PMP596 Homework 2

Due October 28, Thursday, Before class (6:00 PM)

Problem 1: Image Classification (40%)

In this assignment, we will solve a simple image classification problem on the CIFAR-10 dataset. It has the classes: 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size. There are 50000 images in the training set, and 10000 images in the testing set.

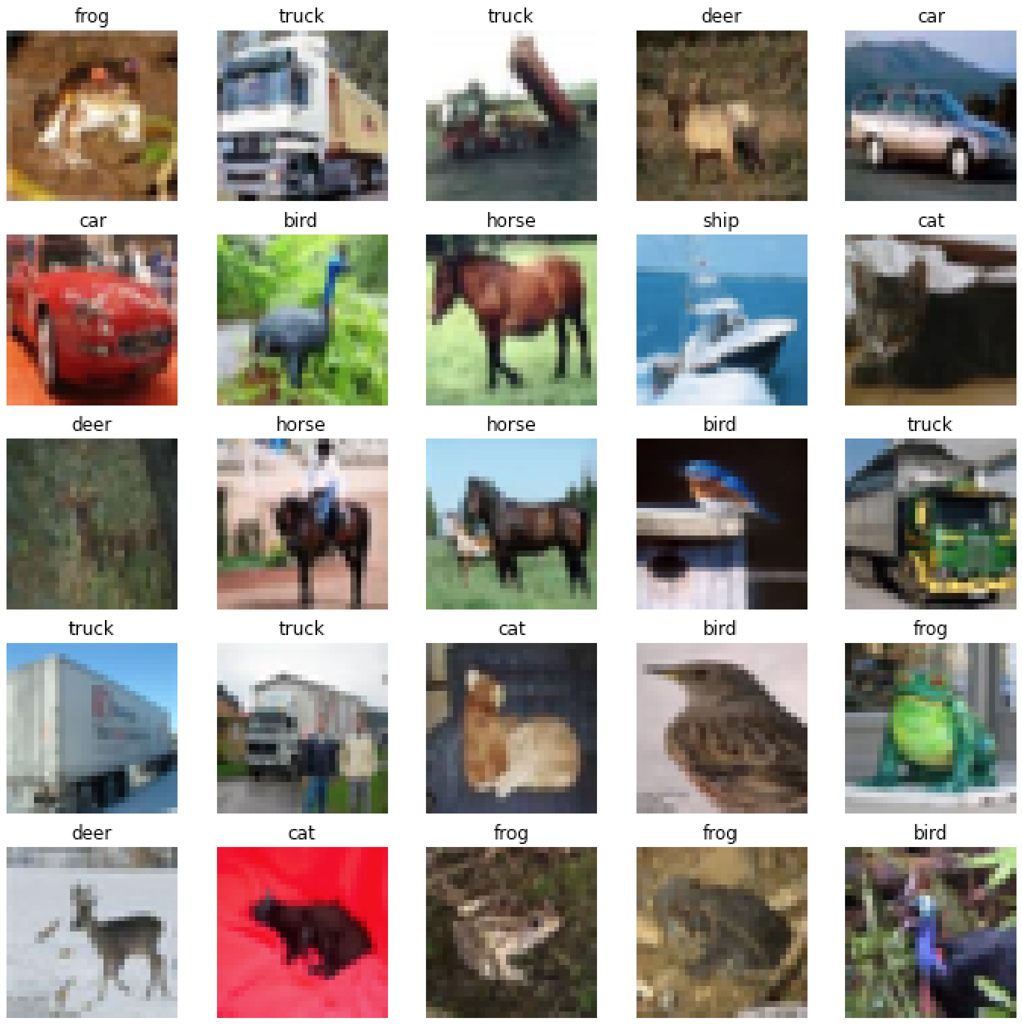

Fig. 1: Some sample images in CIFAR-10 dataset.

(a) Multilayer Perceptron (MLP) (15%)

In HW1P2, we loaded MNIST dataset from a `.mat` file. Here, we use `torchvision` to load and visualize some built-in datasets.

1) Follow the instructions in `HW2_P1.ipynb` to **load and visualize** the MNIST dataset.
2) Write code to **load and visualize the CIFAR-10** dataset in a similar manner. Note that we typically **normalize** the data during the loading process. The code for this transform is written in `HW2_P1.ipynb`.

Use MLP to achieve image classification.

3) Define the following **MLP** using PyTorch to do image classification on CIFAR-10 dataset according to Fig. 2:
   - # of hidden layers: 3
   - # of neurons in the layers: [2048, 1024, 512]
   - Activation functions: ReLU()
   - Dropout for all hidden layers: 30%
   - Output layer + softmax

   Note: to feed images into MLP, each 3D image (3x32x32) needs to be **flatten** into a vector (3072).

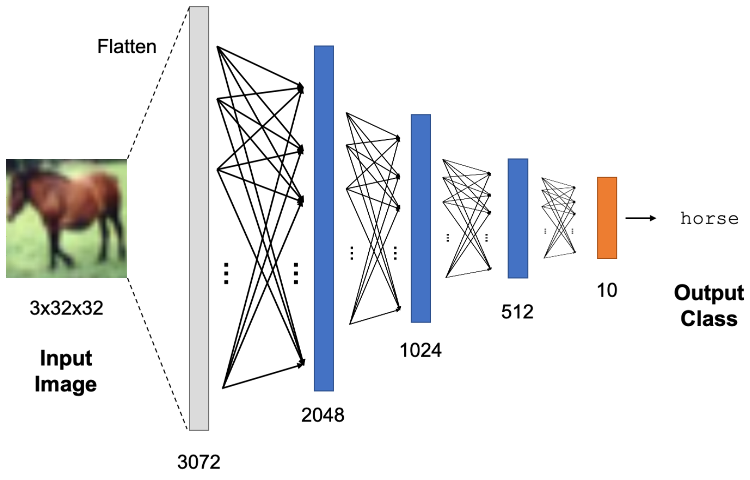

Fig. 2: Architecture of the MLP.

4) Use the following **data augmentation** techniques to the training set:
   - Shifting: randomly shift the images up/down and left/right by within 10%.
   - Rotating: randomly rotate the images by some angles.
   - Flipping: horizontally flip the images.
   - Adding Noise: randomly add some small Gaussian noise to the images.

   Thus, after the above data augmentation, you will get in total 50000x5 training images. Use the augmented training images to train the MLP with the best performance and report the new accuracy performance. (Hint: you can use PyTorch's built-in function for data augmentation.)

5) Evaluate the classification accuracy on the testing set. You can also try other parameters and report the testing accuracies. Please report the performance with its parameters in a table.

(c) Convolution Neural Network (CNN) (20%)

1) Define a **CNN** using PyTorch to do image classification. Develop the CNN from the previous PyTorch NN tutorial.

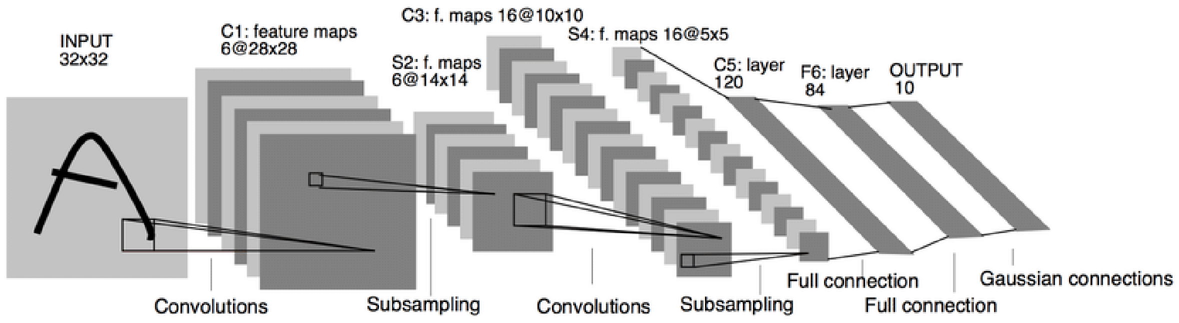

Fig. 3: LeNet-5 architecture for you to start from.

2) Try different network **parameters/configurations**:
    - # of conv: 2, 5, 9
    - Kernel size (conv): 3x3, 5x5
    - Stride (cov): 1x1, 3x3
    - Dilation: 1x1, 3x3
    - Dropout in FC: 0%, 30%

   Similar to (b), please include data augmentation for your CNN. You should try to achieve **at least 80%** accuracy on the CIFAR-10 testing set by using different network configurations (can use other configurations and techniques not mentioned above).
3) Evaluate and report the accuracies using different configurations in a table.

(d) Discussions (5%)

From the results in (b) and (c), discuss:

1) Which framework can achieve higher accuracy, MLP or CNN? Briefly explain the reason.
2) Which parameter can potentially affect your performance most?

Problem 2: GAN (25%)

In this assignment, we will implement a generative adversarial network (GAN) to generate new celebrities after showing pictures of many real celebrities using the CelebA dataset.

Fig. 4: Some sample images in the CelebA face dataset.

(a) Prepare CelebA Dataset (3%)

1) Download the CelebA dataset from Canvas (~1GB) and upload to your Google Drive. Then, unzip the data to your Google Colab server.
2) Load the dataset into PyTorch dataloader and visualize some sample images in this dataset.

(b) GAN Implementation (20%)

The discriminator is made up of strided convolution layers. The input is a 3x64x64 input image and the output is a scalar probability that the input is from the real data distribution. The generator is composed of convolutional-transpose layers. The input is a latent vector z, that is drawn from a standard normal distribution $N(0, 1)$ and the output is a 3x64x64 RGB image. The strided conv-transpose layers allow the latent vector to be transformed into a volume with the same shape as an image.

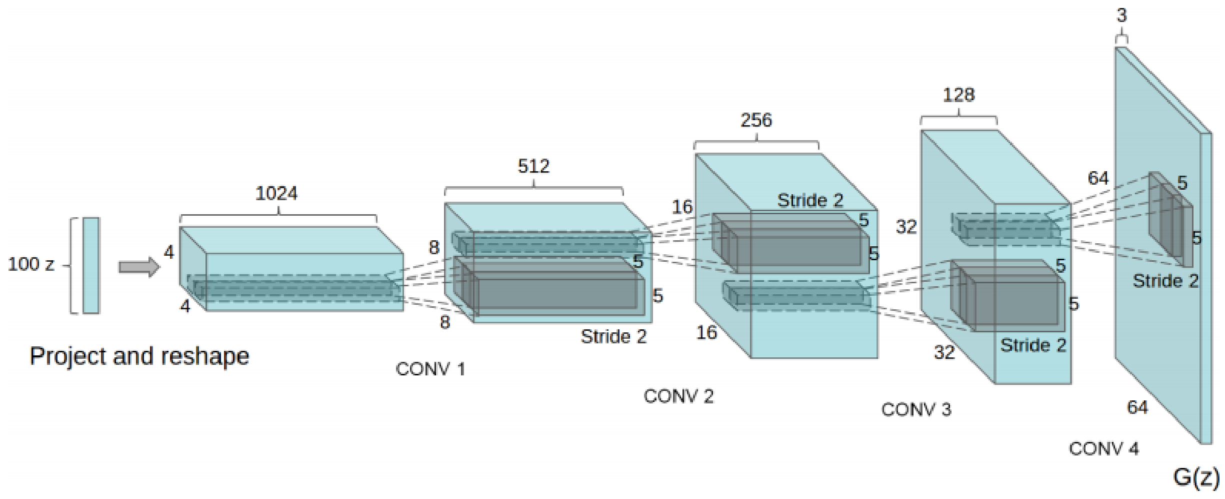

Fig. 5: The architecture for the GAN generator.

The details of the implementation are introduced in `HW2_P2.ipynb`. Please read it carefully before working on this HW. The implementation of the GAN includes the following components (please follow the provided sample code):

- Weight Initialization
- Define Generator
- Define Discriminator
- Loss Functions and Optimizers
- Training Strategy

Please do the following:

1) Read the implementation code of GAN in `HW2_P2.ipynb`.
2) Run the GAN and visualize the results using the code in `HW2_P2.ipynb`.
3) Train for more epochs to see if the results can get better.
4) Change the size of images from 64x64 to 128x128, and retrain the network. Note that after changing the image size, some of the network parameters also need to be changed.

(c) Discussions (2%)

After GAN implementation, think about the answers for following questions:

1) Why does the training strategy for GAN need to be carefully designed?
2) Besides generating fake images from the noise, what other applications can you come up with for GAN?

Problem 3: CycleGAN (10%)

Image-to-image translation is the task of transforming an image from one domain (e.g., images of zebras), to another (e.g., images of horses). Ideally, other features of the image — anything not directly related to either domain, such as the background — should stay recognizably the same. As we might imagine, a good image-to-image translation system could have an almost unlimited number of applications. Changing art styles, going from sketch to photo, or changing the season of the landscape in a photo are just a few examples.

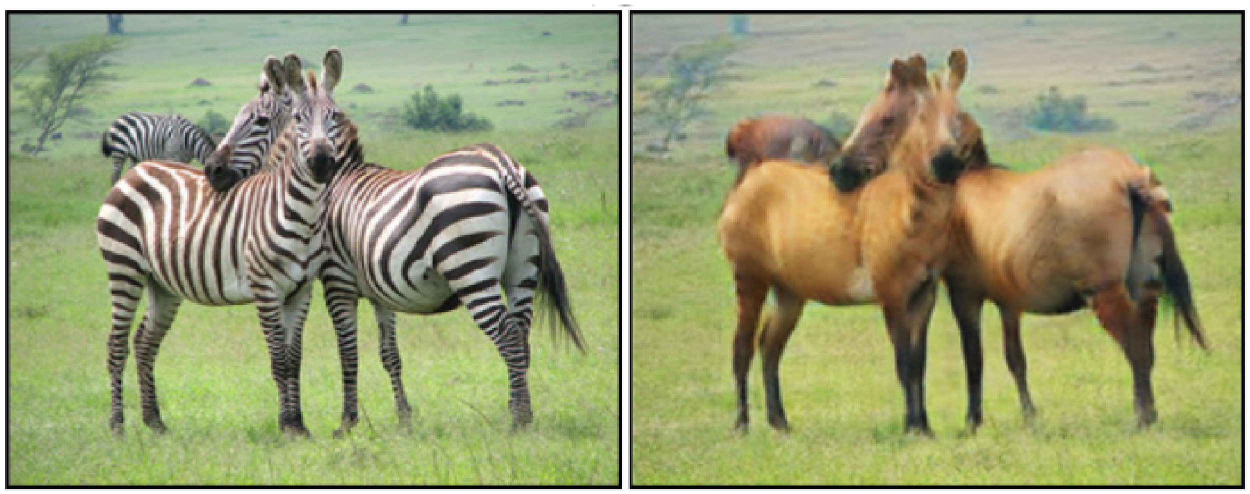
Fig. 6: An image of zebras translated to horses, using a CycleGAN

In this problem, you will implement a CycleGAN on the horse2zebra dataset.

(a) Install CycleGAN and Prepare Dataset

1) Install CycleGAN from the official GitHub repository.
2) Download the `horse2zebra` dataset. Visualize some sample images.

(b) Training and Inference (5%)

1) Train CycleGAN for about 5 epochs. Test your trained model and visualize some results. (The results may not be very good since the model cannot converge within 5 epochs.)
2) Download the pre-trained model for `horse2zebra` and `zebra2horse`. Test the pretrained model and visualize some results. Compare and discuss the pre-trained model and your model in 1).
3) Download some horse/zebra images from the internet and do the style transform using the pre-trained model.

(c) Implement on Other Datasets (5%)

Try (a) and (b) on two other dataset pairs selecting from the following and visualize your results. (You can just use the pre-trained model for this dataset)

```
apple2orange, orange2apple, summer2winter_yosemite,
winter2summer_yosemite, horse2zebra, monet2photo, style_monet,
style_cezanne, style_ukiyoe, style_vangogh, sat2map, map2sat,
cityscapes_photo2label, cityscapes_label2photo,
facades_photo2label, facades_label2photo, iphone2dslr_flower
```

Problem 4: Paper Reading (25%)

Title: Deep Residual Learning for Image Recognition
Authors: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

Abstract: Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers. The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions, where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.