Department of Computer Science, Rutgers University
CS534 – Computer Vision
Assignment 4

In this assignment you will implement convolutional neural networks to recognize traffic sign images. You will need to use the *keras* library to finish the task. First, read this amazing tutorial that walks you through the process of building a baseline model: https://chsasank.github.io/keras-tutorial.html.

Part I: Build a baseline network and reproduce the recognition accuracy

- You need to first download the German Traffic Sign Recognition Benchmark dataset: http://benchmark.ini.rub.de/?section=gtsrb&subsection=news.
- The dataset contains 43 types of traffic signs. There are 31367 images in the training set and 12630 images in the testing set. Your task is to train a network using the training images then classifies each traffic sign image in the testing set.
- Follow the tutorial to build the baseline network introduced in it. This network has 6 convolutional layers and 1 fully connected layer, where each pair of consecutive layers are bridged by a max-pooling and a dropout layer. In order to draw a network model, you can refer to the official document about model visualization in here: https://keras.io/visualization/.
- Train the network using the 31367 training images. Try to tune the training parameters, such as learning rate, decay rate, momentum, batch size, to see how the training loss is affected. **Note**: If you run the training on a CPU, it can take at most 10 minutes per epoch, so you might want to adjust the number of epochs you want to run for.
- Test the network using the 12630 testing images. You should get an accuracy near 97.92%, which is produced in the tutorial. Double-check your implementation if you get something far away from it.

Part II: Enhancement on the baseline

1) **Enhancement 1:**
   Replace the baseline network with VGG16. You need to write the VGG16 model by yourself. Specifically, you need to replace the code in cnn_model() with the VGG16 definition code. You can go online and refer to any open-sourced VGG16 definition code written in keras, such as this: https://github.com/fchollet/deep-learning-models. However, the code in cnn_model() must be written by you. The reason is to let you have a sense of how the network is built and understand the meaning of components of each layer. There is no better way to achieve that than letting you write down the model structure layer by layer.
   A more comprehensive introduction about VGG networks can be found here: http://www.robots.ox.ac.uk/~vgg/research/very_deep/.
   Now retrain the whole system. **Note**: When using VGG16, you need to initialize the model by the weights pre-trained on ImageNet dataset (which contains millions of images and 1000 image types). This is a common trick in deep learning, and it usually helps because such initialization can give a better starting point of our training process, since the weights are already pre-trained to understand a large volume of images (the ImageNet dataset). Thus, it is better than randomly initializing the weights then training only on ~30000 images. Details about how to load pre-trained models are shown in the first link in 1).

Once you finish training, do the same testing as in Part I and report the new accuracy. **Note**: You may not necessarily get a better result, but it should be on par with the baseline. If not, you will need to double-check your code or try to tune the training parameters a little bit. An accuracy that is 10% lower than the baseline is considered as abnormal and some credits will be deducted.

2) **Enhancement 2:**
Add BatchNormalization layer to the **baseline** network. This layer is widely used to accelerate converging speed during training. A brief introduction and the keras API of this layer can found here: https://keras.io/layers/normalization/.
Again, retrain the model and report the testing accuracy. The result should also be on par with the baseline, if not better. An accuracy that is 10% lower than the baseline is considered as abnormal and some credits will be deducted.

Part III: Basic questions of deep learning

In this part, you need to answer the following questions in a concise manner. These questions involve important factors during training and testing of the baseline model. You might not feel their existence since the tutorial has provided good examples for you. However, these are usually critical things to concern when you want to build and train a new network from scratch for a new task.
In order to answer these questions, you need to fully understand the functionality of each type of layer in the baseline network, as well as other parameters during training and testing. The explanation of most concepts are very easy to find online, so they should not be too hard to answer.

1. *What is epoch? What is the relation between an epoch and an iteration?*
2. *What is learning rate? What is the effect if increasing it?*
3. *What is batch size? Suppose the memory is unlimited, what is the most ideal batch size then? In that case, how many batches are taken per epoch?*
4. *How much is a feature map shrunk by a 2x2 max-pooling layer? The baseline network has 3 2x2 max-pooling layers, thus what is the size of the output feature map after the last max-pooling layer, if the input feature map is 48x48?*
5. *What is a Dropout layer and why do we need it?*

Deliverables

train_baseline.py:	Code to train the baseline network. You code should be able to run by `python train_baseline.py` to start the training process
test_baseline.py:	Code to test the baseline network. You code should be able to run by `python test_baseline.py` to evaluate the model on all testing images and output accuracy
train_enhanced_1.py:	Code to train the network with Enhancement 1. You code should be able to run by `python train_enhanced_1.py`
test_enhanced_1.py:	Code to test the network with Enhancement 1. You code should be able to run by `python test_enhanced_1.py`
train_enhanced_2.py	Code to train the network with Enhancement 2. You code should be able to run by `python train_enhanced_2.py`
test_enhanced_2.py:	Code to test the network with Enhancement 2. You code should be able to run by `python test_enhanced_2.py`
Answers.txt	Answers to all the questions in Part III. Keep the answers concise