A project report on

IMAGE CAPTION GENERATION

Natural Language Processing (CSE4022)

by

KIRTI BHAGAT (17BLC1046)

DIBYANSHU GAUTAM (17BCE1328)

SCOPE

May,2020

ROLE AND RESPONSIBILITES

This project was possible due to the combined efforts of both the members. This project required both theoretical and practical knowledge of the subjects of natural language processing, computer vision, and deep learning.

Kirti Bhagat

- Data collection
- Feature extraction and cleaning
- Exploring the possibilities of CNN
- Code implementation.

Dibyanshu Gautam

- Model survey
- Exploring possibilities of RNN and LSTM
- Model evaluation
- Code restructuring

Table of Contents

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 IMAGE CAPTION GENERATION

There is a history of variations in development of models that are able to detect objects but the future of AI surely lies in providing a human-like description. Knowledge about recent advancements in this field is critical for future evolution. The ultimate aim of creating such types of solutions is to make machines that can better replicate humans. There have been many variations and combinations of different techniques since the first such model made in 2014. The project aims to create one such application to be able to generate a textual description given an image that is related to it semantically.

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given image input.

It requires methods from both computer vision and natural language processing. Computer vision to understand the content and features of the image and natural language processing to turn the understanding of the image into words in the right order. Recently, deep learning methods have achieved state-of-the-art results on examples of this problem.

Deep learning methods have demonstrated state-of-the-art results on caption generation problems. What is most impressive about these methods is that a single end-to-end model can be defined to predict a caption, given an image, instead of requiring sophisticated data preparation or a pipeline of specifically designed models.

The Project Report consists of a vivid description of the various modules used to make this project. We have tried to explain each step in-depth. We also provide an explanation for our choice of modules and display final results and accuracy that we could achieve for the image captioning problem.

1.2 OVERVIEW OF THE SYSTEM

In the past few years, the problem of generating descriptive sentences automatically for images has garnered a rising interest in natural language processing and computer vision research. Image captioning is a fundamental task which requires semantic understanding of images and the ability of generating description sentences with proper and correct structure. In this project, we propose a hybrid system employing the use of multilayer Convolutional Neural Network (CNN) to generate vocabulary describing the images and a Recurrent Neural Network (RNN) to accurately structure meaningful sentences using the generated keywords. The convolutional neural network compares the target image to a large dataset of training images, then generates an accurate description using the trained captions.

The model is trained in such a way that if input image is given to model it generates captions which nearly describes the image. We showcase the efficiency of our proposed model using the Flickr8K datasets and show that the model gives superior results compared with the state-of-the-art models utilizing the BLEU metric. The BLEU metric is an algorithm for evaluating the performance of a machine translation system by grading the quality of text translated from one natural language to another. The performance of the proposed model is evaluated using standard evaluation matrices, which outperform previous benchmark models.

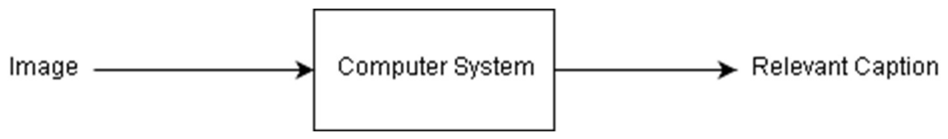


Figure 1: A basic overview

Being a somewhat new area of research in Artificial Intelligence, Image Caption generation is one of the most sought-after challenges in recent times. Researchers have been working on them for quite a few years and surely there are some challenges one might face while working on these. We would explore them in the next section.

1.3 CHALLENGES PRESENT

Although there are a lot many caption generators and various different mechanisms, experimental results show that there can be a lot of improvement in terms of accuracy and speed. It mainly faces the following three challenges: first, how to generate complete natural language sentences like a human being; second, how to make the generated sentence grammatically correct; and third, how to make the caption semantics as clear as possible and consistent with the given image content.

These challenges require us to constantly look for better and faster alternatives for both encoding and decoding the data in order to create an acceptable result for future use.

1.4 OBJECTIVE

Machine translation is advancing at an alarming rate due to the technical developments in the field of machine learning. The rapid developments in the field of machine learning is affecting and improving many branches of the technical industry. The application of Artificial Intelligence and Neural Networks, to complicated natural language processing challenges like speech recognition and machine translation, is leading to remarkably rapid advancements.

There are majorly two main objectives of this system, one is to generate features from the images and the other is to predict a caption by combining these features with the textual features in order to generate semantically correct captions.

Image caption, automatically generating natural language descriptions according to the content observed in an image, is an important part of scene understanding, which combines the knowledge of computer vision and natural language processing. The application of image caption is extensive and significant, for example, the realization of human-computer interaction.

This project works towards improving the current scenario and to make scene understanding or Interaction easier bringing computer systems closer to the ultimate goal of artificial intelligence. All of these would be discussed in the coming sections.

1.5 SCOPE OF THE PROJECT

Image caption generation have loads of application right now as its directly connected to the human interaction. It can be used as recommendations in editing applications, usage in virtual assistants, for image indexing, for visually impaired people, for social media, and several other natural language processing applications. We have so many major companies that are working towards Image caption generation and its related applications. Some of them are:

- Skin Vision: Lets you confirm whether a skin condition can be skin cancer or not.
- Google Photos: Classify your photo into Mountains, sea etc.
- Deep mind: Achieved superhuman level playing Game Atari.
- Facebook: Using AI to classify, segmentate and finding patterns in pictures.

These are some of the major applications in the real world that are currently using Image Caption generative models to improve on their already created system. Mapping natural language to images and vice versa has proven to be very effective and applicable. Thus, such a model with good accuracy and performance can have a wide scope in the future ranging from medical applications to even social media platforms.

CHAPTER 2

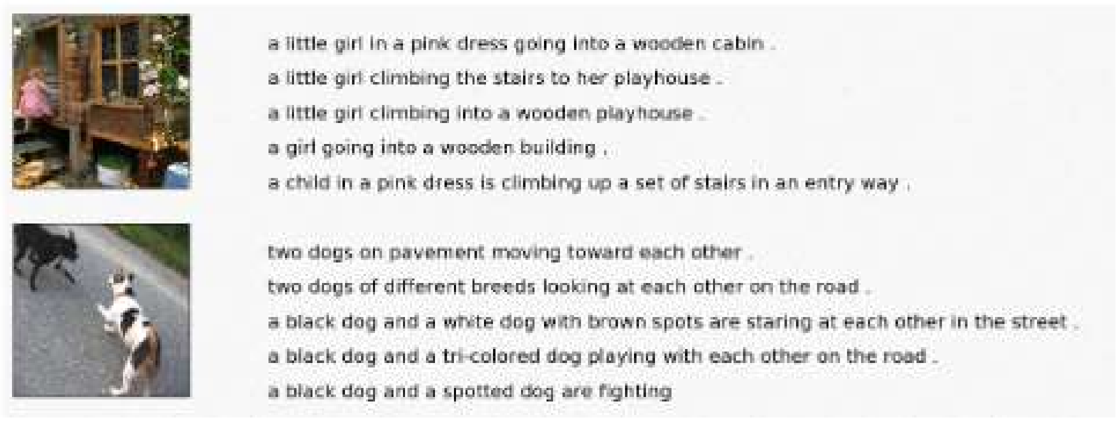
METHODOLOGY AND MATERIALS

2.1 DATASET TO BE USED

Since we are starting at a very beginner's level to tackle this problem, we thought of using the most famous dataset Flickr8K dataset.

The reason to start with this dataset is because it is relatively small and realistic so that we can download and then build our own models easily. The definitive description of the dataset is in the paper "Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics" from 2013. It has been described here as a benchmark collection for sentence-based image generation consisting of 8000 images paired with 5 different caption that provide clear entries about the photograph. These images were randomly selected from 6 different Flickr groups.
It is a free dataset available online and when downloaded it has two different files: -

- Flickr8k_Dataset.zip - Archive of all photographs.
- Flickr8k_text.zip – Text descriptions of all photos.



a little girl in a pink dress going into a wooden cabin .
a little girl climbing the stairs to her playhouse .
a little girl climbing into a wooden playhouse .
a girl going into a wooden building .
a child in a pink dress is climbing up a set of stairs in an entry way .

two dogs on pavement moving toward each other .
two dogs of different breeds looking at each other on the road .
a black dog and a white dog with brown spots are staring at each other in the street .
a black dog and a tri-colored dog playing with each other on the road .
a black dog and a spotted dog are fighting

Figure 2: Flickr Dataset Each picture has 5 captions

2.2 MODULES OF THE PROJECT

2.2.1 Large Scale Visual Recognition

Large Scale Visual Recognition requires algorithms for object detection and image classification at large scale. Many CNN models are available which are pre-trained. They act as encoders and along with their pre-trained weights, are used to encode an image to its feature set.
The CNN starts with convolution layer which applies a mask over a portion of the image continuously to obtain a matrix of extracted features of the entire image. This is then followed by applying rectified linear unit (ReLU). ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. We then incorporate MaxPool layer to extract relevant features describing the context of the image. In this phase the dimensionality of convlayer or feature map gets reduced keeping only the important information. Then, the image encoding is computed by the fully connected layer depending upon the distinct weights of activation. The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

2.2.2.1 VGG-16

VGG-16 is a pre-trained convolutional neural network on ImageNet. Its name VGG-16 comes from the fact that it has 16 layers. This CNN is in the form of repeated blocks of conv-ReLU architecture, each terminating with a maxpool layer which selects features with the high significance relative to other features in the image.
It consists of 13 convolutional layers, 5 Max Pooling layers and 3 Dense layers which sums up to 21 layers but only 16 weight layers.

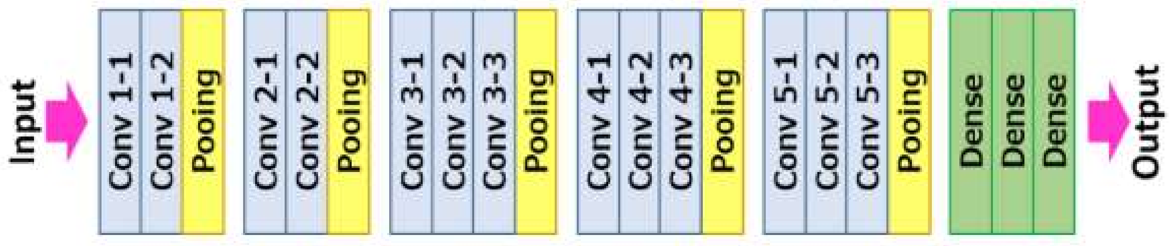


Figure 3: VGG Layer by Layer

Small-size convolution filters allows VGG to have a large number of weight layers. This leads to improved performance. VGG16 also has fewer parameters**.** This is better for faster convergence and less overfitting problem. However, vanishing gradient problem is quite prevalent in VGG16 as it has a sequential stack of layers. As we go deeper in this stack the gradient value that updates the weight parameters becomes so small such that the update is equivalent to zero resulting in dead neurons.
Some challenges faced by VGG16 are that it is very slow to train (the original VGG model was trained on Nvidia Titan GPU for 2-3 weeks). The size of VGG-16 trained ImageNet weights is 528 MB. So, it takes quite a lot of disk space and bandwidth that makes it inefficient.

2.2.2.2 VGG16 with batch normalization

Batch normalization was first introduced in a 2015 paper by researchers at Google, Sergey Ioffe and Christian Szegedy in their paper 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'.
Batch Normalization is that small component in a neural network that continuously takes the output of a particular layer and normalizes it before sending it across to the next layer as input. It is used for improving the speed, performance, and stability of neural networks.
Neural networks have hidden layers and these hidden unit's input distribution changes every time there is a parameter update in the previous layer. This is **internal covariate shift which** makes training slow and requires a very small learning rate and a good parameter initialization. This problem is solved by normalizing the layer's inputs over a mini-batch by the process called **Batch Normalization**.
VGG lacks batch normalization, because at the time it was created batch normalization didn't exist. We can however add this in our VGG16 model. We do this by initializing the scaling and shifting parameters for batch normalization as the standard deviation and mean of the inputs. In our first pass through, the normalization transformation is effectively undone, and the next layer's weights are still optimal. Then, back-propagation updates the scaling and shifting parameters in an appropriate matter.

2.2.2.3 Layers in VGG

VGG has been tried and tested using many different numbers of layers like 11,13,16 and 19. The error evaluated with these different models were:

- VGG-11 obtains 10.4% error rate
- VGG-13 obtains 9.9% error rate
- VGG-16 obtains 8.8% error rate
- VGG-19 obtains 9.0% error rate

This analysis shows us that initially additional convolution layers helped in the classification accuracy. However, after VGG16 the error rate increased. This means the deep learning network is not improving by adding number of layers. Thus, authors stop adding layers.

2.2.2 Model Architectures and Computational Resources.

In this part, we will discuss about the architecture that are generally being used for Caption generation problems and what are some of the recent researches that have worked behind it. The most used architecture is the merge-model as described by Marc Tanti in two of his papers in 2017. Based on the Encoder-decoder architecture, merge-model was one of the first successful models for Caption generation and currently this project works on that.
A large portion of the models depend on the across the board encoder–decoder system, which is adaptable and
compelling. Now and then it is characterized as a structure of CNN + RNN. Generally, a convolutional neural network (CNN) is the encoder, and a recurring neural network (RNN) the decoder. The encoder is the one which inputs a picture and it removes an elevated level of features. The decoder is the one which produces words—given the features extracted from the encoder (encoded picture), it produces words to related to the picture with a full syntactically and elaborately right manner in a sentence.

2.2.2.1 Encoder

As discussed in our first section, VGGNet is the most popular choice amongst all others due to its simplicity and power of accessing Large scale images and getting features accurately. But there are some other networks too which have been used in recent times., One of the popular ones amongst them is ResNet due to its computational efficiency. There are comparisons among many such encoders so we have found these four major ones who have been the best in the lot for Caption generation purposes.

As seen in the table, we had four major encoders and Resnet seems to have better accuracy, but VGG remains easy to use and hence it is the popular choice right now. But it completely depends on what is important, either to go with simplicity or to be completely performance centric.

Table 1: Comparison of Different Encoder Architectures

ENCODER (CNN) ARCHITECTURES

Architecture	Parameters	Multiply-Add Ops	Top-1 Accuracy	Top-5 Accuracy	Year
AlexNet	61M	724M	57.1	80.2	2012
VGG	138M	15.5B	70.5	91.2	2013
Inception – V1	7M	1.43B	69.8	89.3	2013
Resnet-50	25.5M	3.9B	75.2	93	2015

2.2.2.2 Decoder

It is seen that mostly Decoders depends on an RNN. These are mainly used in all kinds of generation models which require prefix results. In a neural language model, an RNN encodes a pre-fix (for example, the caption generated so far) and either itself predicts the next item in the sequence with the help of a feed forward layer or else it passes the encoding to the next layer which will make the prediction itself. This new item is then used or feeded again to the network in order to generate the next one in the sequence until the sequence ultimately reaches an end state. As shown in the image, <beg> is the beginning of sequence, that is a start of sequence token given at the beginning to start of the network. After each iteration the prediction takes place using a Feed-Forward (FF layer) which usually consists of an activation function. <end> is the end-of sequence.

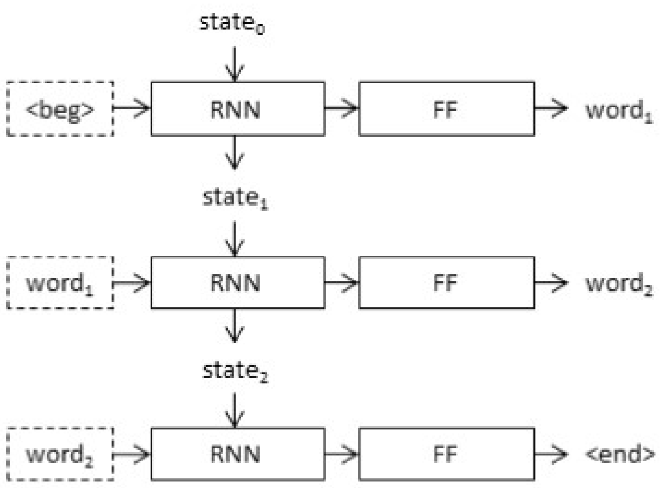


Figure 4: A generic RNN Model

LSTM (long-short-term memory) was developed from RNN, aimed to work for sequential based data. Widely popular, LSTM's are used due to their prowess in memorizing long term dependencies through a memory cell. They work by generating a caption by making one word at each step together with the previous hidden state and generated words. That's the reason why scientists ultimately try using LSTM's for such systems even though they are not that simple as compared to its alternatives.

2.2.2.3 Architecture

There are two major architectural models when it comes to Image caption generation. These two models were explained by Marc Tanti in his research about such systems. These were the Inject and Merge models. Both these methodologies are explained in detail in further sections.

2.2.2.3.1 Inject Model

Inject model as the name suggests injects the encoded data into the decoder. The catch here is that both the image encoded data and the word data is injected simultaneously to the same RNN/LSTM to get the next result.

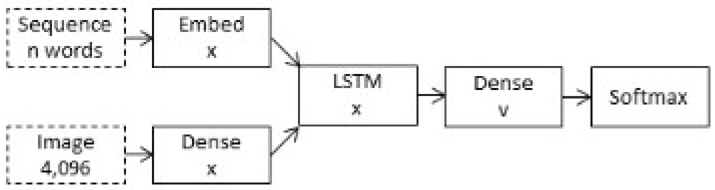


Figure 5: Inject Model Architecture

In this model both visual and linguistic features are injected directly into the RNN layer. In an inject model, the RNN is trained to predict sequences based on histories consisting of both linguistic and perceptual features.
The RNN here is used primarily for image conditioned language. Inject model can be done in three ways. First of them is the Init-inject method where the image vector is used as an initial hidden state for the RNN. Second is the Pre-inject method where the image vector is used as a first word prefix. Another one is Par inject where the RNN accepts two inputs at each step. All of these sub architectures can be clearly seen below.

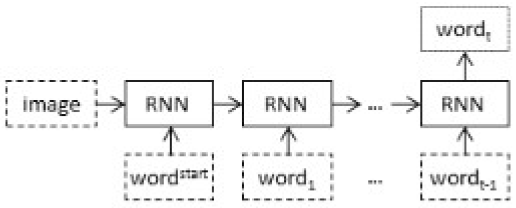


Figure 7: Init Inject Model

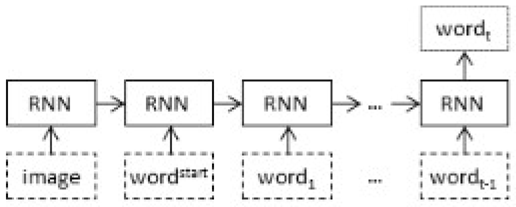


Figure 6 Pre-Inject Model

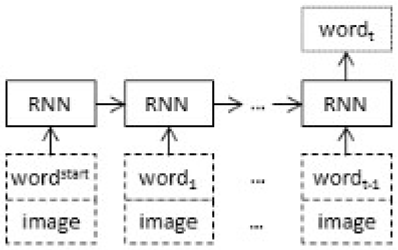


Figure 8: Par inject method

2.2.2.3.2 Merge Model

Merge model differs in the very basics of the architecture we just saw now. Rather than combining image features along with the linguistic features, the combination happens only after the caption prefix has been vectorized. Here the image and the words are worked upon separately and then ultimately added in a multimodal layer to predict the result. In easier words, here the RNN handles only the linguistic information or the encoded prefix and then is ultimately combined with image features later. The number of parameters of a merge model is also different from the inject ones. One of the most vital aspect about this model is that it permits a pre trained language model also in addition with a pre trained image encoding model. That is the reason why merge model performs better and quicker as well since it deals with less amounts of data. The success of merge model suggests that the role of RNN is to encode the input rather than to generate output.

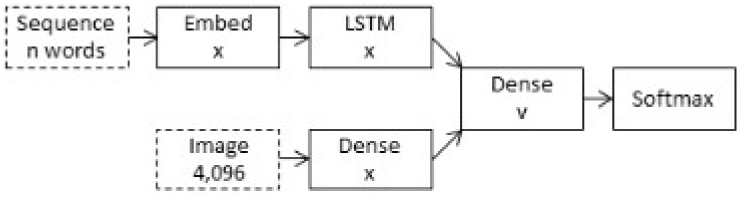


Figure 9: Merge model architecture

Both of these architectures have been used sometime or the other but merge model remains to be the one having more advantages due to its optimized approach towards the output generation. Hence, it is quite popular and used frequently by others when predicting sequence to sequence(seq2seq) tasks in the system. The comparison also shows that RNN’s are not the generators but they can be better used to encode data here.

2.3 SYSTEM ARCHITECTURE

The System architecture of the project consists of a User which interacts with the System. The System is connected to the Database and the source code. Whenever the system is called, the Source code starts functioning and the database is used for model creation and evaluation. This can be better explained using the below Use Case and Class Diagrams respectively.

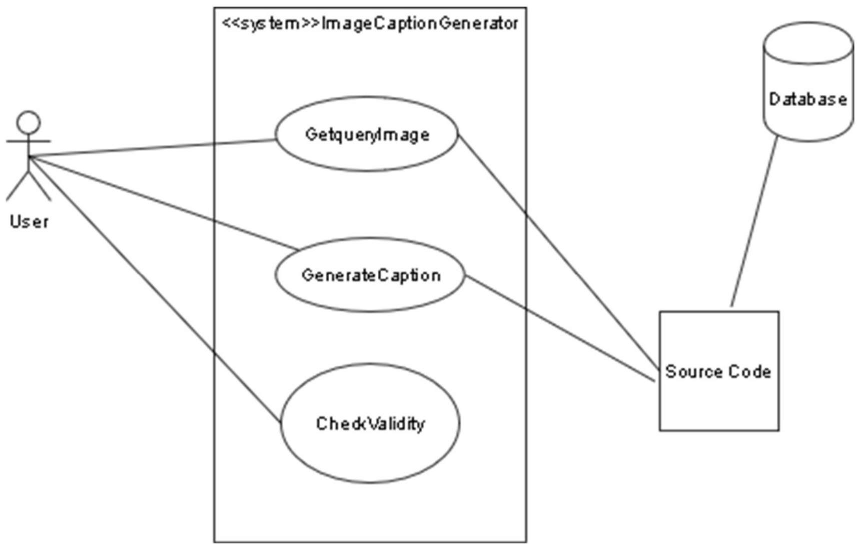


Figure 10 Use case Diagram of System Architecture

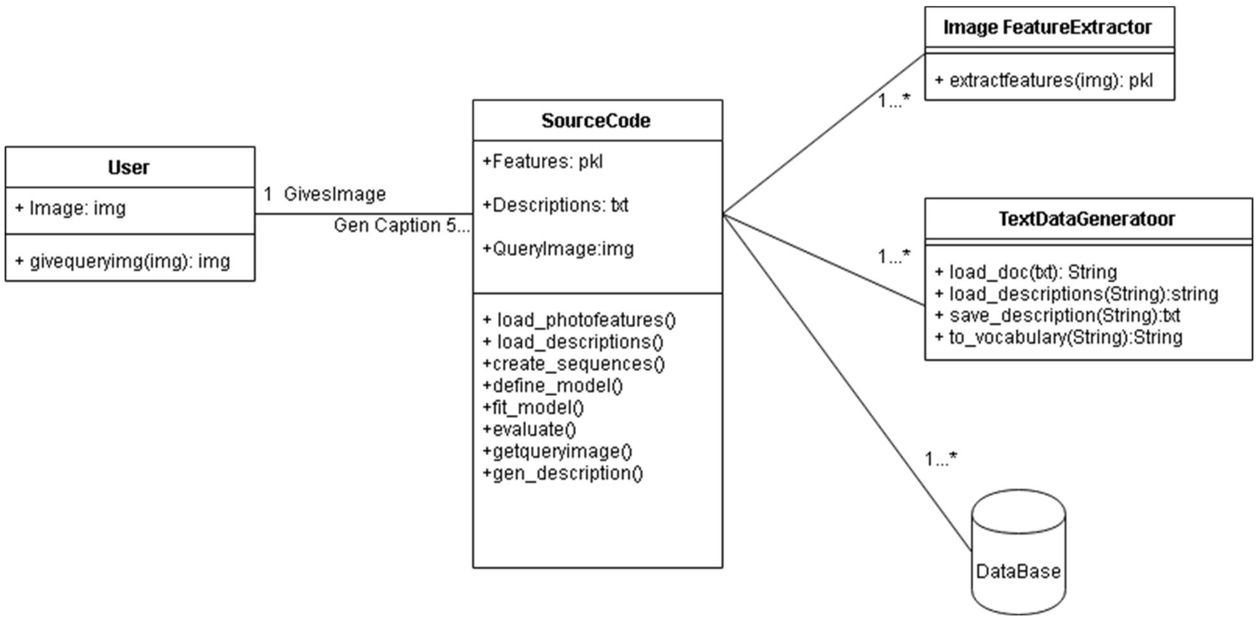


Figure 11 Class Diagram of System Architecture

2.4 EVALUATION METRIC

2.4.1 BLEU

BLEU or the Bilingual Evaluation Understudy, is score for comparing a candidate translation of text to one or more reference translations. BLEU is quick to calculate and is language independent as well, hence is widely adopted as well. Its approach is to count matching n-grams in candidate to the same in reference text. The counting of matching n-grams is modified to ensure that it takes the occurrence of the words in the reference text into account, not rewarding a candidate translation that generates an abundance of reasonable words. This is referred to in the paper as modified n-gram precision.

There are different BLEU versions depending on the value of N in the process of taking words and then calculating them cumulatively in order to get the value. Due to its easily available libraries and easier way of calculation, BLEU becomes the top choice in order to judge the captions generated by the system with respect to a description text.

The approach works by counting matching n-grams in the candidate translation to n-grams in the reference text, where 1-gram or unigram would be each token and a bigram comparison would be each word pair. The comparison is made regardless of word order. The counting of matching n-grams is modified to ensure that it takes the occurrence of the words in the reference text into account, not rewarding a candidate translation that generates an abundance of reasonable words.

CHAPTER 3

IMPLEMENTATION AND RESULTS

3.1 FLOW OF PROJECT

The project is divided int o major checkpoints for easier understanding. These are the vital stages in the system on which the output depends upon. All of these are in a chronological order for better understanding of the user or anyone related.

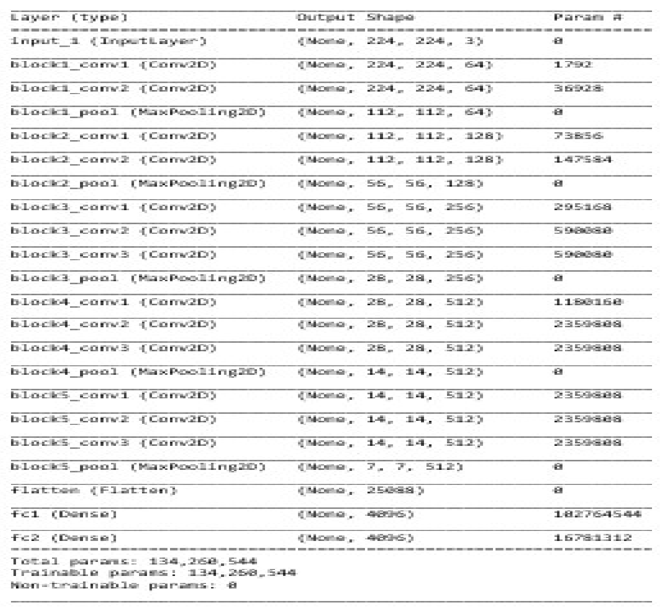
- Prepare the Image Dataset
- Prepare the Text Dataset
- Develop the Model for loading data and Fit it.
- Train the model
- Evaluation and Final results

All of these stages are explained in detail in the following sections of the chapter for your reference.

3.2 PREPARE THE IMAGE DATASET

The objective here is to get the features from each of the image so that we can classify the images and actually see any trends between similar images or similar objects in different images. For this purpose, we used one of the most famous pre trained models to generate the feature vector, Oxford Visual Geometry group or the VGG Model. VGG is used for large scale Visual Recognition.

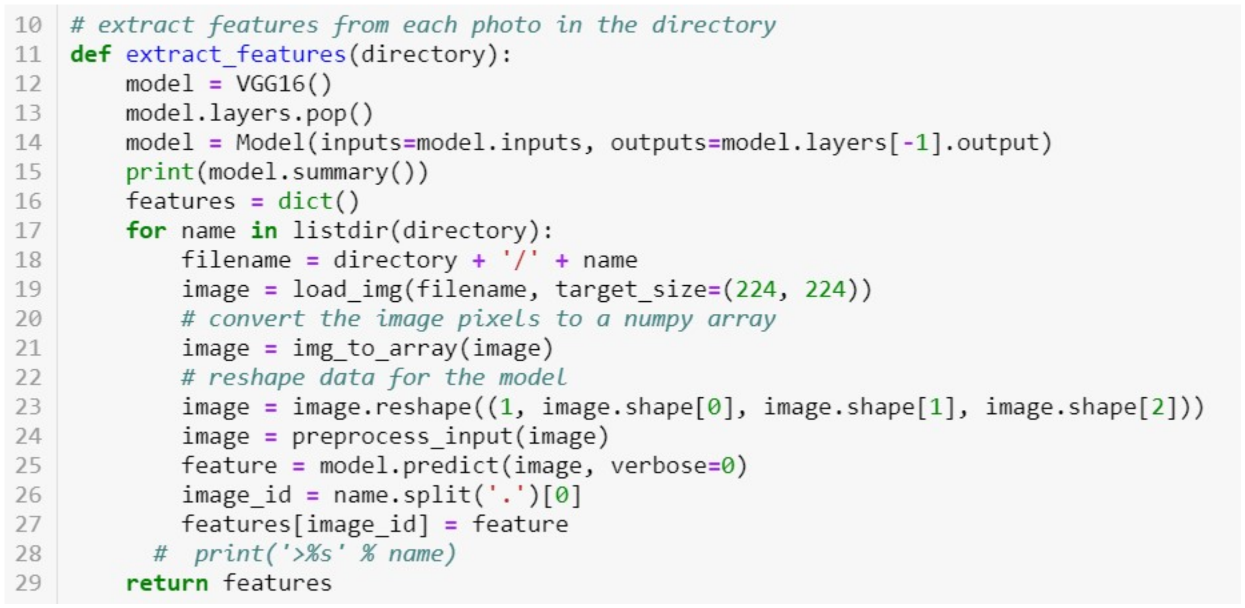
We load the VGG model in Keras using the VGG class and removed the last layer from the loaded model, as this is the model used to predict a classification for a photo. We are not interested in classifying images, but we are interested in the internal representation of the photo right before a classification is made. These are the "features" that the model has extracted from the photo.
Keras also provides tools for reshaping the loaded photo into the preferred size for the model (e.g. 3 channels 224 x 224pixel image).



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: VGG Model Summary

Code:



```python
# extract features from each photo in the directory
def extract_features(directory):
    model = VGG16()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    print(model.summary())
    features = dict()
    for name in listdir(directory):
        filename = directory + '/' + name
        image = load_img(filename, target_size=(224, 224))
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        # reshape data for the model
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = preprocess_input(image)
        feature = model.predict(image, verbose=0)
        image_id = name.split('.')[0]
        features[image_id] = feature
      #  print('>%s' % name)
    return features
```

In this function, we call the VGG class which loads the 512MB model. We then pop the last layer from the model which classifies the objects in the image. We only want the extracted features from the model.

We then use this loaded model to obtain the features of our image dataset and save them in a dictionary called features.

3.3 PREPARE TEXT DATASET

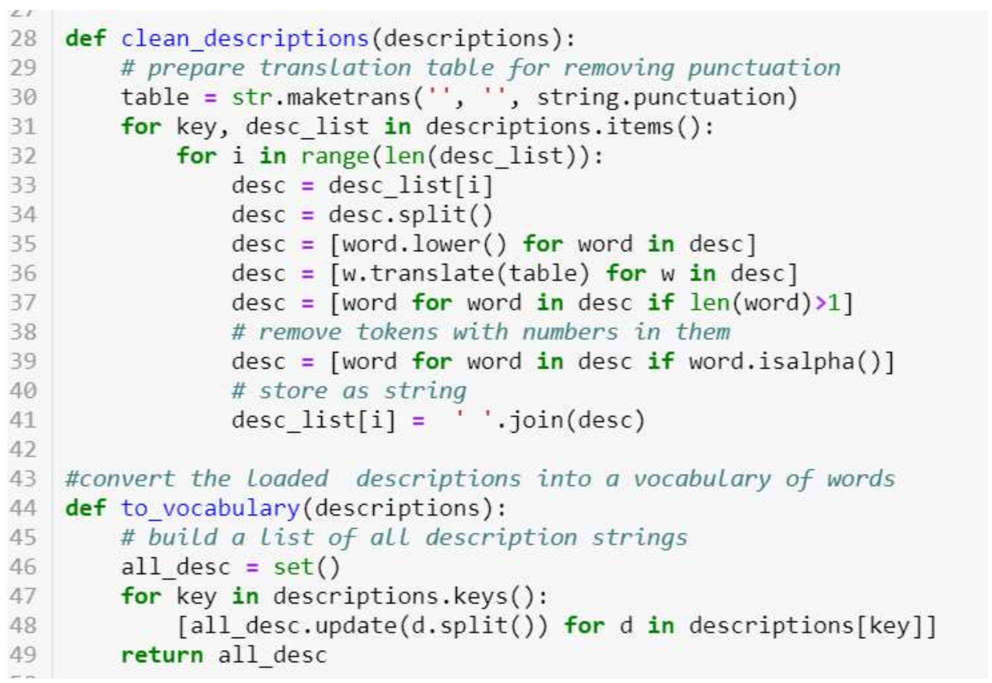
Like any other text dataset, All the descriptions in Flickr8k dataset aren't perfect and require minimal cleaning to get better and more accurate predictions when we fit it in the model. The best part of this dataset is that descriptions are already tokenized so it becomes easier to at least one step.

The text is cleaned in the following ways in order to reduce the size of the vocabulary of words we will need to work with:

- Convert all words to lowercase.
- Remove all punctuation.
- Remove all words that are one character or less in length (e.g. 'a').
- Remove all words with numbers in them.

Once cleaned, we summarize the size of the vocabulary. Ideally, a vocabulary must be both expressive and as small as possible. A smaller vocabulary will result in a smaller model that will train faster. Finally, we save the dictionary of image identifiers and descriptions to a new file named *descriptions.txt*, with one image identifier and description per line.

Code:



```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    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(table) for w in desc]
            desc = [word for word in desc if len(word)>1]
            # remove tokens with numbers in them
            desc = [word for word in desc if word.isalpha()]
            # store as string
            desc_list[i] =  ' '.join(desc)

#convert the loaded  descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc
```

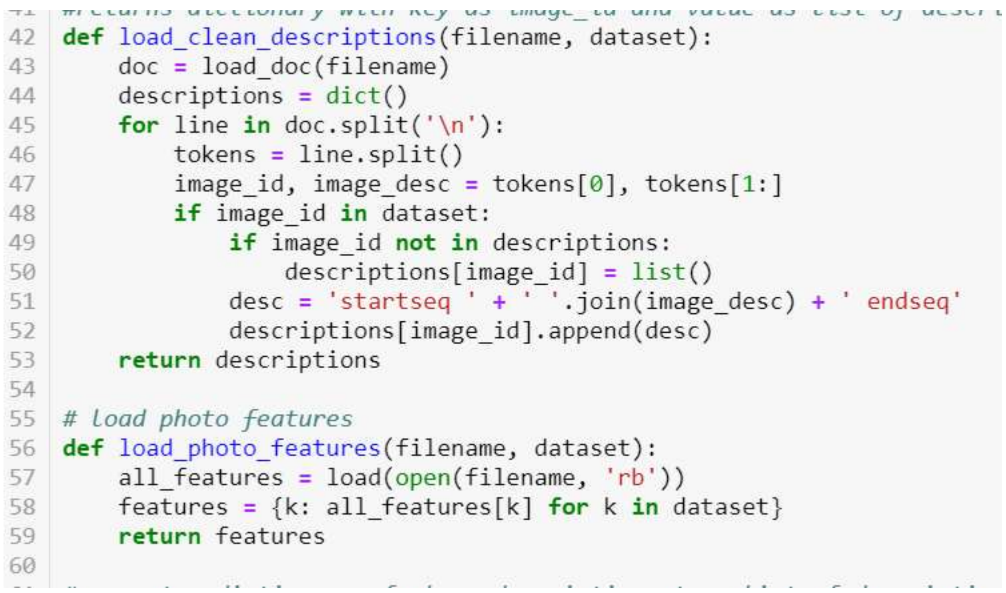
The first function cleans the descriptions of each image. The second function makes a set of all the words encountered in the dataset to make our vocabulary.

3.4 DEVELOPING THE MAIN MODEL

There are three major steps while developing the model and training it to get the system ready.

- Loading the Training data
- Defining the model
- Evaluating with Test data

3.4.1 Loading Data



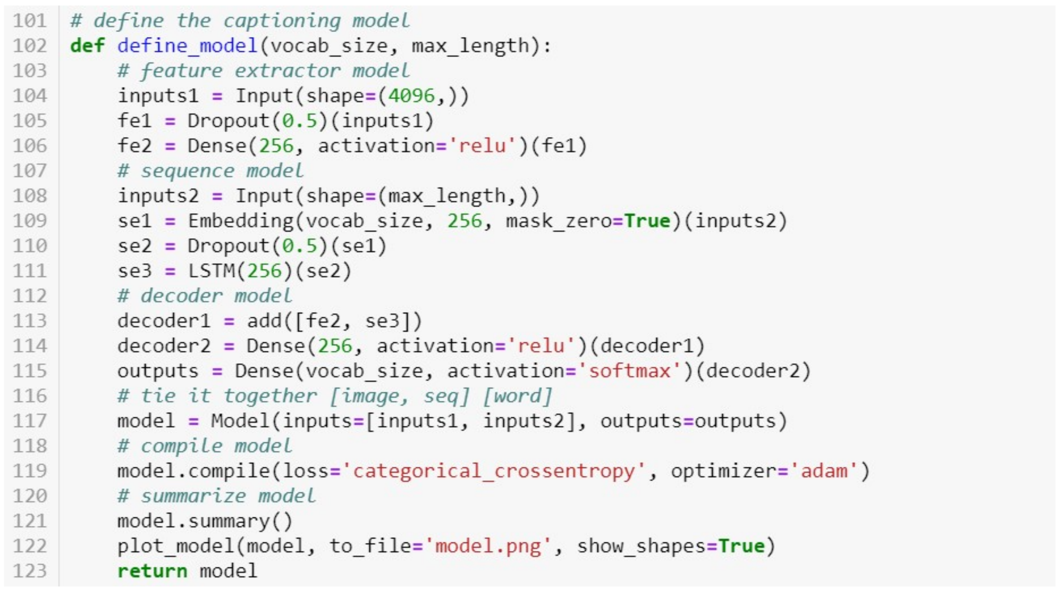
```python
#returns dictionary with key as image_id and value as list of descri
def load_clean_descriptions(filename, dataset):
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in dataset:
            if image_id not in descriptions:
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            descriptions[image_id].append(desc)
    return descriptions

# load photo features
def load_photo_features(filename, dataset):
    all_features = load(open(filename, 'rb'))
    features = {k: all_features[k] for k in dataset}
    return features

```

We use the function load_clean_descriptions to load clean descriptions into memory and get a dictionary with key as image_id and value as list of descriptions.
Function load_photo_features is used to load all the features of image from training dataset into the memory.
We now use these features as combined input for our deep learning model.

3.4.2 Define the Model



```python
# define the captioning model
def define_model(vocab_size, max_length):
    # feature extractor model
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    # decoder model
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    # compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    model.summary()
    plot_model(model, to_file='model.png', show_shapes=True)
    return model
```

This function is the heart of our project which defines the deep learning model. We initialized a sequential neural network with two set of input layers. One from the image features and second from the clean descriptions. The following layers are used to process these inputs before combining them using the Model class:

Dropout layer: The Dropout layer randomly sets input units to 0 with a frequency of rate (0.5 here) at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/ (1 - rate) such that the sum over all inputs is unchanged.

Dense layer: Provides an output with the specified units as its dimensions after applying the activation function.

LSTM: This helps our model learn what information to store in long term memory and what to get rid of.

We now compile the entire model and the summary is as follows:

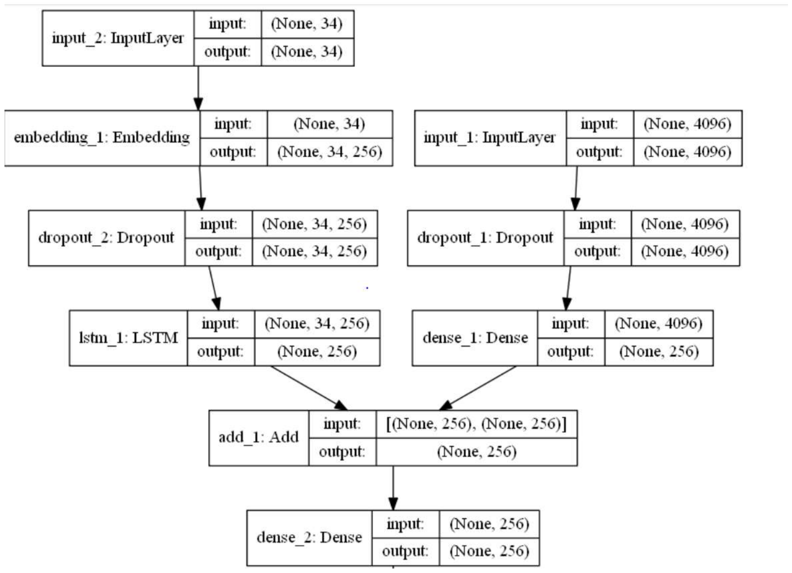
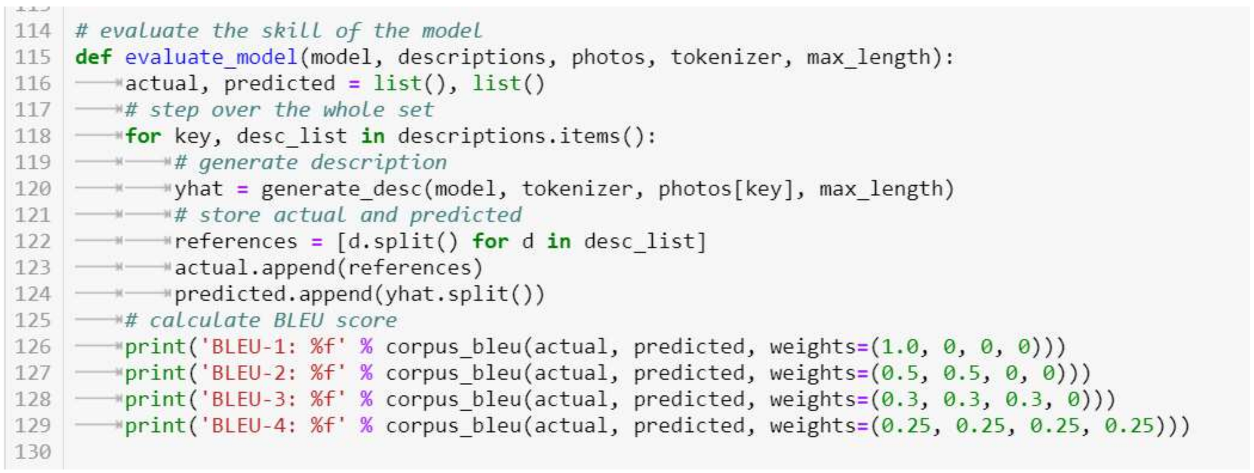


Figure 13 Main Model Summary

3.4.3 Evaluating the Model

After fitting our model with 20 epochs, we are ready to evaluate the model using the bleu score.



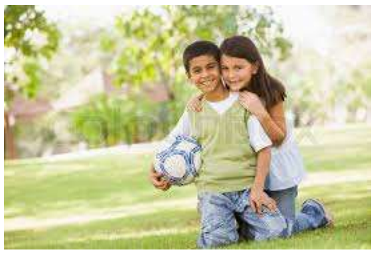
```
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
	actual, predicted = list(), list()
	# step over the whole set
	for key, desc_list in descriptions.items():
		# generate description
		yhat = generate_desc(model, tokenizer, photos[key], max_length)
		# store actual and predicted
		references = [d.split() for d in desc_list]
		actual.append(references)
		predicted.append(yhat.split())
	# calculate BLEU score
	print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
	print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
	print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
	print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

```

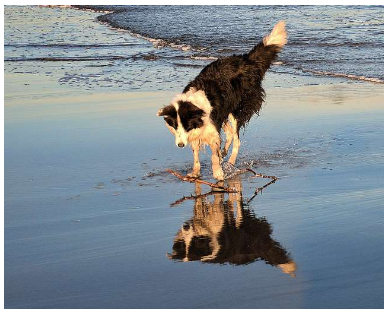
This function compares the actual captions with the predicted one to give the blue score.

3.5 RESULTS AND OUTPUT

We finally tried our model with our own dataset images to see the output. The following are the results:



```
startseq two girls are playing in the grass endseq
```



```
startseq dog is running through the water endseq
```

As seen, the model was successful in getting majority of the description accurate and worked quite well.

CHAPTER 4

CONCLUSION AND FUTURE WORK

In the end, we can conclude that image caption generation remains one of the most interesting and sought-after challenge after its first major challenge emerged in 2015. There are proofs that scientists are working every day to dive deep into how to improve the pre-established system. While going through various articles and paper, we got to know that this topic is surely to remain active for few years from now. Since the amount of data is increasing day by day, it would be really interesting to see what other things can also be done in this prevalent structure. Each module of this particular project has and can be seen develop rapidly over the course of few years.

In future we can be seeing models that perform better than this current one. We already are seeing better CNN Models like VGG- 16 with Batch Normalization or VGG 19 for this purpose. Other modules also have been under change. There are a lot of Challenges that are organized worldwide on this topic. One of the first of them was the Microsoft COCO Challenge organized in 2015 that changed the complete scenario of Caption generation and opened a new dimension of work for researchers and developers as well.

The Projects bring the best of Computer Vision techniques and Natural Language Processing into one big package that generates a lot of interest towards it. Maybe in future we might see the models improve drastically and be used in the daily interaction as well.

CHAPTER 5

REFERENCES

1. Raimonda Staniut˙e and Dmitrij Šešok: A Systematic Literature Review on Image Captioning (2019)
   Sauletekio
2. Marc Tanti, Albert Gatt and Kenneth P Camilleri: Where to put Image in an Image Caption Generator (2018) University of Malta.
3. Marc Tanti, Albert Gatt and Kenneth P Camilleri: What is the Role of Recurrent Neural Networks in an Image Caption Generator? (2018) University of Malta.
4. Jason Brownlee: Caption Generation with Inject and Merge Encode-Decoder Models (2017).
5. Kaustubh: ResNet, AlexNet, VGGNet, Inception: Understanding Various Architectures of

   Convolutional Networks (2019)

6. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-JingZhu: BLEU: A method for Automatic Evaluation of Machine Translation (2002) IBM, New York, USA
7. Rafaella Bernardi, Ruket Cakici, Desmond Elliot: Automatic Description Generation from Images: A Survey of Models, Datasets, and Evaluation Measures (2017)
8. Sergey Ioffe and Christian Szegedy: 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (2015) Google USA.