CS6910: Deep Learning
Assignment 1

Team 7
CS21B007 Arcot Renusree
CS21B036 Janapati Varshita Devi
ED21B042 Naidu Sushant Chandra

April 7, 2025

1 Task 1: Transfer learning using VGGNet/GoogLeNet

1.1 Objective

The objectives of Task 1 are:

- **Download a pre-trained model for Transfer Learning:** Utilize a model pre-trained on a large dataset, such as ImageNet, to leverage its learned features.
- **Fine-tune the pre-trained model on a custom dataset:** Adapt the model to a specific task by training it on a new dataset, adjusting its architecture (Top layer) as needed.

1.2 Dataset Description

The dataset consists of colored images categorized into five classes. Each class corresponds to a unique WordNet ID (used as folder names) and represents a specific object. The dataset structure is suitable for classification tasks and is processed using PyTorch's `ImageFolder` format.

- **Number of classes:** 5
- **Class labels and corresponding object names:**
  - `n01685808` – Lizard
  - `n01729977` – Snake
  - `n01775062` – Spider
  - `n01798484` – Bird (likely a partridge)
  - `n01930112` – Microorganism / Protist (e.g., Euglena or Amoeba)
- **Number of training images:** 3500
- **Number of test images:** 700
- **Original image dimensions:** 224×224 pixels
- **Image dimensions used during training:** 224×224 pixels

Figure 1.1: Sample images from each class: Lizard, Snake, Spider, Bird, and Microorganism.

1.3 VGGNet

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102,764,544
dense_1 (Dense)	(None, 1072)	4,391,984
dropout (Dropout)	(None, 1072)	0
dense_2 (Dense)	(None, 5)	5,365

Total params: 121,876,581 (464.92 MB)
Trainable params: 107,161,893 (408.79 MB)
Non-trainable params: 14,714,688 (56.13 MB)

Figure 1.2: VGGNet architecture attached to an MLFFNN with two hidden layers

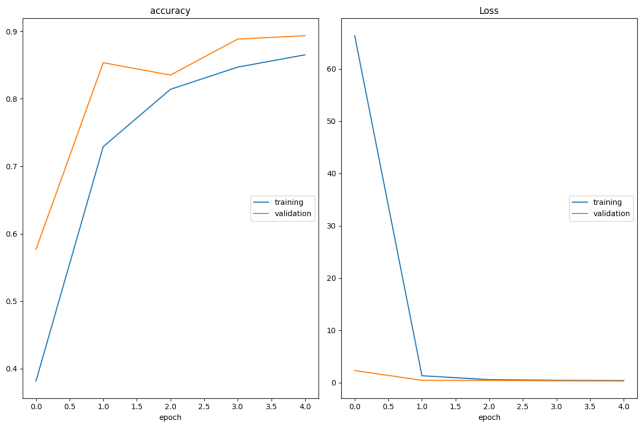


Figure 1.3: Loss and accuracy plots for VGGNet during training of top MLFFNN

1.4 GoogLeNet

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21,802,784
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 256)	13,107,456
batch_normalization_94 (BatchNormalization)	(None, 256)	1,024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32,896
batch_normalization_95 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645

Total params: 34,945,317 (133.31 MB)
Trainable params: 13,141,765 (50.13 MB)
Non-trainable params: 21,803,552 (83.17 MB)

Figure 1.4: GoogLeNet architecture attached to an MLFFNN with two hidden layers

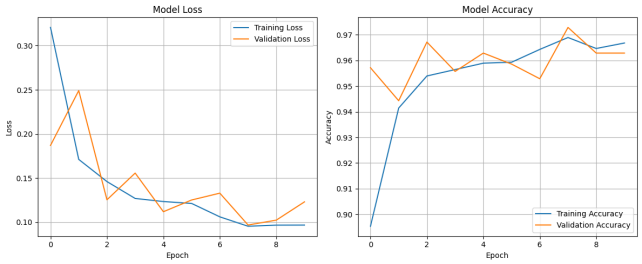


Figure 1.5: Loss and accuracy plots for GoogLeNet model with training only the top MLFFNN

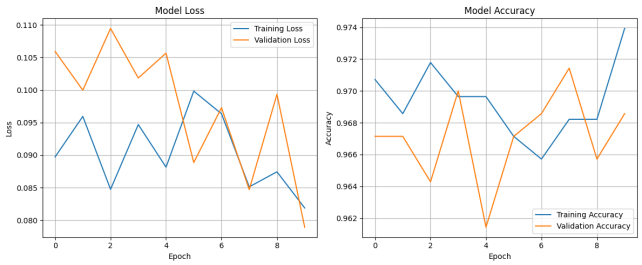


Figure 1.6: Loss and accuracy plots for GoogLeNet model during learning of top and some inner layers of the pre-trained model

1.5 Analysis of Results

The performance metrics indicate that GoogleNet has a higher accuracy compared to VG-GNet. This suggests that GoogleNet is more effective in correctly classifying images, making it a more reliable model for tasks requiring high precision. The superior performance of GoogleNet can be attributed to its innovative architecture:

- **Inception Modules**: GoogleNet's use of Inception modules allows the model to capture multi-scale features by performing convolutions with different parallel filter sizes (1x1, 3x3, 5x5). This capability enhances the model's ability to detect and recognize complex patterns in the data, leading to higher accuracy.
- **Efficiency**: Despite its complex architecture, GoogleNet is designed to be computationally efficient, with fewer parameters compared to VGGNet. This reduces the risk of overfitting, which results in better accuracy.

1.6 Learning Outcomes

- **Utilizing Pre-trained Models**: Learned to leverage pre-trained models to reduce training time and improve accuracy.
- **Transfer Learning**: Applied transfer learning to use knowledge from pre-trained models.

This task provided a comprehensive learning experience in using and fine-tuning pre-trained deep-learning models for image classification. By integrating VGGNet and GoogleNet with a top MLFFNN, We learned to utilize pre-trained models for improved classification accuracy. Additionally, We acquired skills in importing models in TensorFlow, adding custom layers, and fine-tuning the model for a specific dataset, enhancing our understanding of transfer learning and model integration for future projects.

2 Task 2: Image classification using a CNN

2.1 Dataset Description

The dataset consists of colored images categorized into five classes. Each class corresponds to a unique WordNet ID (used as folder names) and represents a specific object. The dataset structure is suitable for classification tasks and is processed using PyTorch's `ImageFolder` format.

- **Number of classes:** 5
- **Class labels and corresponding object names:**
  - `n01685808` – Lizard
  - `n01729977` – Snake
  - `n01775062` – Spider
  - `n01798484` – Bird (likely a partridge)
  - `n01930112` – Microorganism / Protist (e.g., Euglena or Amoeba)
- **Number of training images:** 3500
- **Number of test images:** 700
- **Original image dimensions:** 224 $\times$ 224 pixels
- **Image dimensions used during training:** 224 $\times$ 224 pixels

Figure 2.1 shows one representative image from each class.

Figure 2.1: Sample images from each class: Lizard, Snake, Spider, Bird, and Microorganism.

2.2 Model Architecture

Layer Configuration

- **CL1:** Convolutional layer with 4 filters, kernel size 3×3, stride 1, ReLU activation
- **PL1:** Mean pooling layer with kernel size 2×2 and stride 2
- **CL2:** Convolutional layer with 8 filters, kernel size 3×3, stride 1, ReLU activation
- **PL2:** Mean pooling layer with kernel size 2×2 and stride 2
- **Flatten:** Output from PL2 is flattened
- **FC:** Fully connected layer with output size equal to number of classes (5)

2.3 Hyperparameters

The model was trained using the following hyperparameters:

- **Loss Function:** CrossEntropyLoss
- **Optimizer:** Adam
- **Learning Rate:** 0.001
- **Batch Size:** 32
- **Number of Epochs:** 100
- **Weight Initialization:** PyTorch default

Note: *CrossEntropyLoss* is suitable for multi-class classification tasks, making it an appropriate choice for this dataset which contains five distinct classes. The *Adam optimizer* was selected for its adaptive learning rate properties and robustness, which helps accelerate convergence and improve generalization on image classification problems.

2.4 Loss vs Epoch Plot

The Loss vs Epoch plot in Figure 2.2 demonstrates the model's convergence behavior. Initially, the loss drops sharply, indicating that the model is learning quickly from the training data. Around epoch 30, the loss begins to plateau, suggesting that the model has reached a stable state of learning. However, a very slow decrease after that indicates diminishing returns in training improvement, and possibly the onset of overfitting.

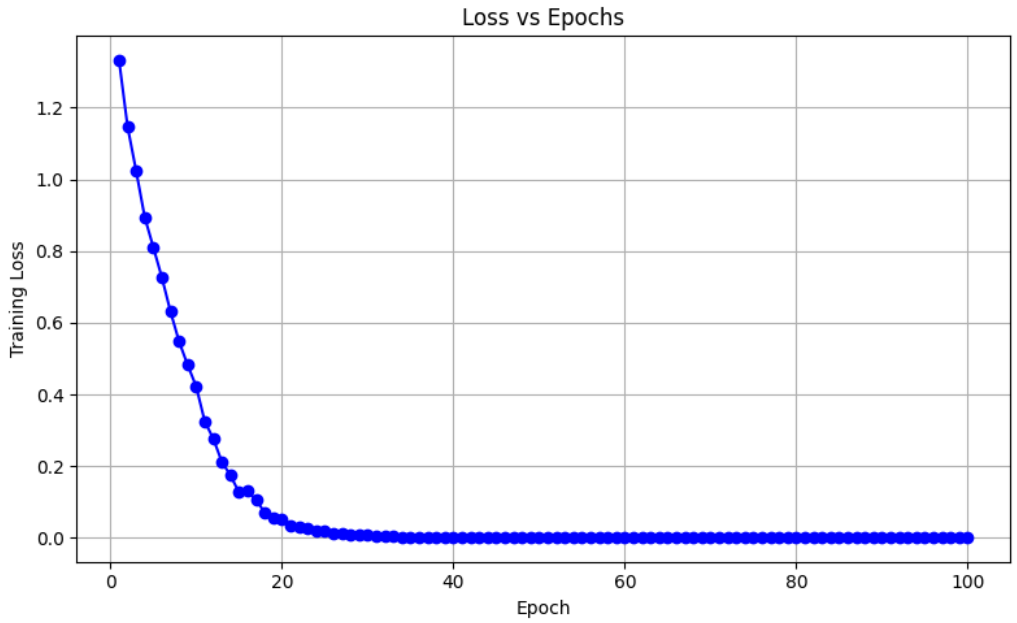


Figure 2.2: Training Loss vs Epochs

2.5 Results

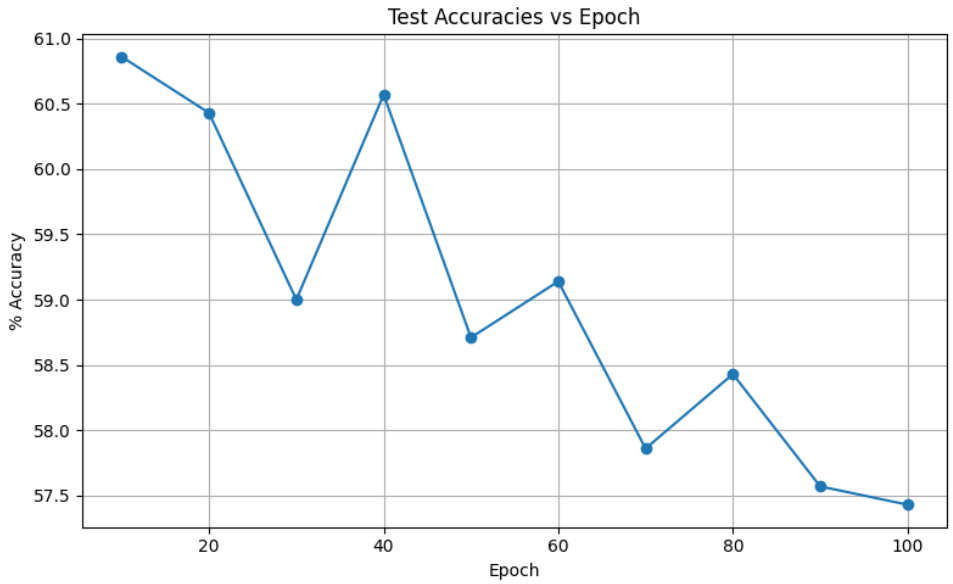
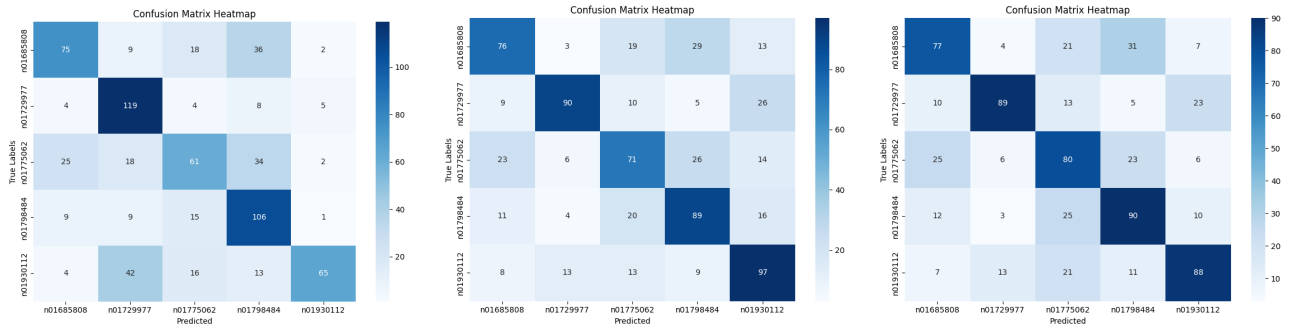


Figure 2.3: Test Accuracies Vs Epoch for Image classification Dataset

As shown in Figure 2.3, the model achieved its highest test accuracy of 60.86% at epoch 10. Post this point, accuracy declined, suggesting that longer training does not yield better generalization. This further confirms the importance of incorporating early stopping mechanisms to prevent overfitting. Additional techniques such as learning rate scheduling or using a validation set for monitoring could help in selecting the optimal epoch for model deployment.

2.6 Confusion Matrix



(a) Epoch 10 (b) Epoch 20 (c) Epoch 40

Figure 2.4: Confusion Matrices for Different Epochs

The confusion matrices in Figure 2.4 provide insights into the per-class performance of the model. Notably, certain classes exhibit strong confusion throughout training, such as:

- **n01930112**: This class is frequently misclassified as **n01729977** and **n01775062**, suggesting poor separability in feature space.
- **n01775062**: Shows improvement in classification accuracy by epoch 40, though some confusion remains with **n01685808**.

Across all epochs, misclassification patterns remain relatively consistent, indicating that certain class pairs are inherently difficult for the model to distinguish. This might be due to visual similarity or lack of distinctive features in the training images.

2.7 Observations

The evaluation results highlight a typical behavior in deep learning models where performance on the training data continues to improve, but the test accuracy peaks early. In this experiment, test accuracy peaked at epoch 10 and gradually declined afterward. This indicates that the model starts to overfit after a certain point, learning noise and specific patterns in the training data that don't generalize to unseen data.

3 Task 3 : Sentiment Analysis using RNN

3.1 Objective

This project focuses on performing sentiment analysis on tweet data using a recurrent neural network (RNN) with a single hidden layer of 25 nodes. The model leverages pre-trained GloVe embeddings (200 dimensions) to represent words and uses hyperparameter tuning to optimize key training parameters such as dropout, recurrent dropout, and learning rate.

3.2 Dataset and Preprocessing

Dataset Overview

- **Training Data:** 5600 samples

- **Test Data:** 1400 samples

Descriptive Statistics for Tweet Length

- **Training Data:**
  - Mean: 12.47 words
  - Standard Deviation: 5.43
  - Range: 1 to 29 words
- **Test Data:**
  - Mean: 12.35 words
  - Standard Deviation: 5.35
  - Range: 2 to 29 words

These statistics indicate that the tweets are relatively short, with most tweets containing between 8 and 16 words. Preprocessing involved standard text cleaning, tokenization, and sequence padding to ensure uniform input length.

3.3 Model Architecture

3.3.1 Embedding Layer

GloVe Embeddings:
The model begins by loading pre-trained 200-dimensional GloVe embeddings. A helper function loads the embeddings from the file, and any word not found in the GloVe dictionary is assigned a vector sampled from a normal distribution.

3.3.2 RNN Layer

- **Hidden Layer:**
  A single SimpleRNN layer with 25 nodes was used.
- **Dropout Parameters:**
  Two types of dropout were employed:
  - **Dropout:** Applied to the inputs of the RNN layer.
  - **Recurrent Dropout:** Applied to the recurrent state transitions.

3.3.3 Output Layer

Dense Layer:
A fully connected layer with a sigmoid activation function producing the binary classification output.

3.3.4 Model Compilation

- **Loss Function:** Binary Crossentropy
- **Optimizer:** Adam with a hyperparameter-tuned learning rate

Metrics: Accuracy

3.4 Hyperparameter Tuning

A hyperparameter tuning pipeline was set up using the Hyperband tuner from Keras Tuner. The tuning process focused on three hyperparameters:

- **Dropout:** Choices included 0.2, 0.3, and 0.4.
- **Recurrent Dropout:** Choices included 0.2, 0.3, and 0.4.
- **Learning Rate:** Choices included 0.001, 0.0005, and 0.0001.

3.4.1 Tuning Process

- **Method:** Hyperband with early stopping based on validation loss (patience of 3 epochs).
- **Trials:** The top 10 best trials were recorded, with performance scores (validation accuracy) ranging from approximately 0.9348 to 0.9411.

3.4.2 Best Hyperparameters

The optimal configuration identified was:

- **Dropout:** 0.2
- **Recurrent Dropout:** 0.2
- **Learning Rate:** 0.001

This configuration was then used to rebuild and retrain the final model.

3.5 Model Training and Evaluation

3.5.1 Training

- **Epochs:** The best model was trained for up to 50 epochs with early stopping.
- **Validation:** Performance was continually monitored on a validation set to avoid overfitting.

3.5.2 Evaluation on Test Data

The final model evaluation on the test set produced the following metrics:

- **Test Accuracy:** 0.9314
- **F1 Score:** 0.9126
- **Precision:** 0.9193
- **Recall:** 0.9314

These results indicate that the model performs robustly in distinguishing sentiment in tweets.

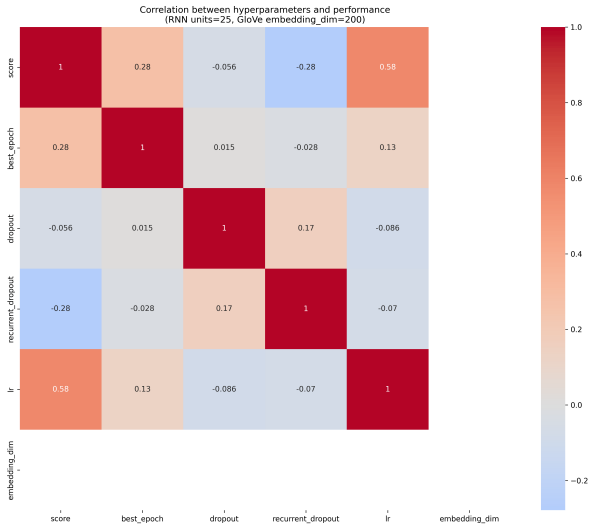


Figure 3.1: Heatmap demonstrating correlation between hyperparameters and performance

3.6 Hyperparameter Effects

A detailed analysis was conducted to understand the impact of each hyperparameter on model performance:

- **Dropout Effect:**

  Mean validation accuracies ranged from approximately 0.92398 to 0.92857, with a slight variation across dropout values. The best performance was observed with a dropout rate of 0.2.

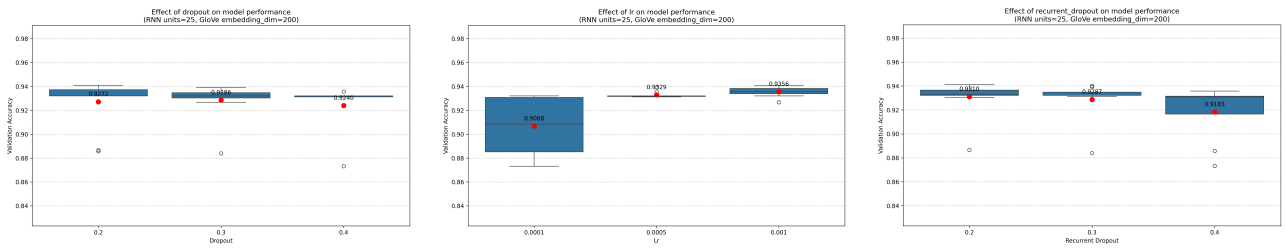
- **Recurrent Dropout Effect:**

  Validation accuracies were highest with a recurrent dropout rate of 0.2, with a mean score of about 0.93104, suggesting that lower recurrent dropout preserves crucial temporal information.

- **Learning Rate Effect:**

  The learning rate of 0.001 yielded the best overall performance (mean $\tilde{0}$.93563) compared to lower learning rates, underscoring the importance of a sufficiently high learning rate for convergence in this setup.

Visualizations (correlation heatmaps, dropout effect, learning rate effect, and recurrent dropout effect plots) along with the CSV file containing the hyperparameter trial results were generated to support these findings.



(a) dropout (b) learning rate (c) recurrent dropout

Figure 3.2: Boxplots of hyperparameter with performance

3.7 Conclusion

This project demonstrates that a sentiment analysis model using a single-layer RNN and pre-trained GloVe embeddings can achieve high performance with proper hyperparameter tuning. The key takeaways are:

- **Effective Use of Pre-trained Embeddings:** The 200-dimensional GloVe vectors provided a strong semantic foundation.
- **Model Simplicity:** A single RNN layer with 25 nodes, when carefully tuned, is sufficient to capture the sentiment in short tweets.
- **Hyperparameter Sensitivity:** Optimal settings (dropout: 0.2, recurrent dropout: 0.2, learning rate: 0.001) significantly impact model performance, as evidenced by the hyperparameter tuning results.
- **Robust Evaluation:** With test accuracy exceeding 93% and strong F1, precision, and recall scores, the model is well-calibrated for sentiment classification on the provided dataset.

Overall, the experiment shows the effectiveness of combining classical word embeddings with RNN architectures for natural language processing tasks in sentiment analysis.

4 Task 4 : Parts-of-speech tagging using RNN

4.1 Dataset and Preprocessing

- The dataset consists of sentences and their corresponding POS tag sequences. Each sentence is a space-separated string of words, and each tag sequence is a space-separated string of POS tags.
- Example:
  - **Sentence:** `She was ready to kill the beef , dress it out ...`
  - **Tags:** `PRON VERB ADJ PRT VERB DET NOUN . VERB PRON PRT ...`
- **Number of training images:** 3500
- **Number of test images:** 700
- **Preprocessing steps:**
  - Sentences are lowercased and split into tokens.
  - Tags are split using whitespace to align with tokens.
  - The same preprocessing steps are applied to the test set.
  - A vocabulary is constructed from training data, with special tokens:
    - `<UNK>` for unknown words
    - `<PAD>` for padding shorter sequences
- **POS Tags:** The set of POS tags present in the dataset: `[., ADJ, ADP, ADV, CONJ, DET, NOUN, NUM, PRON, PRT, VERB, X]`.

4.2 Descriptive Statistics for Sentence Length

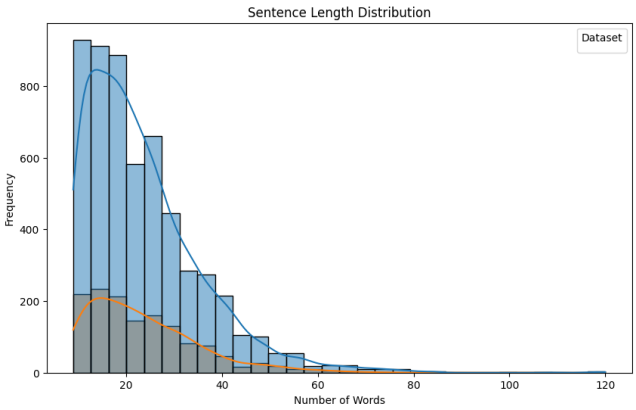
Training Data:

- **Mean:** 23.55 words
- **Standard Deviation:** 12.17
- **Range:** 9 to 120 words

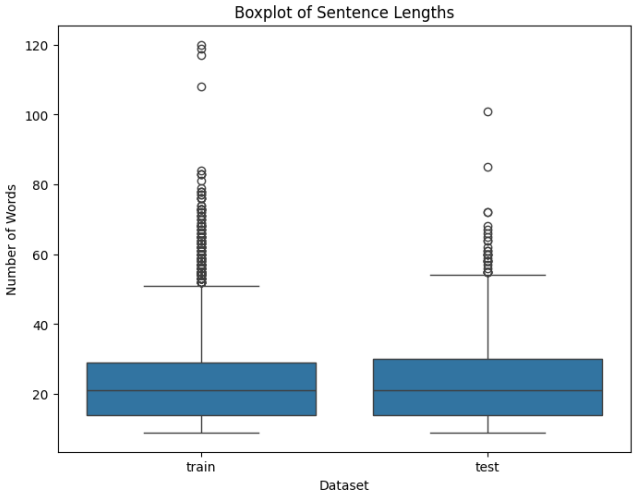
Test Data:

- **Mean:** 23.52 words
- **Standard Deviation:** 11.69
- **Range:** 9 to 101 words

These statistics indicate that the sentences are moderately long, with the majority containing between 14 and 30 words. However, there are a few significantly longer sentences, as suggested by the high maximum values and standard deviation. We use sequence padding to standardize input lengths for model training and batch processing.



(a) Sentence Length Distribution for POS dataset



(b) Boxplot of Sentence Lengths for POS dataset

Figure 4.1: Statistics for Sentence Length

4.3 Model Architecture

- **Word Embeddings:** Pre-trained GloVe embeddings with 200 dimensions are used to represent each word as a dense vector, capturing semantic similarity.
- **RNN Layer:** A single hidden layer RNN with 25 hidden units processes word sequences and captures temporal dependencies.
- **Output Layer:** A linear layer maps RNN outputs to the POS tag space.

4.4 Training Setup

- **Loss Function:** CrossEntropyLoss with `ignore_index = -100` to exclude padded tokens.
- **Optimizer:** Adam optimizer is used to train the model efficiently.
- **Evaluation Metrics:**
  - Avg token-wise Loss
  - Test and Train Accuracy
  - F1-Score per POS tag

Results

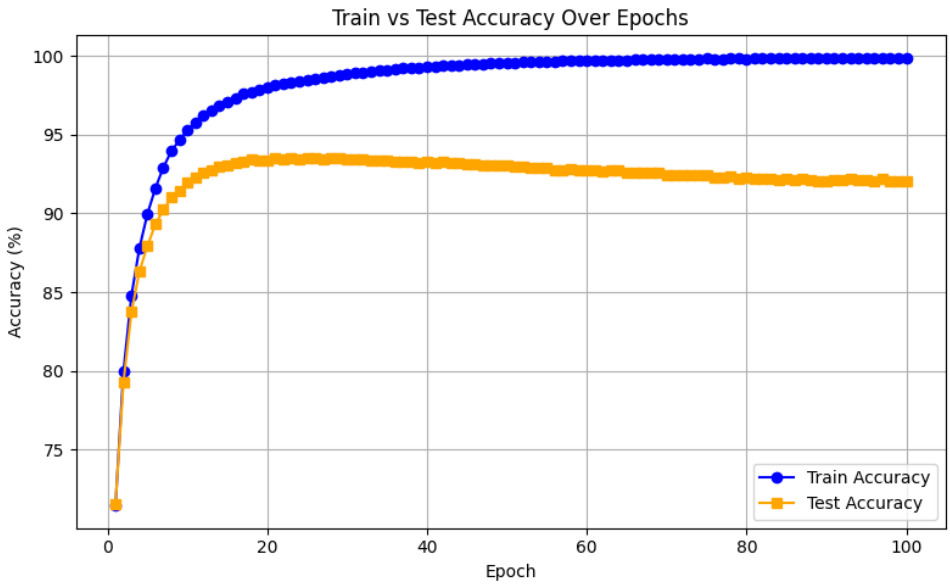


Figure 4.2: Train vs Test Accuracies over epochs

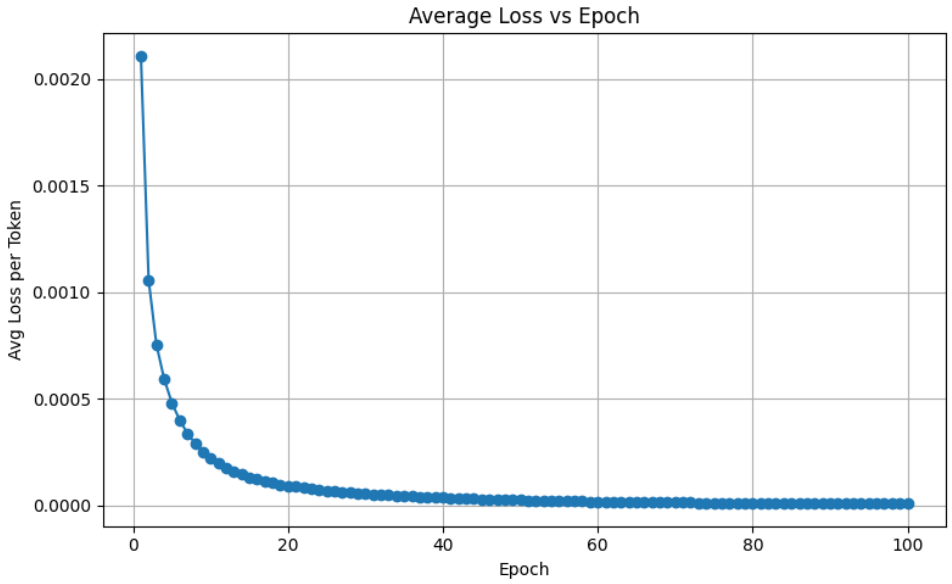


Figure 4.3: Avg Loss per Token

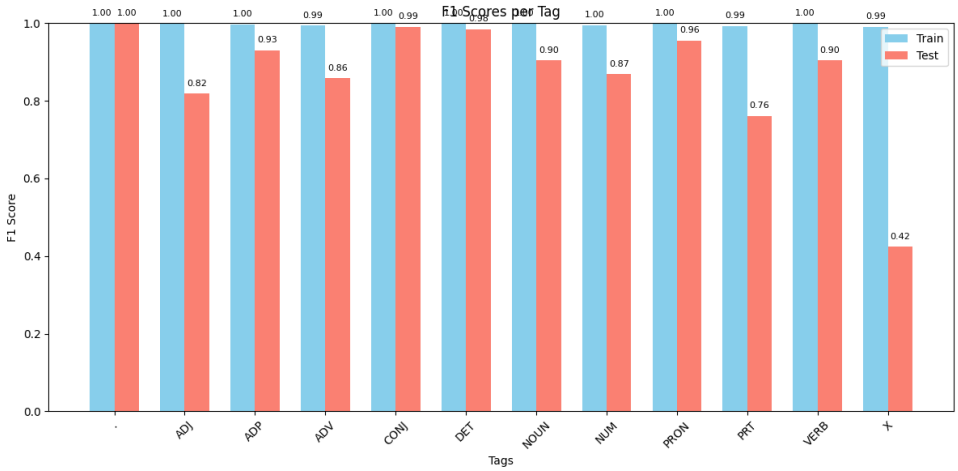


Figure 4.4: F1 scores for all Tags after training for 100 epochs

4.5 Observations

- **F1 Scores per Tag:**
  - Training F1 scores are nearly perfect (≈ 1.00) for all POS tags.
  - Test F1 scores are slightly lower, with significant drops for certain tags:
    - `X` tag has the lowest test F1 score (≈ 0.42), indicating poor generalization.
    - Tags like `PRT`, `ADJ`, and `NUM` also show noticeable gaps between train and test scores.
  - These gaps suggest overfitting on the training data, especially for tags with low frequency or ambiguous usage.
- **Average Loss per Token vs Epoch:**
  - The average loss decreases sharply during the initial epochs.
  - Loss stabilizes and converges near zero after approximately 50 epochs.

  - The final low loss value indicates the model has effectively minimized the error on the training data.

- **Train vs Test Accuracy Over Epochs:**
  - Training accuracy steadily increases to nearly 100%.
  - Test accuracy peaks around 93% and starts to plateau or slightly decline after 20 epochs.
  - The growing gap between train and test accuracy indicates overfitting.

4.6 Conclusions

- The model fits the training data very well but shows limited generalization capability on the test set.
- Overfitting is evident from:
  - High training accuracy versus lower test accuracy.
  - Disparity in F1 scores between train and test, especially for less frequent tags.
- The poor performance on the `X` tag suggests a need for better handling of rare or noisy labels.
- Despite some overfitting, the model achieves high accuracy and strong F1 scores for the majority of POS tags.

5 Task 5: Image Captioning Task with RNN

5.1 Dataset and Problem Statement

The task is to generate captions for images using an Encoder-Decoder architecture. Image features are extracted using a pretrained VGGNet model and are passed through a linear layer to ensure their dimensions match the RNN's hidden state. These transformed features are then used to initialize the hidden state of the decoder. The decoder consists of a single-hidden-layer RNN with 50 nodes in the hidden layer. It is in missing external inputs—during training, the output at each time step t is used as the input at time step t+1, continuing this process for a sequence of length T.

Coming to dataset, It comprises of 4000 images and each image have 5 different captions making it a dataset of 20000 examples. 80:20 train test split is made to create train and test datasets for the task.

5.2 Model Architecture

- **Encoder (CNN-based):** A pre-trained VGG16 model is used to extract deep visual features from input images. The output of the final convolutional layer is average-pooled across spatial dimensions to produce a 512-dimensional feature vector summarizing the image content.
- **Word Embeddings:** Pre-trained GloVe embeddings with 200 dimensions are used to represent each word as a dense vector. These embeddings capture semantic relationships and remain fixed during training to retain their learned structure.
- **Decoder (RNN-based):** A single-layer RNN with 50 hidden units is used to decode the image representation into a sequence of words. The 512-dimensional image vector is transformed to match the hidden size and is used to initialize the decoder's hidden state.

- **Output Layer:** A linear layer maps each hidden state of the decoder to a vocabulary-sized score vector, from which the next word is predicted at each time step.
- **Training Objective:** The model is trained using cross-entropy loss, with padding tokens (<pad>) ignored. Evaluation is performed using BLEU scores (1–4) to assess the fluency and accuracy of generated captions.

5.3 Training Details

- **Training Dataset:** 16,000 image-caption pairs were used for training.
- **Training Mode:** Mini-batch training with a batch size of 160.
- **Loss Function:** Cross-entropy loss was used, ignoring the <pad> token.
- **Optimizer:** Adam optimizer was used
- **Epochs:** The model was trained for 3 epochs considering vanishing and exploding gradients problem and large dataset.
- **Evaluation Metric:** BLEU scores (BLEU-1 to BLEU-4) were computed to assess the quality of generated captions based on n-gram overlap with ground-truth captions.

5.4 Results

BLEU-k	**Training Score**	**Test Score**
BLEU-1	0.1568	0.1347
BLEU-2	0.0950	0.0720
BLEU-3	0.0563	0.0763
BLEU-4	0.0590	0.0420

Table 1: BLEU scores for $k = 1$ to 4 on training and test datasets

6 Task 6: Image Captioning Task with LSTM

6.1 Dataset and Problem Statement

The task objective is to generate natural language captions for images using an Encoder-Decoder architecture same as task 5. Image features are extracted using a pre-trained VGGNet model and passed through a linear transformation to match the LSTM decoder's hidden state dimensions. These image features are then used to initialize the hidden and cell states of the decoder. The decoder consists of a single-layer LSTM with 50 hidden units. During training, the decoder is auto-regressive—i.e., the output word at time step t is used as the input at time step $t + 1$, for a sequence of length T.

The dataset comprises 4000 images, each associated with 5 unique captions, resulting in 20,000 image-caption pairs. An 80:20 train-test split is used, yielding 16,000 training and 4,000 test examples.

6.2 Model Architecture

- **Encoder (CNN-based):** A pre-trained VGG16 model is used to extract deep features from the input images. These features are average-pooled to produce a 512-dimensional representation for each image.
- **Word Embeddings:** Pre-trained GloVe embeddings (200-dimensional) are used to represent words. These embeddings are fixed during training to preserve semantic information.
- **Decoder (LSTM-based):** A single-layer LSTM with 50 hidden units is used to decode the image features into a word sequence. The image feature vector is transformed to match the hidden size and is used to initialize both the hidden and cell states of the LSTM.
- **Output Layer:** A linear layer maps the LSTM's hidden states to vocabulary-sized logits, from which the most probable word is predicted at each timestep.
- **Training Objective:** The model is trained using cross-entropy loss, ignoring padding tokens (<pad>). BLEU scores (1–4) are used as evaluation metrics to assess the quality of the generated captions.

6.3 Training Details

- **Training Dataset:** 16,000 image-caption pairs
- **Batch Size:** 160
- **Loss Function:** Cross-entropy loss with <pad> token ignored
- **Optimizer:** Adam optimizer
- **Epochs:** 3 epochs (to mitigate issues such as vanishing/exploding gradients on a large dataset)
- **Evaluation Metric:** BLEU-1 to BLEU-4 scores computed on predicted captions

6.4 Results

BLEU-k	**Training Score**	**Test Score**
BLEU-1	0.2492	0.2507
BLEU-2	0.1819	0.1832
BLEU-3	0.1563	0.1575
BLEU-4	0.1381	0.1391

Table 2: BLEU scores for $k = 1$ to 4 on training and test datasets (LSTM-based model)

6.5 Comparison and Observations

The performance of the RNN-based and LSTM-based models was compared based on BLEU scores across both training and test datasets. It was observed that the LSTM model outperformed the RNN model at all levels of k (BLEU-1 to BLEU-4). This is expected, as LSTMs are more effective in capturing long-term dependencies and mitigating issues such as vanishing gradients, which are common in standard RNNs.

Summary of Observations:

- The LSTM model generated more coherent and fluent captions, with a higher n-gram overlap with ground-truth captions.
- BLEU scores decreased with increasing k for both models, reflecting the increased difficulty of matching longer n-gram sequences.
- The gap between training and test BLEU scores remained consistent, indicating minimal overfitting in both models.