CS6910
Fundamentals of Deep Learning

Programming Assignment 2

Aditya Mallick ee21b005@smail.iitm.ac.in
Ayushman Agarwal ee21b027@smail.iitm.ac.in
Srinivasan Kidambi ee21b139@smail.iitm.ac.in

Indian Institute of Technology, Madras

May 27, 2024

Contents

1 Task 1 **3**
- 1.1 Data . . . 3
- 1.2 Model . . . 3
- 1.3 Training . . . 3
- 1.4 Hyperparameters . . . 4
- 1.5 Comparison of Optimization Methods . . . 4
- 1.6 VGGNet . . . 6
- 1.7 GoogleNet . . . 7
- 1.8 Inferences . . . 9
- 1.9 Conclusion . . . 9

2 Task 2 **11**
- 2.1 Data . . . 11
  - 2.1.1 Training Data . . . 11
  - 2.1.2 Validation Data . . . 11
  - 2.1.3 Test Data . . . 11
- 2.2 Models . . . 11
- 2.3 Training . . . 12
- 2.4 Hyperparameters . . . 12
- 2.5 Tabulated Results . . . 13
- 2.6 Plots and Results for Optimal Model Configurations . . . 15
- 2.7 Inferences . . . 19
  - 2.7.1 Dependence on Batch Size . . . 19
  - 2.7.2 Dependence on Learning Rate . . . 19
  - 2.7.3 Dependence on Number of Feature Maps in Second Layer . . . 19
- 2.8 Conclusion . . . 20

3 Image Captioning **21**
- 3.1 Introduction . . . 21
- 3.2 Encoder . . . 21
- 3.3 Decoder . . . 21
- 3.4 Theory . . . 21
  - 3.4.1 NetVLAD . . . 21
  - 3.4.2 GloVe . . . 22
  - 3.4.3 Recurrent Neural Network . . . 22
  - 3.4.4 LSTM : Long Short Term Memory . . . 23

4 Task 3 and 4 **23**
- 4.1 Data . . . 23
  - 4.1.1 Training Data . . . 24
  - 4.1.2 Validation Data . . . 24
  - 4.1.3 Test Data . . . 24
- 4.2 Data Preprocessing . . . 24
- 4.3 Model . . . 24
  - 4.3.1 Pre-trained CNN . . . 25
  - 4.3.2 NetVLAD . . . 25
  - 4.3.3 Fully connected Layer . . . 25
  - 4.3.4 Decoder . . . 25

4.4 Training . . . 25
4.5 Hyperparameters . . . 27
4.6 Experimental Studies . . . 28
4.6.1 Choice of Embedding . . . 28
4.6.2 Choice of CNN . . . 28
4.6.3 Varying hidden and embedding dimensions . . . 28
4.7 Results for Optimal Model Configuration . . . 29
4.7.1 Prediction samples . . . 29
4.7.2 BLEU Scores . . . 37
4.8 Inferences . . . 37
4.9 Conclusion . . . 37

5 Machine Translation **38**
5.1 Introduction . . . 38
5.2 Encoder . . . 38
5.3 Decoder . . . 38
5.4 Training . . . 38
5.5 Inference . . . 38
5.6 Teacher Forcing . . . 38

6 Task 5 **39**
6.1 Data . . . 39
6.1.1 Training Data . . . 39
6.1.2 Validation Data . . . 39
6.1.3 Test Data . . . 39
6.2 Data Pre-Processing . . . 39
6.3 English Sentence Embedding . . . 40
6.4 Hindi Sentence Embedding . . . 40
6.5 Model . . . 40
6.6 Training . . . 40
6.7 Hyperparameters . . . 41
6.8 Tabulated Experimentation Results . . . 41
6.9 Results for Optimal Model Configuration . . . 42
6.10 Inferences . . . 43
6.11 Conclusion . . . 43

1 Task 1

Comparison of pretrained models used in transfer learning for classification of images.

1.1 Data

Training data consists of 2000 images equally distributed over 5 classes which have been identified visually. The test and validation datasets have 500 images each, which are again equally divided into sets of 100 images for each class. Therefore, we have a balanced dataset which is a good thing since our model gets equal to exposure to all the 5 classes while training.

1.2 Model

A an image classifying network has been created with the following parameters:

- Pre-trained CNN used for feature extraction
- Flattening layer to prepare input for MLFFNN
- MLFFNN based classifier
  - Number of hidden layers : 2
  - Hidden layer activation function : ReLU
  - Number of nodes in output layer : 5
  - Normalization : Not used
  - Softmax applied on output layer

1.3 Training

To implement the following different pre-trained CNNs:

- VGGNet
- GoogleNet

The model has been trained in the respective manner:

- Mode : Mini-batch mode training
- AdaM optimizer
- Loss function : Categorical cross entropy loss
- Stopping criterion : Trained for a fixed number of epochs

1.4 Hyperparameters

Listed below are the various hyperparameters of the models. We have performed a grid search to compare model performance using a modest variety of values of these parameters. The results have been tabulated for each CNN.

- Number of nodes in the hidden layers
- Learning rate
- Batch Size

Number of nodes in the hidden layers : The number of nodes controls the network's architecture. Choosing the right number of nodes impacts how well the network learns and performs, requiring experimentation to find the optimal value. Given that the training dataset is quite small (2000 images), we should expect a small number of nodes per hidden layer to perform adequately. However, the number of features in the flattened output of the CNN is also something to keep in mind, we do not want to reduce this to too small a number.

Learning rate : Learning rate, like the number of nodes, is set before training. It controls the step size for weight adjustments, impacting how fast and well the model learns. We need to experiment to find the optimal rate that avoids getting stuck or jumping past the best solution.

Batch size : The Batch size refers to the number of samples of data present in each batch during mini batch mode training. Mini-batch training allows for more frequent updates of model parameters compared to batch training, leading to faster convergence, therefore resulting in quicker training times. Mini-batch training often results in better generalization performance compared to stochastic (pattern mode) gradient descent, as it updates the parameters based on the average error of the whole batch instead of a single sample.

Mini-batches therefore provide a compromise between the high variance of stochastic gradient descent and the slow convergence of batch gradient descent.

1.5 Comparison of Optimization Methods

The tabulated results for the explored hyperparameter space has been presented in a series of tables below:

Model	HL1	HL2	LR	BatchSize	TrainErr	TrainAcc	ValErr	ValAcc	TestErr	TestAcc
VGGNet	64	64	0.0005	64	0.001	1.000	0.089	0.974	0.072	**0.980**
VGGNet	128	128	0.0005	32	0.000	1.000	0.115	0.966	0.071	0.974
VGGNet	64	128	0.0005	32	0.001	1.000	0.118	0.962	0.122	0.974
VGGNet	64	64	0.0005	32	0.001	1.000	0.099	0.970	0.072	0.972
VGGNet	64	128	0.0010	32	0.000	1.000	0.105	0.976	0.083	0.972
VGGNet	128	64	0.0005	32	0.001	1.000	0.106	0.972	0.106	0.972
VGGNet	128	64	0.0010	64	0.000	1.000	0.096	0.974	0.089	0.970
VGGNet	128	128	0.0005	64	0.000	1.000	0.101	0.970	0.107	0.970
VGGNet	128	128	0.0010	64	0.000	1.000	0.119	0.964	0.112	0.970
VGGNet	128	128	0.0010	32	0.000	1.000	0.152	0.952	0.127	0.970
VGGNet	64	64	0.0010	32	0.000	1.000	0.131	0.966	0.119	0.968
VGGNet	128	64	0.0010	32	0.000	1.000	0.147	0.958	0.103	0.966
VGGNet	128	64	0.0005	64	0.000	1.000	0.108	0.964	0.105	0.966
VGGNet	64	128	0.0005	64	0.000	1.000	0.123	0.966	0.096	0.964
VGGNet	64	128	0.0010	64	0.000	1.000	0.184	0.966	0.113	0.964
VGGNet	64	64	0.0010	64	0.000	1.000	0.113	0.962	0.119	0.964
GoogleNet	128	128	0.0005	64	0.044	0.989	0.090	0.964	0.086	**0.974**
GoogleNet	128	64	0.0010	32	0.032	0.992	0.066	0.972	0.088	0.974
GoogleNet	128	64	0.0005	64	0.050	0.989	0.079	0.972	0.093	0.974
GoogleNet	128	64	0.0010	64	0.037	0.988	0.069	0.976	0.087	0.972
GoogleNet	64	64	0.0010	64	0.032	0.993	0.055	0.980	0.089	0.972
GoogleNet	64	64	0.0005	32	0.069	0.989	0.088	0.978	0.111	0.972
GoogleNet	64	128	0.0005	64	0.062	0.984	0.089	0.972	0.098	0.970
GoogleNet	64	64	0.0005	64	0.071	0.985	0.098	0.964	0.108	0.970
GoogleNet	128	64	0.0005	32	0.051	0.988	0.077	0.970	0.096	0.968
GoogleNet	64	64	0.0010	32	0.036	0.992	0.060	0.984	0.087	0.966
GoogleNet	64	128	0.0005	32	0.051	0.990	0.072	0.982	0.101	0.966
GoogleNet	64	128	0.0010	64	0.058	0.979	0.107	0.960	0.118	0.964
GoogleNet	128	128	0.0005	32	0.047	0.989	0.082	0.978	0.104	0.958
GoogleNet	64	128	0.0010	32	0.039	0.990	0.061	0.980	0.103	0.956
GoogleNet	128	128	0.0010	64	0.046	0.985	0.085	0.974	0.130	0.954
GoogleNet	128	128	0.0010	32	0.044	0.987	0.084	0.970	0.123	0.952

1.6 VGGNet

VGGNet, or Visual Geometry Group Network, is a type of Convolutional Neural Network (CNN) architecture designed for image recognition tasks.

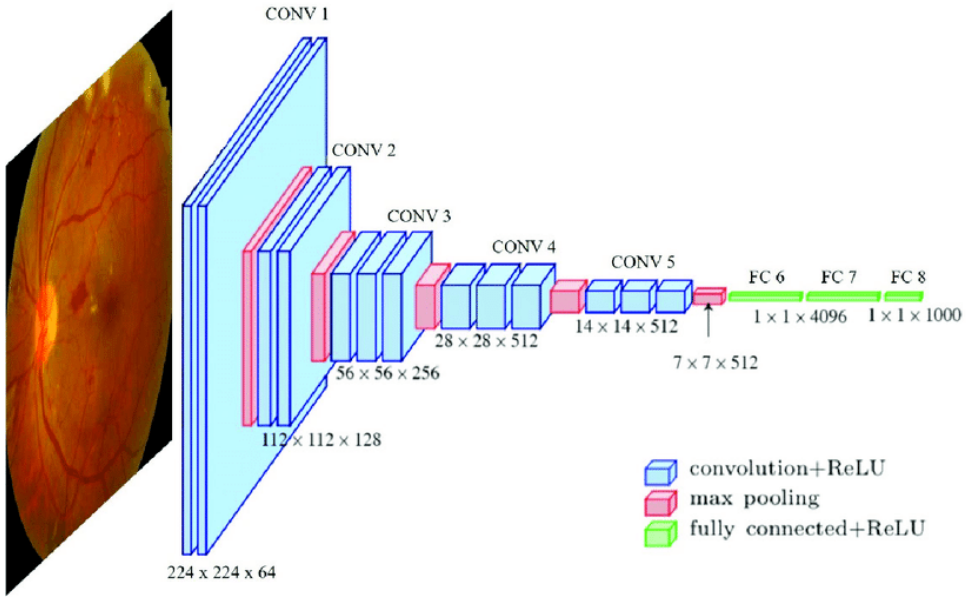
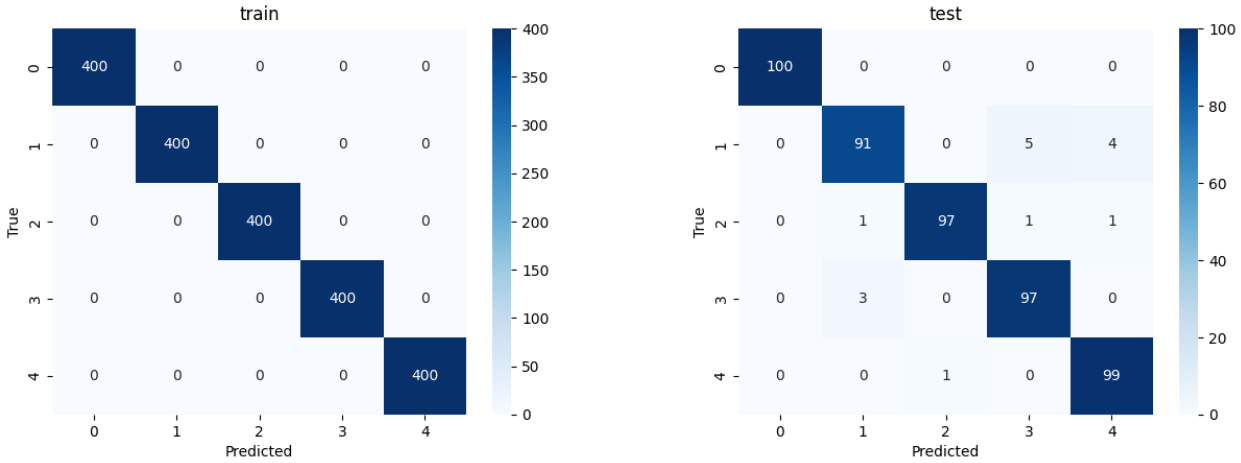


Figure 1.1: Architecture of VGGNet

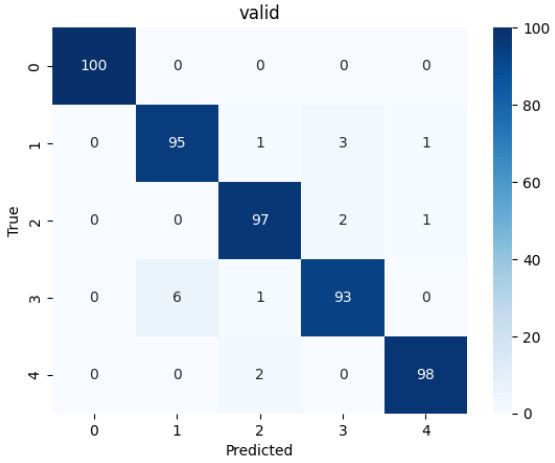
Key Points:

- 3x3 Filters: Small filters throughout the network.
- Depth: We used VGG16, which has 16 layers.
- Consistency: Each convolutional layer is followed by a ReLU activation.
- Pooling: Periodic max-pooling layers reduce spatial dimensions.
- Fully Connected: Ends with several fully connected layers



(a) Train

(b) Test



(c) Validation

Figure 1.2: Confusion Matrices for CNN using VGGNet for feature extraction

1.7 GoogleNet

GoogleNet, also known as Inception v1, is a Convolutional Neural Network (CNN) architecture developed by Google. It introduces a novel approach called the Inception module, which allows for more efficient and deeper networks.

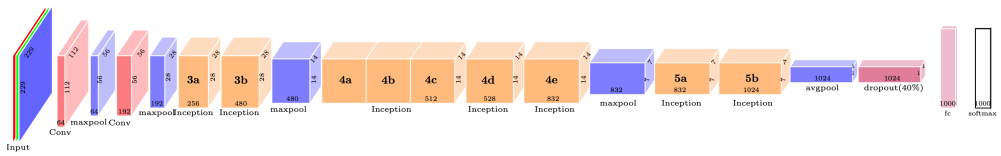


Figure 1.3: Architecture of GoogleNet

Inception Module

- 1x1 Convolution: Reduces dimensionality and computational cost.
- 3x3 Convolution: Captures medium-sized features.
- 5x5 Convolution: Captures larger features.

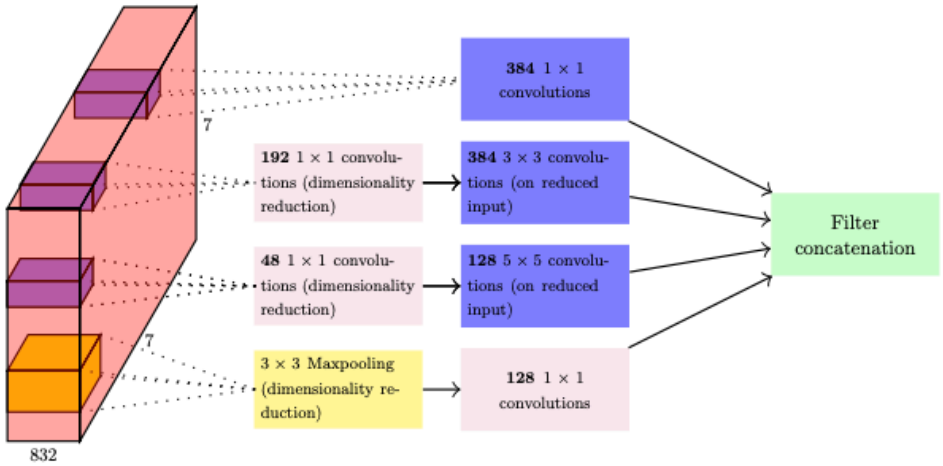
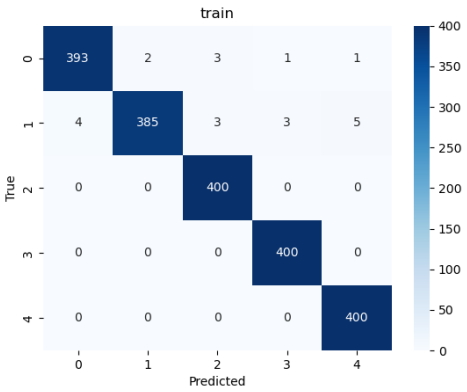


Figure 1.4: Architecture of Inception Module

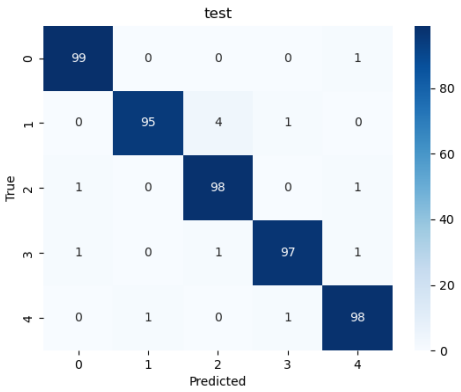
- Max-Pooling: Captures spatial hierarchies and reduces dimensions.
- The outputs of these layers are concatenated to form the output of the Inception module.

Key Points

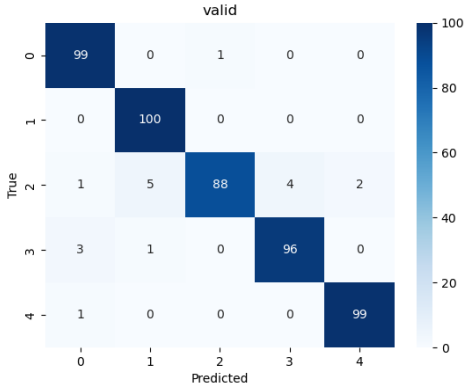
- Parameter Efficiency: The use of 1x1 convolutions and reduced filter sizes helps keep the number of parameters manageable.
- Modular Design: Inception modules can be stacked to create deep networks efficiently.



(a) Train



(b) Test



(c) Validation

Figure 1.5: Confusion Matrices for CNN using GoogleNet for feature extraction

1.8 Inferences

From the results of the grid search, inferences can be drawn regarding the performances of each pre-trained CNN model by analysing the validation error, test error, etc.

- Both architectures achieve very high accuracy in image classification, regardless of variation in hyperparameter values. This is due to the fact that they are pretrained on a vast number of images, so they have high generalization ability.
- Due to its design, using VGGNet is more computationally expensive. It has more parameters and thus takes longer during both training and inference phase.
- GoogleNet's innovative design allowed it to achieve high performance with fewer parameters and computational resources.

1.9 Conclusion

From our findings, we can conclude the following about different optimization methods:

- In general, both VGGNet and GoogleNet are excellent options for transfer learning.
- Due to the advanced architecture of GoogleNet, it can be used in a wider variety of applications where efficiency is critical.

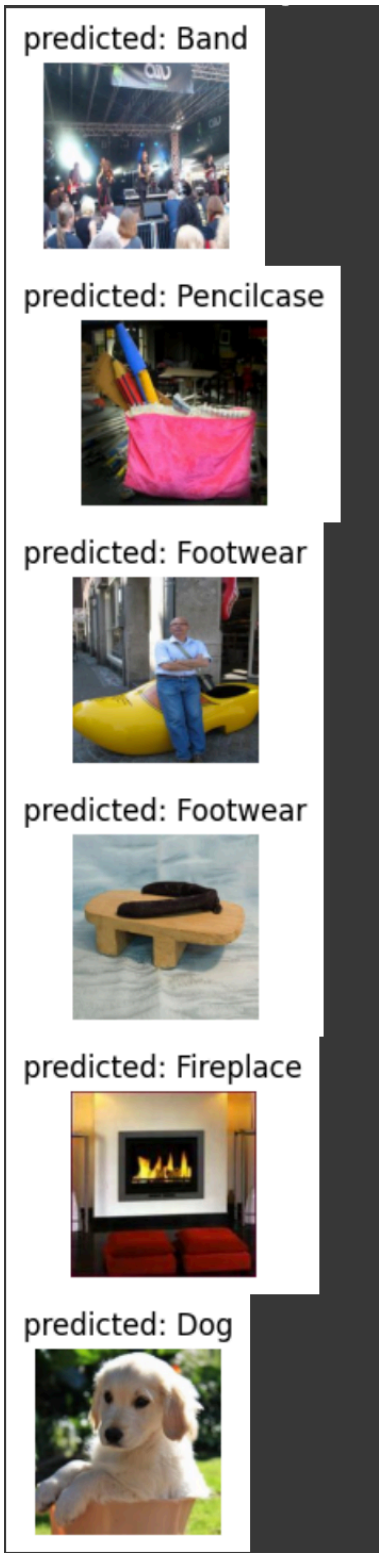


Figure 1.6: Some predictions from the models

2 Task 2

Image classification using a CNN with CL1, PL1, CL2 and PL2 as the layers. Use kernels of size 3x3, stride of 1 in the convolutional layers. Use the mean pooling with a kernel size of 2x2 and stride of 2 in the pooling layers. Use 4 feature maps in CL1. The number of feature maps in CL2 is a hyperparameter.

2.1 Data

A brief description of the provided datasets has been given below.

2.1.1 Training Data

Similar to task 1, the training data has 2000 images corresponding to 5 classes.

Since there are a total of 5 classes present, the output layer of our neural network should have 5 nodes. There is a uniform distribution of class labels over the 2000 data points, i.e. each class has 400 data points. Therefore, we have a balanced dataset which is a good thing since our model gets equal to exposure to all the 5 classes while training.

While training we have taken care that each batch has a uniform distribution of classes.

2.1.2 Validation Data

The validation data has 500 images. The label distribution is uniform, i.e. each class has 100 datapoints.

2.1.3 Test Data

The test data has 500 images. The label distribution is uniform, i.e. each class has 100 datapoints.

2.2 Models

We have built a model following the stated specifications

Model architecture

- Convolutional layer 1
  - Kernel size : 3 x 3
  - Stride : 1
  - Padding : 1
  - Feature maps : 4
- Average Pooling layer 1
- Convolutional layer 2
  - Kernel size : 3 x 3
  - Stride : 1

  - Padding : 1
  - Feature maps : hyperparameter
- Average Pooling layer 2
- Linear output layer (with Softmax activation)

2.3 Training

The model has been trained as specified below

- Mode : Mini batch training
- Optimizer : Adam (Adaptive Moments)
- Loss function : Cross entropy loss
- Stopping criterion : Fixed number of epochs

2.4 Hyperparameters

Listed below are the various hyperparameters of the models. We have demonstrated results and shown comparisons between the two MLFFNNs for variations in each of these:

- Batch size
- Number of feature maps in the second convolution layer
- Learning rate

A brief explanation of the various hyperparameters and their expected effects on the model has been provided below:

Batch size : The Batch size refers to the number of samples of data present in each batch during mini batch mode training. Mini-batch training allows for more frequent updates of model parameters compared to batch training, leading to faster convergence, therefore resulting in quicker training times. Mini-batch training often results in better generalization performance compared to stochastic (pattern mode) gradient descent, as it updates the parameters based on the average error of the whole batch instead of a single sample.

Mini-batches therefore provide a compromise between the high variance of stochastic gradient descent and the slow convergence of batch gradient descent.

Number of feature maps in the second convolution layer : This impacts its representation power, computational requirements, risk of overfitting or underfitting, model size, and overall performance. Finding the right balance is crucial and usually requires experimentation and tuning based on the specific characteristics of the problem and the dataset at hand.

Learning rate : The learning rate is a critical hyperparameter in training neural networks, as it determines the size of the update step taken during gradient descent. The choice of learning rate can significantly impact the training process and the performance of the neural network.

With a very low learning rate, the optimization algorithm progresses slowly, requiring many iterations to converge to a minimum. This can result in longer training times.
A very low learning rate may also cause the process to get stuck in local minima or saddle points. The algorithm may struggle to escape these regions of the parameter space, leading to sub-optimal solutions.

If the learning rate is too high, the gradient descent algorithm may overshoot the minimum of the loss function. This can lead to oscillations or instability in the training process.

Keeping the above points in mind, we have chosen appropriate learning rates for the considered models.

2.5 Tabulated Results

Tabulated results for multiple model configurations have been presented below:

Note

- CL2: Refers to the number of feature maps in the second convolutional layer

Batch Size	CL2	Learning Rate	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy
20	4	0.0001	1.1527	57.45	1.3655	44.0
20	4	0.0005	1.0882	58.20	1.2904	44.8
20	4	0.0010	1.0580	63.00	1.2411	48.6
20	8	0.0001	1.1141	58.50	1.3466	45.2
20	8	0.0005	1.0922	58.90	1.2512	49.8
20	8	0.0010	1.0565	61.65	1.2474	49.2
20	16	0.0001	1.0788	55.90	1.3041	48.0
20	16	0.0005	1.0961	60.00	1.1474	54.2
20	16	0.0010	1.0078	**68.75**	1.1386	**55.0**
50	4	0.0001	1.1406	56.25	1.3488	43.2
50	4	0.0005	1.1009	56.80	1.2809	45.0
50	4	0.0010	1.0534	58.50	1.3108	43.8
50	8	0.0001	1.1624	57.60	1.4158	43.6
50	8	0.0005	1.1088	57.15	1.3211	45.6
50	8	0.0010	1.1319	58.70	1.3441	47.4
50	16	0.0001	1.1169	57.45	1.2485	49.8
50	16	0.0005	1.0650	59.70	1.1591	52.2
50	16	0.0010	1.0769	62.80	1.1401	54.2
100	4	0.0001	1.1552	55.70	1.4765	45.4
100	4	0.0005	1.1451	57.80	1.3994	44.6
100	4	0.0010	1.1531	58.80	1.3907	44.0
100	8	0.0001	1.1336	57.00	1.4263	44.8
100	8	0.0005	1.1479	59.70	1.4121	43.8
100	8	0.0010	1.0797	58.15	1.3746	43.6
100	16	0.0001	1.1391	57.80	1.3572	43.0
100	16	0.0005	1.1558	59.10	1.8217	44.6
100	16	0.0010	1.1971	57.05	1.3187	46.4

2.6 Plots and Results for Optimal Model Configurations

We now provide the plots for average training error vs epoch as well as the confusion matrices for the best model obtained among the explored configurations.

We have considered highest validation accuracy as the criteria for choosing the best model.

After extensively exploring various configurations and different sets of values of hyperparameters, we have found that **the models with the best validation accuracy have the highest number of feature maps in the second convolutional layer**.

The hyperparameter configurations and performance metrics for the best 3 models as per highest validation accuracy criteria have been tabulated below:

Batch Size	CL2	Learning Rate	Train Loss	Train Accuracy	Val Loss	Val Accuracy
20	16	0.0010	1.0078	**68.75**	1.1386	**55.0**
20	16	0.0005	1.0961	60.00	1.1474	54.2
50	16	0.0010	1.0769	62.80	1.1401	54.2

Table 2.1: Metrics for best 3 overall models, they all have 16 feature maps in the second CL

It is reasonable to hypothesize that the representation power of the model increases with a higher number of feature maps. We will try and draw inferences from our grid search to support this hypothesis.

The confusion matrices as well as the error vs epoch plots for the best 3 models along have been provided on the following pages:

Batch Size	CL2	Learning Rate
20	16	0.0010

Table 2.2: The graphs below have been plotted for the above model configuration:

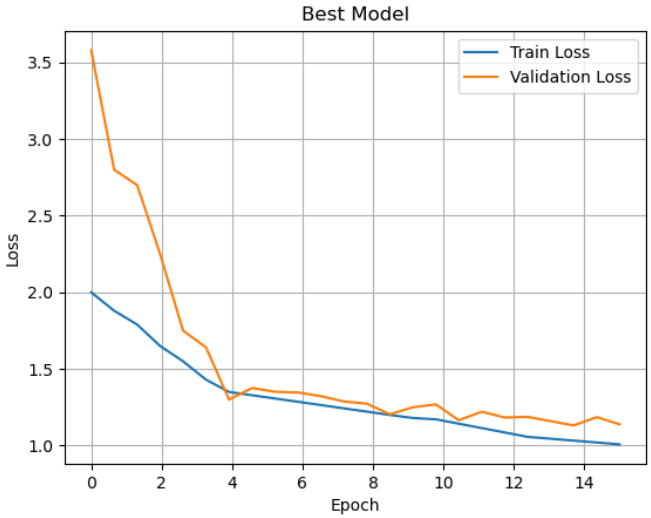
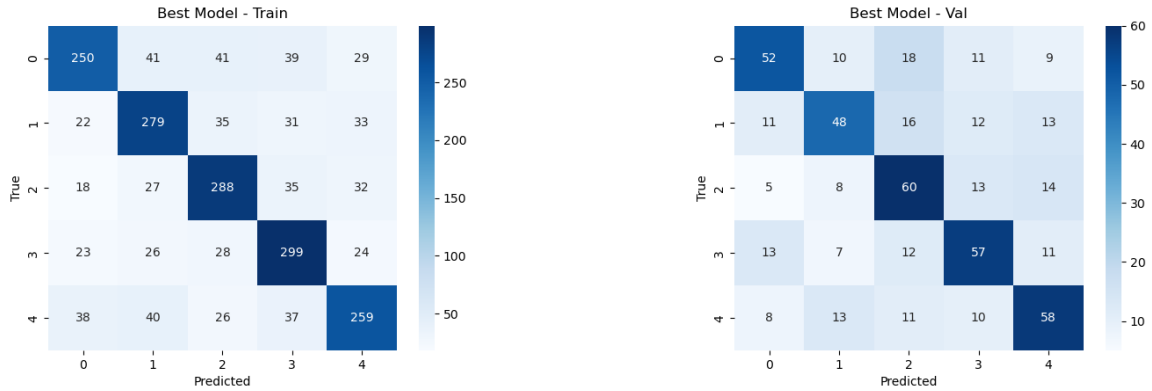


Figure 2.1: Average train and validation error plotted as a function of epochs



(a) Train

(b) Validation

Figure 2.2: Confusion Matrices

Batch Size	CL2	Learning Rate
20	16	0.0005

Table 2.3: The graphs below have been plotted for the above model configuration:

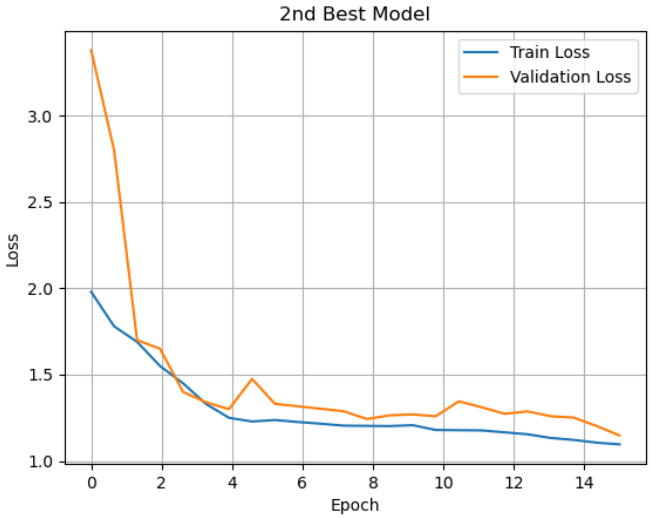
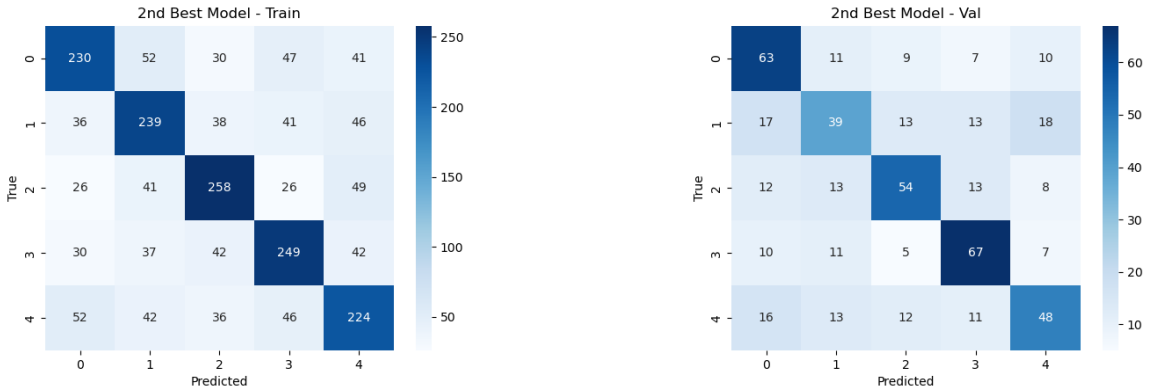


Figure 2.3: Average train and validation error plotted as a function of epochs



(a) Train

(b) Validation

Figure 2.4: Confusion Matrices

Batch Size	CL2	Learning Rate
50	16	0.001

Table 2.4: The graphs below have been plotted for the above model configuration:

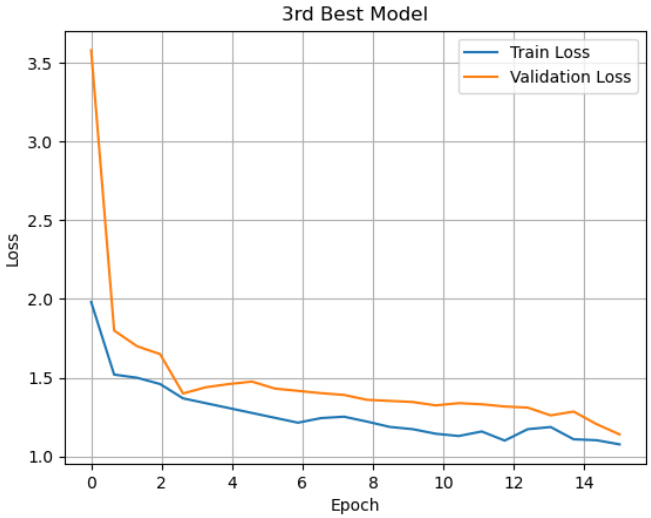
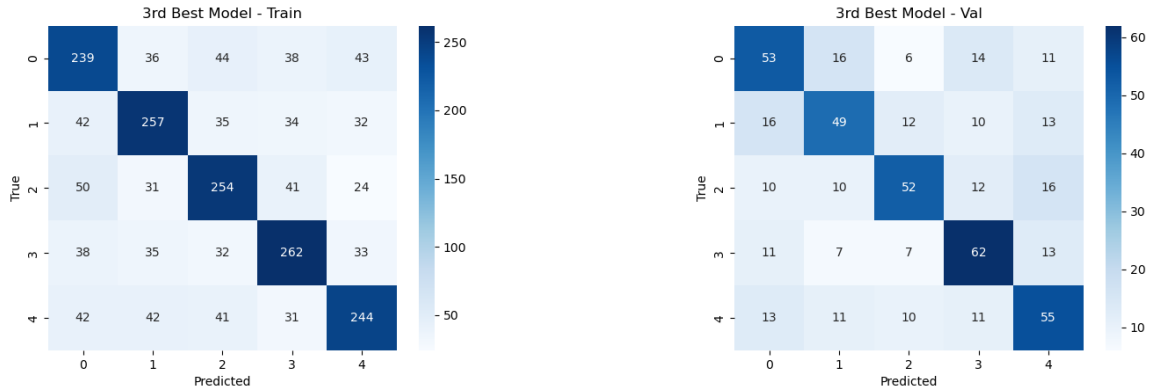


Figure 2.5: Average train and validation error plotted as a function of epochs



(a) Train

(b) Validation

Figure 2.6: Confusion Matrices

2.7 Inferences

Below, we observe the grid search data and draw inferences regarding the performance of models by analysing dependence of the validation accuracy and error on the number of feature maps, learning rate, and batch size.

2.7.1 Dependence on Batch Size

Batch Size	Validation Accuracy	Validation Error
20	48.75555556	1.261666667
50	47.2	1.282944444
100	43.13333333	1.441911111

Table 2.5: Average Validation loss and accuracy over model space for various batch sizes

From the table above, we can infer that a small batch size improves the convergence performance of the CNN.

2.7.2 Dependence on Learning Rate

Learning Rate	Validation Accuracy	Validation Error
0.0001	44.55555556	1.365477778
0.0005	46.84444444	1.345111111
0.001	47.68888889	1.275933333

Table 2.6: Average Validation loss and accuracy over model space for various learning rates

As can be seen above, a moderately high learning rate seems to improve the training performance of the model.

2.7.3 Dependence on Number of Feature Maps in Second Layer

Number of Feature Maps in CL2	Validation Accuracy	Validation Error
4	44.15555556	1.3449
8	45.88888889	1.3488
16	49.04444444	1.292822222

Table 2.7: Average Validation loss and accuracy over model space by varying number of feature maps in CL2

We observe that increasing the number of feature maps reduces the error and increases the accuracy of our model.

2.8 Conclusion

Given the above extensive analysis, we can conclude that using the following:

- Small batch size improves the training process when training for a fixed number of epochs because each mini-batch is smaller leading to more updates within a single epoch.
- A higher learning rate is better since it helps the model converge to a lower loss and better accuracy quicker.
- Increasing the number of parameters by using more feature maps in the second convolutional layer helps the model generalize better.

3 Image Captioning

3.1 Introduction

The process of creating a textual explanation for a set of images is known as image captioning. The captions are generated using both Computer vision and NLP. Because it converts images from a sequence of pixels to a series of words, image captioning can be viewed of as an end-to-end Sequence to Sequence challenge. Both the language or remarks as well as the images must be processed for this purpose. We use RNNs and LSTMs for the Language component and CNNs for the Image part.

3.2 Encoder

The encoder takes the input image and generates a fixed dimensional vector representing the image features. Various models can be employed to achieve this like custom CNNs or pre-trained CNNs(like VGG or ResNet). We can also incorporate architectures like NetVLAD in this process.

3.3 Decoder

The decoder takes the feature vector produced by the encoder as its initial hidden state and generates the output sequence, i.e. the caption describing the input image. It does so one time step at a time. At each time step the inputs to the decoder are the hidden state and the output word generated by the previous time step. For the very first time step, we give "SOS", i.e. Start of Sequence token to initiate the generation process.

3.4 Theory

In this subsection, we cover some foundational theory related to the architectures and methods used by us to complete our task.

3.4.1 NetVLAD

NetVLAD is a method designed for visual localization and mapping tasks in computer vision. It was proposed in the paper "NetVLAD: CNN architecture for weakly supervised place recognition".

At the heart of NetVLAD is the VLAD (Vector of Locally Aggregated Descriptors) encoding technique, which aggregates local image descriptors into a fixed-length global representation. Unlike traditional VLAD, which relies on handcrafted descriptors, NetVLAD employs a convolutional neural network (CNN) to learn discriminative features from image data. This allows NetVLAD to capture complex spatial relationships and semantic information present in images, leading to improved accuracy and robustness.

NetVLAD is trained along with the whole architecture in an end to end manner.

The mathematics for NetVLAD has been given below:

NetVLAD Formulation:

Given a set of M local image descriptors $\{\bar{x_1}, \bar{x_2}, \ldots, \bar{x_M}\}$ extracted from an input image, NetVLAD computes a fixed-length global descriptor $\bar{z}$ as follows:

$$\bar{z_k} = \sum_{i=1}^{M} a_{ki}(\bar{x_i} - \bar{\mu_k}) \tag{3.1}$$

$$\bar{z} = \begin{bmatrix} \bar{z_1} \\ \bar{z_2} \\ . \\ . \\ \bar{z_K} \end{bmatrix}$$

$$a_{ki} = \frac{e^{-\beta\|\bar{x_i} - \bar{\mu_i}\|^2}}{\sum_{j=1}^{K} e^{-\beta\|\bar{x_i} - \bar{\mu_j}\|^2}} \tag{3.2}$$

$$a_{ki} = \frac{e^{\bar{w_k}^t \bar{x_i} + b_k}}{\sum_{j=1}^{K} e^{\bar{w_j}^t \bar{x_i} + b_j}} \tag{3.3}$$

$$\bar{w_k} = 2\beta\bar{\mu_k} \tag{3.4}$$

$$b_k = -\beta\|\bar{\mu_k}\|^2 \tag{3.5}$$

where:

- $\bar{x_i}$ is the i-th local descriptor,
- M is the number of local descriptors,
- a_{ki} is the assignment weight for the i-th descriptor,
- $\bar{\mu_k}$ is the kth cluster centroid.
- K is the number of clusters

3.4.2 GloVe

GloVe(Global Vectors for Word Representation) is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

3.4.3 Recurrent Neural Network

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data. These deep learning architectures are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such as Siri, voice search, and Google Translate. Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output.

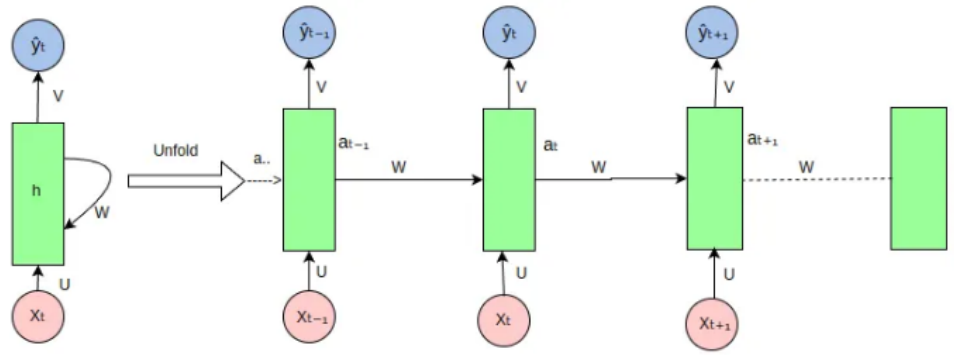


Figure 3.1: Architecture of an unfolded RNN

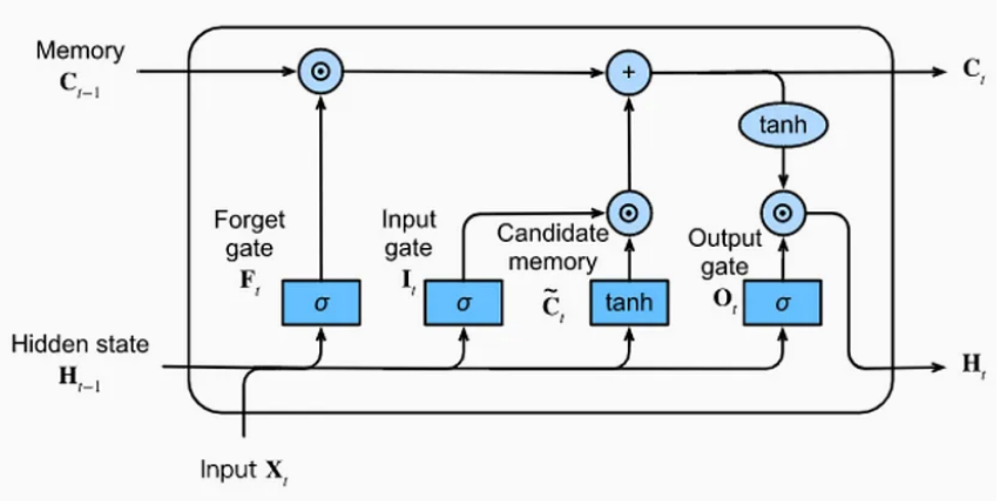


Figure 3.2: Architecture of an LSTM Unit

3.4.4 LSTM : Long Short Term Memory

Long Short-Term Memory Networks is a deep learning, sequential neural network that allows information to persist. It is a special type of Recurrent Neural Network which is capable of handling the vanishing gradient problem faced by RNN.

4 Task 3 and 4

Tasks 3 and 4 are based on developing a model for Image Captioning. In task 3 we use an RNN based decoder whereas in Task 4 we use an LSTM based decoder. Due to the similarities in the two tasks and for ease of comparison, we provide a combined report for the two tasks in this section.

4.1 Data

A brief description of the provided datasets has been given below.

The whole dataset contains a total of 8102 images, each of which has 5 appropriate captions. Each team was assigned 4000 images out of this. We have split this data into the training, validation and testing datasets. While splitting, care has been taken that images in the training data are not present in the validation or testing datasets to avoid leakage of data.

4.1.1 Training Data

The Training dataset has 3400 images, and therefore a total of 17000 image and caption pairs.

4.1.2 Validation Data

The Training dataset has 500 images, and therefore a total of 2500 image and caption pairs.

4.1.3 Test Data

The Training dataset has 100 images, and therefore a total of 500 image and caption pairs.

The final details of the three datasets are as follows:

Dataset	Datapoints (Image-Caption pairs)
Training	17000
Validation	2500
Test	500

4.2 Data Preprocessing

The following steps of pre-processing were carried out on the data before building the model:

- Extra spaces were trimmed from the captions.
- Characters like 'a' were removed from the captions and the vocabulary since they don't contribute to the caption meaning.
- English sentences were converted to lowercase to avoid case sensitivity in embeddings.
- Start of Sequence and End of Sequence tokens were added to all the captions to indicate the start and end of each caption.
- Special tokens like "<UNK>" and "<PAD>" were introduced in the vocabulary. For each batch, all captions were padded such that the sequence length is the same for all captions within that batch.
- The frequency of unique words was tracked across all three datasets. Then, the vocabulary was determined by considering the unique words with a frequency of greater than equal to 5. This was done to ensure a more generalized training as many words in the caption data occurred only once. The words not satisfying this criteria were assigned the "<UNK>" token.

After completing the pre-processing steps, the final vocabulary size is 2088 (including the 4 special tokens).

4.3 Model

Our model consists of the following components:

- Pre-trained CNN

- NetVLAD
- Fully connected layer
- Decoder (RNN or LSTM based)

4.3.1 Pre-trained CNN

We opted to use a pre-trained CNN to obtain meaningful representation features of our images as opposed to using a simple CNN architecture which would provide relatively inferior results. We experimented with multiple possible options like ResNet, Inception module and VGG. We have used VGG in our final architecture since its output has a relatively lesser number of feature maps thus reducing dimensionality of the input to NetVLAD, helping reduce the number of parameters to be trained.

We have used only the features and average pooling layers of the VGG architecture and not the fully connected layers since we want the feature map representation of the images which has to be sent to NetVLAD.

Using this portion of VGG gives an output with the dimensions (batch size, 512, 7, 7) where 512 is the number of feature maps and 7 is the height and width of each feature map. This is now sent as input to NetVLAD.

4.3.2 NetVLAD

For NetVLAD, we have used a total of 32 clusters. Thus the dimensions of the output given by NetVLAD is (batch size, num_clusters x descriptor dimension) which is equivalent to (batch size, 32 x 512)

4.3.3 Fully connected Layer

We have used a fully connected layer to map the NetVLAD feature representation vector to a lower dimensional space of size 256. This is now fed to the decoder.

4.3.4 Decoder

RNN based Decoder

In the case of RNN, there is a single hidden state. This is initialized to be the generated feature representation vector.

LSTM based Decoder

In the case of LSTM, there is a hidden state as well as a cell state, we initialize both of them to be the generated feature representation vector.

4.4 Training

The model has been trained in the manner as specified below :

- Mode : Mini batch training (Batch size : 50)
- Optimizer : Adam (Adaptive Moments)
- Loss function : Cross entropy loss

- Stopping Criterion : Fixed number of epochs (20)
- Teacher forcing ratio has been employed

We apply cross entropy loss once predictions for all the time steps for a given batch of images have been made. There are a total of 340 update steps in each epoch since our training data has 17000 image, caption pairs and our batch size is 50.

The average training and validation loss vs epoch plot for the optimum RNN based and LSTM based model configurations have been shown below respectively:

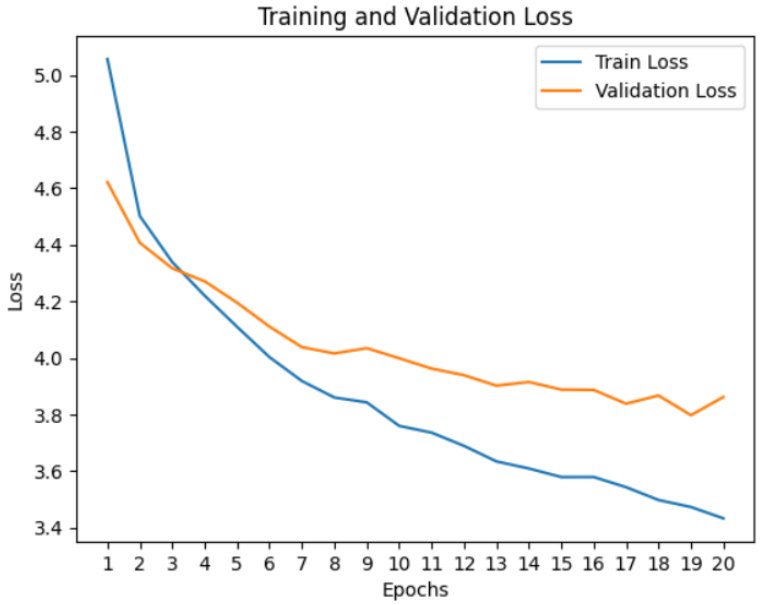


Figure 4.1: Train and Val loss vs Epoch for RNN

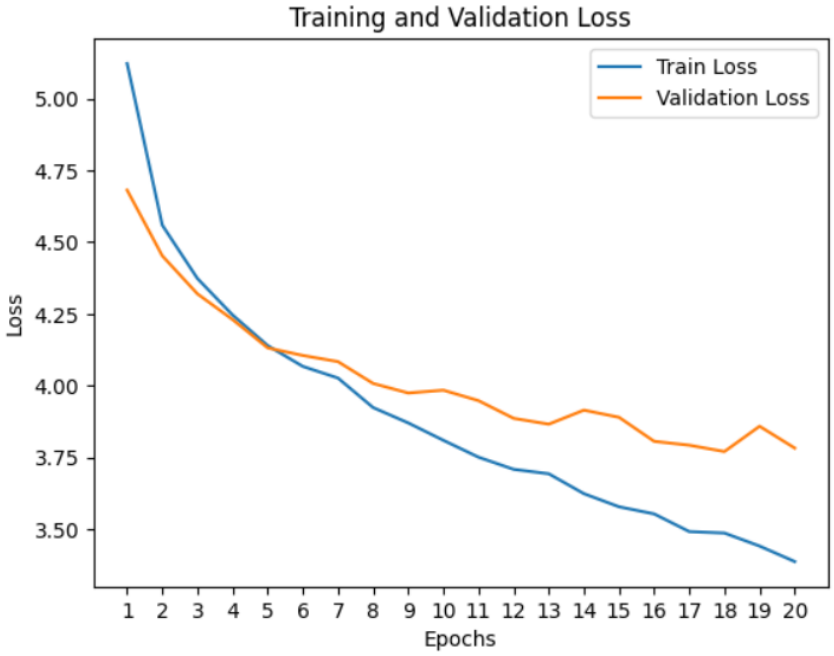


Figure 4.2: Train and Val loss vs Epoch for LSTM

As we can see, LSTM gives lower validation error than RNN(i.e. 3.78 as opposed to 3.86) for the same number of epochs as expected.

4.5 Hyperparameters

Listed below are the various hyperparameters of the model:

- Batch size
- Embedding dimension for learnable embedding layer
- Hidden size of RNNs/LSTMs
- Teacher forcing ratio
- Number of clusters in NetVLAD

A brief explanation of the various hyperparameters and their expected effects on the model has been provided below:

Batch size : Mini-batch training allows for more frequent updates of model parameters compared to batch training, leading to faster convergence, therefore resulting in quicker training times. Mini-batch training often results in better generalization performance compared to stochastic (pattern mode) gradient descent, as it updates the parameters based on the average error of the whole batch instead of a single sample.

Embedding dimension : A higher embedding dimension allows for learning finer differences between different words in the vocabulary, increasing the capability of the model to differntiate words.

Hidden size of LSTMs : Increasing the hidden size increases the capacity of the LSTM model to capture complex patterns in the data. A larger hidden size allows the network to learn more intricate relationships between input sequences and their corresponding outputs. A larger hidden size enables the network to remember information from earlier time steps for a more extended period, potentially improving its ability to capture long-term dependencies in the data.

Teacher forcing ratio : Teacher forcing helps the model learn faster on training data, but with very high teacher forcing the model may never learn to generate sequences on its own. Therefore, this parameter is decayed as training progresses and the capacity of the model to autoregressively generate sequences improves.

Number of Clusters for NetVLAD : The number of clusters, often denoted as K, represents the number of cluster centroids used to aggregate local descriptors into the global descriptor. With more clusters, the clustering process can represent finer distinctions between local features, making the global descriptor more sensitive to subtle variations in the image, but the computational expense during training also increases. If we reduce the number of clusters, important nuances in the image data might be lost, as fewer clusters can lead to over-generalization, where different local features are mapped to the same cluster, reducing the distinctiveness of the representation.

4.6 Experimental Studies

4.6.1 Choice of Embedding

We experimented using both GloVe embeddings as well as learning embeddings from scratch. The obtained results are tabulated below.

Embedding Type	No. of Epochs	Val Loss
Glove	20	3.86
Learning Embedding	20	3.94

Table 4.1: Table for RNN

Embedding Type	No. of Epochs	Val Loss
Glove	20	3.78
Learning Embedding	20	3.84

Table 4.2: Table for LSTM

4.6.2 Choice of CNN

CNN	Val Loss
VGG16	3.86
Simple CNN	4.01

Table 4.3: Loss after 20 epochs for RNN

CNN	Val Loss
VGG16	3.78
Simple CNN	3.98

Table 4.4: Loss after 20 epochs for LSTM

4.6.3 Varying hidden and embedding dimensions

RNN Hidden Size	Embedding Dimension	Val Loss
512	64	4.01
512	128	3.95
256	64	3.89
256	**128**	**3.86**

Table 4.5: For RNN

LSTM Hidden Size	Embedding Dimension	Val Loss
512	64	3.88
512	128	3.85
256	64	3.80
256	**128**	**3.78**

Table 4.6: for LSTM

4.7 Results for Optimal Model Configuration

4.7.1 Prediction samples

In this section we present some of the caption predictions made by the model on train and test images from the dataset.

Predictions for RNN based model

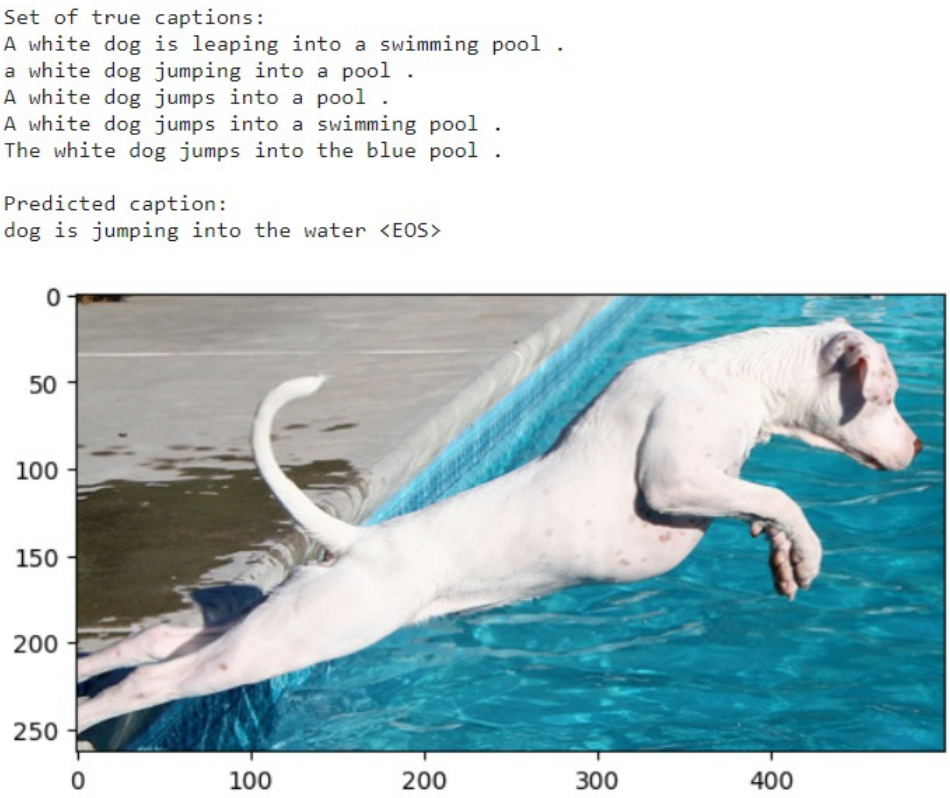


Figure 4.3: Training Image 1

```
Set of true captions:
A boy wearing black shorts is jumping into the ocean .
A child in swimming trunks jumping into the ocean .
A child jumps over ocean waves at a beach .
A child wearing blue and white shorts is jumping in the surf .
A little boy in black shorts is jumping in the water at the beach .

Predicted caption:
boy in blue shorts is on the beach <EOS>
```

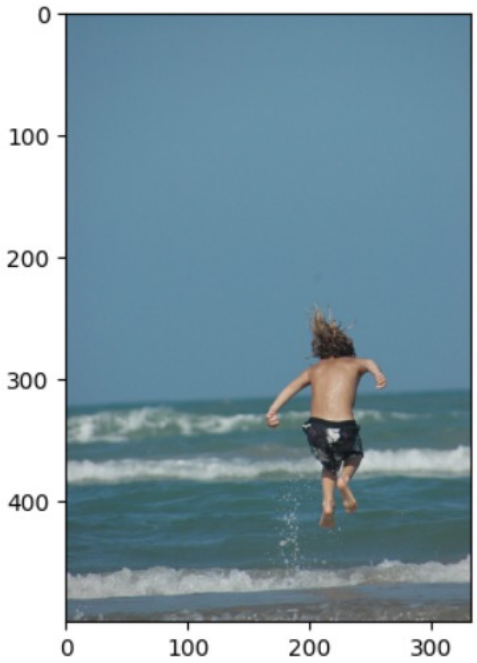


Figure 4.4: Training Image 2

```
Set of true captions:
A group of men play soccer on the field .
A man in a red uniform leaps in the air as one in a white uniform tries to kick a ball .
During a soccer game , one man is in the air while another has one foot down .
The soccer player in white is challenging the player in red for the ball in midair .
Two soccer teams are playing , one is in white the other in red .

Predicted caption:
football player in red and white uniform <EOS>
```

Figure 4.5: Training Image 3

```
Set of true captions:
A football player in a purple uniform holds the ball while a player in red tries to tackle him .
A football player in white is being tackled by a Oklahoma University player .
An American footballer in a red outfit is attacking the player in white who is running with the ball , whilst other players ar
e nearby .
A Sooners football player is tackling an opposing player .
The receiver is tackled after making the catch .

Predicted caption:
football player in red and white uniform <EOS>
```

Figure 4.6: Test Image 1

```
Set of true captions:
A skier doing a high jump from a snowy ramp near a mountain range .
A ski jumper enjoys winter fun .
A snowboarder flies off of a jump which is next to a big red atomic sign .
a snowboarder is up in the air having taken off from a big jump in a tournament .
A snowboarder jumps off a large ramp in the mountains .

Predicted caption:
person is skiing down snowy hill <EOS>
```

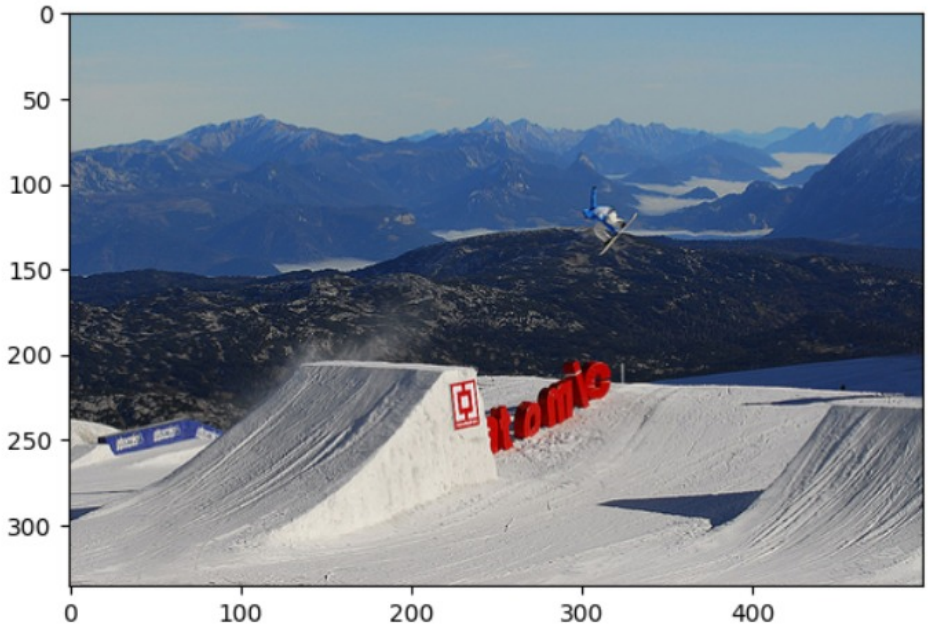


Figure 4.7: Test Image 2

```
Set of true captions:
A black dog runs through the snow .
A black poodle frolics in the snow .
A black poodle is running in the snow .
A black poodle plays in the snow .
A black standard poodle running through the snow in a fenced-in yard .

Predicted caption:
black dog is running through the snow <EOS>
```

Figure 4.8: Test Image 3

Predictions for LSTM based model

```
Set of true captions:
A guy in a red jacket is snowboarding in midair .
A man is snowboarding down a railing in the snow .
A man skiing down a hill and over a red obstacle .
A person snowboarding leaps over an obstacle to continue down the hill .
A snowboarder jumps over a ramp on a mountain course .

Predicted caption:
person in red jacket is snowboarding <EOS>
```

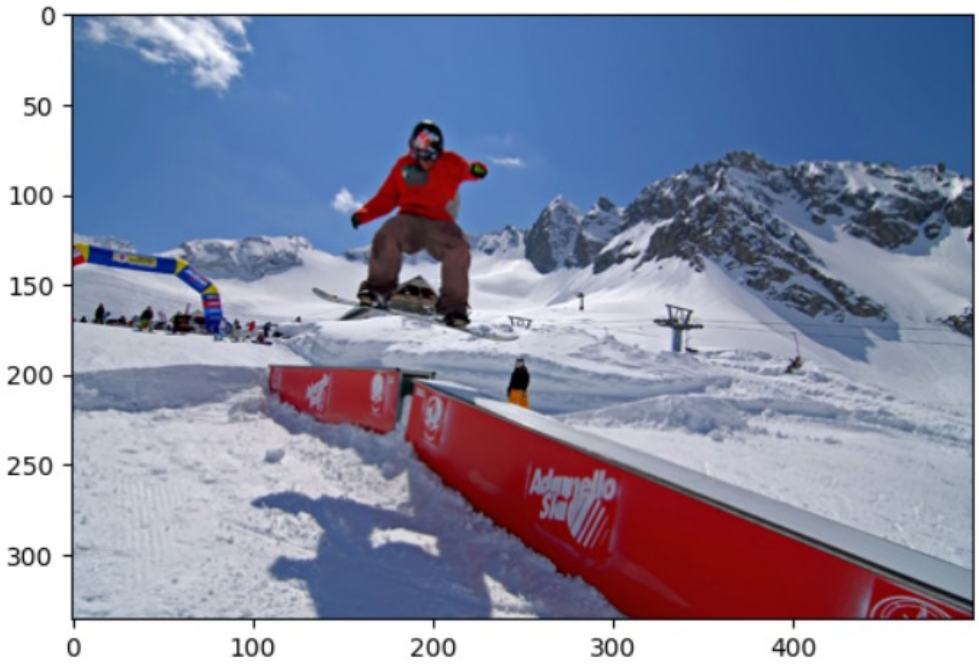


Figure 4.9: Train Image 1

```
Set of true captions:
A brown dog jumps over a barrier that is black and white .
A dog jumping high to clear the bars .
a dog jumps over the pole .
Orange dog jumps over striped posts on course .
The brown dog is performing a jump in an obstacle course .

Predicted caption:
dog jumps over hurdle <EOS>
```

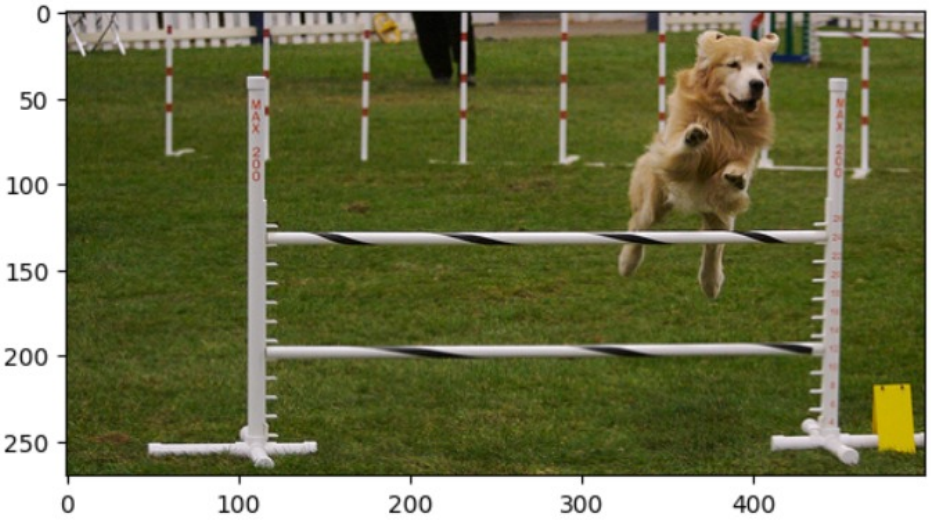


Figure 4.10: Train Image 2

```
Set of true captions:
A man in blue and black surfs on a huge white wave .
A man on a surfboard jumping off the top of a wave .
a surfer is doing an airborne stunt whilst riding waves .
Man boogie boarding on wave .
man surfing

Predicted caption:
man is surfing on wave <EOS>
```

Figure 4.11: Train Image 3

```
Set of true captions:
A child steps up on a playground toy as arms reach for her .
A girl climbs the stairway of a playground as arms reach for her .
An adult is encouraging a young girl to try the slide .
A small child is climbing up playground equipment while hands reach toward her from behind bars .
The girl is at the park playing on the jungle gym .

Predicted caption:
young girl in pink shirt is playing with her arms crossed <EOS>
```

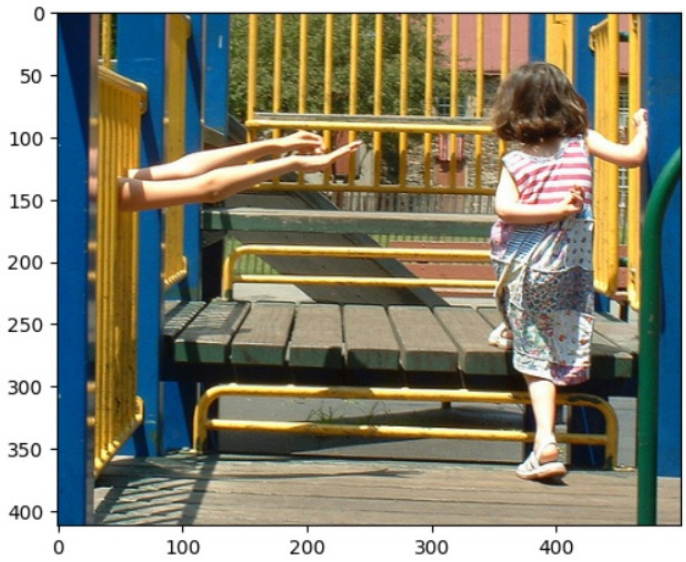


Figure 4.12: Test Image 1

```
Set of true captions:
A bicyclist becomes airborne among dirt hills at night .
A man jumps his dirt bike at night .
A motocross rider at night high in the air above the dirt .
Someone is performing an aerial skateboard trick at night .
The rider is jumping his BMX bike .

Predicted caption:
person on bike is riding dirt bike in the woods <EOS>
```

Figure 4.13: Test Image 2

```
Set of true captions:
A black dog is running through the shallow edge of a large body of water .
A black dog is running through the water .
A black dog jumps high out of the water .
a black and tan dobbermen running through the ocean .
The dog runs through the water .

Predicted caption:
black dog is running through the water <EOS>
```

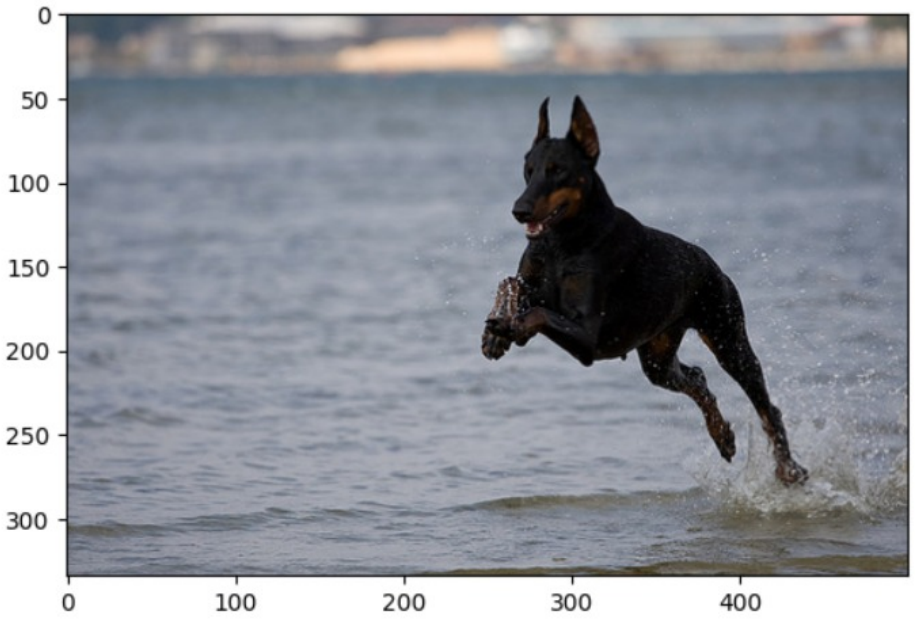


Figure 4.14: Test Image 3

4.7.2 BLEU Scores

The obtained bleu scores for the RNN based model are as follows :

Dataset	BLEU 1	BLEU 2	BLEU 3	BLEU 4
Training	0.6529	0.4679	0.3596	0.2954
Validation	0.5782	0.3919	0.2816	0.2184
Test	0.5600	0.3611	0.2604	0.2047

The obtained bleu scores for the LSTM based model are as follows :

Dataset	BLEU 1	BLEU 2	BLEU 3	BLEU 4
Training	0.7036	0.4966	0.3716	0.3021
Validation	0.6225	0.4147	0.2908	0.2234
Test	0.6042	0.4035	0.2871	0.2018

4.8 Inferences

Some inferences drawn from the experimental data are:

- LSTM gives better performance as compared to a simple RNN. This can be attributed to its higher memory retention over long sequences.
- A higher dimension of embedding helps a model of given configuration learn the sequence prediction task better. This is because each word can be represented more precisely due to the higher number of dimensions.
- Using a pre-trained CNN helps us produce better captions as opposed to using a simpler custom CNN architecture which has to be trained from scratch. This is expected since deep pre-trained architectures like VGG are able to better capture the complexities in diverse images and hence produce better representations.
- Using GloVe representations for generating embeddings results in quicker convergence as opposed to learning embeddings from scratch. This is expected since by using GloVe, we are providing the model with more meaningful embeddings right from the start.
- Although we are getting satisfactory results, we can further improve performance by using Attention based architectures which helps the model understand context and generate more meaningful captions.
- The model is able to learn the words that make up the sentence but not necessarily get their order correct in the prediction. This can be inferred from the significantly higher 1 gram bleu score as compared to the 2 gram, 3 gram and 4 gram bleu scores.

4.9 Conclusion

We obtain decent performance using both RNN and LSTM based models for the Image Captioning Task, although LSTM gives relatively better performance.

5 Machine Translation

5.1 Introduction

Machine translation is the task of translating text from one language to another. In recent years, neural machine translation models, particularly those based on encoder-decoder architectures with Long Short-Term Memory (LSTM) networks, have shown remarkable performance in this task.

5.2 Encoder

The encoder takes the input sequence in the source language and processes it into a fixed-dimensional representation called the context vector. Each word in the input sequence is embedded into a high-dimensional vector representation and fed into the LSTM layer. The final hidden state of the LSTM serves as the context vector, which encodes the entire input sequence.

5.3 Decoder

The decoder takes the context vector produced by the encoder and generates the output sequence in the target language. At each time step, the decoder LSTM receives the context vector along with the previously generated words (or tokens) in the target sequence. It predicts the next word in the target sequence based on this information.

5.4 Training

The encoder-decoder LSTM model is trained using a parallel corpus consisting of source-target language pairs. The model is trained to minimize the cross-entropy loss between the predicted target sequence and the ground truth target sequence. During training, the parameters of the encoder and decoder LSTMs are updated using backpropagation through time (BPTT).

5.5 Inference

During inference, the encoder-decoder LSTM model is used to translate new input sequences. The input sequence is first encoded into a context vector using the encoder LSTM. Then, the decoder LSTM generates the output sequence word by word based on the context vector and previously generated words.

5.6 Teacher Forcing

Teacher forcing is a training technique used in sequence prediction tasks, such as sequence-to-sequence models like encoder-decoder LSTMs. In teacher forcing, during training, the model is fed with the ground truth target sequence at each time step, rather than its own previously generated output. This approach helps stabilize training by providing the model with correct information during the learning process. However, during inference, when the model generates predictions in a real-world scenario, it relies solely on its own previously generated output, which may lead to error accumulation and performance degradation.

6 Task 5

Machine translation with encoder and decoder, each built using a single hidden layer LSTM network.

6.1 Data

A brief description of the provided datasets has been given below.

6.1.1 Training Data

The training data has 70,000 pairs of English and Hindi sentences.

6.1.2 Validation Data

The training data has 20,000 pairs of English and Hindi sentences.

6.1.3 Test Data

The training data has 10,000 pairs of English and Hindi sentences.

6.2 Data Pre-Processing

The following steps of pre-processing were carried out on the data before building the model:

- Extra spaces, stray quotes and digits were removed from both languages' datapoints.
- English sentences were converted to lowercase to avoid case sensitivity in embeddings.
- Start and end tokens were added to Hindi sentences in all datasets.
- The sentence length was clipped to 20 and pairs with longer sentences in either language removed from all the datasets. This was done in order to speed up training on the limited gpu resources available.
- The frequency of unique words in each language was tracked separately across all three datasets. Then, the vocabulary was determined by considering the unique words with a frequency of greater than equal to 5. This was done to ensure a more generalized training as many words in the Hindi data occurred only once. Post this, the sentences with words that did not make it to the vocabulary were removed from all datasets.

After the above pre-processing steps, the details of the datasets are as follows:

Dataset	Datapoints (Pairs of English - Hindi sentences)
Training	15926
Validation	4625
Test	2293

Further, the vocabulary of the leftover Hindi sentences is 7909 distinct words, including the start ('START_') , end ('_END') and padding ('0') tokens.

6.3 English Sentence Embedding

The embedding process of English sentences in all three datasets is described below:

- Glove embedder was used to embed English sentences. The chosen model of glove uses 50 dimensional embeddings for each word.
- Random embeddings were used for any words not in the glove embedder's lookup table.
- Sentences were padded to max length of 20 with the padding token '0' assumed to have a 50 dimensional zero vector as embedding.

6.4 Hindi Sentence Embedding

The embedding process of Hindi sentences in all three datasets is described below:

- The embeddings for the Hindi language were learnt as a learnable 128 dimension embedding layer in the model.
- This embedding learns a glove like lookup table of embeddings for all 7909 words in the vocabulary.

6.5 Model

The model specification is as follows:

- **Embedding Layer** - 128 dimensional
- **Encoder LSTM** - input size = 50 (English embedding dimension), hidden size = 256
- **Decoder LSTM** - input size = 128 (Hindi embedding dimension from embedding layer), hidden size = 256
- **FC Layer** - input dimension = 256, number of nodes = 7909 (vocabulary size)
- Softmax applied on FC layer

6.6 Training

The model has been trained in the manner specified below :

- Mode : Mini batch training
- Optimizer : Adam (Adaptive Moments)
- Loss function : Cross entropy loss
- Stopping criterion : Maximum number of episodes (20 epochs)
- Teacher forcing decayed over time

6.7 Hyperparameters

Listed below are the various hyperparameters of the model:

- Batch size
- Embedding dimension for learnable embedding layer
- Hidden size of LSTMs
- Teacher forcing ratio (start value and decay)

A brief explanation of the various hyperparameters and their expected effects on the model has been provided below:

Batch size : Mini-batch training allows for more frequent updates of model parameters compared to batch training, leading to faster convergence, therefore resulting in quicker training times. Mini-batch training often results in better generalization performance compared to stochastic (pattern mode) gradient descent, as it updates the parameters based on the average error of the whole batch instead of a single sample.

Embedding dimension : A higher embedding dimension allows for learning finer differences between different words in the vocabulary, increasing the capability of the model to differntiate words.

Hidden size of LSTMs : Increasing the hidden size increases the capacity of the LSTM model to capture complex patterns in the data. A larger hidden size allows the network to learn more intricate relationships between input sequences and their corresponding outputs. A larger hidden size enables the network to remember information from earlier time steps for a more extended period, potentially improving its ability to capture long-term dependencies in the data.

Teacher forcing ratio : Teacher forcing helps the model learn faster on training data, but with very high teacher forcing the model may never learn to generate sequences on its own. Therefore, this parameter is decayed as training progresses and the capacity of the model to autoregressively generate sequences improves.

6.8 Tabulated Experimentation Results

In the course of experimentation, the batch size was kept fixed at 64 and the teacher forcing ratio starts at 1 and decays 10% every epoch. Tabulated results for the model configurations tried have been presented below:

LSTM Hidden Size	Embedding Dimension	Val Loss
512	64	3.46
512	128	3.28
256	64	3.12
256	**128**	**3.07**

The loss curves for the best model are shown below:

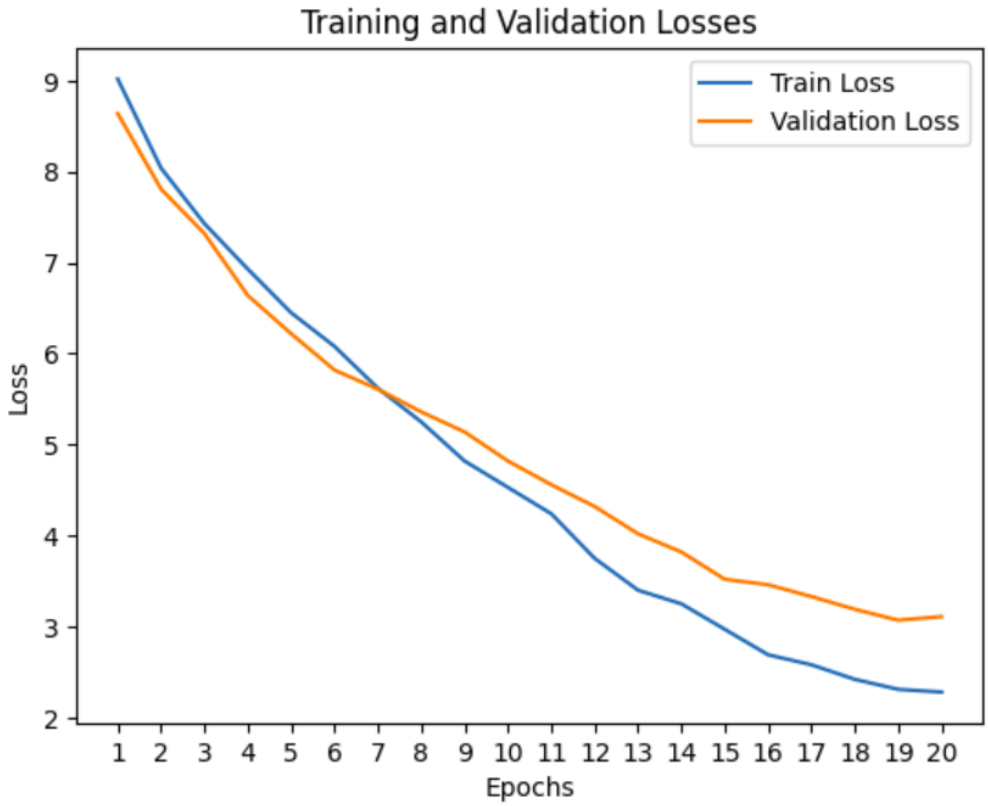


Figure 6.1: Loss curves

6.9 Results for Optimal Model Configuration

Based on experimental observations, the model chosen for further analysis was with embedding dimension 128 and hidden size 256.

The 1,2,3 and 4 gram BLEU scores for the model are as tabulated below:

Dataset	BLEU 1	BLEU 2	BLEU 3	BLEU 4
Training	0.636	0.446	0.391	0.246
Validation	0.578	0.353	0.252	0.200
Test	0.500	0.275	0.180	0.136

Some examples of translation on training data are enumerated below:

- – English: 'Mithali To Anchor Indian Team Against Australia in ODIs'
  – Predicted Hindi: आस्ट्रेलिया के खिलाफ इंडियन टीम की कमान मिताली
  – Expected Hindi: आस्ट्रेलिया के खिलाफ वनडे टीम की कमान मिताली को
- – English: 'Im fine by the grace of God.'
  – Predicted Hindi: ईश्वर की कृपा से बिल्कुल ठीक
  – Expected Hindi: ईश्वर की कृपा से मैं बिल्कुल ठीक हूं
- – English: 'I must go.'
  – Predicted Hindi: मुझे जाना चाहिए
  – Expected Hindi: मुझे जाना चाहिए

Some examples of translation on test data are enumerated below:

- – English: 'however it is a matter of investigation which is in progress he said'
  – Predicted Hindi: उन्होंने कहा कि मामले की जांच चल रही है।
  – Expected Hindi: उन्होंने बताया कि मामले की जांच चल रही है।
- – English: 'this information was'
  – Predicted Hindi: यह जानकारी
  – Expected Hindi: यह जानकारी एन
- – English: 'police are searching for the accused'
  – Predicted Hindi: पुलिस की की तलाश कर कर रही है।
  – Expected Hindi: पुलिस दोषियों की तलाश कर रही है।

6.10 Inferences

Some inferences drawn from the experimental data are:

- While a deeper LSTM learns better, it also overfits easily, hence the higher validation loss.
- A higher dimension of embedding helps a model of given configuration learn the sequence prediction task better.
- While the model is able to translate with some amount of accuracy, the results would be expected to be better with an attention layer as the LSTM encoder decoder model alone cannot focus on different parts of the input sequence when generating each part of the output sequence.
- The model is able to learn the words that make up the sentence but not necessarily get their order correct in the prediction. This can be inferred from the significantly higher 1 gram bleu score as compared to the 2 gram, 3 gram and 4 gram bleu scores.

6.11 Conclusion

The configured LSTM encoder decoder model was trained to translate English to Hindi with decent performance.