# Homework 3 — Convolutional Neural Networks

## 1 Scope

This assignment works on the basics of computer vision from implementing convolution to building object detectors

## 2 Instructions

- Deadline for this assignment is **Feb 25** via Gradescope
- The submission portal will remain open till end of the semester. Do keep in mind the late submission policy (Check slides from first lecture)
- Your submission has two parts: (i) a PDF report that details all the requested deliverables (without any code!!) and (ii) the code for the assignment (as .py files). You will upload the PDF report in gradescope, marking which parts of the report corresponds to which deliverable. The code will be uploaded separately, and we will run "software similarity" software on it to detect violations of the integrity policy.
- You can discuss this HW with others, but you are **not** allowed to share, borrow, copy or look at each other's codes.
- All code that you submit must be yours (except for the starter code that we provide). Specifically, you are **not** allowed to use any software packages or code from the internet or other resources, except for `numpy`, `torch`, `torchvision`, `scipy`, `PIL`, `matplotlib`, and other built-in packages such as `math`.

## 3 Problem Set

**Deliverable 1: Object Classification with CIFAR-100 Dataset** In this part, your will be using the CIFAR-100 dataset. This dataset contains images of various objects grouped into 100 classes. You'll build both a fully connected neural network and a CNN and your models will classify objects from the CIFAR-100 dataset. You will train the models, experimenting with different hyper parameters, learning rate schedulers, augmentations, initializations etc to achieve optimal results

You can use PyTorch layers like usual for the fully connected network. However for the CNN, you will be implementing your own `nn.Conv2d` and `nn.MaxPool2d` (see attached started code). To convince yourself (and us) of its correctness, you will also write a minimum of two test cases per function.

**Expected results:**

- Above 10% accuracy for both FCNN and CNN

**Deliverables:**

- Code
- Two test-cases for each custom layer function
- Max Accuracy and model architecture + hyper-parameters used for FCNN
- One pair of train and test losses plots for FCNN
- Max Accuracy and model architecture + hyper-parameters used for CNN
- One pair of train and test losses plots for CNN
- Explain all learning and insights from changes made to achieve highest possible accuracy
- Visualization of FCNN predictions
- Visualization of CNN predictions

**Deliverable 2: Training for Optimal Results** In this section of the assignment, you'll take your models to the next level by modifying the model and achieving benchmark performance. You'll experiment with different architectures, hyper-parameters, learning rates, and optimization techniques to achieve the best possible accuracy for object classification (CIFAR-100). Your journey will involve analyzing the impact of these changes on the models' accuracy and convergence. Feel free to write your own modules, and make changes to starter code to achieve highest possible accuracy. Results are expected to be submitted in the same format as previous sections however. For your reference, here are the benchmark standards achieved by state of the art classifiers: image-classification-on-cifar-100
**Note**: You can implement one of the **AlexNet, ResNet18, ResNet50, etc.** architectures discussed in class, and use tricks/improvements including but not limited to data augmentations, learning rate schedulers, different loss functions, etc. However, **No transfer learning or pre-trained models allowed.**

**Expected results:**

- Test and Train Accuracy of 60% or higher for full credit for your best model on CIFAR-100 dataset.
- Visualization of results and number of correct predictions visually out of 10.

**Deliverables:**

- Code
- Max Accuracy and model architecture + hyper-parameters used
- One pair of train and test losses plots
- Explain all learning and insights from changes made to achieve highest possible accuracy
- Show visualization results

**Deliverable 3: Naive Object Detection** Now we will switch to object detection. We have worked on classification so far. Our models can tell us if an image contains a cat/dog/car/etc. But what if an image contains multiple objects? A more powerful way to solve this problem would be to not only identify which objects exist in an image, but where they exist in the image.

We will start with a naive approach to object detection. For these two cat and dog images attached, we want you to crop the image into multiple sub-images and classify each sub-image. Then combine these sub-image results to create a bounding box around each cat and dog. For this problem, you may use any of the classification architectures above with their pre-trained weights.

This is left intentionally vague to allow for your own approach. There are no wrong answers.

**Expected results:**

- There must be several bounding boxes in the general region of the cats and dogs. We don't expect perfect results

**Deliverables:**

- Code
- Explain your approach to making a detector using a classifier
- The two images annotated with bounding boxes

What you just did was similar to how some object detection models work! This is known as two shot detectors. The first step is to generate a ”Region Proposal” (your subimages) and the second step is to perform classification on the region in question.

This can be quite slow, and so led the way for Single Shot Detectors (SSDs) like YOLO.

**Deliverable 4: Performance Profiling** For this part, we will compare the YOLO and Faster-RCNN models on the COCO Detection dataset. You can obtain YOLO from the ultralytics package and the rest are obtained from PyTorch.

Measure the mean average precision (mAP) and detection latency for both models.

**Deliverables:**

- Code
- mAP and latency for YOLO
- mAP and latency for Faster-RCNN

You can find COCO dataset at https://cocodataset.org/#download. You should download and evaluate on COCO 2017 Val images and their corresponding detection annotations. You should implement mAP yourself based on its definition without using packages like COCO API.

Here we define latency as the amount of time it takes from inputting an image to getting detection results with a batch size of 1.