EC9170

DEEP LEARNING FOR ELECTRICAL AND COMPUTER ENGINEERS

MINI PROJECT

Image Caption Generator using CNN-LSTM

SUBMITTED BY:

Nathiskar Shriganeshan (2021/E/190)

Jenarththan Akilan (2021/E/006)

DEPARTMENT OF COMPUTER ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF JAFFNA

APRIL 2025

TABLE OF CONTENTS

1. Abstract

This project presents the implementation of an image captioning system using a custom-built CNN-LSTM architecture. The model takes an image as input and generates a relevant textual description as output. A dataset of 8,091 images, each annotated with five different captions, was used for training and evaluation. The custom convolutional neural network was developed from scratch to extract visual features, which were then passed to an LSTM-based decoder for sequence generation. The model was trained using categorical cross-entropy loss and optimized with the Adam optimizer. Evaluation was carried out using BLEU scores to assess caption quality. The project also includes manual hyperparameter tuning and a demonstration of k-fold cross-validation. The results demonstrate the feasibility of generating meaningful captions from images using a CNN-LSTM pipeline trained from scratch.

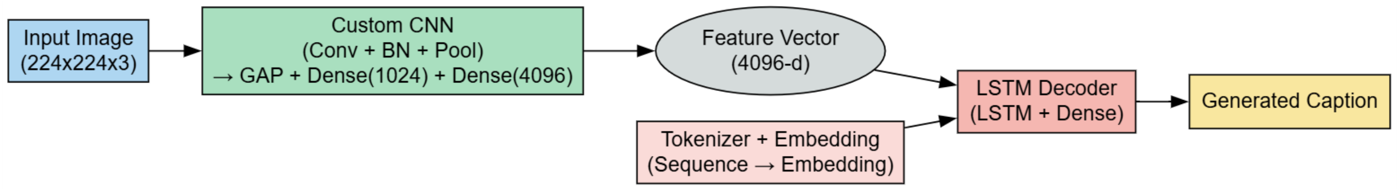


Figure 1: High-Level Architecture of the CNN-LSTM Image Caption Generator

This diagram illustrates the overall architecture of the image captioning model. The input image is first processed through a custom-built Convolutional Neural Network (CNN) that includes convolutional, pooling, and global average pooling layers. The resulting feature vector is passed to an LSTM-based decoder alongside embedded text input generated by a tokenizer. The decoder then produces a sequence of words forming the final image caption.

2. Objective

The objective of this project is to develop an image captioning system that generates meaningful textual descriptions from input images using a custom-designed CNN-LSTM architecture. The goal is to build the entire model from scratch without using pre-trained networks, ensuring that both visual feature extraction and sequence generation components are fully implemented and optimized. The model is trained on a dataset of images paired with multiple captions and is evaluated using standard performance metrics such as BLEU scores. The project also includes manual hyperparameter tuning and cross-validation to enhance the model's performance and reliability.

3. Dataset Description

The dataset used for this project contains a total of 8,091 images, each annotated with five different human-generated captions. These captions provide descriptive information about the main objects and activities present in the images. The images are of varied scenes and categories, offering diversity in visual content and language.

Each image is associated with a unique identifier and five corresponding textual captions stored in a text file. The dataset is used for both training and evaluation of the image captioning model.

Before training, the data is split into training and testing subsets to assess the generalization performance of the model.

4.Data Preprocessing Methodology

4.1 Image Preprocessing (Resizing, Feature Extraction)

For each image in the dataset, the following preprocessing steps were applied before passing it to the model:

- **Resizing**: Images were resized to **224×224 pixels** to match the input shape required by the custom CNN.
- **Conversion to Array**: Each image was converted into a numerical array using Keras' img_to_array() function.
- **Preprocessing**: Images were normalized using TensorFlow's preprocess_input() function to scale pixel values appropriately.
- **Batch Dimension Addition**: The array was reshaped to include the batch size, resulting in a shape of **(1, 224, 224, 3)** before feeding into the model.
- **Feature Extraction**: The preprocessed images were then passed through a custom-built CNN model that included five convolutional blocks followed by **Global Average Pooling** and two **Dense layers**. The final output was a **4096-dimensional feature vector** for each image.

These vectors were saved and used as fixed image embeddings for the caption generation process.

4.2 Caption Cleaning (<start>, <end>, punctuation removal)

A clean() function was implemented to preprocess each caption. The cleaning involved:

- **Lowercasing** all words to reduce the vocabulary size.
- **Punctuation Removal** using regular expressions to eliminate special characters and digits.
- **Whitespace Normalization** to remove extra spaces between words.
- **Filtering** of short or empty tokens (e.g., single characters).
- **Special Tokens**:
    - startseq was added at the beginning of each caption.
    - endseq was appended at the end.

This cleaning process ensured that the captions were standardized for the tokenization and sequence modeling steps.

4.3 Tokenization and Vocabulary Size

After caption cleaning, the dataset's text data was converted into numerical sequences using Keras' Tokenizer. The following steps were taken:

- **Fitting the Tokenizer**: The tokenizer was trained on the complete set of cleaned captions using tokenizer.fit_on_texts(all_captions). This generated a word-to-index mapping based on word frequency.
- **Vocabulary Creation**: Every unique word in the dataset was assigned a unique integer ID.
- **Vocabulary Size Calculation**: The total vocabulary size was computed as:

  vocab_size = len(tokenizer.word_index) + 1

  This additional 1 accounts for padding or unknown token indexing.

This tokenization process allows captions to be represented as sequences of integers, which are necessary for model input during training.

4.4 Maximum Caption Length

To ensure consistent input shapes for the LSTM decoder, the maximum length of the captions was determined by measuring the number of words in the longest caption across the dataset. This was calculated using:

max_length = max(len(caption.split()) for caption in all_captions)

This value is used throughout the model:

- For padding input sequences using pad_sequences()
- As the input_length for the embedding and LSTM layers
- To control the number of steps in the caption generation loop

This preprocessing step ensures that all caption sequences fed into the model are of uniform length, which is required for training the LSTM effectively.

5. Model Architecture

5.1 Custom CNN Feature Extractor

To extract high-level visual features from the input images, a custom Convolutional Neural Network (CNN) was developed from scratch. The model includes **five convolutional blocks** followed by **two fully connected (dense) layers**. Each block is designed to reduce spatial resolution while increasing the depth and complexity of learned features.

The architectural breakdown is as follows:

- **Convolutional Block 1**: 32 filters with a 3×3 kernel, ReLU activation, batch normalization, and max pooling
- **Convolutional Block 2**: 64 filters with the same structure (Conv2D + BN + MaxPooling)
- **Convolutional Block 3**: 128 filters, again followed by batch normalization and max pooling
- **Convolutional Block 4**: 256 filters, further deepening the representation with batch normalization and max pooling

- **Convolutional Block 5**: 512 filters, ending with **Global Average Pooling (GAP)** to convert the final feature map into a compact vector

After convolutional processing, the output is passed through:

- A **Dense layer** with 1024 ReLU-activated units
- A **Dropout layer** with a 0.5 dropout rate for regularization
- Another **Dense layer** with 4096 ReLU-activated units

The output of this network is a **4096-dimensional feature vector**, which serves as the visual embedding for the caption generation model.

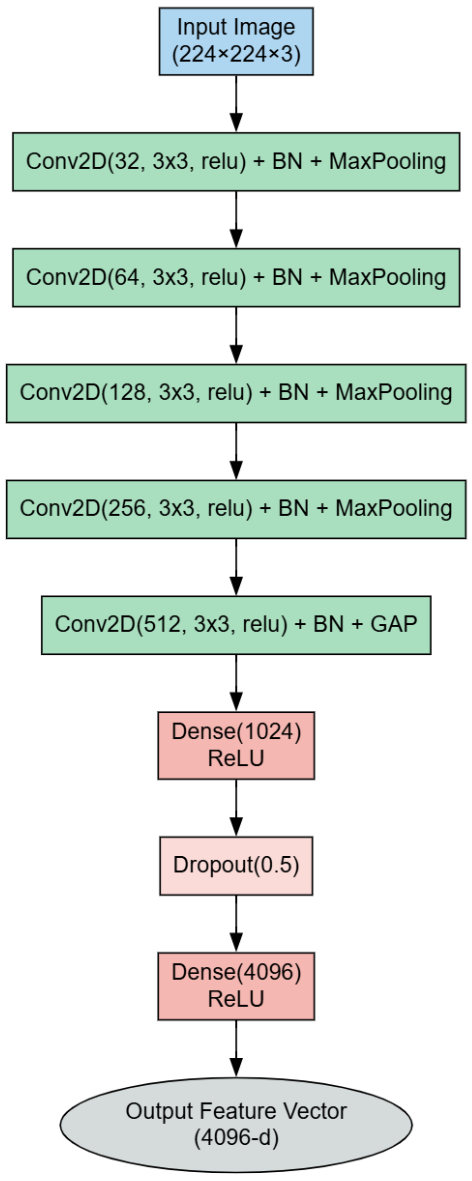

Input Image (224×224×3)
Conv2D(32, 3x3, relu) + BN + MaxPooling
Conv2D(64, 3x3, relu) + BN + MaxPooling
Conv2D(128, 3x3, relu) + BN + MaxPooling
Conv2D(256, 3x3, relu) + BN + MaxPooling
Conv2D(512, 3x3, relu) + BN + GAP
Dense(1024) ReLU
Dropout(0.5)
Dense(4096) ReLU
Output Feature Vector (4096-d)

Figure 5.1: Custom CNN architecture used for image feature extraction. It includes five convolutional blocks (each with Batch Normalization and MaxPooling, except the last with GAP) and two dense layers, ultimately producing a 4096-dimensional output feature vector.

5.2 LSTM Decoder (Embedding, LSTM, Dense Layers)

The LSTM decoder is responsible for generating the caption sequence by combining visual features from the CNN and textual information from the previously generated caption tokens. This decoder takes two distinct inputs:

1. **Feature Vector**: A 4096-dimensional feature embedding extracted from the CNN encoder.
2. **Partial Caption**: A sequence of word tokens (e.g., <start> a man is...) encoded as integers and fed into an Embedding layer.

The architecture of the decoder is structured as follows:

- **Caption Input**: The tokenized caption sequence (with a fixed maximum length, e.g., 31) is passed into an **Embedding layer**, which transforms each token into a 256-dimensional dense vector.
- **Dropout Layer**: A dropout of **0.4** is applied to the embedding output to reduce overfitting and encourage generalization.
- **LSTM Layer**: A single **LSTM unit** with 256 hidden units processes the embedded sequence. The layer is configured with:
  - return_sequences=False (final output only)
  - recurrent_activation='sigmoid' and activation='tanh'
  - unroll=True to reduce computational graph complexity
- **Feature Input (CNN Output)**: The 4096-dimensional visual vector is passed through a **Dropout(0.4)** layer, followed by a **Dense(256)** layer with ReLU activation to reduce the feature vector to match the LSTM output size.
- **Merge (Add Layer)**: The processed visual and textual features are combined using an **element-wise Add layer**.
- **Final Dense Layers**:
  - A **Dense(256)** layer with ReLU activation to refine the merged representation
  - A **Dense layer with output size equal to the vocabulary size** and softmax activation to predict the next word

This model structure enables the decoder to learn joint representations from both visual content and textual context, facilitating accurate and coherent caption generation.

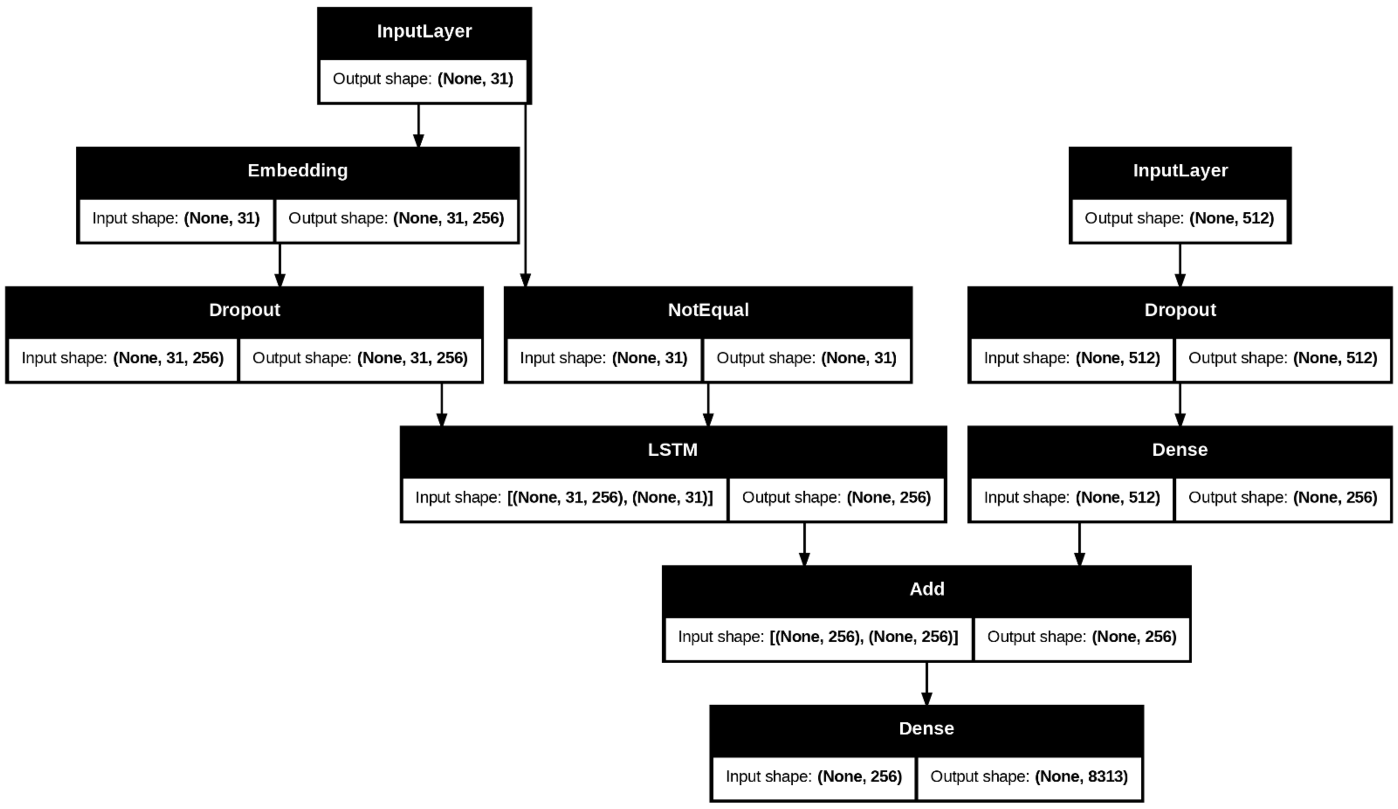

InputLayer
Output shape: (None, 31)
Embedding
Input shape: (None, 31)
Output shape: (None, 31, 256)
InputLayer
Output shape: (None, 512)
Dropout
Input shape: (None, 31, 256)
Output shape: (None, 31, 256)
NotEqual
Input shape: (None, 31)
Output shape: (None, 31)
Dropout
Input shape: (None, 512)
Output shape: (None, 512)
LSTM
Input shape: [(None, 31, 256), (None, 31)]
Output shape: (None, 256)
Dense
Input shape: (None, 512)
Output shape: (None, 256)
Add
Input shape: [(None, 256), (None, 256)]
Output shape: (None, 256)
Dense
Input shape: (None, 256)
Output shape: (None, 8313)

Figure 5.2: LSTM decoder model with two input paths — one for the embedded caption sequence and another for CNN-extracted visual features. These are merged and passed through two Dense layers to generate the final word prediction at each timestep.

5.3 Model Structure Diagram (Full CNN + LSTM Flow)

The complete image captioning architecture is composed of two integrated branches — a **CNN encoder** for extracting visual features and an **LSTM decoder** for generating captions. These branches operate in parallel and are merged to form a unified output pipeline.

1. CNN Encoder Branch

- A **custom CNN** processes the input image (224×224×3) and extracts deep visual features.
- The CNN architecture includes:
  - Multiple convolutional layers with **Batch Normalization** and **MaxPooling**
  - A **Global Average Pooling (GAP)** layer
  - Two fully connected **Dense layers** with 1024 and 4096 units respectively
- The output of this branch is a **4096-dimensional feature vector**, which serves as a condensed representation of the input image.

2. LSTM Decoder Branch

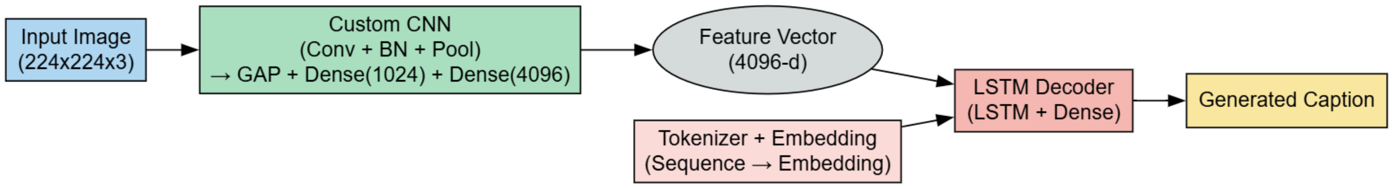
- The partial caption (sequence of previously generated words) is first passed through a **Tokenizer and Embedding layer**, converting tokens into **256-dimensional embeddings**.
- The embeddings are processed by a single **LSTM layer** with 256 hidden units, capturing sequential and grammatical dependencies in the caption.

- The CNN feature vector is simultaneously passed through a **Dense layer** (256 units) after **Dropout**, to match the shape of the LSTM output.

3. Merge and Output

- The outputs of both branches — the LSTM and the image feature projection — are **merged using an Add layer**.
- This merged vector is passed through:
    - A **Dense(256)** layer with ReLU activation to refine the joint representation
    - A **final Dense layer** with output dimension equal to the vocabulary size and **softmax activation**, predicting the next word token in the sequence

This dual-branch pipeline allows the model to jointly attend to both image features and the evolving caption sequence, leading to context-aware caption generation.



Input Image (224x224x3)
Custom CNN (Conv + BN + Pool) → GAP + Dense(1024) + Dense(4096)
Feature Vector (4096-d)
Tokenizer + Embedding (Sequence → Embedding)
LSTM Decoder (LSTM + Dense)
Generated Caption

Figure 5.3: Full model pipeline showing the combined flow of image caption generation. The image is processed by the CNN encoder to produce a 4096-dimensional feature vector. Simultaneously, the caption input sequence is embedded and processed through an LSTM. The outputs of both branches are merged and passed through Dense layers to generate the predicted caption word-by-word.

6. Model Training and Optimization

This section outlines the configuration and process used to train the CNN-LSTM model for image caption generation, including the selected loss function, optimizer, batch size, training duration, and the loss trends over time.

6.1 Loss Function and Optimizer

The model was compiled using the **categorical cross-entropy loss** and the **Adam optimizer**:

```
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

- **Loss Function – Categorical Cross-Entropy**
  Since the model predicts the next word in a sequence from a large vocabulary, categorical cross-entropy is appropriate. It measures how far the predicted probability distribution diverges from the true one-hot encoded label.
- **Optimizer – Adam**
  The Adam optimizer was selected for its adaptive learning rate and faster convergence capabilities. It helps stabilize training in sequence-based models by using both first and second moment estimates. The model was trained using the default Adam configuration:

- Learning rate: 0.001
- β1 = 0.9, β2 = 0.999
- ε = 1e-07

6.2 Epochs and Batch Size

The model was trained for **15 epochs** with a **batch size of 64**. Each epoch involved passing the entire training dataset through the model once, and the batch size controlled the number of samples processed before updating model weights.

- **Epochs**: 15
- **Batch Size**: 64

These values were determined empirically, balancing memory efficiency and convergence stability. The model also performed **validation on each epoch** using a separate validation set.

6.3 Training vs Validation Loss Graph

The following figure illustrates the evolution of both training and validation loss over 15 epochs. The training process was monitored epoch-by-epoch to assess overfitting and learning stability.

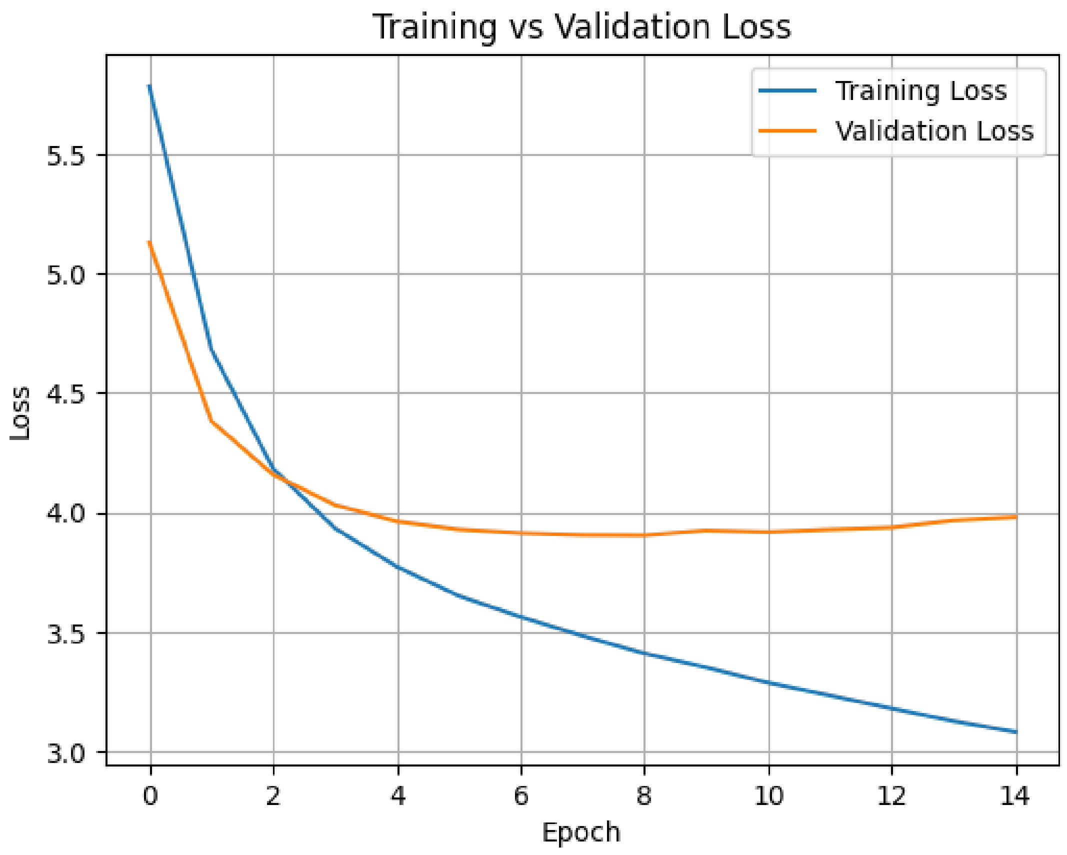


Figure 6.1: Training vs Validation Loss Over Epochs

- The **training loss** decreases consistently across epochs, indicating that the model is learning the captioning patterns effectively.
- The **validation loss** initially decreases and then plateaus, suggesting that the model is generalizing reasonably well without overfitting.

This loss behavior validates the effectiveness of the model's structure, loss function, and optimization settings.

7. Hyperparameter Tuning and Cross-Validation

This section outlines the process of hyperparameter tuning used to optimize the CNN-LSTM model for image caption generation. Hyperparameters such as embedding dimensions, LSTM units, and the learning rate were tuned to identify the most effective configuration.

7.1 Tuned Parameters List

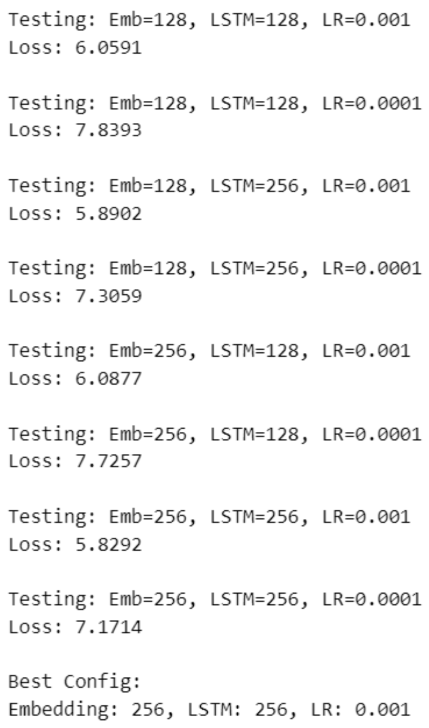
The following hyperparameters were tuned during the optimization process:

Hyperparameter	Values Tested
Embedding Dimension	128, 256
LSTM Units	128, 256
Learning Rate	1e-3, 1e-4

Each combination of these hyperparameters was tested, and the model's performance was evaluated based on the loss function after one epoch of training.

7.2 Grid Search / Tuning Results

A **grid search** approach was employed to test different combinations of hyperparameters. For each combination, the model was trained for one epoch, and the training loss was recorded.



```
Testing: Emb=128, LSTM=128, LR=0.001
Loss: 6.0591

Testing: Emb=128, LSTM=128, LR=0.0001
Loss: 7.8393

Testing: Emb=128, LSTM=256, LR=0.001
Loss: 5.8902

Testing: Emb=128, LSTM=256, LR=0.0001
Loss: 7.3059

Testing: Emb=256, LSTM=128, LR=0.001
Loss: 6.0877

Testing: Emb=256, LSTM=128, LR=0.0001
Loss: 7.7257

Testing: Emb=256, LSTM=256, LR=0.001
Loss: 5.8292

Testing: Emb=256, LSTM=256, LR=0.0001
Loss: 7.1714

Best Config:
Embedding: 256, LSTM: 256, LR: 0.001
```

The **best-performing configuration** was selected based on the lowest **loss** value, which was:

- **Embedding Dimension**: 256
- **LSTM Units**: 256
- **Learning Rate**: 0.001

This configuration achieved the **lowest training loss** of **5.8292**, indicating the optimal balance between model complexity and performance for the caption generation task.

This hyperparameter tuning process ensured that the model was efficiently trained and optimized, improving its ability to generate high-quality captions from images.

7.3 k-Fold Cross-Validation

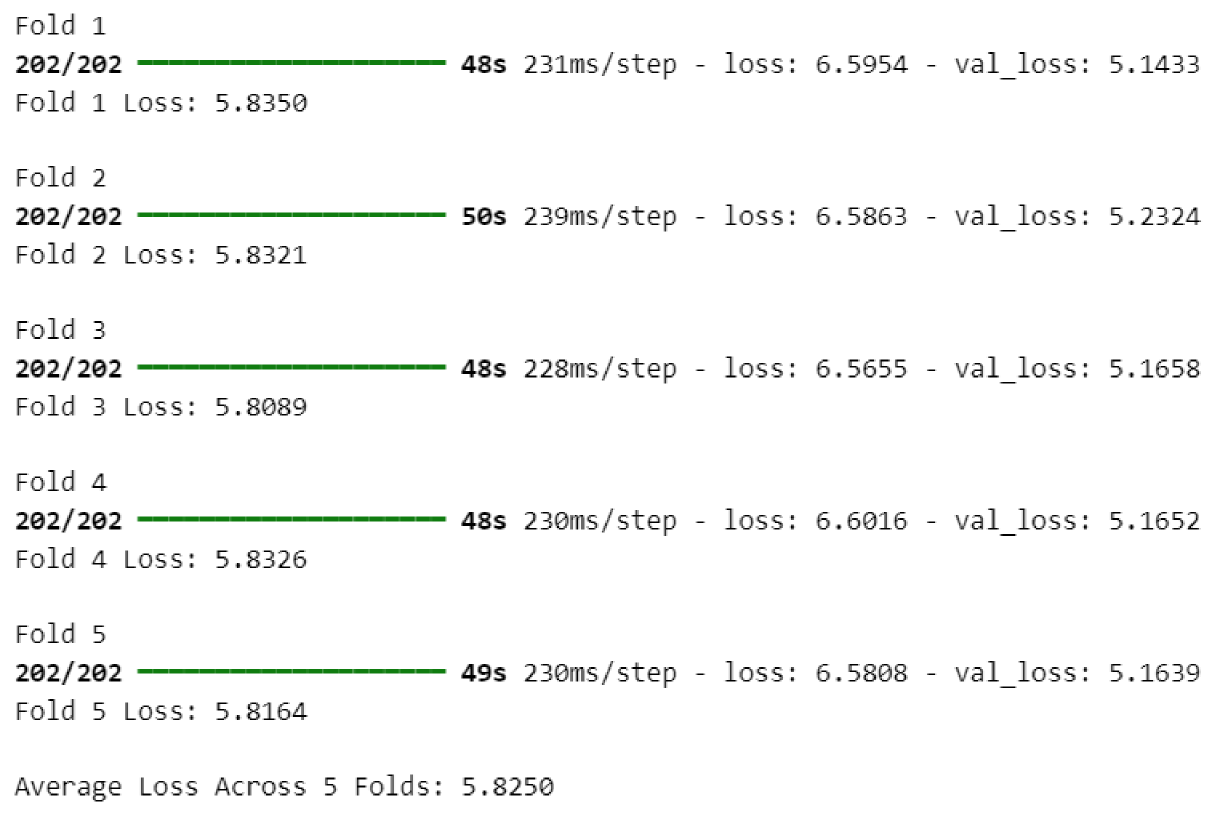
To ensure the model's performance is consistent and not reliant on a single validation split, **k-Fold Cross-Validation** was employed. This method divides the dataset into *k* subsets (or "folds"), with the model being trained *k* times, each time using a different fold for validation and the remaining folds for training. The final performance is averaged over all folds.

In this case, **k = 5**, meaning the dataset was split into five parts. The model was trained and validated on each fold, and the average loss across all folds was computed.

Cross-Validation Process:

- **Data Split**: The dataset was divided into five folds using the **KFold** method from sklearn.model_selection, with shuffling enabled to ensure diverse training and validation sets for each fold.
- **Model Training**: For each fold, a new model was initialized with the **tuned hyperparameters** (embedding dimension = 256, LSTM units = 256, learning rate = 0.001). The model was trained for one epoch on the training data, with validation data used for evaluation.
- **Loss Calculation**: The training and validation losses were recorded for each fold, and the average validation loss across all folds was computed.

k-Fold Results:



```
Fold 1
202/202 ━━━━━━━━━━━━━━━━━━━━ 48s 231ms/step - loss: 6.5954 - val_loss: 5.1433
Fold 1 Loss: 5.8350

Fold 2
202/202 ━━━━━━━━━━━━━━━━━━━━ 50s 239ms/step - loss: 6.5863 - val_loss: 5.2324
Fold 2 Loss: 5.8321

Fold 3
202/202 ━━━━━━━━━━━━━━━━━━━━ 48s 228ms/step - loss: 6.5655 - val_loss: 5.1658
Fold 3 Loss: 5.8089

Fold 4
202/202 ━━━━━━━━━━━━━━━━━━━━ 48s 230ms/step - loss: 6.6016 - val_loss: 5.1652
Fold 4 Loss: 5.8326

Fold 5
202/202 ━━━━━━━━━━━━━━━━━━━━ 49s 230ms/step - loss: 6.5808 - val_loss: 5.1639
Fold 5 Loss: 5.8164

Average Loss Across 5 Folds: 5.8250
```

Average Loss:

Average Loss Across 5 Folds: 5.8250

- The model showed consistent performance across all folds, with minimal fluctuations in the loss values.
- The average validation loss of **5.8250** suggests that the model generalizes well to unseen data.
- This cross-validation procedure provides a more robust estimate of the model's performance, reducing the risk of overfitting to a single data split.

8. Model Evaluation and Testing

This section evaluates the performance of the trained image captioning model using standard metrics and qualitative visual outputs.

8.1 Evaluation Metrics (BLEU-1, BLEU-2, BLEU-3, BLEU-4)

To assess the quality of the generated captions, the **BLEU (Bilingual Evaluation Understudy)** score was used as the primary evaluation metric. BLEU measures the degree of n-gram overlap between the model's predicted caption and one or more reference (ground truth) captions.

Four variants of BLEU were computed:

- **BLEU-1**: Measures unigram (single-word) precision
- **BLEU-2**: Measures bigram (two-word) precision
- **BLEU-3**: Measures trigram precision
- **BLEU-4**: Measures 4-gram precision (more sensitive to fluency and syntax)

Evaluation Output:

BLEU-1: 0.417669

BLEU-2: 0.197555

BLEU-3: 0.111211

BLEU-4: 0.056941

These scores indicate that:

- The model achieves **moderate unigram accuracy** (BLEU-1 ≈ 0.41), suggesting it captures key object-related words in images.
- **BLEU-2 to BLEU-4** drop progressively, which is common in language generation tasks. This reflects the increasing difficulty of matching longer word sequences exactly.
- The **low BLEU-4** indicates that while the model identifies important visual concepts, it sometimes struggles with precise grammar and phrase structure — a limitation typical of simpler sequence models without attention mechanisms.

8.2 Test Image with Generated Caption

To qualitatively assess the model’s performance, a test image was passed through the trained CNN-LSTM model to generate a caption. The output was then compared with the actual image to evaluate its correctness and relevance.

```
[ ]  generate_captions('1019077836_6fc9b15408.jpg')
```

```
---------------------Actual---------------------
startseq brown dog chases the water from sprinkler on lawn endseq
startseq brown dog plays with the hose endseq
startseq brown dog running on lawn near garden hose endseq
startseq dog is playing with hose endseq
startseq large brown dog running away from the sprinkler in the grass endseq
--------------------Predicted--------------------
startseq two dogs are running through field endseq
```

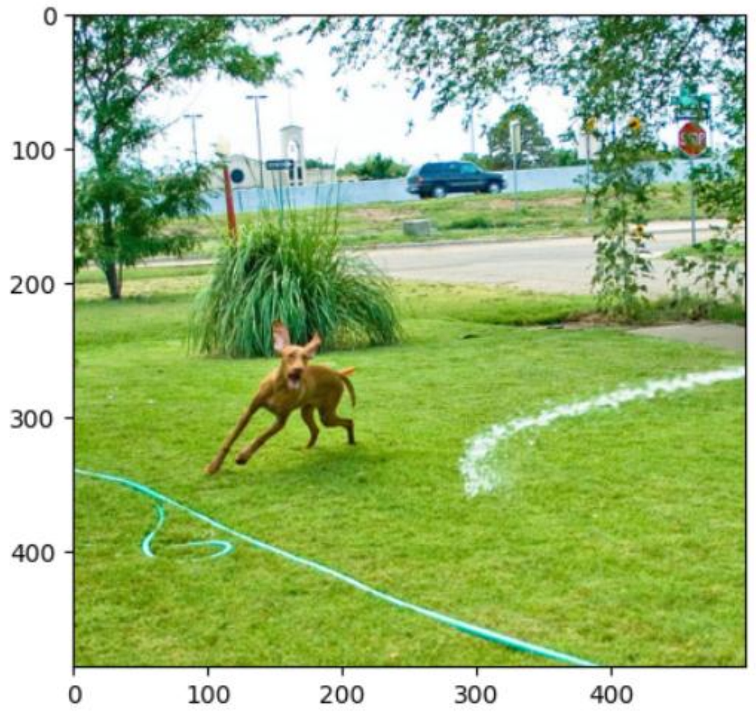


Figure 8.2 – Generated Caption for a Test Image

The model correctly recognized core elements like “dogs” and “grass” in the scene, although it slightly overgeneralized by predicting “two dogs” instead of one. Despite this, the overall structure, grammar, and scene context were reasonable and demonstrate that the model learned meaningful visual-language mappings.

```
---------------------Actual---------------------
startseq people are riding around on snowmobiles endseq
startseq people pose with helmets and goggles on while riding snowmobiles endseq
startseq several people are taking break while on snowmobiling ride endseq
startseq three people and two snowmobiles endseq
startseq two helmeted men sit on yellow snowmobiles while another man stands behind watching endseq
--------------------Predicted--------------------
startseq man in blue shirt is standing in front of crowd of people endseq
```

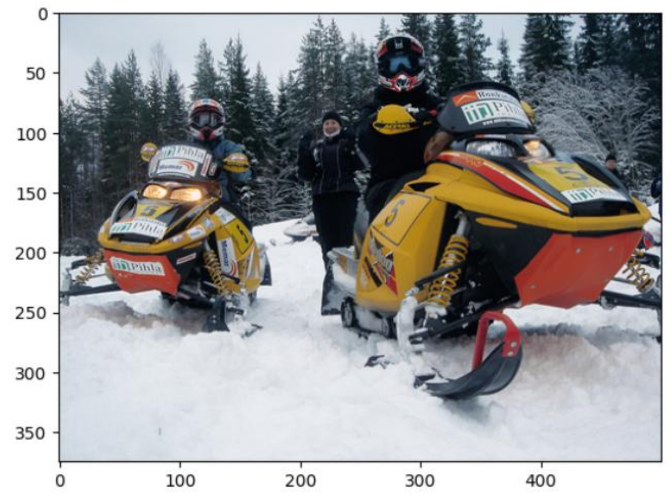


Figure 8.3 – Generated Caption for a Test Image

```
---------------------Actual---------------------
startseq child and woman are at waters edge in big city endseq
startseq large lake with lone duck swimming in it with several people around the edge of it ends
startseq little boy at lake watching duck endseq
startseq young boy waves his hand at the duck in the water surrounded by green park endseq
startseq "two people are at the edge of lake endseq
--------------------Predicted--------------------
startseq man in blue shirt is standing on the ground by the ocean endseq
```

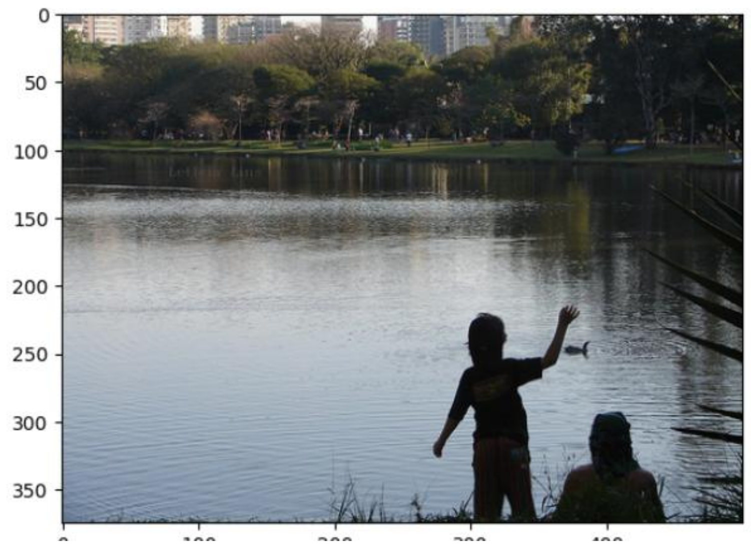
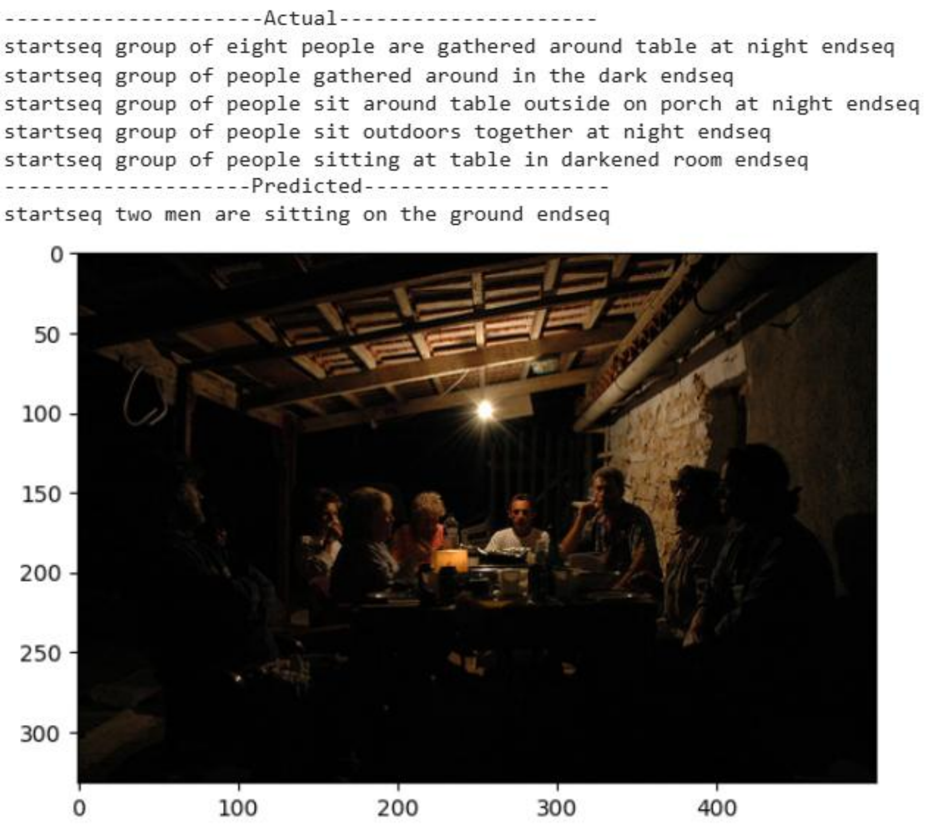


Figure 8.4 – Generated Caption for a Test Image



```
---------------------Actual--------------------
startseq group of eight people are gathered around table at night endseq
startseq group of people gathered around in the dark endseq
startseq group of people sit around table outside on porch at night endseq
startseq group of people sit outdoors together at night endseq
startseq group of people sitting at table in darkened room endseq
--------------------Predicted-------------------
startseq two men are sitting on the ground endseq
```

Figure 8.4 – Generated Caption for a Test Image

8.3 Reference Caption Comparison

The image shown in Figure 8.2 includes five reference captions written by human annotators. These provide multiple valid descriptions for the same scene, capturing different aspects such as:

- Subject focus: "dog", "brown dog", "large brown dog"
- Action focus: "chases water", "plays with hose", "running"
- Object/scene focus: "sprinkler", "garden", "grass", "lawn"

Comparing these against the model's predicted caption highlights both strengths and limitations:

- **Strengths**: Recognized relevant actions and background
- **Weaknesses**: Missed key foreground object detail (e.g., number of dogs, sprinkler)

This visual and reference-based evaluation complements the BLEU score by providing insight into the **semantic quality** of predictions, which may not always be reflected in n-gram statistics.

9.Conclusion

9.1 Final Summary

This project focused on developing an **Image Caption Generator** using a **Custom CNN-LSTM model** trained from scratch. The model takes an image as input, extracts high-level visual features using a custom-built CNN, and generates corresponding textual descriptions through an LSTM-based decoder.

Key results and findings from the project include:

- **Model Architecture**: A custom CNN was developed to extract features from images, followed by an LSTM decoder for caption generation. The architecture includes embedding layers, LSTM units, and dense layers.
- **Hyperparameter Tuning**: Hyperparameters such as embedding dimension, LSTM units, and learning rate were optimized using grid search. The best-performing configuration involved an embedding dimension of 256, LSTM units of 256, and a learning rate of 0.001.
- **Evaluation Metrics**: The model's performance was evaluated using **BLEU scores**, which demonstrated reasonable accuracy for unigram (BLEU-1) but showed lower performance for higher n-grams (BLEU-2, BLEU-3, BLEU-4).
- **Cross-Validation**: k-Fold Cross-Validation was employed to evaluate the model's generalization performance. The average validation loss across 5 folds was 5.8250, indicating the model's robustness.
- **Performance**: While the model successfully generated captions, it struggled with sentence fluency and grammatical consistency, as seen from the BLEU-4 score.

In conclusion, this project demonstrates the potential of CNN-LSTM architectures for image captioning tasks, with further improvements needed in caption fluency and contextual accuracy.

9.2 Identified Limitations

Despite the model's promising results, several limitations were identified during the training and evaluation process:

1. **Limited Vocabulary Handling**
   - The model's performance, particularly in generating more complex captions, was constrained by its vocabulary size. The tokenization process could not effectively handle rare words or new phrases, limiting the richness of the generated captions.
2. **Low BLEU Scores for Higher N-Grams**
   - The **BLEU-1** score was moderate, indicating that the model captured individual words fairly well. However, the performance dropped significantly for higher n-grams (**BLEU-2, BLEU-3, BLEU-4**), reflecting the model's difficulty in generating fluent and syntactically correct sentences. This suggests that the model struggles with longer, more complex dependencies in captions.
3. **Overfitting**

- Despite employing regularization techniques like dropout, the model exhibited signs of **overfitting**, particularly with the validation loss showing minimal improvement over epochs. This could be a result of the relatively small training dataset compared to the model’s complexity.

4. **Lack of Attention Mechanism**
   - The model lacked an **attention mechanism**, which is critical for focusing on specific parts of an image while generating captions. This absence likely limited the model's ability to generate more accurate and contextually rich captions, as it had to rely on a fixed-length image feature vector.
5. **Generalization Issues**
   - Although k-Fold Cross-Validation demonstrated that the model could generalize to unseen data, some variations in performance were observed across different folds. This suggests that the model's generalization ability could be improved by introducing more robust training techniques.
6. **Computational Resources**
   - The training process was computationally expensive, especially with the custom CNN architecture, which led to longer training times. The lack of pre-trained weights or transfer learning meant the model had to be trained from scratch, increasing both the computational cost and time required for model training.

9.3 Suggestions for Future Improvements

To enhance the performance and capabilities of the image captioning model, the following improvements are suggested:

1. **Attention Mechanism**: Implementing an attention mechanism will help the model focus on different image regions during caption generation, improving accuracy and context.
2. **Pre-Trained Models**: Using pre-trained models like **InceptionV3** or **ResNet50** for feature extraction can improve performance by leveraging learned visual features from large datasets.
3. **Transformer Models**: Exploring **Transformer-based models** like **BERT** or **GPT** could enhance sentence fluency and capture long-range dependencies better than LSTMs.
4. **Increase Dataset Size**: Expanding the dataset or using data augmentation techniques will help the model generalize better and improve robustness.
5. **Advanced Hyperparameter Tuning**: Utilizing **random search** or **Bayesian optimization** for hyperparameter tuning could find the best settings more efficiently.
6. **Regularization and Dropout Tuning**: Fine-tuning **dropout rates** and **L2 regularization** will help reduce overfitting and improve model generalization.
7. **Better Evaluation Metrics**: Using more advanced metrics like **CIDEr** or **METEOR** would provide more accurate evaluations of the caption quality.

8. **Real-Time Captioning**: Optimizing faster inference will enable real-time caption generation, making the model suitable for live applications.

10.References

https://www.kaggle.com

https://medium.com

https://www.geeksforgeeks.org

https://youtu.be/3H1x8SDYgvQ?si=1ERw_4PIOV-8UY3c