Computer Vision - Spring 25

Assignment 5

Deadline : 21 April 2025 11:55 P.M.

Instructors: Prof Ravi Kiran Sarvadevabhatla
and Prof Makarand Tapaswi

April 9, 2025

General Instructions

- Your assignment must be implemented in Python.
- While you're allowed to use LLM services for assistance, you must explicitly declare in comments the prompts you used and indicate which parts of the code were generated with the help of LLM services. **If you haven't specified the prompts, then 50% of your marks will be deducted.**
- Plagiarism will only be taken into consideration for code that is not generated by LLM services. Any code generated with the assistance of LLM services should be considered as a resource, similar to using a textbook or online tutorial.
- The difficulty of your viva or assessment will be determined by the percentage of code in your assignment that is not attributed to LLM services. If during the viva you are unable to explain any part of the code, that code will be considered as plagiarized.
- Clearly label and organize your code, including comments that explain the purpose of each section and key steps in your implementation.
- Properly document your code and include explanations for any non-trivial algorithms or techniques you employ.
- Ensure that your files are well-structured, with headings, subheadings, and explanations as necessary. Your assignment will be evaluated not only based on correctness but also on the quality of code, the clarity of explanations, and the extent to which you've understood and applied the concepts covered in the course.
- Make sure to test your code thoroughly before submission to avoid any runtime errors or unexpected behavior.

- **Report all the analysis, comparison and any metrics in the notebook or a separate report that is part of the submission itself. No external links to cloud storage files, wandb logs or any other alternate will be accepted as part of your submission. Only the values and visualizations as part of your commits will be graded.**
- Save your visualizations in a separate folder, or display them in the notebook in a lower resolution, in case size of the file is huge.

1 Vision Transformer [50 Marks]

In this task, you will implement and train the Vision Transformer (ViT) from scratch on the CIFAR-10 dataset.
Reference paper: https://arxiv.org/abs/2010.11929

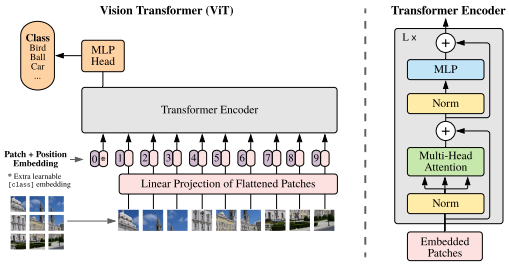

Figure 1: ViT architecture

1.1 Tasks

- Implement scaled dot product attention and multi-headed attention blocks as described in the ViT paper.
- Use these, and incorporate them in the transformer encoder layer.
- Add a [CLS] token. The [CLS] token is a special learnable embedding prepended to the input sequence. It is intended to aggregate global information from the entire input.
- Add learnable positional embedding as implemented in the paper.
- You can use patch size = 4 for your initial experiments.
- Train the ViT model on CIFAR-10 (Do not resize), plot the loss curves, and report the test accuracy.

1.2 Experiments

1. **Patch Size Variation:**
   - Train and evaluate the model with patch sizes of 2, 4 (default), and 8.
   - Plot the loss curves and report the test accuracies.
2. **Hyperparameter exploration:**
   - Experiment with hyperparameters such as embedding dimension, number of transformer layers, MLP hidden dimension, and number of attention heads etc.
   - Identify the best hyperparameter combination that achieves more than 80% test accuracy within 50 epochs.

- Report your best configuration and discuss your findings.

3. **Data Augmentations:**

   - Explore the effect of different data augmentation techniques on model performance. Report the best test accuracy achieved and specify the augmentation techniques (and combinations) used. You can try three different augmentation techniques or a combination thereof to improve the performance further.

1.3 Positional Embeddings

Using the best hyperparameters identified, implement the following types of positional embeddings (refer to **Appendix D.4** in the ViT paper):

- No positional embedding
- 1D learned (default and used in previous experiments)
- 2D learned positional embedding (Refer point 3 of Appendix D.4)
- Classic sinusoidal positional embedding from the original Transformer paper (*Attention Is All You Need*).

Report and compare the loss curves and test accuracy for each positional embedding.

1.4 Visualizations

1. **DINO Attention Maps:**

   - Visualising the attention map from [CLS] token to patch tokens was popularized in DINO. This approach highlighted ViT's capability to capture the semantic layout of an image through their attention maps. In this section, you will be running the DINO's code to visualize the attention map of a pre-trained ViT given by DINO.
   - Use the following implementation here and run `visualize_attention.py` using any two images of your own choosing and visualize the attention maps

2. **ViT CIFAR-10 Attention Maps:**

   - Visualize the attention maps of the ViT model trained on CIFAR-10 in the previous experiments.
   - Specifically, visualize the attention from the [CLS] token to all patch tokens for each attention head of the last attention layer, as well as the aggregated attention map (e.g., by averaging across heads). You can use images from the CIFAR-10 test set for this task.
   - Additionally, visualize the attention from [CLS] to all patch tokens for all the attention layers in the model.

3. **Attention Rollout:**

   - To compute the attention maps from the output token to the input space, ViT paper used attention rollout. Refer to **Appendix D.8** of the ViT paper for details.
   - Attention Rollout was introduced in the paper *Quantifying Attention Flow in Transformers*. You can refer to Section 3 (i.e. the attention rollout sections only) of the paper below.

     Reference Paper: https://arxiv.org/abs/2005.00928
   - In this task, you have to implement the attention rollout as implemented in the ViT paper and visualize the attention maps. You can use images from the CIFAR-10 test set for visualization.

4. **Positional Embedding Visualization:**

   - Use the trained ViT with default learned 1D positional embeddings from the initial task. Visualize the similarities between the learned positional embedding with itself by taking their dot-product.

2 Differential Vision Tranformer [50 Marks]

In this part, you will extend the Vision Transformer (ViT) architecture by incorporating the Differential Attention Mechanism as introduced in the Differential Transformer paper. Differential attention was originally propsed for large language models to amplify context and cancel out noisy information. The idea behind the differential attention mechanism is to eliminate attention noise and encourage the model to focus on critical information. The paper compares their idea with noise-canceling headphones, where the difference between two signals cancels out common-mode noise.
You will adapt this mechanism to the vision domain, replacing standard multi-headed attention in the ViT with Multi-Head Differential Attention. The rest of the ViT architecture—including patch embedding, class token, and positional encoding—should remain consistent with the original ViT implementation.
Reference Paper: https://arxiv.org/abs/2410.05258

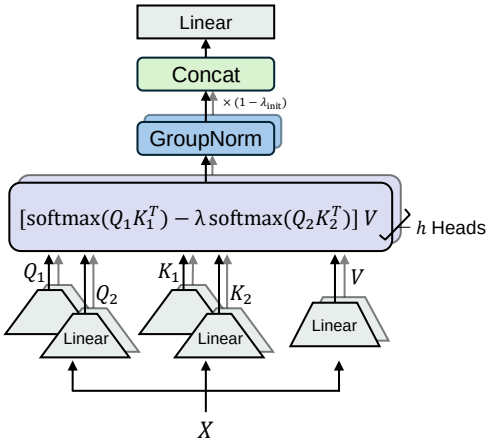

Figure 2: Multi-head Differential Attention Mechanism.

2.1 Tasks

1. **Differential Attention Implementation:**
   - Implement the differential attention mechanism and the multi-headed differential attention modules in the paper.
2. **Integrate Differential Attention into ViT:**
   - Replace the Multi-Head Attention module in the Transformer encoder block with your Multi-Head Differential Attention.
   - Retain other ViT components, such as CLS token, MLP block, residual connections, and learnable positional embeddings.
3. **Training:**
   - Use the CIFAR-10 dataset with a patch size of 2/4 for experiments.
   - Train your Diff-ViT from scratch and plot loss curves.
   - Report test accuracy and compare with the standard ViT results.

2.2 Experiments

Repeat the experiments conducted with the vanilla ViT (Ques 1) using your Differential ViT (Diff-ViT). Report and compare results for:

- Hyperparameters
- Data augmentation
- Positional embedding types

2.3 Visualization

- Visualize the attention from the [CLS] token to all patch tokens, as done previously, but now using Diff-ViT.
- Perform attention map visualization using attention rollout, following the same procedure as in question 1.

3 CLIP [30 Marks]

3.1 Tasks

1. **Inference using CLIP.** Load the ResNet-50 (RN50) model, initialized in two different ways:
   - ImageNet pretraining (`torchvision.models` can be used, specifically look at `IMAGENET1K_V1`); and
   - OpenAI's CLIP (see link).

   Do the visual encoders have the same architecture? If not, please describe and explain the differences.

   Hint: When you load the CLIP model, you will get both the vision and text encoders - be sure to differentiate between them as necessary.

2. **Setup data.** Understand the ImageNet challenge dataset (1000 labels of ILSVRC).
   - What label hierarchy is used in ImageNet?
   - What does a synset mean?
   - Could grouping objects based on synsets lead to problems for visual recognition?
   - State 3 types of visual differences we can expect to see in images with objects corresponding to the same synset.

3. **Setup zero-shot CLIP.** Similar to the ImageNet pretrained RN50, set up CLIP to generate probability scores for the 1000 ImageNet categories. Test it with a few example images to check that it identifies the correct object category.

   Hint: You may treat the cosine similarities as "logits".

4. **CLIP vs ImageNet pretraining.** Pick 10 classes from ImageNet (not all from the same branch, e.g., not all dogs). For each class:
   - Find 2 images that work well with CLIP, but not with ImageNet pretrained RN50. Reason about why this may be the case. From where did you get these images?

- Find 1 image that works well with ImageNet pretraining but not CLIP. Reason about why this may be the case. From where did you get these images?

  Note: For the purpose of this question, we will say that the model "works well" if it generates the correct category label within the top-5 highest scoring labels.

  Hint: Reading the CLIP paper (`https://arxiv.org/abs/2103.00020`) can help you solve this question quickly as it shows examples where ImageNet does not work, but CLIP works.

5. **FP16.** While deep learning has primarily relied on "single" floating precision (32 bits or 4 bytes per parameter), 16-bit floating point numbers are useful for saving on memory. In this question, we will only consider CLIP's image encoder.

   - Convert the RN50 CLIP image encoder model to fp16. Calculate the (wall-clock) time required to encode an image. Estimate the difference between the fp32 (original) model and the fp16 model. Note, it is common to report the mean over 100 runs and indicate both the mean and standard deviation.
   - For 5 images (1 per class), recalculate the probabilities using the fp16 model. Are there significant differences between the fp32 and fp16 outputs? Why?
   - Using `nvidia-smi` or a profiler, note and explain the differences in memory usage for a forward pass between fp32 and fp16 models. If you do not use the profiler, please include some screenshots for the assignment.

     Hint: https://pytorch.org/blog/understanding-gpu-memory-1/ Categorized Memory Usage may help.

Good luck with the assignment!