Enhancing Image Captioning using Deep Learning: A Comprehensive Approach

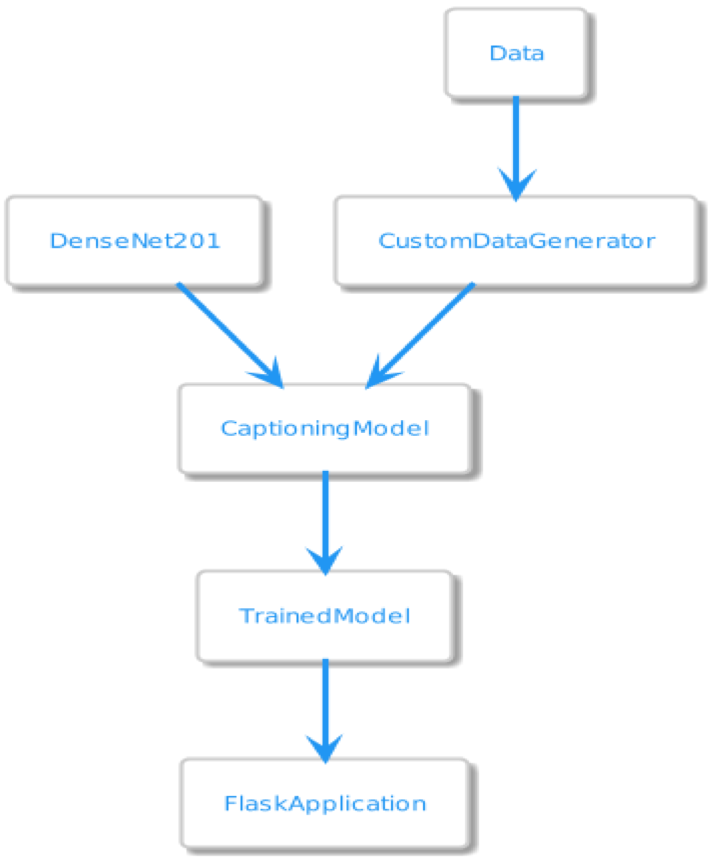
Introduction:

In today's digital landscape, the seamless integration of visual and textual information has become a cornerstone for various applications, from content creation to accessibility tools. At the heart of this integration lies the fascinating field of image captioning, where machines are endowed with the ability to not just "see" images but also articulate detailed, human-like descriptions for them.

Our project embarks on a journey to push the boundaries of image captioning, aiming to create a more nuanced and contextually rich understanding of visual content. Through the fusion of DenseNet201, a robust convolutional neural network architecture, with a carefully crafted recurrent neural network design, we endeavour to capture intricate relationships between pixels and semantics, ultimately elevating the quality of generated captions.

This introduction provides a glimpse into the profound implications of image captioning in our technology-driven society. By harnessing the power of deep learning, we aspire to contribute to the evolution of intelligent systems capable of not just recognizing images but comprehending and narrating their essence

Technical Architecture:



Our project Folder Structure Contains:

Our project boasts a well-organized folder structure to ensure clarity and ease of navigation. Here's an overview of the key components within the project directory:

1.Saves:

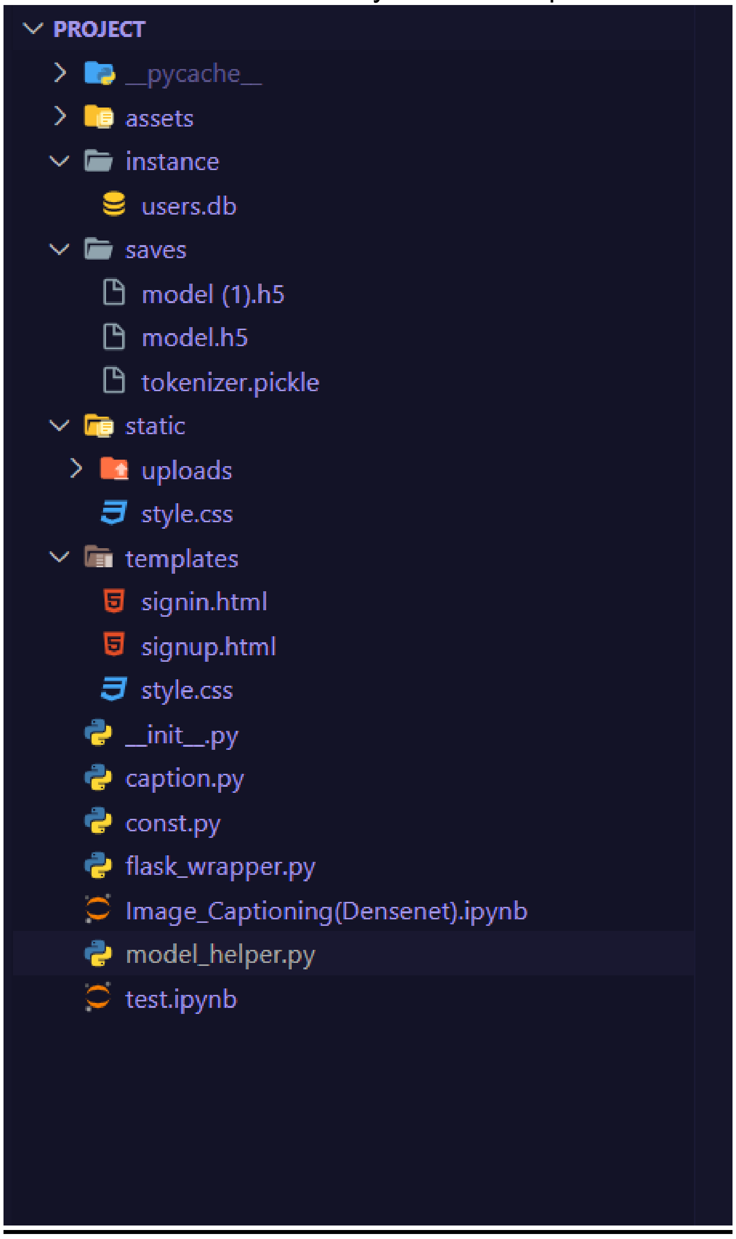
- Model Weights: This directory houses the saved model weights, specifically the model.h5 file, which encapsulates the knowledge acquired during training.
- Tokenizer: The folder contains the tokenizer.pickle file, preserving the vocabulary mapping established during the tokenization process.

2.Static:

- Uploads: Within this folder, the application stores user-uploaded images. It ensures a streamlined process for managing and accessing the images.

3.Templates:

- This directory encapsulates HTML templates essential for the user interface. Notable files include index.html, signin.html, and signup.html, each contributing to the seamless functionality and user experience of the web application.



```
PROJECT
  > __pycache__
  > assets
  v instance
      users.db
  v saves
      model (1).h5
      model.h5
      tokenizer.pickle
  v static
    > uploads
      style.css
  v templates
      signin.html
      signup.html
      style.css
    __init__.py
    caption.py
    const.py
    flask_wrapper.py
    Image_Captioning(Densenet).ipynb
    model_helper.py
    test.ipynb
```

Pre-requisites:

1. Development Environment:
Install Visual Studio Code for a comprehensive code editing and debugging experience. Set up Python extensions for seamless integration.

2. Kaggle Account:
Sign up for a Kaggle account to access datasets, kernels, and competitions directly from Kaggle.

3. Anaconda Distribution (Optional):
Download and install Anaconda for Python and essential libraries. Although optional, Anaconda provides convenient environment management.

4. Python Libraries:
Install essential Python libraries using Anaconda or pip within your Python environment.
   - TensorFlow and Keras for deep learning.
   - PIL for image processing.
   - NumPy, Pandas, Matplotlib for data manipulation and visualization.
   - Flask for web development
   - SQLite for Database

5. Kaggle API:
Install the Kaggle API to access datasets directly from Kaggle. Follow Kaggle's API setup instructions.

6. Dataset and Models:
Identify or download the image captioning dataset from Kaggle or other reputable sources.
Download or access pre-trained models like DenseNet201, possibly using TensorFlow Hub

7. Visual Studio Code:
Utilize Visual Studio Code for coding and project management. Customize the environment as per your preferences.

Milestone 1: Collection of Data

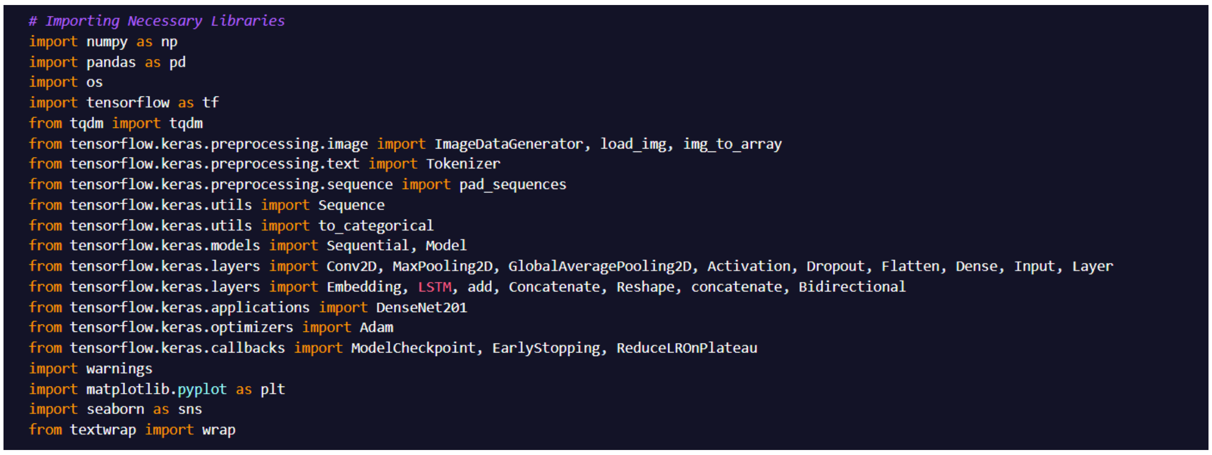
The chosen dataset for this image captioning project is the Flickr8k Dataset, a widely recognized and utilized collection of images sourced from the Flickr platform. Comprising approximately 8,000 images, the dataset provides a diverse and comprehensive set of visual content, covering a wide range of subjects, scenes, and contexts. Each image in the dataset is accompanied by multiple descriptive captions, contributing to the richness and variability of linguistic expressions associated with the same visual content. The annotations exhibit a conversational and expressive style, reflecting how individuals naturally describe scenes.

The dataset's credibility and reliability are enhanced by its extensive use in the research community. As part of the preprocessing steps, captions underwent text normalization, and images were resized and normalized to ensure consistency in input features for the image captioning model. Overall, the Flickr8k Dataset serves as a valuable resource for training models to generate human-like and contextually relevant captions for diverse visual content.

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

Milestone 2: Image Pre-processing and Model Building

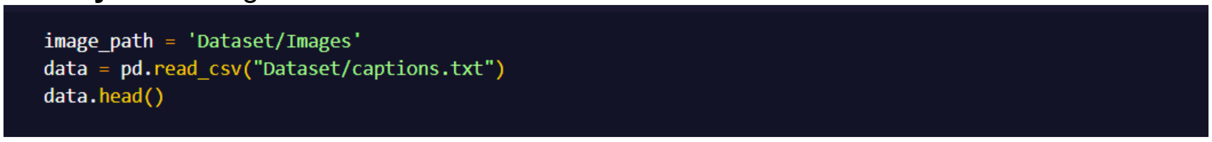
Activity-1: Importing the required libraries



```
# Importing Necessary Libraries
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import Sequence
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Activation, Dropout, Flatten, Dense, Input, Layer
from tensorflow.keras.layers import Embedding, LSTM, add, Concatenate, Reshape, concatenate, Bidirectional
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from textwrap import wrap
```

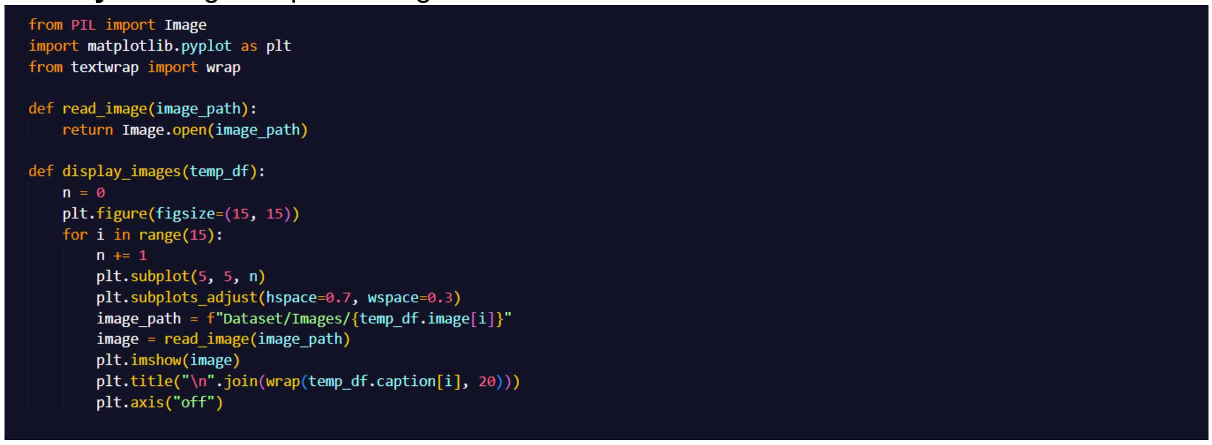
.

Activity-2: Loading Dataset



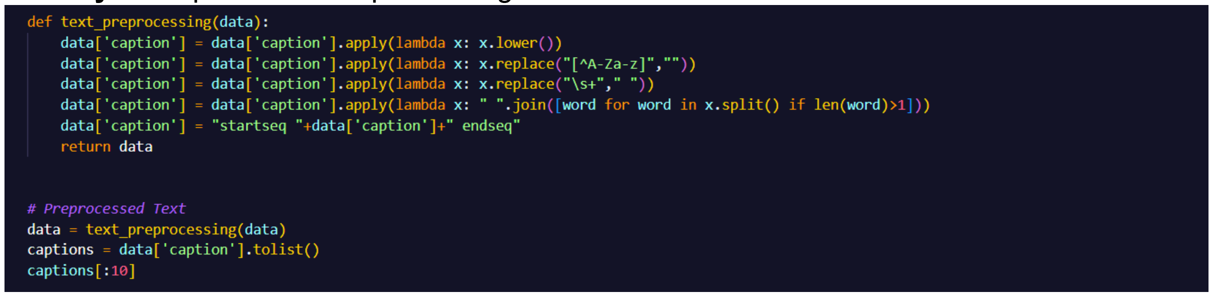
```
image_path = 'Dataset/Images'
data = pd.read_csv("Dataset/captions.txt")
data.head()
```

Activity-3: Image Preprocessing



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

def read_image(image_path):
    return Image.open(image_path)

def display_images(temp_df):
    n = 0
    plt.figure(figsize=(15, 15))
    for i in range(15):
        n += 1
        plt.subplot(5, 5, n)
        plt.subplots_adjust(hspace=0.7, wspace=0.3)
        image_path = f"Dataset/Images/{temp_df.image[i]}"
        image = read_image(image_path)
        plt.imshow(image)
        plt.title("\n".join(wrap(temp_df.caption[i], 20)))
        plt.axis("off")
```

Activity-4: Caption Text Preprocessing



```
def text_preprocessing(data):
    data['caption'] = data['caption'].apply(lambda x: x.lower())
    data['caption'] = data['caption'].apply(lambda x: x.replace("[^A-Za-z]",""))
    data['caption'] = data['caption'].apply(lambda x: x.replace("\s+"," "))
    data['caption'] = data['caption'].apply(lambda x: " ".join([word for word in x.split() if len(word)>1]))
    data['caption'] = "startseq "+data['caption']+" endseq"
    return data


# Preprocessed Text
data = text_preprocessing(data)
captions = data['caption'].tolist()
captions[:10]
```

Activity-5: Tokenization and Encoder Representation

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(captions)
vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(caption.split()) for caption in captions)

images = data['image'].unique().tolist()
nimages = len(images)

split_index = round(0.85*nimages)
train_images = images[:split_index]
val_images = images[split_index:]

train = data[data['image'].isin(train_images)]
test = data[data['image'].isin(val_images)]

train.reset_index(inplace=True,drop=True)
test.reset_index(inplace=True,drop=True)

tokenizer.texts_to_sequences([captions[1]])[0]
```

Activity-6: Image Feature Extraction

```
model = DenseNet201()
fe = Model(inputs=model.input, outputs=model.layers[-2].output)

img_size = 224
features = {}
for image in tqdm(data['image'].unique().tolist()):
    img = load_img(os.path.join(image_path,image),target_size=(img_size,img_size))
    img = img_to_array(img)
    img = img/255.
    img = np.expand_dims(img,axis=0)
    feature = fe.predict(img, verbose=0)
    features[image] = feature
```

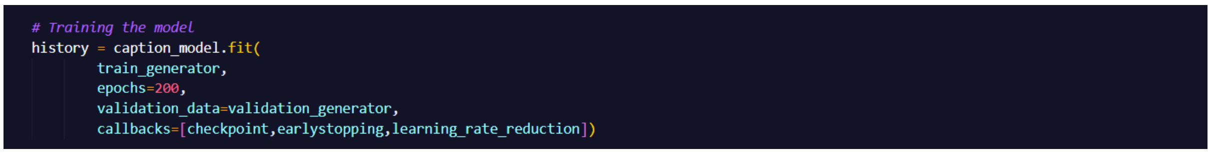
```
100%|██████████| 8091/8091 [10:40<00:00, 12.63it/s]
```

Activity-7: Modelling

```
input1 = Input(shape=(1920,))
input2 = Input(shape=(max_length,))

img_features = Dense(256, activation='relu')(input1)
img_features_reshaped = Reshape((1, 256), input_shape=(256,))(img_features)

sentence_features = Embedding(vocab_size, 256, mask_zero=False)(input2)
merged = concatenate([img_features_reshaped,sentence_features],axis=1)
sentence_features = LSTM(256)(merged)
x = Dropout(0.5)(sentence_features)
x = add([x, img_features])
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(vocab_size, activation='softmax')(x)

caption_model = Model(inputs=[input1,input2], outputs=output)
caption_model.compile(loss='categorical_crossentropy',optimizer='adam')
```

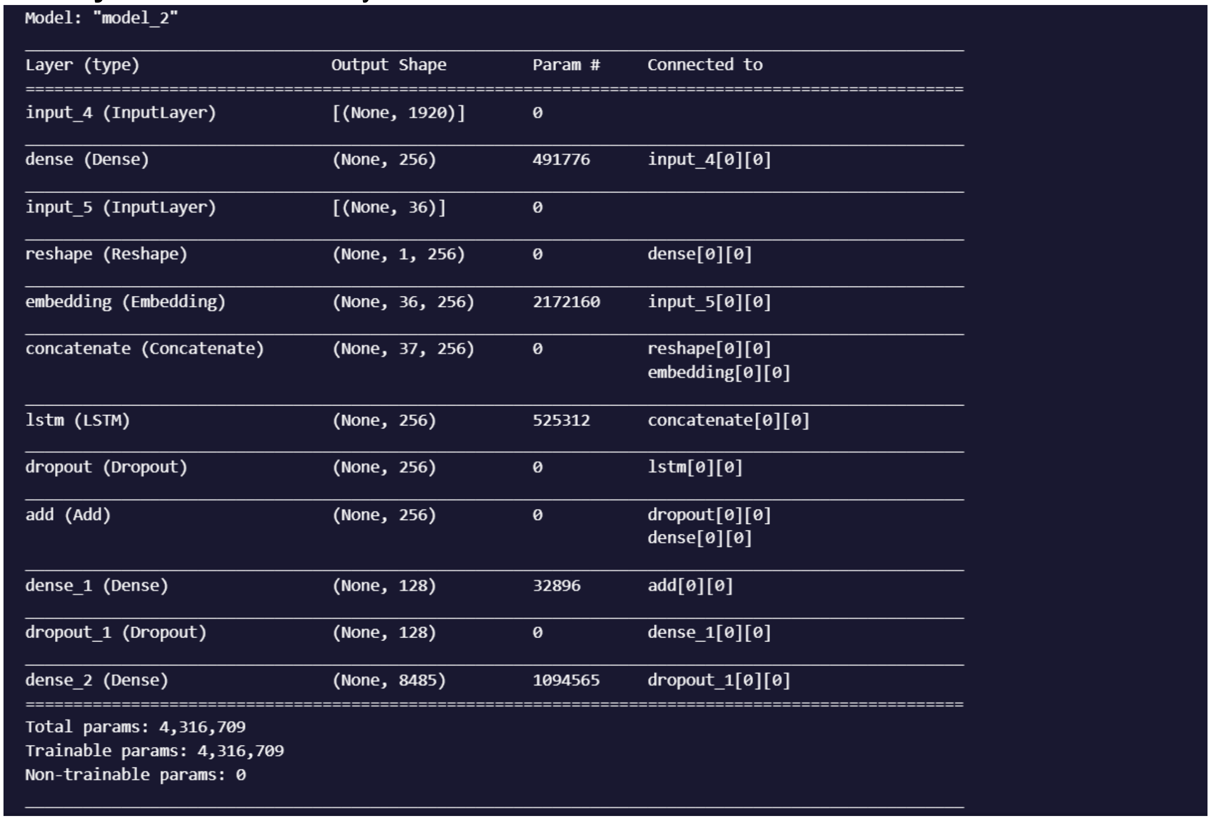
Activity-8: Training the model

```
train_generator = CustomDataGenerator(df=train,X_col='image',y_col='caption',batch_size=64,directory=image_path,
                                      tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)

validation_generator = CustomDataGenerator(df=test,X_col='image',y_col='caption',batch_size=64,directory=image_path,
                                      tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)
model_name = "model.h5"
checkpoint = ModelCheckpoint(model_name,
                            monitor="val_loss",
                            mode="min",
                            save_best_only = True,
                            verbose=1)

earlystopping = EarlyStopping(monitor='val_loss',min_delta = 0, patience = 5, verbose = 1, restore_best_weights=True)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                            patience=3,
                                            verbose=1,
                                            factor=0.2,
                                            min_lr=0.00000001)
```



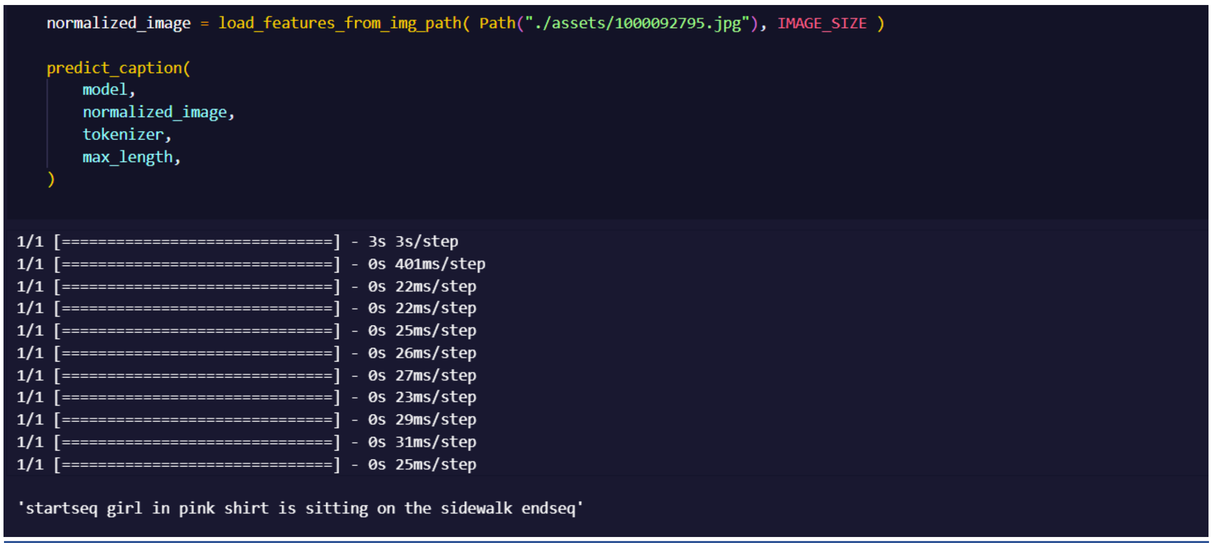
```python
# Training the model
history = caption_model.fit(
        train_generator,
        epochs=200,
        validation_data=validation_generator,
        callbacks=[checkpoint,earlystopping,learning_rate_reduction])
```

Activity-9: Model Summary



```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 1920)]       0

 dense (Dense)                  (None, 256)          491776      input_4[0][0]

 input_5 (InputLayer)           [(None, 36)]         0

 reshape (Reshape)              (None, 1, 256)       0           dense[0][0]

 embedding (Embedding)          (None, 36, 256)      2172160     input_5[0][0]

 concatenate (Concatenate)      (None, 37, 256)      0           reshape[0][0]
                                                                 embedding[0][0]

 lstm (LSTM)                    (None, 256)          525312      concatenate[0][0]

 dropout (Dropout)              (None, 256)          0           lstm[0][0]

 add (Add)                      (None, 256)          0           dropout[0][0]
                                                                 dense[0][0]

 dense_1 (Dense)                (None, 128)          32896       add[0][0]

 dropout_1 (Dropout)            (None, 128)          0           dense_1[0][0]

 dense_2 (Dense)                (None, 8485)         1094565     dropout_1[0][0]

==================================================================================================
Total params: 4,316,709
Trainable params: 4,316,709
Non-trainable params: 0
__________________________________________________________________________________________________
```

Milestone 3: Testing the Model:



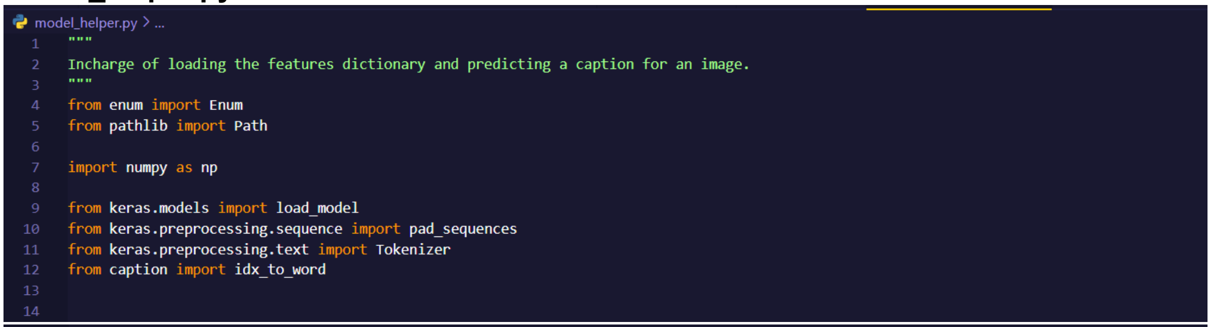
```python
normalized_image = load_features_from_img_path( Path("./assets/1000092795.jpg"), IMAGE_SIZE )

predict_caption(
    model,
    normalized_image,
    tokenizer,
    max_length,
)
```

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 0s 401ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 25ms/step

'startseq girl in pink shirt is sitting on the sidewalk endseq'
```

Milestone 4: Loading Pre Trained Model

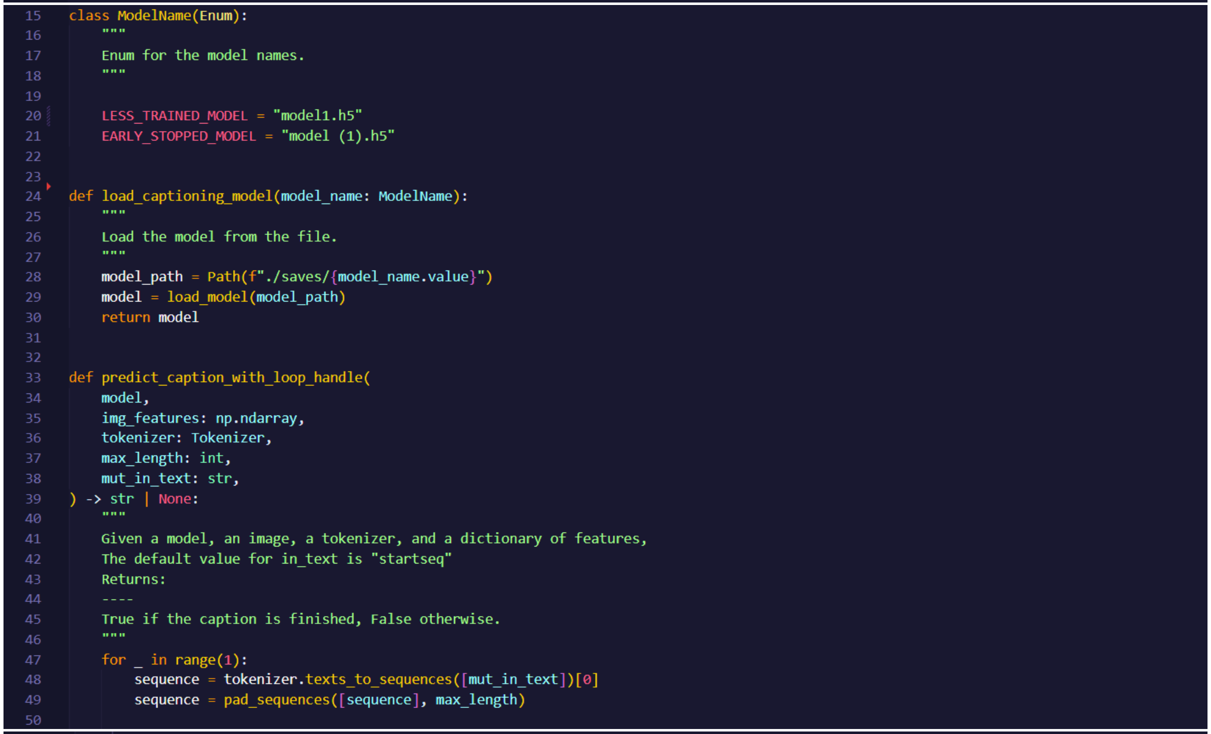
After successfully training the image captioning model on the Flickr8k dataset, the trained model is saved to the 'saves' directory within the project folder as 'model.h5'. This pre-trained model serves as a crucial component in our project, providing a foundation for accurate image captioning.

Model_helper.py:

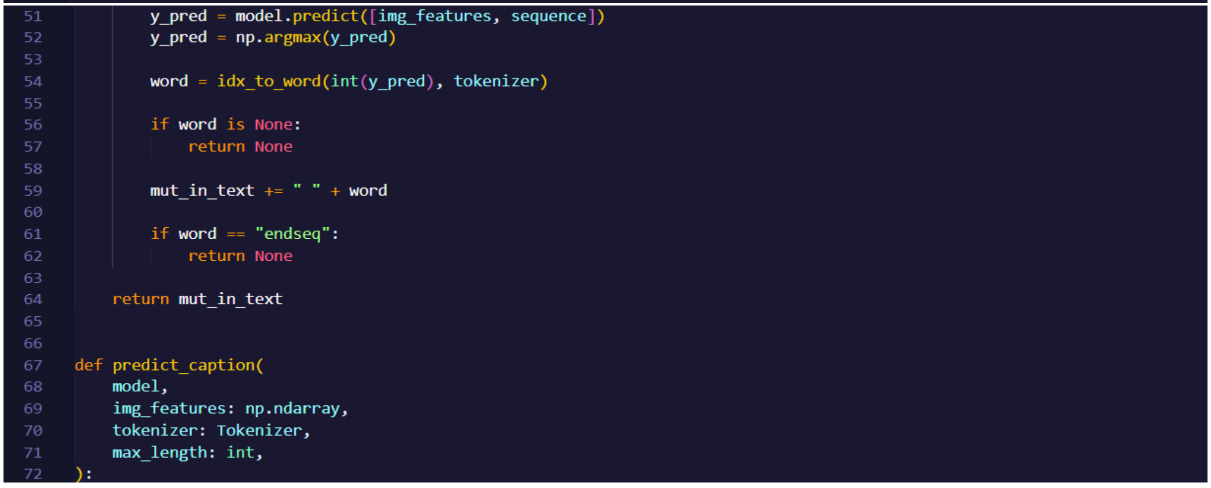


model_helper.py > ...

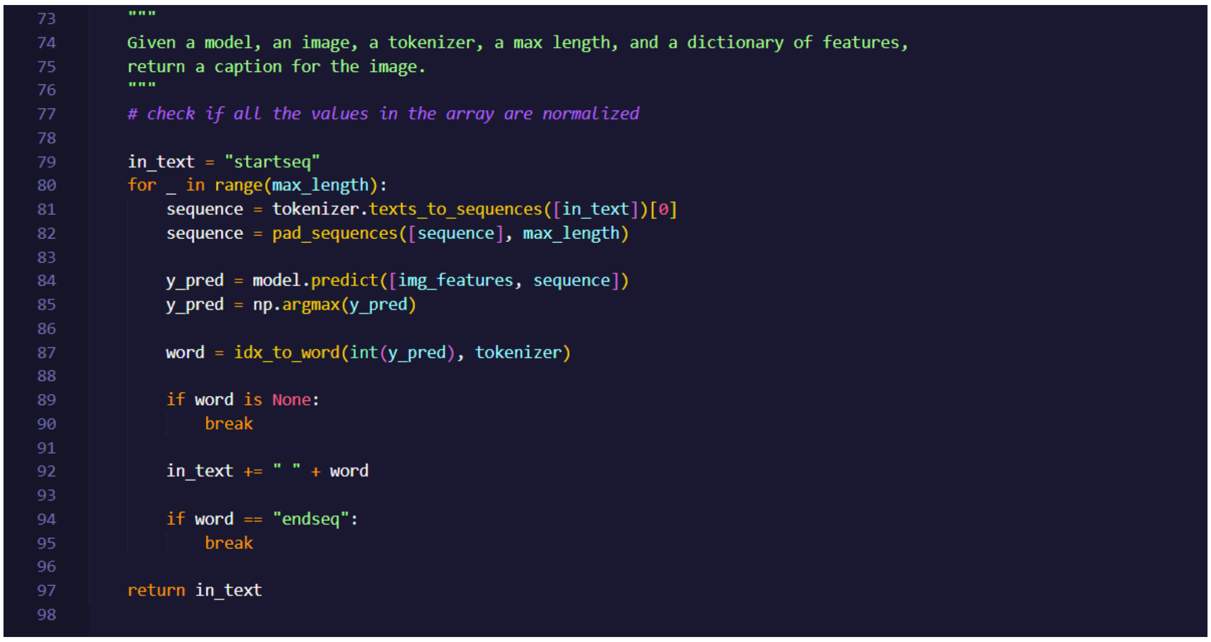
```python
"""
Incharge of loading the features dictionary and predicting a caption for an image.
"""
from enum import Enum
from pathlib import Path

import numpy as np

from keras.models import load_model
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from caption import idx_to_word


```



```python
class ModelName(Enum):
    """
    Enum for the model names.
    """

    LESS_TRAINED_MODEL = "model1.h5"
    EARLY_STOPPED_MODEL = "model (1).h5"


def load_captioning_model(model_name: ModelName):
    """
    Load the model from the file.
    """
    model_path = Path(f"./saves/{model_name.value}")
    model = load_model(model_path)
    return model


def predict_caption_with_loop_handle(
    model,
    img_features: np.ndarray,
    tokenizer: Tokenizer,
    max_length: int,
    mut_in_text: str,
) -> str | None:
    """
    Given a model, an image, a tokenizer, and a dictionary of features,
    The default value for in_text is "startseq"
    Returns:
    ----
    True if the caption is finished, False otherwise.
    """
    for _ in range(1):
        sequence = tokenizer.texts_to_sequences([mut_in_text])[0]
        sequence = pad_sequences([sequence], max_length)

```



```python
        y_pred = model.predict([img_features, sequence])
        y_pred = np.argmax(y_pred)

        word = idx_to_word(int(y_pred), tokenizer)

        if word is None:
            return None

        mut_in_text += " " + word

        if word == "endseq":
            return None

    return mut_in_text


def predict_caption(
    model,
    img_features: np.ndarray,
    tokenizer: Tokenizer,
    max_length: int,
):
```



```python
    """
    Given a model, an image, a tokenizer, a max length, and a dictionary of features,
    return a caption for the image.
    """
    # check if all the values in the array are normalized

    in_text = "startseq"
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], max_length)

        y_pred = model.predict([img_features, sequence])
        y_pred = np.argmax(y_pred)

        word = idx_to_word(int(y_pred), tokenizer)

        if word is None:
            break

        in_text += " " + word

        if word == "endseq":
            break

    return in_text
```

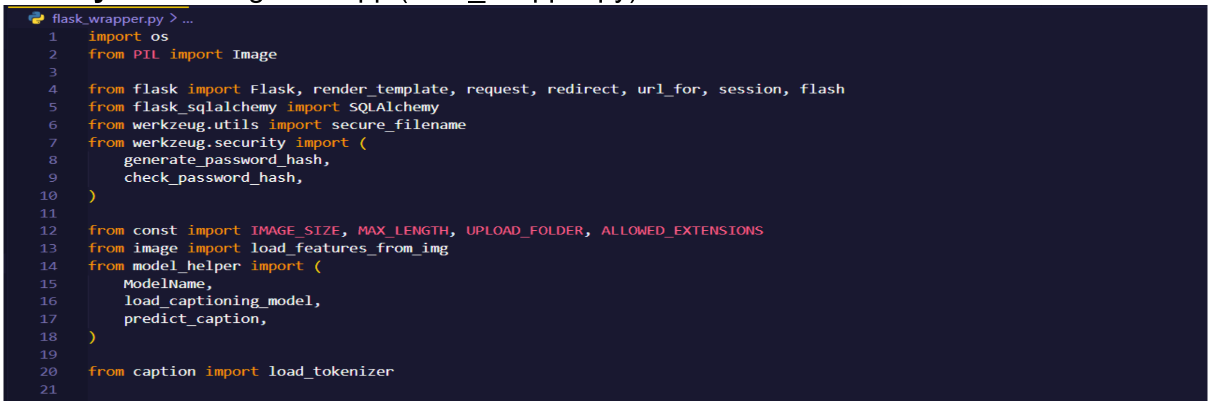
The model_helper.py module serves as a critical component in our project, playing a pivotal role in managing the image captioning model. One of its key functionalities is to load the pre-trained model ('model.h5') using Keras, ensuring seamless integration and reusability of the trained model for generating captions. Additionally, the module includes functions for predicting captions based on input image features, providing a convenient interface for leveraging the model's capabilities.

Another crucial aspect is the loading of the tokenizer used during the model training phase, facilitating the conversion of text data into sequences compatible with the model. In summary, model_helper.py encapsulates the essential operations for working with the image captioning model, contributing to the overall efficiency and functionality of our project.

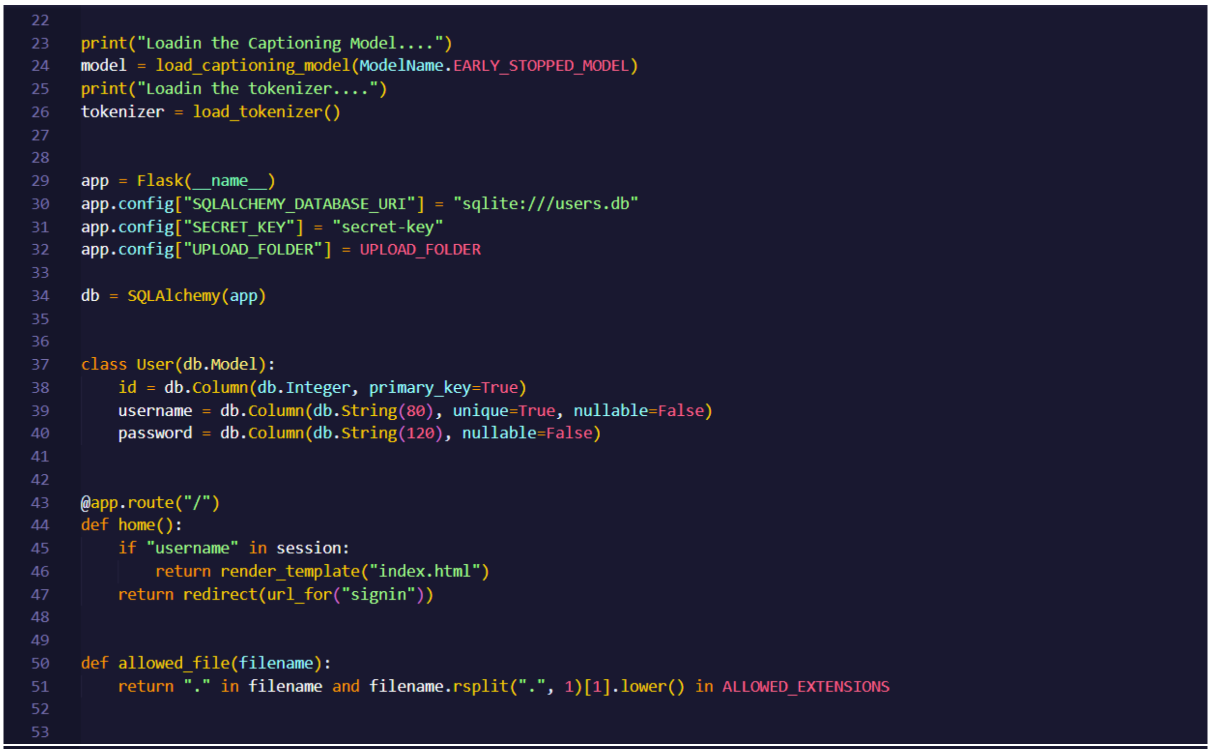
Milestone 5: Application Building

In the Application Building milestone, we focus on creating a user-friendly interface for our image captioning model. Leveraging the Flask web framework, we establish a robust foundation for building a web application. The flask_wrapper.py file serves as the backbone of this development, encapsulating the integration of the pre-trained image captioning model with the Flask application.

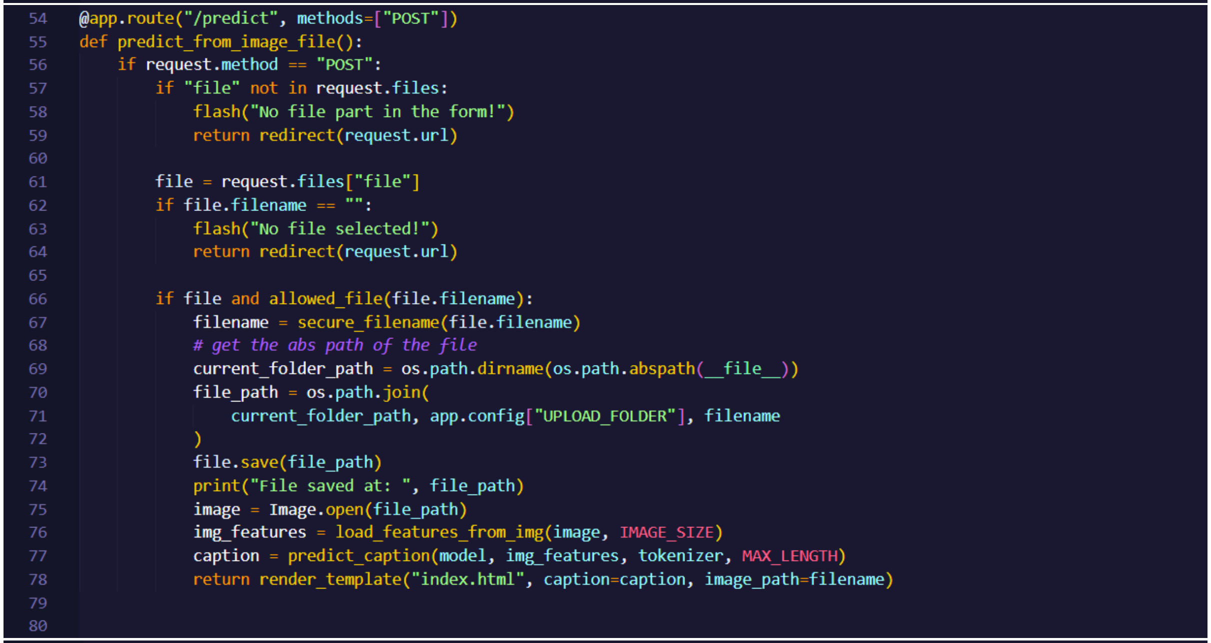
Activity-1: Creating flask app (flask_wrapper.py)



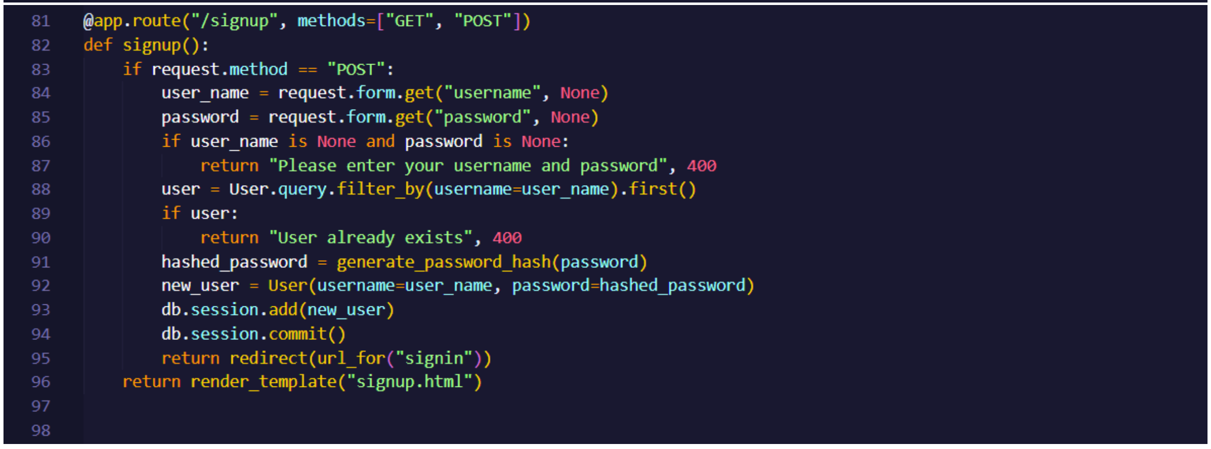
```python
import os
from PIL import Image

from flask import Flask, render_template, request, redirect, url_for, session, flash
from flask_sqlalchemy import SQLAlchemy
from werkzeug.utils import secure_filename
from werkzeug.security import (
    generate_password_hash,
    check_password_hash,
)

from const import IMAGE_SIZE, MAX_LENGTH, UPLOAD_FOLDER, ALLOWED_EXTENSIONS
from image import load_features_from_img
from model_helper import (
    ModelName,
    load_captioning_model,
    predict_caption,
)

from caption import load_tokenizer
```



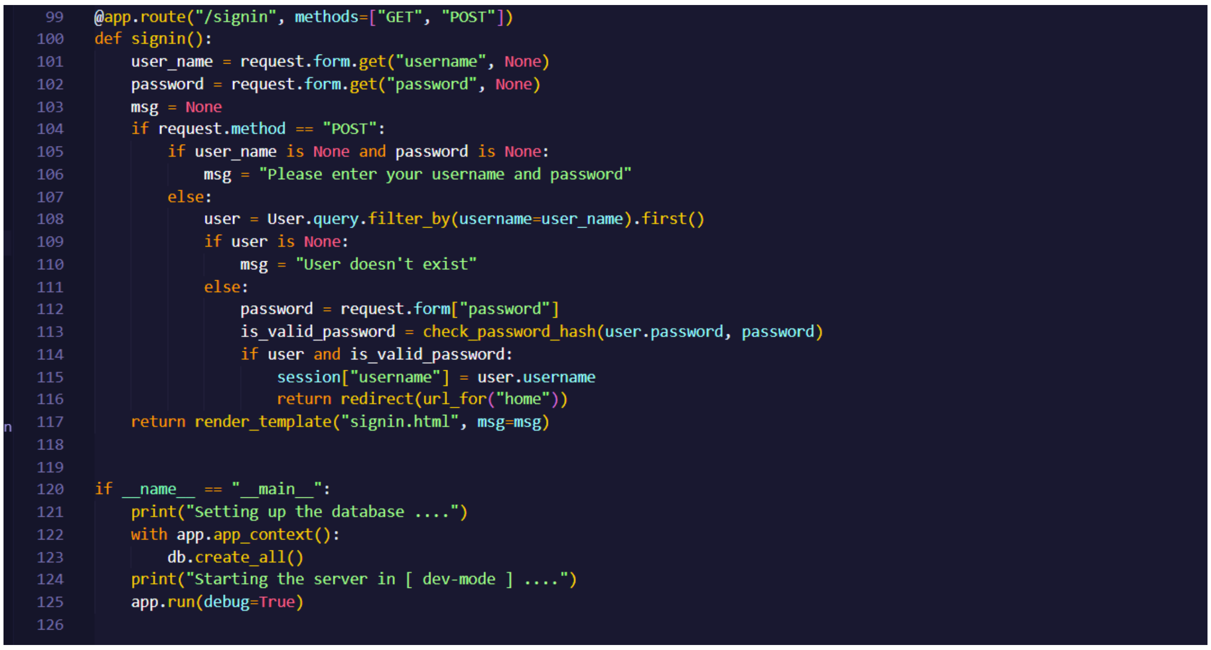
```python

print("Loadin the Captioning Model....")
model = load_captioning_model(ModelName.EARLY_STOPPED_MODEL)
print("Loadin the tokenizer....")
tokenizer = load_tokenizer()


app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///users.db"
app.config["SECRET_KEY"] = "secret-key"
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

db = SQLAlchemy(app)


class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)


@app.route("/")
def home():
    if "username" in session:
        return render_template("index.html")
    return redirect(url_for("signin"))


def allowed_file(filename):
    return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS


```



```python
@app.route("/predict", methods=["POST"])
def predict_from_image_file():
    if request.method == "POST":
        if "file" not in request.files:
            flash("No file part in the form!")
            return redirect(request.url)

        file = request.files["file"]
        if file.filename == "":
            flash("No file selected!")
            return redirect(request.url)

        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            # get the abs path of the file
            current_folder_path = os.path.dirname(os.path.abspath(__file__))
            file_path = os.path.join(
                current_folder_path, app.config["UPLOAD_FOLDER"], filename
            )
            file.save(file_path)
            print("File saved at: ", file_path)
            image = Image.open(file_path)
            img_features = load_features_from_img(image, IMAGE_SIZE)
            caption = predict_caption(model, img_features, tokenizer, MAX_LENGTH)
            return render_template("index.html", caption=caption, image_path=filename)


```



```python
@app.route("/signup", methods=["GET", "POST"])
def signup():
    if request.method == "POST":
        user_name = request.form.get("username", None)
        password = request.form.get("password", None)
        if user_name is None and password is None:
            return "Please enter your username and password", 400
        user = User.query.filter_by(username=user_name).first()
        if user:
            return "User already exists", 400
        hashed_password = generate_password_hash(password)
        new_user = User(username=user_name, password=hashed_password)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for("signin"))
    return render_template("signup.html")


```



```python
@app.route("/signin", methods=["GET", "POST"])
def signin():
    user_name = request.form.get("username", None)
    password = request.form.get("password", None)
    msg = None
    if request.method == "POST":
        if user_name is None and password is None:
            msg = "Please enter your username and password"
        else:
            user = User.query.filter_by(username=user_name).first()
            if user is None:
                msg = "User doesn't exist"
            else:
                password = request.form["password"]
                is_valid_password = check_password_hash(user.password, password)
                if user and is_valid_password:
                    session["username"] = user.username
                    return redirect(url_for("home"))
    return render_template("signin.html", msg=msg)


if __name__ == "__main__":
    print("Setting up the database ....")
    with app.app_context():
        db.create_all()
    print("Starting the server in [ dev-mode ] ....")
    app.run(debug=True)
```

The load_captioning_model function from model_helper.py is utilized to load the pre-trained image captioning model (saved as model.h5). This function ensures that the trained model is ready for making predictions.

The Flask application begins with the definition of routes. The / route renders the home page, prompting users to sign in. The /predict route handles the image captioning functionality. When a user uploads an image through the web interface, the application preprocesses the image and passes it to the loaded model for generating a descriptive caption.

The predict_caption function in model_helper.py takes care of the actual caption generation process. It uses the loaded model to predict a caption based on the uploaded image's features.

The application is also equipped with user authentication features. Users can sign up, sign in, and navigate through the different sections of the web interface. Passwords are hashed using the werkzeug.security library for security.

Furthermore, the application utilizes an SQLite database (users.db) to store user information, providing a persistent and secure storage solution for user accounts.

Activity-2: HTML Templates

In addition to the Python scripts, our project incorporates HTML templates to define the structure and layout of the web pages in our Flask application. These templates are stored in the "templates" folder and play a crucial role in creating a visually appealing and interactive user interface. The following HTML files are part of our project:

1.index.html:
This file represents the main landing page of our application. It includes elements such as the image upload form, prediction display area, and user navigation options.

The structure of this HTML file is designed to provide a clean and intuitive interface for users interacting with the image captioning functionality.

2.signin.html:
The sign-in page allows users to log in to their accounts. It features a form for entering the username and password, along with options for account creation and navigation. The HTML structure encourages a seamless and secure login experience.

3.Signup.html:
The sign-up page is responsible for user registration. It contains a form where new users can enter their desired username and password. The page emphasizes data validation and user-friendly feedback during the sign-up process, ensuring a smooth onboarding experience.

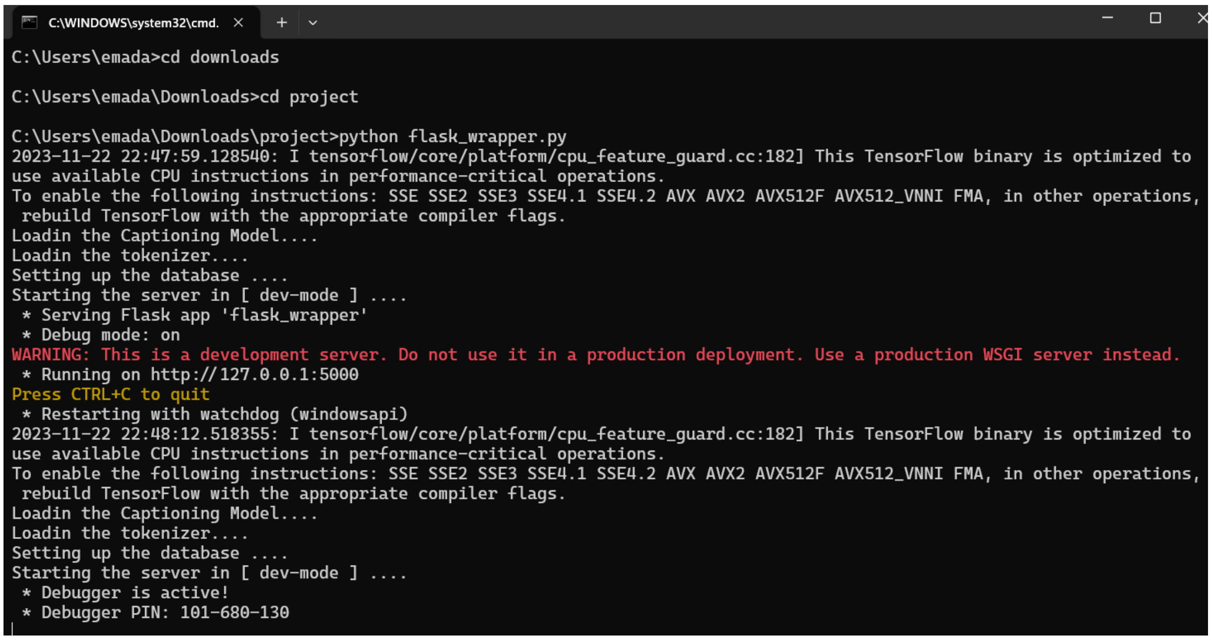
These HTML templates leverage the Flask templating engine, which allows for dynamic content generation and integration with the back-end Python scripts. By combining these templates with the Flask application logic, we achieve a well-integrated and responsive web application for image captioning.

Milestone 6: Result Visualization:

To seamlessly launch the web application, navigate to the project directory in the Visual Studio Code terminal. Execute the following command to initiate the application:

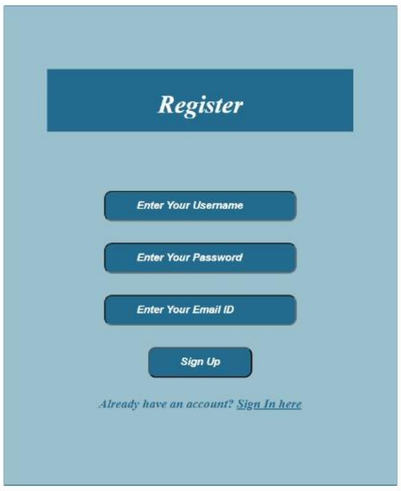
```
python flask_wrapper.py
```

This command will efficiently start the Flask web server, allowing us to interact with the image captioning application effortlessly.



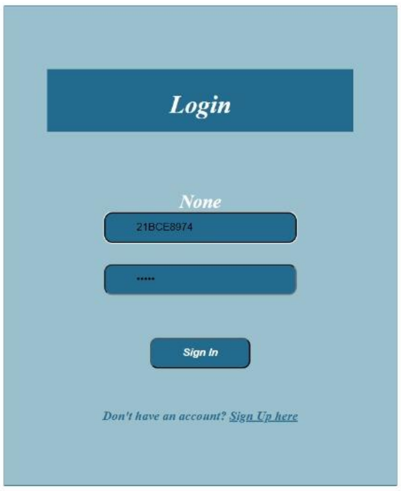
```
C:\Users\emada>cd downloads

C:\Users\emada\Downloads>cd project

C:\Users\emada\Downloads\project>python flask_wrapper.py
2023-11-22 22:47:59.128540: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations,
 rebuild TensorFlow with the appropriate compiler flags.
Loadin the Captioning Model....
Loadin the tokenizer....
Setting up the database ....
Starting the server in [ dev-mode ] ....
 * Serving Flask app 'flask_wrapper'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
2023-11-22 22:48:12.518355: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations,
 rebuild TensorFlow with the appropriate compiler flags.
Loadin the Captioning Model....
Loadin the tokenizer....
Setting up the database ....
Starting the server in [ dev-mode ] ....
 * Debugger is active!
 * Debugger PIN: 101-680-130
```

Here we should go through the link “localhost:5000” in our computer

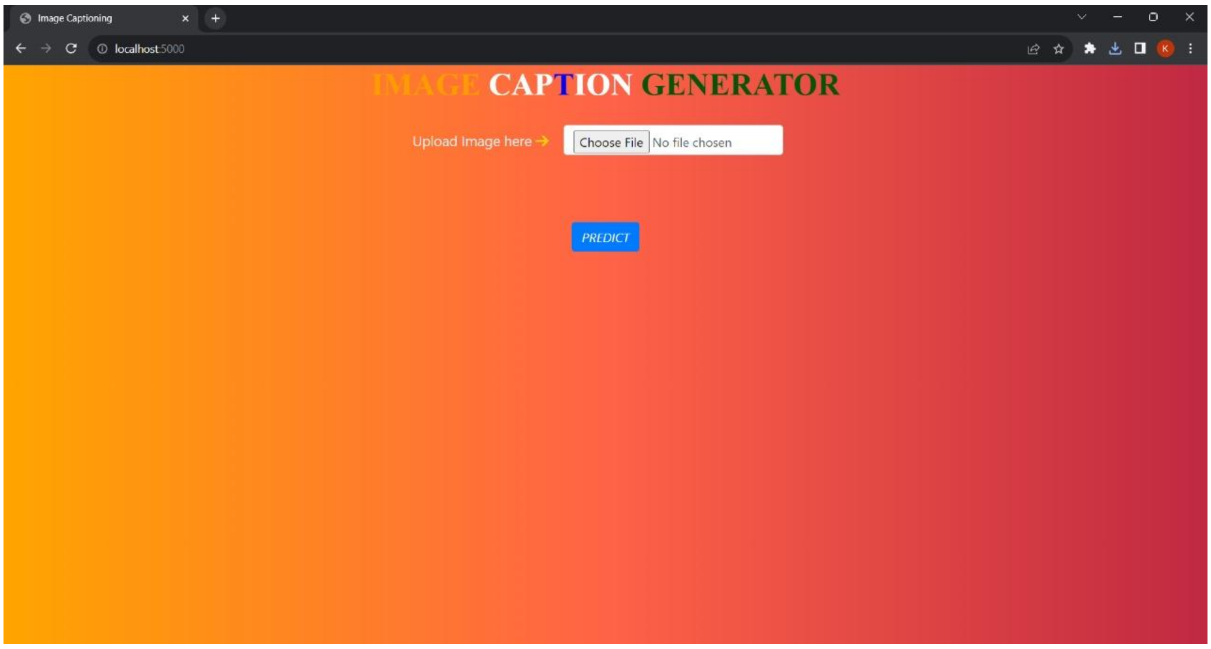


So first any user doesn’t have any account, Users can create their own accounts using their passwords and sign up

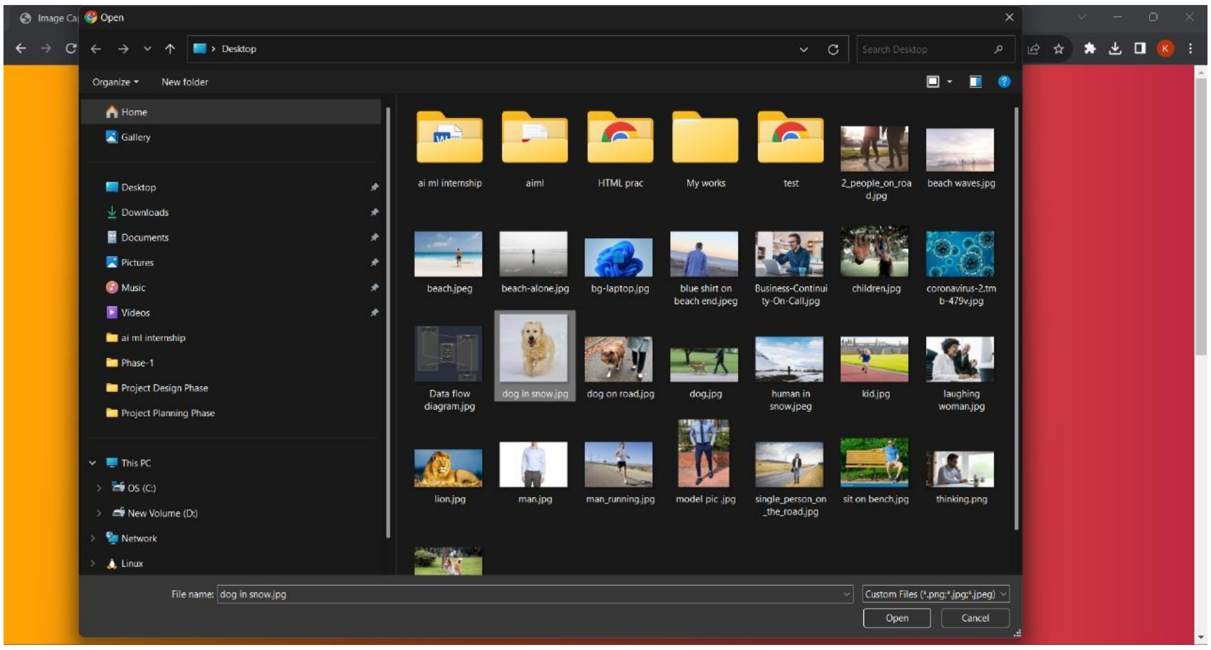
If the user already have an account then he can directly sign in by entering username and password



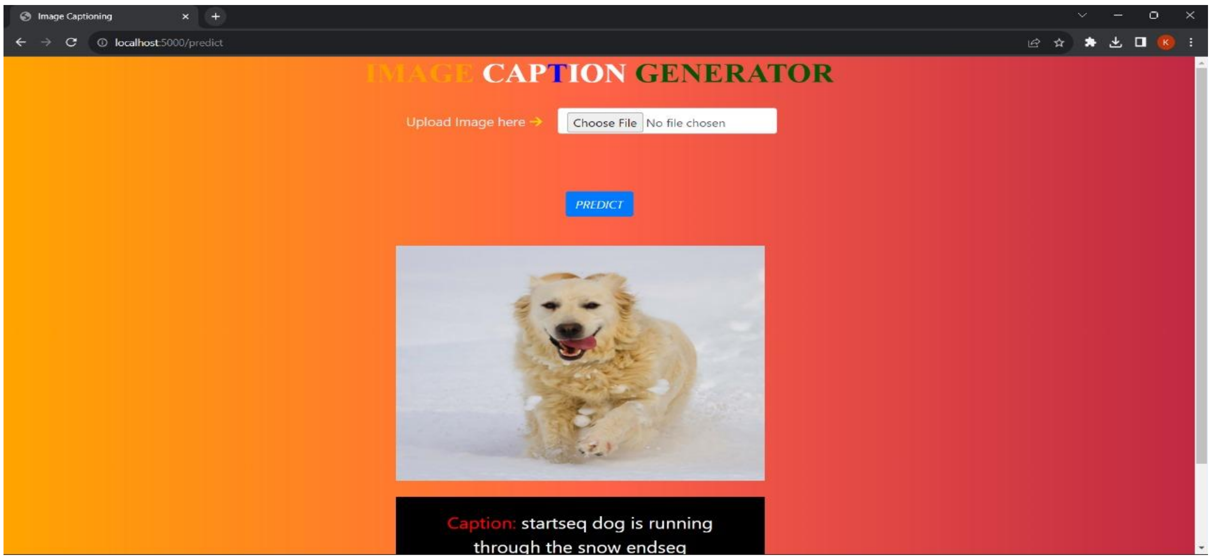
After clicking Sign in the user gets redirected to the main page(index.html)



Now we can choose any file(of jpg,png format) which we want to generate caption

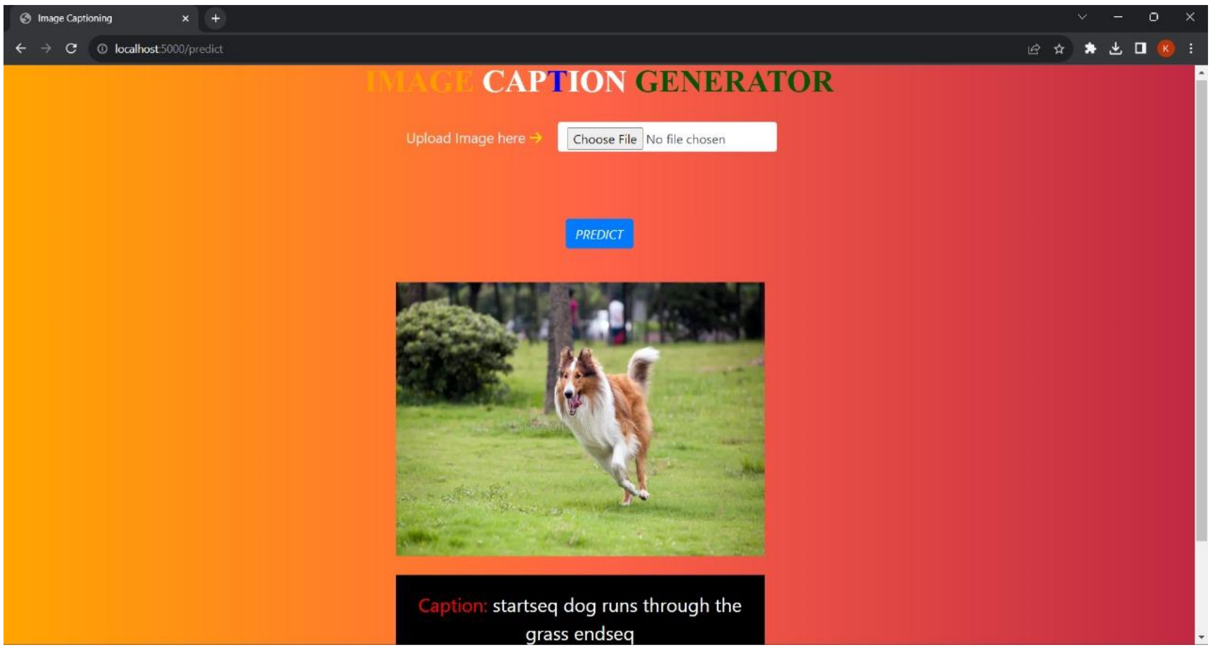


After Selecting file, If we click Predict button it predicts the caption

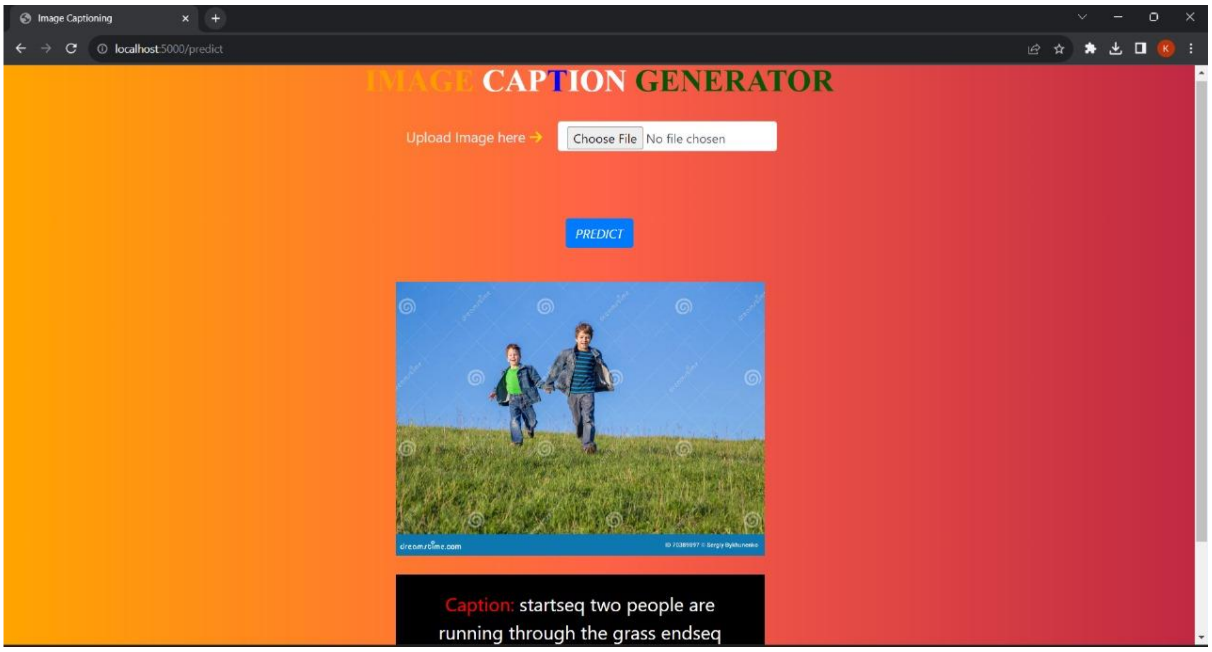


Caption: startseq dog is running through the snow endseq

We can also try for captions for different images



Caption: startseq dog runs through the grass endseq



Caption: startseq two people are running through the grass endseq