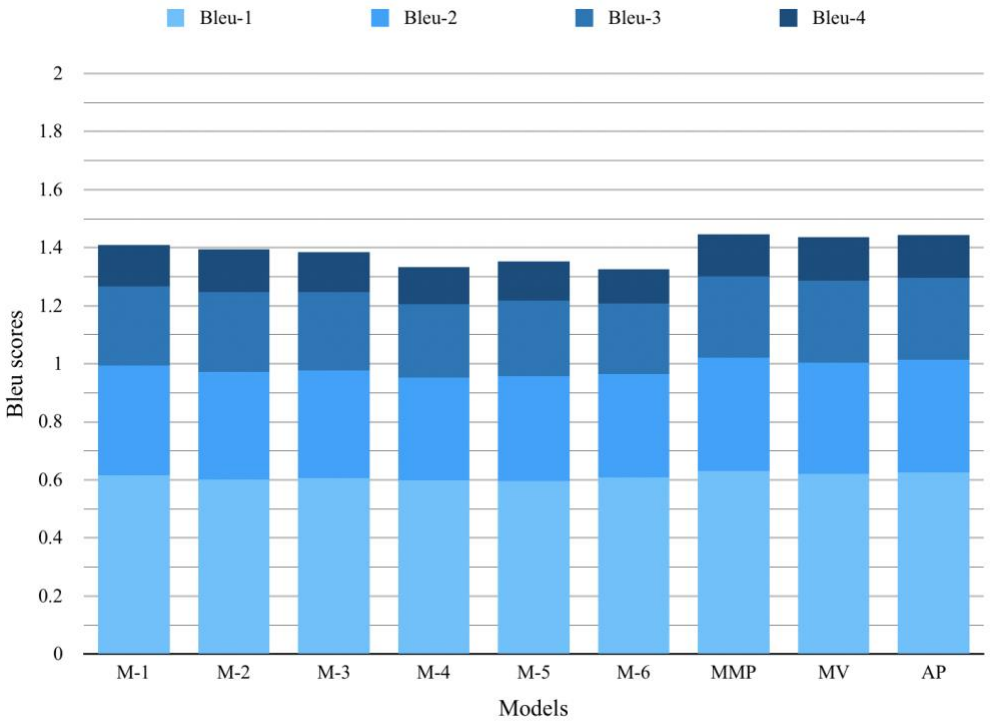

Figure 36. Bleu Score Comparision for Bootstrap Aggregation Ensemble-3.

It can be observed from Figure 36 that using ensemble-3 showed good increase in prediction accuracies over all of the individual models using all combination methods. It can be inferred from the result that using combinations of different RNN models with the adam and rmsprop activations improves the performance of image captioning.

It can be concluded that using bootstrap aggregation ensemble technique will show significant increase in prediction accuracies with using the right combination of models. But, using a greater number of layers would affect the performance of the overall ensemble. Hence, a right combination of RNN models with activations having a smaller number of layers would give an increase in accuracy in the blue-1 score. The three models tested in this section showed an average increase of 1.7% in the bleu-1 score.

6.3 Analysis of Boosting

As explained in section 5.2.4, in boosting weights are assigned to each data point and the weights keep changing as more models are trained on them and error in predictions calculated. A series of ensemble combinations have been tested out with the bleu scores of all of the ensembles presented in this section.

6.3.1 Boosting Ensemble Member Combination-1

For this ensemble, the dataset is sampled 4 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 9. Other hyperparameters namely the RNN type, number of neurons and batches remain constant with values LSTM, 256 and 256 respectively.

Model	No. of layers	Activation function	Learning rate	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	6	Adam	0.001	0.612	0.373	0.275	0.142
M-2	6	Adam	0.0001	0.510	0.314	0.224	0.108
M-3	10	Rmsprop	0.001	0.568	0.298	0.194	0.084
M-4	6	Rmsprop	0.001	0.610	0.368	0.261	0.128
MMP	N/A	N/A	N/A	0.630	0.390	0.280	0.143
MV	N/A	N/A	N/A	0.621	0.378	0.272	0.140
AP	N/A	N/A	N/A	0.625	0.386	0.278	0.143

Table 9. Model Types and the Corresponding Bleu Scores for Boosting Ensemble-1.

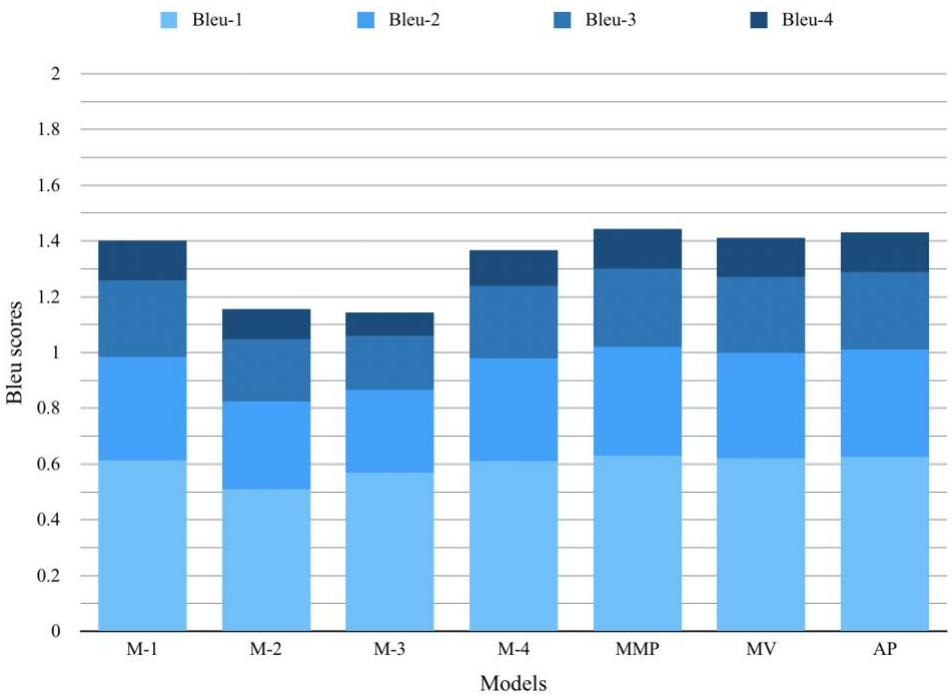

Figure 37. Bleu Score Comparision for Boosting Ensemble-1.

It can be observed from Figure 37 that using ensemble-1 showed an increase in prediction accuracies over all of the individual models using all combination methods. A lot cannot be inferred from this ensemble as it few ensemble members with random variations in hyperparameters.

6.3.2 Boosting Ensemble Member Combination-2

For this ensemble, the dataset is sampled 6 times with each sample having a size of 4000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 10. Other hyperparameters namely the number of neurons and batches remain constant with values 256 and 256 respectively.

Model	RNN type	No. of layers	Activation function	Learning rate	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	Simple RNN	6	Adam	0.001	0.593	0.358	0.260	0.135
M-2	LSTM	6	Adam	0.001	0.611	0.363	0.255	0.130
M-3	GRU	6	Adam	0.001	0.595	0.365	0.270	0.143
M-4	LSTM	6	Adam	0.0001	0.455	0.201	0.064	0.012
M-5	LSTM	10	Rmsprop	0.001	0.576	0.270	0.184	0.072
M-6	LSTM	6	Rmsprop	0.001	0.594	0.354	0.255	0.130
MMP	N/A	N/A	N/A	N/A	0.628	0.380	0.271	0.140
MV	N/A	N/A	N/A	N/A	0.618	0.380	0.275	0.146
AP	N/A	N/A	N/A	N/A	0.593	0.358	0.260	0.135

Table 10. Model Types and the Corresponding Bleu Scores for Boosting Ensemble-2.

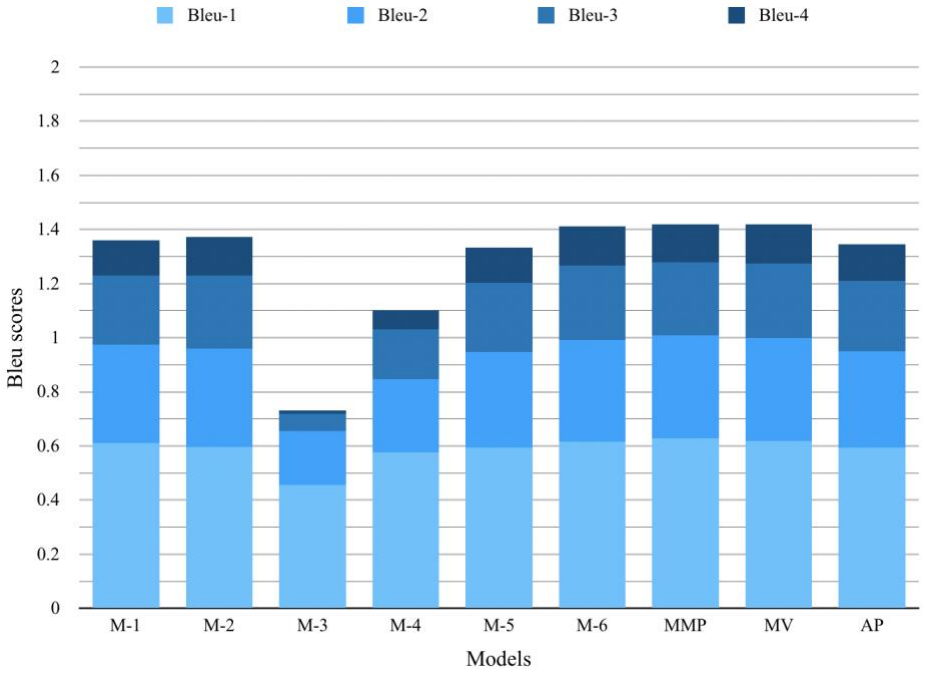

Figure 38. Bleu Score Comparision for Boosting Ensemble-2.

It can be observed from Figure 38 that using ensemble-2 showed an increase in prediction accuracies over all of the individual models using all combination methods. This shows that using different combinations of activations and learning rates with the same RNN type could result in the models capturing different trends in data.

6.3.3 Boosting Ensemble Member Combination-3

For this ensemble, the dataset is sampled 7 times with each sample having a size of 4000 data points. Different combinations of four hyperparameters have been used in building the ensemble and shown in Table 11. Other hyperparameters namely the number of layers, number of neurons, learning rate and batches remain constant with values 6, 256, 0.001 and 256 respectively.

Model	RNN type	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	SimpleRNN	Adam	0.607	0.380	0.278	0.148
M-2	LSTM	Adam	0.591	0.366	0.270	0.140
M-3	GRU	Adam	0.599	0.367	0.266	0.140
M-4	GRU	Adam	0.511	0.213	0.122	0.044
M-5	LSTM	Adam	0.532	0.234	0.134	0.043
M-6	LSTM	Rmsprop	0.561	0.335	0.240	0.120
M-7	LSTM	Adam	0.542	0.244	0.144	0.057
MMP	N/A	N/A	0.618	0.374	0.273	0.142
MV	N/A	N/A	0.629	0.391	0.284	0.151
AP	N/A	N/A	0.634	0.400	0.287	0.151

Table 11. Model Types and the Corresponding Bleu Scores for Boosting Ensemble-3.

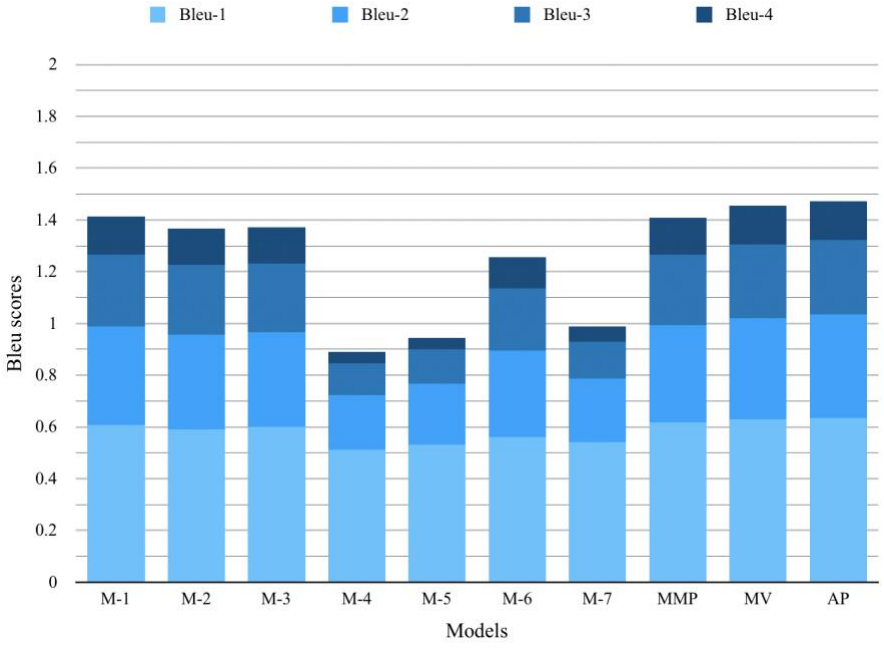

Figure 39. Bleu Score Comparision for Boosting Ensemble-3.

It can be observed from Figure 39 that using ensemble-3 showed an increase in prediction accuracies over all of the individual models using all combination methods. This shows that using different combinations of activations with the different RNN types would also result in higher prediction accuracies.

The boosting ensemble technique has shown the best results among the three techniques that have been analyzed out in this thesis. All combinations methods have shown better performance than the individual models by 2.1% increase in bleu-1 scores on an average. But, one thing can be inferred from the models used for boosting. Using a constant learning rate of 0.001 for all ensemble members has shown better results with a 100% guarantee.

6.4 Sample Captions Generated for Images in Test Dataset

Example - 1

Figure 40. Sample Image in Test Dataset-1

Best Model – “two young boys are playing on the grass.”

k-fold Ensemble – “boy in blue shirt is running in the grass.”

Bootstrap Aggregation Ensemble – “boy in red shirt is running on the grass.”

Boosting Ensemble – “boy in blue shirt is running on the grass.”

Example – 2

Figure 41. Sample Image in Test Dataset-2

Best Model – “skier is walking through the snow.”

k-fold Ensemble – “skier is skiing down snowy hill.”

Bootstrap Aggregation Ensemble – “skier is in the snow.”

Boosting Ensemble – “man in red jacket is skiing down snowy hill.”

Example – 3

Figure 42. Sample Image in Test Dataset-3

Best Model – “man in red shirt is jumping down ramp.”

k-fold Ensemble – “skateboarder is jumping off of the railing.”

Bootstrap Aggregation Ensemble – “man is jumping down the wall.”

Boosting Ensemble – “skateboarder is jumping off ramp.”

Example – 4

Figure 43. Sample Image in Test Dataset-4

Best Model – “man in yellow helmet is riding bike on the road.”

k-fold Ensemble – “motorcycle racer is riding motorcycle.”

Bootstrap Aggregation Ensemble – “man in red helmet is riding the bike.”

Boosting Ensemble – “man in red and white helmet is riding bike on the track.”

Example – 5

Figure 44. Sample Image in Test Dataset-5

Best Model – “man with sunglasses and sunglasses.”

k-fold Ensemble – “man in black shirt and black hat is standing in front of an old building.”

Bootstrap Aggregation Ensemble – “man in black shirt and black shirt is standing on the street.”

Boosting Ensemble – “man in black shirt is standing on the street.”

6.5 Combined Analysis

Model	Bleu-1	Bleu-2	Bleu-3	Bleu-4
Best Model	0.616	0.377	0.273	0.144
k-fold	0.628	0.403	0.302	0.166
Bootstrap Aggregation	0.631	0.390	0.280	0.146
Boosting	0.634	0.400	0.287	0.151

Table 12. Comparison of all Ensemble Methods with the Best Individual Model.

It can be observed from Table 12 that all three ensemble techniques have clearly shown improvements in prediction accuracies over the best individual model. Among these ensembles, boosting has shown the highest increase. The captions generated for images in the training dataset by the ensemble models also clearly show the generation of better captions.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

The idea of using ensemble learning techniques on the image captioning problem has been proven to be advantageous over using a single model. With the results presented in the analysis section, it can be clearly understood that forming an ensemble with the right combination of models would show significant improvements in performance. As the traditional technique followed to solve image captioning has been established, there is a lot of research going on in making improvements to this existing technique. But, using ensemble learning is an already existing and efficient technique that is demonstrated to work well with deep learning models. Hence, this justifies the results obtained with the use of ensemble learning on image captioning.

All the three ensemble learning techniques used in this thesis promised better predictions in almost every case. On an average all the techniques showed an increase in overall accuracy of 2.2% over the best individual model in the ensemble. An increase in accuracies by 2.2% is considered very crucial for deep learning models. Though training many models is computationally expensive when compared to training just one model, the use of a sampled dataset for training each model reduces this expense by a huge amount. Hence, the compute cost would be very close to that of training a single model on the entire dataset. Also, using sampled datasets is a crucial part of ensemble learning which would result in faster training of the models. Choosing the right number of ensemble members to balance the computational cost with increase in accuracies is critical in this case. From the analysis done in this thesis, using 4-5 ensemble members

and training each of the members with 25% of the entire dataset would be sufficient to attain significant increase in prediction accuracies.

Ensemble learning thus helped in solving the problem of high variance occurring in deep neural networks. Different models used as ensemble members learned different patterns from the training dataset. Though each of them individually could not generalize well on the data, when combined together had the advantage of knowing diverse patterns in data learned by each of them. Another advantage here is using a sampled dataset. All models could observe only subsets of the training data and so each of them would generate varied predictions. These diverse predictions when combined in a standard way generated the desired results, thus decreasing the problem of high variance.

For the future, different variations to the traditional model for solving image captioning can be used as model members for the ensemble. This way a more differing set of patterns can be learned by different models which when combined would result in giving even better predictions. Another improvement could be using a larger dataset and training the ensemble models with sampled datasets of the training set. This would help the different models to learn and capture different patterns from subsets and together make stronger predictions. Also, image captioning is a sequence to sequence modeling problem, and it has been proved that ensemble learning works well on this problem. This gives scope for trying out ensemble learning on other sequence to sequence problems like speech recognition, language translation, video captioning, etc.

REFERENCES

[1] Ting Yao, Yingwei Pan, Yehao Li, and Tao Mei. “Incorporating Copying Mechanism in Image Captioning for Learning Novel Objects,” IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[2] Xuedan Du, Yinghao Cai, Shuo Wang, and Leijie Zhang. “Overview of Deep Learning”. 31st Youth Academic Annual Conference of Chinese Association of Automation, 2016.

[3] Senmao Ye, Junwei Han, and Nian Liu. “Attentive Linear Transformation for Image Captioning,” IEEE Transactions on Image Processing, 2018.

[4] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a Convolutional Neural Network,” International Conference on Engineering and Technology (ICET), 2017.

[5] [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

[6] Aya Abdelsalam Ismail, Timothy Wood, and Hector Corrada Bravo. “Improving Long-Horizon Forecasts with Expectation-Based LSTM,” 2018.

[7] K. Simonyan and A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition,” ICLR, 2015.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens. “Rethinking the Inception Architecture for Computer Vision,” 2015.

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. “ You Only Look Once: Unified, Real-Time Object Detection,”, 2016.

[10] Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. “ Boosting Image Captioning with Attributes,” IEEE International Conference on Computer Vision, 2017.

[11] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Chourville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. “ Show, Attend and Tell: Neural Image Caption Generation with Visual Attention,” 2016.

[12] [Online]. Available: https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff.

[13] Junyi Xu, Li Yao, and Le Li. “Argumentation Based Joint Learning: A Novel Ensemble Learning Approach,” 2015.

[14] [Online]. Available: https://machinelearningmastery.com/ensemble-methods-for deep-learning-neural-networks/

[15] [Online]. Available: https://www.datascience.com/resources/notebooks/word-embeddings-in-python.

[16] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "Bleu: A Method for Automatic Evaluation of Machine Translation," Proceedings for the 40th Annual Meeting of the Association for Computational Linguistics (ACL), 2002.

[17] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. "Show and Tell: A Neural Image Caption Generator," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[18] Kenneth Tran, Xiaodong He, Lei Zhang, Jian Sun, Cornelia Carapcea, Chris Thrasher, Chris Buehler, Chris Sienkiewicz. "Rich Image Captioning in the Wild," IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016.

[19] Urminder Singh, Sucheta Chauhan, A. Krishnamachari, Lovekesh Vig. "Ensemble of Deep Long Short Term Memory Networks for Labelling Origin of Replication Sequences," IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2015.

[20] Inwoong Lee, Doyoung Kim, Seoungyoon Kang, Sanghoon Lee. "Ensemble Deep Learning for Skeleton-based Action Recognition using Temporal Sliding LSTM Networks," IEEE International Conference on Computer Vision (ICCV), 2017.

APPENDIX A

ADDITIONAL ENSEMBLE MEMBER COMBINATIONS

I. K-fold Ensemble Member Combination-4

For this ensemble, the dataset is divided into 3-folds which requires 3 ensemble models to be trained on the 3 samples of datasets. Different combinations of RNN types have been used in building the ensemble and shown in Table 13. Other hyperparameters namely the number of layers, number of neurons, activation function, learning rate and batches remain constant with values 8, 256, Rmsprop, 0.001 and 256 respectively.

Model	RNN type	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	0.571	0.319	0.212	0.098
M-2	GRU	0.600	0.337	0.227	0.105
M-3	SimpleRNN	0.586	0.338	0.233	0.111
MMP	N/A	0.598	0.339	0.228	0.108
MV	N/A	0.599	0.345	0.237	0.113
AP	N/A	0.600	0.343	0.234	0.109

Table 13. Model Types and the Corresponding Bleu Scores for k-fold Cross Validation Ensemble-4

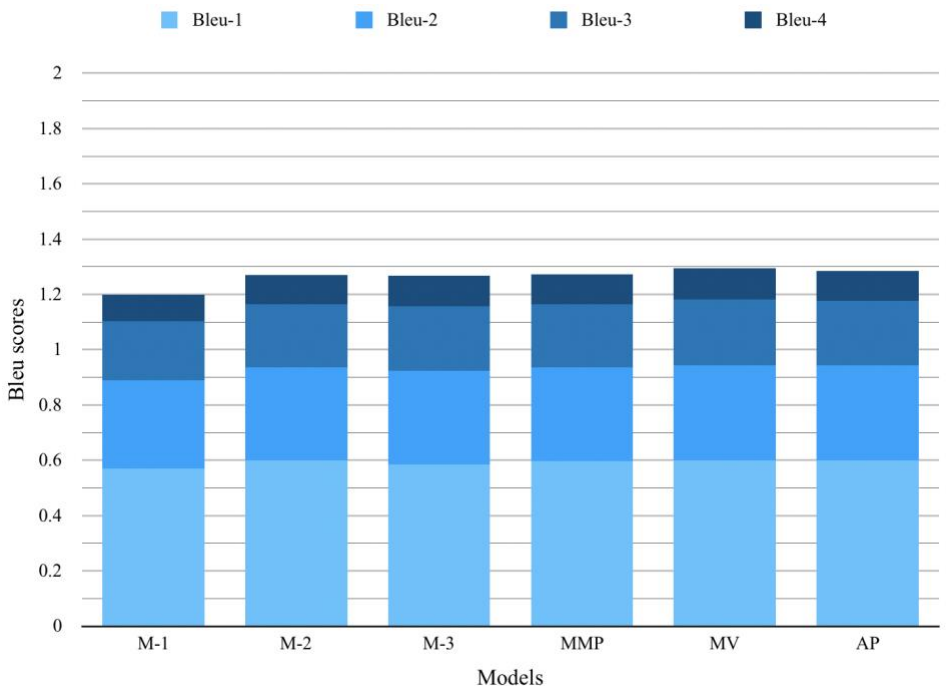

Figure 45. Bleu Score Comparision for k-fold Cross Validation Ensemble-4.

It can be observed from Figure 45 that using ensemble-4 showed an overall increase in prediction accuracies over any of the individual models with all three combination methods. But this increase is not very significant and the reason for this could be that only a different RNN model is used with all other hyperparameter remaining constant. The other reason is the same as for ensemble-3 i.e. the number of ensemble members being very small.

II. K-fold Ensemble Member Combination-5

For this ensemble, the dataset is divided into 3-folds which requires 3 ensemble models to be trained on the 3 samples of datasets. Different

combinations of the RNN types have been used in building the ensemble and shown in Table 14. Other hyperparameters namely the number of layers, number of neurons, activation function, learning rate and batches remain constant with values 8, 256, Adam, 0.001 and 256 respectively.

Model	RNN type	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	0.598	0.341	0.230	0.108
M-2	GRU	0.566	0.326	0.233	0.117
M-3	SimpleRNN	0.593	0.359	0.264	0.142
MMP	N/A	0.614	0.368	0.267	0.143
MV	N/A	0.605	0.350	0.243	0.120
AP	N/A	0.608	0.364	0.260	0.135

Table 14. Model Types and the Corresponding Bleu Scores for k-fold Cross Validation Ensemble-5

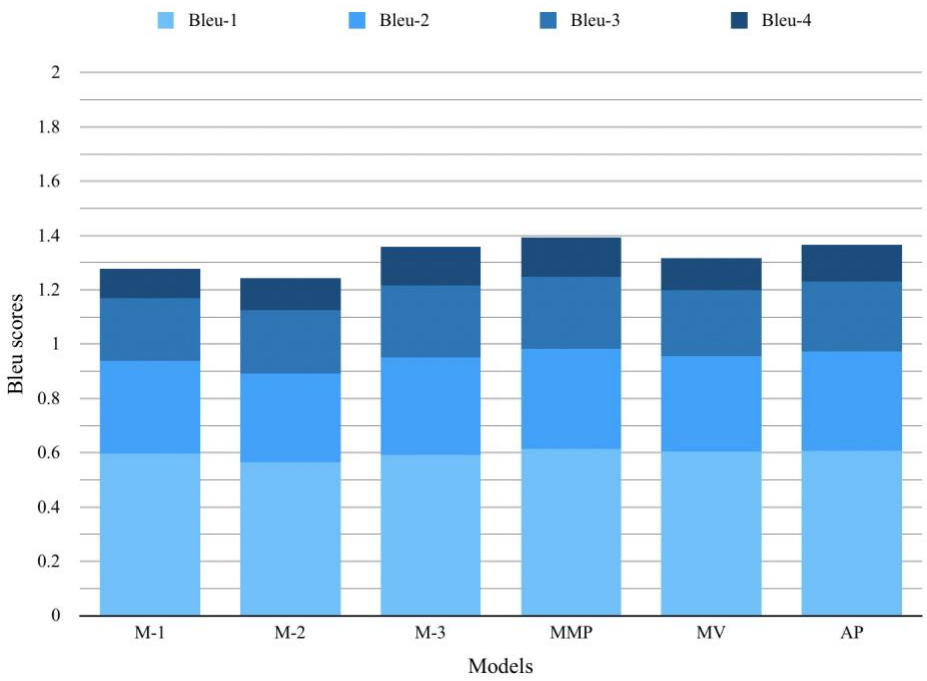

Figure 46. Bleu Score Comparision for k-fold Cross Validation Ensemble-5.

It can be observed from Figure 46 that using ensemble-5 showed an increase in prediction accuracies using only MMP and AP over any of the individual models. But this increase is not very significant and the reasons for this could be the same as the reasons mentioned for ensemble-4 i.e. varying only the RNN models and using less ensemble members.

III. Bootstrap Aggregation Ensemble Member Combination-4

For this ensemble, the dataset is sampled 5 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 15. Other hyperparameters namely the RNN type, number of

neurons, learning rate and batches remain constant with values LSTM, 256, 0.001 and 256 respectively.

Model	No. of layers	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	6	Adagrad	0.474	0.205	0.060	0.092
M-2	8	Adagrad	0.463	0.204	0.085	0.020
M-3	6	Adam	0.580	0.351	0.257	0.133
M-4	6	Rmsprop	0.600	0.364	0.263	0.136
M-5	8	Adam	0.581	0.344	0.247	0.124
MMP	N/A	N/A	0.597	0.356	0.254	0.130
MV	N/A	N/A	0.607	0.369	0.270	0.139
AP	N/A	N/A	0.605	0.371	0.268	0.137

Table 15. Model Types and the Corresponding Bleu Scores for Bootstrap Aggregation Ensemble-4

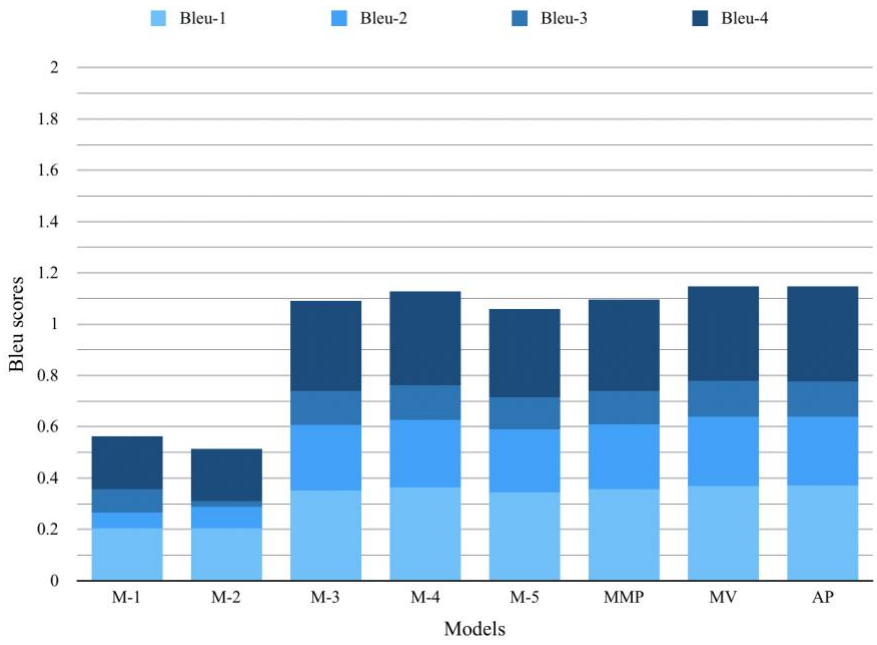

Figure 47. Bleu Score Comparision for Bootstrap Aggregation Ensemble-4.

It can be observed from Figure 47 that using ensemble-4 showed an increase in prediction accuracies with MV and AP over the individual models. But, using MMP gave bad result when compared to Model 4. Since Adagrad is used for activations in this ensemble as well, the prediction accuracies have gone down for the same reasons as mentioned in k-fold ensemble.

IV. Bootstrap Aggregation Ensemble Member Combination-5

For this ensemble, the dataset is sampled 6 times with each sample having a size of 4000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 16. Other hyperparameters namely the number of layers, number

of neurons, learning rate and batches remain constant with values 8, 256, 0.001 and 256 respectively.

Model	RNN type	Activation function	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	LSTM	Adam	0.577	0.331	0.235	0.117
M-2	GRU	Adam	0.588	0.324	0.215	0.095
M-3	SimpleRNN	Adam	0.560	0.334	0.245	0.130
M-4	LSTM	Rmsprop	0.598	0.343	0.236	0.112
M-5	GRU	Rmsprop	0.575	0.327	0.228	0.105
M-6	SimpleRNN	Rmsprop	0.585	0.329	0.227	0.110
MMP	N/A	N/A	0.583	0.339	0.243	0.123
MV	N/A	N/A	0.603	0.351	0.246	0.121
AP	N/A	N/A	0.603	0.356	0.253	0.127

Table 16. Model Types and the Corresponding Bleu Scores for Bootstrap Aggregation Ensemble-5

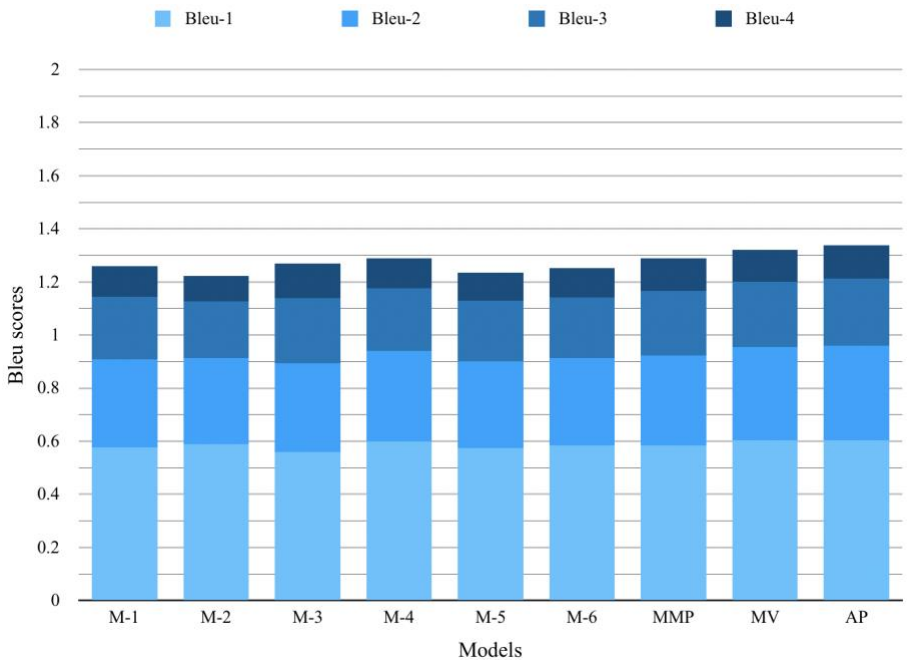

Figure 48. Bleu Score Comparision for Bootstrap Aggregation Ensemble-5.

It can be observed from Figure 48 that using ensemble-5 showed an increase in prediction accuracies over all of the individual models using all combination methods. It can be inferred from the result that using combinations of different RNN models with the adam and rmsprop activations improves the performance of image captioning. But using a greater number of layers effected the performance of bootstrap aggregation as opposed to using a smaller number of layers.

V. Boosting Ensemble Member Combination-4

For this ensemble, the dataset is sampled 7 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in

Table 17. Other hyperparameters namely the RNN type, number of neurons and batches remain constant with values LSTM, 256 and 256 respectively.

Model	No. of layers	Activation function	Learning rate	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	6	Adagrad	0.001	0.450	0.207	0.104	0.028
M-2	6	Rmsprop	0.001	0.603	0.356	0.250	0.120
M-3	8	Rmsprop	0.0001	0.488	0.233	0.123	0.034
M-4	6	Adam	0.0001	0.500	0.313	0.227	0.111
M-5	10	Rmsprop	0.001	0.564	0.315	0.219	0.101
M-6	8	Adam	0.001	0.582	0.352	0.256	0.132
M-7	8	Adagrad	0.001	0.458	0.206	0.096	0.022
MMP	N/A	N/A	N/A	0.624	0.373	0.260	0.130
MV	N/A	N/A	N/A	0.583	0.357	0.258	0.134
AP	N/A	N/A	N/A	0.560	0.340	0.243	0.121

Table 17. Model Types and the Corresponding Bleu Scores for Boosting Ensemble-4.

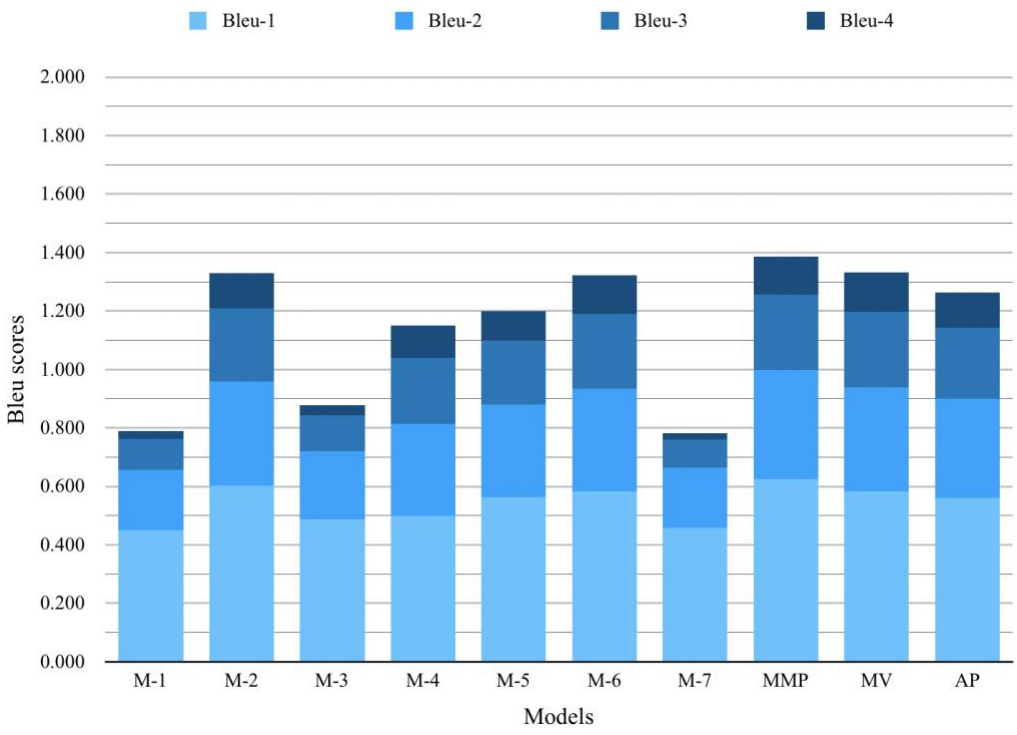

Fig 49. Bleu Score Comparision for Boosting Ensemble-4.

It can be observed from Figure 49 that using ensemble-4 showed an increase in prediction accuracies with MMP and MV over the individual models. But there is a significant decrease in prediction accuracy with AP when compared to models 2 and 6. The reason for this is again the use of adagrad activation.

VI. Boosting Ensemble Member Combination-5

For this ensemble, the dataset is sampled 8 times with each sample having a size of 3000 data points. Different combinations of three hyperparameters have been used in building the ensemble and shown in Table 18. Other hyperparameters namely the RNN type, number of

neurons and batches remain constant with values LSTM, 256 and 256 respectively.

Model	No. of layers	Activat-ion function	Learn-ing rate	Bleu-1	Bleu-2	Bleu-3	Bleu-4
M-1	6	Adagrad	0.001	0.460	0.213	0.100	0.026
M-2	6	Adam	0.001	0.585	0.355	0.255	0.130
M-3	5	Rmspro p	0.0001	0.433	0.191	0.093	0.137
M-4	6	Adam	0.001	0.573	0.364	0.266	0.137
M-5	10	Rmspro p	0.001	0.593	0.318	0.201	0.083
M-6	6	Rmspro p	0.001	0.581	0.350	0.253	0.131
M-7	5	Adagrad	0.001	0.461	0.208	0.083	0.017
M-8	6	Rmspro p	0.001	0.596	0.351	0.250	0.123
MMP	N/A	N/A	N/A	0.617	0.366	0.255	0.126
MV	N/A	N/A	N/A	0.613	0.376	0.273	0.140
AP	N/A	N/A	N/A	0.600	0.368	0.262	0.133

Table 18. Model Types and the Corresponding Bleu Scores for Boosting Ensemble-5.

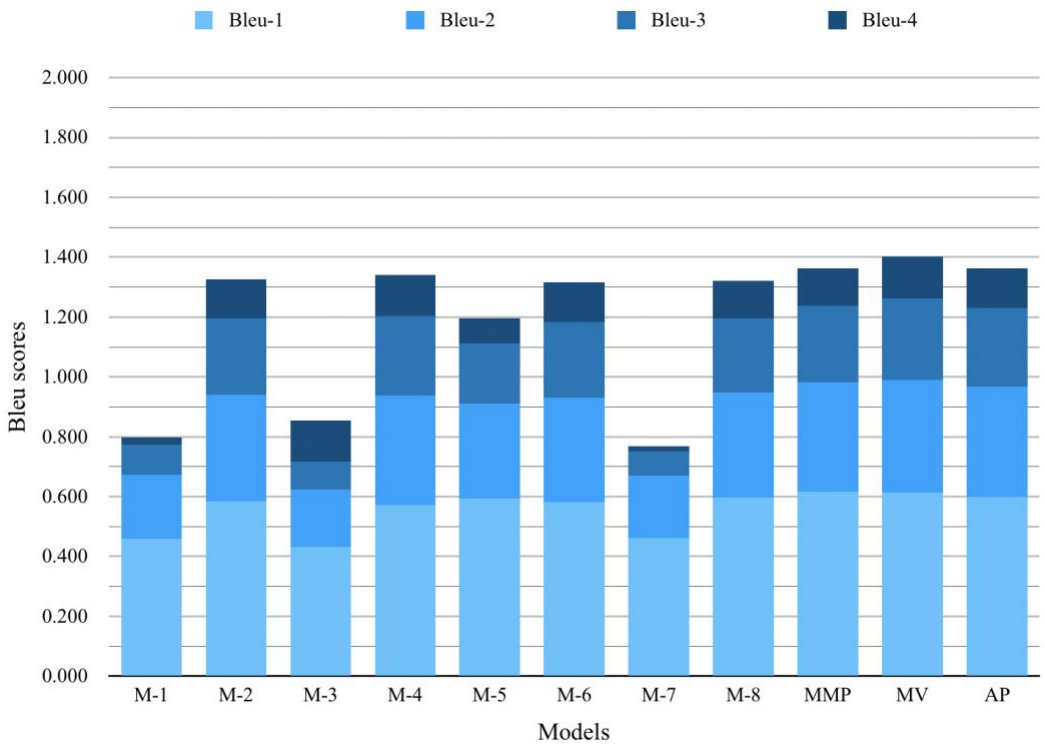

Fig 50. Bleu Score Comparision for Boosting Ensemble-5.

It can be observed from Figure 50 that using ensemble-5 showed an increase in prediction accuracies over all of the individual models using all combination methods. This shows that using different combinations of activations with the same RNN type could also result in the models capturing different trends in data.

APPENDIX B

APPLICATION SOURCE CODE

I. Preparing Text (Python source code)

```
def clean_descriptions(descriptions):
    translationTable = string.maketrans('', '')
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            desc = desc.split()
            desc = [word.lower() for word in desc]
            desc = [w.translate(translationTable) for w in desc]
            desc = [word for word in desc if len(word)>1]
            desc = [word for word in desc if word.isalpha()]
            desc_list[i] =  ' '.join(desc)
    return desc_list
```

II. Feature Extraction (Python source code)

```
# Extract features vectors from a pre-trained model
def extract_features(directory):
    model = ResNet50()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    features = dict()
    for image_name in listdir(directory):
```

```
        filename = directory + '/' + image_name
        image = load_img(filename, target_size=(224, 224))
        image = img_to_array(image)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = preprocess_input(image)
        feature_vector = model.predict(image, verbose=0)
        image_id = image_name.split('.')[0]
        features[image_id] = feature_vector
    return features

# Extract feature vectors for every image and dump into a pickle file
directory = 'Flicker8k_Dataset'
extracted_features = extract_features(directory)
dump(extracted_features, open('resnetfeatures.pkl', 'wb'))
```

III. K-fold Ensemble (Python source code)

```
# load a file into memory
def load_file(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
```

```
# load a pre-defined list of photo identifiers of training set
def load_trainset(filename):
    doc = load_file(filename)
    identifier_list = list()
    for l in doc.split('\n'):
        if len(l) < 1:
            continue
        identifier = l.split('.')[0]
        identifier_list.append(identifier)
    return (identifier_list)

# load all descriptions into memory
def load_descriptions(filename, identifiers):
    doc = load_file(filename)
    descriptions = dict()
    for l in doc.split('\n'):
        tokens = l.split()
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in identifiers:
            if image_id not in descriptions:
                descriptions[image_id] = list()
            desc = 'startword ' + ' '.join(image_desc) + ' endword'
            descriptions[image_id].append(desc)
```

```
    return descriptions

# load features of images
def load_image_features(filename, identifiers):
    features = load(open(filename, 'rb'))
    all_features = {k: features[k] for k in identifiers}
    return all_features

# covert a dictionary of descriptions to list of descriptions
def convert_to_lines(descriptions):
    desc_list = list()
    for key in descriptions.keys():
        [desc_list.append(d) for d in descriptions[key]]
    return desc_list

# fit a tokenizer for the caption descriptions
def create_tokenizer(descriptions):
    lines = convert_to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# create input output sequences for training
```

```
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            seq = tokenizer.texts_to_sequences([desc])[0]
            for i in range(1, len(seq)):
                in_seq, out_seq = seq[:i], seq[i]
                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                X1.append(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)


# define the captioning model
def define_model0(vocab_size, max_length):
    adam = optimizers.Adam(lr = 0.001, decay = 0.0, clipnorm = 1.)
    inputs1 = Input(shape=(2048,))
    l1 = Dropout(0.5)(inputs1)
    l2 = Dense(256, activation='relu')(l1)
    l3 = Dense(256, activation='relu')(l2)
    inputs2 = Input(shape=(max_length,))
    l4 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
```

```
    l5 = Dropout(0.5)(l4)
    l6 = SimpleRNN(256)(l5)
    #se4 = Activation('softmax')(se3)
    decoder1 = add([l3 , l6])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer = adam, metrics =
['accuracy'])
    print(model.summary())
    return model

# define the captioning model
def define_model1(vocab_size, max_length):
    adam = optimizers.Adam(lr = 0.001, decay = 0.0, clipnorm = 1.)
    inputs1 = Input(shape=(2048,))
    l1 = Dropout(0.5)(inputs1)
    l2 = Dense(256, activation='relu')(l1)
    l3 = Dense(256, activation='relu')(l2)
    inputs2 = Input(shape=(max_length,))
    l4 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    l5 = Dropout(0.5)(l4)
    l6 = LSTM(256)(l5)
```

```
    decoder1 = add([l3 , l6])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer = adam, metrics =
['accuracy'])
    print(model.summary())
    #plot_model(model, to_file='model.png', show_shapes=True)
    return model

# define the captioning model
def define_model2(vocab_size, max_length):
    adam = optimizers.Adam(lr = 0.001, decay = 0.0, clipnorm = 1.)
    inputs1 = Input(shape=(2048,))
    l1 = Dropout(0.5)(inputs1)
    l2 = Dense(256, activation='relu')(l1)
    l3 = Dense(256, activation='relu')(l2)
    inputs2 = Input(shape=(max_length,))
    l4 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    l5 = Dropout(0.5)(l4)
    l6 = GRU(256)(l5)
    decoder1 = add([l3 , l6])
    decoder2 = Dense(256, activation='relu')(decoder1)
```

```
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer = adam, metrics =
['accuracy'])
    print(model.summary())
    #plot_model(model, to_file='model.png', show_shapes=True)
    return model

# train dataset
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_trainset(filename)
train = np.array(train)
train_descriptions = load_descriptions('descriptions.txt', train)
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = 34
print('Description Length: %d' % max_length)

kfolds = KFold(3, True)
count = 0
for train_x, test_x in kfolds.split(train):
    train_x, test_x = train[train_x], train[test_x]
```

```
    train_descriptions = load_descriptions('descriptions.txt', train_x)
    train_features = load_image_features('featuresresnet.pkl', train_x)
    X1train, X2train, ytrain = create_sequences(tokenizer, max_length,
train_descriptions, train_features)

    test_descriptions = load_descriptions('descriptions.txt', test_x)
    test_features = load_image_features('featuresresnet.pkl', test_x)
    X1test, X2test, ytest = create_sequences(tokenizer, max_length,
test_descriptions, test_features)

    if(count == 0):
        model = define_model0(vocab_size, max_length)
        filepath = 'model-kfold-' + str(count) + '.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20,
verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest),
shuffle=True)

    if(count == 1):
        model = define_model1(vocab_size, max_length)
        filepath = 'model-kfold-' + str(count) + '.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
```

```
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20,
    verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest),
    shuffle=True)

    if(count == 2):
        model = define_model2(vocab_size, max_length)
        filepath = 'model-kfold-' + str(count) + '.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20,
    verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest),
    shuffle=True)

    count = count + 1
```

IV. Bootstrap Aggregation Ensemble (Python source code)

```
    def random_sampling(identifiers):
        identifiers = array(list(identifiers))
        p = np.random.choice(identifiers, 4000, replace = True)
        return p

    # train dataset
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_trainset(filename)
```

```
train = np.array(train)
train_descriptions = load_descriptions('descriptions.txt', train)
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = 34
print('Description Length: %d' % max_length)

#load validation dataset
filename = 'Flickr8k_text/Flickr_8k.devImages.txt'
val = load_testset(filename)
val_descriptions = load_descriptions('descriptions.txt', val)
val_features = load_image_features('featuresresnet.pkl', val)
X1val, X2val, yval = create_sequences(tokenizer, max_length, val_descriptions,
val_features)

for i in range(3):
    train_sampled = random_sampling(train)
    train_descriptions = load_descriptions('descriptions.txt', train_sampled)
    train_features = load_image_features('featuresresnet.pkl', train_sampled)
    print('Photos: train=%d' % len(train_features))
    X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions,
train_features)
```

```
    # define the model
    if(i == 0):
        model = define_model0(vocab_size, max_length)
        filepath = 'model-batches200-' + str(i) + '-ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
        # fit model
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)

    elif(i == 1):
        model = define_model1(vocab_size, max_length)
        filepath = 'model-batches200-' + str(i) + '-ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
        # fit model
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)

    elif(i == 2):
        model = define_model2(vocab_size, max_length)
```

```
filepath = 'model-batches200-' + str(i) + '-ep{epoch:03d}-loss{loss:.3f}-
val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
# fit model
model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)
```

V. Boosting Ensemble (Python source code)

```
# load a file into memory
def load_file(filename):
   file = open(filename, 'r')
   text = file.read()
   file.close()
   return text
# load a pre-defined list of photo identifiers of training set amd define weights
def load_trainset(filename):
   doc = load_file(filename)
   identifier_list = list()
   for l in doc.split('\n'):
      if len(l) < 1:
         continue
      identifier = l.split('.')[0]
```

```
        identifier_list.append(identifier)
    norm = len(identifier_list)
    train_weights  = dict()
    for i in range(len(identifier_list)):
        train_weights[identifier_list[i]] = 1.0 / norm
    return (set(identifier_list), train_weights)


#Sampling the dataset based on weights
def weight_sampling(identifiers, train_weights):
    identifiers = array(list(identifiers))
    train_weights = list(train_weights.values())
    p = np.random.choice(identifiers, 3000, p = train_weights, replace = False)
    return p


# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    word_seq = 'startword'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([word_seq])[0]
        sequence = pad_sequences([sequence], maxlen=34)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = argmax(yhat)
```

```
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        word_seq += ' ' + word
        if word == 'endword':
            break
    return in_text


# evaluate the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, inference = list(), list()
    reference, predicted = dict(), dict()
    for key, desc_list in descriptions.items():
        yhat = generate_desc(model, tokenizer, photos[key], max_length)
        references = [d.split() for d in desc_list]
        actual.append(references)
        inference.append(yhat.split())
        for d in desc_list:
            reference[key] = d
        predicted[key] = yhat
    with open("reference.txt", "w") as output:
        for key, item in reference.items():
```

```
            output.write("%s\t%s\n" % (key, item))
    with open("predicted.txt", "w") as output:
        for key, item in predicted.items():
            output.write("%s\t%s\n" % (key, item))
    print('BLEU-1: %f' % corpus_bleu(actual, inference, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, inference, weights=(0.5, 0.5, 0, 0)))
    print('BLEU-3: %f' % corpus_bleu(actual, inference, weights=(0.3, 0.3, 0.3, 0)))
    print('BLEU-4: %f' % corpus_bleu(actual, inference, weights=(0.25, 0.25, 0.25,
0.25)))
    return (reference, predicted)


def change_weights(train, train_weights, references, predicted):
    bleu_scores = dict()
    for key, item in references.items():
        bleu = sentence_bleu(list(references[key]), list(predicted[key]), weights =
(0.25, 0.25, 0.25, 0.25), smoothing_function = SmoothingFunction().method1)
        new_weight = train_weights[key] + 1 - bleu
        train_weights[key] = new_weight
        bleu_scores[key] = bleu
    with open("bleuscores.txt", "w") as output:
        for key, item in bleu_scores.items():
            output.write("%s\t%s\n" % (key, item))
```

```
    norm = sum(train_weights.values())
    for key, item in train_weights.items():
        train_weights[key] = item / norm
    return train_weights

# train dataset
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train, train_weights = load_trainset(filename)
train_sampled = weight_sampling(train, train_weights)
train_descriptions = load_descriptions('descriptions.txt', train)
train_features = load_image_features('featuresresnet.pkl', train)
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = 34

#load validation dataset
filename = 'Flickr8k_text/Flickr_8k.devImages.txt'
val = load_testset(filename)
val_descriptions = load_descriptions('descriptions.txt', val)
val_features = load_image_features('featuresresnet.pkl', val)
```

```
X1val, X2val, yval = create_sequences(tokenizer, max_length, val_descriptions,
val_features)


for i in range(6):
   train_descriptions = load_descriptions('descriptions.txt', train_sampled)
   # define the models
   if(i == 0):
      model = define_model0(vocab_size, max_length)
      train_features = load_image_features('featuresresnet.pkl', train_sampled)
      print('Photos: train=%d' % len(train_features))
      X1train, X2train, ytrain = create_sequences(tokenizer, max_length,
train_descriptions, train_features)
      filepath = 'model-batches200-' + str(i) + '-BestWeights.h5'
      checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
      # fit model
      model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)
      model = load_model(filepath)
      # evaluate model
      reference, predicted = evaluate_model(model, train_descriptions,
train_features, tokenizer, max_length)
   elif(i == 1):
```

```
        model = define_model0(vocab_size, max_length)
        train_features = load_image_features('featuresresnet.pkl', train_sampled)
        print('Photos: train=%d' % len(train_features))
        X1train, X2train, ytrain = create_sequences(tokenizer, max_length,
train_descriptions, train_features)
        filepath = 'model-batches200-' + str(i) + '-BestWeights.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
        # fit model
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)
        model = load_model(filepath)
        # evaluate model
        reference, predicted = evaluate_model(model, train_descriptions,
train_features, tokenizer, max_length)
    elif(i == 2):
        model = define_model0(vocab_size, max_length)
        train_features = load_image_features('featuresresnet.pkl', train_sampled)
        print('Photos: train=%d' % len(train_features))
        X1train, X2train, ytrain = create_sequences(tokenizer, max_length,
train_descriptions, train_features)
        filepath = 'model-batches200-' + str(i) + '-BestWeights.h5'
        checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1)
```

```
        # fit model
        model.fit([X1train, X2train], ytrain, batch_size = 256, epochs=20, verbose=2,
callbacks=[checkpoint], validation_data=([X1val, X2val], yval), shuffle=True)
        model = load_model(filepath)
        # evaluate model
        reference, predicted = evaluate_model(model, train_descriptions,
train_features, tokenizer, max_length)


    train_weights = change_weights(train, train_weights, reference, predicted)
    train_sampled = weight_sampling(train, train_weights)
```

VI. Evaluate Model (Python source code)

```
# generate a description for an image
def generate_desc(models, comb_method, tokenizer, photo, max_length):
    word_seq = 'startword'
    if(comb_method == 'MMP'):
        for i in range(max_length):
            pred = []
            max_value = []
            for each_model in model:
                sequence = tokenizer.texts_to_sequences([word_seq])[0]
                sequence = pad_sequences([sequence], maxlen = max_length)
```

```
            yhat = each_model.predict([photo,sequence], verbose=0)
            max_value.append(amax(yhat))
            pred.append(argmax(yhat))
        yhat = max(max_value)
        max_index = max_value.index(yhat)
        yhat = pred[max_index]
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        word_seq += ' ' + word
        if word == 'endword':
            break
elif(comb_method == 'MV'):
    for i in range(max_length):
        pred = []
        max_value = []
        for each_model in model:
            sequence = tokenizer.texts_to_sequences([word_seq])[0]
            sequence = pad_sequences([sequence], maxlen = max_length)
            yhat = each_model.predict([photo,sequence], verbose=0)
            pred.append(argmax(yhat))
        yhat = max(pred, key = pred.count)
        word = word_for_id(yhat, tokenizer)
```

```
            if word is None:
                break
            word_seq += ' ' + word
            if word == 'endword':
                break
    else:
        for i in range(max_length):
            pred = []
            max_value = []
            sequence = tokenizer.texts_to_sequences([word_seq])[0]
            sequence = pad_sequences([sequence], maxlen = max_length)
            yhats = [model.predict([photo, sequence], verbose=0) for model in
models]
            summed = np.sum(yhats, axis=0)
            yhat = argmax(summed, axis=1)
            word = word_for_id(yhat, tokenizer)
            if word is None:
                break
            word_seq += ' ' + word
            if word == 'endword':
                break
    return word_seq
```

```
# evaluate the model
def evaluate_model(model, comb_method, descriptions, photos, tokenizer,
max_length):
    actual, inference = list(), list()
    reference, predicted = dict(), dict()
    for key, desc_list in descriptions.items():
        yhat = generate_desc(model, comb_method, tokenizer, photos[key],
max_length)
        references = [d.split() for d in desc_list]
        actual.append(references)
        inference.append(yhat.split())
        for d in desc_list:
            reference[key] = d
        predicted[key] = yhat
    with open("reference.txt", "w") as output:
        for key, item in reference.items():
            output.write("%s\t%s\n" % (key, item))
    with open("predicted.txt", "w") as output:
        for key, item in predicted.items():
            output.write("%s\t%s\n" % (key, item))
    print('BLEU-1: %f' % corpus_bleu(actual, inference, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, inference, weights=(0.5, 0.5, 0, 0)))
```

```
    print('BLEU-3: %f' % corpus_bleu(actual, inference, weights=(0.3, 0.3, 0.3,
0)))
    print('BLEU-4: %f' % corpus_bleu(actual, inference, weights=(0.25, 0.25, 0.25,
0.25)))
    return (reference, predicted)

# prepare train set
filename = 'Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_document(filename)
train_descriptions = load_descriptions('descriptions.txt', train)
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = 34

# prepare test set
filename = 'Flickr8k_text/Flickr_8k.testImages.txt'
test = load_document(filename)
test_descriptions = load_descriptions('descriptions.txt', test)
test_features = load_image_features('featuresresnet.pkl', test)
print('Photos: test=%d' % len(test_features))
filename1 = 'model-kfold-0-5.h5'
filename2 = 'model-kfold-1-5.h5'
```

```
filename3 = 'model-kfold-2-4.h5'
model1 = load_model(filename1)
model2 = load_model(filename2)
model3 = load_model(filename3)
model = [model1, model2, model3]
combination_method = 'AV'
# evaluate model
evaluate_model(model, combination_method, test_descriptions, test_features,
tokenizer, max_length) 118 118
```