NEURAL IMAGE CAPTIONING

MAJOR PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(COMPUTER SCIENCE AND ENGINEERING)

Submitted By:

Ankit Nitesh (1606645)

Kumar Ambuj (1606719)

Manish Kumar (1606723)

Submitted To:

Prof. Jasdeep Kaur

Assistant Professor

Department of Computer Science & Engineering

Guru Nanak Dev Engineering College

Ludhiana - 141006

ABSTRACT

Image captioning means automatically generating a caption for an image. As a recently emerged research area, it is attracting more and more attention. To achieve the goal of image captioning, semantic information of images needs to be captured and expressed in natural languages. Connecting both research communities of computer vision and natural language processing, image captioning is a quite challenging task. Various approaches have been proposed to solve this problem. In this paper we present a survey on advances in image captioning research. Based on the technique adopted, we classify image captioning approaches into different categories. Representative methods in each category are summarized, and their strengths and limitations are talked about. In this paper, we first discuss methods used in early work which are mainly retrieval and template based. Then, we focus our main attention on neural network based methods, which give state of the art results. After that, state of the art methods are compared on benchmark datasets. Following that, discussions on future research directions are presented.

Language models based on recurrent neural networks have dominated recent image caption generation tasks. In this paper, we introduce a language CNN model which is suitable for statistical language modeling tasks and shows competitive performance in image captioning. In contrast to previous models which predict next word based on one previous word and hidden state, 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 MS COCO. Our extensive experimental results show that our method outperforms the vanilla recurrent neural network based language models and is competitive with the state-of-the-art methods.

Image captioning models typically follow an encoder-decoder architecture which uses abstract image feature vectors as input to the encoder. One of the most successful algorithms uses feature vectors extracted from the region proposals obtained from an object detector. In this work we introduce the Object Relation Transformer that builds upon this approach by explicitly incorporating information about the spatial relationship between input detected objects through geometric attention. Quantitative and qualitative results demonstrate the importance of such geometric attention for image captioning, leading to thw improvements on all common captioning metrics on the MS-COCO dataset.

ACKNOWLEDGEMENT

We are highly grateful to the Dr. Sehajpal Singh, Principal, Guru Nanak Dev Engineering College (GNDEC), Ludhiana, for providing this opportunity to carry out the major project work at premises itself.

The constant guidance and encouragement received from Dr. Parminder Singh, H.O.D. CSE Department, GNDEC Ludhiana has been of great help in carrying out the project work and is acknowledged with reverential thanks.

We would like to express a deep sense of gratitude and thanks profusely to Prof. Gurjit Kaur, without her wise counsel and able guidance, it would have been impossible to complete the project in this manner.

We express gratitude to other faculty members of computer science and engineering department of GNDEC for their intellectual support throughout the course of this work.

Finally, we are indebted to all whosoever have contributed in this report work.

Ankit Nitesh

Kumar Ambuj

Manish Kumar

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1

INTRODUCTION

1.1 Introduction to project:

Image captioning is a task of mapping images to text for human consumption. Broadly speaking, in order for a caption to be good it must satisfy two requirements: it should be a fluent, well-formed phrase or sentence in the target language; and it should be informative, or descriptive, conveying meaningful non-trivial information about the visual scene it describes. Our goal in the work presented here is to improve captioning on both of these fronts.

This is a challenging problem as the system should understand the subcomponents of an image and understand the complete context of the image. An ideal image retrieval system should display images which are more relevant to the query. Suppose that we asked you to caption an image, that is to describe the image using a sentence. This, when done by computers, is the goal of image captioning research. To train a network to accurately describe an input image by outputting a natural language sentence. Image retrieval by a textual query is being used in most of the image search systems. The search of images using textual query primarily depends on metadata of images. Images with metadata similar to the textual query are displayed as results. The above methodology relies on humans to annotate images. Desired results might not be obtained if there is an error in human annotations of metadata or if the metadata doesn't define the context behind an image. People are busy in their professional life and they forget to pay attention to their surrounding problems, which can lead to many hazards. One of the main problems is the Garbage collection. Many times, it has been observed that the garbage which should be inside the dustbin is actually lying outside the dustbin and causing the pollution of the surrounding environment, which further leads to numerous diseases.

Artificial Intelligence(AI) is now at the heart of innovation economy and thus the base for this project is also the same. In 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 increating image encodings. We use the VGG16 architecture proposed by with some modifications. Ever since researchers started working on object recognition in images, it became clear that only providing the names of the objects recognized does not make such a good impression as a full human-like description. As long as machines do not think, talk, and behave like humans, natural language descriptions will remain a challenge to be solved.

1.2 Project Category:

Research Based:

Image captioning means automatically generating a caption for an image. As a recently emerged research area, it is attracting more and more attention. To achieve the goal of image captioning, semantic information of images needs to be captured and expressed in natural languages. Connecting both research communities of computer vision and natural language processing, image captioning is a quite challenging task. Various approaches have been proposed to solve this problem. In this paper, we present a survey on advances in image captioning research. Based on the technique adopted, we classify image captioning approaches into different categories. Representative methods in each category are summarized, and their strengths and limitations are talked about. In this paper, we first discuss methods used in early work which are mainly retrieval and template based. Then, we focus our main attention on neural network based methods, which give state of the art results. Neural network based methods are further divided into subcategories based on the specific framework they use. Each subcategory of neural network based methods are discussed in detail. After that, state of the art methods are compared on benchmark datasets. Following that, discussions on future research directions are presented.

1.3 Objectives:

I. Automatically generate a sentence for an Image in a real type environment.
II. Capture semantic information of images and express in natural language.
III. Creating a model with high accuracy and reducing the time as much as possible while generating captions.

1.4 Problem Formulation:

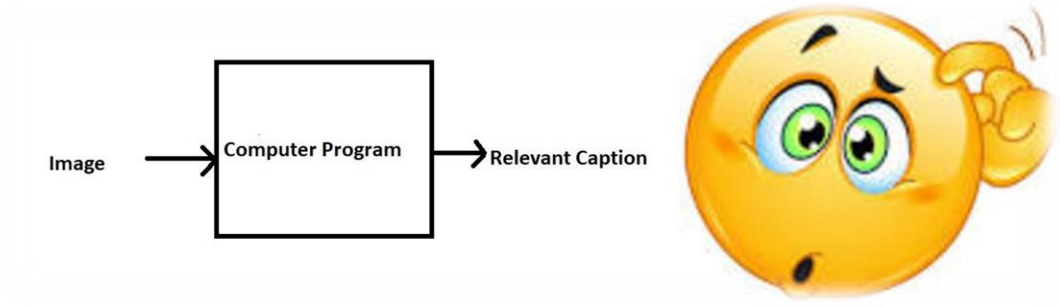
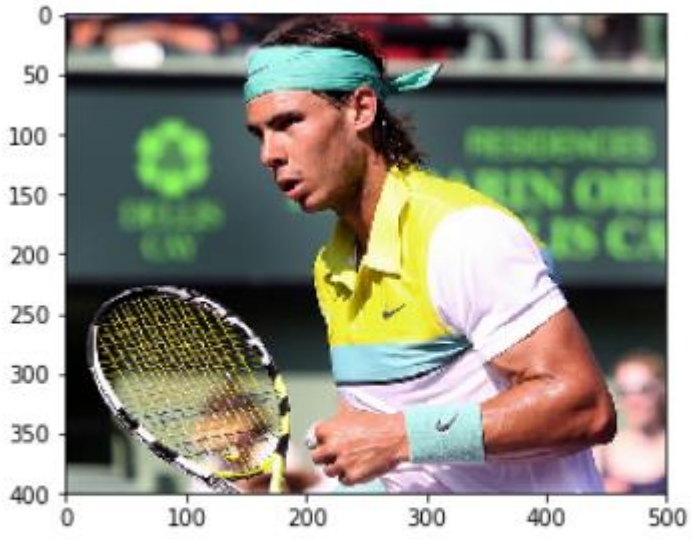


Fig 1.4.1 Problem



```
Greedy: a woman in a tennis racket on the court .
```

Fig 1.4.2 Required 1st Caption

```
Greedy: a little boy is on the water on the floor with a over the floor
```

Fig 1.4.3 Required 2nd Caption

Our target is to bring captions to these images like given above.

We must understand that the images used for testing must be semantically related to those used for training the model. For example, if we train our model on the images of cats, dogs, etc. we must not test it on images of air planes, waterfalls, etc. This is an example where the distribution of the train and test sets will be very different and in such cases no Machine Learning model in the world will give good performance.

1.5 Identification / Reorganization of Need:

I. SkinVision : Lets you confirm weather a skin condition can be skin cancer or not.
II. Google Photos: Classify your photo into Mountains, sea etc.
III. Deepmind: Achieved superhuman level playing Game Atari.
IV. Facebook: Using AI to classify, segmentate and finding patterns in pictures.
V. A U.S. company is predicting crop yield using images from satellite.
VI. Fed Ex and other courier services: Are using hand written digit recognition system.
VII. Picasa : Using facial Recognition to identify your friends and you in a group picture.
VIII. Tesla/Google Self Drive Cars.
IX. Automatic Image captioning requires both Image analysis and neural network.

1.6 Existing System:

In early image captioning work, another type of methods that are commonly used is template based. In template based methods, image captions are generated through a syntactically and semantically constrained process. Typically, in order to use a template based method to generate a description for an image, a specified set of visual concepts need to be detected first. Then, the detected visual concepts are connected through sentence templates or specific language grammar rules or combinatorial optimization algorithms to compose a sentence. A method to use a sentence template for generating image descriptions is presented in by Yang et al., is utilized as a sentence template. To describe an image, the authors first use detection algorithms to estimate objects and scenes in this image. Then, they employ a language model trained over the Gigaword corpus 3 to predicate verbs, scenes and prepositions that may be used to compose the sentence. With probabilities of all elements computed, the best quadruplet is obtained by using Hidden Markov Model inference. Finally, the image description is generated by filling the sentence structure given by the quadruplet. Kulkarni et al. employ Conditional Random Field to determine image contents to be rendered in the image caption. In their method, nodes of the graph correspond to objects, object attributes and spatial relationships between objects, respectively. In the graph model, unary potential functions of nodes are obtained by using corresponding visual models, while pairwise potential functions are obtained by making statistics on a collection of existing descriptions. Image contents to be described are determined by performing Conditional Random Field inference.

1.7 Proposed System:

Neural Image Captioning is a visual pattern recognition problem. Image Captioning, as one of the research based field, became more and more important owing to rapid advances in technologies such as digital cameras, internet and mobile devices, and increased demands on security. Image Captioning has several advantages over other biometric technologies: It is natural, nonintrusive, and easy to use. The proposed research is carried out in the following manner using unimodal and multimodal biometric trait to identify the best approach for Multimodal Biometric Authentication System (MMBAS). ¾ Face detection system with a combination of Cryptographic techniques Message Digest (MD5) and Secure Hash Algorithm (SHA1) with Elastic Bunch Graph Matching (EBGM) algorithm ¾ Face recognition using Scale Invariant Feature Transform (SIFT) ¾ A combination of face and finger print authentication system using Bit Plane Complexity Segmentation (BPCS) method with steganography Many recent researches show that local features are more effective to describe the detailed and stable information of an image. Applications are in the real time environment to implement the recognition system, like E-trading, online 43 banking, E-voting or internet voting etc. But among these, the online remote voter registration systems face some critical security problems (Election Law Blog 2007). These problems are mainly related to the inability to accurately verify the identity of the voter, which can facilitate impersonation or multiple registrations by the same voter with different data. The proposed system takes this issue as an application domain and it introduces a remote voter registration scheme.

CHAPTER - 2

Requirement Analysis and System Specification

2.1 Feasibility Study:

Feasibility is the determination of whether or not a project is worth doing. The process followed in making this determination is called feasibility study. Feasibility Study concentrates on the following areas:

I. Technical Feasibility
II. Economic Feasibility
III. Operational Feasibility

I. Technical Feasibility:

This is concerned with specifying equipment and software that will successfully satisfy the use considerably, but might include the feasibility to produce output in a given time because system is fast enough to handle multiple users.

II. Economic Feasibility:

Economic analysis is the most frequent used technique used for evaluating the effectiveness of a proposed system. This System is Cost effective as there is no need of having separate peoples for evaluating the messages of peoples is smart enough to predict.

III. Operational Feasibility:

It is mainly related to human organizational as social aspects. The points to be considered are – The system interface is standard,user friendly and provides extensive help. Hence no special training is required.

2.2 Software Requirement Specification:

2.2.1 Data Requirement:

There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

But for the purpose of this case study, I have used the Flickr 8k dataset which you can download by filling this **form** provided by the University of Illinois at Urbana-Champaign. Also training a model with large number of images may not be feasible on a system which is not a very high end PC/Laptop. This dataset contains 8000 images each with 5 captions (as we have already seen in the Introduction section that an image can have multiple captions, all being relevant simultaneously).
These images are bifurcated as follows:

- Training Set — 6000 images
- Dev Set — 1000 images
- Test Set — 1000 images

2.2.2 Functional Requirement:

It requires following tools for implementation:

Python:

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales.

Tensorflow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
Google colaboratory: Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

Jupyter Notebook:

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself.

Keras:

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow. Designed to enable fast experimentation with deep neural networks, it focuses on being

user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System),[5] and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network mode.

2.2.3 Performance Requirement:

The system must be interactive and the delays involved must be less .So in every action-response of the system, there are no immediate delays. In case of opening windows forms, of popping error messages and saving the settings or sessions there is delay much below 2 seconds, In case of opening databases, sorting questions and evaluation there are no delays and the operation is performed in less than 2 seconds for opening ,sorting, computing, posting > 95% of the files. Also when connecting to the server the delay is based editing on the distance of the 2 systems and the configuration between them so there is high probability that there will be or not a successful connection in less than 20 seconds for sake of good communication.

2.2.4. Dependability Requirement:

Software dependability is a pressing concern for several reasons:

- Developing software to meet existing dependability criteria is notoriously difficult and expensive. Large software projects fail at a rate far higher than other engineering projects, and the cost of projects that deliver highly dependable software is often exorbitant.
- Software failures have caused serious accidents that resulted in death, injury, and large financial losses. Without intervention, the increasingly pervasive use of software may bring about more frequent and more serious accidents.
- Existing certification schemes that are intended to ensure the dependability of software have a mixed record. Some are largely ineffective, and some are counterproductive.
- Software has great potential to improve safety in many areas. Improvements in dependability would allow software to be used more widely and with greater confidence for the benefit of society.

2.2.5 Maintainability requirement:

Maintainability is the ability of the application to go through changes with a fair degree of effortlessness. This attribute is the flexibility with which the application can be modified, for fixing issues, or to add new functionality with a degree of ease. These changes could impact components, services, functionality, and interfaces when modifying for fixing issues, or to meet future demands. Maintainability has a direct baring on the time it takes to restore the application to normal status following a failure or an upgrade. Enlightened maintainability attributes will enhance availability and reduce runtime defects. Maintainability is a function of the overall software quality attributes.

2.2.6 Security requirement:

Security requirements can be formulated on different abstraction levels. At the highest abstraction level they basically just reflect security objectives. An example of a security objectives could be "The system must maintain the confidentially of all data that is classified as confidential".

More useful for a SW architect or a system designer are however security requirements that describe more concretely what must be done to assure the security of a system and its data. OSA suggests to distinguish 4 different security requirement types:

- **Secure Functional Requirements**, this is a security related description that is integrated into each functional requirement. Typically this also says what shall not happen. This requirement artifact can for example be derived from misuse cases
- **Functional Security Requirements**, these are security services that needs to be achieved by the system under inspection. Examples could be authentication, authorization, backup, server-clustering, etc. This requirement artifact can be derived from best practices, policies, and regulations.
- **Non-Functional Security Requirements**, these are security related architectural requirements, like "robustness" or "minimal performance and scalability". This requirement type is typically derived from architectural principals and good practice standards.
- **Secure Development Requirements**, these requirements describe required activities during system development which assure that the outcome is not subject to vulnerabilities. Examples could be "data classification", "coding guidelines" or "test methodology". These requirements are derived from corresponding best practice frameworks like "CLASP".

2.2.7. Look And Feel Requirement:

The look and feel requirements describe the intended spirit, the mood, or the style of the product's appearance. These requirements specify the intention of the appearance, and are not a detailed design of an interface. For example, suppose you have a look and feel requirement like this:

This requirement does not say the company logo must be prominent, nor does it talk about the colors to be used. It simply states that the product must comply with whatever branding standards your organization has. These standards are published else where your own organization has a department or group responsible for these standard the designer has access to them. The fit criterion, when you add it, measures compliance with the standards.

Consider the look and feel requirements that you might build into your next product. Among other appearances appropriate for your product, you might want it to have the following characteristics:

2.3 Validation:

The validation in the project basically refers to the testing of the project. The evaluations of Neural Style Transfer algorithm remain an open and important problem in this field. In general, there are two major types of evaluation methodologies that can be employed in this field, i.e., qualitative evaluation and quantitative evaluation. Qualitative evaluation relies on the aesthetic judgements of observers. The evaluation results are related to lots of factors (e.g., age and occupation of participants). While quantitative evaluation focuses on the precise evaluation metrics, which include time complexity, loss variation, etc.

2.4 Expected Hurdles:

- Improper knowledge of the project.
- Improper knowledge of frameworks and languages.
- Absence of GPU

2.5 SDLC model to be used:

SDLC model used in this project is Waterfall Model. In the waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The waterfall Model illustrates the software development process in a linear sequential flow.

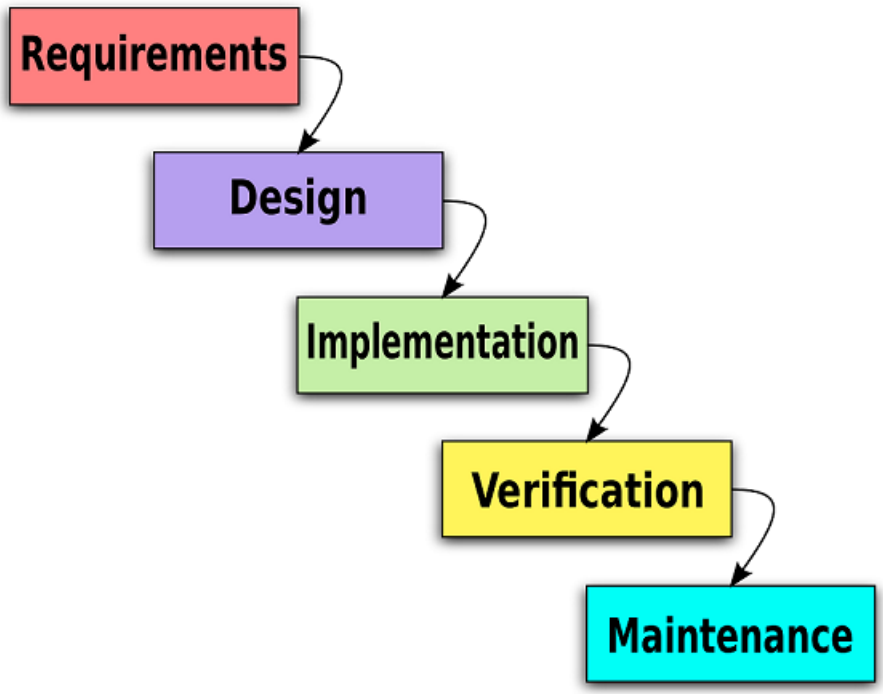


Fig 2.5.1 Waterfall Model

2.5.1 SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

2.5.2 Use cases for the Waterfall SDLC model:

- The requirements are precisely documented.
- Product definition is stable.
- The technologies stack is predefined which makes it not dynamic
- No ambiguous requirements
- The project is short

CHAPTER - 3

Implementation, Testing, and Maintenance

3.1. Implementation:

The implementation steps are as follows:

I. Detect a set of words that may be part of the image caption. We detect the words from the given vocabulary according to the content of the corresponding image based on the weak monitoring method in multi-instance learning (MIL) in order to train the detectors iteratively

II. Running a fully convolutional network on an image, we get a rough spatial response graph. Each position in the response map corresponds to a response obtained by applying the original CNN to the region of the input image where the shift is shifted (thus effectively scanning different locations in the image to find possible objects). By up-sampling the image, we get a response map on the final fully connected layer and then implement the noisy-OR version of MIL on the response map for each image. Each word produces a single probability.

III. The process of caption generation is searching for the most likely sentence under the condition of the visually detected word set. The language model is at the heart of this process because it defines the probability distribution of a sequence of words. Although the maximum entropy language model (ME) is a statistical model, it can encode very meaningful information. For example, "running" is more likely to follow the word "horse" than "speaking." This information can help identify the wrong words and encode commonsense knowledge.

IV. There are similar ways to use the combination of attribute detectors and language models to process image caption generation. Devlin et al used a combination of CNN and k-NN methods and a combination of a maximum entropy model and RNN to process image description generation tasks. Kenneth Tran proposed an image description system, using CNN as a visual model to detect a wide range of visual concepts, landmarks, celebrities, and other entities into the language model, and the output results are the same as those extracted by CNN. The vectors together are used as input to the multichannel depth-similar model to generate a description.

- **Python:**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python.

- **Jupyter Notebook:**

The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text. Jupyter Notebook is maintained by the people at Project Jupyter.

Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

- **Tensorflow:**

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

- **Google Colaboratory:**

Colaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser.

3.2 Coding standards of Language used:

3.2.1 Data Collection:

There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

But for the purpose of this case study, I have used the Flickr 8k dataset which you can download by filling this form provided by the University of Illinois at Urbana-Champaign. Also training a model with large number of images may not be feasible on a system which is not a very high end PC/Laptop.

This dataset contains 8000 images each with 5 captions (as we have already seen in the Introduction section that an image can have multiple captions, all being relevant simultaneously).

These images are bifurcated as follows:

- Training Set — 6000 images
- Dev Set — 1000 images
- Test Set — 1000 images

3.2.2 Understanding the data:

We will clean the text in the following ways in order to reduce the size of the vocabulary of words we will need to work with:

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

Ideally, we want a vocabulary that is both expressive and as small as possible. A smaller vocabulary will result in a smaller model that will train faster.

For reference, we can transform the clean descriptions into a set and print its size to get an idea of the size of our dataset vocabulary.

3.2.3 Data Cleaning:

Data cleaning is the process of preparing data for analysis by removing or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. This data is usually not necessary or helpful when it comes to analyzing data because it may hinder the process or provide

inaccurate results. There are several methods for cleaning data depending on how it is stored along with the answers being sought.

- Data cleaning is not simply about erasing information to make space for new data, but rather finding a way to maximize a data set's accuracy without necessarily deleting information.
- When we deal with text, we generally perform some basic cleaning like lower-casing all the words (otherwise"hello" and "Hello" will be regarded as two separate words), removing special tokens (like '%', '$', '#', etc.), eliminating words which contain numbers (like 'hey199', etc.).
- Create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (corpus) in the data set.
- If we think about it, many of these words will occur very few times, say 1, 2 or 3 times. Since we are creating a predictive model, we would not like to have all the words present in our vocabulary but the words which are more likely to occur or which are common. This helps the model become more robust to outliers and make less mistakes.

3.2.4 Loading the training set:

Now, we load the descriptions of these images from "descriptions.txt" (saved on the hard disk) in the Python dictionary "train_descriptions". You can load this using pandas, and pass the NumPy arrays to TensorFlow. If you need to scale up to a large set of files, or need a loader that integrates with TensorFlow and tf.data then use the tf.data.experimental.make_csv_dataset function.

The only column you need to identify explicitly is the one with the value that the model is intended to predict.

However, when we load them, we will add two tokens in every caption as follows (significance explained later):

- 'startseq' -> This is a start sequence token which will be added at the start of every caption.
- 'endseq' -> This is an end sequence token which will be added at the end of every caption.

3.2.5 Data Preprocessing – Images:

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. For this purpose, we opt for transfer learning by using the InceptionV3 model (Convolutional Neural Network) created by Google Research.
- This model was trained on Imagenet dataset to perform image classification on 1000 different classes of images. However, our purpose here is not to classify the image but just get fixed-length informative vector for each image. This process is called automatic feature engineering.

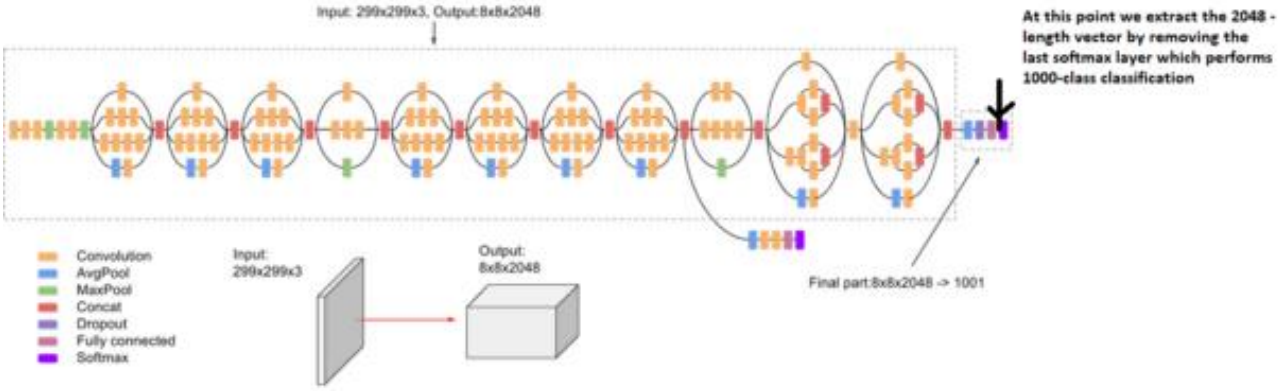


Fig 3.2.5 Feature Vector Extraction (Feature Engineering) from InceptionV3

3.2.6 Data Preprocessing - Captions:

We must note that captions are something that we want to predict. So during the training period, captions will be the target variables (Y) that the model is learning to predict.

- But the prediction of the entire caption, given the image does not happen at once. We will predict the caption word by word.
- Thus, we need to encode each word into a fixed sized vector. However, this part will be seen later when we look at the model design, but for now we will create two Python Dictionaries namely “wordtoix” (pronounced — word to index) and “ixtoword” (pronounced — index to word).
- Stating simply, we will represent every unique word in the vocabulary by an integer (index). As seen above, we have 1652 unique words in the corpus and thus each word will be represented by an integer index between 1 to 1652.

3.2.7 Data Preparation using Generator Function:

This is one of the most important steps in this case study. Here we will understand how to prepare the data in a manner which will be convenient to be given as input to the deep learning model. Hereafter, I will try to explain the remaining steps by taking a sample example as follows:

Consider we have 3 images and their 3 corresponding captions as follows:

Fig 3.2.7.1 (Train image 1) Caption -> The black cat sat on grass

Fig 3.2.7.2 Train image 2) Caption -> The white cat is walking on road

Fig 3.2.7.3 (Test image) Caption -> The black cat is walking on grass

Now, let's say we use the first two images and their captions to train the model and the third image to test our model.

Now the questions that will be answered are:

- **How do we frame this as a supervised learning problem?**
- **What does the data matrix look like? How many data points do we have?, etc.**

Ans:

- First we need to convert both the images to their corresponding 2048 length feature vector as discussed above. Let "Image_1" and "Image_2" be the feature vectors of the first two images respectively.
- Secondly, let's build the vocabulary for the first two (train) captions by adding the two tokens "startseq" and "endseq" in both of them: (Assume we have already performed the basic cleaning steps).
- Caption_1 -> "startseq the black cat sat on grass endseq" Caption_2 -> "startseq the white cat is walking on road endseq"
- vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}.

So how does using a generator function solve this problem?

If you know the basics of Deep Learning, then you must know that to train a model on a particular dataset, we use some version of Stochastic Gradient Descent (SGD) like Adam, Rmsprop, Adagrad, etc.

- With SGD, we do not calculate the loss on the entire data set to update the gradients. Rather in every iteration, we calculate the loss on a batch of data points (typically 64, 128, 256, etc.) to update the gradients.
- This means that we do not require to store the entire dataset in the memory at once. Even if we have the current batch of points in the memory, it is sufficient for our purpose.
- A generator function in Python is used exactly for this purpose. It's like an iterator which resumes the functionality from the point it left the last time it was called.

3.2.8 Word Embedding:

A word embedding is a learned representation for text where words that have the same meaning have a similar representation. It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

Word embedding are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

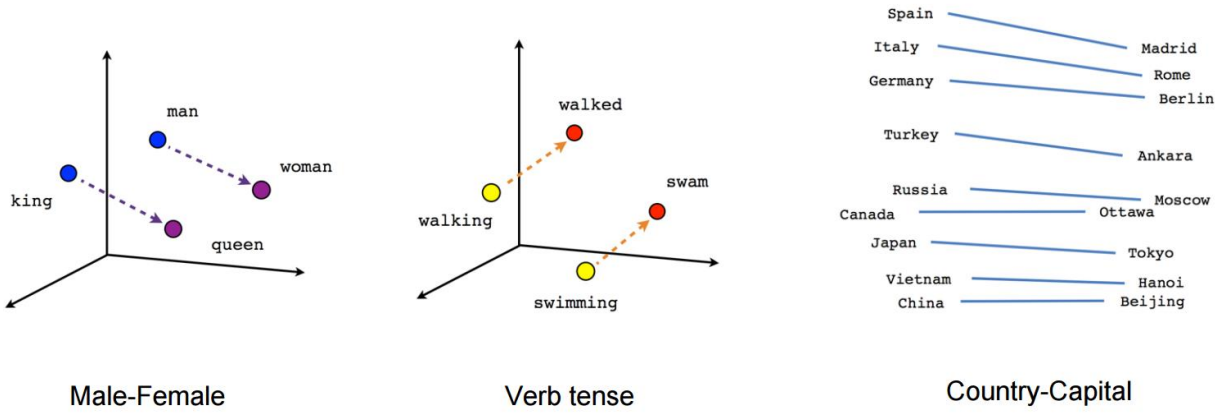


Fig 3.2.8 From Count Vectors to Word2Vec

- Key to the approach is the idea of using a dense distributed representation for each word.
- Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one-hot encoding.
- We will reviews three techniques that can be used to learn a word embedding from text data.
    - Embedding Layer
    - Word2Vec
    - GloVe

3.2.9 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.

This generally makes sense because during the later stages of training, since the model is moving towards convergence, we must lower the learning rate so that we take smaller steps towards the minima. Also increasing the batch size over time helps your gradient updates to be more powerful.

First, let's look at the brief architecture which contains the high level sub-modules:

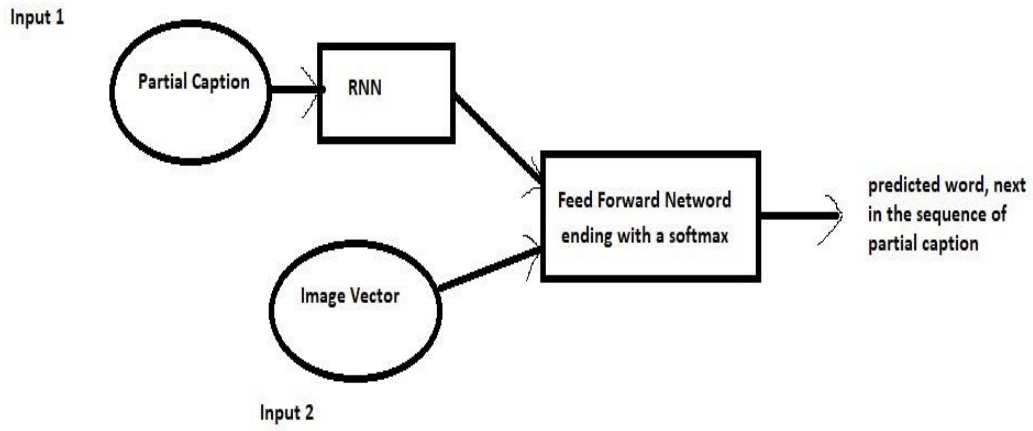


Fig 3.2.9 High level architecture

We will describe the model in three parts:

I. Photo Feature Extractor. This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.

II. Sequence Processor. This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.

III. Decoder (for lack of a better name). Both the feature extractor and sequence processor output a fixed- length vector.

3.2.10 Caption Caption Generator:

So till now, we have seen how to prepare the data and build the model. In the final step of this series, we will understand how do we test (infer) our model by passing in new images, i.e. how can we generate a caption for a new test image.

Fig 3.2.10 Test image

Now let's use the third image and try to understand how we would like the caption to be generated. The third image vector and caption were as follows:

Caption: the black cat is walking on grass

Also the **vocabulary** in the example was:

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

We will generate the caption iteratively, one word at a time as follows:

Iteration 1:

- Input: Image vector + "startseq" (as partial caption)
- Expected Output word: "the"

(You should now understand the importance of the token 'startseq' which is used as the initial partial caption for any image during inference).

This is called as Maximum Likelihood Estimation (MLE) i.e. we select that word which is most likely according to the model for the given input. And sometimes this method is also called as Greedy Search, as we greedily select the word with maximum probability.

This is where we **stop** the iterations.

So we stop when either of the below two conditions is met:

- We encounter an '**endseq**' token which means the model thinks that this is the end of the caption. (You should now understand the importance of the 'endseq' token)
- We reach a maximum **threshold** of the number of words generated by the model.

If any of the above conditions is met, we break the loop and report the generated caption as the output of the model for the given image.

3.3 Project Scheduling:

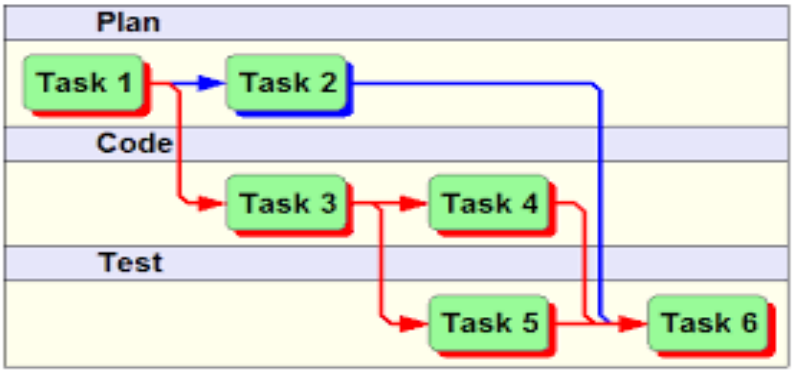


Fig 3.3.1 Pert chart

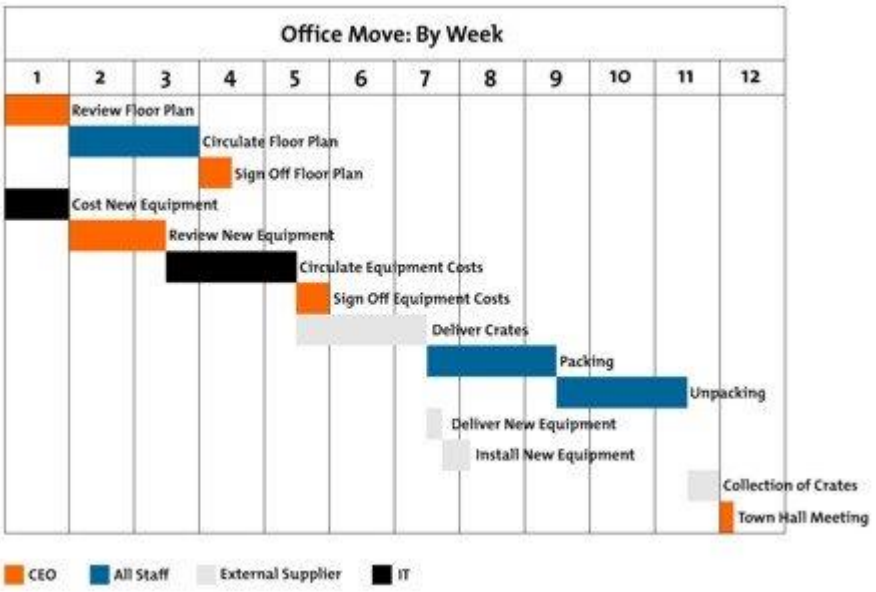


Fig 3.3.2 Gantt Chart

3.4 Testing Techniques and Test Plans:

1) Alpha Testing:

It is the most common type of testing used in the Software industry. The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user.

Alpha Testing is carried out at the end of the software development phase but before the Beta Testing. Still, minor design changes may be made as a result of such testing.

Alpha Testing is conducted at the developer's site. In-house virtual user environment can be created for this type of testing.

#2) Acceptance Testing:

An Acceptance Test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end-user. Client accepts the software only when all the features and functionalities work as expected.

It is the last phase of the testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

#3) Accessibility Testing:

The aim of Accessibility Testing is to determine whether the software or application is accessible for disabled people or not.

Here, disability means deaf, color blind, mentally disabled, blind, old age and other disabled groups. Various checks are performed such as font size for visually disabled, color and contrast for color blindness, etc.

#4) Backward Compatibility Testing:

It is a type of testing which validates whether the newly developed software or updated software works well with the older version of the environment or not.

Backward Compatibility Testing checks whether the new version of the software works properly with file format created by an older version of the software; it also works well with data tables, data files, data structure created by the older version of that software.

If any of the software is updated then it should work well on top of the previous version of that software.

#5) Branch Testing:

It is a type of White box Testing and is carried out during Unit Testing. Branch Testing, the name itself suggests that the code is tested thoroughly by traversing at every branch.

#6) Comparison Testing:

Comparison of a product's strength and weaknesses with its previous versions or other similar products is termed as Comparison Testing.

#7) Compatibility Testing:

It is a testing type in which it validates how software behaves and runs in a different environment, web servers, hardware, and network environment.

Compatibility testing ensures that software can run on a different configuration, different database, different browsers, and their versions. Compatibility testing is performed by the testing team.

#8) Component Testing:

It is mostly performed by developers after the completion of unit testing. Component Testing involves testing of multiple functionalities as a single code and its objective is to identify if any defect exists after connecting those multiple functionalities with each other.

#9) Example Testing:

It means real-time testing. Example Testing includes the real-time scenario, it also involves the scenarios based on the experience of the testers.

#10) Exploratory Testing:

Exploratory Testing is informal testing performed by the testing team. The objective of this testing is to explore the application and looking for defects that exist in the application.

Sometimes it may happen that during this testing major defect discovered can even cause a system failure.

During Exploratory Testing, it is advisable to keep a track of what flow you have tested and what activity you did before the start of the specific flow.

An Exploratory Testing technique is performed without documentation and test cases.

#11) Functional Testing:

This type of testing ignores the internal parts and focuses only on the output to check if it is as per the requirement or not. It is a Black-box type testing geared to the functional requirements of an application. For detailed information about Functional Testing click here.

CHAPTER – 4

RESULTS AND DISCUSSIONS

4.1 Various Modules of the system:

Following are the modules followed to achieve the objective of the project during the project development:

1. Data collection
2. Understanding the data
3. Data Cleaning
4. Loading the training set
5. Data Preprocessing Images
6. Data Preprocessing Captions
7. Data Preparation using Generator Function
8. Word Embedding
9. Model Architecture
10. Caption Generator

4.2 Snapshots of system with brief detail of each:

4.1 Data collection:

Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc. open source datasets available for this problem.

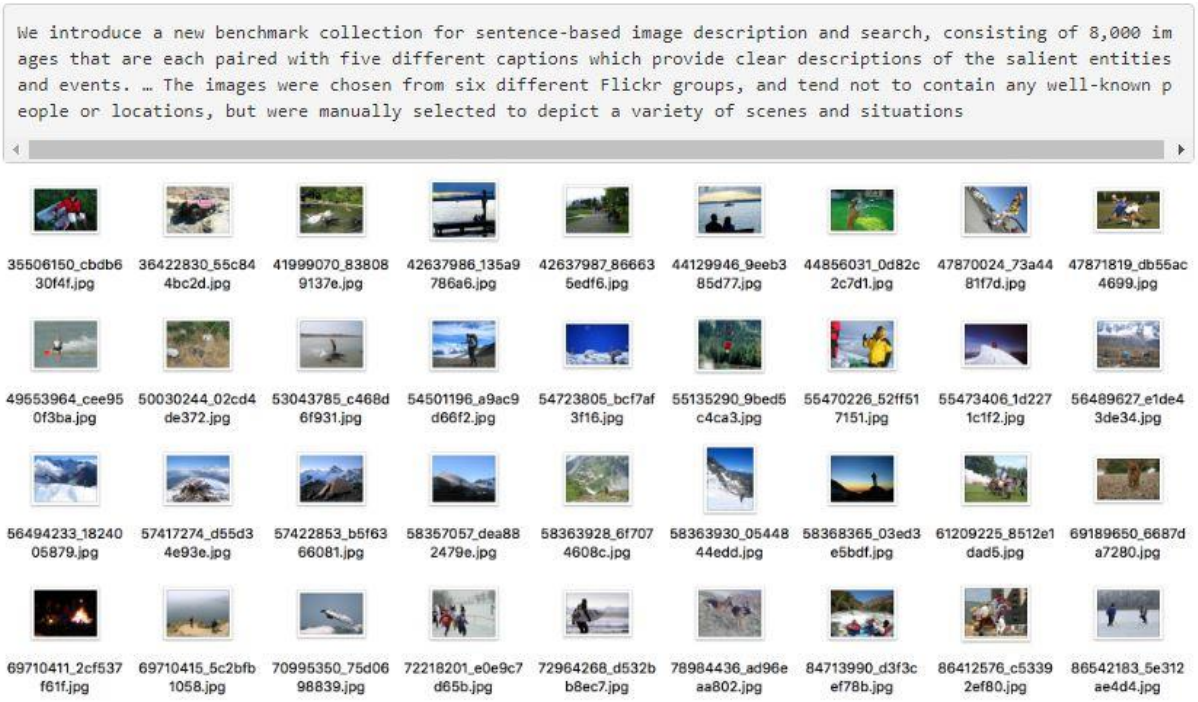


Fig 4.2.1 (8,000 Photos and up to 5 captions for each photo)

4.2 Understanding the data:

```
101654506_8eb26cfb60.jpg#0      A brown and white dog is running through the snow .
101654506_8eb26cfb60.jpg#1      A dog is running in the snow
101654506_8eb26cfb60.jpg#2      A dog running through snow .
101654506_8eb26cfb60.jpg#3      a white and brown dog is running through a snow covered field .
101654506_8eb26cfb60.jpg#4      The white and brown dog is running over the surface of the snow .

1000268201_693b08cb0e.jpg#0     A child in a pink dress is climbing up a set of stairs in an entry
1000268201_693b08cb0e.jpg#1     A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2     A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3     A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4     A little girl in a pink dress going into a wooden cabin .
```

Fig 4.2.2 The text file looks as follows

```
descriptions = dict()
for line in doc.split('\n'):
    # split line by white space
    tokens = line.split()

    # take the first token as image id, the rest as description
    image_id, image_desc = tokens[0], tokens[1:]

    # extract filename from image id
    image_id = image_id.split('.')[0]

    # convert description tokens back to string
    image_desc = ' '.join(image_desc)
    if image_id not in mapping:
        descriptions[image_id] = list()
    descriptions[image_id].append(image_desc)
```

Fig 4.2.3 Creating a dictionary named "descriptions"

4.3 Data Cleaning:

```
# prepare translation table for removing punctuation
table = str.maketrans('', '', string.punctuation)
for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc = desc_list[i]
        # tokenize
        desc = desc.split()
        # convert to lower case
        desc = [word.lower() for word in desc]
        # remove punctuation from each token
        desc = [w.translate(table) for w in desc]
        # remove hanging 's' and 'a'
        desc = [word for word in desc if len(word)>1]
        # remove tokens with numbers in them
        desc = [word for word in desc if word.isalpha()]
        # store as string
        desc_list[i] =  ' '.join(desc)
```

Fig 4.3.1 Creating a vocabulary of all the unique words present

```
# Create a list of all the training captions
all_train_captions = []
for key, val in train_descriptions.items():
    for cap in val:
        all_train_captions.append(cap)

# Consider only words which occur at least 10 times in the corpus
word_count_threshold = 10
word_counts = {}
nsents = 0
for sent in all_train_captions:
    nsents += 1
    for w in sent.split(' '):
        word_counts[w] = word_counts.get(w, 0) + 1

vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]

print('preprocessed words %d ' % len(vocab)))
# preprocessed words 1651
```

Code to

Fig 4.3.2 Retaining only those words which occur at least 10 times in the corpus

4.4 Loading the training set:

```
doc = load_doc('descriptions.txt')
train_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:
        if image_id not in descriptions:
            train_descriptions[image_id] = list()

        # wrap description in tokens
        desc = 'startseq ' + ' '.join(image_desc) + ' endseq'

        # store
        train_descriptions[image_id].append(desc)

        print('Descriptions: train=%d' % len(train_descriptions))
# Descriptions: train=6000
```

Fig 4.4 Loading these names into a list “train”

4.5 Data Preprocessing — Images:

```
# Get the InceptionV3 model trained on imagenet data
model = InceptionV3(weights='imagenet')
# Remove the last layer (output softmax layer) from the inception v3
model_new = Model(model.input, model.layers[-2].output)
```

```
# Convert all the images to size 299x299 as expected by the
# inception v3 model
img = image.load_img(image_path, target_size=(299, 299))
# Convert PIL image to numpy array of 3-dimensions
x = image.img_to_array(img)
# Add one more dimension
x = np.expand_dims(x, axis=0)
# preprocess images using preprocess_input() from inception module
x = preprocess_input(x)
# reshape from (1, 2048) to (2048, )
x = np.reshape(x, x.shape[1])
```

Fig 4.5 Get the corresponding 2048 length feature vector

4.6 Data Preprocessing — Captions

```
ixtoword = {}
wordtoix = {}

ix = 1
for w in vocab:
    wordtoix[w] = ix
    ixtoword[ix] = w
    ix += 1
```

```
# convert 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

# 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)

# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Max Description Length: %d' % max_length)
Max Description Length: 34
```

Fig 4.6 So the maximum length of any caption is 34

4.7 Data Preparation using Generator Function:

	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	[9]	10
2	Image_1	[9, 10]	1
3	Image_1	[9, 10, 1]	2
4	Image_1	[9, 10, 1, 2]	8
5	Image_1	[9, 10, 1, 2, 8]	6
6	Image_1	[9, 10, 1, 2, 8, 6]	4
7	Image_1	[9, 10, 1, 2, 8, 6, 4]	3
8	Image_2	[9]	10
9	Image_2	[9, 10]	12
10	Image_2	[9, 10, 12]	2
11	Image_2	[9, 10, 12, 2]	5
12	Image_2	[9, 10, 12, 2, 5]	11
13	Image_2	[9, 10, 12, 2, 5, 11]	6
14	Image_2	[9, 10, 12, 2, 5, 11, 6]	7
15	Image_2	[9, 10, 12, 2, 5, 11, 6, 7]	3

Fig 4.7.1 Data matrix after replacing the words by their indices

```
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
    X1, X2, y = list(), list(), list()
    n=0
    # loop for ever over images
    while 1:
        for key, desc_list in descriptions.items():
            n+=1
            # retrieve the photo feature
            photo = photos[key+'.jpg']
            for desc in desc_list:
                # encode the sequence
                seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                # 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)
            # yield the batch data
            if n==num_photos_per_batch:
                yield [[array(X1), array(X2)], array(y)]
                X1, X2, y = list(), list(), list()
                n=0
```

Fig 4.7.2 Code for data generator

4.8 Word Embeddings:

```
# Load Glove vectors
glove_dir = 'dataset/glove'
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-
8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

```
embedding_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out
vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in wordtoix.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

Fig 4.8 Creating an embedding matrix

4.9 Model Architecture:

```
# image feature extractor model
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

# partial caption sequence model
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)

# decoder (feed forward) model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

# merge the two input models
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

Fig 4.9 Code to define the Model

4.10 Caption Generator:

```
# prepare translation table for removing punctuation
table = str.maketrans('', '', string.punctuation)
for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc = desc_list[i]
        # tokenize
        desc = desc.split()
        # convert to lower case
        desc = [word.lower() for word in desc]
        # remove punctuation from each token
        desc = [w.translate(table) for w in desc]
        # remove hanging 's' and 'a'
        desc = [word for word in desc if len(word)>1]
        # remove tokens with numbers in them
        desc = [word for word in desc if word.isalpha()]
        # store as string
        desc_list[i] =  ' '.join(desc)
```

```
def greedySearch(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final
```

Fig 4.10.1 Inference with greedy search

Greedy: motorcyclist is riding an orange motorcycle

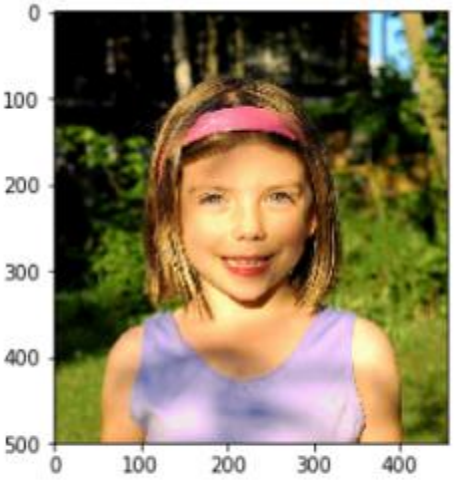
Fig 4.10.2 Output-1

Greedy: white crane with black begins to take flight from the water

Fig 4.10.3 Output-2

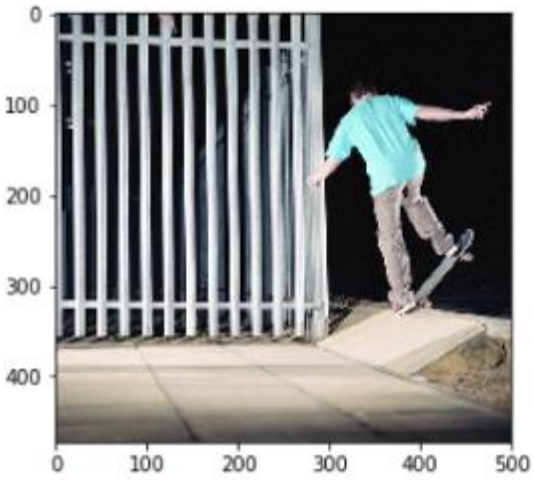
15 Greedy: race car spins down the road as spectators watch

FFig 4.10.4 Output-3



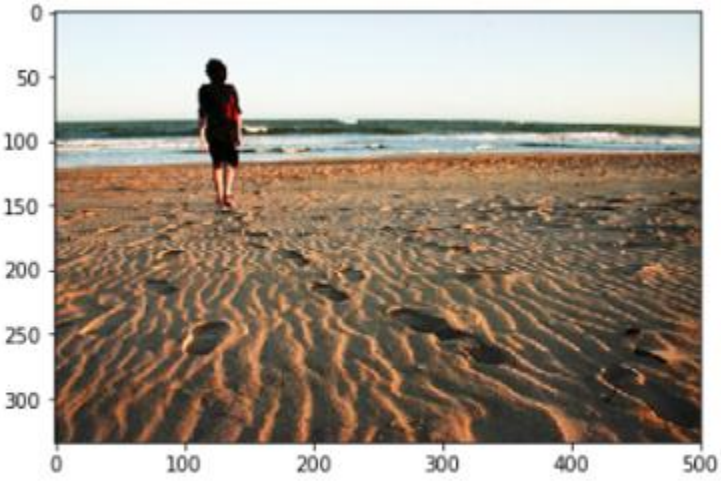
Greedy: girl in pink shirt is smiling whilst standing in front of tree

Fig 4.10.5 Output-4



Greedy: man in black shirt is skateboarding down ramp

Fig 4.10.6 Output-5



Greedy: a boy is walking on the beach with the ocean .

Fig 4.10.7 Output-6

CHAPTER - 5

CONCLUSION & FUTURE SCOPE

5.1 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. The purpose of the model is to maximize the likelihood of the sentence given the image. 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. Furthermore, it will be interesting to see how one can use unsupervised data, both from images alone and text alone, to improve image description approaches.

5.2 Future Scope:

With improvement in the machine learning and deep learning algorithms the current system can be improved to a level that there are negligible number of false positives and true negatives. The existing system can become a base model for more comprehensive and agile framework for a system that can analyses and compare the work done by different chancellors in their respective area.

Future Prospects 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.

REFERENCES

[1] Vinyals, Oriol, et al. "Show and tell: A neural image caption generator." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[3] Fang, Hao, et al. "From captions to visual concepts and back." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

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

[5] Johnson, Justin, Andrej Karpathy, and Li Fei-Fei. "Densecap: Fully convolutional localization networks for dense captioning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[6] Wang, Cheng, et al. "Image captioning with deep bidirectional LSTMs." Proceedings on the 2016 ACM on Multimedia Conference. ACM, 2016.

[7] Nivetha Vijayaraju, et al. "Show, attend and tell: Neural image caption generation with visual attention." International Conference on Machine Learning. 2019.

[8] Papineni, Kishore, et al. "BLEU: a method for automatic evaluation of machine translation." Proceedings of the 40th annual meeting on association for computational linguistics. Association for Computational Linguistics, 2002.