Shireesh Reddy Pyreddy
CS548 - Assignment 05

A brief summary of your approaches and the differences between them:

- **NaiveDeepNetwork:** Basic one convolutional layer with kernel size as 2 followed by 5 linear layers with Tanh activation function and finally the output layer.
- **SimpleNetwork:** Taken from class is a CNN with 4 convolutional layers and kernel size as 3 with ReLU activations, and 2 max-pooling layers. Finally a fully connected layer is added with ReLU activation just before the actual output layer.
- **VanillaCNN:** Plain CNN with two convolutional layers, kernel size as 5x5, ReLU activations, and 2 max-pooling layers with stride as 2x2. Finally there are 3 fully connected layers added just before the output layer.
- **VanillaCNNWithDropOut:** Same as **VanillaCNN** with the dropout layers added after each layer.
- **DeepCNN:** Deep CNN with 6 convolutional layers, kernel size as 3x3, stride as 1x1, ReLU activations, and 3 max-pooling layers. Larger and deeper compared to VanillaCNN.
- **DeepCNNWithBatchNormDropOut:** Same as **DeepCNN** with batch normalization and dropout added to reduce overfitting.
- **VGG19Bn:** Fine tuned the VGG19 architecture with batch normalization and added a custom classifier with 2 fully connected layers, ReLU activation and dropout in the fully connected layers. Also, the size of the features are all the same.
- **ResNet34:** Fine tuned ResNet34 architecture and replaced with a custom classifier which has 2 fully connected layers followed by LeakyReLU instead of ReLU used for VGG19Bn. Also, the size of the features are different in each layer.

A brief discussion as to WHY you thought these variations would have made a difference:

My thought process was to show the advantages or disadvantages of adding a new concept in each network inorder to increase the accuracy or to show how it behaves with this dataset. So, I started from basic networks and built or fine tuned all the way to advanced network architectures like VGG19Bn and ResNet34. I used ResNet 34 in particular because most of the links or open source python notebooks implemented it and achieved at least 94% accuracy. Also, for VGG19Bn, I used 300 epochs just to test my laptop GPU capacity and I found out that it took 6.3 hours to train.

Results:

Approach	TRAINING_acc	TRAINING_f1	TESTING_acc	TESTING_f1
NaiveDeepNetwork	0.4754	0.4711	0.4574	0.4529
SimpleNetwork	0.9552	0.9553	0.7239	0.7250
VanillaCNN	0.8879	0.8872	0.6656	0.6627
VanillaCNNWithDropOut	0.8793	0.8790	0.7383	0.7388
DeepCNN	0.9589	0.9587	0.7813	0.7801
DeepCNNWithBatchNormDropOut	0.8682	0.8672	0.8181	0.8164
VGG19Bn	0.9981	0.9981	0.9131	0.9127
ResNet34	0.9974	0.9974	0.9641	0.9641

Citations:

- https://github.com/chengyangfu/pytorch-vgg-cifar10/
- https://www.kaggle.com/code/shadabhussain/cifar-10-cnn-using-pytorch
- https://www.kaggle.com/code/francescolorenzo/96-fine-tuning-resnet34-with-pytorch
- https://pytorch.org/vision/0.8/_modules/torchvision/models/vgg.html
- https://discuss.pytorch.org/t/how-to-extract-features-of-an-image-from-a-trained-model/119/63?page=4
- https://pytorch.org/vision/0.8/_modules/torchvision/models/resnet.html