Image Caption Generation Using ML

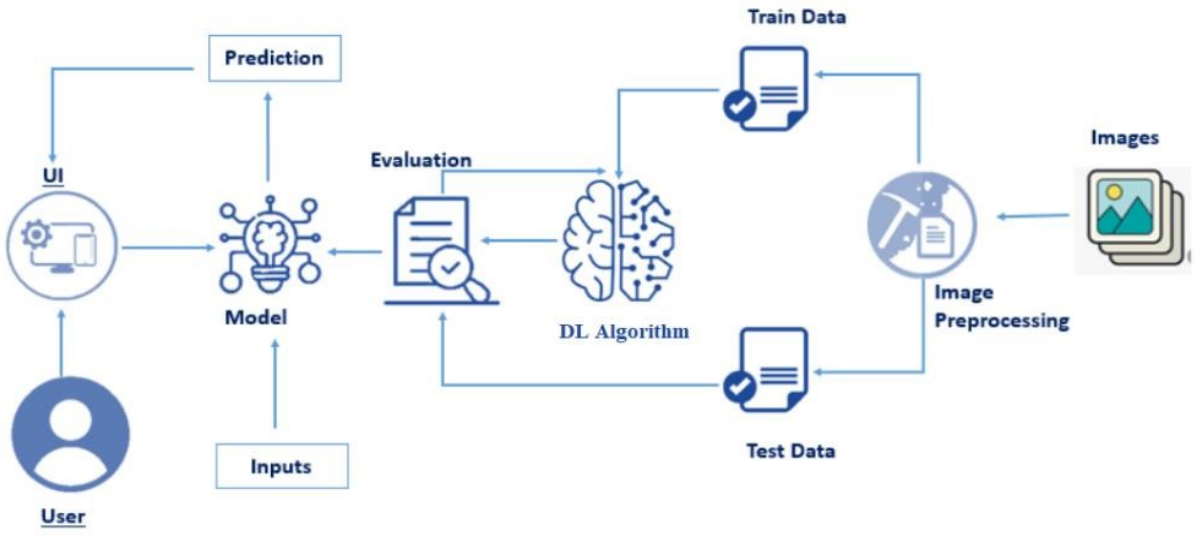
Date	21 November 2023
Team ID	Team-592072
Project Name	Image Caption Generation

Introduction:

In the age of artificial intelligence, novel applications that bridge the gap between visual and textual understanding have been made possible by the integration of computer vision and natural language processing. The Image Caption Generator is one such application; it's a system made to automatically provide insightful captions for pictures.

This paper examines the application of an Image Caption Generator that generates captions by combining thick layers and recurrent neural networks with the VGG16 model for feature extraction. In this study, significant features are extracted from photos using a pre-trained VGG16 model, and the correlations between these visual features and linguistic descriptions are learned using a deep learning architecture.

Technical Architecture:



Pre-requisites:

To successfully build and run the Image Caption Generator project, you must ensure that you have the required software, concepts, and packages installed on your system. Follow the steps below to set up your environment:

1. Anaconda Navigator:
Anaconda Navigator is a powerful platform for data science and machine learning applications. It provides tools such as Jupyter Notebook, Spyder, and more. Install Anaconda Navigator on your system by following the instructions in the installation video.

2. Required Packages:

Python Libraries:

- **Numpy:** An open-source numerical Python library used for multidimensional array and matrix operations. Install it by opening the Anaconda Prompt as an administrator and typing: **pip install numpy**

- **Pandas:** A data manipulation library for Python. Install it using: **pip install pandas**

- **Scikit-learn:** A free machine learning library for Python that supports various algorithms. Install it with: **pip install scikit-learn**

- **TensorFlow and Keras:** Install specific versions for compatibility with the provided code: **pip install tensorflow==2.3.2**
- **pip install keras==2.3.1**

- **Flask:** A web framework used for building web applications. Install it by running:
- **pip install Flask**

Additional Concepts:

- **Jupyter Notebook and Spyder:** The project utilizes Jupyter Notebook and Spyder for development. Make sure you are familiar with how to use these tools through Anaconda Navigator. Refer to the installation video for guidance.

3. GPU Support (Optional):
If you have access to a GPU, consider installing the GPU version of TensorFlow for faster training. Follow the TensorFlow documentation for GPU installation instructions.

Deep Learning Concepts

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

CNN Basic

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:

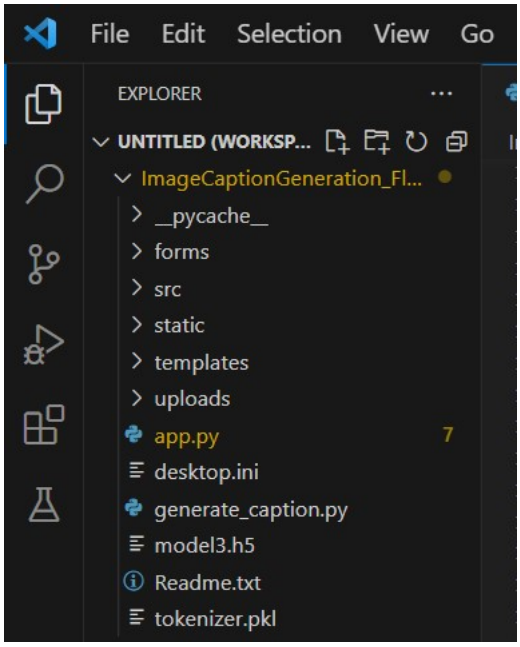
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with flask application.
- VGG16 Model analyse the image, then LSTM is used to process the captions in form of text, and prediction is showcased on the Flask UI.

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



- 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
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

```
# extracting features from the images
features = {}
directory = '/kaggle/input/small-flicker-data-for-image-captioning/flickr1k/images'
for img_name in tqdm(os.listdir(directory)):
    #loading images from files
    img_path = os.path.join(directory , img_name)
    image = load_img(img_path , target_size=(224,224))
    # converting image to numpy array
    image = img_to_array(image)
    #resshape img for vgg16
    image = image.reshape(1 , image.shape[0] , image.shape[1] , image.shape[2])
    #preprocess for vgg
    image = preprocess_input(image)
    #extracting features
    feature = feature_extractor.predict(image , verbose=0)
    #get image ids
    image_id = img_name.split('.')[0]
    #storing features
    features[image_id] = feature
```

Activity 3: 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 4: 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 5: 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

```
# loading vgg16
pre_trained_model = VGG16()
feature_extractor = Model(inputs = pre_trained_model.inputs , outputs = pre_trained_model.layers[-2].output)
```

```
feature_extractor.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (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

block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

flatten (Flatten)           (None, 25088)             0

fc1 (Dense)                 (None, 4096)              102764544

fc2 (Dense)                 (None, 4096)              16781312

=================================================================
Total params: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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.

```
# extracting features from the images
features = {}
directory = '/kaggle/input/small-flicker-data-for-image-captioning/flickr1k/images'
for img_name in tqdm(os.listdir(directory)):
    #loading images from files
    img_path = os.path.join(directory , img_name)
    image = load_img(img_path , target_size=(224,224))
    # converting image to numpy array
    image = img_to_array(image)
    #resshape img for vgg16
    image = image.reshape(1 , image.shape[0] , image.shape[1] , image.shape[2])
    #preprocess for vgg
    image = preprocess_input(image)
    #extracting features
    feature = feature_extractor.predict(image , verbose=0)
    #get image ids
    image_id = img_name.split('.')[0]
    #storing features
    features[image_id] = feature
```

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)
```

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['1001773457_577c3a7d70']
```

```
['A black dog and a spotted dog are fighting',
 'A black dog and a tri-colored dog playing with each other on the road .',
 'A black dog and a white dog with brown spots are staring at each other in the street .',
 'Two dogs of different breeds looking at each other on the road .',
 'Two dogs on pavement moving toward each other .']
```

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

```
# after preprocess of text
mapping['1001773457_577c3a7d70']
```

```
['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']
```

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 initialising the tokenizer function to tokenize all the captions by passing all the captions to it.

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

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

```
32
```

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.

```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

```
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 True:
        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
```

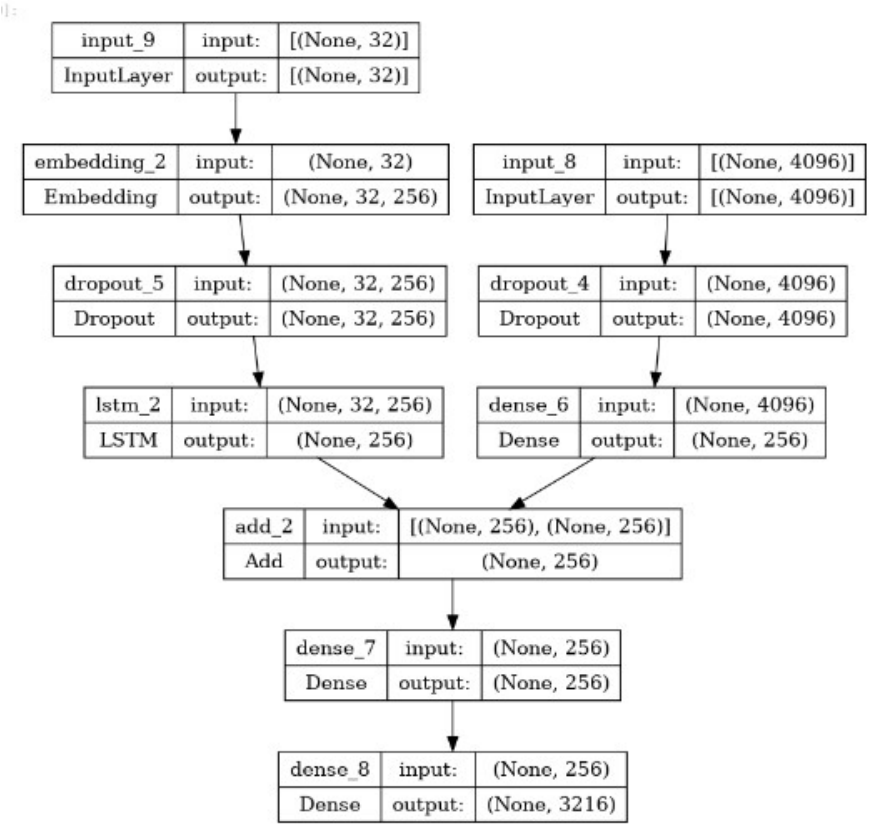
Analysing the trained model by looking at the accuracy through the epocs:

```
from tensorflow.keras.optimizers import Adam

epochs = 30
batch_size = 32
steps = len(train)
learning_rate = 0.001  # You can adjust this value

for i in range(epochs):
    # create datagen
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)

    # Compile the model with a specific learning rate
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=['accuracy'])

    # Fit the model
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

```
900/900 [==============================] - 136s 145ms/step - loss: 385.5274 - accuracy: 0.1437
900/900 [==============================] - 133s 142ms/step - loss: 3.2206 - accuracy: 0.3328
900/900 [==============================] - 132s 141ms/step - loss: 2.1238 - accuracy: 0.4874
900/900 [==============================] - 129s 138ms/step - loss: 1.5777 - accuracy: 0.5989
900/900 [==============================] - 130s 139ms/step - loss: 1.2962 - accuracy: 0.6684
900/900 [==============================] - 131s 140ms/step - loss: 1.3439 - accuracy: 0.7083
900/900 [==============================] - 133s 141ms/step - loss: 1.0013 - accuracy: 0.7311
900/900 [==============================] - 132s 142ms/step - loss: 0.9888 - accuracy: 0.7417
900/900 [==============================] - 133s 143ms/step - loss: 0.9858 - accuracy: 0.7459
900/900 [==============================] - 132s 141ms/step - loss: 0.9290 - accuracy: 0.7490
900/900 [==============================] - 133s 142ms/step - loss: 0.8973 - accuracy: 0.7511
900/900 [==============================] - 132s 141ms/step - loss: 1.0168 - accuracy: 0.7529
900/900 [==============================] - 131s 140ms/step - loss: 0.8770 - accuracy: 0.7536
900/900 [==============================] - 130s 139ms/step - loss: 0.8766 - accuracy: 0.7541
900/900 [==============================] - 130s 139ms/step - loss: 0.9198 - accuracy: 0.7540
900/900 [==============================] - 134s 143ms/step - loss: 0.8973 - accuracy: 0.7541
900/900 [==============================] - 133s 142ms/step - loss: 0.9694 - accuracy: 0.7551
900/900 [==============================] - 132s 142ms/step - loss: 0.8744 - accuracy: 0.7546
900/900 [==============================] - 133s 142ms/step - loss: 0.8666 - accuracy: 0.7546
900/900 [==============================] - 131s 140ms/step - loss: 0.8861 - accuracy: 0.7559
900/900 [==============================] - 132s 141ms/step - loss: 0.8733 - accuracy: 0.7564
900/900 [==============================] - 130s 139ms/step - loss: 0.8508 - accuracy: 0.7562
900/900 [==============================] - 131s 139ms/step - loss: 0.8711 - accuracy: 0.7570
900/900 [==============================] - 132s 141ms/step - loss: 0.8808 - accuracy: 0.7571
900/900 [==============================] - 131s 139ms/step - loss: 0.8485 - accuracy: 0.7568
900/900 [==============================] - 132s 141ms/step - loss: 0.8661 - accuracy: 0.7569
900/900 [==============================] - 132s 141ms/step - loss: 0.8591 - accuracy: 0.7570
900/900 [==============================] - 133s 140ms/step - loss: 0.8458 - accuracy: 0.7575
900/900 [==============================] - 132s 141ms/step - loss: 0.8446 - accuracy: 0.7576
900/900 [==============================] - 131s 140ms/step - loss: 0.8564 - accuracy: 0.7575
```

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)
```



Activity 6: 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')
```

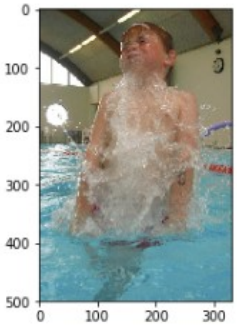
Activity 7: Test The model

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.

```
from nltk.translate.bleu_score import corpus_bleu
actual , predict = list() , list()
key = '17273391_55cfc7d3d4'
captions = mapping[key]
y_pred = predict_caption(model , features[key] , tokenizer , max_length)
    #split into words
y_real = [caption.split() for caption in captions]
y_pred = y_pred.split()

actual.append(y_real)
predict.append(y_pred)


print('bleu 1 : %f' % corpus_bleu(actual , predict , weights=(1.0 , 0 , 0 , 0)))
print('bleu 2 : %f' % corpus_bleu(actual , predict , weights=(0.5, 0.5 , 0 , 0)))
```

```
bleu 1 : 0.454545
bleu 2 : 0.213201
```

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



young boy in pool wearing blue shorts is splashing in the water

two motorcycles on the track

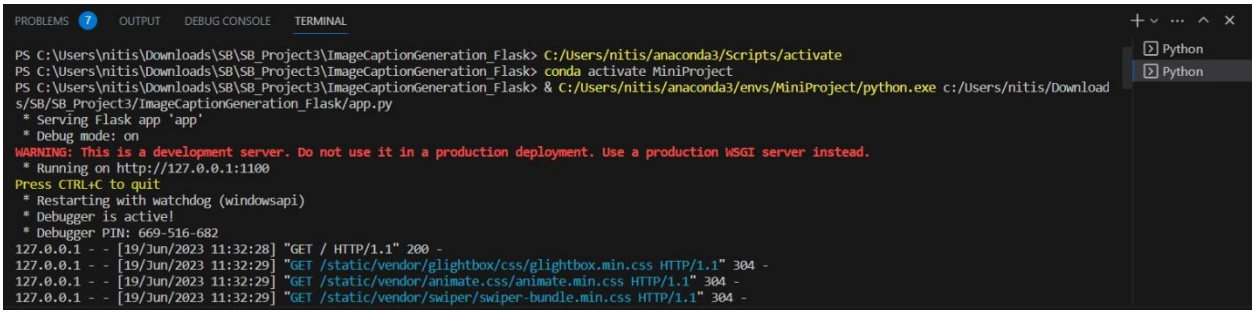
man in yellow kayak paddling through rough waters

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.

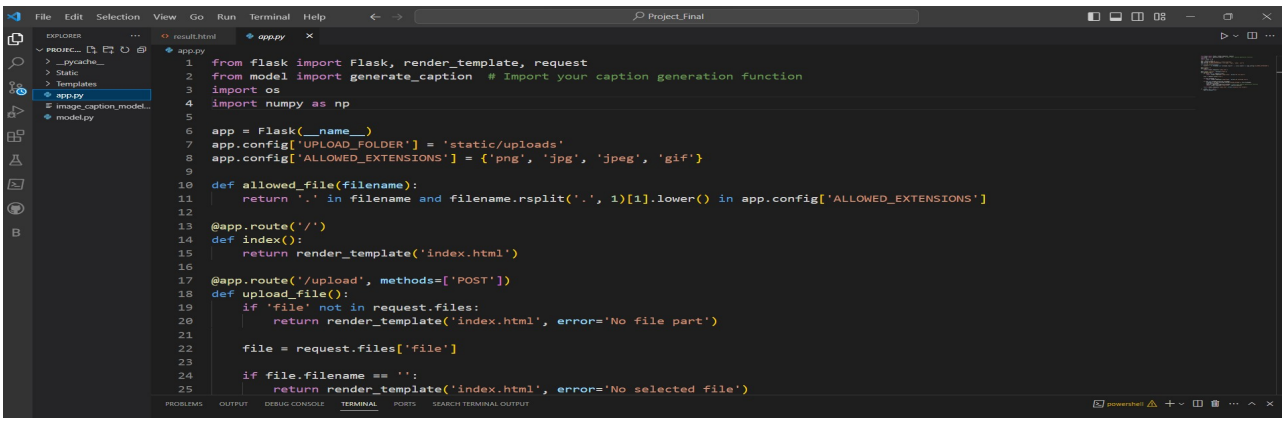
Activity 1: Create HTML Pages

- 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
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- Link :CSS , JS



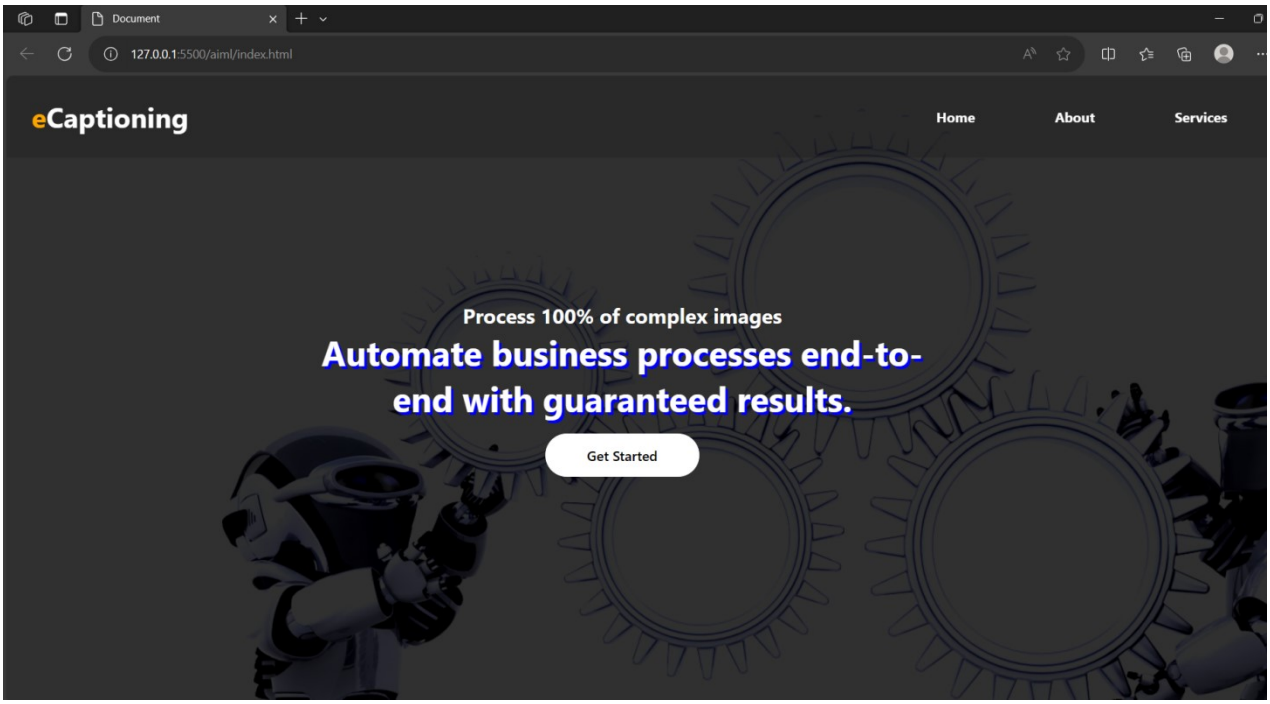
Create app.py (Python Flask) file: -

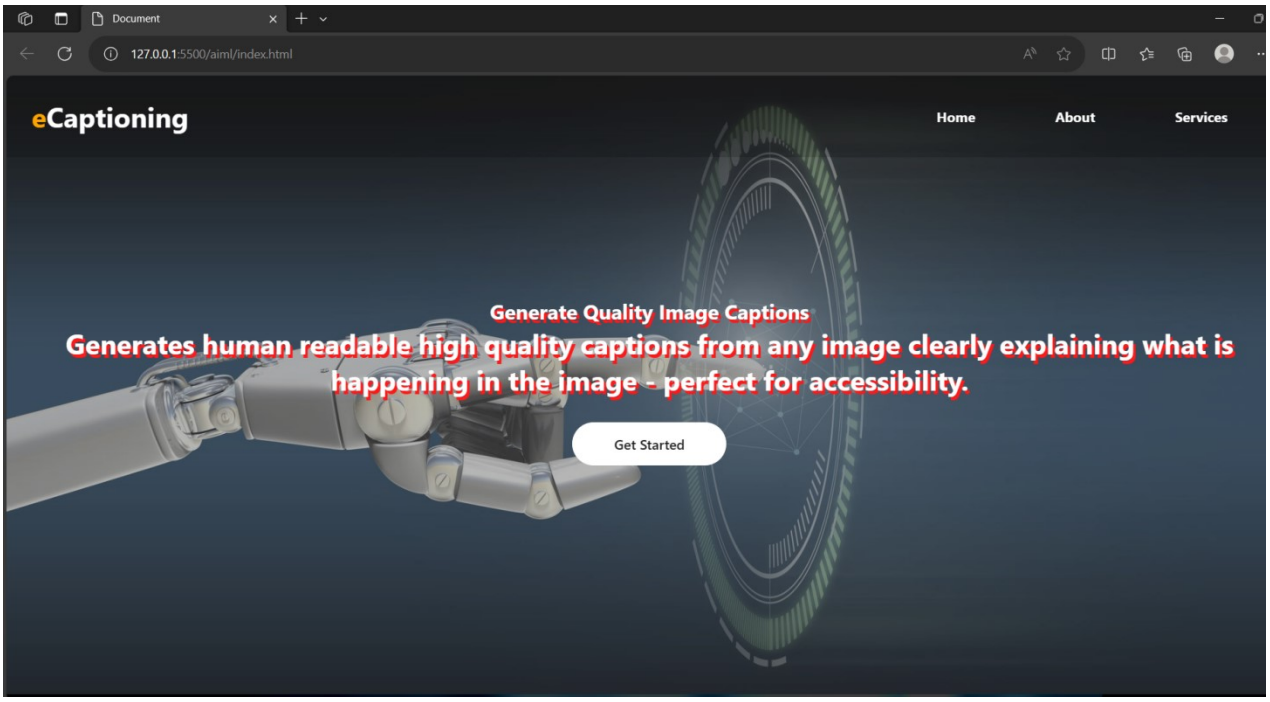
Write below code in Flask app.py python file script to run Micro-Organism Classification Project.

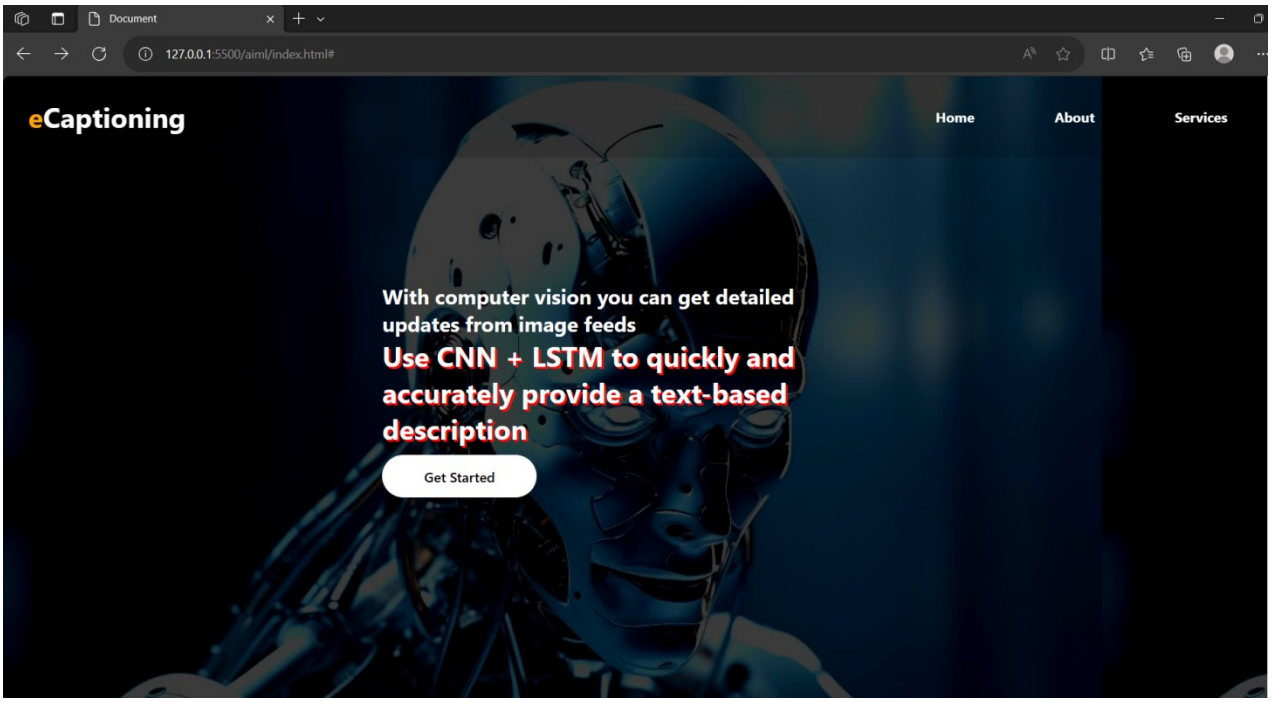


```python
@app.route('/upload', methods=['POST'])
def upload_file():
    if 'file' not in request.files:
        return render_template('index.html', error='No file part')

    file = request.files['file']

    if file.filename == '':
        return render_template('index.html', error='No selected file')

    if file and allowed_file(file.filename):
        filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filename)
        caption = generate_caption(filename)  # Call your caption generation function
        return render_template('index.html', caption=caption)

    return render_template('index.html', error='Invalid file format')

if __name__ == '__main__':
    app.run(debug=True)
```

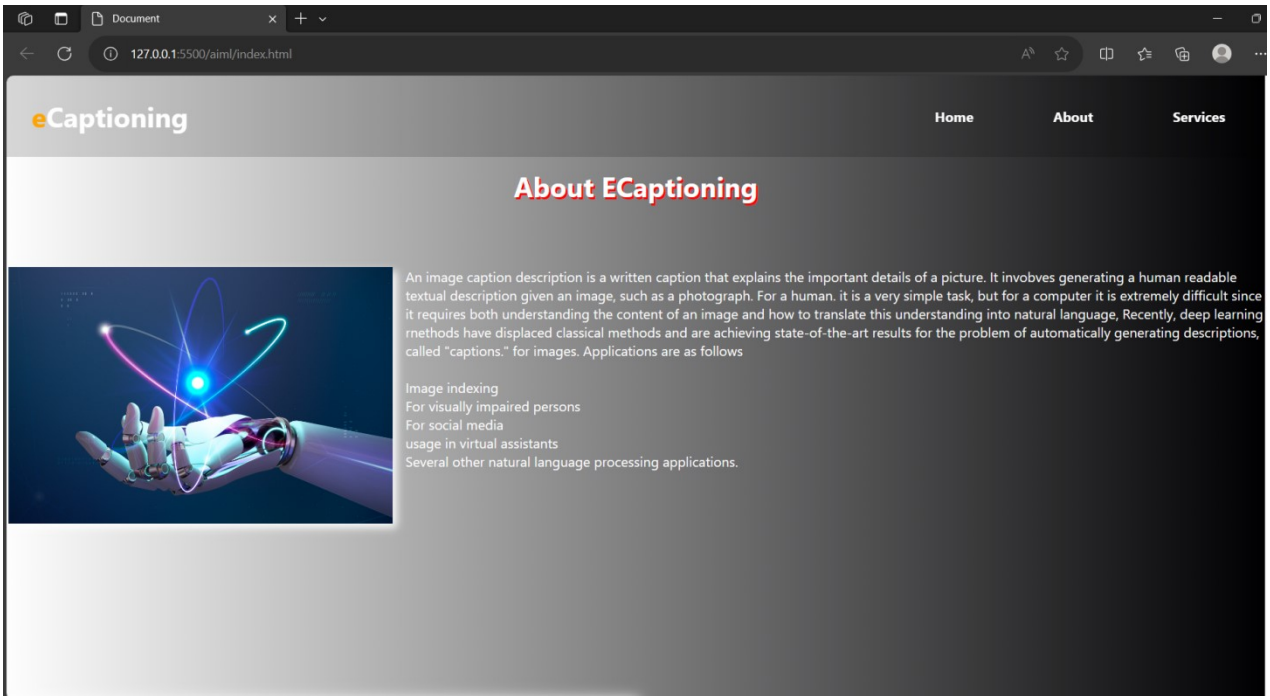
Index.html is displayed below (Webpage named "eCaptioning"):



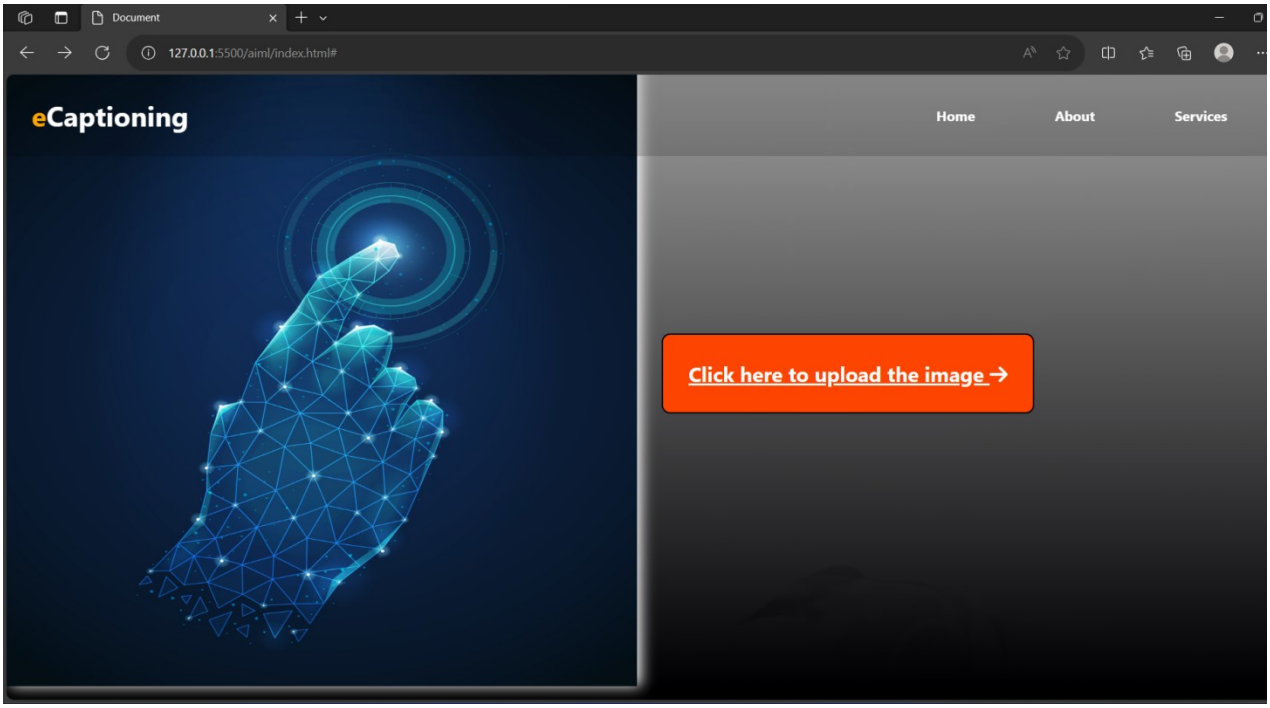

Document
127.0.0.1:5500/aiml/index.html
eCaptioning
Home
About
Services
Generate Quality Image Captions
Generates human readable high quality captions from any image clearly explaining what is happening in the image - perfect for accessibility.
Get Started


Document
127.0.0.1:5500/aiml/index.html#
eCaptioning
Home
About
Services
With computer vision you can get detailed updates from image feeds
Use CNN + LSTM to quickly and accurately provide a text-based description
Get Started

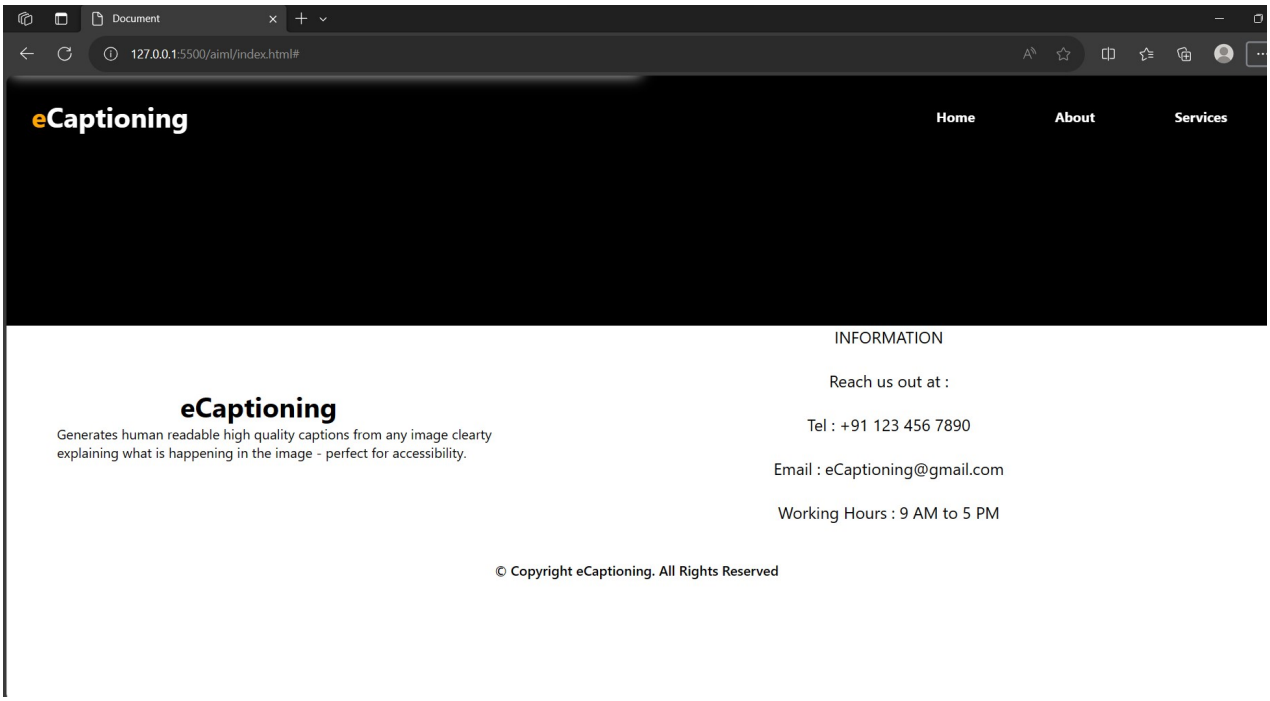
About Section is displayed below:



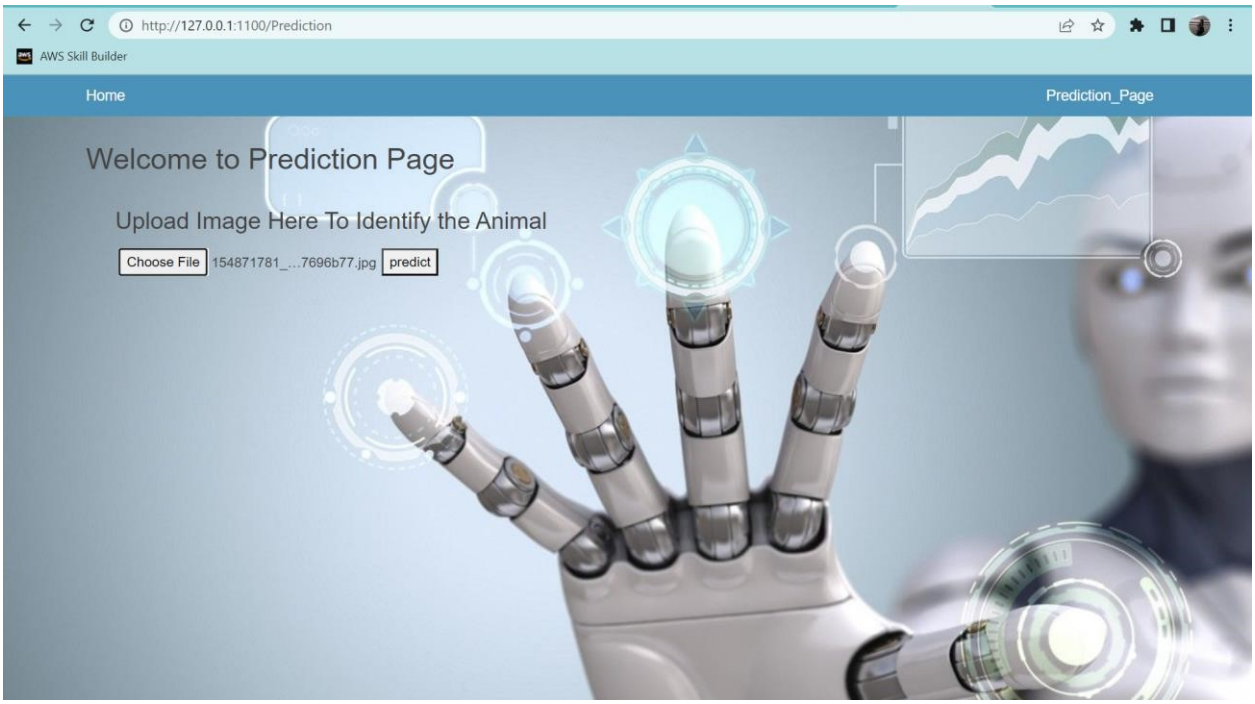
Services section is displayed below with a click button.

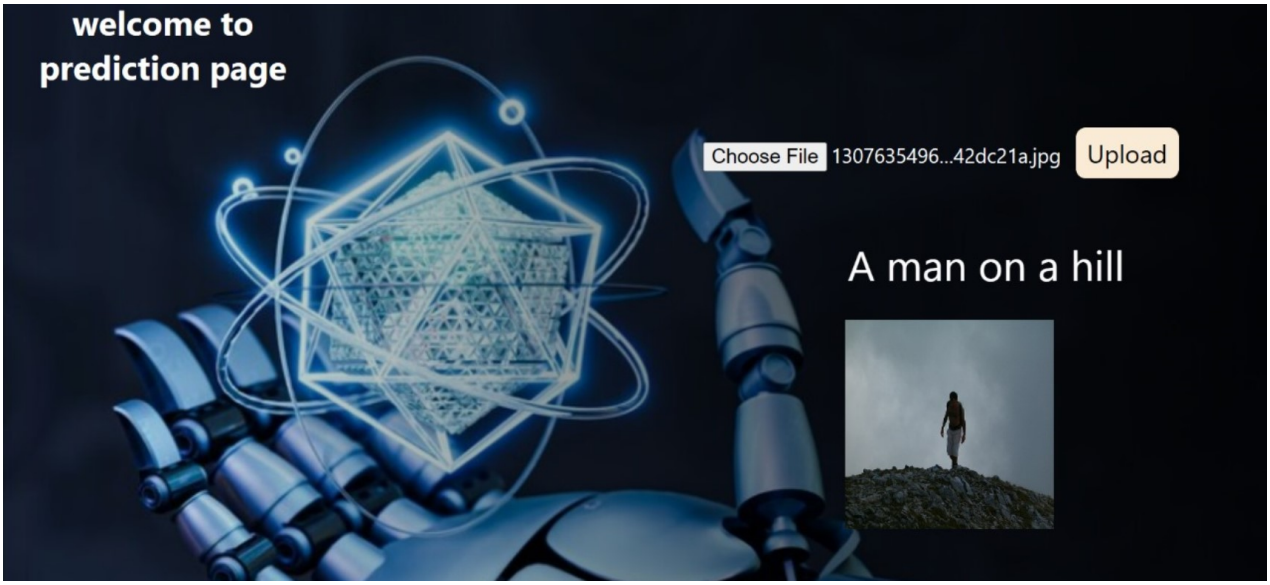


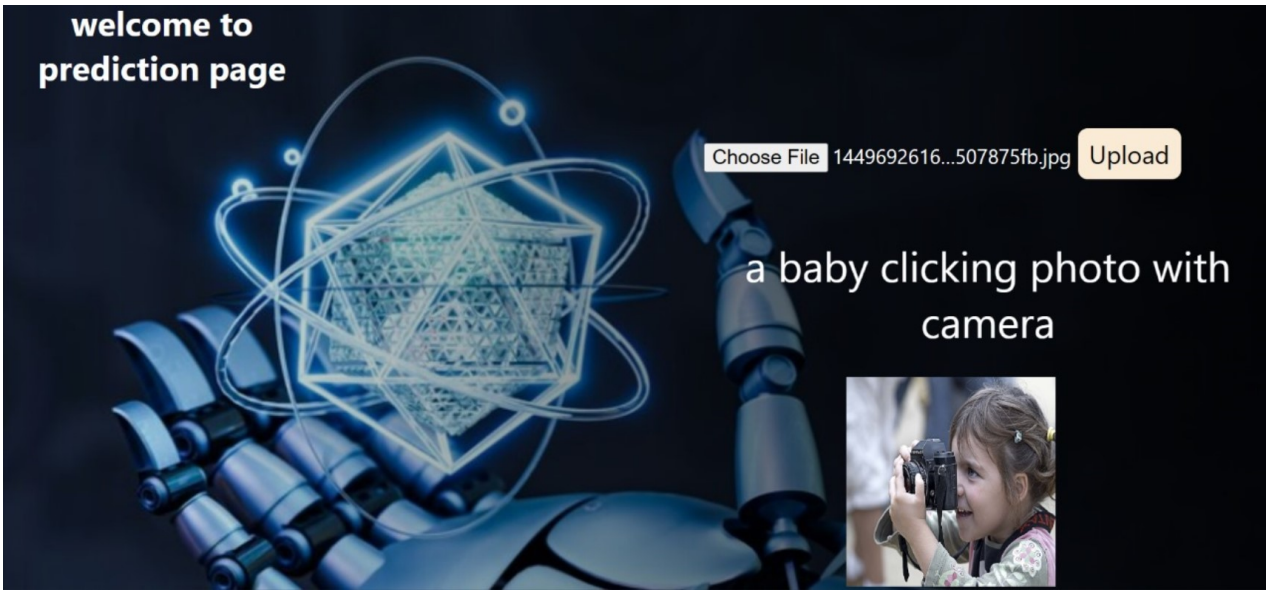
Finally, the Footer of the Webpage along with the choosing of Image for prediction.

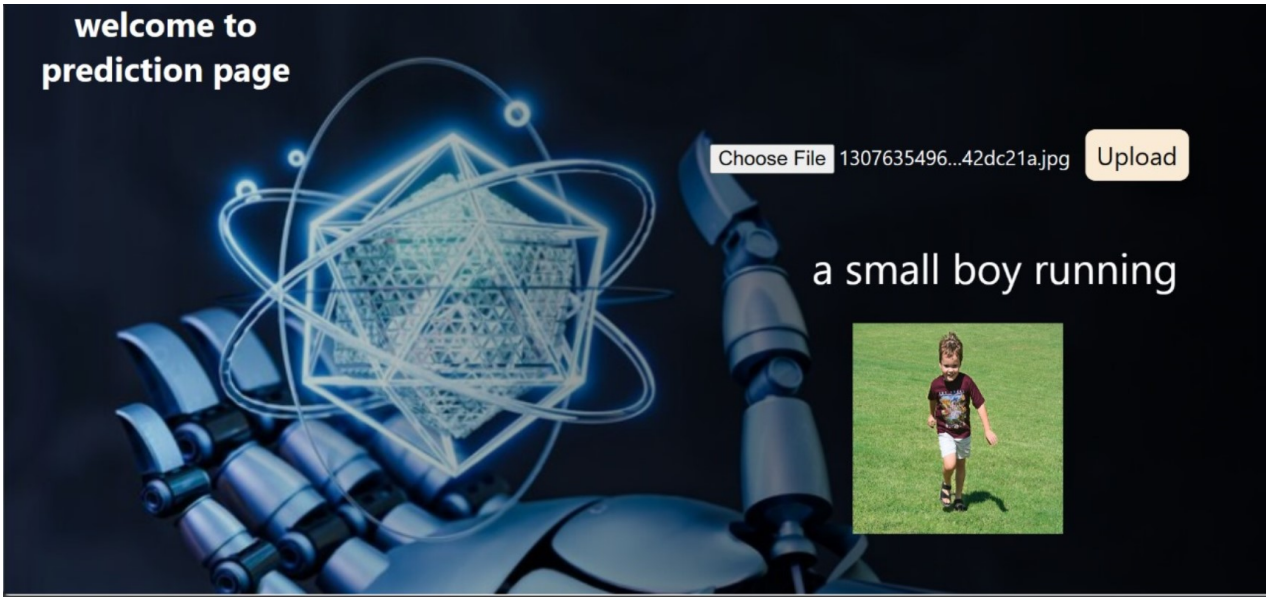


Final Predicted Output (after I click on the Predict Button) is displayed as follows:




welcome to
prediction page
Choose File
1307635496...42dc21a.jpg
Upload
A man on a hill


welcome to
prediction page
Choose File
1449692616...507875fb.jpg
Upload
a baby clicking photo with
camera


welcome to
prediction page
Choose File
1307635496...42dc21a.jpg
Upload
a small boy running

****Thank You****