MAI Exercise 2 - Report of group 11

Viktor Maximilian Lehnhausen, Mikael Alves Brito, Georg Matthes
May 28, 2025

Contents

Question Part

1. ConvNet vs. ViT
ConvNets use convolutions and spatial pooling. It extracts the features through stacked convolution and pooling. Input pixels may overlap due to convolutions.
Drawbacks of ConvNet approach:

- Architecture is fixed, may not adopt to all tasks.
- Convolutions are local, to capture long-range dependencies deep architectures or large kernels are required.

ViT splits an image into a grid of fixed-size, non-overlapping patches. These patches are projected into an embedding. The patch embeddings can interact with each other via self-attention in the transformer encoder.
Drawbacks of ViT:

- Requires massive datasets to compensate of inductive biases.
- Depend on learned positional embeddings, may not generalize to other aspect ratios or resolutions.
- Memory and computational expensive.

2. Invariance and equivariance

Conv-Layer:
→ translation-equivariance, but not translation invariance.
The picture will be "scanned" with the same filter at each position. For that reason the output will be just shifted

ViT-Block:
→ neither
The positional embedding stays the same. For example, when the top-left corner of an image is normally at Patch 1 and with translation at Patch 3, the content of that corner is now associated with a different positional embedding and therefore the whole Output is changed.

ConvNet:
→ translation-equivariance and approximately translation-invariance
A complete ConvNet is translation-equivariance, because all the layers are translation-equivariance A ConvNet is per definition not translation-invariant. But with mechanics like pooling translation-invariant can be achieved approximately. Pooling helps with concentrating on the existence of features and not their position

ViT:
→ neither
When transforming an image the fixed positional embeddings change and thus whole output is changed. This is true for all the blocks and therefore for the whole Transformer

3. Model components
A ConvNet consists out of parameterized modules, composite activations and normalization layers.
ConvNet:

- Conv2D: learns kernels to identify features. Helps with translation-equivariance.
- BatchNorm2d: normalization, stabilizes the activation function.
- Activation function e.g. ReLU: non-linear activation function used in Neural Networks
- Pooling: reduces resolution by using a filter e.g. with the max operation. Helps with translation-invariant.
- Residual Connection: preserve information, helps training deep Neural Networks.

ViT:

- Linear Projection of Flattended Patches: makes batch to input vector
- Patch embedding: transforms patches into embeddings
- Pos embedding: optional with cls-token (learnable classification token)
- LayerNorm: normalization. Used to stabilize the activation
- Multi-Head Self-Attention: gather contextual information between patches
- Residual Connection: preserve information, helps training deep Neural Networks
- MLP: Processing each token individually to extract more information

Coding Part

1. b) A. Identify and explain the cause for the lower training accuracy in our code.
On the train loader we apply a different transformation, this is to enhance the model's ability to learn spatial invariance, as previously discussed in the question part task 2 a ViT has no invariance or equivariance to spatial translations. The validation loader uses a simpler transformation without any spatial translations, thus the correct evaluation method applies the same transformation to the train and validation loaders. Either we use the train transform on both of them or the test transform. But it is important that our training data used during training is still using the train transform with the random translation to learn the spatial translation. We decide to use the test transform for evaluation, as this aligns the most with the real images from the dataset.
1. c) ii. Write about your observations, drawing on your intuition and hypothesizing to explain the results. BasicBlockv2:
We tested the ConvNet with applying the non-linear ReLU-activation after the first Conv-Layer, the first batchNorm and the second Conv-Layer. Our hypothesis is that the network performs worse, because two linear layers without a non-linear activation in between act like a bigger single linear layer. Also the normalization layer in between has no large influence.
Before: Val loss: 1.0371, acc: 0.8015
After: Val loss: 1.0792, acc: 0.7825
This somehow confirms our hypothesis, but we expected the performance decrease to be bigger. We assume, that the two Conv layers without a nonlinear part in between limit the models ability to learn more complex pattern in the data, but since they form a bigger linear layer than before, they are still able to learn some of the patterns in the data.
BasicBlockv3:
We moved the second ReLU activation before the residual connection. Our hypothesis is, that we move the ReLU function behind the second normalization therefore our ReLU input is normalized, unlike before when applying a residual connection to the normalized output with the initial input data and then handing it to the second ReLU.
Before: Val loss: 1.0371, acc: 0.8015
After: Val loss: 1.0226, acc: 0.8089
The results show minor improvements for our Blockv3 compared to the original Block. We assume that the residual connection already provides enough stabilization and that the position of the activation is not that important.
2. b) ii. Analyse the results: Discuss and explain how the metrics for ViT compare to those you obtained for ConvNet. Are they better, worse, or similar? Why?

ConvNet: Val loss: 1.0371, acc: 0.8015
ViT: Val loss: 1.6160, acc: 0.5153
As the results show, the ViT perform significantly worse than the ConvNet. This is because ViTs need larger datasets and longer

training to achieve a good performance. With a larger training set, the performance of the ViT would likely improve significantly. ConvNets are known to perform well even with limited amounts of data, which is the case here.

2. c) ii. Write about your observations, drawing on your intuition and hypothesizing to explain the results.

ViT Blockv2

In ViT, pre-normalization is used. Pre-norm is more stable than post-norm when training deep Transformer models, as it improves gradient flow by applying normalization before each sublayer. This helps prevent vanishing or exploding gradients. Post-norm could perform the same way but only in small networks, so we suggest that post-norm will be worst or comparable.

Before: acc: 0.4963
After: acc: 0.4889

The performance is worst but comparable. We assume the difference will become bigger when using a larger network

ViT Blockv3

Our hypothesis is that by applying the residual connection in one step, the sub-modules can interact more closely with the data, rather than operating in sequence in behind. This could lead to improved performance.

Before: acc: 0.4963
After: acc: 0.5026

The performance is indeed slightly better than the old one. To finally confirm our hypothesis more tests are needed.

3. a) List all the hyperparameters of the ViT training setting that you expect would improve the validation accuracy. Explain why you believe each of these changes would help classification performance.

Feasible hyperparameter changes:

- increase p and attn_p: helps against overfitting. Might increase the validation accuracy if the model has a problem with overfitting.
- decrease patch_size: smaller but more patches might help the Model to learn more details.
- Increase embed_dim: model is able to learn more per patch, but training is more complex
- increase num_heads: model is able to learn different things at the same time
- increase depth: More blocks might help to learn more complex pattern. Training will take longer

Smaller patches -> more patches for the same image input size, more information that can be extracted and shared among each embedded patch, also we end up with more transformer encoders. But it comes with the cost of a bigger model needing more computation.
Dropout rates -> mitigate overfitting to trainingdata

3. c) Briefly write about the hyperparameter change(s) that you implemented, and how they changed accuracy.

Without changes: Val loss: 1.6160, acc: 0.5153

- 1. Test: p & attn_p from 0.0 to 0.2:
  - Val loss: 1.6528, acc: 0.5026
  - performed worse
  - model has no problem with overfitting
- 2. Test: patch_size from 32 $\rightarrow$ 16:
  - Val loss: 1.4731, acc: 0.5681
  - decreasing patch_size increased accuracy
- 3. Test embed_dim from 128 to 180:
  - Val loss: 1.6613, acc: 0.5090
  - no significant accuracy changes
- 4. Test depth from 16 to 24:
  - Val loss: 1.6233, acc: 0.5121
  - no significant accuracy changes
- 5. Test num_heads from 4 to 8:
  - Val loss: 1.6355, acc: 0.5143
  - no significant accuracy changes