

# CS 470: Assignment 05

---

## Programming Assignments

The goal of this assignment is to write code to **train and evaluate PyTorch neural network models to predict classes from the CIFAR10 dataset.**

**You must train and evaluate at least TWO different variations.**

### A05.py

You will write the following functions in this python script:

- `get_approach_names()`
- `get_approach_description(approach_name)`
- `get_data_transform(approach_name, training)`
- `get_batch_size(approach_name)`
- `create_model(approach_name, class_cnt)`
- `train_model(approach_name, model, device, train_dataloader, test_dataloader)`

Note that most of these functions rely on the `approach_name` parameter. This is how you will code different behavior per approach.

#### `get_approach_names()`

Returns a list of the names of all combinations you will be testing. It is largely up to you what names you use, but self-documenting names are encouraged.

A rather boring example: [ "CNN0", "CNN1" ]

#### `get_approach_description(approach_name)`

Given the `approach_name`, return a text description of what makes this approach distinct. This does not have to be very long: one sentence is sufficient.

#### `get_data_transform(approach_name, training)`

Given the `approach_name` and whether this is for training data, return the appropriate dataset transform. You may add data augmentation if you like for training data transforms, BUT:

- Do NOT do data augmentation for non-training data!
- At minimum, you must perform the following (depending on your version of torchvision):

**(0.15)** `data_transform = v2.Compose([v2.ToImageTensor(), v2.ConvertImageDtype()])`

**(0.16)** `data_transform = v2.Compose([v2.ToImage(), v2.ToDtype(torch.float32, scale=True)])`

`get_batch_size(approach_name)`

Given the `approach_name`, return the preferred batch size. This can be hardcoded to a single value if you do not have a preference, BUT this allows you to use a smaller batch size if your architecture is large enough where that is a concern.

`create_model(approach_name, class_cnt)`

Given the `approach_name` and output `class_cnt`, build and return a PyTorch neural network that takes a color 32 x 32 image and outputs a vector with `class_cnt` elements.

You do not have to move this model to the GPU (that is done for you in the provided scripts).

Any of your networks should have the following attributes:

1. The input channels should be 3.
2. The output layer should have `class_cnt` nodes.
3. The network must have at least 3 convolutional and/or fully-connected layers, not including the last output layer.
4. **You are free to use ANY of the layer types available in PyTorch.**
5. You may choose to use one of the existing pre-trained networks (like VGGNet) and fine-tune it.

`train_model(approach_name, model, device, train_dataloader, test_dataloader)`

Given the provided model, the device it is located, and the relevant dataloaders, train this model and return it.

A couple of caveats:

- **You can ONLY train on the data from `train_dataloader`.**
- You may print out how the network is doing on `test_dataloader`. **However, you may NOT use `test_dataloader` as a validation set** (e.g., you cannot use it to tune learning rates dynamically).
- You are free to choose any optimizer you wish.
- You are free to decide how many epochs of training you use.
- **Your loss SHOULD be `CrossEntropyLoss`.** You are, however, welcome to add other losses if you believe it will help.

**Note that, in general, differences here do NOT count as acceptable approach differences, such as:**

- **Epoch counts**
- **Different optimizers**

## Acceptable Approach Differences

Your approaches should be sufficiently different from each other! Acceptable differences include:

- Different sizes of filters
- Notably different filter counts (32 vs. 64, not 32 vs. 33)
- Notably different node counts
- Consistently different activation functions.
- Adding 1 or more additional layers
- Adding skip connections
- With or without significant data augmentation (or sufficiently different variations of data augmentation)

**I actively encourage you to look for examples of working networks.**

**HOWEVER, if you do use code from another source, cite it in the body of the code like so:**

**# Taken from: <https://keras.io/getting-started/sequential-model-guide/>**

**NOTE: Even if you cite it, you may NOT use code from another student (whether presently in the course or from a previous semester).**

**However, any code in the slides for this course is freely usable.**

**If you are unsure whether any variants you intend to experiment with are different enough, ASK ME.**

**TRAINING CAN BE VERY TIME-CONSUMING. DO NOT LEAVE THIS UNTIL LAST MINUTE!**

## Provided Programs

### Train\_A05.py

- Asks you which approach you wish to train.
- Loads the data.
- Trains the model.
- Saves the model to assign05/output

### Eval\_A05.py

- Asks you which approach you wish to evaluate (if -1, evaluates all).
- Evaluates the model(s) on training and testing data.
- Prints and saves the results for each approach to assign05/output with filename: <approach name>\_RESULTS.txt or ALL\_RESULTS.txt.

## BONUS POINTS

For the entire assignment, bonus points will be awarded to the submission that has the following:

- (+%5) Best **testing accuracy** in class
- (+%5) Best **testing F1 score** in the class
- (+%5) Experimenting with 4 variants or more

## Submission

Make sure the following is committed on your repo:

- **A05.py**
- The contents of **assign05/output**, including:
  - o **Your model files (model\_\*.pth) UNLESS they are over 10MB!**
  - o **ALL\_RESULTS.txt**

**PLEASE add the following to your .gitignore file:**

- data/
  - o This is where CIFAR10 is downloaded
- assign05/output/bigmodel.pth
  - o Any models over 10MB

## Grading

- (80%) Programming
- (20%) Output folder