

# **CS6910: Fundamentals of Deep Learning**

## **Assignment # 2**

**Team-8**

**Aditya Raj - ED21B006**

**Annangi Shashank Babu - EE21B021**

**Ata Karim Subhani - EE21B025**

**Juswanth T - EE21B063**

May 27, 2024

## Contents

<b>1</b>	<b>Task 1</b>	<b>2</b>
1.1	Objective	2
1.2	VGGNet	2
1.3	GoogLeNet	4
1.4	Results	5
1.5	Analysis of Results	6
1.6	Learning Outcomes	6
<b>2</b>	<b>Task 2</b>	<b>6</b>
2.1	Objectives	6
2.2	Results	8
2.3	Ananalysis of Results	10
2.4	Learning Outcomes	10
<b>3</b>	<b>Task 3 and 4</b>	<b>11</b>
3.1	Objective	11
3.2	Project Setup	11
3.3	NetVLAD layer	11
3.4	Model Parameters and Hyperparameters	12
3.5	Results	13
3.6	Analysis	18
<b>4</b>	<b>Task 5</b>	<b>20</b>
4.1	Objective	20
4.2	Experimental Setup	20
4.2.1	Encoding	20
4.2.2	Decoder Initialization	20
4.2.3	Decoding Loop	20
4.2.4	Output	21
4.2.5	seq2seq model	21
4.2.6	Parameters of the model	21
4.3	Results	22
4.3.1	Loss Curve	22
4.3.2	Translations on Train Data	22
4.3.3	Translations on Test Data	23
4.3.4	BLEU table	23
4.4	Conclusion	23
4.5	Learning outcomes	24

## List of Figures

1	Loss and accuracy plots for VGGNet during training of top MLFFNN	2
2	VGGNet architecture attached to an MLFFNN with two hidden layers	3
3	GoogLeNet model architecture attached to an MLFFNN with two hidden layers	4
4	Loss and accuracy plots for GoogLeNet model with training only the top MLFFNN	4
5	Loss and accuracy plots for GoogLeNet model during learning of top and some inner layers of the pre-trained model (conv2d-945 layer to be precise)	5

6	model architecture summary . . . . .	7
7	Loss and accuracy plots . . . . .	7
8	Confusion matrix for train data . . . . .	8
9	Confusion matrix for valid data . . . . .	9
10	Confusion matrix for test data . . . . .	9
11	Architecture of encoder that we used. . . . .	12
12	Loss curve for both RNN and LSTM as decoder . . . . .	13
13	Encoder-Decoder Framework . . . . .	20
14	seq2seq model . . . . .	21
15	Parameters of the model . . . . .	21
16	Loss curve . . . . .	22

## List of Tables

1	BLEU Scores for image captioning task . . . . .	13
2	BLEU Scores . . . . .	23

# 1 Task 1

## 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 VGGNet

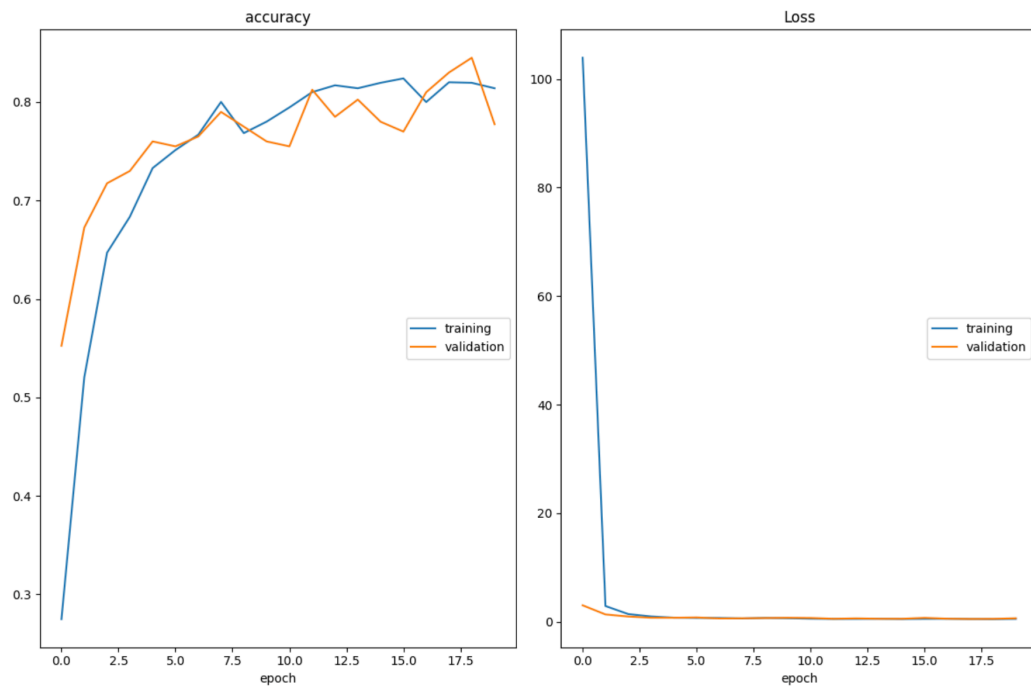


Figure 1: Loss and accuracy plots for VGGNet during training of top MLFFNN

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102764544
dense_1 (Dense)	(None, 1072)	4391984
dropout (Dropout)	(None, 1072)	0
dense_2 (Dense)	(None, 5)	5365

Total params: 121876581 (464.92 MB)  
 Trainable params: 107161893 (408.79 MB)  
 Non-trainable params: 14714688 (56.13 MB)

Figure 2: VGGNet architecture attached to an MLFFNN with two hidden layers

### 1.3 GoogLeNet

Model: "sequential"

Layer (type)	Output Shape	Param #
inception_v3 (Functional)	(None, 5, 5, 2048)	21802784
flatten (Flatten)	(None, 51200)	0
dense (Dense)	(None, 256)	13107456
batch_normalization_94 (Batch Normalization)	(None, 256)	1024
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
batch_normalization_95 (Batch Normalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 5)	645

Total params: 34945317 (133.31 MB)  
Trainable params: 13141765 (50.13 MB)  
Non-trainable params: 21803552 (83.17 MB)

Figure 3: GoogLeNet model architecture attached to an MLFFNN with two hidden layers

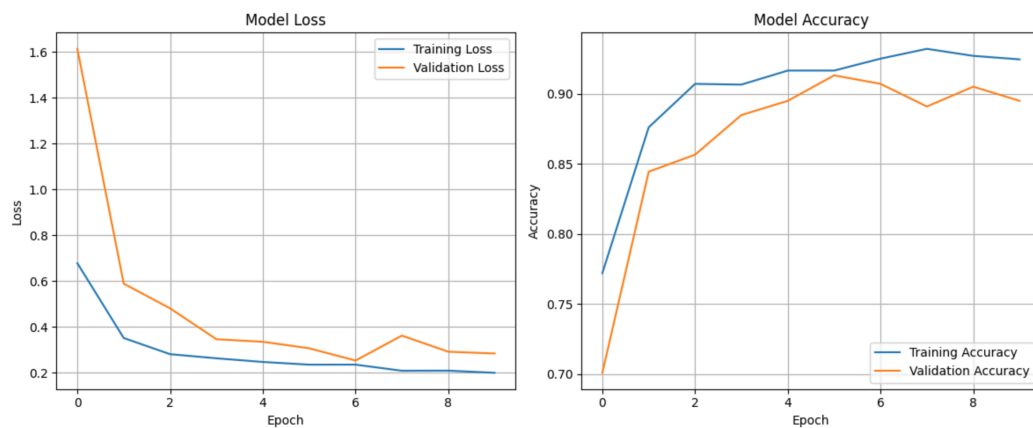


Figure 4: Loss and accuracy plots for GoogLeNet model with training only the top MLFFNN

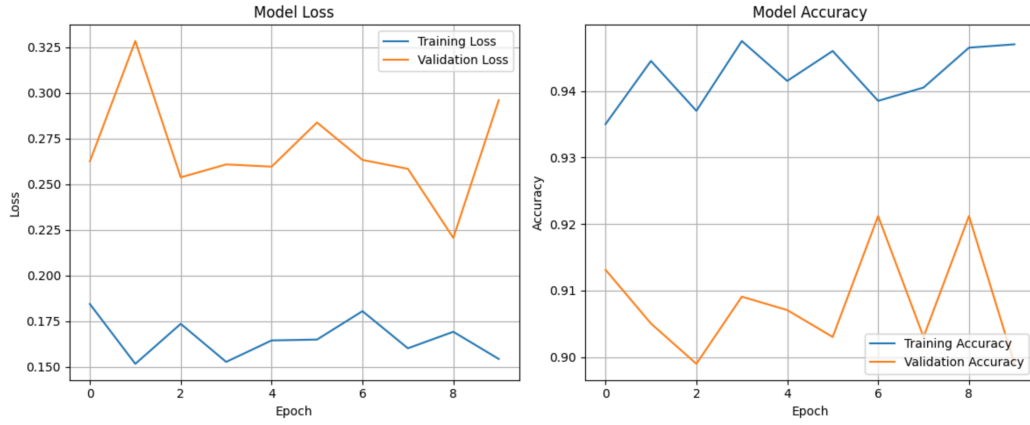


Figure 5: Loss and accuracy plots for GoogLeNet model during learning of top and some inner layers of the pre-trained model (conv2d-945 layer to be precise)

## 1.4 Results

For the VGGNet model, one training process was followed, where the top layer alone was trained. The accuracy achieved with this approach was 93.8%.

The hyperparameters used for training the VGGNet model were as follows:

- **Batch Size:** 200
- **Optimizer:** Adam
- **Learning Rate:** 0.001
- **Number of Epochs:** 20
- **EarlyStopping:** Monitor validation loss with patience of 5 epochs and restore the best weights
- **Dropout Rate:** 0.2

For the GoogLeNet model, two training methods were followed:

### 1. Only Top Layer Learning:

The accuracy achieved with only the top layer learning was 98.4%. In this method, only the top layers of the GoogLeNet model were trained, while the weights of the other layers remained frozen.

### 2. Internal Layers Learning:

When some internal layers were also allowed to learn, the accuracy increased to 98.8%. This approach involved fine-tuning the pre-trained GoogLeNet model by unfreezing and training some of its internal layers along with the top layers.

The hyperparameters used for training the GoogLeNet model (both methods) were as follows:

- **Batch Size:** 100
- **Learning Rate:** 1e-5
- **Optimizer:** RMSprop

- **Number of Epochs:** 10
- **Dropout Rate:** 0.4

## 1.5 Analysis of Results

The performance metrics indicate that GoogleNet has a higher accuracy compared to VGGNet. 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 filter sizes (1x1, 3x3, 5x5) in parallel. 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

## 2.1 Objectives

Objectives for task 2 are :

- To implement image classification using a specific CNN architecture.
- To compare the results with those of VGGNet and GoogLeNet pre-trained models.



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 4)	112
average_pooling2d (AveragePooling2D)	(None, 111, 111, 4)	0
conv2d_1 (Conv2D)	(None, 109, 109, 90)	3,330
average_pooling2d_1 (AveragePooling2D)	(None, 54, 54, 90)	0
flatten (Flatten)	(None, 262440)	0
dense (Dense)	(None, 1000)	262,441,000
batch_normalization (BatchNormalization)	(None, 1000)	4,000
dropout (Dropout)	(None, 1000)	0
dense_1 (Dense)	(None, 100)	100,100
batch_normalization_1 (BatchNormalization)	(None, 100)	400
dropout_1 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 5)	505

Total params: 787,643,943 (2.93 GB)  
 Trainable params: 262,547,247 (1001.54 MB)  
 Non-trainable params: 2,200 (8.59 KB)  
 Optimizer params: 525,094,496 (1.96 GB)

Figure 6: model architecture summary

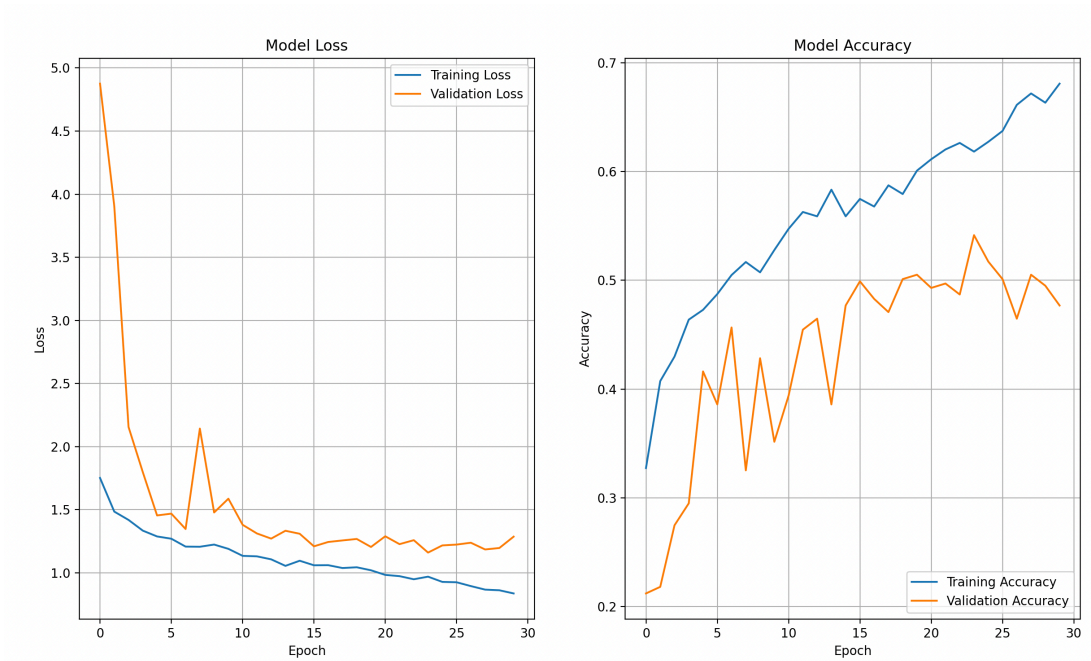


Figure 7: Loss and accuracy plots

## 2.2 Results

After conducting hyperparameter tuning, the best results were achieved with the following values:

- **Batch Size:** 128
- **Number of Epochs:** 50
- **Learning Rate (lr):** 0.001
- **Number of Feature Maps in CL2:** 90
- **Dropout Rate:** 0.3

These hyperparameters resulted in the accuracy of **58%** on the test set. These optimal hyperparameters were determined through a combination of grid search and subjective knowledge, aiming to achieve the highest accuracy and performance in the image classification task. Parameters found through grid search (Dropout Rate was fixed throughout the search):

Batch Size	Epochs	Learning Rate	Feature Maps in CL2	Validation Accuracy (%)
128	50	1e-4	100	57
128	50	1e-4	80	54
128	40	1e-3	80	54
128	40	1e-3	90	60
128	40	1e-3	80	56

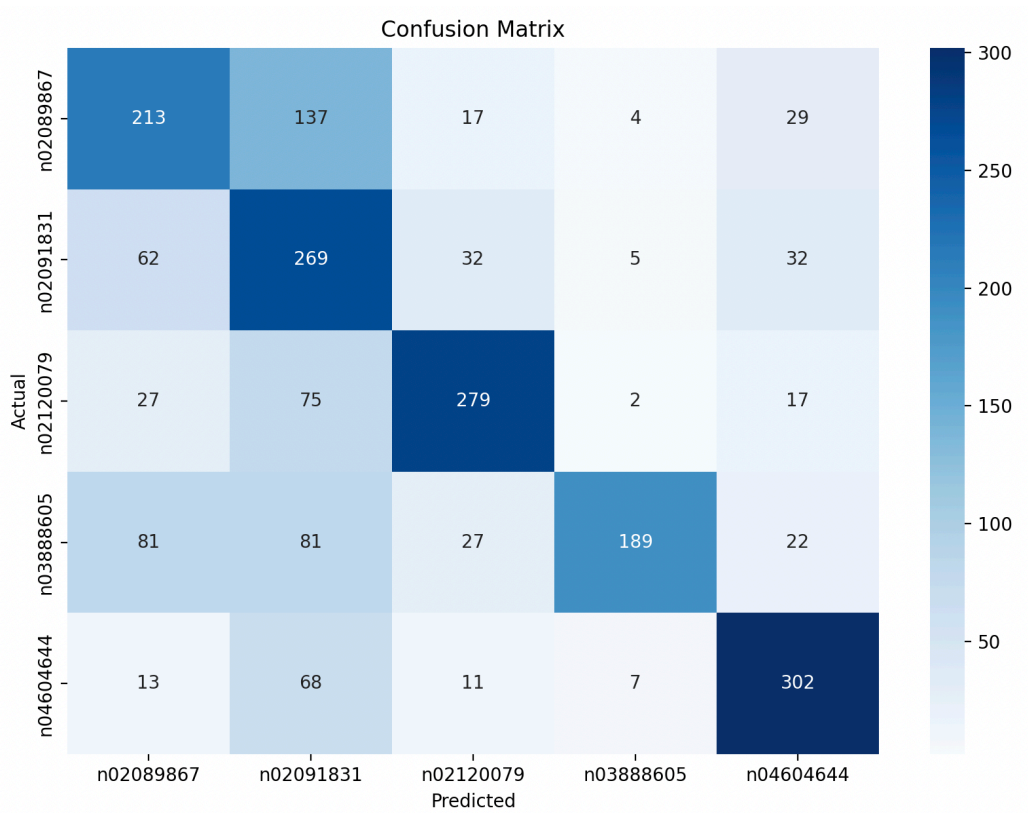


Figure 8: Confusion matrix for train data

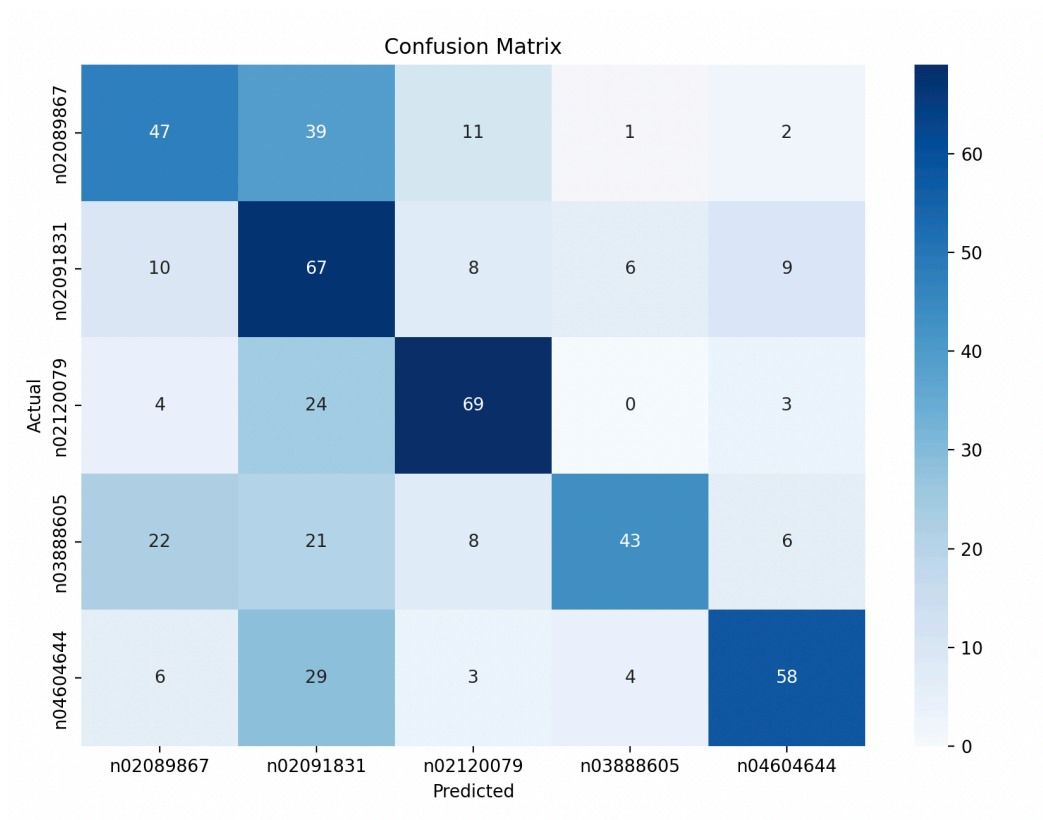


Figure 9: Confusion matrix for valid data

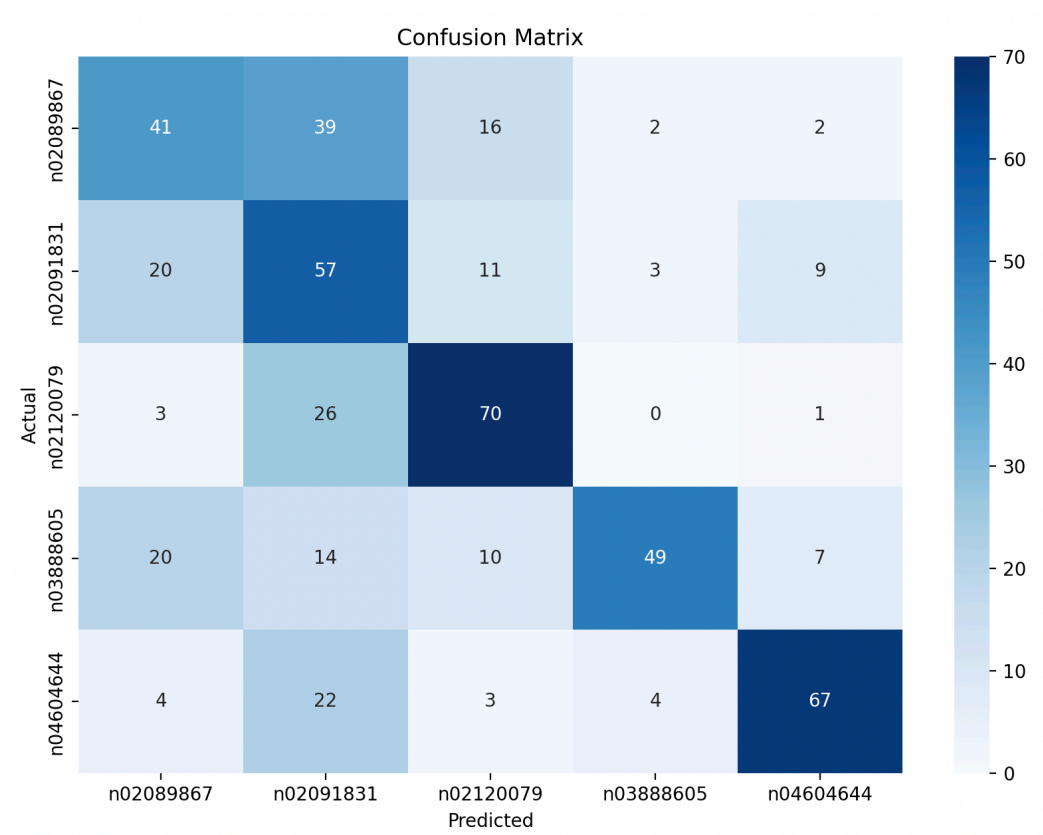


Figure 10: Confusion matrix for test data

## 2.3 Analysis of Results

Comparing the results from Task 1, the higher accuracy of the pre-trained model can be attributed to the following reasons:

1. **Transfer Learning:** pre-trained models, such as VGGNet and GoogleNet, are trained on large datasets, typically ImageNet, which contains a vast amount of diverse images. During this training, the pre-trained model learns generic features that apply to a wide range of image classification tasks. When fine-tuned on a specific dataset, the pre-trained model can leverage these learned features, resulting in better performance compared to a newly built model that starts training from scratch.
2. **Feature Extraction:** The convolutional layers of pre-trained models act as feature extractors, capturing meaningful patterns and representations from the input images. By utilizing these learned features, the pre-trained model requires less training data and fewer epochs to converge compared to a newly built model, which needs to learn these features from scratch.
3. **Regularization:** pre-trained models often incorporate regularization techniques, such as dropout and batch normalization, during training on ImageNet. These techniques help prevent overfitting and improve generalization performance. In contrast, a newly built model may require additional tuning and experimentation to achieve similar levels of regularization.

Overall, the pre-trained model's higher accuracy demonstrates the effectiveness of transfer learning in leveraging pre-existing knowledge from large-scale datasets to improve performance on specific image classification tasks.

## 2.4 Learning Outcomes

- **Model Construction:** Learned how to construct a custom CNN model in TensorFlow.
- **Grid Search:** Learned how to perform grid search to find the best hyperparameters, including the number of feature maps in CL2.
- **Hyperparameter Optimization:** Understood the impact of different hyperparameters on the model's performance and how to optimize them.

Task 2 provided comprehensive learning experiences in building and optimizing a custom CNN model from scratch using TensorFlow. Through this task, We gained valuable skills in model construction, layer configuration, and hyperparameter tuning.

### 3 Task 3 and 4

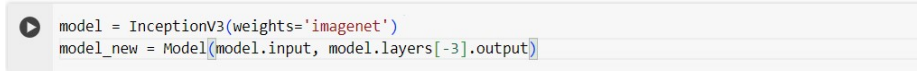
#### 3.1 Objective

The objective is to generate captions using an encoder-decoder model where the image encoder is CNN with NetVLAD layer and the decoder has to be RNN (task 3) or LSTM (task 4) with a single hidden layer.

#### 3.2 Project Setup

Tasks 3 and 4 are similar. Only we have to change the decoder part.

- First stage was to pre-process the data and split the data for training and validation. There were a total **8091** unique images present in the dataset. Initially, total **8102** images were there but few were repeated/corrupted images. So total number of images are 8091, out of which We used **7002** random images for training. **800** images for validation and the rest **300** images for testing the model.
- We used a pre-trained inceptionV3 module to extract the final 3-D volume to be fed into NetVLAD layer. We used this because it has a smaller weight file i.e. approx 96 MB and hence, fast to use for inference. Dimensions of the final convolution layer of



```
model = InceptionV3(weights='imagenet')
model_new = Model(model.input, model.layers[-3].output)
```

the inceptionV3 module is 8x8x2048. This 3-D volume is fed to the NetVlad layer.

- We resized the training image size to 299x299 as this is the expected dimension of the image by the Inception module.
- there are 5 captions for each image and we have preprocessed and encoded them in the below format “startseq “ + caption + “ endseq”.

The reason behind startseq and endseq is, **startseq**: Will act as our first word when feature extracted image vector is fed to decoder. It will kick-start the caption generation process. **endseq**: This will tell the decoder when to stop. We will stop predicting word as soon as endseq appears or we have predicted all words from the training dictionary whichever comes first.

#### 3.3 NetVLAD layer

$$\bar{a}_k(\bar{x}_i) = \frac{e^{W_k^T \bar{x}_i + \bar{b}_k}}{\sum_{k'} e^{W_{k'}^T \bar{x}_i + \bar{b}_{k'}}}$$

The soft-assignment of the input array of descriptors  $\bar{x}_i$  into K clusters can be seen as a two step process:

- A convolution with a set of K filters  $W_k$  that have spatial support  $1 \times 1$  and biases  $b_k$  producing the output  $\bar{s}_k(\bar{x}_i) = W_k^T \bar{x}_i + \bar{b}_k$
- The convolution output is then passed through the soft-max function to obtain the final “soft membership value” of the descriptor to all clusters.

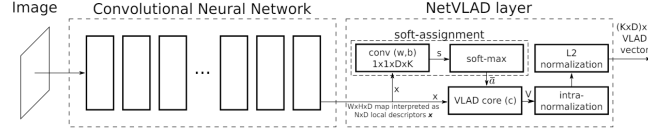


Figure 11: Architecture of encoder that we used.

- All the residual values are then passed through intra-normalization followed by L2-Normalization to get  $Kd \times 1$  VLAD vector which is the final representation of image. Where  $K$  is the number of clusters (a hyperparameter) and  $d$  is the dimension of each descriptor (2048 in inceptionV3 module case).

We have used NetVLAD layer implementation from the research paper which used this pooling technique for Place recognition task.

- Finally to match the dimension of the image vector with the dimension of hidden state vectors of decoder, we add a linear layer at the end of the encoder model and feed the encodings to the initial state of the decoder module.
- The GloVe embeddings of the reference captions are generated, padded for each batch, and fed to the decoder model as timestep inputs. The learning method used for training the RNN and LSTM is **Teacher Forcing**.

### 3.4 Model Parameters and Hyperparameters

- **Encoder Module** - CNN (InceptionV3 final 3-d volume) + NetVLAD layer followed by a linear layer with 256 neurons.
- **Decoder Module** - RNN and LSTM
- **Embedding and cell state size** = 256
- **Vocabulary size** = 8775
- **Epochs** = 100
- **Training mode** = mini-batch with **batch size 32**.
- **Learning Rate** =  $3e-4$
- **Optimizer** = Adam
- **Decoder training** = Teacher Forcing
- **Loss function** = CrossEntropy Loss
- **Training Time** = 10 hours (LSTM) and 10 hours (RNN)
- **Drop-out rate** = 0.4
- **Number of clusters**  $k = 10$

### 3.5 Results

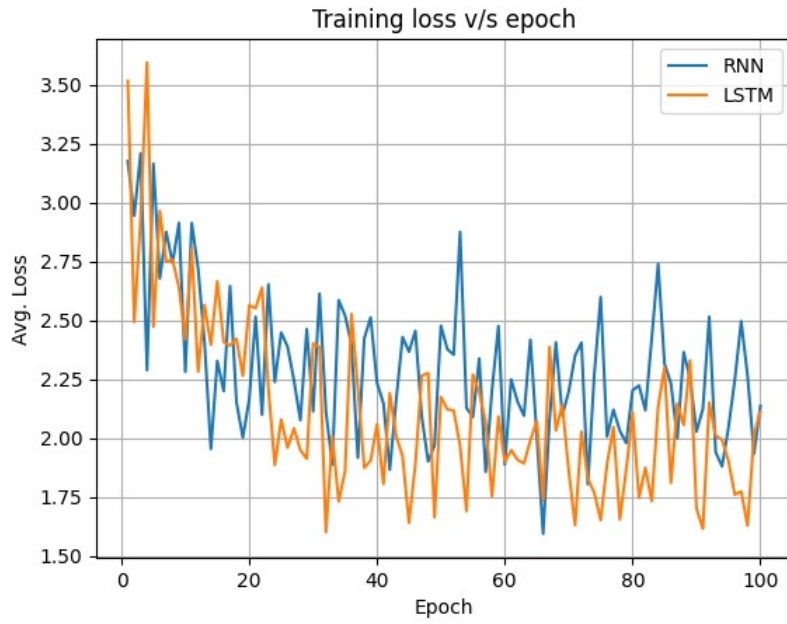


Figure 12: Loss curve for both RNN and LSTM as decoder

Decoder	BLEU-1	BLEU-2	BLEU-3	BLEU-4
RNN	0.3330	0.2135	0.3172	0.4060
LSTM	0.3349	0.2156	0.3190	0.4065

Table 1: BLEU Scores for image captioning task

#### Captions generation using RNN as decoder Training Dataset



Model's Output:

a man in a black shirt and jeans is sitting on a bench in a city .

References:

A person is reading a magazine on a snowing sidewalk .  
A person wearing jeans and a brown coat is reading and smoking on the sidewalk by a road covered with dirty snow .  
A woman reads on newspaper on a snow covered street .  
A woman smokes a cigarette and looks at a magazine in the middle of a snowy road .  
A woman smoking a cigarette is standing next to the road during a wintery day .





-----  
Model's Output:

a man is sitting on a bench in a city .

-----  
References:

A cyclist sits on some steps with his bike .  
A helmeted man sits on steps with graffiti on them , and a bike nearby .  
A man sits on the stairs next to his bike .  
A man with his bike amid a demolished building .  
Man with a bike sitting next to rubble .



-----  
Model's Output:

a child in a red shirt is jumping off a swing set .

-----  
References:

A boy on a black surfboard riding the wave on his stomach .  
A child in a red shirt is using a black wakeboard to splash over a beach wave .  
A kid with red hair boy boarding in the ocean  
A young boy with a red shirt surfing on a black surfboard .  
A young child wearing a red t-shirt rides a surfboard over a wave .

## Test Dataset





-----  
Model's Output:

a man and a woman are sitting on a bench in a park .

-----  
References:

A soccer player in blue is chasing after the player in black and white .  
The girl in the white strip is falling down as the girl in the blue strip challenges for the soccer ball .  
The girls are playing soccer .  
Two women in soccer uniforms playing soccer .  
Two young women on different teams are playing soccer on a field .



-----  
Model's Output:

a child in a red shirt is playing with a toy in a park .

-----  
References:

A brown , shaggy dog has a plastic snowman toy in his mouth .  
A furry brown dog carries his toy in mouth outside .  
A shaggy dog plays with a toy .  
A shaggy dog with a toy in its mouth .  
A small dog with long hair holds a snowman toy in its mouth .



-----  
Model's Output:

a man in a black shirt and jeans is sitting on a bench with a dog .

-----  
References:

A black and white dog running on a green grassy field .  
A black dog is running across a grassy field by a tall fence .  
a black dog runs through a field .  
A black dog runs very fast in a field in a park .  
A dog running at a park , with a park bench in the background .

### Captions generation using LSTM as decoder Training Dataset



-----  
Model's Output:

a man is standing on a rock with a cloud of water and a drum .

-----  
References:

A person is reading a magazine on a snowing sidewalk .  
A person wearing jeans and a brown coat is reading and smoking on the sidewalk by a road covered with dirty snow .  
A woman reads on newspaper on a snow covered street .  
A woman smokes a cigarette and looks at a magazine in the middle of a snowy road .  
A woman smoking a cigarette is standing next to the road during a wintery day .



-----  
Model's Output:

a man is sitting on a bench in the middle of a lake .

-----  
References:

A cyclist sits on some steps with his bike .  
A helmeted man sits on steps with graffiti on them , and a bike nearby .  
A man sits on the stairs next to his bike .  
A man with his bike amid a demolished building .  
Man with a bike sitting next to rubble .



-----  
Model's Output:

a small child is playing with a colorful toy outside .

-----  
References:

A boy on a black surfboard riding the wave on his stomach .  
A child in a red shirt is using a black wakeboard to splash over a beach wave .  
A kid with red hair boy boarding in the ocean  
A young boy with a red shirt surfing on a black surfboard .  
A young child wearing a red t-shirt rides a surfboard over a wave .

## Test Dataset



-----  
Model's Output:

a woman in a white shirt and black pants is playing with a black poodle .  
-----

References:

A soccer player in blue is chasing after the player in black and white .  
The girl in the white strip is falling down as the girl in the blue strip challenges for the soccer ball .  
The girls are playing soccer .  
Two women in soccer uniforms playing soccer .  
Two young women on different teams are playing soccer on a field .



-----  
Model's Output:

a small child is playing with a colorful toy outside .  
-----

References:

A brown , shaggy dog has a plastic snowman toy in his mouth .  
A furry brown dog carries his toy in mouth outside .  
A shaggy dog plays with a toy .  
A shaggy dog with a toy in its mouth .  
A small dog with long hair holds a snowman toy in its mouth .

### 3.6 Analysis

The following things can be inferred from the results demonstrated above:-

- **Decoder:** The model with the LSTM Decoder can be observed to be performing better than that with the RNN decoder. In the loss curve of the training, the model with LSTM decoder can be seen to converge faster than RNN. This is because the different gates in LSTM model help in selectively learning rather than learning the

entire sequence without special weightage, which is critical for NLP tasks such as image captioning. Here, in our application, since the caption’s length is not very large, both models have similar BLEU scores.

- **Transfer Learning:** We used the InceptionNet pre-trained model for the NetVLAD module. The InceptionNet module was not finetuned to save the training time and memory. Since the model was pre-trained and well-tested, the knowledge was transferred and used directly for our needs. The model proved to extract important descriptions from the images in our dataset and helped the model converge faster.
- **NetVLAD:** Since the model takes the NetVLAD representation of images as input, instead of relying upon the fully connected layer output of the CNN, the model was able to learn the pieces of information from the image more accurately.
- **GloVe:** The GloVe embeddings of the reference captions helped in better representation of them than the traditional SVD-based embeddings. This helped the model to converge faster than it would in case of SVD embeddings.
- **Training Time:** Both RNN and LSTM models were trained for 10 hours each with 100 epochs and 15 GB of GPU. This helped the model to learn the embeddings for a long duration which favored it.



## 4 Task 5

### 4.1 Objective

To investigate the performance of an LSTM-based encoder-decoder architecture for machine translation between English and Hindi. The effectiveness of the system is assessed using BLEU scores at different levels (BLEU@1, BLEU@2, BLEU@3, BLEU@4)

### 4.2 Experimental Setup

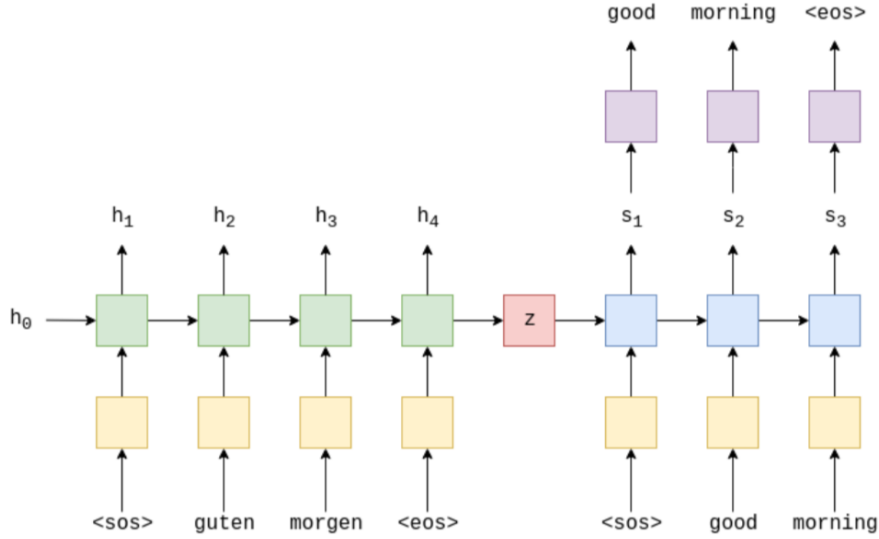


Figure 13: Encoder-Decoder Framework

#### 4.2.1 Encoding

The encoder processes the input sequence `src` and produces a final hidden state and a cell state. This hidden + cell state serves as the context vector, encapsulating the meaning of the input sequence, and is passed to the decoder.

#### 4.2.2 Decoder Initialization

The decoder is initialized with the context vector received from the encoder and a special start-of-sequence (`<sos>`) token as the initial input. This token signals the beginning of the generation process.

#### 4.2.3 Decoding Loop

The decoder iteratively generates the output sequence token by token. At each time step:

1. **Input:** The decoder receives its previous prediction as input. During training, **teacher forcing** is applied with a certain probability, where the ground truth token from the target sequence is used instead.
2. **Hidden State Update:** The decoder updates its hidden state based on the current input and the previous hidden state. This captures the evolving context as the output sequence is generated.

3. **Token Prediction:** The decoder produces a probability distribution over the vocabulary, representing the likelihood of each token being the next one in the sequence. The token with the highest probability is chosen as the output for that time step.

#### 4.2.4 Output

After iterating through all time steps, the model returns the complete generated output sequence, representing the translated version of the input sequence.

#### 4.2.5 seq2seq model

```
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.device = device

    def forward(self, src, trg, teacher_forcing_ratio):

        batch_size = trg.shape[1]
        trg_length = trg.shape[0]
        trg_vocab_size = self.decoder.output_dim
        outputs = torch.zeros(trg_length, batch_size, trg_vocab_size).to(self.device)
        # last hidden state of the encoder is used as the initial hidden state of the decoder
        hidden, cell = self.encoder(src)
        # first input to the decoder is the <sos> tokens
        input = trg[0, :]
        for t in range(1, trg_length):
            # insert input token embedding, previous hidden and previous cell states
            # receive output tensor (predictions) and new hidden and cell states
            output, hidden, cell = self.decoder(input, hidden, cell)
            # place predictions in a tensor holding predictions for each token
            outputs[t] = output
            # decide if we are going to use teacher forcing or not
            teacher_force = random.random() < teacher_forcing_ratio
            # get the highest predicted token from our predictions
            top1 = output.argmax(1)
            # if teacher forcing, use actual next token as next input
            # if not, use predicted token
            input = trg[t] if teacher_force else top1
        return outputs
```

Figure 14: seq2seq model

#### 4.2.6 Parameters of the model

```
Seq2Seq(
  (encoder): Encoder(
    (embedding): Embedding(15476, 100)
    (rnn): LSTM(100, 256)
    (dropout): Dropout(p=0, inplace=False)
  )
  (decoder): Decoder(
    (embedding): Embedding(23086, 768)
    (rnn): LSTM(768, 256)
    (fc_out): Linear(in_features=256, out_features=23086, bias=True)
    (dropout): Dropout(p=0, inplace=False)
  )
)
```

Figure 15: Parameters of the model

- English vocabulary size = 15,476
- Hindi vocabulary size = 23,086
- Optimizer = Adam
- Loss function = Cross Entropy
- Teacher forcing ratio = 0.5
- Clip = 1 (To prevent exploding gradient)
- No. of layers = 1
- Trainable parameters = 26,627,966
- Batch size = 10

## 4.3 Results

### 4.3.1 Loss Curve

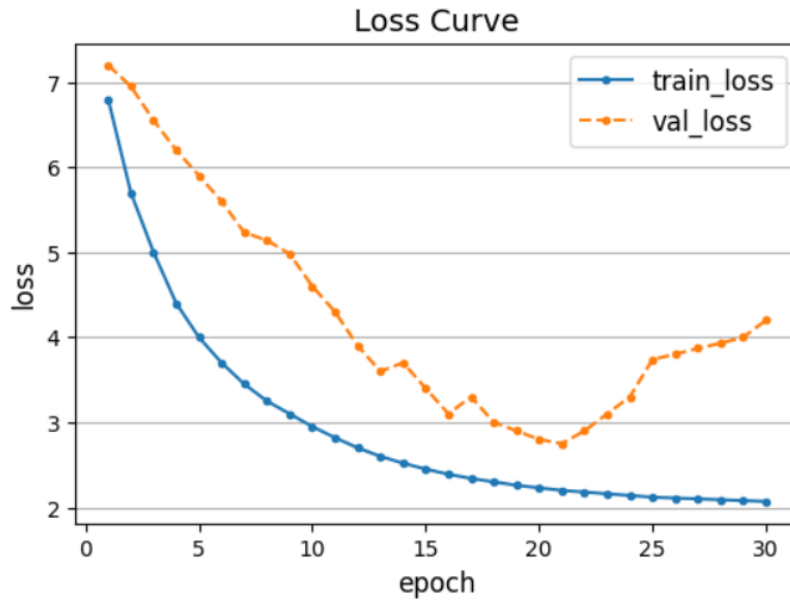


Figure 16: Loss curve

### 4.3.2 Translations on Train Data

- English : The film is directed by Kedar Sharma.  
Actual Hindi : यह फ़िल्म केदार शर्मा डायरेक्ट कर रहे थे.  
Predicted Hindi : यह रहा के केदार शर्मा डायरे के.
- English : These rooms are without attached bathrooms.  
Actual Hindi : इस कमरे में अटैच्ड बाथरूम नहीं है।  
Predicted Hindi : इस पर बाथरूम नह रहा है है
- English : Open rebuke is better than secret love.  
Actual Hindi : खुली हुई डांट गुप्त प्रेम से उत्तम है।  
Predicted Hindi : खुली डांट प्यार से गुप्त है।



### 4.3.3 Translations on Test Data

1. English: India is going to pay an extremely heavy price.  
Actual Hindi : भारत को बड़ी कीमत चुकानी पड़ी है।  
Predicted Hindi : भारत रहा रहा बड़ी पर चुकान है
2. English :Police reached the spot and controlled the situation.  
Actual Hindi : इस पर पुलिस ने मौके पर पहुंचकर हालात का जायजा लिया  
Predicted Hindi : इस होने पुलिस है मौके स्थिति का होने तीन है।
3. English : He was arrested in this connection.  
Actual Hindi : इस मामले में उसे गिरफ्तार किया गया था।  
Predicted Hindi : इस की में की गिरफ्तार की था

### 4.3.4 BLEU table

BLEU Score	Train	Valid	Test
BLEU-1	0.450626	0.324319	0.313974
BLEU-2	0.271154	0.146782	0.149421
BLEU-3	0.108062	0.081923	0.079822
BLEU-4	0.026438	0.009189	0.019023

Table 2: BLEU Scores

## 4.4 Conclusion

The BLEU scores suggest that the machine translation system for English to Hindi is struggling with translation accuracy, particularly with longer n-gram matches.

- **GPU Limitation:**

- We could not fine-tune the model much because of the unavailability of GPUs after a certain limit. As the training dataset is huge, the time taken for each epoch is also very huge (There are close to 26M parameters) and hence We trained until We had free GPU in Kaggle.
- the model is overfitting after the 21st epoch as the valid loss keeps increasing and it loses its ability to generalize. This could be possible because the amount of parameters is too high, by reducing it and after proper fine-tuning we could get better results.
- The loss curve for training is kind of smooth because it took only 10 as the batch size which is very small. And also We couldn't take higher batch sizes because of the constraints on the memory of GPU imposed in Kaggle.

- **Inadequate Model Capacity:** The model architecture ( a single-layer LSTM) might not be complex enough to capture the intricacies of translation, especially for longer sentences.

- **Use More Powerful Models:** Use more complex architectures like Transformer models, which have shown excellent results in machine translation tasks (As it uses attention)

## 4.5 Learning outcomes

Through the implementation of a machine translation model with LSTM encoder-decoder architecture, We gained a deeper understanding of LSTM networks and their role in sequential data processing. Additionally, We gained a deeper understanding about encoder-decoder architecture, machine translation techniques, and the integration of IndicBERT, which are essential for building effective translation models in natural language processing tasks.