Image Caption Generation: A Fusion of CNN and LSTM Deployed through Flask for Web Interface

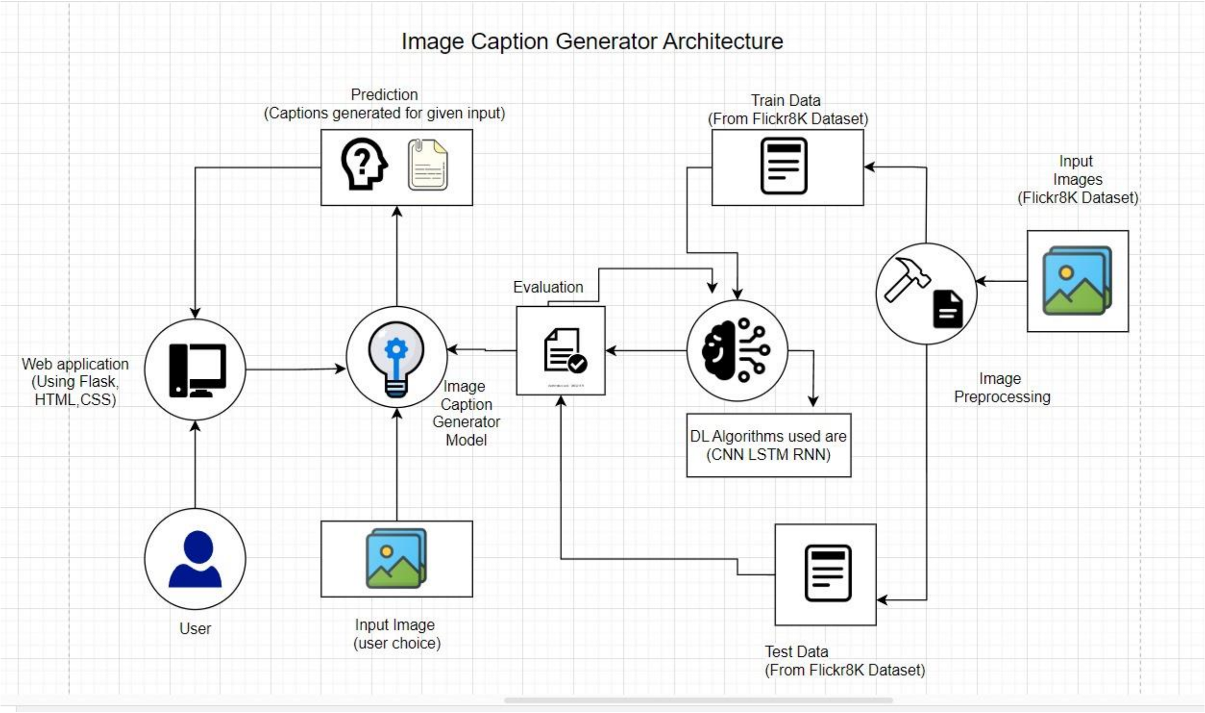
Introduction:

In the rapidly evolving landscape of artificial intelligence, the convergence of computer vision and natural language processing has unlocked unprecedented possibilities in understanding and interpreting visual content. This project delves into the realm of image caption generation, a captivating intersection where Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks harmonize to bridge the gap between visual perception and linguistic expression.

The ability to generate coherent and contextually relevant captions for images is not merely an AI feat but a gateway to a multitude of applications, from enhancing accessibility for the visually impaired to revolutionizing content indexing for search engines. The project sets out to explore the synergy between CNN, renowned for its prowess in image feature extraction, and LSTM, a stalwart in processing sequential data. By combining these two powerful neural network architectures, our aim is to create a model that not only discerns intricate patterns and objects within images but also crafts linguistically sound and contextually meaningful captions.

This report unfolds the various stages of our journey, from the preprocessing of image data to the intricacies of training CNN and LSTM networks. We delve into the challenges encountered, the methodologies employed, and the outcomes achieved. As we navigate through the technical intricacies, the goal remains clear: to contribute to the evolving landscape of AI applications, where machines not only see but comprehend and communicate the visual narratives encapsulated in images.

Technical Architecture:



Pre-requisites:

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

1. Jupyter notebook
2. VS Code

To build Machine learning models you must require the following packages

- Numpy: 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: 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” and click enter.
    - Type “pip install keras” and click enter.
    - Type “pip install Flask” and click enter.
    - Type “pip install opencv-python” and click enter.

Deep Learning Concepts

CNN (Convolutional Neural Network): A Convolutional Neural Network (CNN) represents a category of deep neural networks primarily utilized for the analysis of visual information. It excels in tasks related to visual imagery, employing convolutional layers to efficiently capture hierarchical features within images.

LSTM (Long Short-Term Memory): LSTM, a type of recurrent neural network (RNN), stands out in handling sequential data. Unlike traditional neural networks, LSTMs are adept at retaining information over extended sequences, making them valuable for tasks involving temporal dependencies, such as language processing.

Flask: Flask stands as a popular Python web framework, serving as a third-party Python library dedicated to the development of web applications. It provides a lightweight and flexible environment for creating web-based interfaces and handling user interactions.

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:

1. **User Interaction:**
   - The user interacts with the UI to choose the image.
2. **Flask Integration:**
   - The chosen image is analyzed by the model integrated with the Flask application.
3. **LSTM Caption Processing:**
   - LSTM processes the extracted features, generating captions in textual form.
4. **Prediction Showcase on Flask UI:**
   - The model's prediction, a descriptive caption, is showcased on the Flask UI.

Tasks to Accomplish:

Data Collection: Gather a diverse dataset of images along with captions for training & testing.

Create Train and Test Folders: Organize the dataset into distinct folders for training and testing.

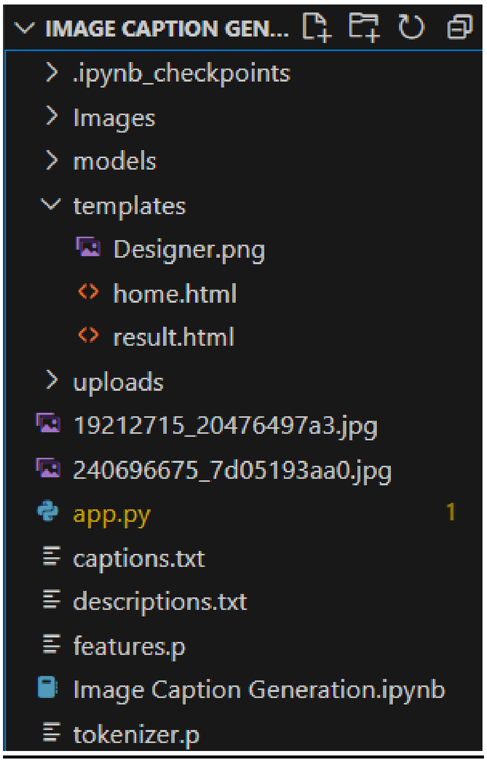
Data Pre-processing: Pre-process the images and textual data for model training.

Model Building:

- Import the Model Building Libraries.
- Initialize the model.
- Add the Input Layer.
- Add Hidden Layers.
- Add the Output Layer.
- Configure the Learning Process.
- Train and test the model.
- Save the Model

Project Structure:

Create a Project folder which contains files as shown below



```
IMAGE CAPTION GEN...
  > .ipynb_checkpoints
  > Images
  > models
  v templates
      Designer.png
      home.html
      result.html
  > uploads
  19212715_20476497a3.jpg
  240696675_7d05193aa0.jpg
  app.py                      1
  captions.txt
  descriptions.txt
  features.p
  Image Caption Generation.ipynb
  tokenizer.p
```

● 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.

Collection of Data

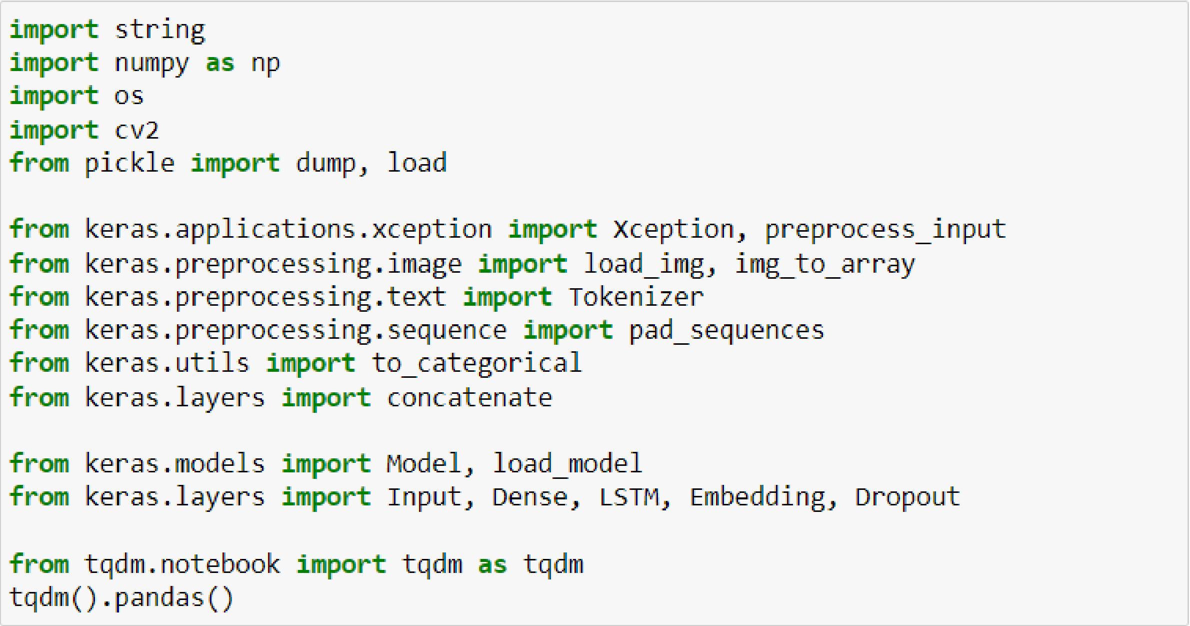
Data Collection: Collect images of events along with 5 captions associated to each image then organized into subdirectories based on their respective names as shown in the project structure. Create folders of images and a text file of captions that need to be recognized.

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

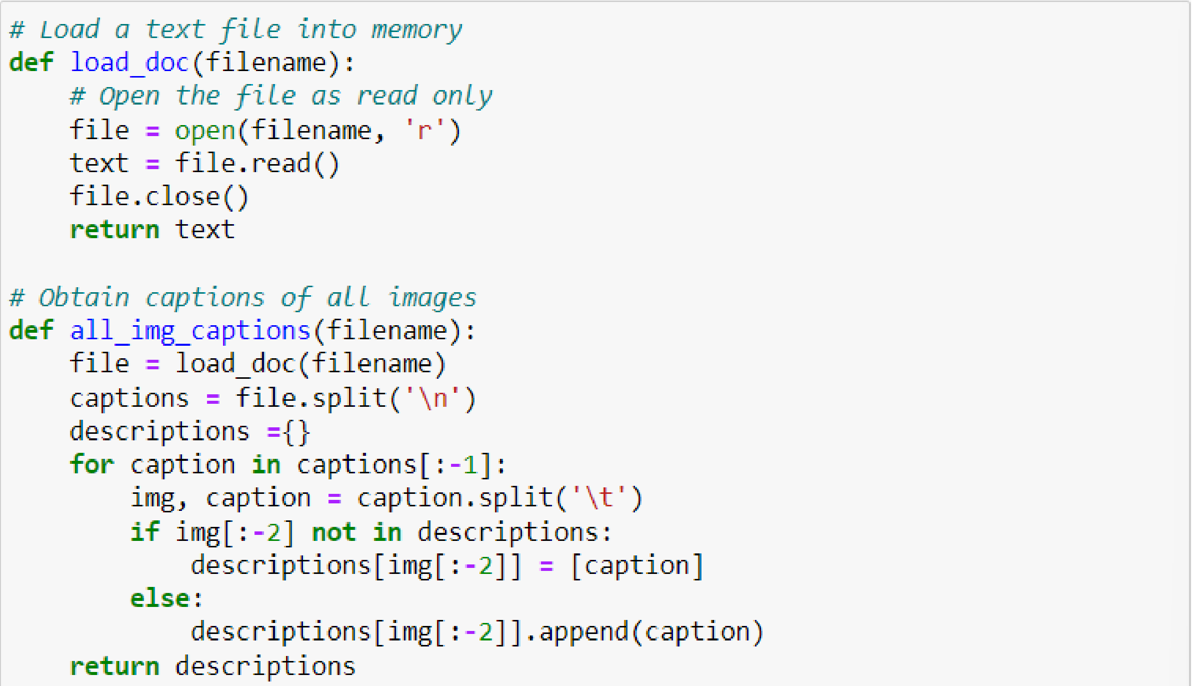
Image Pre-processing and Model Building

1. Importing necessary modules



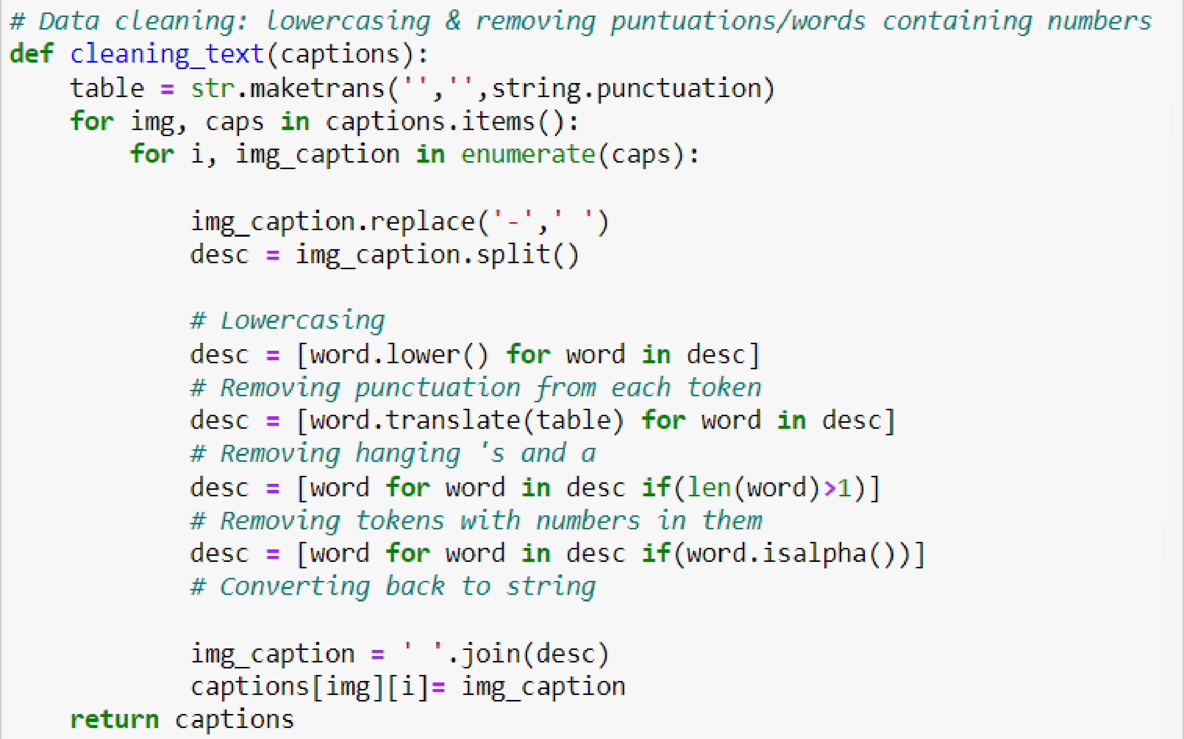
```python
import string
import numpy as np
import os
import cv2
from pickle import dump, load

from keras.applications.xception import Xception, preprocess_input
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.utils import to_categorical
from keras.layers import concatenate

from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout

from tqdm.notebook import tqdm as tqdm
tqdm().pandas()
```

2. Data Loading



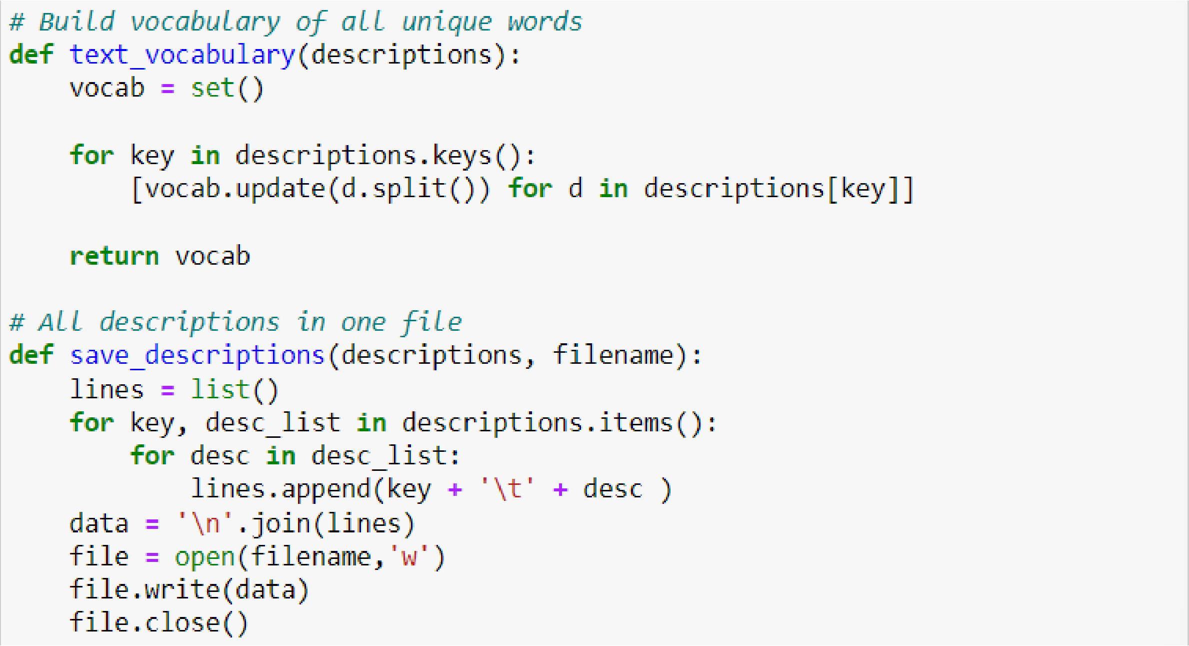
```python
# Load a text file into memory
def load_doc(filename):
    # Open the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

# Obtain captions of all images
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions ={}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions
```

3. Data Cleaning



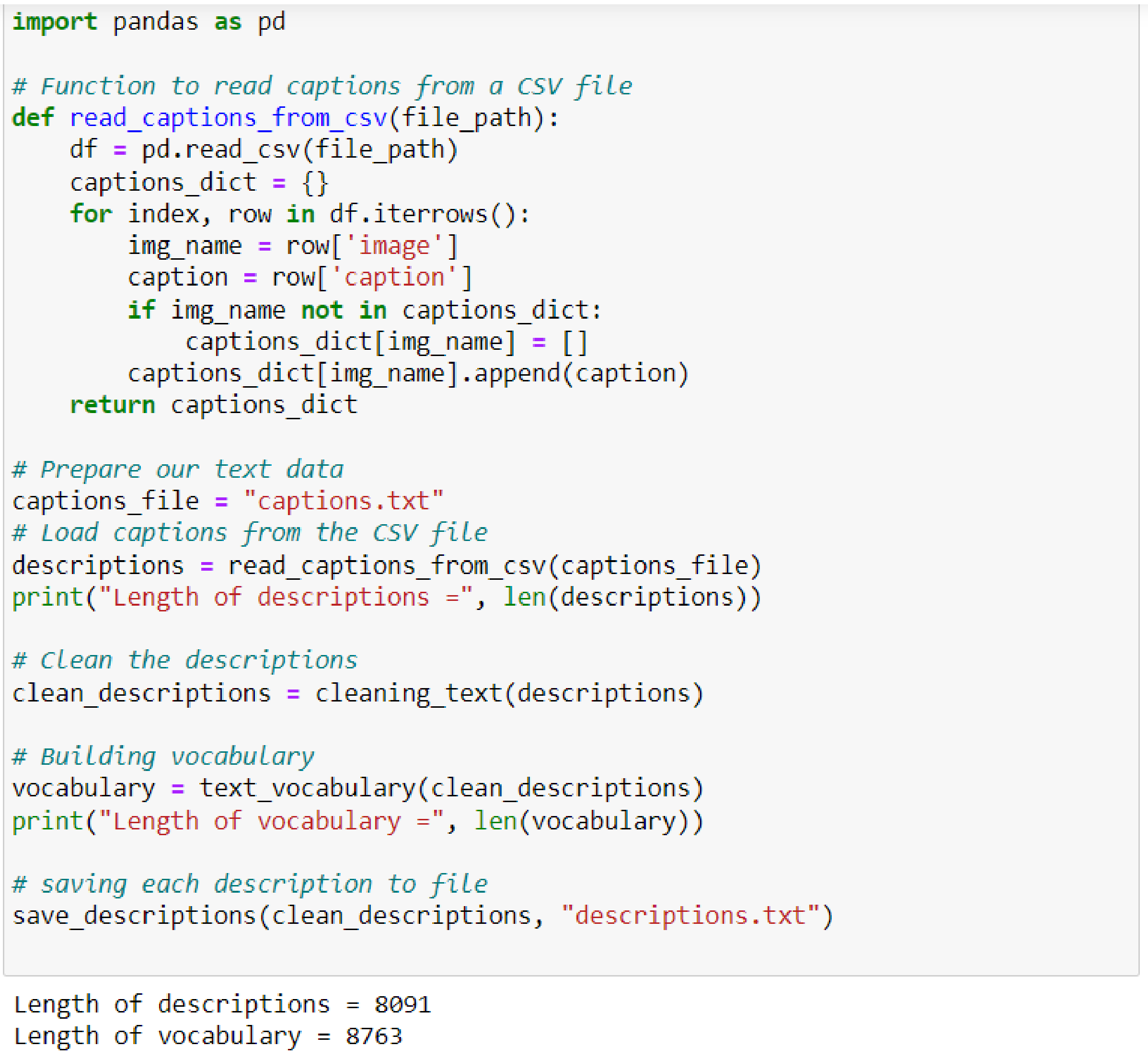
```python
# Data cleaning: lowercasing & removing puntuations/words containing numbers
def cleaning_text(captions):
    table = str.maketrans('','',string.punctuation)
    for img, caps in captions.items():
        for i, img_caption in enumerate(caps):

            img_caption.replace('-',' ')
            desc = img_caption.split()

            # Lowercasing
            desc = [word.lower() for word in desc]
            # Removing punctuation from each token
            desc = [word.translate(table) for word in desc]
            # Removing hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            # Removing tokens with numbers in them
            desc = [word for word in desc if(word.isalpha())]
            # Converting back to string

            img_caption = ' '.join(desc)
            captions[img][i]= img_caption
    return captions
```

4. Build vocabulary and saving file descriptions



```python
# Build vocabulary of all unique words
def text_vocabulary(descriptions):
    vocab = set()

    for key in descriptions.keys():
        [vocab.update(d.split()) for d in descriptions[key]]

    return vocab

# All descriptions in one file
def save_descriptions(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()
```

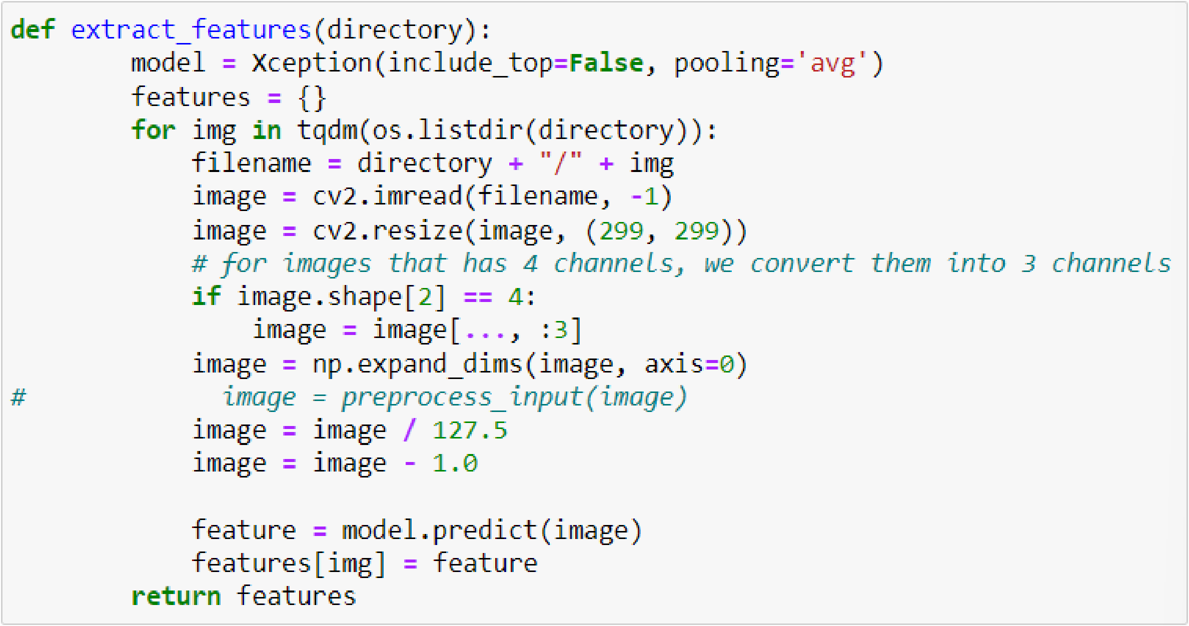
5. Reading captions from CSV



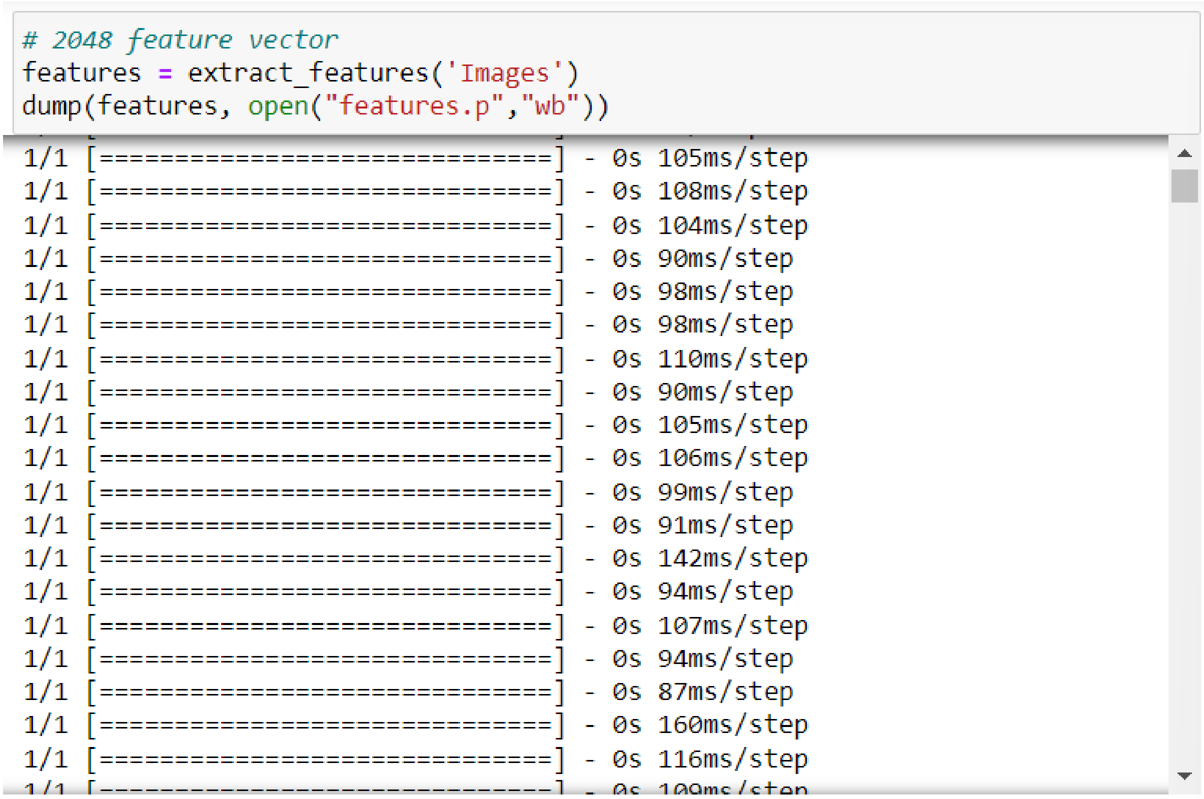
```
import pandas as pd

# Function to read captions from a CSV file
def read_captions_from_csv(file_path):
    df = pd.read_csv(file_path)
    captions_dict = {}
    for index, row in df.iterrows():
        img_name = row['image']
        caption = row['caption']
        if img_name not in captions_dict:
            captions_dict[img_name] = []
        captions_dict[img_name].append(caption)
    return captions_dict

# Prepare our text data
captions_file = "captions.txt"
# Load captions from the CSV file
descriptions = read_captions_from_csv(captions_file)
print("Length of descriptions =", len(descriptions))

# Clean the descriptions
clean_descriptions = cleaning_text(descriptions)

# Building vocabulary
vocabulary = text_vocabulary(clean_descriptions)
print("Length of vocabulary =", len(vocabulary))

# saving each description to file
save_descriptions(clean_descriptions, "descriptions.txt")
```

```
Length of descriptions = 8091
Length of vocabulary = 8763
```

6. Feature extraction



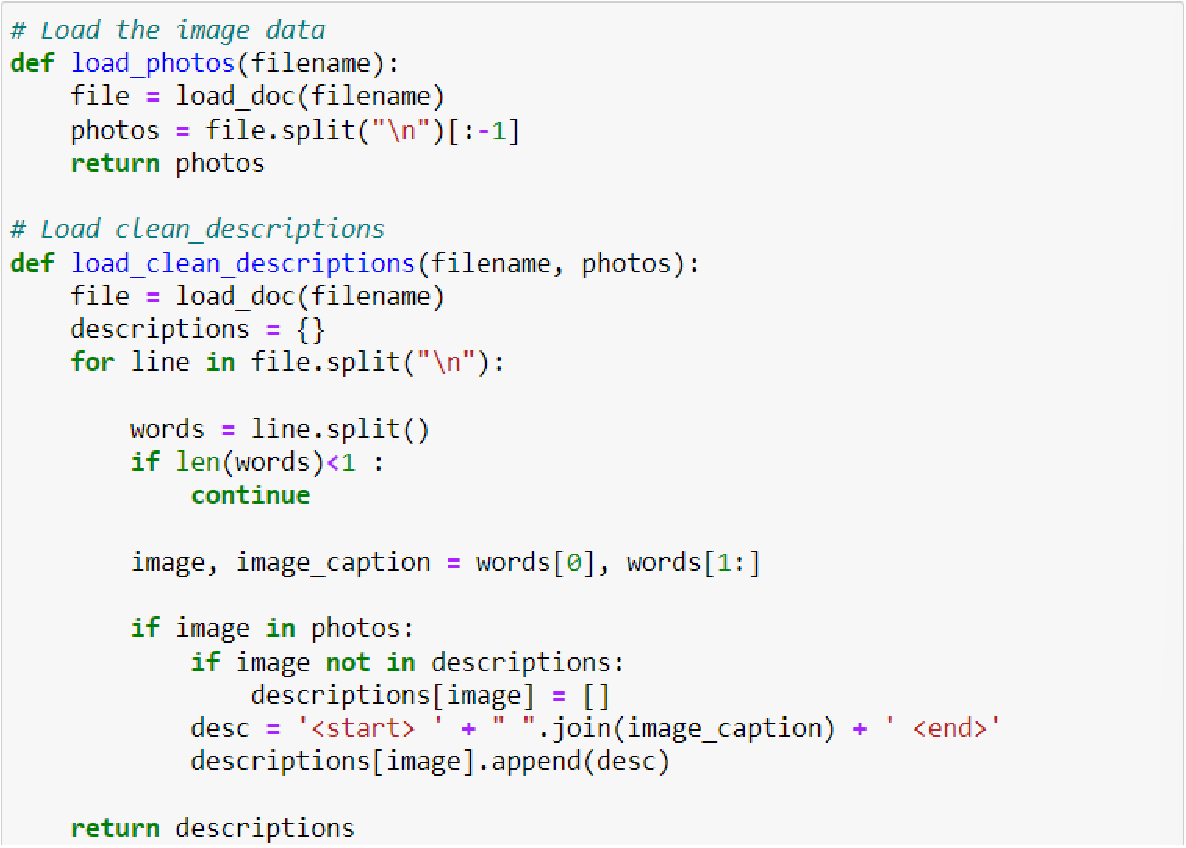
```
def extract_features(directory):
        model = Xception(include_top=False, pooling='avg')
        features = {}
        for img in tqdm(os.listdir(directory)):
            filename = directory + "/" + img
            image = cv2.imread(filename, -1)
            image = cv2.resize(image, (299, 299))
            # for images that has 4 channels, we convert them into 3 channels
            if image.shape[2] == 4:
                image = image[..., :3]
            image = np.expand_dims(image, axis=0)
#             image = preprocess_input(image)
            image = image / 127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```



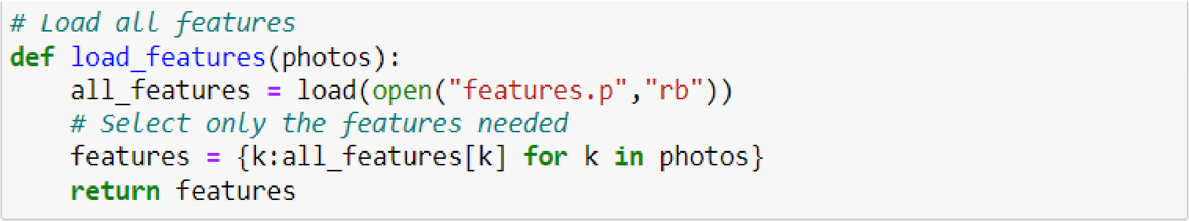
```
# 2048 feature vector
features = extract_features('Images')
dump(features, open("features.p","wb"))
```

```
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 142ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 160ms/step
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 109ms/step
```

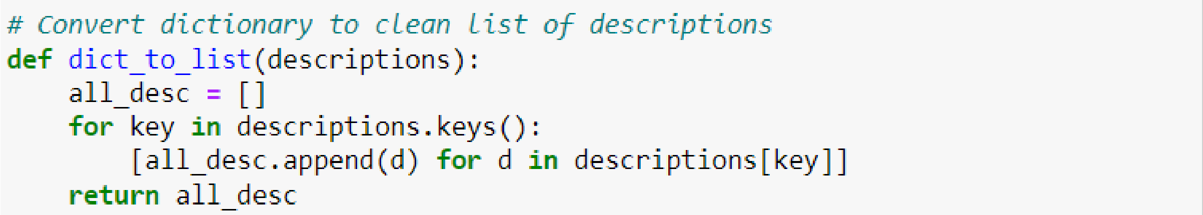
7. Loading data, descriptions, features



```
# Load the image data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos

# Load clean_descriptions
def load_clean_descriptions(filename, photos):
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions
```

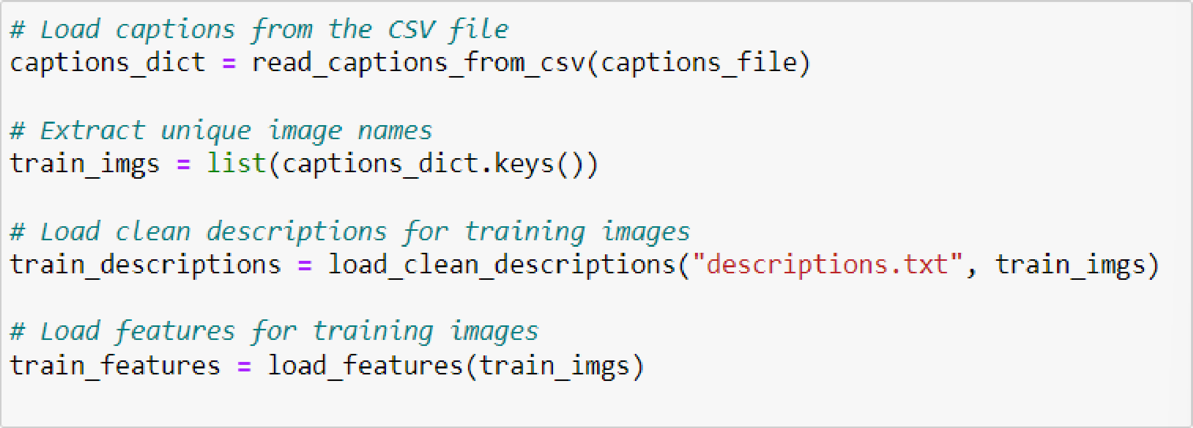


```python
# Load all features
def load_features(photos):
    all_features = load(open("features.p","rb"))
    # Select only the features needed
    features = {k:all_features[k] for k in photos}
    return features
```



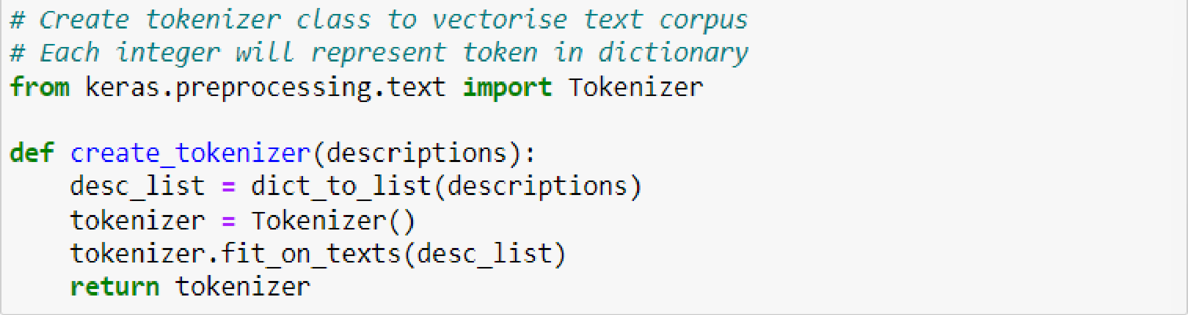
```python
# Convert dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc
```

8. Loading data for training

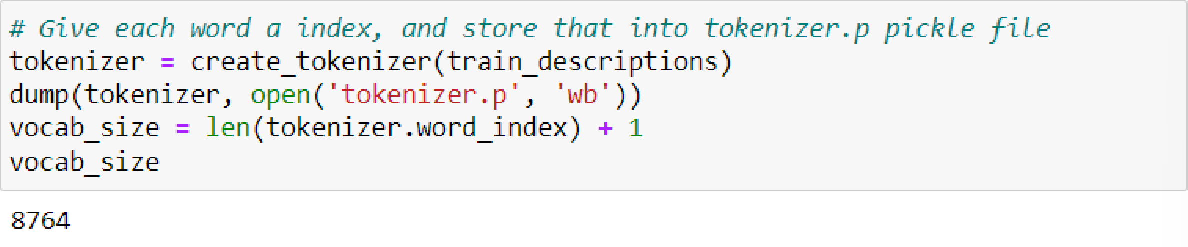


```python
# Load captions from the CSV file
captions_dict = read_captions_from_csv(captions_file)

# Extract unique image names
train_imgs = list(captions_dict.keys())

# Load clean descriptions for training images
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)

# Load features for training images
train_features = load_features(train_imgs)
```

9. Tokenizer

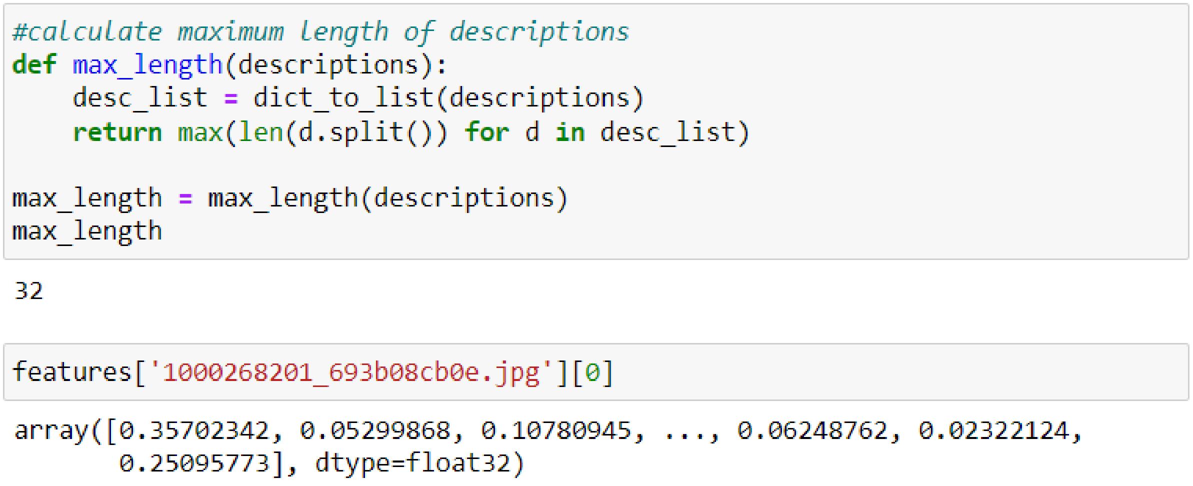


```python
# Create tokenizer class to vectorise text corpus
# Each integer will represent token in dictionary
from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```



```python
# Give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

```
8764
```



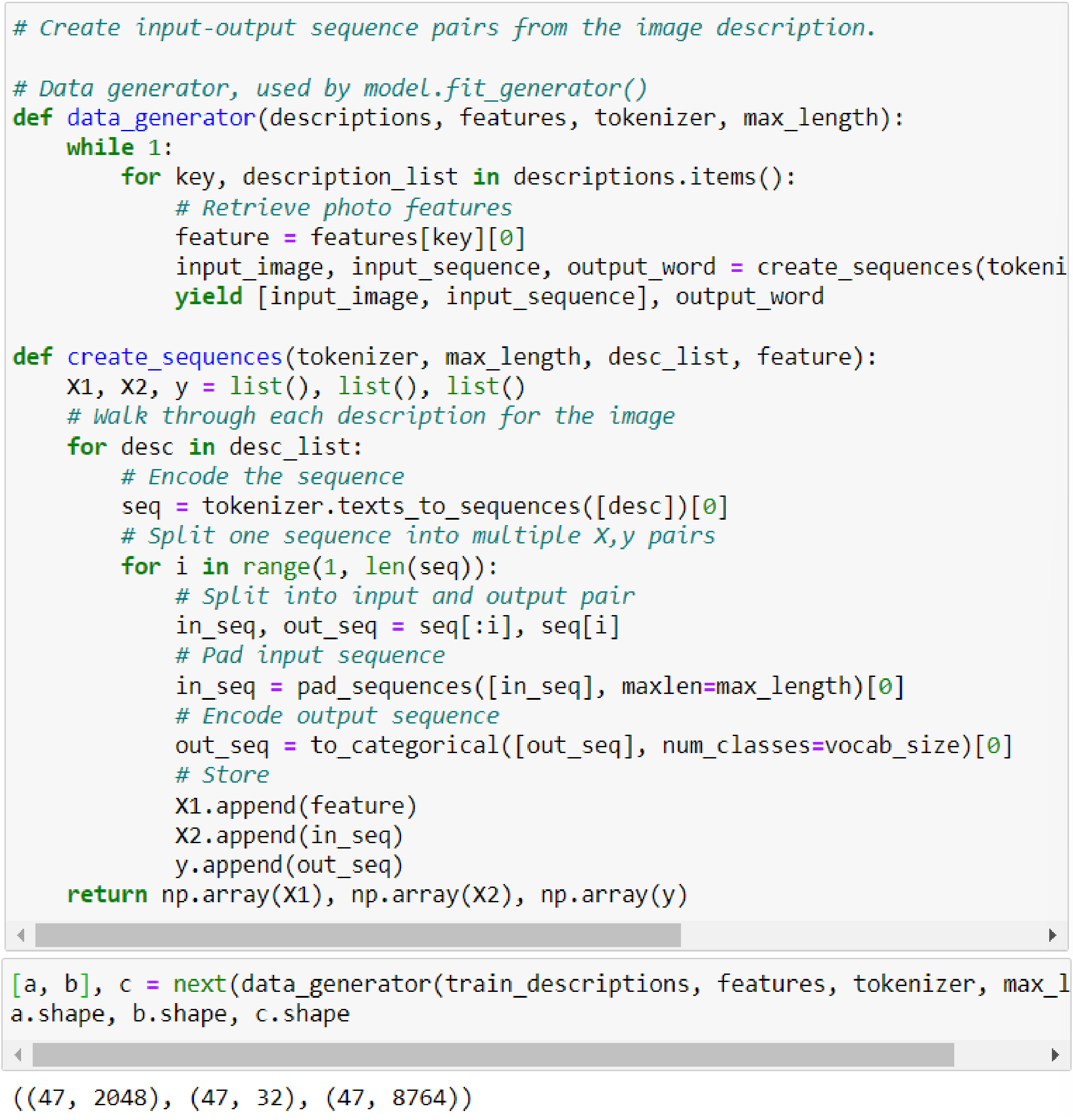
```
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

```
32
```

```
features['1000268201_693b08cb0e.jpg'][0]
```

```
array([0.35702342, 0.05299868, 0.10780945, ..., 0.06248762, 0.02322124,
       0.25095773], dtype=float32)
```

10. Input-output sequence

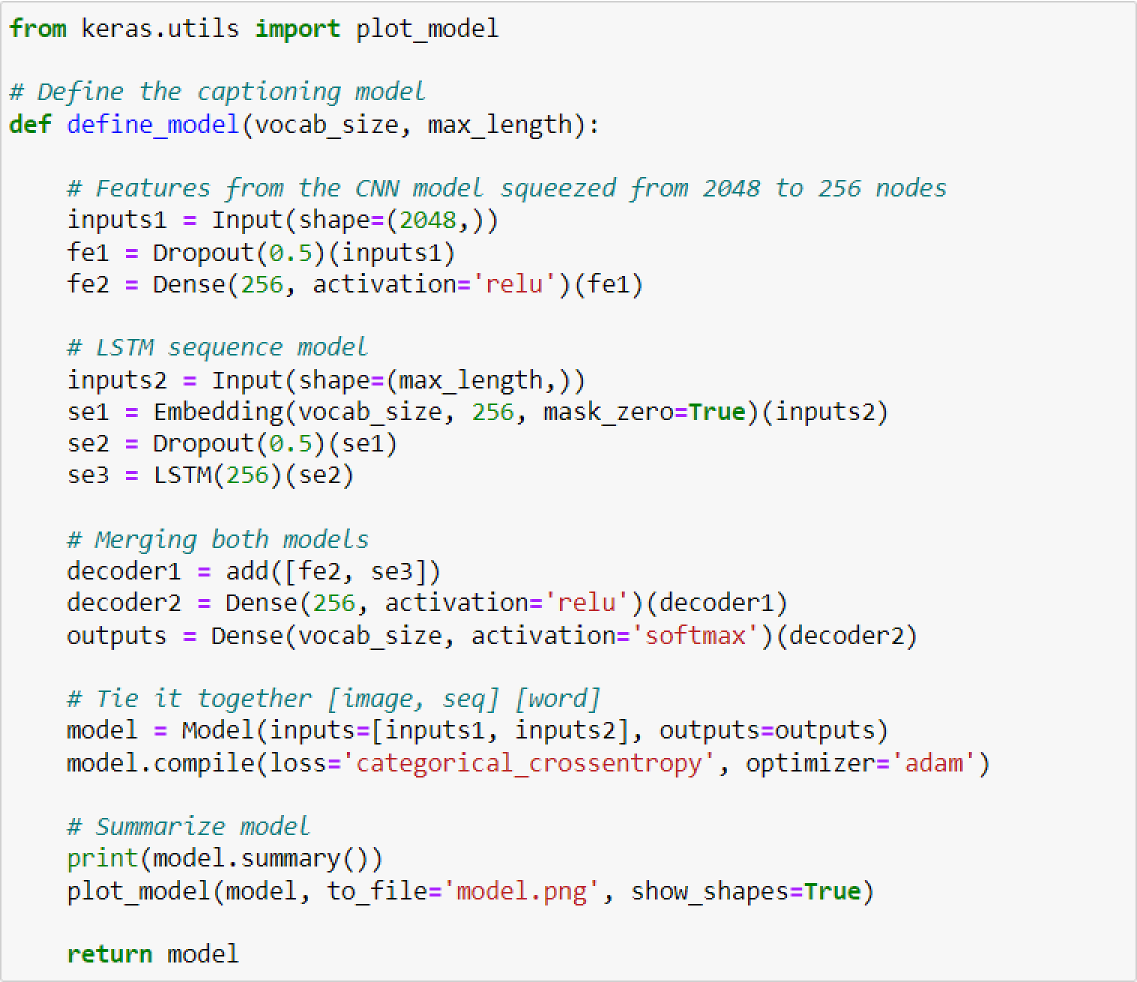


```
# Create input-output sequence pairs from the image description.

# Data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            # Retrieve photo features
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokeni
            yield [input_image, input_sequence], output_word

def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # Walk through each description for the image
    for desc in desc_list:
        # Encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # Split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # Split into input and output pair
            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
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

```
[a, b], c = next(data_generator(train_descriptions, features, tokenizer, max_l
a.shape, b.shape, c.shape
```

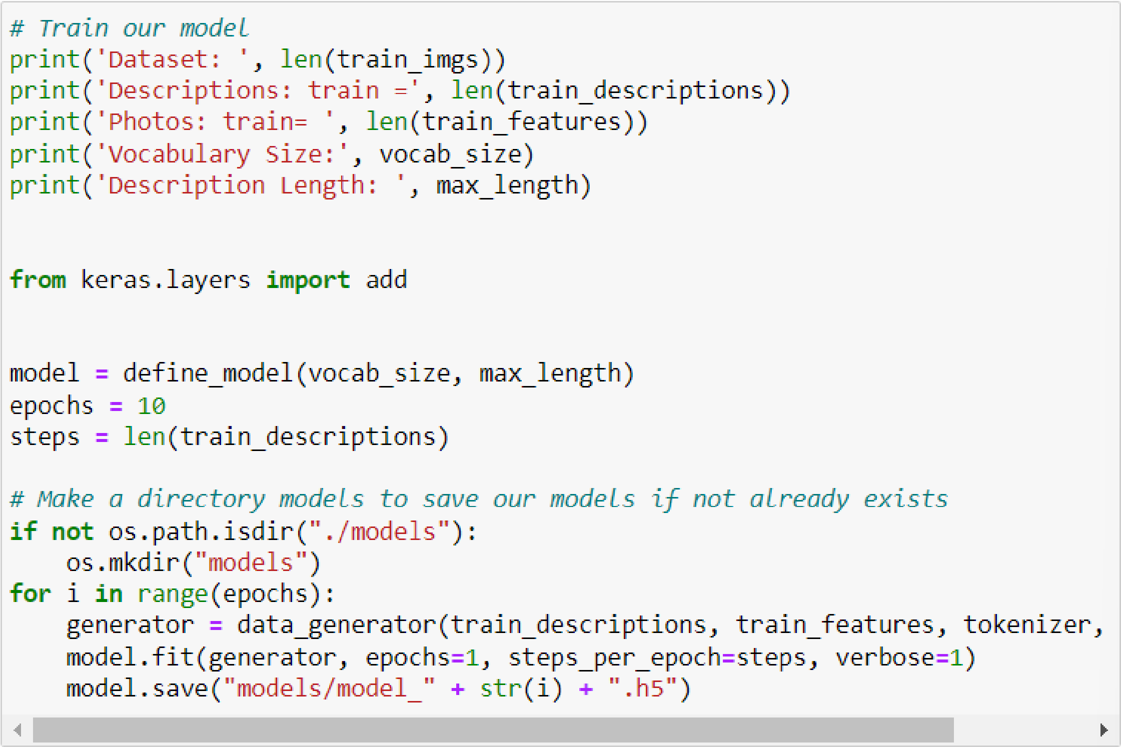
```
((47, 2048), (47, 32), (47, 8764))
```

11. Model Building

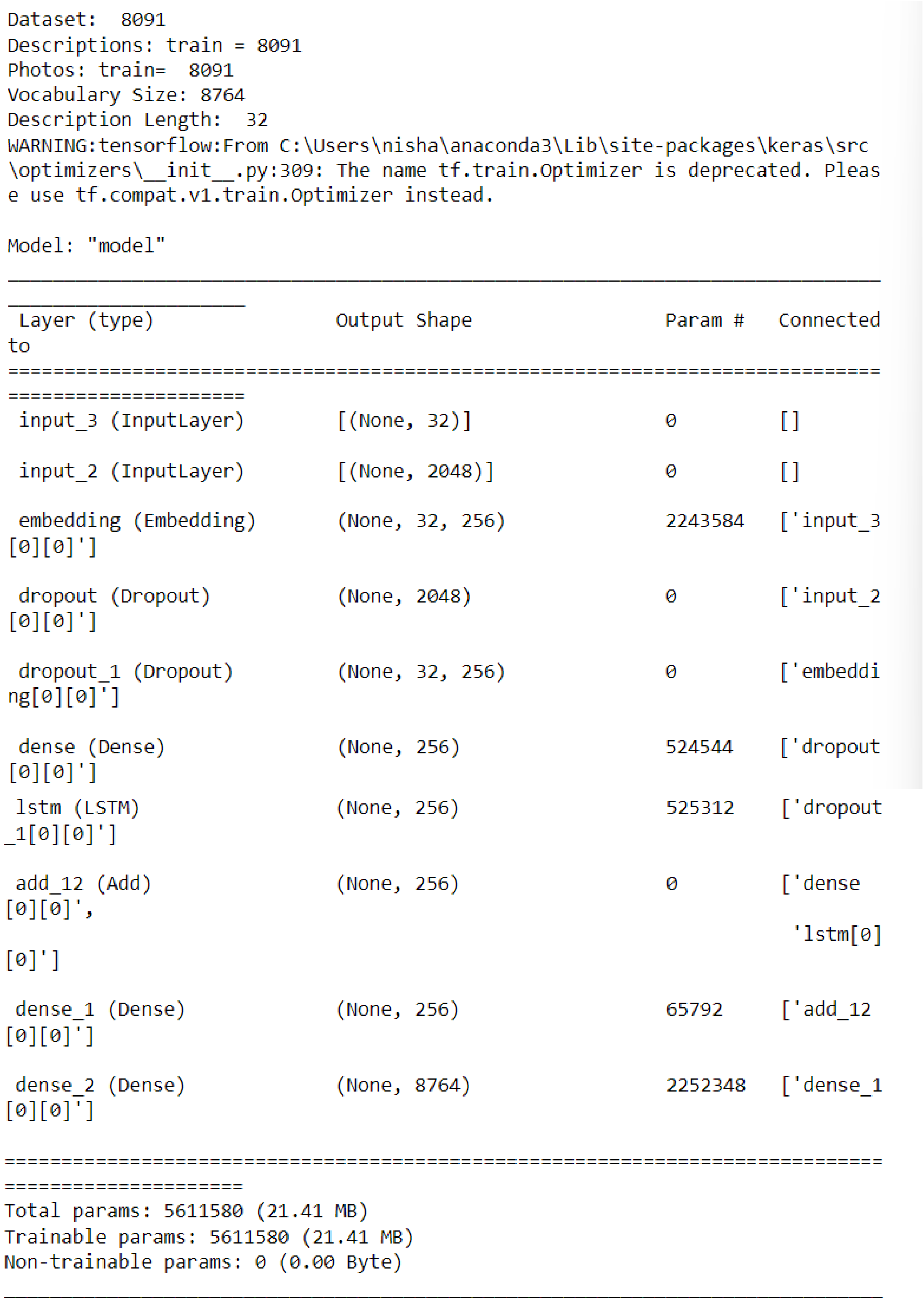


```python
from keras.utils import plot_model

# Define the captioning model
def define_model(vocab_size, max_length):

    # Features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

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

    # Tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # Summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

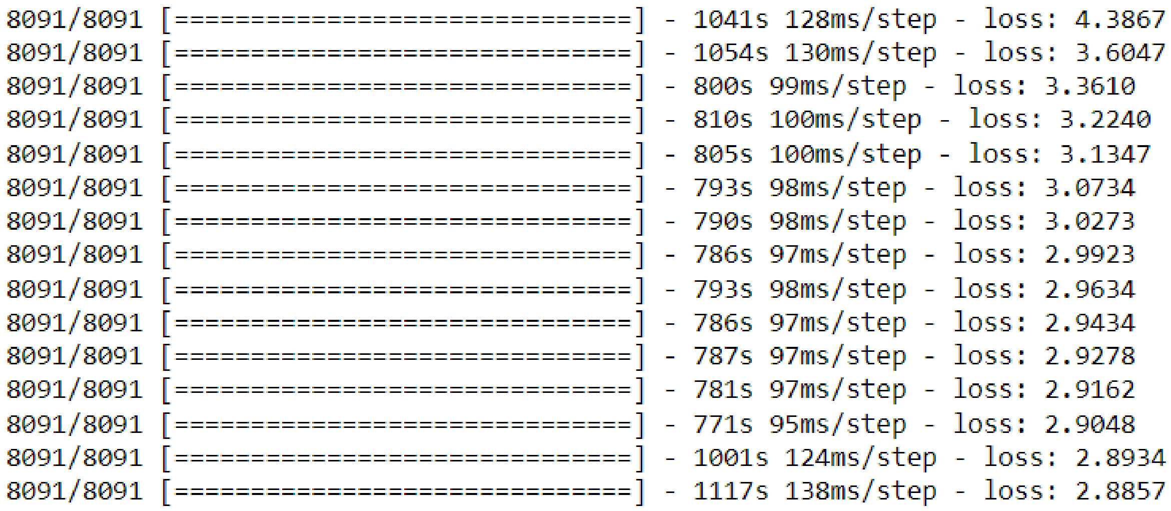
12. Train the model



```python
# Train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train =', len(train_descriptions))
print('Photos: train= ', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)


from keras.layers import add


model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)

# Make a directory models to save our models if not already exists
if not os.path.isdir("./models"):
    os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer,
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

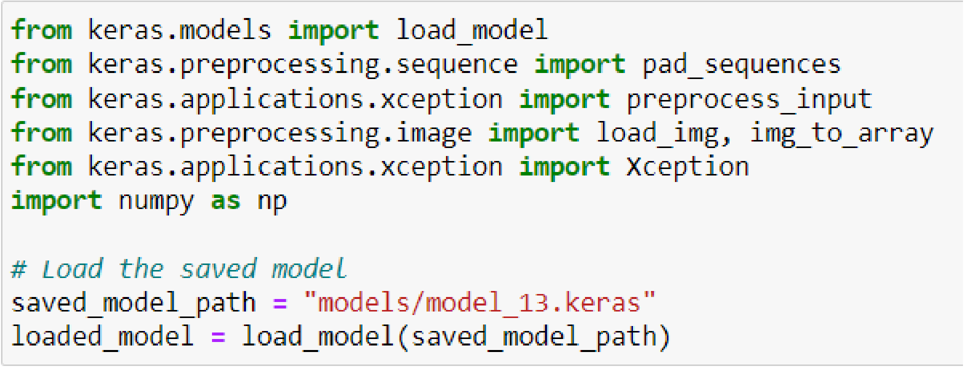


```
Dataset:  8091
Descriptions: train = 8091
Photos: train=  8091
Vocabulary Size: 8764
Description Length:  32
WARNING:tensorflow:From C:\Users\nisha\anaconda3\Lib\site-packages\keras\src
\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Pleas
e use tf.compat.v1.train.Optimizer instead.

Model: "model"
_____________________________________________________________________________
_____________________
 Layer (type)               Output Shape                Param #   Connected
to
=============================================================================
=====================
 input_3 (InputLayer)       [(None, 32)]                0         []

 input_2 (InputLayer)       [(None, 2048)]              0         []

 embedding (Embedding)      (None, 32, 256)             2243584   ['input_3
[0][0]']

 dropout (Dropout)          (None, 2048)                0         ['input_2
[0][0]']

 dropout_1 (Dropout)        (None, 32, 256)             0         ['embeddi
ng[0][0]']

 dense (Dense)              (None, 256)                 524544    ['dropout
[0][0]']
 lstm (LSTM)                (None, 256)                 525312    ['dropout
_1[0][0]']

 add_12 (Add)               (None, 256)                 0         ['dense
[0][0]',

                                                                   'lstm[0]
[0]']

 dense_1 (Dense)            (None, 256)                 65792     ['add_12
[0][0]']

 dense_2 (Dense)            (None, 8764)                2252348   ['dense_1
[0][0]']

=============================================================================
=====================
Total params: 5611580 (21.41 MB)
Trainable params: 5611580 (21.41 MB)
Non-trainable params: 0 (0.00 Byte)
_____________________________________________________________________________
```

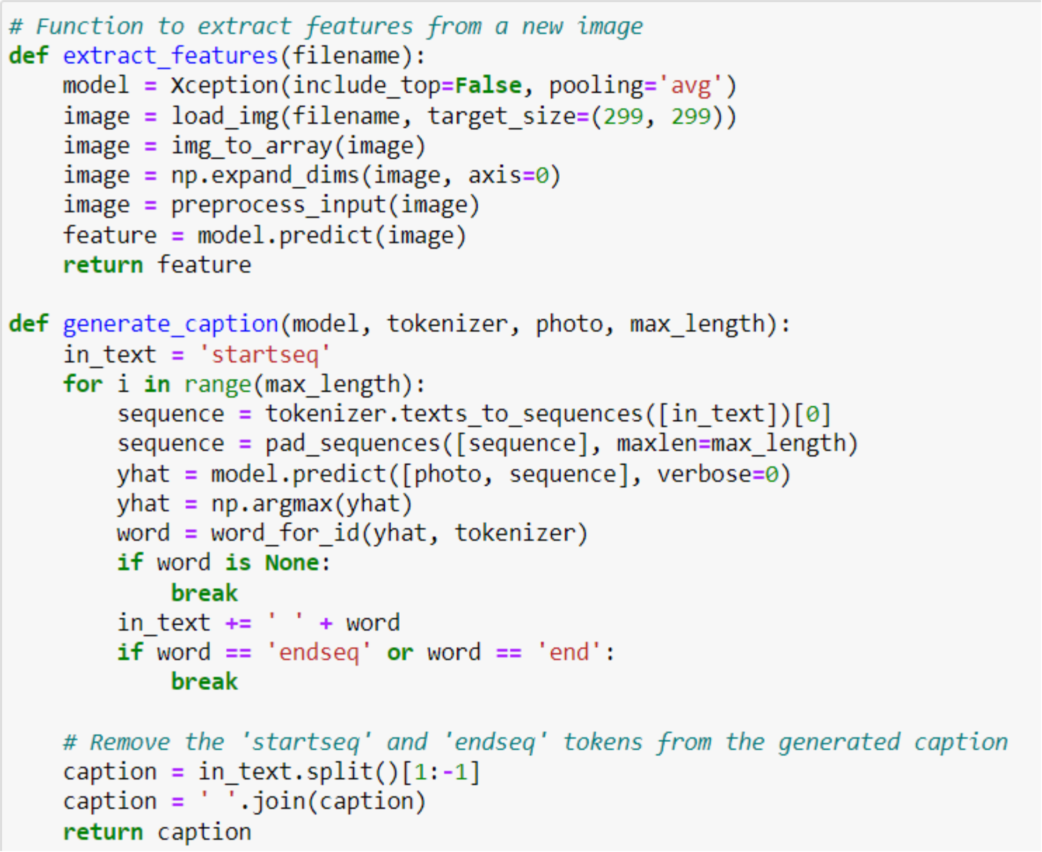


```
8091/8091 [==============================] - 1041s 128ms/step - loss: 4.3867
8091/8091 [==============================] - 1054s 130ms/step - loss: 3.6047
8091/8091 [==============================] - 800s 99ms/step - loss: 3.3610
8091/8091 [==============================] - 810s 100ms/step - loss: 3.2240
8091/8091 [==============================] - 805s 100ms/step - loss: 3.1347
8091/8091 [==============================] - 793s 98ms/step - loss: 3.0734
8091/8091 [==============================] - 790s 98ms/step - loss: 3.0273
8091/8091 [==============================] - 786s 97ms/step - loss: 2.9923
8091/8091 [==============================] - 793s 98ms/step - loss: 2.9634
8091/8091 [==============================] - 786s 97ms/step - loss: 2.9434
8091/8091 [==============================] - 787s 97ms/step - loss: 2.9278
8091/8091 [==============================] - 781s 97ms/step - loss: 2.9162
8091/8091 [==============================] - 771s 95ms/step - loss: 2.9048
8091/8091 [==============================] - 1001s 124ms/step - loss: 2.8934
8091/8091 [==============================] - 1117s 138ms/step - loss: 2.8857
```

Test Results



```python
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.xception import Xception
import numpy as np

# Load the saved model
saved_model_path = "models/model_13.keras"
loaded_model = load_model(saved_model_path)
```



```python
# Function to extract features from a new image
def extract_features(filename):
    model = Xception(include_top=False, pooling='avg')
    image = load_img(filename, target_size=(299, 299))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    feature = model.predict(image)
    return feature

def generate_caption(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq' or word == 'end':
            break

    # Remove the 'startseq' and 'endseq' tokens from the generated caption
    caption = in_text.split()[1:-1]
    caption = ' '.join(caption)
    return caption
```

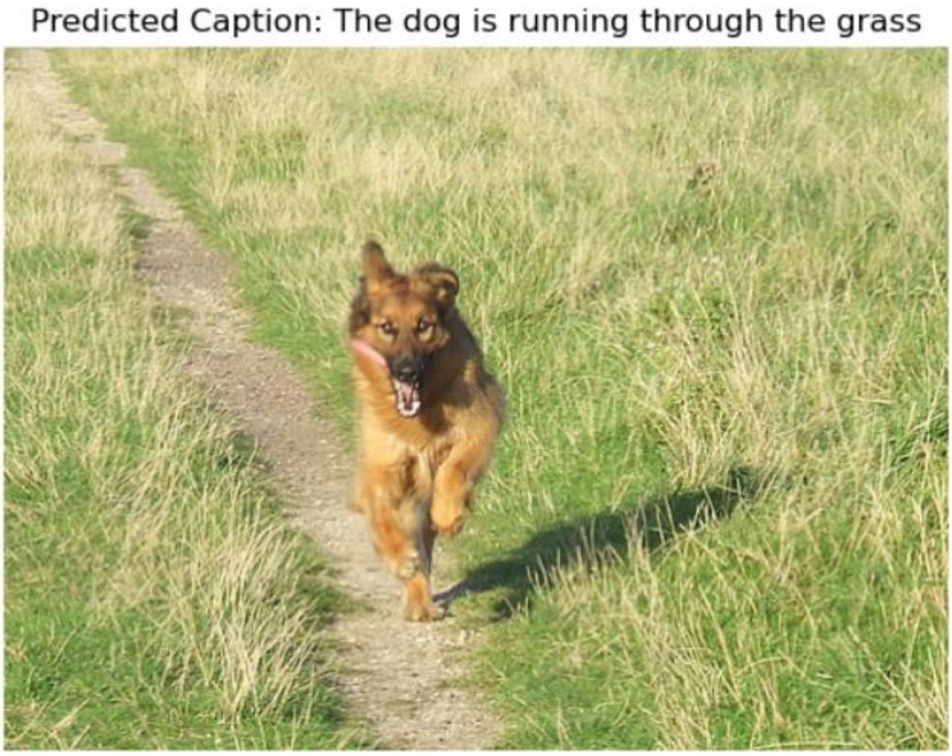
```
# Function to map an integer to its corresponding word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

```
max_length
```

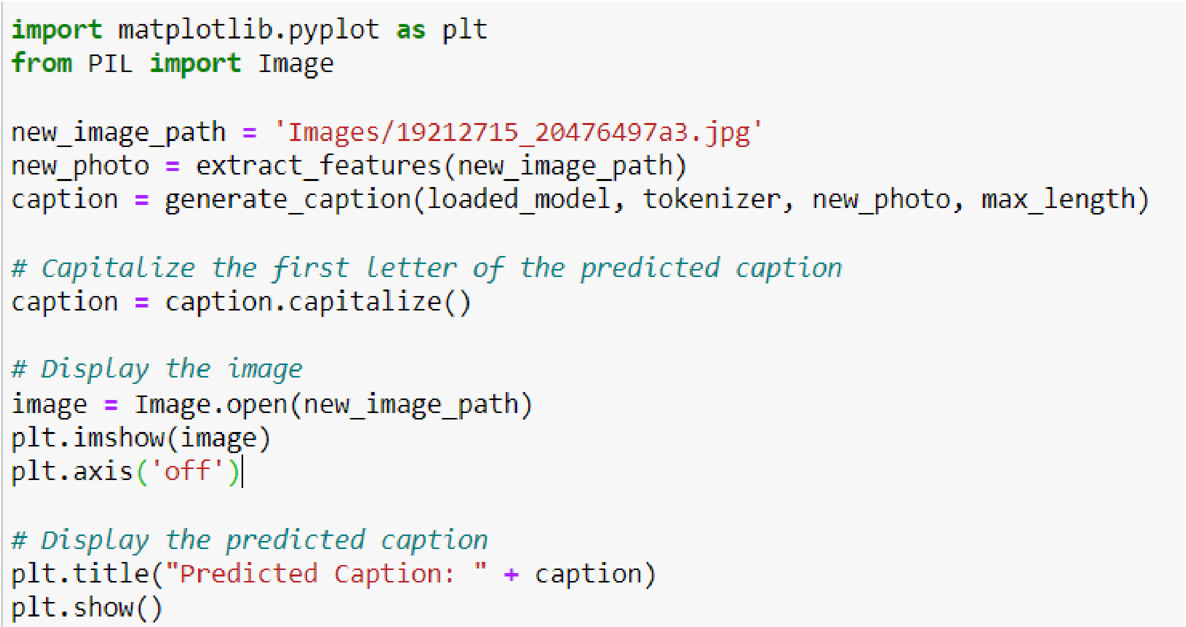
```
32
```

```
import matplotlib.pyplot as plt
from PIL import Image

new_image_path = 'Images/240696675_7d05193aa0.jpg'
new_photo = extract_features(new_image_path)
caption = generate_caption(loaded_model, tokenizer, new_photo, max_length)

# Capitalize the first letter of the predicted caption
caption = caption.capitalize()

# Display the image
image = Image.open(new_image_path)
plt.imshow(image)
plt.axis('off')

# Display the predicted caption
plt.title("Predicted Caption: " + caption)
plt.show()
```

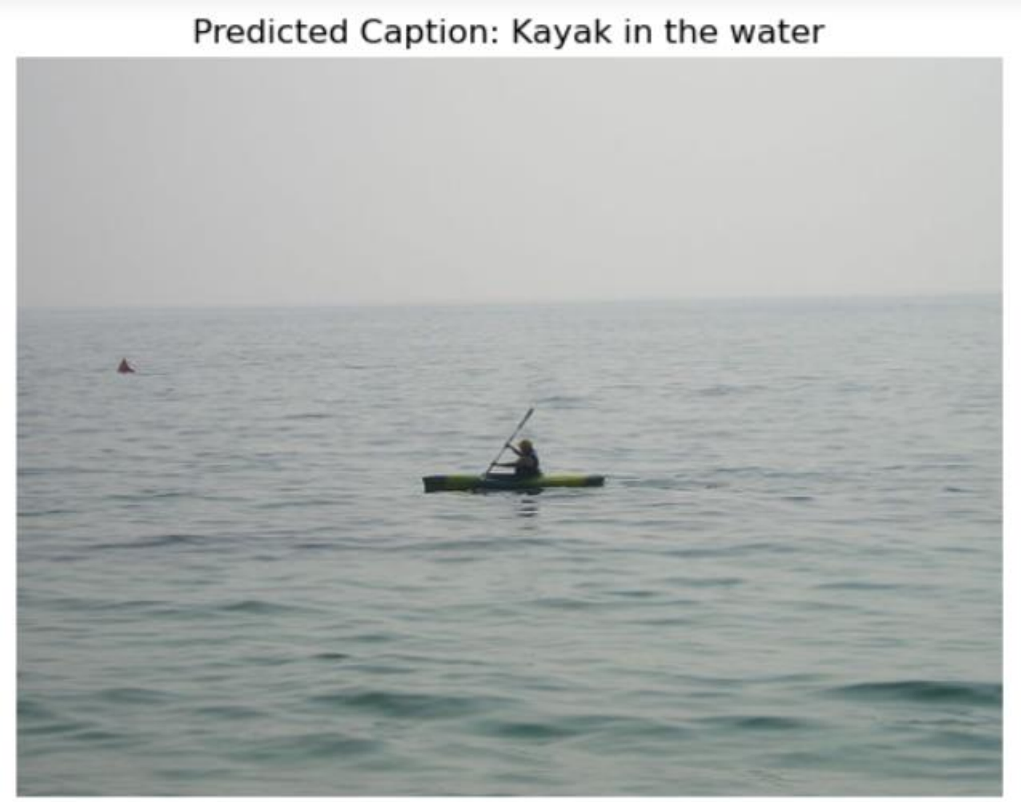
```
1/1 [==============================] - 1s 1s/step
```



Predicted Caption: The dog is running through the grass



```
import matplotlib.pyplot as plt
from PIL import Image

new_image_path = 'Images/19212715_20476497a3.jpg'
new_photo = extract_features(new_image_path)
caption = generate_caption(loaded_model, tokenizer, new_photo, max_length)

# Capitalize the first letter of the predicted caption
caption = caption.capitalize()

# Display the image
image = Image.open(new_image_path)
plt.imshow(image)
plt.axis('off')

# Display the predicted caption
plt.title("Predicted Caption: " + caption)
plt.show()
```



Predicted Caption: Kayak in the water

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.

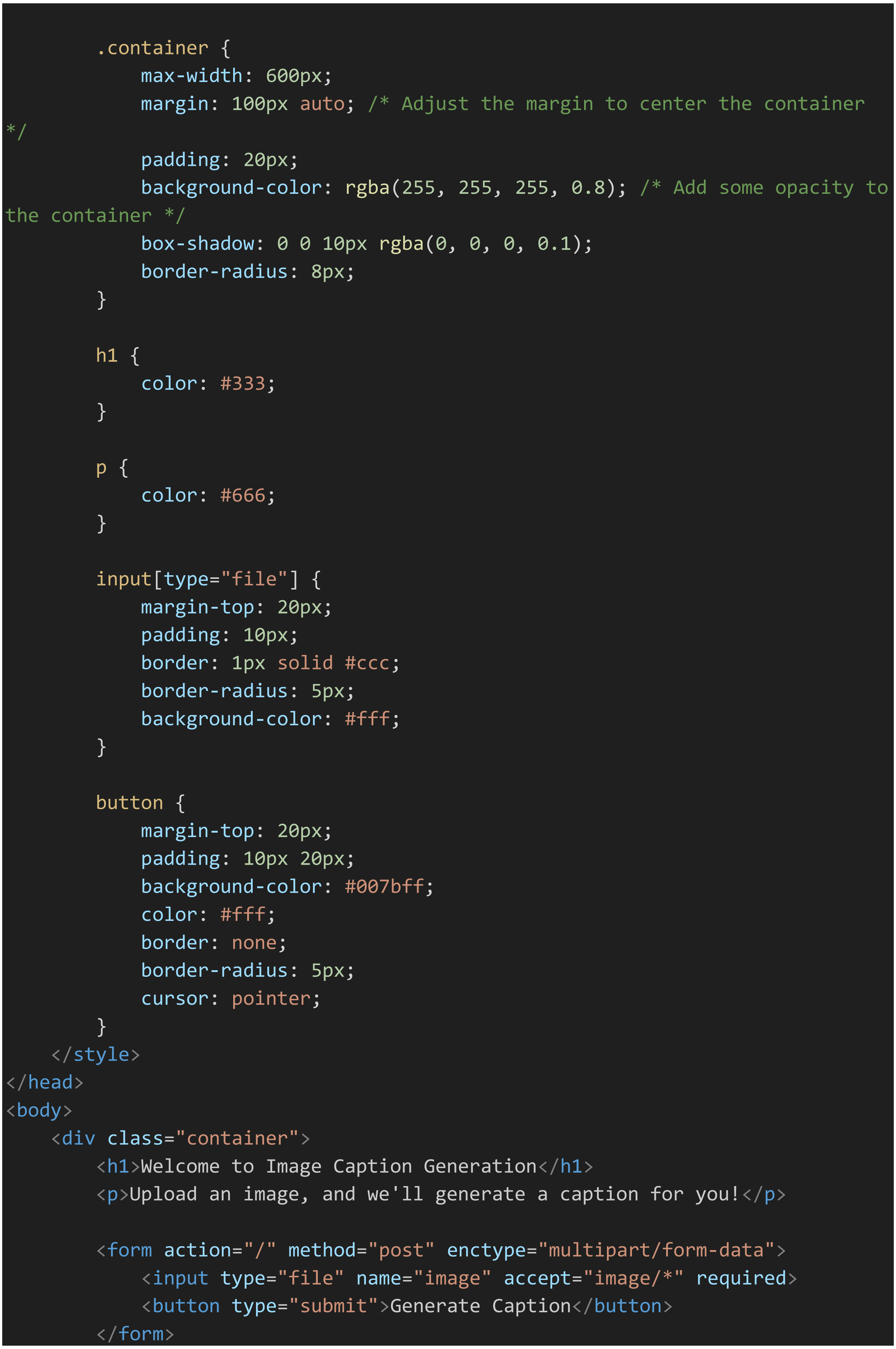
Create HTML Pages

● We use HTML to create the front-end part of the web page.

● Here, we have created 3 HTML pages- home.html, result.html

Home.html

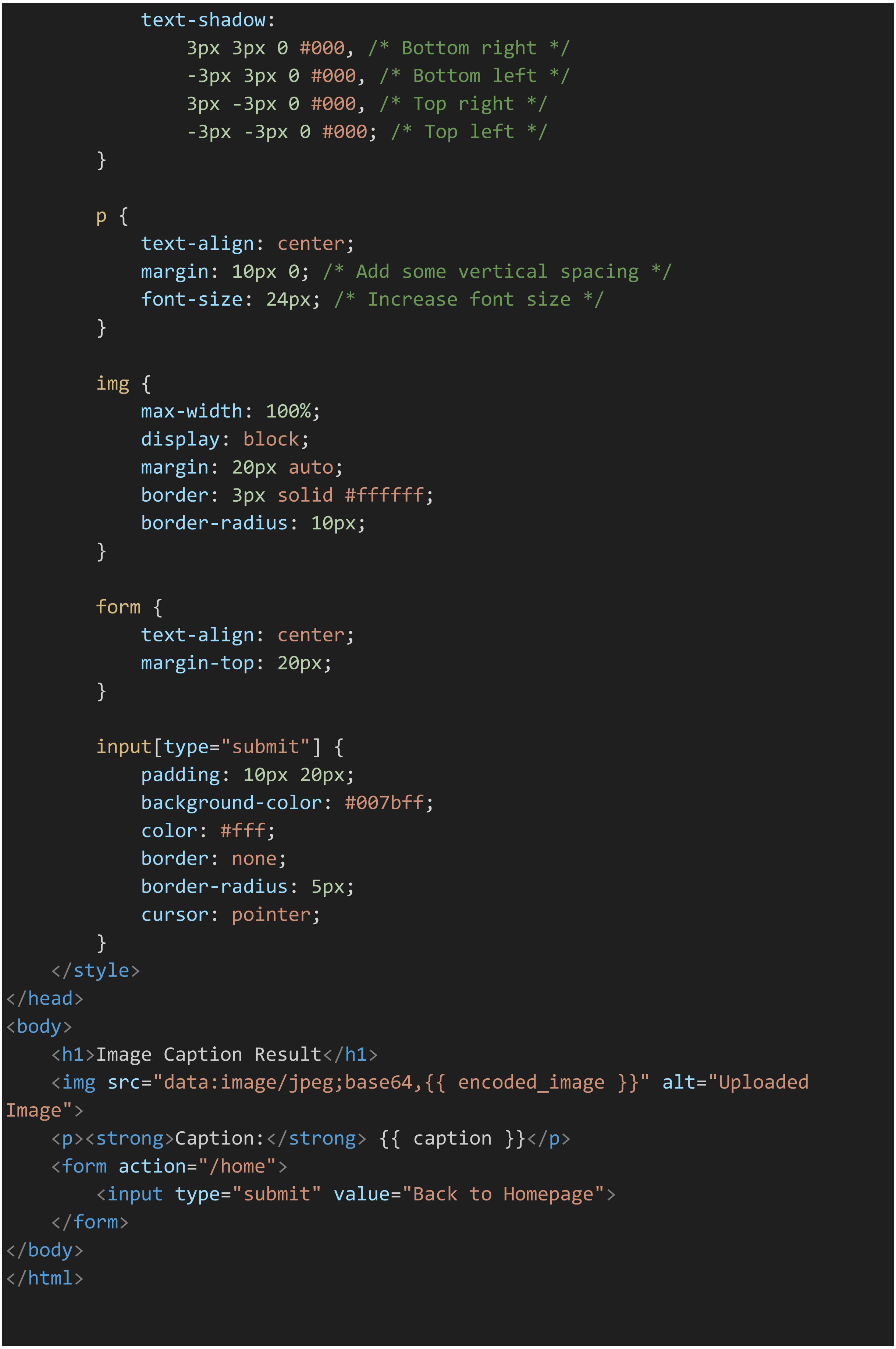
```
<!DOCTYPE html>

<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Caption Generation</title>

    <!-- Add your custom CSS styles here -->
    <style>
        html, body {
            margin: 0;
            padding: 0;
        }


        body {
            background-image:
url("https://wallpapercave.com/wp/wp3377178.jpg");
            background-size: cover;
            background-position: center;
            min-height: 100px;
            background-size: cover;
            background-repeat: no-repeat;
            font-family: 'Arial', sans-serif;
            text-align: center;
            display: flex;
            align-items: center;
            justify-content: center;
            height: 100vh;
            margin: 0;
            margin: 0; /* Add this to remove default margin */
            padding: 0; /* Add this to remove default padding */
        }
```



```
        .container {
            max-width: 600px;
            margin: 100px auto; /* Adjust the margin to center the container
*/
            padding: 20px;
            background-color: rgba(255, 255, 255, 0.8); /* Add some opacity to
the container */
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            border-radius: 8px;
        }

        h1 {
            color: #333;
        }

        p {
            color: #666;
        }

        input[type="file"] {
            margin-top: 20px;
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            background-color: #fff;
        }

        button {
            margin-top: 20px;
            padding: 10px 20px;
            background-color: #007bff;
            color: #fff;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to Image Caption Generation</h1>
        <p>Upload an image, and we'll generate a caption for you!</p>

        <form action="/" method="post" enctype="multipart/form-data">
            <input type="file" name="image" accept="image/*" required>
            <button type="submit">Generate Caption</button>
        </form>
    </div>
</body>
</html>
```

Result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Caption Result</title>

    <!-- Add your custom CSS styles here -->
    <style>
        body {
            background-image:
url("https://wallpapercave.com/wp/wp3377178.jpg");
            background-size: cover;
            background-position: center center;
            background-repeat: no-repeat;
            font-family: 'Helvetica', sans-serif; /* Change the font */
            text-align: center;
            margin: 0;
            padding: 0;
            display: flex;
            flex-direction: column; /* Display children in a column */
            align-items: center; /* Center items horizontally */
            justify-content: center; /* Center items vertically */
            height: 100vh;
            color: #ffffff; /* Change text color to white */
        }

        h1 {
            text-align: center;
            margin: 10px 0; /* Add some vertical spacing */
            font-size: 36px; /* Increase font size */
            font-family: 'Times New Roman', Times, serif; /* Change the font
*/
            background-color: #000000; /* Black background */
            color: #ffffff; /* White text color */
            display: inline-block; /* Display as inline block to apply shadow
to each letter */
            padding: 10px; /* Add padding for spacing */
            border-radius: 10px; /* Add rounded corners */
```



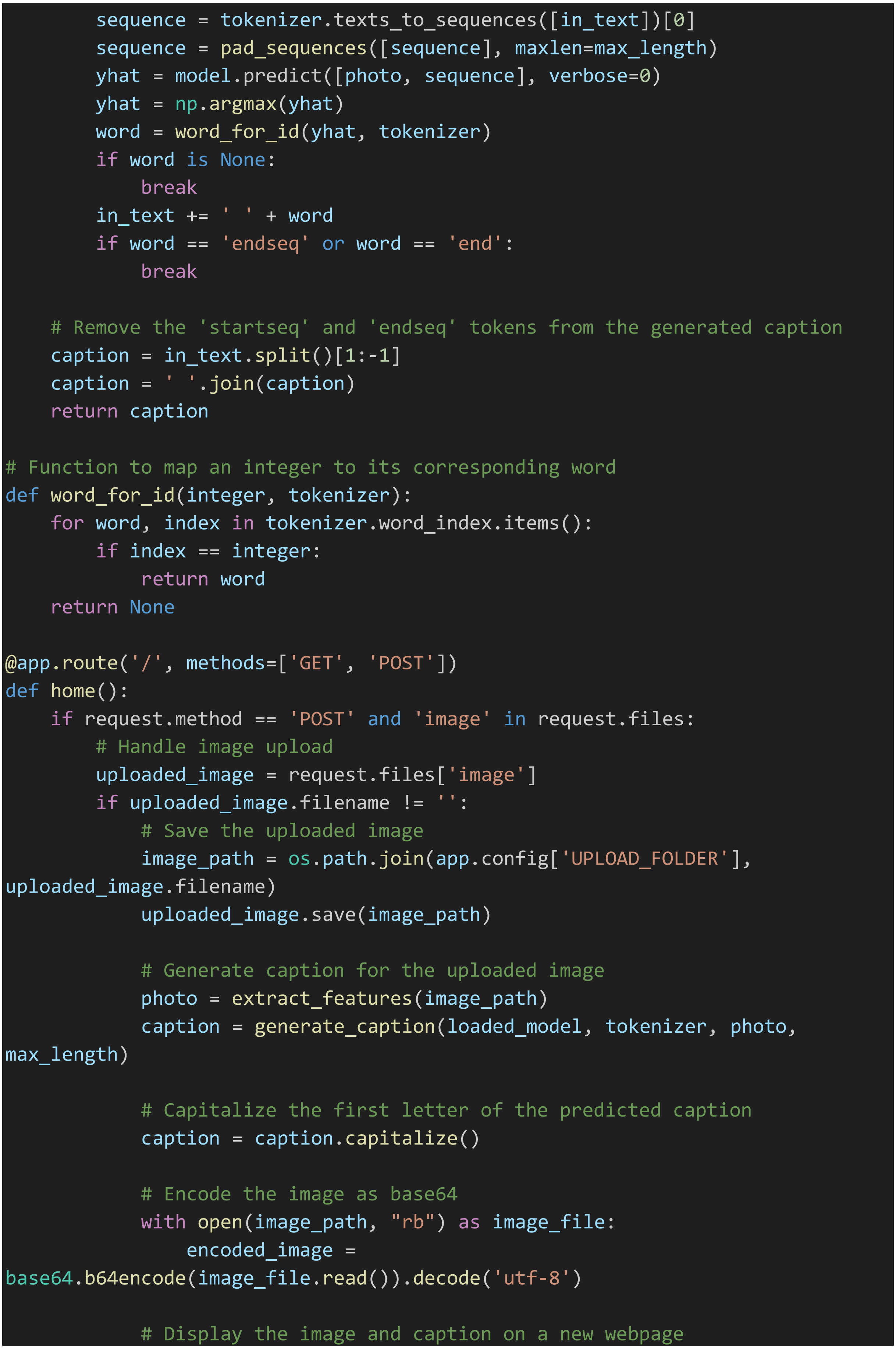
```
            text-shadow:
                3px 3px 0 #000, /* Bottom right */
                -3px 3px 0 #000, /* Bottom left */
                3px -3px 0 #000, /* Top right */
                -3px -3px 0 #000; /* Top left */
        }

        p {
            text-align: center;
            margin: 10px 0; /* Add some vertical spacing */
            font-size: 24px; /* Increase font size */
        }

        img {
            max-width: 100%;
            display: block;
            margin: 20px auto;
            border: 3px solid #ffffff;
            border-radius: 10px;
        }

        form {
            text-align: center;
            margin-top: 20px;
        }

        input[type="submit"] {
            padding: 10px 20px;
            background-color: #007bff;
            color: #fff;
            border: none;
            border-radius: 5px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <h1>Image Caption Result</h1>
    <img src="data:image/jpeg;base64,{{ encoded_image }}" alt="Uploaded
Image">
    <p><strong>Caption:</strong> {{ caption }}</p>
    <form action="/home">
        <input type="submit" value="Back to Homepage">
    </form>
</body>
</html>
```

Create app.py (Python Flask) file: -

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

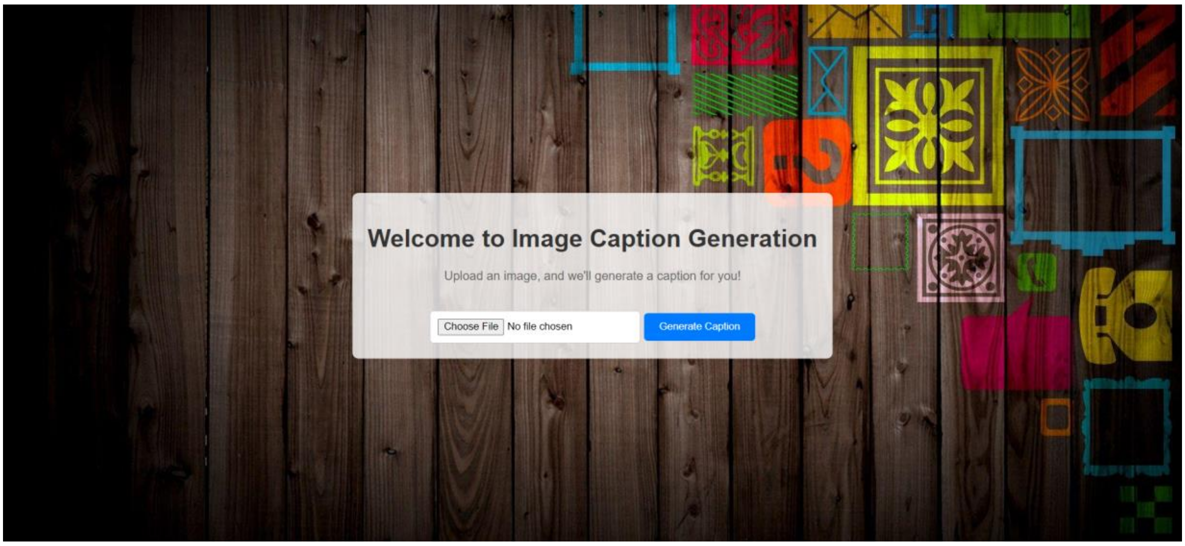
App.py

```
from flask import Flask, render_template, request, redirect, url_for
import os
import matplotlib.pyplot as plt
from PIL import Image
import pickle
import base64
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.xception import Xception
import numpy as np

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'

# Load the saved model
saved_model_path = "models/model_9.h5"
loaded_model = load_model(saved_model_path)

# Load the features and tokenizer
with open("features.p", "rb") as f:
    features = pickle.load(f)

with open("tokenizer.p", "rb") as f:
    tokenizer = pickle.load(f)

# Set max_length
max_length = 32

# Function to extract features from a new image
def extract_features(filename):
    model = Xception(include_top=False, pooling='avg')
    image = load_img(filename, target_size=(299, 299))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    feature = model.predict(image)
    return feature

def generate_caption(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for i in range(max_length):
```



```
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq' or word == 'end':
            break

    # Remove the 'startseq' and 'endseq' tokens from the generated caption
    caption = in_text.split()[1:-1]
    caption = ' '.join(caption)
    return caption

# Function to map an integer to its corresponding word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST' and 'image' in request.files:
        # Handle image upload
        uploaded_image = request.files['image']
        if uploaded_image.filename != '':
            # Save the uploaded image
            image_path = os.path.join(app.config['UPLOAD_FOLDER'],
uploaded_image.filename)
            uploaded_image.save(image_path)

            # Generate caption for the uploaded image
            photo = extract_features(image_path)
            caption = generate_caption(loaded_model, tokenizer, photo,
max_length)

            # Capitalize the first letter of the predicted caption
            caption = caption.capitalize()

            # Encode the image as base64
            with open(image_path, "rb") as image_file:
                encoded_image =
base64.b64encode(image_file.read()).decode('utf-8')

            # Display the image and caption on a new webpage
            return render_template('result.html', encoded_image=encoded_image, caption=caption)

    # Render the homepage with an image upload form
    return render_template('home.html')


@app.route('/home')
def return_home():
    return redirect(url_for('home'))

if __name__ == '__main__':
    if not os.path.exists(app.config['UPLOAD_FOLDER']):
        os.makedirs(app.config['UPLOAD_FOLDER'])
    app.run(debug=True)
```

Webapp Output Screen shots:



Welcome to Image Caption Generation

Upload an image, and we'll generate a caption for you!

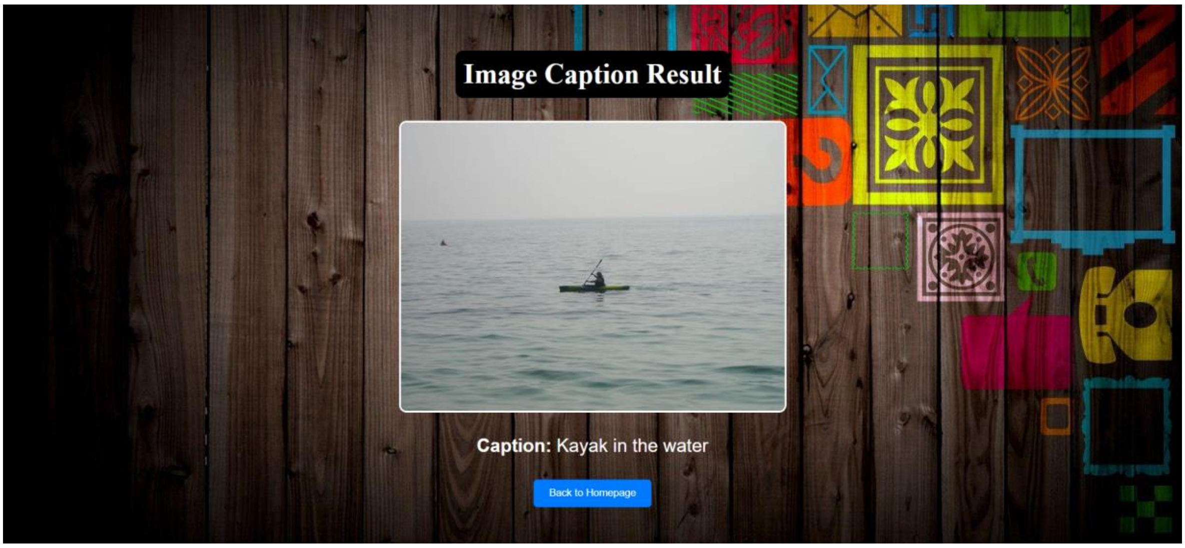


Image Caption Result

Caption: Kayak in the water