Advanced Deep Learning
Winter 2023/2024
Martin Mayr, Vincent Christlein, Katharina Breininger

Sheet 1, starting from October 25th, 2023, due November 15th, 2023, 14:00

Exercise 1.1: Basic structure of a PyTorch project

We will give you the main structure of a PyTorch project for image classification. This will also be the basis for the following two tasks in this exercise. **Your very first task is therefore to make yourself familiar with the given code (`ex1_main.py` and `my_models_skeleton.py`) to be able to implement all subsequent tasks efficiently and successfully.**

Note: Please refrain from using e.g. pytorch-lightning or fast-ai but stay in native pytorch/torchvision for this exercise as we cannot guarantee support for additional frameworks during the exercises. We provide a python environment with the corresponding packages installed on the cip-pool machines. For more information, please see below.

In detail the main code structure encapsulates the following steps:

1. Loading CIFAR10 dataset
2. Creating a ResNet18 model architecture using torchvision w.o. pre-trained weights.
3. Defining loss function and optimizer w. cross-entropy as the loss function and SGD as optimizer. Default learning rate and momentum can be used to get started.
4. Training loop that trains and validates the model in each epoch.
5. Evaluating the model on the CIFAR10 test set and reporting the classification accuracy.

Your task: Improve the main structure with **(a) adding augmentation**. Currently no augmentation is applied. To improve results apply standard data augmentation techniques such as random crop, random horizontal flip, and normalization using PyTorch's transforms. **(b) Save the model with the highest validation accuracy.**

Exercise 1.2: VisionTransformer (ViT): Self-attention for image classification

The key idea behind the Vision Transformer[1] is to apply self-attention mechanisms to image embeddings. The image is first divided into a set of non-overlapping patches, which are flattened and linearly transformed to produce token embeddings. These token embeddings are then used as input to a standard Transformer architecture, which consists of multiple layers of self-attention and feedforward networks.

The self-attention mechanism in the Vision Transformer allows the model to attend to different parts of the image and capture long-range dependencies between them. This is done by computing an attention score between each pair of token embeddings, and using these scores to weight the importance of each token in the final image representation. The feedforward networks in the Transformer architecture then process the attended token embeddings to produce the final output logits.

Note for everyone looking deeper into the topic: In the vanilla Vision Transformer the class token is

[1] `https://arxiv.org/abs/2010.11929`

used for the extracting the output logits. By contrast, in a newer version there is no class token needed but the mean of all output tokens is used as output logits.

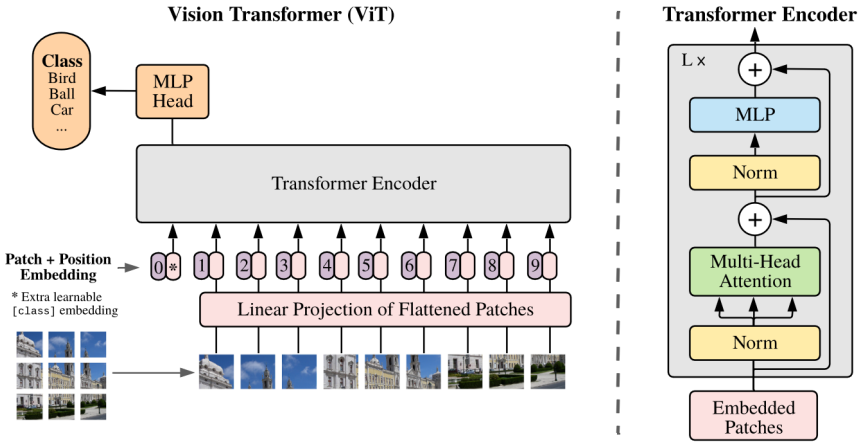

Abbildung 1: Vision Transformer approach.

Your task: The task here is replace the ResNet18 from the previous task with a Vision Transformer (ViT) model. The given ViT class in `my_model_skeleton.py` is nearly complete but lacks the important attention mechanism. Your task is to implement this: Have a look at the code skeleton and follow the instructions there.

Exercise 1.3: Cross-Attention Multi-Scale ViT: Adding cross-attention

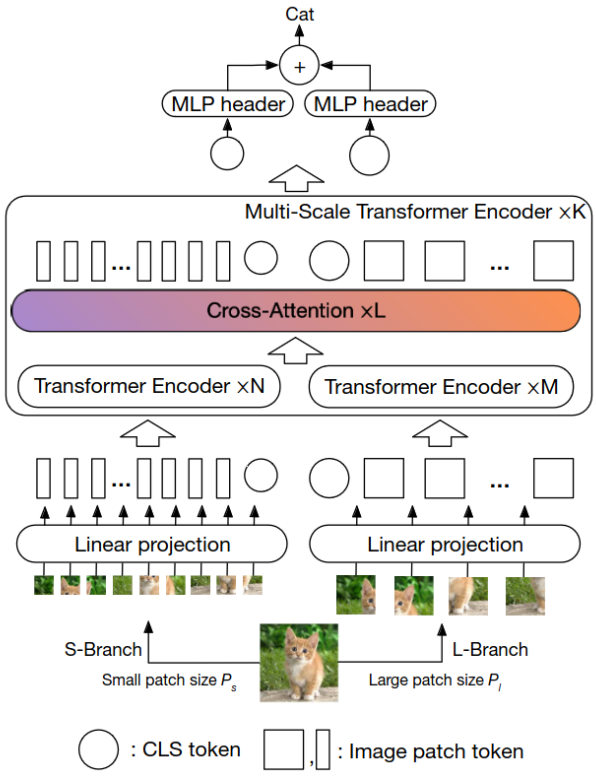

Abbildung 2: CrossViT approach.

CrossViT is motivated by the observation that different parts of an image can have complex interactions and dependencies that are difficult to capture with self-attention alone. Cross-attention allows the model to attend to different scales of the image and the token embeddings simultaneously, and to learn complex interactions between them.

It starts by projecting small and large patches similar to the standard ViT. Both branches are then fed into the multi-scale transformer encoder. In this encoder the two different projections are fed into N/M transformer blocks. Afterwards the cls token of the small patch embeddings is cross-attended to the embeddings of the large embeddings and vice-versa. Finally, both embeddings go into individual MLP heads whose logits are added.

Your task: Implement the CrossViT architecture using PyTorch, following the guidelines in the paper "CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification"[2]. Train and evaluate the model in the already existing pipeline.

We provide you with a skeleton of the Cross-ViT, please feel free to adjust it to your needs[3].

Exercise 1.4: Optional: Visualize the decision making of ViT

Extract the attention weights of the last self-attention block between the class token and the other tokens. Check which patch has the biggest impact on the class token.

Additional information: Using a prepared conda environment

Since the quota on the cip-pool machines is limited and we may need additional packages during the semester, we have prepared a conda environment `adl23_2` that you can use for this exercise. To use this environment, you can do the following:

Option 1: Activate the environment directly:

```
conda activate /proj/aimi-adl/envs/adl23_2
```

Option 2: Adapt your conda config: Check if you have a file `.condarc` in your home directory (`/.condarc`). If not, run

```
conda config
```

If you have it or after running above command, open the file in your favorite editor and add the following line(s) to the file (amend if you already have the `envs_dirs` option):

```
envs_dirs:
- /proj/aimi-adl/envs/
```

You should then be able to activate the environment using

```
conda activate adl23_2
```

Let us know if you encounter any issues!

Hint: You might want to run your training in the background using **tmux(1)**. Don't forget to assign a low priority to your task using **nice(1)**. This prevents the removal of your process in case a local user allocates the GPU.

[2] `https://arxiv.org/abs/2103.14899`
[3] Terms and conditions (see above) apply ;)