Due Date: February 28, 24:00

Instructions

- *This assignment is involved – please start well ahead of time.*
- *For all questions, show your work!*
- *Submit your report (PDF) and your code electronically via the course Gradescope page.*
- *You can use the Jupyter notebook* `main.ipynb` *to run your experiments (optional).*
- *For open-ended experiments (i.e., experiments that do not have associated test-cases), you do not need to submit code – a report will suffice.*
- *TA for this assignment are* ***Johan Obando*** *(IFT6135B) and* ***Jerry Huang*** *(IFT6135A).*

Summary:

In this assignment, you will build several models for image classification on the CIFAR10 dataset. The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. You are provided a PyTorch dataset class (torch.utils.data.Dataset) named CIFAR10 from the torchvision and have the train, valid and test splits given. Throughout this assignment, the shape of CIFAR10 data is (Batch, Channels, Height, Width).

In the first problem, you are expected to build and train an MLP. In the second problem, you are expected to build and train a ResNet18. In the third problem you are expected to build and train a recent architecture called MLPMixer. As you can see, the evolution of these architecture also reflects the recent research landscape in computer vision. It starts with MLP, and then CNN dominates, and now we find that MLP, when done right, is actually not bad!

Coding instructions You will be required to use PyTorch to complete all questions. Moreover, this assignment **requires running the models on GPU** (otherwise it will take an incredibly long time); if you don't have access to your own resources (e.g. your own machine, a cluster), please use Google Colab. The main entry point for launching experiment is `main.py` or respectively `main.ipynb`. During development, you can use the unit tests defined in `test.py` to help you. Remember these are only basic tests, and more tests will be run on Gradescope.

Submission You are expected to submit `mlp.py`, `resnet18.py`, `mlpmixer.py` and `utils.py` to gradescope for autograding, as well as a PDF report for experimental and open-ended problems.

Problem 1

Implementing an MLP (5pts) In this problem, you will implement an MLP. An MLP consists of multiple linear layers each followed by a non-linear function. You can find the code template in `mlp.py`.

1. You are expected to complete `class Linear` in the file `mlp.py`. The class must have two attributes: `weight` and `bias`. Complete the forward function accordingly.
2. Now you can move on to the `class MLP`. In `_build_layers`, you are expected to build the hidden layers as well as the output (classification) layer. In `_initialize_linear_layer`, you are expected to fill the bias with zeros, and use the Glorot & Bengio (2010) normal initialization for the weights. Try to refer to the pytorch document for this part.
3. Complete `activation_fn` so that it can process inputs according to different `self.activation`. Complete the forward function according to docstring.

Problem 2

Implement ResNet18 (15pts) ResNet proposes to leverage the residual connections so that the network can be trained with many more layers. It is shown that the increasing depth helps neural networks achieve better performance in many vision tasks. In this problem, you are going to implement a ResNet18 architecture in `resnet18.py`. You can find the architecture details in Figure 1.

1. We provide the code template for the ResNet block in `class BasicBlock`. You can see that there some question marks in the initialization function for missing arguments. Please fill-in the arguments so that the shape of each component makes sense. After that, complete the forward function according to the diagram in Figure 1.
2. Complete the ResNet18 architecture in `class ResNet18`. Fill-in the missing arguments and complete the forward function.

Problem 3

Implement MLPMixer (20pts) In this problem you are going to implement MLPMixer architecture. Please provide your solution in `mlpmixer.py`. MLPMixer is a recent paper and the authors find that, if done right, MLP only architectures can also be very competitive. The architecture is shown in Figure 2.

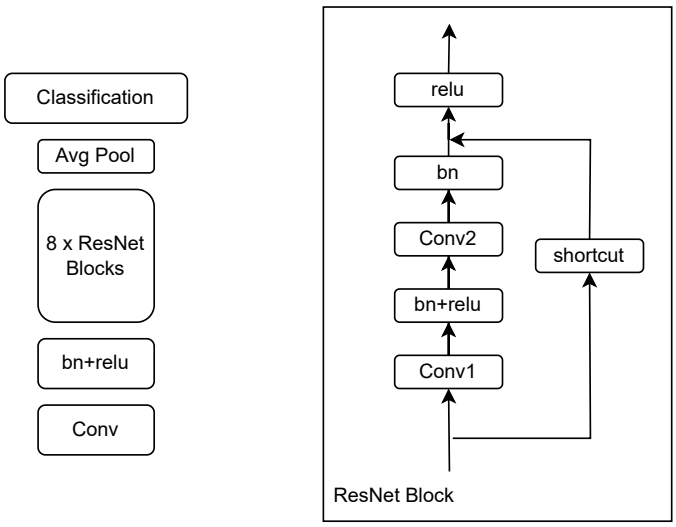

Figure 1: ResNet18 Architecture. **Left**: The architecture diagram of a ResNet18. It begins with a 3x3 convolution layer. The main body consists of eight ResNet blocks. At the end there is the global average pooling and the classification head. **Right**: This is a diagram of the ResNet block. In one branch, the inputs go through two convolution layers with batch normalizations and non-linearity functions. In the other branch, the inputs go though a shortcut module, which could be identity function if the shape matches. Finally the outputs of two branches are added together. The shortcut branch is also called the *residual connections*.

1. The first component you are going to build is `class PatchEmbed`. Unlike the previous architectures, MLPMixer divides the whole image into small patches, and performs a common linear transformation to these patches. E.g., if the image size is 256×256, and the patch size is 8×8. Then we have $32 \times 32 = 1024$ patches. Assuming our image has RGB channels and we embed each patch to a 512 dimension vector, then we need a linear layer of shape (192, 512). While the above procedure is quite complex, in practice we can achieve the above operation with a simple convolution layer, which you will implement. Please fill-in the missing argument of `self.proj`.
2. Implement the `class MixerBlock` and `class MLPMixer` according to the architecture diagram.

Problem 4

Training Models (60pts) In this problem you are going to train the architecture implemented above and perform some hyper-parameter search. Please see `main.py` or `main.ipynb`. For each architecture, we provide the model json config files in `model_configs`. For each of the experiment, train for at least 10 epochs. Please submit a PDF report to answer the following questions.

1. (5pts) Complete the function `cross_entropy_loss` in `utils.py` **without** using the built-in

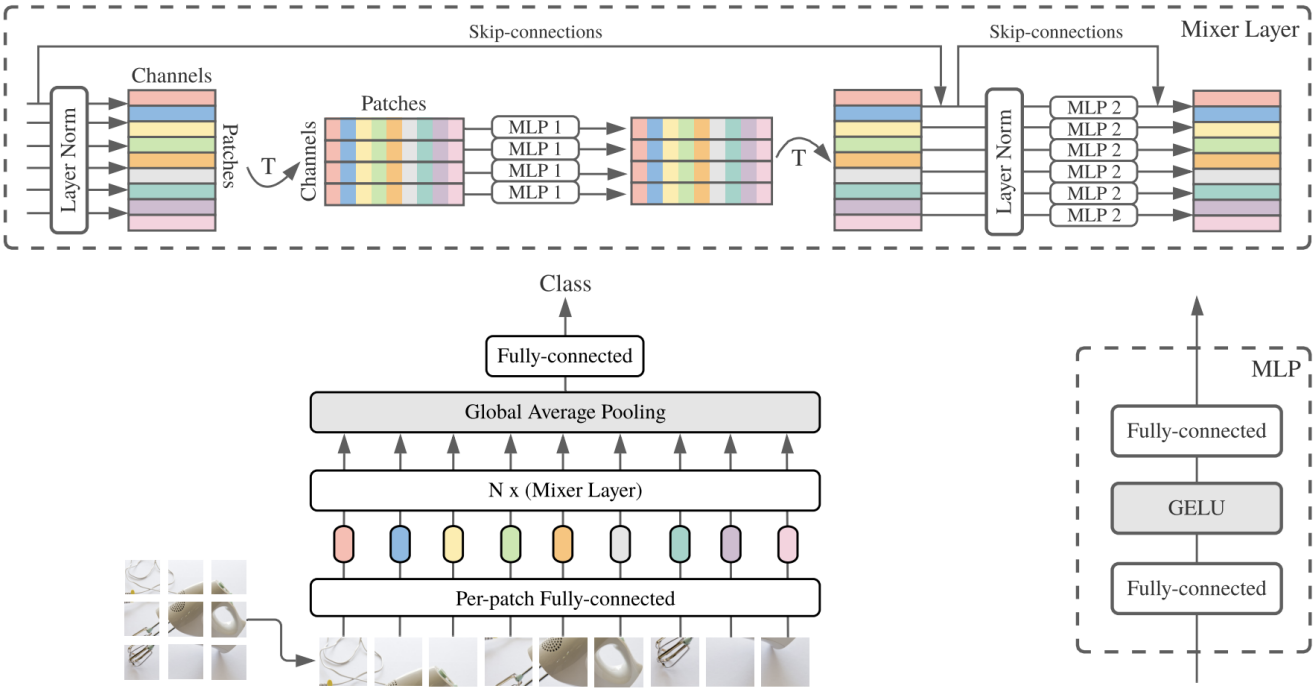

Figure 2: Borrowed from the MLPMixer paper. MLP-Mixer consists of per-patch linear embeddings, Mixer layers, and a classifier head. Mixer layers contain one token-mixing MLP and one channel-mixing MLP, each consisting of two fully-connected layers and a GELU nonlinearity. Other components include: skip-connections, dropout, and layer norm on the channels.

`torch.nn.CrossEntropyLoss` or `torch.nn.functional.cross_entropy`.

2. (10pts) For the MLP architecture, investigate the effect of the choice of non-linearity while keeping the other hyperparameters the same as the default. You are expected to provide four figures corresponding to training loss, validation loss, training accuracy, and validation accuracy, where the x-axis is the number of epochs. For each figure, use the legend to denote the non-linearity being used. Conclude which non-linearity is the best and give your explanation. Optionally, we provide the plotting utility function in `utils.py`.

3. (10pts) For the ResNet18 architecture, investigate the effect of learning rate with the Adam optimizer. Perform experiments with learning rates of 0.1, 0.01, 0.001, 0.0001, 0.00001. Provide the figures and explain your findings.

4. (10pts) For MLPMixer, investigate the effect of patch size. No recommended values are given, and you are expected to run at least 3 experiments. Remember there are only a few valid values for patch size for the given image size. Please provide figures and explain your findings. Also explain in text the effect on the number of model parameters and running time.

5. (5pts) Find your best ResNet18 model by experimenting with different hyper-parameter choices. Provide the hyperparameters in your report. Visualize the kernels of the first layer, which has a weight of shape (out_channel, in_channel, kernel_size, kernel_size). You can modify the `main.py` or add extra cell in `main.ipynb` for visualization. Since we have 64 output

channels and 3 input channels (RGB), one can view this as sixty-four 3×3 small images, where each image represent the kernel corresponding to that output channel. Note that this is an open-ended question. You can perform different pre-processing for visualization, e.g., standardizing the weight values, averaging across the channels to have gray scale images etc. You can see more details and examples in this blogpost. Please describe your visualization procedure in your report.

6. (5pts) Set the patch size to be 4, and find hyper-parameters for your best MLPMixer model. Provide the full hyper-parameters in your report. Visualize the weights (only first layer) of token-mixing MLP in the first block as is described in Figure 5 of the MLPMixer paper. Comment and compare your results with the convolution visualizations. Explain what you think is the reason behind the success of the MLPMixer, especially over normal MLP?

7. (10pts) Investigate how the width of the token-mixing and channel-mixing MLP layers affects the model's generalization ability. Train the MLP-Mixer model with varying widths for the token-mixing and channel-mixing MLP layers (e.g., 256, 512, 1024). Compare the training and validation accuracies across different widths. Analyze how the capacity of the MLP layers impacts overfitting or underfitting.

8. (5pts) Compare the gradient flow (e.g., norms of gradients at different layers) during backpropagation for all three architectures (MLP, ResNet18 and MLPMixer). Analyze and compare the behavior of gradients in each architecture.