Dissertation on

"Image Description using CNN"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

DILIP KUMAR N	**01FB15ECS417**
LAKKANNA	**01FB15ECS426**
SWETHA DEVI D	**01FB15ECS454**

Under the guidance of

Internal Guide
Dr. NAGEGOWDA K.S
Assistant Professor

January – May 2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

'Image Description Using CNN'

is a bonafide work carried out by

DILIP KUMAR N	**01FB15ECS417**
LAKKANNA	**01FB15ECS426**
SWETHA DEVI D	**01FB15ECS454**

In partial fulfillment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2018 – May. 2018. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature	Signature	Signature
Dr. Nagegowda K S	Dr.Shylaja S S	Dr.B.K.Keshavan
Assistant Professor	Chairperson	Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1._____	_____
2._____	_____

DECLARATION

We hereby declare that the project entitled **"Image Description using CNN"** has been carried out by us under the guidance of Dr. Nagegowda K S, Assistant Professor and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology in Computer Science and Engineering of PES University, Bengaluru** during the academic semester January – May 2018. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

Submitted By:

DILIP KUMAR N	**01FB15ECS417**
LAKKANNA	**01FB15ECS426**
SWETHA DEVI D	**01FB15ECS454**

ACKNOWLEDGEMENT

A successful project is a fruitful culmination of efforts of individuals directly of those who quietly supported. A project is considered incomplete and devoid of satisfaction if one fails to ACKNOWLEDGE all these individuals who have been instrumental in the successful completion of this project.

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance, encouragement and cooperation of intellectuals, elders and friends. A number of personalities, in their own capacities have helped us in carrying out this project work. We would like to take this opportunity to thank them all.

.

First and foremost we would like to thank our guide **Dr. Nagegowda K S**, Assistant Professor, PES University Bangalore, for his regular guidance, encouragement and assistance throughout the project

We deeply express our sincere gratitude to Project Co-ordinator **Prof. V R Badri Prasad**, Associate Professor, PES University Bangalore, for his valuable suggestions and expert advice.

We would like to thank **Dr. Shylaja S Sharath**, Head of the department, Computer Science and Engineering, PES University Bangalore, for his moral support towards the completion of the project.

We would like to thank Principal, **Dr. K S Sridhar**, PES University Bangalore, for his moral support towards the completion of the project.

We would like to thank Vice Chancellor**, Dr. K.N.B Murthy**, PES University Bangalore, for his moral support towards the completion of the project.

We would like to thank CEO and Pro chancellor**, Prof. Jawahar Doreswamy**, PES University Bangalore, for his moral support towards the completion of the project

Our most sincere and grateful acknowledgement to th**e P.E.S UNIVERSITY** for providing us with the opportunity to pursue our degree, and thus, helping us in shaping our career.

We extend our deep sense of attitude towards our **PARENTS & our beloved friends**, for their constant support, encouragement and co-operation.

ABSTRACT

Image Description using CNN, i.e. transferring an image with the textual description, is a difficult image processing task. Recent advances in machine learning, especially Convolutional Neural Networks optimized for object recognition, have made it possible to extract high level image information. Using this information learned by the Convolutional Neural Networks, Automatic image description brings together recent advances in natural language processing and computer vision.

In this project we try to understand and implement the model of VGG16, this models is used for the prediction, feature extraction, and fine tuning. VGG neural network is an image classification convolutional neural network, it usually refers to a deep convolution network for object recognition developed and trained

Keywords: Image Description, deep learning, VGG16

CHAPTER 1

INTRODUCTION

Artificial Intelligence (AI) is now at the main building block of the innovation economy and thus the base for this project is also the same. In the recent past, a field of AI, namely Deep Learning has moved a lot of heads due to its impressive results in terms of accuracy when compared to the already algorithms. The task of being able to generate a meaningful sentence from an image is a difficult task but can have a great impact, for instance helping the blind people to get understanding the image by providing the description of the image into speech.

The task of image captioning is significantly harder than that of image classification, which has been the main focus in the computer vision community. A description of an image must capture the relationship between the objects in the image. To have understanding of the image the above knowledge has expressed in natural English language, this means English language model is needed. The attempts made in the past have all been to stitch the two models together.

The model proposed in the paper, we try to combine this into a single model which consists of a Convolutional Neural Network (CNN) encoder which helps in creating image encodings. We could have used some of the recent and advanced classification architectures, but that would have increased the training time significantly. These encoded images are pass to a RNN networks. The network architecture used for the LSTM network work in similar fashion as the ones used in machine translators. The input to the network is an image which is first converted in a 224*224 dimension. We use the Flickr8k dataset to train the model. The model outputs a generated caption based on the dictionary it forms from the tokens of the caption in the training set.

The feature of concern of this project is to provide a correct description for the given an image. Basically, this is not an easy work to a machine for generating a Language for the given an image, it should need a better algorithm and larger dataset to train the neural network. We have prepared some the related papers which help us to get an idea of the neural network topics and machine learning. We make sure that we have achieved the possible accuracy of a neural network by training the model and we succeed in getting the description for input an image.

1.1 PURPOSE

The purpose of this document is to relay important functional and specific requirements, algorithms, assumptions, constraints, etc. Needed to build the application, as well as to provide some perspective on why this application is being built. This is written as a part of final year project report.

We have included the detail about the concepts used and the technologies and their detailed explanation about why we used such concepts. This document contains the problem statement about for what we are going to give solution and how we are going to a get solution. The main thing is that what is the prerequisite for the project and how one can learn new things from this to adopt machine learning concepts into their project or their own idea.

Detail about the design and implementation of the project with an explanation of the concepts with a resource where one can get a clear picture about concepts like CNN, RNN and their usages. In this project we have used Keras API with Tensor flow backend to achieve large data to render.

Have mentioned the pre-requisite application to be installed before using this project, and information about the minimum system and hardware requirements. Along with this we have given each module detail used in this project and explained how each model works and what is the role of each module in this project.

This document contains some additional concepts which are related to the machine learning concepts which is the most important requirement for the project.

1.2 SCOPE

"Image Description using CNN" understanding the whole image scenario, not individual objects. Image Description extracts the details of specific or individual object and their connection from an image. Finally, neural network generates the description for the given image. These information will be produced as a vectors, which is then passed into to the LSTM. LSTM will generates the English language description according to received parameters.

VGG16 and LSTM model is merged to get a description of an image. VGG16 model is to identify or classify the objects in the image and LSTM is to generate the English language based on the VGG16 model predicted objects with an action in an image. Concentrated on to get the highest accuracy in neural network by using better algorithms as possible.

Making GUI for the user to make a job easier to get the description by choosing the image from the file selection dialog and button to get a description after choosing an image. The result description should show in the GUI that makes the user to understandable who don't know how to use the terminals and what the commands to use for the functionality are.

Created a simple GUI for the user to make the use of the application be in the simpler way the user will easily get to know how to operate and will get an idea about the application without giving any external information about the GUI/Application. The GUI is designed and developed using WxPython, this is very simple and attractive library this can support the Windows and Linux both.

The score of this project/application is limited to, the following actions:

- Fetching the New image for the user
- Applying the encoding for the given image
- Applying the train for the given image dataset
- Providing an output as the description of an image

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- **CNN:** Convolution Neural Network
- **ANN:** Artificial Neural Network
- **VGG16:** Visual Geometry Group
- **LSTM:** Long Short Term Memory
- **RNN:** Recurrent Neural Network
- **BLEU:** Bilingual Evaluation Understudy
- **FC:** Fully Connected Layer
- **LRCN:** Long-term Recurrent Convolutional Network
- **AMT:** Amazon Mechanical Turk

CHAPTER 2

PROBLEM DEFINITION

2.1 PROBLEM STATEMENT

In the recent past a field of AI, namely Deep Learning has turned a lot of heads due to its impressive results in terms of accuracy when compared to the already existing intelligence algorithms. The task of being able to generate a meaningful sentence from an image is a difficult task but can have great impact, for instance helping the visually impaired to have a good understanding of an images, which they can feel to recognize the what's there in the images.

2.2 EFFICIENCY METRICS

There are no easy metrics to define if the description of the image has been applied accurately to the content image. The algorithm/procedure efficiency is calculated by manually looking at the generated output images.

Some of the common things to consider for the manual descriptions by the human are:

- **Perceptions:** The Perceptions that we look are the objects and the people's interactions with its environment.
- **Main Objects:** The objects like people, things, and places.
- **Behavioral observation:** Involves identifying behavior of the human in certain perspective.

If the human generated descriptions and the machine generated descriptions are approximate to 90% or 85%, then it should be considered as an acceptable description.

2.3 RUNTIME

We consider the Runtime into two separate phases, they are as follows:

Epochs: One epoch is when an entire dataset is passed forward and backward through the neural network only once.

- **Training phase:** Training phase involves the train of our model, by pairing the input image and the expected description. The training phase never ends until we get the acceptable or maximum accuracy, based on this the number of epochs are needed to train the model. **One epoch** takes around eight to nine hours to complete to train the entire dataset once.
- **Evaluation phase:** Evaluation phase involves the validating of the trained model by providing the test input images. It takes one minute to complete the evaluation of one input image.

CHAPTER 3

LITERATURE SURVEY

- **Long-term Recurrent Convolutional Networks for Visual Recognition and Description[1]**

The authors Jeff Donahue, Lisa Anne Hendricks and Marcus defines the researches and propose different models which relates to the concepts of the deep convolution neural networks and have provided various features which is based on the recent image interpretation related tasks, they check whether the models are recurrent and efficient for the tasks involving an order wise or sequences, visually and can further describe the different classes of the convolution network architectures which is further trainable is applicable for the large scale development processes. It also demonstrates the visually understandable tasks, and the value of these models for activity recognition systems and video descriptions.

In comparison to the previously developed model which assumes a fixed representation or performs a simple temperament average leveling for the sequential order for processing, RNN models are “doubly deep” in this they learn the spatial and time models.

These models incorporate important steps of spatial and time series development. Either fully time varying weighting or provide an initiative to their proposed architectures in three settings. Firstly, provide directly connecting visually convolutional model to the deep learning LSTM model networks, they are able to train the video recognition system models that capture the different state dependencies sections. While considering the different video activity label datasets it might have a great significant impact on the activity datasets and may have that particular complicated dynamic dependencies. We have nonetheless observe the significant improvements on the convolutional neural network architectures benchmarks.

RNN is better in speech identification, and the English textual language generation. It would be a difficult task to train them to learn the changing models, like due in part to the vanishing and exploding gradients to train the model. LSTM provide a solutions by memory units that explicitly allow the network to learn itself when to “forget” previous hidden states and when to update hidden states given new information.

This paper presents the set of an extension to the neural network image captioning, this basically involves the object and the scene recognition with the template or a tree based approaches to generate the descriptional captions. Such sentences are most typically simple and can be distinguished from more fluent human generated description. The own algorithm passes the image to have the description in simple linear methods for transferring the image with the description. The different methods proposed are the activity recognition and LRCN model.

This work suggests a LRCN model which includes the deep hierarchical feature extractor (such as a Convolution Neural Network) can learn to identify the temporal changing for tasks including sequential data (inputs or outputs), visual and linguistic data.

Hence, the conclusion is that they evaluate their description model for retrieval and generation tasks. They firstly provide the description about the effectiveness of their model by continuously evaluating it on the image and caption retrieval tasks. They report results on the flickr30k, and COCO 2014 datasets, both with five captions annotated per image.
They have presented the LRCN, which is a class of models that is both structural and non-structural deep, and flexible enough to be added to other types of visionary job to include consecutive inputs and outputs. Our results on this research have demonstrated that by learning sequential dynamics with a deep ordering of a deep hierarchy of parameters only in the visual domain and can be included to models which takes a fixed visual representation of the input and learn only for the dynamics of the output sequence.
As the field of computer vision has been increased in demand, the tasks with input and predictions, deep sequence modeling tools like LRCN are increasingly central vision systems for problems with sequential data structure. The simpler way with which these tools can be incorporated into the visual identification pipelines makes them a natural choice for the problems with the time-varying visual input of the orderly sequence outputs which are provided with the capability to handle the little preprocessing and no hand-designed features.

- **Learning a Recurrent Visual Representation for Image Caption Generation [2]**

The authors Lawrence Zitnick and Chen Xeini have defined a method of representation of the visual capable of generating the novel descriptions from images and visual representations from descriptions. The main focus for the research of this paper is that to provide the technique to find the approachable representations of image with captions in the convolutional neural network are well separable. Considering both of these tasks the novel representation that effectively holds the visual parameters of the scene that have already been applied and described. That is, a word is produced or get from the visual render is updated to update the new information contained in the word.

The authors propose their method on the numerous datasets. The approaches that the authors have used are the recurrent neural networks, they have two goals.

Firstly they want to be able to generate the sentences given a sequence set of words and the visual observations or features. Intentionally, want to calculate the probability of the given word and a set of the previously generated words and observed visual features. Secondly, they want to allow the capability of computing the similarity of the visual features and the read word for performing the image search.

The authors construct the RNN model structure and builds on the prior models defined by the Mikolov, proposed a RNN language model. The basic feature or property of the visual features id their ability to remember the visual concepts over the long term. One may assume that the visual features should not be estimated until the sentence is defined or finished.

The convolutional network can understand the structure and builds on the prior of the different models being proposed in a RNN language model. The basic property of the recurrent visual features is that to predict the visuality features from sentences and generate various sentences. We define the details of the language model and how the authors of this paper learn their network.
The language model: the author of this paper propose their language model that had between 3,000 to 20,000 words. In this language model evry word is predicted independently, this approach is exponentially too costly to be implemented. Therefore they adopted the idea of the word classifying and factorized the distribution policy into a product of two or many terms

Learning model: for the learning purpose they use the Back propagation through Time (BPTT) algorithm. Specifically, the network structure is unrolled and implemented for several different words. Note that we reset the model after an End-of Sentence (EOS) is encountered, so that the prediction does not cross the sentence boundaries.

The authors define and evaluate the efficiency of their RNN model on multiple tasks. Then they begin to describe the datasets used for the training, evaluating and testing followed by their baselines. Their first set of the evaluation process was to measure the model's capability to generate novel descriptions of the images. Many words provides a way to correspond the feature localization of the different images and could improve the performance of the specified model. Since their model is bi-directional, they evaluate the models performance both on image retrieval applications and sentence retrieval. To provide a caption generation, they first sample a target sentence length from the multinomial distribution learned from the training data model, then for this fixed length we do a sampling of 100 random sentences, and use the one with the lowest loss output. The image captioning describes both the objects in the image and their relationships. An area of future work is to examine the sequential oedering of the exploration of an image.
Hence, the authors provide the conclusion that they describe the first bi-directional model which is capable of then generating both the novel descriptions and the visuality features. When compared with many previous approaches using recurrent neural networks, the model is capable of learning long-term interactions in the model. This emerge from a RNN visual memory that study to rebuild the visual features as new words are read or created. They implemented state of the model results on the task of caption generation, image retrieval and sentence retrieval on numerous datasets. The results presented in this paper were generated and defined and based on the VGG-Network, a Convolutional Neural Network

- **Deep visual-semantic alignments for image description [3]**

  An immediate view at an image is sufficient for a human to figure out and describe a large amount of details about the visual scene. The authors of this paper are Andrej Karpathy and Li Fei-Fei details the model that would generate the natural language descriptions of the images and their regions.

  The alignment model is involved based on the convolution neural networks. The primary focus towards the goal of this paper is to design a model such that the model can simultaneously provide information about the contents of the images and the domain representation of the language model. In respect to that this model should be free from the assumptions about the specific hard-coded template. And must be independent apart from the learning from the training data

  The second, practical challenge is that datasets of image captions are available on the internet, They develop a deep neural network model that equate to the latent adjustment between segments of sentences and the image related regions that they describe. The overview of the model used is to generate descriptions of image regions. While training, the input to the model is image and with corresponding captions for each image. They first shows a model that aligns sentence chunks to the visual regions that explain through a multimodal embedding.
  Then they do correspondences as training data for a second, the recurrent neural network learns to generate the snippets. Their main goal is that sentences are written by people to make the frequency reference to some particular input, but unknown location in images must be described. Since the assumptions are at the level of the entire images and sentences. The basic strategy is to formulate the image – sentences core as a function of the region word scores. Sentence image match should have a maximum matching score if its words have a possible support in the image.

  The various methods proposed by the authors are learning to align visual and language data, representing images, representing images, decoding text segment alignments to images, RNN for creating the captions.

  This paper going to show on deep visual-semantic alignment which makes enough use of machine learning methods and provides a model that aligns sentence chunks to the visual image regions that they describe through a multimodal embedding, then treat these correspondences as training data for a second, multimodal RNN learns to generate the snippets. They first check the equality of the given text and image alignments with ranking experiments. To help the evaluation of the training used AMT to gather the image dataset annotations that use at the test time. Similar to the full image Description task, they evaluate this data as a prediction task from two dimensional array of pixels to the sequence of words and the BLEU score. The ranking basis fetches training sentence sub content most similar with each region as judged by the model. The RNN

model produces the output descriptions that is similar with the gathered data. The model is trained on full images so smaller images may lead to get wrong or not acceptable output.

Hence the conclusion, the authors have introduced a model that generates natural language descriptions image regions based on weak labels in the form of a dataset of images along with description to rank the novels, while ranking of the novels should consider the modulations of the word they have used, it should figure out the modulations of each word generated and model provides state of the art performance on Image-sentence ranking experiments. Second, described about the RNN architecture that generates textual language. We evaluated its accuracy by checking of the possible outcomes.

CHAPTER 4

PROJECT REQUIREMENT SPECIFICATION

4.1 Gantt Chart

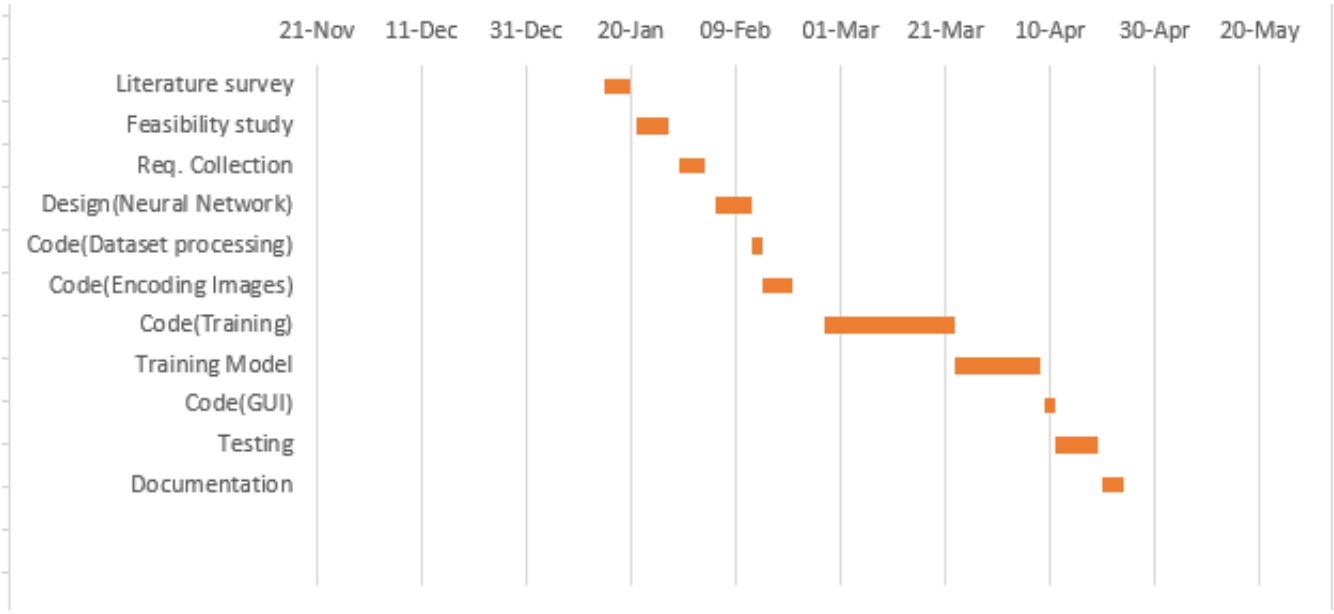

Fig. 4.1: Gantt chart

4.2 Activity Diagram

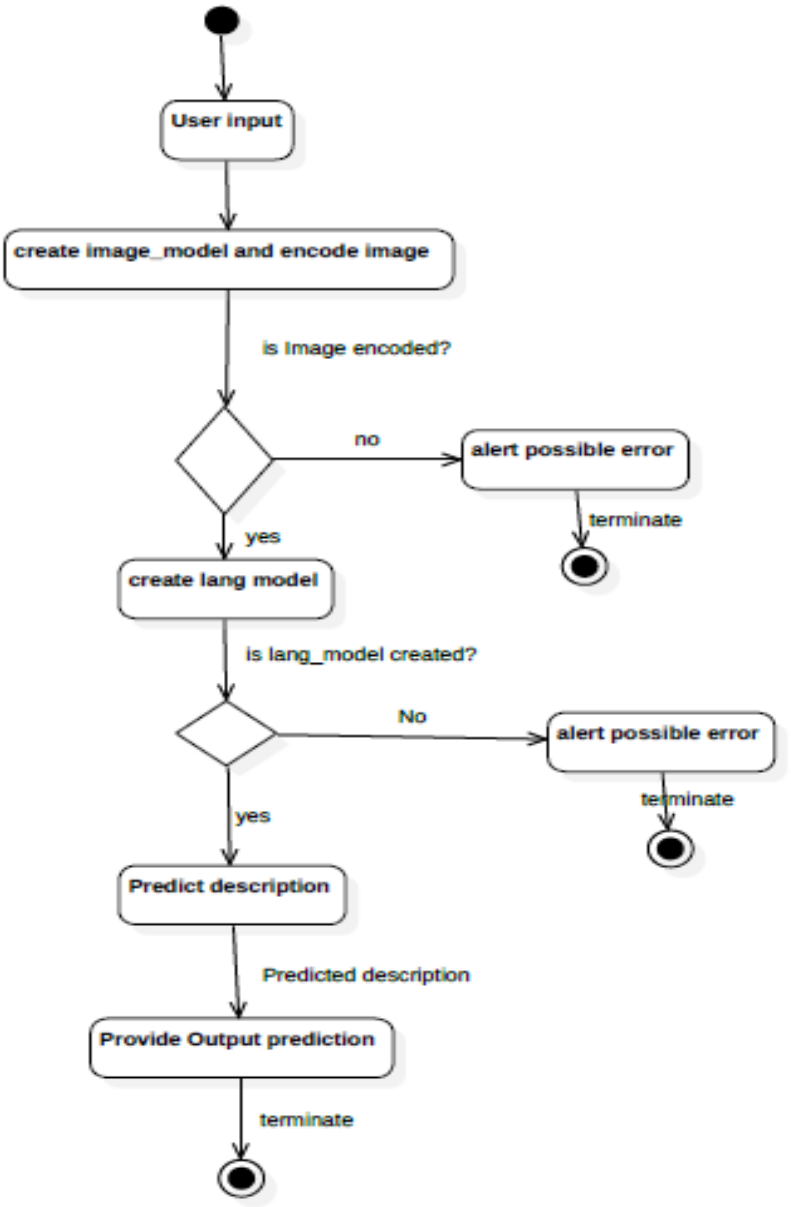

Fig. 4.2: Activity Diagram

4.3 Workflow Diagram

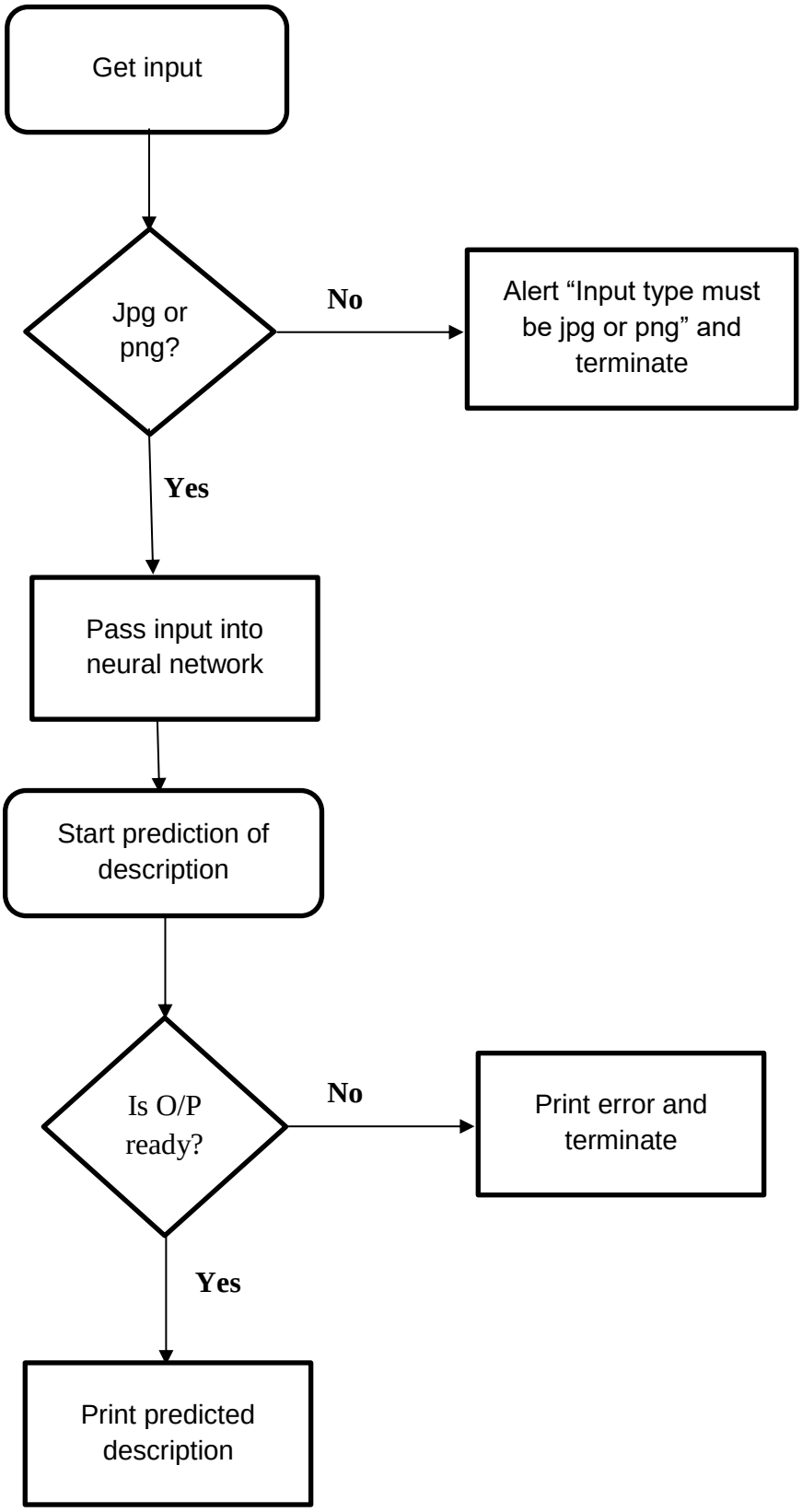

Fig. 4.3: Workflow Diagram

4.4 Class Diagram

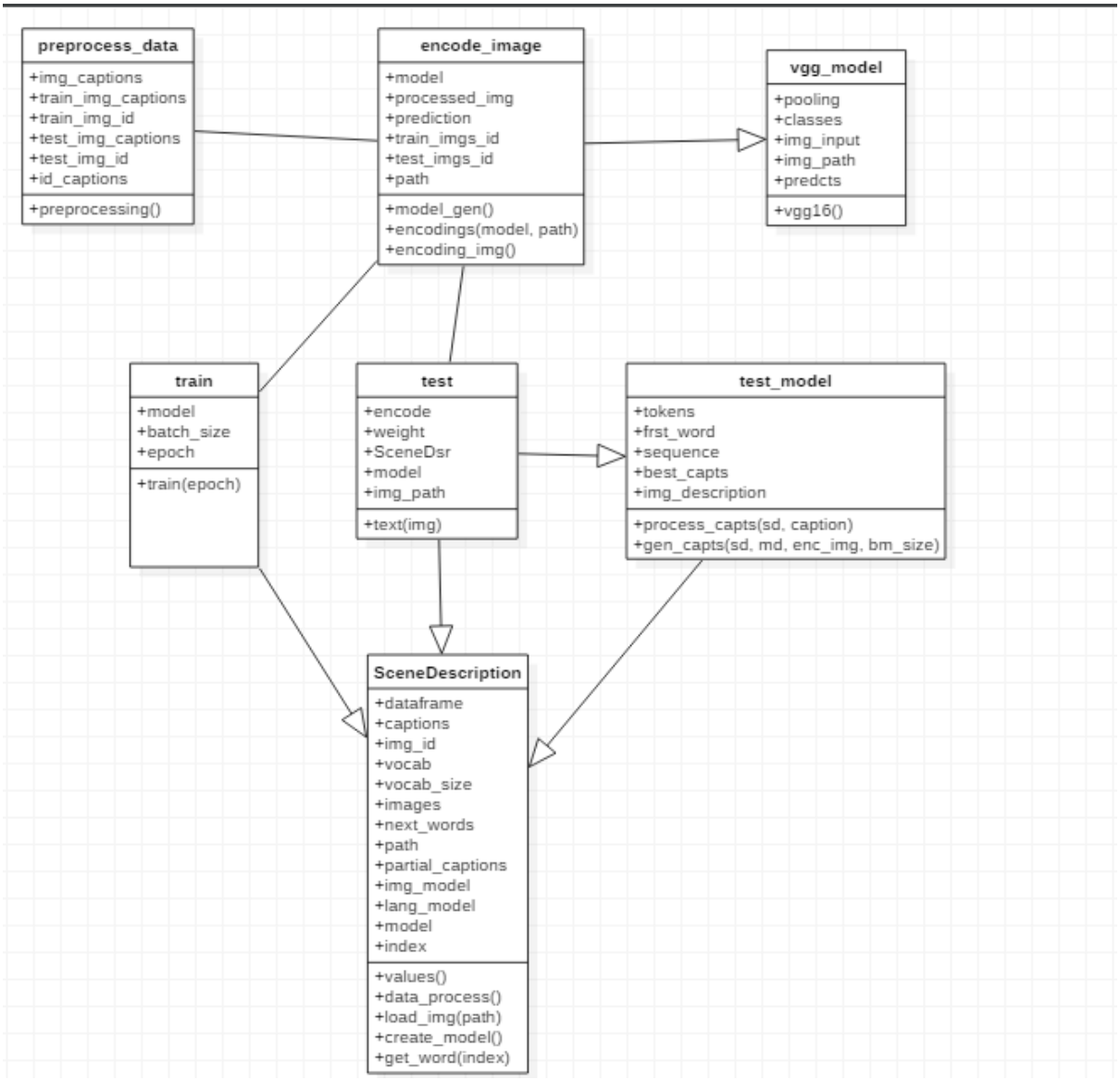

Fig. 4.4: Class diagram

4.5 Use case diagram

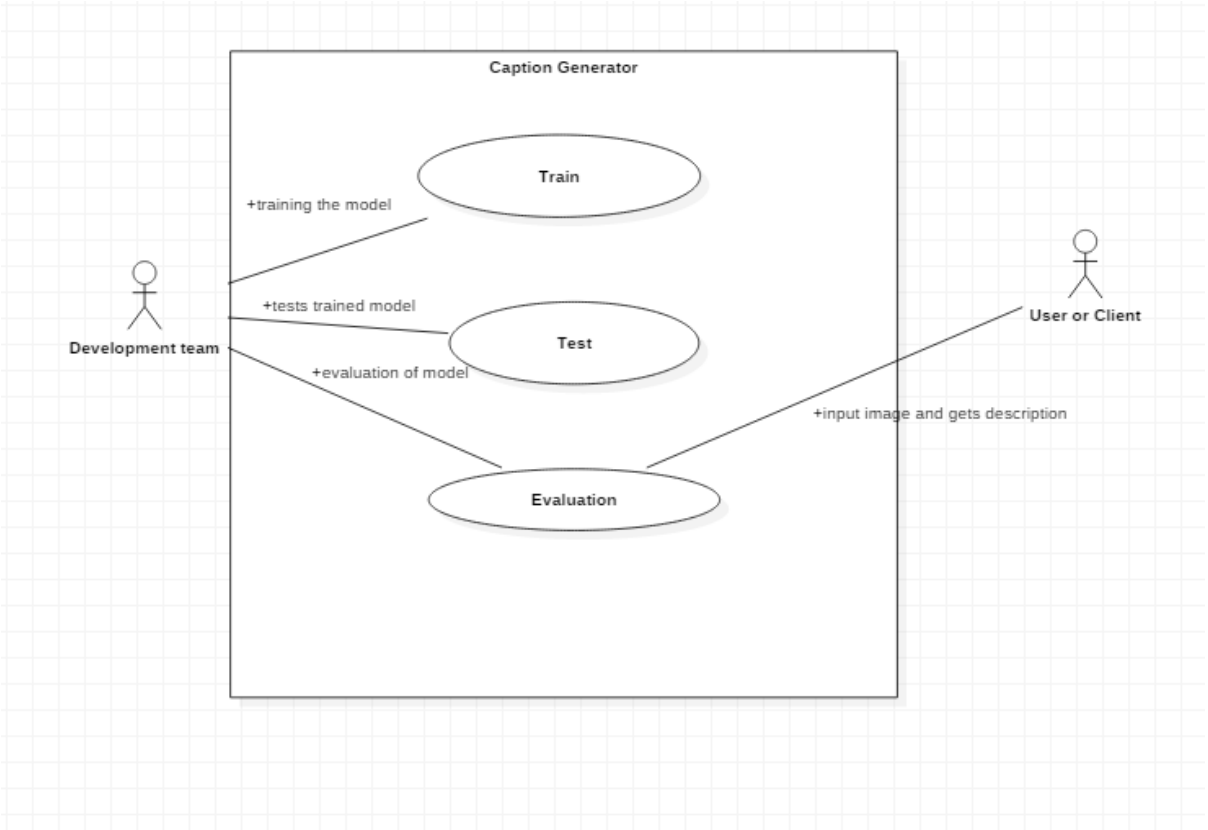

Fig. 4.5: Use case diagram

4.6 Deployment diagram

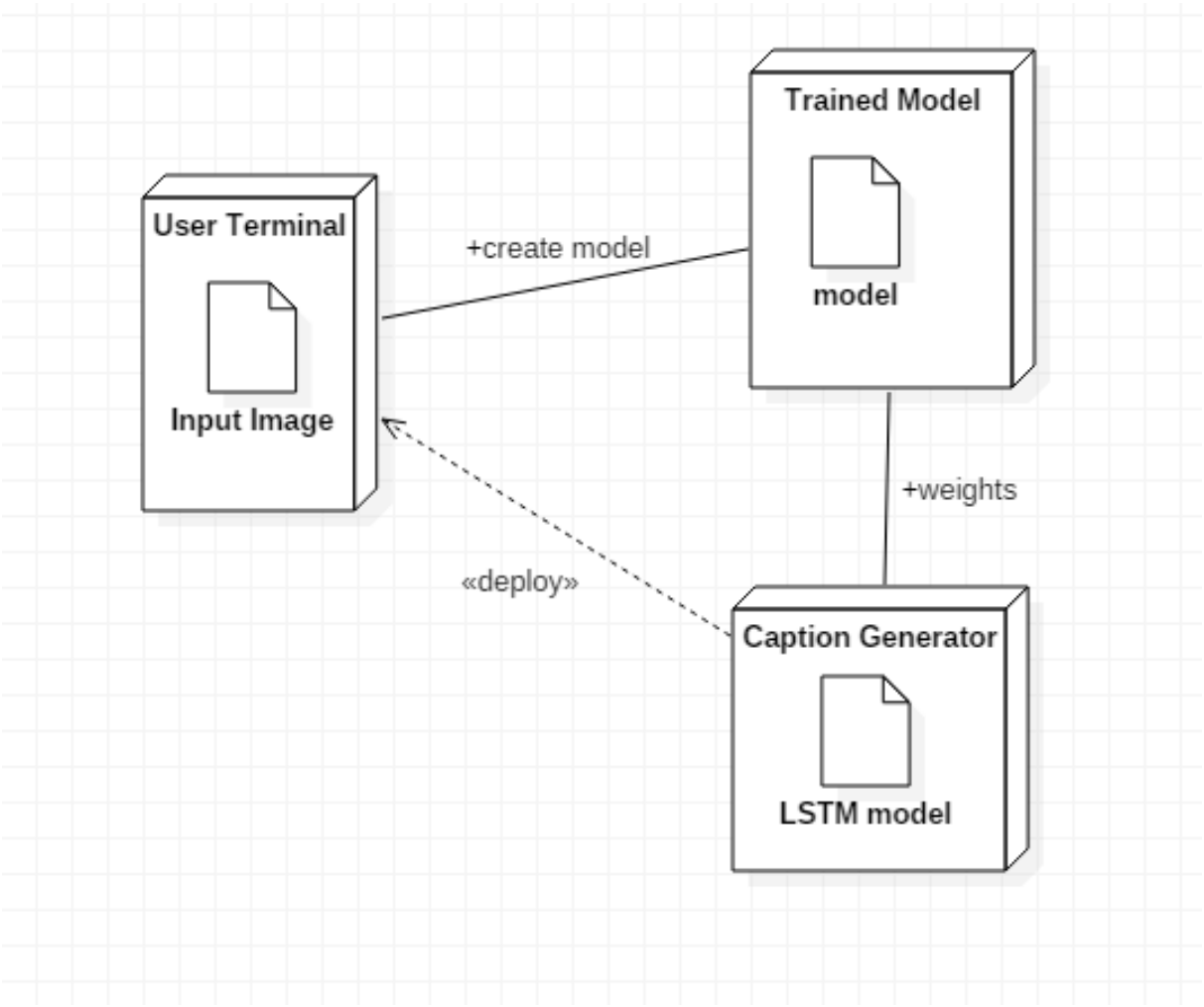

Fig. 4.6: Deployment diagram

4.7 Constraints

4.7.1 Design Constraints

Since WxPython is used to render the GUI, the application is constrained by it. We cannot run it on systems which do not support this feature. Another constraint is that, our application requires high end GPUs which enable faster processing of neural network. Due to monetary reasons we use only CPUs.

4.7.2 User Constraints

- A user must provide the image only with image supporting formats like JPEG, PNG.
- Image size should be greater or equal to 49 x 49, less than this image size application will not provide an acceptable description and it may throw an exception.
- Input image should not be blurred, blurred images cannot be correctly classified by the CNN Neural Network. By providing blurred image as input may lead to get wrong description.

CHAPTER 5

IMAGE CLASSIFICATION

For the task of image captioning we first have to determine a fit model for the task of encoding the image. Correctly classifying the image is main functionality before we going to get an description of the corresponding image.

5.1 K-nearest neighbor classifier (KNN)

Suppose we have a training set of 50,000 images divided into 10 different 'labels' and we wish to label the remaining 10,000. The k - nearest neighbor classifier takes a images which are declared as test images, matches with every single image in the training dataset images, and predicts the output description based on the higher probability. A very simple method is used to compare the images - they are compared pixel by pixel and the difference in values is summed up. In other words, given two images and representing them as vectors I1 and I2, the L1 (or Manhattan) distance between them can be calculated as:

$$d_1\ (I1, I2) = \Sigma_p |I_1^p - I_2^p|$$

Test Images-

10	23	45	22
41	100	34	52
36	49	90	150
33	20	200	110

Training Images-

19	17	23	167
23	56	49	112
45	34	155	109
67	22	44	12

Pixel-wise differences-

9	6	22	145
18	44	15	60
9	15	65	41
34	2	156	98

Table. 5.1 Pixel-wise Difference

$d_1 = 739$

Alternatively, the L2 (or Euler) distance can be used to compare images:

$$d_2\ (I_1, I_2) = \sqrt{\Sigma\ (I_1^p - I_2^p)^2}$$

Fine-tuning of hyper parameters:

The K-nearest neighbor classifier need a setting fork to classify the images. TK-nearest neighbour uses the distance functions to find out the objects on the image. Our goal is to find

the best such values of the hyper parameters so as to maximize the accuracy of our classifier. One would think that an easy way to achieve this would be to try out all possible values of k and pick the one that gives us maximum accuracy on our test data. However, this method should never be used to pick hyper parameters. Using test data to pick hyper parameters results in overfitting i.e. Our model's results will be too optimistic with respect to what we might actually observe when we deploy your model. In other words, it might fail to generalize to other data. To overcome this, we used 'cross-validation'. The training data is divided into several 'folds', one fold is used as the 'test fold' and the other folds are used as training folds. Ex, in 5-layers cross-validation, split the training data into 5 equal layers, we will use 4 of them for training, and 1 for validation. Then iterate over which layer validation layer is then evaluate the performance of model, finally take the avg the performance across the all non-similar layers.

KNN in practice - its pros and cons:

One advantage of KNN is that it is very easy to implement. However, what we save on implementation time, we lose in computation time later on. KNN classifier does not take time to train the model, it should need is storing and indexing the training data. However, its takes the computational cost at the test the model. Since matching the test images with the each training images takes time. This is backwards, since in practice we often care about the test time, efficiency much more than the efficiency at training time. For example, the L2 distance between the following images is the same:

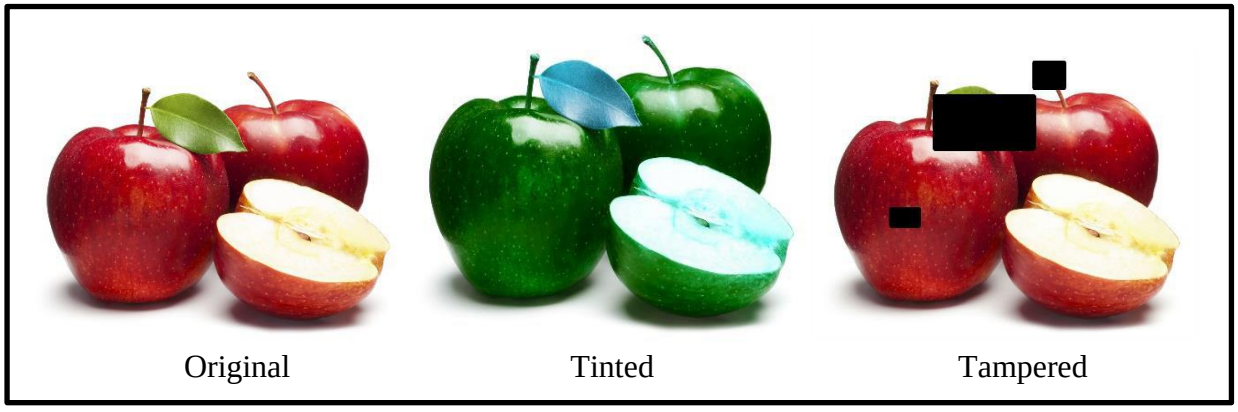

Fig. 5.1: Same L1 distance of three diff. images

5.2 Linear Classifier

In this section, we explore a more powerful approach to image classification that eventually extends to entire Neural Networks and Convolutional Neural Networks.

Linear classifiers are an example of parametric classifiers components, those are:

- **Score function:** Checks and matches the raw data from an model to class score of generated by the class score. This helps to find out the score of the correct classes.
- **Loss function:** Checks and qualifies the bond between the model predicted score and the ground truth labels. This helps to find out the loss in the model.

Minimizing the loss in the neural network is helps to get higher accuracy in the model. If the Score function values increase by the higher values than loss automatically gets reduced by the same value.

Score function

Let's assume a training dataset of images $xi \in RD$, each associated with a label y_i. Here i =1…N and $y_i \in 1…K$. That is, we have **N** examples and **K** distinct categories. For example, in the CIFAR dataset we have a training set of **N** = 50,000 images, each with **D** = 32 x 32 x 3 = 3072 pixels, and **K** = 10, since there are 10 distinct labels (dog, cat, car, etc.). We will now define the score function $f: R_D \rightarrow R_K$ that maps the raw image pixels to class scores.

$$f(x_i, W, b) = Wx_i + b$$

Where W is the weight matrix and B is the bias vector. W has dimensions 10x3072, xi has dimensions 3072x1 and b has dimensions 10x1. Essentially, each row of W acts as a classifier for one class of y. Example- say we have a 4-pixel image that needs to be classified into one of 3 classes. The image will be stretched out into a 12x1 column vector, multiplied with a 3x12 weight matrix and added to a bias vector of dimension 3x1. The result will be a 3x1 column vector where each row represents the image score for that class.

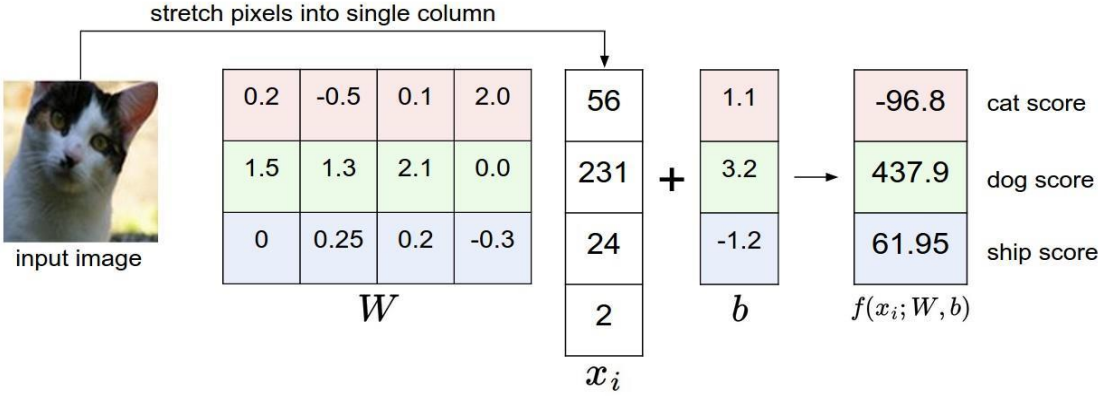

Fig. 5.2.1: Visualization of score function with 4 pixels

Another way to interpret a linear classifier is looking at each image as a point in a high-dimensional space. CIFAR-10 dataset is used, training and testing images are 3072 dimensional space of 32 x 32 x 3 pixels. Analogously, we could say that the entire dataset is a (labeled) set of points, and the linear classifier is drawing gradients in the direction of decreasing similarity with a given class of objects.

Yet another way to interpret linear classifiers is that each row of the weight matrix W corresponds to a template for one of the classes. This is quite similar to nearest neighbor search, except for the fact that one 'template' is being constructed per class instead of comparing the test image to 1000 of images in each class.

Fig. 5.2.2: Templates of weights generated by linear classifier

Loss function

Loss functions are used to measure the discrepancy between the model's prediction and the desired output. In other words, the loss function quantifies our unhappiness with predictions on the training set. Intuitively, the value of the loss function will be higher if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

Loss function is most important to get how much loss we have in our neural network and how much amount of loss we needed to reduce to get higher accuracy. Loss is higher when we have not proper dataset or the proper algorithm to get train our model.

How can we get rid of higher loss?

- By using the proper dataset
- By eliminating the noisy parameters in the dataset
- By implementing strong algorithm to handle some noisy data

5.2.1 Hinge Loss (SVM Classifier)

Hinge loss is set up so that it "wants" the correct class for each image to a have a score higher than the incorrect classes by some fixed margin Δ.

The score function takes the pixels and computes the vector $f(x_i, W)$ of class scores. For example, the score for the j-th class is the j-th element: $s_j = f(x_i, W)_j$. The Multiclass SVM loss for the i-th example is then formalized as follows:

$$L_i = \Sigma_{i \neq y} \max(0, s_j - s_{yi} + \Delta)$$

For example, suppose that we have three classes that receive the scores s= [13, −7, 11], and that the first class is the true class (i.e. $y_i = 0$). Take $\Delta = 10$. The expression above sums over all incorrect classes ($j \neq y_i$), so we get two terms:

$$L_i = \max(0, -7 - 13 + 10) + \max(0, 11 - 13 + 10)$$

The first term gives zero since [-7 - 13 + 10] gives a negative number, which is then threshold to zero with the max() function. We get zero loss for this pair because the correct class score (13) was greater than the incorrect class score (-7) by at least the margin of 10.

5.2.2 Cross Entropy Loss (SoftMax Classifier)

The SoftMax classifier uses a variant loss function- namely, the cross-entropy loss. Unlike the SVM which acts the outputs $f(x_i, W)$ as (Uncalibrated and possibly difficult to interpret) scores for each class, the Softmax classifier gives a lightly more intuitive output (normalized class probabilities) and also has a probabilistic interpretation. Instead of interpreting each row of $(x_i,)$ as the score for that particular class, we interpret it as the probability of the image belonging to that class. The cross-entropy loss has the form-

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j f_j}\right),$$

$$or\ L_i = -f_{y_i} + \log \sum_j f_j$$

Where we are using the notation f_j to mean the j-th element of the vector of class scores. As before, the full loss for the dataset is the mean of L_i over all training examples together with a regularization term R(W) (please see next section). The function $(z) = e^{z_j} / \Sigma e^{z_k}$ is called the SoftMax function.

5.2.3 SVM vs SoftMax

Unlike the SVM which computes Uncalibrated and not easy to interpret scores for all classes, the SoftMax classifier allows us to compute “probabilities” for all labels. For example, given an image the SVM classifier might give us scores [12.5, 0.6, -23.0] for the classes “cat”, “house” and “dog”. The SoftMax classifier can instead compute the

probabilities of the three labels as [0.9, 0.09, 0.01], which allows us to interpret its confidence in each class.

The SVM does not care about the details of the individual scores: if they were instead [10, -100, -100] or [10, 9, 9] the SVM would be indifferent since the margin of delta = 1 is satisfied and hence the loss is zero. However, these scenarios are not equivalent to a SoftMax classifier, which would accumulate a much higher loss for the scores [10, 9, 9] than for [10, -100, -100]. In other words, the Softmax classifier has been never fully happy with the scores it produces: the correct class could always have a higher probability and the incorrect classes always a lower probability and the loss would always get better. However, the SVM is happy once the margins are satisfied and it does not micromanage the exact scores beyond this constraint. This can be thought of as a feature: For example, a 'dog classifier' should be spending most of its "effort" on the difficult problem of classifying different breeds of dogs, and should completely ignore the cat examples, which it already assigns very low scores to, and which likely cluster around a completely different side of the data cloud.

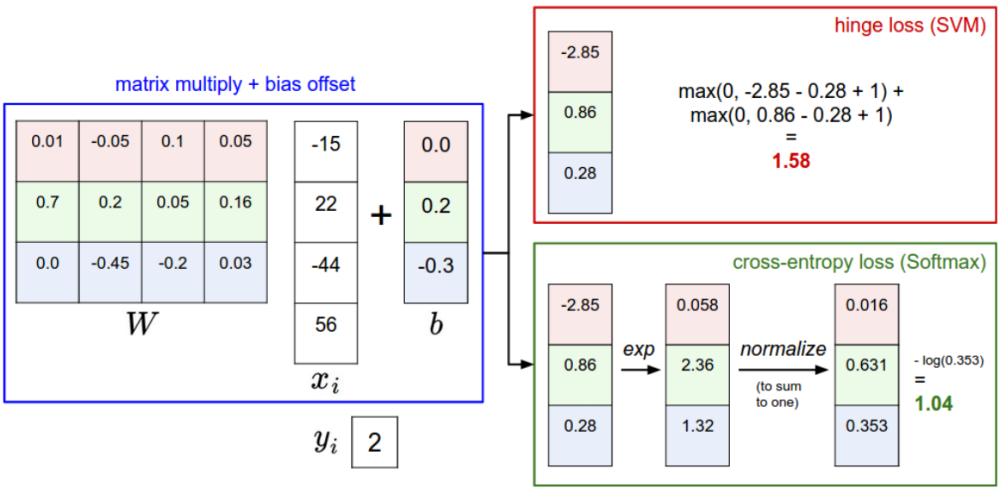

Fig. 5.2.3: Score function comparison of SoftMax and SVM

Regularization

Suppose that we have a data and parameters correctly classify every sample. The issue is that this set of W is not necessarily unique: there might be many similar W that correctly classify the examples. Different values of W don't affect the loss function - the loss function yields a score of zero for all W. In such situations, we must have some

criteria to choose from the given set of W. This can be solved by using loss function with regularization R(W).

$$R\,(W) = \Sigma_k \Sigma_l W^2_{k,\,l}$$

In the expression above, we are summing up all the squared elements of W. For example, suppose that we have some input image vector x=[1,1,1,1] and two weight vectors w1=[1,0,0,0] and w2=[0.25,0.25,0.25,0.25]. Let's ignore the bias for the sake of simplicity. Then $w_1^T x = w_2^T x = 1$ so both weight vectors lead to the same dot product,

but the L_2 penalty of w_1 is 1.0 while the L_2 penalty of w_2 is only 0.25. Intuitively, this is because the weights in w_2 are smaller and more diffuse. Since the L_2 penalty prefers smaller and weight vectors, the final classifier is encouraged to take into account all input dimensions to small amounts rather than a few input dimensions and very strongly. This effect can improve in the generalization high performance of the classifiers on test images and lead to less overfitting.

Optimization (Stochastic Gradient Descent)
SVM loss function:

$$L_i = \sum_{j \neq y_i} \max(0,\, w_j^T x_i - w_{y_i}^T x_i + \Delta)$$

Differentiate the function with respect to the weights. For example, taking the gradient with respect to wyi we obtain:

$$\nabla w_{y_i} L_i = -(\sum_{j \neq y_i} (w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)) x_i$$

Where 1 is the indicator function that is one if the condition inside is true or zero otherwise. This means that we simply count the number of classes that did not met the desired margin and then scale the data vector xi by this number. This gives us a way to calculate the gradient analytically. But this is the gradient only with respect to the row of W that corresponds to the correct class. For the other rows where $j \neq y_i$ the gradient is:

$$\nabla w_j L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)) x_i$$

Once we have calculated the gradient, it is straight-forward to implement the expressions and use them to perform the gradient update:

$$W = W - mu * gradient,$$

Where mu is the step size.

5.3 Convolution Neural Network (CNN)

CNN (**ConvNets** or **CNNs**) are a category of ANN which have proven to be very effective in the field of image recognition and classification. They have been used extensively for the task of object detection, self-driving cars, image captioning etc. A basic convnet is shown in the fig. below

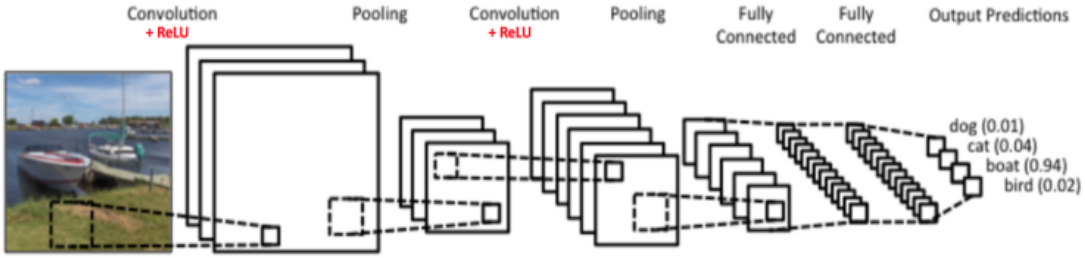

Fig. 5.3.1: A simple convnet architecture

The entire architecture of a convnet can be explained using four main operations namely,

1. Convolution
2. Non- Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

These operations are the basic building blocks of *every* Convolutional Neural Network, so understanding how these work is an important step to developing a sound understanding of ConvNets. We will discuss each of these operations in detail below.

Essentially, every image can be represented as a matrix of pixel values. An image from a standard digital camera will have three channels – red, green and blue – you can imagine those as three 2d-matrices stacked over each other (one for each color), each having pixel values in the range 0 to 255.

Digital camera have three channels red, green and blue each channel has its own value to be represented in the bits, each channel is different in the cinematic looks. These channels are useful to 2d-array to identify some of the objects by making an invert of

the current image color. Inverted images are odd in the position where high contrast and light overlap or darkness in the images.

Three color channels helpful to recognize the individual aspects of an image in some certain moment with accurate mapping of light over the image.

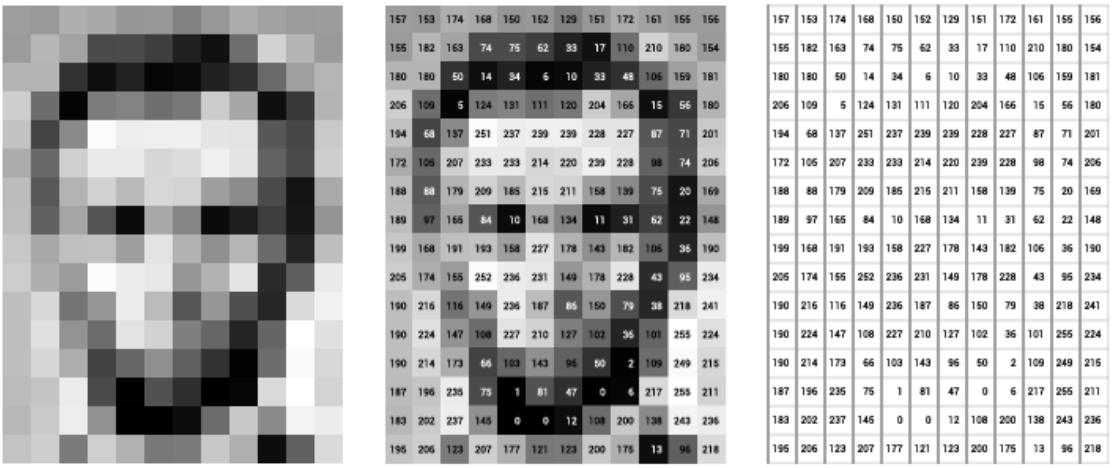

Fig. 5.3.2: A grayscale image as matrix of numbers

Convolution Operator

The purpose of convolution operation is to extract features from an image. We consider filters of size smaller than the dimensions of image. The entire operation of convolution can be understood with the example below.

Consider a small 2-dimensional 5*5 image with binary pixel values. Consider another 3*3 matrix shown in Fig. 5.3.3.

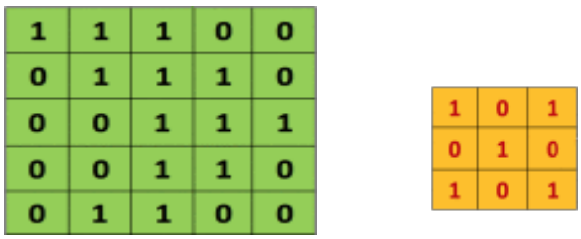

Fig. 5.3.3: Image (in green) and Filter (in orange)

We slide this orange 3*3 matrix over the original image by 1 pixel and calculate element-wise multiplication of the orange matrix with the sub-matrix of the original image and add the final multiplication outputs to get the final integer which forms a single element of the output matrix which is shown in the Fig. 8 by the pink matrix.

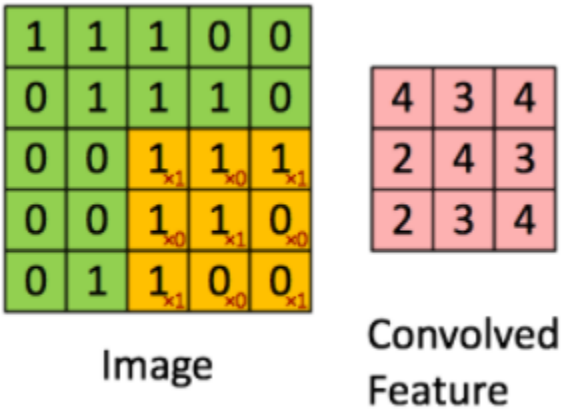

Fig. 5.3.4: Convolution operation

Introducing Non-Linearity

An additional operation is applied after every convolution operation. The most commonly used non-linear function for images is the ReLU which stands for Rectified Linear Unit. The ReLU operation is an element-wise operation which replaces the negative pixels in the image with a zero.

Since most of the operations in real-life relate to non-linear data but the output of convolution operation is linear because the operation applied is elementwise multiplication and addition. The output of the ReLU operation is shown in the figure below.

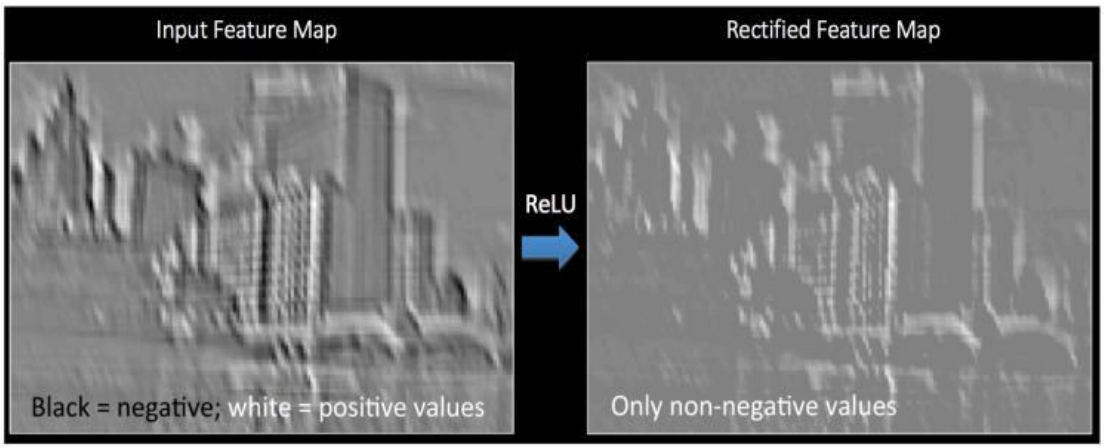

Fig. 5.3.5: Output after a ReLU operation

Some other commonly used non-linearity functions are sigmoid and tanh.

Spatial Pooling

The pooling operation reduces the dimensionality of the image but preserves the important features in the image. Max pooling is the most common used pooling type. In max pooling you slide a window of n*n where n is less than the side of the image and determine the maximum in that window and then shift the window with the given stride length. The complete process is specified by the fig. 5.3.6.

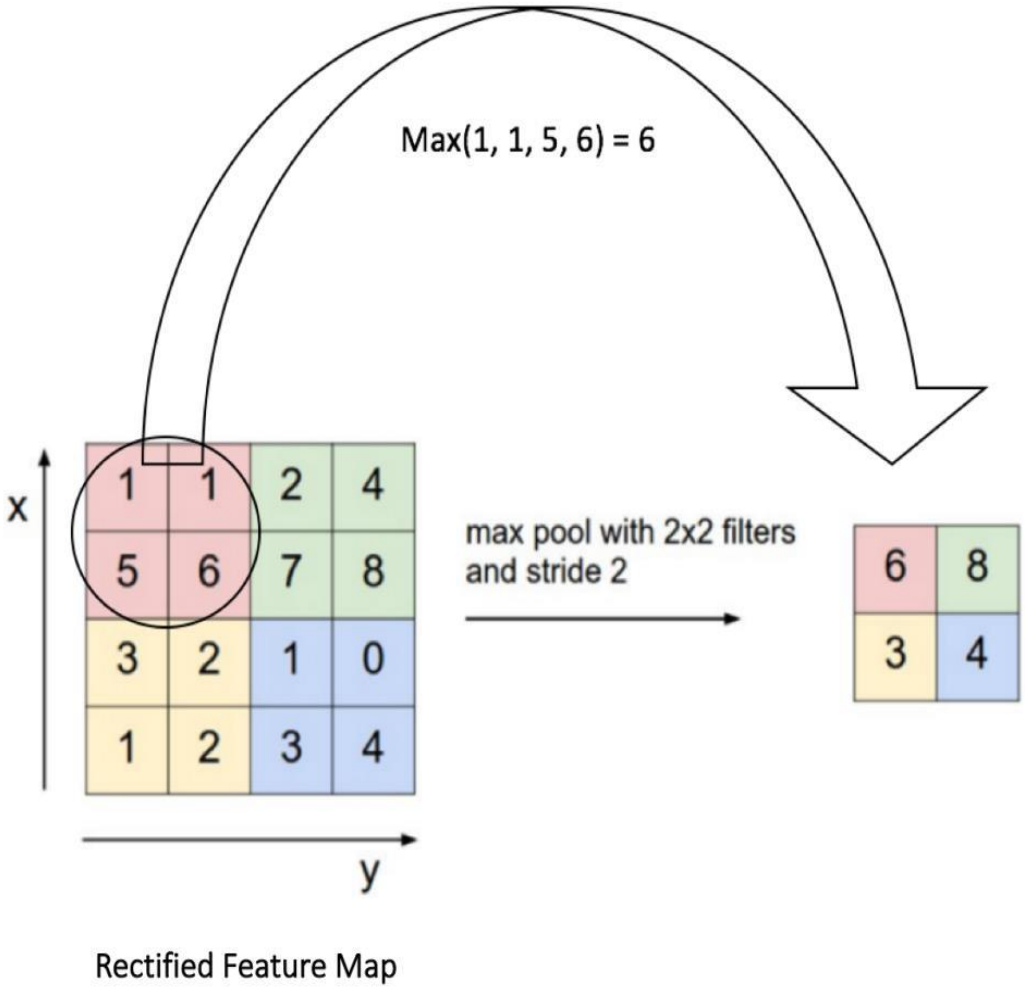

Fig. 5.3.6: Max pooling operation

Fully-Connected layer

The convolution and pooling operation generate features of an image. The task of the fully connected layer is to map these feature vectors to the classes in the training data.

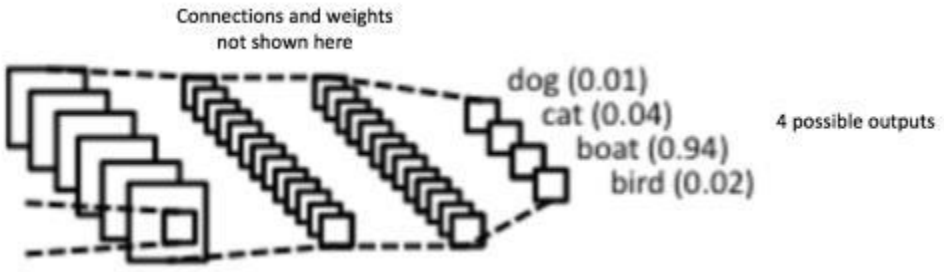

Fig. 5.3.7: An example of fully connected layer of data with 4 classes

The task of image classification on cifar-10 has shown state of the art results with the use of ConvNets. We use the Alex net architecture proposed by Alex krizhevsky with

a few tweaks. Alexnet is trained for images having 224*224 dimensions and hence need to be modified to be used for cifar-10 since the images in cifar-10 are 32*32. The model

used by us has alternate layers of convolution and non-linearity's. We use a fully connected layer at the end which uses SoftMax activation to give the scores of the 10 classes present in the cifar-10 dataset.
The dataset on these ConvNets yield an accuracy of 85% within around 1.5 hrs of training on GPU's. The plots of loss and accuracy on test and validation set are shown in the figures below.

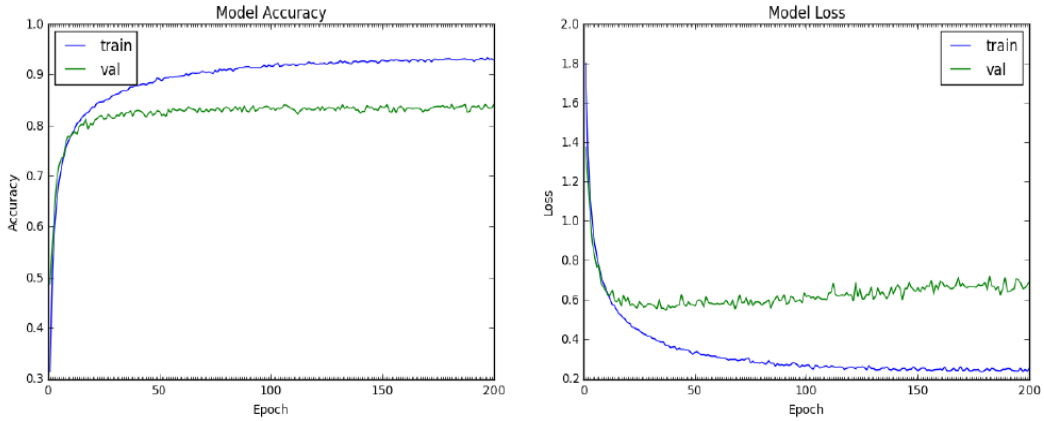

Fig. 5.3.8: Accuracy and loss plot on training and validation set

The model is built using tensorflow. Tensorflow is an open source library developed by Google brain team for machine learning. Though being a python API, most of the code of tensorflow is written in C++ and CUDA which is nvidia's programming language for gpus. This helps tensorflow in faster execution of code since python is slower than CPP. Also, the use of gpu enhances the performance of the code significantly.

CHAPTER 6

IMAGE DESCRIPTION MODEL

6.1 Model Overview

The model proposed takes an image *I* as input and is trained to maximize the probability of p(S|I) where S is the sequence of words generated from the model and each word St is generated from a dictionary built from the training dataset. The input image I is fed into a deep vision Convolution Neural Network (CNN) which helps in detecting the objects present in the image. The image encodings are passed on to the Language Generating Recurrent Neural Network (RNN) which helps in generating a meaningful sentence for the image as shown in the fig.6.1.1. An analogy to the model can be given with a language translation RNN model where we try to maximize the p(T|S) where T is the translation to the sentence S. However in our model the encoder RNN which helps in transforming an input sentence to a fixed length vector is replaced by a CNN encoder. Recent research has shown that the CNN can easily transform an input image to a vector.

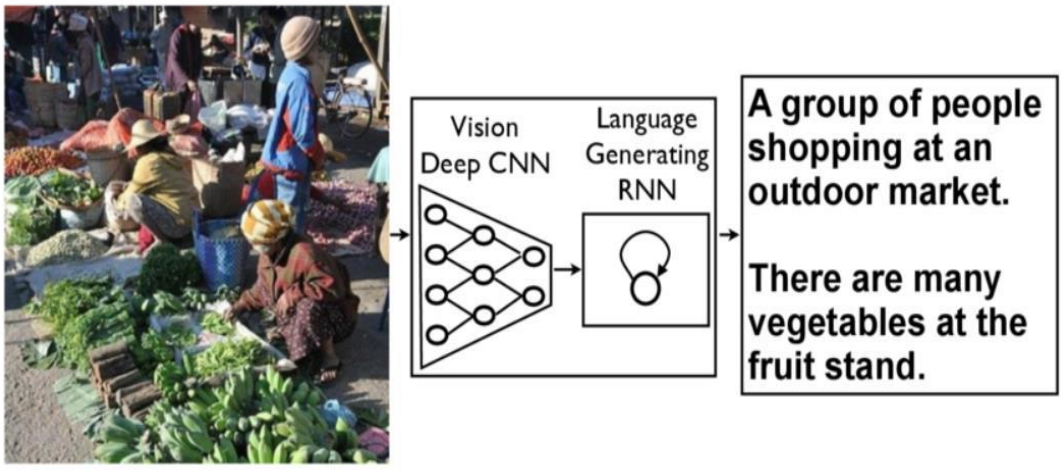

Fig. 6.1: An overview of the image captioning model

For the task of image classification, we use a pre-trained model VGG16. The details of the models are discussed in the following section. A Long Short-Term Memory (LSTM) network follows the pre-trained VGG16. The LSTM network is used for language generation. LSTM differs from traditional Neural Networks as a current token is dependent on the previous tokens for a sentence to be meaningful and LSTM networks take this factor into account. In the following sections we discuss the components of the model i.e. the CNN encoder and the Language generating RNN in details.

6.2 Dataset

For the task of image captioning we use Flickr8k dataset. The dataset contains 8000 images with 5 captions per image. The dataset by default is split into image and text folders. Each image has a unique id and the caption for each of these images is stored corresponding to the respective id.

The dataset contains 6000 training images, 1000 development images and 1000 test images. A sample from the data is given in fig. 6.2.

- Two dogs play in the grass.
- Two small white dogs playing on short grass.
- Two white dogs are playing on the grass.
- Two white dogs play in the grass.
- Two white dogs play on the green grass.

Fig. 6.2: Sample image and corresponding captions from the Flickr8k dataset

Other datasets like Flickr30k and MSCOCO for image captioning exist but both these datasets have more than 30,000 images thus processing them becomes computationally very expensive. Captions generated using these datasets may prove to be better than the ones generated after training on Flickr8k because the dictionary of words used by RNN decoder would be larger in case of Flickr30k and MSCOCO.

Choosing the suitable and larger Dataset is the most important work for the neural network developers. Neural network is based on what data we are providing to learn. So while selecting the dataset for neural network we need figure out what kind of dataset is needed and which dataset is suitable to get train the model. Good dataset can make the good neural network.

Prepare the dataset as you needed before you are using it in neural network by removing unrelated factors or parameters by analyzing the requirement of the neural network.

6.3 Deep CNN Architecture

The details of the CNN were discussed in section 5.3. Convolutional Neural Network (CNN) have improved the task of image classification significantly. ImageNet Large Scale Visual Recognition competition(ILSVRC) have provided various open source deep learning frameworks like ZFnet, Alexnet, Vgg16, Resnet etc. For the task of image encoding in our model we use Vgg16 which is a 16-layered network proposed in ILSVRC 2014 [2]. VGG16 significantly decreased the top-5 error rate in the year 2014 to 7.3%.

The image taken for classification needs to be a 224*224 image. The only preprocessing done is by subtracting the mean RGB values from each pixel determined from the training images.

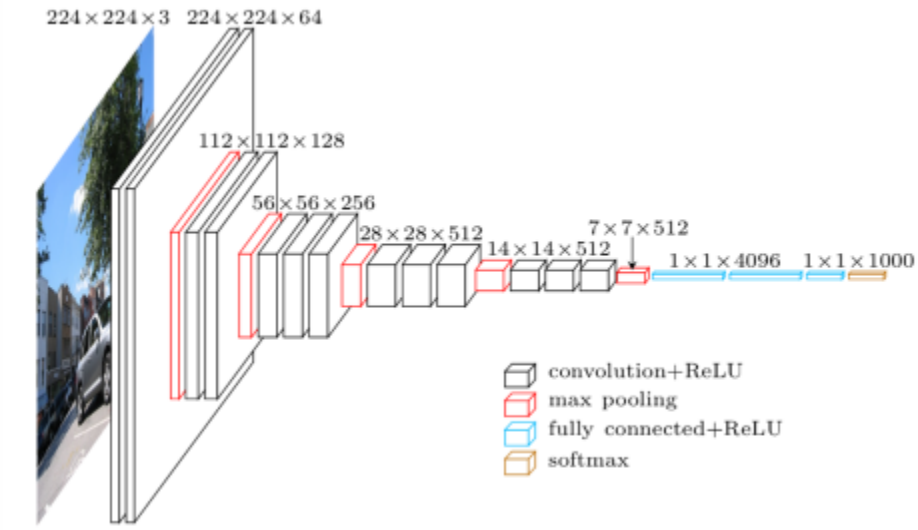

Fig. 6.3.1: VGG16 architecture

The convolution layer consists of 3*3 filters and the stride length is fixed at 1. Max pooling is done using 2*2-pixel window with a stride length of 2. All the images need to be converted into 224*224-dimensional image. A Rectified Linear Unit (ReLU) activation function is follows every convolution layer. A ReLU computes the function (x)=max $(0,x)$. The output of the ReLU function is given below:

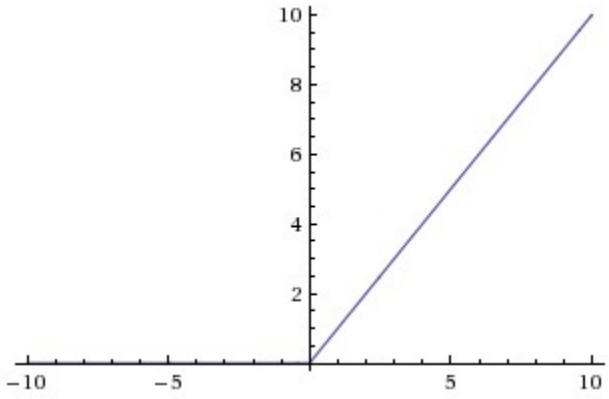

Fig. 6.3.2: Rectified linear unit activation function

The advantage of using a ReLU layer over sigmoid and tanh is that it accelerates the stochastic gradient descent. Also unlike the extensive operations (exponential etc.) the ReLU operation can be easily implemented by thresholding a matrix of activations at zero. For our purpose however, we need not classify the image and hence we remove the last 1*1*1000 classification layer.

The output of our CNN encoder would thus be a 1*1*4096 encoded which is then passed to the language generating RNN. There have been more successful CNN frameworks like ResNet but they are computationally very expensive since the number of layers in ResNet was 152 as compared to VGG16 which is only 16-lauered network. A comparison between the layers vs top-5 error rate in ILSVRC challenge is given below:

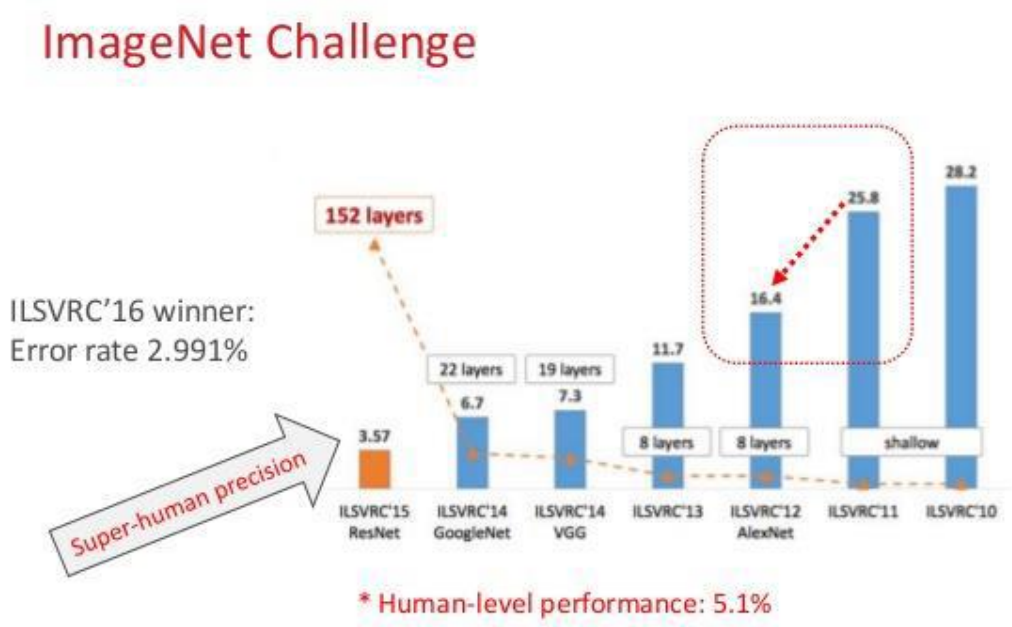

Fig. 6.3.3: Top-5 error rate vs the no. of layers

6.4 Recurrent Neural Net (RNN) Decoder Architecture

Recurrent neural nets are a type of artificial neural network in which connection between units form a directed cycle. The advantage of using RNN over conventional feed forward net is that the RNN can process arbitrary set of inputs using its memory. RNNs were discovered in the year 1980 by John Hopfield who gave the famous Hopfield model. Recurrent neural nets in simple terms can be considered as networks with loops which allows the information to persist in the network.

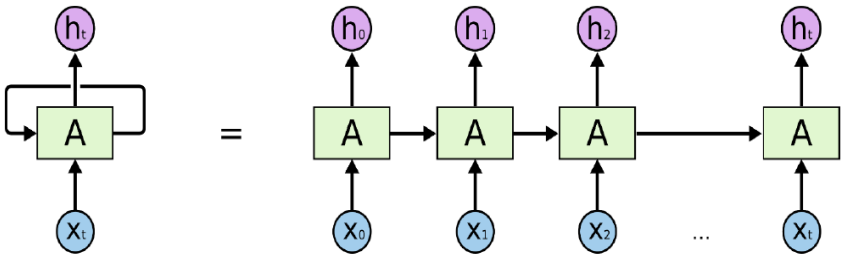

Fig. 6.4.1: A simple neural network unrolled into simple neural net

As shown in the figure above a recurrent neural network can be considered as multiple copies of same network with each network passing the message to its successor.

One of the problems with RNNs is that they do not take long-term dependencies into account. Consider a machine that tries to generate sentences on its own. For instance, the sentence is “I grew up in England, I speak fluent English”, if the machine is trying to predict the last word in the sentence i.e. *English*, the machine needs to know that the language name to be followed by fluent is dependent on the context of the word England. It is possible that the gap between the relevant information and the point where it is needed becomes very large in which case the conventional RNNs fail.

To overcome the above-mentioned problem of “long term dependencies”, Hoch Reiter and Schmidhuber provided the LSTM networks in the year 1997. Since then LSTM networks have revolutionized the fields of speech recognition, machine translation etc. Like the conventional RNNs, LSTMs also have a chain like structure, but the repeating modules have a different structure in case of a LSTM network. A simple LSTM network is shown in Fig. 6.4.2.

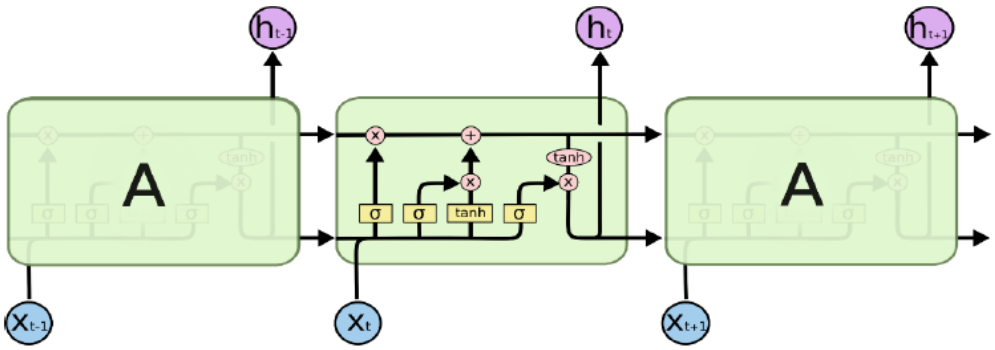

Fig. 6.4.2: Four interacting layers in a LSTM layer

The key behind the LSTM network is the horizontal line running on the top which is known as the cell state. The cell state runs through all the repeating modules and is modified at every module with the help of gates. This causes the information in a LSTM network to persist.

We use this LSTM network with a slight variation. The architecture of the LSTM network used is given below.

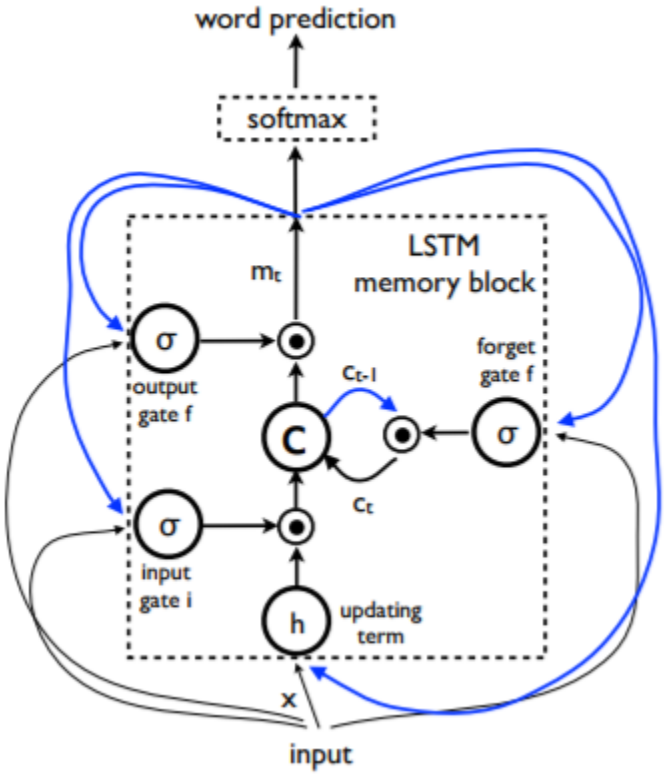

Fig. 6.4.3: LSTM architecture for language generation

The entire network is governed by the following equations

$$i_t = (W_{ix}x_t + W_{im}m_{t-1}),$$

Where i_t is the input gate at time t, W represents the trained parameters. The variable m_{t-1} denotes the output of the module at time t-1 and σ represents the sigmoid operation which outputs numbers between zero and one, describing how much of each component should be let through.

$$ft = (W_{fx}x_t + W_{fm}m_{t-1}),$$

Where ft represents the forget gate which control whether to forget the current cell value.

$$ot = (W_{ox}x_t + W_{om}m_{t-1}),$$

Where o_t represents the output gate which determines whether to output the new cell value or not.

$$ct = f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}),$$

Where c_t is the cell state that runs through all the modules and ⊙ represents the tensor product with a gate value.

$$m_t = o_t \odot c_t,$$

Where m_t is the encoded vector which is then fed into the SoftMax function.

$$p_{t+1} = (m_t)$$

The output p_{t+1} of a module gives the word prediction. The same LSTM network is repeated until an end token (.) is encountered by the network. The series of these word prediction generate the caption for a given image. The complete training process for the combined model (CNN encoder + RNN language generator) and the LSTM network in unravelled form is given below in Fig. 6.4.4.

The Long Short Term Memory (LSTM) model is trained to predict each word of the sentence after it has seen the image as well as all preceding words as defined by $p(St|I, S0, \ldots, St-1)$.

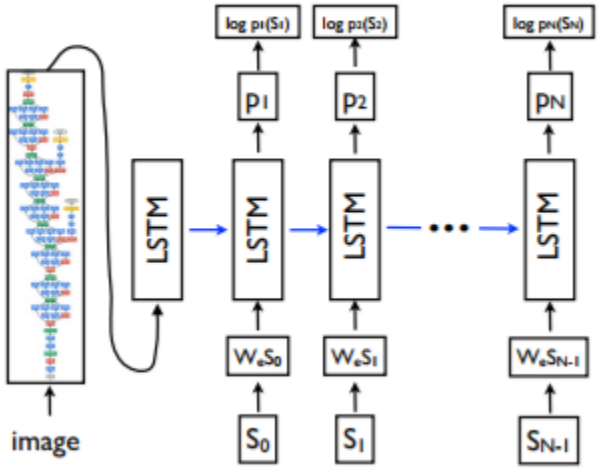

Fig. 6.4.4: Working of the image captioning model

CHAPTER 7

SYSTEM REQUIREMENTS SPECIFICATION

7.1 FUNCTIONAL REQUIREMENTS

7.1.1 Description generator

The input will be an image and the output would be the description of the image with the textual English language sentence. At the final step, we need a way to add the information of one image with the description. This is the final step in which the description of the input image is provided at the output.

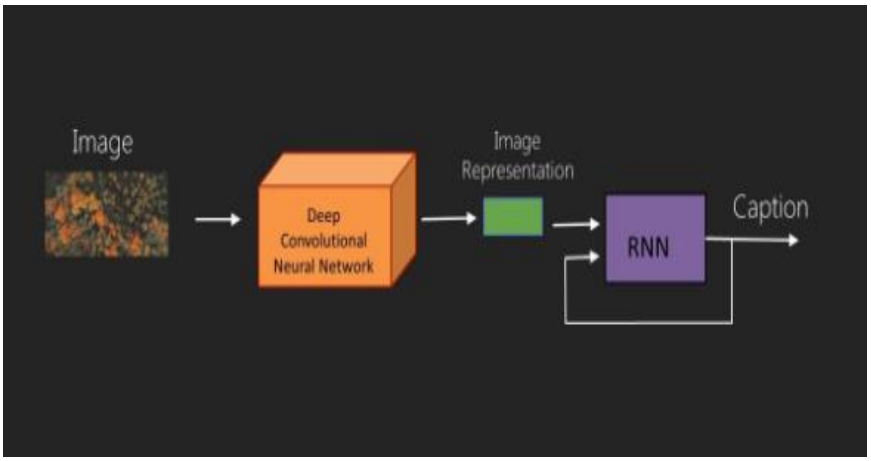

Fig. 7.1.1: Description generator

7.1.2 Model Retrain

Learning of neural network is based on the repeated trainings, neural networks learns from the previous training and improves in the network by feeding the previous trained model weights. Retraining the neural network is better way to get improvement in accuracy of the network.

Here we are allowing to retrain the model by feeding the previous model weights it is one of the way to improve accuracy in the neural network. There is no constraints to stop the training model that's dependent on the user or developer until they get the acceptable error rate(minimum error rate, maximum accuracy).

7.2 NON-FUNCTIONAL REQUIREMENTS

7.2.1 Dependencies:

- **Windows or Linux OS:** We need a Windows or a Linux operating system which can support CPU drivers. We would prefer Windows 10 or Ubuntu 16+ for this application.
- Keras
- Tensorflow
- Python 2.7
- Anaconda 2.5
- Numpy
- Scipy
- Pandas

7.2.2 User Requirements

There are two kinds of users for the proposed application:

- **Developer:** Developer is allowed to retrain the model by whatever epoch developer wants to.
- **End User:** User is allowed get description of the images by providing the image input.

7.3 HARDWARE REQUIREMENT

PARAMETERS	SPECIFICATIONS
CPU ARCHITECTURE	X86_64
SYSTEM MEMORY	8-32 GB
CPU'S	1

Table. 7.3 Hardware requirement

7.4 SOFTWARE REQUIREMENT

PRODUCT NAME	DESCRIPTION	VERSION
UBUNTU	OPERATING SYSTEM	16.4
CPU PROCESSORS	CPU DRIVER	Intel i3
KERAS	KERAS IS A MULTIPLICATIVE API TO HANDLE LARGE COMPUTATIONS.	2.0.3
WX PYTHON	WXPYTHON IS A CROSS-PLATFORM GUI TOOLKIT FOR THE PYTHON PROGRAMMING LANGUAGE	3.0.2
PYTHON	HIGH LEVEL PROGRAMMING LANGUAGE	2.7
TENSORFLOW	OPEN SOURCE SOFTWARE LIBRARY	1.5

Table. 7.4 Software requirement

CHAPTER 8

SYSTEM DESIGN

8.1 SYSTEM ARCHITECTURE

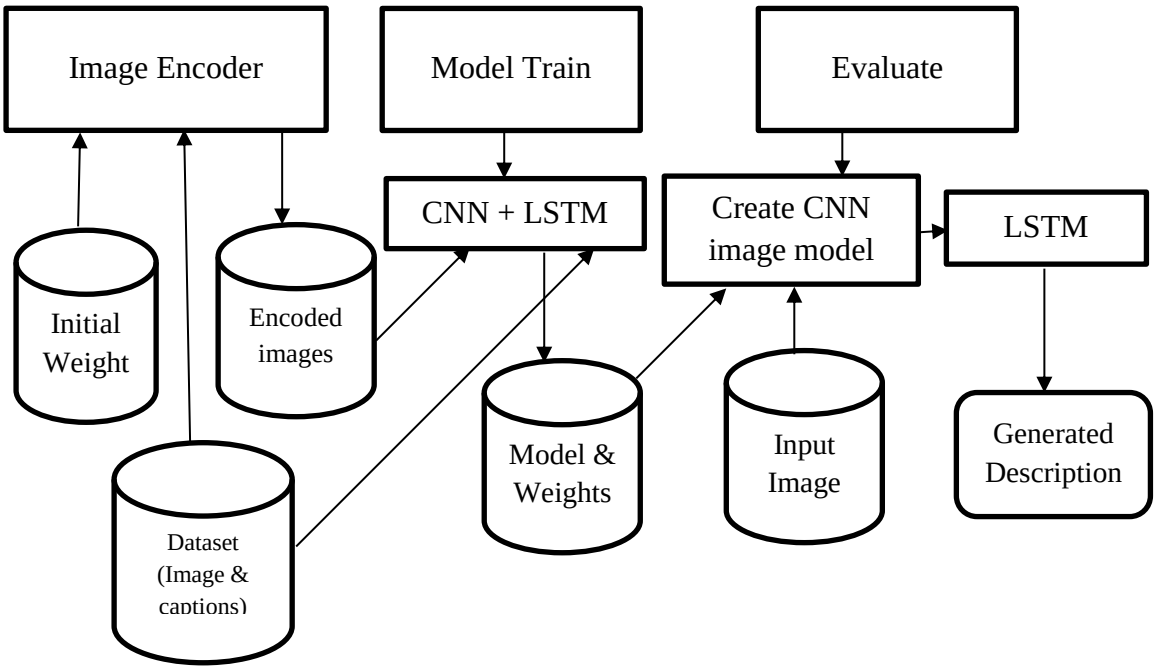

Fig.8.1: System Architecture

The system architecture contains the following components:

- **Image Encoder:**

  Image encoder is the process of encoding the images into an array which helps us to use datasets in training and evaluation. Image encoder is uses the ImageNet initial weights to encode the images. This phase generates **encoded_images.p,** here .p is the file type to store the large array data.

  - **Initial Weights**: We need initial weights to train the model, these weights are pre-trained which boosts the neural network get good encode of the image.
  - **Dataset**: Flickr8K Dataset is used for the training of the images. This dataset is contain 8000 images in that we have used 7000 images for training and 1000 images for testing/validation.

- **Encoding of images**: Encoding is transformed into constructs, representing physical features and objects. Encodes the train and test images to use further in training and testing module.

➢ Model Train:

Training the model is the most important part of the neural network, accuracy of neural network is based on the training. One thing we have to consider here is the number of epoch's to train the model. Epoch means the number of time the whole dataset is goes through our algorithm. Example One epoch means the training dataset is about 7000 images, these 7000 images are once goes through the algorithm for one epoch, and probably non gpu system's takes more than 8 hours to complete the one epoch training. If we have using GPU's training for one epoch takes 1 to 2 hours to complete the training. Here there is no constraint to use the specific amount of epoch it is purely based on the developer to when to stop the training of the neural network, we can stop the training by giving specific amount epoch when we get acceptable or higher accuracy and lower error rate.

Some factors to consider when to retrain the neural network:

- If the neural network accuracy is lesser than the loss
- If the neural network is repeatedly providing wrong output than the expected output.

Some factors to consider while training are:

- **Accuracy:** Measurement of efficiency of the trained model, we can stop the training of neural network based on this accuracy if we get better accuracy. If developer wants to achieve more accuracy they can continue training by providing some amount epoch by feeding the last trained model weights. Accuracy is the key factor to the neural network, this can give us how much our neural network is trained well. Here we consider if there is higher accuracy than the loss than that neural network is well trained. If we get 100% accuracy then we can stop the neural network training if we get accuracy about 65% to 70% then that model is to retrain by giving some amount epochs.

- **Loss:** Measurement of wrong or not correctly classified by neural network, we can retrain the model if there is higher loss in neural network. Higher loss indicates that our neural network is not trained well. Loss will be decreases when the accuracy is increases and when Loss increases accuracy of the neural network decreases. Here key factor is too well trained neural network is too maintain lower Loss in the network.

  If we are getting higher Loss in the neural network that means our neural network is not trained well. We can decrease Loss by considering some factors.

  How we can decrease the Loss in Neural Network:

  - By using the best loss vector functions.
  - By using the better dataset related to the project.
  - By using larger dataset can able to train well by decreasing the Loss.
  - Using best activation functions can help to remove wrong classifications.

- **VGG16**: One of the best neural network architecture in the image classification and object identifying. VGG16 is the convolution neural network which includes 16 layers architecture. These 16 layers are Convolution layer, Pooling layer and Fully Connected (FC) layers. Convolution layers are represents the images into objects by the help of filters. The filters are the n x m matrix which holds the numeric value in that specific grid of an image. Pooling layer are the reducing the input sizes by using the Max pooling or Average pooling. Max pooling is takes max value in the each filter and reduces the input size to the next output layer. Average pooling is takes the avg value of the each grid and reduces the input size to the next output layer. Fully connected layers are output layers of the neural network, these fully connected layers are get the input values from all the 16 layers and produces the outputs. The produced output is the classification of an image or an object. The classified class can next input to the LSTM model to produce the description of an image or an object.

VGG16 is most important to the image description if this is correctly classifies of an image or an object it'll help to get correct description according to an image.

- **LSTM**: Long Short Term Memory is the Recurrent Neural Network (RNN). RNN is purely based dependent on the previous neural network output which input to the current neural network. RNN works repeated connection because of the each output layer is depends on the previous one. LSTM is generates the English language by getting input from the VGG16 model. Takes the inputs from the VGG16 model and generates the description to the corresponding object with higher matching value.

  LSTM remembers the output of the each connected neural networks outputs sequentially and find out the next sequence which probably be the next word in the neural network. This decides the next word by possible words and finalize based on the higher probability of each possible words.

  Generated Output: The Neural network provides the English language textual description for the particular input image.

➢ Evaluation:

Evaluation phase is the finding the output description for given an image. Neural network should provide the correct description for given an image this the main feature of our Neural network.

- **Create Image Model:** Taking input an image from user and creates the VGG16 model. After creating the image model for given input an image passing through algorithm by using the trained model weights, which can help to identify the objects and passes to the LSTM model generate description.

- **LSTM:** Takes the input from the VGG16 model (CNN Image model) and generates the description by performing the predictions based on the all connected neural network output layers. LSTM holds the array of possible description it returns the higher probability description to the user.

8.2 Module Description

- **Encoding Module:**

  First module is very important to further modules they rely on the Encoding Module. Encoding of the image Flickr 8K dataset is to help full to pass the encoding images to train module. Here it encodes the images into an array.

- **Training Module:**

  Creates VGG16 model and reads the images by encoded images (image_encodings.p) file. Trains the model by feeding the captions to that particular.

- **Evaluation/Testing Module:**

  Takes the image input from the user and creates VGG16 model on that given input image and next LSTM model will generates the language for the description and predicts the description for that particular image. Here while predicting it come up with some predictions final it return which as the higher accuracy rate.

CHAPTER 9

DETAILED DESIGN

9.1 ENCODING MODULE:

- **Input:** Flickr 8K images dataset
- **Description:** First module is very important to further modules they rely on the Encoding Module. Encoding of the image Flickr 8K dataset is helpful to pass the encoding images to the train module. Here it encodes the images into an array.
- **Output:** Generates image_encodings.p
- **Data Flow Diagram:**

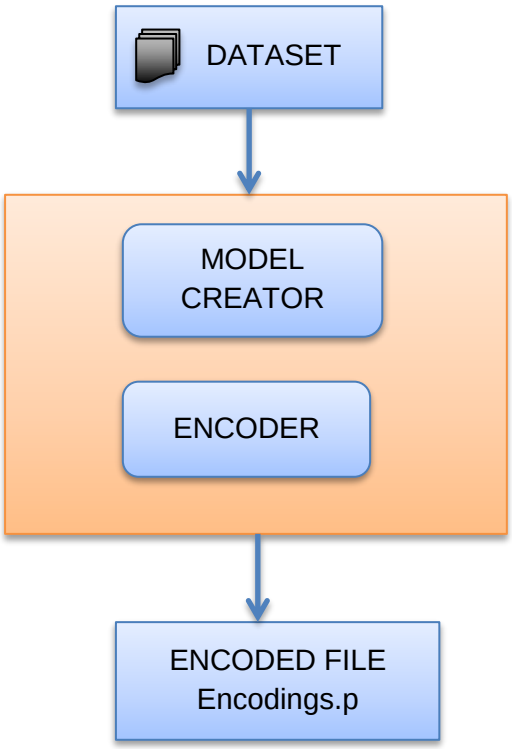

Fig. 9.1: Encoding Module

9.2 TRAINING MODULE:

- **Input:** Epoch size, ex: **python train.py 200**, here 200 is epoch.
- **Description:** Creates VGG16 model and reads the images by encoded images (image_encodings.p) file. Trains the model by feeding the captions to each particular image.
- **Output:** Generates Weights.h5 (trained model weights).
- **Data Flow Diagram:**

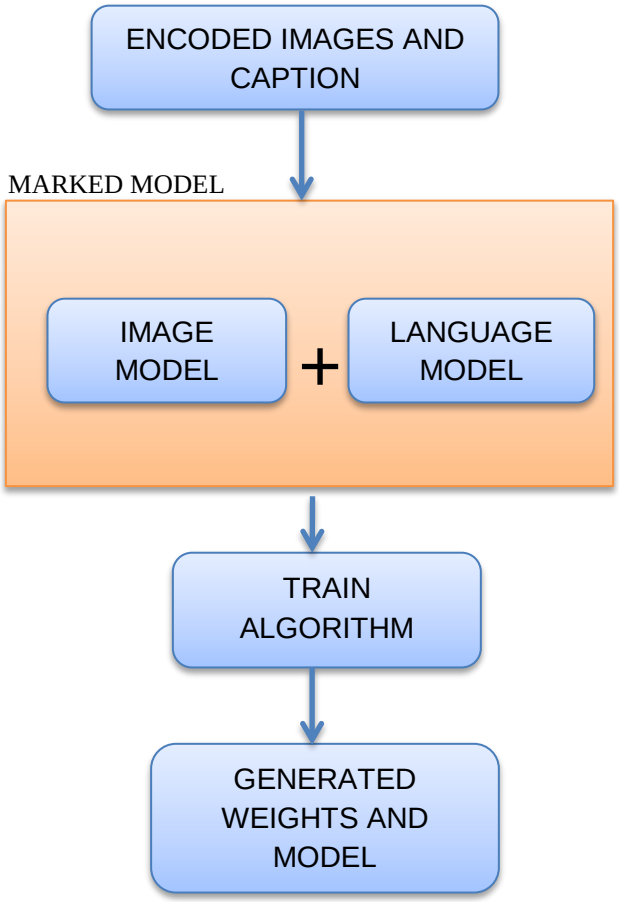

Fig. 9.2: Training Module

9.3 EVALUATION MODULE:

- **Input:** Image
- **Description:** Takes the image input from the user and creates VGG16 model on that given input image and next LSTM model will generates the language for the description and predicts the description for that particular image. Here while predicting it come up with some predictions final it return which as the higher accuracy rate.
- **Output:** Description for the given image.

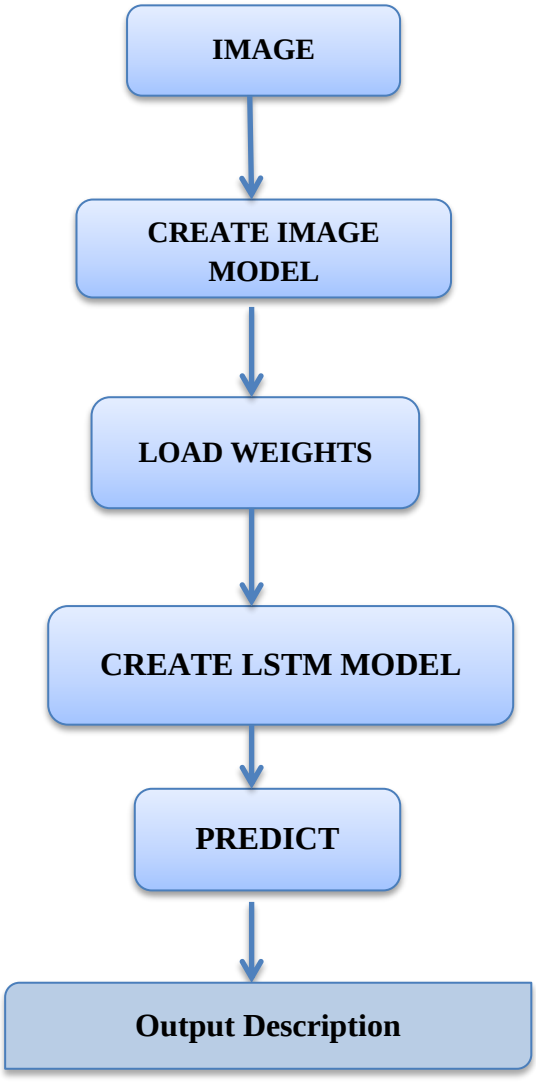

Fig. 9.3: Evaluation Module

CHAPTER 10

IMPLEMENTATION/PSUEDO CODE

The code for the model was built on keras. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. We use tensorflow as the backend to build the code.

- **Code for Image Encoding:**

```
from keras.layers import Flatten, Input, Dense, Conv2D, Maxpooling2D,
GlobalMaxPooling2D
x = Conv2D(64, (3, 3), activation='relu', padding='same',
name='block1_conv1')(img_input)
x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)
# Block 2
x = Conv2D(128, (3, 3), activation='relu', padding='same',
name='block2_conv1')(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same',
name='block2_conv2')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)
# Block 3
x = Conv2D(256, (3, 3), activation='relu', padding='same',
name='block3_conv1')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same',
name='block3_conv2')(x)
x = Conv2D(256, (3, 3), activation='relu', padding='same',
name='block3_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)
# Block 4
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv1')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block4_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv1')(x)  # Block 5
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv2')(x)
x = Conv2D(512, (3, 3), activation='relu', padding='same',
name='block5_conv3')(x)
x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)
if include_top:
# Classification block
x = Flatten(name='flatten')(x)
x = Dense(4096, activation='relu', name='fc1')(x)
x = Dense(4096, activation='relu', name='fc2')(x)
        x = Dense(classes, activation='softmax', name='predictions')(x)
```

Following are the briefly described important functions in the above mentioned code.

1. Conv2D

This operation creates a layer which is a convolution kernel that is convolved with the layer input to produce a tensor of outputs. The important arguments to this function are-

Filters: The first parameter to the function is filters i.e. an Integer that determines the dimensionality of the output space (the number output of filters in the convolution).

Strides: The second argument to the function is an integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height.

Activation: The third argument to the function is which Activation function to use which in this case is defined as ReLU. If you don't specify anything, linear activation is applied.

2. MaxPooling2D

This operation performs maximum spatial pooling. The important arguments are-

Pool_size: First argument to the function is an integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). If only one integer is given, then the same window length will be applied for both the dimensions.

Strides: Second argument to the function is an Integer, row of 2 integers, or None. The argument governs the same operation as discussed in Conv2D layer. If None, it will default to pool_size.

3. Flatten

The flatten() function in keras is used to flatten the input. It does not affect the batch size. For instance, the output of a particular layer is of dimension (32, 32, 64) then after applying the flatten function the aspects of the feature vector would become 65536 i.e. 32*32*64. The operation of the flatten function is similar to that of numpy.reshape() function.

4. Dense

The dense layer is fully connected layer in the neural network model, so all the neurons in a layer are attached to those in a next layer. One important parameter to the dense function is –

Activation: It specifies the activation function to be used. The softmax activation function in the last line maps the encoded vector in the previous step to all possible output classes.

- **Code for the language generating RNN:**

```
from keras.models import Sequential
from keras.layers import LSTM, Embedding, TimeDistributed, Dense, RepeatVector,
Merge, Activation

image_model = Sequential()
image_model.add(Dense(EMBEDDING_DIM, input_dim = 4096, activation = 'relu'))
image_model.add(RepeatVector(self.max_length))

lang_model = Sequential()
lang_model.add(Embedding(self.vocab_size, 265, input_length = self.max_length))
lang_model.add(LSTM(256, return_sequences = True))
lang.model.add(TimeDistributed(Dense(EMBEDDING_DIM)))

model = Sequential()
model.add(Merge([image_model, lang_model], mode = 'concat'))
model.add(LSTM(1000, return_sequences = False))
model.add(Dense(self.vocab_size))
model.add(Activation('softmax'))
```

Some important functions in the above-mentioned code are listed below

1. Sequential
The Sequential model is usually a similar type stack of layers one after the other.

2. Embedding
This layer changes positive integers (indexes) into dense vectors of fixed value size. This layer can only be applied as the initial layer in a neural network model. The first parameter to the function is input dimensions which is like to the size of the vocabulary and the parameter defined as 256 refers to the output dimensions of the layer.

3. LSTM
This function introduces the Long-Short Term Memory layer. The first parameter specified as 1000 in this case is the output dimensions of the layer. One can also specify the activation functions to be used in the layer.

4. Merge
The merge functionality is used to merge two models together. For instance, in the above-mentioned case the mode 'concat' specifies that lang_model is concatenated after the image_model.

CHAPTER 11

TESTING

Test Case ID	TS_1
Test Name	File type identification
Description	Given input file should be identified and should accept only images(.jpg or .png)
Prerequisites	Input file
Test Strategy	Black Box

Table. 11.1: Test case 1

Test Case ID	TS_2
Test Name	Image Encoding
Description	Given input images must encode
Prerequisites	Flickr8K image dataset
Test Strategy	Black Box

Table. 11.2: Test case 2

Test Case ID	TS_3
Test Name	Load weights
Description	Should load the initial weights of ImageNet
Prerequisites	Imagenet should be initialized
Test Strategy	Black Box

Table. 11.3: Test case 3

Test Case ID	TS_4
Test Name	Model create and weights update
Description	Should create neural network and create the model.h5 file, update the weights and save weights.h5 file
Prerequisites	VGG16 model architecture and LSTM
Test Strategy	Black Box

Table. 11.4: Test case 4

Test Case ID	TS_5
Test Name	UI (File Chooser)
Description	Checking the file chooser selects the path of the selected image
Prerequisites	Image must be in directory or in the system
Test Strategy	Black Box

Table. 11.5: Test case 5

Test Case ID	TS_6
Test Name	UI Update description
Description	Checking the description will update after returning from the test.py
Prerequisites	Must be clicked on the Get Caption button after selecting or choosing the image to get description.
Test Strategy	Black Box

Table. 11.6: Test case 6

CHAPTER 12

RESULTS AND DISCUSSIONS

	**Generated Caption** A tennis player hitting the ball. **Human provided caption** A young boy rides a purple bike
	**Generated Caption** A child in a helmet riding a bike **Human provided caption** A young boy rides a purple bike

Fig. 12.1: Output 1

	**Generated Caption** A group of people are walking on a busy street. **Human provided caption** Group of boys and girls walking on the street.
	**Generated Caption** A brown dog is running in the water. **Human provided caption** A dog is carrying wood stick in its mouth and walking in the beach.

Fig. 12.2: Output 2

CHAPTER 13

SNAPSHOTS

13.1 Encoding Images:

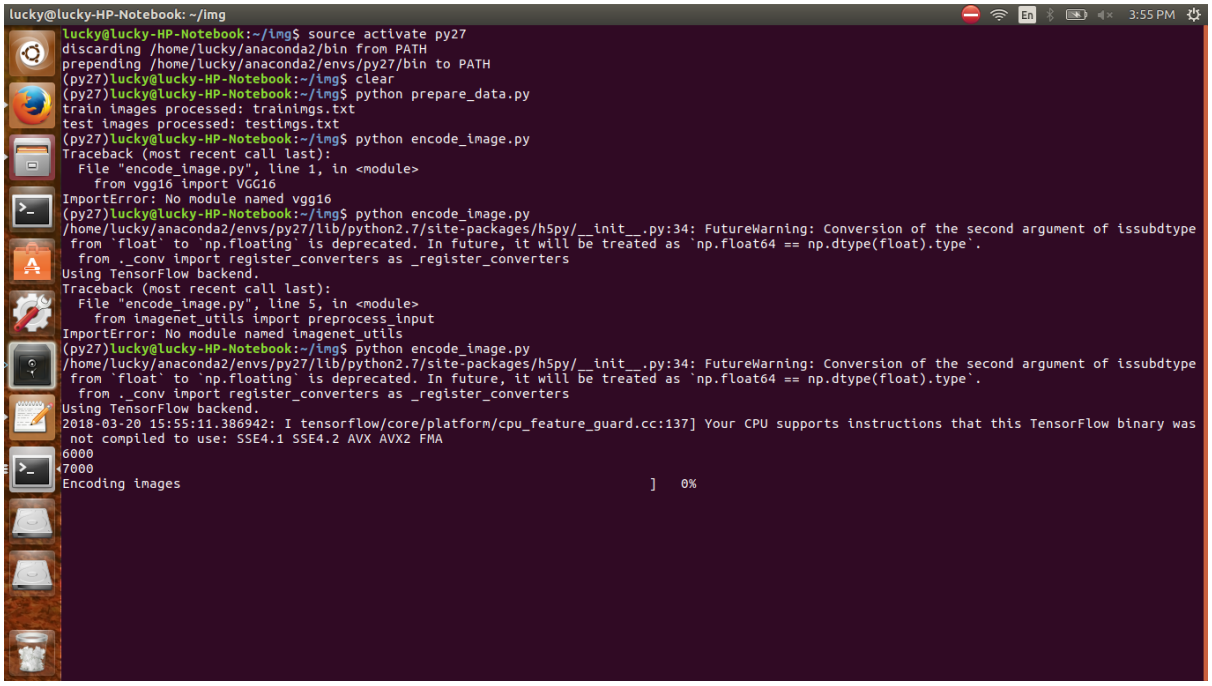

Fig. 13.1.1: Encoding 0%

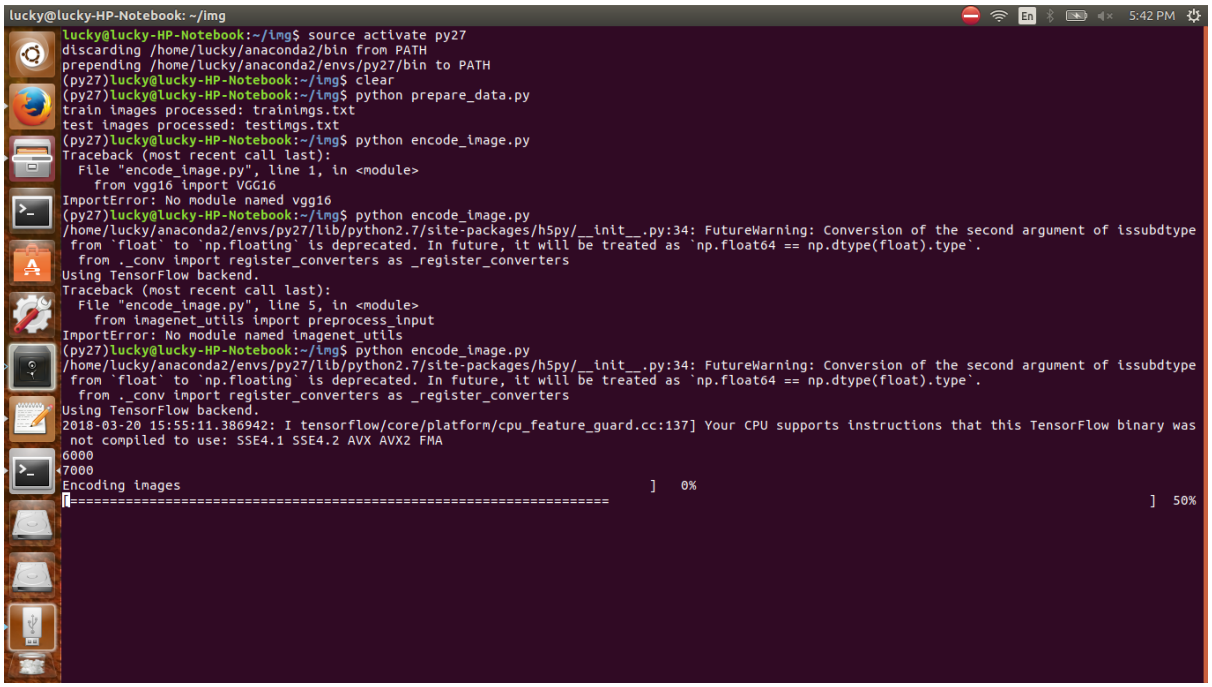

Fig. 13.1.2: Encoding 50%

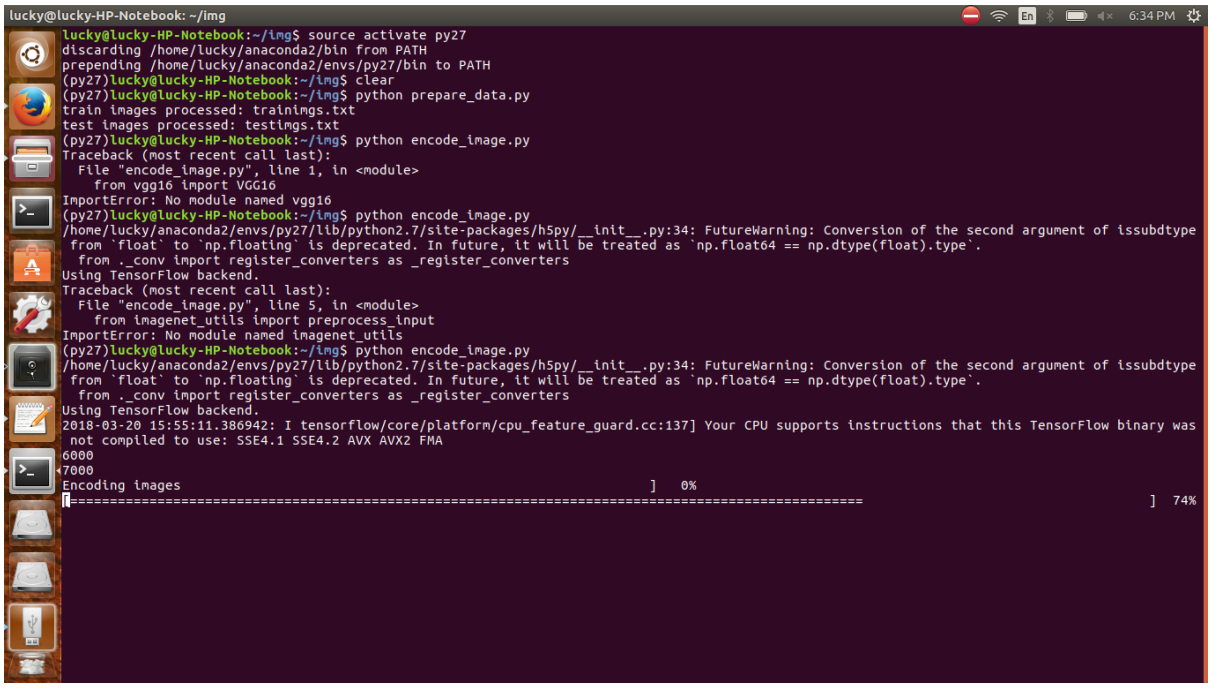

Fig. 13.1.3: Encoding 74%

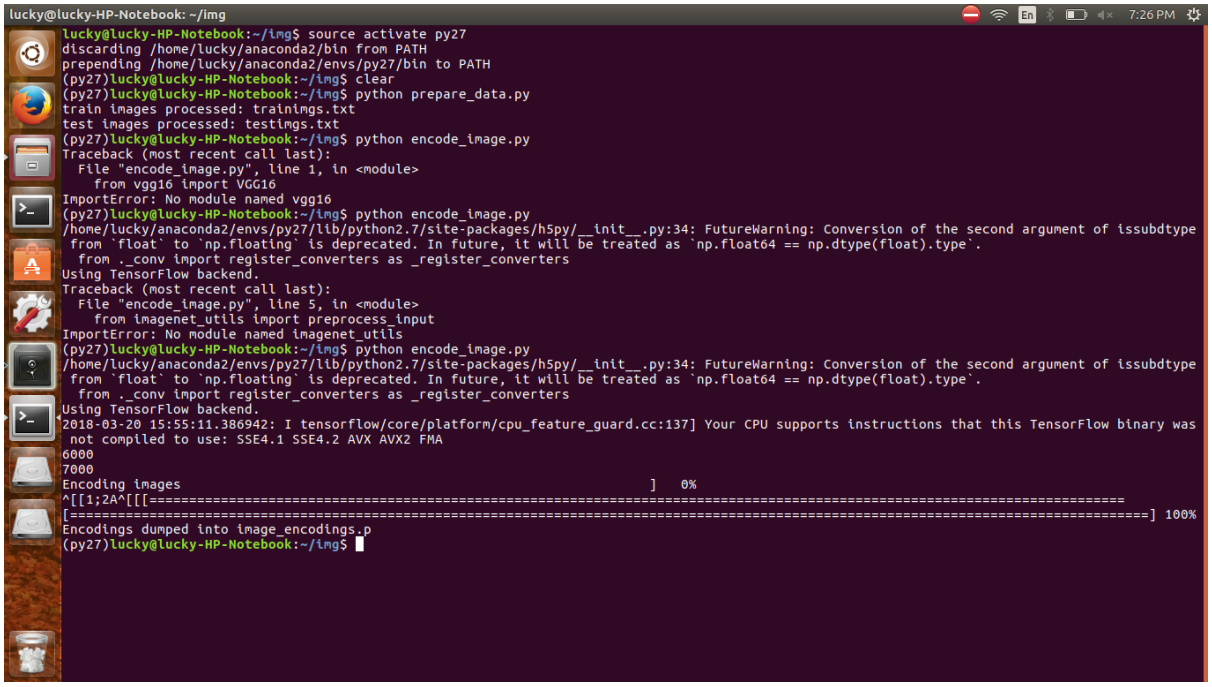

Fig. 13.1.4: Encoding 100%

13.2 Training Model:

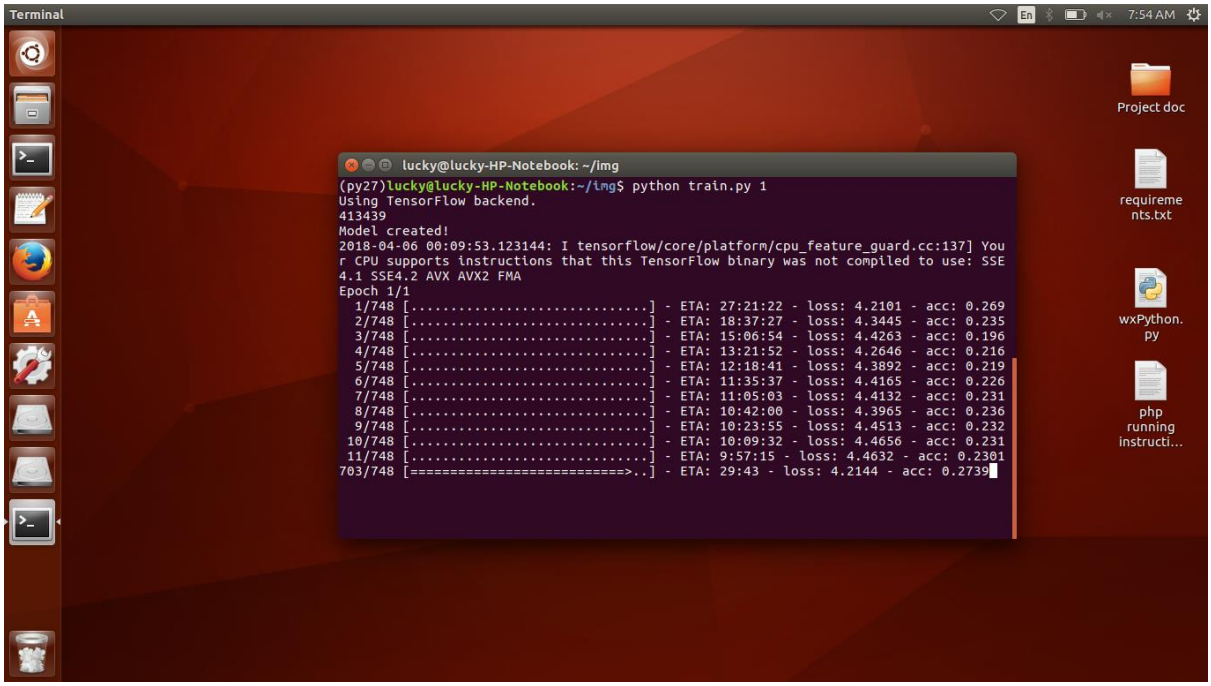

Fig. 13.2.1: Training accuracy 0.2739

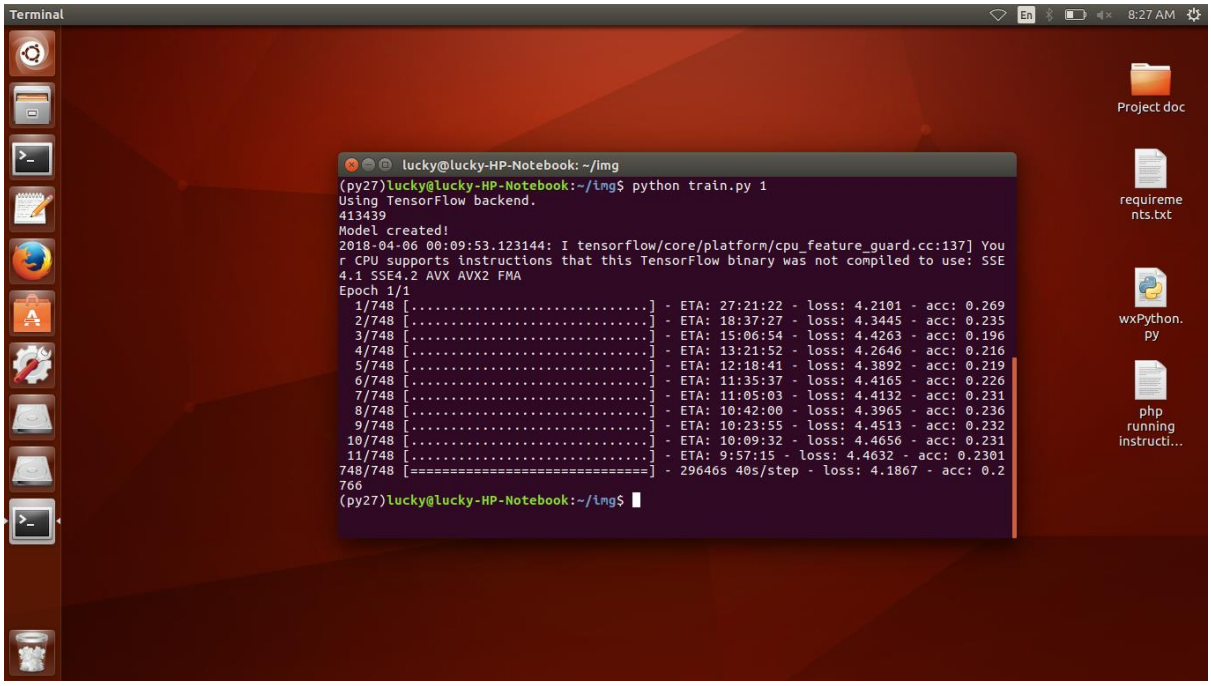

Fig. 13.2.2: Training accuracy 0.2766

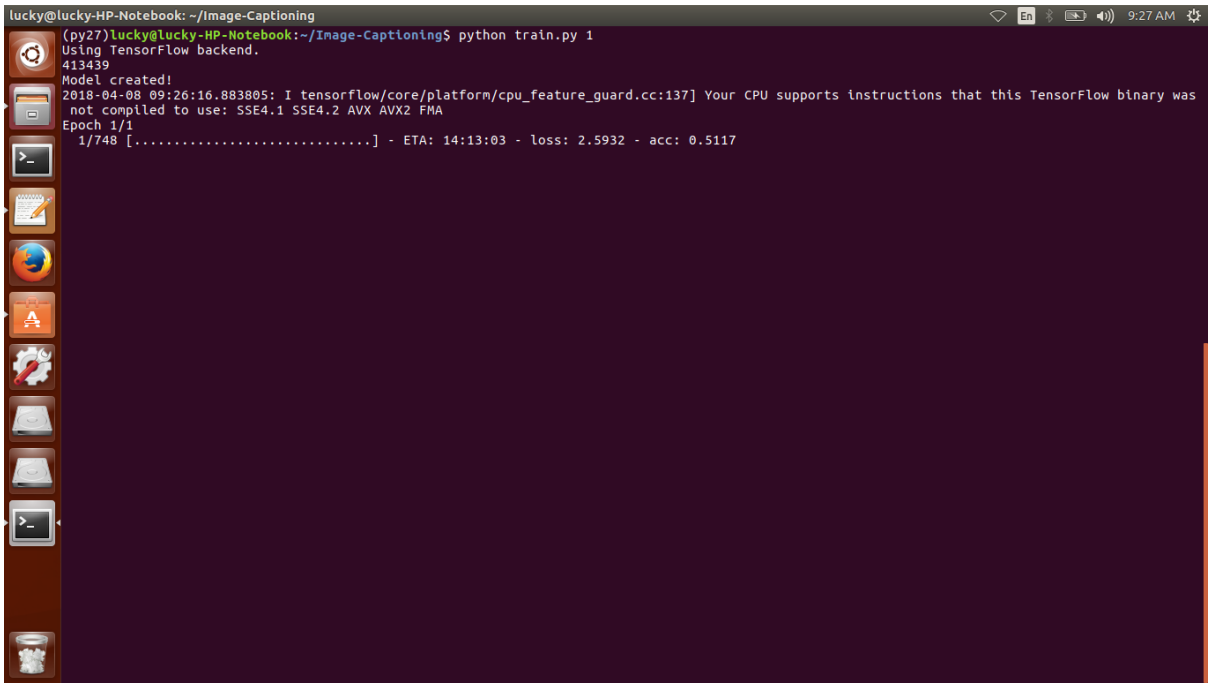

Fig. 13.2.3: Training accuracy 0.5117

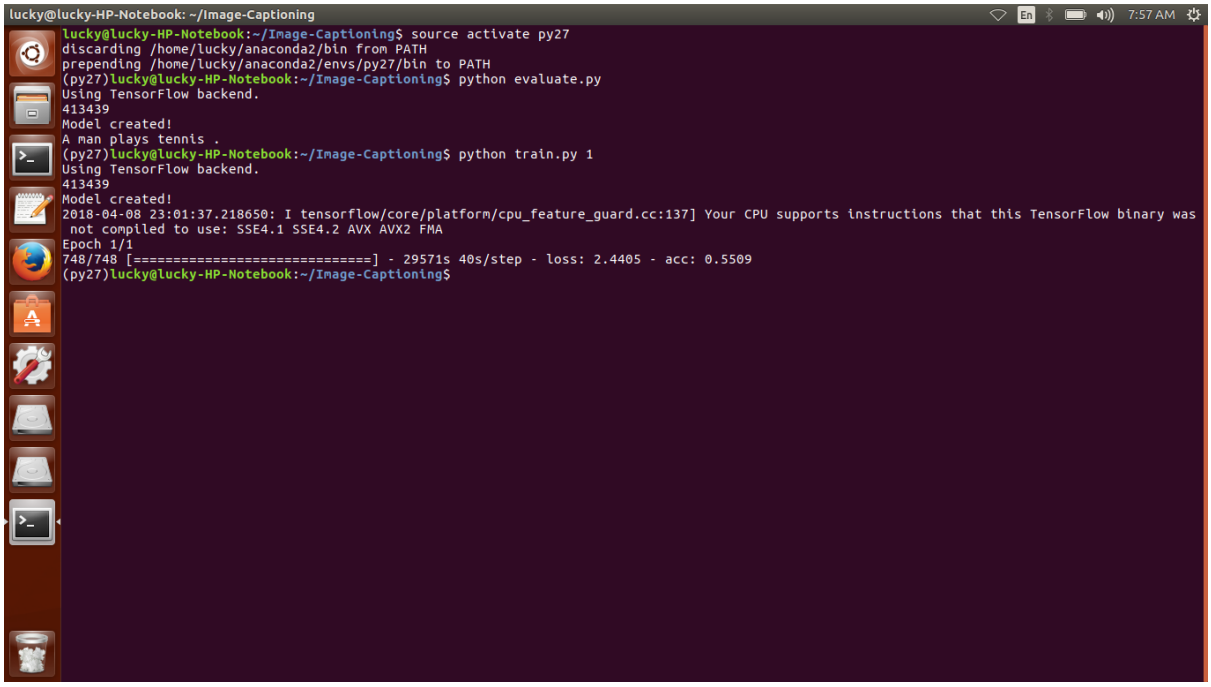

Fig. 13.2.4: Training accuracy 0.5509

13.3 Evaluation:

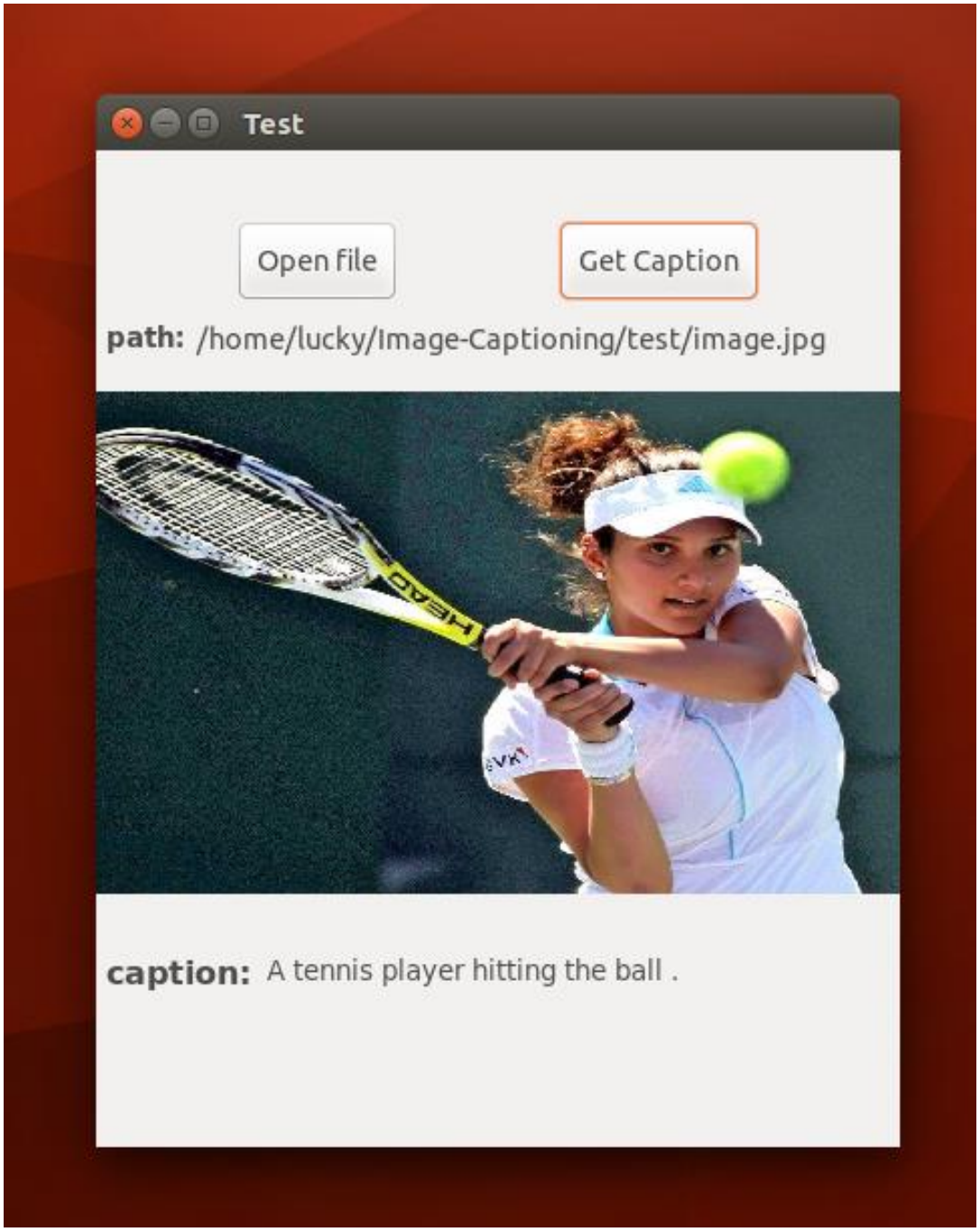

Fig. 13.3.1: Evaluation1

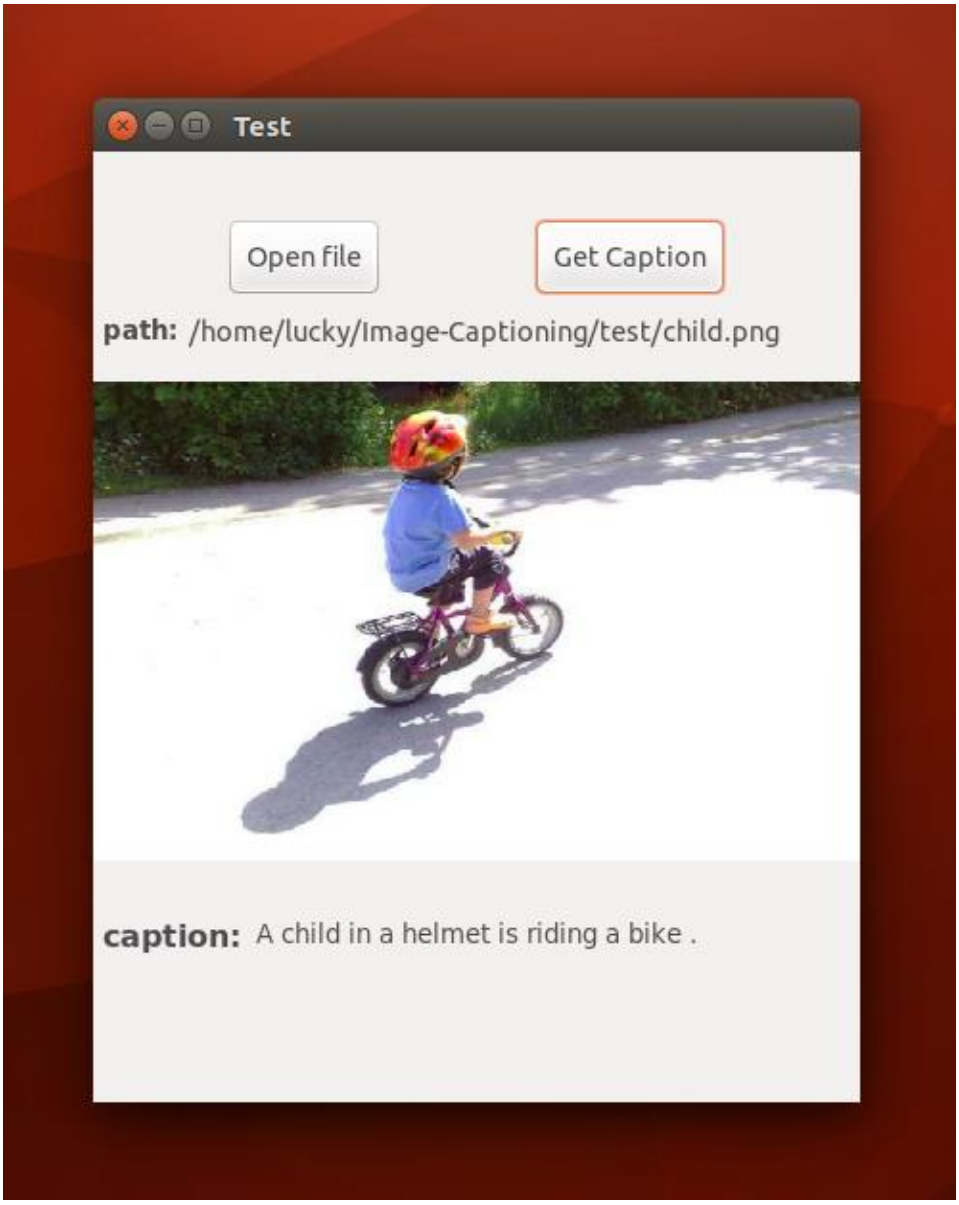

Fig. 13.3.2: Evaluation 2

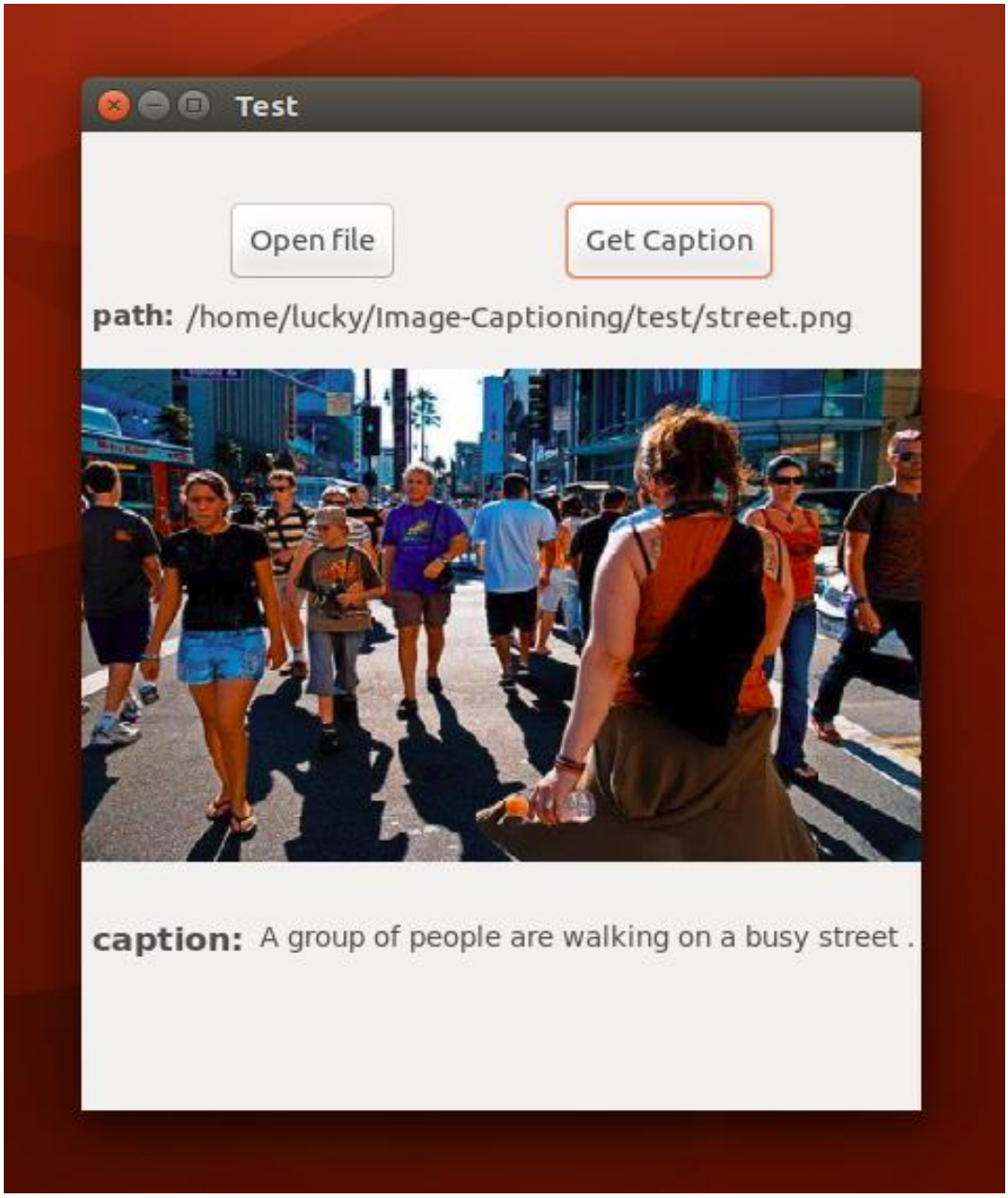

Fig. 13.3.3: Evaluation 3

CHAPTER 14

CONCLUSION

Our end-to-end system neural network system is capable of viewing an image and generating a reasonable description in English depending on the words in its dictionary generated on the basis of tokens in the captions of train images. The model has a convolutional neural network encoder and a LSTM decoder that helps in generation of sentences.

Experimenting the model with Flickr8K dataset show decent results. We evaluate the accuracy of the model on the basis of BLEU score. The accuracy can be increased if the same model is worked upon a bigger dataset.

CHAPTER 15

FURTHER ENHANCEMENT

The task of image captioning can be put to great use for the visually impaired. The model proposed can be integrated with an android or iOS application to work as a real-time scene descriptor. The accuracy of the model can be improved to achieve state of the art results by hyper tuning the parameters.

The model's accuracy can be boosted by deploying it on a larger dataset so that the words in the vocabulary of the model increase significantly. The use of relatively newer architecture, like ResNet and GoogleNet can also increase the accuracy in the classification task thus reducing the error rate in the language generation.

Apart from that the use of bidirectional LSTM network and Gated Recurrent Unit may help in improving the accuracy of the model.

CHAPTER 16

BIBLIOGRAPHY

[1] Jeff Donahue, Lisa Anne Hendricks and Marcus Rohrbach “Long-term Recurrent Convolutional Networks for Visual Recognition and Description”.

https://arxiv.org/abs/1411.5654

[2] X. Chen and C. L. Zitnick “Learning a Recurrent Visual Representation for Image Caption Generation“.
https://arxiv.org/abs/1411.5654

[3] Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015.

https://cs.stanford.edu/people/karpathy/cvpr2015.pdf

[4] Convolution Neural Network-

https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

[5] Long Short Term Memory (LSTM) Model-

https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

[6] VGG16 Model-

https://keras.io/applications/#vgg16

[7] Keras Models-

https://keras.io/models/about-keras-models/

CHAPTER 17

USER MANUAL

Image description is describing an image fed to the model. The task of object detection has been studied for a long time but recently the task of image description is coming into light.

- **Dataset:** The dataset used is Flickr8K. You can request the data [here] (). An email for the links of the data to be downloaded will be mailed to your email id. Extract the images in Flickr8K_Data directory and text data in Flickr8K_Text.
- **Requirements:** (To be installed)
    - Tensorflow
    - Keras
    - Numpy
    - H5py
    - Pandas
    - Pillow
    - Pyttsx
    - WxPython
    - Progress bar
- **Steps to execute:**
    1. After extracting the data, execute the prepare_data.py file by locating the file directory and execute "python prepare_data.py". This file adds "start" and "end" token to the training and testing text data. On execution the file creates new txt files in Flickr8K_Text folder.
    2. Execute the encode_image.py file by typing "python encode_image.py" in the terminal window of the file directory. This creates image_encodings.p which generates image encodings by feeding the image to VGG16 model. In case the weights are not directly available in your temp directory, the weights will be downloaded first.
    3. Create Output directory in project directory, its holds the weights of the trained model. Execute the train.py file in terminal window as "python train.py (int)". Replace "(int)" by any integer value ex: "python train.py 200". The variable will denote the number of epochs for which the model will be trained. The models will be saved in the Output folder in this directory. The weights

4. and model for evaluation now you can download the weights here https://drive.google.com/drive/folders/1aukgi_3xtuRkcQGoyAaya5pP4aoDzl7r . After downloading **model.h5** and **weights.h5** place them in Output directory. These weights trained about 70 epochs. If you downloaded weights and wants to see how the output comes then go to step 4.
5. After training or after downloading the weights from above given link you can evaluate the neural network by using WxPython GUI or in terminal.
    - **WxPython GUI**: Execute "python evaluate.py" for generating a description of an image. This python file will opens the GUI which contains two buttons one is **Open file** for choosing the image to get description and another one is **Get Caption** button, this button calls the text(image) method which is existed in test.py file and prints the description of the given image when it gets return back from text(image) method.

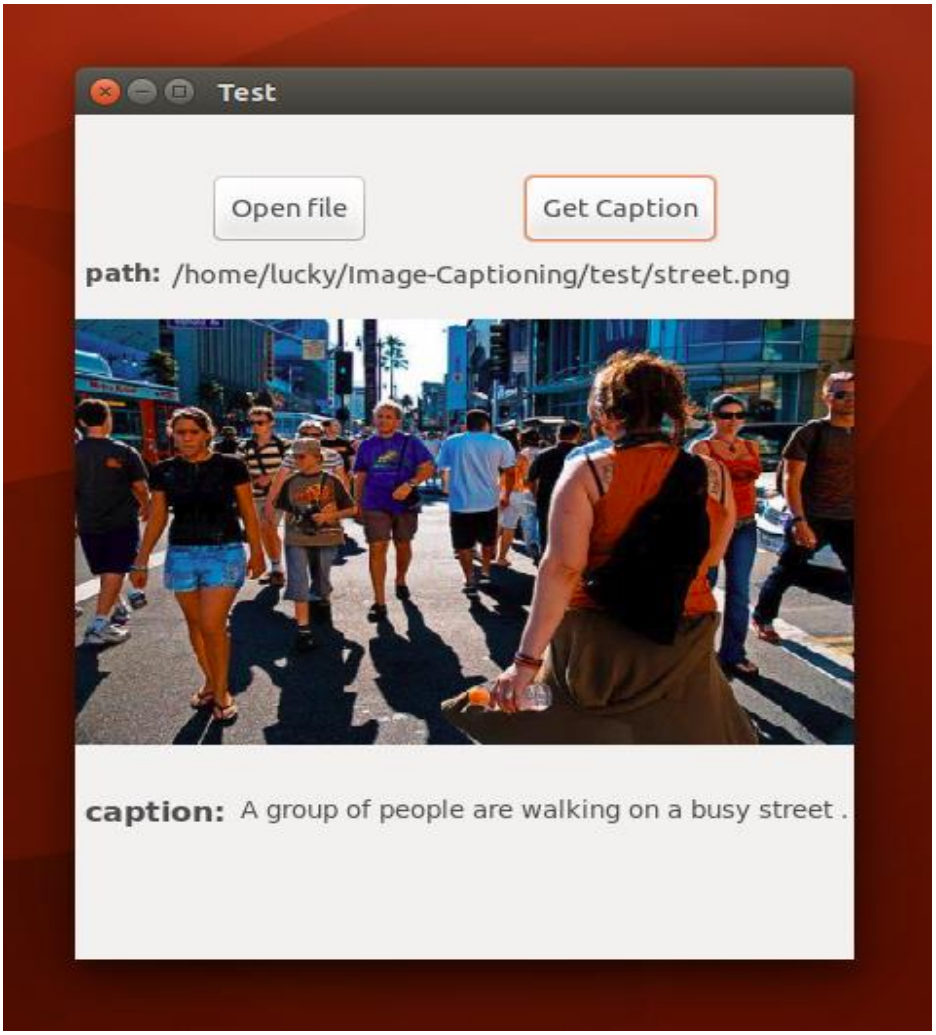

Fig. 17.1: Output 1

- **Terminal:** Execute "python test.py image" for generating a caption of an image. Pass the extension of the image along with the name of the image file for example, "python test.py beach.jpg". The image file must be present in the test folder.

NOTE- You can skip the training part by directly downloading the weights and model file and placing them in the Output folder since the training part will take a lot of time if working on a non-GPU system. A GTX 1050 Ti with 4 gigs of RAM takes around 10-15 minutes for one epoch. For normal CPU systems it takes around 8 hours for one epoch.