Develop a Deep Learning Model to find the Caption or Description of an Image given an Input Image.

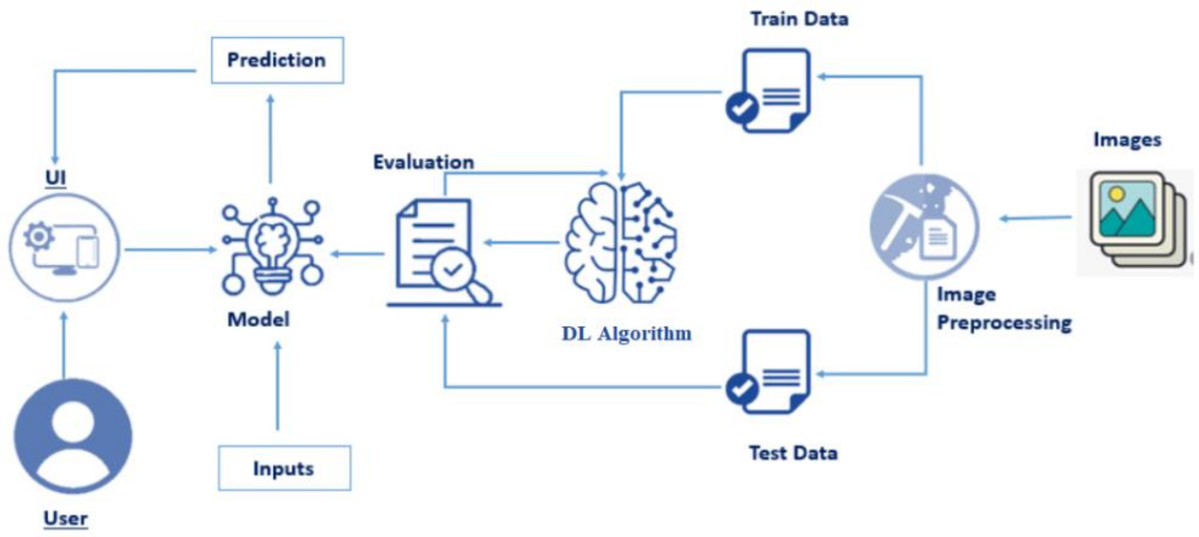
Introduction:

An image caption description is a written caption that explains the important details of a picture. It involves generating a human readable textual description given an image, such as a photograph. For a human, it is a very simple task, but for a computer it is extremely difficult since it requires both understanding the content of an image and how to translate this understanding into natural language.

Recently, deep learning methods have displaced classical methods and are achieving state-of-the-art results for the problem of automatically generating descriptions, called “captions,” for images. In this project, we will see how deep neural network models can be used to automatically generate descriptions for images, such as photographs.

In this project, we use CNN and LSTM to generate the caption of the image. As the deep learning techniques are growing, huge datasets and computer power are helpful to build models that can generate captions for an image. This is what we are going to implement in this Python based project where we will use deep learning techniques like CNN and RNN.

Technical Architecture:



Pre-requisites:

To complete this project, you must require the following software's, concepts, and packages

Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS. Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, Spyder, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. To build Machine learning models you must require the following packages

- Numpy: o It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- Scikit-learn: o It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- Flask: Web framework used for building Web applications
- Python packages:
    - open anaconda prompt as administrator
    - Type “pip install numpy” and click enter.
    - Type “pip install pandas” and click enter.
    - Type “pip install scikit-learn” and click enter.
    - Type “pip install tensorflow==2.3.2” and click enter.
    - Type “pip install keras==2.3.1” and click enter.
    - Type “pip install Flask” and click enter.

Deep Learning Concepts

CNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery.

<u>CNN Basic</u>

Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Flask Basics

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project, you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data pre-processing techniques.
- know how to build a web application using the Flask framework.

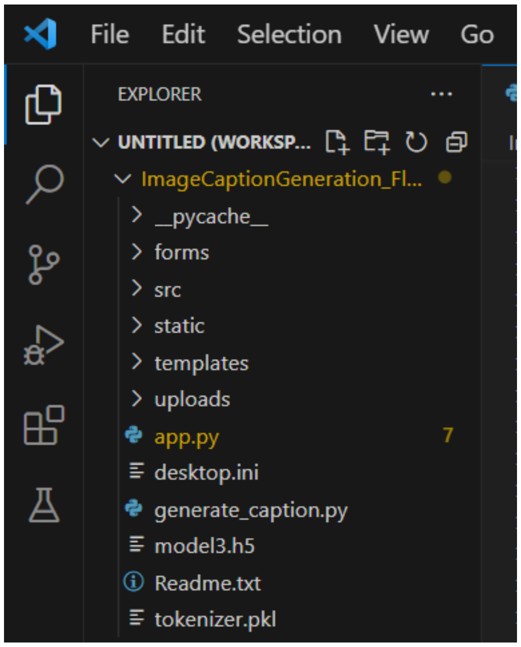
Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with flask application.
- VGG16 Model analyse the image, then LSTM is used to process the captions in form of text, and prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Create Train and Test Folders.
- Data Pre-processing.
- Model Building
    - Import the model building Libraries
    - Initializing the model
    - Adding Input Layer
    - Adding Hidden Layer
    - Adding Output Layer
    - Configure the Learning Process
- Training and testing the model
    - Save the Model
    - Application Building
    - Create an HTML file
    - Build Python Code
-

Project Structure:
Create a Project folder which contains files as shown below



```
UNTITLED (WORKSP...
  ImageCaptionGeneration_Fl...
    > __pycache__
    > forms
    > src
    > static
    > templates
    > uploads
    app.py                7
    desktop.ini
    generate_caption.py
    model3.h5
    Readme.txt
    tokenizer.pkl
```

● The Dataset folder contains the training and testing images for training our model.
● We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting
● We need the model which is saved as model.h5 and the captions as tokenizer.pkl
The templates folder contains index.html and prediction.html pages.

Milestone 1: Collection of Data

Data Collection: Collect images of events along with 5 captions associated to each image then organized into subdirectories based on their respective names as shown in the project structure. Create folders of images and a text file of captions that need to be recognized.
The given dataset has 7k+ different types of images and 40k+ high quality human readable text captions.

Download the Dataset- https://www.kaggle.com/datasets/adityajn105/flickr8k

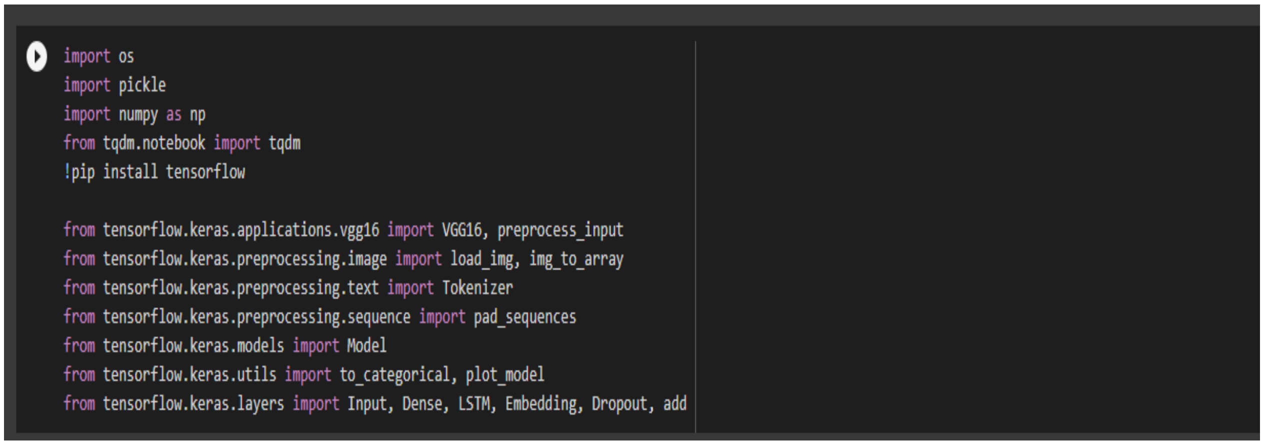
Milestone 2: Image Pre-processing and Model Building

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Then clean the text captions and map them together. Then it's time to build our Vgg16 model which contains an input layer along with the convolution, max-pooling, and finally an output layer and the LSTM model.

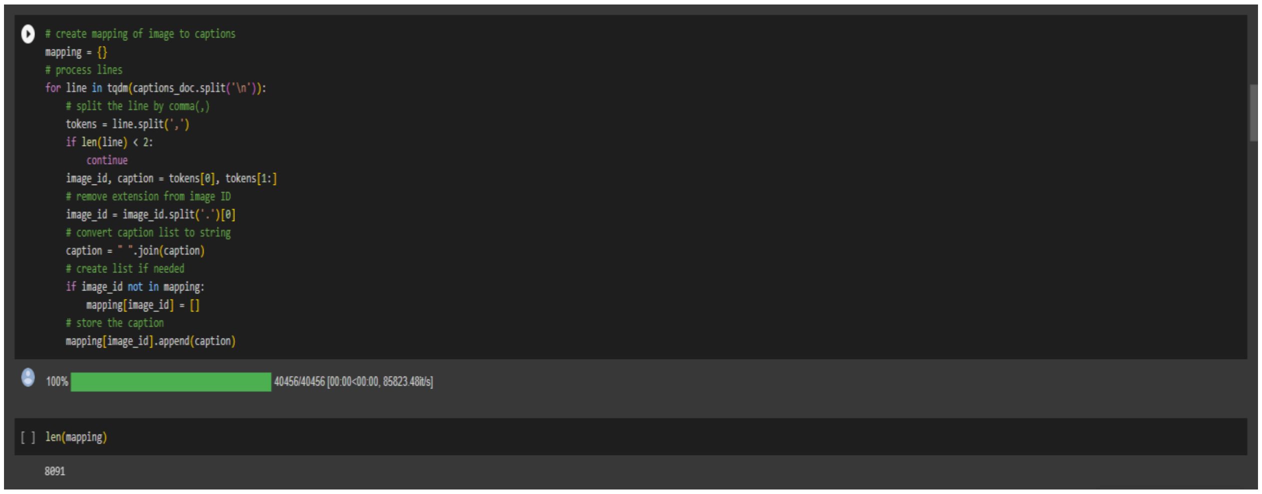
Activity 1: Importing the Model Building Libraries

Importing the necessary libraries



```
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm
!pip install tensorflow

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
```

Activity 2: Exploratory Data Analysis

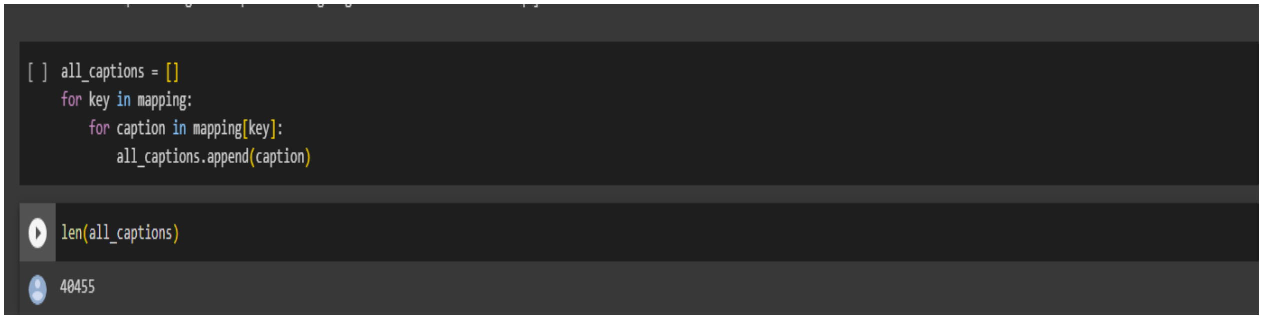


```
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

```
100% 40456/40456 [00:00<00:00, 85823.48it/s]
```

```
len(mapping)
```

```
8091
```



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

```
len(all_captions)
```

```
40455
```

Activity 4: Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

Activity 5: Configure the Learning Process

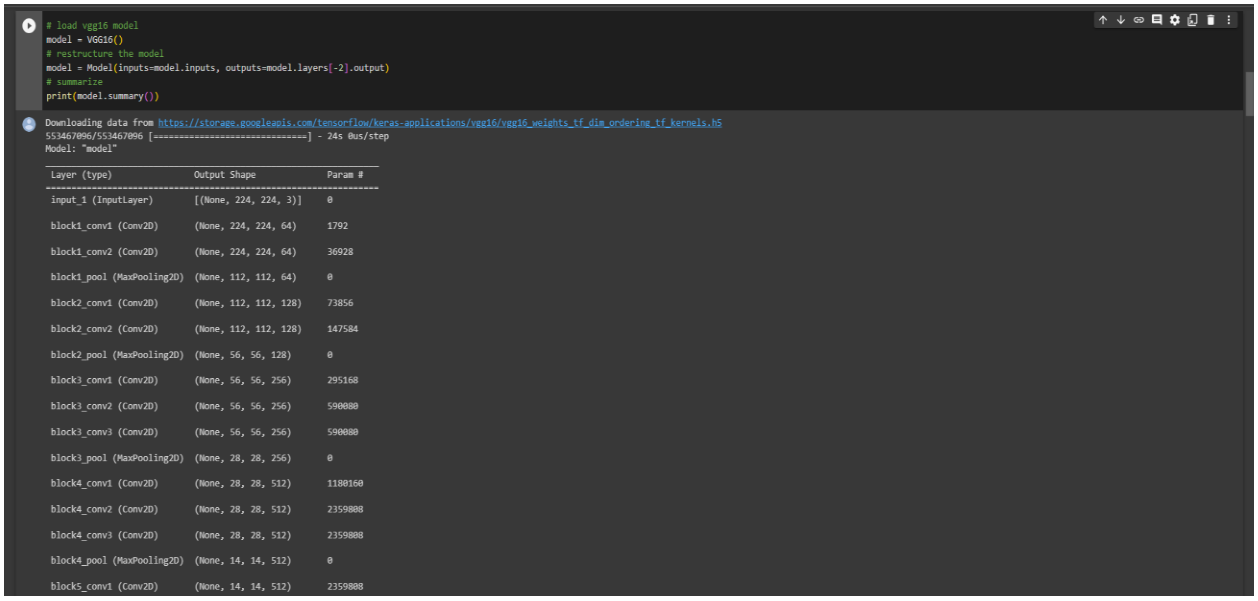
- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Activity 6: Train The model

Now, let us train our model with our image dataset. The model is trained for 20 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 20 epochs and probably there is further scope to improve the model.

Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
- Epochs: an integer and number of epochs we want to train our model for.
- validation_data can be either:
- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size

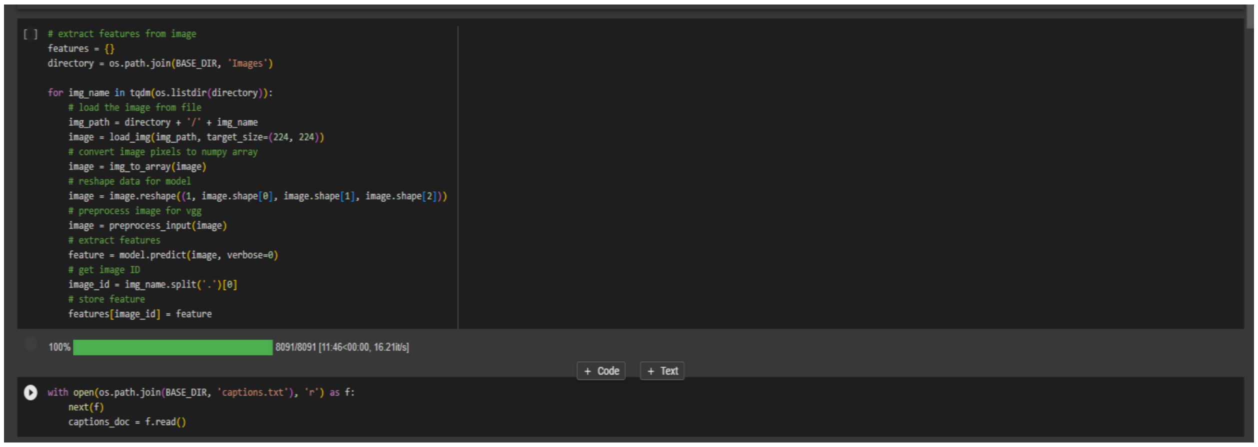


```
# load vgg16 model
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 24s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
```

Here we are creating a dictionary to have a key-value pair, wherein key will be the image id and value is the features.

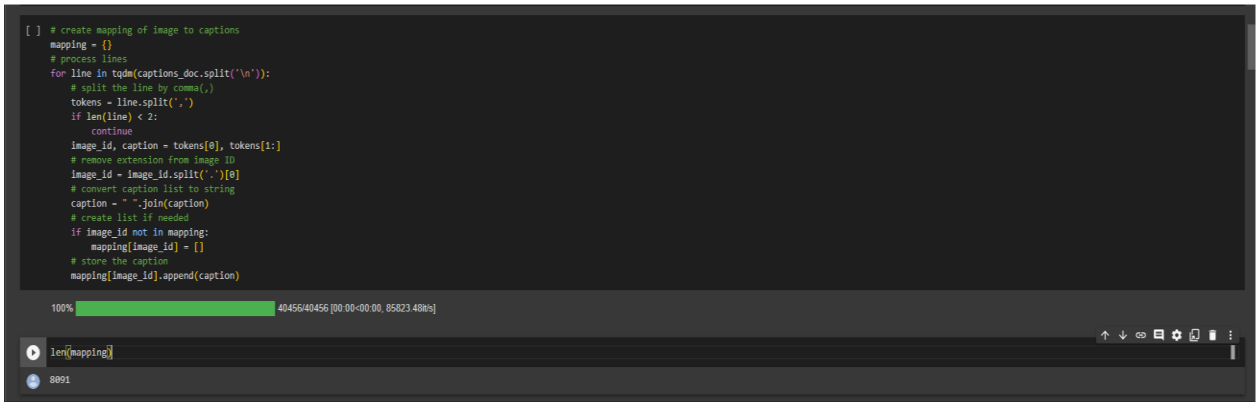
Then we are iterating through all the images, concatenating the image name to get the whole file path/image path.



```
[ ] # extract features from image
    features = {}
    directory = os.path.join(BASE_DIR, 'Images')

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

100% 8091/8091 [11:46<00:00, 16.21it/s]
```

```
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

In this stage, we are loading the captions data.

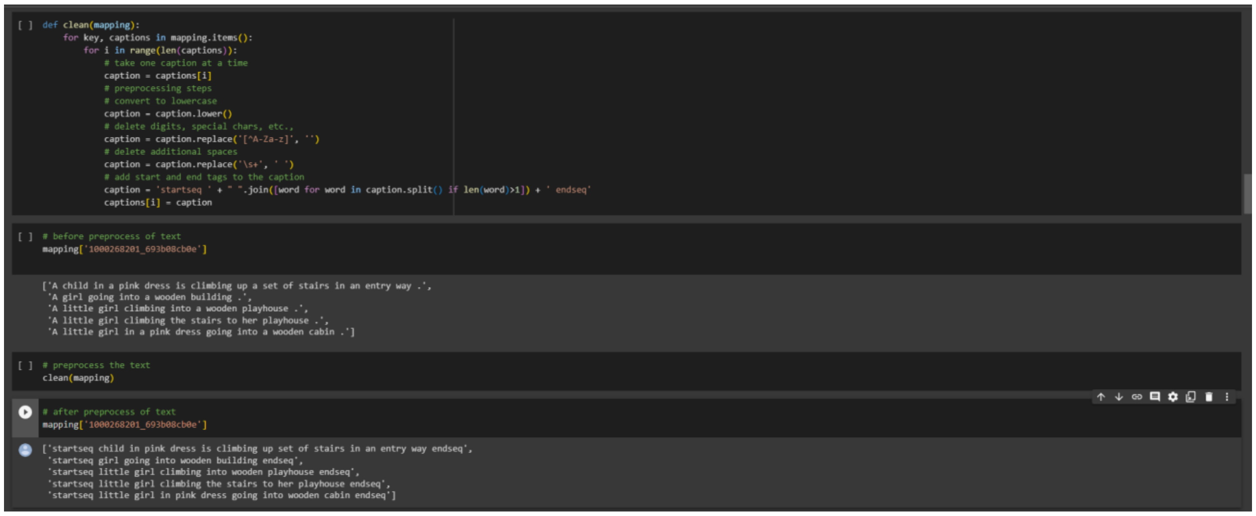


```
[ ] # create mapping of image to captions
    mapping = {}
    # process lines
    for line in tqdm(captions_doc.split('\n')):
        # split the line by comma(,)
        tokens = line.split(',')
        if len(line) < 2:
            continue
        image_id, caption = tokens[0], tokens[1:]
        # remove extension from image ID
        image_id = image_id.split('.')[0]
        # convert caption list to string
        caption = " ".join(caption)
        # create list if needed
        if image_id not in mapping:
            mapping[image_id] = []
        # store the caption
        mapping[image_id].append(caption)

100% 40456/40456 [00:00<00:00, 85823.48it/s]
```

```
len(mapping)
```

```
8091
```

Here, we are pre-processing the text data by mapping of the images for generating textual description of the images



```
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
```

```
# before preprocess of text
mapping['1000268201_693b08cb0e']
```

```
['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

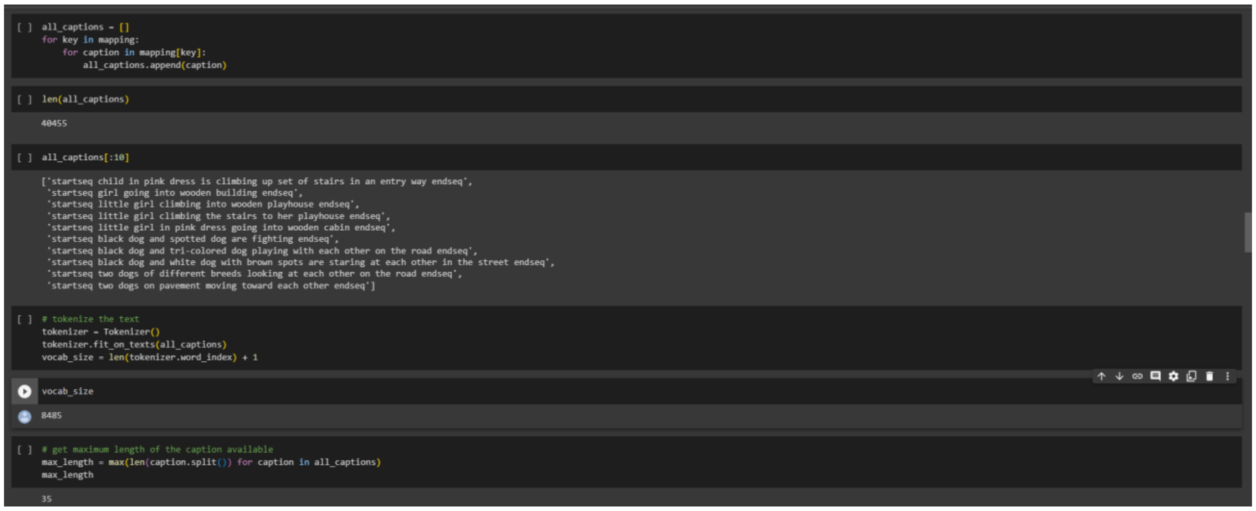
```
# preprocess the text
clean(mapping)
```

```
# after preprocess of text
mapping['1000268201_693b08cb0e']
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

This is the pre-processing stage of the entire mapping data. Here we need to do some cleaning operation on the mapping data like converting it to lower case and remove digits, special characters and additional spaces. After, pre-processing we can see the special characters are removed, single letters are removed and the start and end tag has been added to each string.

Here, we are printing the first 10 captions of data. After that, we are initialising the tokenizer function to tokenize all the captions by passing all the captions to it.



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

```
len(all_captions)
```

```
40455
```

```
all_captions[:10]
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 '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']
```

```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

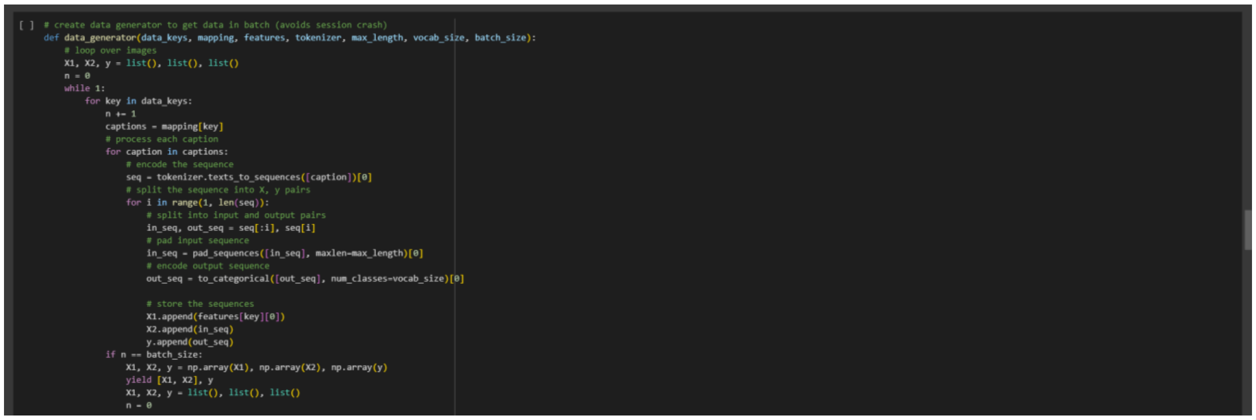
```
vocab_size
```

```
8485
```

```
# get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

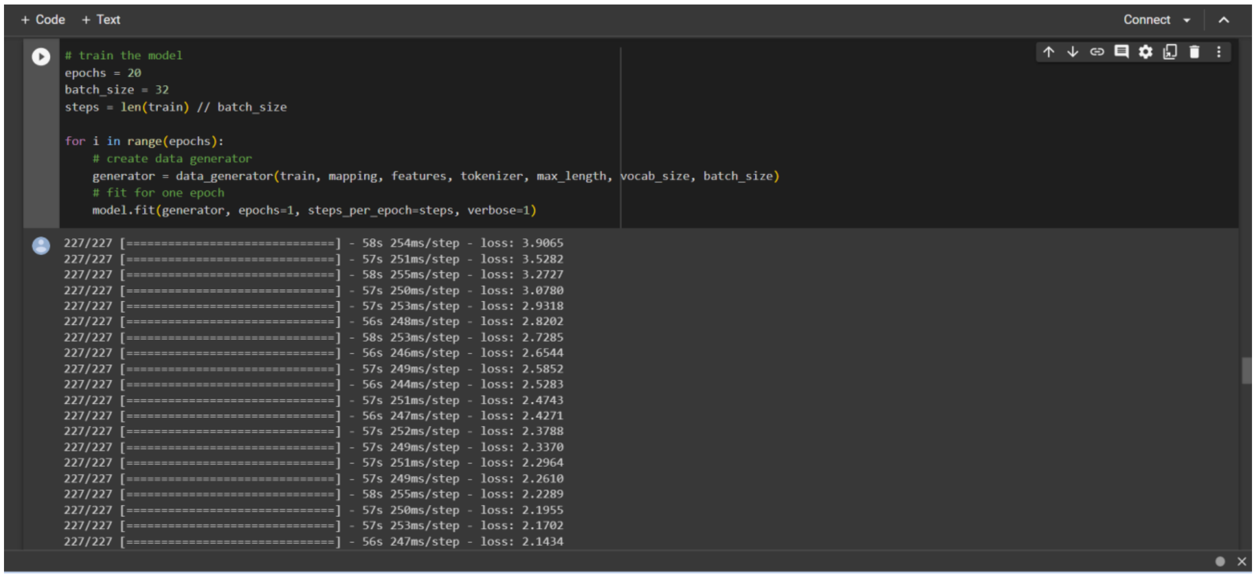
```
35
```

Here, the data is divided into train data and test data.90% of the data is trained and remaining 10% of the data is tested.



```
[ ] # create data generator to get data in batch (avoids session crash)
    def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
        # loop over images
        X1, X2, y = list(), list(), list()
        n = 0
        while 1:
            for key in data_keys:
                n += 1
                captions = mapping[key]
                # process each caption
                for caption in captions:
                    # encode the sequence
                    seq = tokenizer.texts_to_sequences([caption])[0]
                    # split the sequence into X, y pairs
                    for i in range(1, len(seq)):
                        # split into input and output pairs
                        in_seq, out_seq = seq[:i], seq[i]
                        # pad input sequence
                        in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                        # encode output sequence
                        out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                        # store the sequences
                        X1.append(features[key][0])
                        X2.append(in_seq)
                        y.append(out_seq)
                if n == batch_size:
                    X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                    yield [X1, X2], y
                    X1, X2, y = list(), list(), list()
                    n = 0
```

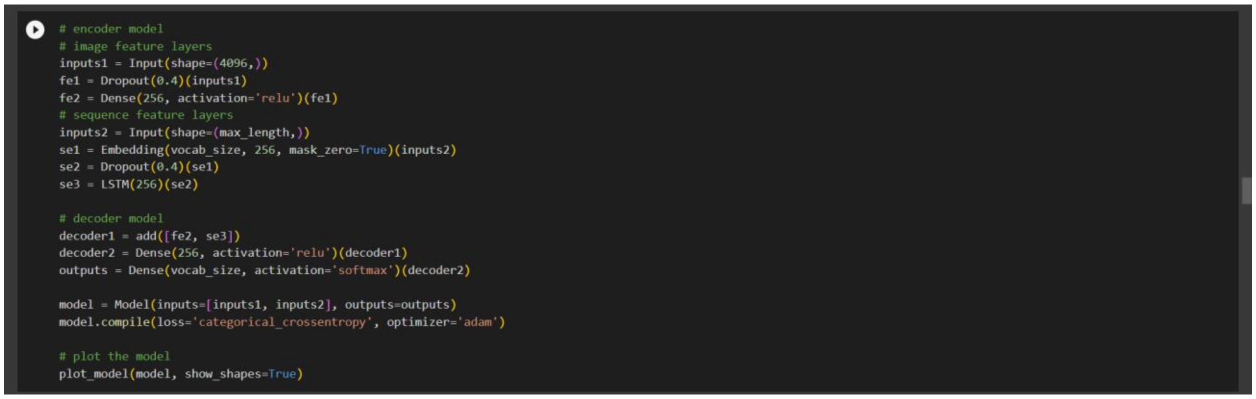
Analysing the trained model by looking at the accuracy through the epocs:



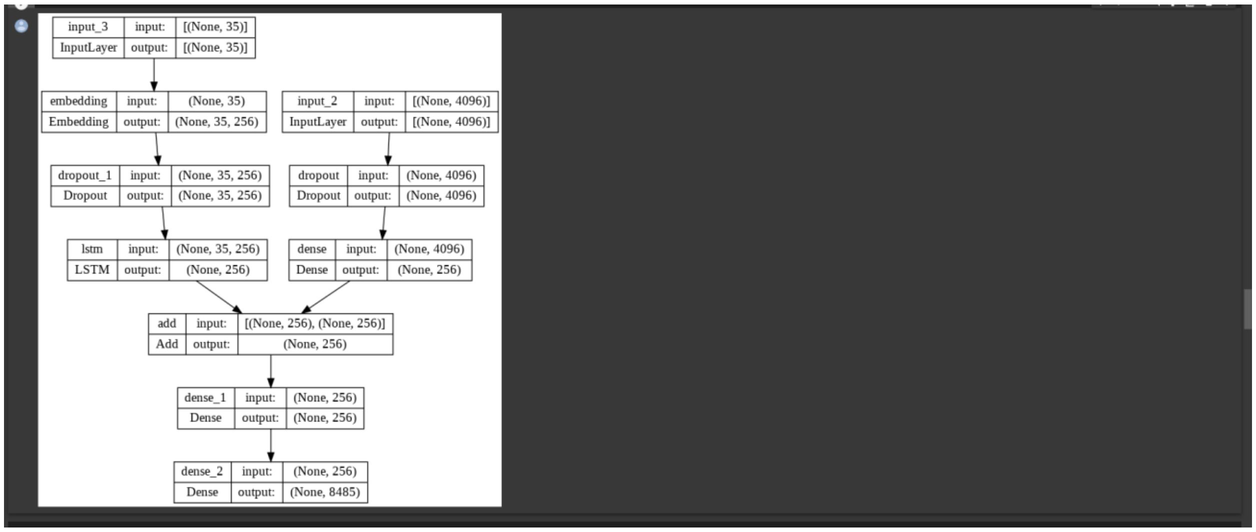
```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

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 [==============================] - 58s 254ms/step - loss: 3.9065
227/227 [==============================] - 57s 251ms/step - loss: 3.5282
227/227 [==============================] - 58s 255ms/step - loss: 3.2727
227/227 [==============================] - 57s 250ms/step - loss: 3.0780
227/227 [==============================] - 57s 253ms/step - loss: 2.9318
227/227 [==============================] - 56s 248ms/step - loss: 2.8202
227/227 [==============================] - 58s 253ms/step - loss: 2.7285
227/227 [==============================] - 56s 246ms/step - loss: 2.6544
227/227 [==============================] - 57s 249ms/step - loss: 2.5852
227/227 [==============================] - 56s 244ms/step - loss: 2.5283
227/227 [==============================] - 57s 251ms/step - loss: 2.4743
227/227 [==============================] - 56s 247ms/step - loss: 2.4271
227/227 [==============================] - 57s 252ms/step - loss: 2.3788
227/227 [==============================] - 57s 249ms/step - loss: 2.3370
227/227 [==============================] - 57s 251ms/step - loss: 2.2964
227/227 [==============================] - 57s 249ms/step - loss: 2.2610
227/227 [==============================] - 58s 255ms/step - loss: 2.2289
227/227 [==============================] - 57s 250ms/step - loss: 2.1955
227/227 [==============================] - 57s 253ms/step - loss: 2.1702
227/227 [==============================] - 56s 247ms/step - loss: 2.1434
```

Visualising Model Summary:

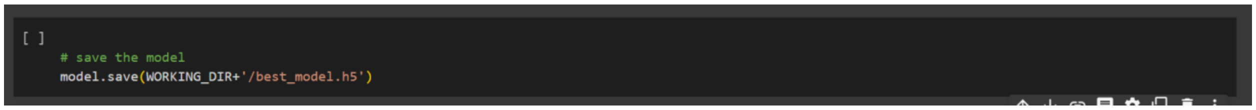


```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# 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)
```


input_3 | InputLayer | input: [(None, 35)] | output: [(None, 35)]
embedding | Embedding | input: (None, 35) | output: (None, 35, 256)
input_2 | InputLayer | input: [(None, 4096)] | output: [(None, 4096)]
dropout_1 | Dropout | input: (None, 35, 256) | output: (None, 35, 256)
dropout | Dropout | input: (None, 4096) | output: (None, 4096)
lstm | LSTM | input: (None, 35, 256) | output: (None, 256)
dense | Dense | input: (None, 4096) | output: (None, 256)
add | Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_1 | Dense | input: (None, 256) | output: (None, 256)
dense_2 | Dense | input: (None, 256) | output: (None, 8485)

Activity 7: Save the Model

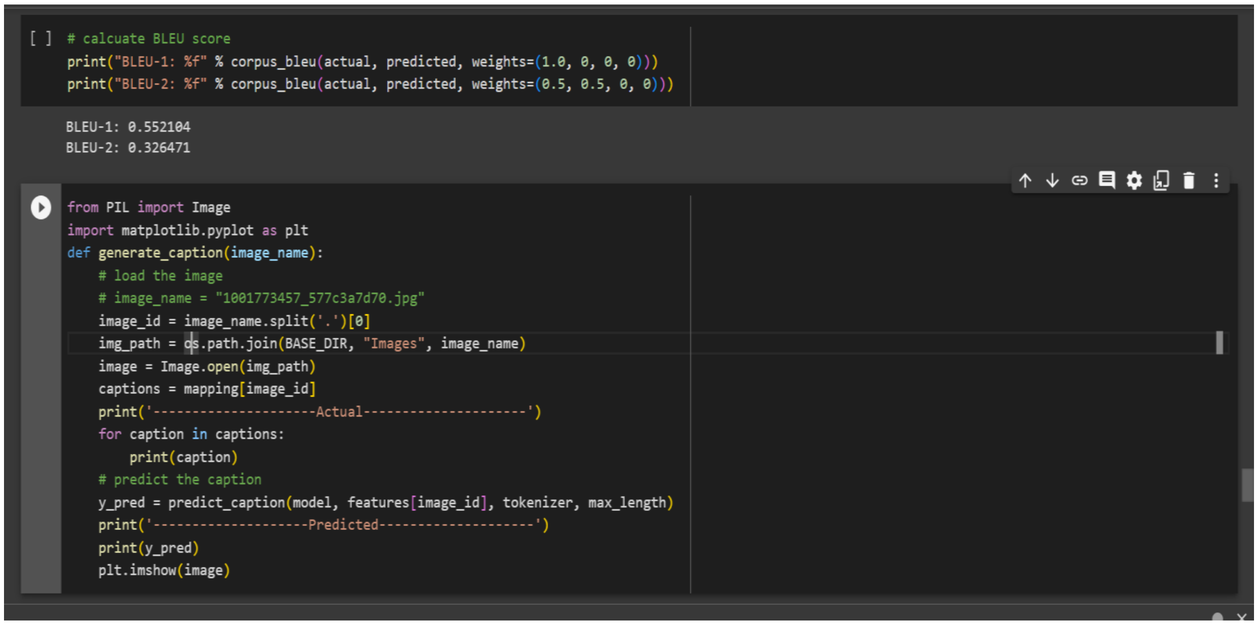
The model is saved with .h5 extension as follows An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



```
[ ]
    # save the model
    model.save(WORKING_DIR+'/best_model.h5')
```

Activity 8: Test The model

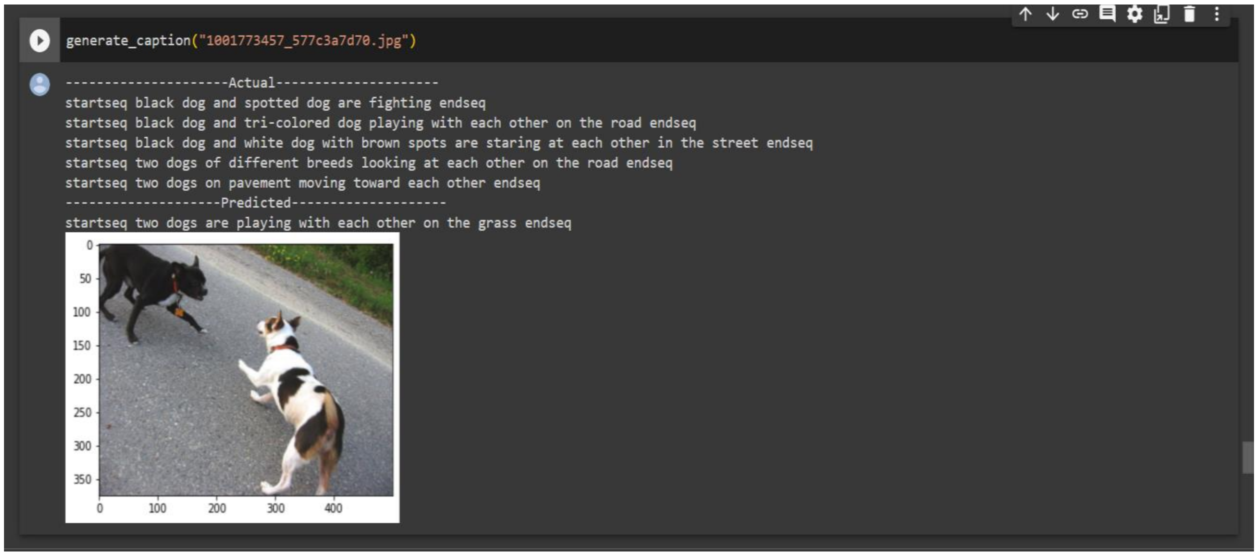
Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model.



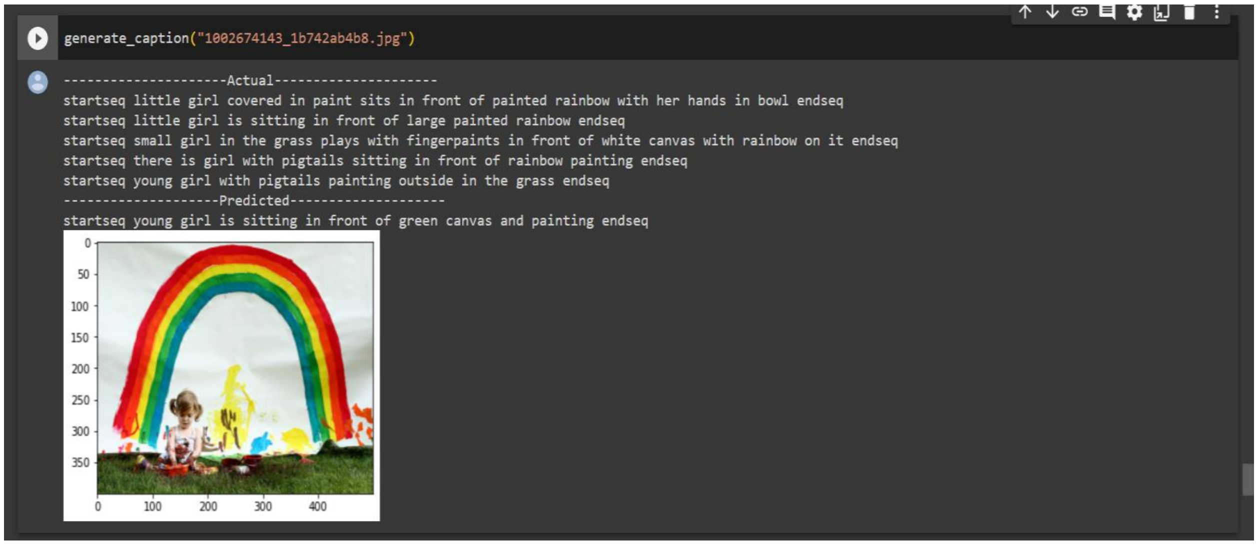
```
[ ] # calcuate BLEU score
    print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))

    BLEU-1: 0.552104
    BLEU-2: 0.326471
```

```
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, "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)
```

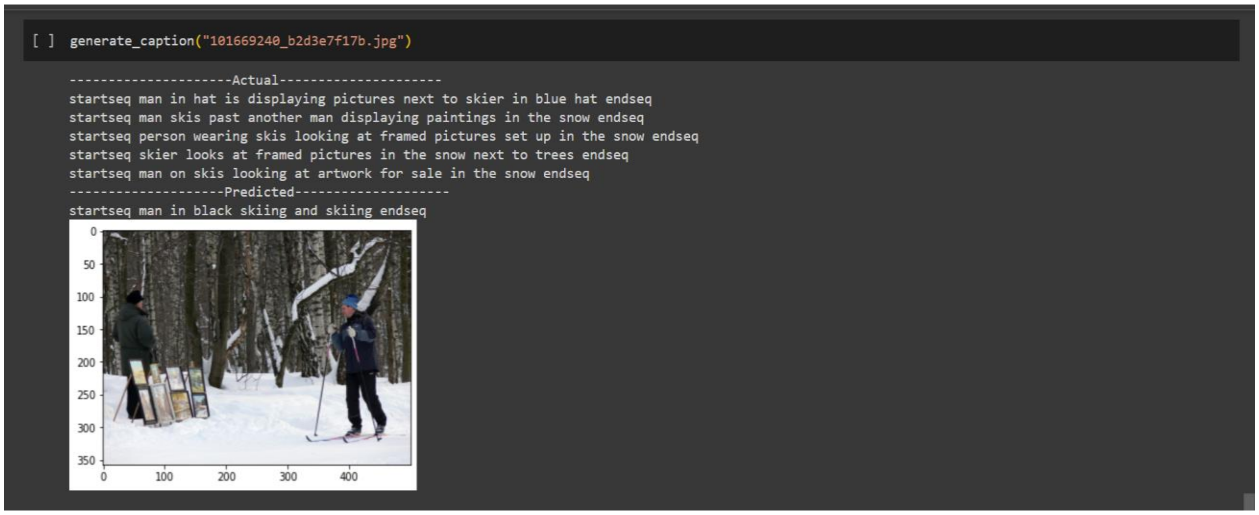
Taking an image as input and checking the results. **Displaying 4 Test Results.**



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

--------------------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 each other on the grass endseq
```



```
generate_caption("1002674143_1b742ab4b8.jpg")

--------------------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 young girl is sitting in front of green canvas and painting endseq
```



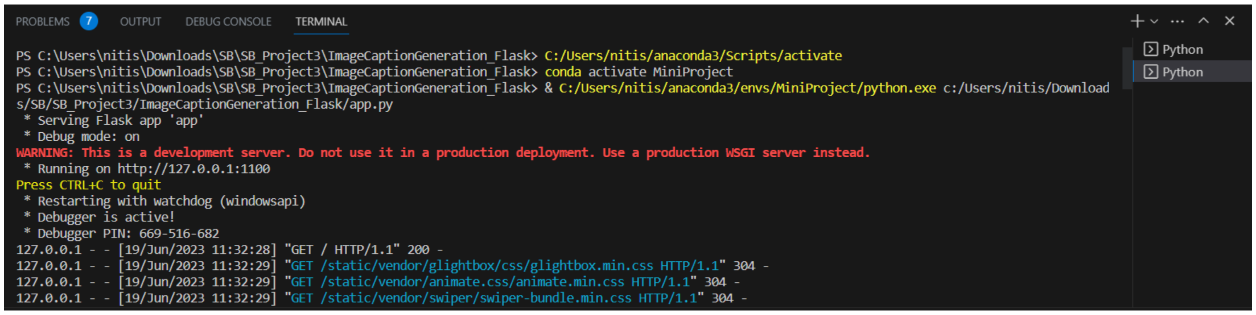
```
[ ] generate_caption("101669240_b2d3e7f17b.jpg")

--------------------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 man in black skiing and skiing endseq
```

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface. In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Image” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

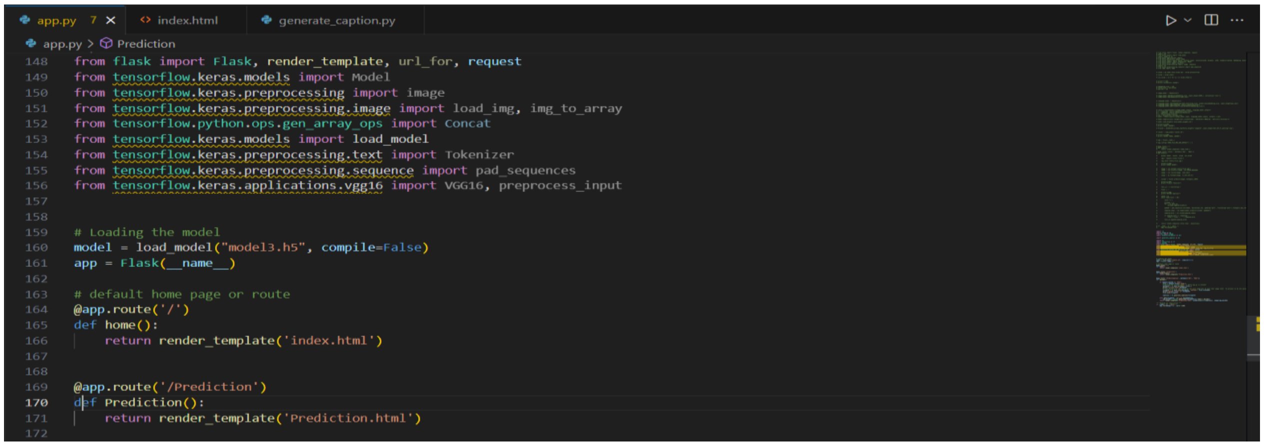
- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
- home.html displays the home page.
- Intro.html displays an introduction about the project
- upload.html gives the emergency alert For more information regarding HTML https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- Link :CSS , JS



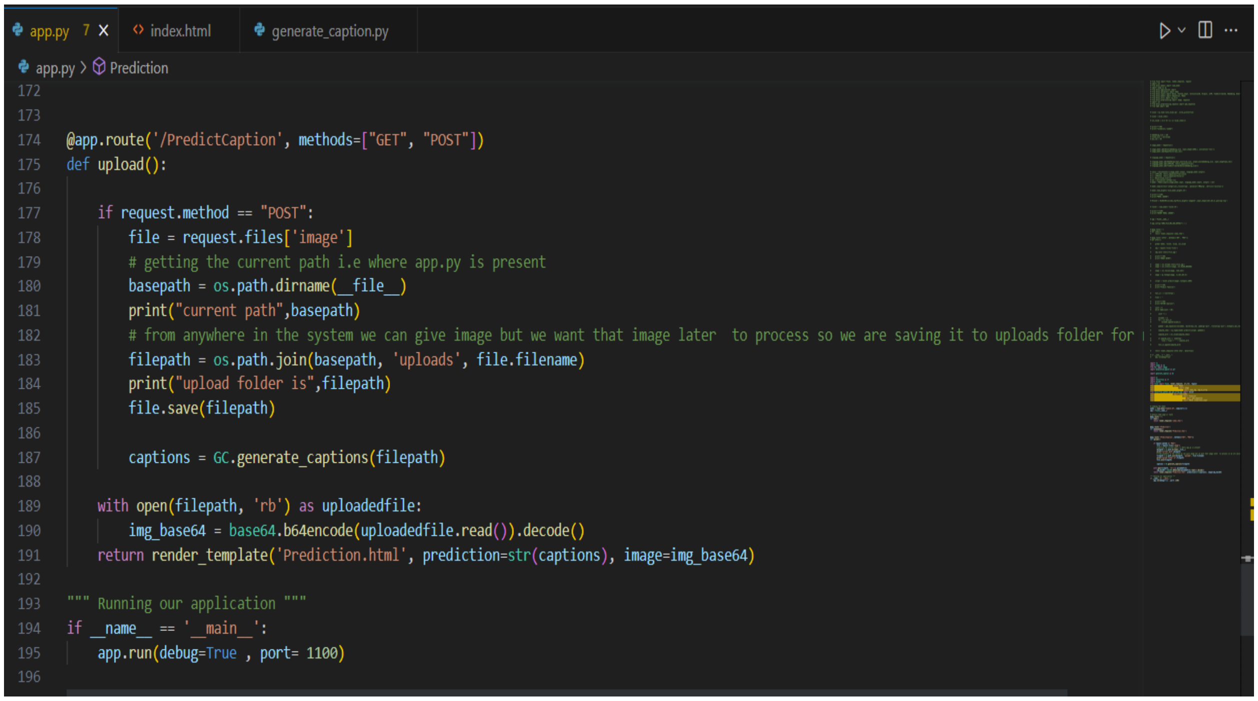
```
PS C:\Users\nitis\Downloads\SB\SB_Project3\ImageCaptionGeneration_Flask> C:/Users/nitis/anaconda3/Scripts/activate
PS C:\Users\nitis\Downloads\SB\SB_Project3\ImageCaptionGeneration_Flask> conda activate MiniProject
PS C:\Users\nitis\Downloads\SB\SB_Project3\ImageCaptionGeneration_Flask> & C:/Users/nitis/anaconda3/envs/MiniProject/python.exe c:/Users/nitis/Download
s/SB/SB_Project3/ImageCaptionGeneration_Flask/app.py
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:1100
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 669-516-682
127.0.0.1 - - [19/Jun/2023 11:32:28] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [19/Jun/2023 11:32:29] "GET /static/vendor/glightbox/css/glightbox.min.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Jun/2023 11:32:29] "GET /static/vendor/animate.css/animate.min.css HTTP/1.1" 304 -
127.0.0.1 - - [19/Jun/2023 11:32:29] "GET /static/vendor/swiper/swiper-bundle.min.css HTTP/1.1" 304 -
```

Create app.py (Python Flask) file: -

Write below code in Flask app.py python file script to run Micro-Organism Classification Project.

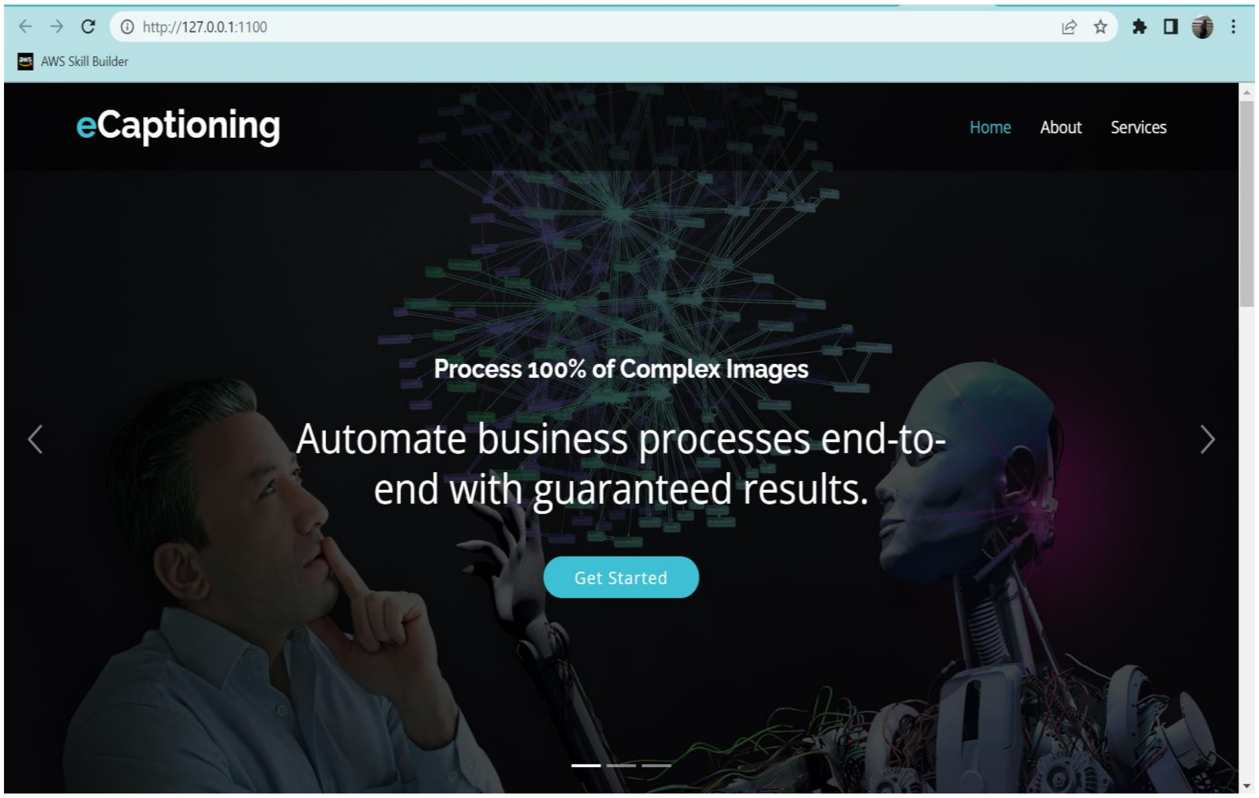


```python
from flask import Flask, render_template, url_for, request
from tensorflow.keras.models import Model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.python.ops.gen_array_ops import Concat
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input


# Loading the model
model = load_model("model3.h5", compile=False)
app = Flask(__name__)

# default home page or route
@app.route('/')
def home():
    return render_template('index.html')


@app.route('/Prediction')
def Prediction():
    return render_template('Prediction.html')

```



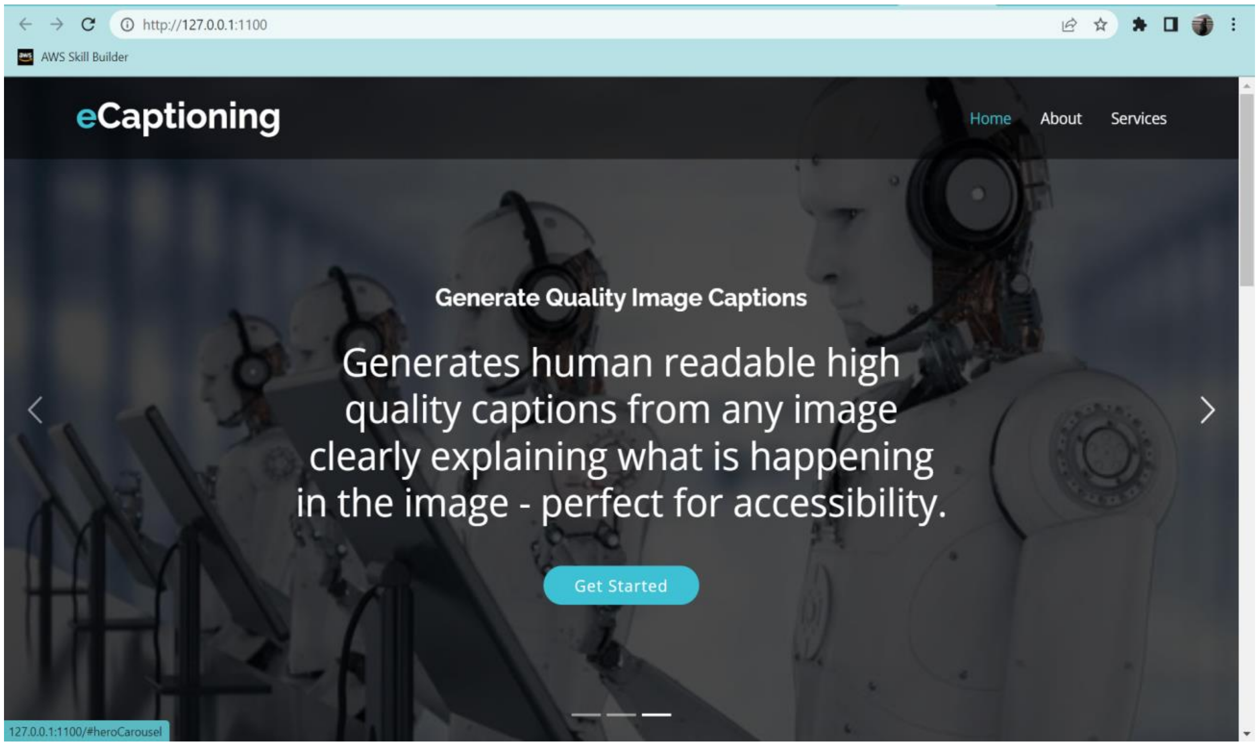
```python

@app.route('/PredictCaption', methods=["GET", "POST"])
def upload():

    if request.method == "POST":
        file = request.files['image']
        # getting the current path i.e where app.py is present
        basepath = os.path.dirname(__file__)
        print("current path",basepath)
        # from anywhere in the system we can give image but we want that image later  to process so we are saving it to uploads folder for
        filepath = os.path.join(basepath, 'uploads', file.filename)
        print("upload folder is",filepath)
        file.save(filepath)

        captions = GC.generate_captions(filepath)

    with open(filepath, 'rb') as uploadedfile:
        img_base64 = base64.b64encode(uploadedfile.read()).decode()
    return render_template('Prediction.html', prediction=str(captions), image=img_base64)

""" Running our application """
if __name__ == '__main__':
    app.run(debug=True , port= 1100)

```

Index.html is displayed below (Webpage named "eCaptioning"):



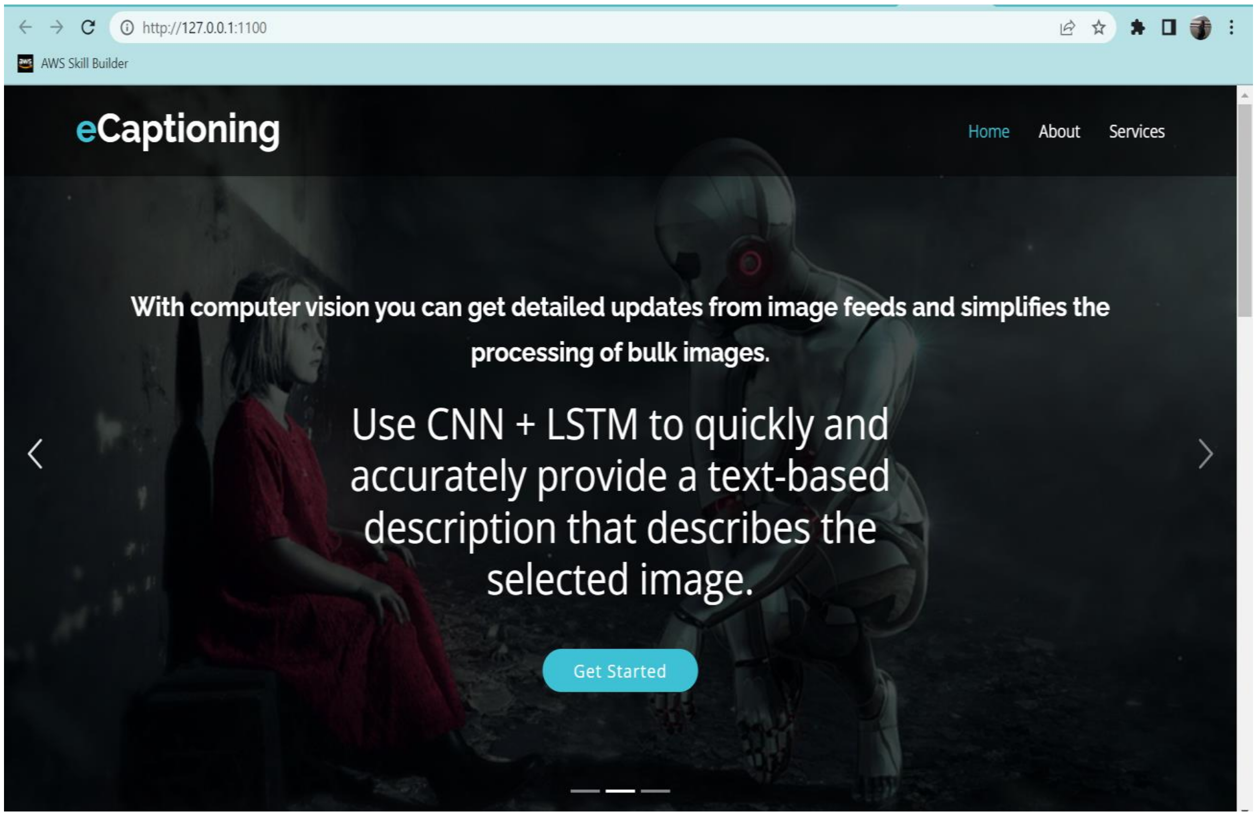
Process 100% of Complex Images

Automate business processes end-to-end with guaranteed results.



Generate Quality Image Captions

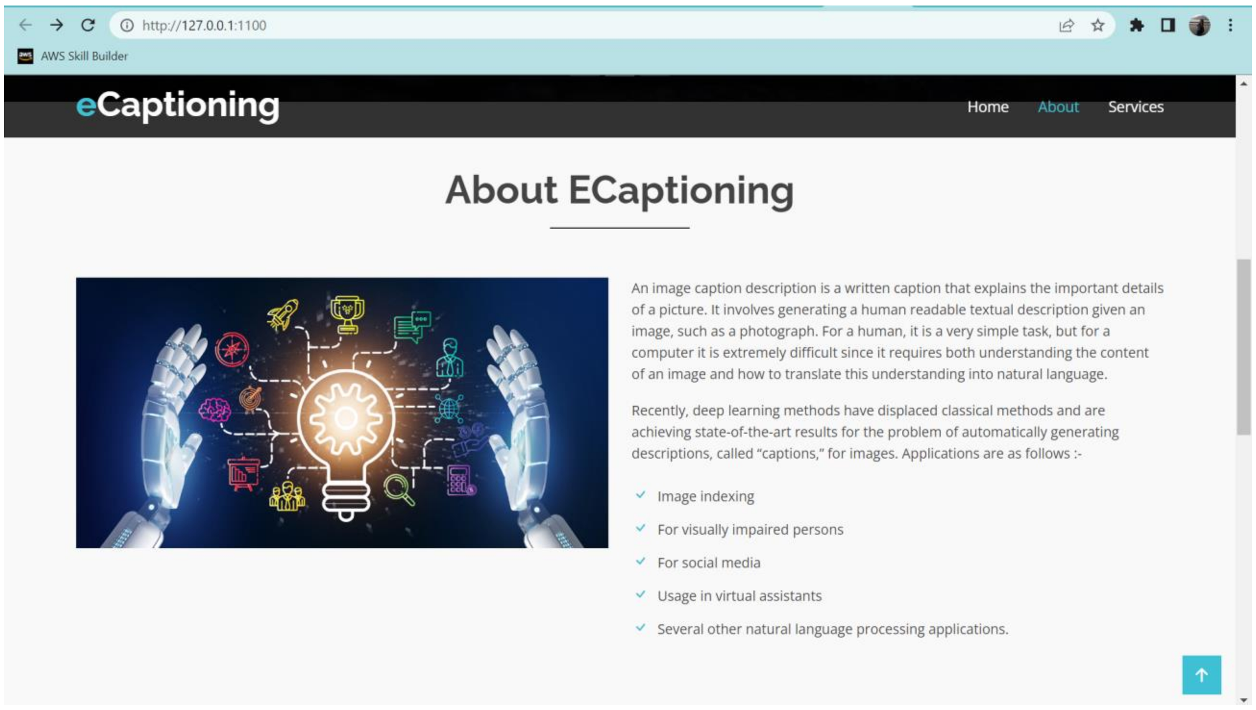
Generates human readable high quality captions from any image clearly explaining what is happening in the image - perfect for accessibility.



With computer vision you can get detailed updates from image feeds and simplifies the processing of bulk images.

Use CNN + LSTM to quickly and accurately provide a text-based description that describes the selected image.

About Section is displayed below:


http://127.0.0.1:1100

AWS Skill Builder

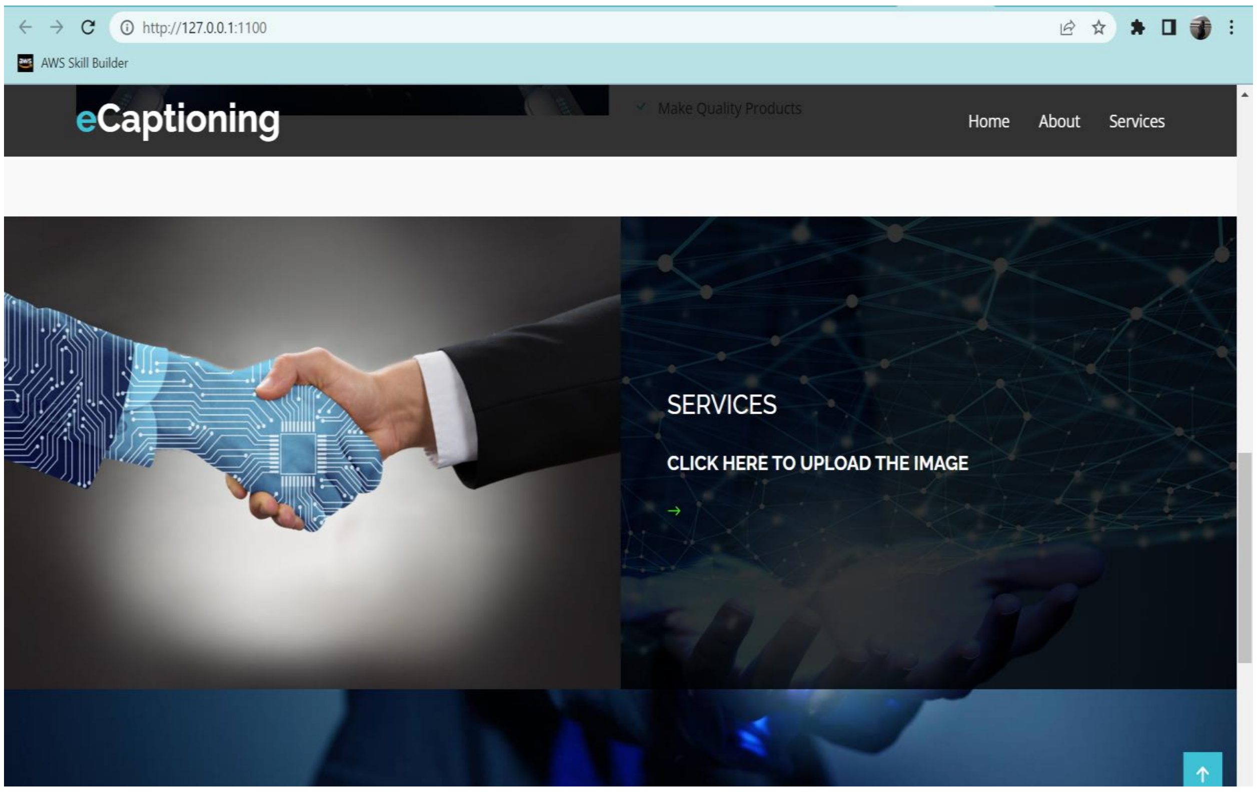
About ECaptioning

An image caption description is a written caption that explains the important details of a picture. It involves generating a human readable textual description given an image, such as a photograph. For a human, it is a very simple task, but for a computer it is extremely difficult since it requires both understanding the content of an image and how to translate this understanding into natural language.

Recently, deep learning methods have displaced classical methods and are achieving state-of-the-art results for the problem of automatically generating descriptions, called "captions," for images. Applications are as follows :-

- Image indexing
- For visually impaired persons
- For social media
- Usage in virtual assistants
- Several other natural language processing applications.

Services section is displayed below with a click button.

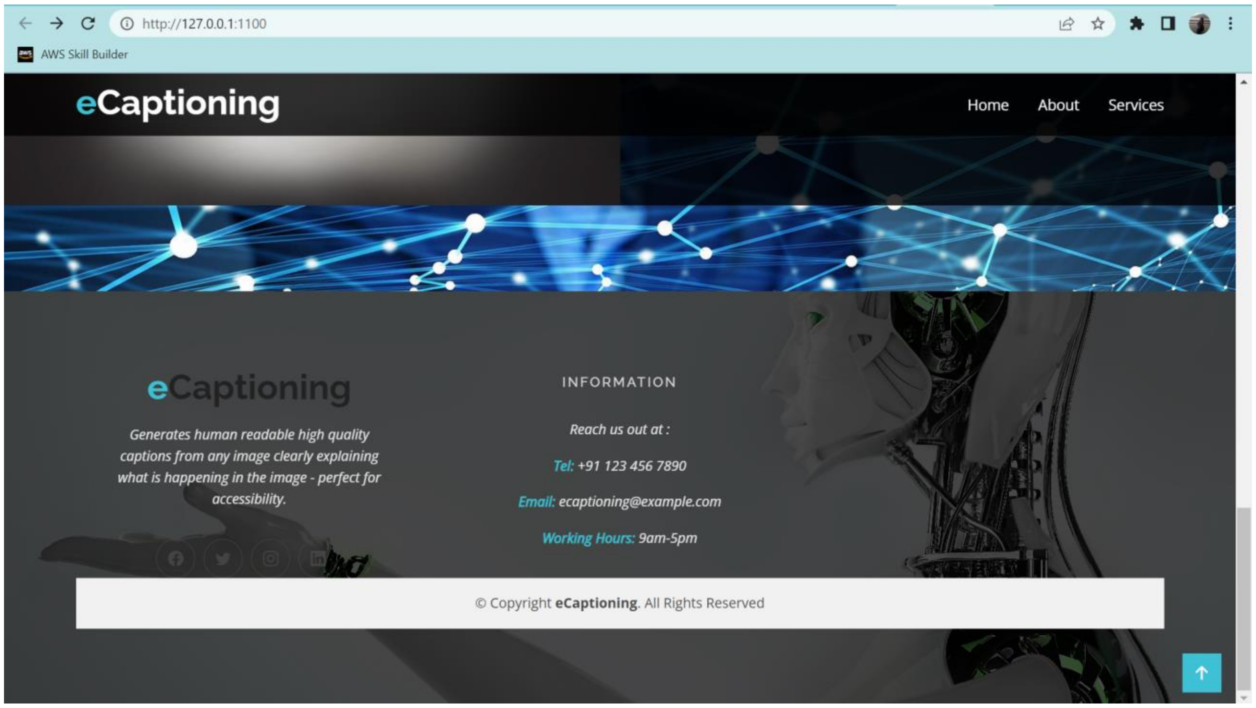

http://127.0.0.1:1100

AWS Skill Builder

SERVICES

CLICK HERE TO UPLOAD THE IMAGE

Finally, the Footer of the Webpage along with the choosing of Image for prediction.



eCaptioning

Generates human readable high quality captions from any image clearly explaining what is happening in the image - perfect for accessibility.

INFORMATION

Reach us out at :

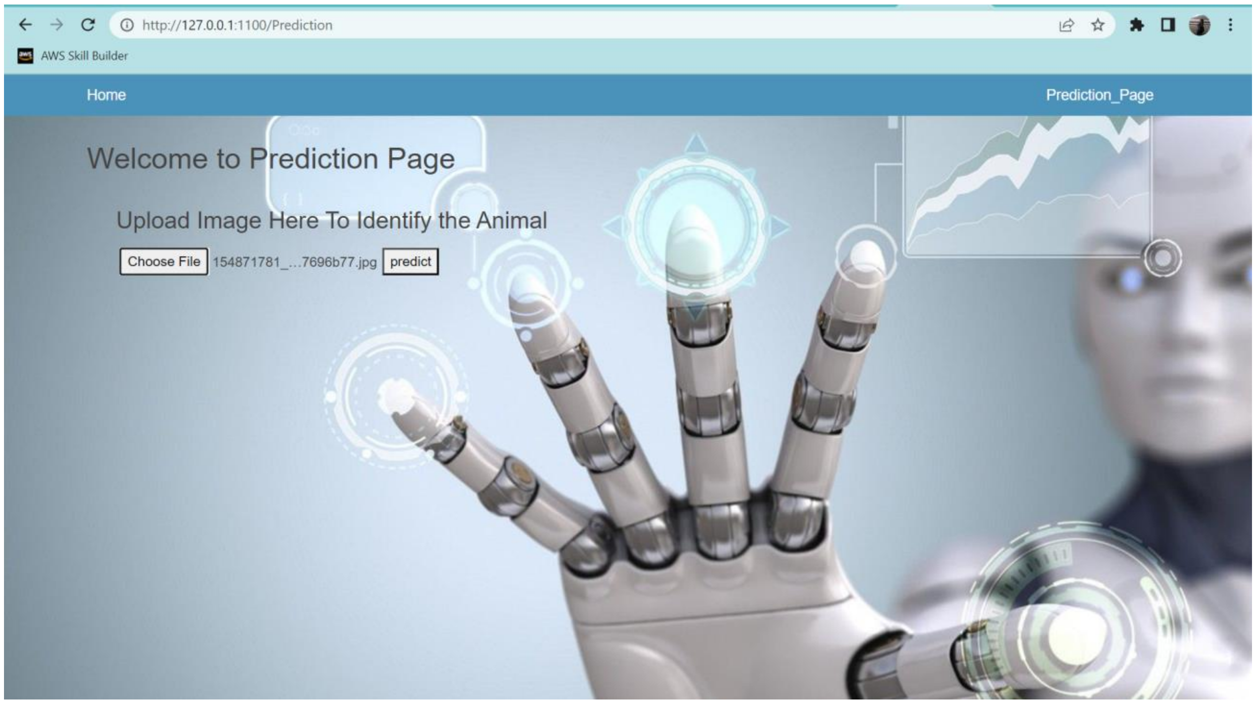
Tel: +91 123 456 7890

Email: ecaptioning@example.com

Working Hours: 9am-5pm

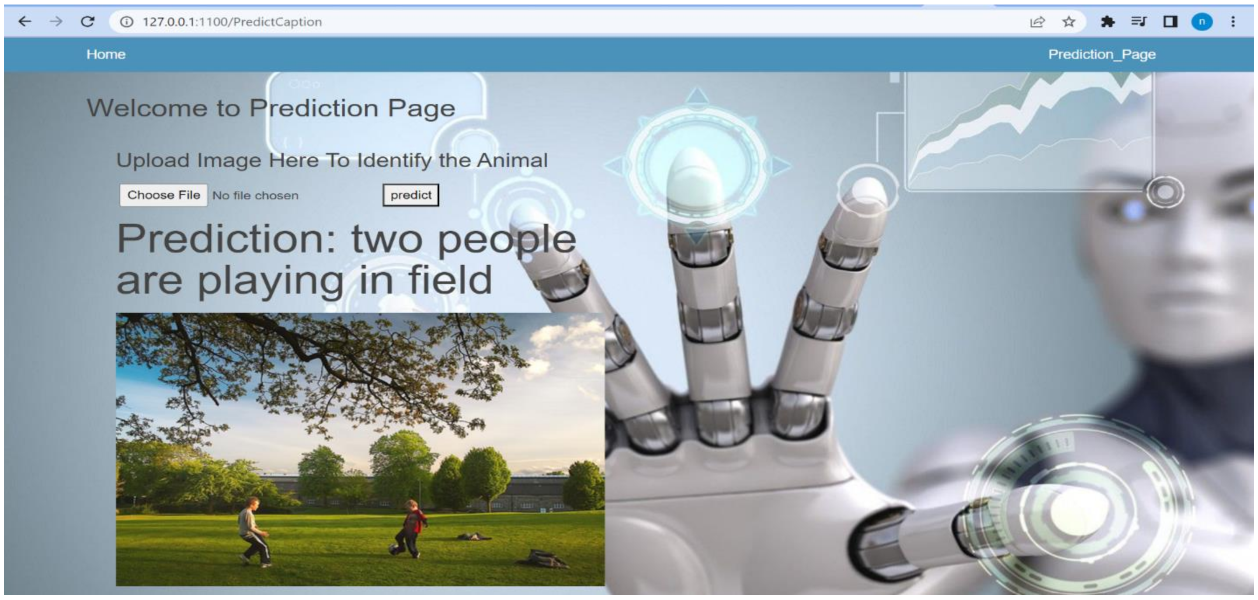
© Copyright **eCaptioning**. All Rights Reserved

Final Predicted Output (after I click on the Predict Button) is displayed as follows:



Welcome to Prediction Page

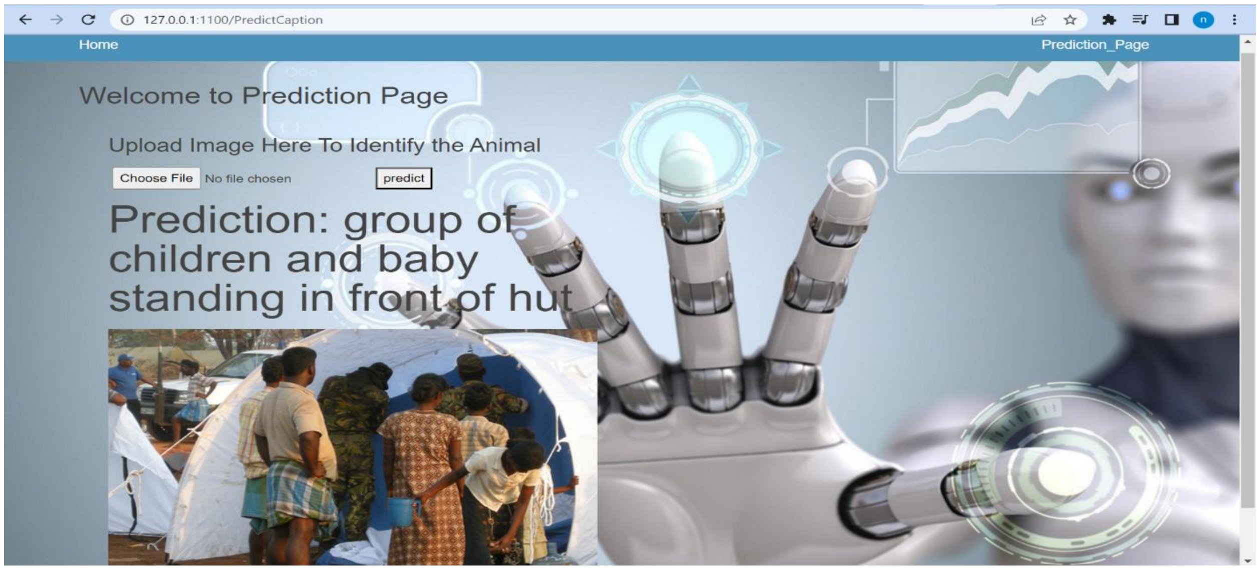
Upload Image Here To Identify the Animal



Welcome to Prediction Page

Upload Image Here To Identify the Animal

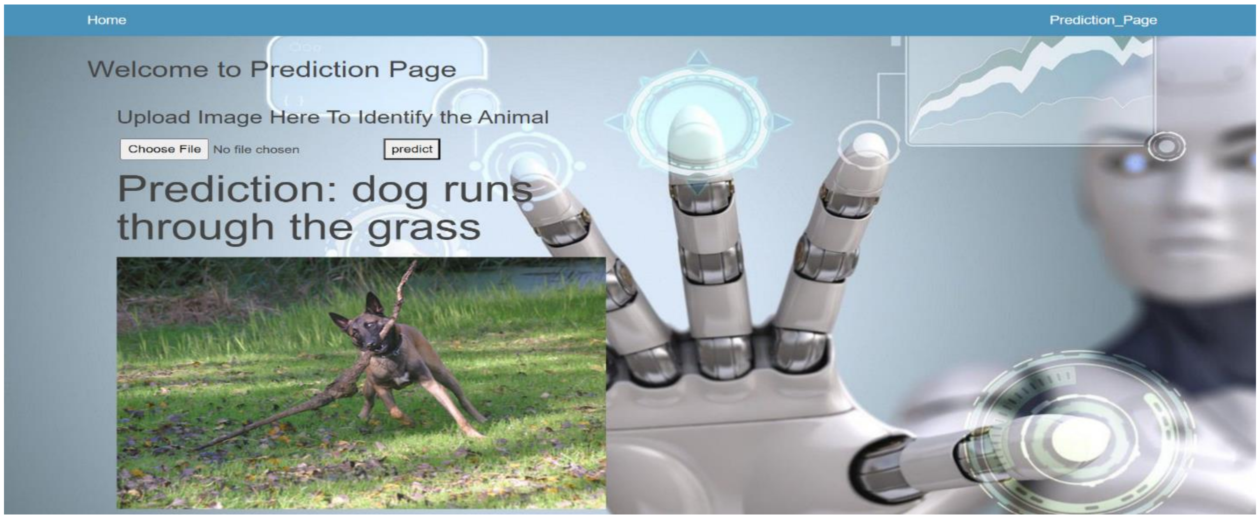
Prediction: two people are playing in field



Welcome to Prediction Page

Upload Image Here To Identify the Animal

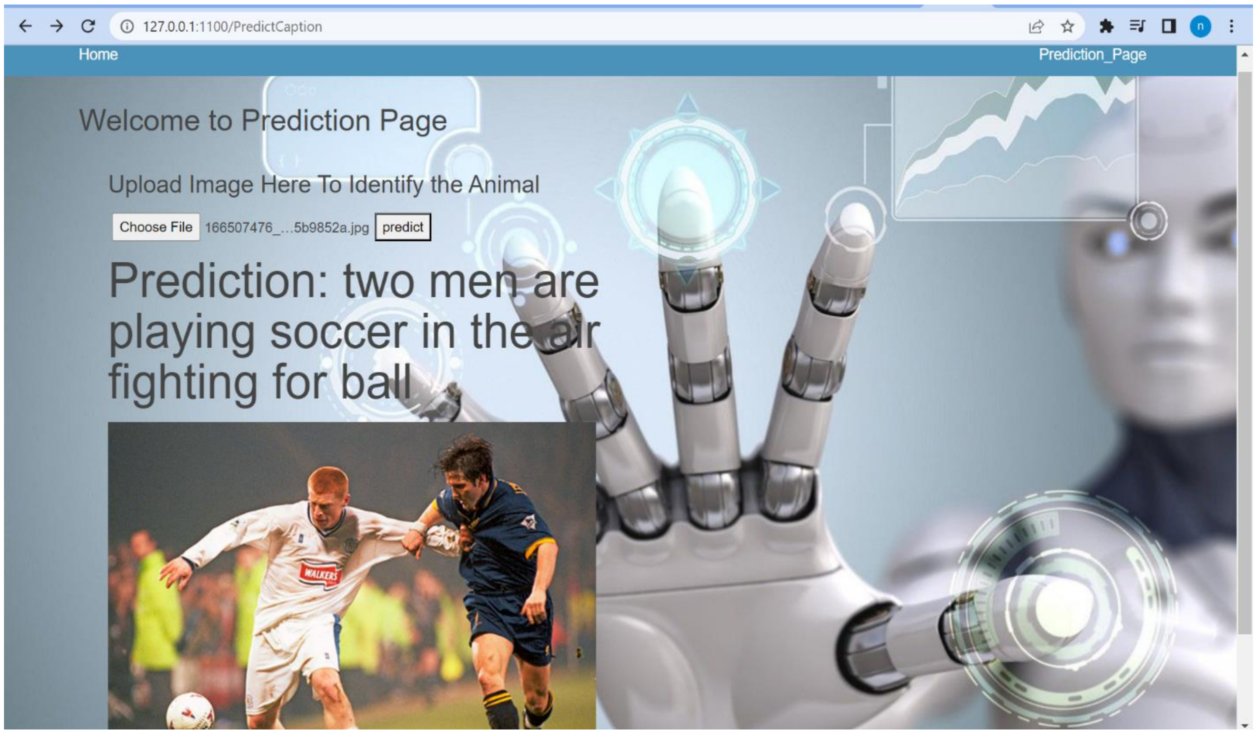
Prediction: group of children and baby standing in front of hut



Welcome to Prediction Page

Upload Image Here To Identify the Animal

Prediction: dog runs through the grass



Welcome to Prediction Page

Upload Image Here To Identify the Animal

Prediction: two men are playing soccer in the air fighting for ball

****Thank You****