Amit Das

Prof. Granger

Intro to Comp. Neuroscience

November 17, 2024

Intro to Comp. Neuroscience Programming Assignment

In this programming assignment, I used a Vision Transformer (ViT) to classify $224x224$ pixel images into their appropriate sports categories. To build the SportsViT, I used PyTorch and TorchVision. The sports dataset I used from Kaggle can be accessed here. This dataset consists of a training, validation, and testing set with images from 100 different sports categories. The model weights I used, which were taken from the *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* paper can be accessed here.

ViT Architecture

Below is the architecture of the SportsViT, which has 12 layers:

```
VisionTransformer(
  ├── conv_proj: Conv2d(in_channels=3, out_channels=768, kernel_size=(16, 16), stride=(16, 16))
  └── encoder: Encoder(
      ├── dropout: Dropout(p=0.0)
      ├── layers: Sequential(
          ├── encoder_layer_{i}: EncoderBlock(
              ├── ln_1: LayerNorm(normalized_shape=(768,), eps=1e-06)
              ├── self_attention: MultiheadAttention(
                  └── out_proj: Linear(in_features=768, out_features=768, bias=True)
              )
              ├── dropout: Dropout(p=0.0)
              ├── ln_2: LayerNorm(normalized_shape=(768,), eps=1e-06)
              └── mlp: MLPBlock(
                  ├── Linear(in_features=768, out_features=3072, bias=True)
                  ├── GELU(approximate='none')
                  ├── Dropout(p=0.0)
                  ├── Linear(in_features=3072, out_features=768, bias=True)
                  └── Dropout(p=0.0)
              )
          )
      )
      └── ln: LayerNorm(normalized_shape=(768,), eps=1e-06)
  )
  └── heads: Sequential(
      └── head: Linear(in_features=768, out_features=100, bias=True)
  )
)
```

conv_proj layer: The layer's main purpose is to break the input image into a sequence of patches. Since the stride is the same as the kernel size, these patches are non-overlapping. For an input image of size $224x224$, the image is represented by 196 patches. Each patch is encoded by a 768-dimensional vector.

Encoder: The encoder consists of multiple layers.The first layer normalizes the input. The self-attention layer computes standard self-attention using Key, Query, and Value vectors, which are all the same vectors. The SportsViT model has 12 heads and each head could attend to different parts of the image. After this attention layer there is another normalization layer followed by a MLP block. The MLP consists of a linear layer and an activation function, followed by another linear layer.

Model Head: This is a classification head, which I adjusted to take as input a 768th dimensional vector and output a 100th dimensional vector.

Implementation of SportsViT

The implementation of SportsViT is in the file `SportsViT.py`. The class that implements the ViT is the `SportsClassificationViT` class. A lot of the functionality comes from the torch.nn module. Within the constructor of this class, I define several model attributes including the model itself. I replaced the original model's classification head so that the model can work with my dataset. I also changed the forward layer of the encoder layers to store the attention weights.

Resources that I used

For implementation, I used the following resources:

- Model Architecture and Weights: I used the vit_b_16 architecture. The model architecture can be found here: https://pytorch.org/vision/main/models/generated/torchvision.models.vit_b_16.html
- Monkey-patching: I needed to replace the forward method of the encoder block and used the following resource to understand how to do that: https://discuss.pytorch.org/t/monkey-patching-the-forward-pass-of-an-nn-module/176095
- Training the ViT: I used this tutorial to understand how training and validation of a ViT works: https://medium.com/@brianpulfer/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c. I mainly used most of the logic for the training and validation loop, but added some more customizations like saving the best checkpoint during training.
- Visualizing Positional Embeddings: I used this tutorial to understanding how to visualize positional embeddings in a ViT and their importance: https://medium.com/@mandalsouvik/transformer-positional-embeddings-f73fee304900
- I used the standard augmentations and transformations from PyTorch. These can be accessed here: https://pytorch.org/vision/main/transforms.html

For understanding of concepts:

- Attention Weights: To gain a better understanding of self-attention, I found this video helpful: https://www.youtube.com/watch?v=eMlx5fFNoYc

I used prior knowledge of NumPy, Matplotlib, and PyTorch for visualizing attention weights and setting up data loaders to make them compatible with my dataset.

Dataset Description

The dataset is stored in the following format:

```
sports_dataset
  ├── train
    ├── golf
    ├── hockey
  ...
    ├── rugby
```

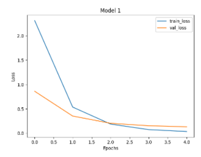
The subcategory represents the splits: train, validation, test. The subcategories within the splits represent the class names: golf, hockey, rugby, etc. Within these folders are images related to those sports. This is a classification dataset for classifying images into the sports they represent.

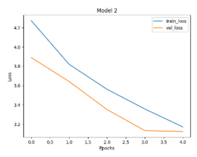
Results

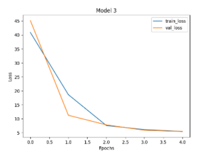
I trained the following models:

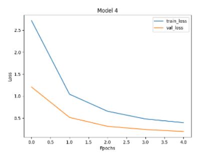
- Model 1: No augmentation during training, LR of $1 \cdot 10^{-5}$, batch size of 16
- Model 2: Augmentations enabled during training, LR of $1 \cdot 10^{-3}$, batch size of 16
- Model 3: Augmentations enabled during training, LR of $1 \cdot 10^{-1}$, batch size of 16
- Model 4: Augmentations enabled during training, LR of $1 \cdot 10^{-5}$, batch size of 16
- Model 5: Augmentations enabled during training, LR of $1 \cdot 10^{-5}$, batch size of 64

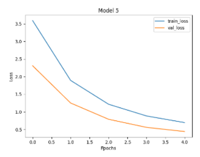
I chose these model configurations to see the effect of augmentations during training, learning rate, and batch size. During training, I observed the following loss curves.











From these plots, training and validation loss are both decreasing; however, the magnitude of the losses differ. For example, Model 1's loss converges below 1.0, but Model 3's loss converges around 5.0.

I trained each of the different configurations for 5 epochs and saved the best checkpoint based on the performance on the validation set during training. I observed the following results on the balanced test set.

Model	***F1 Score***
Model 1	0.9898
Model 2	0.2089
Model 3	0.00020
Model 4	0.96979
Model 5	0.94964

Model 1 has the best performance. As the learning rate decreases, we see better performance, which makes sense since we are using pre-trained weights. A high learning rate may lead to catastrophic forgetting. Interestingly, adding data augmentations did not lead to the model that performed the best on the testing set.

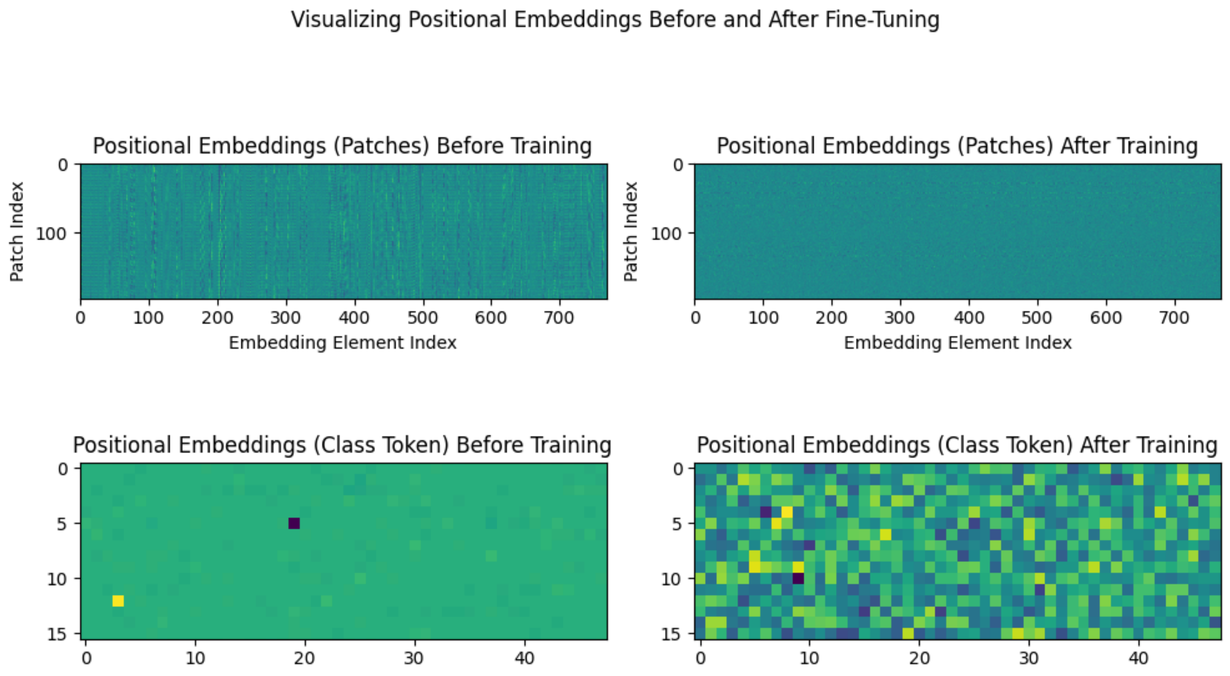
What conclusions can I draw?

Most of the code I added to the codebase was dealing with visualizations. Specifically, I visualized the positional embeddings of the tokens, attention maps across layers, attention maps of the last layer across heads, and aggregated attention maps. Here are the results I obtained from those experiments.

I. Positional Embeddings

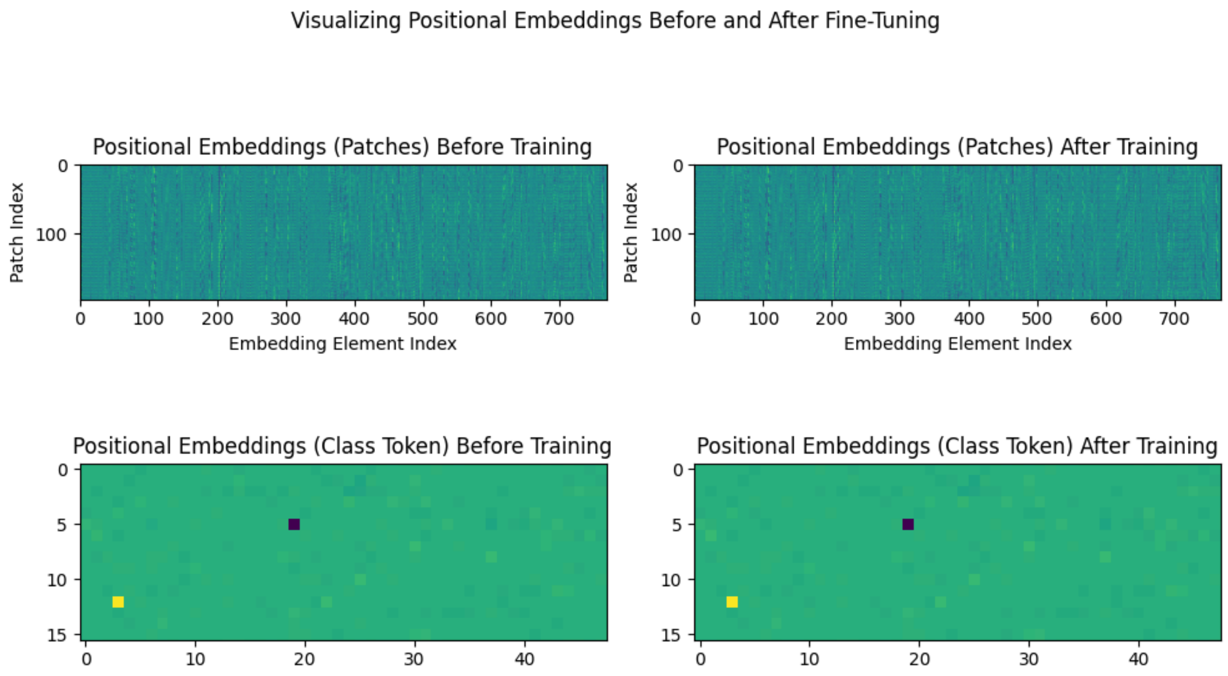
The ViT sees each image as a grid of $14x14$ or 196 patches. At the beginning of this sequence, a class token (CLS) is added to the sequence. The role of this token is to aggregate global information across the sequence. Each image is represented by 197 tokens. Furthermore, the position of these tokens, or patches, are important. A patch from the center of the image may be more important than a patch from the center of the image. I was interested in seeing how positional embeddings of tokens changed after fine-tuning the classification model.

A. Visualizing change in Position Embeddings for Model 3



The above subplot shows the effect of fine-tuning using Model 3. This model results in suboptimal performance, due to its high learning rate. When we visualize a heatmap of the positional embeddings before training, there is quite a bit of variation. This indicates that different locations of the patches are translated to different embeddings. However, after fine-tuning, the heatplot appears to be uniform. This indicates that the positional context within the image is lost, meaning a patch being in the center of the image is no different than a patch being from the corners. We also see that the positional embeddings of the class token drastically change after fine-tuning.

B. Visualizing change in Position Embeddings for Model 1

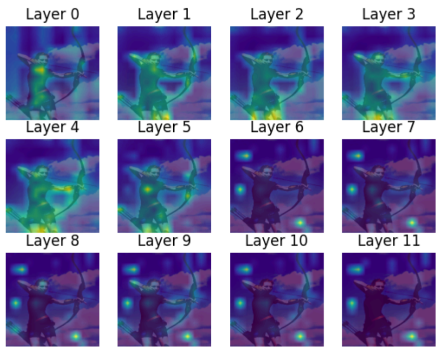


We visualize the position embeddings after fine-tuning with the best performing model, Model 1. Now, there are no major differences before training and after fine-tuning. This is because Model 1 uses a very small learning rate, unlike the model above which led to bad performance. With this model, positional information is still retained, likely leading to its great performance.

II. Visualizing Attention

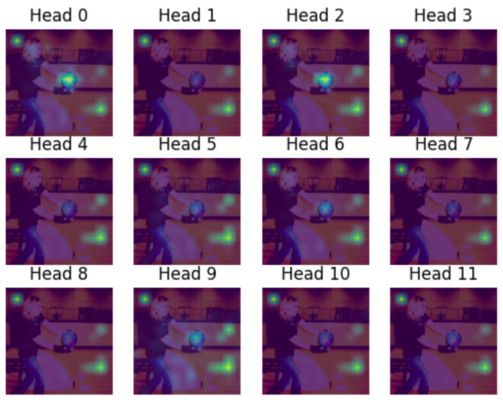
As I discussed earlier, ViTs represent the images as a sequence and the first token in this sequence is the CLS token. The CLS token aggregates information from the other tokens and the class prediction is based on this token. Thus, the CLS token is the most important token in the sequence and visualizing the attention weights of that token can lead to insights in how well the model is performing. To extract the attention weights, I made modifications in the original architecture of SportsViT. Specifically, I used monkey patching to modify the forward layer of the layers within the encoder block. This modification included retrieving the attention weights that led to the output of the attention layer and adding them to a global variable, which I could access after evaluation. I ensured that the forward layer did not change the actual performance of the ViT. Below I describe my results.

A. Visualizing Attention Across Layers



This is a visualization I obtained by analyzing the attention weights from my model across different layers. This sample is not cherry-picked and is a common trend I see across the images in the test set. From this image, one can see that the first 6 layers segment the image. In this example, the archer is fully segmented from the background. The later layers do not seem to add too much valuable information, possibly indicating that a ViT with 6 layers is sufficient for the task. However, one can clearly see why this image could be classified as "Archery" as the model is able to learn and detect the main object from the background.

B. Looking at Attention through different heads



In this image, I visualized the last layer, but created a subplot of the different heads. This image is "cherry picked" as we have seen from the last example, the last 6 layers do not seem to be adding much value. However, this image makes it clear that different heads focus on different things. For example, heads 0, 2, and 9 all seem to put attention on the bowling ball, while the other heads focus more on the background.

Summary/Additional Information

In this document, I have only shown attention map visualizations from a few images. However, I generated these visualizations across the entire testing set (500 images) and these can be found in the provided GitHub link.

In this class, we have seen examples of models where interpreting them can be challenging. One can get a sense of how neurons at the later layers are affecting the output, but connecting those neurons all the way back to the inputs is perplexing. Furthermore, within the models, the concept of which input did what ("positionality") seems to get lost. All of the models we looked at so far were non-topographical, so it was unsure what part of the input was correlated with the output. In fact, this seems to be how the brain works in some circumstances.

The ViT is interesting in that it treats specific parts of the image as tokens and the tokens add up to form the larger image. By doing this, we can clearly see which patch attends to the CLS token the most and interpretability of our model becomes much clearer. From the visualizations generated, I am confident that the method of using attention scores for interpreting and understanding ViTs work well.