THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Yash Manish Doshi

G33250233

Final Project Report

1. Introduction

Visual content is a dominant medium for communication and interaction in the modern world. However, for visually impaired individuals, this reliance on visual information creates barriers that hinder access to critical aspects of daily life and digital interactions. The Vision Voice project seeks to tackle this challenge by developing an accessible system that transforms visual inputs into meaningful audio outputs, combining cutting-edge AI technologies to address this real-world need.

This project involves the integration of image captioning models to describe images and text-to-speech (TTS) modules to provide audio descriptions. As part of the development process various neural network architectures were explored to check the effectiveness in generating accurate and relevant captions. Experiments with models like ResNet50, ResNet101, and VGG16 highlighted the trade-offs between computational complexity and accuracy, ultimately guiding the team toward informed design decisions.

The text-to-speech (TTS) module plays a crucial role in ensuring that the generated captions are accessible to users. By converting textual descriptions into clear and natural-sounding audio, the TTS system provides a seamless auditory experience. Furthermore, the integration of multilingual capabilities allows captions to be translated into multiple languages which can help users from diverse linguistic backgrounds to benefit from the application. These features make Vision Voice a versatile tool that breaks language barriers and significantly enhancing its accessibility and usability for a variety audience.

This report provides an in-depth look into the methodologies, challenges, and outcomes of these contributions. It focuses on the iterative experimentation and problem-solving processes that underpin Vision Voice's development, showcasing how AI can be utilized to bridge the accessibility gap in innovative ways.

2. Shared Work and Collaboration

The Vision Voice project involved teamwork across several areas, such as dataset preparation, model experimentation, system integration, and application enhancements. Some of the significant contributions to the shared work include:

2.1. Dataset Selection

- Researched and evaluated various datasets to identify those with comprehensive captions and diverse image categories.
- Analyzed dataset suitability based on size, quality of captions, and alignment with project goals.
- Finalized the COCO 2017 dataset for its richness in annotations and widespread use in image captioning research.

2.2. Experimentation with CNN Architectures

- Conducted experiments with models like Custom CNN, ResNet50, ResNet101, and VGG16 to assess their suitability for the encoder in the image captioning pipeline.
- Compared models based on performance metrics, computational efficiency, and their ability to extract meaningful features.
- Contributed insights from these experiments, guiding the team to select ResNet50 for its balance between accuracy and processing time.

2.3. Application Modifications and Text-to-Speech (TTS) Integration

- Worked on refining the app interface to ensure it is user-friendly and intuitive for visually impaired users.
- Added various features in order to make the app user friendly and more dynamic
- Implemented multilingual support in the TTS system, enabling captions to be translated and delivered in multiple languages.
- Optimized the TTS module to produce clear and natural-sounding audio, enhancing the overall accessibility of the system.

3. Exploring Different Models

3.1. Custom CNN on a Smaller Dataset

In order to build a foundational knowledge of image captioning an initial experimentation was done using a custom CNN architecture on a smaller Flickr8k dataset. This approach aimed to test a simplified model pipeline and validate the concept of combining CNN for feature extraction with an RNN for caption generation. A smaller dataset was used in order to get the intial idea of the working of the models.

- The simplicity of the architecture made it easier to implement providing a hands-on understanding of how image features are extracted.
- Using a smaller dataset like Flickr8k allowed for faster experimentation without requiring extensive computational resources but at the same time the model did not have more data to explore.
- The custom CNN struggled to capture complex visual features due to its shallow architecture.
- The small dataset and the relatively few convolutional layers led to overfitting during training.
- It is difficult to come to a specific custom CNN model as it is difficult to find the right number of layers which indicated the need for a much better feature extraction.

3.2. VGG16

VGG16 was the next model explored, chosen for its proven track record in feature extraction tasks. Its hierarchical structure of convolutional layers promised better feature representation compared to the custom CNN.

- Its simplicity and effectiveness in extracting features from images with smaller filters (3x3 convolutions).
- It’s a well structured architecture with widespread use in computer vision tasks.
- Despite its simplicity, VGG16 required more computational resources than anticipated for training as it generates a large number of parameters.
- The large number of parameters generated caused overfitting when fine-tuning the model on the COCO dataset.
- While VGG16 captured basic features effectively, it struggled with complex and abstract visual concepts, affecting caption quality.

These limitations faced due to VGG16 highlighted the need for a more modern architecture with better feature extraction capabilities and computational efficiency.

3.3 ResNet50

In order to explore better models than Custom CNN and VGG16. ResNet50 emerged as the optimal choice for the encoder in the image captioning pipeline. Its balance of depth, efficiency, and feature extraction capabilities made it the much suitable architecture for the task.

- A moderate depth with 50 layers provided a good trade-off between computational efficiency and feature richness.
- The residual connections ensured smooth training and improved gradient flow.
- Seamless integration with the decoder resulted in consistent and stable training outcomes.

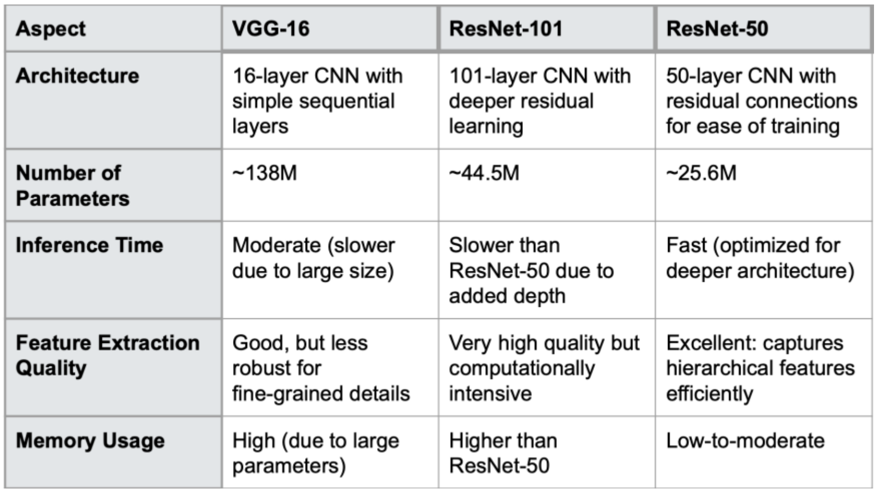
3.4 ResNet101

I explored ResNet101 for its deeper architecture. This was because ResNet50 worked better on the dataset. So in order to explore for a better and deeper model, ResNet101 was used to mitigate vanishing gradient issues and improve training efficiency. But ResNet101 has its own issues.

- The use of skip connections allowed for better gradient flow, enabling the model to learn deeper features.
- Its 101-layer depth offered the potential to capture more deeper visual patterns and relationships.

- The deep architecture led to higher training times and computational demands which makes even 1 epoch run for significantly higher amount of time.
- The performance improvement over simpler architectures like ResNet50 were marginal, making the additional complexity unjustifiable.
- ResNet101's depth complicated its integration with the decoder, leading to unstable training.

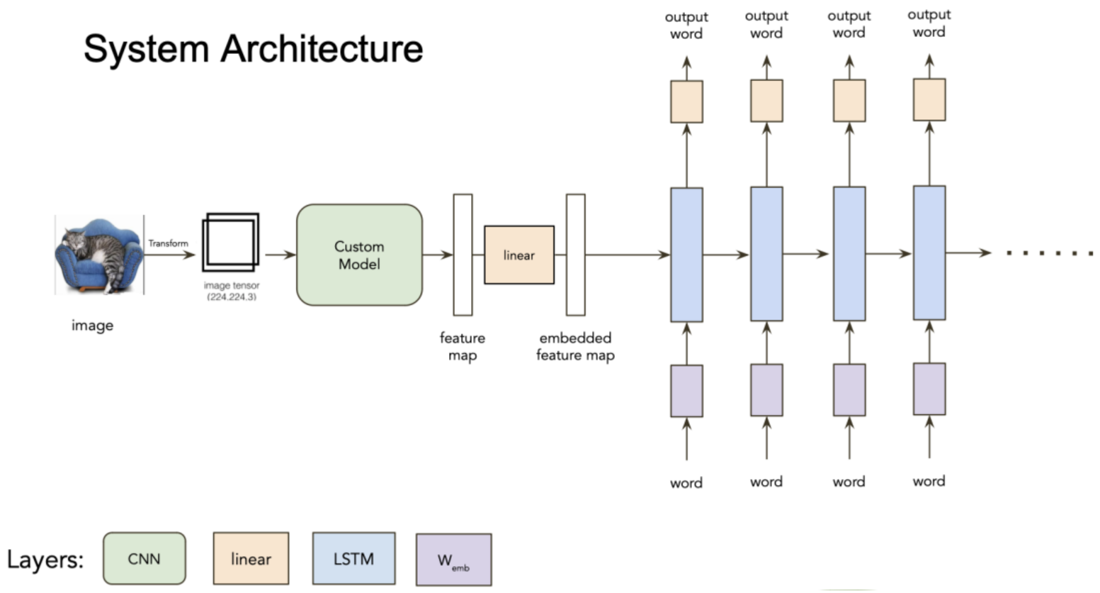
This iterative process of evaluating different architectures demonstrated the importance of balancing computational efficiency with feature extraction depth. By experimenting with diverse models, I gained valuable insights into the trade-offs between simplicity, performance, and usability which led me to select ResNet50 as a better option for the image captioning system.



Aspect	VGG-16	ResNet-101	ResNet-50
Architecture	16-layer CNN with simple sequential layers	101-layer CNN with deeper residual learning	50-layer CNN with residual connections for ease of training
Number of Parameters	~138M	~44.5M	~25.6M
Inference Time	Moderate (slower due to large size)	Slower than ResNet-50 due to added depth	Fast (optimized for deeper architecture)
Feature Extraction Quality	Good, but less robust for fine-grained details	Very high quality but computationally intensive	Excellent: captures hierarchical features efficiently
Memory Usage	High (due to large parameters)	Higher than ResNet-50	Low-to-moderate

A comparison between all the models

4. ResNet50 (System Architecture and Code breakdown)



4.1 system architecture

4.1. Vocabulary Class

The Vocabulary class handles the required text processing required for caption generation. It maps words to unique indices for machine processing and provides reverse mapping for readability. It also supports tokenization of captions into sequence of integers and detokenization of sequences back into words.

- build_vocab(captions): Constructs a vocabulary by assigning unique indices to words from a list of captions.
- tokenize(caption): Converts a caption string into a sequence of integers using the word-to-index mapping.
- detokenize(tokens): Converts a sequence of integers back into a human-readable caption.

4.2 Dataset Class

The dataset class help in the loading and preprocessing of the COCO dataset making sure that the input data is ready for training and evaluation.

- It handles the image caption pairs for training and validation.
- Prepares images and tokenized captions for input to the model.
- Loads captions and image metadata from a JSON file.
- Applies preprocessing various transformations images.
- It tokenizes captions and adds <start> and <end> tokens to point the start and end of sequences.

4.3. Collate Function

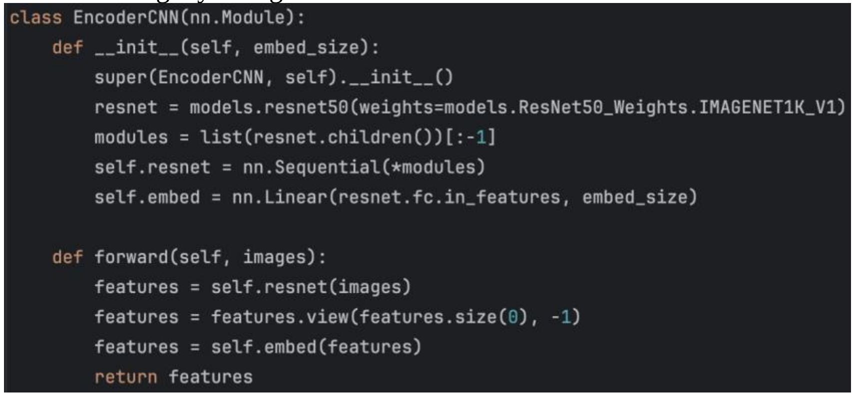
The collate function manages batching for the data loader, ensuring compatibility with the model's requirements.

- Aligns images and captions within each batch for consistent processing.
- Pads captions to handle variable-length sequences.
- Stacks images into a tensor for batch processing.
- Pads captions to the maximum length in the batch for uniformity.
- Returns processed images, padded captions, and their respective lengths.

4.4 EncoderCNN

The encoder is the image feature extraction module in the image captioning pipeline. It processes the input image into a compact feature vector that represents the high-level information needed by the decoder to generate captions. It Extracts significant features from input images. It Reduces the dimensionality of image data while saving the critical information required for caption generation. It Prepares features into a format compatible with the decoder.
The encoder uses a pre-trained ResNet50 model from PyTorch model leveraging weights trained on the ImageNet dataset. The fully connected layer of ResNet50 gets removed only the convolutional layers are retained to extract spatial and semantic features.

After extracting features a custom linear fully connected layer is added to project the ResNet50 output into a lower-dimensional embedding space (size = embed_size). This dimensionality reduction ensures compatibility with the decoder while minimizing computational overhead.
The input image is passed through the ResNet50 model to produce a feature map. The feature map is flattened and fed through the embedding layer to generate the embedded feature vector.



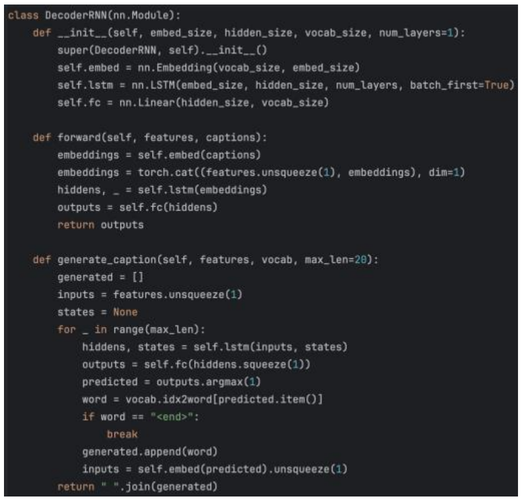
```
class EncoderCNN(nn.Module):
    def __init__(self, embed_size):
        super(EncoderCNN, self).__init__()
        resnet = models.resnet50(weights=models.ResNet50_Weights.IMAGENET1K_V1)
        modules = list(resnet.children())[:-1]
        self.resnet = nn.Sequential(*modules)
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)

    def forward(self, images):
        features = self.resnet(images)
        features = features.view(features.size(0), -1)
        features = self.embed(features)
        return features
```

4.5 DecoderRNN

The DecoderRNN is the language generation module that takes the image feature vector from the encoder and sequentially generates a descriptive caption. Decodes the image feature vector into a coherent sequence of words. Maintains context across word predictions using sequential processing.

Converts each word token in the caption into a dense vector representation. This layer maps vocabulary indices to dense embeddings, allowing the decoder to process the input more effectively. The LSTM's output at each timestep is passed through a fully connected layer to predict the next word. The output is a probability distribution over the entire vocabulary, indicating the likelihood of each word being the next in the sequence. LSTM is used in the decoder because they handle sequential data, making them ideal for processing captions where the context of each word depends on previous ones. LSTM overcome the vanishing gradient problem through their gating mechanisms which help in learning across sentences of variable length.
At training time the decoder receives the actual tokenized caption and predicts the next word for each timestep. At inference time the decoder starts with the <start> token and predicts the next word iteratively. The predicted word is fed back as input for the next timestep until the <end> token is reached, or the maximum caption length is exceeded.



```
class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(DecoderRNN, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, features, captions):
        embeddings = self.embed(captions)
        embeddings = torch.cat((features.unsqueeze(1), embeddings), dim=1)
        hiddens, _ = self.lstm(embeddings)
        outputs = self.fc(hiddens)
        return outputs

    def generate_caption(self, features, vocab, max_len=20):
        generated = []
        inputs = features.unsqueeze(1)
        states = None
        for _ in range(max_len):
            hiddens, states = self.lstm(inputs, states)
            outputs = self.fc(hiddens.squeeze(1))
            predicted = outputs.argmax(1)
            word = vocab.idx2word[predicted.item()]
            if word == "<end>":
                break
            generated.append(word)
            inputs = self.embed(predicted).unsqueeze(1)
        return " ".join(generated)
```

4.6 Training Function

The train_model function is the one where everything boils down. This function optimizes the encoder and decoder to minimize the error between the predicted captions and the ground truth. For each batch of data, the encoder extracts image features, and the decoder generates captions. The predicted captions are compared to the actual captions using cross-entropy loss, a measure of how far the predictions is from the truth. Gradients are calculated and backpropagated to update the model's weights, improving its performance over time.
Training is an iterative process that requires patience and fine-tuning, but it's essential for building a robust model.

5. StreamLit and Application Overview

The Vision Voice application is an accessible tool developed to generate image captions and provide an auditory output for visually impaired users. It integrates advanced deep learning techniques, multilingual text-to-speech capabilities, and an interactive user interface making it an innovative solution for bridging the gap between visual and auditory content.
Key Features:

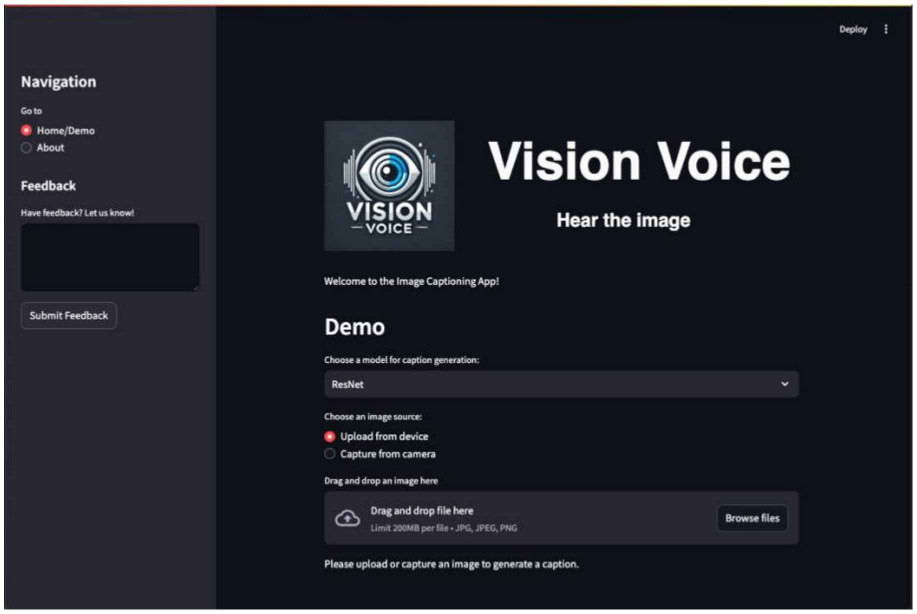
1. The app utilizes state-of-the-art models like Custom CNN, EfficientNet (with beam search), and Blip to generate descriptive captions for images.
2. Users can upload images or can directly capture them using their camera for instant caption generation.
3. Generated captions are converted into clear and natural-sounding audio helping visually impaired users.
4. The app supports caption translation and audio output in multiple languages, catering to a diverse, global audience.
5. Users can provide feedback directly within the app to suggest improvements or report issues.

The application is built using Streamlit, a Python framework for developing interactive web applications. Streamlit provides a lightweight, easy-to-use environment that enables quick development and deployment of machine learning models with a user friendly interface. It allows smooth integration of deep learning models and external APIs like gTTS for TTS and Google Translate for multilingual support. Provides real-time updates and responsiveness for image uploads, camera inputs, and caption generation. Enables developers to use HTML/CSS for enhanced UI design ensuring a visually appealing and intuitive layout.

Key Streamlit Features in the Application:

- Users can upload images or capture them directly using their webcam.
- The st.file_uploader and st.camera_input components handle these inputs seamlessly.
- A dropdown menu lets you choose between different caption generation models, such as Custom CNN, EfficientNet, and Blip.
- Captions can be converted into audio with a single button click, thanks to the st.audio component.

- ○ Navigation between the home/demo page and the about page is streamlined using a sidebar.
- ○ Feedback can also be submitted directly from the sidebar.



Welcome to the Image Captioning App!

Demo

Choose a model for caption generation: ResNet

Choose an image source: [✓ Upload from device] Capture from camera

Drag and drop an image here

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG
Browse files

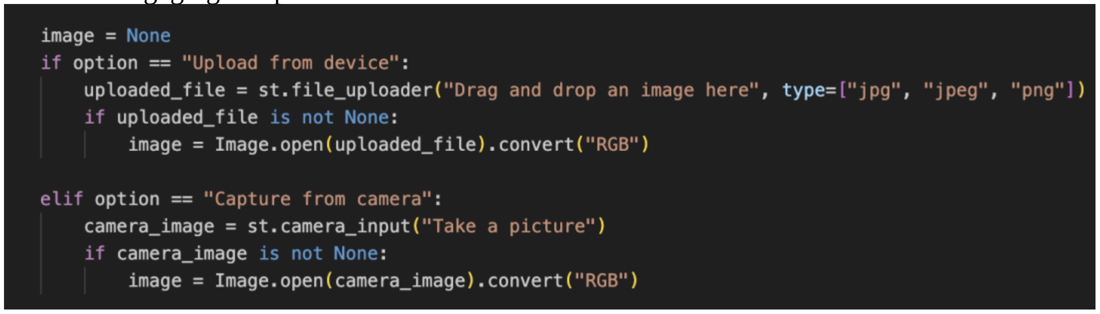
Please upload or capture an image to generate a caption.

5.1 My Contributions to the Application

My contributions to the development of the Vision Voice app focused on enhancing its functionality, usability, and accessibility. Below are the key aspects of my work:

5.1.1. Added Camera Support for Real-Time Captioning

- To make the application more dynamic and interactive, I implemented the camera input feature. This allows users to capture images in real time and generate captions instantly.
- Integrated Streamlit's camera_input functionality to enable users to take pictures directly through their webcam.
- Ensured compatibility with the image captioning models, allowing seamless caption generation for both uploaded and captured images.
- This feature significantly improves the app's usability by providing real-time captioning capabilities, making it more engaging and practical for users.



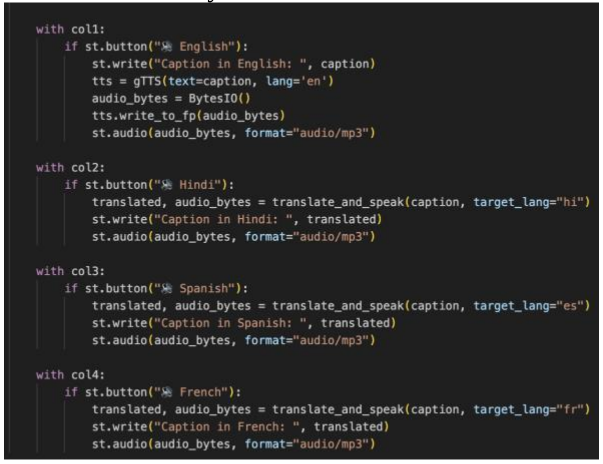
```python
image = None
if option == "Upload from device":
    uploaded_file = st.file_uploader("Drag and drop an image here", type=["jpg", "jpeg", "png"])
    if uploaded_file is not None:
        image = Image.open(uploaded_file).convert("RGB")

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")
```

5.1.2. Multilingual Text-to-Speech Integration

- Recognizing the need for inclusivity, I added **multilingual support** to the text-to-speech (TTS) module. This allows users to hear captions in their preferred language.
- ○ Used the Google Text-to-Speech (gTTS) library to implement speech synthesis in multiple languages.

  - Provided options for users to select their desired language for audio output.
  - This enhancement broadened the app's accessibility for a global audience, ensuring that language is no longer a barrier to usability



```
with col1:
    if st.button("🔊 English"):
        st.write("Caption in English: ", caption)
        tts = gTTS(text=caption, lang='en')
        audio_bytes = BytesIO()
        tts.write_to_fp(audio_bytes)
        st.audio(audio_bytes, format="audio/mp3")

with col2:
    if st.button("🔊 Hindi"):
        translated, audio_bytes = translate_and_speak(caption, target_lang="hi")
        st.write("Caption in Hindi: ", translated)
        st.audio(audio_bytes, format="audio/mp3")

with col3:
    if st.button("🔊 Spanish"):
        translated, audio_bytes = translate_and_speak(caption, target_lang="es")
        st.write("Caption in Spanish: ", translated)
        st.audio(audio_bytes, format="audio/mp3")

with col4:
    if st.button("🔊 French"):
        translated, audio_bytes = translate_and_speak(caption, target_lang="fr")
        st.write("Caption in French: ", translated)
        st.audio(audio_bytes, format="audio/mp3")
```

5.1.3. Exploration with Alternative TTS Libraries

Before finalizing the TTS module, I explored multiple libraries to find the most effective solution:

- tested pyttsx3 and pyPlayground for their TTS capabilities but the audio output produced by these tools sounded mechanical and lacked clarity.
- Additionally, these tools had limited support for multilingual audio synthesis, making them less suitable for the application's requirements.
- After testing, gTTS proved to be the most reliable and versatile solution.
- It produced clear, natural-sounding audio and provided seamless support for multiple languages.
- While experimenting with pyttsx3 and pyPlayground, I encountered difficulties in maintaining a natural tone and ensuring compatibility with the app's multilingual goals.
- Transitioning to gTTS resolved these issues, offering a better user experience.

5.1.4. Optimizing the TTS Workflow

- Integrated the TTS module into the application workflow, enabling users to convert captions into speech at the click of a button.
- Ensured that the TTS feature is lightweight and fast, minimizing latency during real-time usage.

Impact of My Contributions

Through my work on the Vision Voice app, I was able to:

- Enhance the real-time usability of the application with camera support.
- Broaden its global accessibility by integrating multilingual TTS capabilities.
- Ensure the app delivers high-quality audio output by carefully selecting and implementing the right TTS tool.

6. Results and Analysis:

The Vision Voice project focused on creating an innovative image captioning system that integrates advanced neural networks and text-to-speech (TTS) capabilities to enhance accessibility for visually impaired users. While the project presented significant challenges in terms of model training and computational demands, it achieved several critical milestones that reflect the depth and breadth of the work undertaken.

- **Custom CNN**:
  - Demonstrated the limitations of shallow architectures, such as poor generalization and overfitting, highlighting the need for more advanced feature extraction.
- **VGG16**:
  - Its structured architecture provided foundational insights into model scalability and computational requirements. It was computationally heavy and did not generate a proper output as the time taken for the model for each epoch was more than one and half hours.
- **ResNet50**:
  - Emerged as the most balanced model combining computational efficiency and feature extraction depth.
  - Enabled stable training for the model.
- **ResNet101**:
  - It introduced significant computational overhead and integration challenges.
  - Insights from ResNet101 experiments helped in understanding the practicality of using ResNet50 for the current project.

Training deeper models like ResNet101 highlighted the importance of balancing computational efficiency with performance. ResNet50 provided an optimal trade-off, ensuring practical usability without compromising accuracy. Custom CNN and VGG16 experiments reinforced the need for regularization techniques and careful model tuning as they led to overfitting because of their shallow nature.

The integration of real-time captioning and multilingual TTS capabilities significantly enhanced the usability and accessibility of the Vision Voice application.

Real-Time Captioning:

- The implementation of camera support provided a dynamic and interactive user experience, allowing real-time caption generation directly from webcam inputs.
- This feature expanded the app's practical applications, such as enabling users to instantly interpret their surroundings.

Multilingual TTS:

- By supporting multiple languages through Google Text-to-Speech (gTTS), the application became useful for a variety of audience.
- The clear and natural-sounding audio output ensured an inclusive and user-friendly experience overcame the mechanical tone limitations of alternatives like pyttsx3 and pyPlayground.

7. Discussion

The Vision Voice project demonstrated the significance of iterative experimentation in developing a robust and accessible image captioning system. Through extensive trials with different models, we gained valuable insights into the inherent trade-offs between architectural complexity and computational efficiency. Each experiment, whether successful or challenging, contributed to a deeper understanding of model performance and system requirements.

1. Insights from Iterative Model Experimentation

- Exploring deeper architectures like ResNet101 revealed the advantages of increased model depth, such as enhanced feature extraction capabilities. However, this came at the cost of overfitting and significant computational overhead, making the model unsuitable for real-time applications.
- Simpler architectures, such as VGG16, proved to be computationally efficient but struggled to capture the intricate details of complex images. This resulted in captions that lacked richness and depth.
- ResNet50 struck an optimal balance between depth and efficiency, offering robust performance while being computationally feasible for deployment.
- These experiments underscored the importance of understanding trade-offs in model selection. A model must not only achieve high accuracy but also align with the system's performance requirements, such as low latency for real-time captioning.

2. Challenges in Training Deep Architectures

- Training deeper models like ResNet101 posed significant computational challenges, including high memory usage and prolonged training times. These limitations highlighted the importance of choosing architectures that are both powerful and resource-efficient.
- With larger models, the risk of overfitting became apparent, especially when training on relatively small datasets. This emphasized the need for regularization techniques, data augmentation, and careful model tuning.
- The COCO dataset, while extensive, presented challenges in ensuring generalizability. Certain captions lacked diversity, which occasionally limited the model's ability to generate varied and meaningful descriptions.

3. Future Improvements

- Vision Transformers (ViT) represent a promising avenue for image captioning tasks. By leveraging self-attention mechanisms, they can potentially capture global image features more effectively than convolutional networks, leading to richer captions.
- Expanding the training dataset to include more diverse and domain-specific images could enhance the model's generalizability. Datasets like Flickr30k and custom datasets representing real-world scenarios could complement the COCO dataset. The BLIP model that we used explores the similar concept.
- Techniques such as reinforcement learning could be further explored to optimize the quality of generated captions. Running the model on more epochs can significantly improve the modekl
- Future iterations could incorporate more advanced translation APIs to refine multilingual text-to-speech capabilities, ensuring better localization for global audiences.

8. Conclusion

The Vision Voice project serves as a testament to the importance of iterative experimentation and collaborative effort in developing accessible AI solutions. My contributions to the project centered around:

- conducting trials with ResNet101, VGG16, and ResNet50 to identify the most effective architecture. This iterative process provided critical insights into the balance between accuracy, efficiency, and practicality.
- enhancing the app with real-time camera input for dynamic captioning and integrating multilingual TTS for broader accessibility.
- handling computational limitations, overfitting, and dataset diversity issues, which deepened my understanding of model behavior and system requirements.

These efforts collectively laid the groundwork for the team to move forward with the project and explore various areas and integrate advanced features like multilingual text-to-speech. While challenges in model training and deployment presented obstacles, they also provided valuable learning experiences that can guide future advancements.

Looking ahead the lessons learned from this project helps in the future improvement including exploring Vision Transformers, expanding dataset diversity, and refining multilingual support. The Vision Voice project underscores the potential of AI captioning systems to enhance accessibility and serve as a bridge between visual and auditory modalities, making a meaningful impact on users lives.

9. Code Utilization

9.1 model_yash.py:

Original lines from internet: 180

Modified: 40

Added: 50

(180-40) / (180+50) * 100 ~ 60.86%

9.2 app_audio.py

Original lines from internet: 95

Modified: 30

Added: 25

(95-30) / (95+25) * 100 ~ 62.5%

10. References

1. Herdade, S., Kappeler, A., Boakye, K. and Soares, J., 2019. Image captioning: Transforming objects into words. *Advances in neural information processing systems*, *32*. https://openaccess.thecvf.com/content_cvpr_2018/html/Aneja_Convolutional_Image_Captioning_CVPR_2018_paper.html

2. Yao, T., Pan, Y., Li, Y., Qiu, Z. and Mei, T., 2017. Boosting image captioning with attributes. In *Proceedings of the IEEE international conference on computer vision* (pp. 4894-4902). https://openaccess.thecvf.com/content_iccv_2017/html/Yao_Boosting_Image_Captioning_ICCV_2017_paper.html

3. https://www.tensorflow.org/text/tutorials/image_captioning

4. https://paperswithcode.com/task/image-captioning

5. OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/chat

6. Sharma, H., Agrahari, M., Singh, S.K., Firoj, M. and Mishra, R.K., 2020, February. Image captioning: a comprehensive survey. In *2020 International Conference on Power Electronics & IoT Applications in Renewable Energy and its Control (PARC)* (pp. 325-328). IEEE. https://ieeexplore.ieee.org/abstract/document/9087226?casa_token=g8MR7u1kdWQAAAAA:D0VaIzEDMM-D2LB1bMLzQ8ossrpwHz0QZYbR95-f80fbk3-LgwZM6hWNqM1nVaM_7SMDbcMuKw