VISUAL DISABILITY HANDLING USING NATURAL LANGUAGE PROCESSING

A PROJECT REPORT

Submitted by

ABILASH K	**180801003**
ARUN KUMAR G	**180801018**
BHARATH KUMAR P	**180801023**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

SRI VENKATESWARA COLLEGE OF ENGINEERING

(An Autonomous Institution; Affiliated to Anna University, Chennai-600025)

ANNA UNIVERSITY :: CHENNAI 600 025

JUNE 2022

SRI VENKATESWARA COLLEGE OF ENGINEERING
(An Autonomous Institution; Affiliated to Anna University, Chennai-600025)

ANNA UNIVERSITY :: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **“VISUAL DISABILITY HANDLING USING NATURAL LANGUAGE PROCESSING”** is the bonafide work of “**ABILASH K (180801003), ARUN KUMAR G (180801018) AND BHARATH KUMAR P (180801023)”** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. V. VIDHYA, M.E, Ph.D.,	**Ms. K. KIRUTHIKA DEVI, M.E.,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
	ASSISTANT PROFESSOR
INFORMATION TECHNOLOGY	**INFORMATION TECHNOLOGY**

Submitted for the project viva-voce examination held on ______________________

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ABSTRACT

Normal people look at an image and give the response to the answer for any questions utilizing their insight. But visually impaired people have to rely on others for help to interpret and understand an image. Several technologies have been developed for assistance of visually impaired people. One of the ways blind people understand their surroundings is by clicking images and relying on descriptions generated by image captioning systems. Being able to describe the content of an image using properly formed English sentences is a challenging task, but it could have great impact by helping visually impaired people better understand their surroundings. These descriptions can then be read out loud to the visually impaired. This project proposes to present a memory-efficient Image captioning model which is trained with batches of data to avoid memory error during training of the model. The Image captioning model uses a Convolutional Neural Network to extract features from an image. These features are then fed into a Long Short Term Memory network to generate a description of the image in a valid English sentence. The description is converted to audio output using a python Text-to-Speech Application Programming Interface. The proposed model can generate highly descriptive captions that can potentially aid the visually impaired people.

ACKNOWLEDGEMENT

We thank our Principal **Dr. S. Ganesh Vaidyanathan, M.E., M.S., Ph.D.,** Sri Venkateswara College of Engineering for being the source of inspiration throughout our study in this college.

We express our sincere thanks to **Dr. V. Vidhya, M.E., Ph.D., Head of the Department**, Information Technology for her encouragement accorded to carry this project.

With profound respect, we express our deep sense of gratitude and sincere thanks to our guide **Ms. K. Kiruthika Devi, M.E., Assistant Professor**, Information Technology for his valuable guidance and suggestions throughout this project.

We are also thankful to our project coordinators **Dr. G. Sumathi, M.E., Ph.D., Professor** & **Ms. R. Saktheeswari, M.E., Assistant Professor** for their continual support and assistance.

We thank our family and friends for their support and encouragement throughout the course of our graduate studies.

ABILASH K
ARUN KUMAR G
BHARATH KUMAR P

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
RL	Reinforcement Learning
ANN	Artificial Neural Network
VSA	Visual Semantic Alignment
RCNN	Region-based Convolutional Neural Network
LRCN	Long-term Recurrent Convolutional Network
NIC	Neural Image Caption
VGG	Visual Geometry Group
YOLO	You Only Look Once
TTS	Text To Speech
ICG	Image Caption Generator
API	Application Programming Interface

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Visual impairment, also known as vision impairment or vision loss, is a decreased ability to see to a degree that causes problems not fixable by usual means, such as glasses. According to the World Health Organization, 285 million people are visually impaired worldwide, including over 39 million blind people. Living with visual impairment can be challenging, since many daily-life situations are difficult to understand without good visual acuity.

Technology has the potential to significantly improve the lives of visually impaired people. Access technology such as screen readers, screen magnifiers, and refreshable Braille displays enable the blind to use mainstream computer applications and mobile phones giving them access to previously inaccessible information. Another such technology that could improve the lives of the visually impaired is image caption generation. Most modern mobile phones are able to capture photographs, making it possible for the visually impaired to make images of their surroundings. These images can be used to generate captions that can be read out loud to give visually impaired people a better understanding of their surroundings. Image caption generation can also make the web more accessible to visually impaired people. The last decade has seen the triumph of the rich graphical desktop, replete with colourful icons, controls, buttons, and images. Automated caption generation of online images can make the web a more inviting place for visually impaired surfers.

Being able to automatically describe the content of an image using properly formed English sentences is a very challenging task. This task is significantly harder, for example, than the well-studied image classification or object recognition tasks, which have been a main focus in the computer vision community. Indeed, a description must capture not only the objects contained in an image, but it also must express how these objects relate to each other as well as their attributes and the activities they are involved in. Moreover, the above semantic knowledge has to be expressed in a natural language like English, which means that a language model is needed in addition to visual understanding.

Image captioning aims to describe the objects, actions, and details present in an image using natural language. Most image captioning research has focused on single-sentence captions, but the descriptive capacity of this form is limited; a single sentence can only describe in detail a small aspect of an image. Recent work has argued instead for image paragraph captioning with the aim of generating a (usually 5-8 sentence) paragraph describing an image. Compared with single-sentence captioning, paragraph captioning is a relatively new task. The main paragraph captioning dataset is the Visual Genome corpus, introduced by Krause (2016). When strong single-sentence captioning models are trained on this dataset, they produce repetitive paragraphs that are unable to describe diverse aspects of images. The generated paragraphs repeat a slight variant of the same sentence multiple times, even when beam search is used.

Likewise, different methods are used for paragraph generation, they are: Long-Term Recurrent Convolutional Network: The input can be an image or sequence of images from a video frame. The input is given to CNN which recognizes the activities in the image and a vector representation is formed and given to the LSTM model where a word is generated and a caption is obtained. Visual Paragraph Generation: This method implies to give a coherent and detailed generation of paragraph. Few semantic regions are detected in an image using attention model and sentences are generated one after the other and a paragraph is generated.

RNN: A recurrent neural network is a neural network that is specialized for processing a sequence of data with time stamp index t from 1 to t. For tasks that involve sequential inputs, such as speech and language, it is often better to use RNNs. In a NLP problem if you want to predict the next word in a sentence it is important to know the words before it.

GRU: The gated recurrent unit is relatively recent development proposed by Cho et al. Similar to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. Gated Recurrent Unit (GRU) calculates two gates called update and reset gates which control the flow of information through each hidden unit. Each hidden state at time-step t is computed using the following equations: Update gate formula, Reset gate formula, new memory formula, Final memory formula. The update gate is calculated from the current input and the hidden state of previous time step. This gate controls how much of portions of new memory and old memory should be combined in the final memory. Similarly, the reset gate is calculated but with different set of weights. It controls the balance between previous memory and the new input information in the new memory.

1.2 MOTIVATION

Generating captions for images is a vital task relevant to the area of both Computer Vision and Natural Language Processing. Mimicking the human ability of providing descriptions for images by a machine is itself a remarkable step along the line of Artificial Intelligence. The main challenge of this task is to capture how objects relate to each other in the image and to express them in a natural language (like English). Traditionally, computer systems have been using pre-defined templates for generating text descriptions for images. However, this approach does not provide sufficient variety required for generating lexically rich text descriptions. This shortcoming has been suppressed with the increased efficiency of neural networks. Many states of art models use neural networks for generating captions by taking image as input and predicting next lexical unit in the output sentence.

1.3 NATURAL LANGUAGE PROCESSING

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken. NLP is a component of artificial intelligence (AI). The development of NLP applications is challenging because computers traditionally require humans to "speak" to them in a programming language that is precise, unambiguous and highly structured, or through a limited number of clearly enunciated voice commands. Human speech, however, is not always precise - it is often ambiguous and the linguistic structure can depend on many complex variables, including slang, regional dialects and social context.

1.4 MACHINE LEARNING

Machine Learning is an idea to learn from examples and experience, without being explicitly programmed. Instead of writing code, you feed data to the generic algorithm, and it builds logic based on the data given. Machine Learning is a field which is raised out of Artificial Intelligence (AI). Applying AI, developers wanted to build better and intelligent machines. But except for few mere tasks such as finding the shortest path between point A and B, developers were unable to program more complex and constantly evolving challenges. There was a realisation that the only way to be able to achieve this task was to let machine learn from itself. This sounds similar to a child learning from its self. So, machine learning was developed as a new capability for computers. And now machine learning is present in so many segments of technology.

Types of machine learning Algorithms:

1. Supervised learning
2. Unsupervised Learning
3. Reinforcement Learning

1.4.1 Supervised Learning

Supervised learning is the most popular paradigm for machine learning. It is the easiest to understand and the simplest to implement. It is the task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labelled training data consisting of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyses the training data and produces an inferred function, which can be used for mapping new examples. Supervised Learning is very similar to teaching a child with the given data and that data is in the form of examples with labels, a learning algorithm can be fed with these example-label pairs one by one, allowing the algorithm to predict the right answer or not. Over time, the algorithm will learn to approximate the exact nature of the relationship between examples and their labels. When fully trained, the supervised learning algorithm will be able to observe a new, never-before-seen example and predict a good label for it.

Most of the practical machine learning uses supervised learning. Supervised learning is where you have input variable (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(x)$$

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for the data. It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. Supervised learning is often described as task oriented. It is highly focused on a singular task, feeding more and more examples to the algorithm until it can accurately perform on that task. This is the learning type that you will most likely encounter, as it is exhibited in many of the common applications like Advertisement Popularity, Spam Classification, face recognition**.**

Two types of Supervised Learning are:

1. Regression:

Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Regression can be used to estimate/ predict continuous values (Real valued output). For example**,** given a picture of a person and predicting the age on the basis of the given picture.

2. Classification:

Classification means to group the output into a class. If the data is discrete or categorical then it is a classification problem. For example**,** given data about the sizes of houses in the real estate market, making our output about whether the house "sells for more or less than the asking price" i.e., Classifying houses into two discrete categories.

1.4.2 Unsupervised Learning

Unsupervised Learning is a machine learning technique, where you do not need to supervise the model. Instead, you need to allow the model to work on its own to discover information. It mainly deals with the unlabelled data and looks for previously undetected patterns in a data set with no pre-existing labels and with a minimum of human supervision. In contrast to supervised learning that usually makes use of human-labelled data, unsupervised learning, also known as self-organization, allows for modelling of probability densities over inputs.

Unsupervised machine learning algorithms infer patterns from a dataset without reference to known, or labelled outcomes. It is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns, and differences without any prior training of data. Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore, machine is restricted to find the hidden structure in unlabelled data by our-self. For example, if some pictures of dogs and cats is provided to the machine to categorized, then initially the machine has no idea about the features of dogs and cats so it categorizes them according to their similarities, patterns and differences. The Unsupervised Learning algorithms allows you to perform more complex processing tasks compared to supervised learning. Although, unsupervised learning can be more unpredictable compared with other natural learning methods.

Unsupervised learning problems are classified into two categories of algorithms:

- **Clustering:** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behaviour.
- **Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

1.4.3 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences. Machine mainly learns from past experiences and tries to perform best possible solution to a certain problem. It is the training of machine learning models to make a sequence of decisions. Though both supervised and reinforcement learning use mapping between input and output, unlike supervised learning where the feedback provided to the agent is correct set of actions for performing a task, reinforcement learning uses rewards and punishments as signals for positive and negative behaviour. Reinforcement learning is currently the most effective way to hint machine's creativity.

1.4.4 Neural Networks

Neural Network (or Artificial Neural Network) has the ability to learn by examples. ANN is an information processing model inspired by the biological neuron system. ANN biologically inspired simulations that are performed on the computer to do a certain specific set of tasks like clustering, classification, pattern recognition etc. It is composed of a large number of highly interconnected processing elements known as the neuron to solve problems. It follows the non-linear path and process information in parallel throughout the nodes. A neural network is a complex adaptive system. Adaptive means it has the ability to change its internal structure by adjusting weights of inputs.

Artificial Neural Networks can be best viewed as weighted directed graphs, where the nodes are formed by the artificial neurons and the connection between the neuron outputs and neuron inputs can be represented by the directed edges with weights. The ANN receives the input signal from the external world in the form of a pattern and image in the form of a vector. These inputs are then mathematically designated by the notations x(n) for every n number of inputs. Each of the input is then multiplied by its corresponding weights (these weights are the details used by the artificial neural networks to solve a certain problem). These weights typically represent the strength of the interconnection amongst neurons inside the artificial neural network. All the weighted inputs are summed up inside the computing unit. If the weighted sum equates to zero, a bias is added to make the output non-zero or else to scale up to the system’s response. Bias has the weight and the input to it is always equal to 1. Here the sum of weighted inputs can be in the range of 0 to positive infinity. To keep the response in the limits of the desired values, a certain threshold value is benchmarked. And then the sum of weighted inputs is passed through the activation function. The activation function is the set of transfer functions used to get the desired output of it. There are various flavours of the activation function, but mainly either linear or non-linear set of functions. Some of the most commonly used set of activation functions are the Binary, Sigmoid (linear) and Tan hyperbolic sigmoidal (non-linear) activation functions.

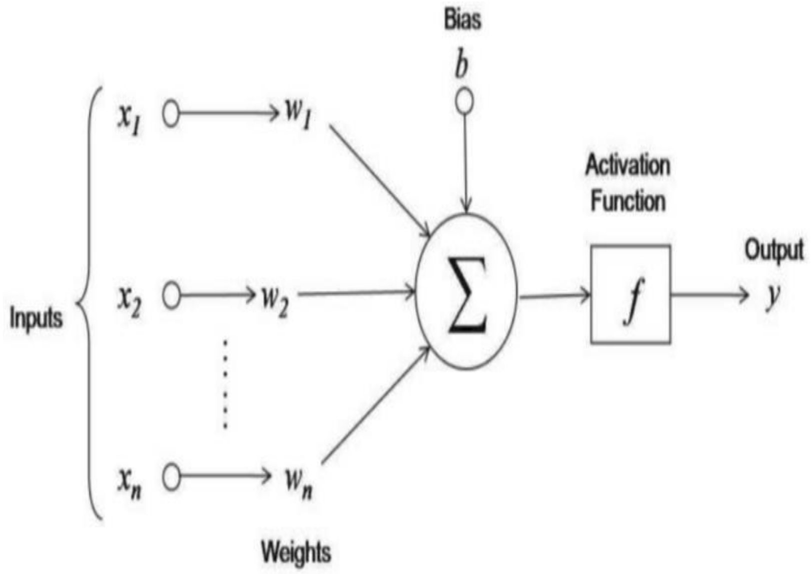


Figure 1.1 Neuron model

The Artificial Neural Network contains three layers

1. **Input Layer:** The input layers contain those artificial neurons (termed as units) which are to receive input from the outside world. This is where the actual learning on the network happens or corresponding happens else it will process.

2. **Hidden Layer:** The hidden layers are mentioned hidden in between input and the output layers. The only job of a hidden layer is to transform the input into something meaningful that the output layer/unit can use in some way. Most of the artificial neural networks are all interconnected, which means that each of the hidden layers is individually connected to the neurons in its input layer and also to its output layer leaving nothing to hang in the air. This makes it possible for a complete learning process and also learning occurs to the maximum when the weights inside the artificial neural network get updated after each iteration.

3. **Output Layer:** The output layers contain units that respond to the information that is fed into the system and also whether it learned any task or not.

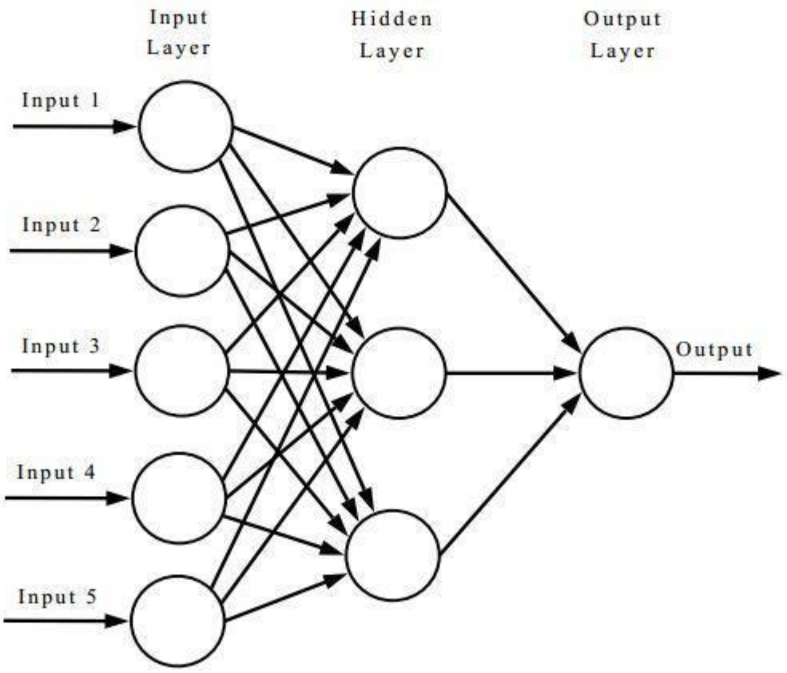


Figure 1.2 Basic Neural Network

Learning process of a neural network:

1. Start with values (often random) for the network parameters (*wij* weights and *bj* biases).
2. Take a set of examples of input data and pass them through the network to obtain their prediction.
3. Compare these predictions obtained with the values of expected labels and calculate the loss with them.
4. Perform the backpropagation in order to propagate this loss to each and every one of the parameters that make up the model of the neural network.
5. Use this propagated information to update the parameters of the neural network with the gradient descent in a way that the total loss is reduced, and a better model is obtained.
6. Continue iterating in the previous steps a good model is achieved.

1.4.5 DEEP LEARNING

Deep learning is a branch of machine learning which is completely based on artificial neural networks. Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabelled. It has a greater number of hidden layers and known as deep neural learning or deep neural network.

Deep learning has evolved hand-in-hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world. This data, known simply as big data, is drawn from sources like social media, internet search engines, ecommerce platforms, and online cinemas, among others. This enormous amount of data is readily accessible and can be shared through fintech applications like cloud computing. However, the data, which normally is unstructured, is so vast that it could take decades for humans to comprehend it and extract relevant information. Companies realize the incredible potential that can result from unravelling this wealth of information and are increasingly adapting to AI systems for automated support. Deep learning learns from vast amounts of unstructured data that would normally take humans decades to understand and process.

Deep learning and utilizes a hierarchical level of artificial neural networks to carry out the process of machine learning. The artificial neural networks are built like the human brain, with neuron nodes connected like a web. While traditional programs build analysis with data in a linear way, the hierarchical function of deep learning systems enables machines to process data with a nonlinear approach.

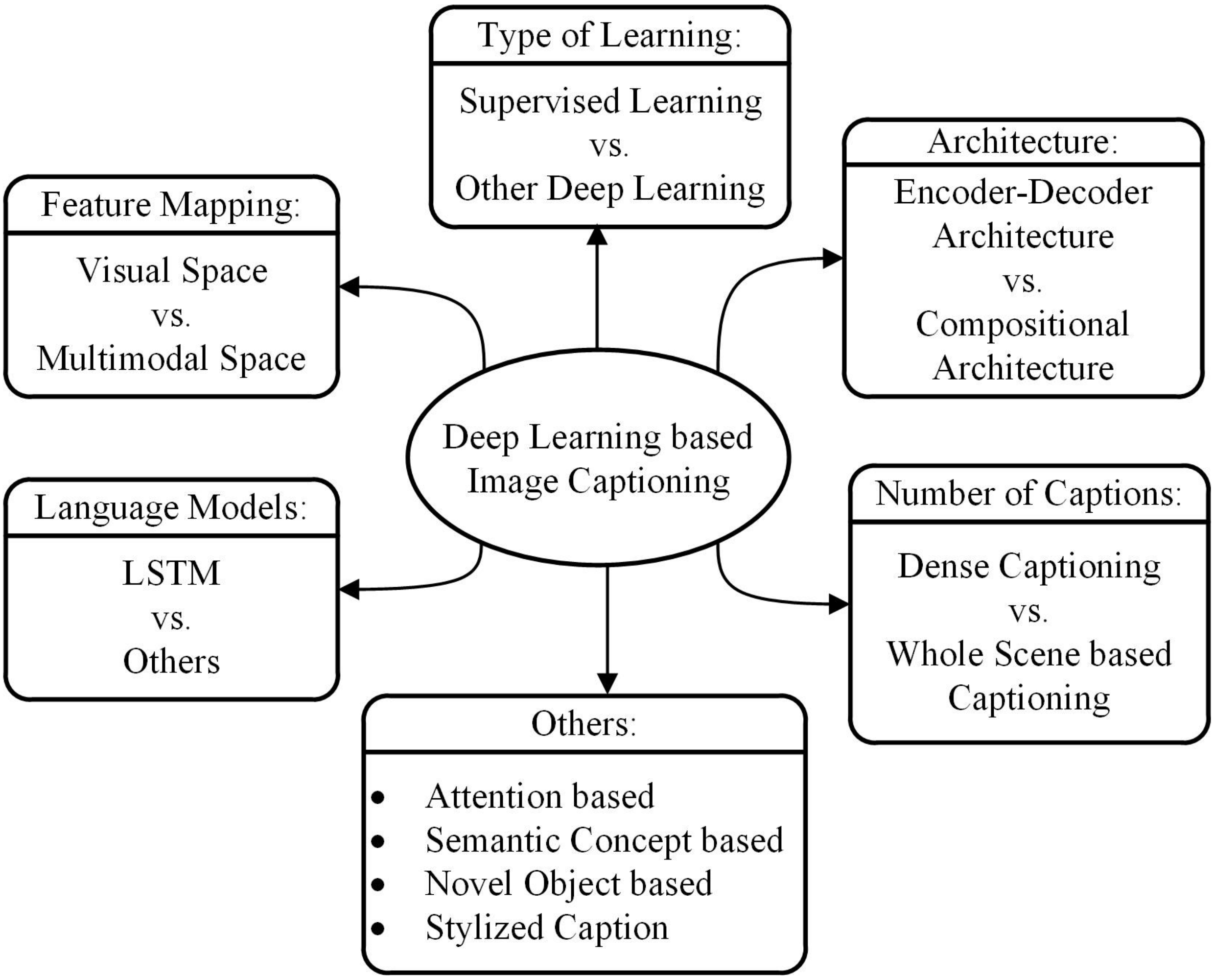


Figure 1.3 An overall taxonomy of deep-learning based image captioning

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

This chapter presents a detailed literature survey of all the existing work carried so far.

2.2 LITERATURE SURVEY

2.2.1 Deep Visual-Semantic Alignments for Generating Image Descriptions

Andrej Karpathy and Li Fei-Fei proposed a visual-semantic alignment (VSA) method. The method generates descriptions of different regions of an image in the form of words or sentences. Technically, the method replaces the CNN with Region-based convolutional Networks (RCNN) so that the extracted visual features are aligned to particular regions of the image. The experiment shows that the generated descriptions significantly outperform retrieval baselines on both full images and on a new dataset of region-level annotations.

MERITS:

- This method generates descriptions of different regions of an image in the form of words or sentences.
- This method generates more diverse and accurate descriptions than the whole image method such as the LRCN and NIC.

DEMERITS:

- The limitation is that the method consists of two separate models VSA method and RCNN method.

2.2.2 An Image Caption Method based on Object Detection

Danyang Cao, Menggui Zhu, Lei Gao proposed a method of generating image caption based on object detection. The object detection algorithm is used to extract image feature, only the features of meaningful regions in the image are used, and then image caption is generated by combining the spatial attention mechanism with the caption generation network. The image feature of the object region and the salient region are sufficient to represent the information of the entire image in the image caption task. For better convergence of the model, they use a new strategy for model training.

MERITS:

- An adaptive attention mechanism can decide "when" and "where" to pay attention which has high flexibility.

DEMERITS:

- However, low-level visual features have been neglected, which is also beneficial for understanding the images.

2.2.3 From Captions to Visual Concepts and Back

Hao Fang, Saurabh Gupta and Forrest Iandola presented a visual concepts-based method instead of end-to-end learning. First, they used multiple instances learning to train visual detectors of words that commonly occur in captions such as nouns, verbs, and adjectives. Then, they trained a language model with a set of over 400,000 image descriptions to capture the statistics of word usage. Finally, they re-ranked caption candidates using sentence-level features and a deep multi-modal similarity model.

MERITS:

- The captions have equal or better quality 34% of the time than those written by human beings.

DEMERITS:

- The limitation of the method is that it has more human controlled parameters which make the system less re-producible.

2.2.4 Long-term Recurrent Convolutional Networks for Visual Recognition and Description

Jeffery Donahue, Lisa Anne Hendricks and Sergio Guadarrama developed a model based on deep convolutional networks. They describe a class of recurrent convolutional architectures which is end-to-end trainable and suitable for large-scale visual understanding tasks, and demonstrate the value of these models for activity recognition, image captioning, and video description. Learning long-term dependencies is possible when nonlinearities are incorporated into the network state updates. Their recurrent sequence models are directly connected to modern visual convolutional network models and can be jointly trained to learn temporal dynamics and convolutional perceptual representations.

MERITS:

- Recurrent convolutional architecture suitable for large-scale visual learning.

DEMERITS:

- The limitation is the difficulty of understanding the intermediate result.

2.2.5 Knowing when to look: Adaptive Attention via a Visual Sentinel for Image Captioning

Jiasen Lu, Caiming Xiong, Devi Parikh and Richard Socher proposed an Attention-based neural encoder-decoder frameworks for Image Captioning. They propose a novel adaptive attention model with a visual sentinel. At each time step, their model decides whether to attend to the image (and if so, to which regions) or to the visual sentinel. The model decides whether to attend to the image and where, in order to extract meaningful information for sequential word generation.

MERITS:

- If the according word is a visual word, the attention weight will larger than the weights for attending word features.

DEMERITS:

- Although this method utilizes visual words to adaptively attend image features and word features, the attention layers are not constrained directly by the loss function, either.

2.2.6 Neural Image Caption Generation with Visual Attention

Kelvin Xu, Jimmy Ba and Riyan Kiros introduced an attention-based model that automatically learns to describe the content of images. They describe how they can train the model in a deterministic manner using standard backpropagation techniques and stochastically by maximizing a variational lower bound. They also show through visualization how the model is able to automatically learn to fix its gaze on salient objects while generating the corresponding words in the output sequence.

MERITS:

- This method can generate image captions that are semantically more accurate.

DEMERITS:

- However, these methods have problems in identifying prominent objects of the image.

2.2.7 Collective Generation of Natural Image Descriptions

Kuznetsova, Ordonez and Berg presented a holistic data-driven approach to image description generation, exploiting the vast amount of (noisy) parallel image data and associated natural language descriptions available on the web. More specifically, given a query image, we retrieve existing human-composed phrases used to describe visually similar images, then selectively combine those phrases to generate a novel description for the query image. They cast the generation process as constraint optimization problems, collectively incorporating multiple interconnected aspects of language composition for content planning, surface realization and discourse structure.

MERITS:

- This method produces general and syntactically correct captions.
- The generated caption is more like a human-generated language than the template-based method.

DEMERITS:

- They cannot generate image specific and semantically correct captions.

2.2.8 Generating Image Descriptions from Computer Vision Detections

Mitchell, Han and Dodge proposed a novel generation system that composes human-like descriptions of images from computer vision detections. The first step of this method is to detect the object, attributes, and actions and scenes in the image, after obtaining the information, the information is filled into a fixed sentence template. They exploited syntactic trees to create a data-driven model. By leveraging syntactically informed word co-occurrence statistics, the generator filters and constrains the noisy detections output from a vision system to generate syntactic trees that detail what the computer vision system sees.

MERITS:

- Can generate grammatically correct captions.
- Template-based methods are simple and intuitive.

DEMERITS:

- Heavily hand designed and unexpressive which are not flexible enough to generate meaningful sentences.
- Templates are predefined and cannot generate variable-length captions.

2.2.9 A Neural Image Caption Generator

Vinyals, Toshev and Bengio proposed a neural image caption (NIC) model only for the image caption generation. Combining the GoogLeNet and single layer of LSTM, this model is trained to maximize the likelihood of the target description sentence given the training images. The performance of the model is evaluated qualitatively and quantitatively. This method was ranked first in the MS COCO Captioning Challenge (2015) in which the result was judged by humans.

MERITS:

- Effective from the traditional RNN by overcoming the limitations of RNN which had short term memory.

DEMERITS:

- Not enough attention to the different objects in the image.

2.2.10 Generating Visually Descriptive Language from Object Layouts

Yin and Ordonez suggested a sequence-to-sequence model in which an LSTM network encodes a series of objects and their positions as an input sequence and an LSTM language model decodes this representation to generate captions. Their model uses the YOLO object detection model to extract object layouts from images (object categories and locations) and increase the accuracy of captions. They also present a variation that uses the VGG image classification model pre-trained on ImageNet to extract visual features. The encoder at each time step takes as input a pair of object category (encoded as a one-hot vector), and the location configuration vector that contains the left-most position, top-most position, width and height of the bounding box corresponding to the object, all normalized.

MERITS:

- Increased accuracy of captions.

DEMERITS:

- The model is trained with back-propagation, but the error is not propagated to the object detection model.

CHAPTER 3

SYSTEM DESIGN

3.1 PROPOSED WORK

The proposed work aims to build a working model of Image caption generator by implementing Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM). The Image features will be extracted from Xception which is a CNN model trained on the ImageNet dataset and then the features are fed as input to the LSTM model which will be responsible for generating the image captions. The image captions will be converted to a computer speech by the Text-to-Speech model.

3.1.1 Module Description

The project is divided into three modules as listed below. Figure 3.1 represents the architecture diagram of the proposed system.

1. Convolutional Neural Network
2. Long Short-Term Memory
3. Text-to-Speech model

3.1.2 Architecture Diagram

- An image is fed as input to Convolutional Neural Network which is trained on ImageNet dataset. The CNN extracts the features of the image and then feeds it to the Long Short Term Memory model.
- The LSTM model combines the features extracted from CNN to generate a caption for the image by sequence prediction technique. Based on the previous text the model can predict what the next word will be.

- The image caption generated from the image caption generated model will be converted into computer speech using text-to-speech model which will enable the person with visual disability to interpret the image.

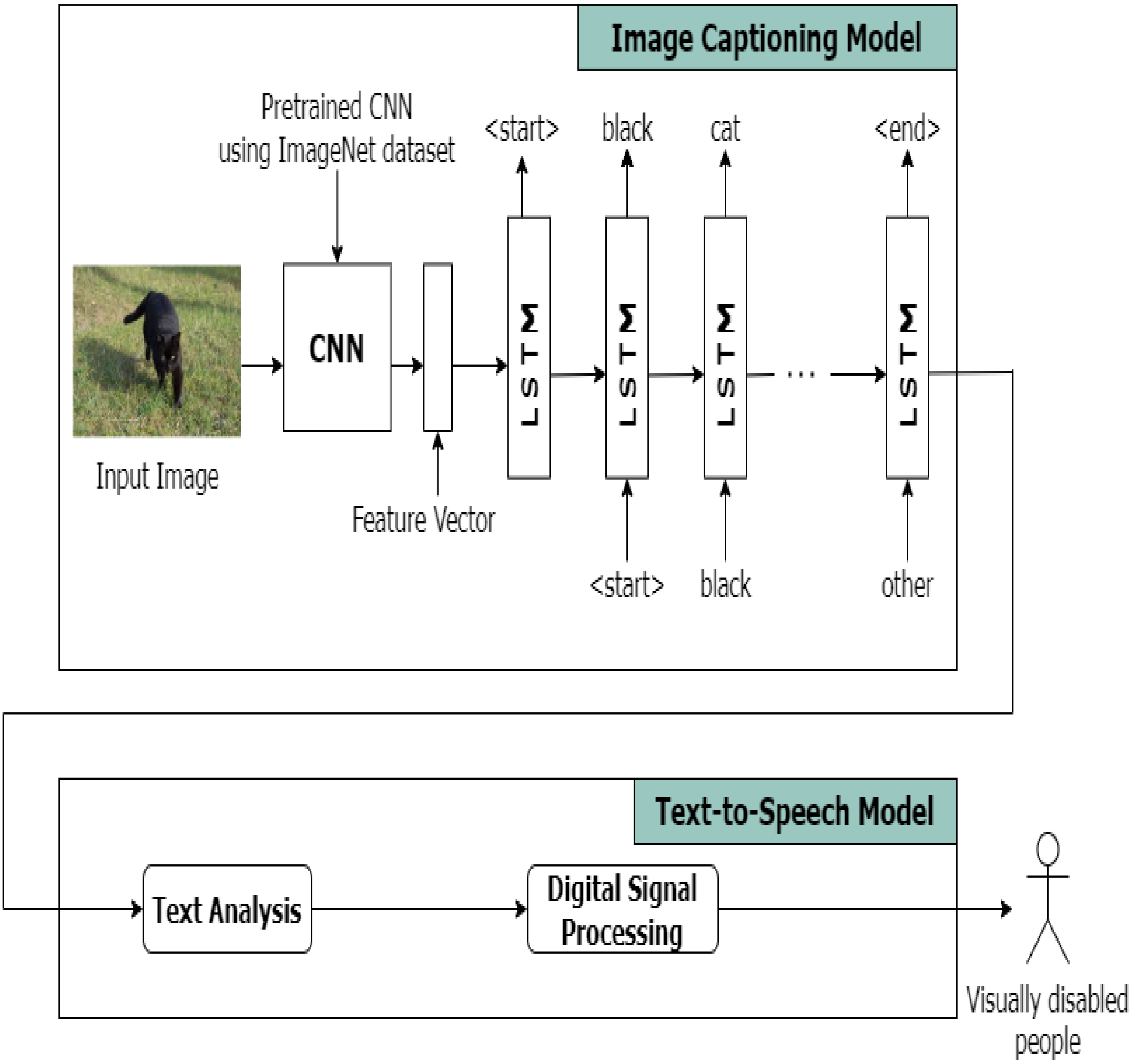


Figure 3.1 ICG and TTS Architecture

3.2 CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.

CNN is basically used for image classifications and identifying if an image is a bird, a plane or Superman, etc. It scans images from left to right and top to bottom to pull out important features from the image and combines the feature to classify images. It can handle the images that have been translated, rotated, scaled and changes in perspective.

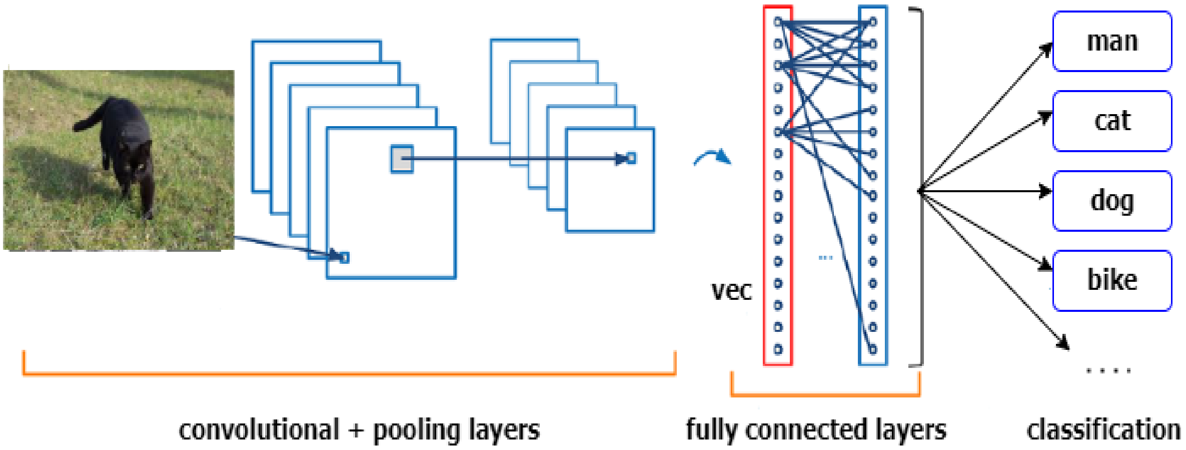


Figure 3.2 Working of CNN

3.2.1 Working of CNN

Pretrained CNN model Xception using ImageNet dataset is used in this project to extract image features. Xception is a convolutional neural network. Xception is the pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. The network has an image input size of 299X299. The Xception architecture has 36 convolutional layers forming the feature extraction base of the network. The data first goes through the entry flow, then through the middle flow which is repeated eight times, and finally through the exit flow.

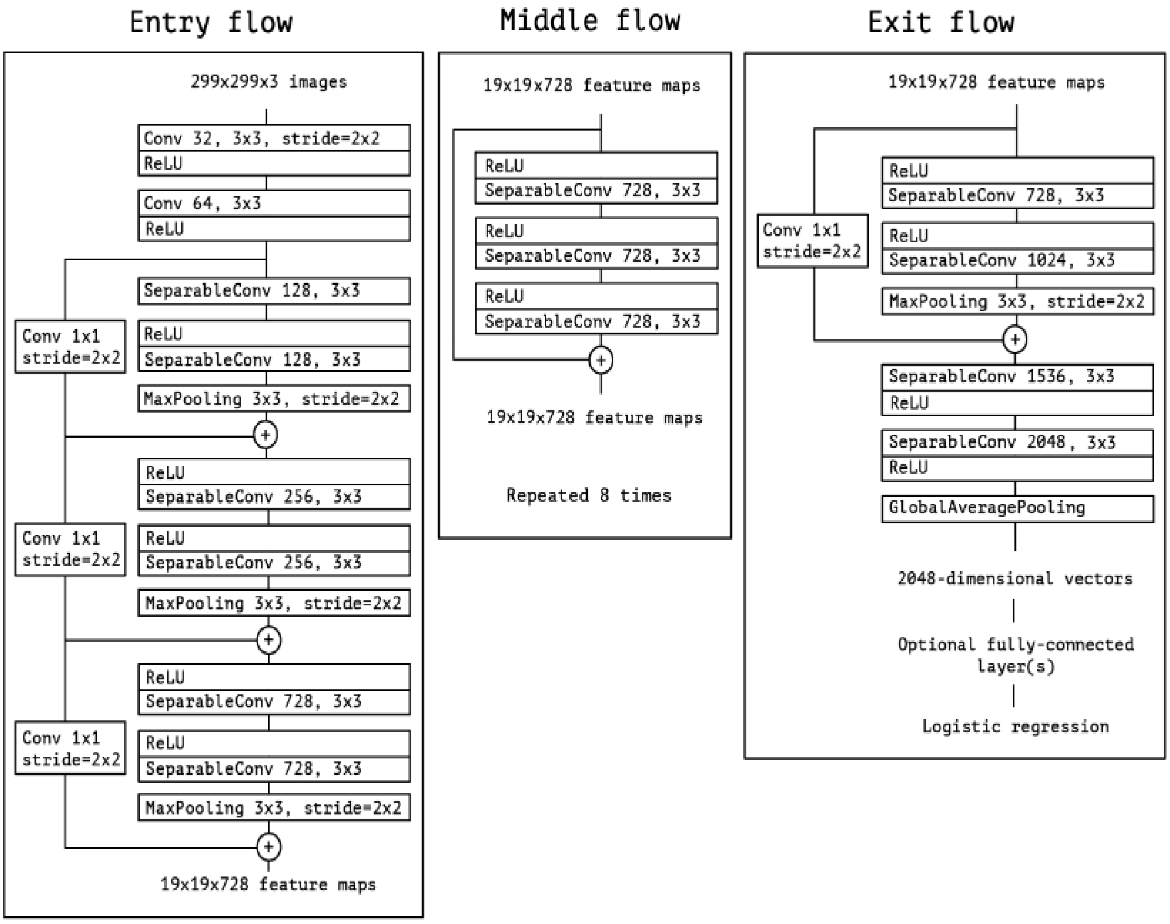


Figure 3.3 Xception model

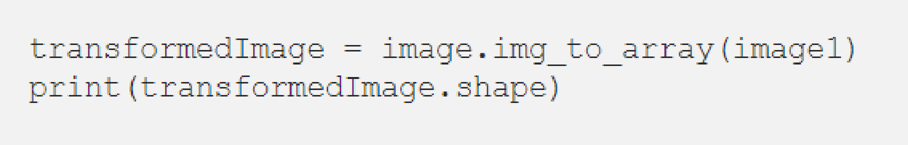
3.2.1.1 Create Xception model

```
model = Xception(
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    input_shape=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)
```

Arguments

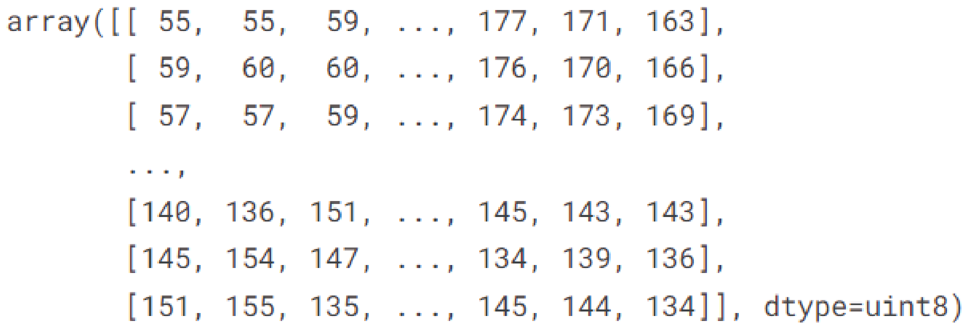
1. ***include_top***: whether to include the fully-connected layer at the top of the network.
2. ***weights***: one of None (random initialization), ‘imagenet’ (pre-training on ImageNet), or the path to the weights file to be loaded.
3. ***input_tensor***: optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model.
4. **input_shape**: optional shape tuple, only to be specified if include_top is False (otherwise the input shape has to be (299, 299, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 71. E.g. (150, 150, 3) would be one valid value.
5. **pooling**: Optional pooling mode for feature extraction when include_top is False.
6. ***None*** means that the output of the model will be the 4D tensor output of the last convolutional block.
7. ***avg*** means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.
8. ***max*** means that global max pooling will be applied.
9. ***classes***: optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified.
10. ***classifier_activation***: A str or callable. The activation function to use on the “top” layer. Set classifier_activation=None to return the logits of the “top” layer. When loading pretrained weights, classifier_activation can only be None or “softmax”.
11. ***Returns***: A keras.Model instance.

3.2.1.2 Transform Image into Array



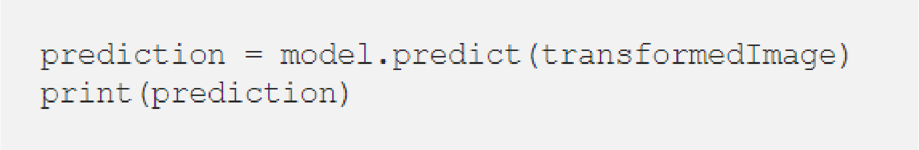
```
transformedImage = image.img_to_array(image1)
print(transformedImage.shape)
```

A computer sees an Image as an array of pixels. For example, if image size is 300 x 300. In this case, the size of the array will be 300x300x3. Where 300 is width, next 300 is height and 3 is RGB channel values. The computer is assigned a value from 0 to 255 to each of these numbers. This value describes the intensity of the pixel at each point. Therefore, the Image is transformed to an array.



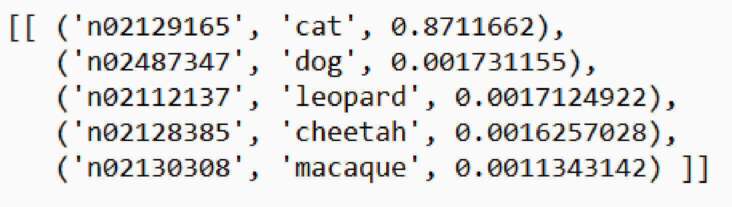
```
array([[ 55,  55,  59, ..., 177, 171, 163],
       [ 59,  60,  60, ..., 176, 170, 166],
       [ 57,  57,  59, ..., 174, 173, 169],
       ...,
       [140, 136, 151, ..., 145, 143, 143],
       [145, 154, 147, ..., 134, 139, 136],
       [151, 155, 135, ..., 145, 144, 134]], dtype=uint8)
```

3.2.1.3 Predict the Image



```
prediction = model.predict(transformedImage)
print(prediction)
```

The top 5 predictions are made using the Xception model.



```
[[ ('n02129165', 'cat', 0.8711662),
   ('n02487347', 'dog', 0.001731155),
   ('n02112137', 'leopard', 0.0017124922),
   ('n02128385', 'cheetah', 0.0016257028),
   ('n02130308', 'macaque', 0.0011343142) ]]
```

The Xception model has predicted that Image consists of **cat** with a probability of 0.87(87%) and other animals with a probability of less than 0.001(less than 1%).

3.3 LONG SHORT-TERM MEMORY

The LSTM (Long Short-Term Memory) layer is nothing but a specialized Recurrent Neural Network to process the sequence input. It is discovered by Hochhreiter and Schmidhuber. It is similar to RNN. It is majorly used for passing information from one cell to another cell and outputs a whole word. It consists of three gates, they are-an input gate, an output gate and a forget gate. The input gate receives a value and passes it to other cells. The forget gate is used for telling which extent the value is used further which is known by is past experiences. The output gate is used for getting all the values and producing the output as a word. It can also process a sequence of data such as video or speech but not just image.

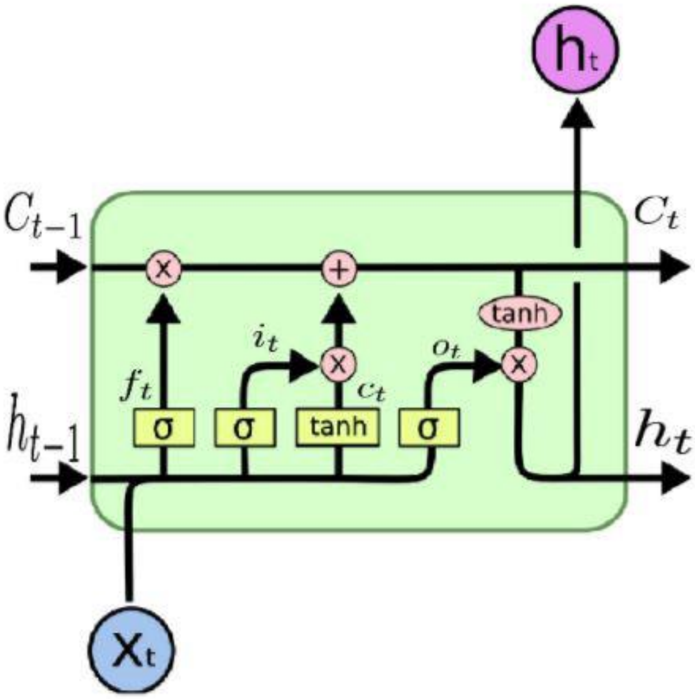


Figure 3.4 LSTM Cell Structure

LSTMs use gated cells to store information outside the regular flow of the RNN. With these cells, the network can manipulate the information in many ways, including storing information in the cells and reading from them. The cells are individually capable of making decisions regarding the information and can execute these decisions by opening or closing the gates. The ability to retain information for a long period of time gives LSTM the edge over traditional RNNs in these tasks. The chain-like architecture of LSTM allows it to contain information for longer time periods, solving challenging tasks that traditional RNNs struggle to or simply cannot solve.

The three major parts of the LSTM include:

Forget Gate - removes information that is no longer necessary for the completion of the task. This step is essential to optimizing the performance of the network.

Input Gate – responsible for adding information to the cells.

Output Gate – selects and outputs necessary information.

3.3.1 Working of LSTM

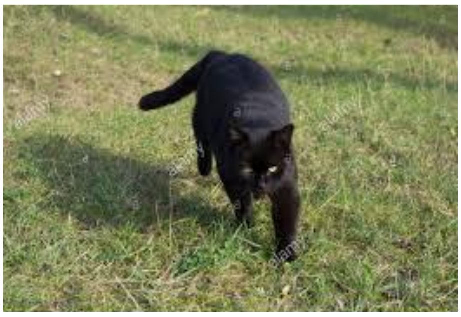


Figure 3.5 Test Image

Caption - the black cat is walking on grass. The vocabulary in the example was:

vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}

The caption is generated iteratively, one word at a time as follows:

3.3.1.1 Iteration 1

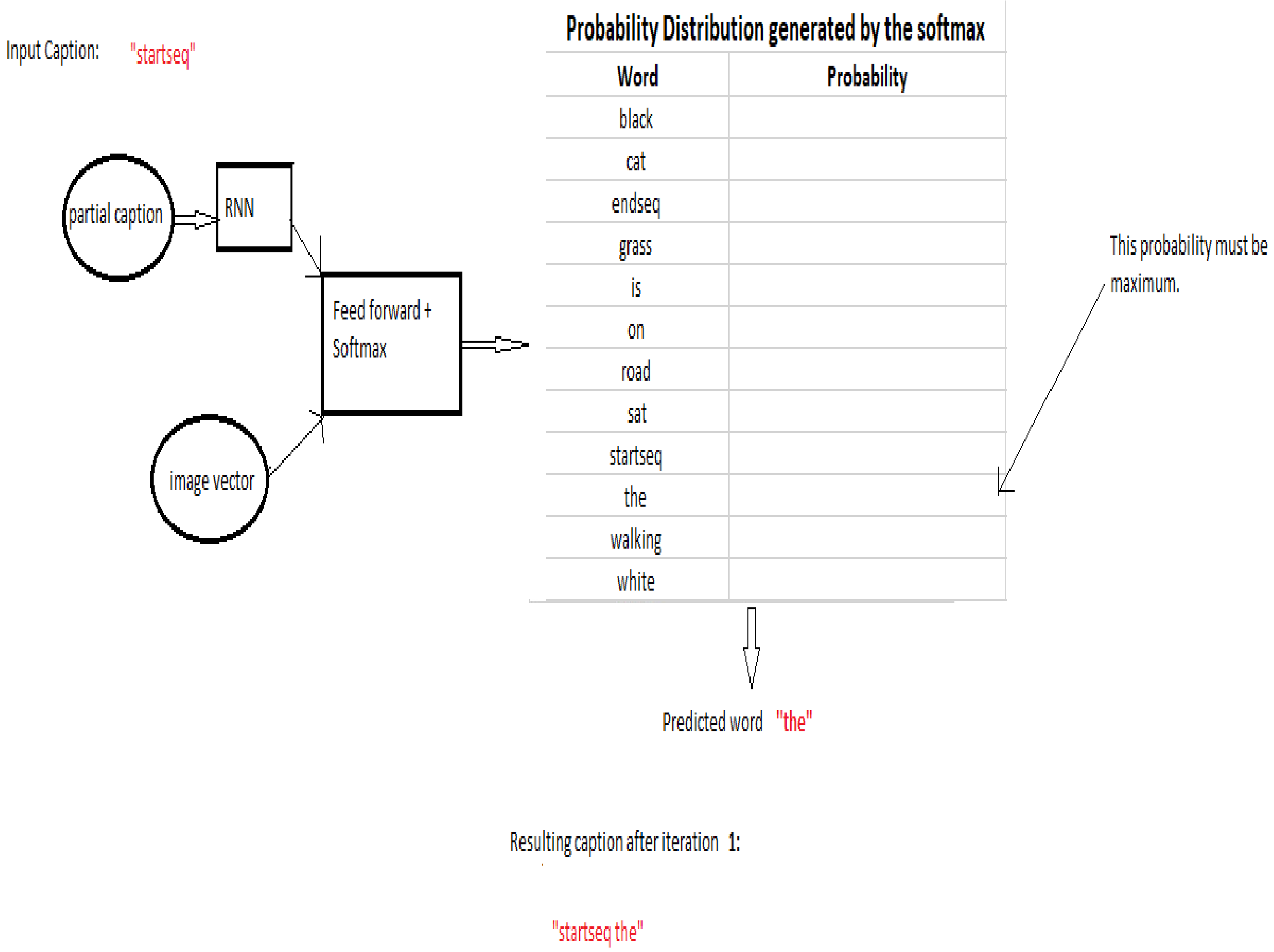
Input: Image vector + “startseq” (as partial caption)

Expected Output word: “the”

(The token ‘startseq’ is used as the initial partial caption for any image during inference).

The model generates a 12-long vector (in the sample example while 1652-long vector in the original example) which is a probability distribution across all the words in the vocabulary. For this reason, the word with the maximum probability is selected, given the feature vector and partial caption.

If the model is trained well, the probability for the word “the” should be maximum, this is called as Maximum Likelihood Estimation (MLE) i.e., the word which is most likely according to the model for the given input is selected. And sometimes this method is also called as Greedy Search.


Input Caption: "startseq"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "the"
Resulting caption after iteration 1:
"startseq the"

Probability Distribution generated by the softmax

Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.6 Iteration 1

3.3.1.2 Iteration 2

Input: Image vector + “startseq the”

Expected Output word: “black”

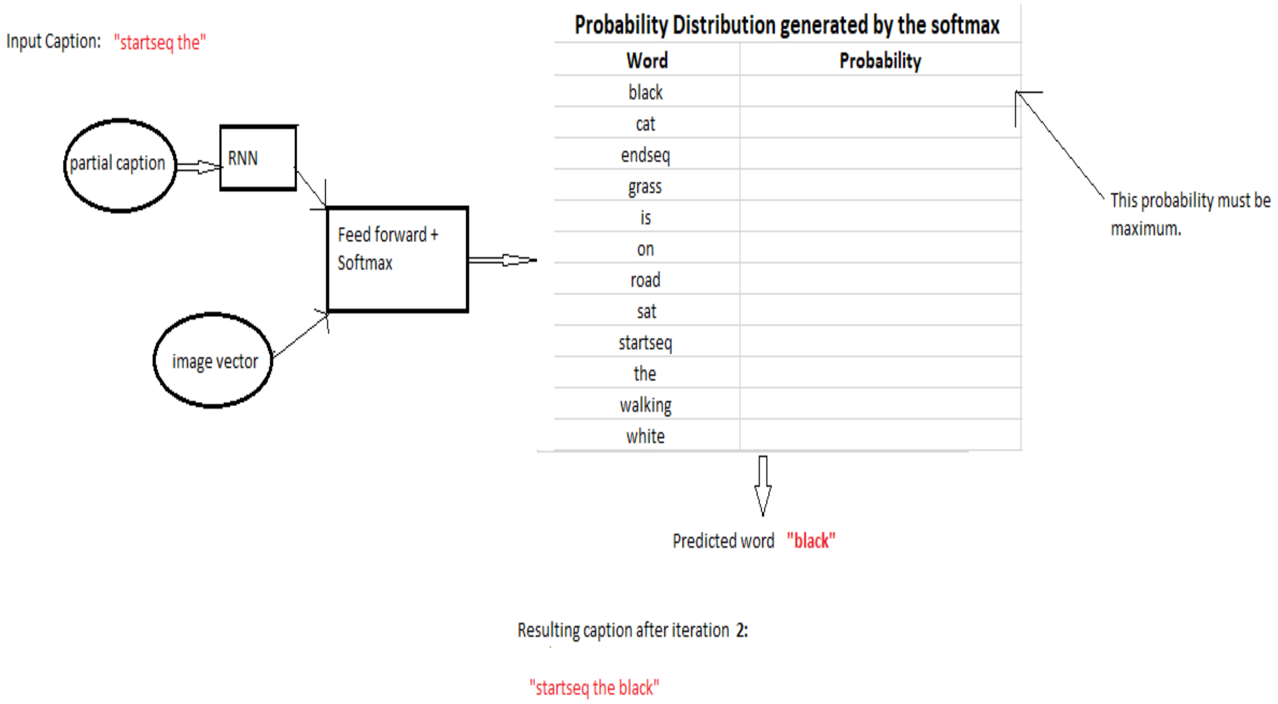


Figure 3.7 Iteration 2

3.3.1.3 Iteration 3

Input: Image vector + “startseq the black”

Expected Output word: “cat”

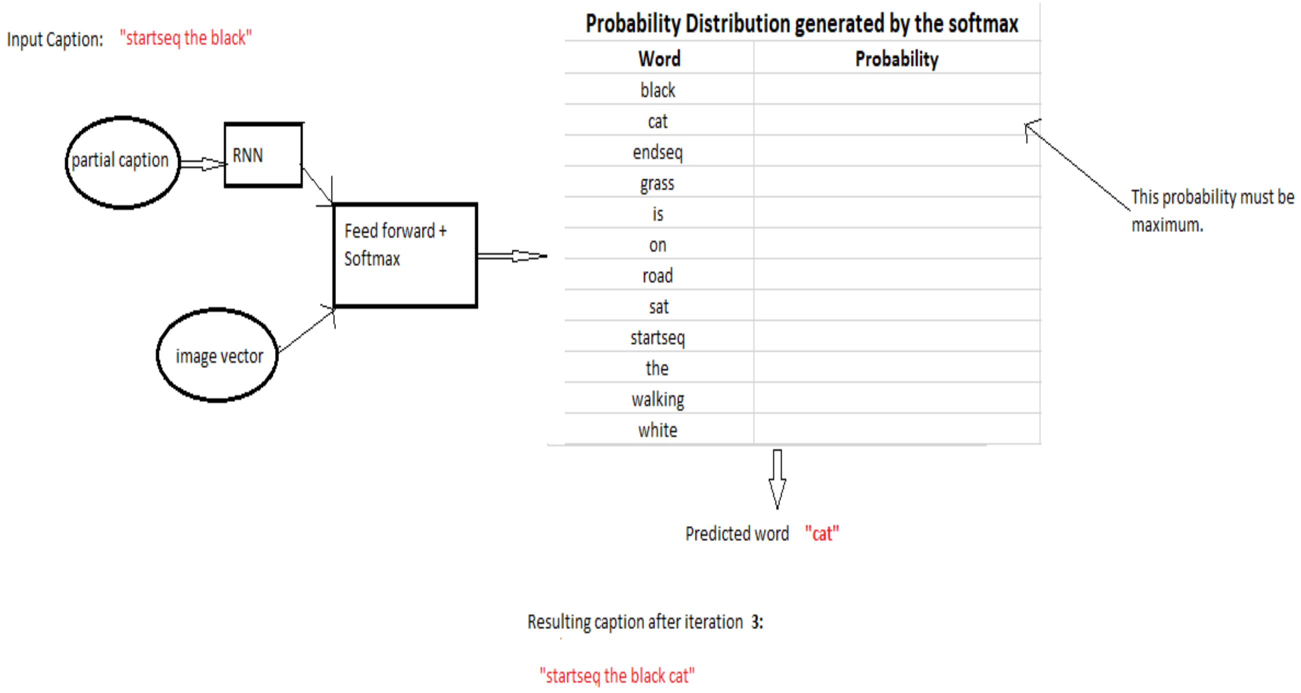
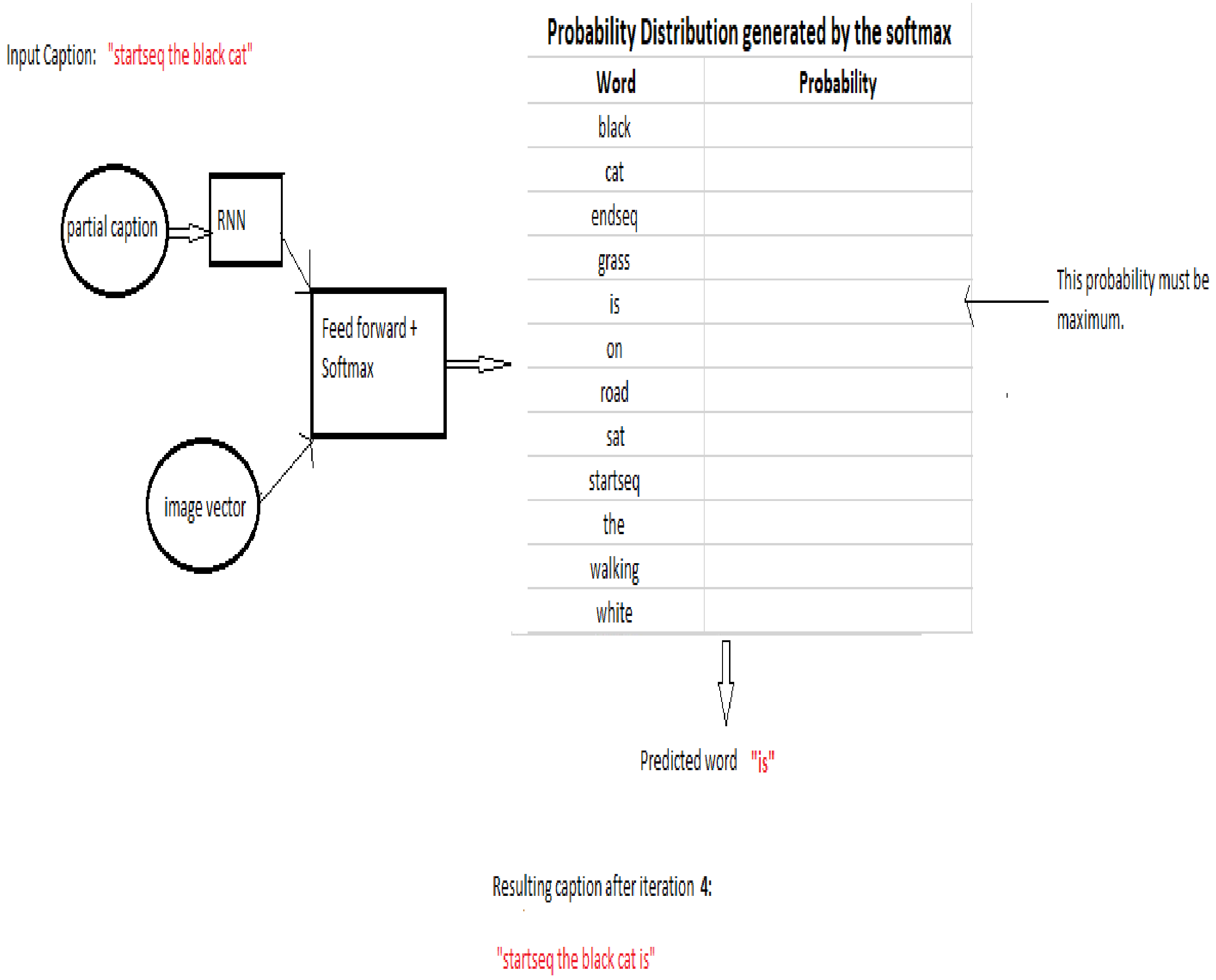


Figure 3.8 Iteration 3

3.3.1.4 Iteration 4

Input: Image vector + “startseq the black cat”

Expected Output word: “is”


Input Caption: "startseq the black cat"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "is"
Resulting caption after iteration 4:
"startseq the black cat is"

Probability Distribution generated by the softmax

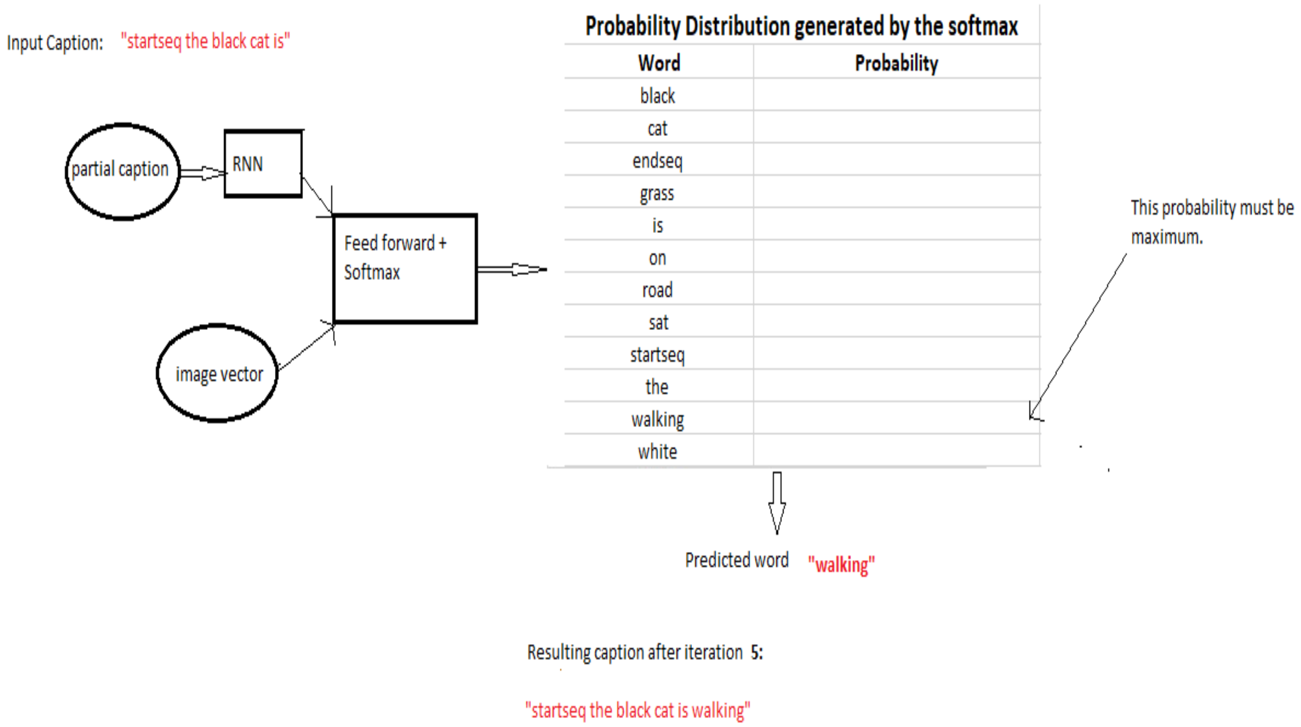
Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.9 Iteration 4

3.3.1.5 Iteration 5

Input: Image vector + “startseq the black cat is”

Expected Output word: “walking”


Input Caption: "startseq the black cat is"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "walking"
Resulting caption after iteration 5:
"startseq the black cat is walking"

Probability Distribution generated by the softmax

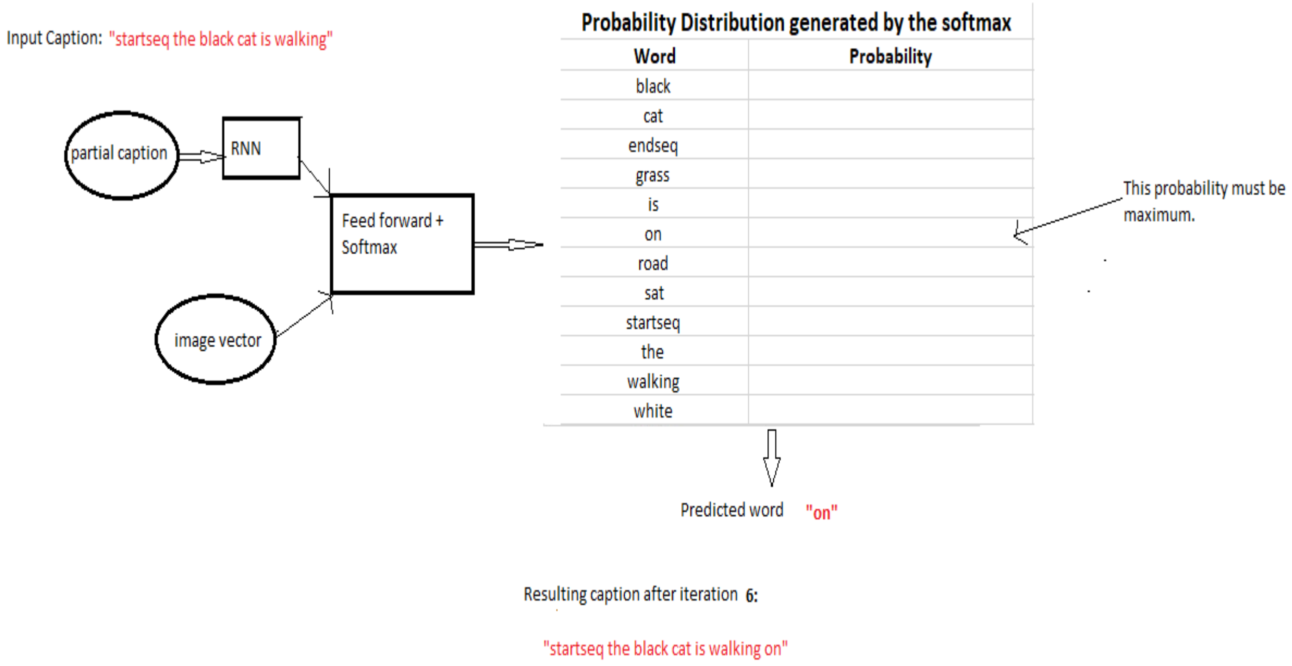
Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.10 Iteration 5

3.3.1.6 Iteration 6

Input: Image vector + “startseq the black cat is walking”

Expected Output word: “on”


Input Caption: "startseq the black cat is walking"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "on"
Resulting caption after iteration 6:
"startseq the black cat is walking on"

Probability Distribution generated by the softmax

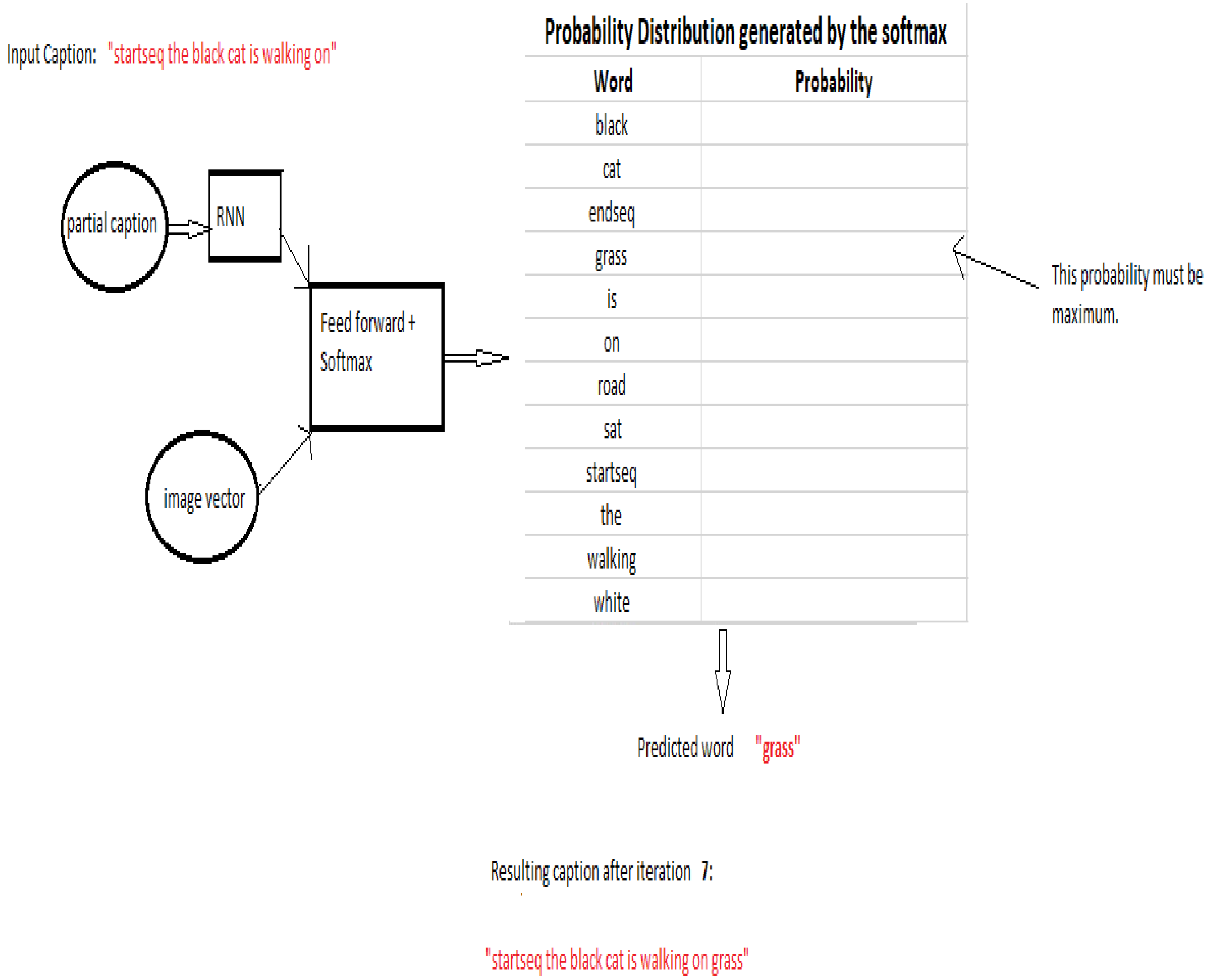
Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.11 Iteration 6

3.3.1.7 Iteration 7

Input: Image vector + “startseq the black cat is walking on”

Expected Output word: “grass”



Input Caption: "startseq the black cat is walking on"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "grass"
Resulting caption after iteration 7:
"startseq the black cat is walking on grass"

Probability Distribution generated by the softmax

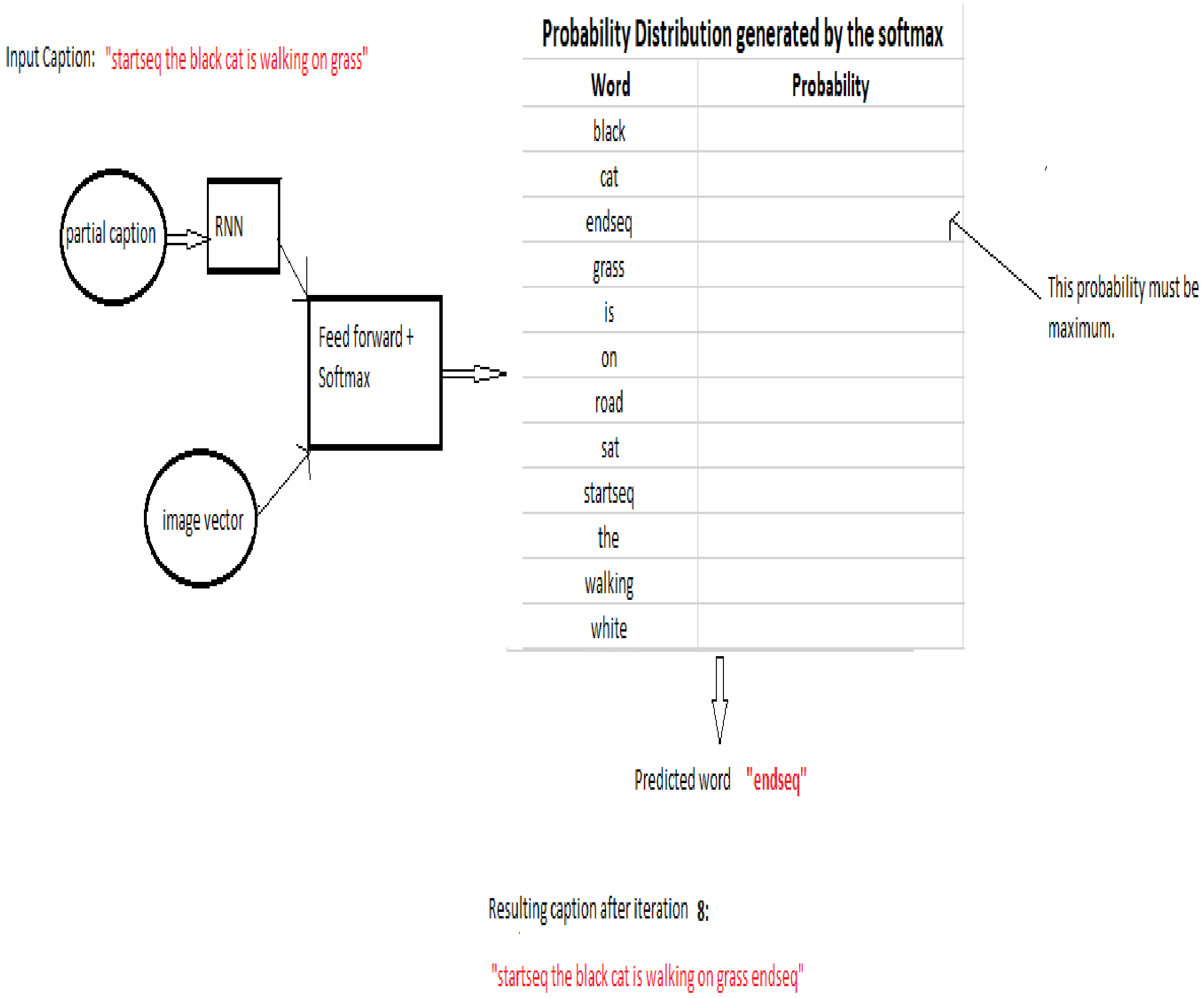
Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.12 Iteration 7

3.3.1.8 Iteration 8

Input: Image vector + “startseq the black cat is walking on grass”

Expected Output word: “endseq”



Input Caption: "startseq the black cat is walking on grass"
partial caption
RNN
Feed forward + Softmax
image vector
This probability must be maximum.
Predicted word "endseq"
Resulting caption after iteration 8:
"startseq the black cat is walking on grass endseq"

Probability Distribution generated by the softmax

Word	Probability
black	
cat	
endseq	
grass	
is	
on	
road	
sat	
startseq	
the	
walking	
white	

Figure 3.13 Iteration 8

This is where it stops the iterations.

So, it stops when either of the below two conditions is met:

- It encounters an ‘endseq’ token which means the model thinks that this is the end of the caption. (You should now understand the importance of the ‘endseq’ token)
- It reaches a maximum threshold of the number of words generated by the model.

3.4 TEXT-TO-SPEECH MODEL

Speech synthesis (TTS) is defined as the artificial production of human voices. The main use (and what induced its creation) is the ability to translate a text into spoken speech automatically. Unlike speech recognition systems that use phonemes (the smallest units of sound) in the first place to cut out sentences, TTS will be based on what are known as graphemes: the letters and groups of letters that transcribe a phoneme. This means that the basic resource is not the sound, but the text. This is usually done in two steps.

Speech synthesis can be found in a multitude of applications. However, it is important to know that this technology was originally designed to help people with disabilities (particularly visually impaired) in their daily lives. For example, the very famous Stephen Hawking, because of his severe disability, used a TTS to communicate with the people around him.

Since then, many cases of use have been developed more or less close to the original virtue of TTS. For example, as mentioned above in the context of transport, this technology is used to generate voices to transmit messages to passengers via voice, whether or not they are disabled. It is very easy today to find traces of TTS in our uses. Another example can be found in language translation engines. These are equipped with this technology to suggest the pronunciation of the translated information in order to complete the textual translation.

Another sector that integrates speech synthesis in embedded systems or cloud applications that continues to revolutionize uses is the broad field of IoT. Indeed, in a rapidly expanding universe, intelligent devices are increasingly equipped with TTS, on the one hand to improve the user experience and on the other hand to improve accessibility and the intelligence of the interfaces. A strong example that continues to progress is that of household appliances (also known as “appliances” in English), increasingly equipping consumer products and robots with voice.

In order to choose the right speech synthesis (text-to-speech), it is essential to take into account several criteria. These parameters are the following: the language spoken, the type of speaker, the quality of the voice and the supplier. With this information, it is easier to select the right solution that meets your needs and constraints. Indeed, not all companies offering TTS have equivalent ranges, so it is very important to source these partners well before you start. Next, the language and the type of voice are important criteria for the user experience proposed, there must be consistency between the voice interface and what it should inspire.

On the integration side, speech synthesis are technologies that are also based on the notions of cloud, embedded or hybrid (also known as "on-premise"). It should be remembered that embedded has technical limits in terms of sentence storage that a cloud will not have, but the embedded voice will work no matter what happens where the cloud needs a connection. These parameters are to be taken into account according to the nature of your projects, in transport for example it is recommended to use embedded to ensure a continuous service.

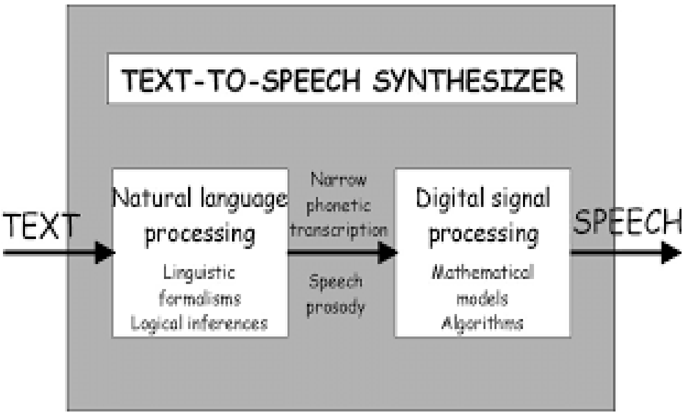

TEXT-TO-SPEECH SYNTHESIZER
TEXT
Natural language processing
Linguistic formalisms
Logical inferences
Narrow phonetic transcription
Speech prosody
Digital signal processing
Mathematical models
Algorithms
SPEECH

Figure 3.14 TTS model

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 SOFTWARE REQUIREMENTS

- Programming language: Python
- Operating system: Windows
- Tools: Anaconda Navigator/ TensorFlow
- Dataset: Flicker8k
- Editor: Jupyter Notebook

4.2 DATASET

Flickr8k dataset is a public benchmark dataset for image to sentence description. This dataset consists of 8000 images with five captions for each image. These images are extracted from diverse groups in Flickr website. Each caption provides a clear description of entities and events present in the image. The dataset depicts a variety of events and scenarios and doesn’t include images containing well-known people and places which makes the dataset more generic. The dataset has 6000 images in training dataset, 1000 images in development dataset and 1000 images in test dataset. Features of the dataset making it suitable for this project are:

- Multiple captions mapped for a single image makes the model generic and avoids overfitting of the model.
- Diverse category of training images can make the image captioning model to work for multiple categories of images and hence can make the model more robust.

4.3 PACKAGES

This project requires the following packages:

- TensorFlow
- Keras
- Pillow
- NumPy
- Tqdm
- Matplotlib

4.3.1 TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries, and community resources that lets researchers push the state-of-the-art in ML, and gives developers the ability to easily build and deploy ML-powered applications.

TensorFlow provides a collection of workflows with intuitive, high-level APIs for both beginners and experts to create machine learning models in numerous languages. Developers have the option to deploy models on a number of platforms such as on servers, in the cloud, on mobile and edge devices, in browsers, and on many other JavaScript platforms. This enables developers to go from model building and training to deployment much more easily.

The single biggest benefit TensorFlow provides for machine learning development is abstraction. Instead of dealing with the nitty-gritty details of implementing algorithms, or figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall application logic. TensorFlow takes care of the details behind the scenes.

TensorFlow offers additional conveniences for developers who need to debug and gain introspection into TensorFlow apps. Each graph operation can be evaluated and modified separately and transparently, instead of constructing the entire graph as a single opaque object and evaluating it all at once. This so-called "eager execution mode," provided as an option in older versions of TensorFlow, is now standard.

4.3.2 Keras

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly.

Keras runs on top of open-source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is deep learning framework developed by Microsoft. It uses libraries such as Python, C#, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks.

Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

4.3.3 Pillow

Python Pillow module is built on top of PIL (Python Image Library). It is the essential modules for image processing in Python. But it is not supported by Python 3. But anyone can use this module with the Python 3.x versions as PIL. It supports the variability of images such as jpeg, png, bmp, gif, ppm, and tiff.

Any operation can be done on the digital images using the pillow module. Various operations on the images such as filtering images, creating thumbnail, merging images, cropping images, blur an image, resizing an image, creating a water mark and many other operations can be done using Pillow library.

4.3.4 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays

of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today’s scientific/mathematical Python-based software, just knowing how to use Python’s built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

4.3.5 Tqdm

Tqdm is a Python library used to display smart progress bars that show the progress of your Python code execution. This library can also be used to see the progress of a machine learning model while training the model on a very large data set.

Sometimes it takes a lot of time while training a machine learning model on a very huge dataset. So, to check the progress of the model training Tqdm library can be used in Python to see how much time is remaining for our machine learning model to be trained.

4.3.6 Matplotlib

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open-source alternative to MATLAB. Developers can also use matplotlib’s APIs (Application Programming Interfaces) to embed plots in GUI applications. Data visualization which is the process of translating the numbers, text, or large data sets into various types of graphs such as histograms, maps, bar plots, pie charts, etc. For visualizations, some tools or technology is needed. Matplotlib is one of the most powerful libraries in python for data visualization.

It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

4.4 PROJECT FILE STRUCTURE

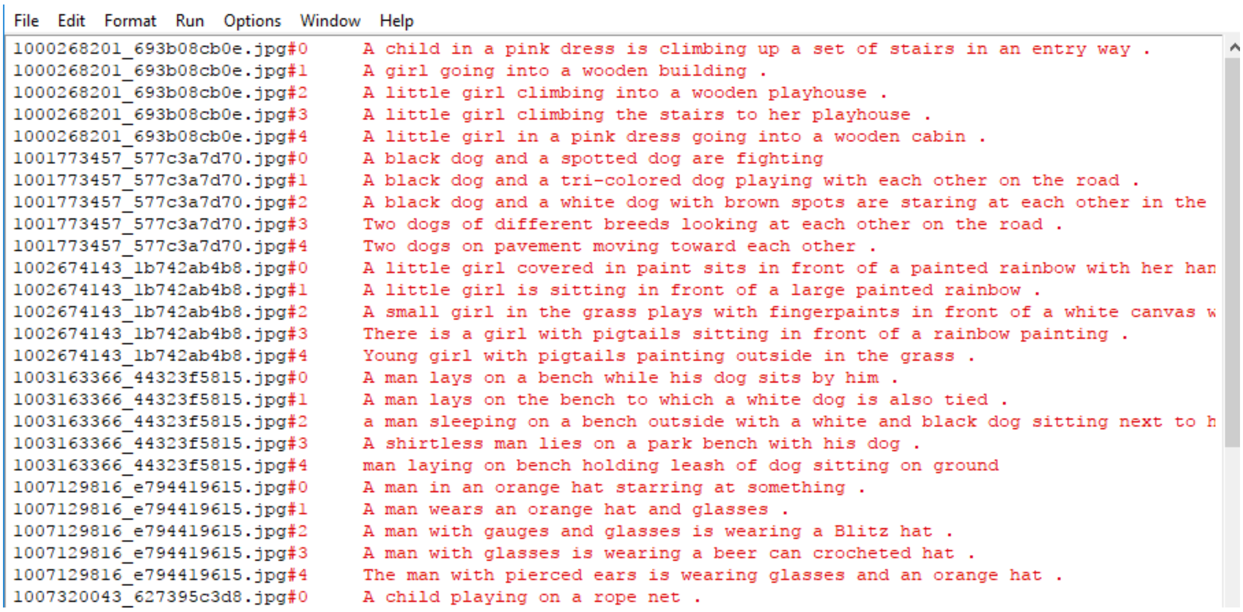
- **Flicker8k_Dataset –** Dataset folder which contains 8091 images.
- **Flickr_8k_text –** Dataset folder which contains text files and captions of images.
- **Models –** It will contain our trained models.
- **Descriptions.txt –** This text file contains all image names and their captions after pre-processing.
- **Features.p –** Pickle object that contains an image and their feature vector extracted from the Xception pre-trained CNN model.
- **Tokenizer.p –** Contains tokens mapped with an index value.
- **Testing_caption_generator.py –** Python file for generating a caption of any image.
- **Training_caption_generator.ipynb –** Jupyter notebook in which we train and build our image caption generator.

CHAPTER 5

IMPLEMENTATION

5.1 DATA PREPROCESSING

The main text file which contains all image captions is **Flickr8k.token** in our **Flickr_8k_text** folder.



```
1000268201_693b08cb0e.jpg#0	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1	A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4	A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0	A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1	A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2	A black dog and a white dog with brown spots are staring at each other in the
1001773457_577c3a7d70.jpg#3	Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4	Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0	A little girl covered in paint sits in front of a painted rainbow with her han
1002674143_1b742ab4b8.jpg#1	A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2	A small girl in the grass plays with fingerpaints in front of a white canvas w
1002674143_1b742ab4b8.jpg#3	There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4	Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0	A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1	A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2	a man sleeping on a bench outside with a white and black dog sitting next to h
1003163366_44323f5815.jpg#3	A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0	A man in an orange hat starring at something .
1007129816_e794419615.jpg#1	A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2	A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3	A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4	The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0	A child playing on a rope net .
```

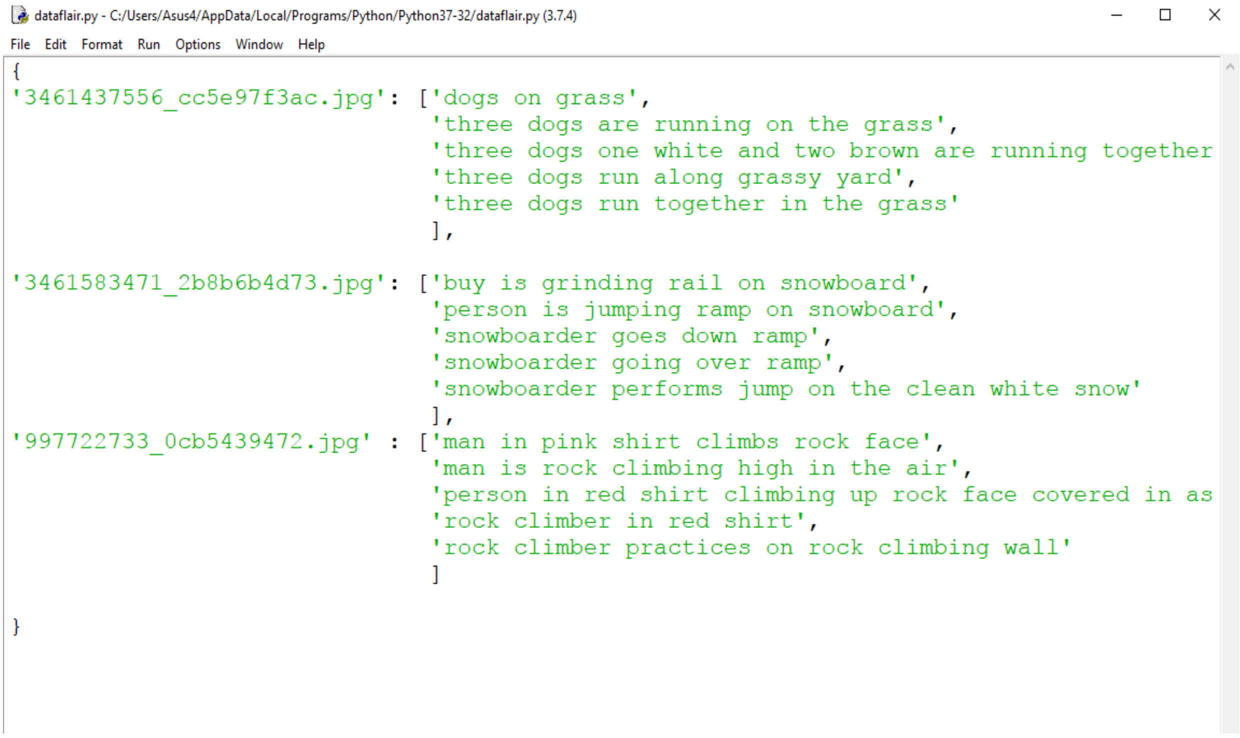
Figure 5.1 Flickr8k.token text file

The format of our file is image and caption separated by a new line (“\n”).

Each image has 5 captions and see that # (0 to 5) number is assigned for each caption.

This module defines 5 functions:

- **load_doc(filename) –** For loading the document file and reading the contents inside the file into a string.
- **all_img_captions(filename)** – This function will create a descriptions dictionary that maps images with a list of 5 captions. The descriptions dictionary will look something like the below Figure.



dataflair.py - C:/Users/Asus4/AppData/Local/Programs/Python/Python37-32/dataflair.py (3.7.4)
File Edit Format Run Options Window Help

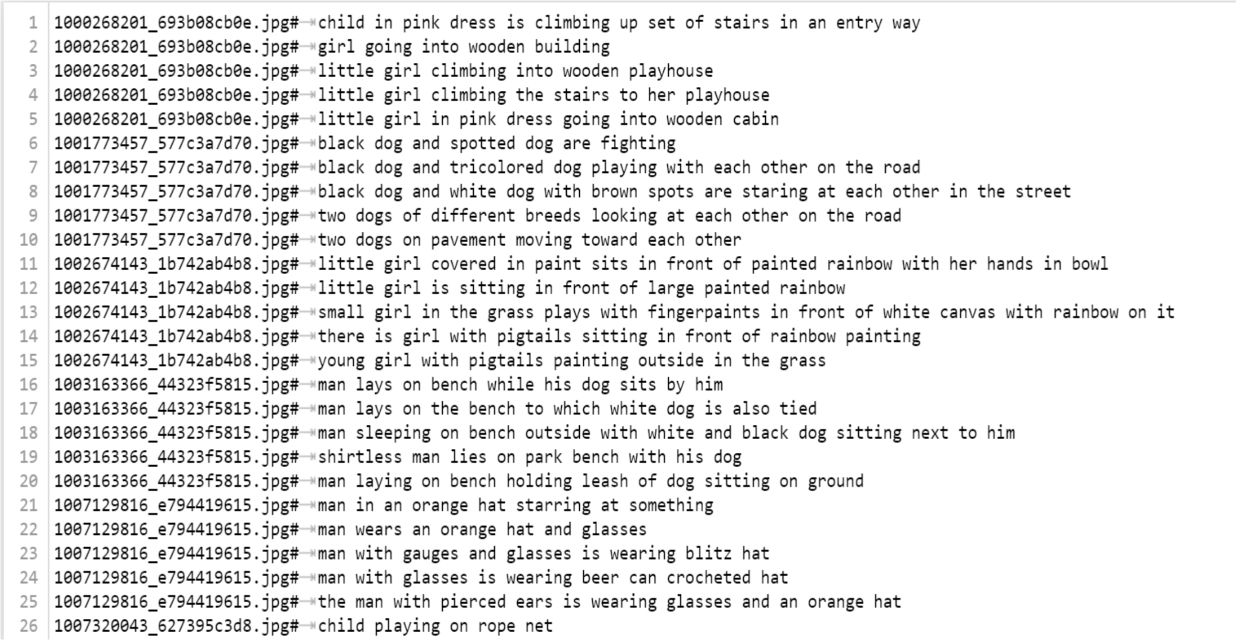
```
{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together
                              'three dogs run along grassy yard',
                              'three dogs run together in the grass'
                              ],

'3461583471_2b8b6b4d73.jpg': ['buy is grinding rail on snowboard',
                              'person is jumping ramp on snowboard',
                              'snowboarder goes down ramp',
                              'snowboarder going over ramp',
                              'snowboarder performs jump on the clean white snow'
                              ],
'997722733_0cb5439472.jpg' : ['man in pink shirt climbs rock face',
                              'man is rock climbing high in the air',
                              'person in red shirt climbing up rock face covered in as
                              'rock climber in red shirt',
                              'rock climber practices on rock climbing wall'
                              ]

}
```

Figure 5.2 Descriptions Dictionary

- **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step when working with textual data, according to the objective, what type of cleaning has to be performed on the text is decided. In this case, punctuations are removed, converting all text to lowercase and removing words that contain numbers. So, a caption like "A man riding on a three-wheeled wheelchair" will be transformed into "man riding on three wheeled wheelchair"
- **text_vocabulary(descriptions) –** This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.
- **save_descriptions(descriptions, filename) –** This function will create a list of all the descriptions that have been preprocessed and store them into a file. A descriptions.txt file is created to store all the captions.



```
1000268201_693b08cb0e.jpg#	child in pink dress is climbing up set of stairs in an entry way
1000268201_693b08cb0e.jpg#	girl going into wooden building
1000268201_693b08cb0e.jpg#	little girl climbing into wooden playhouse
1000268201_693b08cb0e.jpg#	little girl climbing the stairs to her playhouse
1000268201_693b08cb0e.jpg#	little girl in pink dress going into wooden cabin
1001773457_577c3a7d70.jpg#	black dog and spotted dog are fighting
1001773457_577c3a7d70.jpg#	black dog and tricolored dog playing with each other on the road
1001773457_577c3a7d70.jpg#	black dog and white dog with brown spots are staring at each other in the street
1001773457_577c3a7d70.jpg#	two dogs of different breeds looking at each other on the road
1001773457_577c3a7d70.jpg#	two dogs on pavement moving toward each other
1002674143_1b742ab4b8.jpg#	little girl covered in paint sits in front of painted rainbow with her hands in bowl
1002674143_1b742ab4b8.jpg#	little girl is sitting in front of large painted rainbow
1002674143_1b742ab4b8.jpg#	small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it
1002674143_1b742ab4b8.jpg#	there is girl with pigtails sitting in front of rainbow painting
1002674143_1b742ab4b8.jpg#	young girl with pigtails painting outside in the grass
1003163366_44323f5815.jpg#	man lays on bench while his dog sits by him
1003163366_44323f5815.jpg#	man lays on the bench to which white dog is also tied
1003163366_44323f5815.jpg#	man sleeping on bench outside with white and black dog sitting next to him
1003163366_44323f5815.jpg#	shirtless man lies on park bench with his dog
1003163366_44323f5815.jpg#	man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#	man in an orange hat starring at something
1007129816_e794419615.jpg#	man wears an orange hat and glasses
1007129816_e794419615.jpg#	man with gauges and glasses is wearing blitz hat
1007129816_e794419615.jpg#	man with glasses is wearing beer can crocheted hat
1007129816_e794419615.jpg#	the man with pierced ears is wearing glasses and an orange hat
1007320043_627395c3d8.jpg#	child playing on rope net
```

Figure 5.3 Pre-processed Description File

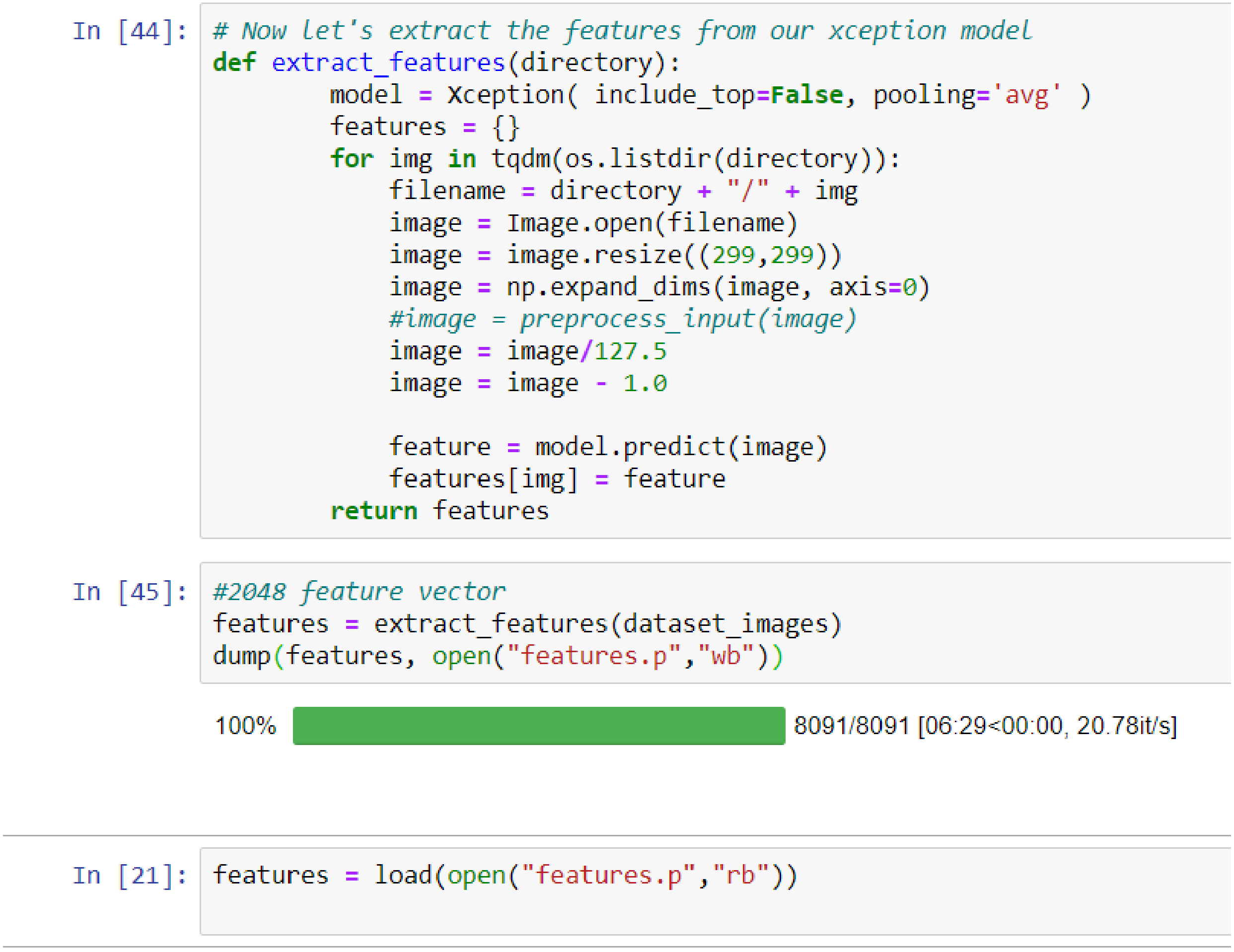
5.2 EXTRACTING FEATURE VECTOR

This technique is also called transfer learning, a pre-trained model is used that have been already trained on large datasets and extract the features from these models and use them for our tasks. The Xception model which has been trained on ImageNet dataset that had 1000 different classes to classify is used. This model can be directly imported from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for ImageNet, some little changes is made for integrating with our model. One thing to notice is that the Xception model takes 299*299*3 image size as input. The last classification layer is removed to get the 2048 feature vector.

model = Xception (include_top=False, pooling=‟avg‟)

The function **extract_features()** will extract features for all images and it maps image names with their respective feature array. Then the features dictionary is dumped into a “features.p” pickle file.

This process can take a lot of time depending on your system. I am using an Nvidia 1050 GPU for training purpose so it took me around 7 minutes for performing this task. However, if you are using CPU then this process might take 1-2 hours. You can comment out the code and directly load the features from our pickle file.



```
In [44]: # Now let's extract the features from our xception model
         def extract_features(directory):
                 model = Xception( include_top=False, pooling='avg' )
                 features = {}
                 for img in tqdm(os.listdir(directory)):
                     filename = directory + "/" + img
                     image = Image.open(filename)
                     image = image.resize((299,299))
                     image = np.expand_dims(image, axis=0)
                     #image = preprocess_input(image)
                     image = image/127.5
                     image = image - 1.0

                     feature = model.predict(image)
                     features[img] = feature
                 return features
```

```
In [45]: #2048 feature vector
         features = extract_features(dataset_images)
         dump(features, open("features.p","wb"))
```

```
         100%  8091/8091 [06:29<00:00, 20.78it/s]
```

```
In [21]: features = load(open("features.p","rb"))
```

Figure 5.4 Image Feature Extraction

5.3 LOADING DATASET

For loading the training dataset, more functions are needed:

- **load_photos(filename) –** This will load the text file in a string and will return the list of image names.

- **load_clean_descriptions(filename, photos) –** This function will create a dictionary that contains captions for each photo from the list of photos. It also appends the <start> and <end> identifier for each caption. This is needed so that our LSTM model can identify the starting and ending of the caption.
- **load_features(photos) –** This function will give us the dictionary for image names and their feature vector which has been previously extracted from the Xception model.

5.4 TOKENIZING THE VOCABULARY

Computers don't understand English words, for computers, words have to be represented with numbers. So, each word of the vocabulary is mapped with a unique index value. Keras library provides us with the tokenizer function that will be used to create tokens from our vocabulary and save them to a **"tokenizer.p"** pickle file.

The vocabulary contains 7577 words. The maximum length of the descriptions is calculated. This is important for deciding the model structure parameters. Maximum Length is 32.

5.5 CREATE DATA GENERATOR

To make this task into a supervised learning task, input and output sequence pair is provided to the model for training. The model is trained on 6000 images and each image will contain 2048 length feature vector and caption is also represented as numbers. This amount of data for 6000 images is not possible to hold into memory so a data generator method is used that will yield batches. The generator will yield the input and output sequence.

For example:

The input to our model is [x1, x2] and the output will be y, where x1 is the 2048 feature vector of that image, x2 is the input text sequence and y is the output text sequence that the model has to predict.

X1 (feature vector)	X2 (text sequence)	Y (word to predict)
feature	startseq,	little
feature	startseq, little,	girl
feature	startseq, little, girl,	running
feature	startseq, little, girl, running,	in
feature	startseq, little, girl, running, in,	field
feature	startseq, little, girl, running, in, field,	endseq

Table 5.1 Word Prediction

5.6 DEFINING THE CNN-RNN MODEL

To define the structure of the model, the Keras Model from Functional API is used. It will consist of three major parts:

- **Feature Extractor –** The feature extracted from the image has a size of 2048, with a dense layer, the dimensions are reduced to 256 nodes.
- **Sequence Processor –** An embedding layer will handle the textual input, followed by the LSTM layer.
- **Decoder –** By merging the output from the above two layers, the model will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Visual representation of the final model is given in the below figure.

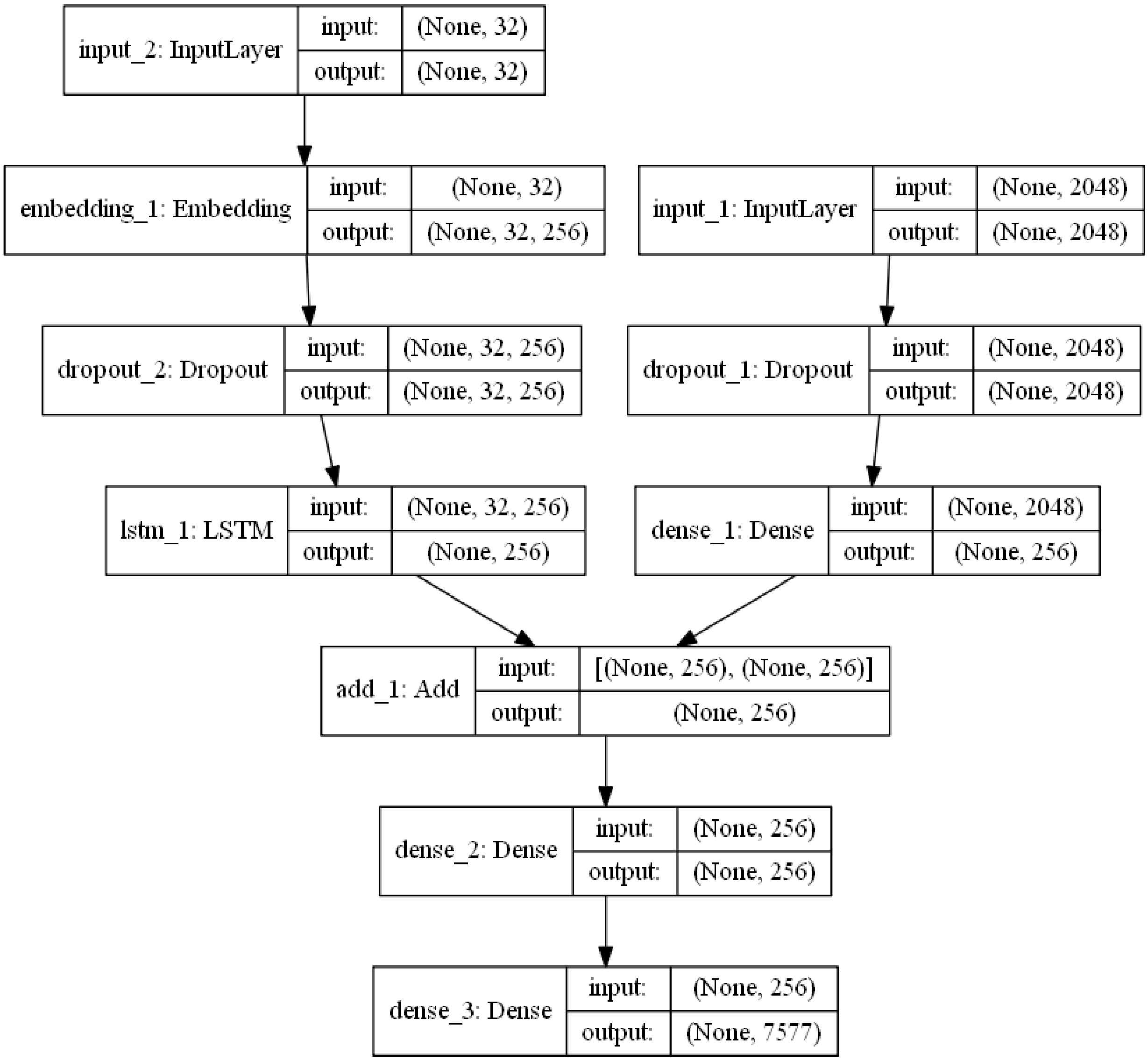


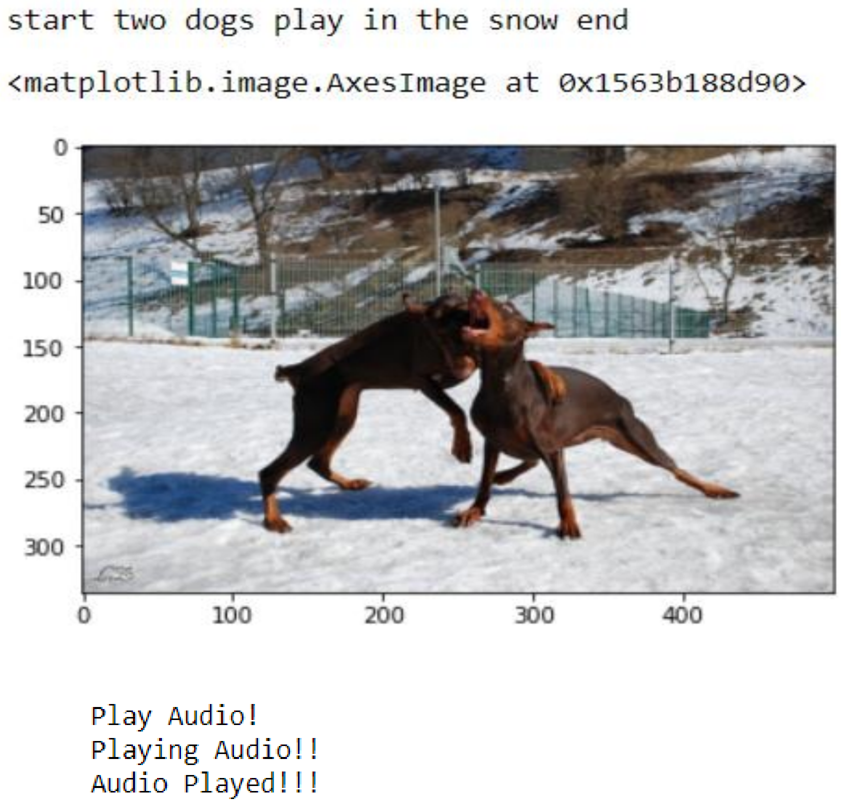
Figure 5.5 Final Model Structure

5.7 TRAINING THE MODEL

To train the model, 6000 training images are used by generating the input and output sequences in batches and fitting them to the model using model.fit_generator() method. The training models are saved to the Models folder. This will take some time depending on the system capability.

5.8 TESTING THE MODEL

The model has been trained, now, a separate file testing_caption_generator.py will load the model and generate predictions. The predictions contain the max length of index values so the same tokenizer.p pickle file is used to get the words from their index values.

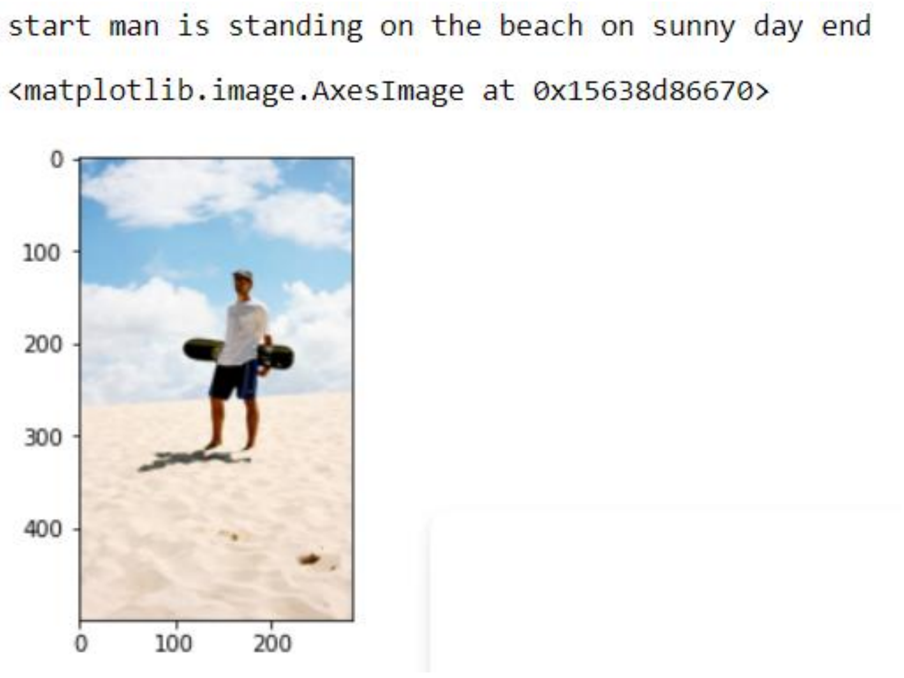


```
start two dogs play in the snow end

<matplotlib.image.AxesImage at 0x1563b188d90>
```

```
Play Audio!
Playing Audio!!
Audio Played!!!
```

Figure 5.6 Image Caption-I

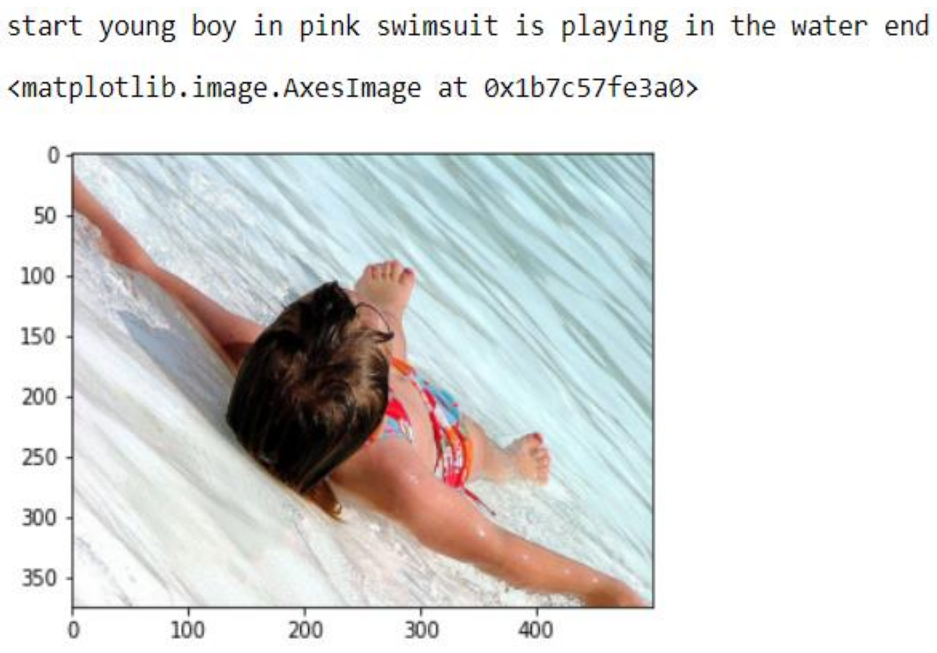


```
start man is standing on the beach on sunny day end

<matplotlib.image.AxesImage at 0x15638d86670>
```

```
Play Audio!
Playing Audio!!
Audio Played!!!
```

Figure 5.7 Image Caption-II

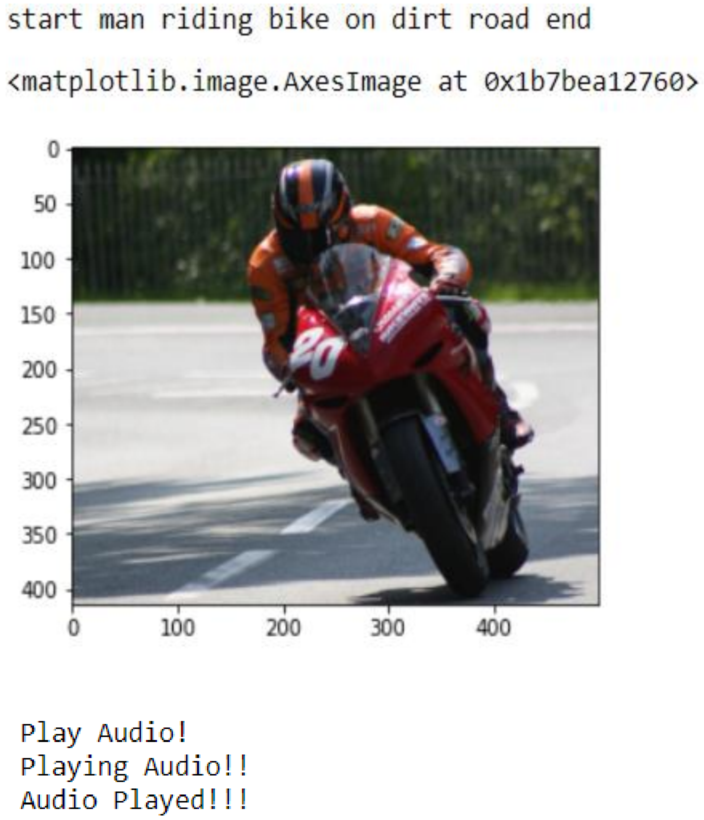


```
start young boy in pink swimsuit is playing in the water end

<matplotlib.image.AxesImage at 0x1b7c57fe3a0>
```

```
Play Audio!
Playing Audio!!
Audio Played!!!
```

Figure 5.8 Image Caption-III



```
start man riding bike on dirt road end
<matplotlib.image.AxesImage at 0x1b7bea12760>
```

```
Play Audio!
Playing Audio!!
Audio Played!!!
```

Figure 5.9 Image Caption-IV

The caption along with the image is plotted using Matplotlib library. Python Text-to-Speech API (pyttsx3) is used to convert the caption to audio. The audio can be played at different rates and different voice modulation using different properties.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

In this project, an Image caption generator is implemented by a CNN-RNN model. Flickr8k dataset is used which includes nearly 8000 images, and the corresponding captions are also stored in the text file. Although deep learning - based image captioning methods have achieved a remarkable progress in recent years, a robust image captioning method that is able to generate high quality captions for nearly all images is yet to be achieved. With the advent of novel deep learning network architectures, automatic image captioning will remain an active research area for some time. The scope of image-captioning is very vast in the future as the users are increasing day by day on social media and most of them would post photos. Generating descriptions for images and reading out the same using a computer can aid the visual disabled. So, this project will help them to a greater extent.

6.2 FUTURE WORK

Future work Image captioning has become an important problem in recent days due to the exponential growth of images in social media and the internet. This report discusses the various research in image retrieval used in the past and it also highlights the various techniques and methodology used in the research. As feature extraction and similarity calculation in images are challenging in this domain, there is a tremendous scope of possible research in the future. Current image retrieval systems use similarity calculation by making use of features such as colour, tags, Image Retrieval Using Image Captioning 54 histogram, etc. There cannot be completely accurate results as these methodologies do not depend on the context of the image. Hence, complete research in image retrieval making use of context of the images such as image captioning will facilitate to solve this problem in the future. This project can be further enhanced in future to improve the identification of classes which has a lower precision by training it with more image captioning datasets. This methodology can also be combined with previous image retrieval methods such as histogram, shapes and can be checked if the image retrieval results get better.

REFERENCES

1. Andrej Karpathy and Li Fei-Fei (April 2017) “Deep Visual-Semantic Alignments for Generating Image Descriptions”, IEEE Trans. Pattern Anal. Mach.Intell39, 664–676.

2. Cao, D., Zhu, M. & Gao, L (2019) “An image caption method based on object detection” Multimedia Tools Application 78**,** 35329–35350.

3. Hao Fang, Saurabh Gupta, Forrest Iandola, Li Deng (2015) "From captions to visual concepts and back", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1473-1482.

4. Jeffery Donahue, Lisa Anne Hendricks, Marcus Rohrbach (April 2017) "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description" in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 4, pp. 677-691.

5. Jiasen Lu, C. Xiong, D. Parikh and R. Socher (2017) "Knowing When to Look: Adaptive Attention via a Visual Sentinel for Image Captioning", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3242-3250.

6. Kelvin Xu., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Bengio, Y. (June 2015). “Show, attend and tell: Neural image caption generation with visual attention”, In International conference on machine learning, pp. 2048-2057.

7. Kuznetsova, Polina & Ordonez, Vicente & Berg, Alexander & Berg, Tamara & Yejin, Choi. (2012). “Collective generation of natural image descriptions”. Association for Computational Linguistics, ACL 2012 - Proceedings of the Conference. 1. 359-368.

8. Margaret Mitchell, Xufeng Han, Jesse Dodge, Alyssa Mensch, Amit Goyal, and Hal Daumé. (2012) “Generating image descriptions from computer vision detections” In Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics, 747–756.

9. O. Vinyals, A. Toshev, S. Bengio and D. Erhan (2015) "Show and tell: A neural image caption generator", IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3156-3164.

10. Yin X, Ordonez V. (July 2017) “Generating visually descriptive language from object layouts”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3356-2364.

APPENDIX

DATA PREPROCESSING CODE

```
def load_doc(file_name):
    file = open(file_name,"r")
    text = file.read()
    file.close()
    return text

def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    print("caption : ", len(captions))
    descriptions ={}
    for caption in captions:
        img, caption = caption.split('\t')
        print("img",img)
        if img[:-1] not in descriptions:
            descriptions[img[:-1]] = list()
            descriptions[img[:-1]].append(caption)
        else:
            descriptions[img[:-1]].append(caption)
    return descriptions

def cleaning_text(captions):
    table = str.maketrans("","",string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):
            img_caption.replace("-"," ")
            desc = img_caption.split()

            #Lowercase
            desc = [word.lower() for word in desc]
            #punctuations
            desc = [word.translate(table) for word in desc]
            #hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            #tokens with numbers
            desc = [word for word in desc if(word.isalpha())]
            #back to string
            img_caption = " ".join(desc)
            captions[img][i] = img_caption
    return captions

def text_vocabulary(description):
    vocab = set()
    for key in description.keys():
        [vocab.update(d.split()) for d in description[key]]
    return vocab

def save_descriptions(descriptions, filename):
    lines = list()
    for key , desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + "\t" + desc)
    data = "\n".join(lines)
    file = open(filename,"w")
    file.write(data)
    file.close()

dataset_text = "D:/NLP"
dataset_images ="D:/NLP/Flicker8k_Dataset"
#prepare text data
filename = dataset_text + "/" + "Flickr8k.token.txt"
#mapping as image to 5 descriptions in description dictionary
descriptions = all_img_captions(filename)
print("Length of descriptions = ",len(descriptions))
#Cleaning the descriptions
clean_descriptions = cleaning_text(descriptions)
#vocabulary
vocabulary = text_vocabulary(clean_descriptions)
print("length of the vocabulary : ",len(vocabulary))
#saving description to file
save_descriptions(clean_descriptions,"descriptions.txt")
```

TOKENIZING CODE

```
from keras.preprocessing.text import Tokenizer

#mapping each unique word to an index and clean
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    print(all_desc)
    return all_desc
```

```
def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer

#Storing into a pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer,open("tokenizer.p","wb"))
vocab_size = len(tokenizer.word_index) + 1

#calculate maximum length od descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)
```

TRAINING THE MODEL CODE

```
def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # split into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            # pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)

def data_generator(descriptions, features,tokenizer,max_length):
    while 1:
        for key, description_list in descriptions.items():
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
            yield ([input_image,input_sequence],output_word)

#captioning model
def define_model(vocab_size, max_length):
    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    # LSTM 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)
    # Merging both models
    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)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)
    return model

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
#directory for saving our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions,train_features, tokenizer, max_length)
    #print(type(next(generator)),len(next(generator)))
    model.fit(generator,epochs = 1,steps_per_epoch = steps, verbose = 1)
    model.save("models/model_" + str(i) + ".h5")
```

TESTING THE MODEL CODE

```
def extract_features(filename,model):
    try:
        image = Image.open(filename)
    except:
        print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
    image = image.resize((299,299))
    image = np.array(image)
    if image.shape[2]  ==4:
        image  = image[..., :3]
    image = np.expand_dims(image, axis = 0)
    image = image/127.5
    image = image - 1.0
    feature = model.predict(image)
    return feature

def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

def generate_desc(model, tokenizer, photo, max_length):
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text
```

```
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")
photo = extract_features(img_path, xception_model)
img = Image.open(img_path)
description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

TEXT-TO-SPEECH CODE

```
import pyttsx3
engine = pyttsx3.init()
text = description[6:len(description)-4]
voices = engine.getProperty("voices")
engine.setProperty("voice", voices[2].id)
engine.setProperty("rate", 130)
engine.say(text)
engine.runAndWait()
```