Final Project: Image Caption Generator Using Deep Learning

Project Title:

Image Caption Generator using CNN and RNN (VGG16 + LSTM)

Overview:

This project implements an image captioning system that generates natural language descriptions for images. The model uses a pre-trained CNN (VGG16) to extract image features and an RNN (LSTM) to generate descriptive captions.

Objective:

To build a model that can interpret visual content and describe it using human-like language. The model is trained on the Flickr8k dataset and aims to learn the relationship between image features and text.

Technologies Used:

- Python
- TensorFlow / Keras
- Streamlit (for web app)
- NumPy, Pandas
- Matplotlib (optional for visualization)

Dataset:

Flickr8k

- 8000 images with 5 captions each
- Captions are stored in `C:/Users/abish/Desktop/Guvi_Files/Project_Final/archive/Text/captions.txt`
- Images are in `C:/Users/abish/Desktop/Guvi_Files/Project_Final/archive/Images`

Project Workflow:

- Load and clean captions (lowercase, remove punctuation, add start/end tokens)
- Use pre-trained VGG16 (without top layer) to extract 4096-dim image features
- Tokenize captions and prepare padded sequences
- Build a hybrid CNN-RNN model:
    - CNN features passed through Dense layers
    - Captions passed through Embedding + LSTM layers
    - Both merged using `Add()` and fed to Dense + Softmax for word prediction
- Train using a custom generator for efficient memory handling
- Save model and tokenizer
- Deploy with a Streamlit app for caption generation on uploaded images

---

Learning Outcomes:

- Learned how to combine computer vision (CNN) and natural language processing (RNN)
- Understood pre-trained model usage (VGG16) for feature extraction
- Implemented a custom data generator to enable training on large datasets
- Built and deployed an interactive image captioning web application using Streamlit

---

Challenges Faced:

Memory Management:

- Initially, the training process failed due to high memory usage from loading all sequences and one-hot labels into memory.
- Solved this by using `sparse_categorical_crossentropy` and creating a custom `DataGenerator` using `keras.utils.Sequence`, which loads captions and features in small batches.
- Also avoided loading all captions per image by using only one caption per image per batch.

Output:

- `caption_model.keras` - trained model
- `tokenizer.pkl` - vocabulary for decoding
- `image_features_vgg.npy` - image features
- `app.py` - Streamlit app

Use Cases:

- Visually impaired accessibility tools
- Social media auto-captioning
- Content moderation
- Surveillance and monitoring