



Autosaved just now

Publish changes



DA6401 - Assignment 2

Learn how to use CNNs: train from scratch and finetune a pre-trained model as it is.

Akshay V me21b014

▼ Instructions

- The goal of this assignment is twofold: (i) train a CNN model from scratch and learn how to tune the hyperparameters and visualize filters (ii) finetune a pre-trained model just as you would do in many real-world applications
- Discussions with other students is encouraged.
- You must use `Python` for your implementation.
- You can use any and all packages from `PyTorch`, `Torchvision` or `PyTorch-Lightning`. NO OTHER DL library such as `TensorFlow` or `Keras` is allowed. Please confirm with the TAs before using any new external library. BTW, you may want to explore `PyTorch-Lightning` as it includes `fp16` mixed-precision training, `wandb` integration and many other black boxes eliminating the need for boiler-plate code. Also, do look out for `PyTorch2.0`.
- You can run the code in a jupyter notebook on colab by enabling GPUs.
- You have to generate the report in the format shown below using `wandb.ai`. You can start by cloning this report using the clone option above. Most of the plots that we have asked for below can be (automatically) generated using the APIs provided by `wandb.ai`

- You also need to provide a link to your GitHub code as shown below. Follow good software engineering practices and set up a GitHub repo for the project on Day 1. Please do not write all code on your local machine and push everything to GitHub on the last day. The commits in GitHub should reflect how the code has evolved during the course of the assignment.
- You have to check Moodle regularly for updates regarding the assignment.

▼ Problem Statement

In Part A and Part B of this assignment you will build and experiment with CNN based image classifiers using a subset of the [iNaturalist dataset](#).

▼ Part A: Training from scratch

▼ Question 1 (5 Marks)

Build a small CNN model consisting of 5 convolution layers. Each convolution layer would be followed by an activation and a max-pooling layer.

After 5 such conv-activation-maxpool blocks, you should have one dense layer followed by the output layer containing 10 neurons (1 for each of the 10 classes). The input layer should be compatible with the images in the [iNaturalist dataset](#) dataset.

The code should be flexible such that the number of filters, size of filters, and activation function of the convolution layers and dense layers can be changed. You should also be able to change the number of neurons in the dense layer.

- What is the total number of computations done by your network? (assume m filters in each layer of size $k \times k$ and n neurons in the dense layer)

- What is the total number of parameters in your network? (assume m filters in each layer of size $k \times k$ and n neurons in the dense layer)

▼ Question 2 (15 Marks)

You will now train your model using the [iNaturalist dataset](#). The zip file contains a train and a test folder. Set aside 20% of the training data, as validation data, for hyperparameter tuning. Make sure each class is equally represented in the validation data. Do not use the test data for hyperparameter tuning.

Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions but you are free to decide which hyperparameters you want to explore

- number of filters in each layer : 32, 64, ...
- activation function for the conv layers: ReLU, GELU, SiLU, Mish, ...
- filter organisation: same number of filters in all layers, doubling in each subsequent layer, halving in each subsequent layer, etc
- data augmentation: Yes, No
- batch normalisation: Yes, No
- dropout: 0.2, 0.3 (BTW, where will you add dropout? You should read up a bit on this)

Based on your sweep please paste the following plots which are automatically generated by wandb:

- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

Also, write down the hyperparameters and their values that you swept over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried.

▼ Question 3 (15 Marks)

Based on the above plots write down some insightful observations.

For example,

- adding more filters in the initial layers is better
- Using bigger filters in initial layers and smaller filters in latter layers is better
-

(Note: I don't know if any of the above statements is true. I just wrote some random comments that came to my mind)

▼ Question 4 (5 Marks)

You will now apply your best model on the test data (You shouldn't have used test data so far. All the above experiments should have been done using train and validation data only).

- Use the best model from your sweep and report the accuracy on the test set.
- Provide a 10×3 grid containing sample images from the test data and predictions made by your best model (more marks for presenting this grid creatively).

▼ Question 5 (10 Marks)

Paste a link to your github code for Part A

Example: https://github.com/<user-id>/da6401_assignment2/partA;

- We will check for coding style, clarity in using functions and a `README` file with clear instructions on training and evaluating the model (the 10 marks will be based on this).
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarised).
- We will also check if the training and test data has been split properly and randomly. You will get 0 marks on the assignment if we

find any cheating (e.g., adding test data to training data) to get higher accuracy.

▼ Part B : Fine-tuning a pre-trained model

▼ Question 1 (5 Marks)

In most DL applications, instead of training a model from scratch, you would use a model pre-trained on a similar/related task/dataset. From `torchvision`, you can load ANY ONE `model` (`GoogLeNet`, `InceptionV3`, `ResNet50`, `VGG`, `EfficientNetV2`, `VisionTransformer` etc.) pre-trained on the ImageNet dataset. Given that ImageNet also contains many animal images, it stands to reason that using a model pre-trained on ImageNet maybe helpful for this task.

You will load a pre-trained model and then fine-tune it using the naturalist data that you used in the previous question. Simply put, instead of randomly initialising the weights of a network you will use the weights resulting from training the model on the ImageNet data (`torchvision` directly provides these weights). Please answer the following questions:

- The dimensions of the images in your data may not be the same as that in the ImageNet data. How will you address this?
- ImageNet has 1000 classes and hence the last layer of the pre-trained model would have 1000 nodes. However, the naturalist dataset has only 10 classes. How will you address this?

(Note: This question is only to check the implementation. The subsequent questions will talk about how exactly you will do the fine-tuning.)

▼ Question 2 (5 Marks)

You will notice that `GoogLeNet`, `InceptionV3`, `ResNet50`, `VGG`,

`EfficientNetV2`, `VisionTransformer` are very huge models as compared to the simple model that you implemented in Part A. Even fine-tuning on a small training data may be very expensive. What is a common trick used to keep the training tractable (you will have to read up a bit on this)? Try different variants of this trick and fine-tune the model using the iNaturalist dataset. For example, '___'ing all layers except the last layer, '___'ing upto k layers and '___'ing the rest. Read up on pre-training and fine-tuning to understand what exactly these terms mean.

Write down the at least 3 different strategies that you tried (simple bullet points would be fine).

▼ Question 3 (10 Marks)

Now fine-tune the model using ANY ONE of the listed strategies that you discussed above. Based on these experiments write down some insightful inferences comparing training from scratch and fine-tuning a large pre-trained model.

▼ Question 4 (10 Marks)

Paste a link to your GitHub code for Part B

Example: https://github.com/<user-id>/da6401_assignment2/partB

Follow the same instructions as in Question 5 of Part A.

▼ Self Declaration

I, AkshayV_ME21B014 (Roll no: XXYY), swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.

