VisionVoice: Accessible Image Captioning with Audio

Name: Nihar Domala
GWID: G34884842

Introduction

VisionVoice is a deep learning application that creates text descriptions for images and turns these captions into audio. Designed for accessibility, it helps visually impaired users understand visual content by combining advanced computer vision and natural language processing technologies.

As a team, we developed the project collaboratively using GitHub to share and manage our work. My primary responsibility was building an efficient data loader to load the images along with their corresponding captions. Once this foundation was established, all team members contributed to designing and training CNN + LSTM architecture models to generate captions. We evaluated the model's performance using the BLEU score metric to assess the quality of the generated captions. Finally, we developed a user-friendly frontend interface using Streamlit to make the application easily accessible.

Description of my Individual work

I initially began by working on data collection and preparation. For this project, we utilized the COCO 2017 dataset, which provides a rich source of annotated images for image captioning tasks. My primary focus was on developing the data loader and vocabulary-generating class. To gain a deeper understanding of dataset preprocessing, I also reviewed several research papers & project repositories, which helped guide our approach. I applied a series of standard image transformations before sending the images to the encoder to ensure consistency and improve model performance. Additionally, I incorporated caption tokenization into the data loader to streamline the training process.

While my teammates focused on the model design, I collaborated with them to explore improvements when initial attempts with custom CNNs and VGG models did not yield satisfactory results. After further exploration, we decided to use a pretrained ResNet50 model as the encoder, paired with an LSTM network with 512 hidden dimensions as the decoder. This approach showed promising results during training.

During my research on image captioning architectures, I discovered the BLIP (Bootstrapped Language-Image Pretraining) model. Since BLIP is a ready-to-use and highly efficient architecture, we decided to integrate it into our project's frontend for additional functionality.

In addition to these tasks, I wrote the "generate caption" functions for each model we experimented with and contributed to integrating all the models into the frontend. This ensured a seamless user experience, enabling users to select and utilize the different model architectures through the application interface.

Detailed description of my contribution

Data Collection

Firstly, I was responsible for data collection. I researched on datasets for image captioning and learnt about standard datasets:

1. MS COCO - Over 330,000 images and 5 captions per image
2. Flikr8k - 8,000 images and 40,000 captions
3. Flikr30k - 30,000 images and over 150,000 captions
4. Visual Genome - Over 100,000 images with detailed annotations
5. Conceptual Captions - 3.3 million images with captions

Among these, I eliminated Flikr8k & Flikr30k as they were too small to capture all kinds of captioning patterns of so many objects existing. The model could only recognize the objects it is trained on and hence gets biased. I eliminated Conceptual captions dataset and it is too huge to train on and we will mot be able to work on it within the project timeframe with given resources. Trading off between Microsoft Common Objects in Context (MS COCO) and Visual Genome, we decided to work on MS COCO dataset 2017.

Preprocessing & loading the data

➔ I started working on creating a COCOCaptionDataset class which initializes vocabulary, images & captions paths and tranformations for the images.
Then, I have defined `*__getitem__*` dunder method to use this class as data loader. In this method, I will get the image from the directory, convert it to RGB format and apply

transformations if given. Then, get it's corresponding captions are loaded, tokenized the captions, made a tensor of the caption.

➔ I created the `*get_data_loader*` function and utilized COCOCaptionDataset class to get the dataset and passed it to torch.utils.data.DataLoader class to utilize it as a data loader for the project with a custom collate function. The purpose of this collate function is to stack the batch of images and pad the tokenized caption.

➔ I developed a Vocabulary class that generates vocab.pkl file from all the captions and includes <start>, <end> and <unk> tokens which are start word, end word and unknown word tokens. It creates 2 dictionaries one mapping each word with the corresponding index and the other mapping each index to the corresponding word. If the vocab.pkl doesn't exist, it creates one, else uses the already existing file as vocabulary.

➔ I researched about standard transformations and made test.py file which shows how to use dataloader using the transformations and displays an image along with it's caption and it's tokenized array just to make sure that dataloader is working fine.

Training

While my teammates worked on model design and implementation, I trained resnet50 & efficientnetb3 models by adding time taken for each epoch in the training loop. Also, helped in debugging some errors in the training file.

Evaluation

I faced some problem of version mismatch while using corpus_bleuscore function from nltk. So, I implemented custom `*compute_bleu*` function along with `*evaluate_model*` function which directly calculates bleu score on validation dataset if you pass the dataloader.

I also created a simple `*generate_and_display_caption*` function just to see how to model is predicting for a given specific image. If an image is passed, it applies necessary transformations and generates caption using the saved model. The image and the caption are displayed.

BLIP

While researching image captioning architectures, I came across the BLIP (Bootstrapped Language-Image Pretraining) model. BLIP is easy to use and highly efficient, so we decided to include it in our project's frontend to enhance its functionality.

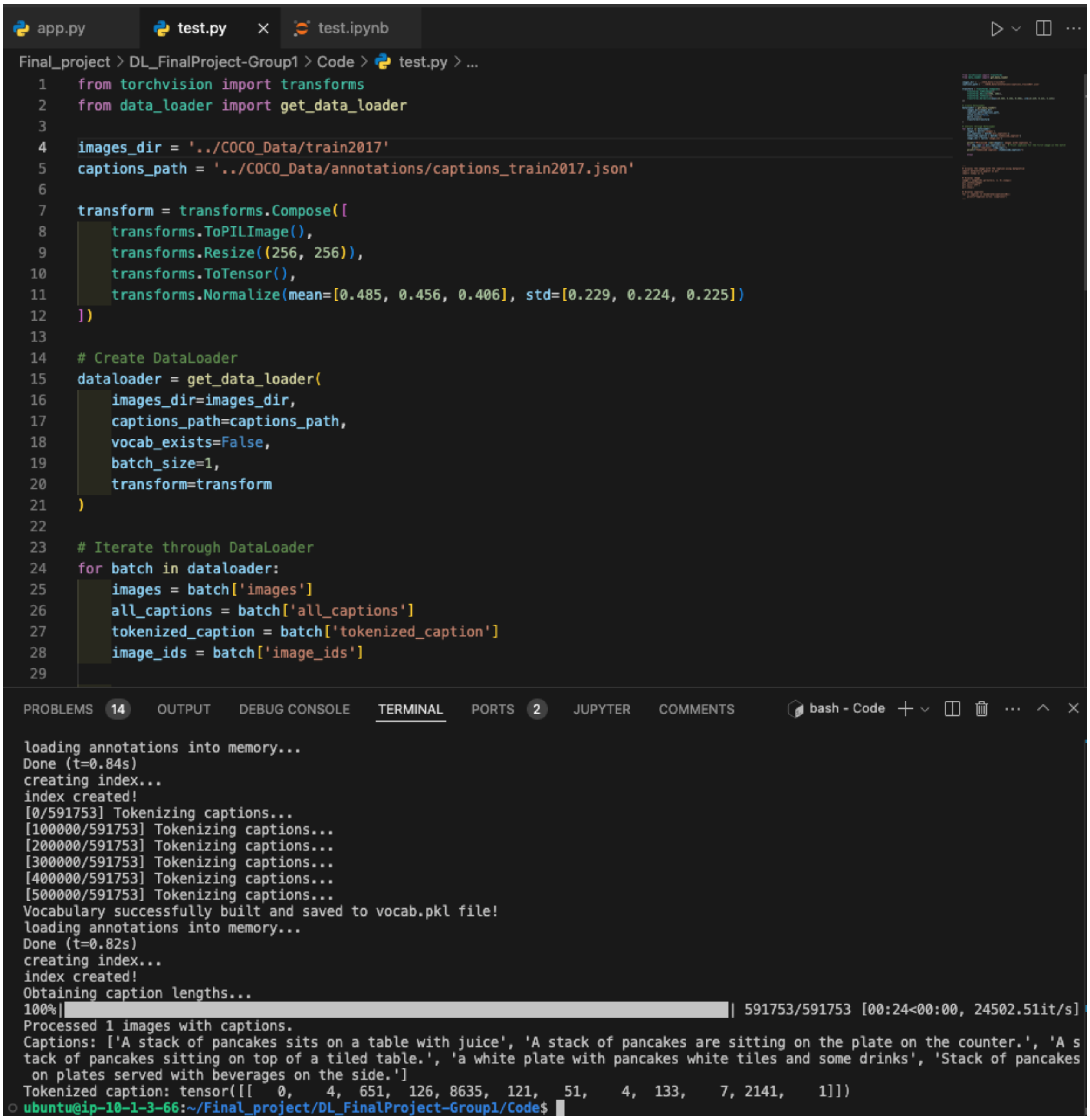
Integrated trained models to Streamlit app

As the models were saved as .pkl & .pth files, I made functions for generating captions (generate_caption_beam.py, generate_caption_blip.py, generate_caption_resnet.py) from each model by loading their architecture & weights. And integrated these functions to the streamlit interface by giving users an option to choose the model for generating the caption.

Results

Dataset preparation and validation

To ensure the data preparation process was effective, I tested the data loader and vocabulary generation functionalities. Using a sample from the MS COCO 2017 dataset, I confirmed that the data loader correctly applied transformations, tokenized captions, and padded sequences as expected. It is in test.py code file.



```python
from torchvision import transforms
from data_loader import get_data_loader

images_dir = '../COCO_Data/train2017'
captions_path = '../COCO_Data/annotations/captions_train2017.json'

transform = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Create DataLoader
dataloader = get_data_loader(
    images_dir=images_dir,
    captions_path=captions_path,
    vocab_exists=False,
    batch_size=1,
    transform=transform
)

# Iterate through DataLoader
for batch in dataloader:
    images = batch['images']
    all_captions = batch['all_captions']
    tokenized_caption = batch['tokenized_caption']
    image_ids = batch['image_ids']

```

```
loading annotations into memory...
Done (t=0.84s)
creating index...
index created!
[0/591753] Tokenizing captions...
[100000/591753] Tokenizing captions...
[200000/591753] Tokenizing captions...
[300000/591753] Tokenizing captions...
[400000/591753] Tokenizing captions...
[500000/591753] Tokenizing captions...
Vocabulary successfully built and saved to vocab.pkl file!
loading annotations into memory...
Done (t=0.82s)
creating index...
index created!
Obtaining caption lengths...
100%|████████████████████████████████████████| 591753/591753 [00:24<00:00, 24502.51it/s]
Processed 1 images with captions.
Captions: ['A stack of pancakes sits on a table with juice', 'A stack of pancakes are sitting on the plate on the counter.', 'A s
tack of pancakes sitting on top of a tiled table.', 'a white plate with pancakes white tiles and some drinks', 'Stack of pancakes
 on plates served with beverages on the side.']
Tokenized caption: tensor([[   0,    4,  651,  126, 8635,  121,   51,    4,  133,    7, 2141,    1]])
ubuntu@ip-10-1-3-66:~/Final_project/DL_FinalProject-Group1/Code$
```

When the vocab_exists argument is set to false, new vocabulary is also getting created as expected. So, overall the data loader is functioning properly without any issues.

Model Training and Evaluation

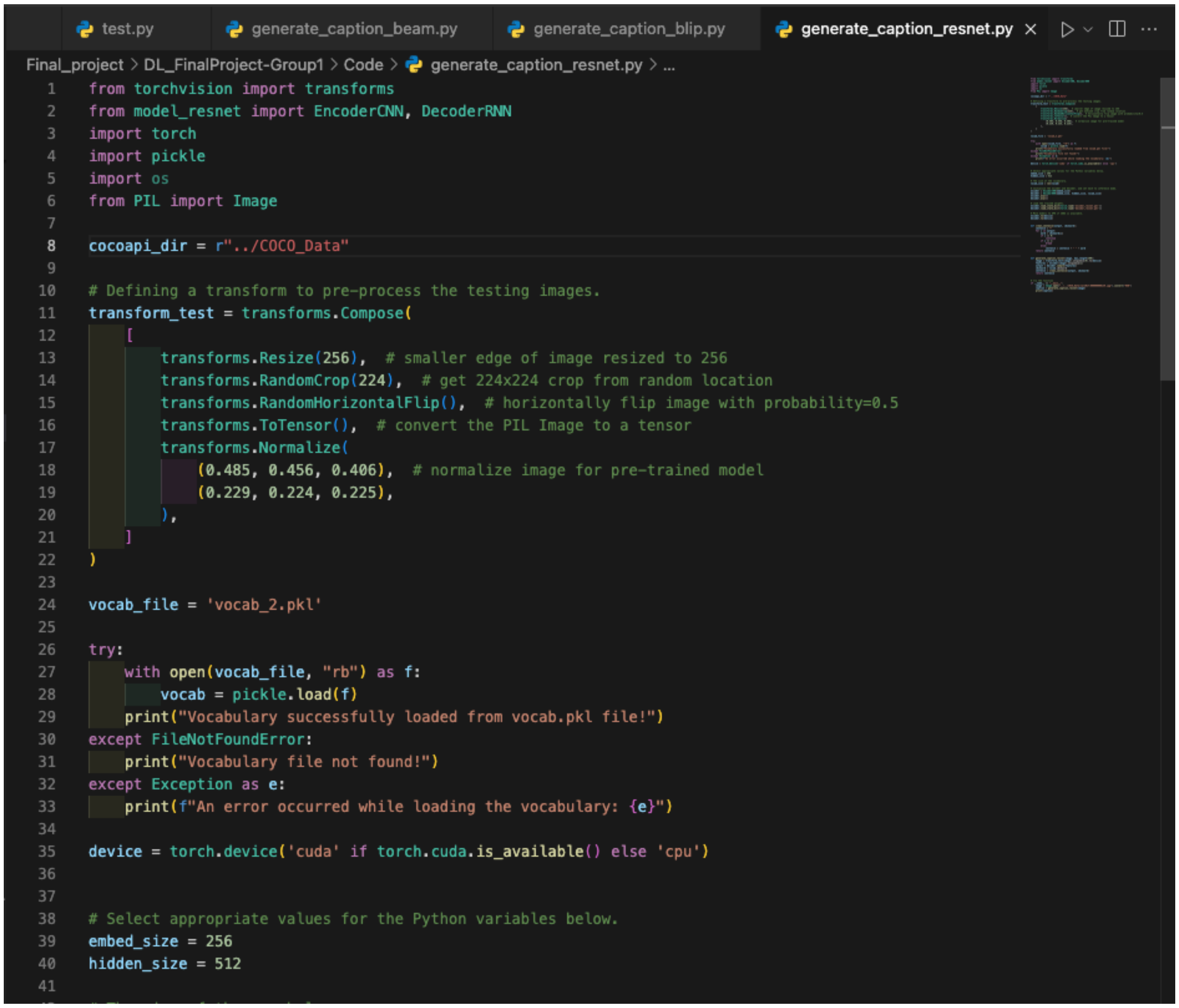
I trained ResNet50 and EfficientNetB3 as encoder models combined with an LSTM decoder by adding snippet to time per epoch. The models were evaluated based on their BLEU score written by me, a standard metric for image captioning.

Table 1: BLEU scores and training time for ResNet50 and EfficientNetB3 models

Model	BLEU-4	Time per epoch (secs)
ResNet50 + LSTM	0.1487	3417.43
EfficientNetB3 + LSTM	0.2002	4408.34

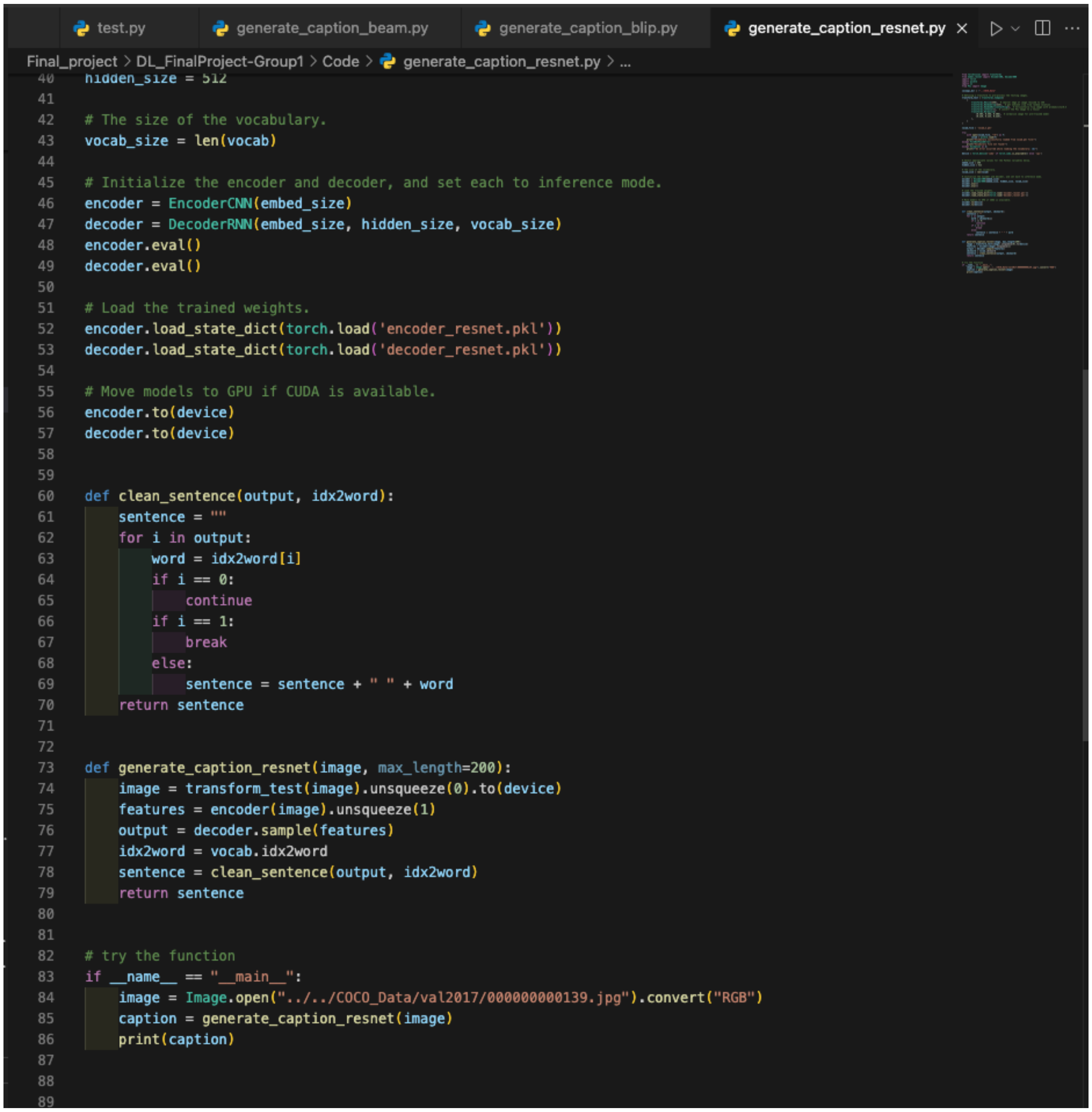
Frontend Integration Results

The generate captions functions code snippets are below:



test.py
generate_caption_beam.py
generate_caption_blip.py
generate_caption_resnet.py
Final_project > DL_FinalProject-Group1 > Code > generate_caption_resnet.py > ...

```python
from torchvision import transforms
from model_resnet import EncoderCNN, DecoderRNN
import torch
import pickle
import os
from PIL import Image

cocoapi_dir = r"../COCO_Data"

# Defining a transform to pre-process the testing images.
transform_test = transforms.Compose(
    [
        transforms.Resize(256),  # smaller edge of image resized to 256
        transforms.RandomCrop(224),  # get 224x224 crop from random location
        transforms.RandomHorizontalFlip(),  # horizontally flip image with probability=0.5
        transforms.ToTensor(),  # convert the PIL Image to a tensor
        transforms.Normalize(
            (0.485, 0.456, 0.406),  # normalize image for pre-trained model
            (0.229, 0.224, 0.225),
        ),
    ]
)

vocab_file = 'vocab_2.pkl'

try:
    with open(vocab_file, "rb") as f:
        vocab = pickle.load(f)
    print("Vocabulary successfully loaded from vocab.pkl file!")
except FileNotFoundError:
    print("Vocabulary file not found!")
except Exception as e:
    print(f"An error occurred while loading the vocabulary: {e}")

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')


# Select appropriate values for the Python variables below.
embed_size = 256
hidden_size = 512

```

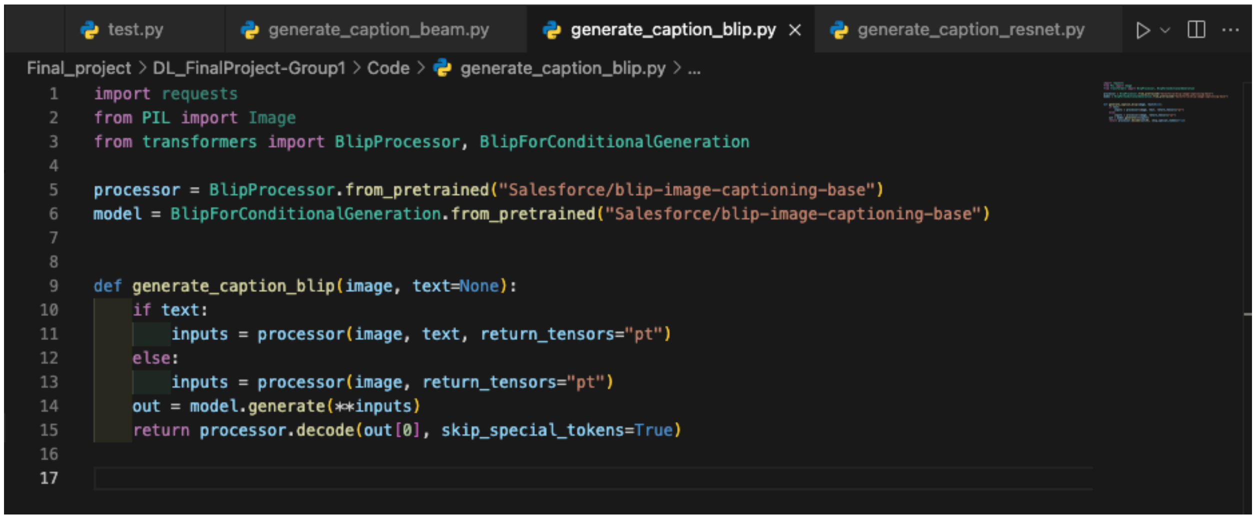


test.py
generate_caption_beam.py
generate_caption_blip.py
generate_caption_resnet.py
Final_project > DL_FinalProject-Group1 > Code > generate_caption_resnet.py > ...

```python
hidden_size = 512

# The size of the vocabulary.
vocab_size = len(vocab)

# Initialize the encoder and decoder, and set each to inference mode.
encoder = EncoderCNN(embed_size)
decoder = DecoderRNN(embed_size, hidden_size, vocab_size)
encoder.eval()
decoder.eval()

# Load the trained weights.
encoder.load_state_dict(torch.load('encoder_resnet.pkl'))
decoder.load_state_dict(torch.load('decoder_resnet.pkl'))

# Move models to GPU if CUDA is available.
encoder.to(device)
decoder.to(device)


def clean_sentence(output, idx2word):
    sentence = ""
    for i in output:
        word = idx2word[i]
        if i == 0:
            continue
        if i == 1:
            break
        else:
            sentence = sentence + " " + word
    return sentence


def generate_caption_resnet(image, max_length=200):
    image = transform_test(image).unsqueeze(0).to(device)
    features = encoder(image).unsqueeze(1)
    output = decoder.sample(features)
    idx2word = vocab.idx2word
    sentence = clean_sentence(output, idx2word)
    return sentence


# try the function
if __name__ == "__main__":
    image = Image.open("../../COCO_Data/val2017/000000000139.jpg").convert("RGB")
    caption = generate_caption_resnet(image)
    print(caption)
```

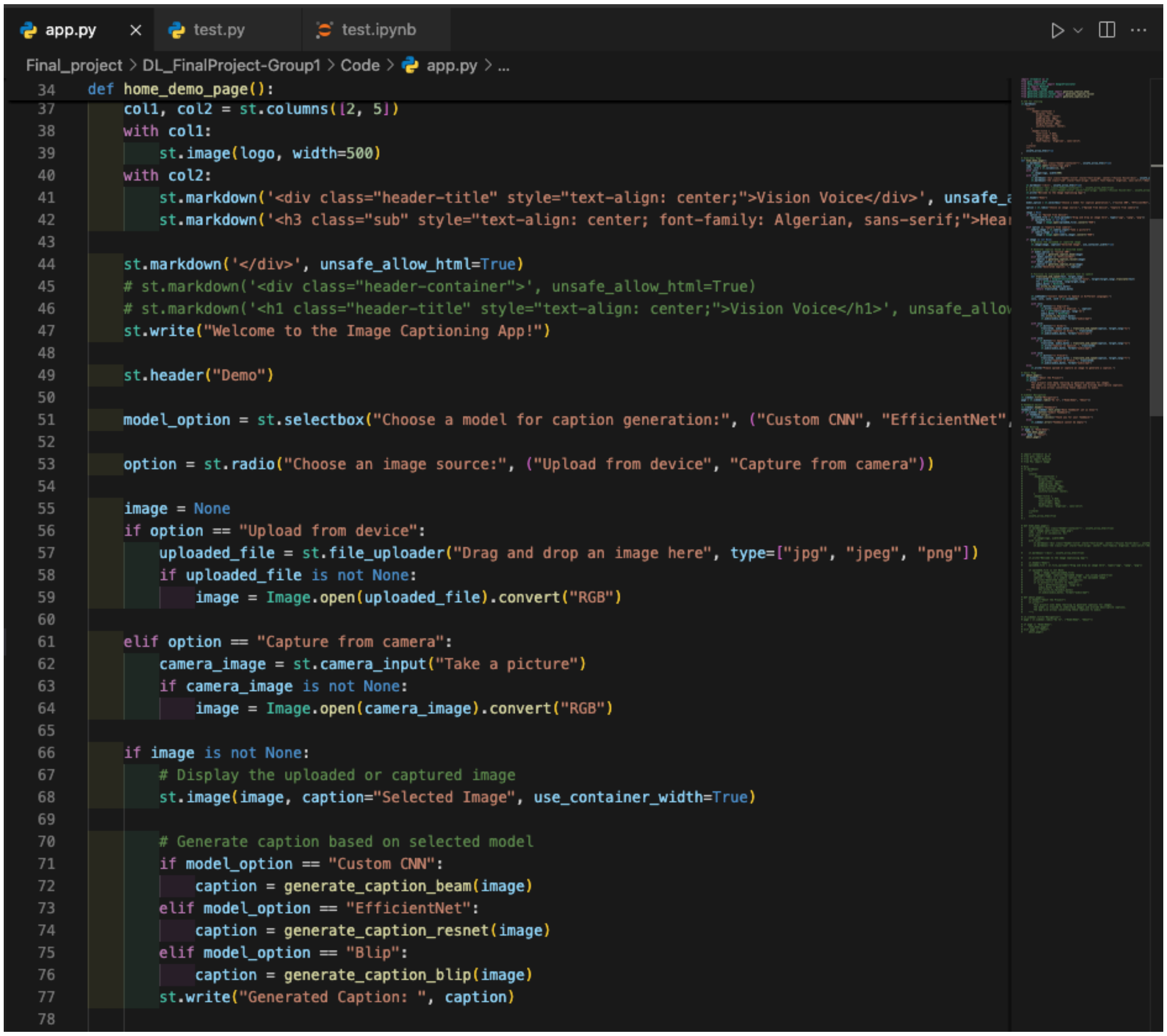
This is one of the implementation, similarly other models are also integrated.

For BLIP model, it is simply to load the pretrained model, it's implementation is below:


test.py
generate_caption_beam.py
generate_caption_blip.py
generate_caption_resnet.py
Final_project > DL_FinalProject-Group1 > Code > generate_caption_blip.py > ...

```python
import requests
from PIL import Image
from transformers import BlipProcessor, BlipForConditionalGeneration

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")


def generate_caption_blip(image, text=None):
    if text:
        inputs = processor(image, text, return_tensors="pt")
    else:
        inputs = processor(image, return_tensors="pt")
    out = model.generate(**inputs)
    return processor.decode(out[0], skip_special_tokens=True)

```

The generate caption functions are simply imported in the streamlit app.py code and uploaded image is passed to it as below:


app.py
test.py
test.ipynb
Final_project > DL_FinalProject-Group1 > Code > app.py > ...

```
def home_demo_page():
    col1, col2 = st.columns([2, 5])
    with col1:
        st.image(logo, width=500)
    with col2:
        st.markdown('<div class="header-title" style="text-align: center;">Vision Voice</div>', unsafe_a
        st.markdown('<h3 class="sub" style="text-align: center; font-family: Algerian, sans-serif;">Hea

    st.markdown('</div>', unsafe_allow_html=True)
    # st.markdown('<div class="header-container">', unsafe_allow_html=True)
    # st.markdown('<h1 class="header-title" style="text-align: center;">Vision Voice</h1>', unsafe_allo
    st.write("Welcome to the Image Captioning App!")

    st.header("Demo")

    model_option = st.selectbox("Choose a model for caption generation:", ("Custom CNN", "EfficientNet",

    option = st.radio("Choose an image source:", ("Upload from device", "Capture from camera"))

    image = None
    if option == "Upload from device":
        uploaded_file = st.file_uploader("Drag and drop an image here", type=["jpg", "jpeg", "png"])
        if uploaded_file is not None:
            image = Image.open(uploaded_file).convert("RGB")

    elif option == "Capture from camera":
        camera_image = st.camera_input("Take a picture")
        if camera_image is not None:
            image = Image.open(camera_image).convert("RGB")

    if image is not None:
        # Display the uploaded or captured image
        st.image(image, caption="Selected Image", use_container_width=True)

        # Generate caption based on selected model
        if model_option == "Custom CNN":
            caption = generate_caption_beam(image)
        elif model_option == "EfficientNet":
            caption = generate_caption_resnet(image)
        elif model_option == "Blip":
            caption = generate_caption_blip(image)
        st.write("Generated Caption: ", caption)
```

Conclusion and future scope

The VisionVoice project successfully developed an accessible image captioning application that generates descriptive text for images and converts it to audio, benefiting visually impaired users. The project utilized the MS COCO dataset and combined CNN and LSTM networks to produce captions, with a user-friendly Streamlit interface for easy access.

Future improvements could focus on:

- Enhancing vocabulary management
- Optimizing model architecture for faster training

- Exploring additional datasets
- Expanding to include video captioning or real-time image analysis
- Produce automatic speech output and integrate it as a chatbotlike Siri

By addressing these challenges, VisionVoice can become an even more effective tool for enhancing accessibility for visually impaired users

References

➔ https://pytorch.org/docs/stable/data.html
➔ https://github.com/iamirmasoud/image_captioning
➔ https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/
➔ https://medium.com/@lucrece.shin/chapter-3-transfer-learning-with-resnet50-from-dataloaders-to-training-seed-of-thought-67aaf83155bc
➔ https://docs.streamlit.io/develop/api-reference/widgets/st.selectbox
➔ https://chatgpt.com/

Percentage of Code Sourced Online

Total lines – 578

Borrowed from external – 220

Modified – 105

Added – 30

Percentage = (220 - 105 / 220 + 30) * 100 = 46%