ECE 6524 – Deep Learning
No. P-01-6524

Project Assignment #1: Convolutional Neural Networks (CNNs)

In this assignment, you are asked to implement and train a Y-network built upon convolutional neural networks (CNN) (using Keras) for classification. For this assignment, you may need to use GPUs to speed up the training of CNNs. If you have access to GPUs locally or remotely (such as VT's ARC cluster), you are ready to follow the tutorial below to prepare the data set for training CNNs. Otherwise, use Google's Colaboratory (Colab) for your access to free GPUs.

1. Please read a tutorial on using Keras to train a CNN for binary classification of cats and dogs: "Cats and Dogs Image Classification using Keras" (https://pythonistaplanet.com/image-classification-using-deep-learning/). Specifically, you need to understand some basics on building a simple CNN using Keras.

2. Implement and train a Y-network to classify the classes in the CIFAR-10 and CIFAR-100 datasets (https://www.cs.toronto.edu/~kriz/cifar.html)
The Y-network to be implemented includes two CNNs (termed as the left CNN and right CNN branches): each with 3 convolution-dropout-pooling layers for feature extraction. A block diagram of the Y-network model is shown in **Fig. 2**.

For each CNN branch, the filter kernel size is (3, 3); the activation function is ReLU; the pooling size is (2, 2). The numbers of filters are 32, 64, 128, respectively, for three layers. The network combines the features using *concatenate* layer. The merge operation concatenate is similar to stacking two tensors of the same shape along the concatenation axis to form one tensor. (For example, concatenating two tensors of shape (3, 3, 16) along the last axis will result in a tensor of shape (3, 3, 32).) After the flatten layer that forms a feature vector, a dropout layer and a final dense layer (with C output nodes, where C is the number of classes, and an activation function of 'softmax') are used for classification. The loss function, 'categorical_crossentropy', should be used.

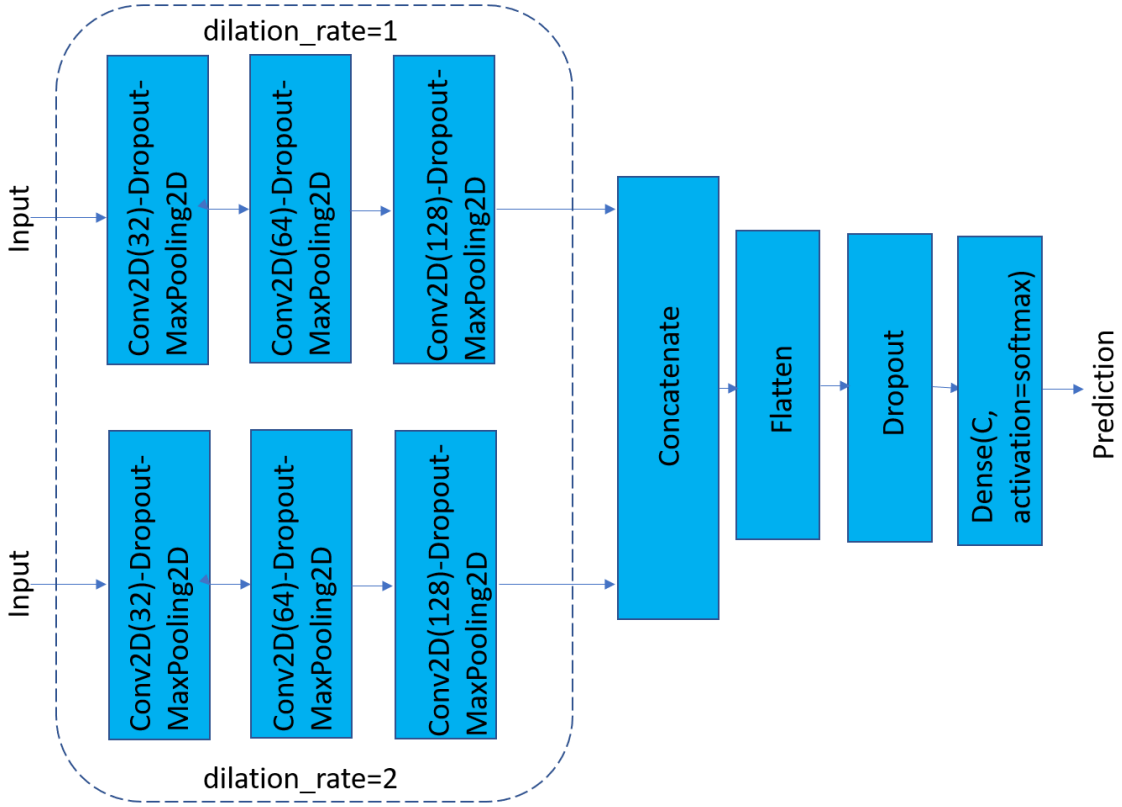

Fig. 2. A Block Diagram of the Y-network Model.

The Keras *Sequential.fit_generator() or Sequential.fit()* method can be used to train the Y-network. As instructed in the tutorial, the trained model can be saved to a file so that the model can be loaded (using *load_model()*) later for testing.

Report the training performance and validation performance. For this project, you should examine the overfitting problem by plotting model accuracies on training and validation data, which are recorded in the returning object (*History*) of the Keras *Sequential.fit()* or *Sequential.fit_generator()* method.

3. **Optimize the Y-network. Study the performances of** Y-networks with different numbers of layers, numbers of the filters, dropout rates, optimization methods, etc. Report the performances of several Y-networks.

4. **Visualize feature maps.** The activation maps, called feature maps, capture the result of applying the filters to input, such as the input image or another feature map. The idea of visualizing a feature map for a specific input image would be to understand what features of the input are detected or preserved in the feature maps.

In Keras, the *Model*() method can be used to build a new model that starts from the input layer and ends at a given convolution layer. The feature maps can then be generated by the *predict*() method (see sample codes below for an example).

```
from keras.models import Model
new_model = Model(inputs=model.inputs, outputs=model.layers[2].output)
feature_maps = new_model.predict(test_image_data_normalized)
```

Plot the feature maps obtained at each convolution layer for an input. From the experiment results, what is your understanding of the features extracted in each convolution layer?

You need to prepare **a written report** (in the pdf format) including the following sections: (1) Approach(es), (2) Experimental Results, and (3) Discussion.

In addition, you need to attach your **implementation codes** as separate files to the report.