Capstone Project on Image Caption Generator with CNN and LSTM

Project Report By Bhavesh Baghele

CONTENTS

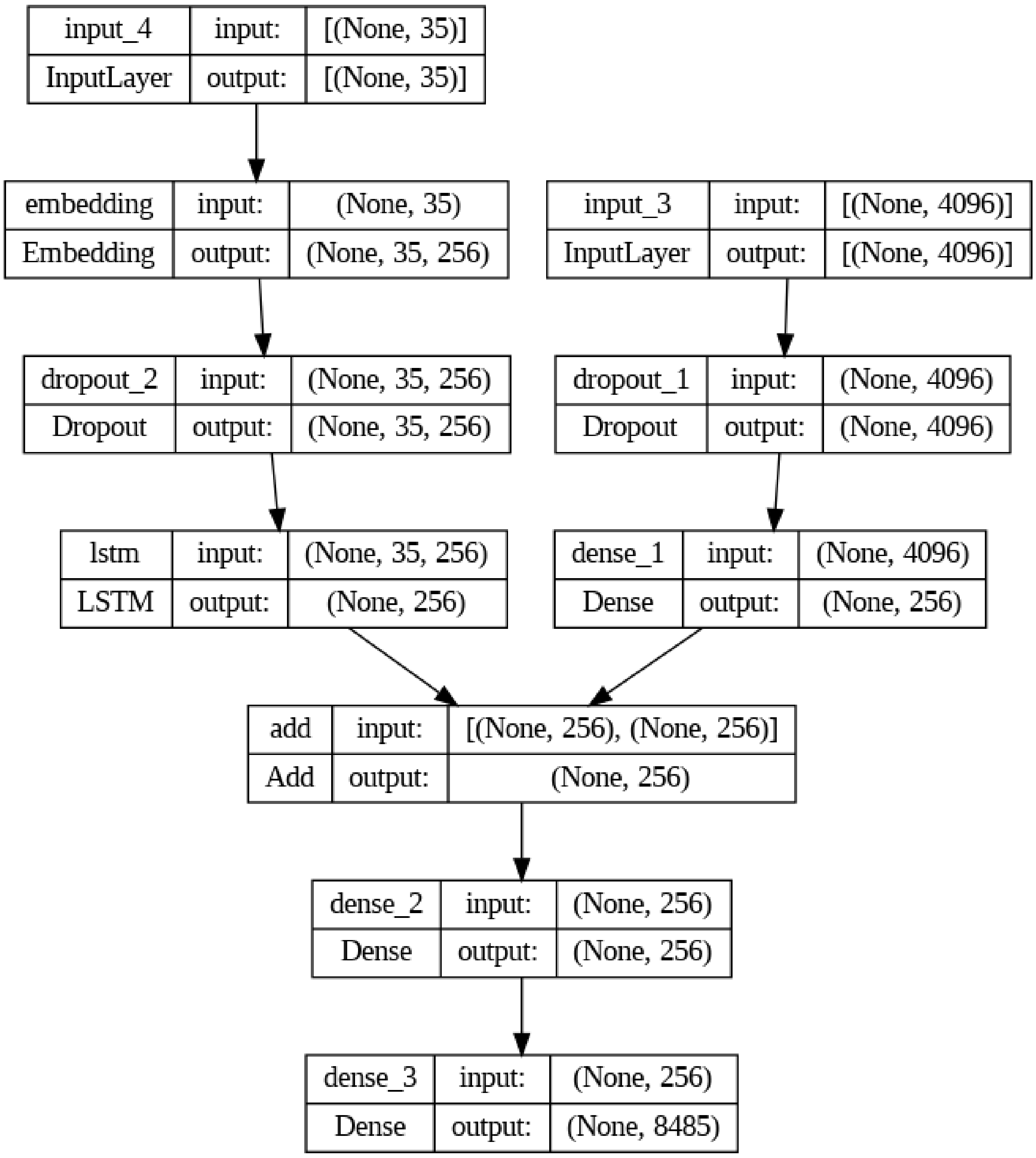
Introduction to the Project

Generating accurate captions for an image has remained as one of the major challenges in Artificial Intelligence with plenty of applications ranging from robotic vision to helping the visually impaired. Long term applications also involve providing accurate captions for videos in scenarios such as security system. "Image caption generator": the name itself suggests that we aim to build an optimal system which can generate semantically and grammatically accurate captions for an image. Researchers have been involved in finding an efficient way to make better predictions, therefore we have discussed a few methods to achieve good results. We have used the deep neural networks and machine learning techniques to build a good model. We have used Flickr 8k dataset which contains around 8000 sample images with their five captions for each image. There are two phases : feature extraction from the image using Convolutional Neural Networks (CNN) and generating sentences in natural language based on the image using Recurrent Neural Networks (RNN). For the first phase, rather than just detecting the objects present in the image, we have used a different approach of extracting features of an image which will give us details of even the slightest difference between two similar images. We have used VGG-16 (Visual Geometry Group), which is a 16 convolutional layers model used for object recognition.

Objective

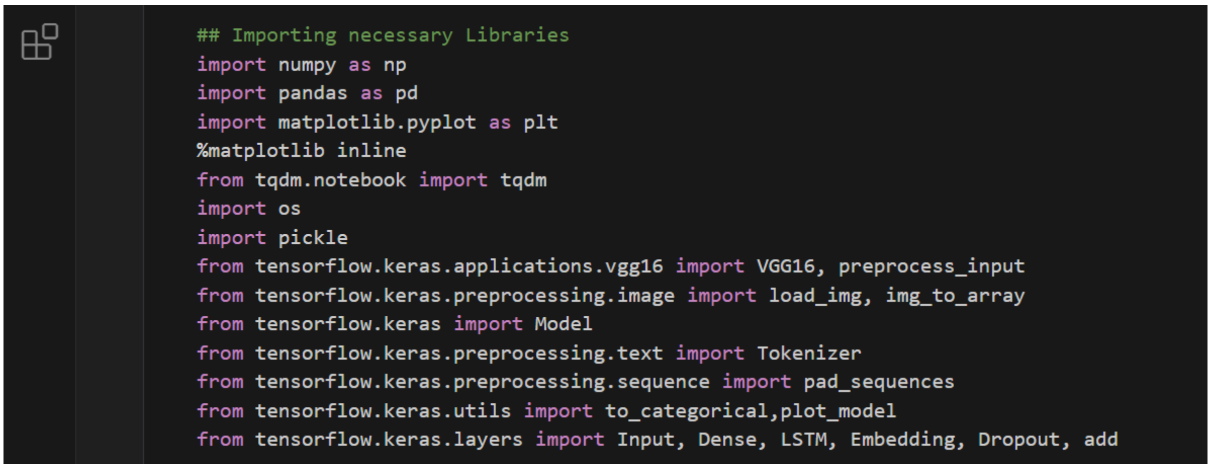
The objective of the project is to predict the captions for the input image. The dataset consists of 8k images and 5 captions for each image. The features are extracted from both the image and the text captions for input. The features will be concatenated to predict the next word of the caption. CNN is used for image and LSTM is used for text. BLEU Score is used as a metric to evaluate the performance of the trained model.

FLOW CHART


input_4
InputLayer
input:
output:
[(None, 35)]
[(None, 35)]
embedding
Embedding
input:
output:
(None, 35)
(None, 35, 256)
input_3
InputLayer
input:
output:
[(None, 4096)]
[(None, 4096)]
dropout_2
Dropout
input:
output:
(None, 35, 256)
(None, 35, 256)
dropout_1
Dropout
input:
output:
(None, 4096)
(None, 4096)
lstm
LSTM
input:
output:
(None, 35, 256)
(None, 256)
dense_1
Dense
input:
output:
(None, 4096)
(None, 256)
add
Add
input:
output:
[(None, 256), (None, 256)]
(None, 256)
dense_2
Dense
input:
output:
(None, 256)
(None, 256)
dense_3
Dense
input:
output:
(None, 256)
(None, 8485)

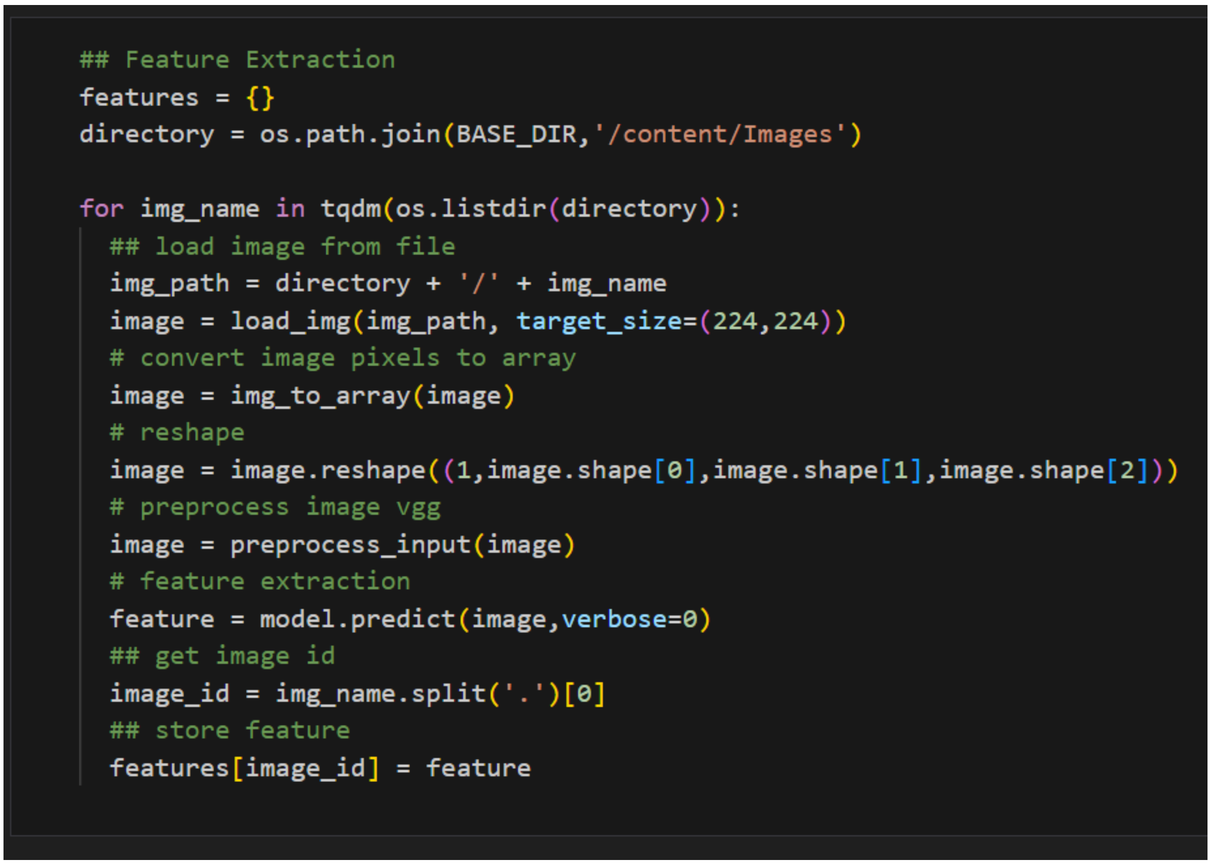
Python Codes

Importing Libraries



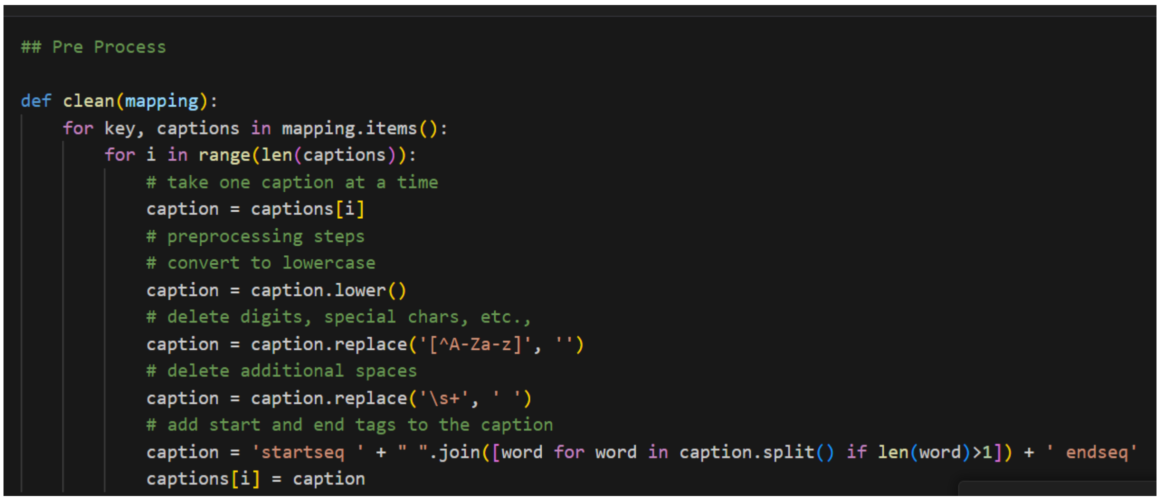
```python
## Importing necessary Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from tqdm.notebook import tqdm
import os
import pickle
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical,plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

Feature Extraction



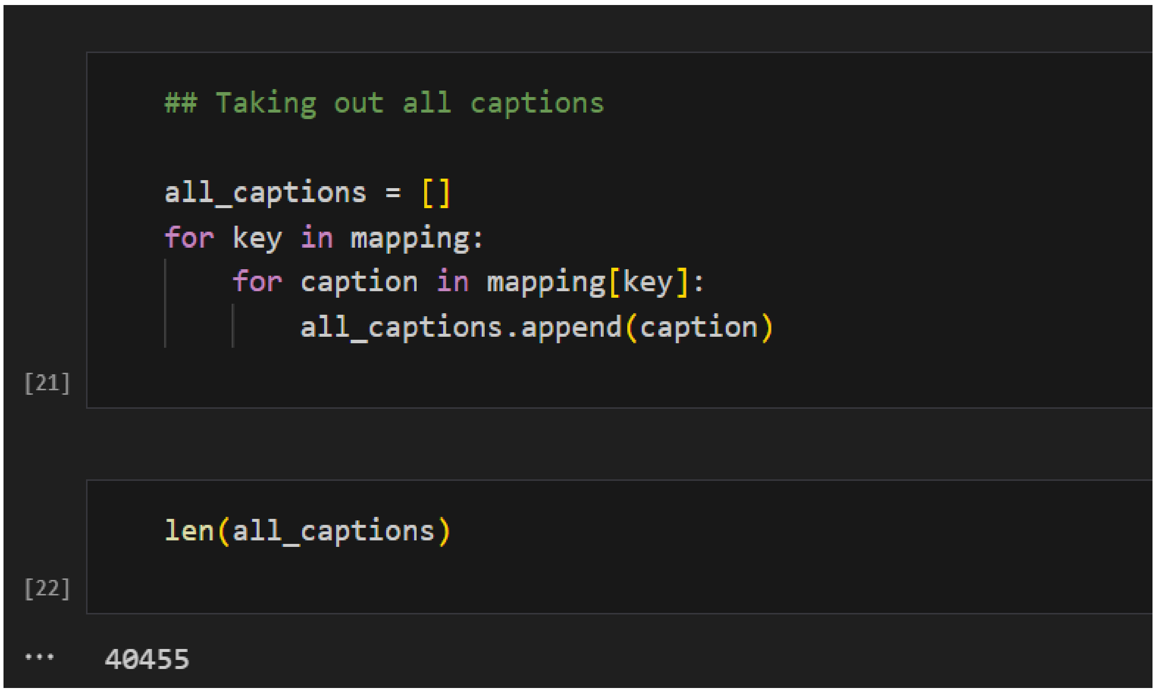
```python
## Feature Extraction
features = {}
directory = os.path.join(BASE_DIR,'/content/Images')

for img_name in tqdm(os.listdir(directory)):
    ## load image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224,224))
    # convert image pixels to array
    image = img_to_array(image)
    # reshape
    image = image.reshape((1,image.shape[0],image.shape[1],image.shape[2]))
    # preprocess image vgg
    image = preprocess_input(image)
    # feature extraction
    feature = model.predict(image,verbose=0)
    ## get image id
    image_id = img_name.split('.')[0]
    ## store feature
    features[image_id] = feature
```

Pre Processing



```
## Pre Process

def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

Captions

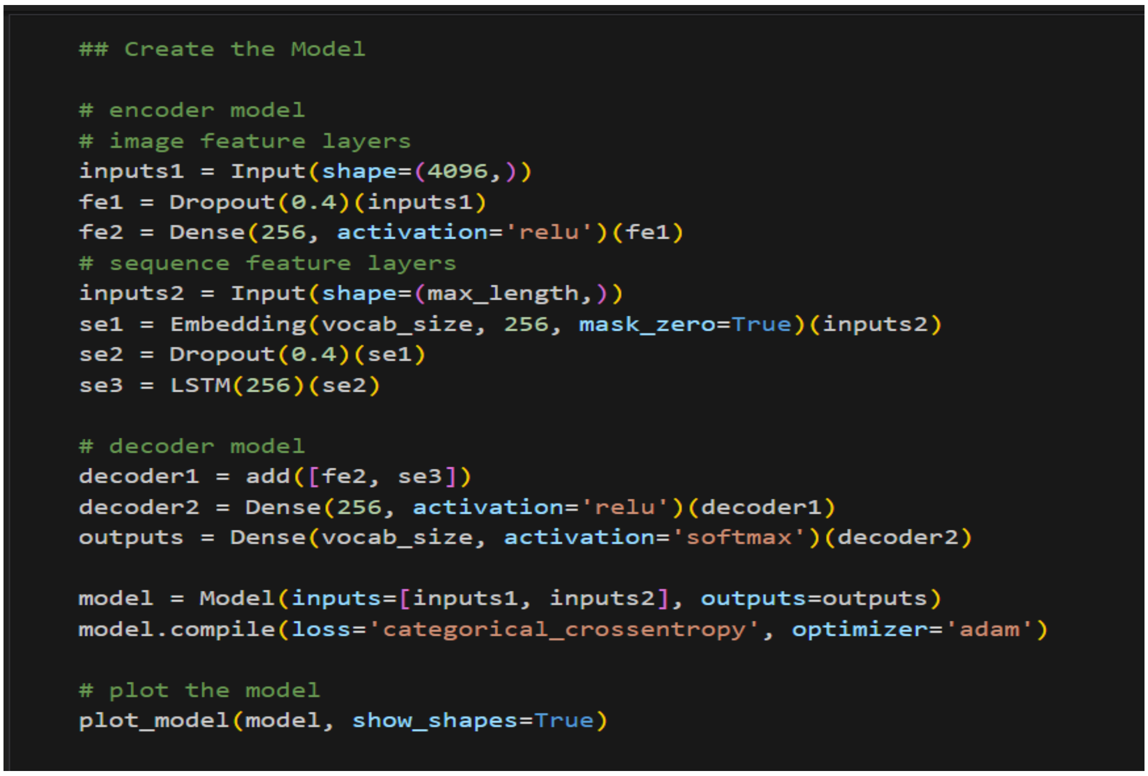


```
## Taking out all captions

all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
```
[21]

```
len(all_captions)
```
[22]

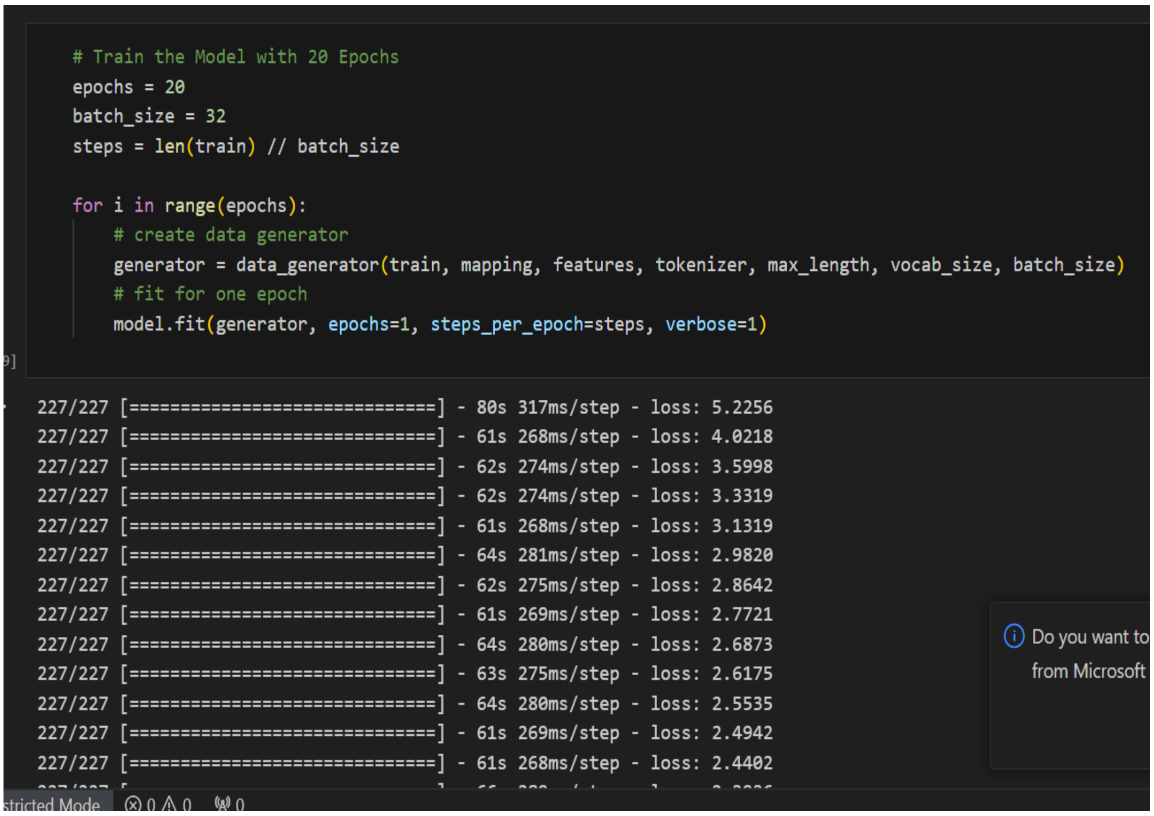
```
... 40455
```

Model Creation



```python
## Create the Model

# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

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

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# plot the model
plot_model(model, show_shapes=True)
```

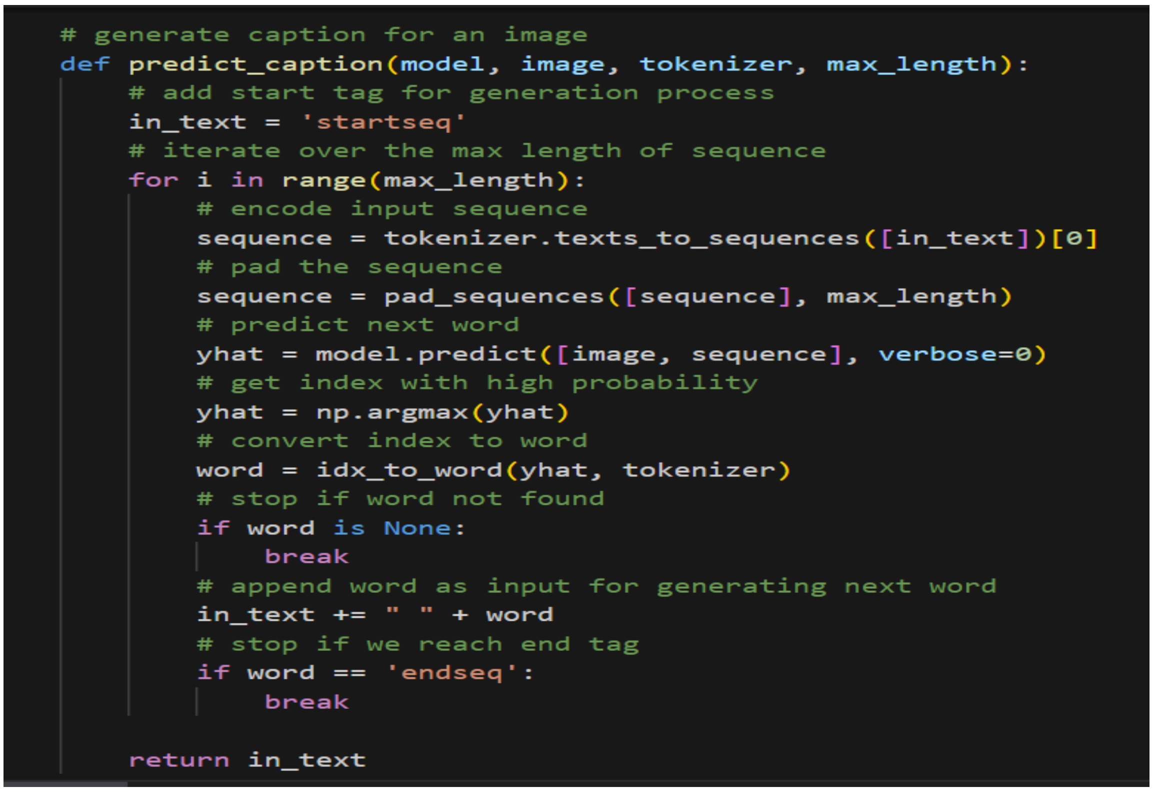
Training the Model



```python
# Train the Model with 20 Epochs
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

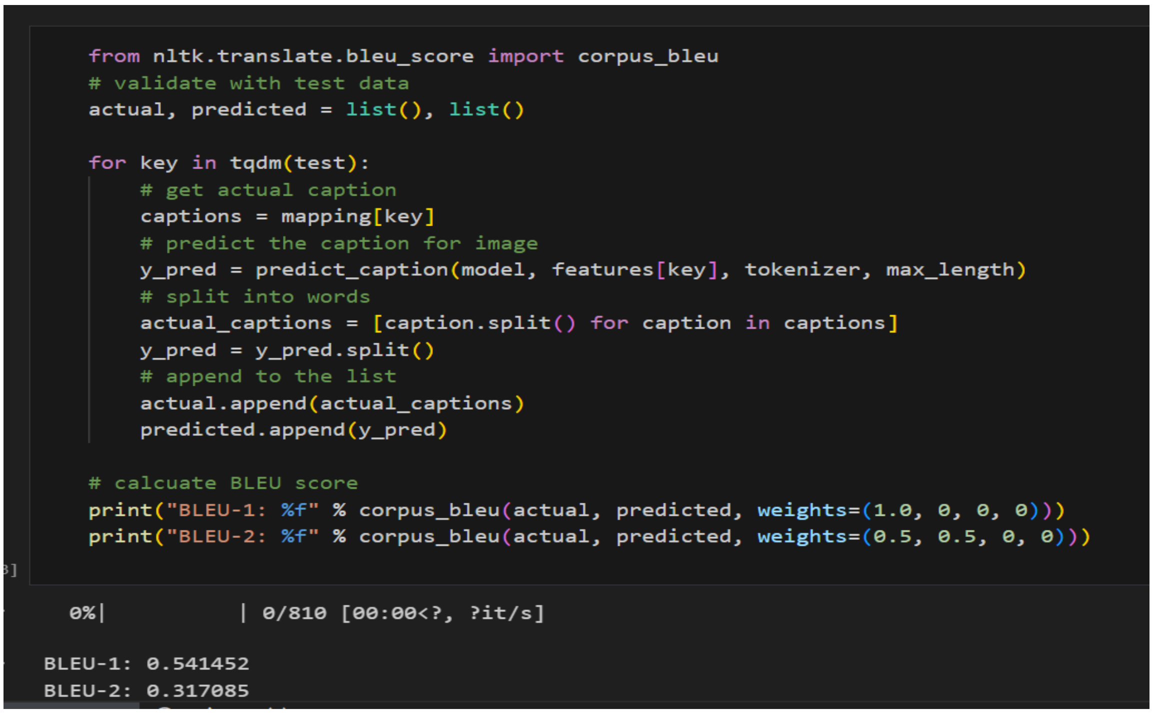
```
227/227 [==============================] - 80s 317ms/step - loss: 5.2256
227/227 [==============================] - 61s 268ms/step - loss: 4.0218
227/227 [==============================] - 62s 274ms/step - loss: 3.5998
227/227 [==============================] - 62s 274ms/step - loss: 3.3319
227/227 [==============================] - 61s 268ms/step - loss: 3.1319
227/227 [==============================] - 64s 281ms/step - loss: 2.9820
227/227 [==============================] - 62s 275ms/step - loss: 2.8642
227/227 [==============================] - 61s 269ms/step - loss: 2.7721
227/227 [==============================] - 64s 280ms/step - loss: 2.6873
227/227 [==============================] - 63s 275ms/step - loss: 2.6175
227/227 [==============================] - 64s 280ms/step - loss: 2.5535
227/227 [==============================] - 61s 269ms/step - loss: 2.4942
227/227 [==============================] - 61s 268ms/step - loss: 2.4402
```

Caption generation



```python
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text
```

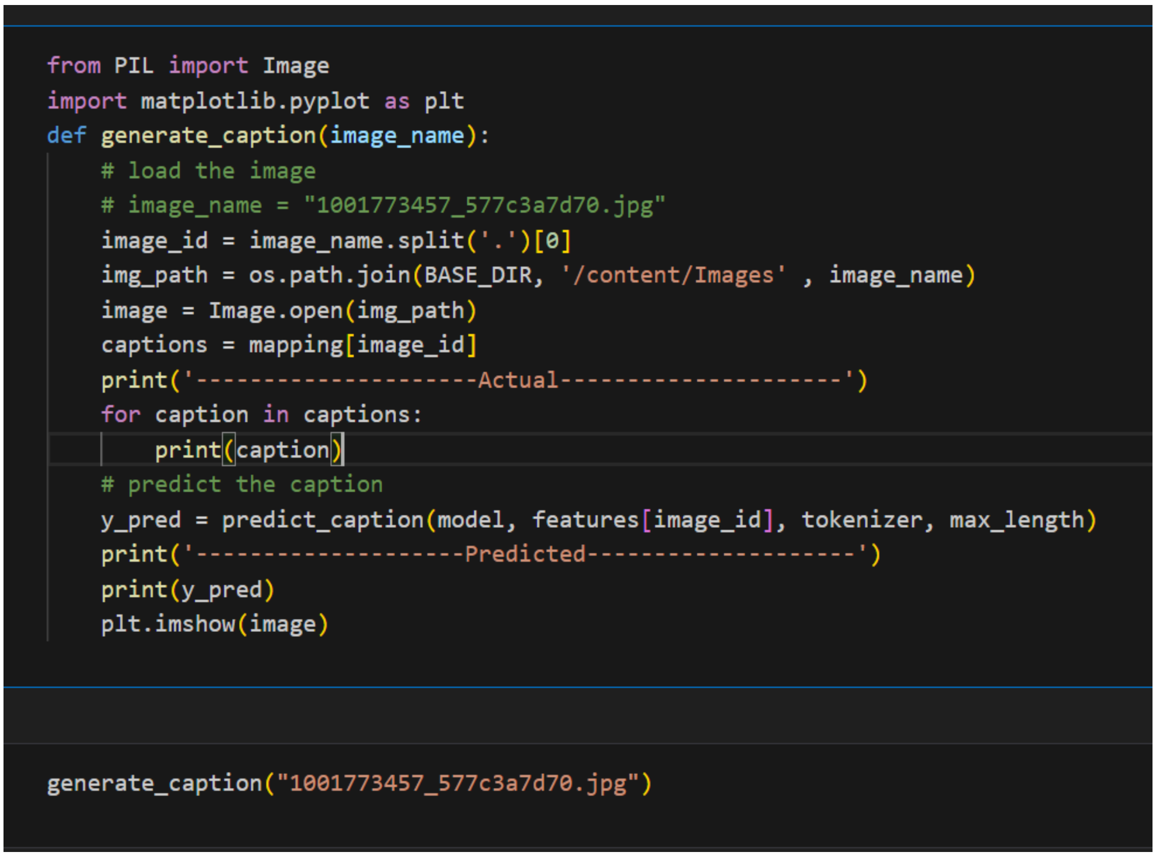
BLEU Score



```python
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
  0%|          | 0/810 [00:00<?, ?it/s]

BLEU-1: 0.541452
BLEU-2: 0.317085
```

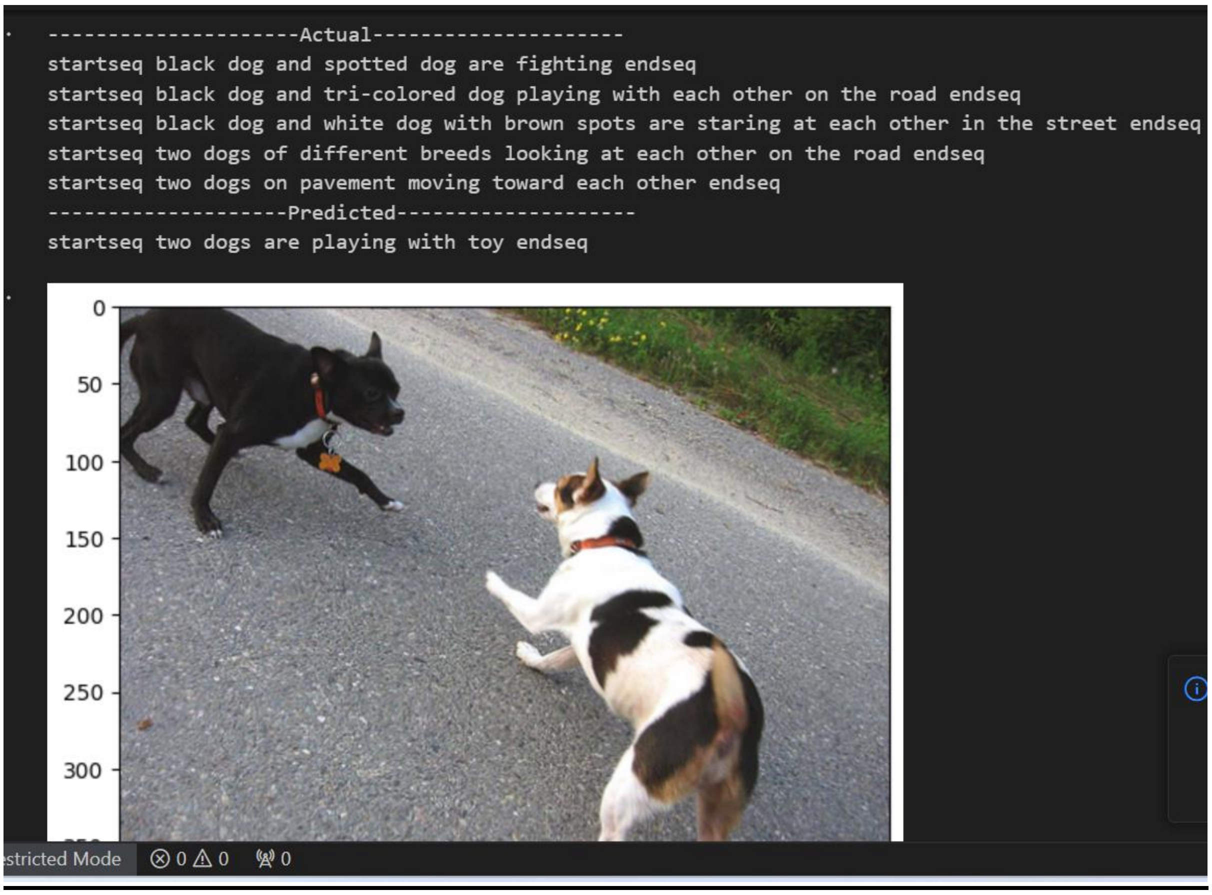
Caption generation for Image



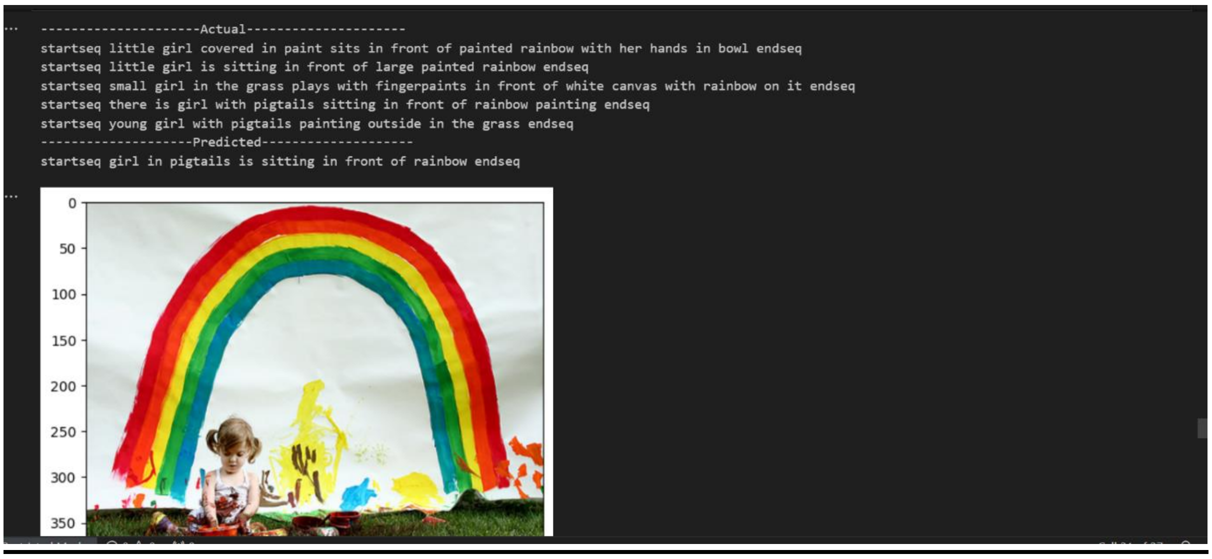
```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, '/content/Images' , image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

```
generate_caption("1001773457_577c3a7d70.jpg")
```

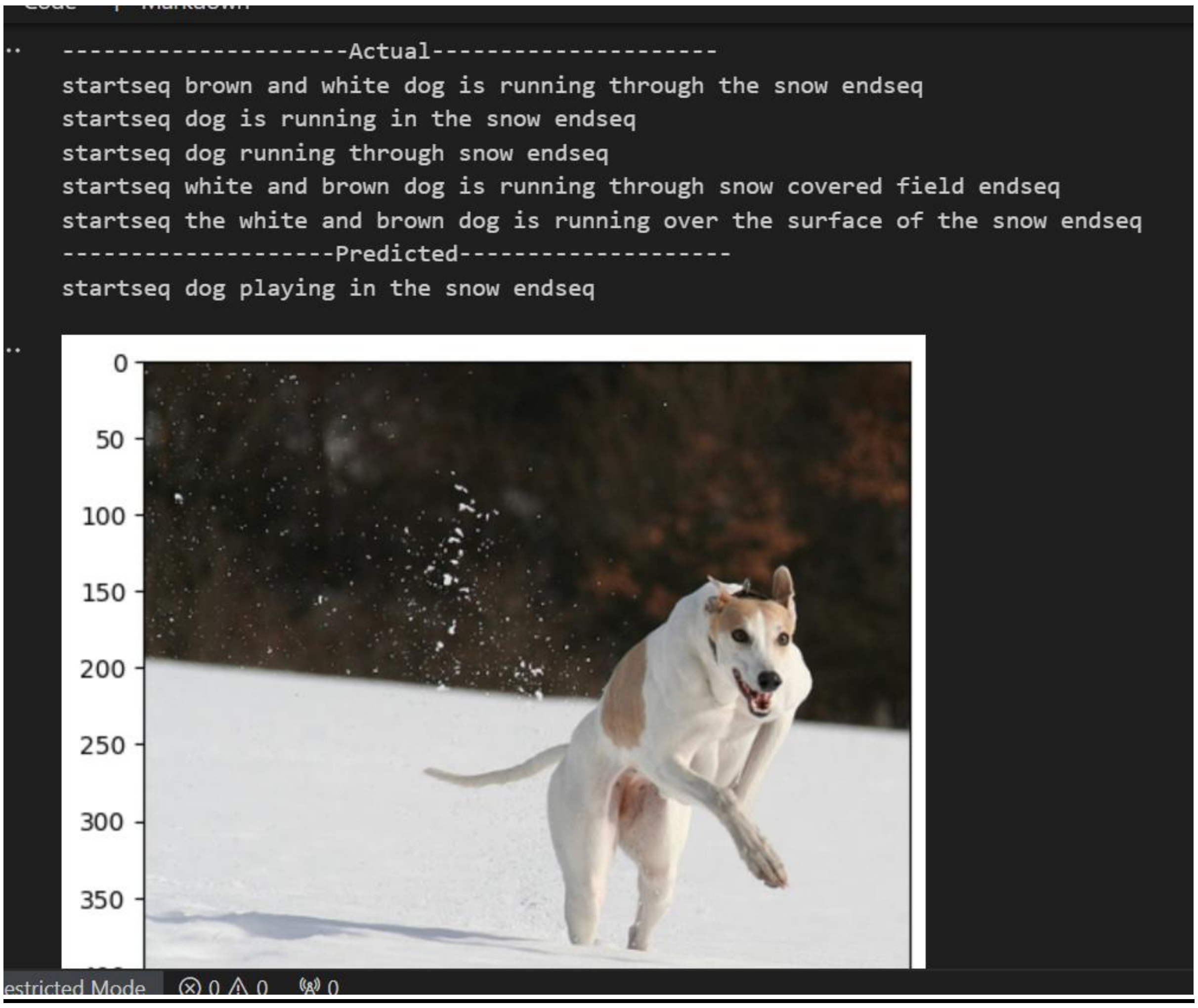
OUTPUT SCREENSHOT



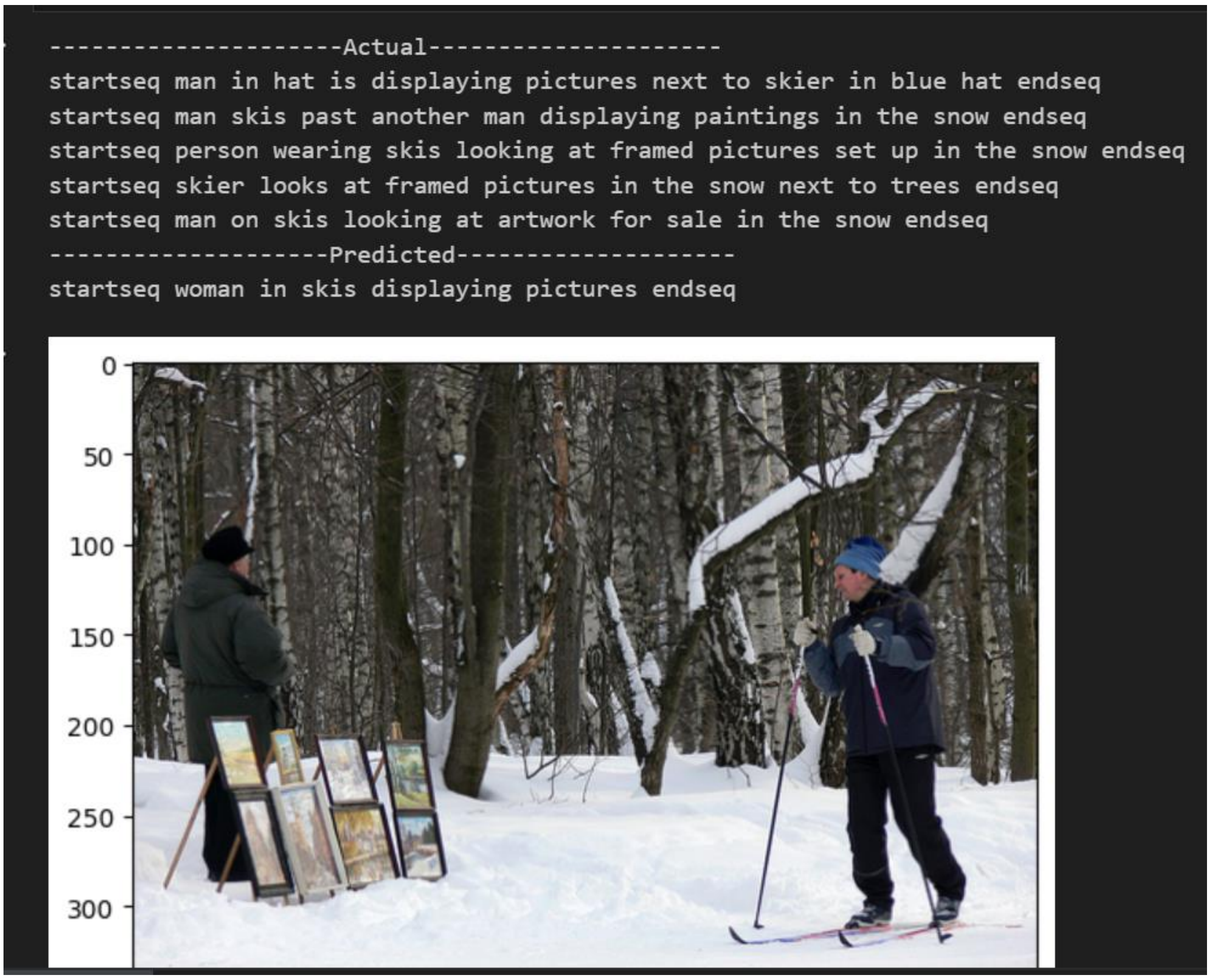
```
---------------------Actual---------------------
startseq black dog and spotted dog are fighting endseq
startseq black dog and tri-colored dog playing with each other on the road endseq
startseq black dog and white dog with brown spots are staring at each other in the street endseq
startseq two dogs of different breeds looking at each other on the road endseq
startseq two dogs on pavement moving toward each other endseq
--------------------Predicted-------------------
startseq two dogs are playing with toy endseq
```



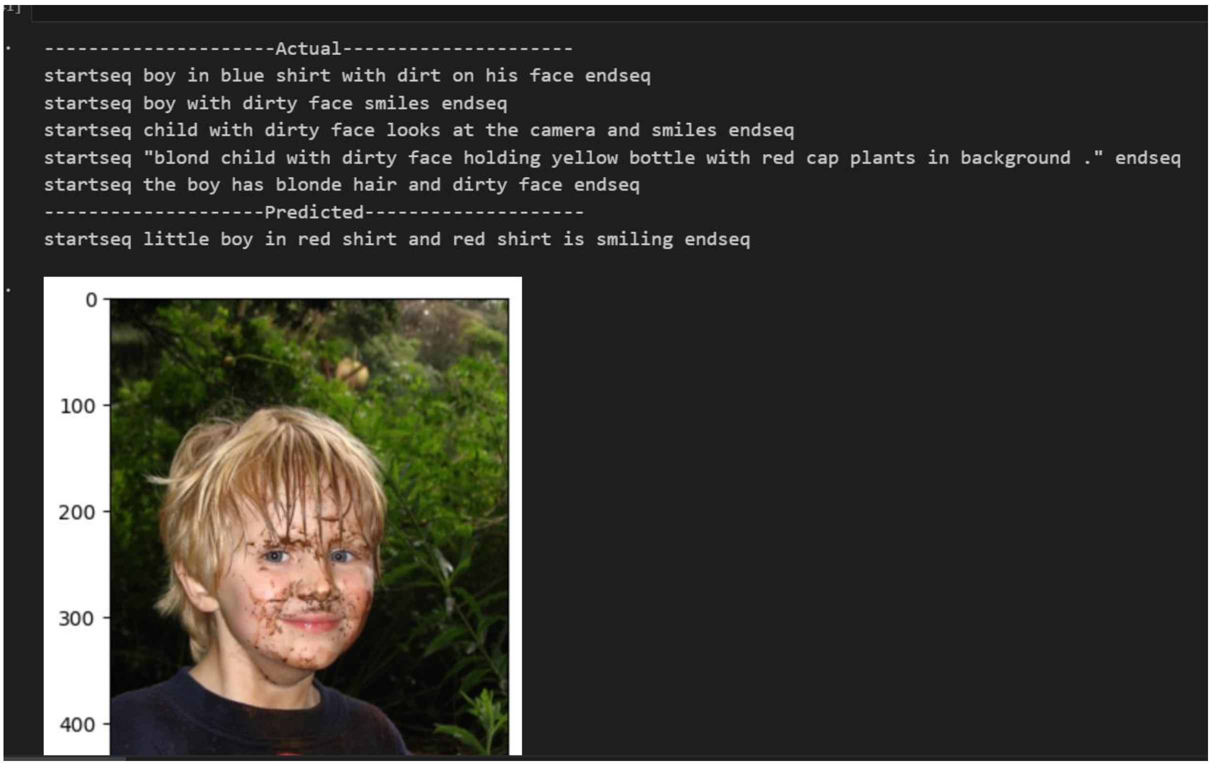
```
--------------------Actual--------------------
startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq
startseq little girl is sitting in front of large painted rainbow endseq
startseq small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it endseq
startseq there is girl with pigtails sitting in front of rainbow painting endseq
startseq young girl with pigtails painting outside in the grass endseq
-------------------Predicted-------------------
startseq girl in pigtails is sitting in front of rainbow endseq
```



```
---------------------Actual---------------------
startseq brown and white dog is running through the snow endseq
startseq dog is running in the snow endseq
startseq dog running through snow endseq
startseq white and brown dog is running through snow covered field endseq
startseq the white and brown dog is running over the surface of the snow endseq
--------------------Predicted--------------------
startseq dog playing in the snow endseq
```



```
---------------------Actual--------------------
startseq man in hat is displaying pictures next to skier in blue hat endseq
startseq man skis past another man displaying paintings in the snow endseq
startseq person wearing skis looking at framed pictures set up in the snow endseq
startseq skier looks at framed pictures in the snow next to trees endseq
startseq man on skis looking at artwork for sale in the snow endseq
--------------------Predicted-------------------
startseq woman in skis displaying pictures endseq
```



```
--------------------Actual--------------------
startseq boy in blue shirt with dirt on his face endseq
startseq boy with dirty face smiles endseq
startseq child with dirty face looks at the camera and smiles endseq
startseq "blond child with dirty face holding yellow bottle with red cap plants in background ." endseq
startseq the boy has blonde hair and dirty face endseq
-------------------Predicted-------------------
startseq little boy in red shirt and red shirt is smiling endseq
```

ALGORITHMS

1)CNN- A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

2)LSTM- Long Short-Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) that is specifically designed to handle sequential data, such as time series, speech, and text. LSTM networks are capable of learning long-term dependencies in sequential data, which makes them well suited for tasks such as language translation, speech recognition, and time series forecasting.

A traditional RNN has a single hidden state that is passed through time, which can make it difficult for the network to learn long-term dependencies. LSTMs address this problem by introducing a memory cell, which is a container that can hold information for an extended period of time. The memory cell is controlled by three gates: the input gate, the forget gate, and the output gate. These gates decide what information to add to, remove from, and output from the memory cell.

The input gate controls what information is added to the memory cell. The forget gate controls what information is removed from the memory cell. And the output gate controls what information is output from the memory cell. This allows LSTM networks to selectively retain or discard information as it flows through the network, which allows them to learn long-term dependencies.

LEARNING OUTCOMES

We have implemented a CNN-LSTM model for building an Image Caption Generator. A CNN-LSTM architecture has wide-ranging applications which include use cases in Computer Vision and Natural Language Processing domains. The CNN-LSTM model was built on the idea of generating the captions for the input pictures. This model can be used for a variety of applications. In this, we studied about the CNN model, RNN models, LSTM models, and in the end we validated that the model is generating captions for the input pictures.

CONCLUSION

We have used Flickr_8k dataset which includes nearly 8000 images, and the corresponding captions are also stored in the text file. Although deep learning -based image captioning methods have achieved a remarkable progress in recent years, a robust image captioning method that is able to generate high quality captions for nearly all images is yet to be achieved. The performance of the model can be improved by training it on a larger dataset and hyperparameter tuning. From above we can observe that unigram bleu scores favour short predictions. Prediction is good if all the BLEU scores are high.

CITATIONS

1)Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, and Tamara L Berg. Baby talk: Understanding and generating image descriptions. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35:2891–2903, June 2013.

2) Peter Young Micah Hodosh and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Artificial Intelligence Research, 47:853–899, 2013.

3) Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. International Conference on Machine Learning, 2048- 2057, 2015.

4) Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. Image captioning with semantic attention. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4651– 4659, 2016

5) R. Chauhan, K. K. Ghanshala and R. C. Joshi, “Convolutional Neural Network (CNN) for Image Detection and Recognition”. First International Conference on Secure Cyber Computing and Communication (ICSCCC), 2018