



**THE GEORGE
WASHINGTON
UNIVERSITY**
WASHINGTON, DC

**Deep Learning
Final Project - Group 1
December 11, 2024
Amir Jafari**

Aravinda Vijayaram Kumar
G36084456

Nemi Makadia
G29362869

Nihar Domala
G34884842

Yash Doshi
G33250233

1. Introduction

VisionVoice project highlights a cutting-edge application designed to improve accessibility for visually impaired users by converting images into descriptive audio. This project employs advanced deep learning techniques, specifically in the realms of computer vision and natural language processing, to generate text captions for images.

These captions are then transformed into audio using text-to-speech (TTS) technology, creating a seamless experience for users who rely on auditory information. The collaborative nature of the project was facilitated through GitHub, where team members developed an efficient data loader to integrate image and caption data effectively. This setup allowed for the smooth handling of large datasets which is the COCO 2017 dataset that contains thousands of images with multiple captions each.

The core of VisionVoice's architecture is a hybrid model combining Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks. This CNN + LSTM architecture was meticulously designed and trained to produce accurate and descriptive captions. The CNN component is responsible for extracting visual features from images, while the LSTM network handles the sequential generation of text descriptions. Various pre-trained models like VGG-16, ResNet50, and EfficientNet-B3 were evaluated for their ability to extract features efficiently and accurately. Each model offers unique benefits; for example, ResNet50 provides a balance between speed and feature extraction quality due to its residual connections, while EfficientNet-B3 offers computational efficiency with its lightweight design.

To ensure the model's effectiveness, its performance was rigorously tested using the BLEU score metric, which evaluates the similarity between generated captions and reference captions. While this metric is useful for objective comparison, it mainly focuses on exact matches rather than semantic understanding. The project also emphasizes user experience by developing a frontend interface using Streamlit. This interface is designed to be intuitive and accessible, allowing users to easily interact with the application. Overall, VisionVoice represents a significant step forward in making visual content accessible to visually impaired individuals through innovative use of AI technologies.

2. Description of the dataset

The MS COCO (Microsoft Common Objects in Context) 2017 dataset is a comprehensive collection designed for image captioning tasks with several key characteristics:

Dataset Specifications:

- Contains over 330,000 images
- Each image is paired with 5 unique captions
- Provides detailed annotations for each image

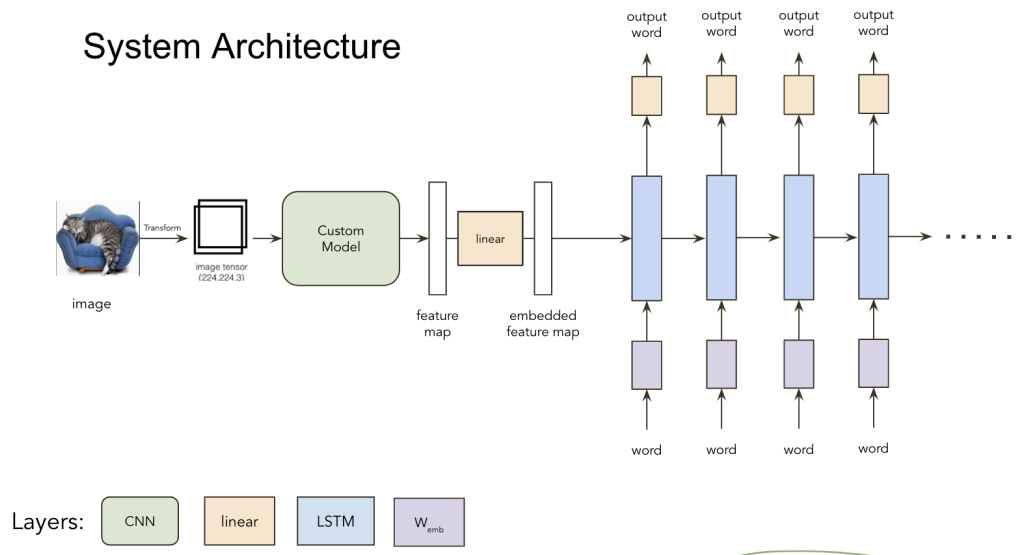
MS COCO was chosen for the VisionVoice project because it:

- Provides sufficient data volume to capture diverse captioning patterns
- Avoids bias issues present in smaller datasets like Flickr8k and Flickr30k
- Offers manageable training time compared to larger datasets like Conceptual Captions
- Maintains a balance between dataset size and practical training requirements

The dataset's comprehensive nature and well-structured annotations make it ideal for training robust image captioning models while remaining within project resource constraints. The MS COCO dataset is widely used for training and evaluating state-of-the-art models in computer vision. It supports complex image understanding tasks

Dataset	Image Count	Caption Count
MS COCO	330,000+	1,650,000+
Flickr8k	8000	40,000
Flickr30k	30,000	150,000+
Visual Genome	100,000+	Detailed annotations
Conceptual captions	3.3 million	3.3 million

3. Deep Learning Network Architecture and Model Overview



3.1 ResNet-50

Encoder: ResNet50 Feature Extractor

Network Type: Convolutional Neural Network (CNN)

Base Model: ResNet50 with transfer learning

3.1.1 Architectural Components

Initialization

- Uses pre-trained ResNet50 model (`models.resnet50(pretrained=True)`)
- Freezes all model parameters to prevent gradient updates
- Removes final classification layer

Feature Extraction Mechanism

- Extracts intermediate CNN features

- Transforms full image representation to compact feature vector
- Embeds features into fixed-dimensional space

Decoder: LSTM

Model Type: Long Short-Term Memory (LSTM) Decoder Primary Function: Generate captions from image features

Core Components

1. Word Embedding Layer
 - Converts word indices to dense vector representations
 - Maps vocabulary to fixed-dimensional embedding space
 - Learns semantic relationships between words
2. LSTM Layer
 - Captures sequential dependencies in text generation
 - Processes caption generation step-by-step
 - Maintains hidden state across generation sequence
3. Linear Output Layer
 - Predicts vocabulary probabilities at each time step
 - Transforms LSTM hidden states to vocabulary space

Initialization Parameters

Key Hyperparameters:

- **embed_size**: Dimensionality of word embeddings
- **hidden_size**: LSTM hidden layer dimensionality
- **vocab_size**: Total number of unique words
- **num_layers**: Number of stacked LSTM layers (default: 1)

Forward Pass Transformation

- Input: Raw image tensor
- Process: Extract convolutional features
- Flatten features to 1D vector
- Linear embedding to desired dimension

Embedding Transformation

- Input feature dimensionality: Determined by ResNet50's final convolutional layer
- Output: Fixed-size embedding vector specified during initialization

Performance Characteristics

- Leverages pre-trained weights for robust feature extraction
- Computationally efficient
- Reduces high-dimensional image to compact representation

3.2 EfficientNet-B3

3.2.1 Image Encoder: EfficientNet Feature Extractor

Network Type: Convolutional Neural Network (CNN) **Base Model:** EfficientNet-B3 with transfer learning

Key Architectural Components

1. **Transfer Learning Strategy**
 - Utilizes pre-trained EfficientNet-B3 model
 - Option to freeze or selectively unfreeze layers
 - Supports fine-tuning through `trainable_layers` parameter
2. **Feature Extraction Mechanism**
 - Extracts rich intermediate features from EfficientNet
 - Adaptive average pooling to reduce spatial dimensions
 - Linear embedding to desired vector space

3.2.2 Caption Decoder: LSTM-based Sequence Generator

Network Type: Long Short-Term Memory (LSTM) Recurrent Neural Network

Decoder Architecture

1. **Word Embedding Layer**
 - Converts discrete word indices to dense vector representations
 - Learns semantic relationships between words
2. **LSTM Layer**
 - Processes sequential caption generation
 - Captures long-term dependencies in text
 - Handles variable-length sequences

3. Output Layer

- Predicts probability distribution over vocabulary
- Generates captions word by word

3.2.3 Caption Generation Technique: Beam Search

Generation Strategy: Beam Search with configurable beam size

- Decoding strategy that significantly enhances the quality of generated captions
- Maintains top-k most promising sequences by considering multiple candidate sequences simultaneously
- Generated captions are more coherent, descriptive, and contextually appropriate

Key Generation Features:

- Maximum length constraint
- Early stopping with end token
- Selects best caption based on cumulative log-probabilities

3.2.4 Model Evaluation

Evaluation Metric: BLEU Score

- Compares generated captions with ground truth
- Measures caption quality and semantic accuracy
- Corpus-level evaluation across multiple images

3.3 BLIP Image Captioning Model

3.3.1 Model Overview

Model Name: BLIP (Bootstrapping Language-Image Pre-training)

Base Model: Salesforce/blip-image-captioning-base

Model Type: Transformer-based Vision-Language Model

Key Characteristics

- Pre-trained on large-scale image-text datasets
- Supports both conditional and unconditional image captioning
- Leverages transformer architecture for multimodal understanding

3.3.2 Captioning Methodology

Generation Approaches:

1. Unconditional Captioning
 - Generates captions without additional text prompts
 - Relies solely on image content
2. Conditional Captioning
 - Allows text conditioning for guided caption generation
 - Enables more specific caption targeting

3.3.3 Implementation Details

Key Components:

- BlipProcessor: Handles image preprocessing
- BlipForConditionalGeneration: Generates captions
- Supports flexible input modes (image-only or image+text)

3.3.4 Project Context

Comparative Analysis In our research, the BLIP model served as a benchmark for:

- Comparing caption generation performance
- Evaluating alternative image captioning approaches
- Providing a pre-trained baseline for model assessment

Comparative Dimensions:

- Caption quality
- Generation flexibility
- Computational efficiency
- Semantic understanding

Significance in Research

The BLIP model provided a standardized, pre-trained approach to compare against our custom-developed image captioning models, offering insights into:

- State-of-the-art image captioning techniques
- Performance variations across different architectural approaches
- Strengths and limitations of transfer learning in multimodal tasks

4. Experimental Setup

The project implementation began with careful consideration of data preparation and model architecture. For training and testing the network, we established a comprehensive setup that encompassed data processing, model implementation, and performance evaluation strategies.

A custom COCO CaptionDataset class was developed to handle the data efficiently. This class manages the initialization of vocabulary, image paths, and caption paths, along with implementing necessary image transformations. The data loading process includes RGB conversion of images, application of standard transformations, and caption tokenization. A specialized collate function was implemented to handle batch processing and caption padding effectively.

Image Transformations

- Resize to 256x256:
 - Ensures all images are of a uniform size, which is necessary for batch processing.
- Random Crop to 224x224:
 - Introduces variability in the training data, helping the model generalize better.
- Random Horizontal Flip:
 - Another form of data augmentation to prevent overfitting by flipping images horizontally.
- ToTensor:
 - Converts images to PyTorch tensors, which are required for model input.
- Normalize:
 - Normalizes the image tensor using mean and standard deviation values specific to the dataset (ImageNet in this case).

The implementation features two primary model architectures. The first combines a pre-trained ResNet50 encoder with an LSTM decoder containing 512 hidden dimensions. The second utilizes an EfficientNetB3 encoder paired with a similar LSTM decoder. These architectures were chosen after initial experiments with custom CNNs and VGG models showed limited success. Additionally, the BLIP (Bootstrapped Language-Image Pretraining) model was integrated to provide enhanced functionality in the frontend interface.

Model Parameters

- Embedding Dimension (256):
 - The size of the vector space in which words and images are embedded. A dimension of 256 is a balance between computational efficiency and the ability to capture semantic meaning.
- Hidden Dimension (512):
 - The size of the hidden state in the LSTM. A larger hidden dimension allows the model to capture more complex patterns in the data.
- Vocabulary Size:

- The number of unique words in the vocabulary. This is determined by the dataset and is crucial for the decoder to generate accurate captions.

Training Parameters

- **Batch Size (128):**
 - Determines the number of samples processed before the model is updated. A batch size of 128 is chosen based on memory constraints and empirical performance.
- **Learning Rate (0.001):**
 - The step size at each iteration while moving toward a minimum of the loss function. A learning rate of 0.001 is a common starting point for the Adam optimizer.
- **StepLR Scheduler:**
 - Adjusts the learning rate every 5 epochs by a factor of 0.1 to fine-tune the training process.

The training process was carefully monitored, with time tracking implemented for each epoch. The models' performance was evaluated using a custom-implemented BLEU score metric, providing quantitative assessment of caption quality. The evaluation results showed promising outcomes, with the EfficientNetB3 + LSTM model achieving a BLEU-4 score of 0.2002, while the ResNet50 + LSTM model reached 0.1487. Training times varied significantly, with ResNet50 requiring 3417.43 seconds per epoch and EfficientNetB3 needing 4408.34 seconds.

To ensure data integrity and proper functionality, a comprehensive validation system was implemented through test.ipynb. This system verifies correct application of transformations, proper caption tokenization, and appropriate sequence padding. The vocabulary generation process includes special tokens (<start>, <end>, <unk>) and maintains two dictionaries for word-to-index and index-to-word mapping, facilitating efficient text processing during training and inference.

The entire system was integrated into a Streamlit-based frontend interface, allowing users to easily select between different models for caption generation. This implementation provides a practical demonstration of the model's capabilities while maintaining accessibility for end-users

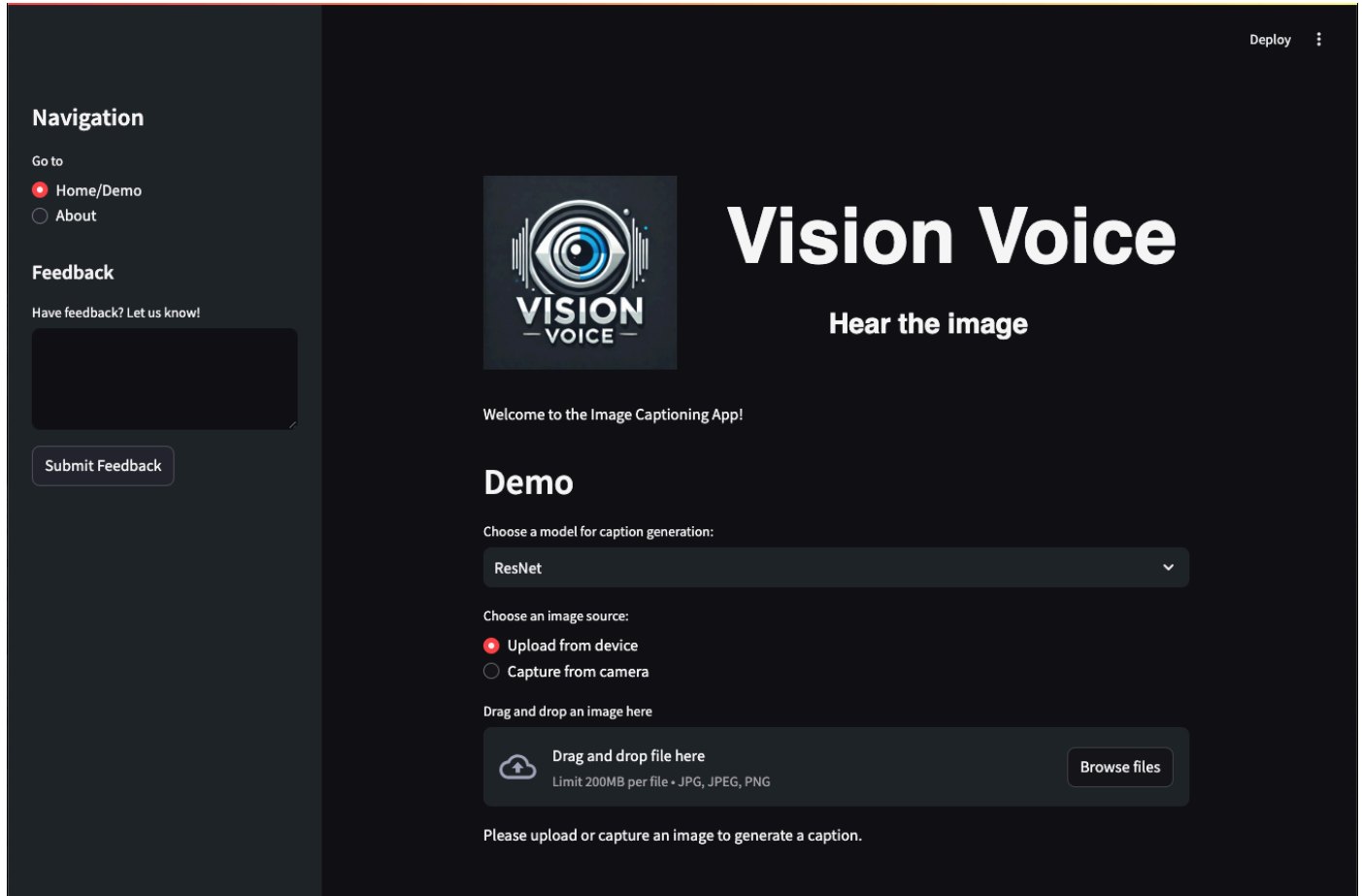
INTEGRATION WITH STREAMLIT AND THE APP

Why streamlit?

- **Ease of Use:** Streamlit requires minimal code to set up and is extremely user-friendly, allowing developers to build apps with just a few lines of Python.
- **Interactive Widgets:** Provides built-in interactive widgets like sliders, buttons, and file uploaders, making it easy to create interactive applications.
- **Rapid Prototyping:** Allows for rapid development and prototyping of data apps. You can iterate quickly and see changes in real-time.

- **Integration with Python:** Seamlessly integrates with popular Python libraries for data science and machine learning, such as Pandas, NumPy, TensorFlow, and PyTorch.

Vision Voice: Enhancing Accessibility with Cutting-Edge Technology



The screenshot displays the Vision Voice application interface. On the left is a dark sidebar with a 'Navigation' section containing 'Go to' links for 'Home/Demo' (selected) and 'About'. Below this is a 'Feedback' section with the text 'Have feedback? Let us know!' and a 'Submit Feedback' button. The main content area has a dark background. At the top right, there is a 'Deploy' button and a menu icon. The central part features the 'Vision Voice' logo, which is a stylized eye with sound waves, and the text 'Vision Voice' in large white font, followed by 'Hear the image' in a smaller white font. Below this, a welcome message reads 'Welcome to the Image Captioning App!'. A 'Demo' section follows, starting with 'Choose a model for caption generation:' and a dropdown menu currently showing 'ResNet'. Then, 'Choose an image source:' is followed by two radio buttons: 'Upload from device' (selected) and 'Capture from camera'. Below these is a 'Drag and drop an image here' instruction and a large dark box with a cloud icon, the text 'Drag and drop file here', a file size limit 'Limit 200MB per file • JPG, JPEG, PNG', and a 'Browse files' button. At the bottom of the main area, a message states 'Please upload or capture an image to generate a caption.'

The Vision Voice application is an innovative tool designed to provide visually impaired users with image captions and auditory feedback. It combines advanced deep learning algorithms, multilingual text-to-speech (TTS) capabilities, and a user-friendly interface to bridge the gap between visual and auditory information.

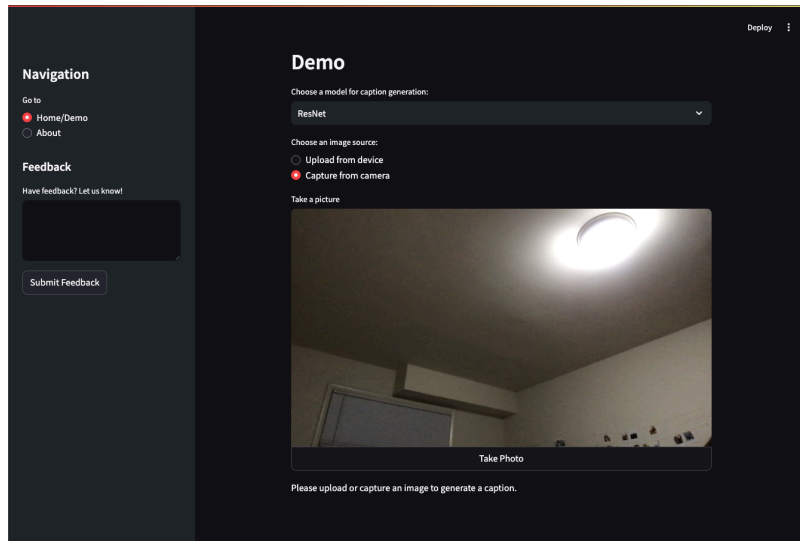
How the Vision Voice Application Works

The Vision Voice application provides an intuitive and accessible way for visually impaired users to generate captions for images and convert them into audio in multiple languages. Here's a detailed, step-by-step breakdown of the application's workflow:

Step 1: Image Input

1. Upload or Capture Image:

- **Browse:** Users can select an image from their device by using the file upload feature.
- **Capture:** Alternatively, users can take a new picture using the app's integrated camera functionality. This ensures flexibility and convenience in obtaining the desired image.



Step 2: Caption Generation

Processing the Image:

- Once the image is uploaded or captured, it is processed through the app's deep learning models.
- The application employs advanced caption generation models such as Custom CNN, EfficientNet (with beam search), and Blip to analyze the image and generate a descriptive caption.

Step 3: Caption to Audio Conversion

3. Generating Audio Output:

- After the caption is generated, it is converted into audio using text-to-speech (TTS) technology.
- The app provides natural-sounding audio output, ensuring that visually impaired users can easily understand the description of the image.

Step 4: Language Options

4. Multilingual Support:

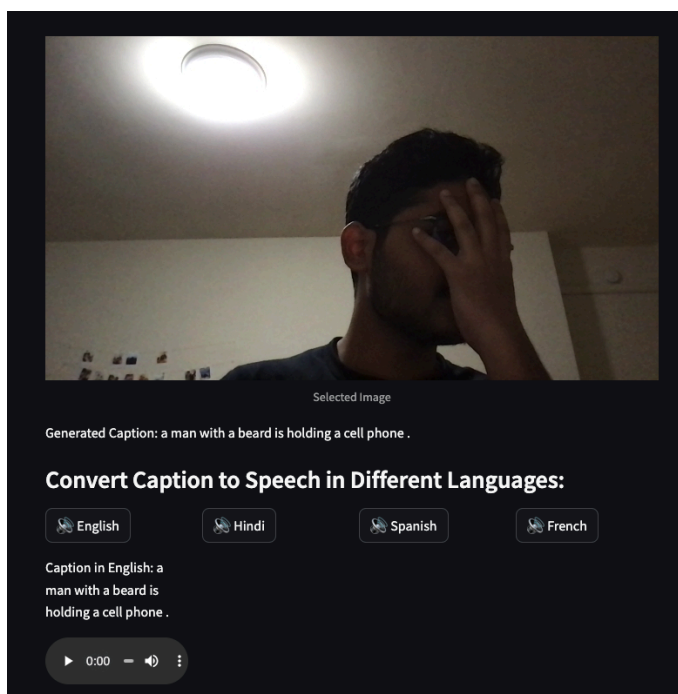
- The application supports audio output in multiple languages. In addition to English, users can choose from Hindi, Spanish, and French.

- This multilingual capability allows the app to cater to a diverse global audience, making it more accessible to users from different linguistic backgrounds.

Step 5: User Feedback

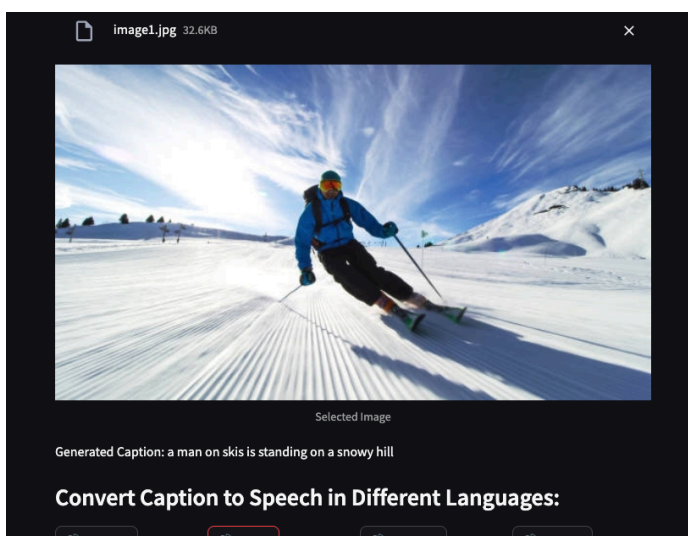
5. Feedback Integration:

- The app includes a feedback mechanism where users can provide suggestions for improvements or report any issues they encounter.
- This continuous feedback loop helps in refining and enhancing the application's functionality over time.




5. Results and Discussion

ResNet50



EfficientNetB3

image1.jpg32.6KB



Selected Image

Generated Caption: a man sitting on a bench next to a tree .

Convert Caption to Speech in Different Languages:

English

Hindi

Spanish

French

Caption in Hindi: एक आदमी पेड़ के पास एक बेंच पर बैठा है।

▶0:00


—

🔊

⋮

BLIP

image1.jpg32.6KB



Selected Image

Generated Caption: a skier skiing down a slope

Convert Caption to Speech in Different Languages:

English

Hindi

Spanish

French

Caption in French: un skieur dévalant une pente

▶0:00

—

🔊

⋮

5.1 Comparative Model Performance

Model	Bleu Score
ResNet50	0.1487
Efficient Net B3	0.2002

The image presents the Bleu Score which is a well-established metric for evaluating image captioning models, for two different architectures: ResNet50 and EfficientNet-B3 which are being used currently in the project.

As shown in the table, the ResNet50-based model achieved a Bleu Score of 0.1487, while the EfficientNet-B3 model scored 0.2002. This indicates that the EfficientNet-based model outperformed the ResNet50 model in terms of generating captions that more closely match the reference captions.

This is ought to be true as EfficientNet-B3 model is much lighter as compared to the ResNet-50.

The Bleu Score is a precision-oriented metric that measures the n-gram overlap between the generated captions and the ground truth captions. A higher Bleu Score suggests that the model is better at producing captions that accurately reflect the semantic content of the images.

5.2 Insights from the Code

The ResNet50 encoder follows a standard transfer learning approach, where a pre-trained ResNet50 model is used as the feature extractor. The final classification layer is removed, and the extracted features are linearly embedded into a desired dimensionality. This approach leverages the robust visual representations learned by the ResNet50 model during pretraining on a large-scale dataset like ImageNet.

However, the ResNet50 model may not be optimally suited for the image captioning task, as its architecture is primarily designed for image classification. The high-level features extracted by ResNet50 may not capture the nuanced semantic information required for accurately describing image content in natural language.

EfficientNet-B3 Encoder In contrast, the EfficientNet-B3 encoder showcases a more sophisticated transfer learning strategy. The code allows for selectively unfreezing the last `trainable_layers` of the EfficientNet-B3 model, enabling fine-tuning on the specific image captioning task. This approach can help the model adapt its feature representations to better align with the demands of caption generation.

EfficientNet-B3 is a more recent and optimized architecture for general computer vision tasks, including those requiring fine-grained understanding of image semantics. Its improved scaling and architectural design choices may contribute to its superior performance compared to the ResNet50 model in this image captioning scenario.

5.3 Theoretical Considerations

The better performance of the EfficientNet-B3 model can be attributed to its stronger capacity for visual feature extraction and semantic understanding. EfficientNet models are known for their efficient scaling and improved representational power, which allows them to capture more nuanced visual information crucial for generating accurate and coherent captions.

Furthermore, the ability to fine-tune the last layers of the EfficientNet-B3 model enables the encoder to specialize its features for the target task of image captioning. This fine-tuning process can help bridge the gap between general visual recognition and the specific requirements of translating image content into natural language.

In contrast, the ResNet50 model, while a powerful and robust feature extractor, may lack the necessary flexibility and task-specific optimization to excel at the complex multimodal task of image captioning. Its architecture and pretraining objectives are primarily geared towards image classification, which may limit its performance on more semantically-driven language generation tasks.

5.4 Future Improvements

To further enhance the image captioning performance, some potential avenues for exploration include:

1. **Architectural Modifications:** Investigating hybrid encoder-decoder architectures that combine the strengths of different CNN and transformer-based models may lead to improved feature extraction and sequence generation capabilities.
2. **Advanced Training Techniques:** Exploring techniques like reinforcement learning, adversarial training, or multi-task learning may help the models better align their caption generation with human-written references.
3. **Multimodal Integration:** Incorporating additional modalities, such as textual descriptions or scene graphs, could provide the models with richer contextual information to generate more comprehensive and coherent captions.
4. **Ensemble Modeling:** Combining the outputs of multiple captioning models, each with its own strengths, could lead to more robust and diverse caption generation.

By building upon the insights gained from the comparative analysis of the ResNet50 and EfficientNet-B3 models, future research can further advance the state-of-the-art in image captioning and push the boundaries of multimodal understanding.

6. 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://proceedings.neurips.cc/paper_files/paper/2019/hash/680390c55bbd9ce416d1d69a9ab4760d-Abstract.html
2. <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00571-w>
3. Ghandi, T., Pourreza, H. and Mahyar, H., 2023. Deep learning approaches on image captioning: A review. *ACM Computing Surveys*, 56(3), pp.1-39.
https://dl.acm.org/doi/full/10.1145/3617592?casa_token=A4cFiNmp1U4AAAAA%3AREZm4gaZpcLtaBMPE_MAdF4qFktDycoJ0F7nx9uevS19Ec4M_csD3ReOCLY0x6ixXCvuVxHivwzdsg
4. Staniūtė, R. and Šešok, D., 2019. A systematic literature review on image captioning. *Applied Sciences*, 9(10), p.2024. <https://www.mdpi.com/2076-3417/9/10/2024>
5. Image Captioning (COCO): https://github.com/iamirmasoud/image_captioning/tree/main (best, pytorch, trained)
6. Image Captioning (Flickr30k):
<https://github.com/nirajankarki5/Flickr30k-Image-Caption-Generator-Using-Deep-Learning?tab=readme-ov-file>
7. Wang, C., Yang, H., Bartz, C. and Meinel, C., 2016, October. Image captioning with deep bidirectional LSTMs. In *Proceedings of the 24th ACM international conference on Multimedia* (pp. 988-997). https://dl.acm.org/doi/abs/10.1145/2964284.2964299?casa_token=LAAe9HDjQtUAAAAA:MOmoMWqEBA8dEr1sgSwhde0gzYlg6_A0v9TkLgzaAZL8yj5x9B0U3dl-K9EcML7eMrhmReF6Pg92Rg
8. OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. <https://chat.openai.com/chat>
9. Transfer Learning basics:
<https://medium.com/@lucrece.shin/chapter-3-transfer-learning-with-resnet50-from-data loaders-to-training-see-d-of-thought-67aaf83155bc>

7. Appendix

- app.py

```
app.py X model_nihar2.py model_nihar.py test.py test.ipynb
Final_project > DL_FinalProject-Group1 > Code > app.py > ...
1 import streamlit as st
2 from gtts import gTTS
3 from deep_translator import GoogleTranslator
4 from io import BytesIO
5 from PIL import Image
6 from generate_caption_beam import generate_caption_beam
7 from generate_caption_resnet import generate_caption_resnet
8 from generate_caption_blip import generate_caption_blip
9
10 # CSS for styling
11 st.markdown(
12     """
13     <style>
14     .header-container {
15         display: flex;
16         align-items: center;
17         padding-top: 20px;
18         padding-bottom: 20px;
19         margin-bottom: 20px;
20         justify-content: center;
21     }
22     .header-title {
23         font-size: 4.8em;
24         font-weight: bold;
25         margin-left: 20px;
26         font-family: 'Algerian', sans-serif;
27     }
28     </style>
29     """
30     ,
31     unsafe_allow_html=True
32 )
33
34 # Home/Demo Page
35 def home_demo_page():
36     st.markdown('<div class="header-container">', unsafe_allow_html=True)
37     logo = Image.open("assets/logo.png")
38     col1, col2 = st.columns([2, 5])
39     with col1:
40         st.image(logo, width=500)
41     with col2:
42         st.markdown('<div class="header-title" style="text-align: center;">Vision Voice</div>', unsafe_allow_html=True)
43         st.markdown('<h3 class="sub" style="text-align: center; font-family: Algerian, sans-serif;">Hear the image</h3>', unsafe_all
44         st.markdown('</div>', unsafe_allow_html=True)
45     # st.markdown('<div class="header-container">', unsafe_allow_html=True)
46     # st.markdown('<h1 class="header-title" style="text-align: center;">Vision Voice</h1>', unsafe_allow_html=True)
47     st.write("Welcome to the Image Captioning App!")
48
49     st.header("Demo")
50
51     model_option = st.selectbox("Choose a model for caption generation:", ("Custom CNN", "EfficientNet", "Blip"))
52
53     option = st.radio("Choose an image source:", ("Upload from device", "Capture from camera"))
54
55     image = None
56     if option == "Upload from device":
57         uploaded_file = st.file_uploader("Drag and drop an image here", type=["jpg", "jpeg", "png"])
58         if uploaded_file is not None:
59             image = Image.open(uploaded_file).convert("RGB")
```

```

59     image = Image.open(uploaded_file).convert("RGB")
60
61     elif option == "Capture from camera":
62         camera_image = st.camera_input("Take a picture")
63         if camera_image is not None:
64             image = Image.open(camera_image).convert("RGB")
65
66     if image is not None:
67         # Display the uploaded or captured image
68         st.image(image, caption="Selected Image", use_container_width=True)
69
70         # Generate caption based on selected model
71         if model_option == "Custom CNN":
72             caption = generate_caption_beam(image)
73         elif model_option == "EfficientNet":
74             caption = generate_caption_resnet(image)
75         elif model_option == "Blip":
76             caption = generate_caption_blip(image)
77         st.write("Generated Caption: ", caption)
78
79
80     # Function to translate and convert text to speech
81     def translate_and_speak(text, target_lang):
82         translated = GoogleTranslator(source="auto", target=target_lang).translate(text)
83         tts = gTTS(translated, lang=target_lang)
84         audio_bytes = BytesIO()
85         tts.write_to_fp(audio_bytes)
86         return translated, audio_bytes
87
88
89     st.subheader("Convert Caption to Speech in Different Languages:")
90     col1, col2, col3, col4 = st.columns(4)
91
92     with col1:
93         if st.button("🇺🇸 English"):
94             st.write("Caption in English: ", caption)
95             tts = gTTS(text=caption, lang='en')
96             audio_bytes = BytesIO()
97             tts.write_to_fp(audio_bytes)
98             st.audio(audio_bytes, format="audio/mp3")
99
100     with col2:
101         if st.button("🇮🇳 Hindi"):
102             translated, audio_bytes = translate_and_speak(caption, target_lang="hi")
103             st.write("Caption in Hindi: ", translated)
104             st.audio(audio_bytes, format="audio/mp3")
105
106     with col3:
107         if st.button("🇪🇸 Spanish"):
108             translated, audio_bytes = translate_and_speak(caption, target_lang="es")
109             st.write("Caption in Spanish: ", translated)
110             st.audio(audio_bytes, format="audio/mp3")
111
112     with col4:
113         if st.button("🇫🇷 French"):
114             translated, audio_bytes = translate_and_speak(caption, target_lang="fr")
115             st.write("Caption in French: ", translated)
116             st.audio(audio_bytes, format="audio/mp3")

```



```

117     else:
118         st.write("Please upload or capture an image to generate a caption.")
119
120 # About Page
121 def about_page():
122     st.header("About the Project")
123     st.write("""
124         This project uses deep learning to generate captions for images.
125         The model is trained to understand images and provide descriptive captions.
126         The app also allows converting these captions to audio.
127     """)
128
129 # Sidebar Navigation
130 st.sidebar.title("Navigation")
131 page = st.sidebar.radio("Go to", ("Home/Demo", "About"))
132
133 # Feedback Section
134 st.sidebar.header("Feedback")
135 feedback = st.sidebar.text_area("Have feedback? Let us know!")
136 if st.sidebar.button("Submit Feedback"):
137     if feedback.strip():
138         st.sidebar.success("Thank you for your feedback!")
139     else:
140         st.sidebar.error("Feedback cannot be empty!")
141
142 # Page Routing
143 if page == "Home/Demo":
144     home_demo_page()
145 elif page == "About":
146     about_page()
147

```

- **model.py**

```

36 class ImageEncoder(nn.Module):
37     def __init__(self, embedding_dim, trainable_layers=0):
38         super(ImageEncoder, self).__init__()
39         base_model = models.efficientnet_b3(pretrained=True)
40         for param in base_model.parameters():
41             param.requires_grad = False
42         if trainable_layers > 0:
43             feature_params = list(base_model.features.parameters())
44             for param in feature_params[-trainable_layers:]:
45                 param.requires_grad = True
46         self.feature_extractor = base_model.features
47         self.avgpool = nn.AdaptiveAvgPool2d(1)
48         self.fc = nn.Linear(1536, embedding_dim)
49
50     def forward(self, images):
51         x = self.feature_extractor(images)
52         x = self.avgpool(x)
53         x = torch.flatten(x, 1)
54         embeddings = self.fc(x)
55         return embeddings
56
57 class CaptionDecoder(nn.Module):
58     def __init__(self, embedding_dim, hidden_dim, vocab_size, vocab, num_layers=1):
59         super(CaptionDecoder, self).__init__()
60         self.hidden_dim = hidden_dim
61         self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)
62         self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers, batch_first=True)
63         self.fc = nn.Linear(hidden_dim, vocab_size)
64         self.hidden_state = (torch.zeros(1, 1, hidden_dim), torch.zeros(1, 1, hidden_dim))
65         self.end_token_id = vocab(vocab.end_word) # Add the end token ID
66         self.vocab = vocab
67         self.start_token_id = vocab(vocab.start_word)
68
69     def forward(self, image_features, captions):
70         caption_embeddings = self.word_embeddings(captions[:, :-1])
71         inputs = torch.cat([image_features.unsqueeze(1), caption_embeddings], dim=1)
72         lstm_output, self.hidden_state = self.lstm(inputs)
73         outputs = self.fc(lstm_output)
74         return outputs

```

```

106 def generate_caption(self, image_features, beam_size=3, max_length=20):
107     device = image_features.device
108     vocab_size = self.fc.out_features
109     assert image_features.size(0) == 1, "Batch size should be 1 for inference."
110     start_token_id = self.start_token_id
111     end_token_id = self.end_token_id
112     h = torch.zeros(self.lstm.num_layers, beam_size, self.hidden_dim, device=device)
113     c = torch.zeros(self.lstm.num_layers, beam_size, self.hidden_dim, device=device)
114     seqs = torch.LongTensor([start_token_id]).to(device)
115     seqs = seqs.repeat(beam_size, 1)
116     top_k_scores = torch.zeros(beam_size, 1, device=device)
117     image_features = image_features.repeat(beam_size, 1)
118     image_features = image_features.unsqueeze(1)
119     with torch.no_grad():
120         lstm_output, (h, c) = self.lstm(image_features, (h, c))
121     complete_seqs = []
122     complete_seqs_scores = []
123     for step in range(max_length):
124         last_word_ids = seqs[:, -1]
125         embeddings = self.word_embeddings(last_word_ids)
126         embeddings = embeddings.unsqueeze(1)
127         lstm_output, (h, c) = self.lstm(embeddings, (h, c))
128         scores = self.fc(lstm_output.squeeze(1))
129         scores = F.log_softmax(scores, dim=1)
130         scores = top_k_scores.expand_as(scores) + scores
131         top_k_scores_flat, top_k_words_flat = scores.view(-1).topk(beam_size, dim=0, largest=True, sorted=True)
132         prev_word_inds = top_k_words_flat // vocab_size
133         next_word_inds = top_k_words_flat % vocab_size
134         new_seqs = torch.cat([seqs[prev_word_inds], next_word_inds.unsqueeze(1)], dim=1)
135         new_scores = top_k_scores_flat
136         complete_inds = (next_word_inds == end_token_id).nonzero(as_tuple=False).squeeze(1)
137         incomplete_inds = (next_word_inds != end_token_id).nonzero(as_tuple=False).squeeze(1)
138         if len(complete_inds) > 0:
139             for i in complete_inds:
140                 complete_seqs.append(new_seqs[i].tolist())
141                 complete_seqs_scores.append(new_scores[i].item())
142             k = beam_size - len(complete_seqs)
143             if k <= 0:
144                 break
145             seqs = new_seqs[incomplete_inds]
146             top_k_scores = new_scores[incomplete_inds].unsqueeze(1)
147
148         h = h[:, prev_word_inds[incomplete_inds], :]
149         c = c[:, prev_word_inds[incomplete_inds], :]
150
151     if len(complete_seqs_scores) == 0:
152         complete_seqs = seqs.tolist()
153         complete_seqs_scores = top_k_scores.squeeze(1).tolist()
154
155     best_idx = complete_seqs_scores.index(max(complete_seqs_scores))
156     best_seq = complete_seqs[best_idx]
157
158     words = [self.vocab.idx2word[idx] for idx in best_seq]
159     # Remove the <end> token if it exists
160     if end_token_id in best_seq:
161         words = words[:best_seq.index(end_token_id)]
162
163     return words
164

```