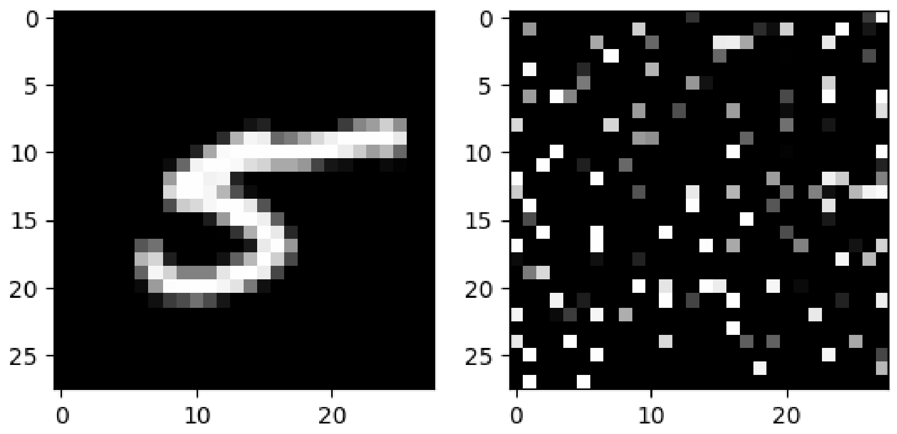
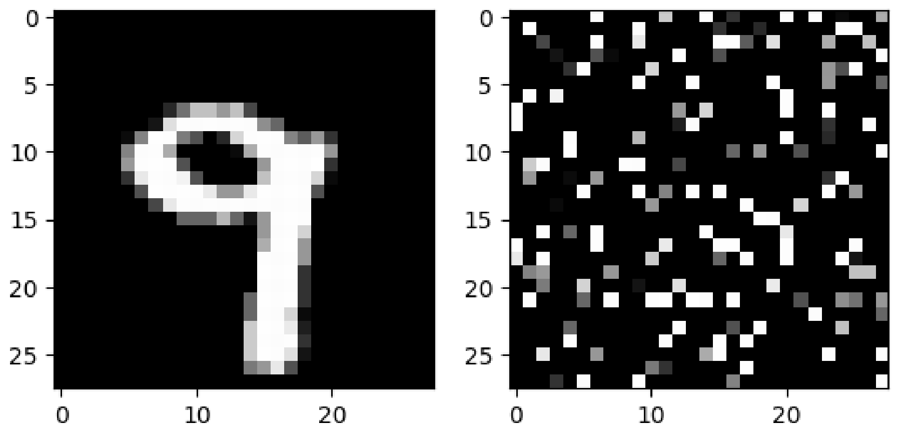
DL Assignment 3

Q1) 3 Random Images -

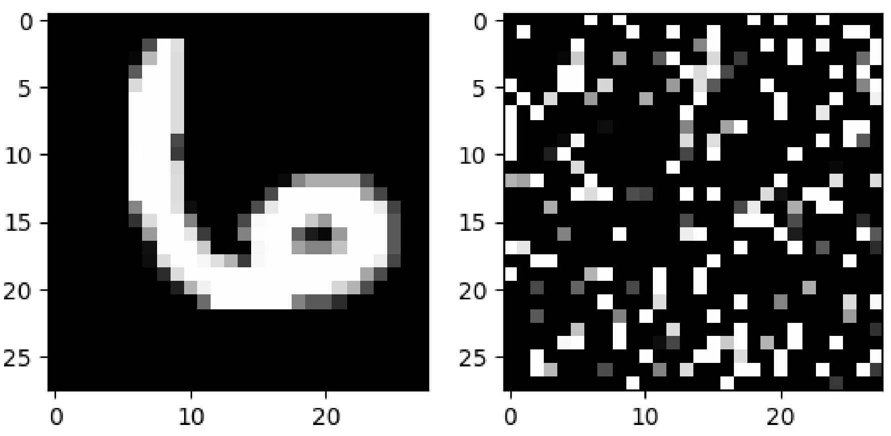
Sequential MNIST vs Permuted MNIST for digit 5



Sequential MNIST vs Permuted MNIST for digit 9

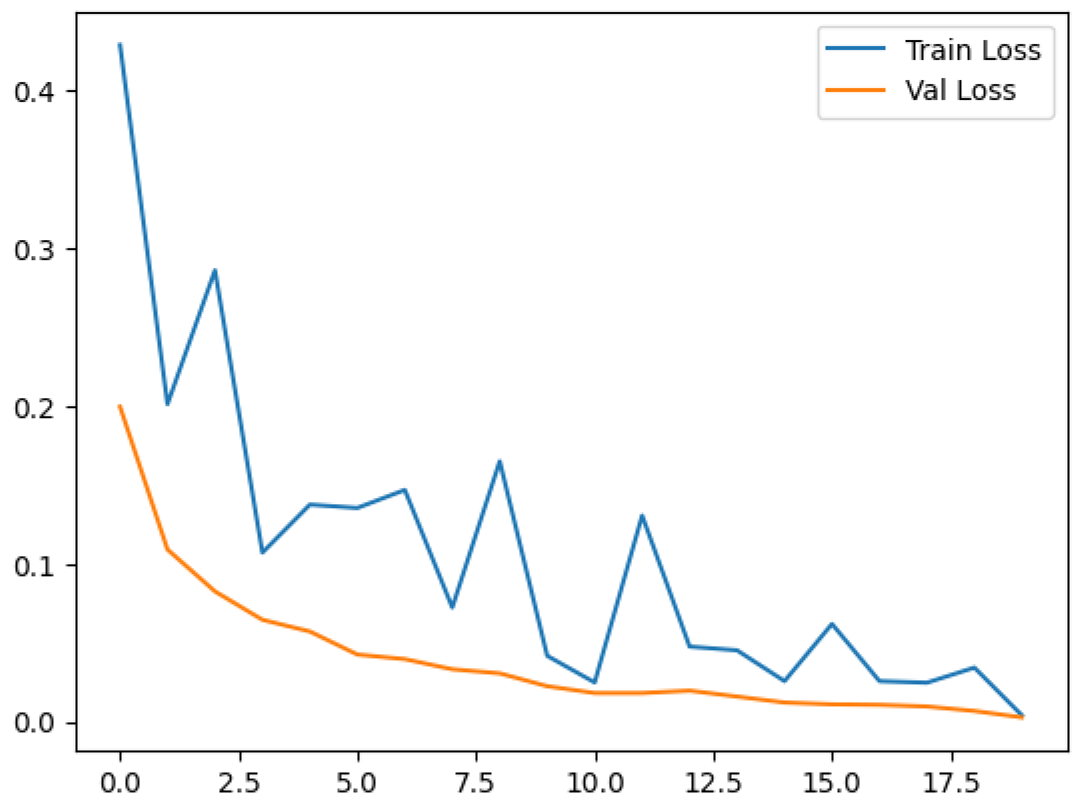


Sequential MNIST vs Permuted MNIST for digit 6

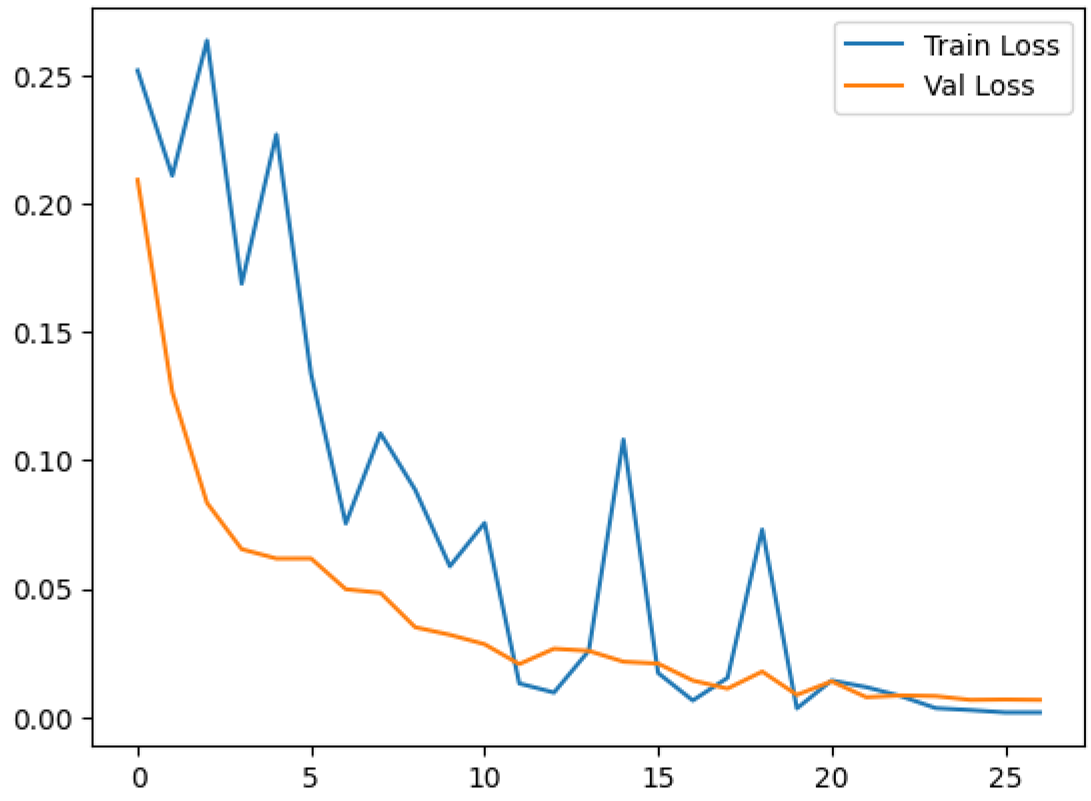


In total we tested 6 different architectures all present in the submission with 2 configurations (input normalized and input raw).

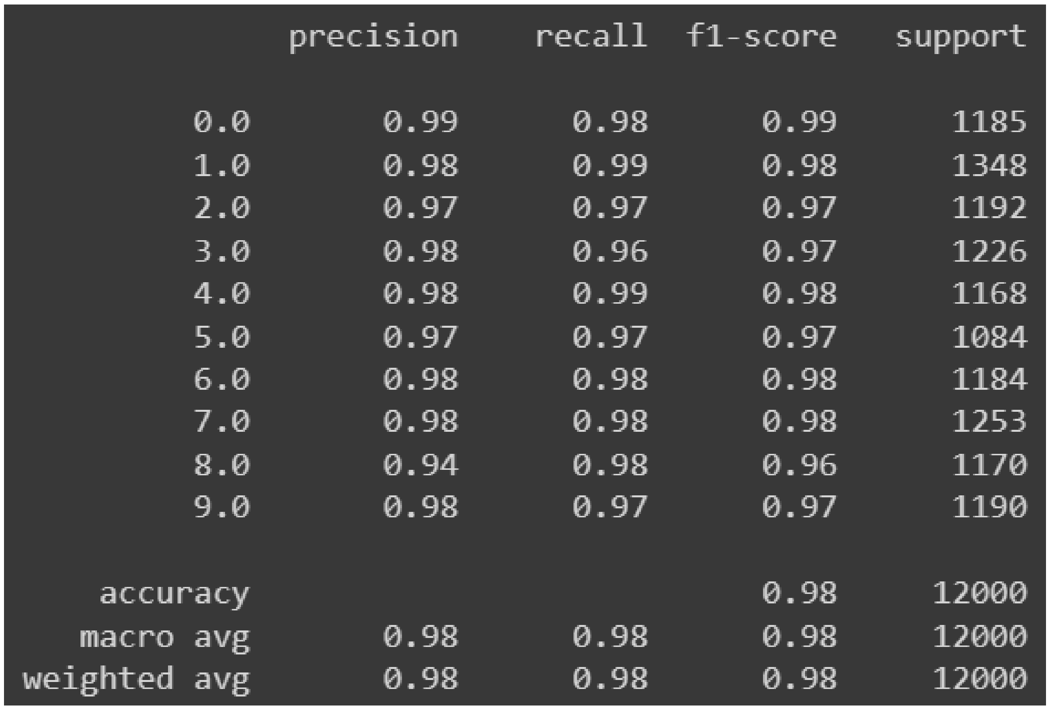
Sequential Loss Curve -



Permuted Loss Curve -

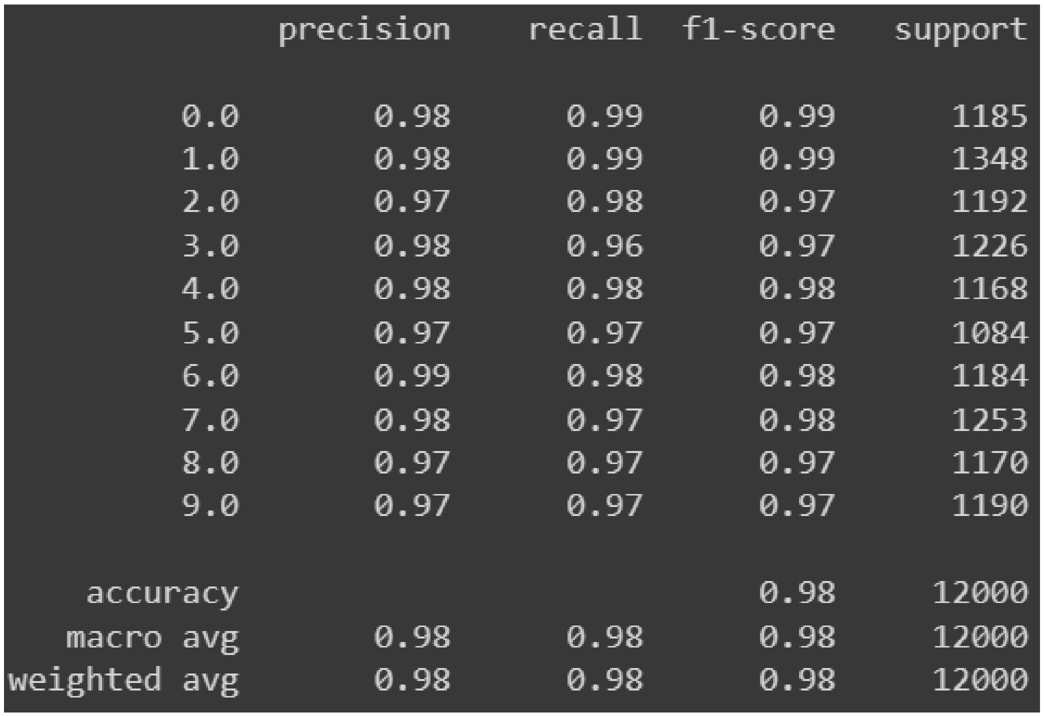


Sequential Class Wise F1 -



```
              precision    recall  f1-score   support

         0.0       0.99      0.98      0.99      1185
         1.0       0.98      0.99      0.98      1348
         2.0       0.97      0.97      0.97      1192
         3.0       0.98      0.96      0.97      1226
         4.0       0.98      0.99      0.98      1168
         5.0       0.97      0.97      0.97      1084
         6.0       0.98      0.98      0.98      1184
         7.0       0.98      0.98      0.98      1253
         8.0       0.94      0.98      0.96      1170
         9.0       0.98      0.97      0.97      1190

    accuracy                           0.98     12000
   macro avg       0.98      0.98      0.98     12000
weighted avg       0.98      0.98      0.98     12000
```

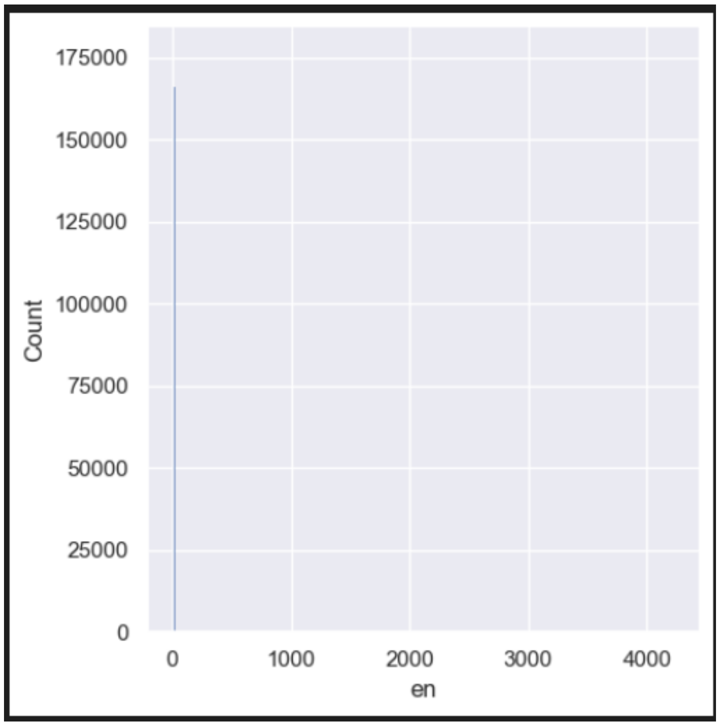
Permuted Class Wise F1 -



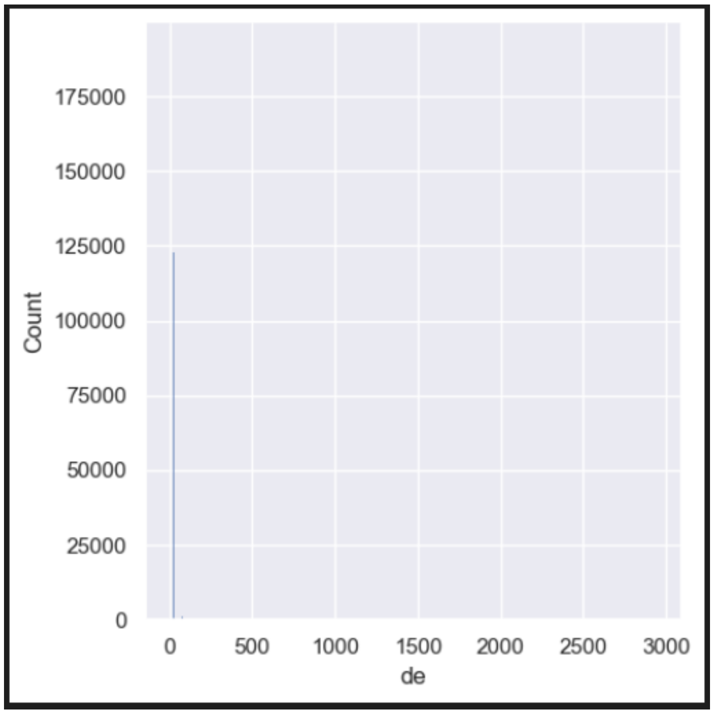
```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      1185
         1.0       0.98      0.99      0.99      1348
         2.0       0.97      0.98      0.97      1192
         3.0       0.98      0.96      0.97      1226
         4.0       0.98      0.98      0.98      1168
         5.0       0.97      0.97      0.97      1084
         6.0       0.99      0.98      0.98      1184
         7.0       0.98      0.97      0.98      1253
         8.0       0.97      0.97      0.97      1170
         9.0       0.97      0.97      0.97      1190

    accuracy                           0.98     12000
   macro avg       0.98      0.98      0.98     12000
weighted avg       0.98      0.98      0.98     12000
```

We ran our models only for one epoch in q2 and q3. We also only trained all our models on 0.5% of the training data.

Q2)
Visualization:
Sequence Length Distribution for English Language:



Sequence Length Distribution for German Sentences:



Top 10 most frequent words along with their frequency in the English Language vocabulary of the dataset:

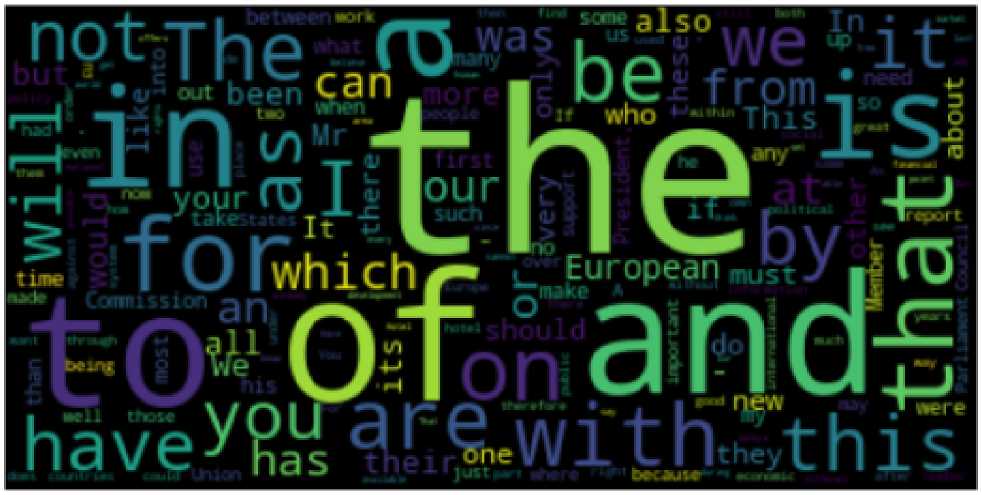
```
[('the', 6648005), ('of', 3481829), ('and', 3085861), ('to', 2937157),
('in', 2118292), ('a', 1874508), ('is', 1523959), ('that', 1154285),
('for', 1117157), ('on', 841380)
```

Top 10 most frequent words along with their frequency in the German Language vocabulary of the dataset:

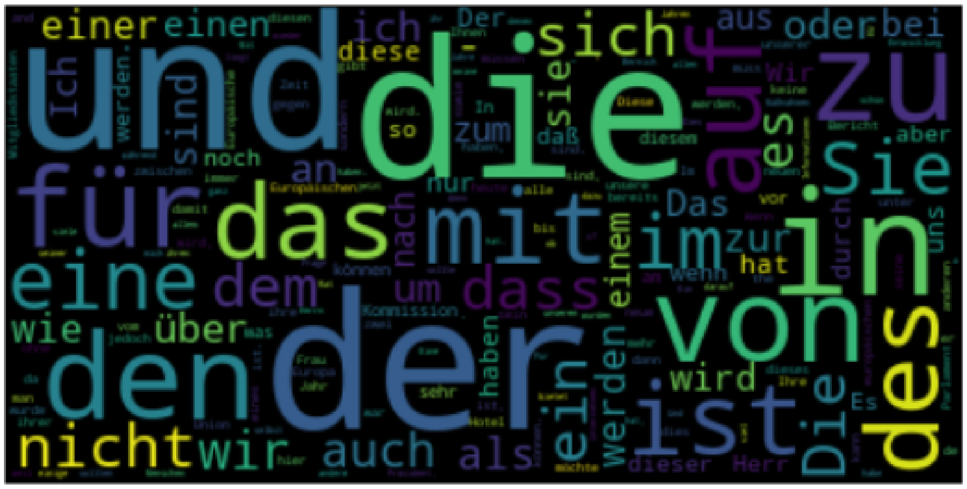
```
[('die', 3156285), ('der', 3044311), ('und', 2933148), ('in', 1559122),
('zu', 1155844), ('von', 1089068), ('den', 1080508), ('für', 894625),
('mit', 777083), ('das', 760142)]
```

English mean token length: 9.159616348596833
German mean token length: 12.019671461133381

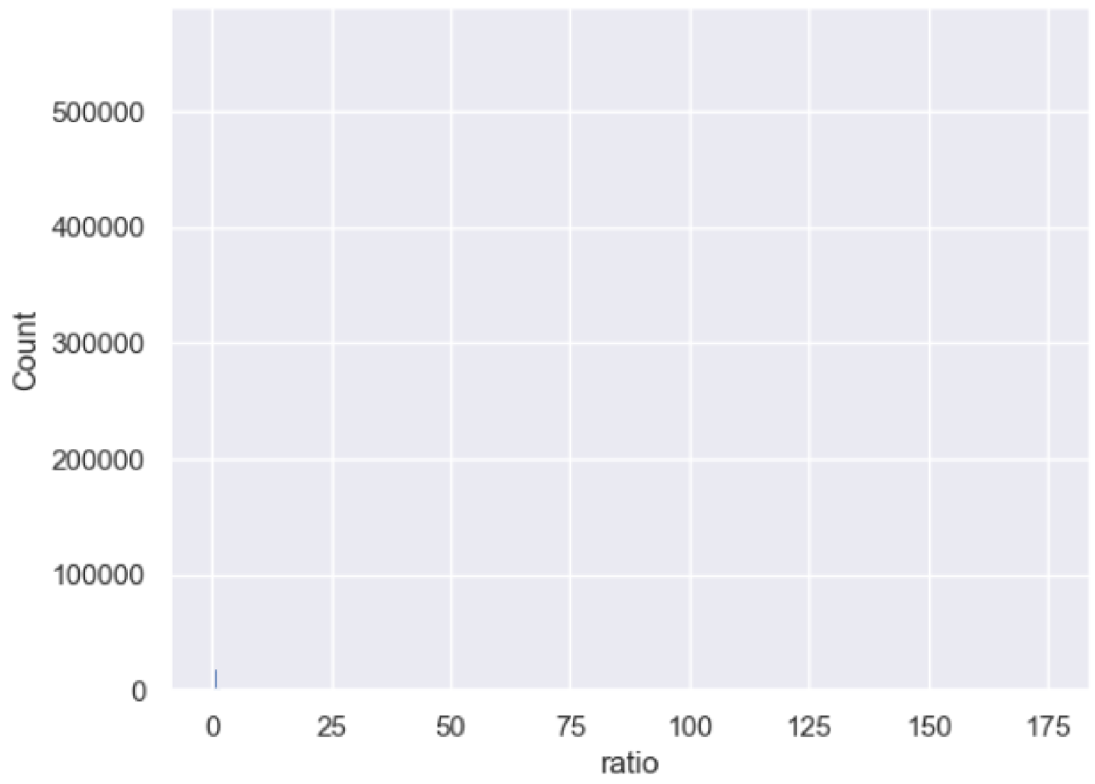
English Language WordCloud:



German Language WordCloud:



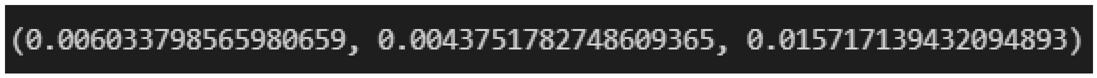
Ratio of token length of English sentences to their respective German translations:



We implemented all four types of attention scoring functions, but only used the “general” scoring for both global and local type attention. This is because in the original paper, scores for models using the “general” and “location” scoring function were the best amongst all.
We also only trained the local-m attention model for the local attention part because that model was the best performing in the original paper.

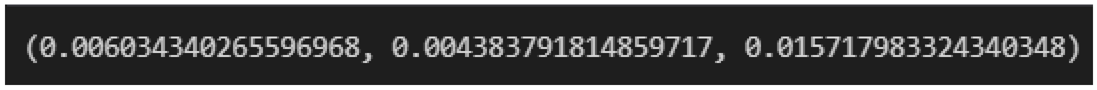
We didn’t get satisfactory results for our training. We reason that this might be because of lack of compute because we couldn’t find any implementation errors in our code.

- LSTM:
    - Training Loss: 10.404
    - Validation Loss: 10.405
    - Inference Bleu-1, Bleu-2, Rouge-L:



```
(0.006033798565980659, 0.0043751782748609365, 0.015717139432094893)
```

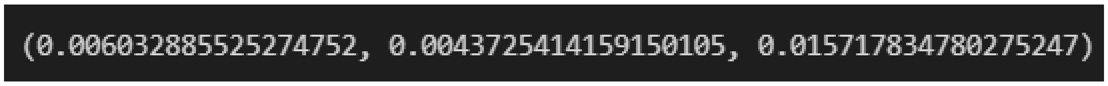
- LSTM + Global Attention
    - Training Loss: 10.410
    - Validation Loss: 10.406
    - Inference Bleu-1, Bleu-2, Rouge-L:



```
(0.006034340265596968, 0.004383791814859717, 0.015717983324340348)
```

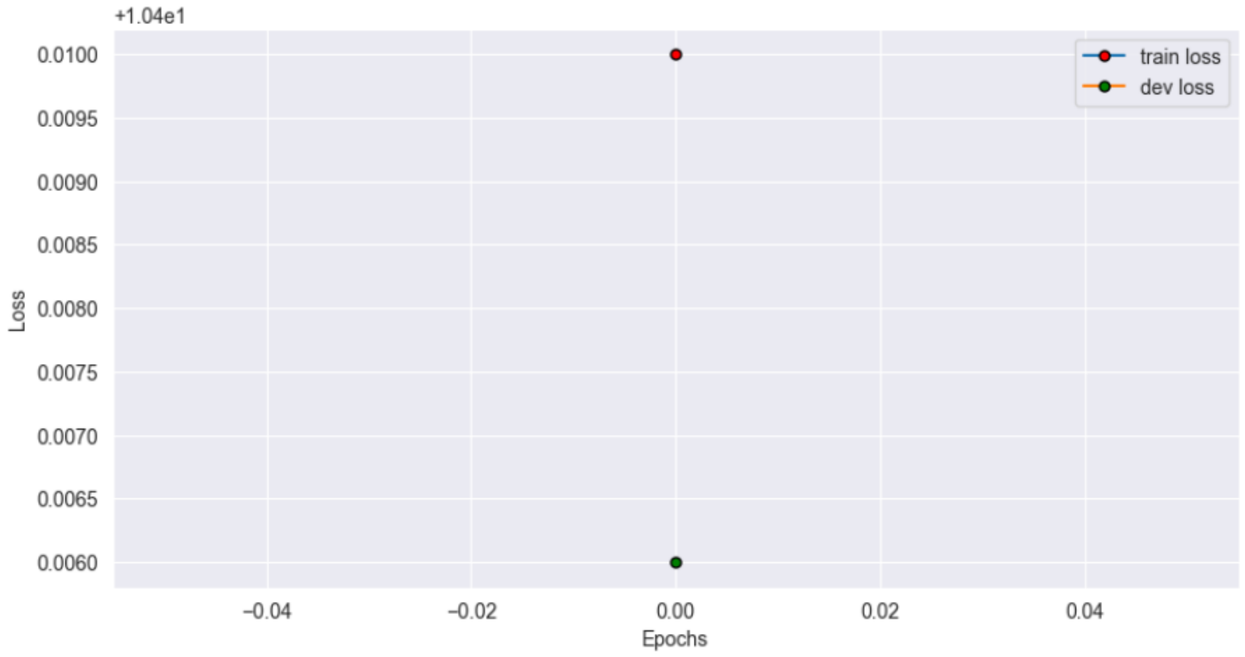
- LSTM + Local Attention

- Train Loss: 10.408
- Val. Loss: 10.405
- Inference Bleu-1, Bleu-2, Rouge-L:



```
(0.006032885525274752, 0.0043725414159150105, 0.015717834780275247)
```

Training and Validation Loss vs Epochs:

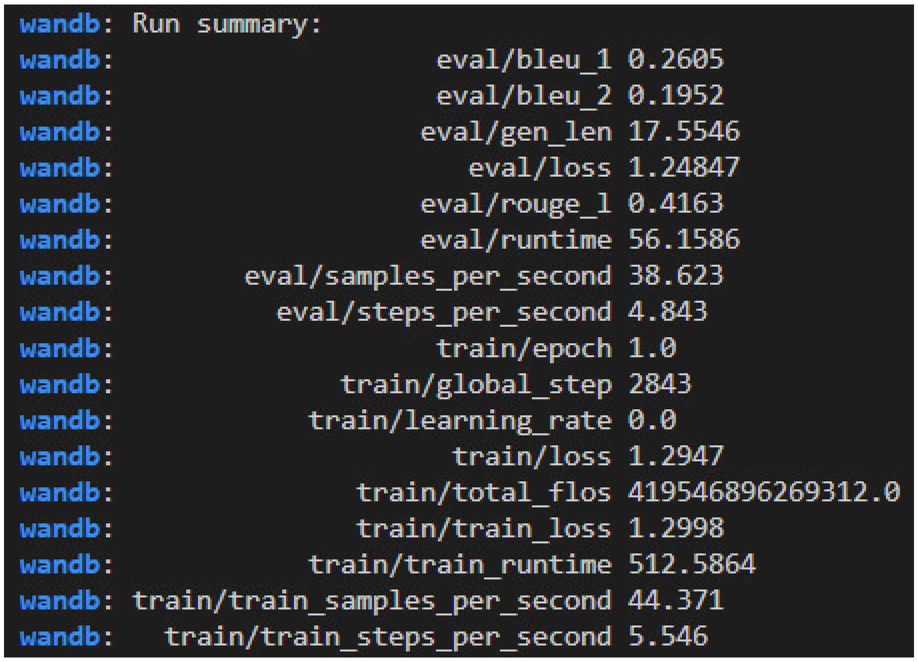


Because the training was only done for 1 epoch, and due to computational bottlenecks not much can be said about it.

Given our current setup and the conditions, its hard to say empirically what would generally perform better as we observed degenerate performance. According to the given scores, LSTM + Global Attention seems to have worked the best amongst the three, which makes sense as we have the full sequence length to attend on and hence we will be able to gather some information or the other from each token.

Q3) We finetuned the T5-small model, made available by huggingface for our machine translation task.

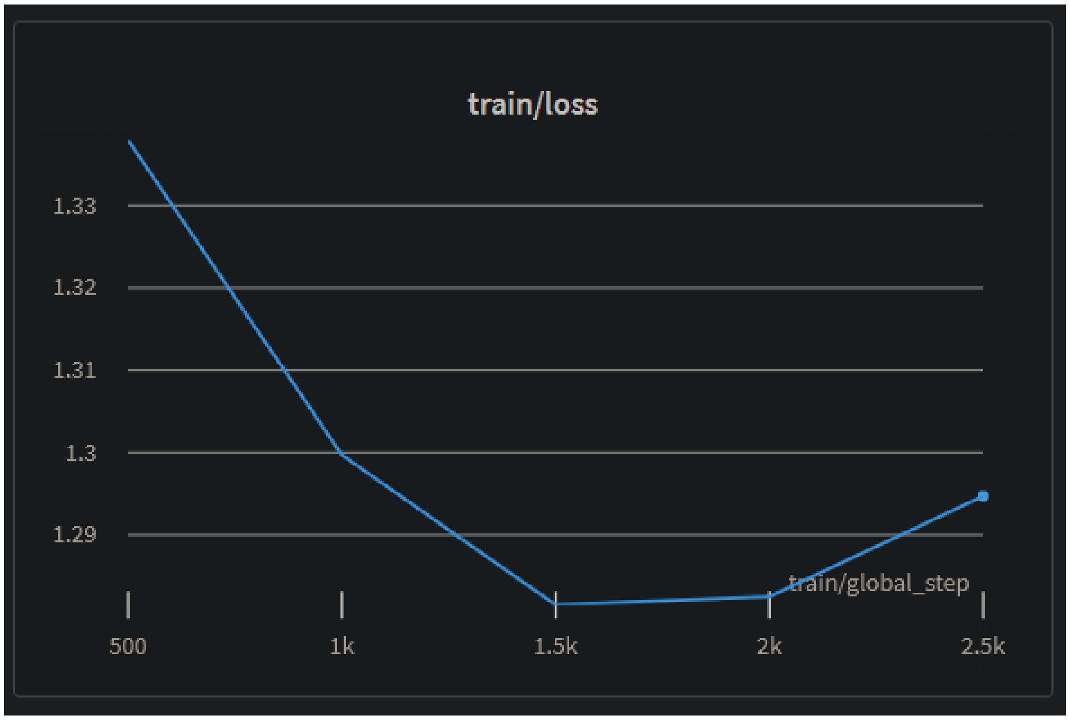
Training Details:



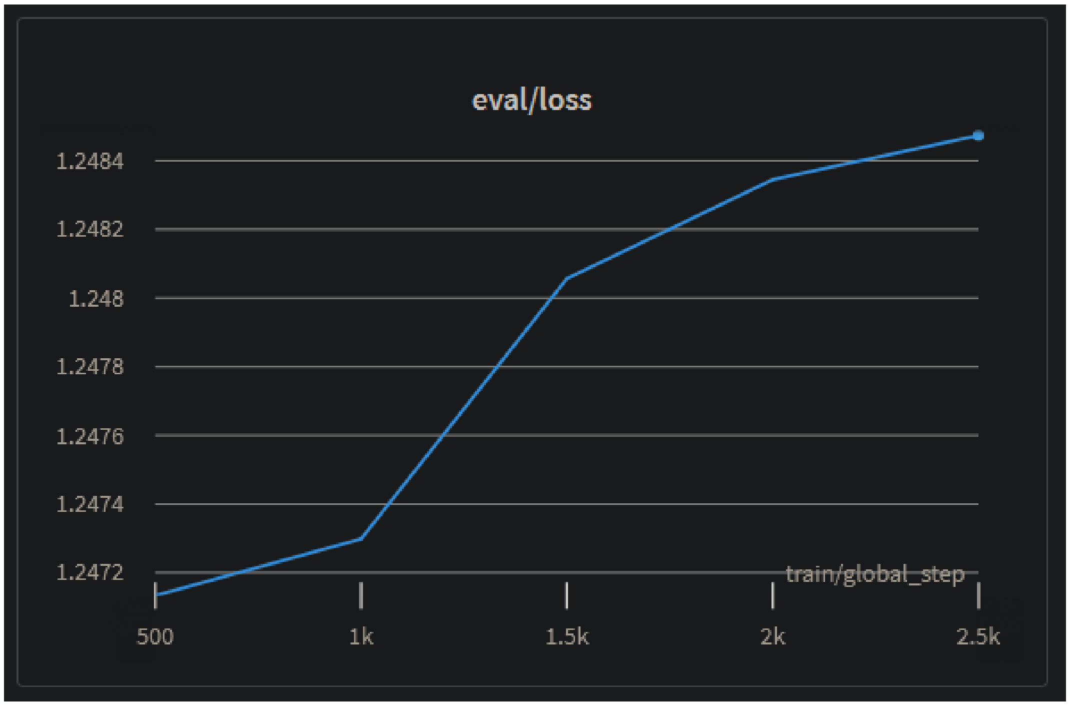
```
wandb: Run summary:
wandb:                   eval/bleu_1 0.2605
wandb:                   eval/bleu_2 0.1952
wandb:                  eval/gen_len 17.5546
wandb:                     eval/loss 1.24847
wandb:                  eval/rouge_l 0.4163
wandb:                  eval/runtime 56.1586
wandb:       eval/samples_per_second 38.623
wandb:         eval/steps_per_second 4.843
wandb:                   train/epoch 1.0
wandb:             train/global_step 2843
wandb:           train/learning_rate 0.0
wandb:                    train/loss 1.2947
wandb:              train/total_flos 419546896269312.0
wandb:              train/train_loss 1.2998
wandb:           train/train_runtime 512.5864
wandb: train/train_samples_per_second 44.371
wandb:   train/train_steps_per_second 5.546
```

Evaluation was done after every 500 steps, and the total steps the model was run for were 2843 steps.

Training Loss vs Steps:



Validation Loss vs Steps:



For inference on the test set, we used beam search with a beam size of 5.

Inference Results:
BLEU-1: 0.6285
BLEU-2: 0.4864
ROUGE-L: 0.5579
We observed that the quality of translations generated in Task-3 are far better than the ones generated in Task-2. This is because here we're finetuning a Transformer based network (T5) which uses many fancy topics like Self-Attention, Positional Embeddings and Parallelization. They are able to capture very long range dependencies because the whole sequence is processed together in transformers, in comparison to what happens with LSTMs, are able to hold the positional information using positional embeddings and also capture encode the input sequence better using self attention. LSTMs process sequences word-by-word which is why they can't capture very long range dependencies like the transformers. For LSTMs with a long input the gradient information still would need to travel across the whole sequence which will result in a lot of gradient loss/vanishing gradients. Transformers have access to information stored in all the steps because of self-attention, and hence they won't suffer from this specific problem.