Image Describer Net:

A Deep Learning Approach to Image Caption Generation

Introduction:

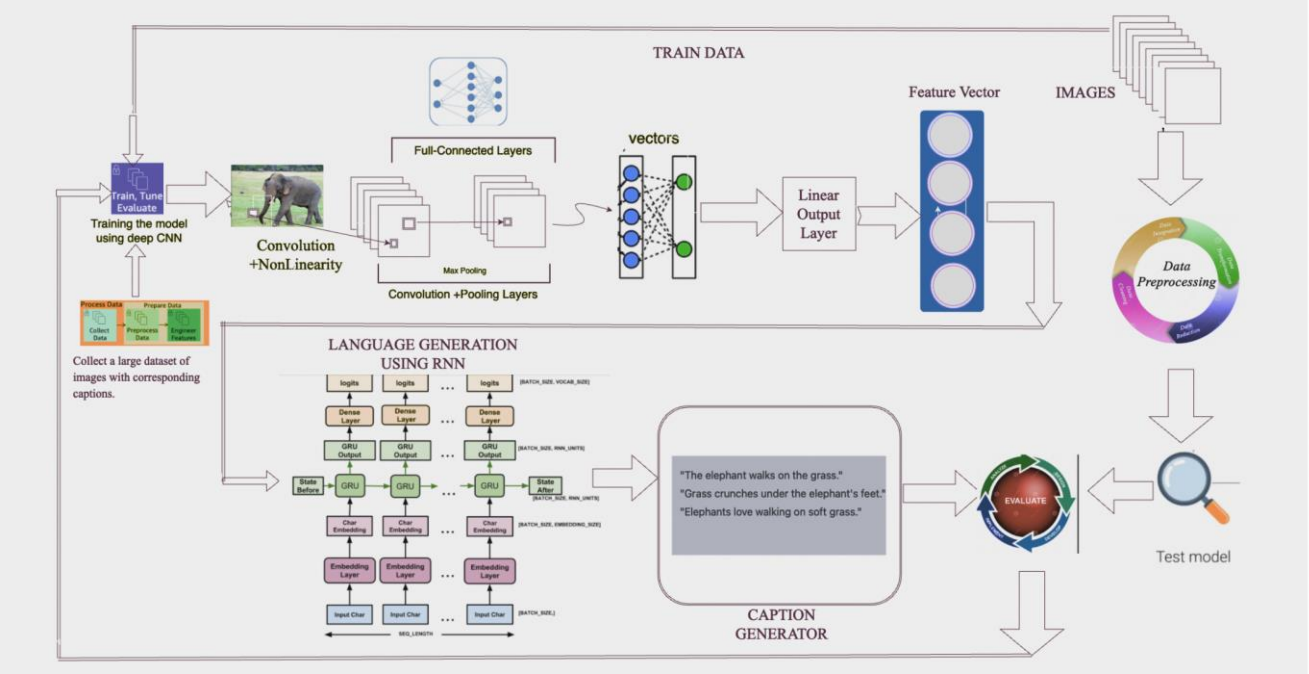
In the realm of artificial intelligence and computer vision, the task of image caption generation involves creating a human-readable textual description based on the content of an image. While this task is intuitive for humans, it poses a significant challenge for computers, as it necessitates the comprehension of visual content and the ability to articulate this understanding in natural language. The advent of deep learning methods, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) like Long Short-Term Memory (LSTM), has revolutionized the field, outperformed classical approaches and achieved state-of-the-art results in automatically generating image captions.

This project harnesses the power of deep neural network models, specifically CNNs and LSTMs, to automatically generate descriptive captions for images, exemplifying the advancements in deep learning techniques. The success of these models is contingent upon extensive datasets and computational resources, enabling the development of sophisticated models capable of interpreting images and translating that understanding into coherent and contextually relevant language. By employing CNNs for image feature extraction and LSTM for sequential data processing, the project illustrates the synergy between computer vision and natural language processing, showcasing the potential of deep learning in bridging the gap between visual content and textual descriptions.

As the project unfolds, it exemplifies the growth and significance of deep learning in image caption generation, demonstrating the effectiveness of techniques like CNNs and LSTMs. The utilization of large datasets and powerful computing resources underscores the data-driven nature of these approaches, emphasizing the importance of scale in training models that can seamlessly generate captions for diverse images. The project serves as an insightful exploration into the application of deep learning for image captioning,

shedding light on the evolving landscape of AI and its impactful role in understanding and describing visual content.

Technical Architecture:



DEFINITIONS:

- **Embedding Layer:** In image captioning, the Embedding layer is like a dictionary that converts words into vectors. It helps the model understand the relationships between different words in the captions.
- **LSTM Layer:** The Long Short-Term Memory (LSTM) layer is a type of memory unit in the neural network that allows the model to remember information from earlier words in the captions, helping it generate coherent and contextually relevant sentences.
- **Dense Layer:** The Dense layer produces the final output, predicting the next word in the caption. It takes the information learned by the previous layers and maps it to the vocabulary, providing the most likely word based on the given context.
- **Vocabulary Size:** This is the total number of unique words in your dataset. It helps the model understand the range of words it might encounter in captions.

- **Embedding Dimensions:** The number of dimensions in which each word will be represented as a vector. Higher dimensions allow the model to capture more intricate relationships between words.
- **Adam Optimizer:** Adam is an optimization algorithm that adjusts the model's weights during training, helping it converge to an optimal solution efficiently.
- **Categorical Cross entropy Loss:** This is a measure of how well the model's predictions match the actual captions. It penalizes the model more when it makes larger mistakes, guiding it to improve its caption predictions.
- **VGG16:** VGG16 is a deep convolutional neural network architecture widely used for image classification. It consists of 16 weight layers, including convolutional and fully connected layers, and is known for its simplicity and effectiveness. Pre-trained on large datasets like ImageNet, VGG16 can extract intricate features from images, making it valuable for various computer vision tasks.
- **CNN (Convolutional Neural Network):** A type of deep neural network designed for visual pattern recognition. It uses convolutional layers to automatically learn hierarchical representations of features from input images, making it effective for tasks like image classification.
- **LSTM (Long Short-Term Memory):** A type of recurrent neural network (RNN) architecture that excels in capturing long-term dependencies in sequential data. LSTMs are well-suited for tasks involving time-series data or natural language processing, where context and memory over time are crucial.
- **RNN (Recurrent Neural Network):** A class of neural networks designed to work with sequential data by maintaining hidden states that capture information from previous inputs. While effective for certain tasks, traditional RNNs struggle with long-term dependencies, leading to the development of more advanced architectures like LSTMs.
- **FEATURE EXTRACTION**: Feature extraction refers to the process of transforming raw input data, often high-dimensional and complex, into a reduced set of relevant and informative features. These features capture essential patterns or characteristics, facilitating more efficient and effective analysis by machine learning algorithms.
- **TOKENIZER: A** tokenizer is a linguistic tool that breaks down a text into individual units, such as words or sub words, to facilitate further analysis or processing. It plays a crucial role in natural language processing tasks by converting raw text into a structured format, enabling

efficient computational understanding and manipulation of language. Tokenization is a fundamental step in text preprocessing for various language-related applications, including machine learning and information retrieval.

- **Corpus BLEU:** Corpus BLEU is a metric used for evaluating the quality of machine-generated text by measuring the overlap between the generated output and reference texts, commonly employed in tasks like machine translation.
- **SoftMax and RELU:** SoftMax is an activation function that converts a vector of raw scores into a probability distribution, while RELU (Rectified Linear Unit) is an activation function commonly used in neural networks, introducing non-linearity by outputting the input directly if positive and zero otherwise.

Project Objectives:

By the project's completion, participants will achieve the following milestones:

Mastering Convolutional Neural Network (CNN) Concepts and Techniques:

Acquire a comprehensive understanding of the fundamental principles and techniques underlying Convolutional Neural Networks, a key technology in image processing and computer vision.

Deepened Insight into Image Data:

Develop a profound comprehension of image data, exploring its characteristics, formats, and the challenges associated with processing visual information.

Proficiency in Data Pre-processing Techniques:

Demonstrate the ability to pre-process and clean image data using a variety of techniques, gaining hands-on experience in enhancing data quality and preparing it for effective machine learning model training.

Web Application Development with Flask:

Learn the process of building a web application using the Flask framework, a lightweight and efficient web development tool. Understand the principles of web application architecture, routing, and integration of machine learning models into web-based interfaces.

Explore Additional Concepts in Computer Vision:

Expand knowledge beyond CNNs by exploring additional concepts in computer vision, such as feature extraction, image augmentation, and transfer learning, enriching the understanding of advanced techniques used in real-world applications.

Hands-on Experience with Deployment:

Gain practical experience in deploying machine learning models within a web application environment, ensuring a holistic understanding of the end-to-end development lifecycle.

Effective Communication of Technical Concepts:

Develop communication skills to articulate complex technical concepts clearly, enabling effective collaboration and knowledge sharing within the project team and with stakeholders.

Understanding Model Evaluation Metrics:

Familiarize yourself with model evaluation metrics specific to image-related tasks, gaining insights into assessing model performance and making informed decisions during the development process.

This project is designed to equip participants not only with technical expertise but also with the practical skills needed to apply these concepts in real-world scenarios, fostering a well-rounded proficiency in the intersection of deep learning, computer vision, and web development.

Project Flow:

- Data Collection.

  Create Train and Test Folders.

- Data Pre-processing.
- Model Building

  Import the model building Libraries

  Initializing the model

  Adding Input Layer

  Adding Hidden Layer

Adding Output Layer

Configure the Learning Process

- Training and testing the model

Save the Model

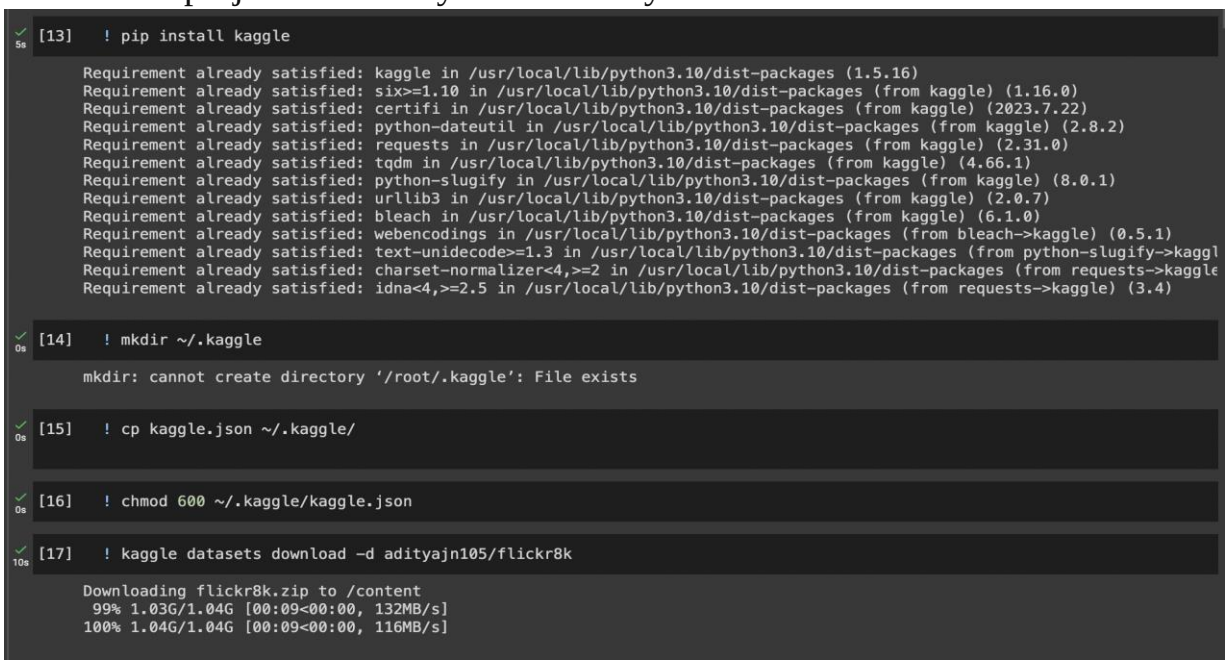
Application Building

Create an HTML file

Build Python Code

Pre-requisites:

We've successfully set up the Kaggle API in Google Colab by installing the Kaggle package, creating the Kaggle directory, copying my API key, and adjusting permissions. With these configurations, I downloaded the "Flickr8k" dataset directly to my Colab environment using the Kaggle CLI. This streamlined process allows for seamless access to Kaggle datasets, facilitating data-driven projects and analyses within my Colab notebook.



IMPORT PACKAGES

STEP-1:

To build machine learning models in Google Colab, essential Python packages are installed using the '!pip install' command. The code imports essential Python

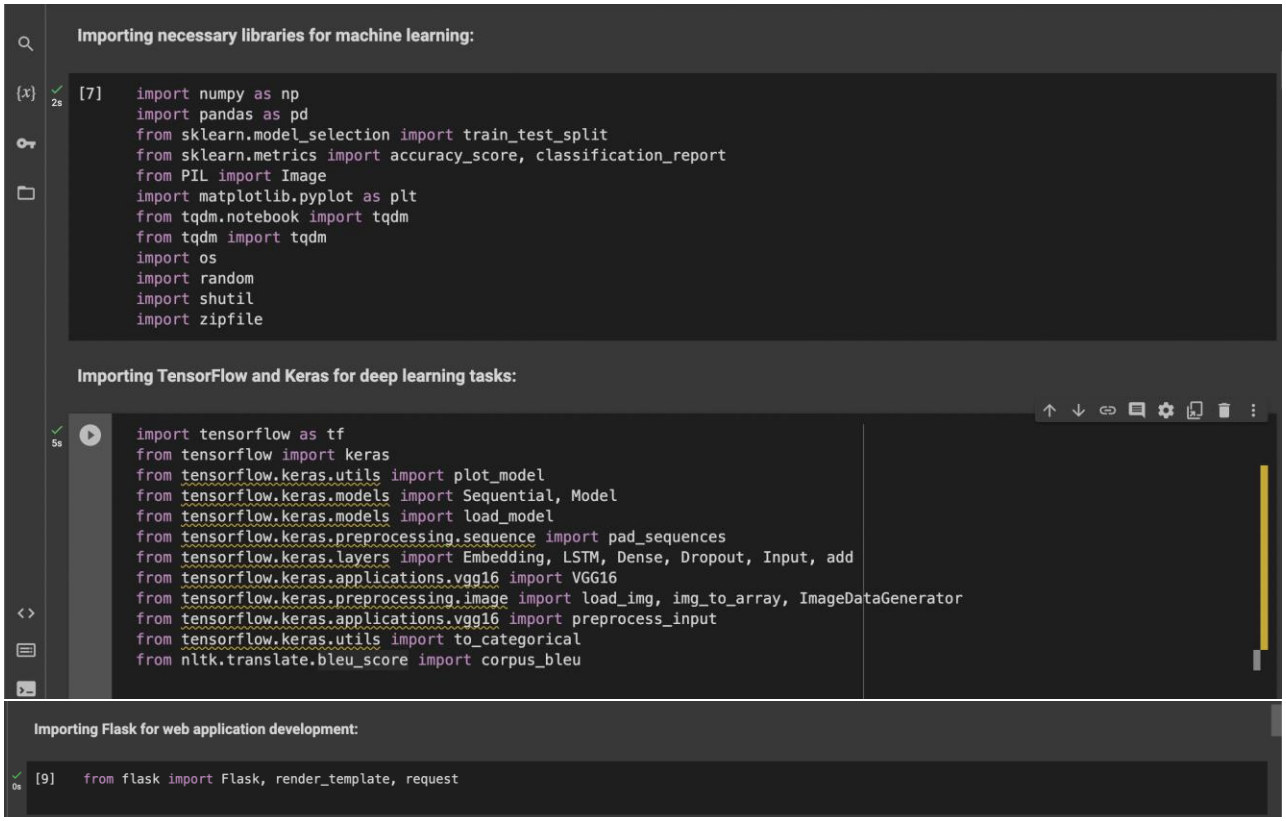
libraries for data manipulation, machine learning, and image processing. NumPy and Pandas are utilized for efficient data handling, Scikit-learn for machine learning tasks, and PIL along with Matplotlib for image processing and visualization. The tqdm library is included for displaying progress bars during iterations. Operating system-related tasks, random number generation, and file operations are facilitated by os, random, and shutil libraries. Additionally, zipfile is imported, suggesting potential involvement in handling compressed files or datasets. This comprehensive set of libraries equips the user with versatile tools for various data science and machine learning tasks.

STEP-2:

The code imports essential modules from TensorFlow and Keras libraries for building and training neural networks. Sequential and Model classes facilitate model creation, while load_model allows loading pre-trained models. Various layers, such as Embedding, LSTM, Dense, and Dropout, are imported for constructing neural network architectures. The VGG16 model, a powerful pre-trained convolutional neural network for computer vision, is included. Functions for preprocessing images, handling sequences, and plotting model architectures are also imported. The corpus_bleu function from NLTK is brought in for evaluating text generation tasks. This set of imports provides a comprehensive toolkit for developing and accessing machine learning models in the context of image and sequence processing tasks.

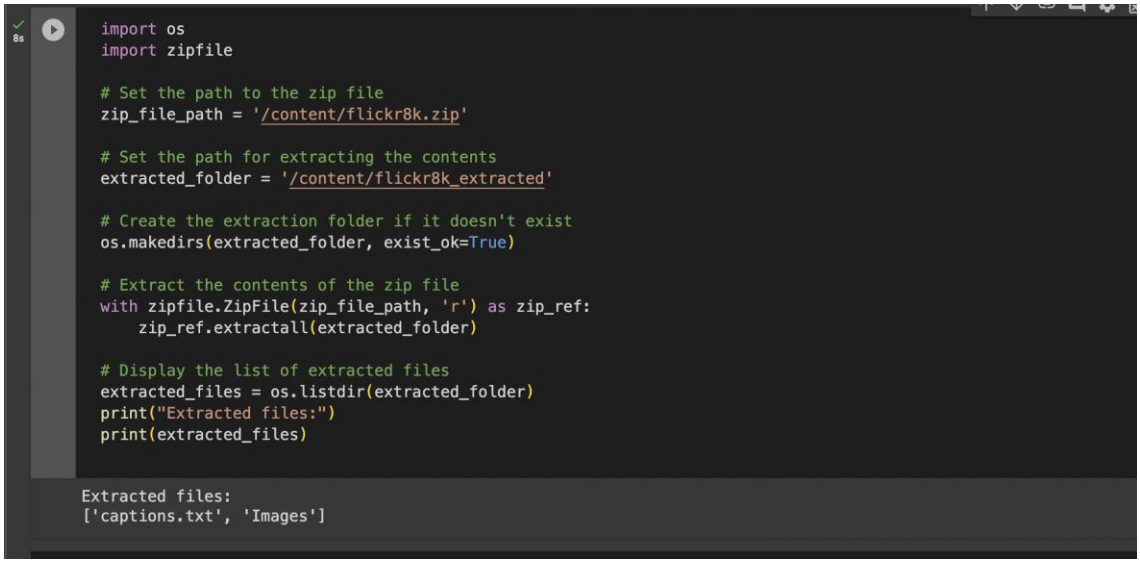
STEP-3

Step 3 encapsulates the entire process, from installing necessary packages to incorporating code snippets for constructing a machine learning model in Google Calab. This comprehensive pipeline enables users to seamlessly work on image caption generation projects in a collaborative and cloud-based environment.



Data Collection:

I established the path for the dataset zip file, naming it "flickr8k.zip". Following that, I designated a folder named "flickr8k_extracted" to serve as the destination for extracting the contents. If this folder didn't exist, I took the initiative to create it. Utilizing the zipfile module, I executed the extraction process, retrieving all files from "flickr8k.zip" and placing them neatly into "flickr8k_extracted". To offer transparency about the dataset's structure after this extraction, I thoughtfully listed and displayed the names of the extracted files.



```
import os
import zipfile

# Set the path to the zip file
zip_file_path = '/content/flickr8k.zip'

# Set the path for extracting the contents
extracted_folder = '/content/flickr8k_extracted'

# Create the extraction folder if it doesn't exist
os.makedirs(extracted_folder, exist_ok=True)

# Extract the contents of the zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extracted_folder)

# Display the list of extracted files
extracted_files = os.listdir(extracted_folder)
print("Extracted files:")
print(extracted_files)
```

```
Extracted files:
['captions.txt', 'Images']
```

Data preparation process

Data preparation process, where you unzip and organize the dataset for further analysis or model training in your project.

Mapping creation process:

I established paths, extracted contents from the Flickr8k dataset, and created a mapping dictionary linking image IDs to their respective captions. This groundwork sets the stage for subsequent tasks in building a machine learning model for image captioning.

In this step, I aggregated all captions from the mapping dictionary into a list named **all_captions**. This list contains textual descriptions associated with images, forming a comprehensive dataset for training a machine learning model in subsequent stages of the project.

```
# Set the path to the extracted folder
extracted_folder = '/content/flickr8k_extracted'

# Set the path to the captions file
captions_file_path = os.path.join(extracted_folder, 'captions.txt')

# Create a mapping dictionary
mapping = {}

# Process lines in the captions file
with open(captions_file_path, 'r') as captions_file:
    captions_doc = captions_file.read()

    # Process lines
    for line in tqdm(captions_doc.split('\n'), desc="Creating Mapping"):
        # Split the line by comma(,)
        tokens = line.split(',')

        if len(tokens) < 2:
            continue

        # Extract image_id and caption
        image_id, caption = tokens[0], tokens[1:]

        # Remove extension from image ID
        image_id = image_id.split('.')[0]
```

```
        # Convert caption list to string
        caption = " ".join(caption)

        # Create a list if needed
        if image_id not in mapping:
            mapping[image_id] = []

        # Store the caption
        mapping[image_id].append(caption)
```

```
Creating Mapping: 100% 40457/40457 [00:00<00:00, 172140.50it/s]
```

```
[13]  len(mapping)
```

```
8092
```

```
[14]  # Initialize a list to store all captions
      all_captions = []

      # Iterate over keys in the mapping dictionary
      for key in mapping:
          # Iterate over captions in the current key
          for caption in mapping[key]:
              # Append the caption to the list
              all_captions.append(caption)
```

```
[15]  len(all_captions)
```

```
40456
```

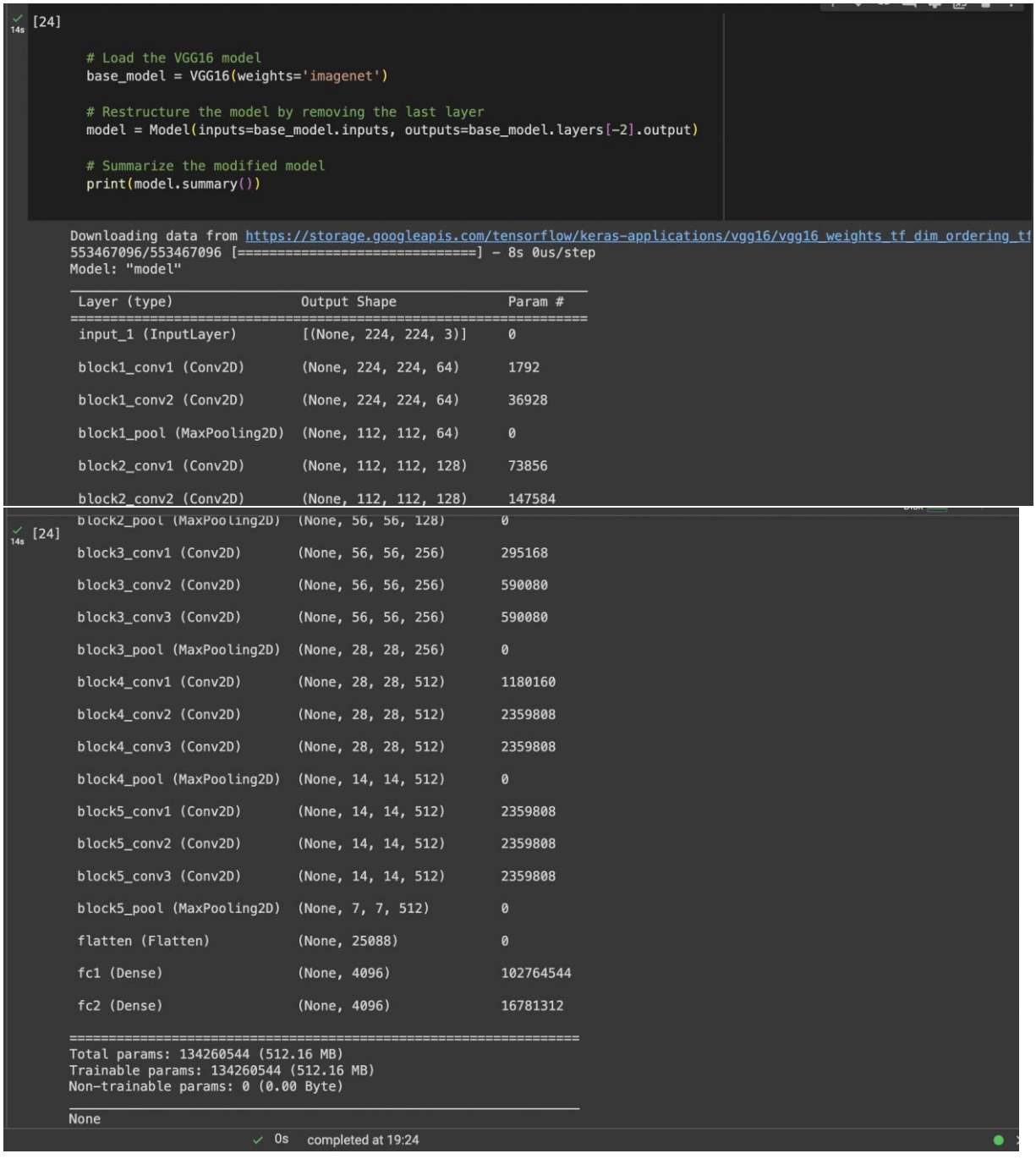
Initializing the model:

VGG16 model

The provided code loads the VGG16 model, a well-known convolutional neural network (CNN) pre-trained on the ImageNet dataset for image classification. It then restructures the model by removing its last layer, which is typically a dense layer responsible for image classification. This modification transforms the VGG16 model into a feature extractor, preserving its ability to capture high-level features from images.

The significance of this modification lies in the concept of transfer learning. By excluding the final classification layer, the modified model can be

used as a powerful feature extractor for various image-related tasks, such as image retrieval or image captioning. This process leverages the knowledge acquired by VGG16 during its extensive training on ImageNet, allowing for effective reuse of learned features in different applications without the need for extensive retraining. This is particularly useful when dealing with limited labeled data or computational constraints.



```
[24]

# Load the VGG16 model
base_model = VGG16(weights='imagenet')

# Restructure the model by removing the last layer
model = Model(inputs=base_model.inputs, outputs=base_model.layers[-2].output)

# Summarize the modified model
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf
553467096/553467096 [==============================] - 8s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

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

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

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

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

=================================================================
Total params: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

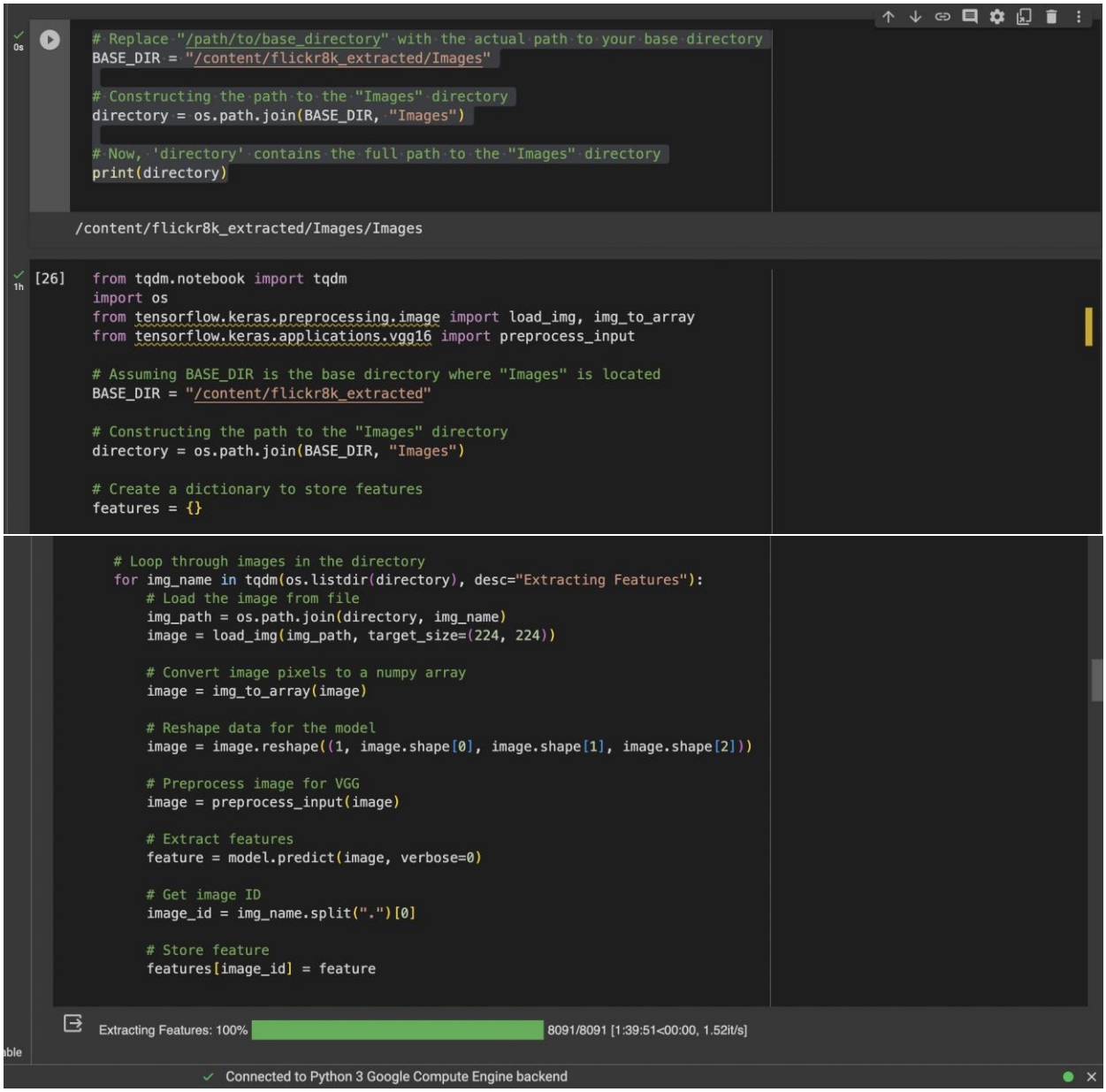
```
0s  completed at 19:24
```

Feature Extraction:

The provided code is contributing to the image captioning pipeline by extracting visual features from images using the VGG16 model. The code initializes a

directory path to the folder containing images, loads each image, preprocesses it, and extracts its features using the VGG16 model. These features are then stored in a dictionary with image IDs as keys.

This step is crucial for image captioning models as it bridges the gap between image content and natural language. The extracted features serve as a rich representation of the visual content, allowing subsequent models to understand the visual context of the images. In the broader image captioning workflow, these features can be used alongside textual information to train models that generate descriptive captions for images. The VGG16 model, with its pre-trained weights on ImageNet, provides a robust feature extraction backbone for this task.



"Caption Mapping" or "Data Parsing":

This code snippet is part of the data preprocessing phase. It reads the content of the 'captions.txt' file, extracts image IDs and captions, removes file extensions, and creates a mapping between image IDs and corresponding captions. This step is crucial for organizing and structuring the textual information associated with each image, laying the foundation for subsequent steps in the image captioning pipeline.

```
[27]
# Open 'captions.txt' file in read mode
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    # Skip the first line (header)
    next(f)

    # Read the remaining content into captions_doc
    captions_doc = f.read()
```

```
[28]  # Create a mapping of image IDs to captions
      mapping = {}

      # Process lines
      for line in tqdm(captions_doc.split("\n")):
          # Split the line by comma(,)
          tokens = line.split(',')

          # Check if the line has enough tokens
          if len(tokens) < 2:
              continue

          # Extract image_id and caption
          image_id, caption = tokens[0], tokens[1:]

          # Remove extension from image ID
          image_id = image_id.split('.')[0]

          # Convert caption list to string
          caption = " ".join(caption)

          # Create a list if needed
          if image_id not in mapping:
              mapping[image_id] = []

          # Store the caption
          mapping[image_id].append(caption)

100% 40456/40456 [00:00<00:00, 227632.23it/s]
```

```
[29]  len(mapping)
8091
```

The expression len(mapping) returns the number of items (key-value pairs) in the mapping dictionary. In this context, it means that there are 8,091 unique image IDs in your dataset, each associated with one or more captions. This count represents the total number of images for which you have captions in your dataset.

Text Cleaning:

This code defines a function called clean that takes a mapping of image IDs to captions and performs several preprocessing steps on each caption. The preprocessing steps include converting the text to lowercase, removing digits and special characters, eliminating extra spaces, and adding start and end tags to the caption. These steps are commonly performed to standardize and clean textual data before training a model.

```python
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # Take one caption at a time
            caption = captions[i]

            # Preprocessing steps
            # Convert to lowercase
            caption = caption.lower()

            # Delete digits, special characters, etc.
            caption = caption.replace("[^A-Za-z]","")
            # Delete additional spaces
            caption = caption.replace("\s",' ')
            # Add start and end tags to the caption
            caption = 'startseq ' + ' '.join([word for word in caption.split() if len(word) > 1]) + ' endseq'

            # Update the caption in the mapping
            captions[i] = caption
```

Before clean and after cleaning sentences look like this:

```
[22]  mapping["1000268201_693b08cb0e"]

['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']

[23]  clean (mapping)

[24]  mapping["1000268201_693b08cb0e"]

['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

Data Aggregation:

The provided code is collecting all captions from the mapping dictionary and consolidating them into a single list named **all_captions**. Each caption is associated with a specific image ID, and this step helps gather all captions together for further processing. This is often a data preparation step, and the resulting list can be used for tasks such as tokenization, vocabulary creation, or training language models.

```
all_captions = []

# Iterate through the mapping
for key in mapping:
    # Iterate through captions for each image ID
    for caption in mapping[key]:
        # Append the caption to the list
        all_captions.append(caption)

# Example: Print the first 5 captions
print(all_captions[:5])
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq', 'startseq girl going into wooden bui
```

```
[33]  len(all_captions)
```

```
40455
```

```
[34]  all_captions[:10]
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tricolored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the street endseq',
 'startseq two dogs of different breeds looking at each other on the road endseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

"Text Tokenization" or "Vocabulary Building"

This code snippet utilizes the TensorFlow Keras library to perform tokenization on a collection of captions. Tokenization involves converting textual data into numerical form by assigning a unique integer to each word in the text. In this context, the code is specifically tokenizing captions, thereby building a vocabulary and transforming the textual information into sequences of integers. This process is crucial for preparing text data to be fed into machine learning models, particularly those involved in natural language processing tasks such as image captioning.

```
from tensorflow.keras.preprocessing.text import Tokenizer

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

# Example: Print vocabulary size
print("Vocabulary Size:", vocab_size)
```

```
Vocabulary Size: 8485
```

The output indicates that the vocabulary size is 8485, which means there are 8485 unique words in the tokenized captions. Each word is assigned a unique integer index. This numerical representation is essential for training

machine learning models, as it allows the model to process and understand the textual information in a format that it can work with.

The output indicates that the maximum length of the captions in the dataset is 35 words. This information is crucial for data preprocessing, particularly when dealing with sequence data like captions.

```
[29]  # Get the maximum length of the caption available
      max_length = max(len(caption.split()) for caption in all_captions)
      print("Maximum Caption Length:", max_length)

Maximum Caption Length: 35
```

Data splitting:

Train-Test Split:

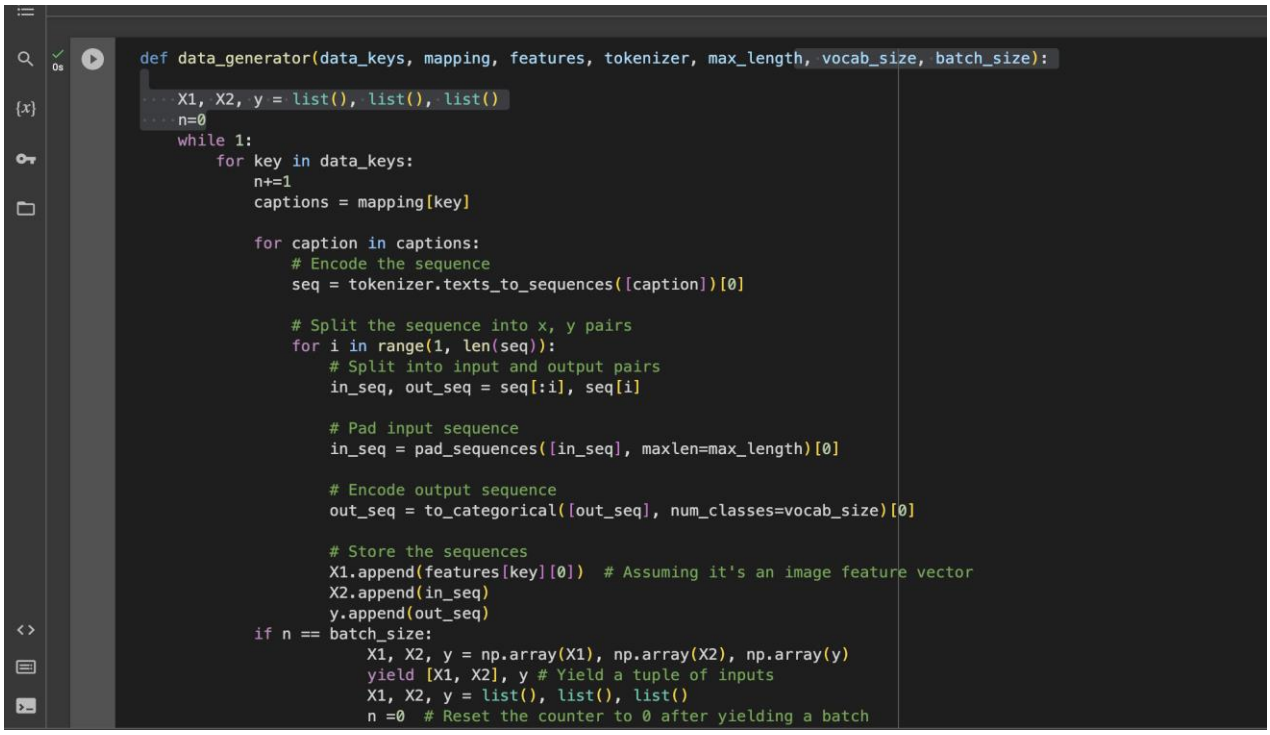
The "train-test split" is a fundamental practice in machine learning aimed at assessing the performance of a model on unseen data. This process involves dividing a dataset into two subsets: a training set, used to train the model by exposing it to labelled examples, and a testing set, held back to evaluate the model's performance on new, unseen instances. The training set enables the model to learn patterns and relationships within the data, while the testing set serves as an independent benchmark to gauge how well the model generalizes to previously unseen samples. The train-test split is essential for ensuring that the model's evaluation reflects its ability to make accurate predictions on new, real-world data, providing valuable insights into its effectiveness and potential for broder application

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

Sequence Encoding and Padding:

The provided code is a data generator for training a neural network model for image captioning. It takes as input a set of image keys, a mapping of image IDs to captions, image features, a tokenizer, maximum sequence length, vocabulary size, and batch size. The generator produces batches of training data containing

input sequences, partial captions, and corresponding output sequences. It iterates through the image keys, encoding the captions, splitting them into input and output pairs, padding the input sequences, and one-hot encoding the output sequences. The generator yields batches continuously for training the image captioning model. In summary, this code contributes to the data preprocessing phase of the image captioning pipeline, preparing the input data for training the neural network.



```
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):

    X1, X2, y = list(), list(), list()
    n=0
    while 1:
        for key in data_keys:
            n+=1
            captions = mapping[key]

            for caption in captions:
                # Encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]

                # Split the sequence into x, y pairs
                for i in range(1, len(seq)):
                    # Split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]

                    # Pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]

                    # Encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    # Store the sequences
                    X1.append(features[key][0])  # Assuming it's an image feature vector
                    X2.append(in_seq)
                    y.append(out_seq)
            if n == batch_size:
                        X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                        yield [X1, X2], y # Yield a tuple of inputs
                        X1, X2, y = list(), list(), list()
                        n =0  # Reset the counter to 0 after yielding a batch
```

ENCODING AND DECODING:

Encoding (in the Encoder Model):

- **Purpose:** The encoder is responsible for extracting meaningful features from the input data, which, in this case, is an image.
- **Details:** The image features are fed into the encoder (**inputs1**). The layers (**fel** and **fe2**) process these features to create a condensed representation (**fe2**) that captures essential information about the input image. This condensed representation serves as a rich feature vector that summarizes the content of the image.

Decoding (in the Decoder Model):

- **Purpose:** The decoder takes the encoded information and generates a meaningful output, which, in this case, is a textual caption for the image.
- **Details:** The decoder takes two inputs—image features (**fe2**) from the encoder and sequence features (**se3**) from the input caption sequence. These features are combined (**decoder1**) and then processed through

additional layers (**decoder2** and **outputs**) to generate a probability distribution over the vocabulary. This distribution represents the likelihood of each word in the vocabulary being the next word in the caption. During training, the model learns to adjust its parameters to maximize the likelihood of generating the correct caption.

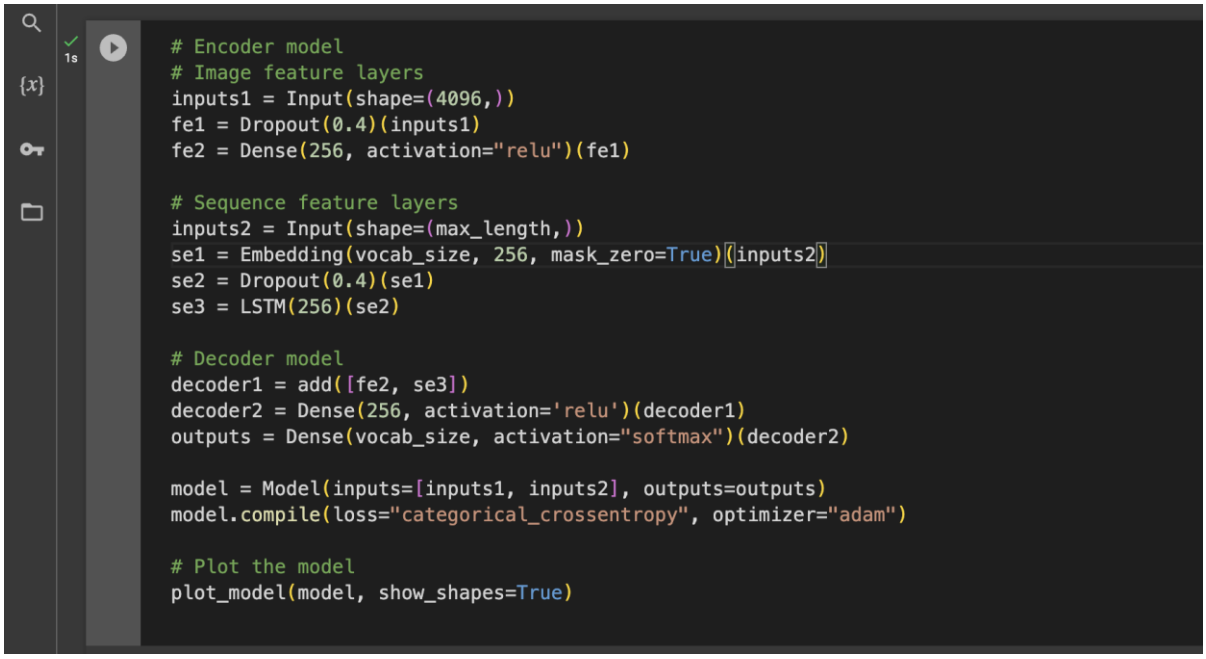
In summary, encoding is the process of extracting relevant features from input data (image), and decoding is the process of using these features to generate meaningful output (caption). The combination of encoding and decoding allows the model to learn a mapping between images and their corresponding captions, enabling it to generate captions for new, unseen images. This is a common approach in image captioning tasks, where the goal is to automatically describe the content of images in natural language

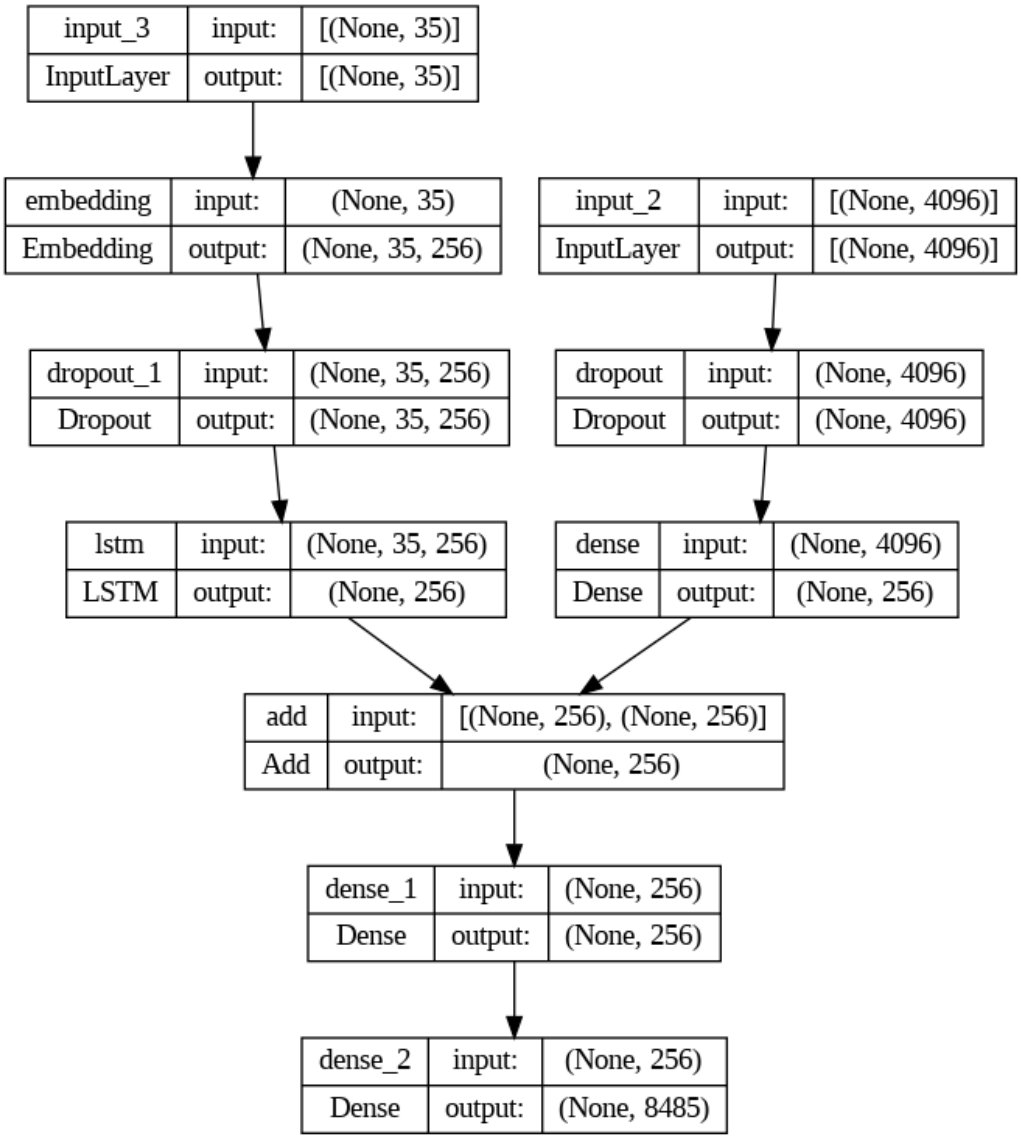
Encoder Model (inputs1):

- **Inputs:** The model takes, as input, a vector of length 4096, representing the features extracted from an image.
- **Dropout Layer:** Introduces dropout to prevent overfitting by randomly setting a fraction of input units to zero during training.
- **Dense Layer (fe2):** A fully connected layer with 256 neurons and ReLU activation. This processes the image features.

Decoder Model (inputs2):

- **Inputs:** Takes a sequence of word indices (captions) with a maximum length of 20.
- **Embedding Layer (sel):** Transforms the input sequences into dense vectors of fixed size (256 dimensions in this case). It also masks zero values.
- **Dropout Layer (se2):** Introduces dropout for regularization in the sequence processing.
- **LSTM Layer (se3):** A Long Short-Term Memory layer with 256 units. It processes the sequential information in the captions.


1s
# Encoder model
# Image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation="relu")(fe1)
# Sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)
# Decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation="softmax")(decoder2)
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss="categorical_crossentropy", optimizer="adam")
# Plot the model
plot_model(model, show_shapes=True)


input_3 | input: | [(None, 35)]
InputLayer | output: | [(None, 35)]
embedding | input: | (None, 35)
Embedding | output: | (None, 35, 256)
input_2 | input: | [(None, 4096)]
InputLayer | output: | [(None, 4096)]
dropout_1 | input: | (None, 35, 256)
Dropout | output: | (None, 35, 256)
dropout | input: | (None, 4096)
Dropout | output: | (None, 4096)
lstm | input: | (None, 35, 256)
LSTM | output: | (None, 256)
dense | input: | (None, 4096)
Dense | output: | (None, 256)
add | input: | [(None, 256), (None, 256)]
Add | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 8485)

Training phase:

The provided code snippet involves training a model for a specified number of epochs using a generator-based approach. The key elements include defining parameters such as the number of epochs, batch size, and steps per epoch. The training process iterates through each epoch, generating batches of training data using a custom data generator that combines image features, tokenized captions, and target sequences. The model is then trained for one epoch using the generated batches, and the process is repeated for the specified number of epochs. This training approach is crucial in refining the model's parameters and improving its ability to generate accurate captions for images. In succinct terms, the code represents the training phase of a machine learning model for image captioning, contributing to the model's learning and adaptation to the provided data.

Number of Epochs: The number of epochs refers to the number of times the entire dataset is passed forward and backward through the neural network during training. One epoch is completed when every sample in the training set has been processed once. Increasing the number of epochs allows the model to see the data more times, potentially improving its performance, but it's essential to monitor for overfitting.

Batch Size: Batch size represents the number of training samples utilized in one iteration. During each epoch, the dataset is divided into batches, and the model's parameters are updated based on the average gradient of the loss calculated on each batch. Smaller batch sizes require less memory but may lead to less stable updates. Larger batch sizes may provide more stable updates but require more memory.

Steps per Epoch: Steps per epoch is the number of batches processed in each epoch. It is typically set to the total number of training samples divided by the batch size. This parameter influences how often the model's parameters are updated. A larger value for steps per epoch results in more frequent updates but may extend the training time. It is crucial to choose an appropriate value to balance computational efficiency and model convergence.

```
[32]  #train the model
      epochs = 20
      batch_size = 32
      steps = len(train) // batch_size

      for i in range(epochs):
          #create data generator
          generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
          #fit for one epoch
          model.fit(generator, epochs=1, steps_per_epoch = steps, verbose=1)
```

```
227/227 [==============================] - 774s 3s/step - loss: 5.2214
227/227 [==============================] - 770s 3s/step - loss: 4.0046
227/227 [==============================] - 771s 3s/step - loss: 3.5804
227/227 [==============================] - 769s 3s/step - loss: 3.3168
227/227 [==============================] - 786s 3s/step - loss: 3.1177
227/227 [==============================] - 774s 3s/step - loss: 2.9685
227/227 [==============================] - 775s 3s/step - loss: 2.8572
227/227 [==============================] - 776s 3s/step - loss: 2.7648
227/227 [==============================] - 776s 3s/step - loss: 2.6827
227/227 [==============================] - 781s 3s/step - loss: 2.6141
227/227 [==============================] - 779s 3s/step - loss: 2.5526
227/227 [==============================] - 783s 3s/step - loss: 2.4997
227/227 [==============================] - 777s 3s/step - loss: 2.4502
227/227 [==============================] - 777s 3s/step - loss: 2.4093
227/227 [==============================] - 783s 3s/step - loss: 2.3660
227/227 [==============================] - 785s 3s/step - loss: 2.3274
227/227 [==============================] - 792s 3s/step - loss: 2.2883
227/227 [==============================] - 782s 3s/step - loss: 2.2635
227/227 [==============================] - 783s 3s/step - loss: 2.2260
227/227 [==============================] - 779s 3s/step - loss: 2.1990
```

Model Saving:

The code provided is saving the trained model to a file named "IMAGE_CAPTION_best_model.h5" in the specified directory (WORKING DIR). This step is essential in machine learning workflows, especially after training a model. Saving the model allows you to reuse it later for making predictions on new data without having to retrain the model each time.
It involves storing the model's architecture, weights, and other configuration details to a file, making it convenient for future use without the need for retraining.

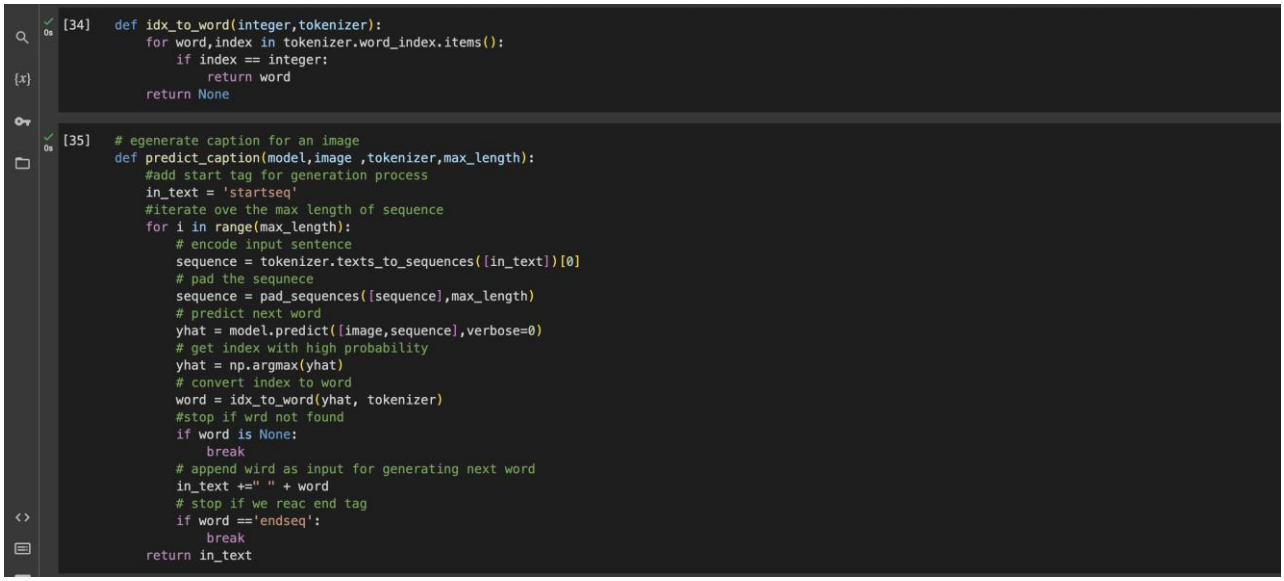
Model Saving

```
[33]  # Replace "WORKING DIR" with the actual path to the directory where you want to save the model
      model.save("/content/IMAGE_CAPTION_best_model.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This f
  saving_api.save_model(
```

Caption Generation.

The provided code is a crucial component of a project focused on image captioning. The function predict caption is designed to generate descriptive captions for images using a pre-trained neural network model. It takes an image, a tokenizer (used for mapping words to numerical indices), and a specified maximum length for the generated sequence as inputs. The process begins with an initial sequence starting with the tag 'startseq,' and the model iteratively predicts the next word in the sequence. The idx_to_word function assists in converting predicted indices back into their corresponding words. This sequence generation process continues until the model predicts the end tag 'endseq' or reaches the specified maximum length. Overall, this code contributes to the generation of meaningful captions for images, enhancing the interpretability and utility of machine learning models in various applications.



```
[34]  def idx_to_word(integer,tokenizer):
          for word,index in tokenizer.word_index.items():
              if index == integer:
                  return word
          return None

[35]  # egenerate caption for an image
      def predict_caption(model,image ,tokenizer,max_length):
          #add start tag for generation process
          in_text = 'startseq'
          #iterate ove the max length of sequence
          for i in range(max_length):
              # encode input sentence
              sequence = tokenizer.texts_to_sequences([in_text])[0]
              # pad the sequnece
              sequence = pad_sequences([sequence],max_length)
              # predict next word
              yhat = model.predict([image,sequence],verbose=0)
              # get index with high probability
              yhat = np.argmax(yhat)
              # convert index to word
              word = idx_to_word(yhat, tokenizer)
              #stop if wrd not found
              if word is None:
                  break
              # append wird as input for generating next word
              in_text +=" " + word
              # stop if we reac end tag
              if word =='endseq':
                  break
          return in_text
```

CALCULATES BLEU SCORES:

Code calculates BLEU scores, a widely used metric for assessing the quality of machine-generated text, in the context of an image captioning project. The BLEU-1 score measures the precision of individual words in the generated captions, while the BLEU-2 score extends this evaluation to consider both unigrams and bigrams. The actual and predicted lists represent tokenized reference and machine-generated captions, respectively. These scores provide a quantitative measure of how well the model-generated captions align with human references. In image captioning, where the goal is to generate contextually relevant and linguistically accurate descriptions for images, BLEU scores offer valuable insights into the model's performance, aiding in the

iterative improvement of the captioning system by providing a standardized measure of caption quality.

CALCULATES BLEU SCORES

```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual , predicted =list(),list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key],tokenizer,max_length)
    #split into words
    actual_captions = [caption.split() for caption in  captions]
    y_pred = y_pred.split()
    #append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

#calculate BLEU Score
print("BLEU-1: %f" % corpus_bleu(actual,predicted,weights=(1.0,0,0,0)))
print("BLEU-2: %f" % corpus_bleu(actual,predicted,weights=(0.5,0.5,0,0)))
```

```
100%|██████████| 810/810 [08:23<00:00,  1.61it/s]
BLEU-1: 0.549099
BLEU-2: 0.324626
```

The BLEU score is a metric commonly used for evaluating the quality of machine-generated text, such as translations or image captions. It ranges from 0 to 1, where a higher score indicates better performance. The BLEU score is computed based on the precision of n-grams (contiguous sequences of n items) in the generated text compared to the reference text.

In your specific output:

- "BLEU-1: 0.54" means that, on average, the precision of unigrams (individual words) in your generated captions compared to the reference captions is approximately 54%.
- "BLEU-2: 0.32" means that, on average, the precision of bigrams (two consecutive words) in your generated captions compared to the reference captions is approximately 32%.

"Caption Evaluation":

Load and Display Image:

- Takes an image name as input, constructs the full path to the image, and loads the image using the Python Imaging Library (PIL).
- Displays the loaded image using Matplotlib.

Display Actual Captions:

- Retrieves the actual captions for the given image from a mapping (**mapping**) based on the image's ID.
- Prints the actual captions to provide a reference for comparison.

Predict Caption:

- Calls the **predict_caption** function (assumed to be defined elsewhere) to generate a caption for the given image using a pre-trained captioning model (**model**) and tokenizer (**tokenizer**).
- Prints the predicted caption.

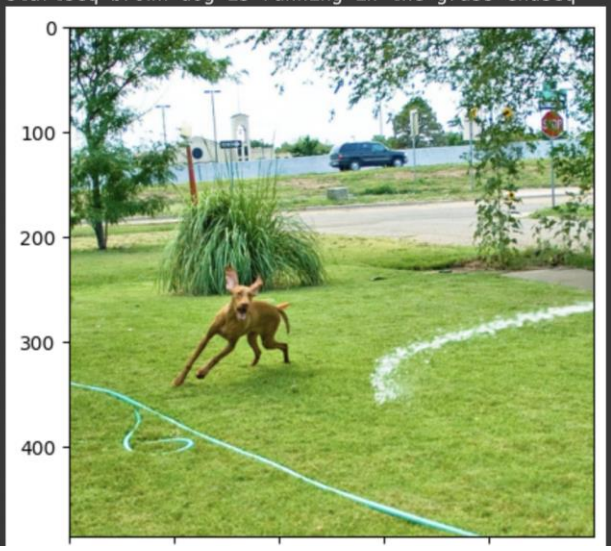
Caption Evaluation

```
[ ]   from PIL import Image
      import matplotlib.pyplot as plt
      def generate_caption(image_name):
          # load the image
      #     image_name = "1001773457_577c3a7d70.jpg"
          image_id = image_name.split('.')[0]
          img_path = os.path.join(BASE_DIR,"Images",image_name)
          image = Image.open(img_path)
          captions = mapping[image_id]
          print('----------------Actual--------------------')
          for caption in captions:
              print(caption)
          y_pred=predict_caption(model,features[image_id],tokenizer , max_length)
          print('----------------Predicted-----------------')
          print(y_pred)
          plt.imshow(image)
```

EXECUTING THE MODEL:

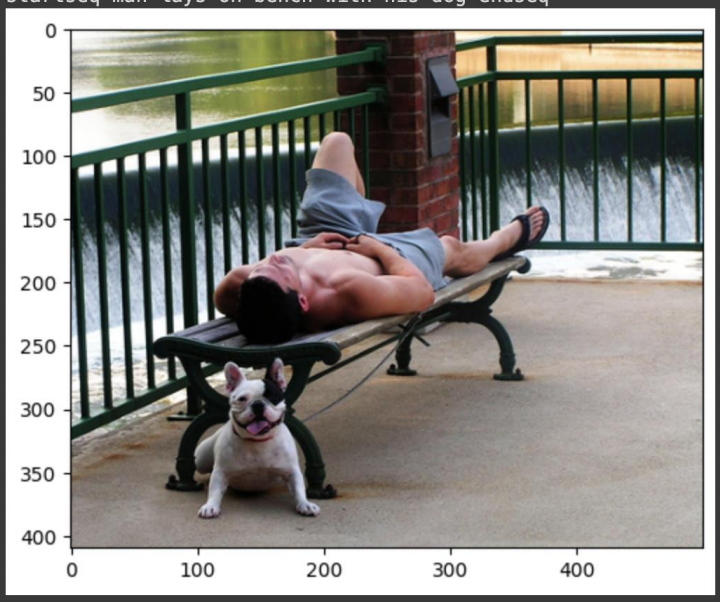
```
[ ]   generate_caption("1019077836_6fc9b15408.jpg")
```

```
----------------Actual--------------------
startseq brown dog chases the water from sprinkler on lawn endseq
startseq brown dog plays with the hose endseq
startseq brown dog running on lawn near garden hose endseq
startseq dog is playing with hose endseq
startseq large brown dog running away from the sprinkler in the grass endseq
----------------Predicted-----------------
startseq brown dog is running in the grass endseq
```



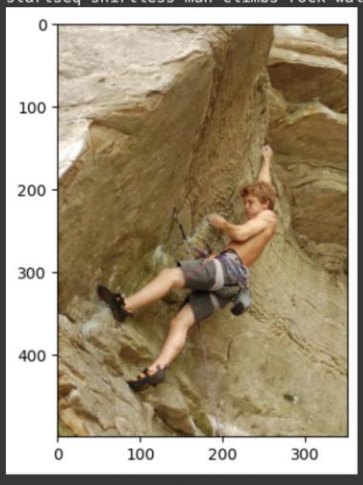
```
generate_caption("1003163366_44323f5815.jpg")
```

```
-----------------Actual--------------------
startseq man lays on bench while his dog sits by him endseq
startseq man lays on the bench to which white dog is also tied endseq
startseq man sleeping on bench outside with white and black dog sitting next to him endseq
startseq shirtless man lies on park bench with his dog endseq
startseq man laying on bench holding leash of dog sitting on ground endseq
----------------Predicted-----------------
startseq man lays on bench with his dog endseq
```




```
generate_caption("236095031_5cb17dc54a.jpg")
```

```
-----------------Actual--------------------
startseq boy rock climbs endseq
startseq man is rock climbing without shirt endseq
startseq young boy rock climbing endseq
startseq the boy looked down as he climbed the steep rock face endseq
startseq the climber is hanging off vertical cliff with gear but no shirt endseq
----------------Predicted-----------------
startseq shirtless man climbs rock wall endseq
```



2s completed at 20:40

+ Code + Text

```
generate_caption("238512430_30dc12b683.jpg")
```

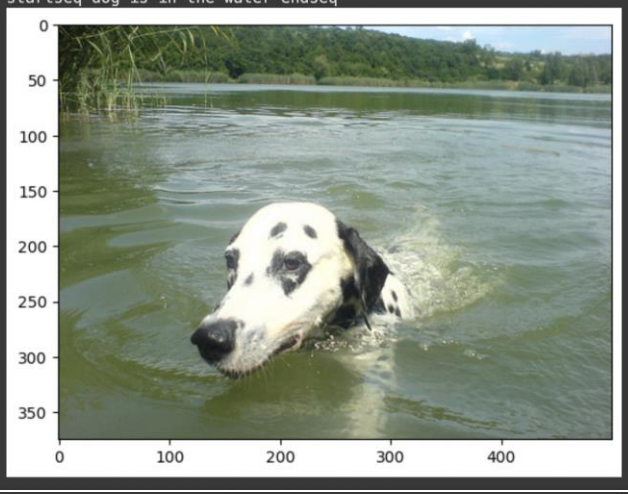
```
----------------Actual----------------
startseq boat is illuminated at night endseq
startseq large ship on the water at dusk endseq
startseq man is sitting on the end of hit boat overlooking the water endseq
startseq person sits on the front deck of ship and city and another ships is lit up in the distance endseq
startseq ship in harbor at night with city skyline behind endseq
----------------Predicted----------------
startseq man is sitting on dock at sunset endseq
```


```
generate_caption("96973080_783e375945.jpg")
```

```
----------------Actual----------------
startseq brown dog is running over snow near leafless trees endseq
startseq brown dog runs across snowy field endseq
startseq dog is running through the snow endseq
startseq dog runs through the snow endseq
startseq medium sized brown dog is running through an open white wooded area endseq
----------------Predicted----------------
startseq dog runs through the snow endseq
```

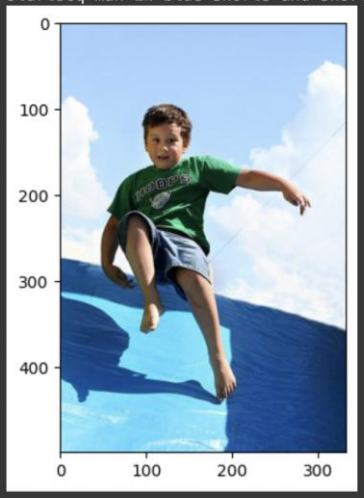
```
generate_caption("3712742641_641282803e.jpg")
```

```
----------------Actual--------------------
startseq black and white dog is swimming in lake endseq
startseq black and white dog is swimming in large green lake endseq
startseq dog swimming in pond endseq
startseq dog treads through dark green water endseq
startseq the black and white dog is swimming in the river endseq
----------------Predicted-----------------
startseq dog is in the water endseq
```




```
generate_caption("3741827382_71e93298d0.jpg")
```

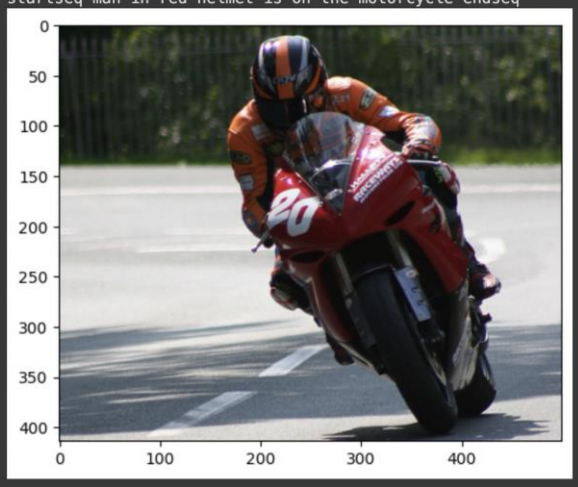
```
----------------Actual--------------------
startseq boy in green shirt above something blue endseq
startseq small boy getting tossed into the air endseq
startseq young boy jumps around on blue mat with half-smile on his face endseq
startseq young boy plays in front of blue object endseq
startseq the boy wearing the green shirt is climbing on blue inflatable endseq
----------------Predicted-----------------
startseq man in blue shorts and shorts is jumping into the air endseq
```



✓ 3s completed at 07:13

```
generate_caption("166321294_4a5e68535f.jpg")
```

```
----------------Actual--------------------
startseq man is riding on red motorcycle endseq
startseq motorcycle driver dressed in orange gear swerves to the right endseq
startseq motorcyclist on red speed bike leans into sharp turn endseq
startseq motorcyclist crouches low as he rounds turn endseq
startseq this person is on red motorcycle endseq
----------------Predicted-----------------
startseq man in red helmet is on the motorcycle endseq
```




```
generate_caption("216172386_9ac5356dae.jpg")
```

```
----------------Actual--------------------
startseq bike sits atop rise with mountains in the background endseq
startseq man wearing red uniform and helmet stands on his motorbike endseq
startseq motocross bike is being ridden over rocks endseq
startseq motocross biker about to descend endseq
startseq the motorcyclist has reached the summit endseq
----------------Predicted-----------------
startseq man in red jacket is riding bicycle on dirt hill endseq
```

APPLICATION BUILDING:

CAPTION CRAFT:

Now that we trained our model, we tried to build a web application **(Caption Craft)** based on the knowledge we got from this project. This web app utilizes a Flask framework to deploy an image captioning model. Users interact with a user-friendly HTML interface, selecting an image for caption generation. The chosen image undergoes preprocessing before being fed into a pre-trained model. The model predicts a descriptive caption for the image, and the results, along with the original image, are displayed on a HTML page. This seamless flow allows users to effortlessly upload images and receive corresponding captions generated by the trained model.

STEP 1:

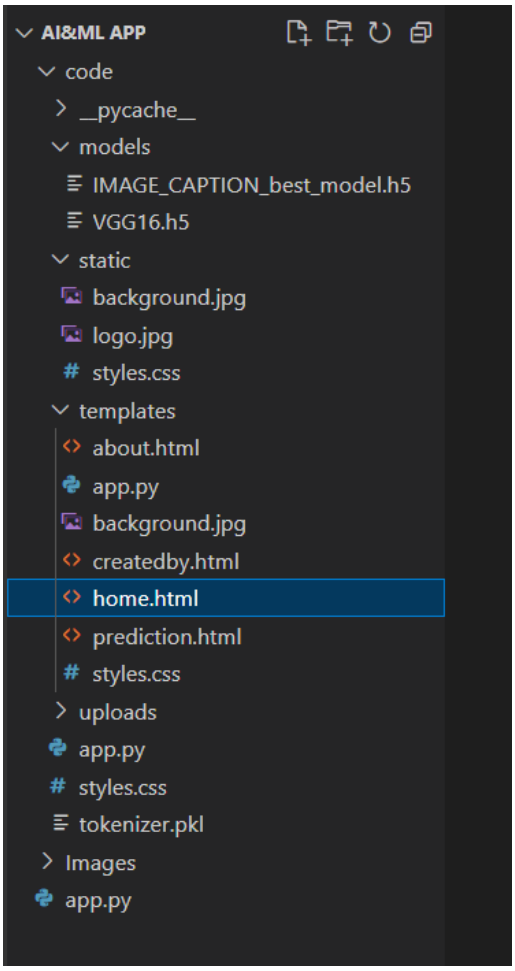
In our web app we use HTML to create the front-end part of the web page.

- **home.html:** This page serves as the main entry point, providing a welcoming interface for users to navigate through the application.

- **about.html:** Users can access this page to gain insights into the project, offering details about the image captioning system and its functionality.
- **createdby.html:** This page offers information about the creators and contributors behind the development of the image captioning web app, acknowledging the individuals involved in building the application.

- These HTML pages are complemented by JavaScript (main.js) and CSS (main.css) to improve functionality and enhance the visual appeal of the web app. The combination of these components provides a comprehensive and user-friendly interface for uploading images and receiving generated captions.

- **STEP 2:**



- Flask provides a lightweight and flexible framework for building web applications, handling various aspects of web development and allowing

developers to focus on implementing application logic.

ENABLE FLASK IN COMMAND PROMPT:

pip install flask
cd "path\to\your\file"
set FLASK_APP=app.py
flask run

STEP 3:

Home.html: Likely contains the main content for the home page, including any introductory information, navigation links, or featured content.

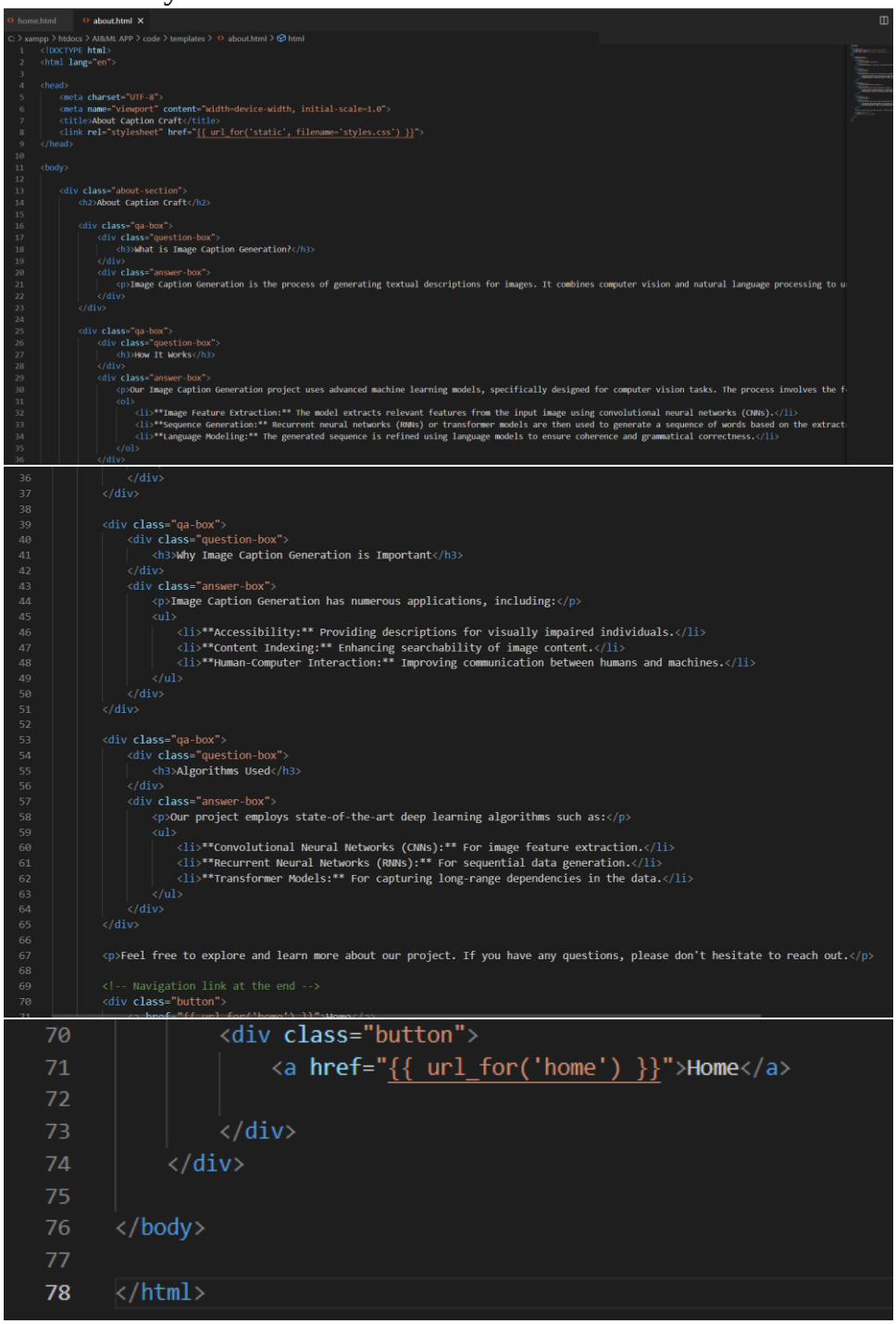
home.html

C: > xampp > htdocs > AI&ML APP > code > templates > home.html > html > head

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Caption Craft</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

</head>

<body>

    <header>
        <img src="logo" alt="Caption Craft Logo">
        <h1>Caption Craft</h1>
    </header>

    <nav>
        <a href="{{ url_for('home') }}">Home</a>

        <a href="{{ url_for('about') }}">About</a>
<a href="{{ url_for('createdby') }}">Created By</a>

    </nav>

    <section class="slide" id="home">
        <h2>Welcome to Caption Craft</h2>
        <p>This is the home page of Caption Craft. Explore our image caption generation project and get started!</p>
        <p>

<a href="{{ url_for('predict') }}" class="button">Get Started</a>

        </p>
    </section>

    <section class="slide" id="about">
        <h2>About Caption Craft</h2>
        <p>Explain completely about image caption generation, how it works, algorithms used, etc.</p>
    </section>

    <section class="slide" id="createdby">
        <h2>Created By</h2>
        <p>Meet the developers:</p>
        <p>BHAVANA</p>
        <p>BHAVANI</p>
    </section>

</body>

</html>
```

STEP 4:About.html:

presents information about the application, its purpose, or details about its functionality.



```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>About Caption Craft</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>

    <div class="about-section">
        <h2>About Caption Craft</h2>

        <div class="qa-box">
            <div class="question-box">
                <h3>What is Image Caption Generation?</h3>
            </div>
            <div class="answer-box">
                <p>Image Caption Generation is the process of generating textual descriptions for images. It combines computer vision and natural language processing to u
            </div>
        </div>

        <div class="qa-box">
            <div class="question-box">
                <h3>How It Works</h3>
            </div>
            <div class="answer-box">
                <p>Our Image Caption Generation project uses advanced machine learning models, specifically designed for computer vision tasks. The process involves the f
                <ol>
                    <li>**Image Feature Extraction:** The model extracts relevant features from the input image using convolutional neural networks (CNNs).</li>
                    <li>**Sequence Generation:** Recurrent neural networks (RNNs) or transformer models are then used to generate a sequence of words based on the extract
                    <li>**Language Modeling:** The generated sequence is refined using language models to ensure coherence and grammatical correctness.</li>
                </ol>
            </div>
            </div>
        </div>

        <div class="qa-box">
            <div class="question-box">
                <h3>Why Image Caption Generation is Important</h3>
            </div>
            <div class="answer-box">
                <p>Image Caption Generation has numerous applications, including:</p>
                <ul>
                    <li>**Accessibility:** Providing descriptions for visually impaired individuals.</li>
                    <li>**Content Indexing:** Enhancing searchability of image content.</li>
                    <li>**Human-Computer Interaction:** Improving communication between humans and machines.</li>
                </ul>
            </div>
        </div>

        <div class="qa-box">
            <div class="question-box">
                <h3>Algorithms Used</h3>
            </div>
            <div class="answer-box">
                <p>Our project employs state-of-the-art deep learning algorithms such as:</p>
                <ul>
                    <li>**Convolutional Neural Networks (CNNs):** For image feature extraction.</li>
                    <li>**Recurrent Neural Networks (RNNs):** For sequential data generation.</li>
                    <li>**Transformer Models:** For capturing long-range dependencies in the data.</li>
                </ul>
            </div>
        </div>

        <p>Feel free to explore and learn more about our project. If you have any questions, please don't hesitate to reach out.</p>

        <!-- Navigation link at the end -->
        <div class="button">
            <a href="{{ url_for('home') }}">Home</a>

            </div>
        </div>

    </body>

    </html>
```

STEP 5: Createdby.html

includes details about the creator of the application, acknowledging contributions or providing contact information.

home.html | about.html | createdby.html

C: > xampp > htdocs > AI&ML APP > code > templates > createdby.html > html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Created By</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

</head>

<body>

    <div class="createdby-section">
        <h2>Developed By</h2>

        <div class="developer-info" id="developer1">
            <div class="developer-names">
                <span>BHAVANA</span>
            </div>

            <div class="additional-info">
                <p>Email: bhavanaperecharla@gmail.com</p>
                <p>Date of Birth: DECEMBER 24, 2003</p>
                <p>Team ID: 591719</p>
                <p>Student ID: 21BCE9184</p>
                <p>College Name: VIT AP College</p>
            </div>
        </div>

        <div class="developer-info" id="developer2">
            <div class="developer-names">
                <span>BHAVANI</span>
            </div>

            <div class="additional-info">
                <p>Email: bhavaniponnapali@gmail.com</p>
                <p>Date of Birth: DECEMBER 23, 2003</p>
                <p>Team ID: 591719</p>
                <p>Student ID: 21BCE9043</p>
                <p>College Name: VIT AP College</p>
            </div>
        </div>

        <!-- Navigation link at the end -->
        <div class="button">
            <a href="{{ url_for('home') }}">Home</a>
        </div>
    </div>

</body>

</html>
```

STEP 6:
app.py

app.py is the main script of the Flask application. It defines the core

functionality, including routes for different URLs (home, about, createdby, predict), file upload handling, and integration with a machine learning model. The script orchestrates the application, allowing it to handle incoming requests, process data, and dynamically render HTML templates, creating a functional web-based image captioning application.

app.py ● | home.html | about.html | createdby.html

C: > xampp > htdocs > AI&ML APP > code > templates > app.py

```python
# Import necessary libraries
from flask import Flask, render_template, request, jsonify
from werkzeug.utils import secure_filename
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing.sequence import pad_sequences
from flask_cors import CORS
import pickle
import numpy as np
import os
import base64

# Create Flask app
app = Flask(_name_, template_folder='templates', static_folder='static')
CORS(app)

# Load your image captioning model
model_path = 'C:\\xampp\\htdocs\\AI&ML APP\\code\\models\\IMAGE_CAPTION_best_model.h5'
model = load_model(model_path, compile=False)

tokenizer_path = 'C:\\xampp\\htdocs\\AI&ML APP\\code\\tokenizer.pkl'  # Specify the ac

with open(tokenizer_path, 'rb') as tokenizer_file:
    tokenizer = pickle.load(tokenizer_file)

max_length = 35

# Function to preprocess image for model input
def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(229, 229))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    img_array = preprocess_input(img_array)
    return img_array
```

```
    return img_array

# Function to generate caption from image features
def generate_caption_from_features(img_array):
    # Assuming your model takes image features as input
    caption = predict_caption(model, img_array, tokenizer, max_length)
    return caption

# Function to predict caption
def predict_caption(model, image, tokenizer, max_length):
    in_text = 'startseq'
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([image, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = idx_to_word(yhat, tokenizer)
        if word is None:
            break
        in_text += " " + word
        if word == 'endseq':
            break
    return in_text

# Helper function to convert index to word
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

# Route for home page
@app.route('/')
def home():
    return render_template('home.html')
```

```python
    return render_template('home.html')

@app.route('/createdby')
def createdby():
    return render_template('createdby.html')

@app.route('/about')
def about():
    return render_template('about.html')

UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

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


@app.route('/predict', methods=["GET", "POST"])
def predict():
    if request.method == "POST":
        try:
            # Get the uploaded image file
            uploaded_file = request.files['image']

            # Check if the file is allowed
            if uploaded_file and allowed_file(uploaded_file.filenam
                # Save the uploaded image to a temporary location
                filename = secure_filename(uploaded_file.filename)
                temp_path = os.path.join(app.config['UPLOAD_FOLDER'
                uploaded_file.save(temp_path)

                # Preprocess the image for model input
                img_array = preprocess_image(temp_path)
```

```
    return render_template('home.html')

@app.route('/createdby')
def createdby():
    return render_template('createdby.html')

@app.route('/about')
def about():
    return render_template('about.html')

UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

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


@app.route('/predict', methods=["GET", "POST"])
def predict():
    if request.method == "POST":
        try:
            # Get the uploaded image file
            uploaded_file = request.files['image']

            # Check if the file is allowed
            if uploaded_file and allowed_file(uploaded_file.filenam
                # Save the uploaded image to a temporary location
                filename = secure_filename(uploaded_file.filename)
                temp_path = os.path.join(app.config['UPLOAD_FOLDER'
                uploaded_file.save(temp_path)

                # Preprocess the image for model input
                img_array = preprocess_image(temp_path)
```

STEP 7:

prediction.html:

prediction.html is an HTML template used by the Flask application to dynamically generate the prediction page. It likely includes placeholders for displaying the predicted image, caption, and any additional information. This template is rendered by Flask in response to the **/predict** route, providing a user-friendly interface to view the results of the image captioning process.

```
C: > xampp > htdocs > AI&ML APP > code > templates > <> prediction.html > ...
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome to the Prediction Page</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>

    <div class="prediction-section">
        <h2>WELCOME TO THE PREDICTION PAGE</h2>

        <p>UPLOAD THE PICTURE TO GET CAPTION</p>

        <form id="predictionForm" enctype="multipart/form-data" action="{{ url_for('predict') }}" method="post">
            <label for="imageUpload">Upload an Image:</label>
            <input type="file" id="imageUpload" name="image" accept="image/*" required>
            <button type="submit">Predict</button>
        </form>

<body>
        <h2>Predicted Image</h2>

            {% if data.image %}
                <img src="{{ data.image }}" alt="Predicted Image" style="max-width: 100%;">
            {% else %}

            <p>No image to display</p>

          {% endif %}

          <h2>Predicted Caption:</h2>
                    <p>{{ data.caption }}</p>
```

```
                    <p>{{ data.caption }}</p>

    </body>

        <!-- Navigation link at the top -->
        <div class="button">
            <a href="{{ url_for('home') }}">Home</a>
        </div>
      </div>

    </body>
    </html>
```

STEP 8:

styles.css

(CSS) file used to define the visual presentation and styling of HTML elements in the web application. It likely contains rules for styling various components such as fonts, colors, layout, and responsiveness

C: > xampp > htdocs > AI&ML APP > code > templates > # styles.css > body

```css
body {
    font-family: 'Raleway', sans-serif;
    margin: 0;
    padding: 0;
    background: url('background.jpg') no-repeat center center fixed;
    background-size: cover;
    color: #fff;
    line-height: 1.5; /* Adjust line height as needed */
}

header {
    text-align: center;
    padding: 1em 0;
}

.logo-container img {
    max-width: 100%;
    height: auto;
    display: inline-block; /* To prevent extra space below the image */
}


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

nav a {
    text-decoration: none;
    color: #fff;
    padding: 10px 20px;
    margin: 0 10px;
    background-color: #4285f4;
    border: 1px solid #4285f4;
    border-radius: 5px;
    font-family: 'Lato', sans-serif;
```

```
    font-family: 'Lato', sans-serif;
}

.slide {
    padding: 50px 0;
    text-align: center;
    border: 2px solid #4285f4;
    border-radius: 10px;
    margin: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    color: #fff;
}

.button {
    display: inline-block;
    padding: 10px 20px;
    font-size: 16px;
    text-align: center;
    text-decoration: none;
    background-color: #4285f4;
    color: #fff;
    border-radius: 5px;
    cursor: pointer;
    font-family: 'Montserrat', sans-serif;
}

.about-section {
    padding: 50px 20px;
    text-align: left;
    border: 2px solid #4285f4;
    border-radius: 10px;
    margin: 20px;
```

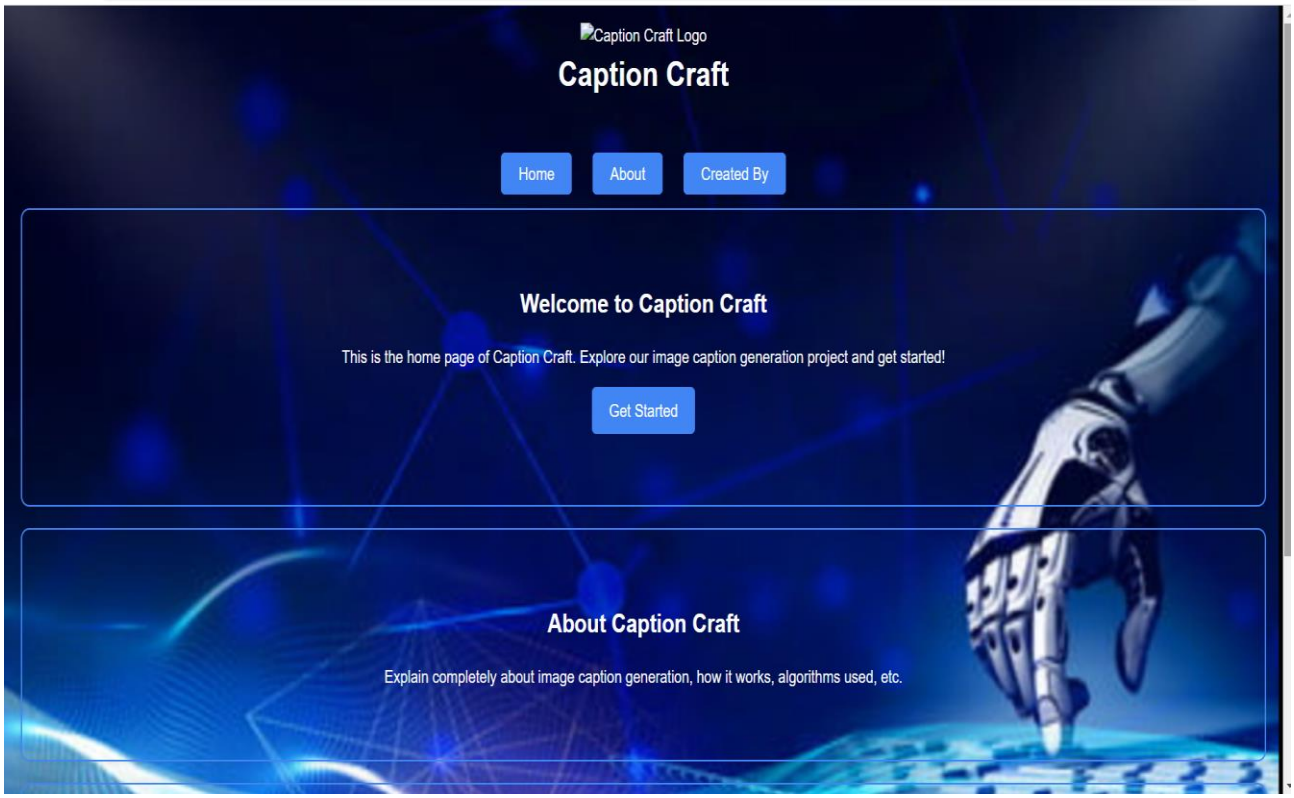
```
    margin: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.about-section h2 {
    color: #fff;
}

.qa-box {
    display: flex;
    flex-direction: column;
    margin-bottom: 20px;
}

.question-box,
.answer-box {
    background-color: #fff;
    padding: 10px;
    border-radius: 5px;
    margin-bottom: 10px;
}

.question-box h3 {
    color: #000;
}

.answer-box p,
.answer-box ol,
.answer-box ul {
    color: #000;
}

.createdby-section {
```

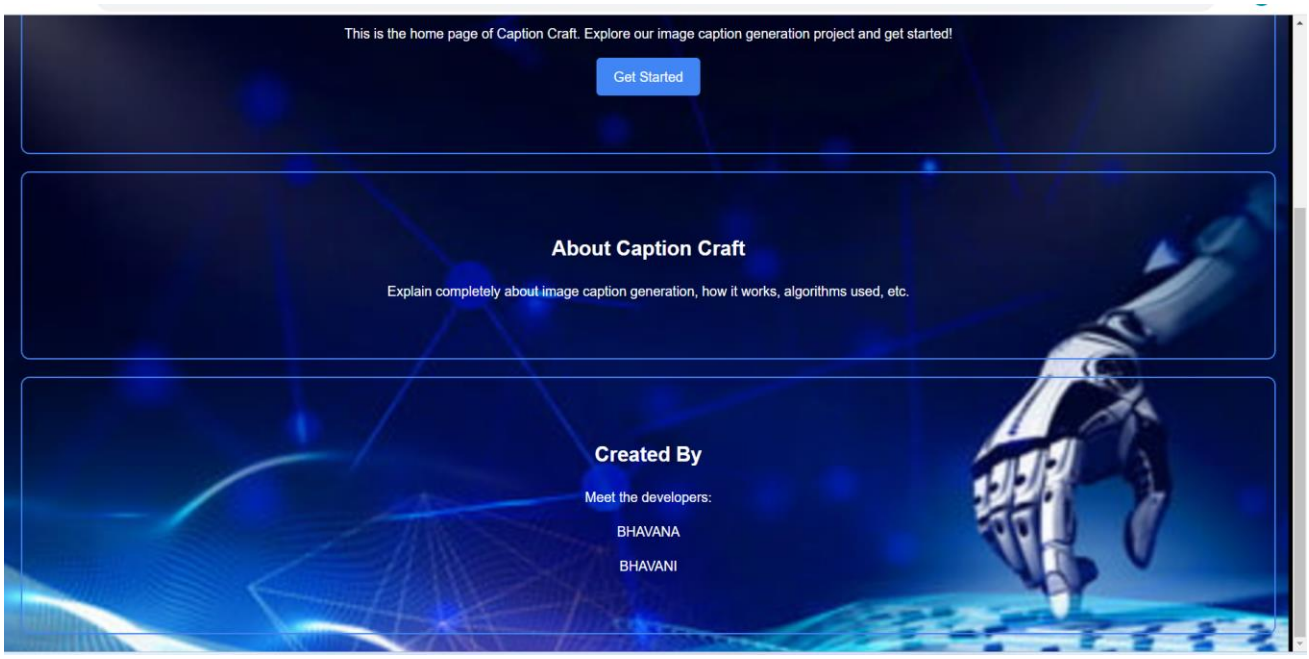
```
}

.createdby-section h2 {
    color: #4285f4;
    font-size: 48px;
    font-family: 'Arial', sans-serif;
    border-bottom: 2px solid #4285f4;
    padding-bottom: 10px;
}

.developer-info {
    margin-top: 30px;
}

.developer-names {
    display: flex;
    justify-content: center;
    align-items: center;
}

.developer-names span {
    color: #fff;
    font-size: 36px;
    font-family: 'Verdana', sans-serif;
    margin: 0 20px;
    border: 2px solid #4285f4;
    padding: 10px;
    border-radius: 10px;
}

.additional-info {
    margin-top: 20px;
    color: #fff;
}

.additional-info p {
```

```css
.additional-info p {
    margin: 10px 0;
}

.prediction-section {
    padding: 100px 20px;
    text-align: center;
    border: 2px solid #4285f4;
    border-radius: 10px;
    margin: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.prediction-section h2 {
    color: #4285f4;
    font-size: 48px;
    font-family: 'Arial', sans-serif;
    border-bottom: 2px solid #4285f4;
    padding-bottom: 10px;
}

.prediction-section p {
    margin-top: 20px;
}

#predictionForm {
    margin-top: 20px;
    display: flex;
    flex-direction: column;
    align-items: center;
}

label {
    color: #fff;
    font-size: 18px;
    margin-bottom: 20px;
```

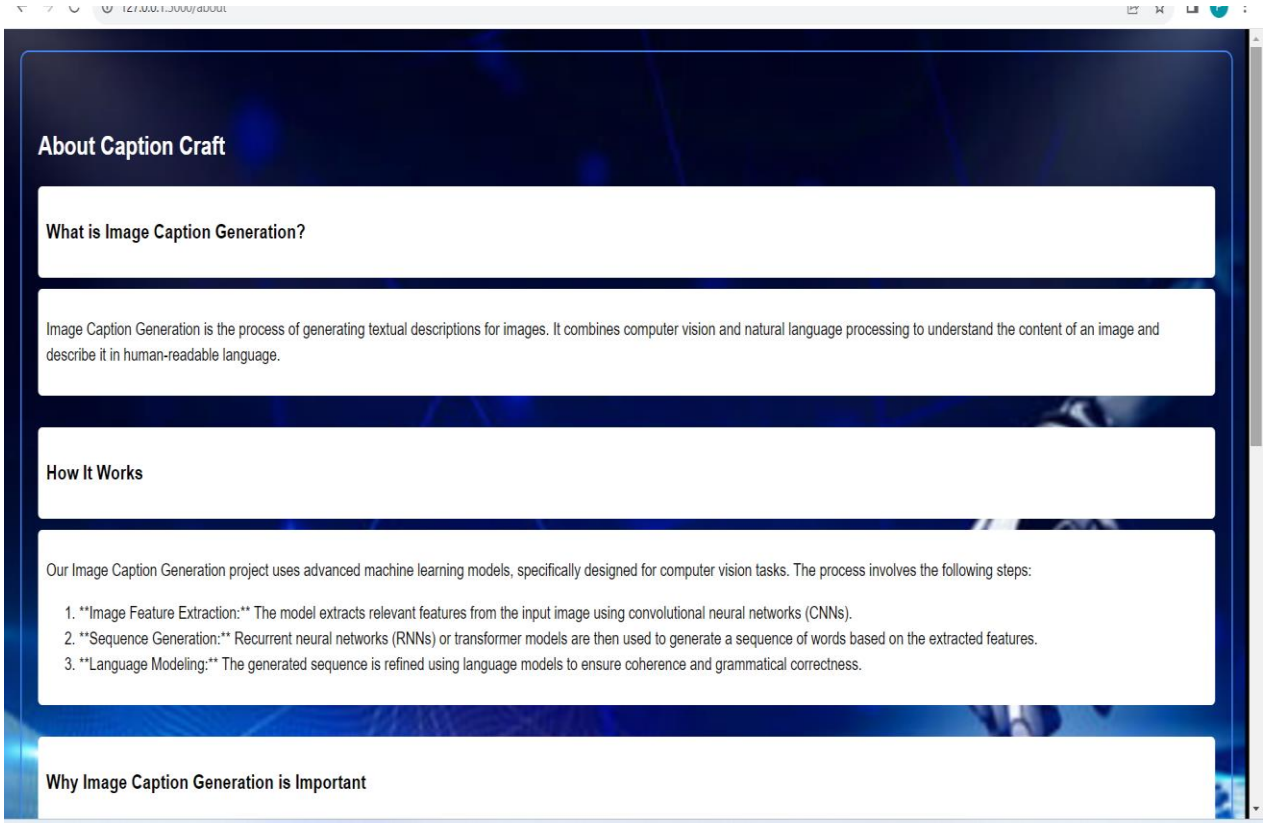
```
170         align-items: center;
171     }
172
173     label {
174         color: #fff;
175         font-size: 18px;
176         margin-bottom: 20px;
177     }
178
179     #imageUpload {
180         margin-top: 10px;
181         margin-bottom: 20px;
182     }
183
184     button {
185         background-color: #4285f4;
186         color: #fff;
187         border: 1px solid #4285f4;
188         padding: 10px 20px
189     }
```

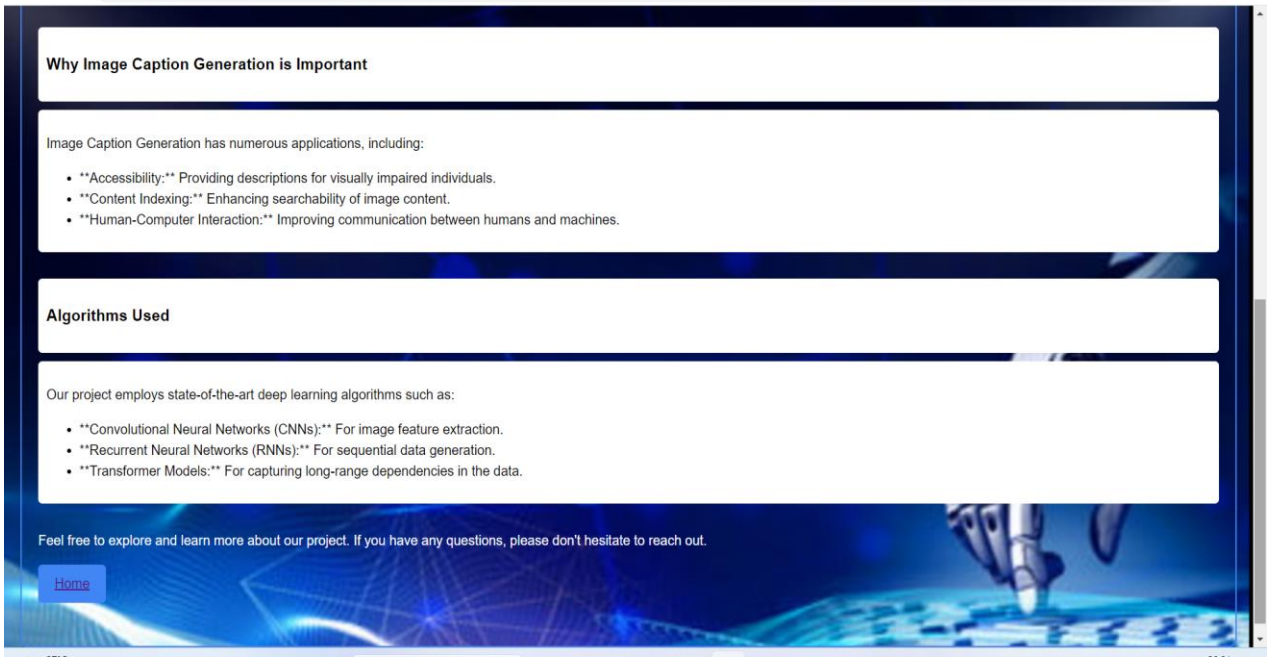
STEP 9:
HOME PAGE :





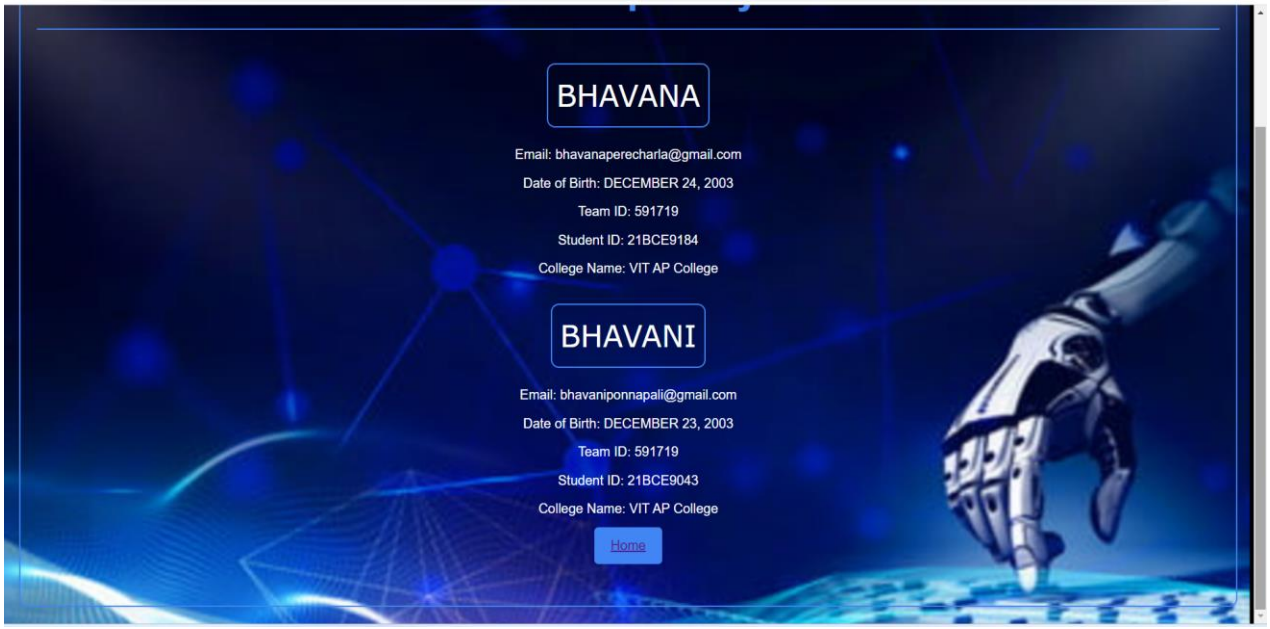
STEP :10
ABOUT PAGE (INFORMATION REGARDING IMAGE CAPTION GENERATION:)





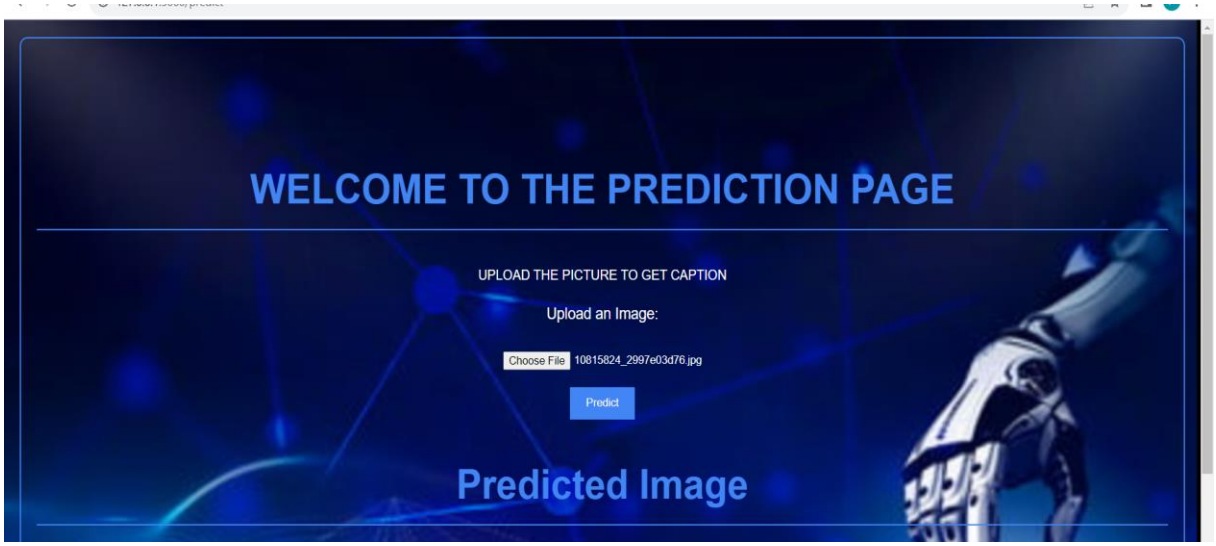
STEP :11

DEVELOPED BY



STEP :12

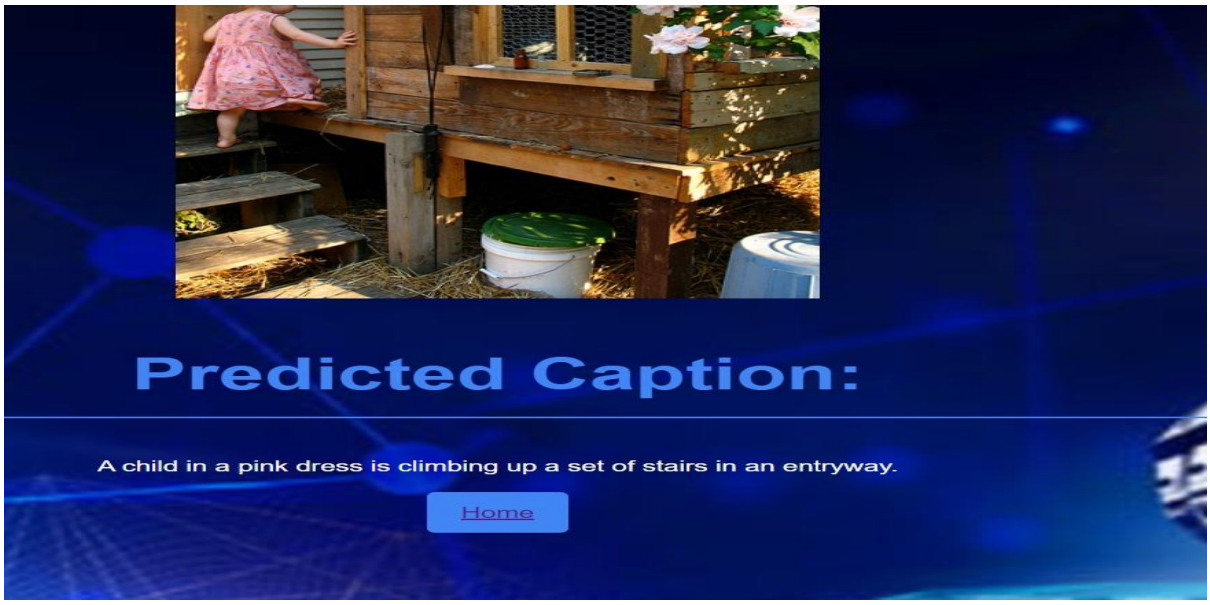
PREDICTION PAGE



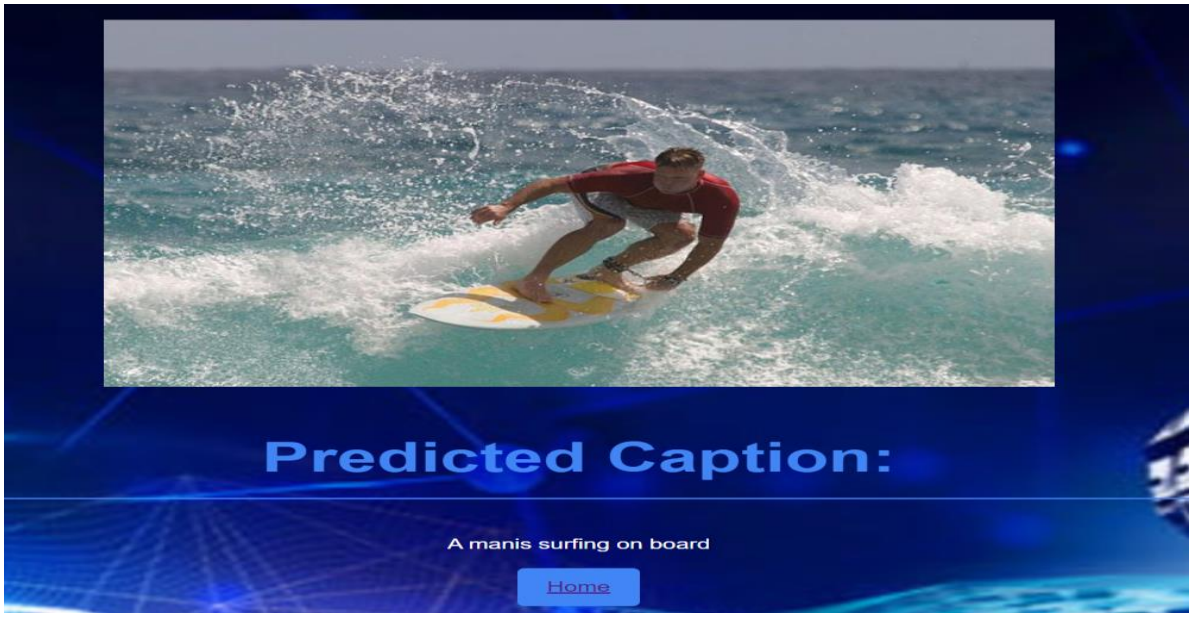
PREDICTION 1:

PREDICTION 2:

PREDICTION 3:



PREDICTION 4:



PREDICTION 5:

❖ THANK YOU