IMAGE CAPTION GENERATION USING DEEP LEARNING TECHNIQUES

INTRODUCTION

Image caption generation using a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) with Long Short-Term Memory (LSTM) cells is a fascinating area of AI where computers learn to understand images and generate descriptive captions for them. CNNs are adept at extracting meaningful features from images, capturing essential visual information, while RNNs with LSTM cells serve as language models, learning the sequence and structure of sentences to generate coherent and contextually relevant captions. By integrating these two types of neural networks, this approach enables machines to not only recognize elements within images but also articulate them in human-like descriptions, bridging the gap between visual perception and natural language understanding.

PURPOSE

Image caption generation serves as a powerful bridge between visual content and human-like understanding, enhancing accessibility and comprehension of images for various applications. By automatically generating descriptive captions for images, this technology aids the visually impaired in accessing visual content by providing textual descriptions. It also facilitates better content organization and searchability, improving information retrieval across vast image databases or social media platforms. Moreover, in fields like autonomous vehicles or robotics, accurate image captions aid in decision-making processes by providing contextual information about the surroundings. Additionally, image captioning finds utility in content recommendation systems, enhancing user experience by offering richer contextual information. Overall, the purpose lies in enriching accessibility, comprehension, and application of visual content across diverse domains and user scenarios.

EXISTING PROBLEM

One big problem with making computers write descriptions for pictures is that they sometimes miss important details or make mistakes. They might struggle to understand the whole story in a picture or might get some things wrong in their descriptions. Also, they might say things that aren't quite right because they learned from some data that had some mistakes or biases in it. Getting computers to write accurate and complete descriptions for pictures, without mistakes or biases, is a tough challenge in this field.

REFERENCES

1. "Show and Tell: A Neural Image Caption Generator" by Oriol Vinyals et al. (2015) - Introduces a model using CNN and LSTM for image captioning.
2. "Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering" by Peter Anderson et al. (2018) - Discusses attention mechanisms for improved image captioning.
3. "Image Captioning with Semantic Attention" by Li Yao et al. (2017) - Explores semantic attention mechanisms in caption generation.
4. "Self-Critical Sequence Training for Image Captioning" by Steven J. Rennie et al. (2017) - Introduces a reinforcement learning approach for improving image captioning models.

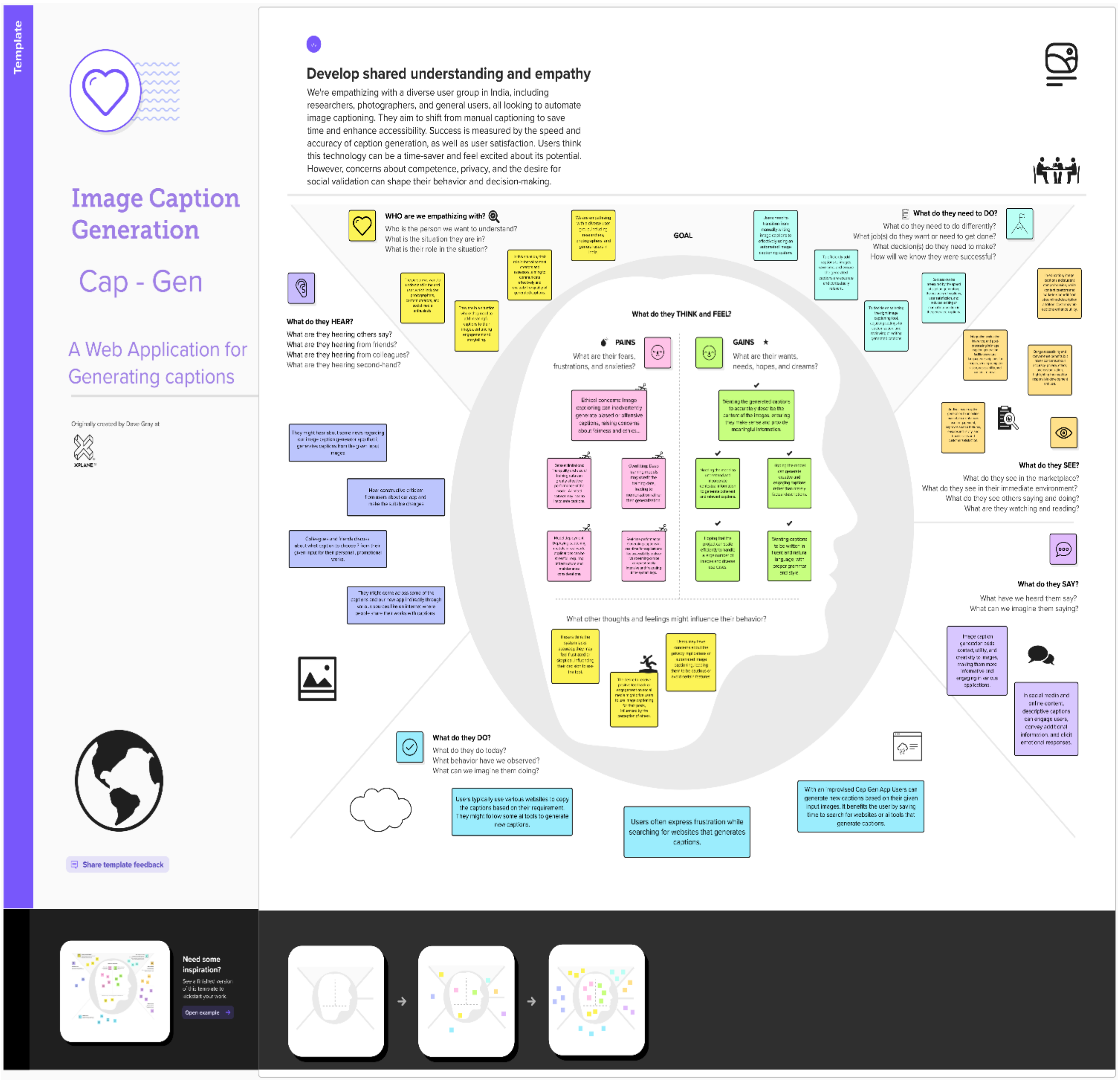
PROBLEM STATEMENT DEFINITION

The problem in image caption generation is to create AI models capable of accurately describing images in natural language. These models must understand diverse visual content, discerning objects, relationships, and context within images to generate meaningful and coherent captions. The challenge involves developing algorithms that capture intricate details, avoid biases, and produce contextually relevant descriptions, addressing complexities like diverse interpretations of images and handling ambiguous scenes. Ensuring the generated captions are both accurate and human-like remains a fundamental goal in this field.

The big puzzle in image captioning is making computers understand pictures and describe them just like humans do. The tricky part is teaching them to see all the things in a picture and then use words to tell what's happening. Sometimes, computers miss important details or mix things up in their descriptions. They might not get the whole story in a picture or might say something that's not quite right. The challenge is to help computers get better at writing accurate and complete descriptions for pictures, just like people do, without making mistakes or getting confused by tricky images. The aim is for them to describe pictures in a way that's clear, accurate, and just like how a person would tell the story.

IDEATION AND PROPOSED SOLUTION

Empathy Map and Canvas



Template

Image Caption Generation

Cap - Gen

A Web Application for Generating captions

Originally created by Dave Gray at XPLANE

Share template feedback

Develop shared understanding and empathy

We're empathizing with a diverse user group in India, including researchers, photographers, and general users, all looking to automate image captioning. They aim to shift from manual captioning to save time and enhance accessibility. Success is measured by the speed and accuracy of caption generation, as well as user satisfaction. Users think this technology can be a time-saver and feel excited about its potential. However, concerns about competence, privacy, and the desire for social validation can shape their behavior and decision-making.

WHO are we empathizing with?
What is the person we want to understand?
What is the situation they are in?
What is their role in the situation?

We are empathizing with a diverse user group, including researchers, photographers, and general users in India.

In this situation, their role is that of content creators and individuals aiming to communicate effectively and evaluate the quality of generated captions.

The person we want to understand is the end user, which includes photographers, content creators, and social media individuals.

They are in a situation where they need to add meaningful captions to their images, enhancing engagement and storytelling.

GOAL

Users need to transition from manually writing image captions to effectively using an automated image captioning system.

What do they need to DO?
What do they need to do differently?
What job(s) do they want or need to get done?
What decision(s) do they need to make?
How will we know they were successful?

To efficiently add captions to images, save time, and ensure the generated captions are accurate and contextually relevant.

To decide on selecting the right image captioning tool, adjusting settings for customization, and reviewing or editing generated captions.

Success can be measured by the speed of caption generation, the accuracy of captions, user satisfaction, and reduced editing [illegible]

What do they HEAR?
What are they hearing others say?
What are they hearing from friends?
What are they hearing from colleagues?
What are they hearing second-hand?

They might hear about some news regarding our image caption generator app that it generates captions from the given input images

Hear constructive criticism from users about our app and make the suitable changes

Colleagues and friends discuss about what caption to choose ? from their given input for their personal , promotional works.

They might come across some of the captions and our new app indirectly through various sources like on internet where people share their works with captions

What do they THINK and FEEL?

PAINS
What are their fears, frustrations, and anxieties?

Ethical concerns: Image captioning can inadvertently generate biased or offensive captions, raising concerns about fairness and ethics...

Data limitations: The quality and size of training data can greatly affect the performance of the model. [illegible] captions.

Overfitting: Deep learning models may overfit the training data, leading to [illegible] their generalization.

Model Deployment: Deploying captioning models in real-world applications can be [illegible] infrastructure and maintenance considerations.

Real-time performance: Generating captions in real-time for applications [illegible]

GAINS
What are their wants, needs, hopes, and dreams?

Wanting the generated captions to accurately describe the content of the images, ensuring they make sense and provide meaningful information.

Needing the model to understand and incorporate contextual information to generate coherent and relevant captions.

Hoping the model can generate creative and engaging captions rather than merely factual descriptions.

Hoping that the project can scale efficiently to handle a large number of images and diverse use cases.

Wanting captions to be written in fluent and natural language, with proper grammar and style.

What other thoughts and feelings might influence their behavior?

If users think the system lacks accuracy, they may feel frustrated or skeptical, influencing their decision to use the tool.

The desire to receive positive feedback or engagement on social media might drive users to use image captioning for their posts, influenced by the perception of others.

Users may have concerns about the privacy implications of automated image captioning, leading them to be cautious or avoid certain features.

What do they SEE?
What do they see in the marketplace?
What do they see in their immediate environment?
What do they see others saying and doing?
What are they watching and reading?

[illegible]

What do they SAY?
What have we heard them say?
What can we imagine them saying?

Image caption generator adds context, utility, and creativity to images, making them more informative and engaging in various applications.

In social media and online content, descriptive captions can engage users, convey additional information, and elicit emotional responses.

What do they DO?
What do they do today?
What behavior have we observed?
What can we imagine them doing?

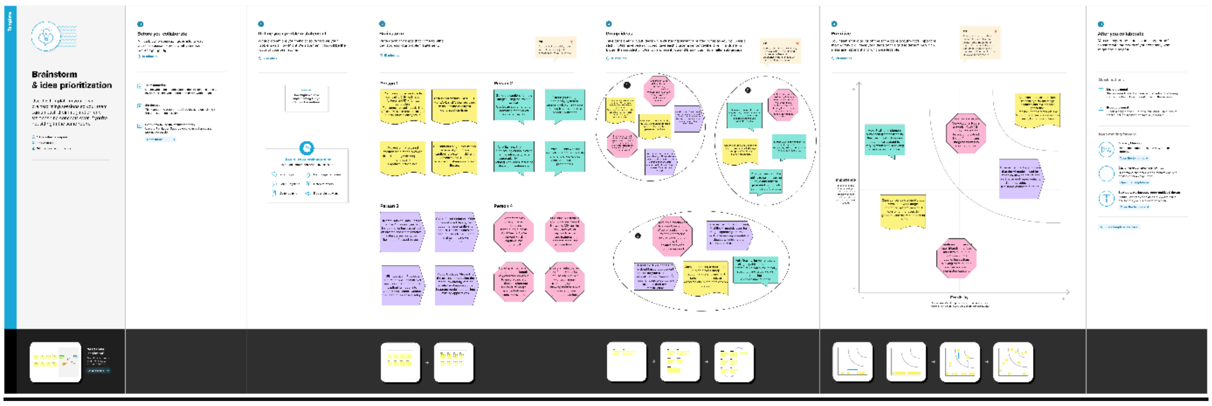
Users typically use various websites to copy the captions based on their requirement. They might follow some ai tools to generate new captions.

Users often express frustration while searching for websites that generates captions.

With an improvised Cap Gen App Users can generate new captions based on their given input images. It benefits the user by saving time to search for websites or ai tools that generate captions.

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

Ideation and Brainstorming



REQUIREMENT ANALYSIS

Functional Requirement

The primary functional requirement for image caption generation is the development of a robust AI system capable of accurately analyzing diverse images and generating clear, contextually relevant, and coherent descriptions in natural language. This system should effectively recognize objects, scenes, and relationships within images, translating this visual understanding into grammatically correct and meaningful sentences. Additionally, it should adapt to varying image complexities, handle ambiguous scenes, and avoid biases in its descriptions. The output should align closely with human-like comprehension, providing insightful and accurate captions that capture the essence of the depicted visual content.

Non Functional Requirement

Non-functional requirements for image caption generation encompass aspects beyond the system's specific functionalities, focusing on qualities like performance, usability, and reliability. These include:

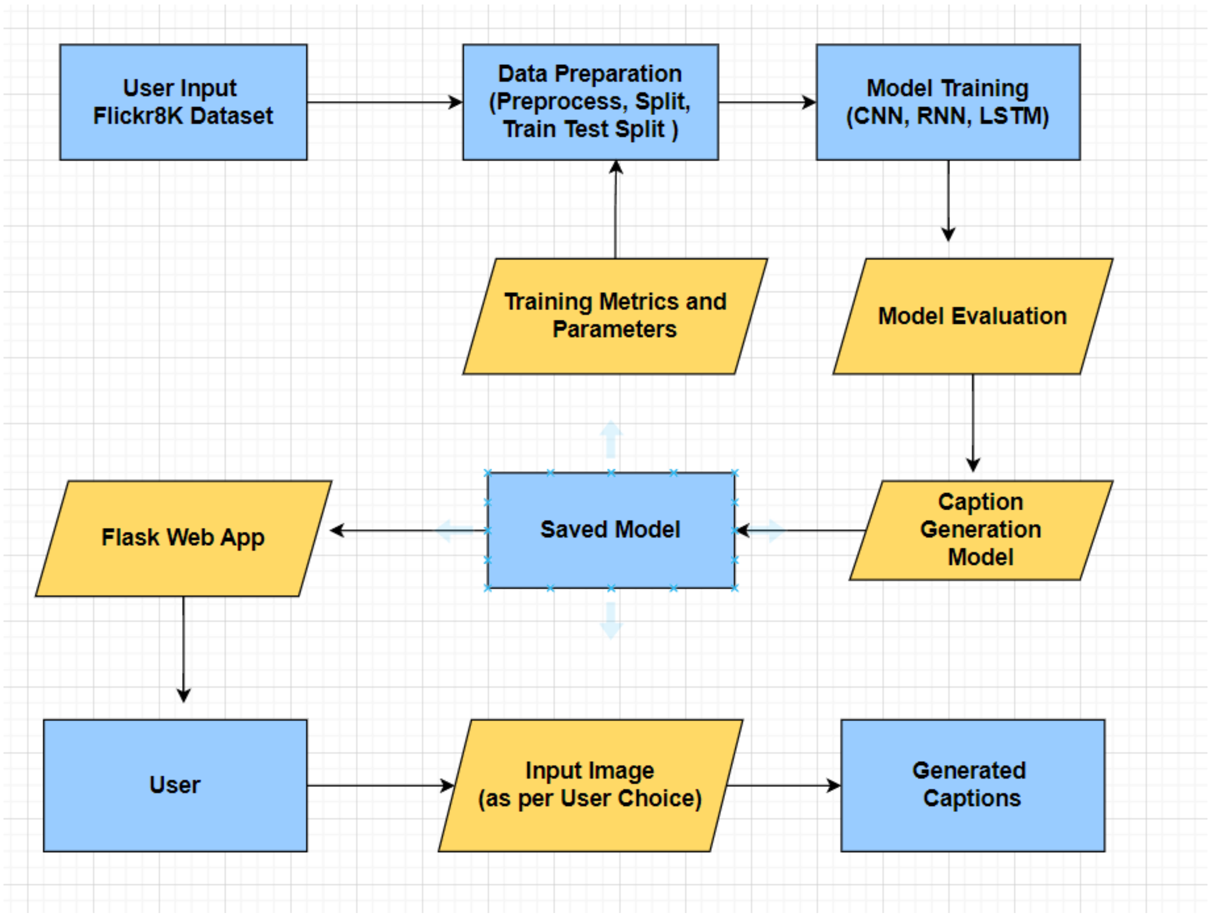
1. Performance: The system should exhibit reasonable response times for caption generation, ensuring timely and efficient processing of images without significant delays. It should also manage computational resources efficiently to handle a variety of image complexities.
2. Accuracy and Quality: Aim for high accuracy in caption generation, minimizing errors or inaccuracies in descriptions. The captions should be contextually

relevant and linguistically coherent, reflecting a comprehensive understanding of the image content.

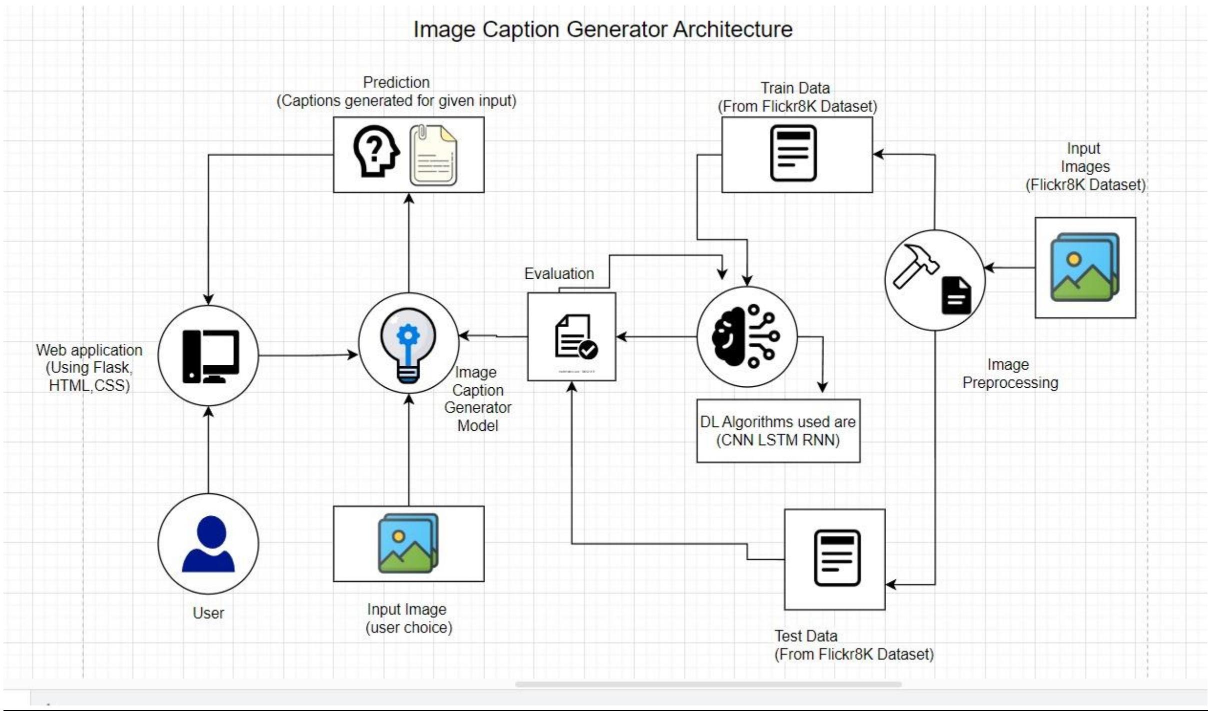
3. Scalability: The system should be capable of scaling with increased data or user demand, accommodating larger image datasets or higher user traffic without compromising performance or quality.
4. Robustness: Ensure the model's stability and resilience against variations in input images, handling diverse image types, sizes, and qualities effectively without a significant decline in performance or accuracy.
5. Ethical Considerations: Address biases in the generated captions, striving for fairness and inclusivity in the descriptions. Ensure that the system adheres to ethical guidelines, avoiding offensive or discriminatory language in its outputs.

<u>PROJECT DESIGN</u>

<u>Data Flow Diagram</u>



SOLUTION ARCHITECTURE



TECHNICAL ARCHITECTURE

Technical architecture encompasses the fundamental structure and design of a software system, outlining its key components, their interactions, and the underlying framework that enables its functionality. This includes hardware infrastructure, software modules, databases, networking protocols, and interfaces. It also addresses critical considerations such as data storage, scalability, security measures, and compliance with industry standards. The architecture dictates how the system handles increasing loads and ensures optimal performance. It encompasses deployment strategies, whether on-premises or on cloud platforms, as well as procedures for error handling, recovery, and backups. Integration with external services and APIs is also specified, enabling seamless interaction with third-party applications. Furthermore, it delineates development, testing, staging, and production environments, ensuring consistency across different stages of the software development life cycle. Monitoring and logging mechanisms are put in place to track system behavior, performance metrics, and errors. Maintenance procedures and strategies for upgrades are defined to keep the system up-to-date and efficient. Documentation is crucial, providing insights into architectural decisions and best practices, facilitating knowledge transfer among team members and aiding in troubleshooting. In essence, technical architecture serves as the foundational blueprint for the construction and maintenance of a robust and effective software system.

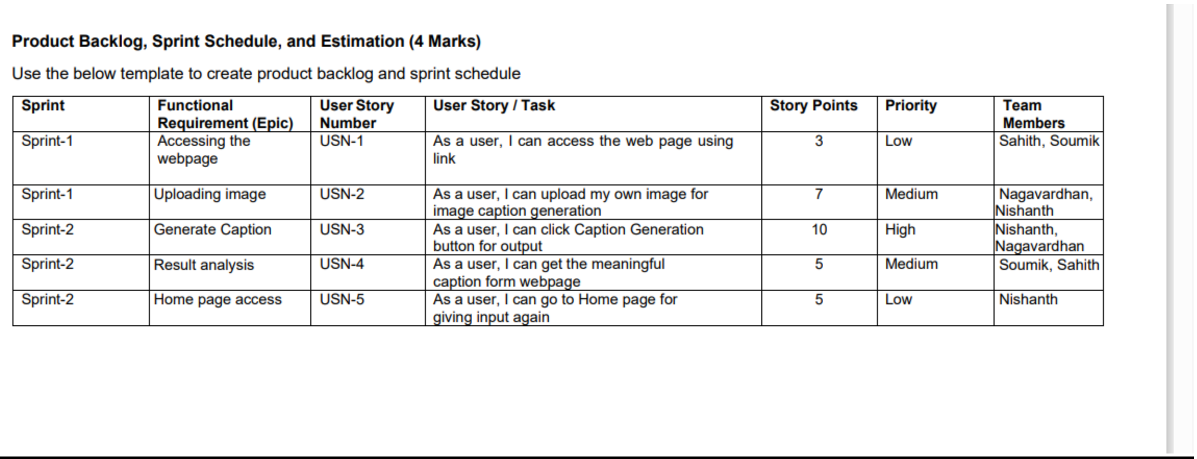
Table-1
Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface Design	To display the image for which the user wants to generate a caption.	HTML, CSS.
2.	User Experience	Image caption generation delivers quick descriptions for images, with user-friendly features like editing, and easy sharing for an enhanced experience.	HTML, CSS.
3.	Application Logic - 1	Image Processing: Preprocess the user-uploaded image.	TensorFlow.
4.	Application Logic - 2	Feature Extraction and Model Prediction: Extract features using a pre-trained model. Predict captions using a sequence-to-sequence model.	TensorFlow.
5.	Application Logic - 3	Post-processing and Output: Post-process the generated caption. Display the final caption to the user.	TensorFlow.
6.	Machine Learning Model	Image caption generation ML models automatically describe images, enhancing accessibility, user experience, content organization, and supporting applications.	TensorFlow.
7.	Flask Integration	Web pages for home page and result are integrate d with flask app.py	VS Code

Table-2
Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	TensorFlow, VS Code
2.	Availability	Achieve high availability for image caption generation with load balancing, redundancy.	Tensorflow, Flask
3.	Performance	Image caption generation performance is assessed through metrics like accuracy, model architecture, and inference speed.	TensorFlow.
4.	Offline Access	Allow utilize the application without a continuous internet connection.	Flask.
5.	Image Upload and Processing	Allow users to upload images for caption generation with backend processing.	Flask
6.	Responsive Web Design	Ensure the web application is accessible and user-friendly on various devices.	HTML, CSS

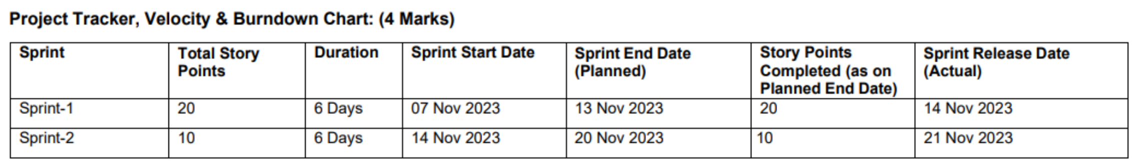
SPRINT PLANNING AND ESTIMATION



Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Accessing the webpage	USN-1	As a user, I can access the web page using link	3	Low	Sahith, Soumik
Sprint-1	Uploading image	USN-2	As a user, I can upload my own image for image caption generation	7	Medium	Nagavardhan, Nishanth
Sprint-2	Generate Caption	USN-3	As a user, I can click Caption Generation button for output	10	High	Nishanth, Nagavardhan
Sprint-2	Result analysis	USN-4	As a user, I can get the meaningful caption form webpage	5	Medium	Soumik, Sahith
Sprint-2	Home page access	USN-5	As a user, I can go to Home page for giving input again	5	Low	Nishanth



Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	07 Nov 2023	13 Nov 2023	20	14 Nov 2023
Sprint-2	10	6 Days	14 Nov 2023	20 Nov 2023	10	21 Nov 2023

$$AV = \frac{sprint\ duration}{velocity}$$

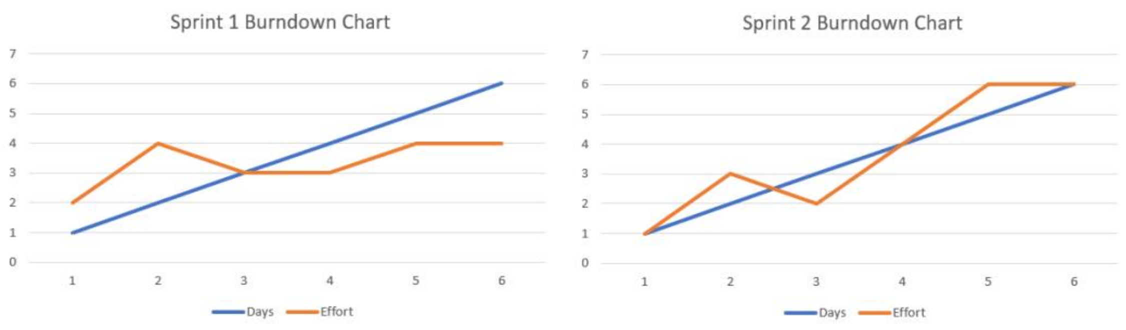
Sprint 1 AV = 20/6 = 3.33

Sprint 2 AV = 10/6 = 1.66

BURNDOWN CHARTS

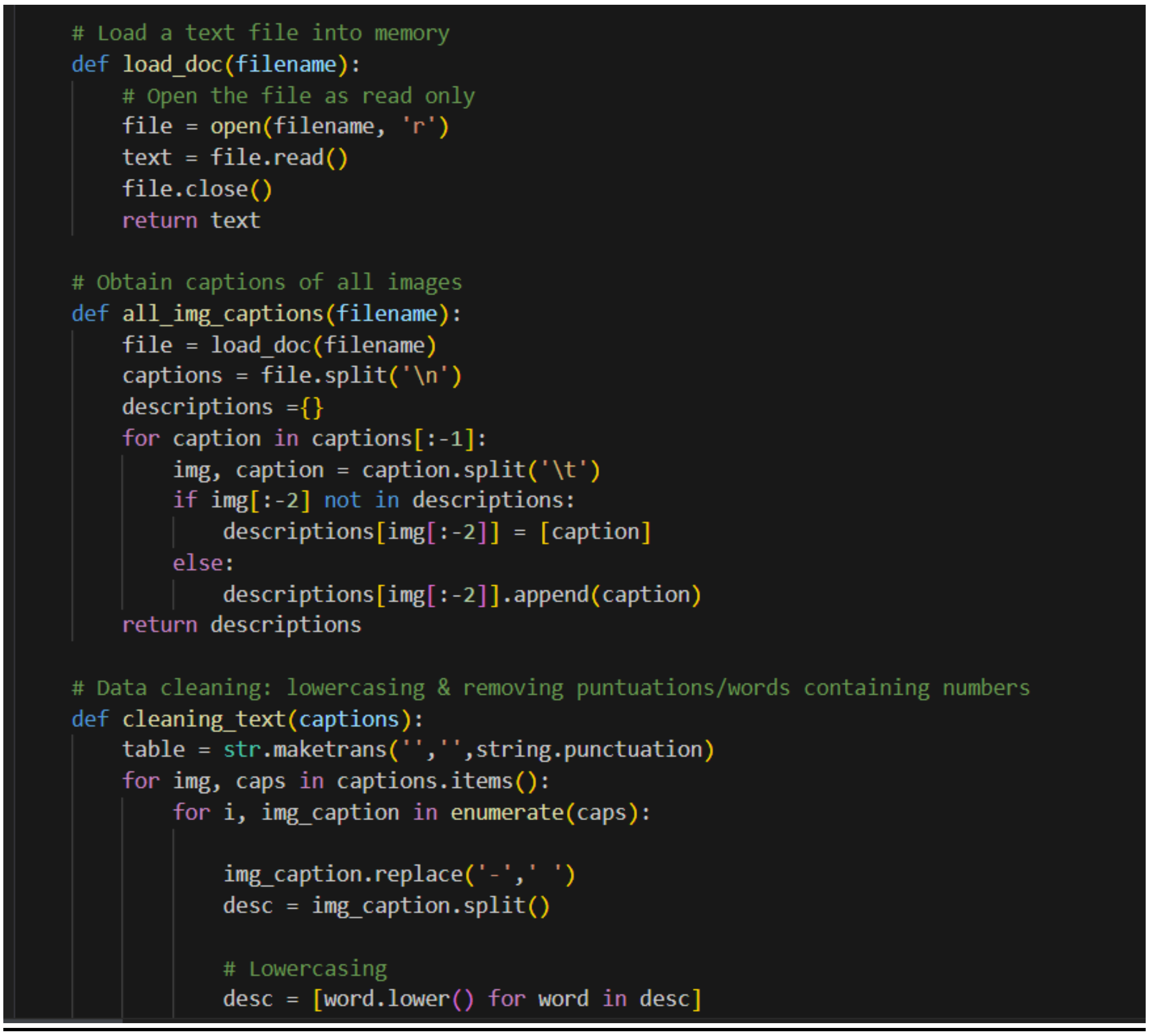
Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

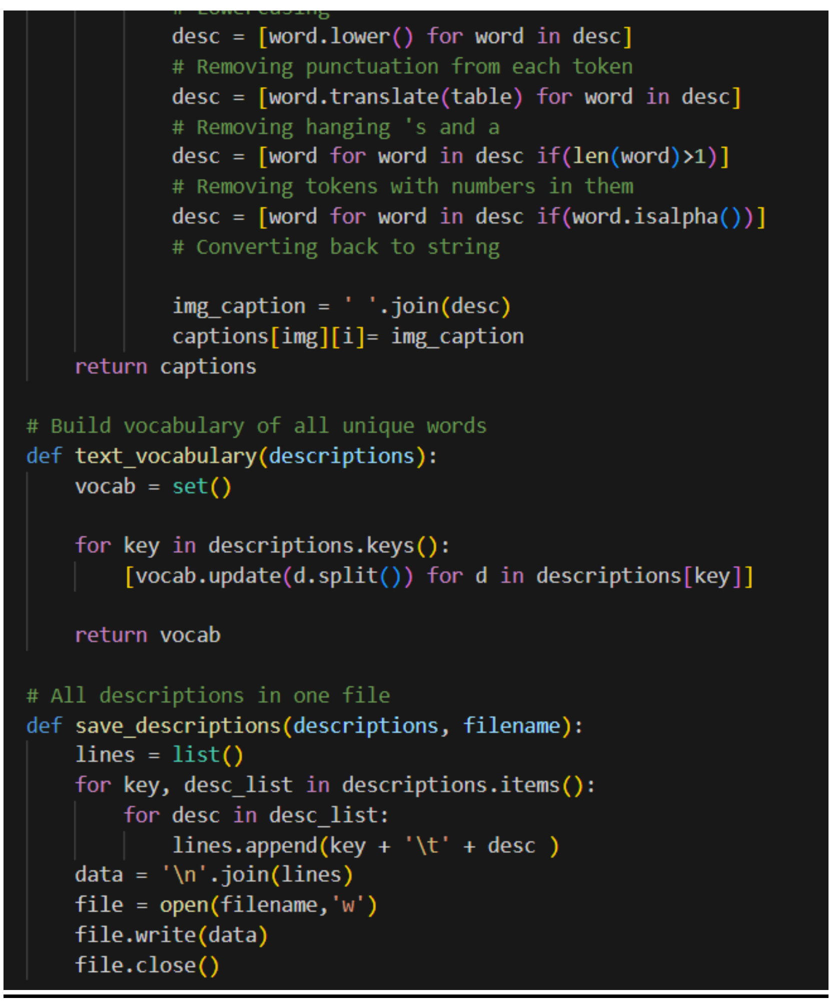


CODING AND SOLUTIONING

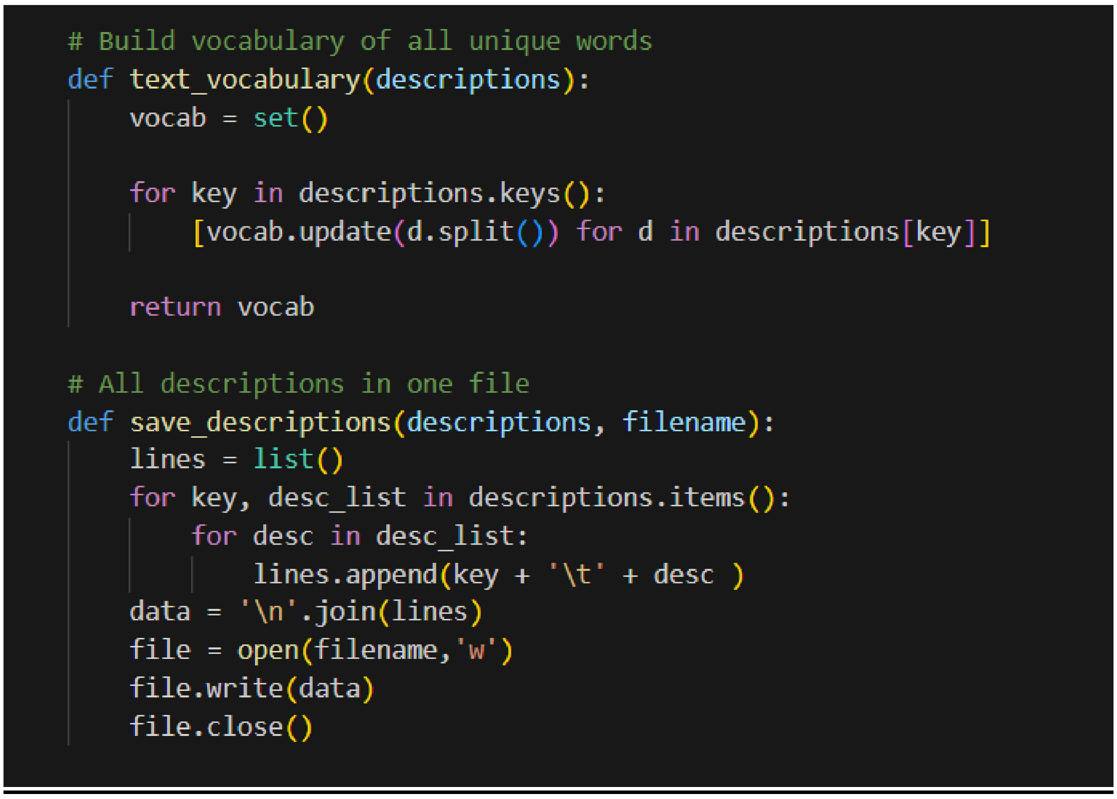
```
import string
import numpy as np
import os
import cv2
from pickle import dump, load

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()
```



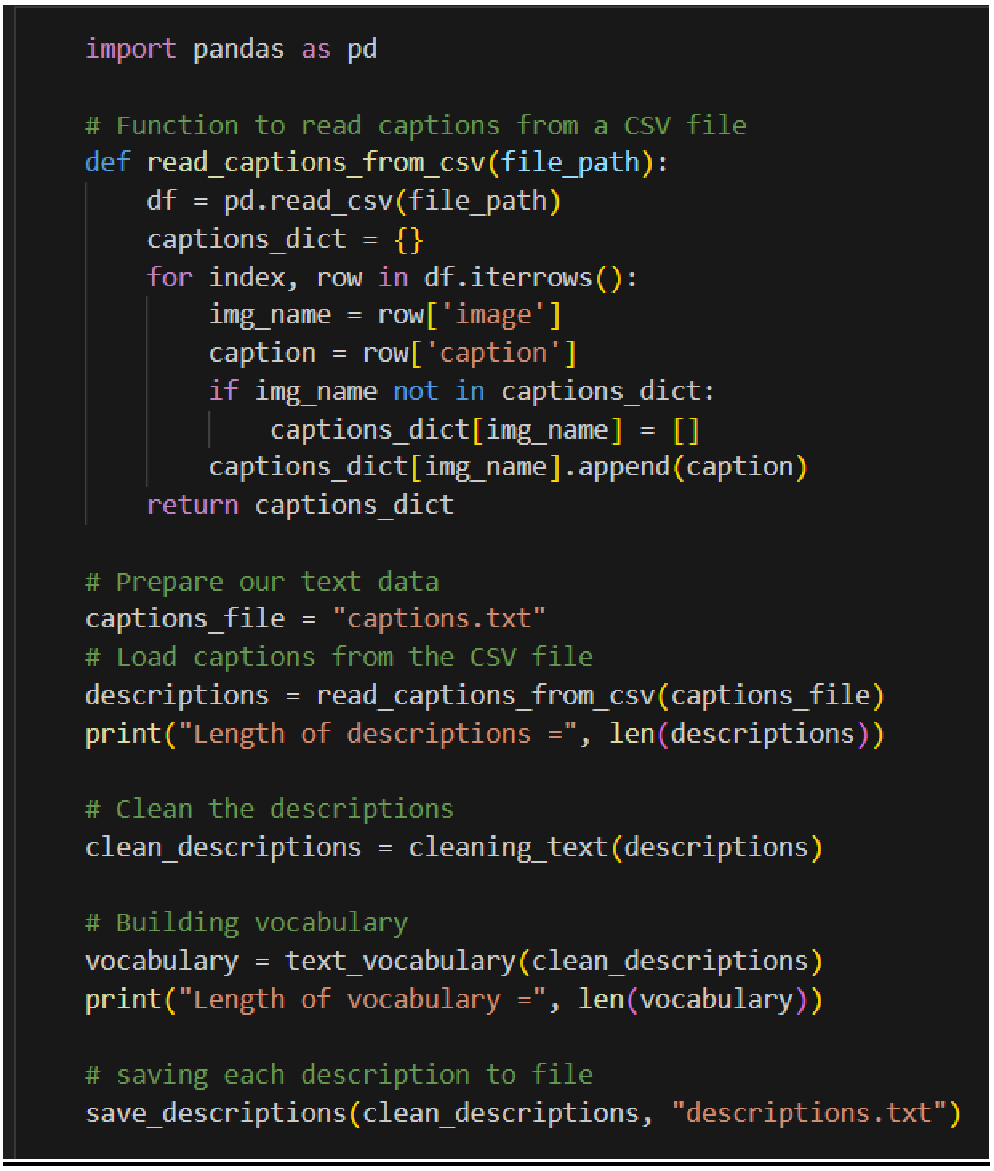
```python
# Load a text file into memory
def load_doc(filename):
    # Open the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text

# 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

# 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]
```

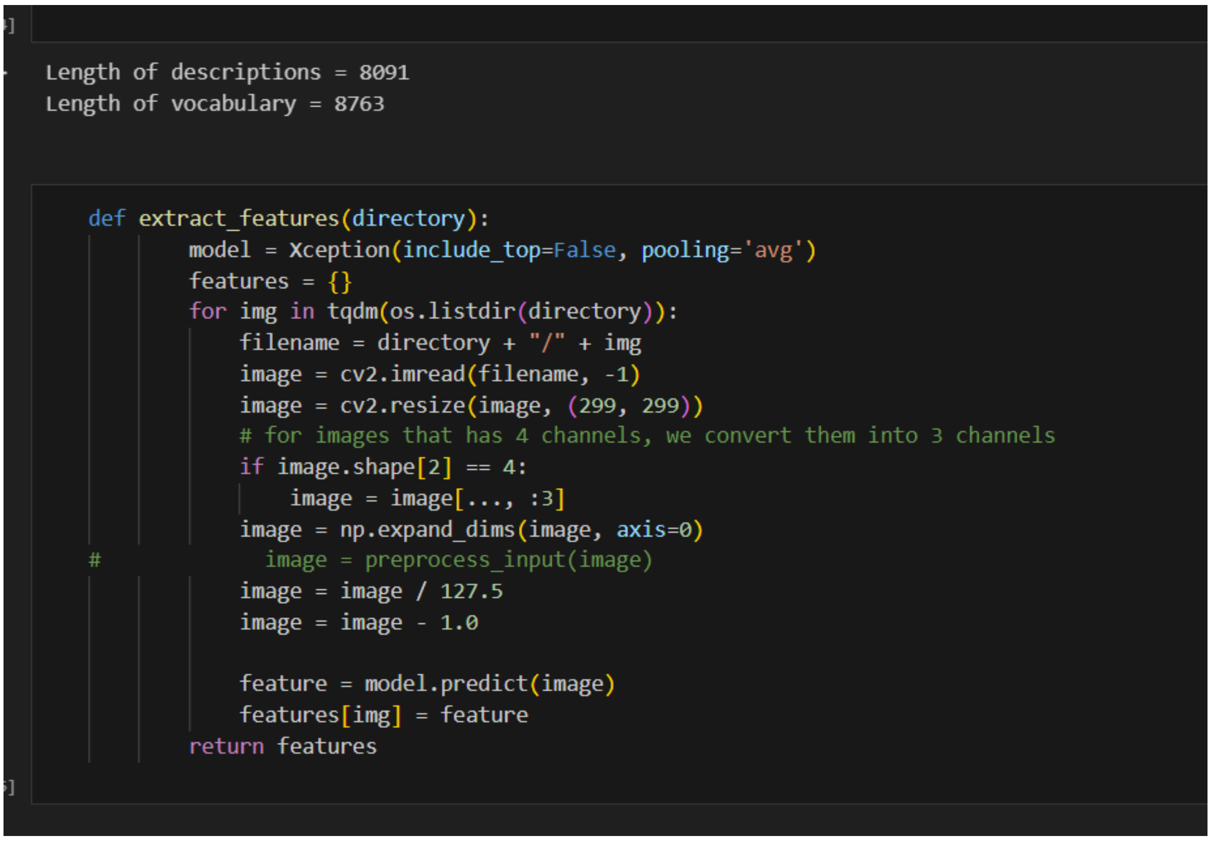


```
            # Lowercasing
            desc = [word.lower() for word in desc]
            # Removing punctuation from each token
            desc = [word.translate(table) for word in desc]
            # Removing hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            # 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

# 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()
```



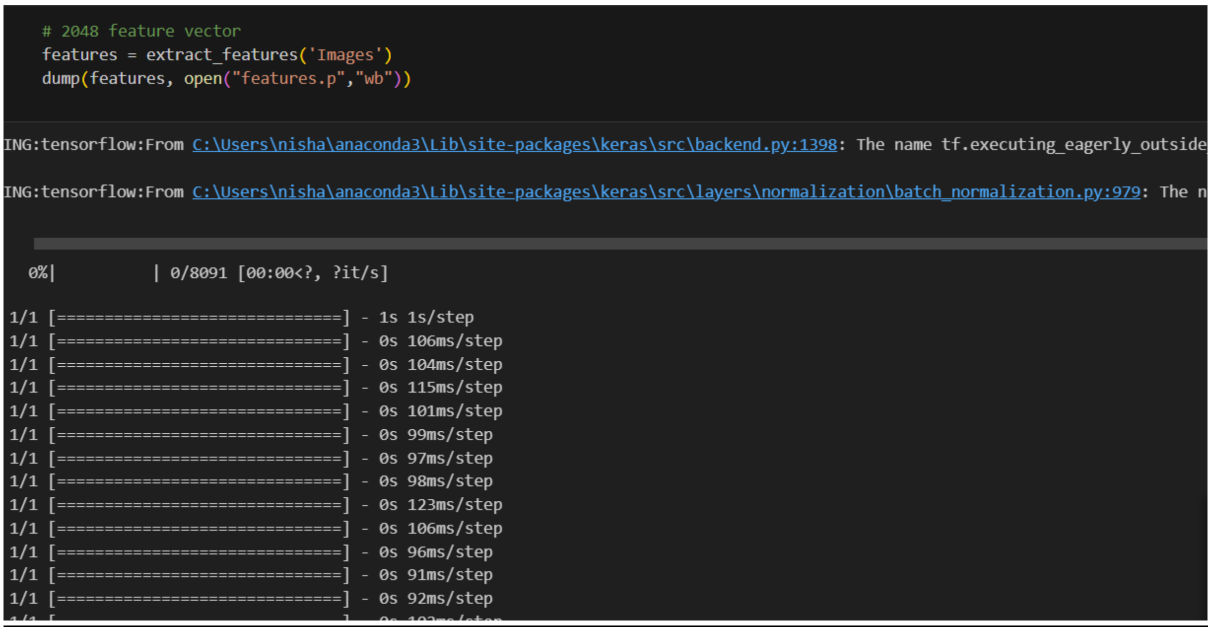
```
# Build vocabulary of all unique words
def text_vocabulary(descriptions):
    vocab = set()

    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()
```


```python
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")
```



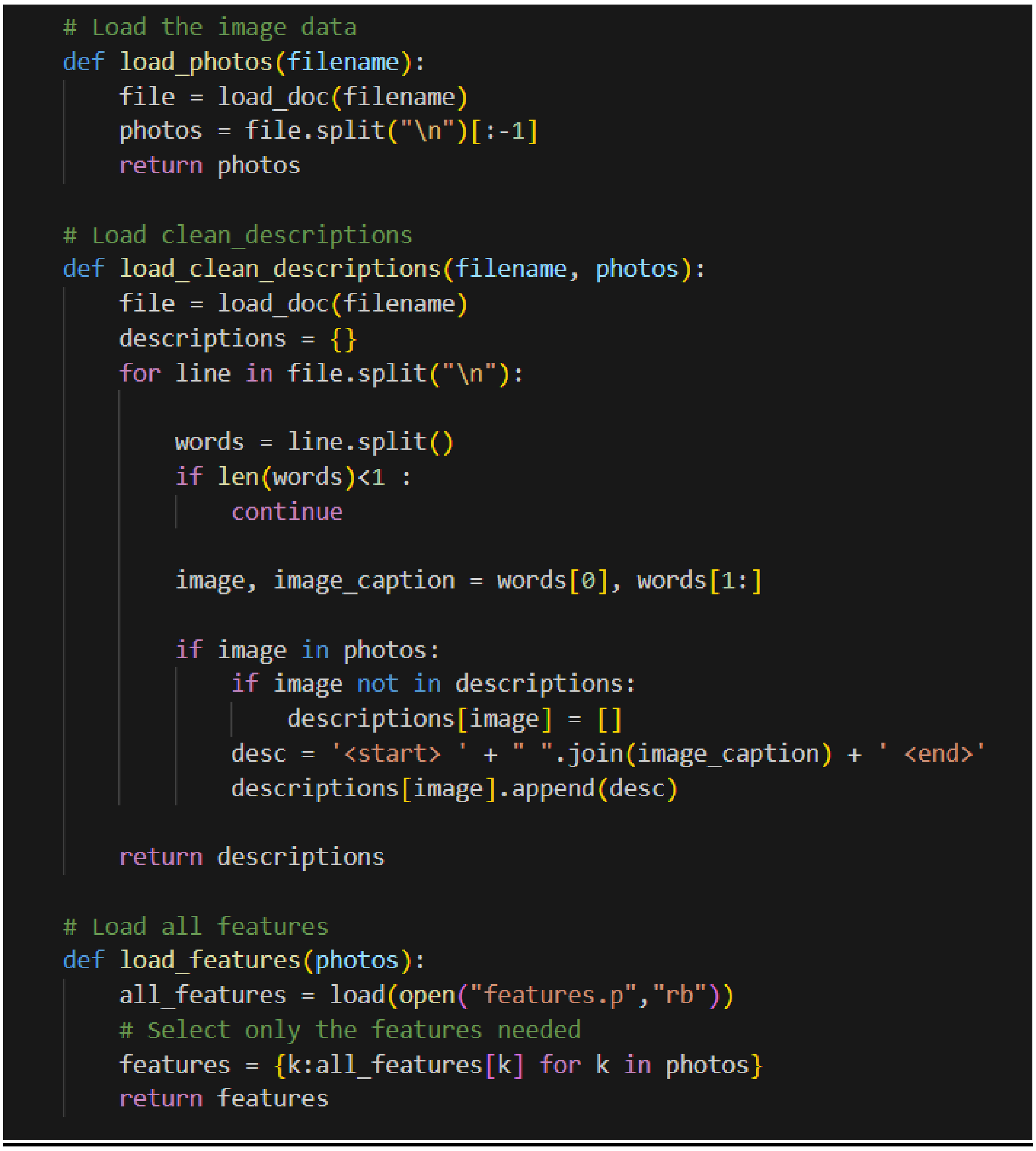
```
Length of descriptions = 8091
Length of vocabulary = 8763
```

```python
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
```

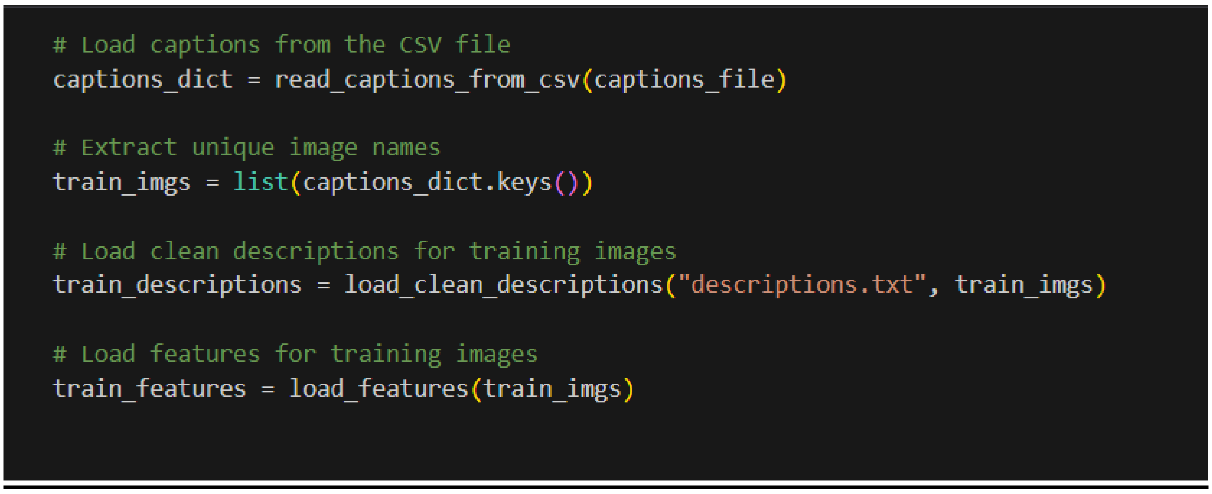


```python
# 2048 feature vector
features = extract_features('Images')
dump(features, open("features.p","wb"))
```

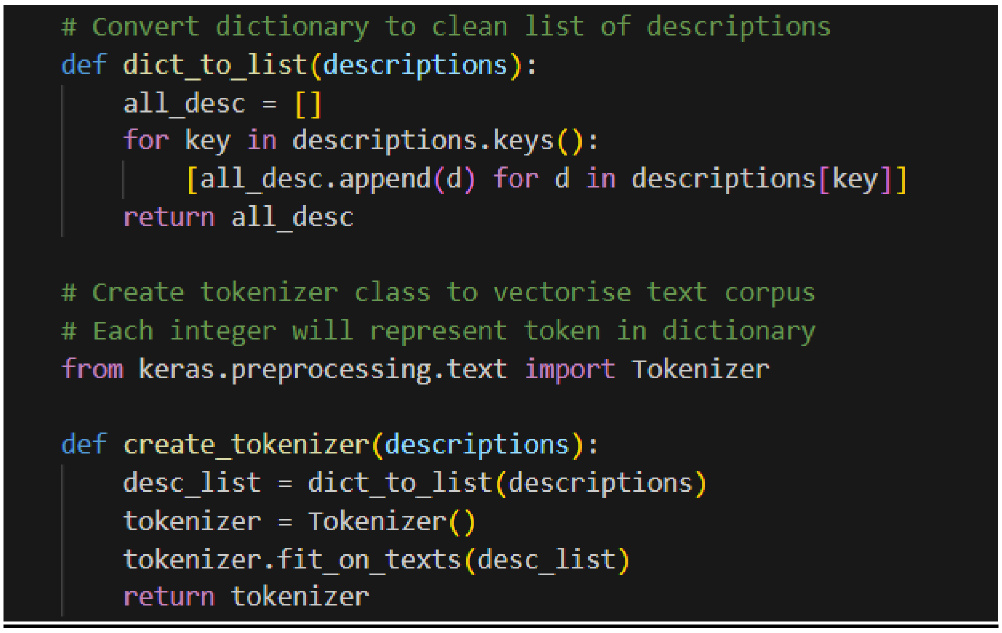
```
ING:tensorflow:From C:\Users\nisha\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside

ING:tensorflow:From C:\Users\nisha\anaconda3\Lib\site-packages\keras\src\layers\normalization\batch_normalization.py:979: The n

  0%|          | 0/8091 [00:00<?, ?it/s]

1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 104ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 98ms/step
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 92ms/step
```



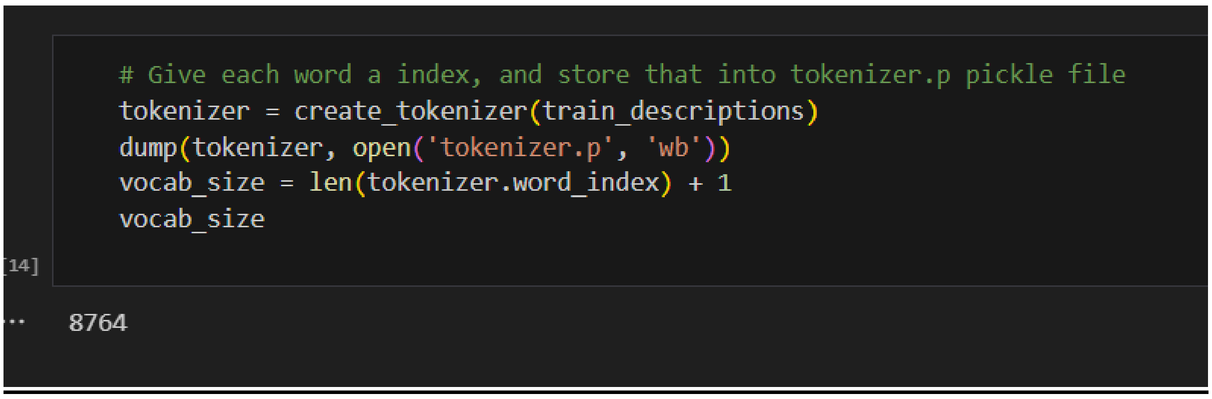
```python
# Load the image data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos

# 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

# 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
# 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)
```

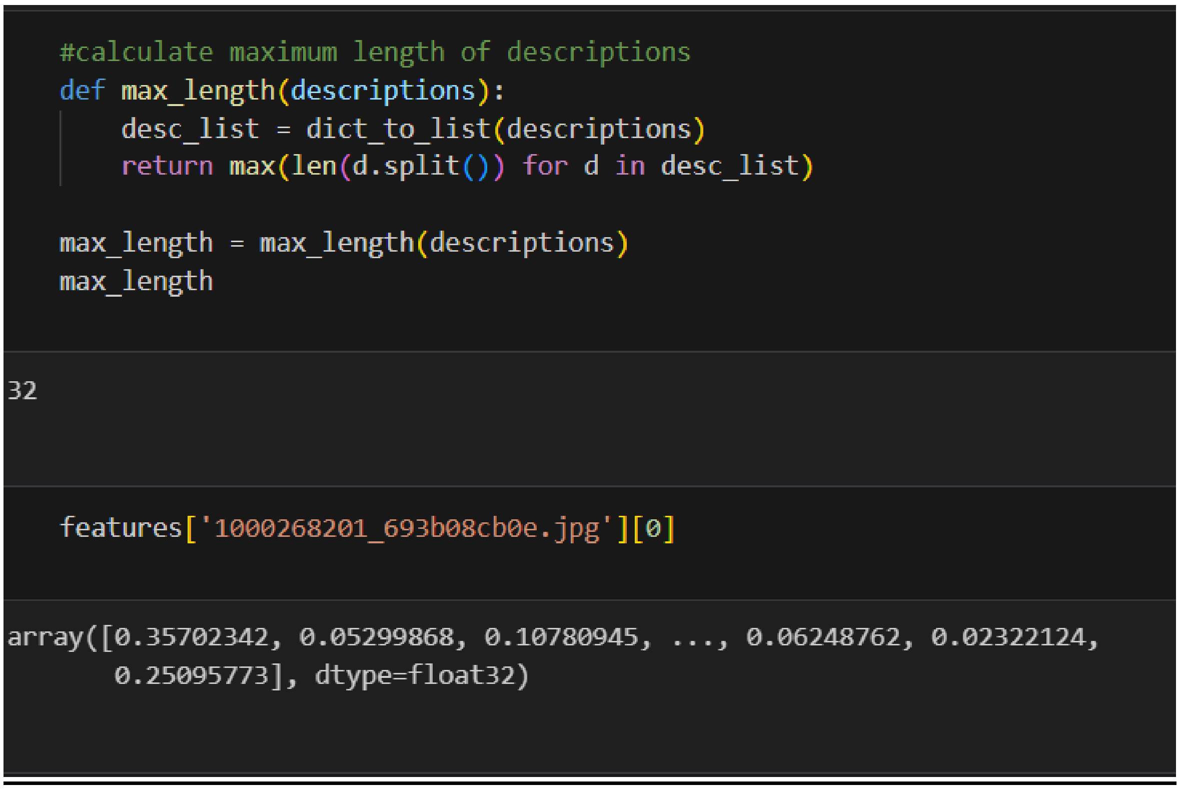


```python
# Convert dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# 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
```

```
[14]
... 8764
```

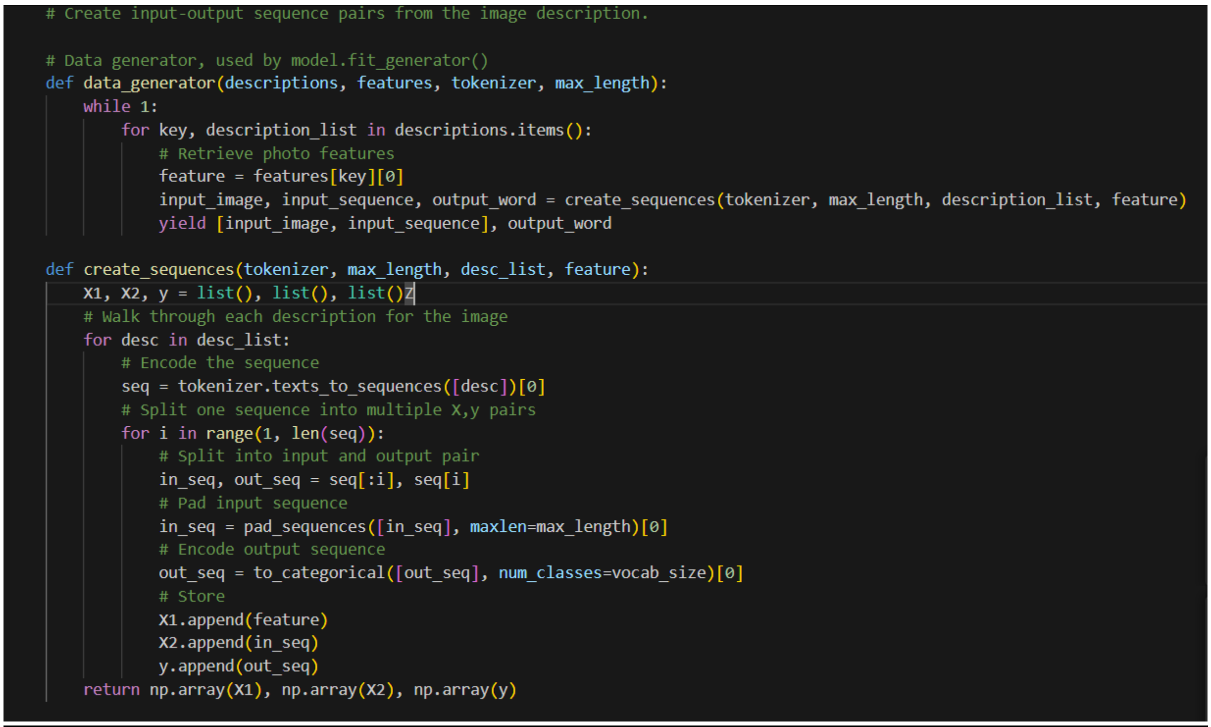


```python
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

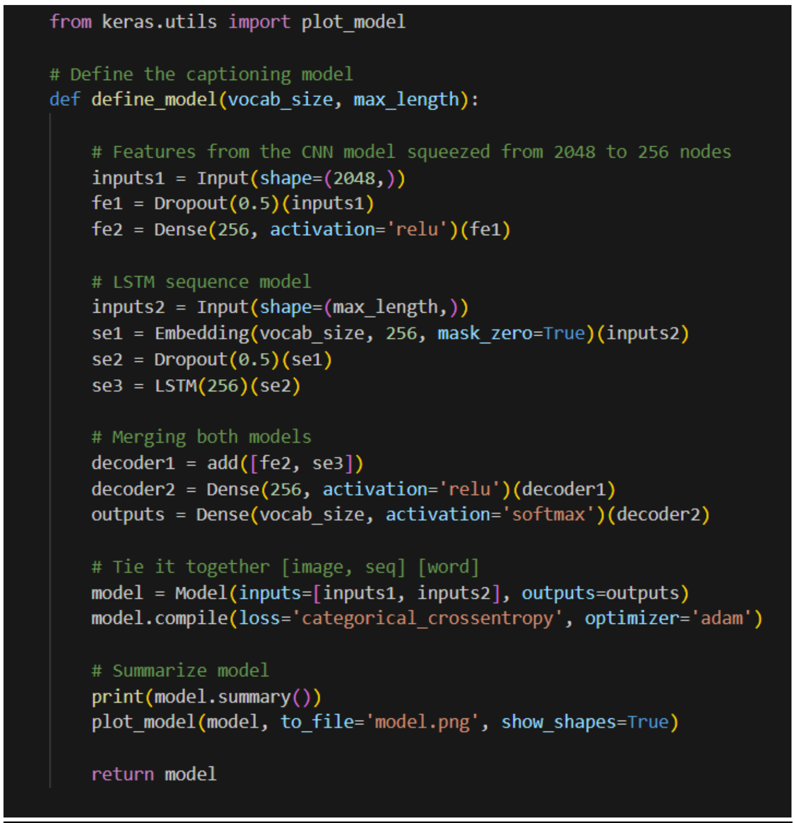
```
32
```

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

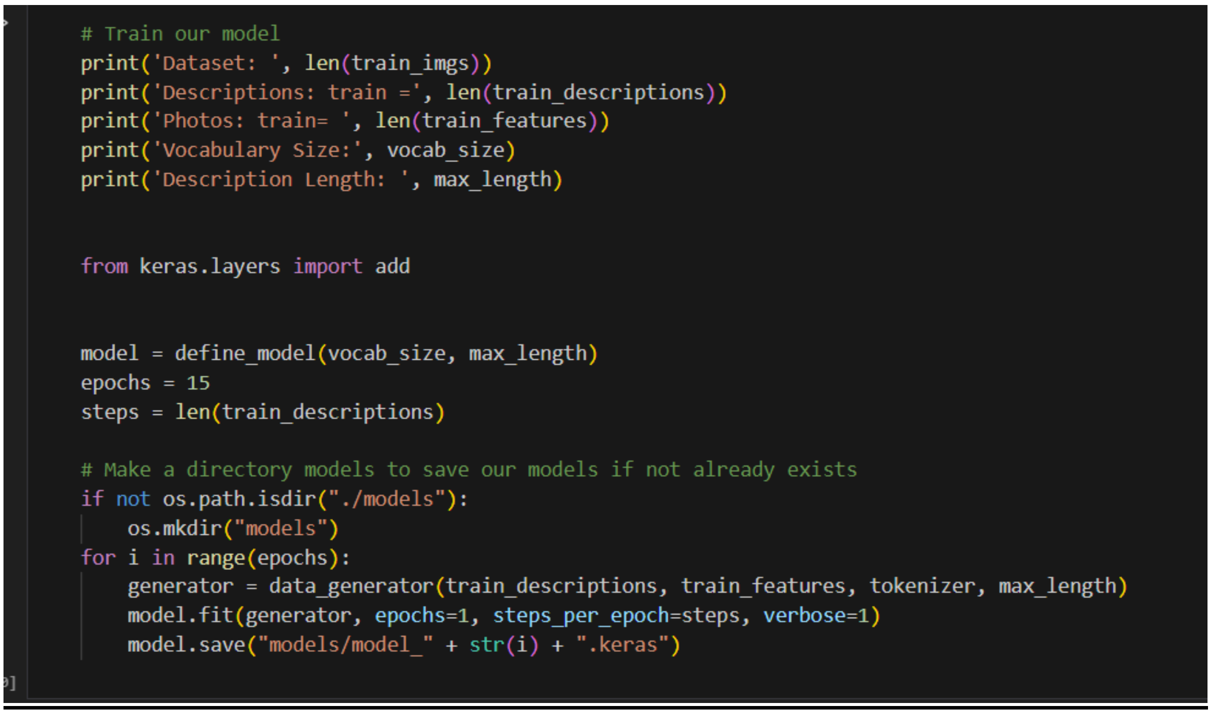
```
array([0.35702342, 0.05299868, 0.10780945, ..., 0.06248762, 0.02322124,
       0.25095773], dtype=float32)
```



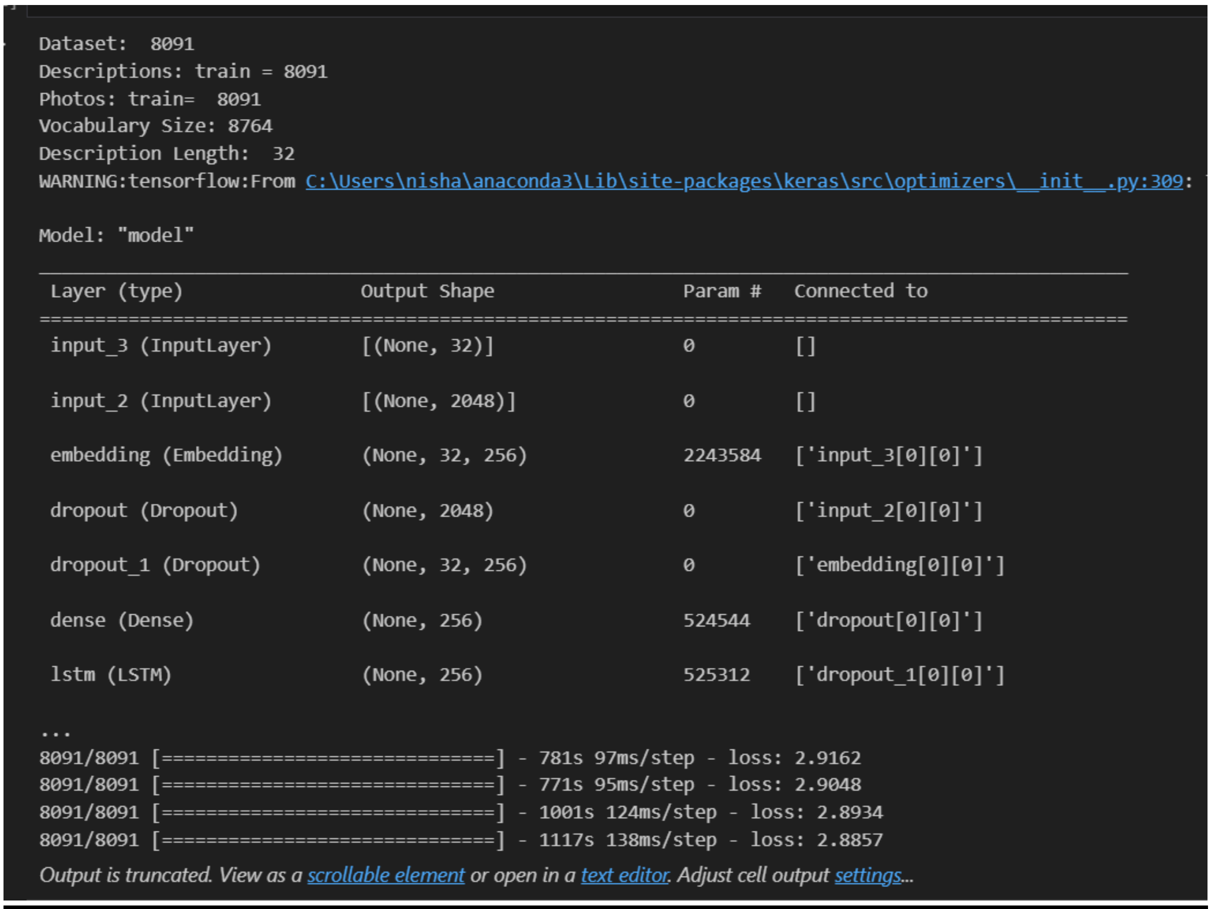
```python
# Create input-output sequence pairs from the image description.

# 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(tokenizer, max_length, description_list, feature)
            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)
```



```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
```

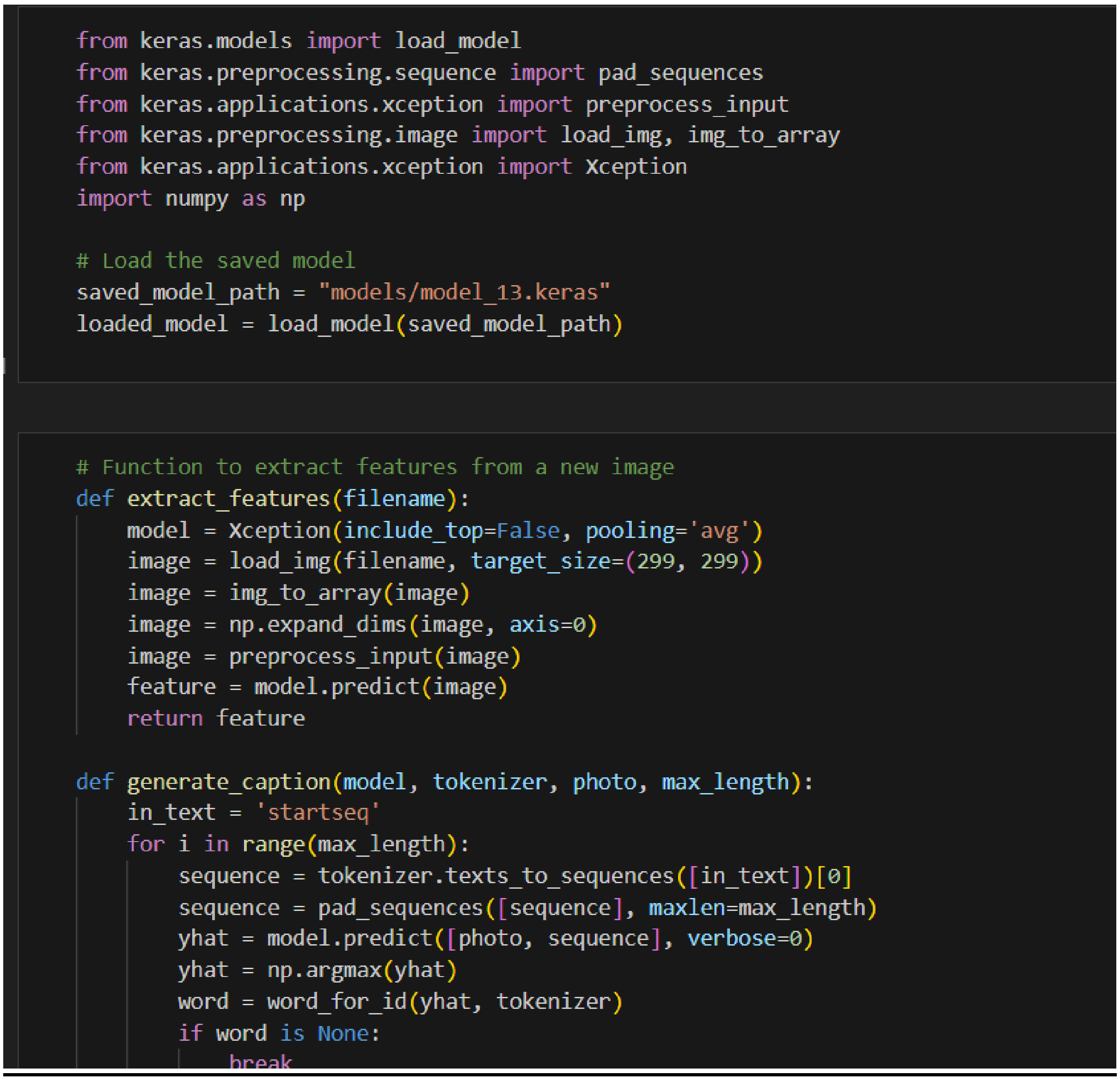


```python
# Train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train =', len(train_descriptions))
print('Photos: train= ', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)


from keras.layers import add


model = define_model(vocab_size, max_length)
epochs = 15
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, max_length)
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save("models/model_" + str(i) + ".keras")
```



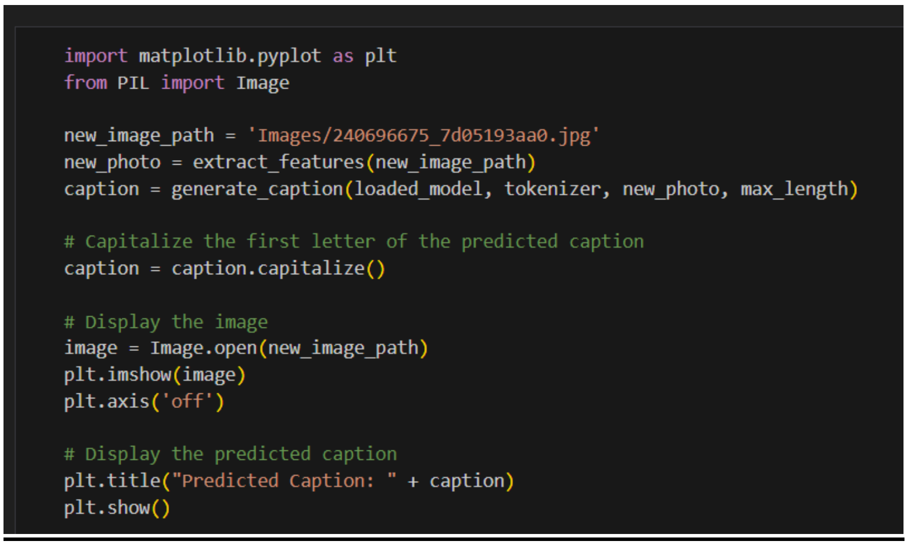
```
Dataset:  8091
Descriptions: train = 8091
Photos: train=  8091
Vocabulary Size: 8764
Description Length:  32
WARNING:tensorflow:From C:\Users\nisha\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309:

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         ['embedding[0][0]']

 dense (Dense)               (None, 256)                  524544    ['dropout[0][0]']

 lstm (LSTM)                 (None, 256)                  525312    ['dropout_1[0][0]']

...
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
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

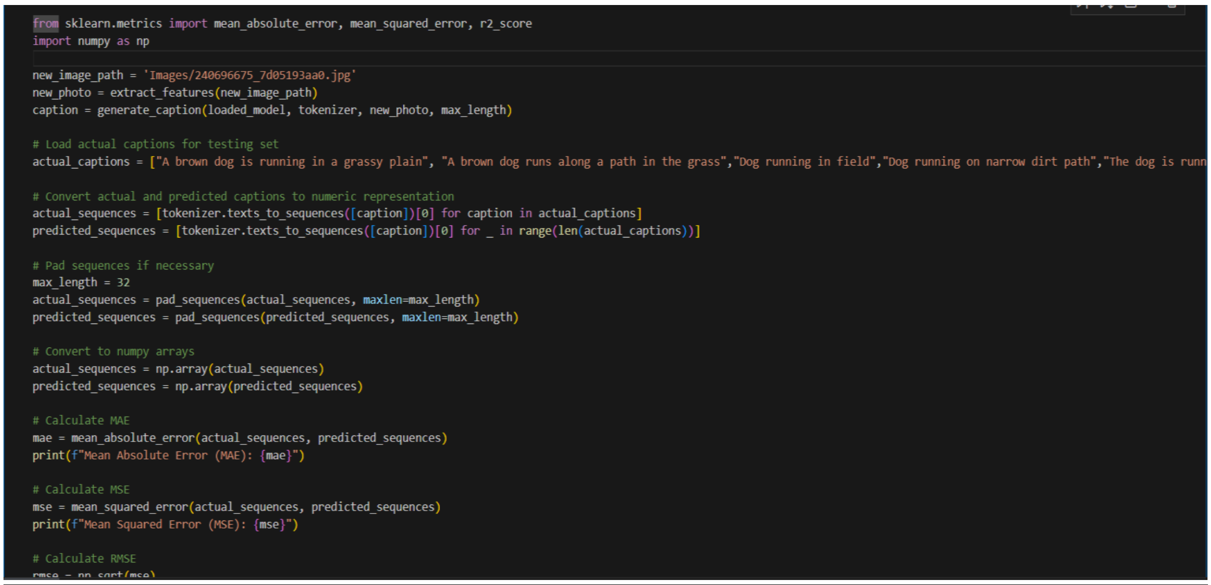


```python
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.xception import Xception
import numpy as np

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

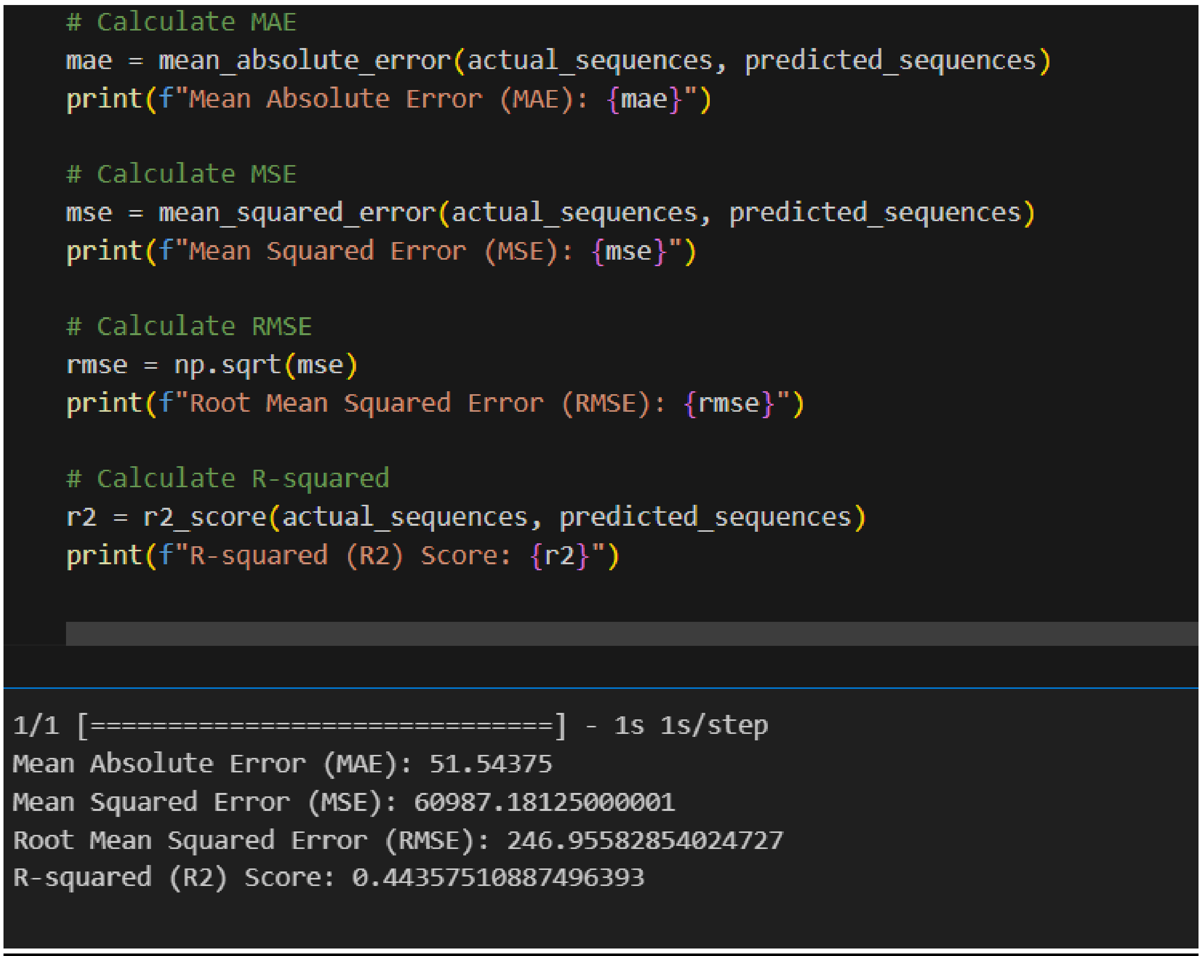
```python
# Function to extract features from a new image
def extract_features(filename):
    model = Xception(include_top=False, pooling='avg')
    image = load_img(filename, target_size=(299, 299))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)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



```python
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()
```



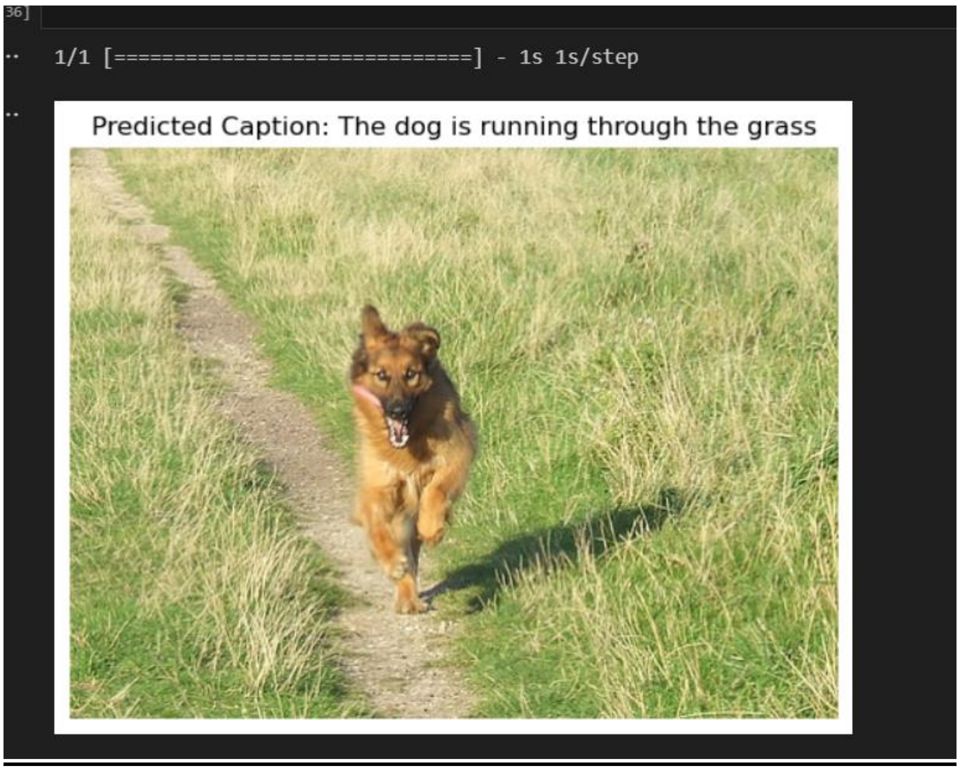
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

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

# Load actual captions for testing set
actual_captions = ["A brown dog is running in a grassy plain", "A brown dog runs along a path in the grass","Dog running in field","Dog running on narrow dirt path","The dog is runn

# Convert actual and predicted captions to numeric representation
actual_sequences = [tokenizer.texts_to_sequences([caption])[0] for caption in actual_captions]
predicted_sequences = [tokenizer.texts_to_sequences([caption])[0] for _ in range(len(actual_captions))]

# Pad sequences if necessary
max_length = 32
actual_sequences = pad_sequences(actual_sequences, maxlen=max_length)
predicted_sequences = pad_sequences(predicted_sequences, maxlen=max_length)

# Convert to numpy arrays
actual_sequences = np.array(actual_sequences)
predicted_sequences = np.array(predicted_sequences)

# Calculate MAE
mae = mean_absolute_error(actual_sequences, predicted_sequences)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate MSE
mse = mean_squared_error(actual_sequences, predicted_sequences)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate RMSE
rmse = np.sqrt(mse)
```



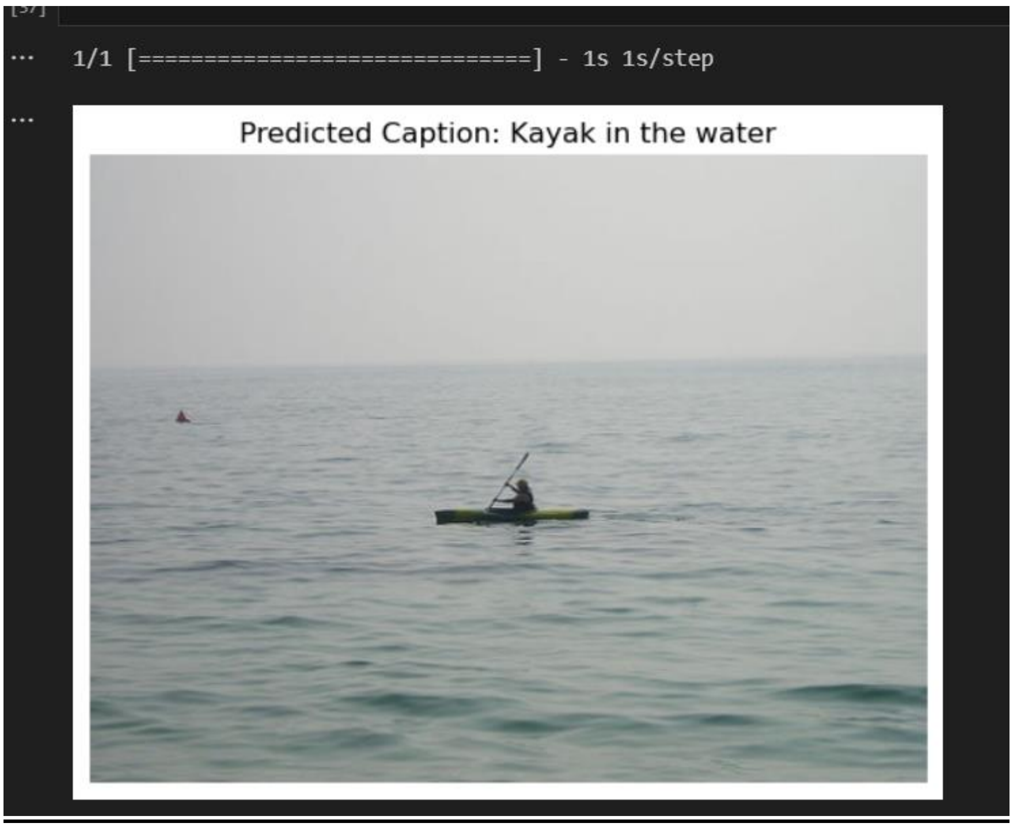
```python
# Calculate MAE
mae = mean_absolute_error(actual_sequences, predicted_sequences)
print(f"Mean Absolute Error (MAE): {mae}")

# Calculate MSE
mse = mean_squared_error(actual_sequences, predicted_sequences)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate RMSE
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")

# Calculate R-squared
r2 = r2_score(actual_sequences, predicted_sequences)
print(f"R-squared (R2) Score: {r2}")
```

```
1/1 [==============================] - 1s 1s/step
Mean Absolute Error (MAE): 51.54375
Mean Squared Error (MSE): 60987.18125000001
Root Mean Squared Error (RMSE): 246.95582854024727
R-squared (R2) Score: 0.44357510887496393
```

RESULTS

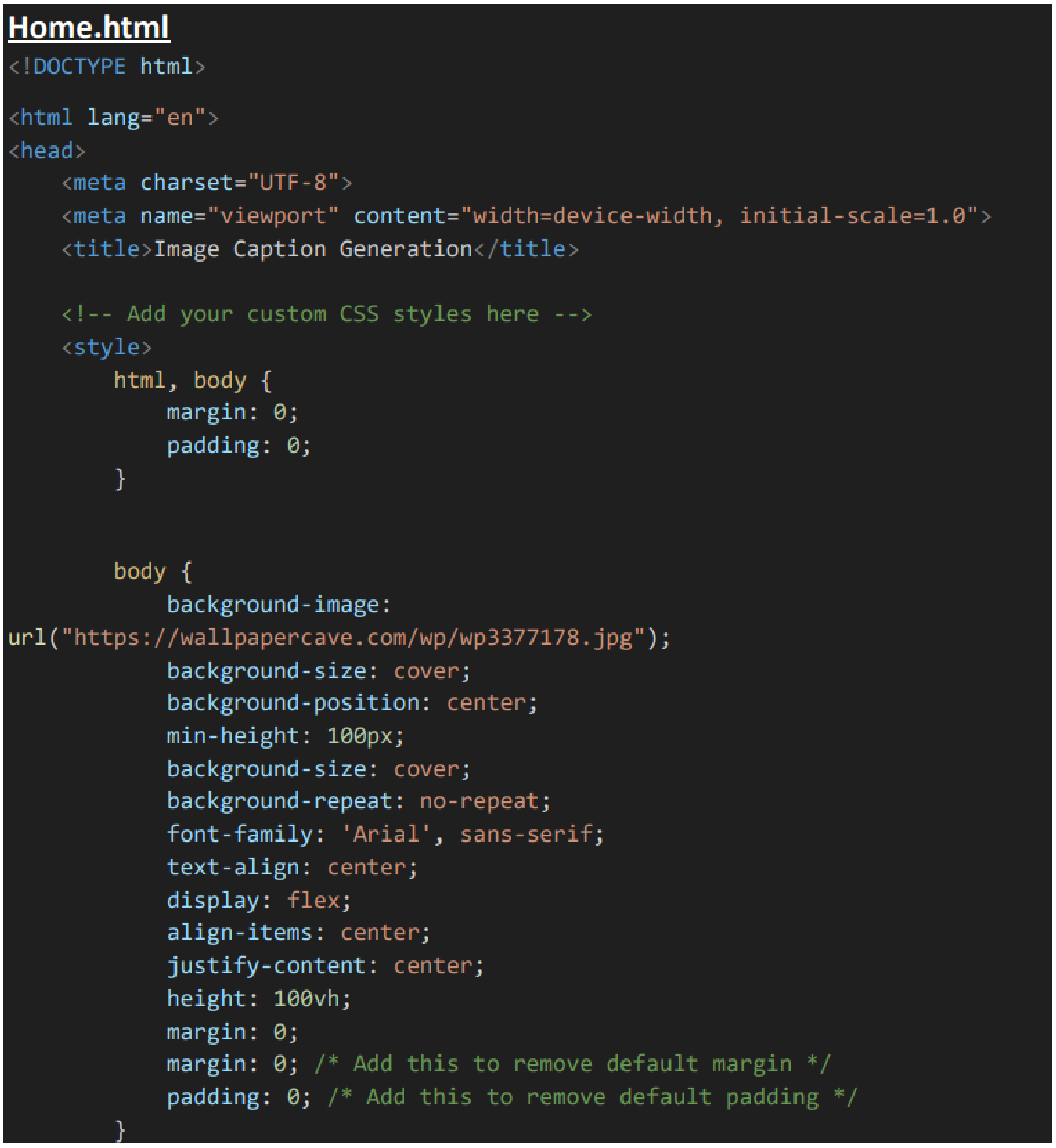


```
36]
..  1/1 [==============================] - 1s 1s/step
..
```

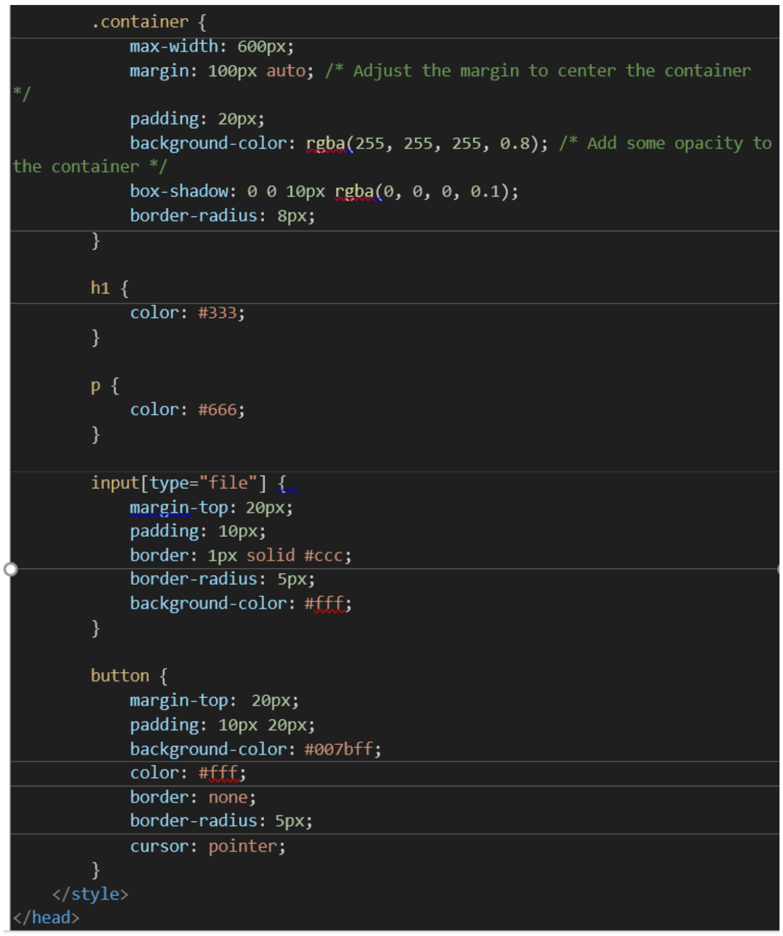


```
[37]
...  1/1 [==============================] - 1s 1s/step
...
```

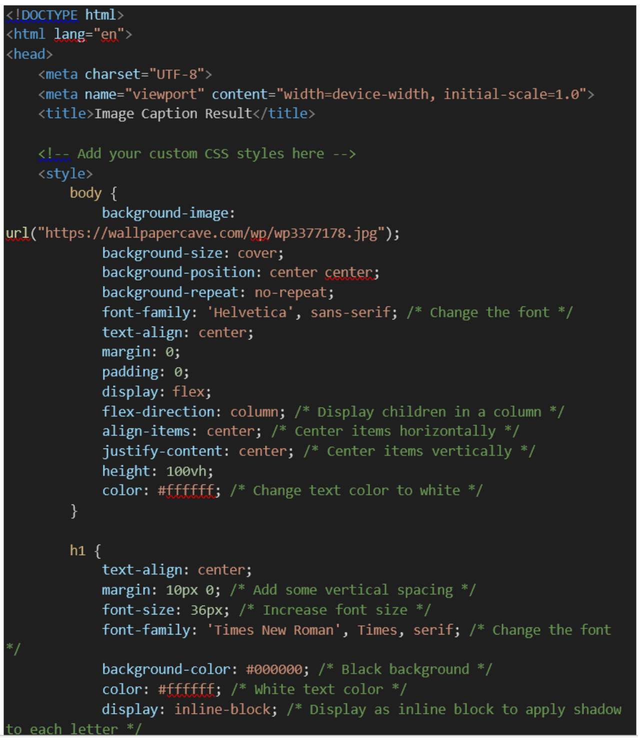
HTML CSS CODE SCREENSHOTS


Home.html

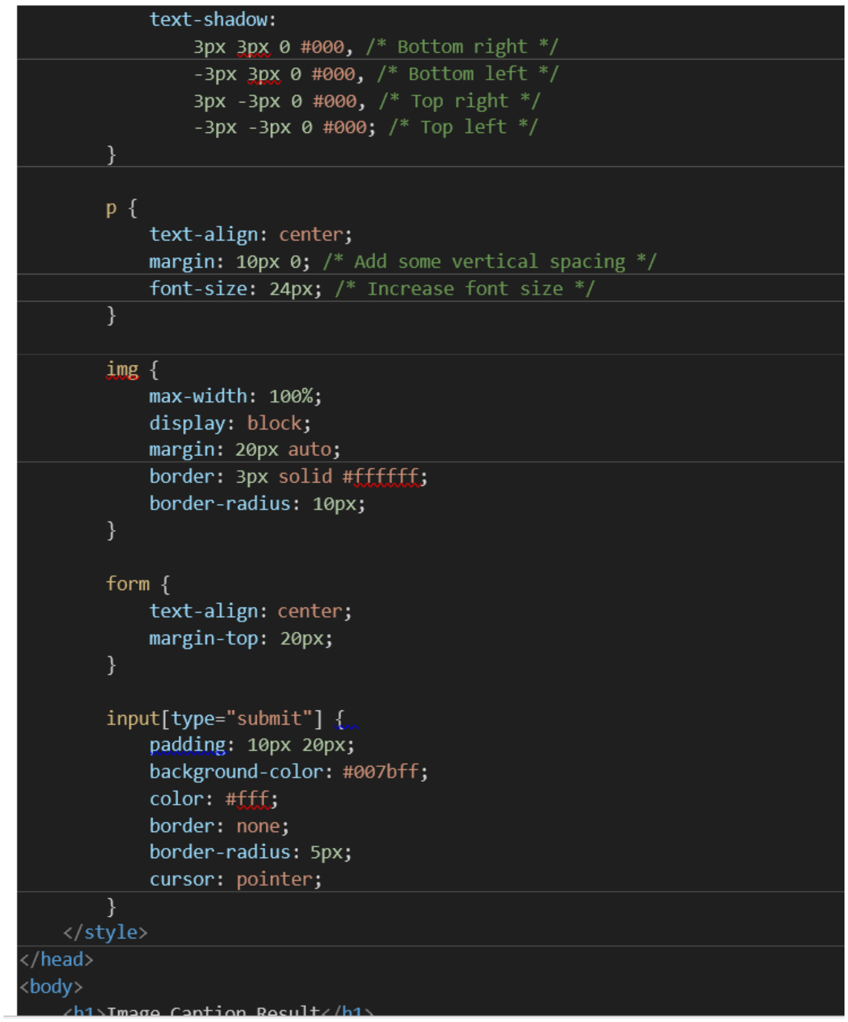
```
<!DOCTYPE html>

<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 */
        }
```



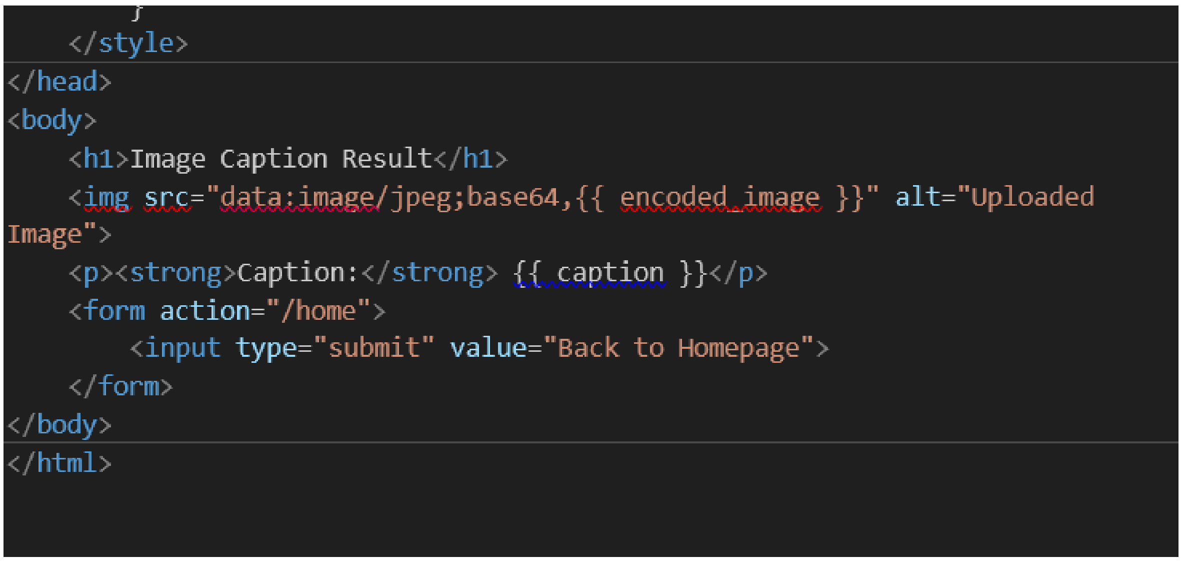
```
        .container {
            max-width: 600px;
            margin: 100px auto; /* Adjust the margin to center the container
*/
            padding: 20px;
            background-color: rgba(255, 255, 255, 0.8); /* Add some opacity to
the container */
            box-shadow: 0 0 10px rgba(0, 0,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



```
<!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 */
```

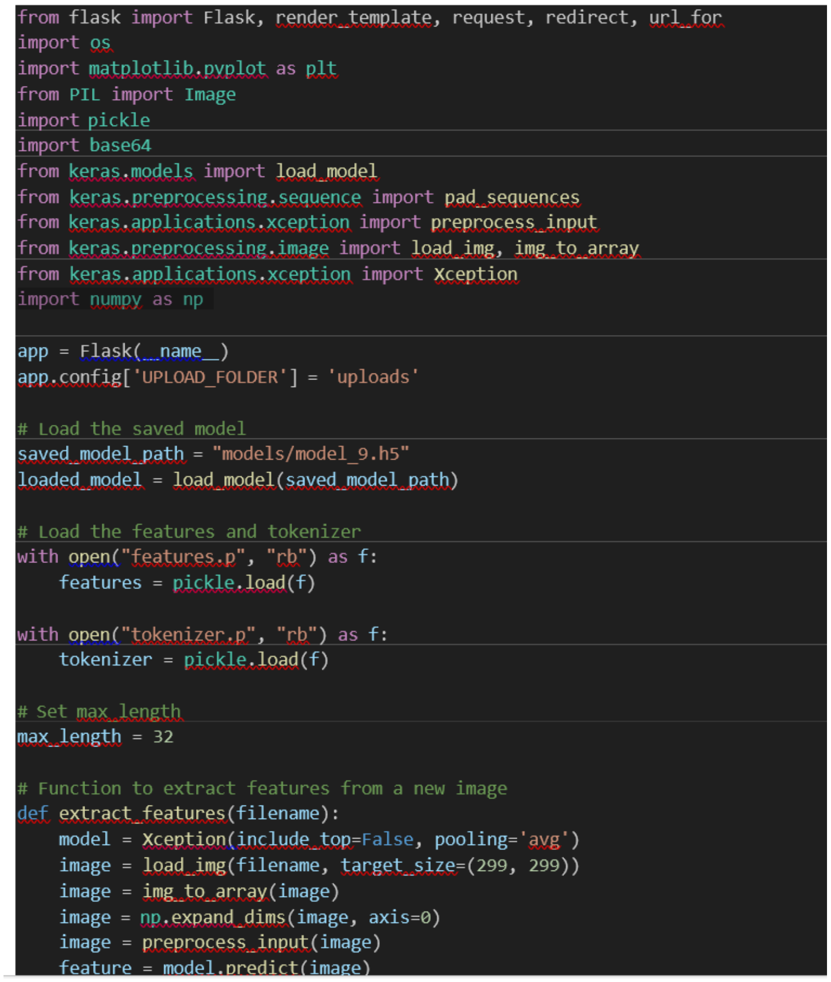


```
            text-shadow:
                3px 3px 0 #000, /* Bottom right */
                -3px 3px 0 #000, /* Bottom left */
                3px -3px 0 #000, /* Top right */
                -3px -3px 0 #000; /* Top left */
        }

        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>
```

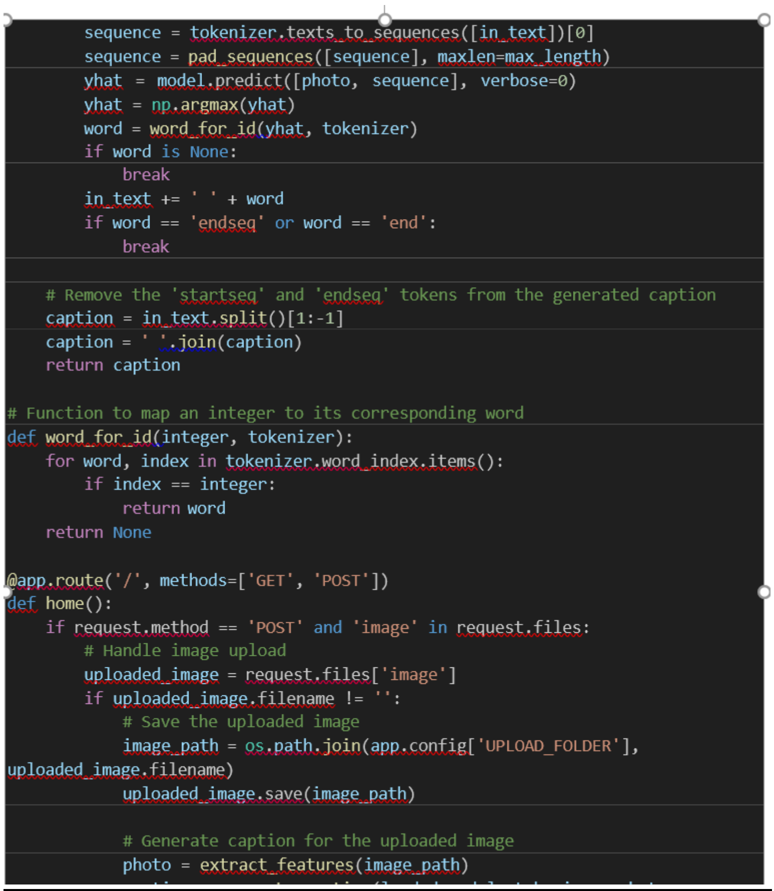


```
        }
    </style>
</head>
<body>
    <h1>Image Caption Result</h1>
    <img src="data:image/jpeg;base64,{{ encoded_image }}" alt="Uploaded
Image">
    <p><strong>Caption:</strong> {{ caption }}</p>
    <form action="/home">
        <input type="submit" value="Back to Homepage">
    </form>
</body>
</html>
```

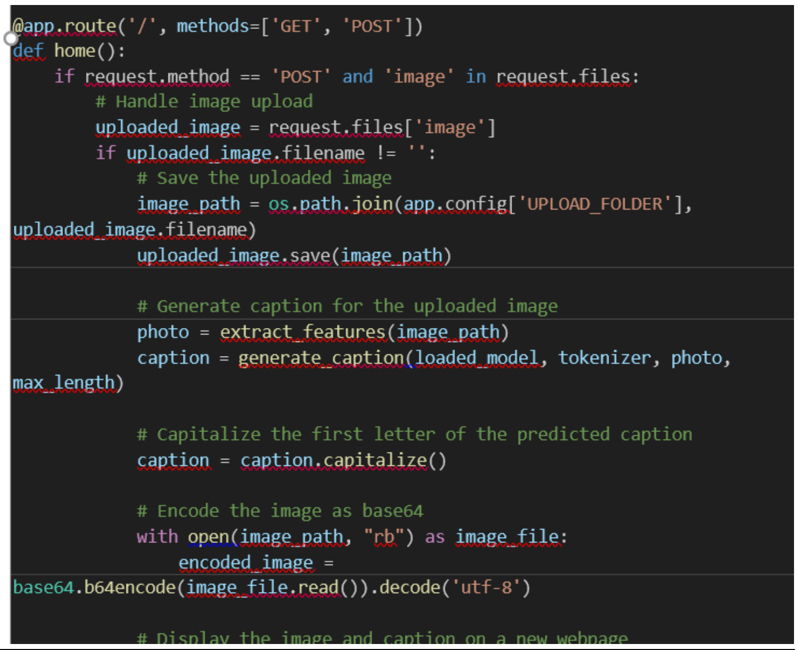
PYTHON FLASK FILE SCREEN SHOTS



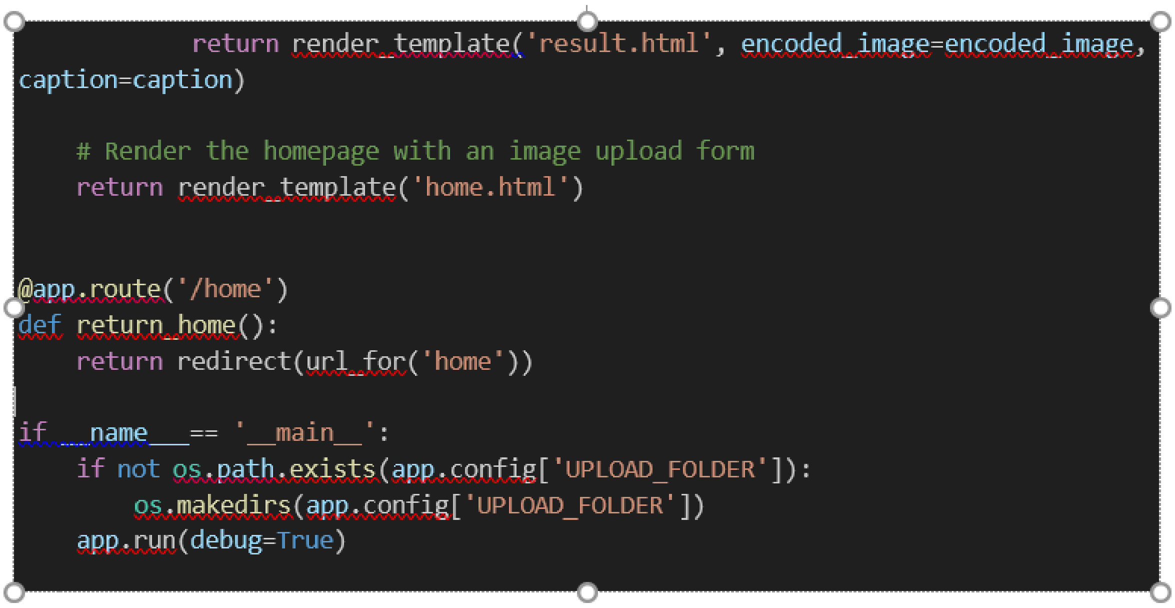
```python
from flask import Flask, render_template, request, redirect, url_for
import os
import matplotlib.pyplot as plt
from PIL import Image
import pickle
import base64
from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import 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)
```



```
        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)
```



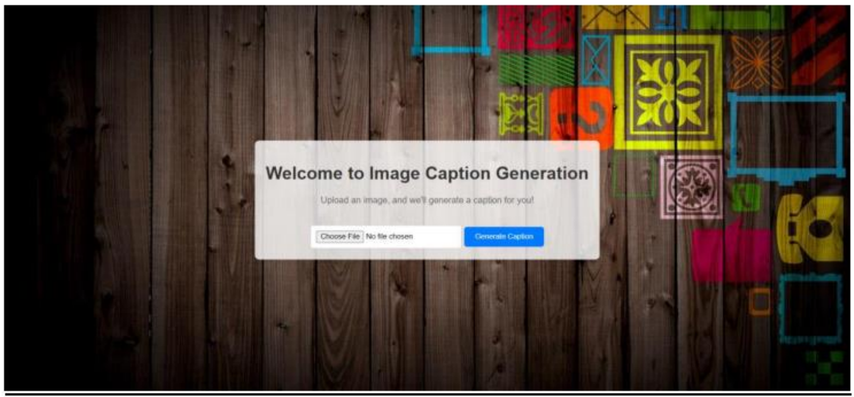
```
@app.route('/', methods=['GET', 'POST'])
def home():
    if request.method == 'POST' and 'image' in request.files:
        # Handle image upload
        uploaded_image = request.files['image']
        if uploaded_image.filename != '':
            # Save the uploaded image
            image_path =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

RESULTS OF FLASK WEB APP

Webapp Output Screen shots:



Welcome to Image Caption Generation

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

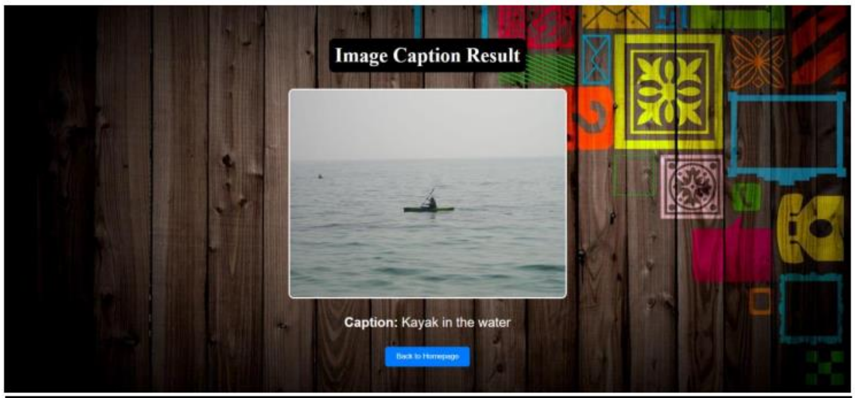


Image Caption Result

Caption: Kayak in the water

ADVANTAGES AND DISADVANTAGES

Advantages:

1. **Helps Everyone Understand Pictures**: It makes pictures easier to understand for people who can't see well by adding descriptions they can hear.
2. **Makes Pictures Easier to Find:** By adding captions, it helps search engines find pictures, so you can look them up more easily.
3. **Improves How We Use Websites:** Captions make it easier to use websites, especially in noisy places or for people who prefer reading over looking at pictures.
4. **Gives More Information:** It adds extra details to pictures, making them clearer or explaining things you might miss.

Disadvantages:

1. **Sometimes Gets It Wrong:** It might not always describe pictures accurately, especially if the picture is complicated or has a lot of details.
2. **Might Miss the Meaning:** It might describe a picture correctly but not understand what it really means or how it feels.
3. **Can Show Biases:** Sometimes, it might describe pictures in a way that's unfair to certain groups of people, like saying things that aren’t true because of prejudices in the data it learned from.
4. **Needs a Lot of Computer Power:** Making these descriptions needs a lot of computer power, so not everyone can do it easily.

CONCLUSION

Image caption generation stands as a pivotal advancement, revolutionizing how we interact with visual content. Its foremost advantage lies in fostering inclusivity, bridging accessibility gaps by providing descriptions for those with visual impairments, thereby ensuring a more inclusive online experience. Additionally, it enhances searchability and user experience by supplementing images with text, aiding search engines in indexing visual content and catering to diverse user preferences. However, challenges persist in accuracy and bias, where complexities in images can lead to inaccurate descriptions, while inherited biases from training data may perpetuate stereotypes or misconceptions. As this technology evolves, continual strides in refining AI models and data curation remain imperative, promising improved accuracy, reduced biases, and broader accessibility for this transformative capability.

FUTURE SCOPE

The future of image caption generation holds immense promise, poised to witness profound advancements driven by evolving AI capabilities and data refinements. Anticipated developments involve enhanced contextual understanding, enabling models to generate captions that grasp deeper nuances, emotions, and diverse cultural contexts within images. Further strides will likely focus on refining accuracy through sophisticated neural networks and multimodal learning, enabling these models to decipher intricate visuals with greater precision. Moreover, strides toward mitigating biases and ensuring ethical AI will play a pivotal role, fostering more inclusive and fair representations in image descriptions. As this field progresses, collaborations between AI, psychology, and linguistics may burgeon, fostering interdisciplinary approaches to refine these models, thus unlocking a future where image captioning becomes more accurate, culturally attuned, and seamlessly integrated into various aspects of everyday life.

APPENDIX

GITHUB LINK -

https://github.com/smartinternz02/SI-GuidedProject-612985-1698850923.git