SMART VOICE ASSISTANT FOR THE BLIND USING ARTIFICIAL INTELLIGENCE

PROJECT REPORT

Submitted by

Shlok Aggarwal 20BCE0679
Rama Krishna Mohapatro 20BCE0877
Pranav Sehgal 20BCE0898
Kunal Kalra 20BCE2035
Abhijay Thakur 20BCE2375

Course Code: CSE 4015

Course Title: Human Computer Interaction

Slot: E2+TE2

Fall Semester 2022-2023

Submitted to

Prof. Deepa K, SCOPE

School of Computer Science and Engineering

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

ABSTRACT

According to WHO reports, at least 2.2 billion individuals worldwide suffer from vision impairment. Currently, with a lot of technological breakthroughs coming out regularly, the need for independent living is recognized in the situation of these 2.2 billion visually impaired people who are socially restricted. They encounter various occasions in their daily lives where they are in an unfamiliar environment and are unable to aid themselves. For them, the only way out may be to trust a random stranger who isn't necessarily reliable.

As a result, we intend to use technological advancements to assist visually impaired persons. For this purpose, we propose developing a system that uses Artificial Intelligence, Image Recognition, Machine Learning, etc. which will assist them in interacting with their surroundings. In simple words, the basic idea behind our system is to turn visual data into an alternate modality (preferably a voice assistant) suitable for blind users and we intend to use Artificial Intelligence for this conversion.

1. Introduction

Millions of people throughout the world suffer from visual impairments, making it difficult for them to comprehend their surroundings. People with vision impairments require assistance with everything from day-to-day tasks to demanding activities.

The idea of a video description generator seems like a useless one when we think of humans describing an image upon visual confirmation, it is easy to differentiate, describe and assign attributes to objects, sceneries, or acts that are a part of or result of various external factors interfering. The interactions of external factors such as a change in lighting, pose or obstruction, and other factors make it nearly impossible for machines to even participate.

But, the recent advancements in the science of Machine Learning and Artificial Intelligence have brought about Computer Vision, Concurrent and Recurrent Neural Networks, Image segmentation, and classification tools. In pursuit of achieving detailed captions, we've used Deep Learning architecture merging RNN as well as CNN to provide a SoftMax prediction to assign attributes to images and output possible detailed descriptions of the image content. The system henceforth proposed uses an isolated approach to improve the existing methods to generate the desired results.

The goal of our project is to create an assistant for the visually impaired to help them become more independent than they were previously.

2. Literature Review

[1] Deep learning methods can help generate captions for images using neural networks. Object detection Feature Extraction Creating attributes Encoder and Decoder. Applying hybrid image caption generator model can be developed in the future for more accurate captions. Time complexity can be reduced using LSTM.
[2] "CNN and RNN (LSTM) are used. CNN for identify the various features or objects that are present in the image, it will fed into the LSTM to produce the sequence of words that properly describe the image. First extracting feature vectors from the images/Feature vectors are then given the LSTM to forecast the caption.
[3] TensorFlow's object detection -API model for object detection. -OpenCV for calculating the distance of one object from the other object. Object Detection consists of object classification & localization. Distance Calculation in which the visually impaired user will be informed about the distance of the object from him/her.
[4] Regular Camera Module was replaced by the Raspberry Camera Module for faster processing. - CNN's are used for the model that in turn uses multilayer perceptrons for minimal processing. Basic idea is to take the picture using a camera and then analyze the image and provide the user with voice navigation facilities.

[5] The image captioning process is carried out using CNN, RNN, and LSTM. The images will be identified using the CNN algorithm, which is implemented in the system's hidden layers. Captions are created using the data contained in the datasets.
[6] "CNN will investigate the image for generating captions or related words and RNN will put words related to the identified parts or objects with the assistance of earlier knowledge, LSTM the modified version of RNN supports to remember the data that it trained from and connects that data with the current one. This will help to organize and label the images in an efficient manner without human intervention.
[7] A VGG-16 net CNN is used to extract feature vectors from real-time video (image frames) and an LSTM is employed to generate captions from these feature vectors. These generated captions are then converted from text to audio

3. Hardware/Software Requirements

- Parallel processing capable processor (Google Colab can be used too)
- Nvidia GTX graphics GPU
- RAM > 4 GB (preferable 8GB and above to avoid session crash)
- Python 3.5 and above
- Keras library • Installing necessary packages like speech recognition, TensorFlow etc.

4. Existing System

Currently, there are some AI models which generate the descriptions for the image but they do not process the video and generate the voice output useful for the blind.

Mainly, there are two existing primary kinds of image caption models – A method based on a statistical probability language model to generate features and a neural-network model based on an encoder-decoder language model to extract features. In deep learning, the recurrent neural network (RNN) has gotten a lot of attention. It was the first popular in the field of natural processing, and it performed well in language modeling.
With the advent and broad application of the recurrent neural network, an image description generation approach based on the encoder-decoder paradigm is proposed. The encoder in the model is a convolutional neural network, and the image features are retrieved from the last fully connected layer or convolutional layer. The decoder is a recurrent neural network that is mostly used to generate image descriptions.

Because RNN training is difficult and there is a general problem of gradient descent, which can be slightly mitigated by regularisation, RNN still has the fatal flaw of only remembering the contents of the previously limited-time unit, and LSTM is a special RNN architecture that can solve problems like gradient disappearance and has long-term memory. The LSTM network has been shown to be effective in dealing with video-related contexts in recent years. The LSTM model structure is commonly utilized in the text context decoding step, just as it is in the video context decoding stage.

4.1 Drawback/limitations of existing System/approach/method

Since we have analyzed the existing models, we have identified some drawbacks/limitations in the model which are as follows:

Video-Input: Though there are some existing models, they mainly focus on generating the output for the image data. Images are always not the only choice for generating captions. So, this is one of the main drawbacks that the models don't work on video inputs.

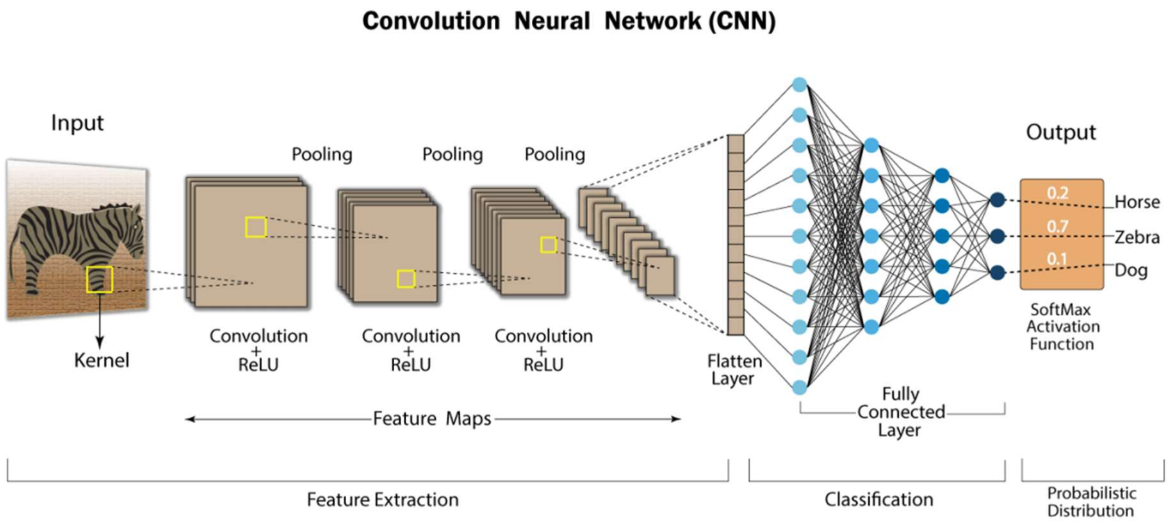
Human-like Characteristics: Despite the numerous uses of Artificial Intelligence to solve various problems, no good system exists that can demonstrate human attributes such as creative or logical reasoning, empathy, and so on.

Data-Set: The use of high-quality data drives and develops AL systems. This is why the usage of the appropriate data collection should be the first step in the AL implementation process. Because multiple types of data will be moving across an organization, deciding which data to use can be difficult.

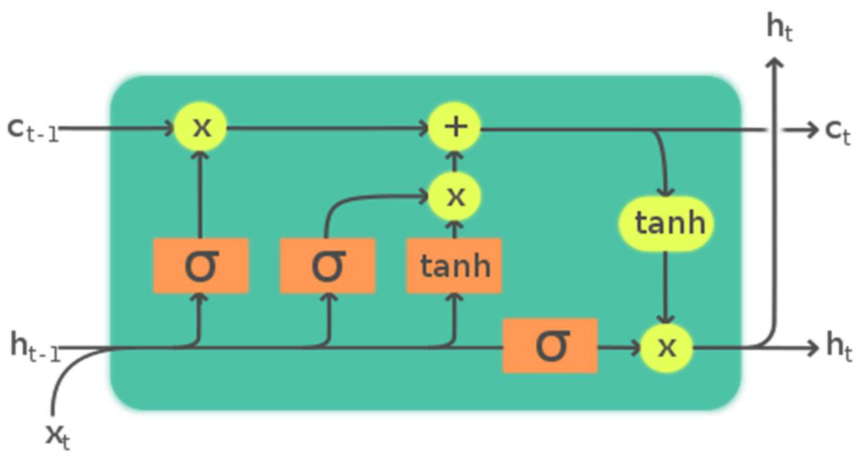
To address the problems raised above, we propose to develop a model that takes the appropriate data (images from the videos of the different environments) as input, trains a model, and then predicts the output and verbally describes it to the user.

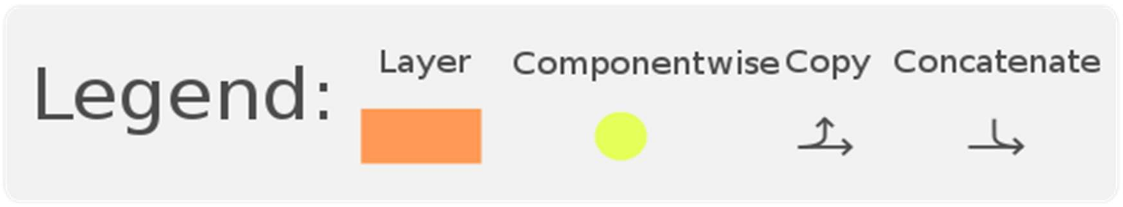
5. Proposed/ Developed Model

Convolutional neural networks (CNNs) are neural networks with one or more convolutional layers that are primarily utilized for image processing, classification, segmentation, and other autocorrelated data.



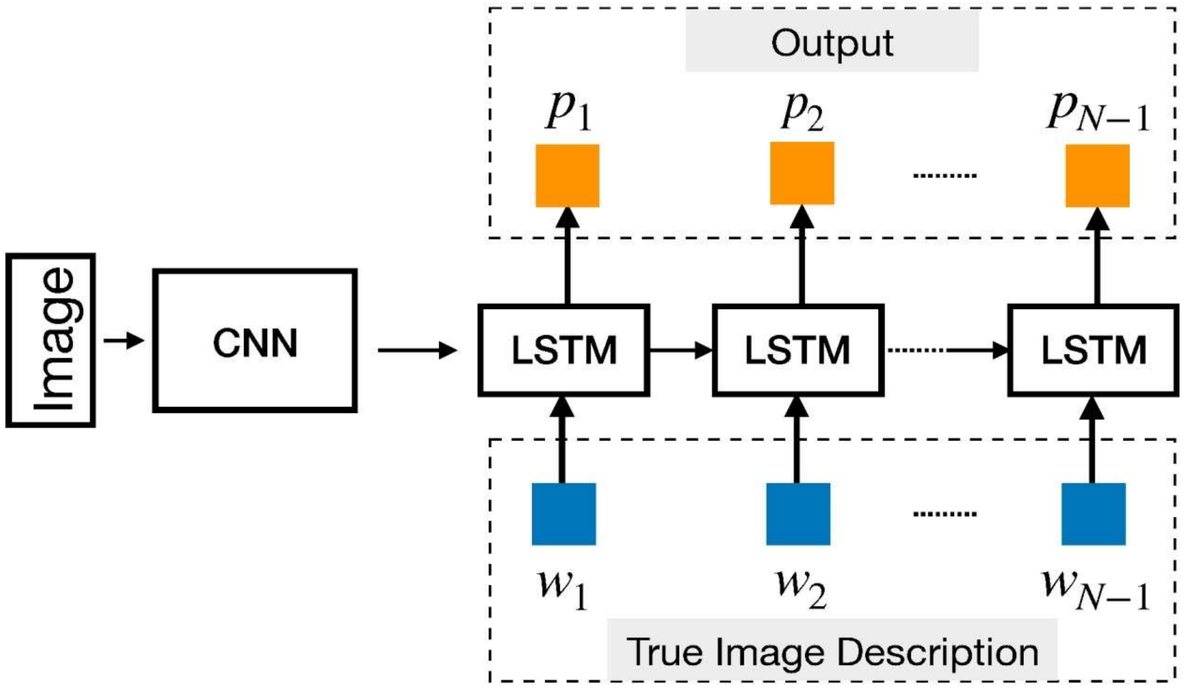
LSTM stands for Long short-term memory, it is an RNN architecture in the field of Deep Learning. It has feedback connections as it is a Recurrent neural network which means it can use bilateral process traversal whenever it requires it. It is mostly used for sequence generation.





5.1. Design:

We employed a Deep Learning architecture combining RNN and CNN to offer a SoftMax prediction to assign attributes to the given video and provide possibly extensive descriptions of the image content to obtain the needed descriptions for the blind. The system now presented takes a stand-alone approach to improving existing approaches in order to achieve the required objectives.



5.2. Module Wise Description:

- Setup the Google Colab Notebook and then mount the Google Drive for the dataset.
- We import all the necessary packages and perform data cleaning.
- The next step is to extract the feature vectors from all images.
- Loading dataset for Training the model – We will be using the Flckr8k dataset as it contains eight thousand images which proves to be a utility for better training our model.
- Tokenizing the vocabulary – mapping of a unique index to a word in the Keras library includes the tokenizer function that will be used to create tokens using our vocabulary.
- Created a Data generator function so that we can send data in batches. Sending data in batches will make sure that there is no session crash which usually happens due to less RAM.
- Defining the RNN-CNN model - we will use the Keras library which is a part of Tensorflow. It consists of three separate layers: Feature Extraction (CNN layer), Sequence Processing Layer (LSTM layer), and Predictor (Merge Layer for SoftMax prediction).
- LSTM model is been used because it takes into consideration the state of the previous cell's output and the present cell's input for the current output. This is useful while generating the descriptions for the images.
- The step involves building the LSTM model with two or three input layers and one output layer where the descriptions are generated. The model can be trained with various numbers of nodes and layers. We start with 256 and try out with 512 and 1024.
- Training the model. For this training, we employed the rectified linear activation function (ReLU), which is a linear function that outputs the input directly if it is

positive and 0 otherwise. We chose this for our model because it is easier to train and generally achieves better performance.

- Also, we have used the Adam Optimizer for CNN. This makes the algorithm reach convergence more efficiently for the given data.
- Testing the model and calculating the time taken for the given video input.
- This time includes the time to process the video, convert it into frames, and then pass one of the given frames to the model so that the model can predict the description and provide the audio output.

5.3. Implementation:

We started by taking the video input file and converting it into image frames. We've handed one of these frames to the image description function, which predicts the description using the trained model.

This trained model consists of three subparts:

Feature Extractor for the image: This is a pre-trained 16-layer VGG model on the ImageNet dataset. We used the extracted features predicted by this model as input after preprocessing the images with the VGG model (minus the output layer). The Feature Extractor model expects a vector of 4,096 elements as input image features. A Dense layer transforms these into a 256-element representation of the image.

Sequence Processing: This is a word embedding layer for handling text input, followed by a recurrent neural network layer using Long Short-Term Memory (LSTM). The Sequence Processor model expects input sequences of a predetermined length (34 words) to be fed into an Embedding layer that ignores padding values using a mask. After that, there's an LSTM layer with 256 memory units.

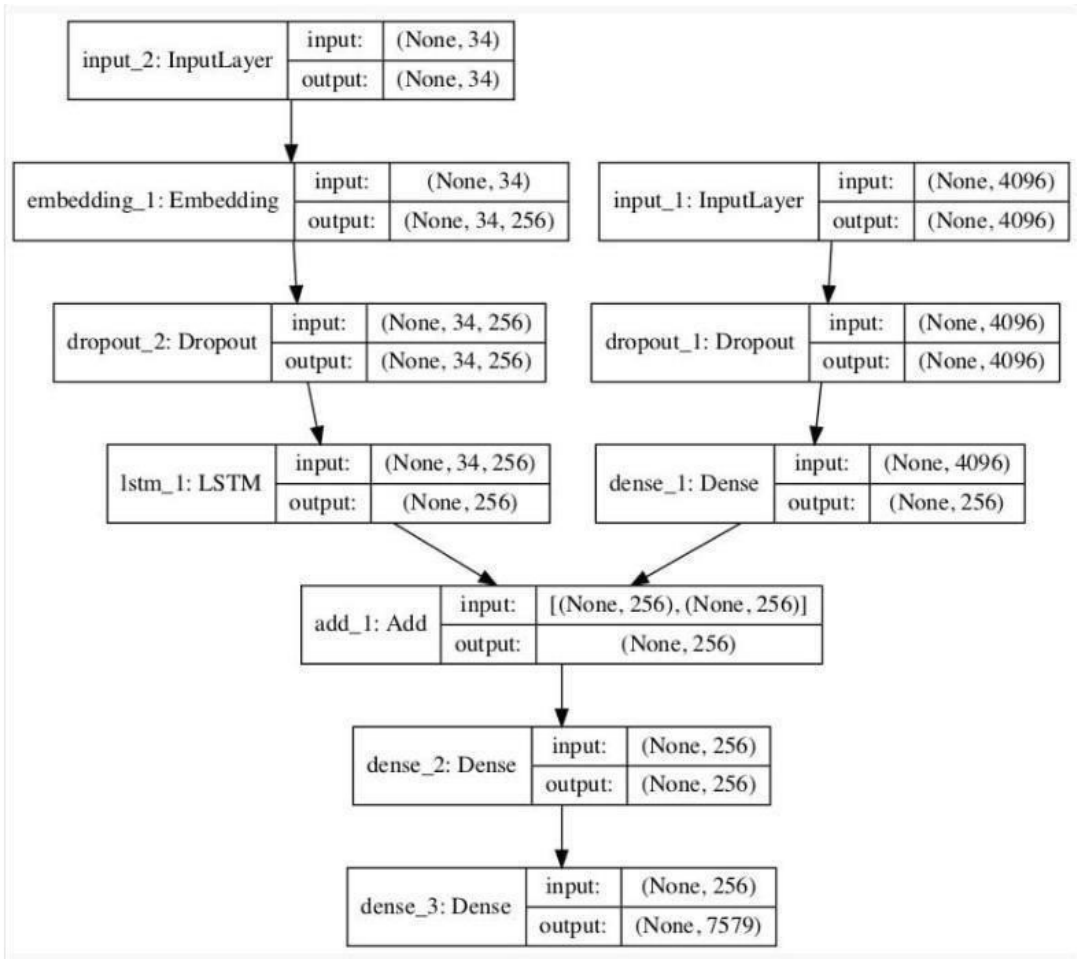
Predictor: A fixed-length vector is produced by both the feature extractor and the sequence processor. To create a final forecast, these are combined and processed by a Dense layer. The Predictor model uses an addition operation to combine the vectors from both input models. This is then sent to a Dense 256 neuron layer, and finally to a final output Dense layer, which generates a SoftMax prediction for the next word in the sequence over the whole output vocabulary.

Working of the model:

The model learns quickly and overfits the training data soon. As a result, we'll keep an eye on the trained model's performance on the holdout development dataset. We will save the entire model to a file when the model's skill on the development dataset improves at the end of an epoch.

We'll test a model by creating descriptions for all of the images in the test dataset and using a typical cost function to evaluate the predictions. This entails giving in the start description token 'starting,' creating one word, and then recursively invoking the model with generated words as input until the end of the sequence token 'ending,' or the maximum description length is reached.

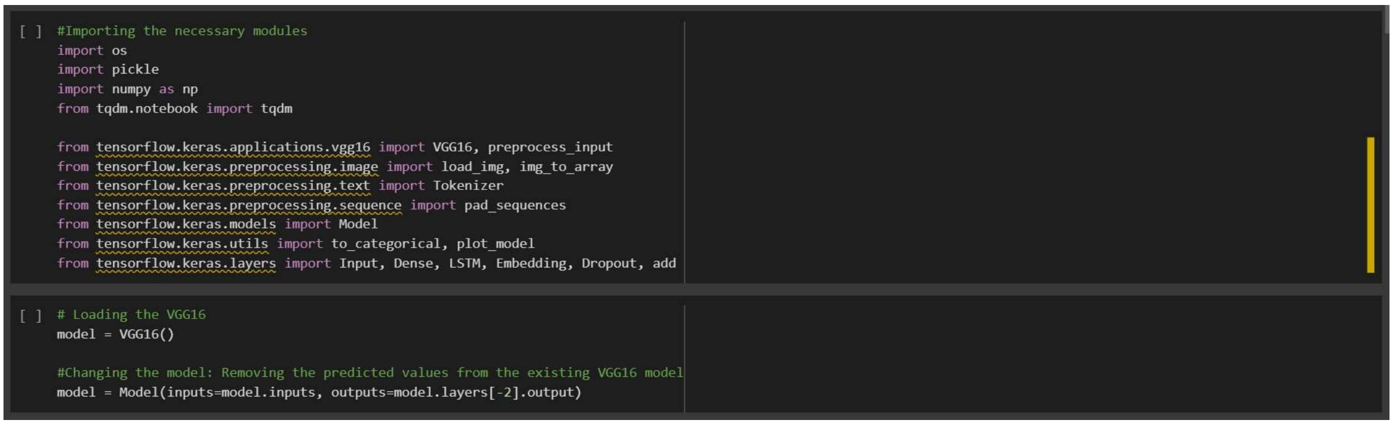
Structure of the model can be visualized as follows:



input_2: InputLayer | input: (None, 34) | output: (None, 34)
embedding_1: Embedding | input: (None, 34) | output: (None, 34, 256)
input_1: InputLayer | input: (None, 4096) | output: (None, 4096)
dropout_2: Dropout | input: (None, 34, 256) | output: (None, 34, 256)
dropout_1: Dropout | input: (None, 4096) | output: (None, 4096)
lstm_1: LSTM | input: (None, 34, 256) | output: (None, 256)
dense_1: Dense | input: (None, 4096) | output: (None, 256)
add_1: Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_2: Dense | input: (None, 256) | output: (None, 256)
dense_3: Dense | input: (None, 256) | output: (None, 7579)

6. Results and Discussions

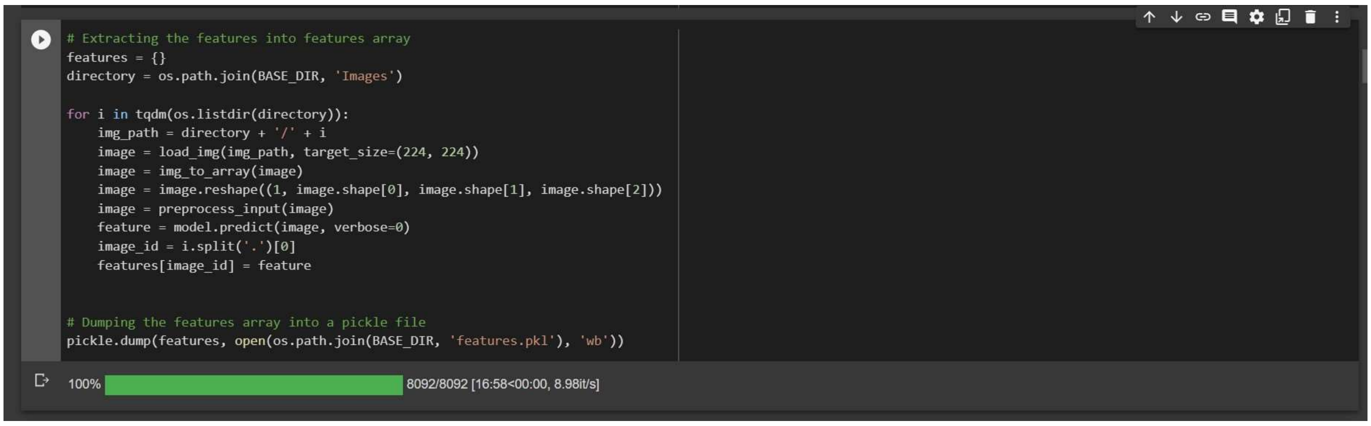
- Importing the modules and loading the VGG16() model



```
[ ] #Importing the necessary modules
    import os
    import pickle
    import numpy as np
    from tqdm.notebook import tqdm

    from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
    from tensorflow.keras.preprocessing.image import load_img, img_to_array
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    from tensorflow.keras.models import Model
    from tensorflow.keras.utils import to_categorical, plot_model
    from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

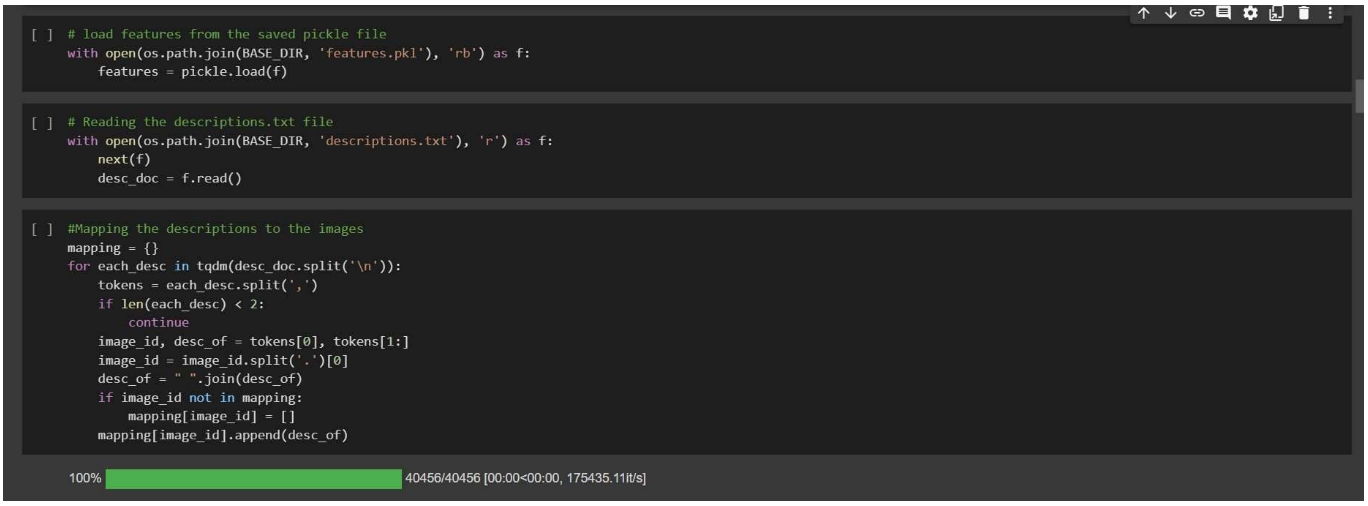
```
[ ] # Loading the VGG16
    model = VGG16()

    #Changing the model: Removing the predicted values from the existing VGG16 model
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

- Extracting features, loading descriptions, and mapping the images with descriptions



```
# Extracting the features into features array
features = {}
directory = os.path.join(BASE_DIR, 'Images')

for i in tqdm(os.listdir(directory)):
    img_path = directory + '/' + i
    image = load_img(img_path, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    feature = model.predict(image, verbose=0)
    image_id = i.split('.')[0]
    features[image_id] = feature


# Dumping the features array into a pickle file
pickle.dump(features, open(os.path.join(BASE_DIR, 'features.pkl'), 'wb'))
```

```
100% 8092/8092 [16:58<00:00, 8.98it/s]
```



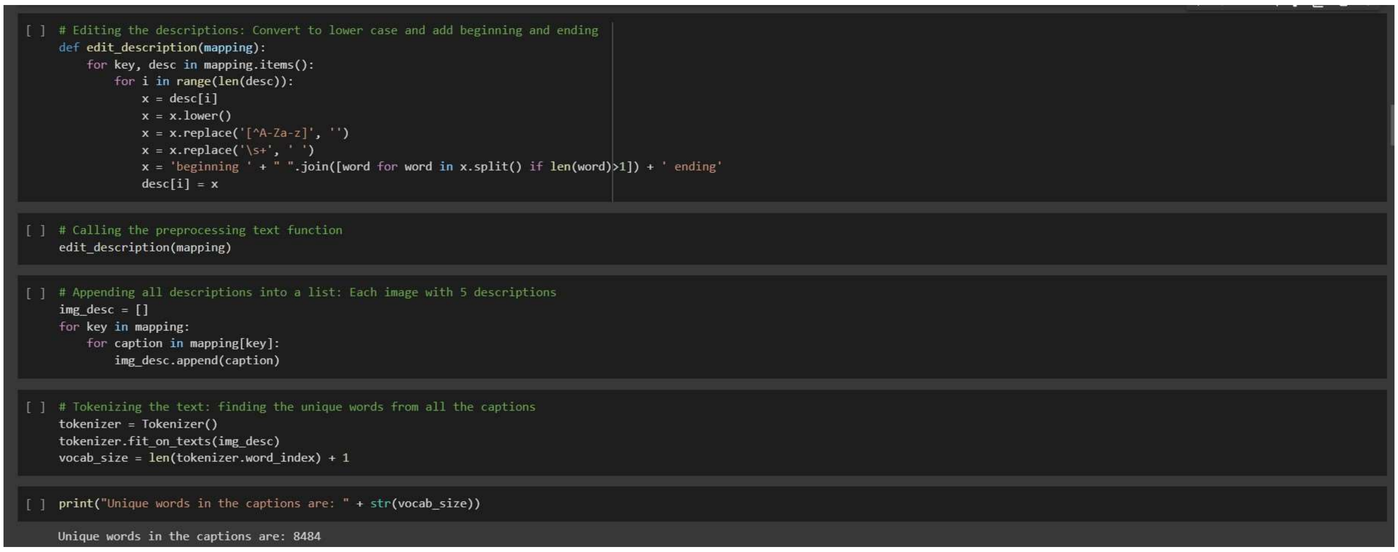
```
[ ] # load features from the saved pickle file
    with open(os.path.join(BASE_DIR, 'features.pkl'), 'rb') as f:
        features = pickle.load(f)
```

```
[ ] # Reading the descriptions.txt file
    with open(os.path.join(BASE_DIR, 'descriptions.txt'), 'r') as f:
        next(f)
        desc_doc = f.read()
```

```
[ ] #Mapping the descriptions to the images
    mapping = {}
    for each_desc in tqdm(desc_doc.split('\n')):
        tokens = each_desc.split(',')
        if len(each_desc) < 2:
            continue
        image_id, desc_of = tokens[0], tokens[1:]
        image_id = image_id.split('.')[0]
        desc_of = " ".join(desc_of)
        if image_id not in mapping:
            mapping[image_id] = []
        mapping[image_id].append(desc_of)
```

```
100% 40456/40456 [00:00<00:00, 175435.11it/s]
```

- After mapping, we edit the descriptions: convert them into lower case, remove special characters, etc. and add ‘beginning’ and ‘ending’ tags and later tokenize them i.e., find the unique words.



```
[ ] # Editing the descriptions: Convert to lower case and add beginning and ending
    def edit_description(mapping):
        for key, desc in mapping.items():
            for i in range(len(desc)):
                x = desc[i]
                x = x.lower()
                x = x.replace('[^A-Za-z]', '')
                x = x.replace('\s+', ' ')
                x = 'beginning ' + " ".join([word for word in x.split() if len(word)>1]) + ' ending'
                desc[i] = x
```

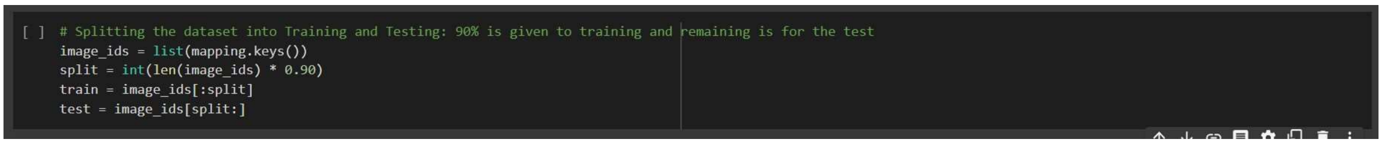
```
[ ] # Calling the preprocessing text function
    edit_description(mapping)
```

```
[ ] # Appending all descriptions into a list: Each image with 5 descriptions
    img_desc = []
    for key in mapping:
        for caption in mapping[key]:
            img_desc.append(caption)
```

```
[ ] # Tokenizing the text: finding the unique words from all the captions
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(img_desc)
    vocab_size = len(tokenizer.word_index) + 1
```

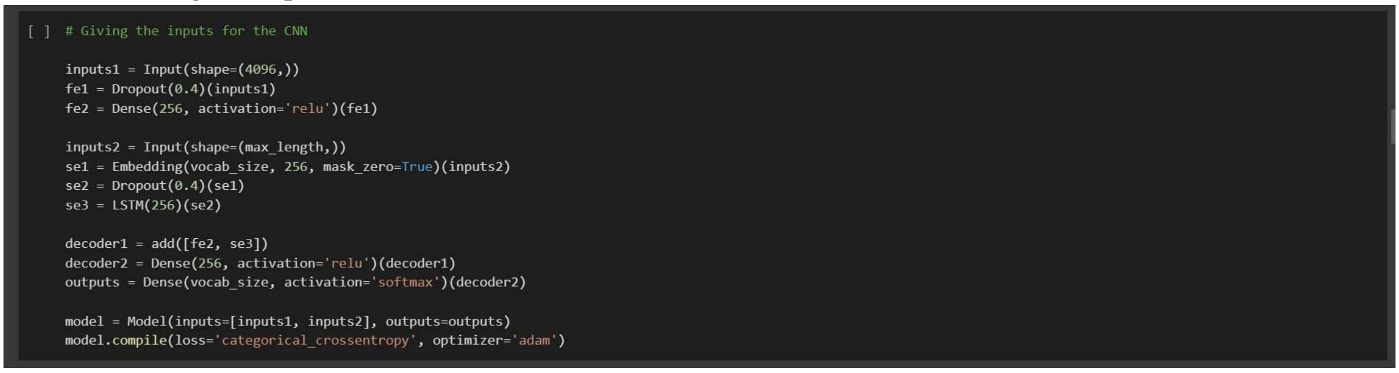
```
[ ] print("Unique words in the captions are: " + str(vocab_size))

    Unique words in the captions are: 8484
```

- Splitting the dataset into training and testing (90% into training and 10% into test data)



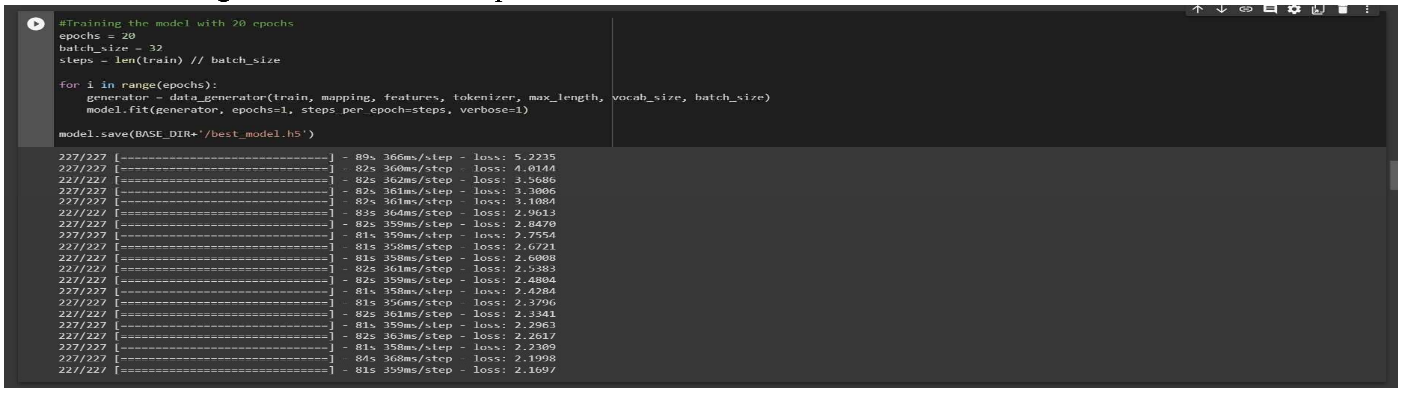
```
[ ] # Splitting the dataset into Training and Testing: 90% is given to training and remaining is for the test
    image_ids = list(mapping.keys())
    split = int(len(image_ids) * 0.90)
    train = image_ids[:split]
    test = image_ids[split:]
```

- Giving the inputs for the CNN.



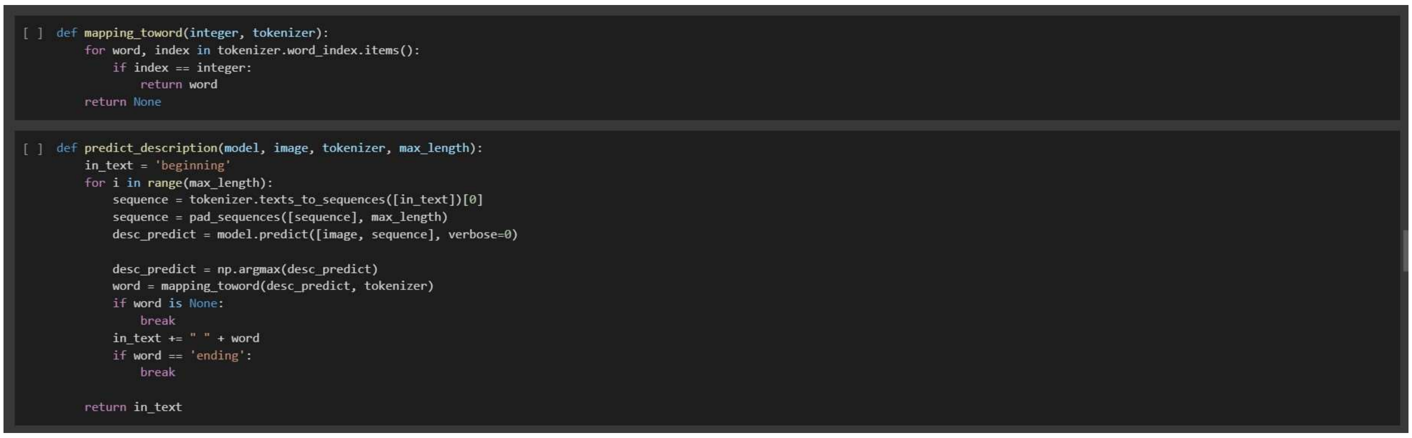
```
[ ] # Giving the inputs for the CNN

    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.4)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.4)(se1)
    se3 = LSTM(256)(se2)

    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')
```

- Training the model with 20 epochs



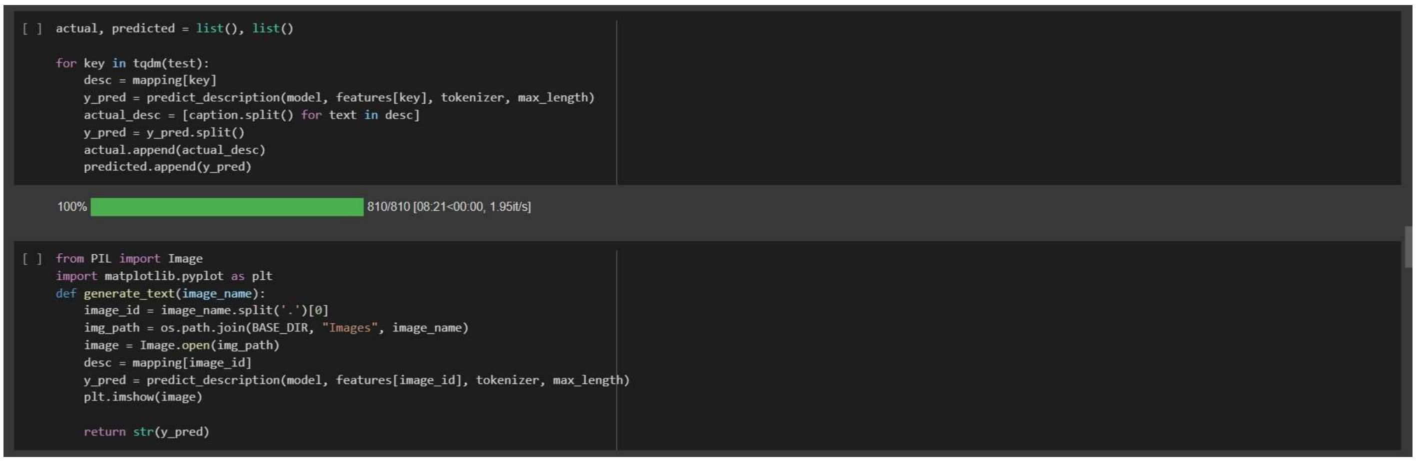
```
#Training the model with 20 epochs
epochs = 20
batch_size = 32
steps = len(train) // batch_size

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

model.save(BASE_DIR+'/best_model.h5')

227/227 [==============================] - 89s 366ms/step - loss: 5.2235
227/227 [==============================] - 82s 360ms/step - loss: 4.0144
227/227 [==============================] - 82s 362ms/step - loss: 3.5686
227/227 [==============================] - 82s 361ms/step - loss: 3.3006
227/227 [==============================] - 82s 361ms/step - loss: 3.1084
227/227 [==============================] - 83s 364ms/step - loss: 2.9613
227/227 [==============================] - 82s 359ms/step - loss: 2.8470
227/227 [==============================] - 81s 359ms/step - loss: 2.7554
227/227 [==============================] - 81s 358ms/step - loss: 2.6721
227/227 [==============================] - 81s 358ms/step - loss: 2.6008
227/227 [==============================] - 82s 361ms/step - loss: 2.5383
227/227 [==============================] - 82s 359ms/step - loss: 2.4804
227/227 [==============================] - 81s 358ms/step - loss: 2.4284
227/227 [==============================] - 81s 356ms/step - loss: 2.3796
227/227 [==============================] - 82s 361ms/step - loss: 2.3341
227/227 [==============================] - 81s 359ms/step - loss: 2.2963
227/227 [==============================] - 82s 363ms/step - loss: 2.2617
227/227 [==============================] - 81s 358ms/step - loss: 2.2309
227/227 [==============================] - 84s 368ms/step - loss: 2.1998
227/227 [==============================] - 81s 359ms/step - loss: 2.1697
```

- Functions to predict and generate the descriptions for the image



```python
def mapping_toword(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

```python
def predict_description(model, image, tokenizer, max_length):
    in_text = 'beginning'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], max_length)
        desc_predict = model.predict([image, sequence], verbose=0)

        desc_predict = np.argmax(desc_predict)
        word = mapping_toword(desc_predict, tokenizer)
        if word is None:
            break
        in_text += " " + word
        if word == 'ending':
            break

    return in_text
```

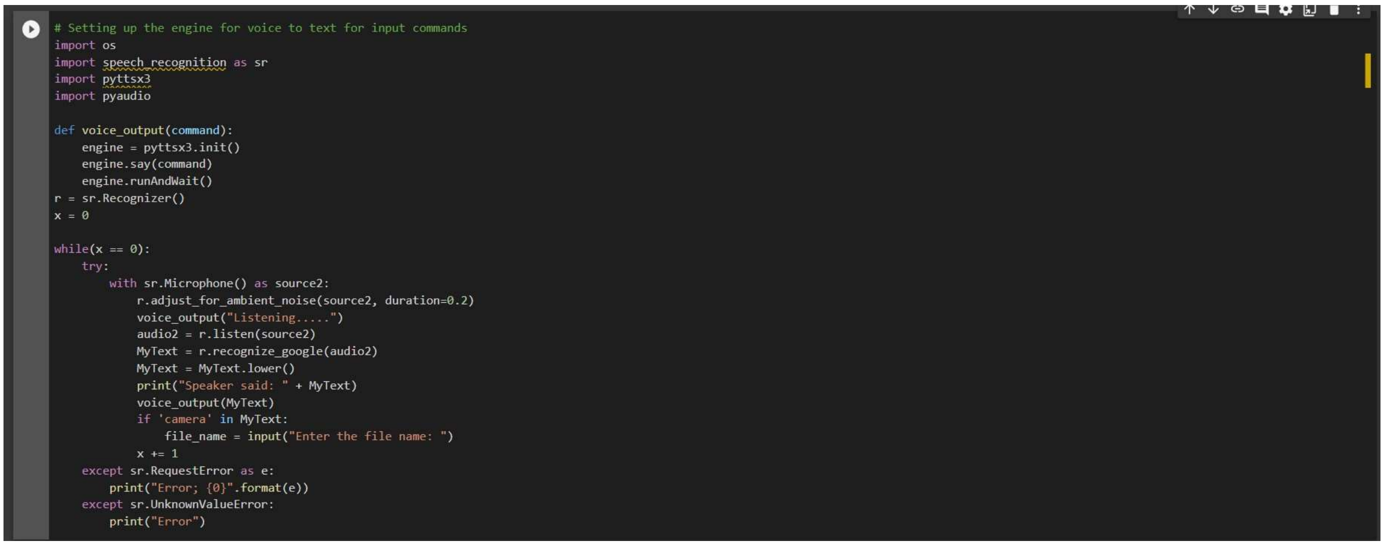


```python
actual, predicted = list(), list()

for key in tqdm(test):
    desc = mapping[key]
    y_pred = predict_description(model, features[key], tokenizer, max_length)
    actual_desc = [caption.split() for text in desc]
    y_pred = y_pred.split()
    actual.append(actual_desc)
    predicted.append(y_pred)
```

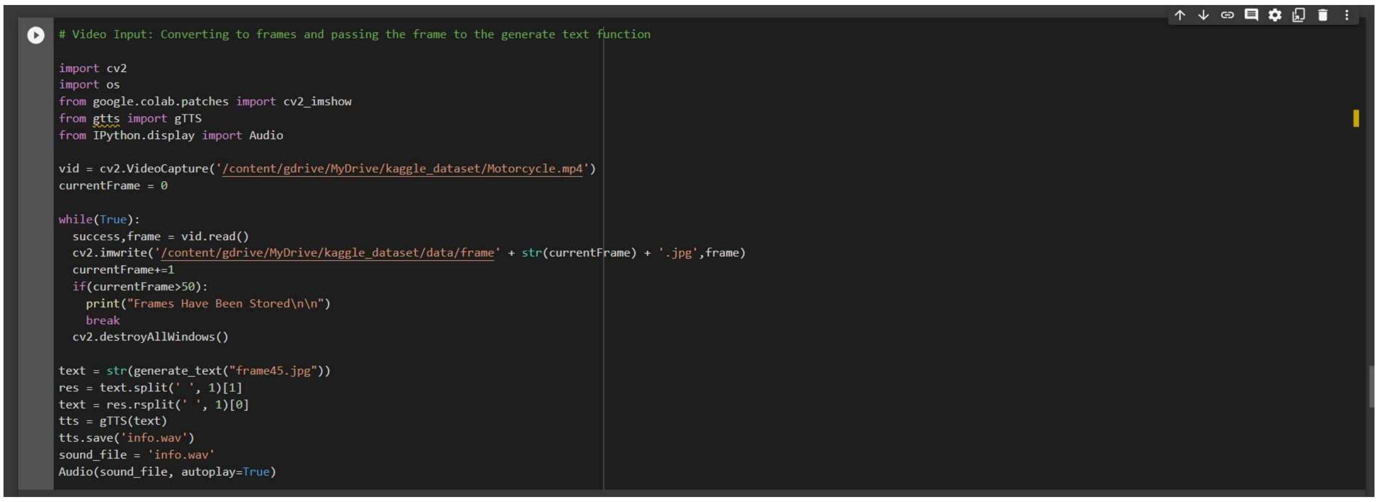
```
100% 810/810 [08:21<00:00, 1.95it/s]
```

```python
from PIL import Image
import matplotlib.pyplot as plt
def generate_text(image_name):
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    desc = mapping[image_id]
    y_pred = predict_description(model, features[image_id], tokenizer, max_length)
    plt.imshow(image)

    return str(y_pred)
```

- Function to take the voice input from the user which is the name of the video file which needs to be processed to generate the description.

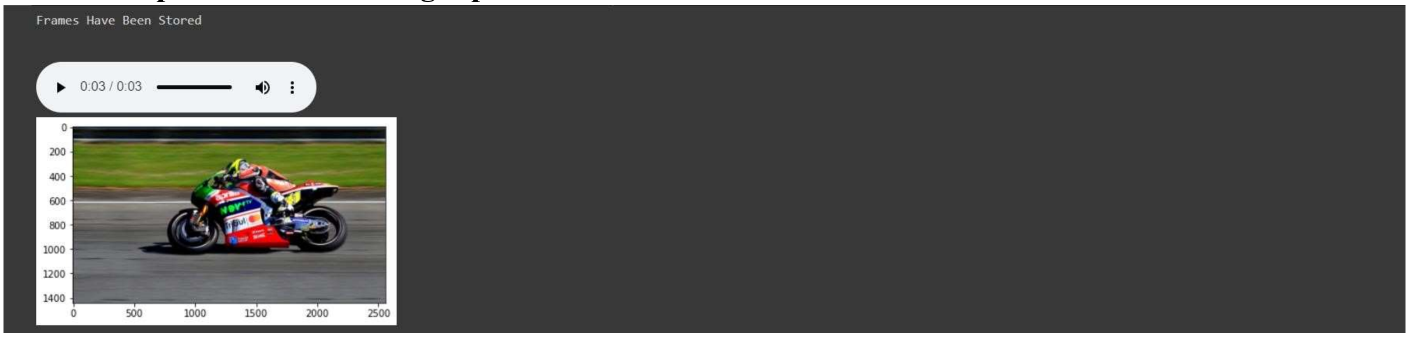


```python
# Setting up the engine for voice to text for input commands
import os
import speech_recognition as sr
import pyttsx3
import pyaudio

def voice_output(command):
    engine = pyttsx3.init()
    engine.say(command)
    engine.runAndWait()
r = sr.Recognizer()
x = 0

while(x == 0):
    try:
        with sr.Microphone() as source2:
            r.adjust_for_ambient_noise(source2, duration=0.2)
            voice_output("Listening.....")
            audio2 = r.listen(source2)
            MyText = r.recognize_google(audio2)
            MyText = MyText.lower()
            print("Speaker said: " + MyText)
            voice_output(MyText)
            if 'camera' in MyText:
                file_name = input("Enter the file name: ")
            x += 1
    except sr.RequestError as e:
        print("Error; {0}".format(e))
    except sr.UnknownValueError:
        print("Error")
```



```python
# Video Input: Converting to frames and passing the frame to the generate text function

import cv2
import os
from google.colab.patches import cv2_imshow
from gtts import gTTS
from IPython.display import Audio

vid = cv2.VideoCapture('/content/gdrive/MyDrive/kaggle_dataset/Motorcycle.mp4')
currentFrame = 0

while(True):
  success,frame = vid.read()
  cv2.imwrite('/content/gdrive/MyDrive/kaggle_dataset/data/frame' + str(currentFrame) + '.jpg',frame)
  currentFrame+=1
  if(currentFrame>50):
    print("Frames Have Been Stored\n\n")
    break
  cv2.destroyAllWindows()

text = str(generate_text("frame45.jpg"))
res = text.split(' ', 1)[1]
text = res.rsplit(' ', 1)[0]
tts = gTTS(text)
tts.save('info.wav')
sound_file = 'info.wav'
Audio(sound_file, autoplay=True)
```

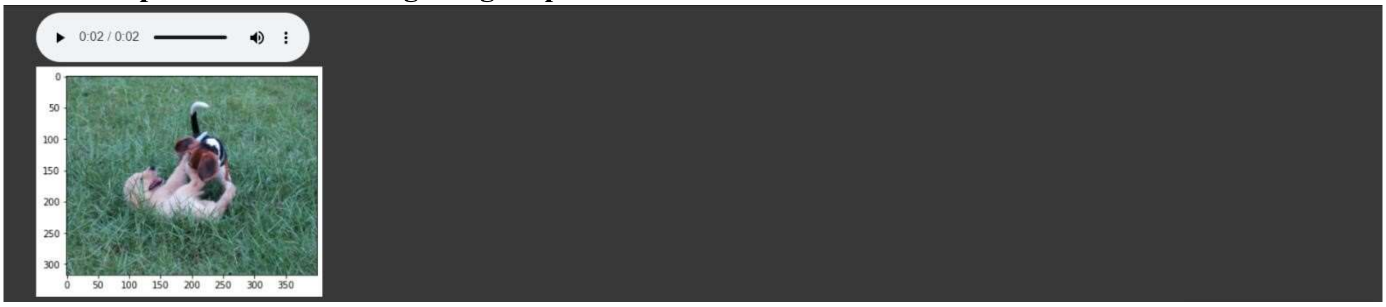
Output for the following input



```
Frames Have Been Stored
```

Output says: "Motorcyclist wearing silver helmet rides his bike". **(Audio output)**

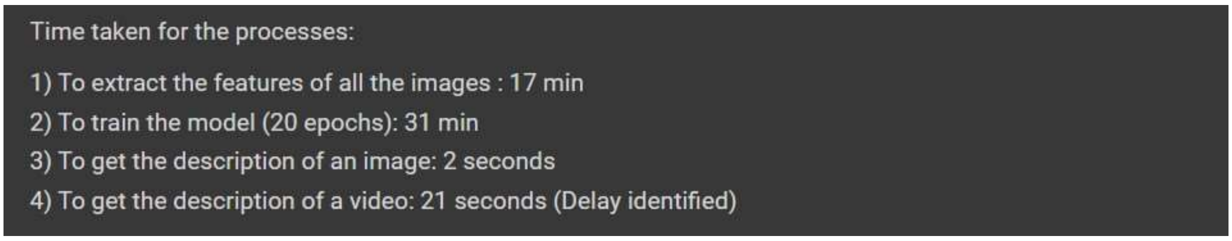
Output for the following image input



Output says: "Two dogs playing in the grass". **(Audio output)**

Given below is the analysis of our model which is the time taken for the major steps in the code and this will help us to understand how long it takes to pre-process the image, extract features, train the model, and most importantly the time taken to produce the audio output given the video input or the image input.

Time taken by the processes to produce the output:



```
Time taken for the processes:

1) To extract the features of all the images : 17 min
2) To train the model (20 epochs): 31 min
3) To get the description of an image: 2 seconds
4) To get the description of a video: 21 seconds (Delay identified)
```

Kaggle Link for the Code:

https://www.kaggle.com/code/pranavsehga1/smart-voice-assistant-for-the-blind

7. Conclusion

The study taught us about LSTM sequencing algorithms that have been generalized for sequence production in the industry, as well as the CNN model for visual imagery-based operations. The combination of the two neural networks resulted in a very practical program. Although the model's implementation was time-intensive, the repetitive changing of the hyperparameters to generate and tune the ideal captions was thought-provoking. The model is highly useful for the provided testing photos, and it also has a lot of room for improvement in the future.

8. Future Work

The future work includes implementing this ML model with help of a hardware device for blind people. The hardware can be a smart spectacle that has video capturing and audio output facilities. We wish to enhance this project by directly processing the video so that we get the output directly from the video. Also, the model has a bit of low accuracy as the caption does not accurately describe the image. Due to limited processing power and other constraints, we only ran for a few epochs. If we run a few more epochs, the model improves, producing better output that accurately characterizes the features in the input image. So, we wish to do the changes in the near future and will try our best to enhance the project.

9. References

[1] Kumar, N. Komal; Vigneswari, D.; Mohan, A.; Laxman, K.; Yuvaraj, J. (2019). [IEEE 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS) - Coimbatore, India (2019.3.15-2019.3.16)] 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS) - Detection and Recognition of Objects in Image Caption Generator System: A Deep Learning Approach. , (), 107–109.

[2] Mohana Priya R;Maria Anu;Divya S; (2021). Building A Voice Based Image Caption Generator with Deep Learning . 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)

[3] Chharia, A., & Upadhyay, R. (2020). Deep Recurrent Architecture based Scene Description Generator for Visually Impaired. 2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).

[4] Sarathi, V., Mujumdar, A., & Naik, D. (2021, April). Effect of Batch Normalization and Stacked LSTMs on Video Captioning. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 820-825). IEEE.

[5] Hossain, M. Z., Sohel, F., Shiratuddin, M. F., &amp; Laga, H. (2019),“A comprehensive survey of deep learning for image captioning”, ACM Computing Surveys (CSUR), 51(6), 1-36

[6] “A survey on vision-based human action recognition,” Image Vision Comput, vol. 28, pp. 976–990, 2011. R.Poppe, “A survey on vision-based human action recognition,” Image Vision Comput, vol. 28, pp. 976–990, 2011

[7] Chharia, A., & Upadhyay, R. (2020). Deep Recurrent Architecture based Scene Description Generator for Visually Impaired. 2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).

[8] Sarathi, V., Mujumdar, A., & Naik, D. (2021, April). Effect of Batch Normalization and Stacked LSTMs on Video Captioning. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 820-825). IEEE.

[9] HafeezAlia, Sanjeev U. Rao, Swaroop Ranganath, T.S.Ashwin, Guddeti Ram Mohana ReddyA “Google Glass Based Real-Time Scene Analysis for the Visually Impaired”, a. (13/12/2021)

[10] Ashwani Kumar, Ankush Chourasia,”International Research Journal of Engineering and Technology (IRJET)”(March 2018)

[11] Ajinkya Badave, Rathin Jagtap, Rizina Kaovasia, Shivani Rahatwad, Saroja Kulkarni,”2020 International Conference on Industry 4.0 Technology (I4Tech)”(Feb 13-15, 2020)

[12] HaoranWang , Yue Zhang, and Xiaosheng Yu, “An Overview of Image Caption Generation Methods”, (CIN-2020)

[13] B.Krishnakumar, K.Kousalya, S.Gokul, R.Karthikeyan, and D.Kaviyarasu, “IMAGE CAPTION GENERATOR USING DEEP LEARNING”, (international Journal of Advanced Science and Technology- 2020)

[14] MD. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga, “A Comprehensive Survey of Deep Learning for Image Captioning” ,(ACM-2019) VII. Rehab Alahmadi, Chung Hyuk Park, and James Hahn, “Sequence-tosequence image caption generator”, (ICMV-2018)

[15] Oriol Vinyals, Alexander Toshev, SamyBengio, and Dumitru Erhan, “Show and Tell: A Neural Image Caption Generator”,(CVPR 1, 2- 2015)

Website for reference:

I. **https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add**
II. **https://www.geeksforgeeks.org/convert-text-speech-python/**
III. **https://www.nbshare.io/notebook/249468051/How-To-Code-RNN-andLSTM-Neural-Networks-in-Python/**