Chapter 3: Transformers

October 28, 2024

In this part, we focus on the architecture of Vision Transformer model by implementing a naive version of the ViT model, experimenting on MNIST dataset, exploring the influences of different hyper parameters.

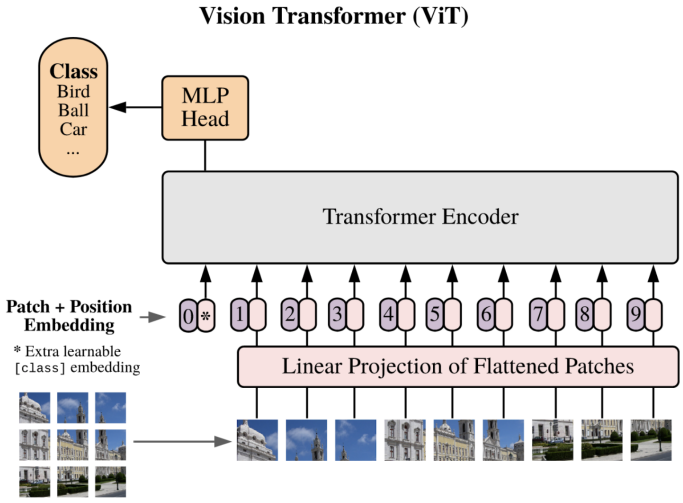

Figure 1: Vision Transformer model

It is demonstrated in the figure that an image is first split into fixed-size patches,then we linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence.

1 Self-Attention

Question 1. What is the main feature of self-attention, especially compared to its convolutional counterpart. What is its main challenge in terms of computation/memory?

- The main feature of self-attention is its ability to dynamically compute the relevance between different elements in an input sequence, allowing for the adjustment of each element's representation based on these computed relationships. This enables the model to capture long-range dependencies and global contextual information across the entire sequence in a single operation. In contrast, convolutional operations typically focus on local features within a fixed receptive field and require multiple layers to achieve a similar level of global understanding.
- The main challenge of self-attention in terms of computation and memory lies in its high computational complexity, which is $O(N^2 \cdot d)$, where N is the length of the input sequence and d is the embedding dimension. This means that as the sequence length increases, the computational load grows quadratically, making it resource-intensive. Additionally, self-attention requires storing a large attention matrix of size $N \times N$, leading to significant memory consumption, especially when dealing with long sequences or high-resolution data. This can become a bottleneck during training deep models.

Question 2. At first, we are going to only consider the simple case of one head. Write the equations and complete the following code. And don't forget a final linear projection at the end!

- the input X is transformed into three different representations : the Query Q, the Key K, and the Value V. This transformation is achieved through matrix multiplication with learnable weight matrices. Each vector type plays a distinct role in the attention mechanism:
  - the Query Q: Represent the current position that is looking for relevant information.
  - the Key K: Represent the information available in other positions that can be matched against the queries.
  - the Value V: Contain the actual information that will be retrieved based on the attention weights.

$$
\begin{aligned}
Q &= XW_q, \\
K &= XW_k, \\
V &= XW_v.
\end{aligned}
$$

- The attention scores are computed by taking the dot product of the query vectors with the key vectors,QK^T. This results in a matrix that quantifies the similarity between each query and all keys.
- The scores are then scaled by dividing by $\sqrt{d_k}$ (where d_k is the dimensionality of the query and key vectors) to prevent the dot products from becoming too large, which helps stabilize the gradients during training.

- The softmax function is applied to the scaled scores to convert them into a probability distribution, ensuring that the attention weights sum to one. This step highlights the most relevant values based on the calculated attention scores.

- Finally, the attention weights are used to compute a weighted sum of the value vectors V. This results in the output of the self-attention mechanism, where each position in the sequence contributes to the representation based on its relevance to the query.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^t}{\sqrt{d_k}}\right)V.$$

- Optionally, it's possible to do a last final linear projection at the then, in this case we have

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^t}{\sqrt{d_k}}\right)VW.$$

2 Multihead-Attention

Question 3.Write the equations and complete the following code to build a Multi-Heads Self-Attention.

- To concatenate multiple attention heads, given h heads, we compute the attention output for each head and concatenate them:

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W_O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$, W_O is the output weight matrix.

3 Transformer Block

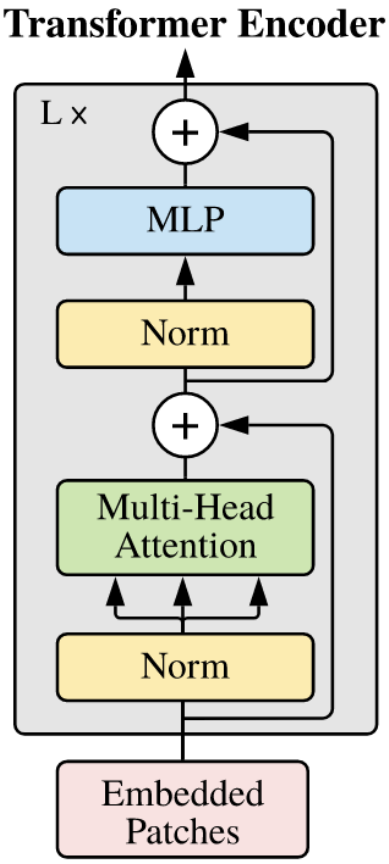

Figure 2: Transformer Encoder

Now, we need to build a Transformer Block as described in the image above.

Question 4. Write the equations and complete the following code.

1. Given an input X, where N is the sequence length and D is the embedding dimension, we first normalize the input:

$$\hat{X}_1 = \text{LayerNorm}(X)$$

2. Then, we compute the multi-head self-attention:

$$Q = \hat{X}_1 W_q,$$
$$K = \hat{X}_1 W_k,$$
$$V = \hat{X}_1 W_v.$$
$$\text{Attention} = \text{MultiHeadAttention}(Q, K, V)$$

3. We add a residual connection:

$$X_{\text{att}} = X + \text{Attention}$$

4. we normalize the result again:

$$\hat{X}_2 = \text{LayerNorm}(X_{\text{att}})$$

5. The MLP consists of two linear transformations with a GELU activation function in between:

$$\text{MLP}(\hat{X}_2) = \text{GELU}(\hat{X}_2 W_1 + b_1) W_2 + b_2$$

where $W_1 \in \mathbb{R}^{D \times \text{mlp\_ratio} \cdot D}$ and $W_2 \in \mathbb{R}^{\text{mlp\_ratio} \cdot D \times D}$.

6. We add another residual connection:

$$X_{\text{out}} = X_{\text{att}} + \text{MLP}(\hat{X}_2)$$

Thus, the final output of the block is given by:

$$\text{Output} = X_{\text{out}}$$

4 Full ViT model

Now we need to build a ViT model based on what are coded in the previous questions. There are additional components that should be coded such as the Class token, Positional embedding and the classification head.

Question 5. Explain what is a Class token and why we use it?

- The class token acts as an extra token that aggregates information from all patches during the self-attention process.
- **Purpose:** The output corresponding to the class token is used as the final representation for classification tasks. It gathers information from all the other patches through the self-attention mechanism, enabling the model to predict the class label based on the holistic understanding of the entire input image.
- **Why Use It?** The class token provides a mechanism for the model to produce a fixed-size output for classification regardless of the input sequence length. This approach is similar to how BERT uses a [CLS] token for natural language processing tasks.

Question 6. Explain what is the the positional embedding (PE) and why it is important?

- Positional embeddings are used to provide information about the position of each patch within the sequence and are added to the patch embeddings before being fed into the transformer layers.
- **Purpose:** The purpose of the positional embedding is to inject information about the spatial arrangement of patches in the original image, allowing the model to understand the relative positions of different patches.
- **Why is it Important?** In images, the arrangement of patches is crucial for recognizing shapes, textures, and objects. Without positional embeddings, the transformer would treat the input patches as a "bag of patches" without any notion of spatial arrangement, leading to suboptimal performance in vision tasks.

Here we use sinusoidal encoding, it uses fixed sinusoidal functions of different frequencies to generate positional embeddings. The advantage is that it does not need to be learned and can generalize to longer sequences than seen during training.

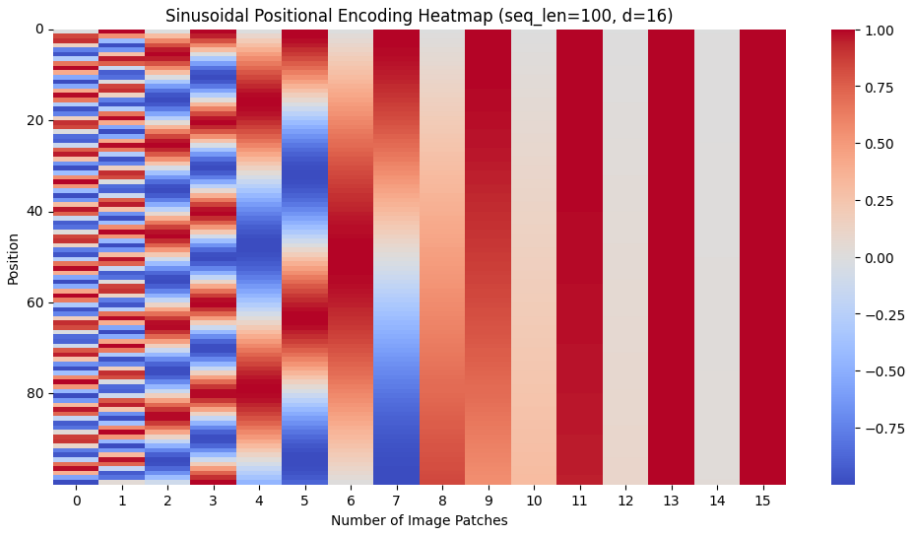

Figure 3: sinusoidal positional encoding with $seqlen$=100, d=16

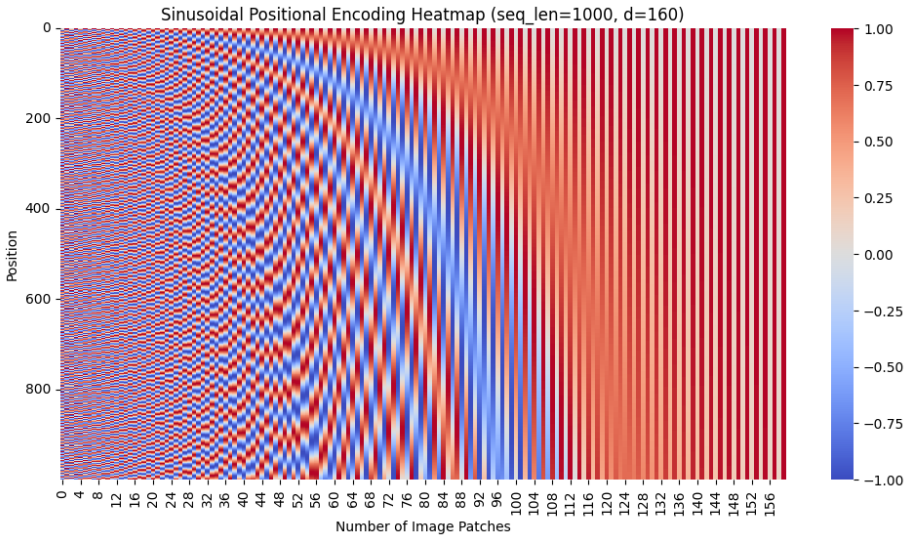

Figure 4: sinusoidal positional encoding with $seqlen$=1000, d=160

Figure 3 and Figure 4 are employed to visualize the appearance of sinusoidal encoding. In the context of an image transformer, the x-axis represents the number of image patches created, while the y-axis represents the size of an image patch embedding.

5 Experiment on MNIST

Question 7. Test different hyperparameters and explain how they affect the performance. In particular embed_dim, patch_size, and nb_blocks.

In this section we analyze the influences of different hyperparameters on the performance of transformer model, and we test it on MNIST dataset.

5.1 Influences of embed dimension

First, we take a look at the influences of embed dimension. In the experiment we test the performances of the model with embed dimension at [16,32,64,128].

In the context of a ViT model, the embed dim (embedding dimension) plays a crucial role in determining the model's capacity to represent and learn features from the input data.

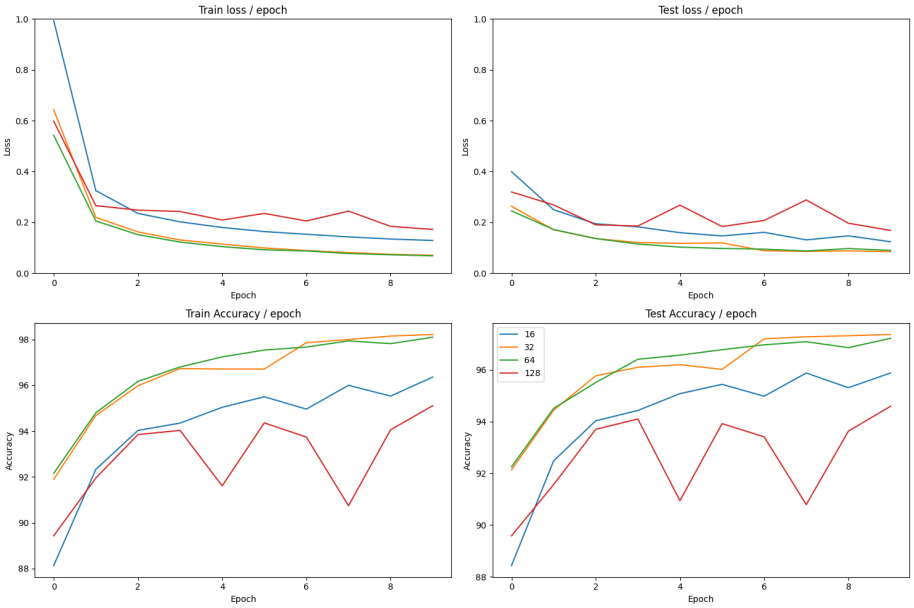

Figure 5: Influences of embed dimension

From Figure 5, we can see that the embed dimension of 16 converges slightly slower and the performance of embed dimension at 128 is worse than the others. And due to the increase of parameters, the embed dimension at 128 requires more training time as well. Meanwhile the embed dimension of 32 has the best performance with lower losses and higher accuracies in both training and testing.

5.2 Influences of patch size

Question 8. What is the complexity of the transformer in terms of number of tokens? How you can improve it?

We now examine the influences of patch sizes. We will also answer the question 8 since the number of tokens is closely related to the patch size.

Patch size by definition determines the size of each patch, and since the size of MNIST image is fixed with 28×28, meaning that with the increase of patch size, each patch covers a larger size of the image therefore we will have less patches and smaller number of tokens.

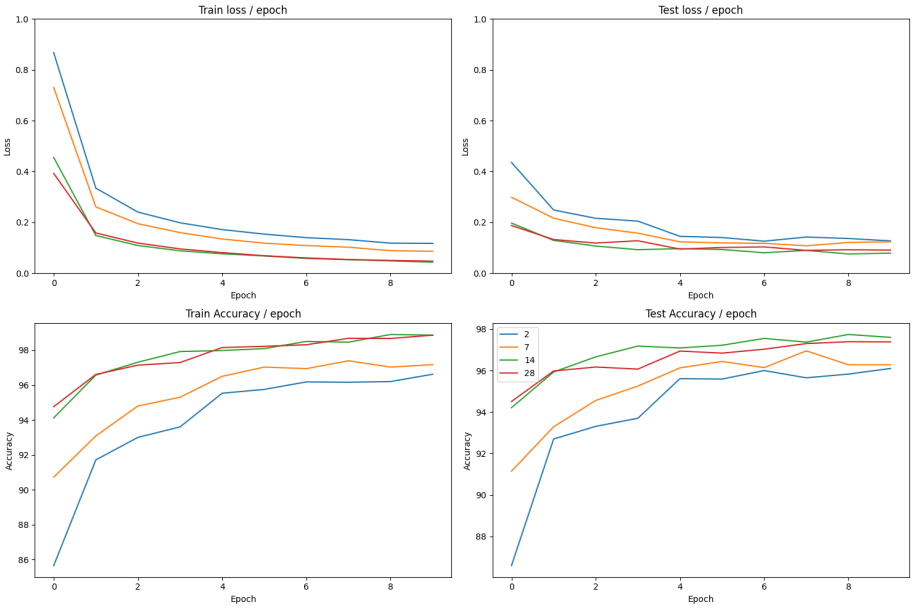

Figure 6: Influences of different patch sizes

From Figure 6, we can see that when the patch size is 2×2 pixels, the model has problem with the learning. And as for the patch size of 28×28, this is exactly the size of a MNIST image and we can consider it as no patch formed. We can conclude that when the patch size is set to 14×14, the results are ideal with more stable convergence and slightly lower losses and higher accuracies in the end.

5.3 Influences of number of blocks

Finally we focus on the influences of number of blocks. And in this section we experiment on different number of blocks on [2,4,6,8].

This parameter directly affects the depth of the model. More blocks mean a deeper architecture, which can capture more complex patterns and relationships in the data.

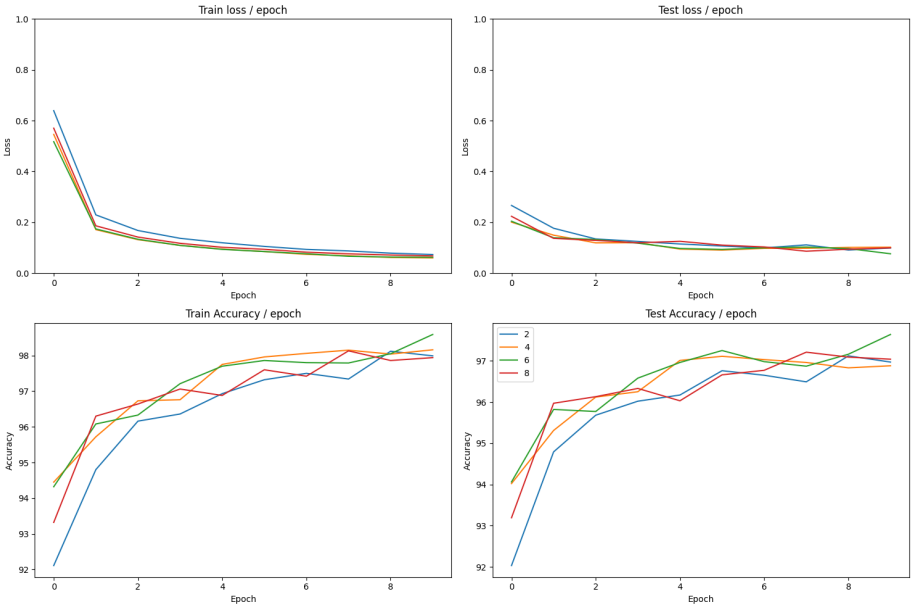

Figure 7: Influences of number of blocks

We can see from Figure 7 that when there are only 2 blocks, it seems to higher initial loss and lower initial accuracy but converges to a relatively good result. Meanwhile, we can also conclude that the higher number of blocks may result in instability as demonstrated when there are 8 blocks. And we see that the one with best performance is that of 6 blocks.

6 Larger Transformers

In this section we try to build model with bigger transformer and utilizes the timm library.

6.1 Questions

Question 9. Load the model using the timm library without pre-trained weights. Try to apply it directly on a tensor with the same MNIST images resolution. What it is the problem and why we have it? Explain if we have also such problem with CNNs. As ViT takes RGB images, the input tensor should have 3 channels.

- **The problem:** The `vit_base_patch16_224` are typically pre-configured to work with higher image resolutions 224x224. When using a ViT model without pretrained weights, the input tensor is still expected to have the corresponding dimensions. When we try to use a smaller resolution like

28x28 for MNIST images , the ViT model will not work as intended because the patch embedding layer and positional encoding might not be compatible with this input size.

- **The case with CNNs: Convolutional Neural Networks (CNNs)** are more flexible when it comes to input resolution because they operate in a fully convolutional manner. The convolutional layers can process different input sizes without requiring a specific image resolution. The only constraint in a CNN occurs in the fully connected layers, where the size of the input feature map needs to match the expected dimensions. In contrast, the ViT model has a strict requirement for the number and size of patches due to the self-attention mechanism, which is why using the wrong resolution leads to issues.

Question 10. Provide some ideas on how to make transformer work on small datasets. You can take inspiration from some recent work.

1. **DINO (Data-efficient Image Transformer):** By digging down the paper of "Emerging Properties in Self-Supervised Vision Transformers " we see how the DINO model applies self-supervised learning with Vision Transformers (ViTs) to achieve better performance on small datasets by using a self-distillation approach without labels.

   (a) **Self-Supervised Pretraining for ViTs:** Traditional ViT approaches rely on large-scale, supervised pretraining, but DINO adopts a self-supervised strategy inspired by NLP tasks like BERT's masked language modeling. The model learns by creating pretext tasks that use image transformations, rather than relying on predefined labels.

   (b) **Knowledge Distillation Framework**: DINO employs a knowledge distillation approach, where a student network learns to predict the output of a teacher network. The teacher network is built using a momentum encoder (an exponential moving average of the student parameters), which provides a high-quality, dynamic target for the student to match. This eliminates the need for a fixed pretrained teacher, making the model adaptable to different data scenarios.

   (c) **Efficient Training with Limited Resources**: DINO demonstrates competitive performance on benchmarks like ImageNet using only moderate computing resources (two 8-GPU servers over three days), which is significantly more efficient than some other self-supervised approaches. The framework's simplicity and flexibility allow it to be applied to both ViTs and conventional convolutional networks without requiring architecture changes.

   Overall, DINO combines the strengths of self-supervised learning, knowledge distillation, and multi-crop training strategies to make ViTs more effective on smaller datasets, providing a versatile and resource-efficient approach.

2. **SPT(Shifted Patch Tokenization)** and **LSA (Locality Self-Attention) :** By exploring in the paper of "Vision Transformer for Small-Size Datasets" , we found two methods to improve the ViT for small datasets:

   (a) **SPT (Shifted Patch Tokenization)** : SPT aims to enhance the locality inductive bias of ViTs by incorporating more spatial information into the visual tokens. This is achieved by spatially shifting the input image in multiple directions (e.g., four diagonal directions) and then concatenating these shifted versions with the original image. By increasing the receptive field of tokenization (the amount of spatial information each token covers), SPT helps the ViT learn better local structures in the data.

   (b) **LSA (Locality Self-Attention)** : LSA aims to make the self-attention mechanism of ViTs more locally focused by sharpening the attention score distribution. It uses **diagonal masking** that removes self-token relations by masking the diagonal elements of the similarity matrix (computed from the Query and Key), thereby emphasizing the relationships between different tokens. And it also uses the **adjusted softmax temperature** during training to sharpen the attention distribution. This helps mitigate the smoothing effect that standard softmax has on attention scores. By focusing more on inter-token relationships and adjusting the temperature, LSA increases the model's ability to learn local features, improving its performance on smaller datasets.

   Together, these two methods aim to boost the locality inductive bias of ViTs, making them more effective for visual tasks on small datasets by enhancing their capacity to learn local patterns.

6.2 Experiments

6.2.1 Learning from scratch

We tried the model with `pretrained=False`, which turned out to be a bad idea, because the learning process takes too long as it took nearly one hour to give results of all epochs, and the accuracies starts from aroung 30 percent.

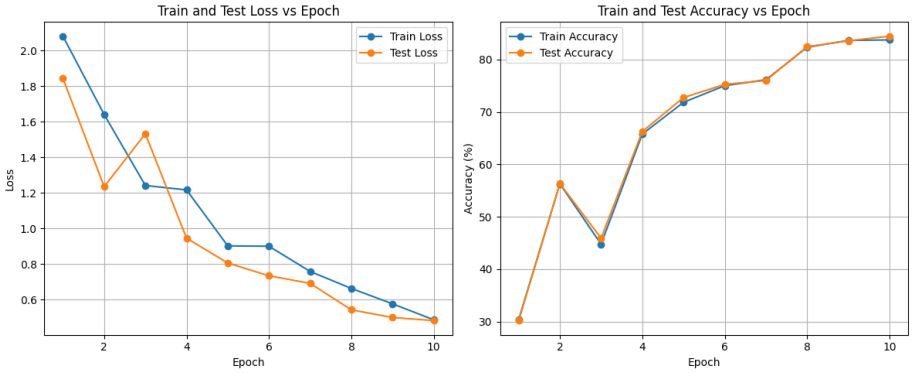

Figure 8: Learning from scratch

From the Figure 8 above we can see that the model starts from a high loss and very low accuracy and the results at the end are not ideal either. So this may be attributed to the limited dataset. And the learning process takes up too much space as well as time, which is not efficient in any way.

6.2.2 Learning with model pretrained on ImageNet

Conducting the experiments in the colab, we actually explored 2 approaches. Since the size of MNIST image are not compatible with the ViT model with higher resolutions. So we either transform the MNIST datasets to make it compatible with the model or the other way around. And by using the "trick" in timm library, we came to the conclusion that the latter where we use the `pretrained_cfg_overlay` is the better approach that learns much quicker.

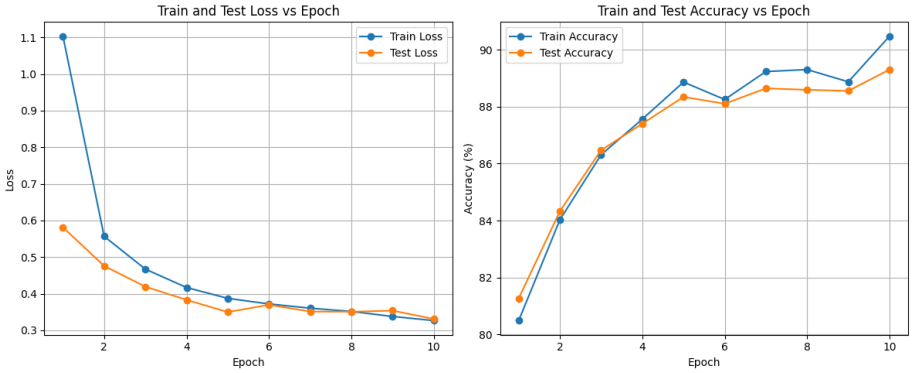

Figure 9: model pretrained

Compared to Figure 8, we can see from above that the Figure 9 shows a better result with the model pretrained on ImageNet. Compared to learning

from scratch, the pretrained model starts with lower losses and higher accuracies and achieves better results.

6.2.3 Learning with smaller pretrained model

Finally we tried a smaller model pretrained on ImageNet, `vit_small_patch16_224`

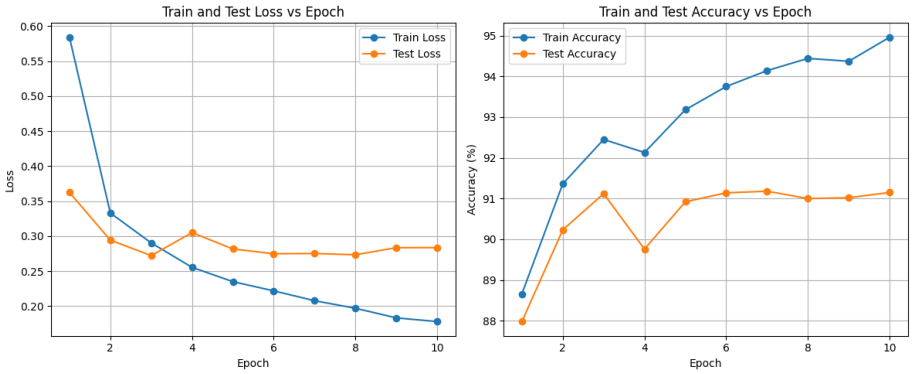

Figure 10: smaller model pretrained

In this smaller model, we obtain a even better result with much lower loss and higher accuracies. But the instability shown on the test accuracy and the insignificancy of the changes on test loss indicate that there is still space to improve the learning process of the model on smaller datasets as given in the answers of **Question 10**.

7 Conclusion

It is quite challenging to fine-tune the ViT model to obtain the optimal results.We need to choose proper patch size and number of blocks while avoiding overly large embedding s=dimensions. And from the experimental results with MNIST dataset, we can derive that the pretrained model evidently demonstrates better results and performs even better with smaller ViTs. We can also improve them with self-supervised learning approaches like **DINO** (Self-Distillation with No Labels) or **MAE** (Masked Autoencoder) and methods to boost capacity to learn local patterns like **SPT(Shifted Patch Tokenization) and LSA (Locality Self-Attention) .** We can also consider introducing additional regularization techniques to reduce overfitting on the small dataset, examples including Dropout and Stochastic Depth (that randomly drop layers during training, improving generalization.).