

Phase 4

Date	26 November 2022
Team ID	591955
Project Name	IMAGE CAPTION GENERATION

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

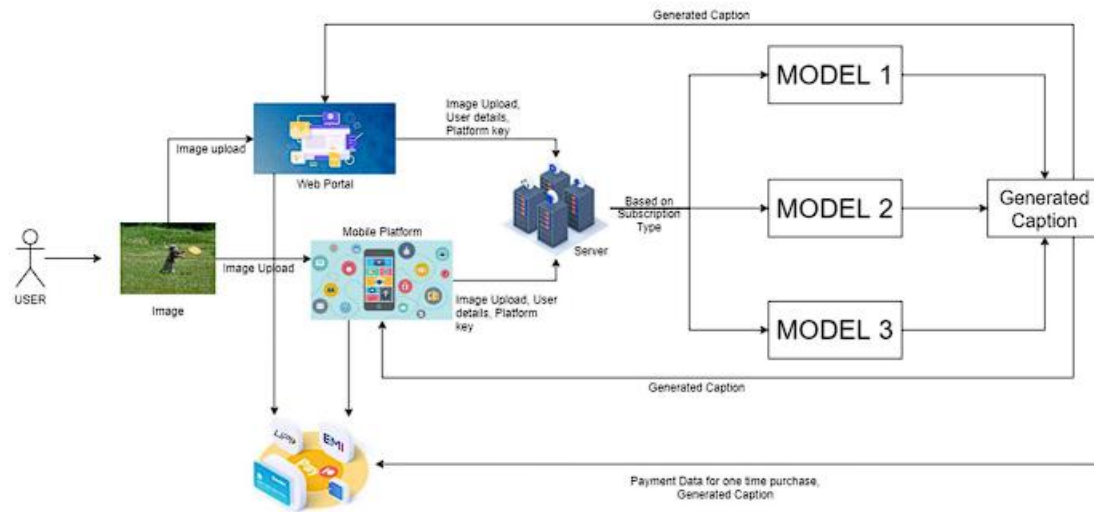
Introduction:

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.

Solution 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 Visual Studio Code

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
- Use any options, from CMD, Powershell or Terminal of VS Code
- 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 and click enter.
- Type pip install keras 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.

CNN Basic

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

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:

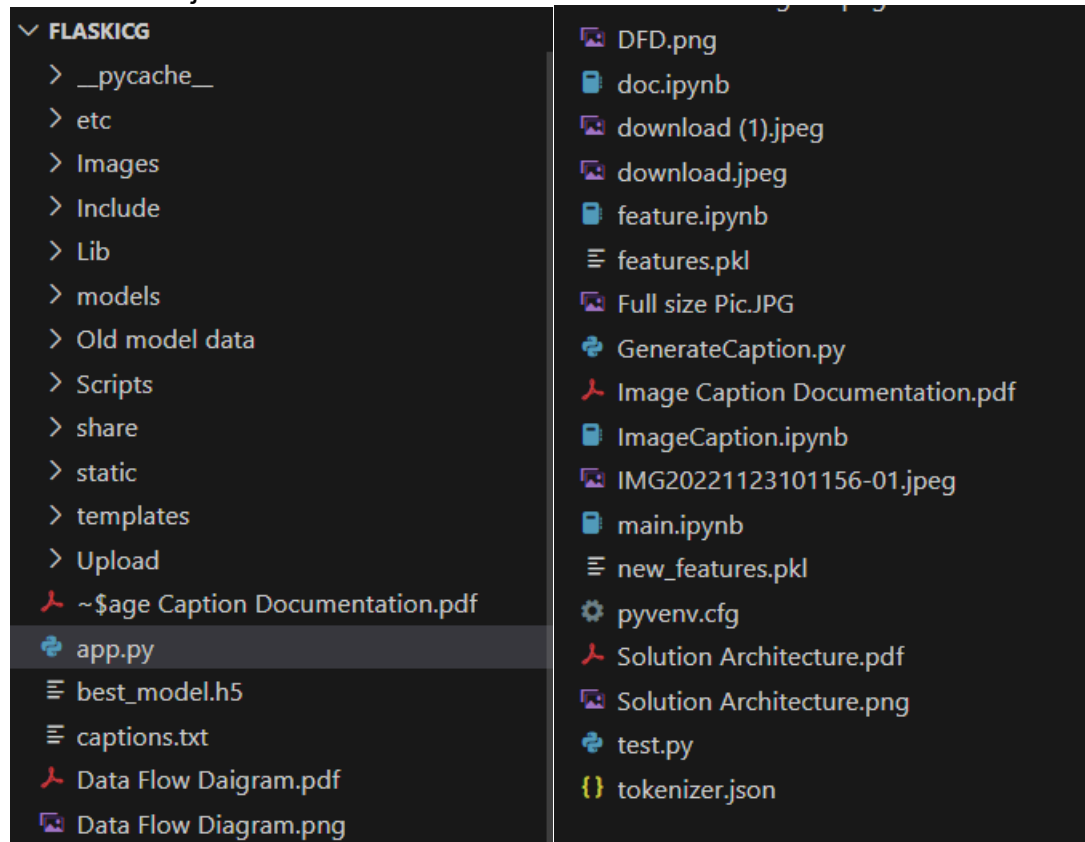
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with flask application.
- VGG16 Model analyse the image, then LSTM is used to process the captions in form of text, and prediction is showcased on the Flask UI.

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 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.json.
- The templates folder contains index.html and required 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 # handling the files
import pickle # storing numpy features
import numpy as np
from tqdm.notebook import tqdm # how much data is process till now

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

[1] ✓ 4.6s Python
... WARNING:tensorflow:From f:\Projects\flask\lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.s
```

Activity 2: Exploratory Data Analysis

```
[16] def txt_vocab(descriptions):
    vocab = set()
    for key in descriptions.keys():
        [vocab.update(d.split()) for d in descriptions[key]]
    return vocab
Python

[18] def save_description(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc)
        data = "\n".join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
Python

dataset_text = "F:\Projects\FlaskICG\captions.txt"
dataset_images = "F:\Projects\FlaskICG\images"

#to prepare our text data
filename = dataset_text

#loading the file that contains all data
#map them into descriptions dictionary
descriptions = img_capt(filename, "img_name.txt")

cleaned_txt = txt_clean(descriptions)
vocabulry = txt_vocab(descriptions)
print("Length of vocabulary", len(vocabulry))

save_description(cleaned_txt, "description.txt")
Python

... Length of vocabulary 8763
```

Activity 3: Preprocessing of Text Data

```
Processing of Text Data

Now we start processing the text data

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

tokenizer_json = tokenizer.to_json()
with open('tokenizer.json', 'w', encoding='utf-8') as f:
    f.write(tokenizer_json)
Python

[21] vocab_size

... 8484

No. of unique words

[22] # get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
Python

... 35
```

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.

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

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.

```
features = {}
directory = os.path.join(root_dir, 'images')

# Initialize an empty list to store image paths
image_paths = []

# Define batch size
batch_size = 32

# Iterate through the images in the directory
for img_name in tqdm(os.listdir(directory)):
    # Load the image from file
    img_path = os.path.join(directory, img_name)
    image_paths.append(img_path)

# If the batch size is reached, process the batch
if len(image_paths) == batch_size:
    # Load and preprocess the batch of images
    batch_images = [img_to_array(load_img_path, target_size=(224, 224)) for path in image_paths]
    batch_images = np.array(batch_images)
    batch_images = preprocess_input(batch_images)

    # Extract features for the batch
    batch_features = model.predict(batch_images, verbose=0)

    # Store features for each image in the batch
    for i, image_id in enumerate(image_paths):
        features[image_id] = batch_features[i]

    # Clear the image paths for the next batch
    image_paths = []

# Process any remaining images (if the total number is not a multiple of the batch size)
if image_paths:
    batch_images = [img_to_array(load_img_path, target_size=(224, 224)) for path in image_paths]
    batch_images = np.array(batch_images)
    batch_images = preprocess_input(batch_images)

    batch_features = model.predict(batch_images, verbose=0)

    for i, image_id in enumerate(image_paths):
        features[image_id] = batch_features[i]
```

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(tokens) < 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)
```

- Dictionary 'mapping' is created with key as image_id and values as the corresponding caption text
- Same image may have multiple captions, if **image_id not in mapping: mapping[image_id] = []** creates a list for appending captions to the corresponding image

Now let us see the no. of images loaded

```
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(100(captions)):
            # keep 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

[11]:
```

Defined to clean and convert the text for quicker process and better results

Let us visualize the text before and after cleaning

```
# before preprocess of text
mapping['10002628201_693088c8de']

[14]:
```

```
... ['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)

[15]:
```

```
# after preprocess of text
mapping['10002628201_693088c8de']

[16]:
```

```
... ['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',
      'startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl 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[10]

[18]:
```

```
... ['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',
      'startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq']
```

Here, the data is divided into train data and test data.

```
# 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:
            # new 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 epochs:

```
# 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)
```

WARNING:tensorflow: From /usr/local/lib/python3.6/dist-packages/tensorflow/python/util/ops.py:402: The name tf.nn.rnn_cell.LSTMCell is deprecated. Please use tf.nn.rnn_cell.LSTMCell instead.

```
227/227 [=====] - 3111 1s/step - loss: 5.2371
227/227 [=====] - 3246 1s/step - loss: 4.8461
227/227 [=====] - 3255 1s/step - loss: 3.6865
227/227 [=====] - 2846 1s/step - loss: 3.3427
227/227 [=====] - 2686 1s/step - loss: 3.1445
227/227 [=====] - 2675 1s/step - loss: 2.9961
227/227 [=====] - 2795 1s/step - loss: 2.8849
227/227 [=====] - 2795 1s/step - loss: 2.7883
227/227 [=====] - 2935 1s/step - loss: 2.7841
227/227 [=====] - 3406 1s/step - loss: 2.6330
227/227 [=====] - 3375 1s/step - loss: 2.5682
227/227 [=====] - 3346 1s/step - loss: 2.5678
227/227 [=====] - 3386 1s/step - loss: 2.4522
227/227 [=====] - 3386 1s/step - loss: 2.4833
227/227 [=====] - 3386 1s/step - loss: 2.3668
227/227 [=====] - 3386 1s/step - loss: 2.3188
227/227 [=====] - 3325 1s/step - loss: 2.2825
227/227 [=====] - 2956 1s/step - loss: 2.2442
227/227 [=====] - 2866 1s/step - loss: 2.2143
227/227 [=====] - 3146 1s/step - loss: 2.1852
```

Visualising Model Summary:

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 35)]	0	[]
input_2 (InputLayer)	[(None, 4096)]	0	[]
embedding (Embedding)	(None, 35, 256)	2171904	['input_3[0][0]']
dropout (Dropout)	(None, 4096)	0	['input_2[0][0]']
dropout_1 (Dropout)	(None, 35, 256)	0	['embedding[0][0]']
dense (Dense)	(None, 256)	1048832	['dropout[0][0]']
lstm (LSTM)	(None, 256)	525312	['dropout_1[0][0]']
add (Add)	(None, 256)	0	['dense[0][0]', 'lstm[0][0]']
dense_1 (Dense)	(None, 256)	65792	['add[0][0]']

...
Trainable params: 5992228 (22.86 MB)
Non-trainable params: 0 (0.00 Byte)

None

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...

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')
```

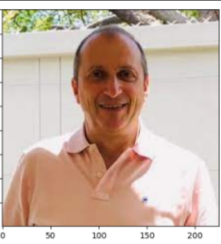
Activity 8: Test The model

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.

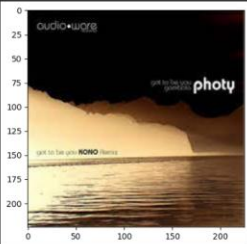
```
from nltk.translate.bleu_score import corpus_bleu  
# calculate with test data  
actual, predicted = list(), list()  
  
for key in tqdm(test):  
    # get actual caption  
    captions = mapping[key]  
    # predict the caption for image  
    y_pred = predict_caption(model, features[key], tokenizer, max_length)  
    # split into words  
    actual_captions = [caption.split() for caption in captions]  
    y_pred = y_pred.split()  
    # append to the list  
    actual.append(actual_captions)  
    predicted.append(y_pred)  
# calculate 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)))
```

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

```
generate_caption("download.jpeg")  
[1]:  
download ('download'): array([[0. , 0.26159553, 0. , ..., 0. , 0. , 0. ,  
1.7319424 ], dtype=float32])  
-----Predicted-----  
startseq  
0  
25  
50  
75  
100  
125  
150  
175  
200  
0 50 100 150 200
```



```
generate_caption("download (1).jpeg")  
[1]:  
download ('download (1)': array([[0. , 0. , 0. , ..., 4.891528, 0. , 0. ,  
dtype=float32])  
-----Predicted-----  
startseq  
0  
25  
50  
75  
100  
125  
150  
175  
200  
0 50 100 150 200
```



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

```
1 # Importing the flask module in the current script.
2 from flask import Flask, request, render_template
3
4 # Creating an instance of the Flask class.
5 app = Flask(__name__)
6
7 # Route for the home page.
8 @app.route('/')
9 def home():
10     return render_template("index.html")
11
12 # Route for the image prediction page.
13 @app.route('/predict', methods=['POST'])
14 def predict():
15     # Getting the image file from the request.
16     image_file = request.files['image']
17     # Saving the image file to a temporary directory.
18     image_path = os.path.join(tempfile.gettempdir(), image_file.filename)
19     image_file.save(image_path)
20     # Loading the image file into the model.
21     image = image_loader(image_path)
22     # Predicting the class of the image.
23     prediction = model.predict(image)
24     # Returning the prediction to the user.
25     return render_template("predict.html", prediction=prediction)
26
27 # Running the flask application.
28 if __name__ == '__main__':
29     app.run(debug=True)
```

```
1 # Importing the flask module in the current script.
2 from flask import Flask, request, render_template
3
4 # Creating an instance of the Flask class.
5 app = Flask(__name__)
6
7 # Route for the home page.
8 @app.route('/')
9 def home():
10     return render_template("index.html")
11
12 # Route for the image prediction page.
13 @app.route('/predict', methods=['POST'])
14 def predict():
15     # Getting the image file from the request.
16     image_file = request.files['image']
17     # Saving the image file to a temporary directory.
18     image_path = os.path.join(tempfile.gettempdir(), image_file.filename)
19     image_file.save(image_path)
20     # Loading the image file into the model.
21     image = image_loader(image_path)
22     # Predicting the class of the image.
23     prediction = model.predict(image)
24     # Returning the prediction to the user.
25     return render_template("predict.html", prediction=prediction)
26
27 # Running the flask application.
28 if __name__ == '__main__':
29     app.run(debug=True)
```


Create app.py (Python Flask) file: -

```
app.py > _
1 from flask import Flask, render_template, request, redirect, url_for
2 import os
3 from GenerateCaption import generate_caption
4
5 app = Flask(__name__)
6
7 logged = False
8 admins = [
9     ("username": "admin", "password": "admin"),
10 ]
11
12 @app.route("/", methods=["GET"])
13 def home():
14     return render_template("home.html")
15
16 @app.route("/about", methods=["GET"])
17 def about():
18     return render_template("about.html")
19
20 @app.route("/get_captions", methods=["GET", "POST"])
21 def get_captions():
22     global logged
23
24     if logged:
25         if request.method == "POST":
26             # Handle the uploaded image and generate caption
27             image_file = request.files["image"]
28
29             # Specify the upload folder with the correct path
30             upload_folder = r"F:\Projects\FlaskICG\Upload"
31
32             # Ensure the upload folder exists
33             os.makedirs(upload_folder, exist_ok=True)
34
35             file_path = os.path.join(upload_folder, image_file.filename)
36             image_file.save(file_path)
37
38             # Correct variable name used for generate_caption
39             caption = generate_caption(file_path)
40
41             print(caption)
42             return render_template("get_captions.html", caption=caption, logged=logged)
43
44             return render_template("get_captions.html", logged=logged)
45
46         else:
47             return redirect(url_for("home"))
48
49 @app.route("/login", methods=["GET", "POST"])
50 def login():
51     global logged, admins
52     if logged:
53         return redirect(url_for("get_captions"))
54     elif request.method == "POST":
55         username = request.form.get("username", "")
56         password = request.form.get("password", "")
57
58         print(username)
59         print(password)
60
61         for each in admins:
62             if (each["username"] == username and (each["password"] == password)):
63                 logged = True
64                 print("here")
65                 return redirect(url_for("get_captions"))
66         else:
67             return render_template("login.html")
68
69 @app.route("/signout", methods=["GET", "POST"])
70 def signout():
71     global logged
72     logged = False
73     return redirect(url_for("home"))
74
75 if __name__ == "__main__":
76     app.run(debug=True)
```