IMAGE CAPTIONING USING MACHINE LEARNING

Submitted in partial fulfilment of the requirements
of the degree of
Bachelor of Engineering

T. E. Computer Engineering

By

Dharmi Hemani	**Roll No.06**	**PID.172031**
Deveshwari Pujari	**Roll No.17**	**PID.172030**
Shivani Raul	**Roll No.20**	**PID.172110**

Guide:

Mrs. Dakshata Panchal
Professor

Department of Computer Engineering
St. Francis Institute of Technology
(Engineering College)

University of Mumbai
2019-2020

CERTIFICATE

This is to certify that the project entitled '**Image Captioning Using Machine Learning'** is a bonafide work of **Dharmi Hemani (Roll No.06), Deveshwari Pujari (Roll No.17) and Shivani Raul (Roll No. 20)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of T.E. in Computer Engineering

Mrs. Dakshata Panchal
Guide

Dr. Kavita Sonawane
Head of Department

Dr. Sincy George
Principal

Project Report Approval for T.E.

This project report entitled ***Image Captioning*** by ***Ms.Dharmi Hemani, Ms.Deveshwari Pujari, Ms.Shivani Raul*** is approved for the degree of ***T.E. in Computer Engineering.***

Examiners

1.----------

2.----------

Date:

Place:

Declaration

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

Dharmi Hemani Roll No.06
Deveshwari Pujari Roll No.17
Shivani Raul Roll No.20

Date:

Abstract

Caption generation is a challenging artificial intelligence problem where a textual description must be generated for a given photograph. It requires both methods from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. The Image Captioning Model has many potential applications in real life. A noteworthy one would be to save the captions of an image so that it can be retrieved easily at a later stage just on the basis of this description. The task is straightforward the generated output is expected to describe in a single sentence what is shown in the image the objects present, their properties, the actions being performed and the interaction between the objects, etc. But to replicate this behaviour in a machine learning system is a huge task, as with any other image processing problem and hence the use of complex and advanced techniques such as Machine Learning and Deep Learning are used to solve the task. Project is implemented by using the Tensorflow library which allows end-to-end training of both CNN and RNN parts. Our objective is to replace the encoder part of the RNN with a Convolutional Neural Network (CNN). Thus, transforming the images into relevant input data to feed into the decoder of the RNN. The image will be converted into a multi-feature dataset, characterizing its distinctive features. This results to a method which creates generalized description of new images.

Contents

List of Figures

List of Tables

List of Abbreviations

Table 2: Full form for most commonly used mnemonics

Sr no.	Mnemonics	Definition
1.	NLP	Natural Language Processing
2.	CNN	Convolutional Neural Network
3.	RNN	Recurrent Neural Network
4.	LSTM	Long short-term memory
5.	VGG16	Visual Geometry Group
6.	NLTK	Natural Language Tool Kit
7.	DFD	Data-flow diagram

Chapter 1
Introduction

The project “Image Captioning” is about defining and training a CNN and RNN Networks in order to automatically generate captions for an Image. Hence we created a Machine learning model to generate captions for images. This project uses VGG16, a CNN model for feature extraction. These features are converted to image captions by RNN. Flickr 8K dataset is used for training of this model.

This neural system for image captioning is roughly based on the IEEE paper "An Empirical Study of Language CNN for Image Captioning”

The input is an image and the output is a sentence describing the content of the image. This project is implemented using the Tensorflow library and allows end-to-end training of both CNN and RNN parts.

Some detailed use cases would be like a visually impaired person taking a picture from his phone and then the caption generator will turn the caption to speech for him to understand.

Advertising industry tries to generate captions automatically without the need to make them separately during production and sales.

1.1 Description

1.1.1 Abstract:

Automatic Image captioning is one of the most recent problems that caught the interest of the computer vision community and Natural Language Processing communities alike. This is a major part of scene understanding in computer vision. Not one only do we have to recognize objects in the image, but we have to find relationships in natural language. This project addresses the problem by using a deep neural network model. The model would make use of Convolution neural networks to read image data and Recurrent neural networks for learning sentences/captions for image. Language models based on recurrent neural networks have dominated recent image caption generation tasks. Here a language CNN model which is suitable for statistical language modeling tasks and shows competitive performance in image captioning. Our language CNN is fed with all the previous words and can model the long-range dependencies in history words, which are critical for image captioning. The effectiveness of our approach is validated on two datasets Flickr30K and Flickr8K used to train the model.

1.1.2 Dataset:

There are popular open datasets that can be used to train deep network models. These are often referred to as a benchmark collection of image caption datasets to train and evaluate a model's accuracy. These datasets have complex day to day scenes with various objects.

1. Flickr8K
2. Flickr30K
3. MSCOCO

Preproccessing: We start preprocessing Flickr30k by converting texts to lowercase and any non-alphanumeric are discarded. All words that don't repeat more than some threshold number, would be ignored while building a vocabulary. Building vocabulary is achieved using NLTK toolkit for lemmatization, stop words . It has been observed that vocabulary size is about three thousand words. Dropout can be used to avoid any overfitting. All images are resized to one common size (224*224) before training.

1.2 Problem Formulation

1.2.1 Model:

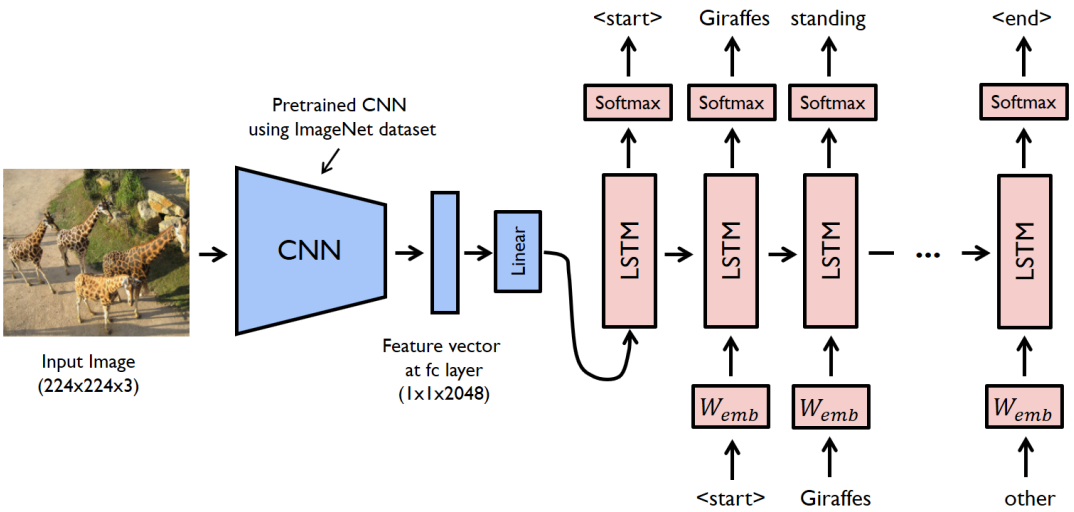


Fig 1.1 A pre-trained CNN model is used as an encoder, and the decoder is an LSTM network.

To solve the problem of image captioning, we need two models.

1. Image model
2. Language model

One of the reasons why the problem is difficult, is because we need two models and seamless integration between them. The network can be viewed as a combination of encoder and decoder. Encoder would be a convolutional neural network i.e CNN. Image is processed by CNN layer and features are extracted. End of the CNN layer is connected to a Long short-term memory i.e LSTM networks, a special kind of Recurrent Neural Network (RNN). LSTM are capable of learning long term dependencies.

Encoder

The Convolutional Neural Network(CNN) can be thought of as an encoder. The input image is given to CNN to extract the features. The last hidden state of the CNN is connected to the Decoder.

Decoder

The Decoder is a Recurrent Neural Network(RNN) which does language modelling up to the word level. The first time step receives the encoded output from the encoder and also the vector.

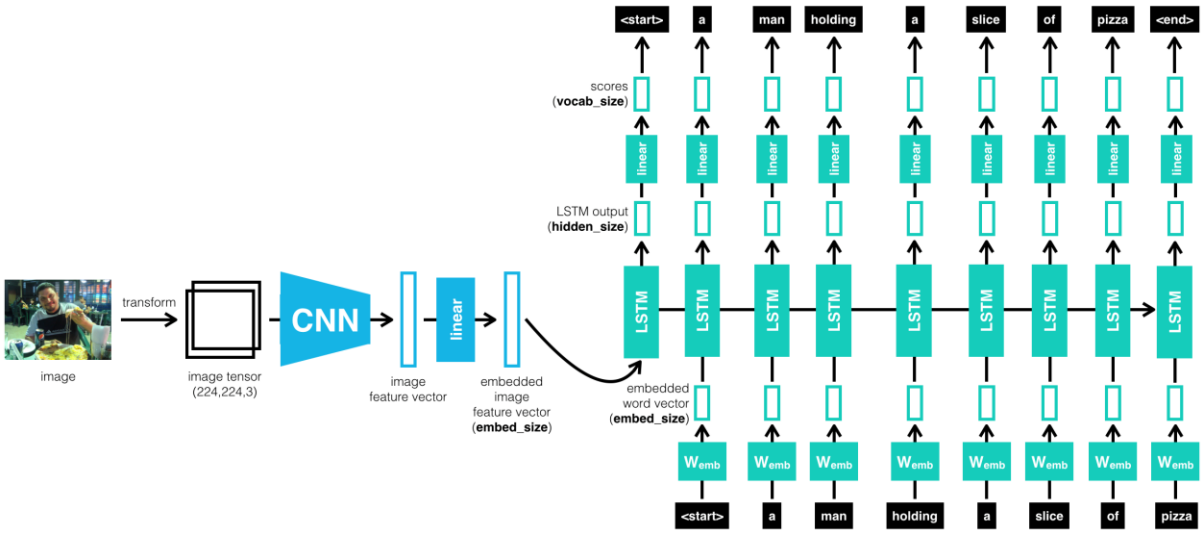


Fig 1.2: Encoder-Decoder.

Our model doesn't make use of spark and hence does not have any sort of data analytic representation. Most of the core work is focused on building efficient deep neural networks. The model is built using Keras, a deep learning library in python. Keras is a high-level library that is above Theano and Tensorflow. The API is very simple and makes use of the Tensorflow backend. Keras is minimalistic in code, and is growing fast in the community. Keras is well documented and can run on GPU.

We designed the above model and trained it. The first subsection of layers is the image model which takes the image as input and feeds it to CNN layer. The CNN gets trained on images and features are extracted. The second subsection of layers is language mode of LSTM

1.Image model

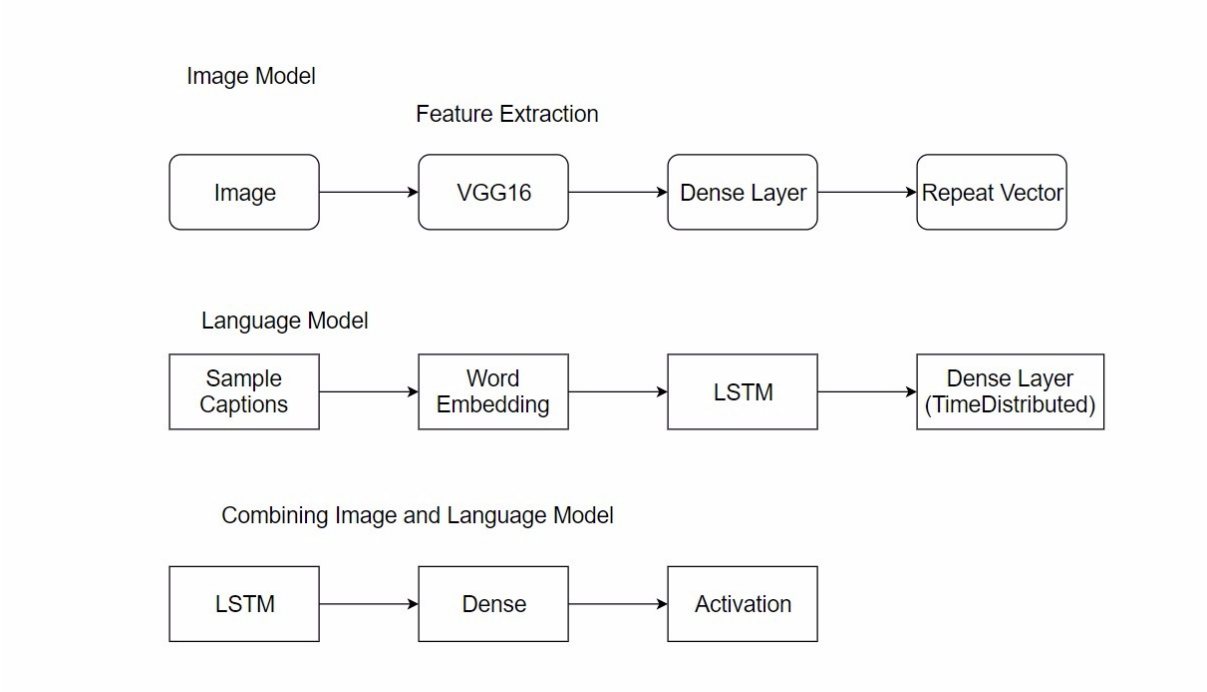


Fig 1.3: Image Model.

We used transfer learning to use one popular model for CNN implementation. Transfer learning is a major topic in machine learning that involves storing knowledge from one model and applying to another problem. The reason we use pretrained network is because, CNN models are difficult to train from scratch and it could be very computationally expensive that it takes several hours on GPU. In the scientific community, it is very common to use pre trained model on a larger dataset and then using the model as a feature extractor.

After rigorous research, we plan to use a pretrained CNN model called “VGG16” or “OxfordNet”. This model has won the 2014 ImageNet challenge - ImageNet Large Scale Visual Recognition Competitionin localization and classification tracks. The model is found to be easily generalizable to other datasets, with very good results[7]. The original VGG model is written on Caffe, however we use Keras implementation [3]. Details about the architecture can be found in the paper – “Very Deep Convolutional Networks for Large-Scale Image Recognition” . The model architecture is portrayed in the below figure. The pooling layers are used as down samplers, and the initial pooling layers detect details like edges, lines etc. As the network comes to end, the pooling layers detect abstract features and shapes. The input would be images of fixed size (224*224), with 3 channels. The last SoftMax layer gives the probability distribution for object recognition. We removed the SoftMax layer, and the updated

model generates a weight vector of dimension 4096.We now use the dense vector to change the output to fixed embedding size which is 128 in our case. Then we use the repeat vector to repeat it as one vector for each word since we will be merging this output with Language model output.

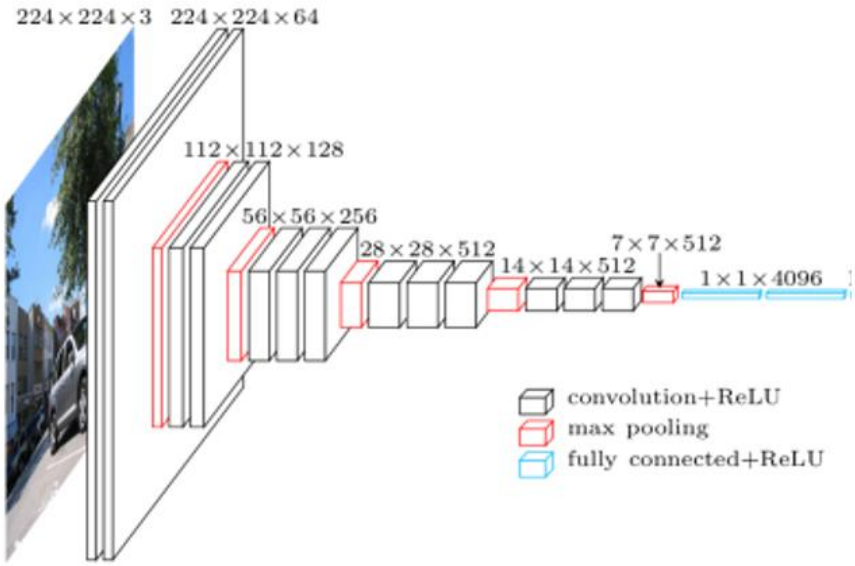


Fig 1.4: The architecture of VGG16 Model.

2. Text model

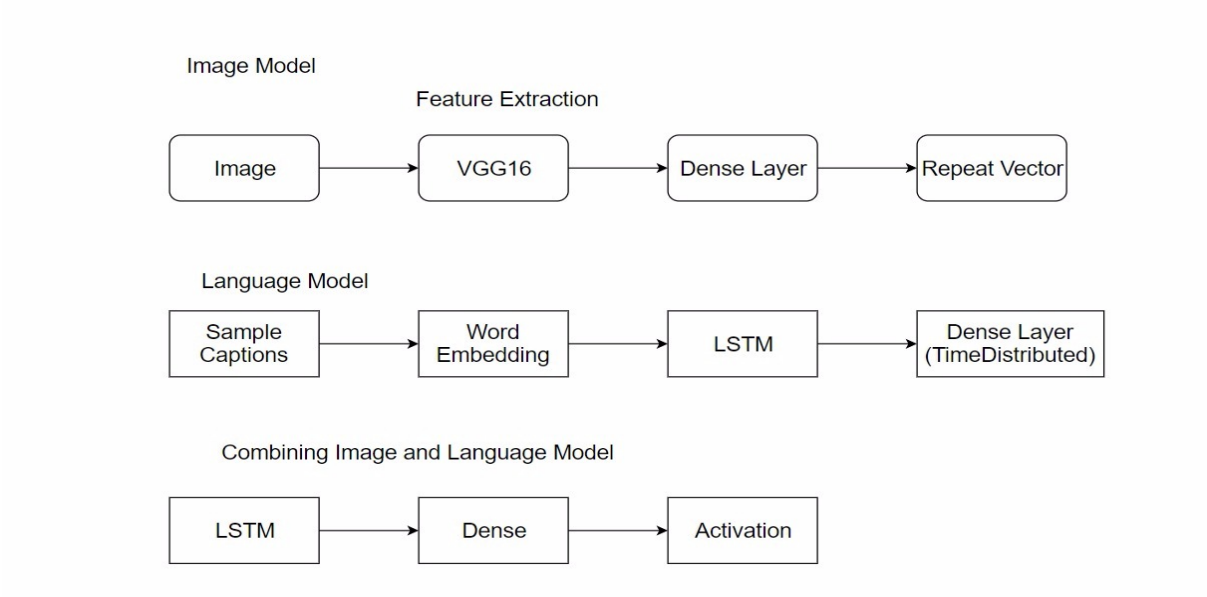
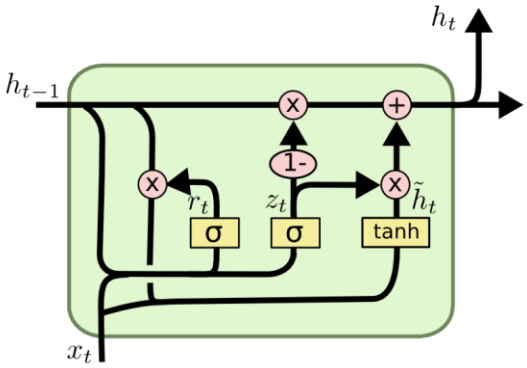


Fig 1.5: Text Model

The output of image model acts as input to language model. To understand the captions under the images, we use a Recurrent Neural Networks(RNN) to solve the problem. Vanilla neural networks work in several scenarios, but the API is very limited. RNN's output is determined not just by the input, but with series of inputs. In practice, we use Long Short-term memory (LSTM), which is a variation of RNN. LSTM works better and has powerful update equation and backpropagation. LSTM is a language model and decoder trained on feature vectors. LSTM had phenomenal influence and success in different problems like language modelling, speech recognition, translation etc. The power of RNN/LSTM has been well documented by Andrej Karpathy in this blog post. Consider the problem of guessing the next word. The prediction depends on last few words, to understand the context and RNN's does that with memory. Before the LSTM were proposed by Hochreiter & Schmidhuber, long term dependencies were a problem. Having a memory of information isn't a problem anymore.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Fig 1.6: Working and Trigonometric calculation of LSTM

LSTM has three inputs – the input from current time step, output from previous time step and memory of the previous unit. LSTM have ability to add/remove information via gates, which consists of sigmoid neural nets and pointwise multiplication operations. LSTMs' are a major reason why image captioning models are successful[5]. LSTM picks part of the image and maps to the appropriate word in the caption. Here, we create an embedding layer to get a vector representation for each word in the caption. Then we input the output vector to LSTM for the model to learn the neighbouring words for each word. We then convert the LSTM output to fixed dimension using a dense layer. In this case we will use Time Distributed because it's a 3D tensor. Now, we combine the outputs from both Language Model and Image model, and input the vector to LSTM. LSTM learns the different captions for that image in the training phase. We convert the LSTM output to the vocabulary size using the dense layer and activate the model using activation method. In the testing phase, LSTM predicts the captions for the

image. LSTM predicts the next word for the given image with the partial caption available at that stage.

1.3 Motivation

The application of Artificial Intelligence and Neural Networks to complicated natural language processing challenges like speech recognition and machine translation is leading to remarkably rapid advancements. Image Captioning is very much important in real world scenarios. Image Captioning can be useful in various scenarios for example:

Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system. This has already been under implementation in self driven cars. Similarly Auto Captioning can help in various ways.

Automatic Captioning can help make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.

CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.

Providing the motivation for information exchange through communication alone is not the complete solution.The objective of the system is simply to optimize information exchange, the system will invent its own optimized caption.Traditionally, methods like template based and retrieval based methods have been used to solve the problem.

However, the major drawback of these methods is the results don’t translate to new images and hence these methods fail in generalization. These methods focus on labeling images with a fixed set of visual categories due to which they fail in describing new images. Therefore the requirement to create a method that can be generalized to create description of new images.

The drawbacks mentioned above are removed with the help of deep learning models such as CNN, RNN and LSTM making the model more convenient and efficient.

1.4 Proposed Solution

To become acquainted with the “image caption generation” topic we first conducted a quick review of articles under it. We obtained an idea of the technologies and models, which are popular under this topic so that our research would be relevant and correct. Moreover, we did not narrow the search query to small details in order to get enough results and an appropriate number of articles from the search—keywords are presented identically to how they were submitted for the search query. Initially we decided to work with a traditional method of clustering although clustering would be a tedious task as it would not give 100% accuracy for the model hence through further study we used the VGG16 model in constitution with Keras. Using **BLEU Score** to evaluate and measure the performance of the model.The neural networks were studied in detail for this purpose.

1.5 Scope of The Project

We combine "Image Labeling" and "Automatic Machine Translation" into an end-to-end hybrid neural network system. The developed model is capable of autonomously viewing an image and generating a reasonable description in natural language with reasonable accuracy and naturalness.We also evaluate the effectiveness of language CNN on the smaller dataset Flickr30K. Through our model we can build up great accuracy for our project by using the VGG16 model which helps out in both preprocessing, training the dataset[6]. This project has a high future scope in terms of machine learning applications as it can be used in various fields for the purpose of training a machine so that the machine works as per human development.

The goal is to design a model which covers all the functions of image description and provides an interface of Digital assistant to the user. A Digital assistant help the user to provide answer to his questions which would be given in speech form as a command.
By using Machine learning techniques and Natural language processing project performes:

Image Captioning: Recognising Different types of objects in an image and creating a meaningful sentence that describes that image to visually impaired persons.

Chapter 2
Review of Literature

2.1 Literature Review:

Generating captions for images is a very intriguing task lying at the intersection of the areas of Computer vision and Natural Language Processing. This task is central to the problem of understanding a scene.

The purpose of this model is to encode the visual information from an image and semantic information from a caption, into a embedding space; this embedding space has the property that vectors that are close to each other are visually or semantically related. For a batch of images and captions, we can use the model to map them all into this embedding space, compute a distance metric, and for each image and for each caption find its nearest neighbours. If you rank the neighbours by which examples are closest, you have ranked how relevant images and captions are to each other.

Traditionally, pre-defined templates have been used to generate captions for images. But this approach is very limited because it cannot be used to generate lexically rich captions.The research in the problem of caption generation has seen a surge since the advancement in training neural networks and the availability of large classification datasets. Most of the related work has been based on training deep recurrent neural networks. The first paper that used neural networks for generating image captions was proposed by Kiros , that used Multi-modal log bilinear model that was biased by the features obtained from input image[1].

Karpathy developed a model that generated text descriptions for images based on labels in the form of a set of sentences and images[2]. They use multi-modal embeddings to align images and text based on a ranking model they proposed. Their model was evaluated on both full frame and region level experiments and it was found that their Multimodal Recurrent Neural Net architecture outperformed retrieval baselines[3][4].

In our project we have used a Convolutional Neural Network coupled with an LSTM based architecture. An image is passed on as an input to the CNN, which yields certain annotation vectors. Based on a human vision inspired notion of attention, a context vector is obtained as a

function of these annotation vectors, which is then passed as an input to the LSTM. Creating a captioning system that accurately generates captions like humans depends on the connection between the importance of objects in image and how they will be related to other objects in image. Image can be described using more than one sentence but to efficiently train the image captioning model we require only a single sentence that can be provided as a caption. This leads to problems of text summarization in natural language processing.

There are mainly two different ways to perform the task of image captioning.

1. Retrieval based method
2. Generative method.

From that most of the work is done based on retrieval based methods. One of the best models of retrieval based methods is the **Im2Txt model**. It was proposed by Vicente Ordonez, Girish Kulkarni and Tamara L Berg.

Their system is divided into mainly two part

1) Image matching

First we will provide our input image to model. Matching image will be retrieved from database containing images and its appropriate caption. Once we find match- ing images we will compare extracted high level objects from original image and matching images.

2) Caption generation.

Images will then reranked based on the content matched. Once it is reranked, the caption of top n ranked images will be returned. The main limitation of these retrieval based method is that it can only produce captions which are already present in database. It can not generate novel captions.

This limitation of retrieval based methods is solved in generative models. Using generative models we can create novel sentences.

Generative models can be of two types -

1. Pipeline based model
2. End to end model.

Pipeline type models uses two separate learning processes, one for language modeling and one for image recognition. They first identify objects in the image and provide the result of it to a language modeling task.

While in **end-to-end models** we combine both language modeling and image recognition models in a single end to end model. Both parts of the model learn at the same time in an end-to-end system. They are typically created by a combination of convolutional and recurrent neural networks.

2.2 Previous work:

In our project we have used a Convolutional Neural Network coupled with an LSTM based architecture. An image is passed on as an input to the CNN, which yields certain annotation vectors. Based on a human vision inspired notion of attention, a context vector is obtained as a function of these annotation vectors, which is then passed as an input to the LSTM.

Chapter 3

System Analysis

3.1 Functional Requirements

In the following project, we used the 16-layer VGGNetwork model to compute CNN features and map the last fully-connected layer's output features to an embedding space via a linear transformation.We used Anaconda Platform to run Python codes.

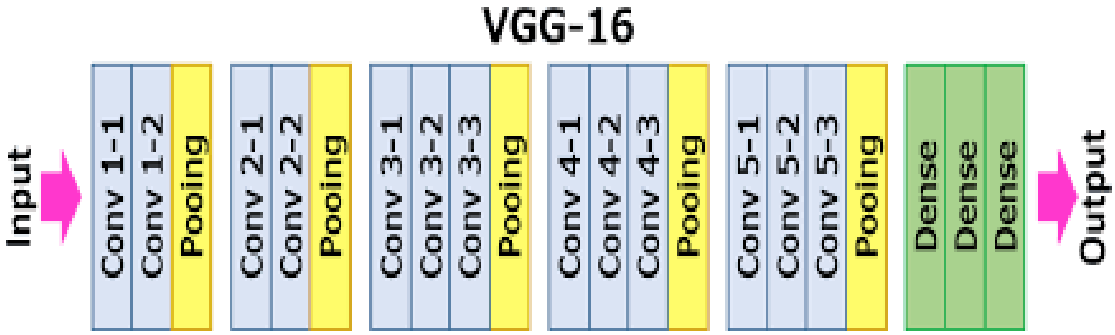


Fig 3.1: 16 Layered VGG Model

3.2 Nonfunctional Requirements

3.2.1 Performance Requirements:

Performance - The model is designed for open sources and can run from a standalone desktop PC. The software will support simultaneous user access only if there are multiple terminals. Only voice information will be handled by the software. Amount of information to be handled can vary from user to user.

Usability – The model has a simple codes and easy to use. It has been designed in such a way that any student can easily use it with minimal problems.

Reliability – The reliability of the model entirely depends upon the availability of the server. As long as the server is available, the model will always work without a problem.Further this simple model can be used in compex codes for largere projects that uses image captioning.
Security: The output can be accessed easily without requirement of any secure or zipped files.
Manageability: Once the image captioning algorithm is devised, no frequent changes will be required. Easily manageable

3.3 Specific Requirements

3.3.1. Hardware Requirements:

1. Windows OS (user)
2. Anaconda

3.3.2 Software Requirements:

Operating System: Windows

Language: Python

Database:Flickr 30K database.

Libraries:

Keras and Tensorflow for Machine learning models.

Dependencies:

1. Keras 2.0.7
2. Theano 0.9.0
3. Numpy
4. Pandas 0.20.3
5. Matplotlib
6. Pickle

3.4 Use-Case Diagrams and description

3.4.1 Use Class:

The goal is to design a model such that any user without any restriction can use it for simplifying their simple works. The model specifically targets Visually Impaired people. Brief knowledge regarding model running is required as the model receives input in the form of image . English language is used for interaction.

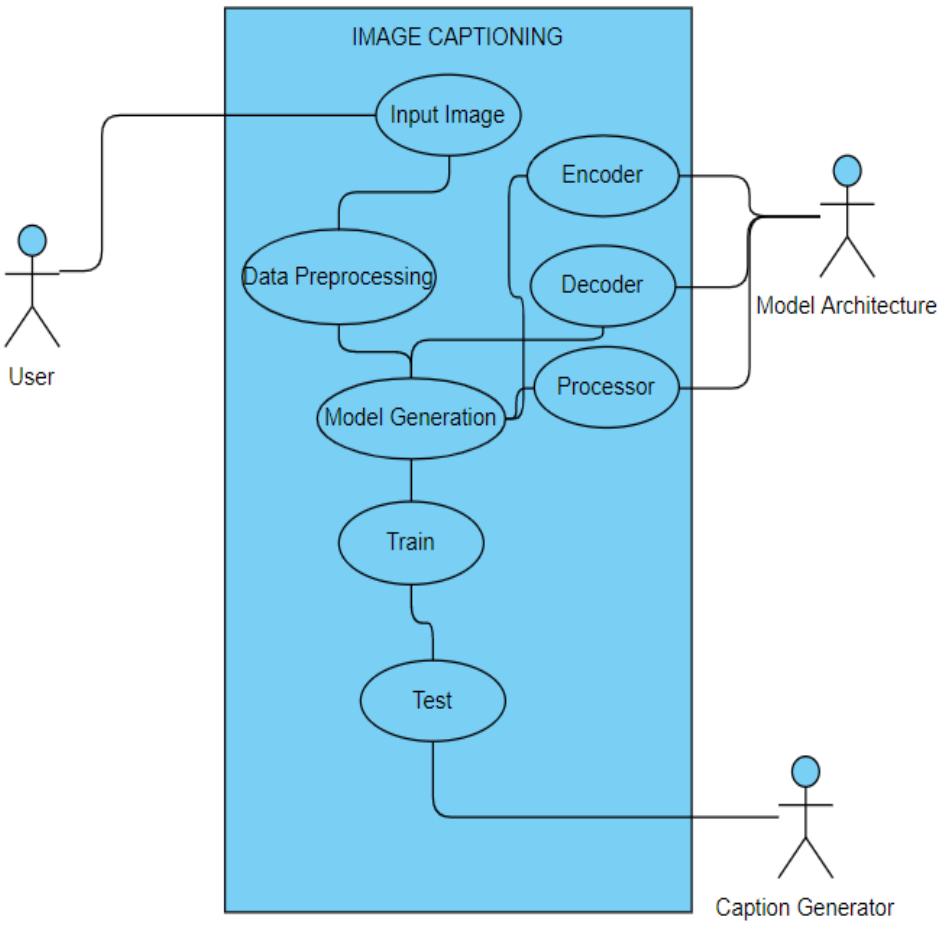


Fig 3.2: Use Class diagram of the image encoder and caption generator working to map the data in a embedding text.

3.4.2 Description:

1. an image encoder—a linear transformation from the 4096 dimensional image feature vector to a 300 dimensional embedding space
2. a caption encoder—a recurrent neural network which takes word vectors as input at each time step, accumulates their collective meaning, and outputs a single semantic embedding by the end of the sentence.
3. A cost function which involved computed a similarity metric, which happens to be cosine similarity, between image and caption embeddings

Product Features

1. Captioning the given image.
2. Output generation in the form of text.

Image Captioning

User receives the description of Image uploaded by them. The user can get description in the form of text .

1. Open the model.
2. Run the Program.
3. Click or Upload the Image.
4. Image caption is generated.

Chapter 4

Analysis Modeling

4.1 Data Modeling

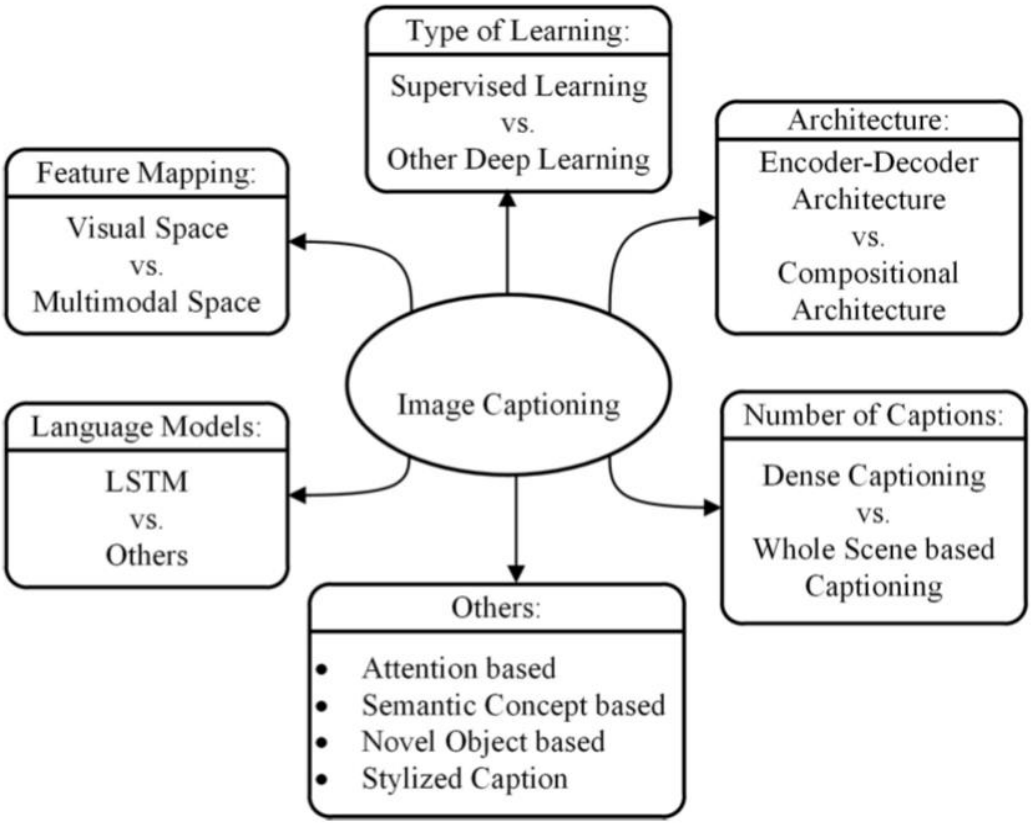


Fig. 4.1. An overall taxonomy of Machine Learning-based image captioning.

The figure illustrates the comparisons of different categories of image captioning methods. Novel caption generation-based image caption methods mostly use visual space and deep machine learning based techniques. Captions can also be generated from multimodal space. Deep learning-based image captioning methods can also be categorized on learning techniques: Supervised learning, Reinforcement learning, and Unsupervised learning. We group the reinforcement learning and unsupervised learning into Other Deep Learning. Usually captions are generated for a whole scene in the image. However, captions can also be generated for different regions of an image (Dense captioning).

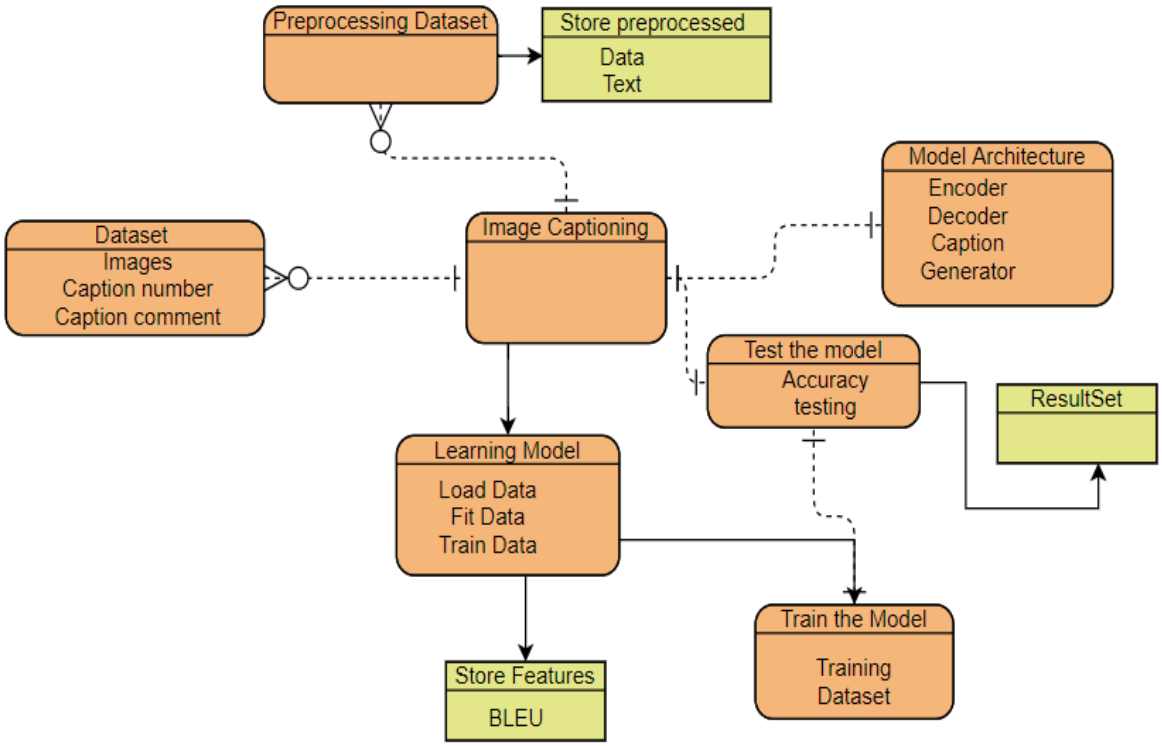


Fig 4.2: ER Diagram

The above diagram represents the Entity Relationship Diagram of Image Captioning System. Through this diagrammatic representation we can get a overview of how the entities are related to each other and their further functionings. Moreover here we also elaborate the Work Flow of our project .

4.2 Activity Diagram

The activity diagram drawn below illustrates the flow of control in the system and refer to the steps involved in the execution of Image Captioning model. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using this activity diagram. This activity diagram focuses on condition of flow and the sequence in which it happens.

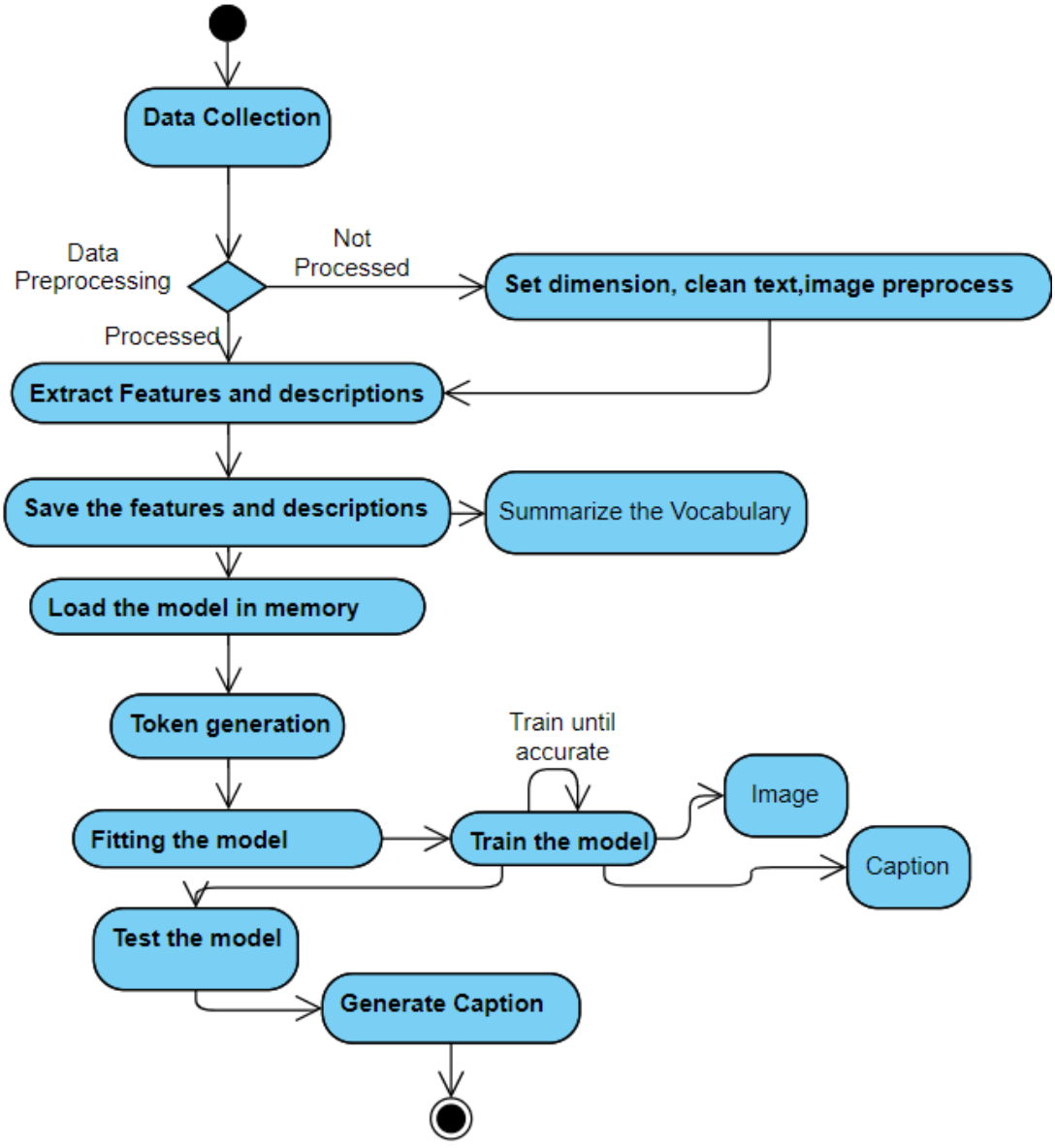


Fig 4.3: Activity Diagram

4.3 Functional Modeling

Data flow diagram

DFD illustrates how data is processed in Image captioning in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

Level 0 DFD

The diagram below represents level 0 DFD for Image Captioning. In this the whole system is represented with one process as System, the rest of the entities are defined which have greater roll in the system along with their essential functionalities.

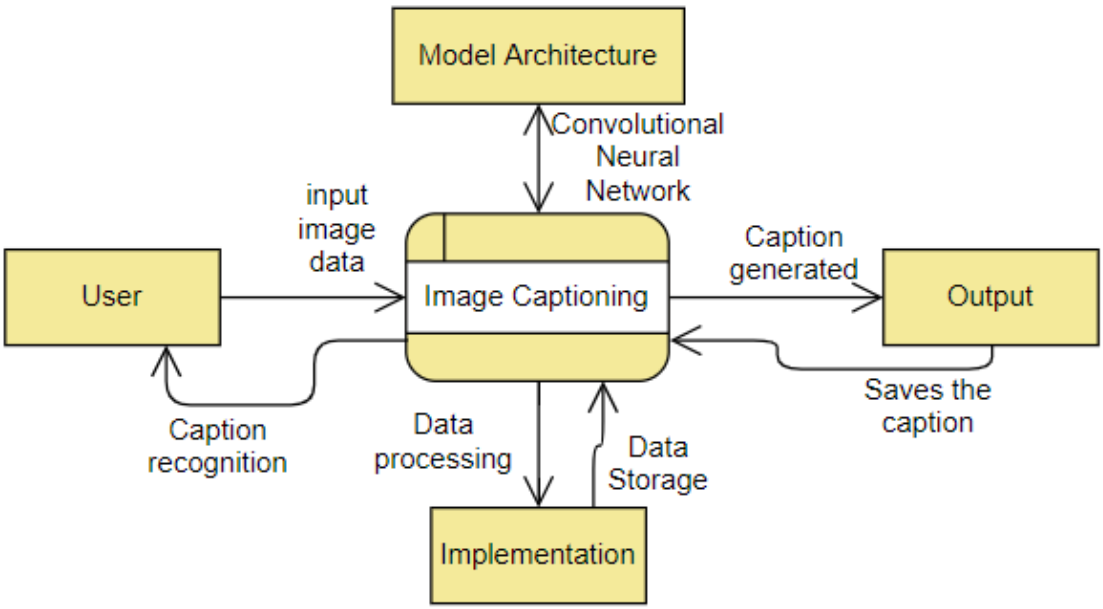


Fig 4.4: Level 0 DFD

Level 1 DFD

In level 1 DFD for Auto Image Captioning we identify processes. Each data-flow into the system must be received by a process. This level shows a detail system model constituting various functionalities associated with the system software. Add data-flows flowing between processes and data stores within the system.

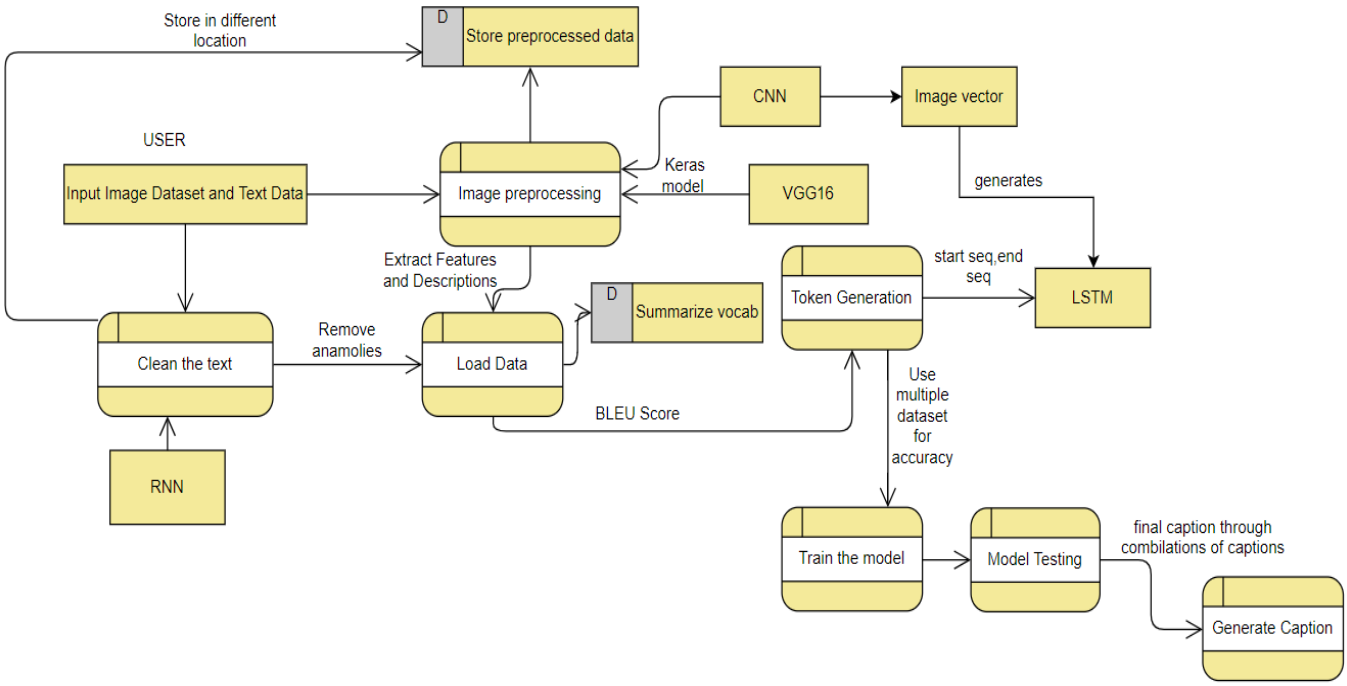


Fig 4.5: Level 1 DFD

Chapter 5

Design

5.1 Flow Design of The project

1. Data collection
2. Data Preprocessing — Text
3. Data Preprocessing — Images
4. Training
5. Testing
6. Model Architecture
7. Evaluation

5.1.1 Data Collection

Image dataset:

Fig 5.1: Image Dataset Sample

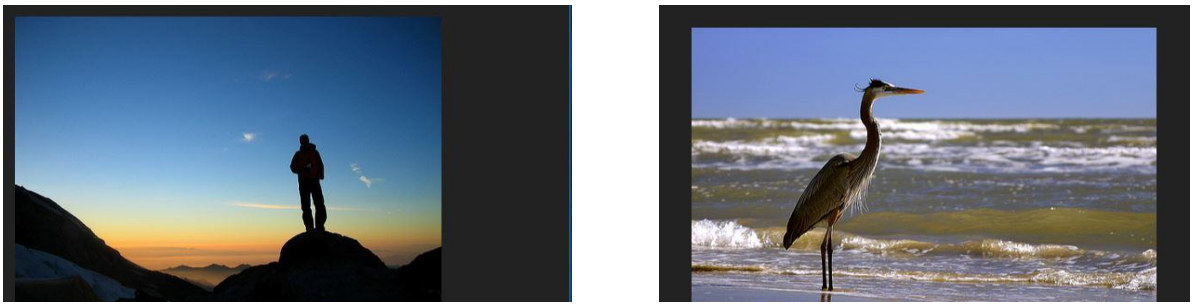


Fig 5.2: Individual Images from dataset

2) Text Dataset:

File Edit Format View Help

1000268201_693b08cb0e.jpg#0 A child in a pink dress is climbing up a set of stairs in an entry way .1000268201_693b08cb0e.jpg#1 A girl going into a wooden build
girl is sitting in front of a large painted rainbow .1002674143_1b742ab4b8.jpg#2 A small girl in the grass plays with fingerpaints in front of a white canvas wit
Blitz hat .1007129816_e794419615.jpg#3 A man with glasses is wearing a beer can crocheted hat .1007129816_e794419615.jpg#4 The man with pierced ears is wearing gla
12859_01547e3f17.jpg#0 A dog shakes its head near the shore , a red ball next to it .1012212859_01547e3f17.jpg#1 A white dog shakes on the edge of a beach with a
a log .1015584366_dfcec3c85a.jpg#1 A grey dog is leaping over a fallen tree .1015584366_dfcec3c85a.jpg#2 A large black dog leaps a fallen log .1015584366_dfcec3c
pictures in the snow next to trees .101669240_b2d3e7f17b.jpg#4 Man on skis looking at artwork for sale in the snow1016887272_03199f49c4.jpg#0 A collage of one person
ch a thrown object in a field with nearby cars .1019604187_d087bf9a5f.jpg#1 A white dog is about to catch a yellow ball in its mouth .1019604187_d087bf9a5f.jpg#2
pg#2 A little boy at a lake watching a duck .1022454332_6af2c1449a.jpg#3 A young boy waves his hand at the duck in the water surrounded by a green park .1022454
515238d8.jpg#4 This is a black dog splashing in the water .102351840_323e3de834.jpg#0 A man drilling a hole in the ice .102351840_323e3de834.jpg#1 A man is drillir
erfall .102455176_5f8ead62d5.jpg#3 A person in blue and red ice climbing with two picks .102455176_5f8ead62d5.jpg#4 Climber climbing an ice wall1026685415_0
ch a red toy .1030985833_b0902ea560.jpg#1 A black dog and a brown dog play with a red toy on a courtyard .1030985833_b0902ea560.jpg#2 A brown and black lab ar
jpg#1 A man sits and reads a newspaper by a sculpture outside of an office building .103195344_5d2dc613a3.jpg#2 a man sits near a large statue .103195344_5d2dc6
are standing in the grass and a person is sitting next to them1032122270_ea6f0beedb.jpg#3 Three dogs on a grassy hill1032122270_ea6f0beedb.jpg#4 Three dogs stand
g#0 People sit on the mountainside and check out the view .104136873_5b5d41be75.jpg#1 Three people are on a hilltop overlooking a green valley .104136873_5b5d
ile another man looks on .1042590306_95dea0916c.jpg#2 A multiracial couple posing for a picture1042590306_95dea0916c.jpg#3 Asian man and blond woman holding hands
0a5c7c.jpg#3 Crowd of people at the beach .1048710776_bb5b0a5c7c.jpg#4 Several young people sitting on a rail above a crowded beach .1052358063_eae6744153.jpg#
tails splashes in the shallow water .1053804096_ad278b25f1.jpg#1 A girls plays in the surf .1053804096_ad278b25f1.jpg#2 a girl with pigtails is playing in the c
a steel beam .1055753357_4fa3d8d693.jpg#4 Two men take a break from construction .1056249424_ef2a2e041c.jpg#0 The children are playing in the water .105624942
rides1056359656_662cee0814.jpg#1 a young blond girl with a magizine in her hands1056359656_662cee0814.jpg#2 A young girl on a train reads a book about trair
little kid is jumping off a high dive at the pool .1057089366_ca83da0877.jpg#4 The boy is jumping off a high diving board into the pool .1057210460_09c6f4c6c1.jpg#0
#2 A young boy in swimming trunks is walking with his arms outstretched on the beach .106490881_5a2dd9b7bd.jpg#3 Children playing on the beach .106490881_5a2dd9b
A dog jumps to catch a toy .1067675215_7336a694d6.jpg#0 A man in blue shorts is laying in the street .1067675215_7336a694d6.jpg#1 A man in blue shorts lays down c
b1b60.jpg#2 A dog is jumping to catch a object thrown at it ,1072153132_53d2bb1b60.jpg#3 A dog leaps while chasing a tennis ball through a grassy field .10721531
6_d86f2d3347.jpg#0 A group of eight people are gathered around a table at night .107582366_d86f2d3347.jpg#1 A group of people gathered around in the dark .1
es .1075881101_d55c46bece.jpg#3 A child covered in foam is climbing on a black inflatable ramp .1075881101_d55c46bece.jpg#4 A person covered in soapy water is getti
he can catch him .1077931201_1e0bb83105.jpg#3 The man is in the pool and throwing a small boy into the air .1077931201_1e0bb83105.jpg#4 While water droplets fly
wo horses watching a fire .1082252566_8c79beef93.jpg#0 A bulldog , a sheep dog , and a boxer standing in a yard .1082252566_8c79beef93.jpg#1 A large brown dog is pus
7d9633581.jpg#2 A white dog has its head on the ground .1084040636_97d9633581.jpg#3 A white dog is resting its head on a tiled floor with its eyes open .1084040636_
.jpg#4 Two young kids are playing in the water on an inflated toy .1087539207_9f77ab3aaf.jpg#0 A family playing on a tractor on a beautiful day1087539207_9f77ab3aaf.jp
dressed in a red bikini and a red feathered headress .108898978_7713be88fc.jpg#0 Skiiers walking up the hill through a forest .108898978_7713be88fc.jpg#1
rown dog runs down a path happily .1089181217_ee1167f7af.jpg#4 Energetic brown dog running1089755335_0bfbfd30e6.jpg#0 A mother and children is fishing on a boardwalk
sits on daft horse near a frozen lake .109202801_c6381eef15.jpg#0 Two draft horses pull a cart through the snow .109202801_c6381eef15.jpg#1 Two golden brown
rs109260218_fca831f933.jpg#2 Spelunkers pose inside a rock cavern while bathed in sunlight from the surface .109260218_fca831f933.jpg#3 three people sit in a ca
aborate chalk illustration .1093737381_b313cd49ff.jpg#4 A woman wearing a blue dress is drawing a picture of a boy and a girl on the sidewalk with sidewalk chalk .10944
le course1095580424_76f0aa8a3e.jpg#3 The dog is running out of the tunnel on a dog obedience course .1095580424_76f0aa8a3e.jpg#4 The small brown and white dog is

Fig 5.3: Text Dataset saved in .csv file

5.1.2 Data Text preprocessing - We transform all letters in the captions to lowercase and remove all the non alphabetic characters. Words occur less than five times are replaced with an unknown token <UNK>. We truncate all the captions longer than 16 tokens and set the maximum number of input words for CNNL to be 16.

Preprocessed Text:

```
print('Vocabulary Size: %d' % len(vocabulary))
# save to file
save_descriptions(descriptions, 'descriptions.txt')
```

```
Loaded: 8092
Vocabulary Size: 8763
```

Fig 5.4: Preprocessed Text

5.1.3 Data Image Preprocessing

Images are nothing but input (X) to our model. As you may already know that any input to a model must be given in the form of a vector. We need to convert every image into a fixed sized vector which can then be fed as input to the neural network.

This model was trained on the VGG Model to perform image classification on 6000 different images. However, our purpose here is not to classify the image but just get a fixed-length informative vector for each image. This process is called automatic feature engineering.

Preprocessed Images:

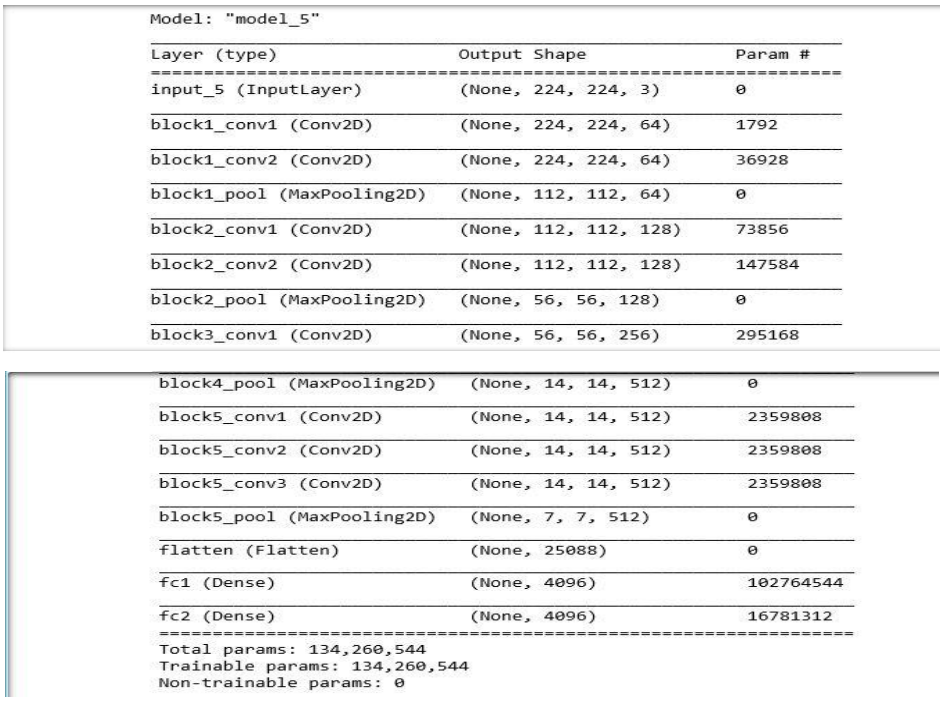


Fig 5.5,5.6 : Features of Images preprocessed

Fig 5.7: Preprocessed Images saved

5.1.4 Training

The output from the last hidden state of the CNN(Encoder) is given to the first time step of the decoder. We set x1 =<START> vector and the desired label y1 = first word in the sequence. Analogously, we set x2 =word vector of the first word and expect the network to predict the second word. Finally, on the last step, xT = last word, the target label yT =<END> token.

During training, the correct input is given to the decoder at every time-step, even if the decoder made a mistake before.

```
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
```

```
Dataset: 6000
Descriptions: train=6000
Photos: train=6000
```

Fig 5.8 : Loading dataset for training.

Fig 5.9: Training Dataset:

Training Output:

```
Dataset: 6000
Descriptions: train=6000
Photos: train=6000
Vocabulary Size: 7579
Description Length: 34
WARNING:tensorflow:From C:\Users\Dharmi Hemani\anaconda3\lib\site-packages\tensorflow\python\keras\backend.py:3794: add_dispat
ch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\Dharmi Hemani\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name t
f.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Epoch 1/1
6000/6000 [==============================] - 12198s 2s/step - loss: 4.6833
Epoch 1/1
6000/6000 [==============================] - 34199s 6s/step - loss: 3.8999
Epoch 1/1
6000/6000 [==============================] - 11154s 2s/step - loss: 3.6448
```

```
6000/6000 [==============================] - 38050s 6s/step - loss: 3.1421
Epoch 1/1
6000/6000 [==============================] - 12857s 2s/step - loss: 3.1241
Epoch 1/1
6000/6000 [==============================] - 14027s 2s/step - loss: 3.1115
Epoch 1/1
6000/6000 [==============================] - 12711s 2s/step - loss: 3.1096
Epoch 1/1
6000/6000 [==============================] - 12422s 2s/step - loss: 3.0912
Epoch 1/1
6000/6000 [==============================] - 45193s 8s/step - loss: 3.0827
Epoch 1/1
6000/6000 [==============================] - 12227s 2s/step - loss: 3.0757
Epoch 1/1
6000/6000 [==============================] - 11753s 2s/step - loss: 3.0705
Epoch 1/1
6000/6000 [==============================] - 13160s 2s/step - loss: 3.0643
Epoch 1/1
6000/6000 [==============================] - 35559s 6s/step - loss: 3.0647
```

Fig 5.10, 5.11: Dataset after training.

5.1.5 Testing

The image representation is provided to the first time step of the decoder. Set x1 =<START> vector and compute the distribution over the first word y1. We sample a word from the distribution (or pick the argmax), set its embedding vector as x2, and repeat this process until the <END> token is generated.

During Testing, the output of the decoder at time t is fed back and becomes the input of the decoder at time t+1

Testing Dataset output:

```
Dataset: 6000
Descriptions: train=6000
Vocabulary Size: 7579
Description Length: 34
Dataset: 1000
Descriptions: test=1000
Photos: test=1000
WARNING:tensorflow:From C:\Users\Dharmi Hemani\anaconda3\lib\site-packages\tensorflow\python\keras\backend.py:3794: add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From C:\Users\Dharmi Hemani\anaconda3\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

BLEU-1: 0.515548
BLEU-2: 0.276066
BLEU-3: 0.192688
BLEU-4: 0.091328
```

Fig 5.12: Output after Testing .

5.1.6 Model Architecture

Since the input consists of two parts, an image vector and a partial caption, we cannot use the Sequential API provided by the Keras library. For this reason, we use the Functional API which allows us to create Merge Models[3].

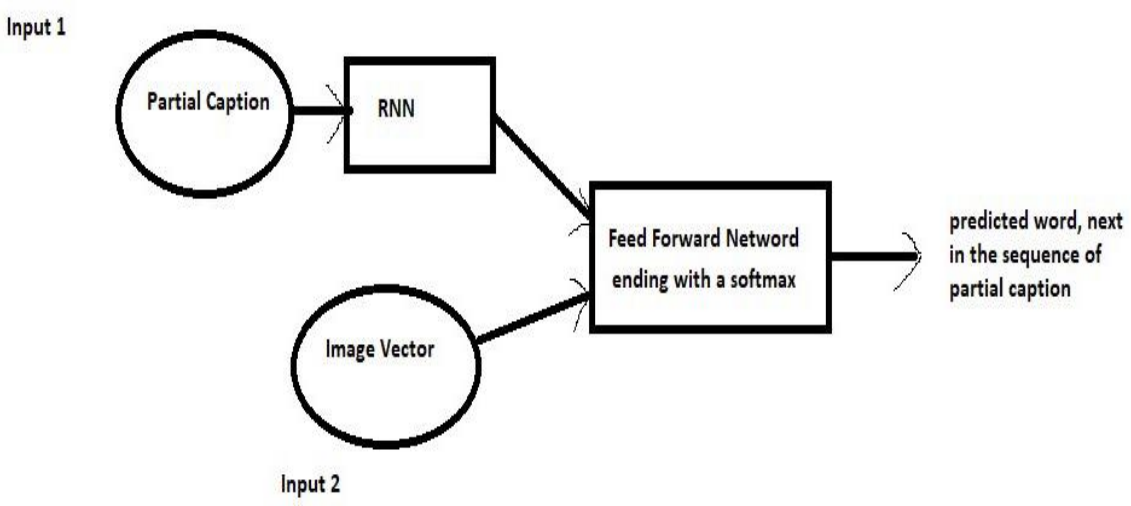


Fig 5.13: Basic Architecture of Image-Captioning.

5.1.7 Evaluation

To understand how good the model is, let's try to generate captions on images from the test dataset (i.e. the images which the model did not see during the training).

The evaluation can be observed in results.(Refer chap 6)

Chapter 6

Implementation & Results

6.1 Input:

STEP1 : TEXT PREPROCESSING

```
from keras.preprocessing.text import Tokenizer
from pickle import dump

# load doc into memory
def load_doc(filename):
# open the file as read only
file = open(filename, 'r')
# read all text
text = file.read()
# close the file
file.close()
return text
# load a pre-defined list of photo identifiers
def load_set(filename):
doc = load_doc(filename)
dataset = list()
# process line by line
for line in doc.split('\n'):
# skip empty lines
if len(line) < 1:
continue
# get the image identifier
identifier = line.split('.')[0]
dataset.append(identifier)
return set(dataset)
# load clean descriptions into memory
```

```
def load_clean_descriptions(filename, dataset):
# load document
doc = load_doc(filename)
descriptions = dict()
for line in doc.split('\n'):
# split line by white space
tokens = line.split()
# split id from description
image_id, image_desc = tokens[0], tokens[1:]
# skip images not in the set
if image_id in dataset:
# create list
if image_id not in descriptions:
descriptions[image_id] = list()
# wrap description in tokens
desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
# store
descriptions[image_id].append(desc)
return descriptions
# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
all_desc = list()
for key in descriptions.keys():
[all_desc.append(d) for d in descriptions[key]]
return all_desc
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
lines = to_lines(descriptions)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
return tokenizer
# load training dataset (6K)
filename=r'C:\Users\Dharmi
Hemani\anaconda3\Scripts\MLPROJ\Flickr8k_text\Flickr_8k.trainImages.txt'
```

```
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
# save the tokenizer
dump(tokenizer, open('tokenizer.pkl', 'wb'))
```

STEP 2: IMAGE PREPROCESSING

```
from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.models import load_model
# extract features from each photo in the directory
def extract_features(filename):
# load the model
model = VGG16()
# re-structure the model
model.layers.pop()
model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
# load the photo
image = load_img(filename, target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
```

```
# prepare the image for the VGG model
image = preprocess_input(image)
# get features
feature = model.predict(image, verbose=0)
return feature
# map an integer to a word
def word_for_id(integer, tokenizer):
for word, index in tokenizer.word_index.items():
if index == integer:
return word
return None
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
# seed the generation process
in_text = 'startseq'
# iterate over the whole length of the sequence
for i in range(max_length):
# integer encode input sequence
sequence = tokenizer.texts_to_sequences([in_text])[0]
# pad input
sequence = pad_sequences([sequence], maxlen=max_length)
# predict next word
yhat = model.predict([photo,sequence], verbose=0)
# convert probability to integer
yhat = argmax(yhat)
# map integer to word
word = word_for_id(yhat, tokenizer)
# stop if we cannot map the word
if word is None:
Break
# append as input for generating the next word
in_text += ' ' + word
# stop if we predict the end of the sequence
if word == 'endseq':
```

```
break
return in_text
# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from training)
max_length = 34
# load the model
model = load_model('model_19.h5')
# load and prepare the photograph
photo=extract_features(r'C:\Users\Dharmi
Hemani\anaconda3\Scripts\MLPROJ\example3.jpg')
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

STEP 3: TRAINING

```
from numpy import array
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint
# load doc into memory
def load_doc(filename):
# open the file as read only
```

```
file = open(filename, 'r')
# read all text
text = file.read()
# close the file
file.close()
return text
# load a pre-defined list of photo identifiers
def load_set(filename):
doc = load_doc(filename)
dataset = list()
# process line by line
for line in doc.split('\n'):
# skip empty lines
if len(line) < 1:
continue
# get the image identifier
identifier = line.split('.')[0]
dataset.append(identifier)
return set(dataset)
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
# load document
doc = load_doc(filename)
descriptions = dict()
for line in doc.split('\n'):
# split line by white space
tokens = line.split()
# split id from description
image_id, image_desc = tokens[0], tokens[1:]
# skip images not in the set
if image_id in dataset:
# create list
if image_id not in descriptions:
descriptions[image_id] = list()
```

```
# wrap description in tokens
desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
# store
descriptions[image_id].append(desc)
return descriptions
# load photo features
def load_photo_features(filename, dataset):
# load all features
all_features = load(open(filename, 'rb'))
# filter features
features = {k: all_features[k] for k in dataset}
return features
# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
all_desc = list()
for key in descriptions.keys():
[all_desc.append(d) for d in descriptions[key]]
return all_desc
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
lines = to_lines(descriptions)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
return tokenizer
# calculate the length of the description with the most words
def max_length(descriptions):
lines = to_lines(descriptions)
return max(len(d.split()) for d in lines)
# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, desc_list, photo, vocab_size):
X1, X2, y = list(), list(), list()
# walk through each description for the image
for desc in desc_list:
# encode the sequence
```

```
seq = tokenizer.texts_to_sequences([desc])[0]
# split one sequence into multiple X,y pairs
for i in range(1, len(seq)):
# split into input and output pair
in_seq, out_seq = seq[:i], seq[i]
# pad input sequence
in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
# encode output sequence
out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
# store
X1.append(photo)
X2.append(in_seq)
y.append(out_seq)
return array(X1), array(X2), array(y)
# define the captioning model
def define_model(vocab_size, max_length):
# feature extractor model
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence model
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)
# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
# tie it together [image, seq] [word]
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
# compile model
model.compile(loss='categorical_crossentropy', optimizer='adam')
# summarize model
```

```
#model.summary()
#plot_model(model, to_file='model.png', show_shapes=True)
return model
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, tokenizer, max_length, vocab_size):
# loop for ever over images
while 1:
for key, desc_list in descriptions.items():
# retrieve the photo feature
photo = photos[key][0]
in_img, in_seq, out_word = create_sequences(tokenizer, max_length, desc_list, photo, vocab_size)
yield [[in_img, in_seq], out_word]
# load training dataset (6K)
filename=r'C:\Users\Dharmi Hemani\anaconda3\Scripts\MLPROJ\Flickr8k_text\Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# define the model
model = define_model(vocab_size, max_length)
# train the model, run epochs manually and save after each epoch
```

```
epochs = 20
steps = len(train_descriptions)
for i in range(epochs):
# create the data generator
generator = data_generator(train_descriptions, train_features, tokenizer, max_length, vocab_size)
# fit for one epoch
model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
# save model
model.save('model_' + str(i) + '.h5')
```

STEP 4: TESTING & EVALUATION

```
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
# load doc into memory
def load_doc(filename):
# open the file as read only
file = open(filename, 'r')
# read all text
text = file.read()
# close the file
file.close()
return text
# load a pre-defined list of photo identifiers
def load_set(filename):
doc = load_doc(filename)
dataset = list()
# process line by line
for line in doc.split('\n'):
```

```
# skip empty lines
if len(line) < 1:
continue
# get the image identifier
identifier = line.split('.')[0]
dataset.append(identifier)
return set(dataset)
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
# load document
doc = load_doc(filename)
descriptions = dict()
for line in doc.split('\n'):
# split line by white space
tokens = line.split()
# split id from description
image_id, image_desc = tokens[0], tokens[1:]
# skip images not in the set
if image_id in dataset:
# create list
if image_id not in descriptions:
descriptions[image_id] = list()
# wrap description in tokens
desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
# store
descriptions[image_id].append(desc)
return descriptions
# load photo features
def load_photo_features(filename, dataset):
# load all features
all_features = load(open(filename, 'rb'))
# filter features
features = {k: all_features[k] for k in dataset}
return features
```

```
# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
all_desc = list()
for key in descriptions.keys():
[all_desc.append(d) for d in descriptions[key]]
return all_desc
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
lines = to_lines(descriptions)
tokenizer = Tokenizer()
tokenizer.fit_on_texts(lines)
return tokenizer
# calculate the length of the description with the most words
def max_length(descriptions):
lines = to_lines(descriptions)
return max(len(d.split()) for d in lines)
# map an integer to a word
def word_for_id(integer, tokenizer):
for word, index in tokenizer.word_index.items():
if index == integer:
return word
return None
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
# seed the generation process
in_text = 'startseq'
# iterate over the whole length of the sequence
for i in range(max_length):
# integer encode input sequence
sequence = tokenizer.texts_to_sequences([in_text])[0]
# pad input
sequence = pad_sequences([sequence], maxlen=max_length)
# predict next word
yhat = model.predict([photo,sequence], verbose=0)
```

```
# convert probability to integer
yhat = argmax(yhat)
# map integer to word
word = word_for_id(yhat, tokenizer)
# stop if we cannot map the word
if word is None:
break
# append as input for generating the next word
in_text += ' ' + word
# stop if we predict the end of the sequence
if word == 'endseq':
Break
return in_text
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
actual, predicted = list(), list()
# step over the whole set
for key, desc_list in descriptions.items():
# generate description
yhat = generate_desc(model, tokenizer, photos[key], max_length)
# store actual and predicted
references = [d.split() for d in desc_list]
actual.append(references)
predicted.append(yhat.split())
# calculate BLEU score
print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))
# prepare tokenizer on train set
# load training dataset (6K)
filename=r'C:\Users\Dharmi
Hemani\anaconda3\Scripts\MLPROJ\Flickr8k_text\Flickr_8k.trainImages.txt'
train = load_set(filename)
```

```
print('Dataset: %d' % len(train))
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# prepare test set
# load test set
filename1=r'C:\Users\Dharmi
Hemani\anaconda3\Scripts\MLPROJ\Flickr8k_text\Flickr_8k.testImages.txt'
test = load_set(filename1)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
# load the model
filename3 = 'model_19.h5'
model = load_model(filename3)
# evaluate model
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

6.2 Output:

Example 1:


```
startseq man in red shirt is standing in front of an outdoor building endseq
Out[1]:
```

Fig 6.1 Image inputed and caption generated

Example 2:

```
startseq two dogs are playing in the grass endseq
Out[2]:
```

Fig 6.2:Image and Caption Generated.

Example 3:

```
startseq man in red jacket is standing in the mountains endseq
Out[2]:
```

Fig 6.3:Image and Caption Generated.

Chapter 7
Conclusion

In this project , we present an image captioning model with language CNN for formation in sequence for image caption generation. Experiments conducted on Flickr30K image captioning datasets validate our purpose and analysis.

We intend to use “BLEU metric”, which is widely used by machine translation experts. BLEU is precision based and compares system output to human translation. BLEU breaks captions to words and we compare these with human translations. It’s like a bag of similar words. There is an inherent problem with this metric system, given the fact that different people use different vocabulary to represent a similar meaning, sometimes with no overlap. This is also due to the fact that language models have limited vocabulary and training set.

So finally in this project, we learned:

1. How a convolutional neural network and LSTM can be combined to generate captions to an image.
2. How to utilize the beam search algorithm to consider multiple captions and select the most probable sentence.

Appendix

Table 1: Definitions for most commonly used terms

Sr. no.	Term	Definition
1.	Natural language processing	Natural language processing is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. As such, NLP is related to the area of human–computer interaction.
2.	recurrent neural network	A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state to process variable length sequences of inputs. This makes them applicable to tasks such as image processing.
3.	Convolutional Neural Network	A convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks, based on their shared-weights architecture and translation invariance characteristics.They have applications in image and video recognition and image classification.
4.	Long short-term memory	Long short-term memory is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data.

References

[1] P. Anderson, B. Fernando, M. Johnson, and S. Gould. Spice: Semantic propositional image caption evaluation, 2016. “Deep visual-semantic alignments for generating image descriptions,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3128–3137, 2015. Germany, 2016.

[2] Jiuxiang Gu1, Gang Wang2, Jianfei Cai3, Tsuhan Chen3, “An Empirical Study of Language CNN for Image Captioning”, in IEEE 2017. ROSE Lab, Interdisciplinary Graduate School, Nanyang Technological University, Singapore 2 Alibaba AI Labs, Hangzhou, China 3 School of Computer Science and Engineering, Nanyang Technological University, Singapore.

[3] Parth Shah, Vishvajit Bakarola and Supriya Pati “Learning phrase representations using rnn encoder-decoder for statistical machine translation”. EMNLP, 2014. pp. 3128–3137, 2015. Chhotubhai Gopalbhai Patel Institute of Technology, Bardoli, India.

[4] Orchid Engg, (Jan 2018). Image Captioning using Deep Neural Architectures. [online]arXiv.org.Available at: https://arxiv.org/abs/1801.05568 [Accessed Feb 24, 2020].

[5] Harshall Lamba, Data Science , s. (2018). Teaching Computers to describe pictures , Image Captioning with Keras.[online]towardsdatascience.org. Available at: https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8 [Accessed 20 March 2020].

[6] Faizan, Analytics Vidya (2018).Automatic Image Captioning using Deep Learning (CNN and LSTM).[online]analyticsvidya.com. Available at: https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/ [Accessed 3 Apr. 2020].

Acknowledgements

This group project that has been assigned, gave us a tremendous impact toward ourselves. The impact was very beneficial for us in terms of enhancing our knowledge in Machine Learning and Deep Learning and how to work with each other to perform all the best for the three of us.

We take the opportunity to thank all those who have helped and guided us through this project and made this experience worthwhile for us. We would like to thank HOD of the Computer department **Dr. Kavita Sonawane** for her support and cooperation. We would like to thank our teacher and guide professor **Mrs. Dakshata Panchal** who gave us her valuable suggestions and ideas and also encouraged us to explore this domain.