Final Report: Image Captioning Project

1. Project Summary

The goal of this project was to build an automated image captioning system capable of generating meaningful and contextually accurate textual descriptions for given images. The system combines deep learning techniques from both computer vision and natural language processing.

Dataset used: Flickr8k, consisting of 8,000 images, each paired with 5 descriptive captions.

Exploratory steps included: Caption length distribution, Top word frequency analysis, and Visual inspection of image categories and objects.

2. Preprocessing Techniques

All standard preprocessing techniques were applied, including:

- Text cleaning: Lowercasing, punctuation removal, special character filtering.

- Tokenization and lemmatization.

- Stopwords removal (tested with and without it).

- Padding for caption sequences.

- Vocabulary thresholding to remove rare words.

- Sequence mapping to prepare the input-output format for training.

3. Embedding Techniques

All major embedding methods were tested to analyze their impact on performance:

- GloVe (pre-trained word vectors)

- FastText – including subword information

- TF-IDF and Bag of Words (for analysis purposes)

- BERT Embeddings – Contextual embeddings capturing deep semantic

- Learned Embeddings: Embedding layer trained jointly with the model

4. Model Development & Experiments

Multiple deep learning architectures were implemented and tested.

CNN + RNN Models: Used CNNs (Inception, ResNet50, Densenet201) for image features and combined with RNN-based decoders: Vanilla LSTM, Bidirectional LSTM, GRU.

Transformer Architecture: Fine-tuned Vision Transformer (ViT) and Transformer decoders significantly improved results.

After testing different models, we chose BLIP (Bootstrapped Language-Image Pretraining) as our final model.
BLIP combines vision and language learning and gave the best results in generating natural, accurate captions.
At first, with the pre-trained version, it gave general or less detailed captions.
But after fine-tuning it on our own dataset, it started giving more specific and meaningful descriptions based on the image content.

5. Optimization

To improve model performance and ensure generalization, we applied multiple optimization techniques:

Hyperparameter Tuning:

- Number of LSTM layers and hidden units (for the decoder)
- Batch size and learning rate
- Dropout rates to prevent overfitting
- Optimizers: Compared Adam and AdamW to evaluate their impact on convergence speed and performance

Regularization Techniques:

- Dropout was applied between layers to reduce overfitting.
- Early stopping monitored validation loss and stopped training when no improvement was observed to avoid unnecessary training and overfitting.
- Learning rate scheduling was used to gradually reduce the learning rate as training progressed, helping the model settle into better minima.

Model Validation:

- Cross-validation was applied to evaluate model stability and consistency across different data splits.

6. Insights and Business Impact

Accurate image captioning enables automation in content generation, improves accessibility, and enhances image search capabilities.

In e-commerce, it leads to better product descriptions, improved user engagement, and SEO.

Results highlighted the importance of context-aware embeddings and advanced decoders.

7. Challenges & Solutions

Challenges faced and how they were overcome:

- Variance in caption structure → Normalized using lemmatization and length trimming

- Overfitting → reduce model complexity and adding dropout

- Long training times → Transfer learning and pre-trained backbones

- Poor RNN results → Shifted to Transformer models

- Embedding incompatibility → Unified token index and embedding formats

8. Deployment Architecture

- **Server Setup:**
    - Flask web server with CORS (Cross-Origin Resource Sharing) enabled
        - CORS allows browsers to access resources from different origins (domains or ports)—useful when the frontend and backend are separate
    - Single-server deployment: web server and model run on the same machine
    - GPU acceleration support: `torch.device('cuda' if torch.cuda.is_available() else 'cpu')`

- **Model Management:**
    - BLIP model loaded once during server startup
    - Model files cached locally (`blip` directory)
    - In-memory model initialization for low-latency inference
- **Resource Efficiency:**
    - All processing done in memory (no temp files on disk)
    - Webcam input optimized to minimize latency
    - Uses base64 encoding for transferring image and audio data between frontend and backend

9. Main Functionalities

1. **Image Upload**
    - Supports PNG, JPG, JPEG, and GIF files
    - Validates image type before processing
    - Outputs caption, processed image, and spoken audio
2. **Webcam Integration**
    - Live webcam capture for real-time captioning
    - Live Mode: Faster, captions only
    - Capture Mode: Captions with optional speech
3. **AI Captioning**
    - Utilizes Salesforce's BLIP model to describe image contents
    - Works with uploaded files and webcam snapshots
4. **Text-to-Speech**
    - Uses Google's gTTS to convert text captions into speech

    - Base64-encoded audio returned for direct browser playback
    - Aids accessibility for visually impaired users

5. **Performance Metrics**
    - Displays captioning time for webcam input
    - Gives users insight into application responsiveness

6. **Experiment Tracking and Model Registry**
    - MLflow integrated to track model versions, parameters, and performance metrics.
    - Every inference or retraining run can be logged to monitor model behavior over time.
    - The BLIP model can be registered in the MLflow Model Registry, enabling version control and easy rollback.

10. Final Outcome

The final model — a fine-tuned Transformer using FastText embeddings and advanced preprocessing — achieved the best performance in terms of BLEU, METEOR, and CIDEr scores.This project showcases the importance of experimentation, thorough preprocessing, and modern architectures for achieving state-of-the-art results in image captioning.

11. Future Improvements

- **Multilingual Support:** Integrating multilingual models or fine-tuning BLIP on caption datasets in different languages would broaden accessibility and use cases.
- **Model Quantization & Pruning:** To improve deployment efficiency, especially on edge devices, applying quantization or pruning can reduce the model's size and inference time without major loss in performance.
- **Feedback-Based Learning:** Incorporating user feedback to iteratively fine-tune the model could help personalize and refine caption quality over time.
- **Visual Question Answering (VQA) Integration:** Extending the system to answer questions about image content can add significant interactivity and deepen image understanding.

- **Better Decoding Strategies:** Exploring advanced decoding strategies like Top-k sampling, nucleus sampling, or diverse beam search may further improve caption creativity and variety.

- **Improved Evaluation Pipeline:** Adding human evaluation and interpretability tools can help assess caption fluency, factual accuracy, and bias—beyond automated metrics.

- **Mobile & Offline Support:** Packaging the model with tools like ONNX or TensorRT for mobile or offline environments can expand the application's reach.