Assignment 4 - Learning and Inference for Visual Recognition

1. **Independent Reading**.

   1. ImageNet Classification with Deep Convolutional Neural Networks (`papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`)
   2. Transformer Networks (`https://arxiv.org/abs/1706.03762`)

2. **(12 pts)** In this assignment, you will be training a Convolutional Neural Network (CNN) from scratch. Recall in Assignment-4, you used pre-trained weights of ResNet50 for feature extraction and trained just a logistic regression model, which is essentially the final layer of a CNN used for classification. In this assignment, you will be using the CIFAR-10 dataset, which contains 32×32 images divided into 10 categories. The training set contains $50,000$ images, and the test set contains $10,000$ images.

   You have to create a file named `CNN_training.ipynb`. It will have a function named `train` and a class named `ConvNet`. For your convenience, starter codes related to these two have been provided in two separate files `train.py` and `cnn_model.py`, available in the assignment folder. Please adhere to the instructions provided in the starter codes and copy them to the `CNN_training.ipynb` file.

   You need to fill in the following.

   [6 pts] Fill in object class named `ConvNet` in `cnn_model.py`. You'll define the network layers in the method `__init__` and the forward pass in the method `forward`. We create an instance of `ConvNet` in `train.py`, which takes a tensor with dimension $(B, 3, 32, 32)$ as an input, where B is the batch size. The output should be a matrix of dimension $(B, 10)$ representing the confidence (before applying the softmax function) for the 10 categories. Your CNN should have 4 convolutional layers with ReLU, Dropout, Batch Normalization and Max pooling operations in each (if required), followed by a linear layer and then a classifier layer at the end. You can use the function `_initialize_weight` in `ConvNet` to initialize weights of the convolutional layers. You are allowed to choose stride, weight decay, dropout rate, filter kernel size, number of output feature maps for each layer and other hyper-parameters. FYI: A 4 layer network (written properly) should be able to obtain at least an accuracy of 65% on the test set.

   [1 pts] Fill in loss function with cross-entropy loss.

   [1 pts] Fill in to obtain accuracy of current batch of data.

   [2 pts] Fill in to obtain test accuracy over the entire test set and append it to the test_accuracy variable.

   [2 pts] Plot training loss, train accuracy and test accuracy from the saved variables. Can you infer based on the plots, whether the model is overfitted, under-fitted or perfectly fitted ?

Submission Protocol. You must submit codes written in Python for every problem. Unless otherwise specified, the code for each problem must be a Jupyter notebook i.e. `.ipynb` file. You are recommended to use Colab. **Your files must have output plotted and/or printed as directed. Failure to follow instructions will lead to a deduction in marks.** You should add comments to your codes to make them reader friendly. All codes must be uploaded in separate folders named after the problem number. Keep all the images necessary to run a code in the same folder as the code while you are submitting. You must provide explanations in your notebooks related to each problem (if required). The zip file to be uploaded must have your name.