[Overview] In this project you will explore the basics of supervised learning using a neural network with several convolutional and fully connected layers. Your experiments will be conducted on the CIFAR-10 dataset.[1] We provide you with starter code in PyTorch,[2] but you are free to use a framework of your choice. In the starter code, several sections are marked as `TODO`, indicating locations where you need to add code (typically a handful of lines that are key to complete an algorithm).

(a) Implementing the training loop [20pts] Define the loss function and optimizer, then complete the implementation of the training loop. After your implementation is complete, your `train_and_test()` function will return a sequence of actual (rather than placeholder) loss-function values on the train and test partitions of the dataset.

(b) Training the model [20pts] Write a function to visualize loss-vs-epoch plots, showing the train-partition loss and the test-partition loss on the same figure. Extend `train_and_test()` to allow the ability to interchange train and test data, thus simulating a data-poor regime; we'll call this data split iCIFAR-10 (*inverted* CIFAR). Decide on the optimization parameters, number of epochs, optimizer settings, etc. Then use those parameters to generate loss-vs-epoch plots, one for the train/test partitions of CIFAR-10 and the for the train/test partitions of iCIFAR-10.

(c) Defining a new model [20pts] In class we'll study several published CNN architectures and *design patterns*, each developed to achieve a particular objective. Decide on an objective (e.g., better accuracy or better computational efficiency, etc.) and implement a (shallow) network `Net_1`, possibly derived from `Net_0`. Compare `Net_1` to `Net_0` on CIFAR-10 using loss-vs-epoch plots.

As a side objective, you need to demonstrate (in your project writeup and comments in your code) that you understand how to perform shape computations e.g., to ensure that a linear layer is specified with the correct number of input features given the preceding convolutional or pooling layer. You can demonstrate this on `Net_1` or on a separate network `Net_1c`. For example, you could keep the `padding` argument of the `nn.Conv2d` layer at its default `padding=0` so that the output shape changes and you have to take that change of shape into consideration when computing the shape of the subsequent layers.

(d) Performance evaluation [20pts] While monitoring the values of the loss function is useful for assessing whether the network is training properly, once trained the network is typically evaluated using statistical performance-evaluation metrics (possibly fine-tuned for a given application). In this project you will implement the precision-recall (PR) metric by completing `csep576_precision_recall_curve()` so that it returns a correct sequence of precision/recall values. Once this function is implemented, evaluate `Net_0` and `Net_1` on CIFAR-10 and iCIFAR-10 using PR plots. In your writeup, comment on which classes are recognized well (in the PR sense), and which ones underperform compared to the rest?

(e) Dealing with overfitting [20pts] In this part of the assignment you will try to mitigate overfitting on iCIFAR-10 via regularization. Select and implement one of the methods described in [CVA2] Ch.5 by extending `Net_1` to `Net_2`. Using the same experimental settings as before (optimizer, number of epochs) as in (d), compare `Net_1` and `Net_2` using "loss curves" and the PR plots.

[What to submit] Your completed p3.py and a PDF report that includes motivation, hypotheses you wanted to test, findings/lessons you learned, and answers to specific questions, e.g., at the end of (d). The report should be backed up by figures/tables you generated in (a)-(e).

[1] https://www.cs.toronto.edu/~kriz/cifar.html
[2] https://pytorch.org