Project 1

It's all about the woof

1 Description

It is normal practice in deep learning to adapt large-scale models pre-trained on general domain data to solve particular down-stream tasks through a process called *fine-tuning*. In full fine-tuning all model parameters are updated by training the whole model on new domain data. With increasingly large state-of-the-art models, this becomes very expensive in terms of storage and compute requirements.

Low-Rank Adaptation (LoRA) [1] is a simple yet effective adaptation technique we can use for fine-tuning pre-trained models without updating all the model parameters directly. This is especially useful for fine-tuning large state-of-the-art models for down-stream applications, since it greatly reduces the compute and memory costs.

In this project, you will fine-tune a pre-trained Vision Transformer (ViT) [2] by using both full fine-tuning and LoRA, and write a report about it. The data you will fine-tuning on is ImageWoof, which is a subset of ImageNet containing images of 10 different dog breeds. The down-stream task is classification of these 10 breeds.

2 Low-Rank Adaption (LoRA)

LoRA is an adaptation technique that indirectly trains dense layers in a neural network by keeping the original pre-trained weights frozen and instead optimizing rank decomposition matrices of the change in weights during the adaptation/fine-tuning. More specifically, the adaptation of a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$ can be written as $W_0 + \Delta W$ where ΔW is the update in the weights that we want to learn. LoRA constrains the update by representing it with a low-rank decomposition $\Delta W = BA$, where $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$ and the rank is $r \ll \min(d,k)$. The output of the LoRA-adapted model given input x is then

$$h = (W_0 + \Delta W)x = W_0 x + BAx. \tag{1}$$

This setup is illustrated in Figure 1.

To apply LoRA to the dense layers of a pre-trained model, you will need to write a wrapper class. This is what you did (or will do) in exercise 3 of the notebook for the lab about ViTs! We, therefore, suggest that you complete the notebook as a warm-up exercise to this project.

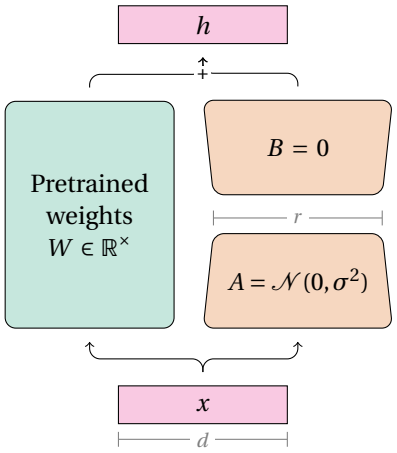

Figure 1: LoRA representation of each dense layer [1].

3 Pre-trained ViT

We will use the `timm` library to get a pre-trained ViT. You can read the `timm` documentation here. The ViT we will use is `'vit_tiny_patch16_224'`, which is an instance of ViT-Tiny pre-trained on ImageNet-21k and fine-tuned on ImageNet2012. It expects input size 224x224 and uses 16×16 patches. You can see how to load this model in the notebook about ViTs.

NB: When fine-tuning a ViT for classification on new data, the old classification head is typically discarded and replaced with a new one (with the correct number of categories) to be trained in the fine-tuning process. This can be done automatically for you in the `timm.create_model` function by specifying the number of categories with the `num_classes` argument.

4 IN9310 students only: visualize the final attention map of the ViT

⚠ All groups with one or more IN9310 students have to complete this extra task. Groups consisting of only IN5310 students may skip this.

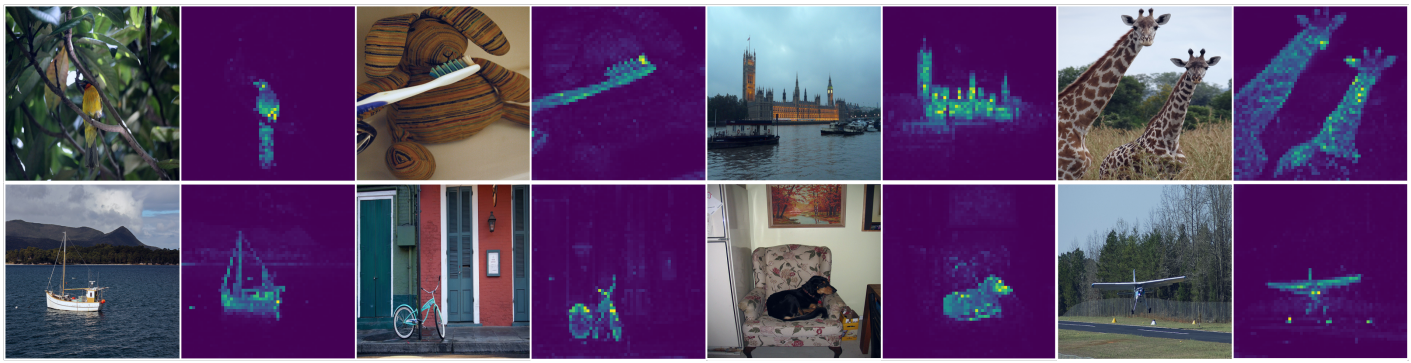

Figure 2: Attention maps from DINO [3].

Visualizing the ViT attention maps is a common way of gaining insight into what the ViT has learned. The attention weights between the class token and the patch tokens shows us which part of the input image contain more important information for the classification task. For this task, you need to write a function that given the model (either the ViT or the ViT+LoRA) and an input image gives you the attention masks for the ground truth class in that image. For instance, you can refer to DINO's maps [3], show in Fig. 2, for an example of what the attention maps will look like.

5 Experiments

Your task is to conduct and report on the following experiments:

1. Fine-tune the ViT on ImageWoof by training the full model (a.k.a. full fine-finetuning). Five epochs should be enough to get good results. Report the model accuracy and training time.
2. Fine-tune the ViT on ImageWoof by using your LoRA wrapper. Report on the accuracy and training time, and compare it with the model trained with full fine-tuning.
3. IN9310 only: Visualize the final attention maps of the ViT. How does the attention changes from the original ViT to the ViT+LoRA model?

Details about how you should organize your report are given in § 6. In the discussion in your report, we want you to reflect on the following in particular:

1. Which fine-tuning method (LoRA vs. full) gave the best results for classification on ImageWoof? Why do you think that is?
2. In which situations do you think LoRA is useful?

6 Project Report

Your final report should be written like a workshop/conference paper. You can take a look at different papers from major conferences (CVPR, ICCV, ECCV, NeurIPS, ICML, or ICLR) to get a general sense of how the papers look and are organized. A good (and relevant!) example is the paper that introduced ViTs [2]. You can use the LaTeX template provided in the example repo (see the `report` branch) as a starting point.

In general, we suggest that you have the following sections in your report:

- Abstract
- Introduction
- Related work
- Method
- Experiments and results
- Discussion

- Conclusion
- References

We restrict the report to have a maximum of 9 pages, excluding the references. However, since it is the first project, our recommended length for this project is 5 pages (more or less). We also encourage you to use figures and tables to present your methods and results. Remember to properly cite any figures you use that you haven't made yourself. If you have more specific questions of the report requirements and style, either refer to the instructions for ICLR 2023 or ask someone from the teaching team.

You must include a "Group Contributions" section (before the references) where you detail what each member of your group did. You could have a paragraph detailing what each member was responsible for and what tasks did they perform, or you could draw a contribution matrix detailing the main steps and responsibilities and report the percentage of involvement of each team member. This section *does not count towards your page limit.*

7 Evaluation

Your submission consists of the project report and the code in your group's GitHub at the deadline. **The deadline is Friday 08/09.** The grade is pass/fail, and will be defined by the following aspects:

1. Project report 80%
   (a) Introduction, motivation, and background 10%
   (b) Related work 10%
   (c) Explanation of your methodology 10%
   (d) Experiments, discussion, and results 50%
2. Code 20%

Your language usage won't be graded, but your ability to present your results, ideas, and how they are supported will be. Each other point will be evaluated according to the completeness and correctness of the requested items.

8 Submission

Your submission must be through your assigned group on Github. Use the repository named `g#-p1` for your group number and commit all the code and report there. You can start out by cloning this example repo, which has one `main` branch for pushing the code and a `report` branch for pushing the report files. For instance,

```
1 $ git clone --bare git@github.uio.no:2023-s2-in5310-in9310/example-p1.git
2 $ cd example-p1
3 $ git remote rm origin
4 $ git remote add origin git@github.uio.no:2023-s2-in5310-in9310/g#-p1.git
5 $ git push --force --all origin
```

will clone the `example-p1` repository into your own group. Note that in line 4 you need to replace `#` with your group number.

8.1 Code

Your submission must have the following subfolders in the `main` branch of your repository:

- `output`: a directory where your application should produce all the generated files (otherwise stated in the problem). This is where you should save your best fine-tuned models. We suggest saving them with the names `output/full_model.pt` and `output/lora_model.pt`, but it is up to you. This folder and all its contents must be added to the artifacts in case you setup a workflow for your code.
- `src`: a directory containing all your source code. You only need to submit files that are not derived from other files or through compilation. In case some processing is needed, prefer to submit a script that does that instead of submitting the files.

- `Makefile`: a makefile that executes your code through a docker image or that is setup in a way that construct its own environment for reproduction. You are encouraged to use the standard pytorch image, `pytorch/pytorch`, but other solutions are possible. ⚠ Docker is not installed on the Educloud servers. If you're using the Educloud servers to run your code, you can use the `in5310` virtual environment instead of docker to ensure that we can execute your code. More detailed instructions can be found here. Discuss with the teaching team early on your setup to ensure its reproduction. The code will be executed through a standard call to `make`, so other dependencies must be provided by you under that constraints. 🚀 Moreover, note that your code **must** run without additional prompt or changes from the user.

Additionally, you need to write a script `validate_project1.py` with a function `load_my_models()`. This should take no arguments and return two fine-tuned ViTs: your best model from fine-tuning with all weights, and your best model from fine-tuning with LoRA. The teaching team will import this script for testing your models when we evaluate your projects. Use the `validate_project1.py` example available in the example repo as a starting point. You only have to complete the `load_my_models()` function. The reason you have to write this script is to ensure that we load your models correctly!

8.2 Report

All files pertaining to the report should be uploaded in a separate `report` branch. You need to include the following:

- All the source files (e.g., `report.tex`) that produce your report must be here.
- 🚀 Your report **must** be written using LaTeX (and friends) and compiled within the workflow in this branch. ⚠ Consequently, **the PDF must not be committed**. You can use the images from `adnrv/texlive` to build your PDFs. **Only the PDF of the report** must be added to the `artifacts` path on the workflow setup, and not other intermediary files of your report (e.g., images, log files, auxiliary files). You can follow the `example` repository (or directly clone it into your own repository) to reuse the existing workflow.
- Your report must show all your work for the given project, including images (labeled appropriately, that is, following the convention given) and other outputs needed to explain and convey your work. **The constraints of this report are explained in § 6.**

🚀 The last commit before the submission deadline will be used to review your code and the report. Do not worry about executing times for building the report. However, **you must ensure that your code works as no attempt will be made to patch or run broken code**.

9 Notes

- ℹ Note that there are several implementations that you can find on the internet. This project is for **you to implement** the algorithms. Thus, **do not submit code from others**. And if you re-use code from someone for a non-restricted part, disclose it in your report and code.
- ℹ All the submissions must be self-contained and must be executable in a Linux environment. Your code could execute in the docker image `pytorch/pytorch`, available at docker hub (`https://hub.docker.com/r/pytorch/pytorch/`). If not, then you must ensure that your entry point sets up the environment needed to reproduce your work without further inputs from the user.
- ℹ It is your responsibility to make sure your code compiles and executes correctly. No effort will be made to run your code besides executing `make`.
- ℹ You must program in Python. If you need to install other packages you must do so within your `Makefile` as automatic prerequisites.

Please contact someone from the teaching team at a reasonable time before the deadline if you are having problems.

References

[1] Edward J. Hu et al. "LoRA: Low-Rank Adaptation of Large Language Models". In: *Inter. Conf. Learn. Represent. (ICLR)*. 2022.

[2] Alexey Dosovitskiy et al. "An Image is Worth 16 × 16 Words: Transformers for Image Recognition at Scale". In: *Inter. Conf. Learn. Represent. (ICLR)*. 2021.

[3] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: *IEEE Inter. Conf. Comput. Vis. (ICCV)*. 2021.