IMAGE CAPTION GENERATOR

A Project Report submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

1215316903 - Akash Dabral

1215316926 - Kati Silas Swarnakanth

1215316935 - Muddurthi Deepak Siddardha

1215316937 - Nimmagadda Sai Ujwal

Under the esteemed guidance of

Ms P. Mansa Devi
Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

VISAKHAPATNAM

APRIL 2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

We, hereby declare that the project work entitled "**IMAGE CAPTION GENERATOR**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfilment of the requirements for the award of the degree of Bachelors of Technology in Computer Science and Engineering.

The work has not been submitted to any other college or university for the award of any degree or diploma.

Date: 20th April 2020

Registration No.	Name	Signature
1215316903	Akash Dabral	
1215316926	Kati Silas Swarnakanth	
1215316935	Muddurthi Deepak Siddardha	
1215316937	Nimmagadda Sai Ujwal	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled **“IMAGE CAPTION GENERATOR”** is a bonafide record of work carried out by **1215316903 - AKASH DABRAL, 1215316926 - KATI SILAS SWARNAKANTH, 1215316935 - MUDDURTHI DEEPAK SIDDARDHA, 1215316937 - NIMMAGADDA SAI UJWAL** students submitted in partial fulfilment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	Head of the Department
Ms. P. Mansa Devi	Dr. K Thammi Reddy
Assistant Professor	Professor
Dept. of CSE	Dept. of CSE

ACKNOWLEDGEMENT

We express our deepest gratitude and special thanks to **Ms. P. Mansa Devi**, Assistant Professor in Dept. of CSE, who in spite of being extraordinarily busy with her duties, took time out to hear, guide and keep us on the correct path during the whole project.

We express our sincerest thanks to **Mr. P. N. R. L. Chandra Sekhar**, Associate Professor in Dept. of CSE, and to **Mr. V. Praneel Ankireddy**, Assistant Professor in Dept. of CSE, for taking part in useful discussions along with giving us necessary advice and guidance. We choose this moment to acknowledge their contribution to the completion of our project.

We would also like to thank our AMC., **Mrs. B. Soujanya**, Assistant Professor in Dept. of CSE, and our HoD, **Mr. Konala Thammi Reddy**, Dept. of CSE, who gave us this opportunity to do a project and helped us in the successful completion of our report.

TABLE OF CONTENTS

ABSTRACT

In artificial intelligence, describing the meaning of a photo or an image automatically is a fundamental issue which links computer vision and natural language processing. In our project, we have tried to implement a reproductive model which is designed on a profound, recurrent framework that merges new developments in machine translation and computer vision which can be utilised to produce natural phrases to describe the image in consideration.

Among all the many challenging artificial intelligence problems, generating captions of an image is also not an easy task. A textual description for a photograph given as input has to be generated which should correctly describe it. A paper released by a few Google researchers named “Show and Tell: A Neural Image Caption Generator” in 2014 talks about the same problem which we used as a reference and implemented our project by combining CNN with LSTM to take a photo as input and output a caption as described in the paper.

Both the methods, computer vision to comprehend the content of the image and a linguistic model from the natural language processing domain, are required to generate words in the correct order from the image’s understanding. Progress in deep learning methods recently has achieved state-of-the-art results for models similar to ours.

In our project, we take inspiration from our reference paper where they have discussed about an end-to-end model where the first part consists of a vision CNN (we used Xception CNN) and the second part consists of a language generation RNN (we used LSTM). The model tries to generate correct descriptions of the image given as input. We have trained our model where the probability of generating a description sentence of the given training image is maximised. Our experiments carried on self-made and Flickr8k datasets demonstrate the ability of our model to generate near accurate descriptions of our images in a fluent language learnt exclusively from our training data.

1. INTRODUCTION

Among several problems in artificial intelligence, caption generation is amongst the challenging ones where a textual explanation for a given image must be produced. It is a tough job to be able to explain the meaning of an image automatically using properly constructed English sentences. With so much visual data present out there in the digital world, it is also useful if we are able to lable those automatical without human need. This will also help in segregating the information while also being helpful later in the future for many other machine learning or artificial intelligence tasks.

When we compare to this to the object recognition or image classification tasks which have been well-studied by the computer vision community, the task is remarkably harder. In reality, the description generated should not only merely contain the objects detected in the image but also lucidly relate various objects to describe the activities and characteristics involved in the image in an articulate manner. Moreover, the model must be able to generate a description in a natural language like English, indicating that a linguistic model is required in addition to visual comprehension.

Many past experiments consist of combining parts of the above-discussed problems to generate descriptions for an image. But here we will be making a single model joining both the computer vision and natural language processing to produce a target sequence for our image. Our work is inspired by new developments in machine translation where a source language sentence S is transformed target language T. For years together, successions of detached tasks were implemented to accomplish machine translation, but presently can be made possible in a much easier way using RNNs.

As discussed in the paper we are referring, the model we would be building would require both the methods, one from computer vision to comprehend the content of the image and another a linguistic model from the natural language processing domain to create a combination of words in correct order describing the image. What is most remarkable about these approaches is that it is possible to create a single combined model to generate descriptions for an image, rather than explicitly constructed or sophisticated models.

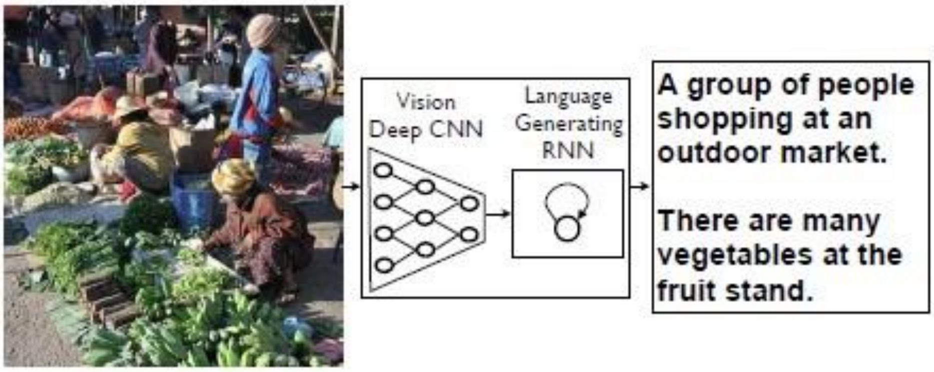

Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

Fig. 1: Framework of our project

For our implementation of the model described in the paper, we combine two models, the CNN and the LSTM, into one (shown in Fig. 1). We researched various CNNs to find which one would perform best for the given problem. According to the research paper “Xception: Deep Learning with Depthwise Separable Convolutions [24]” by a Google researcher, we came to know that the best CNN to use for our project would be the Xception CNN.

2. LITERATURE SURVEY

In artificial intelligence, describing the meaning of a photo or an image automatically is a common research area which links computer vision and natural language processing. The perception of images requires to identify objects and recognise them. It also has to consider the nature or position of the scene, the properties of the artefacts and their interactions. For a well-formed sentence, both syntactic and semantic language comprehension is needed. There are several methods used for this purpose, but we can broadly classify them into two categories: (1) Traditional machine learning-based techniques and (2) Deep machine learning-based techniques.

In traditional machine learning methods, handcrafted features like Local Binary Patterns (LBP) [1], Scale-Invariant Feature Transform (SIFT) [2], the Histogram of Oriented Gradients (HOG) [3], and a blend of similar features are commonly used. Features are mined from data and are given as input in these techniques. With the aim to classify an object, these features are then forwarded to a classifier such as Support Vector Machines (SVM) [4]. Extraction of features from a large dataset is not feasible in such methods since handcrafted features are task-specific. Moreover, in real-world, videos and images related data is very complex and can have different interpretations semantically.

Conversely, in deep machine learning-based techniques, features are trained automatically from the given training data where such models can handle a complex range of variety of videos and images related data. Let us take Convolutional Neural Networks (CNN) [5], for example, where they are commonly used for the learning of features, and a classifier like Softmax where it used for classification tasks. In our case, Recurrent Neural Networks (RNN) is usually preceded by CNN for the production of captions.

For designing a predictive modelling problem, it is essential to consider and test multiple ways. In our case, we indeed have several ways to view and analyse the problem of generation of photographic captions.

2.1 VARIOUS CAPTION GENERATION MODELS

For our caption generation problem, we have mainly three models which are mentioned below:

1. A model which generates the whole sequence
2. A model which generates words from other words
3. A model which generates words from a sequence

2.1.1 MODEL WHICH GENERATES THE WHOLE SEQUENCE

The model here is a one-to-many sequence prediction model. Given an input photo to this model, it generates the entire description for it in a single-shot manner.

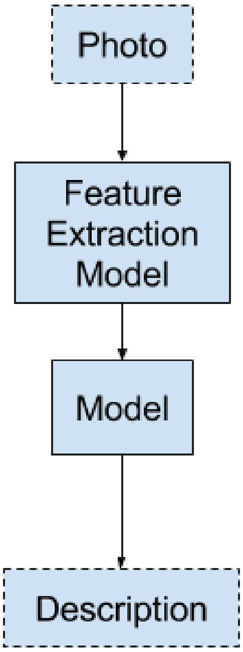


Fig. 2: Generating textual sequence from photo

In this method, we heavily rely on the linguistic model to create the most accurate description for our image by generating the words in exact order. We pass the photo to a model which extracts features such as a pre-trained model on the dataset from ImageNet. For the sequence produced in the output, a one-hot encoding is used, which allows for predicting the probability distribution over every word of the sequence. Each series is padded to be the same length. That means, in the output sequence, the model is obligated to generate multiple “no word” time steps.

2.1.2 MODEL WHICH GENERATES WORDS FROM OTHER WORDS

The model here implements a one-to-one sequence prediction. Given an input photo and a previously generated word (or start of sequence token) to this model, the next word is recursively generated in the sequence. In simple words, given an image and a word as input, the LSTM produces a prediction for one word.

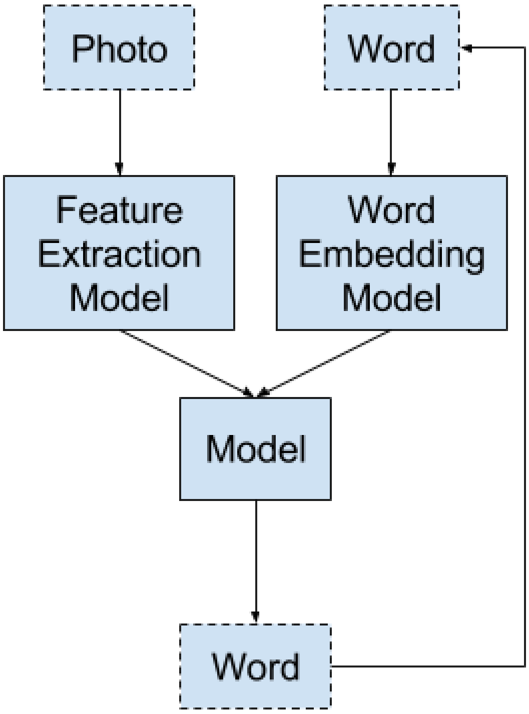


Fig. 3: Generating sequence's next word from the previous word

While calling the model, the word given as input is sometimes a sequence start word (if the model is called for the first time) or a word that was created from the last period the model was called based on the situation. Again, similar to the previous model, we pass the photo to a model which extracts features such as a pre-trained model on the dataset from ImageNet. Here, we encode the input word into an integer and pass it through an embedding script. For allowing the model to guess the word probabilities, one-hot encoding is used at the output. In this model, we recursively call the same model until a sequence end token is produced.

2.1.3 MODEL WHICH GENERATES WORDS FROM A SEQUENCE

The model here implements a many-to-one sequence prediction technique. Given an input photo and a sequence of words that were already created to this model, it attempts to guess the next word in the definition. The end result is a textual description generated through recursively making calls to the model.

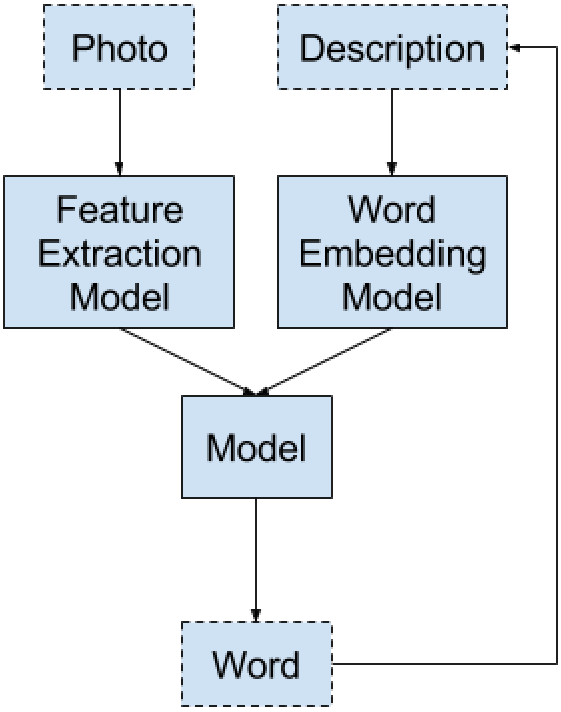


Fig. 4: Generating next word from already generated sequence

This model is very similar to the above model where instead of a single word given as input, we give here a sequence of words to the algorithm to produce the next word. Here too, we pass the photo to a model which extracts features such as a pre-trained model on the dataset from ImageNet. It might be preferable to pass the photo only once in the beginning than sending it at every cycle. Each series is padded to be the same length. Here also, we encode the input word as an integer and pass it through an embedding script. For the model to predict word probabilities, the output word is one-hot encoded. In this model too, we recursively call the same model until a sequence end token is produced.

.

3. PROBLEM STATEMENT

In our fast-growing digital world, we come across numerous photographs and digital images on various platforms from social media sites to advertisements to the internet. These images are usually understandable by humans, even if they do not come with any descriptions or detailed captions. This ability of humans can be induced into the machine and could be put to use for many purposes. For example, it may be conducive for the people who are impaired visually to “see” the world. We can also use it for automatic image indexing as it is vital for Content-Based Image Retrieval (CBIR) systems which can be further extended to many other fields like military, biomedicine, web search and education. Also, we could use social networking sites like Twitter and Facebook to create picture descriptions straight away. The descriptions could include where we are, what we wear, and importantly what we are doing.

However, for all this to work, if humans have a requirement of automatically produced image captions from the machine, the machine needs some sort of image descriptions to be understood. Hence, with the work described in our referred paper [25], we designed a model which will help us do such tasks od automating the generation of image captions.

3.1 MODEL

Most of the earlier work is typically a combination of various existing methods of the problems we have discussed [6, 16]. On the contrary, the model discussed in the reference paper [25] has a somewhat different approach. They have proposed a unique solo joint model which, as they have mentioned, works like – “the model takes input an image *I* and is trained to maximise the likelihood $p(S|I)$ of generating a target arrangement of words $S = \{S1, S2, \ldots\}$ where every individual word S_t comes from a given dictionary, that describes the image adequately [25]”.

Since, over the past few years, it has been realistically demonstrated what CNNs are capable of. They can elegantly generate a profuse depiction of the given image and embed it into a vector of fixed length. This fixed-length vector can then be utilised for a diversity of vision-related tasks [21]. Hence, the researchers have proposed a model, where they have swapped the RNN encoder with a CNN. The CNN is first pre-trained for the task of classifying the images and then for the RNN decoder, the final hidden layer of the CNN is given as input for the generation of sentences (see Fig. 5).

To produce the descriptions of the images, the research paper [25] has also talked about a neural and probabilistic structure. Current developments in statistical machine translation prove that best in class results can be achieved in robust sequential models by optimising the likelihood of correct translation in an “end-to-end” manner. These models use an RNN that encodes a variable-length input into a static dimension vector and uses this illustration to decode it to our desired word output. The researchers, in their research paper [25], have used the same method in which, a picture is given as input (as a replacement to giving a source language input sentence), the same concept of translating an image into its definition is applied. They are able to do this by taking an input image and maximising the probability of the right description using the formula:

$$\theta^\star = \arg\max_\theta \sum_{(I,S)} \log p(S|I;\theta) \qquad (1)$$

where θ are the model specifications described in the research paper, I is an image and S is its correct description as mentioned. The length of S is unrestrained since it characterises any sentence. Therefore, for the model in the discussion, the researchers implemented the chain rule to calculate the cumulative probability over $S_0, \ldots, S_N$, where the length of the example in the discussion is N as is shown below:

$$\log p(S|I) = \sum_{t=0}^{N} \log p(S_t|I, S_0, \ldots, S_{t-1}) \qquad (2)$$

For the convenience of the project, reliance on θ has been left out. *(S, I)* depicts an example training pair at the training stage. And using stochastic gradient descent technique, the researchers have optimised the sum of log probabilities on the whole training set, as shown in (2). $p(S_t \mid I, S_0, \ldots, S_{t-1})$ can be customarily modelled using an RNN, where words of variable length till $t - 1$ represented by a hidden state of fixed-length and is represented by a memory h_t. After looking at a new input x_t, this memory h_t is updated using a non-linear function f as follows:

$$h_{t+1} = f(h_t, x_t)\ . \qquad (3)$$

The researchers made two crucial design adoptions to make the above-mentioned RNN more reliable and robust. They had to figure out the true form of f and in what way would be the words and photos given as inputs x_t. The researchers came up with a solution to use an LSTM network for f and a CNN for image representation since they have been commonly implemented in object detection and recognition tasks in computer vision.

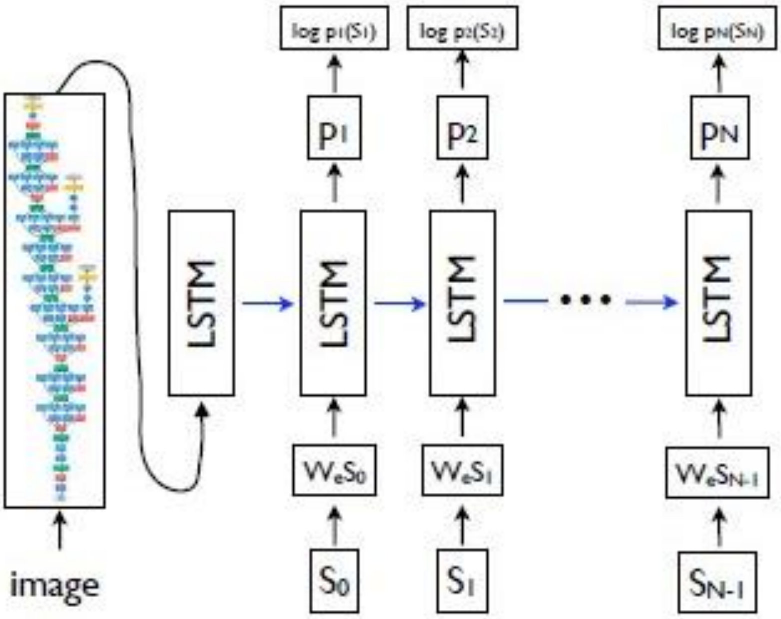


Fig. 5: Overview of the model

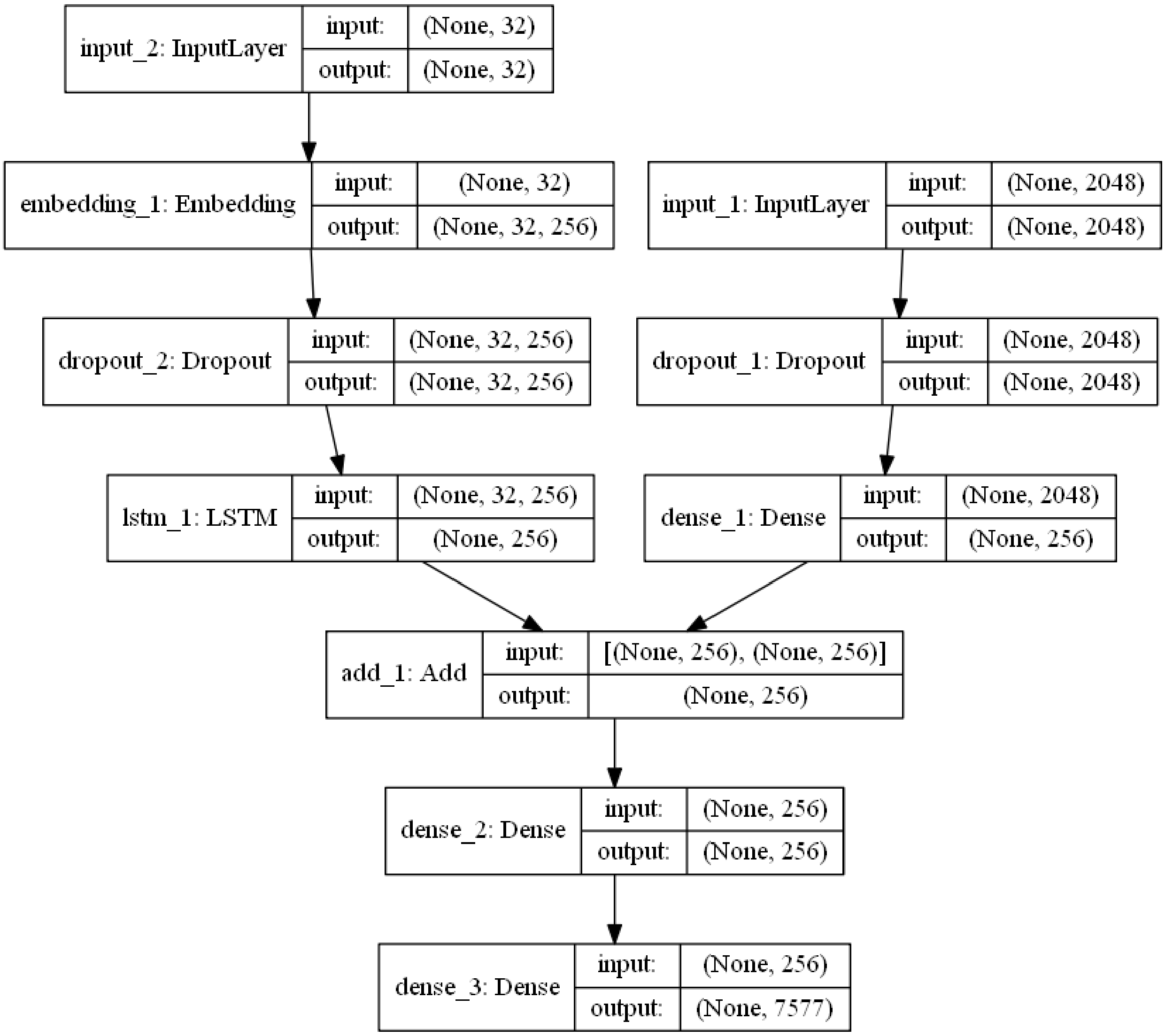


Fig. 6: Model structure

3.1.1 XCEPTION CNN

Even though our referred paper [25] does not mention which CNN to use, we decided to go with the Xception CNN as it performed with higher accuracy of various image datasets as compared to other CNNs as described in the research paper “Xception: Deep Learning with Depthwise Separable Convolutions [24]”.

Xception CNN is based on the Inception v3 architecture and inherits many properties of it.

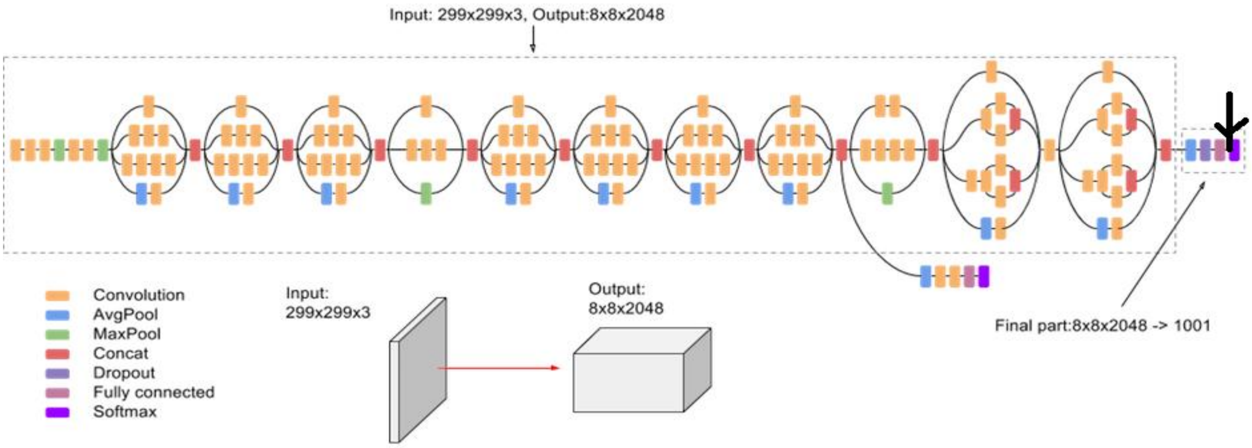


Fig. 7: Model structure of Xception CNN

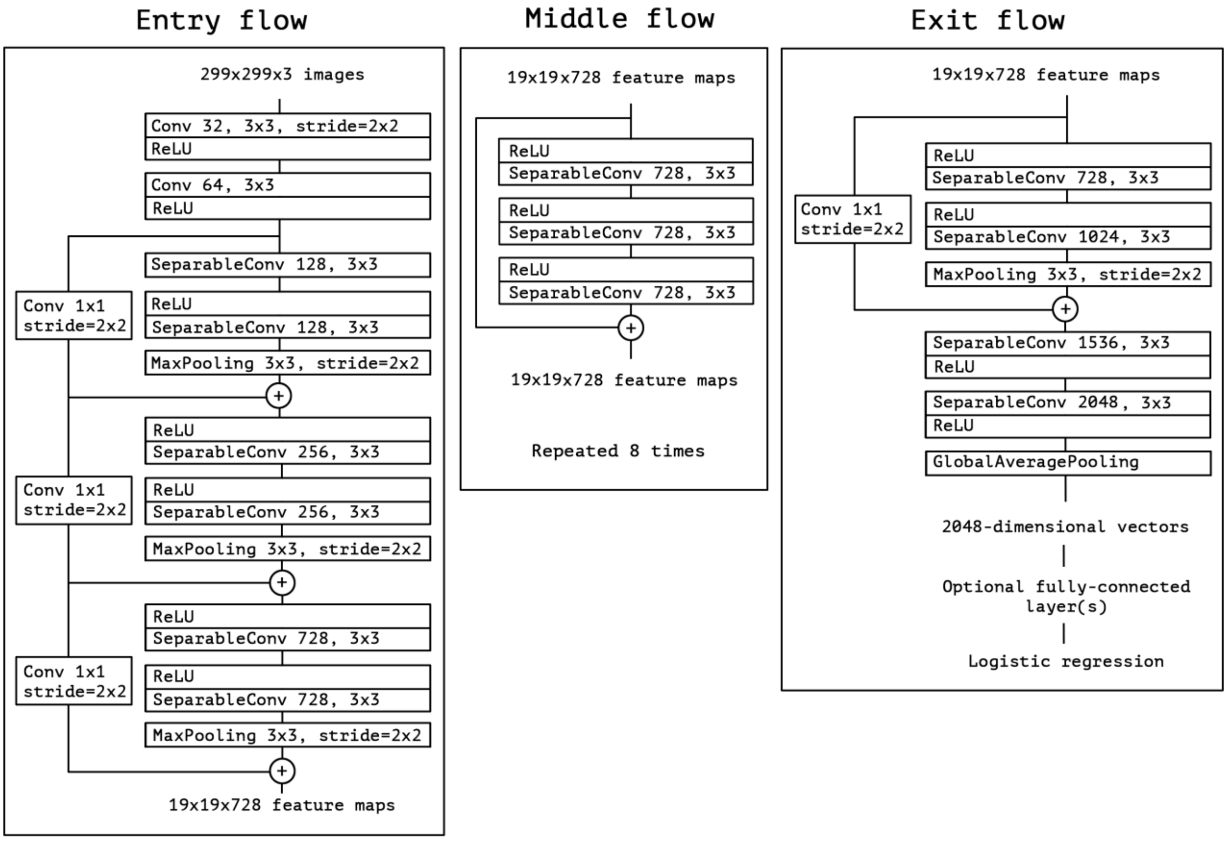


Fig. 8: Xception architecture

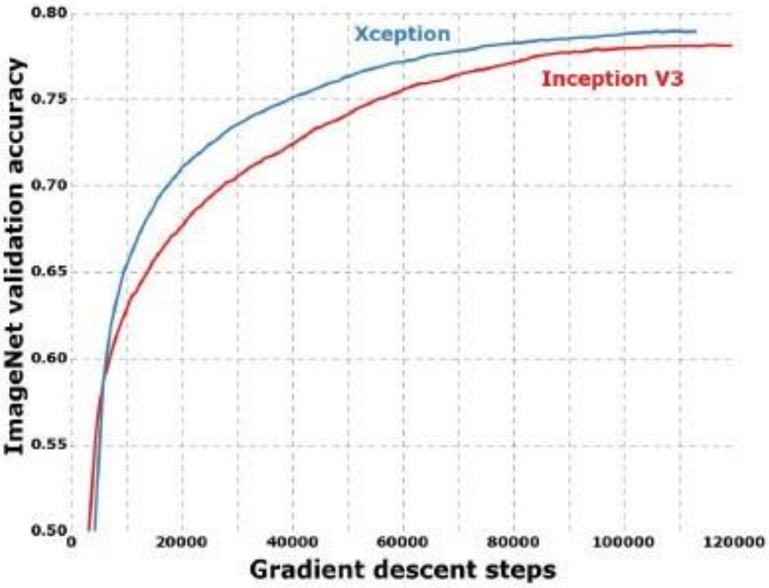


Fig. 9: Comparision of Inception V3 and Xception on ImageNet Dataset as in paper [24]

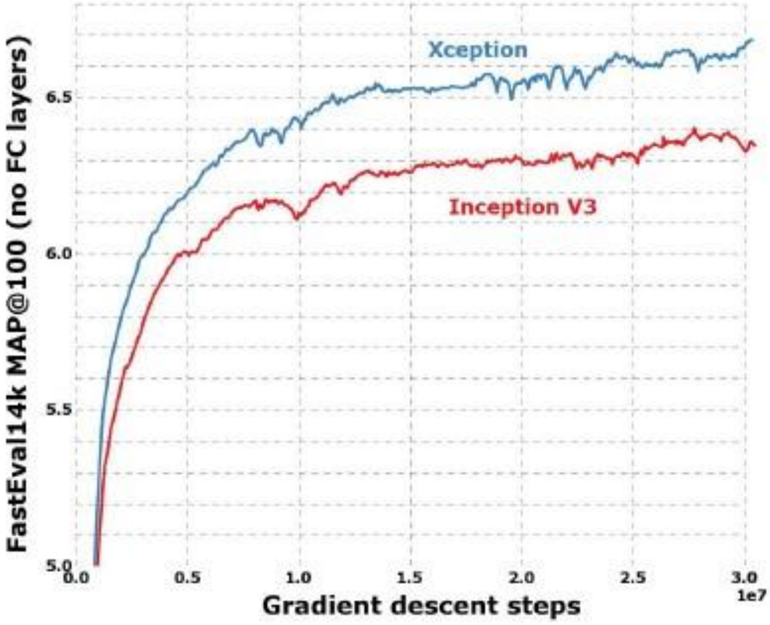


Fig. 10: Comparision of Inception V3 and Xception on JFT Dataset as in paper [24]

	Steps/sec	Param count
Xception	28	2,28,55,952
Inception V3	31	2,36,26,728

Table 1: Comparision between the speed of training and size as in paper [24]

In the research paper [24], the Xception CNN model is designed in a manner where the feature extraction base for the model consists of 36 convolutional layers which are organised into 14 modules. Except for the initial and latter modules, all the layers consist of linear residual contacts around them. The flow of data in the architecture (Fig. 8) is that it initially enters the entry flow, after which it repeats itself eight times in the middle flow of the architecture, only to later come out of the exit flow.

	Top – 1 accuracy	**Top – 5 accuracy**
VGG - 16	0.715	0.901
ResNet - 152	0.770	0.933
Inception v3	0.782	0.941
Xception	**0.790**	**0.945**

Table 2: Xception CNN performance on ImageNet dataset when compared to other CNNs [24]

3.1.2 LSTM

Among the most typical challenges while designing an RNN is the exploding and vanishing gradients [10] problem. For our model, our choice of *f* in (3) hugely depends on how well it handles this problem. Hence, we chose a particular RNN called LSTM for our model to tackle this problem. Other studies have introduced [10] showed how LSTM works successfully when applied to translation [6, 22] and sequence generation [9].

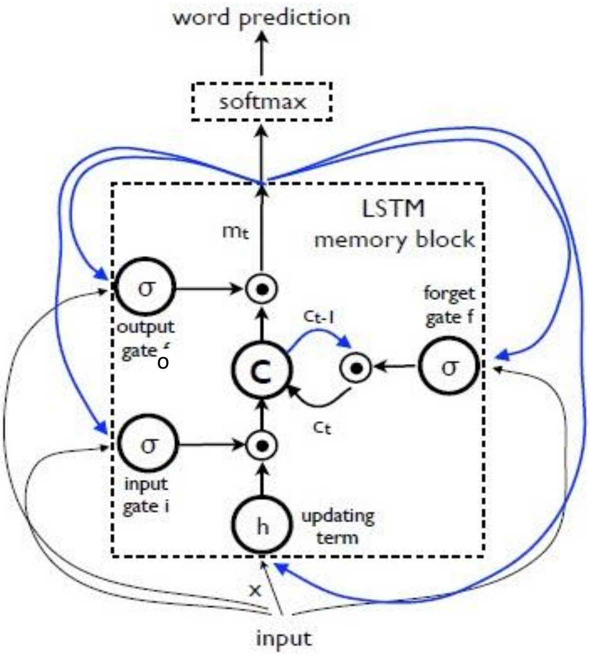


Fig. 11: LSTM memory block

In the principal of the LSTM model lies a memory cell *c* which is shown in Fig. 11. At each time step, the memory cell *c* keeps encoding the knowledge of the input words perceived until this step. The memory block comprises of three gates in particular; the input gate *I* which determines whether to read the input or not; the output gate *o* which determines whether the newly calculated cell value should be output or not; and lastly, a forget gate *f* which determines whether or not to forget the value of the current cell.

The lines coloured in blue are repeated connections. At time *t*, three inputs are given to the LSTM; the output *m* from *t – 1* time is given to the forgotten gate, input gate and output gate. Additionally, the cell value is given to the forgotten gate. For word prediction, Softmax is passed an expected word from *t – 1* time along with the time *t*'s memory output *m*.

Since the layers are applied multiplicatively, we could either delete the memory cell *c*'s data if the gate value is 0 or keep the data in the memory cell for if the gate value is 1; as the whole behaviour of the memory cell is controlled only by the gates. As per the research paper [25], the workings of the gates and the update of the cells, as well as its output, are as follows:

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1}) && (4)\\
f_t &= \sigma(W_{fx}x_t + W_{fm}m_{t-1}) && (5)\\
o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1}) && (6)\\
c_t &= f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}) && (7)\\
m_t &= o_t \odot c_t && (8)\\
p_{t+1} &= \text{Softmax}(m_t) && (9)
\end{aligned}
$$

Where *W* denotes the various matrices which are trained parameters and $\odot$ characterises the gate value product. Since these multiplicative gates help tackle exploding and vanishing gradients [10] problem, they make our LSTM robust. Only the hyperbolic tangent *h(·)* and sigmoid *σ(·)* are non-linear. The Softmax function is passed the value of end equation m_t which then produces a probability distribution p_t overall words.

The LSTM model, as described in the research paper, is a model following many-to-one sequence prediction. When given an input photo and a sequence of previous words described by $p(St|I, S_0, . . . , S_{t-1})$ that were already created by the model, it attempts to guess the succeeding word in the definition. Think of it as if a virtual copy of the model is created for the input image and every word of the sentence. As depicted in Fig. 5, one can see that the output m_{t-1} of the previous copy of the LSTM at the time $t - 1$ is given to the present copy LSTM at the time *t*. Here, all the LSTM share the same constraints. The entire recurrent connections present in our model are converted into unrolled versions of feed-forward connections. So, the unrolling process for our input image *I* and a corresponding sentence *S = (S0, . . . , SN)* describing the image will look like this:

$$
\begin{aligned}
x_{-1} &= \text{CNN}(I) && (10)\\
x_t &= W_e S_t, \quad t \in \{0 \ldots N-1\} && (11)\\
p_{t+1} &= \text{LSTM}(x_t), \quad t \in \{0 \ldots N-1\} && (12)
\end{aligned}
$$

Every word here is represented by a one-hot dimension S_t vector, which equals the size of the dictionary. The researchers, in their research paper [25], have assigned the beginning of the sentence with a start word S_0 and have ended the sentence with a stop word S_N. So, in that way, the LSTM can signal that the generation of the sequence of words has been completed by producing the stop word. In this model, mapping of both the words and the images are done in the same space; CNN is employed for mapping the image and the word embedding W_e is used to map the words.

The researchers found it beneficial to give the image *I* as input at $t = -1$, for the LSTM to know the contents of the image, only once because they found that, at every step, continuously giving the image *I* as input produced diminishing results. This also introduced noise into the network and easily overfitted it.

3.2 INPUT

For our project, we give an image as input to the model.

3.3 OUTPUT

The output is the image as well as the caption of the image that has been generated.

4. OVERVIEW OF TECHNOLOGIES

4.1 PROGRAMMING LANGUAGES

Python 3.7.3 64 bit (AMD Processor), Anaconda Installation

4.2 TOOLS AND ENVIRONMENTS

Anaconda (Version 3.17.8, Build: py37_0)

Spyder (Version: 3.3.4, Build: py37_0)

Jupyter (Version: 1.0.0, Build: py37_0)

Google Colab

5. IMPLEMENTATION

5.1 DATASETS

In the research paper [25], these were the following datasets that they mentioned.

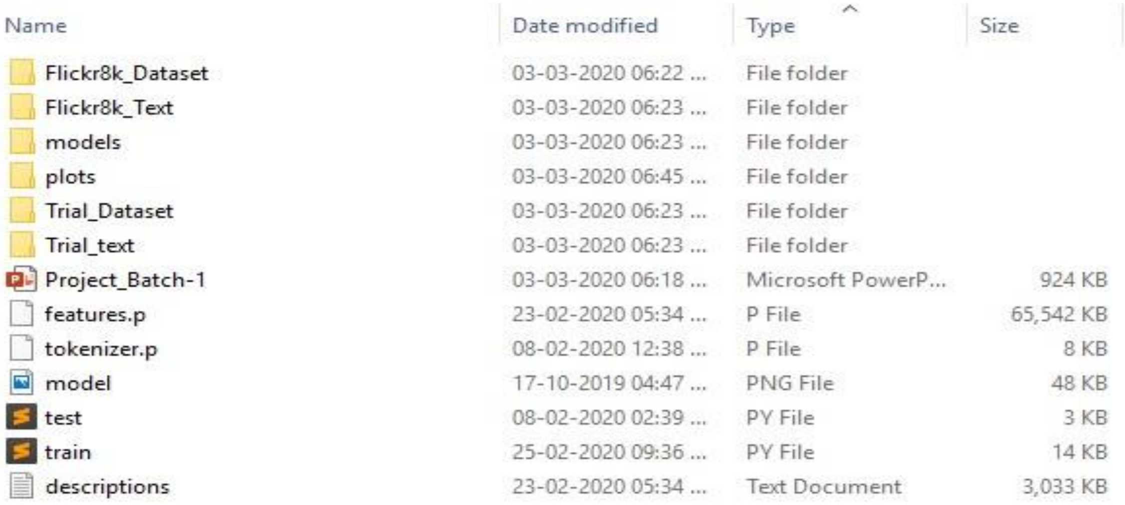
Dataset	Size		
	Train	Valid.	Test
Pascal VOC 2008 [6]	-	-	1000
Flickr8k [19]	6000	1000	1000
Flickr30k [22]	28000	1000	1000
MSCOCO [17]	82783	40504	40775
SBU [18]	1M	-	-

Table 3: Statistics of the datasets mentioned in the research paper [25]

Each of these datasets consists of many images where each image has a set of sentences describing the image. For evaluation purposes, we have used the Flickr8k dataset in our project—this dataset as 8000 images with every image having five descriptions each. We also created a small dataset for trial purposes consisting of 15 images.

5.2 PROJECT STRUCTURE

Our project workspace (Fig. 10) consists of the datasets, python code for the training of model and prediction.



Name	Date modified	Type	Size
Flickr8k_Dataset	03-03-2020 06:22 ...	File folder	
Flickr8k_Text	03-03-2020 06:23 ...	File folder	
models	03-03-2020 06:23 ...	File folder	
plots	03-03-2020 06:45 ...	File folder	
Trial_Dataset	03-03-2020 06:23 ...	File folder	
Trial_text	03-03-2020 06:23 ...	File folder	
Project_Batch-1	03-03-2020 06:18 ...	Microsoft PowerP...	924 KB
features.p	23-02-2020 05:34 ...	P File	65,542 KB
tokenizer.p	08-02-2020 12:38 ...	P File	8 KB
model	17-10-2019 04:47 ...	PNG File	48 KB
test	08-02-2020 02:39 ...	PY File	3 KB
train	25-02-2020 09:36 ...	PY File	14 KB
descriptions	23-02-2020 05:34 ...	Text Document	3,033 KB

Fig. 12: Complete project directory

Flickr8k_Dataset: This folder consists of all the 8000 images.

Flickr8k_Text: This folder consists of all the descriptions and the image segregation into various train, validation and test data.

models: This folder comprises of all the models that are trained. Each model is saved here in this folder.

plots: In this folder, the graphs related to the training of our model (Accuracy and Loss graphs) are stored.

Trial_Dataset: This folder consists of 15 randomly searched images from the net.

Trail_text: This folder consists of all the self-made descriptions and the image segregation into various train, validation and test data.

features.p: This is a file generated after we pass our dataset through the Xception CNN and extract all the features related to our dataset.

tokeniser.p: This is a file created when we collect all the unique words from the descriptions and the possible relations between them.

model.png: This is our model structure.

test.py: This is the code for testing our model.

train.py: This is the code for training our model.

descriptions.txt: This file is created after we pre-process our descriptions. We remove any special characters, numbers, punctuations and convert all the descriptions to lower-case.

5.3 CODING

5.3.1 TRAINING

This is the project’s most significant code file. The code to training our model is in the file train.py. The code flow for the training phase of our model is as follows:

1. Import all the required files and packages.
2. Read all descriptions and pre-process them by removing any special characters, numbers, punctuations and converting all the descriptions to lower-case and then save it.
3. Extract features using Xception CNN from all the dataset images and saving that file.
4. Segregate the train data and generate a tokeniser from the train image descriptions.
5. Calculate the maximum length of all the descriptions.
6. Create the LSTM model.
7. Train the LSTM model using our training images and descriptions.
8. Save the model.
9. Plot the training graphs (Accuracy and Loss) for the trained model and save them.

5.3.2 TESTING

The code for testing our model is in the file test.py. The code flow for the testing phase of our model is as follows:

1. Import all the required files and packages.
2. Open the image given as input.
3. Extract features from that image using the Xception CNN.
4. Load the created LSTM model earlier.
5. Pass the image and its features along with the tokeniser to the LSTM model.
6. Print the description.

5.4 TRAINING PHASE

We initially began with training our model with a self-created dataset of 10 images. This dataset was created by searching random images from the internet and all of our team members describing five captions for each image. However, during this, we faced many problems like overfitting of the data. Clearly, the dataset that we had created was minimal compared to our problem as gigantic volumes of data are required for purely supervised approach machine learning techniques.

Even though the task of generating image descriptions is much more laborious the traditional object recognition problem, hefty datasets like ImageNet have made it possible to do so.

Nonetheless, various strategies were tried by us to cope with the overfitting of data. In one of the techniques, we used the weights of a pre-trained model on ImageNet to initialise the weights of the Xception CNN we used. This helped a lot in generalising the data. We also did some overfitting-avoiding techniques at the model level. We tried ensembling models and dropout techniques [34]. We also tried and tweaked the model's size while trading on the hidden units of the model with its depth. We also achieved few results by making changes to the ensembling and the dropout.

After getting pleased with the results and working of our model with the dataset we created, we ran it on the Flickr8k dataset. Due to limited computing power, we could only train our model with 1000 images rather than the complete dataset.

With the pre-trained weights of the ImageNet dataset for our CNN, by means of stochastic gradient descent algorithm, we again trained all the weights of our model with no momentum and at a fixed learning rate. Except for our CNN weights, all the other weights we randomly initialised. We used a 256 dimension memory as size for our LSTM and the embedding. We also pre-processed the descriptions with the necessary tokenisation where we preserved all the words that appeared in training set for at least five times.

5.4.1 TRAINING ON SMALL DATASET

Resources:

Laptop – HP Pavillion 15 (Model: hp 15ac120tx)

Processor: Intel i3 5005U – 2.00GHz

RAM: 16GB

Libraries:

TensorFlow: 2.0.0

Keras: 2.2.8

Dataset size

Total: 15

Train: 10

Test: 5

Training Time

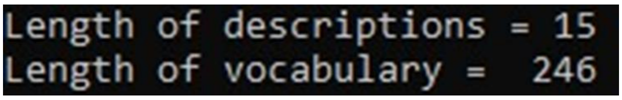
Time: 150 sec

Training statistics:

Epochs - 50

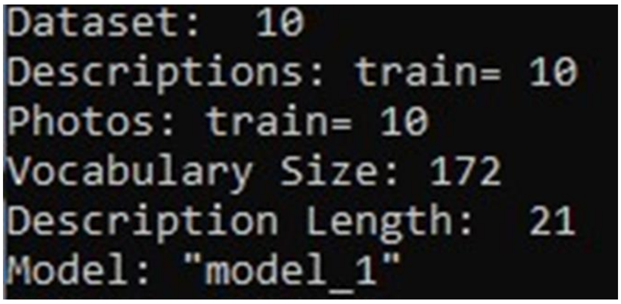
Accuracy - ~90%

Loss - ~0.35



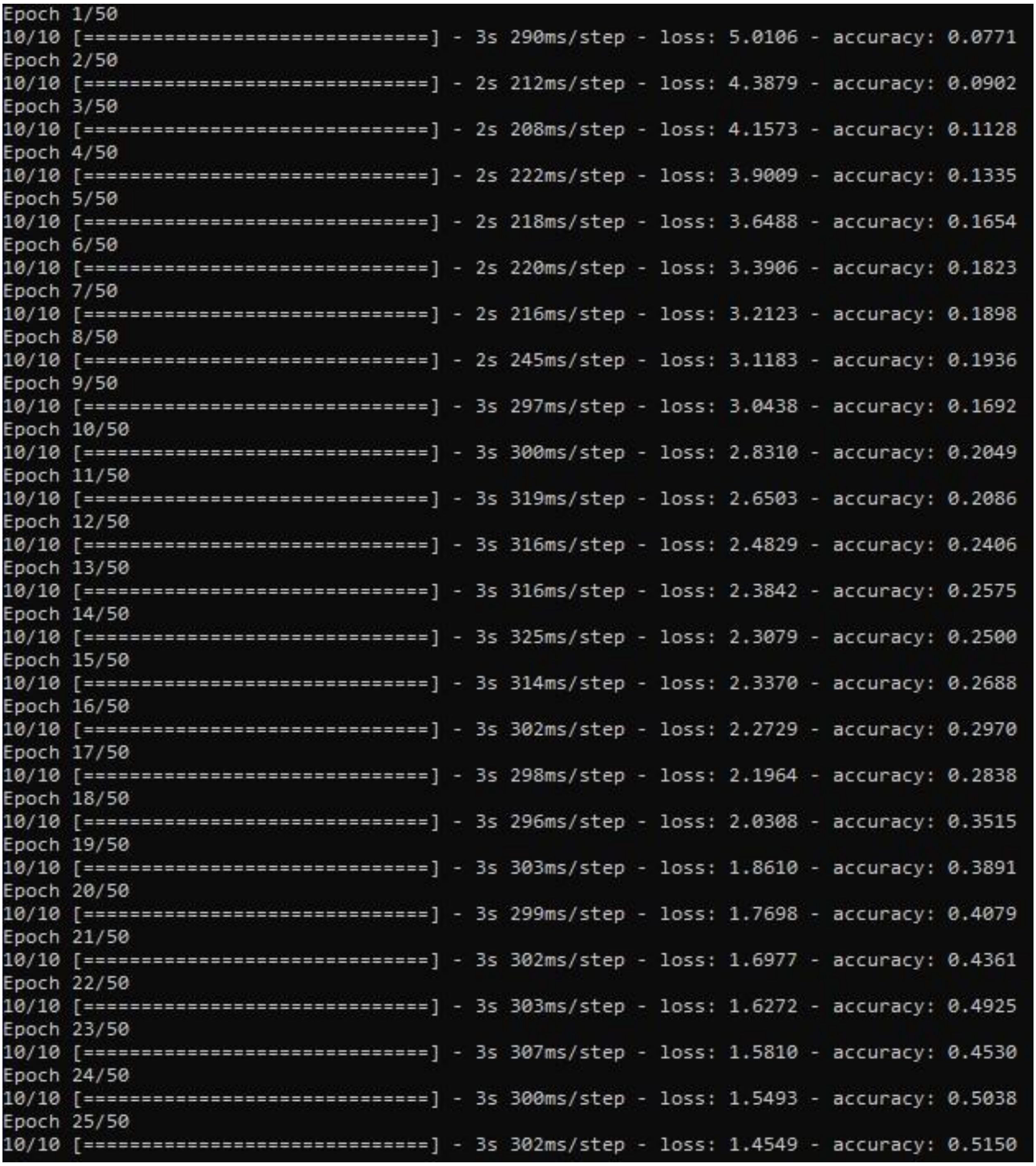
```
Length of descriptions = 15
Length of vocabulary =  246
```

Fig. 13: Total descriptions



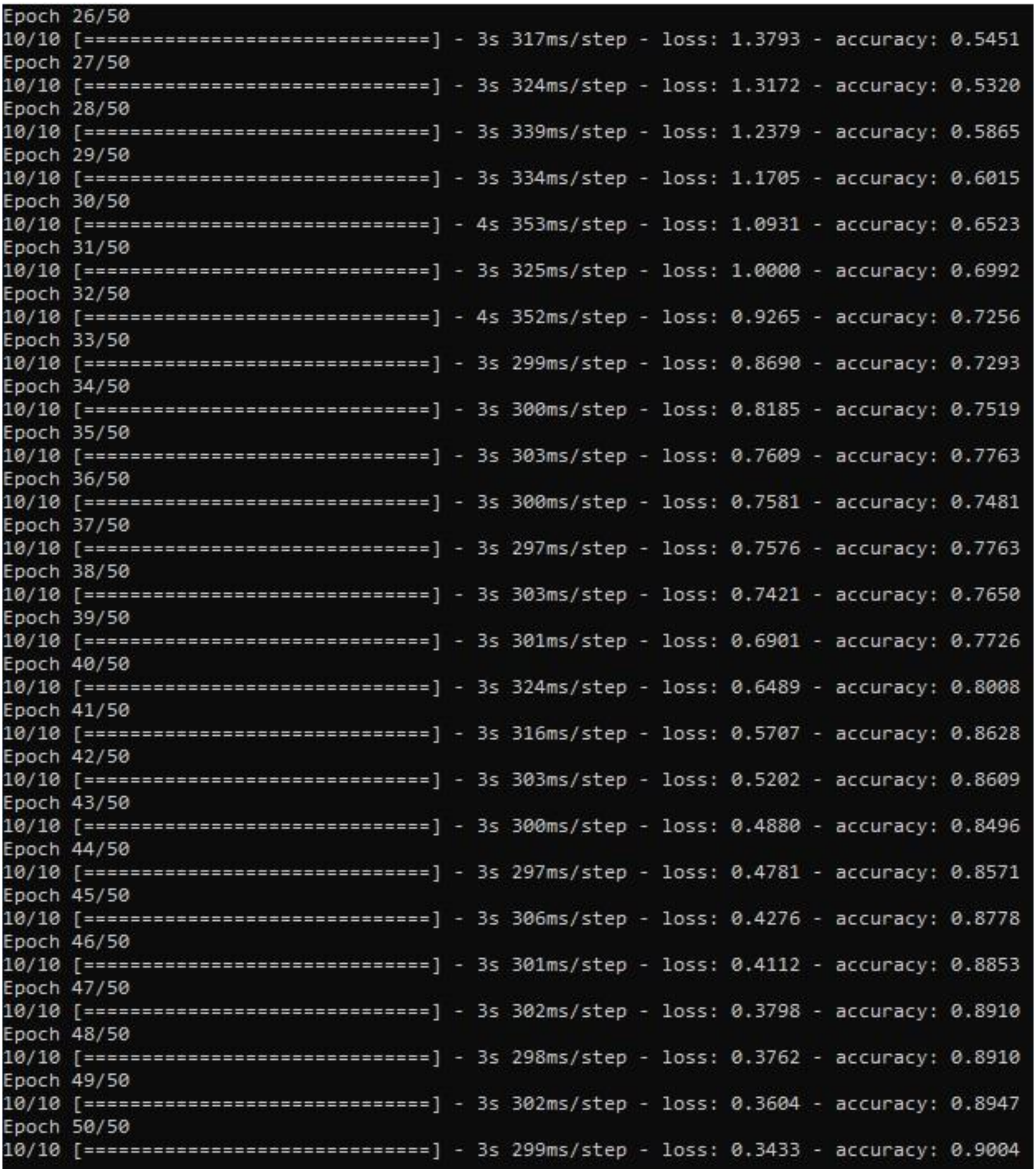
```
Dataset:  10
Descriptions: train= 10
Photos: train= 10
Vocabulary Size: 172
Description Length:  21
Model: "model_1"
```

Fig. 14: Train dataset



```
Epoch 1/50
10/10 [==============================] - 3s 290ms/step - loss: 5.0106 - accuracy: 0.0771
Epoch 2/50
10/10 [==============================] - 2s 212ms/step - loss: 4.3879 - accuracy: 0.0902
Epoch 3/50
10/10 [==============================] - 2s 208ms/step - loss: 4.1573 - accuracy: 0.1128
Epoch 4/50
10/10 [==============================] - 2s 222ms/step - loss: 3.9009 - accuracy: 0.1335
Epoch 5/50
10/10 [==============================] - 2s 218ms/step - loss: 3.6488 - accuracy: 0.1654
Epoch 6/50
10/10 [==============================] - 2s 220ms/step - loss: 3.3906 - accuracy: 0.1823
Epoch 7/50
10/10 [==============================] - 2s 216ms/step - loss: 3.2123 - accuracy: 0.1898
Epoch 8/50
10/10 [==============================] - 2s 245ms/step - loss: 3.1183 - accuracy: 0.1936
Epoch 9/50
10/10 [==============================] - 3s 297ms/step - loss: 3.0438 - accuracy: 0.1692
Epoch 10/50
10/10 [==============================] - 3s 300ms/step - loss: 2.8310 - accuracy: 0.2049
Epoch 11/50
10/10 [==============================] - 3s 319ms/step - loss: 2.6503 - accuracy: 0.2086
Epoch 12/50
10/10 [==============================] - 3s 316ms/step - loss: 2.4829 - accuracy: 0.2406
Epoch 13/50
10/10 [==============================] - 3s 316ms/step - loss: 2.3842 - accuracy: 0.2575
Epoch 14/50
10/10 [==============================] - 3s 325ms/step - loss: 2.3079 - accuracy: 0.2500
Epoch 15/50
10/10 [==============================] - 3s 314ms/step - loss: 2.3370 - accuracy: 0.2688
Epoch 16/50
10/10 [==============================] - 3s 302ms/step - loss: 2.2729 - accuracy: 0.2970
Epoch 17/50
10/10 [==============================] - 3s 298ms/step - loss: 2.1964 - accuracy: 0.2838
Epoch 18/50
10/10 [==============================] - 3s 296ms/step - loss: 2.0308 - accuracy: 0.3515
Epoch 19/50
10/10 [==============================] - 3s 303ms/step - loss: 1.8610 - accuracy: 0.3891
Epoch 20/50
10/10 [==============================] - 3s 299ms/step - loss: 1.7698 - accuracy: 0.4079
Epoch 21/50
10/10 [==============================] - 3s 302ms/step - loss: 1.6977 - accuracy: 0.4361
Epoch 22/50
10/10 [==============================] - 3s 303ms/step - loss: 1.6272 - accuracy: 0.4925
Epoch 23/50
10/10 [==============================] - 3s 307ms/step - loss: 1.5810 - accuracy: 0.4530
Epoch 24/50
10/10 [==============================] - 3s 300ms/step - loss: 1.5493 - accuracy: 0.5038
Epoch 25/50
10/10 [==============================] - 3s 302ms/step - loss: 1.4549 - accuracy: 0.5150
```

Fig. 15: 0th to 25th cycle of training



```
Epoch 26/50
10/10 [==============================] - 3s 317ms/step - loss: 1.3793 - accuracy: 0.5451
Epoch 27/50
10/10 [==============================] - 3s 324ms/step - loss: 1.3172 - accuracy: 0.5320
Epoch 28/50
10/10 [==============================] - 3s 339ms/step - loss: 1.2379 - accuracy: 0.5865
Epoch 29/50
10/10 [==============================] - 3s 334ms/step - loss: 1.1705 - accuracy: 0.6015
Epoch 30/50
10/10 [==============================] - 4s 353ms/step - loss: 1.0931 - accuracy: 0.6523
Epoch 31/50
10/10 [==============================] - 3s 325ms/step - loss: 1.0000 - accuracy: 0.6992
Epoch 32/50
10/10 [==============================] - 4s 352ms/step - loss: 0.9265 - accuracy: 0.7256
Epoch 33/50
10/10 [==============================] - 3s 299ms/step - loss: 0.8690 - accuracy: 0.7293
Epoch 34/50
10/10 [==============================] - 3s 300ms/step - loss: 0.8185 - accuracy: 0.7519
Epoch 35/50
10/10 [==============================] - 3s 303ms/step - loss: 0.7609 - accuracy: 0.7763
Epoch 36/50
10/10 [==============================] - 3s 300ms/step - loss: 0.7581 - accuracy: 0.7481
Epoch 37/50
10/10 [==============================] - 3s 297ms/step - loss: 0.7576 - accuracy: 0.7763
Epoch 38/50
10/10 [==============================] - 3s 303ms/step - loss: 0.7421 - accuracy: 0.7650
Epoch 39/50
10/10 [==============================] - 3s 301ms/step - loss: 0.6901 - accuracy: 0.7726
Epoch 40/50
10/10 [==============================] - 3s 324ms/step - loss: 0.6489 - accuracy: 0.8008
Epoch 41/50
10/10 [==============================] - 3s 316ms/step - loss: 0.5707 - accuracy: 0.8628
Epoch 42/50
10/10 [==============================] - 3s 303ms/step - loss: 0.5202 - accuracy: 0.8609
Epoch 43/50
10/10 [==============================] - 3s 300ms/step - loss: 0.4880 - accuracy: 0.8496
Epoch 44/50
10/10 [==============================] - 3s 297ms/step - loss: 0.4781 - accuracy: 0.8571
Epoch 45/50
10/10 [==============================] - 3s 306ms/step - loss: 0.4276 - accuracy: 0.8778
Epoch 46/50
10/10 [==============================] - 3s 301ms/step - loss: 0.4112 - accuracy: 0.8853
Epoch 47/50
10/10 [==============================] - 3s 302ms/step - loss: 0.3798 - accuracy: 0.8910
Epoch 48/50
10/10 [==============================] - 3s 298ms/step - loss: 0.3762 - accuracy: 0.8910
Epoch 49/50
10/10 [==============================] - 3s 302ms/step - loss: 0.3604 - accuracy: 0.8947
Epoch 50/50
10/10 [==============================] - 3s 299ms/step - loss: 0.3433 - accuracy: 0.9004
```

Fig 16: 26th to 50th cycle of training

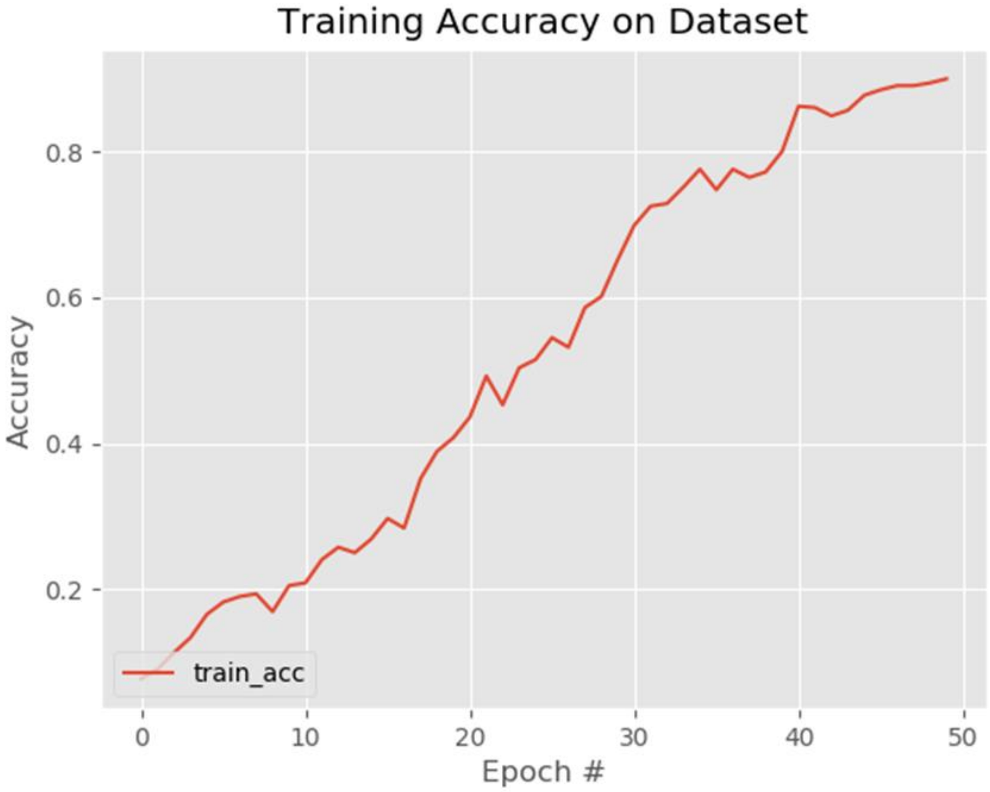


Fig. 17: Training accuracy

The above graph is a plot of the accuracy of the model when we trained it on an image dataset of 10 images which was self-created. As observed in the above graph (Fig. 17), we were able to attain an accuracy of around 90%. Though the graph is not a smooth one, we can see that our model as consistently increased its accuracy over the span of 50 epochs.

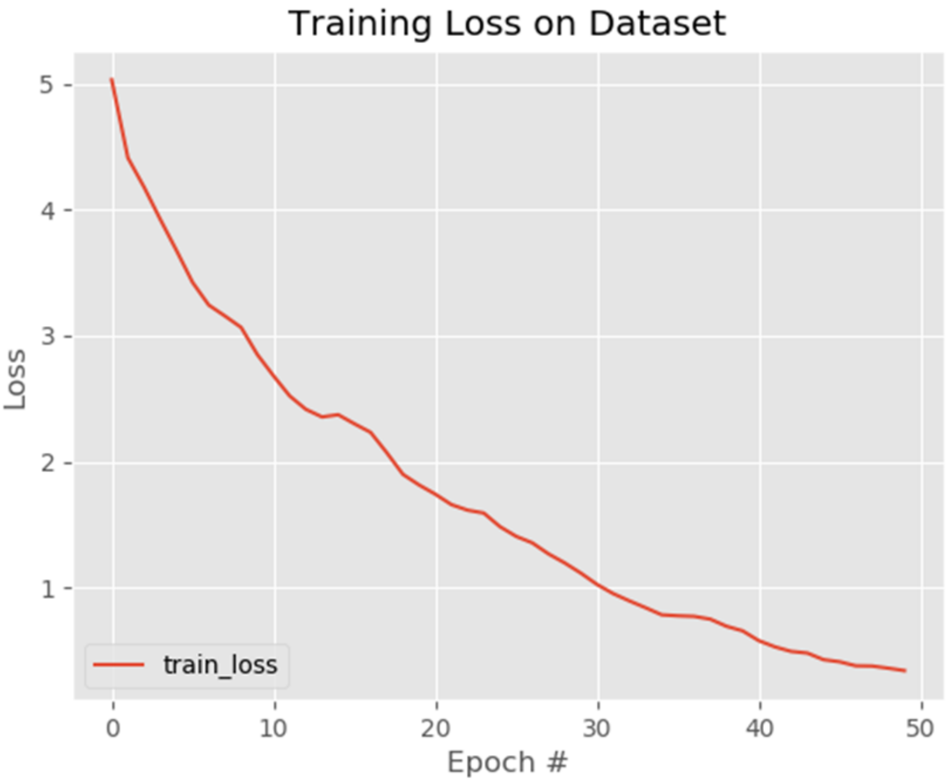


Fig. 18: Training loss

The above graph is a plot of the loss of the model when we trained it on an image dataset of 10 images which was self-created. As one can observe in Fig. 18, the loss has drastically reduced from 5 to 0.34 over a span of 50 epochs which is quite a significant result.

5.4.2 TRAINING ON FLICKR8K DATASET

Resources:

Laptop – HP Pavillion 15 (Model: hp 15ac120tx)

Processor: Intel i3 5005U – 2.00GHz

RAM: 16GB

Libraries:

TensorFlow: 2.0.0

Keras: 2.2.8

Dataset size

Total: 8000

Train: 1000

Test: 1000

Training Time

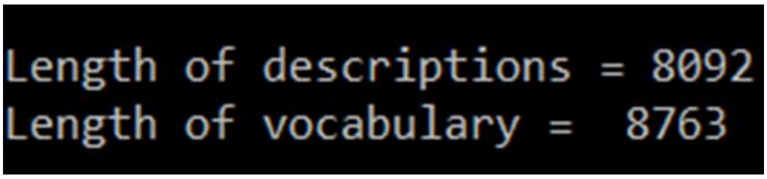
Time: sec

Training statistics:

Epochs - 75

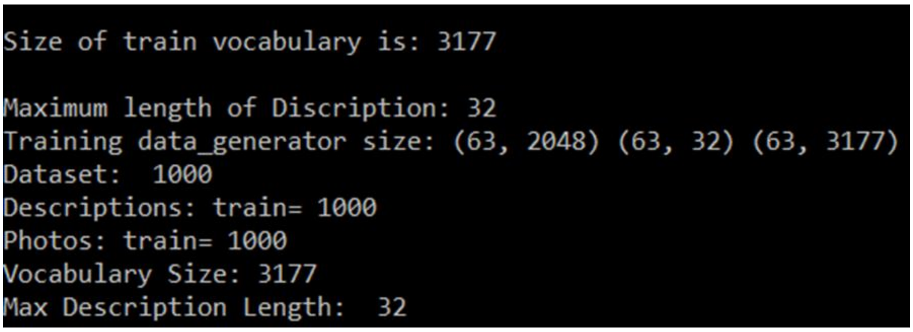
Accuracy - ~78.5%

Loss - ~0.65



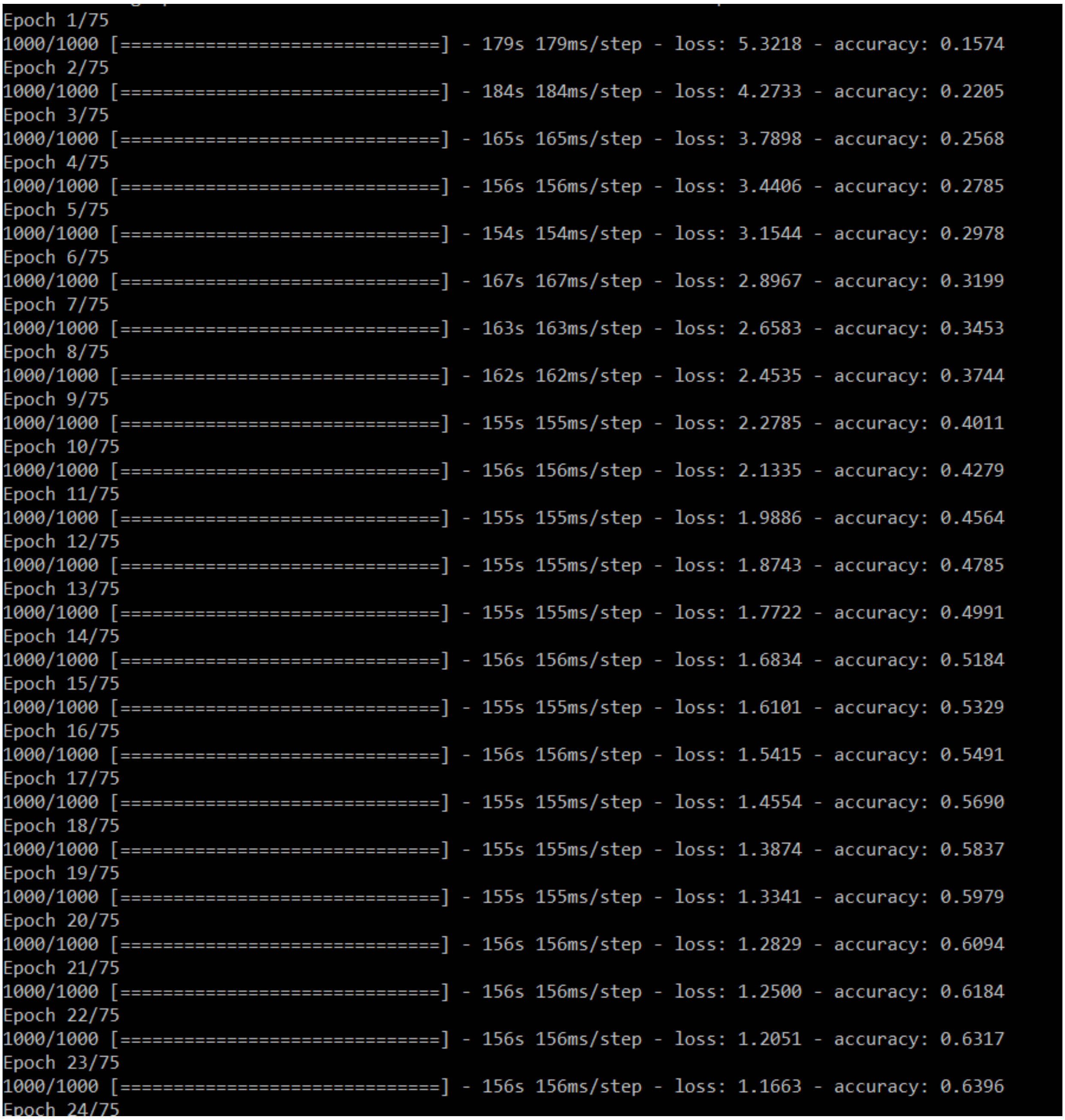
```
Length of descriptions = 8092
Length of vocabulary =  8763
```

Fig. 19: Total descriptions



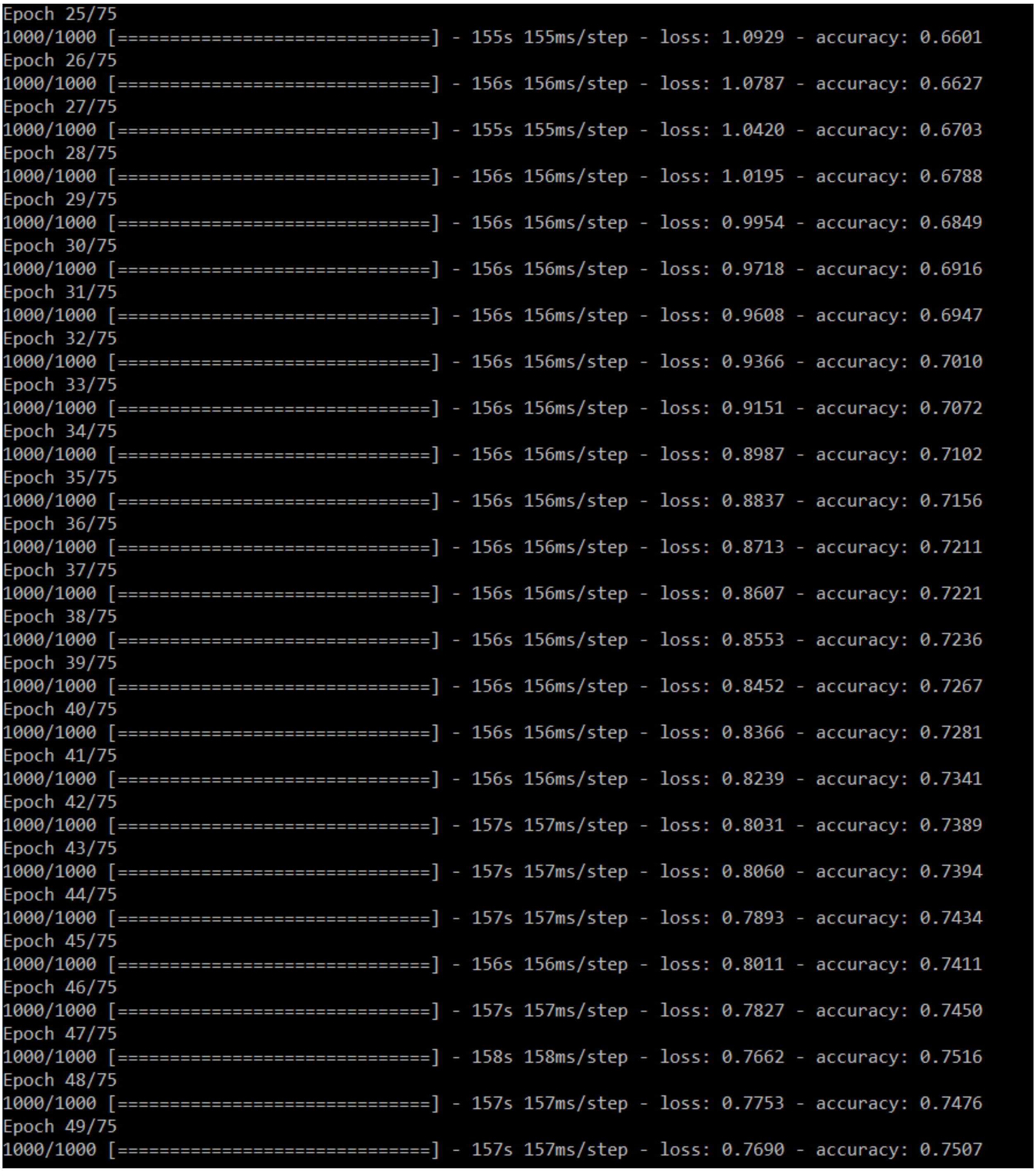
```
Size of train vocabulary is: 3177

Maximum length of Discription: 32
Training data_generator size: (63, 2048) (63, 32) (63, 3177)
Dataset:  1000
Descriptions: train= 1000
Photos: train= 1000
Vocabulary Size: 3177
Max Description Length:  32
```

Fig. 20: Train dataset



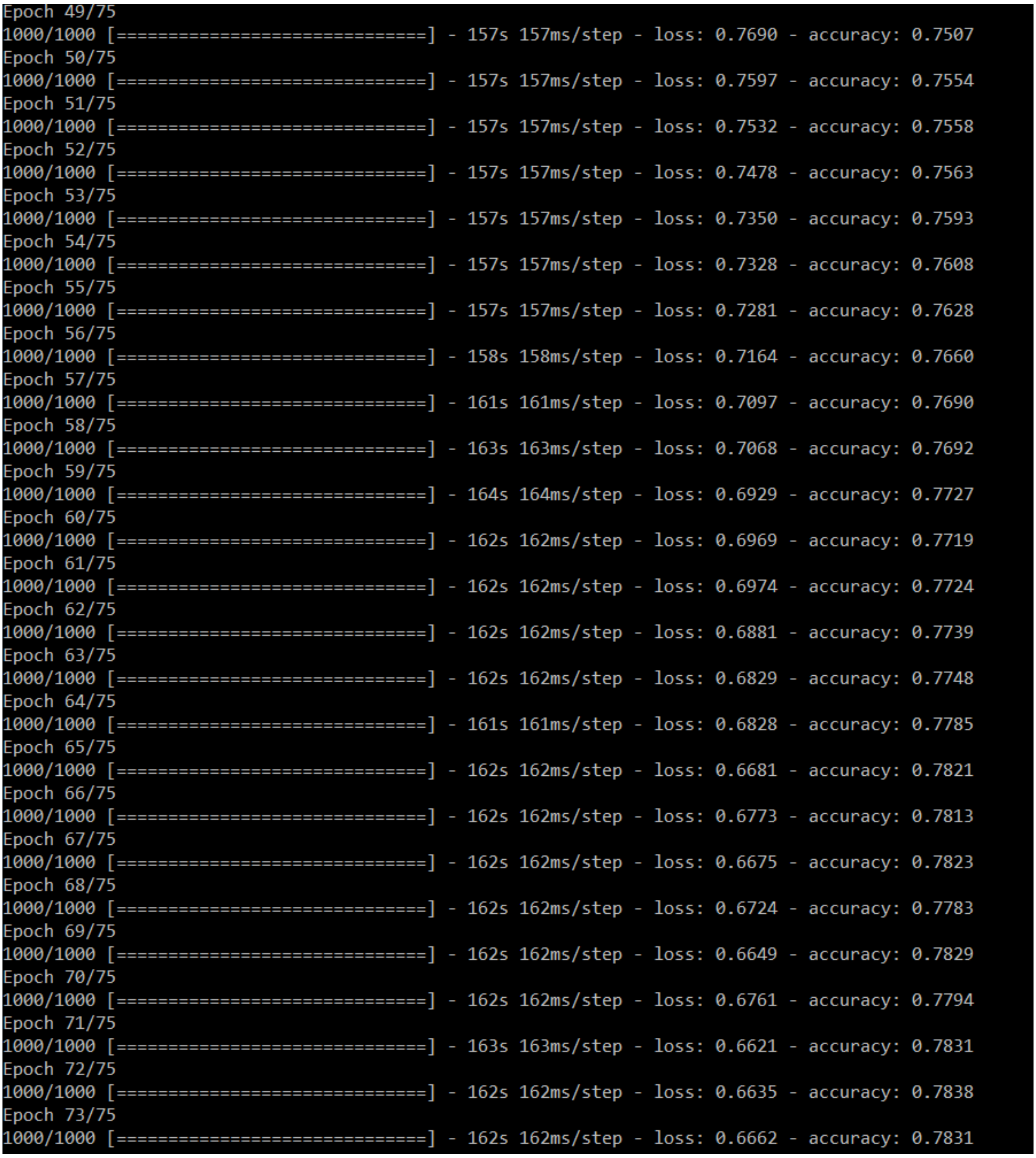
```
Epoch 1/75
1000/1000 [==============================] - 179s 179ms/step - loss: 5.3218 - accuracy: 0.1574
Epoch 2/75
1000/1000 [==============================] - 184s 184ms/step - loss: 4.2733 - accuracy: 0.2205
Epoch 3/75
1000/1000 [==============================] - 165s 165ms/step - loss: 3.7898 - accuracy: 0.2568
Epoch 4/75
1000/1000 [==============================] - 156s 156ms/step - loss: 3.4406 - accuracy: 0.2785
Epoch 5/75
1000/1000 [==============================] - 154s 154ms/step - loss: 3.1544 - accuracy: 0.2978
Epoch 6/75
1000/1000 [==============================] - 167s 167ms/step - loss: 2.8967 - accuracy: 0.3199
Epoch 7/75
1000/1000 [==============================] - 163s 163ms/step - loss: 2.6583 - accuracy: 0.3453
Epoch 8/75
1000/1000 [==============================] - 162s 162ms/step - loss: 2.4535 - accuracy: 0.3744
Epoch 9/75
1000/1000 [==============================] - 155s 155ms/step - loss: 2.2785 - accuracy: 0.4011
Epoch 10/75
1000/1000 [==============================] - 156s 156ms/step - loss: 2.1335 - accuracy: 0.4279
Epoch 11/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.9886 - accuracy: 0.4564
Epoch 12/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.8743 - accuracy: 0.4785
Epoch 13/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.7722 - accuracy: 0.4991
Epoch 14/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.6834 - accuracy: 0.5184
Epoch 15/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.6101 - accuracy: 0.5329
Epoch 16/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.5415 - accuracy: 0.5491
Epoch 17/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.4554 - accuracy: 0.5690
Epoch 18/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.3874 - accuracy: 0.5837
Epoch 19/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.3341 - accuracy: 0.5979
Epoch 20/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.2829 - accuracy: 0.6094
Epoch 21/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.2500 - accuracy: 0.6184
Epoch 22/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.2051 - accuracy: 0.6317
Epoch 23/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.1663 - accuracy: 0.6396
Epoch 24/75
```

Fig 21: 1st to 24th cycle of training



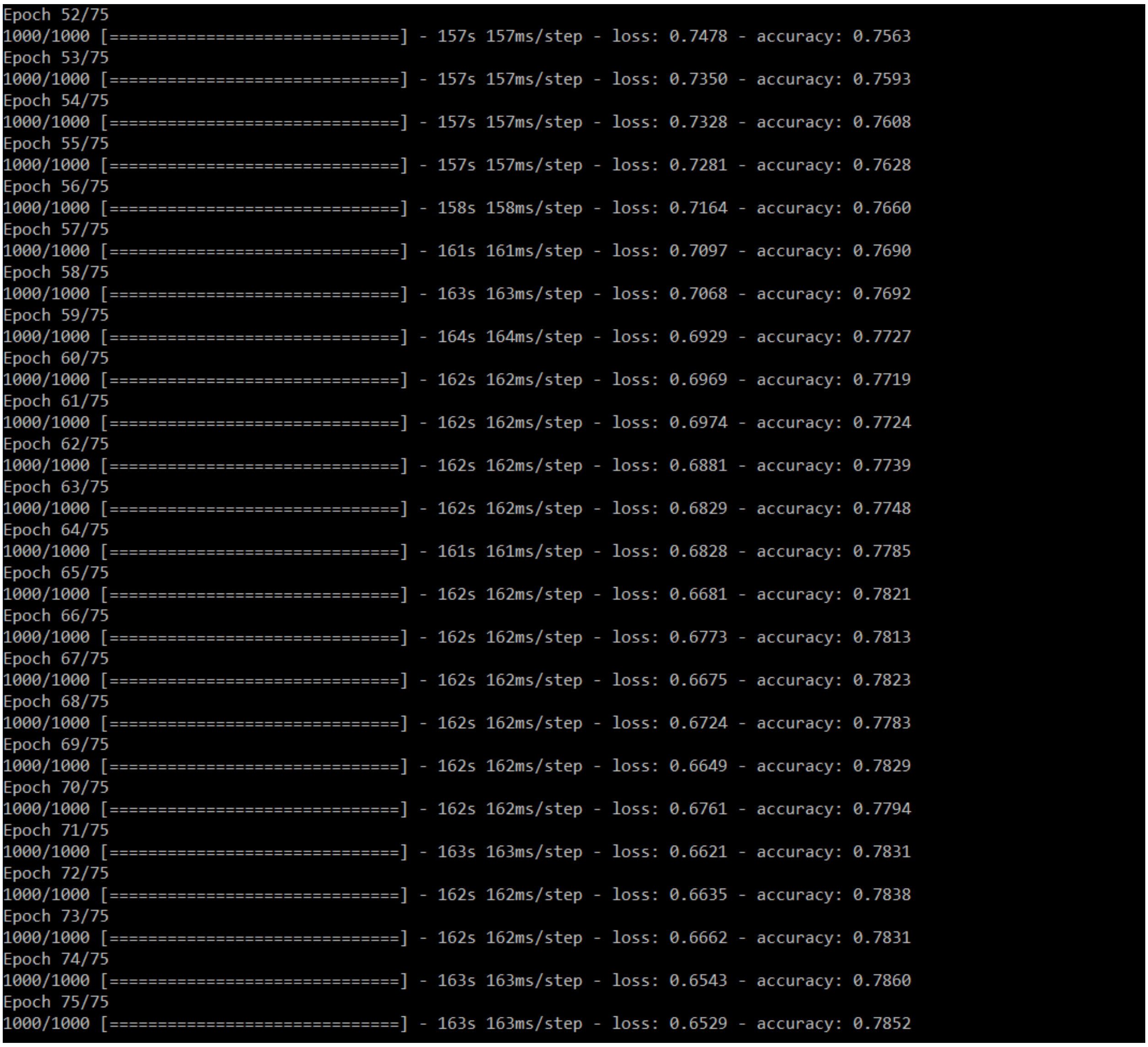
```
Epoch 25/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.0929 - accuracy: 0.6601
Epoch 26/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.0787 - accuracy: 0.6627
Epoch 27/75
1000/1000 [==============================] - 155s 155ms/step - loss: 1.0420 - accuracy: 0.6703
Epoch 28/75
1000/1000 [==============================] - 156s 156ms/step - loss: 1.0195 - accuracy: 0.6788
Epoch 29/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.9954 - accuracy: 0.6849
Epoch 30/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.9718 - accuracy: 0.6916
Epoch 31/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.9608 - accuracy: 0.6947
Epoch 32/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.9366 - accuracy: 0.7010
Epoch 33/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.9151 - accuracy: 0.7072
Epoch 34/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8987 - accuracy: 0.7102
Epoch 35/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8837 - accuracy: 0.7156
Epoch 36/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8713 - accuracy: 0.7211
Epoch 37/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8607 - accuracy: 0.7221
Epoch 38/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8553 - accuracy: 0.7236
Epoch 39/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8452 - accuracy: 0.7267
Epoch 40/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8366 - accuracy: 0.7281
Epoch 41/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8239 - accuracy: 0.7341
Epoch 42/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.8031 - accuracy: 0.7389
Epoch 43/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.8060 - accuracy: 0.7394
Epoch 44/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7893 - accuracy: 0.7434
Epoch 45/75
1000/1000 [==============================] - 156s 156ms/step - loss: 0.8011 - accuracy: 0.7411
Epoch 46/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7827 - accuracy: 0.7450
Epoch 47/75
1000/1000 [==============================] - 158s 158ms/step - loss: 0.7662 - accuracy: 0.7516
Epoch 48/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7753 - accuracy: 0.7476
Epoch 49/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7690 - accuracy: 0.7507
```

Fig 22: 25th to 49th cycle of training



```
Epoch 49/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7690 - accuracy: 0.7507
Epoch 50/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7597 - accuracy: 0.7554
Epoch 51/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7532 - accuracy: 0.7558
Epoch 52/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7478 - accuracy: 0.7563
Epoch 53/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7350 - accuracy: 0.7593
Epoch 54/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7328 - accuracy: 0.7608
Epoch 55/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7281 - accuracy: 0.7628
Epoch 56/75
1000/1000 [==============================] - 158s 158ms/step - loss: 0.7164 - accuracy: 0.7660
Epoch 57/75
1000/1000 [==============================] - 161s 161ms/step - loss: 0.7097 - accuracy: 0.7690
Epoch 58/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.7068 - accuracy: 0.7692
Epoch 59/75
1000/1000 [==============================] - 164s 164ms/step - loss: 0.6929 - accuracy: 0.7727
Epoch 60/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6969 - accuracy: 0.7719
Epoch 61/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6974 - accuracy: 0.7724
Epoch 62/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6881 - accuracy: 0.7739
Epoch 63/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6829 - accuracy: 0.7748
Epoch 64/75
1000/1000 [==============================] - 161s 161ms/step - loss: 0.6828 - accuracy: 0.7785
Epoch 65/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6681 - accuracy: 0.7821
Epoch 66/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6773 - accuracy: 0.7813
Epoch 67/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6675 - accuracy: 0.7823
Epoch 68/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6724 - accuracy: 0.7783
Epoch 69/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6649 - accuracy: 0.7829
Epoch 70/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6761 - accuracy: 0.7794
Epoch 71/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.6621 - accuracy: 0.7831
Epoch 72/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6635 - accuracy: 0.7838
Epoch 73/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6662 - accuracy: 0.7831
```

Fig 23: 49th to 73rd cycle of training



```
Epoch 52/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7478 - accuracy: 0.7563
Epoch 53/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7350 - accuracy: 0.7593
Epoch 54/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7328 - accuracy: 0.7608
Epoch 55/75
1000/1000 [==============================] - 157s 157ms/step - loss: 0.7281 - accuracy: 0.7628
Epoch 56/75
1000/1000 [==============================] - 158s 158ms/step - loss: 0.7164 - accuracy: 0.7660
Epoch 57/75
1000/1000 [==============================] - 161s 161ms/step - loss: 0.7097 - accuracy: 0.7690
Epoch 58/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.7068 - accuracy: 0.7692
Epoch 59/75
1000/1000 [==============================] - 164s 164ms/step - loss: 0.6929 - accuracy: 0.7727
Epoch 60/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6969 - accuracy: 0.7719
Epoch 61/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6974 - accuracy: 0.7724
Epoch 62/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6881 - accuracy: 0.7739
Epoch 63/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6829 - accuracy: 0.7748
Epoch 64/75
1000/1000 [==============================] - 161s 161ms/step - loss: 0.6828 - accuracy: 0.7785
Epoch 65/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6681 - accuracy: 0.7821
Epoch 66/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6773 - accuracy: 0.7813
Epoch 67/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6675 - accuracy: 0.7823
Epoch 68/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6724 - accuracy: 0.7783
Epoch 69/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6649 - accuracy: 0.7829
Epoch 70/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6761 - accuracy: 0.7794
Epoch 71/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.6621 - accuracy: 0.7831
Epoch 72/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6635 - accuracy: 0.7838
Epoch 73/75
1000/1000 [==============================] - 162s 162ms/step - loss: 0.6662 - accuracy: 0.7831
Epoch 74/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.6543 - accuracy: 0.7860
Epoch 75/75
1000/1000 [==============================] - 163s 163ms/step - loss: 0.6529 - accuracy: 0.7852
```

Fig 24: 52nd to 75th cycle of training

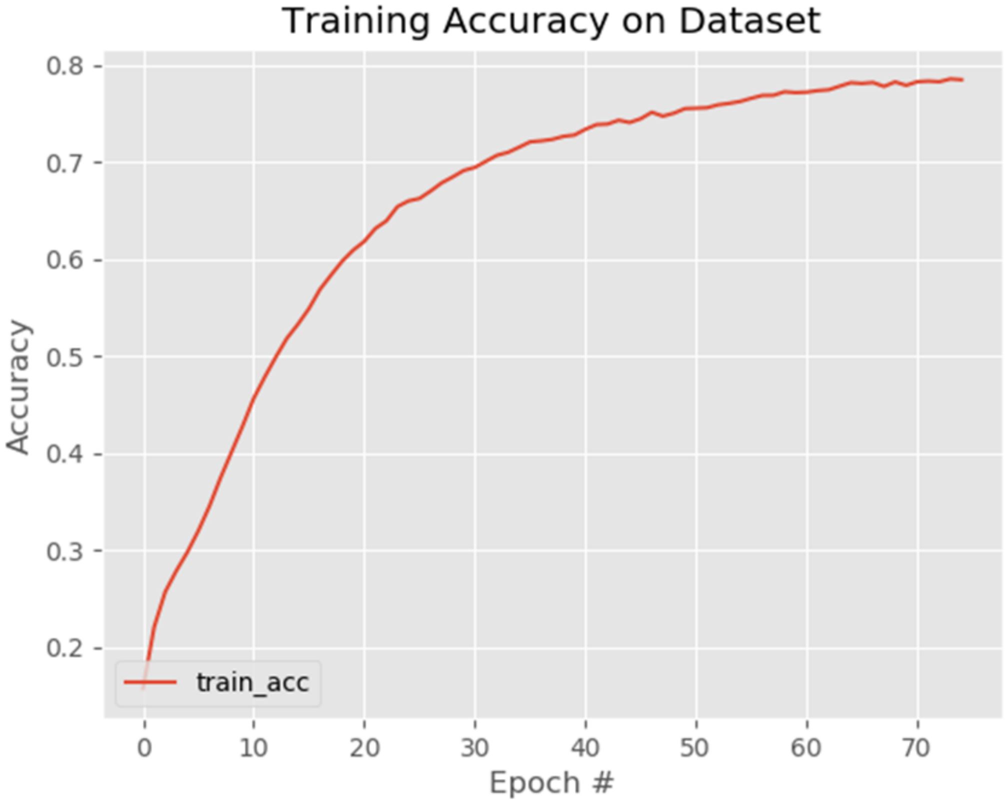


Fig. 25: Training accuracy

The above graph is a plot of the accuracy of the model when we trained it on Flickr8k image dataset. Out of the 6000 images in the training set, we selected only 1000 images as we had limited computational resources and time. As observed in the above graph (Fig. 25), we were able to attain an accuracy of around 78.5%. Here the graph has a smooth one, and we can see that our model as smoothly increased its accuracy over the span of 75 epochs.

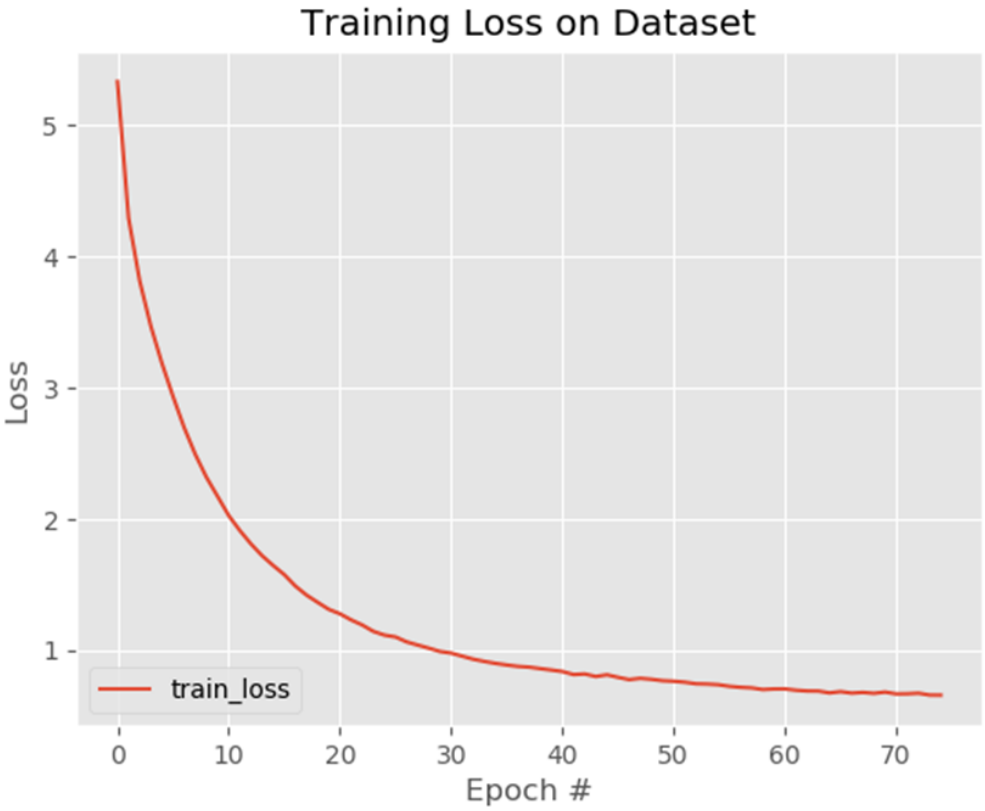


Fig. 26: Training loss

The above graph is a plot of the loss of the model when we trained it on Flickr8k image dataset where out of 6000 train images, we only selected 1000. As one can observe in Fig. 26, the loss has drastically reduced from 5.5 to 0.65 over a span of 75 epochs which is quite a significant result.

5.5 TESTING

Following are some samples of how our model generates captions for the image provided.

Resources:

Laptop – HP Pavillion 15 (Model: hp 15ac120tx)

Processor: Intel i3 5005U – 2.00GHz

RAM: 16GB

Libraries:

TensorFlow: 2.0.0

Keras: 2.2.8

1.

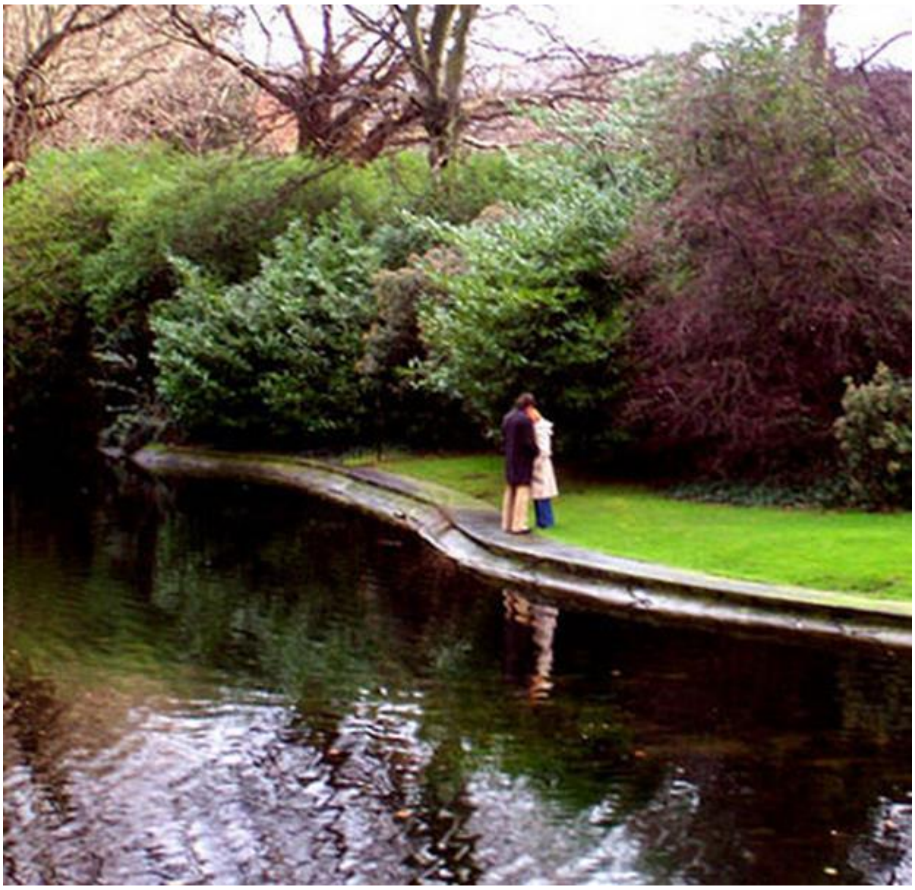
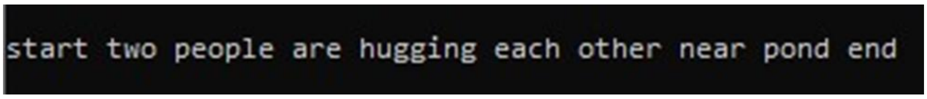


Fig. 27: Sample photo#1 passed as input



```
start two people are hugging each other near pond end
```

Fig. 28: Caption generated for the input image#1

2.

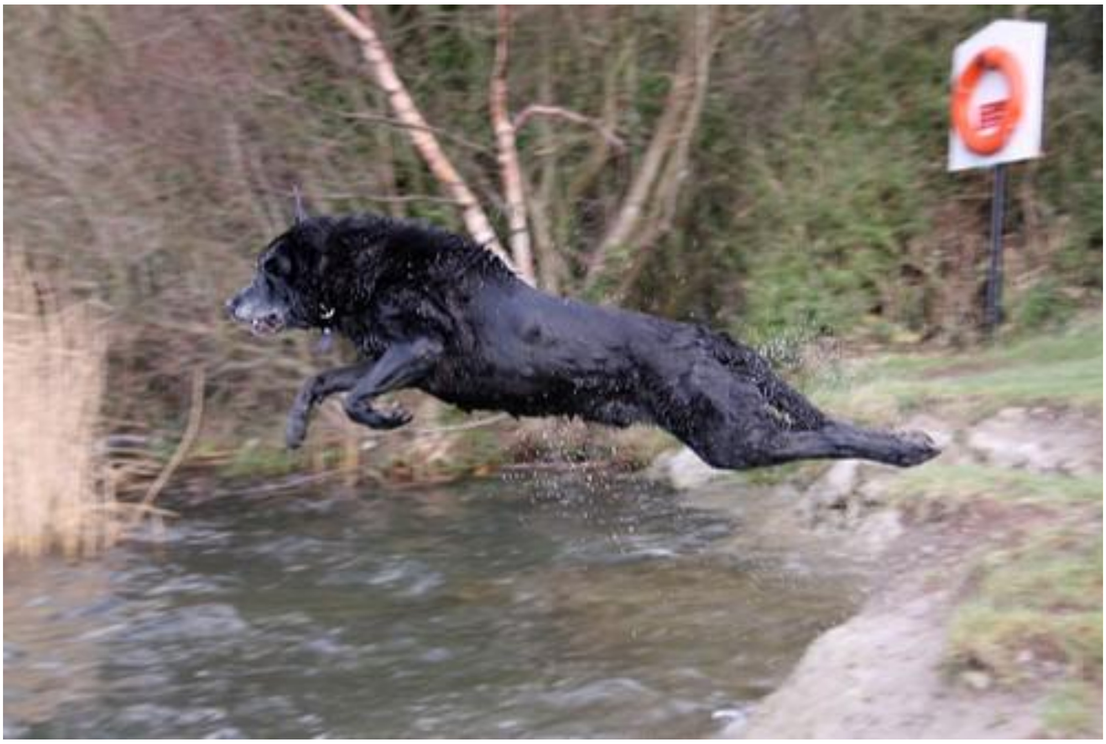
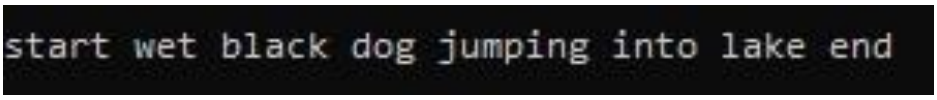


Fig. 29: Sample photo#2 passed as input



```
start wet black dog jumping into lake end
```

Fig. 30: Caption generated for the input image#2

3.

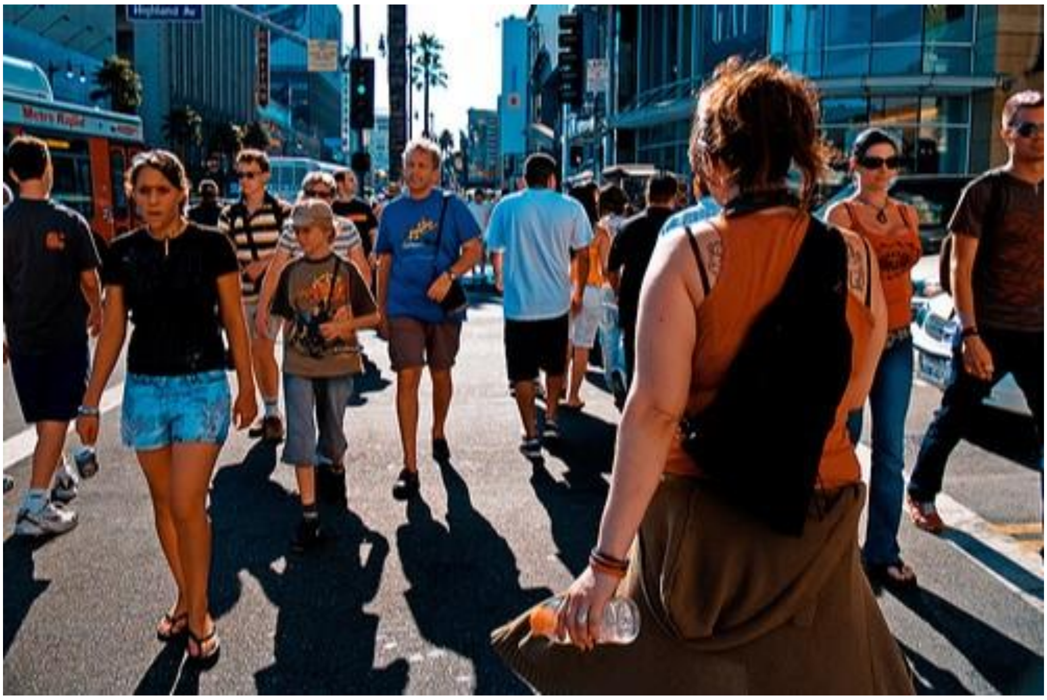
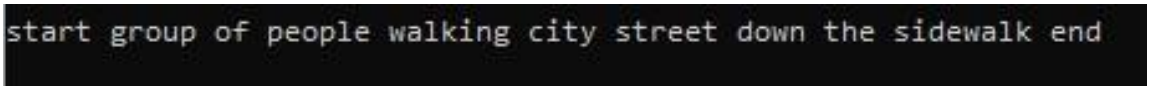


Fig. 31: Sample photo#3 passed as input



```
start group of people walking city street down the sidewalk end
```

Fig. 32: Caption generated for the input image#3

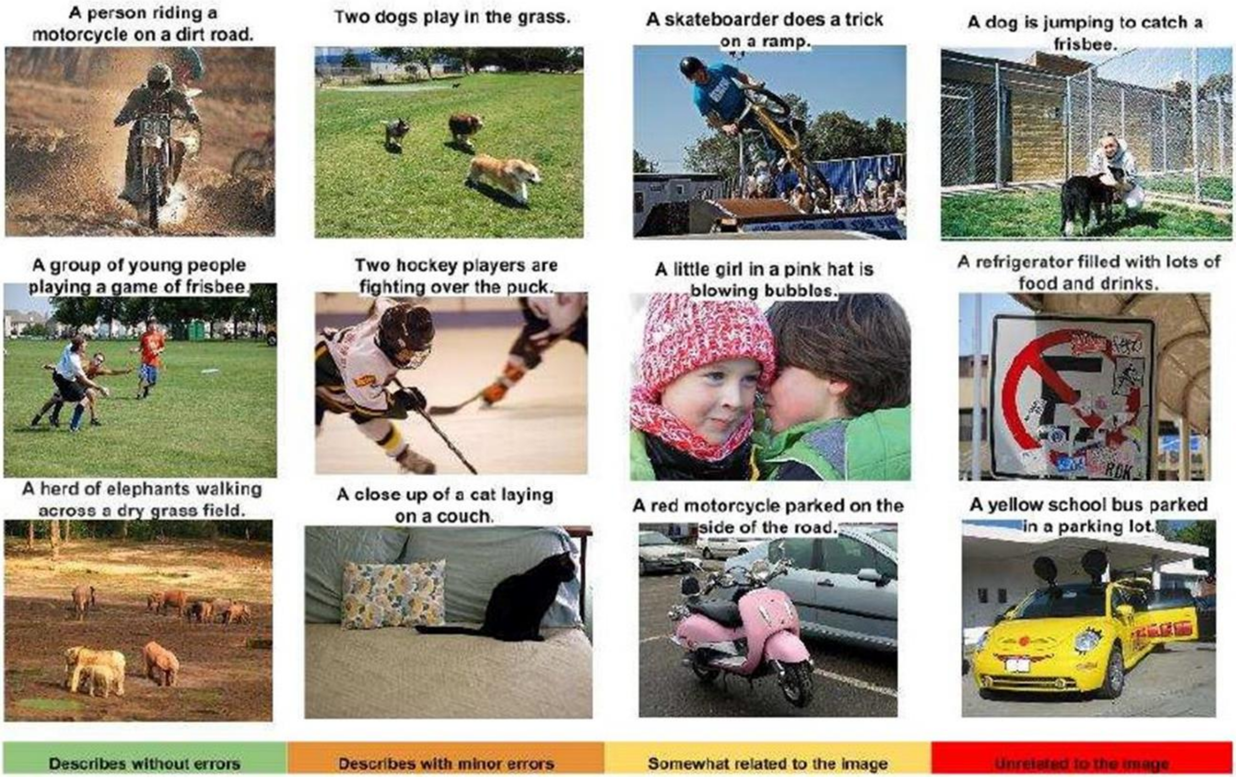

A person riding a motorcycle on a dirt road.
Two dogs play in the grass.
A skateboarder does a trick on a ramp.
A dog is jumping to catch a frisbee.
A group of young people playing a game of frisbee.
Two hockey players are fighting over the puck.
A little girl in a pink hat is blowing bubbles.
A refrigerator filled with lots of food and drinks.
A herd of elephants walking across a dry grass field.
A close up of a cat laying on a couch.
A red motorcycle parked on the side of the road.
A yellow school bus parked in a parking lot.
Describes without errors
Describes with minor errors
Somewhat related to the image
Unrelated to the image

Fig. 33: Evaluation of results based on human ratings

We performed testing on our model by sending several images and then compared and categorised the sentences generated by our model based on human rating. Fig. 33 shows the collated results.

6. RESULTS AND LIMITATIONS

6.1 RESULTS

After we trained our model, we tested it on a few images to check whether the model generated an appropriate caption or not. The captions were pretty much describing the image. The preciseness of the captions that our model generates mostly depends on how elaborate the captions were during the training phase. More extensive the vocabulary, higher will be the result. Sometimes, the model might not be able to describe the image accurately, but by training the model on a larger dataset, the results can be improved.

6.2 LIMITATIONS

The accuracy of generating a clear caption depends mainly on the size of the dataset and the descriptions of the image. More the variety of the images used during the training; more will the model know how to generate correct captions.

The other limitation would be that the model describing close or similar-looking things with the most common name. For example, if the model had been trained with several horse images, but very few donkeys or pony images, when generating a caption of an image consisting of a donkey, the model is more likely to describe the image with a horse because it is not able to differentiate much between the horse, donkey and a pony as there are not many visible differences between them.

CONCLUSION AND FUTURE SCOPE

In this project, we implement a model as discussed in the research paper we referenced [25], which is a single-joint model based on a neural network system that is capable of taking an image as input, analyses it and then automatically produces a plain logical explanation of that image in English. The model uses the Xception CNN to extract features from the photo or image given as input and then give it to an LSTM RNN which later generates the description of that image. We trained our model to enhance the probability of generating the correct description for the image. Our several experiments show that the sentences that are produced are a very reasonable description of the image, which makes our model robust in terms of qualitative outcomes. From our experiments, it is also clear that the higher the size of the dataset used, the greater will be the model's performance.

We believe that, out of all the human-developed models present for the Image Captioning problem, our system has the upper hand in it. Even though our obtained results are pretty good, we hope that with the daily growth in datasets, the models will keep getting increasingly better and better results will keep intensifying. Furthermore, it would be amazing to see if one could design image captioning approaches using unsupervised data (both from images alone and text alone) to achieve such results.

References:

A. Journals / Articles

[1] Timo Ojala, Matti PietikÃď'inen, and Topi MÃďenpÃďÃď. 2000. Grayscale and rotation invariant texture classification with local binary patterns. In **European Conference on Computer Vision. Springer, 404–420**.

[2] David G Lowe. 2004. Distinctive image features from scale-invariant keypoints. **International journal of computer vision 60, 2 (2004), 91–110**.

[3] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition, 2005. **CVPR 2005. IEEE Computer Society Conference on, Vol. 1. IEEE, 886–893**.

[4] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In Proceedings of the fifth annual workshop on **Computational learning theory. ACM, 144–152**.

[5] Yann LeCun, LÃĺ'on Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. **Proc. IEEE 86, 11 (1998), 2278–2324**.

[6] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. Every picture tells a story: Generating sentences from images. In **ECCV, 2010**.

[7] R. Gerber and H.-H. Nagel. Knowledge representation for the generation of quantified natural language descriptions of vehicle traffic in image sequences. In **ICIP. IEEE, 1996**.

[8] Y. Gong, L. Wang, M. Hodosh, J. Hockenmaier, and S. Lazebnik. Improving image-sentence embeddings using large weakly annotated photo collections. In **ECCV, 2014**.

[9] A. Graves. Generating sequences with recurrent neural networks. **arXiv:1308.0850, 2013**.

[10] S. Hochreiter and J. Schmidhuber. Long short-term memory. **Neural Computation, 9(8), 1997**.

[11] M. Hodosh, P. Young, and J. Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. **JAIR, 47, 2013**.

[12] S. Ioffe and C. Szegedy. Batch normalisation: Accelerating deep network training by reducing internal covariate shift. In **arXiv:1502.03167, 2015**.

[13] A. Karpathy, A. Joulin, and L. Fei-Fei. Deep fragment embeddings for bidirectional image sentence mapping. **NIPS, 2014**.

[14] R. Kiros, R. Salakhutdinov, and R. S. Zemel. Unifying visual-semantic embeddings with multimodal neural language models. In **arXiv:1411.2539, 2014**.

[15] R. Kiros and R. Z. R. Salakhutdinov. Multimodal neural language models. In **NIPS Deep Learning Workshop, 2013**.

[16] G. Kulkarni, V. Premraj, S. Dhar, S. Li, Y. Choi, A. C. Berg, and T. L. Berg. Baby talk: Understanding and generating simple image descriptions. In **CVPR, 2011**.

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll' ar, and C. L. Zitnick. Microsoft coco: Common objects in context. **arXiv:1405.0312, 2014**.

[18] V. Ordonez, G. Kulkarni, and T. L. Berg. Im2text: Describing images using 1 million captioned photographs. In **NIPS, 2011**.

[19] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier. Collecting image annotations using amazon's mechanical turk. In **NAACL HLT Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk, pages 139– 147, 2010**.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[21] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localisation and detection using convolutional networks. **arXiv preprint arXiv:1312.6229, 2013**.

[22] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. In **ACL, 2014**.

[23] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularisation. In **arXiv:1409.2329, 2014**.

[24] Francois Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. **IEEE CVPR.2017.195**.

[25] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan. 2015. Show and Tell: A Neural Image Caption Generator. **IEEE 2015 ISBN 978-1-4673-6964-0/15**.

B. E-websites

1. https://machinelearningmastery.com/deep-learning-caption-generation-models/
2. https://colab.research.google.com/