

2019 Spring COM526000 Deep Learning - Homework 3

Convolutional Neural Network for Image Recognition

105061210 楊雅婷

Problems

1. Image Preprocessing (file: data_prepro.py)

- ✚ read every directory name (the same as class name) that can be use to construct a dict {name: number} to map between class names and labels
- ✚ use these class names to acknowledge paths to each class directory, and then we know the paths to every image
- ✚ use **opencv 'cv2'** to read images and **resize images to 64*64** (at first I try to resize those images to larger size, ex: 192*192 or 128*128, however, due to the lack of computation resources, I decide to use 64*64)
- ✚ return images and labels as **numpy array with shape (7411, 64, 64, 3) and (7411,)**

2. CNN Implementation (file: hw3_105061210.py)

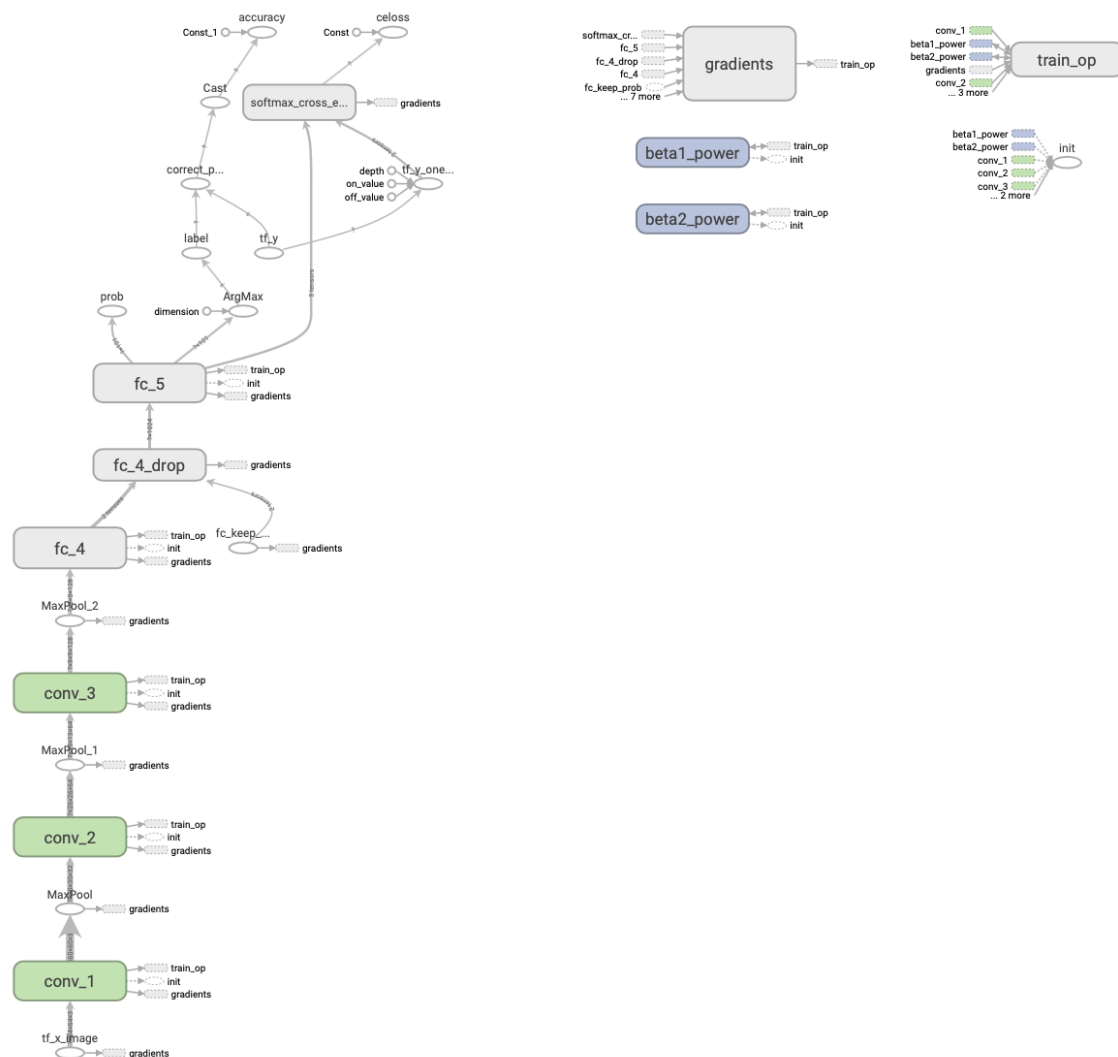
- ✚ I construct a CNN model as a class
- ✚ parameters:

batch size	256
epoch	30
learning rate	0.0005
conv: activation function	relu
conv: kernel size	(5, 5)
conv: strides	(1, 1, 1, 1)
pool: ksize	[1, 2, 2, 1]
pool: strides	[1, 2, 2, 1]
output:	32 -> 64 -> 128 -> 1024 -> 101

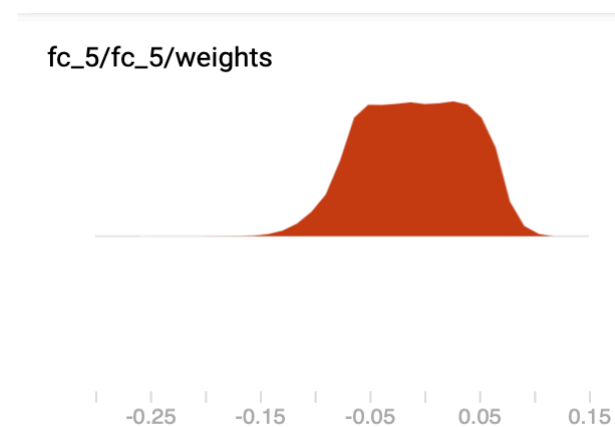
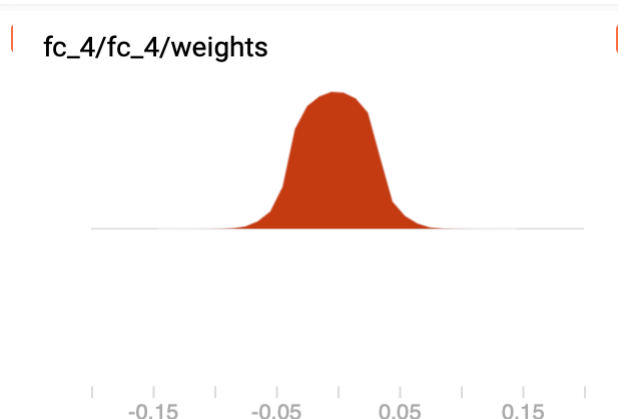
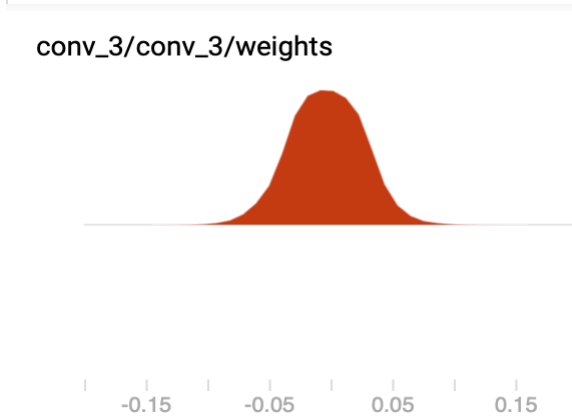
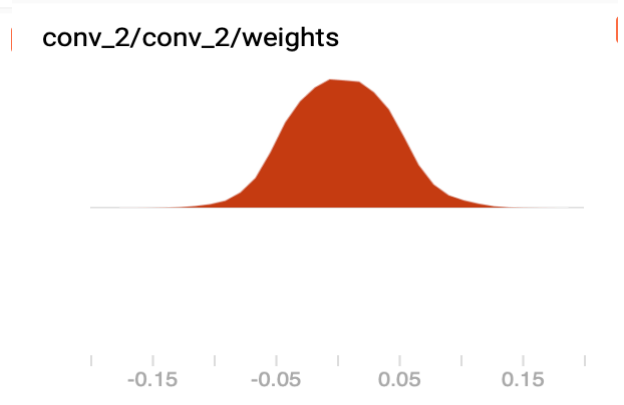
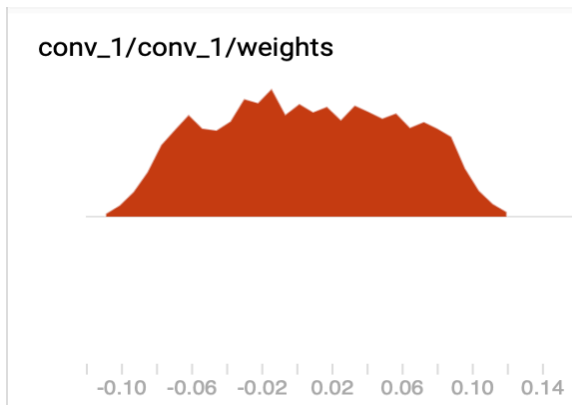
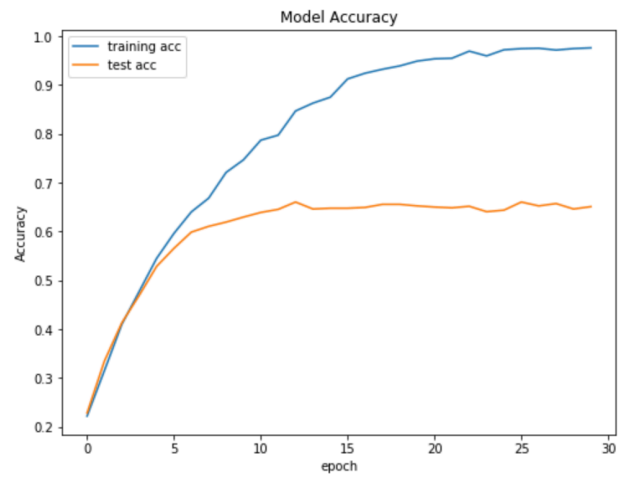
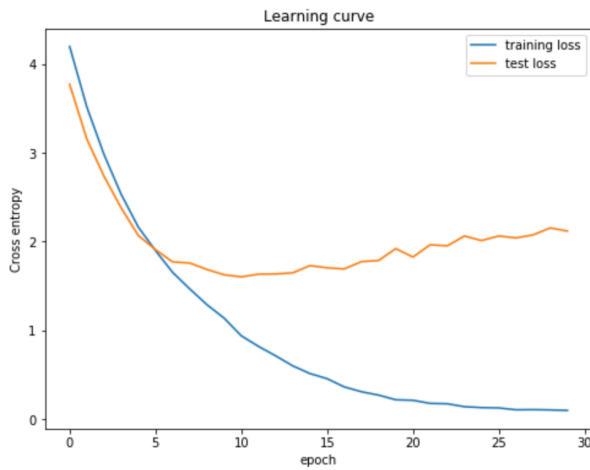
- Different kernel-size filters (3×3 , 5×5 , 7×7) will detect different size of features
- Stride has an effect both on how the filter is applied to the image and the size of the resulting feature maps
- Padding allows us to build very deep models and the feature maps do not dwindle away to nothing (SAME calculates and adds the padding to the input image (or feature map) to ensure that the output has the same shape as the input)



architecture (by Tensorboard):



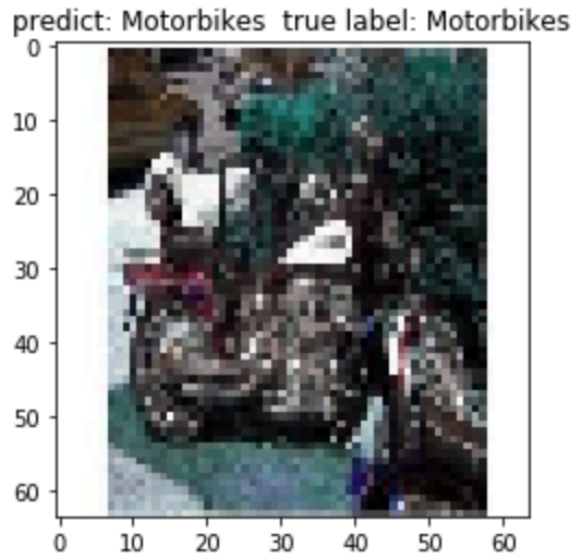
3. Learning curve, accuracy rate, and distribution of weights



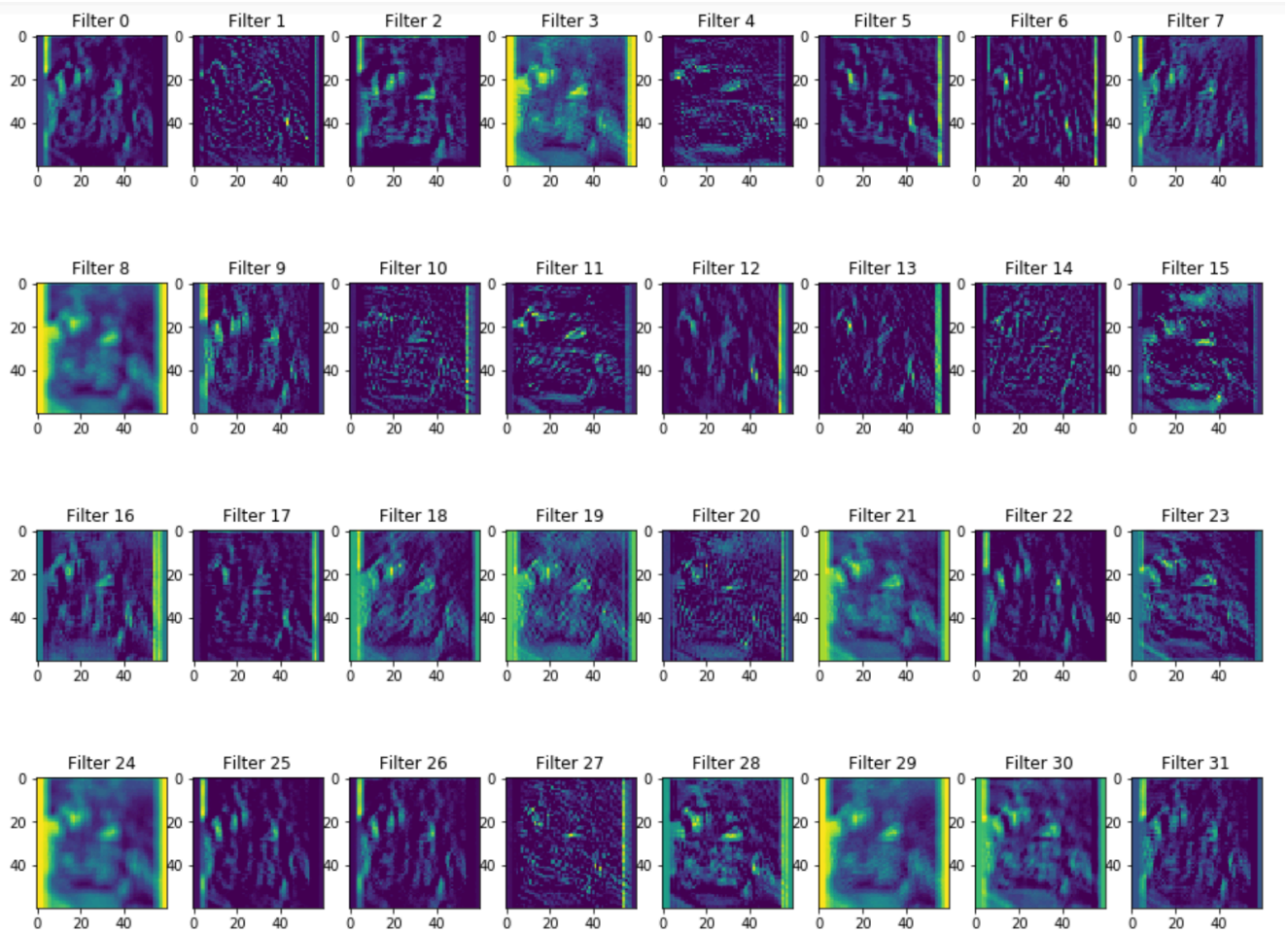
Those weights I want are in the conv_layer function. Instead of rewriting it, I use Tensorboard to record and view them

epoch: 30	train loss: 0.09640267
epoch: 30	train accuracy: 0.9761166
epoch: 30	val loss: 2.118379
epoch: 30	val accuracy: 0.6508689

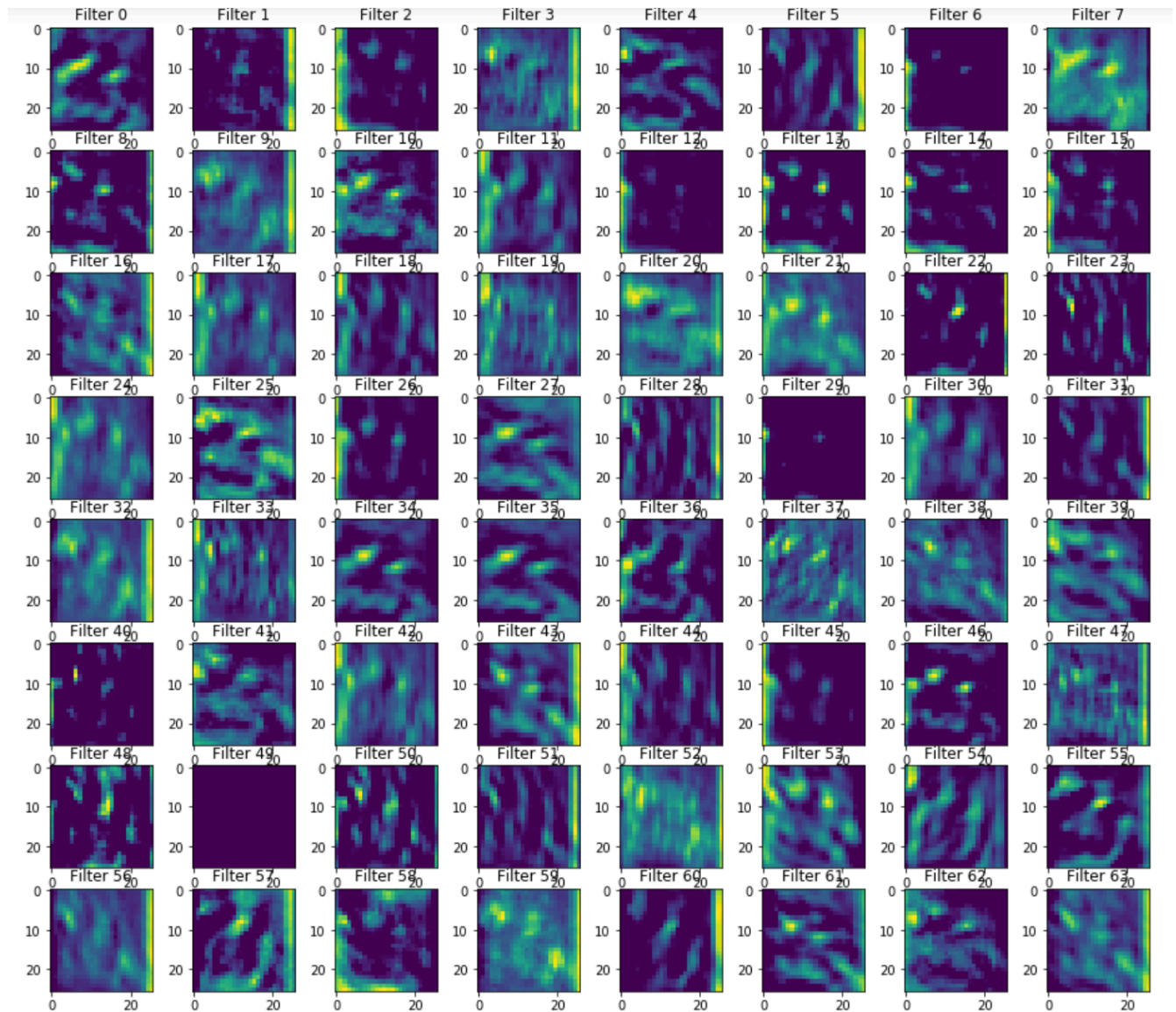
4. Image with prediction and label, and corresponding activation of the first and second convolution layers







 **First convolution layer: (32 channels)**



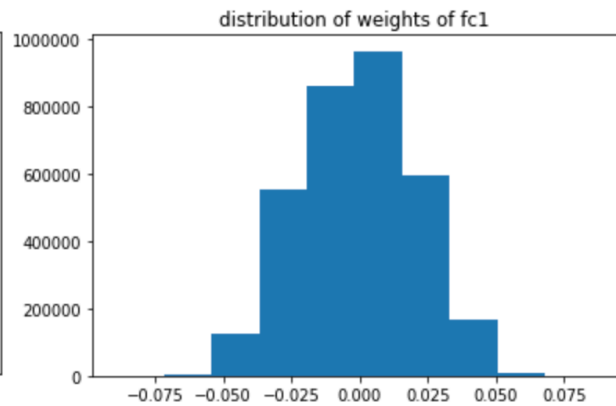
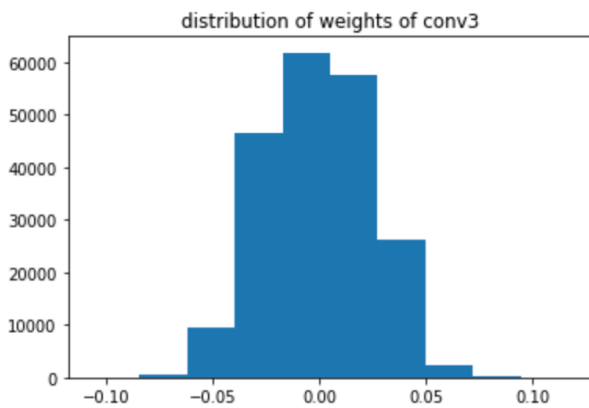
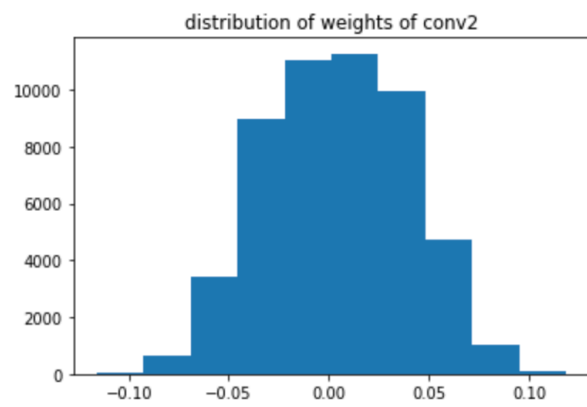
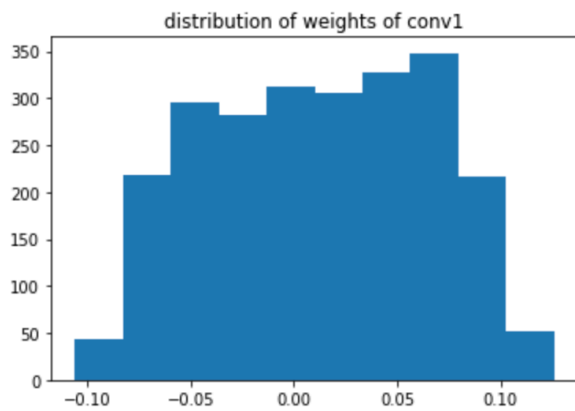
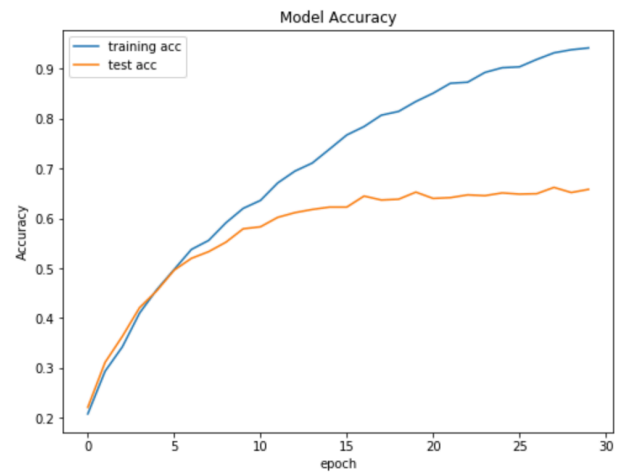
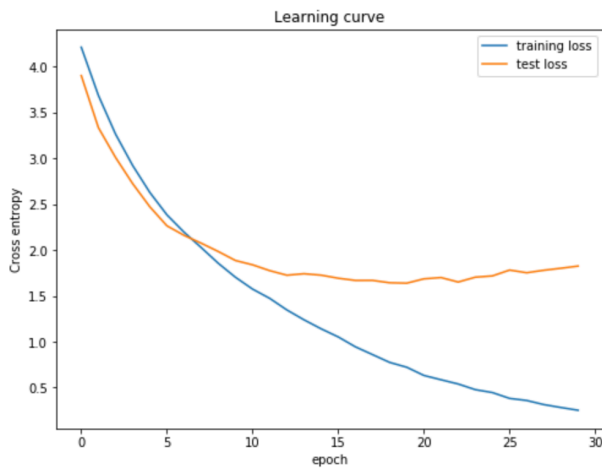
Second convolution layer: (64 channels)

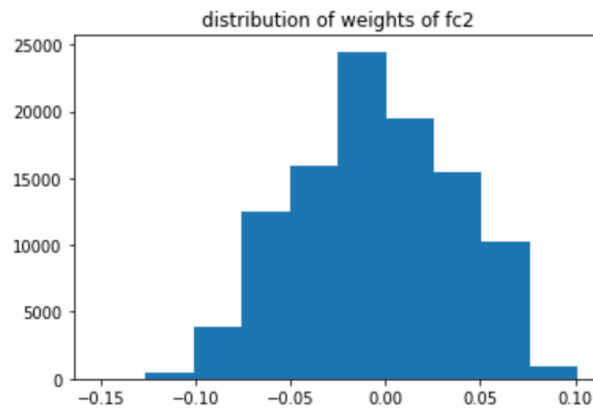


-  The first convolution layer retains more information present in the original picture than the second convolution layer.
-  The activations become **more abstract and less visually** interpretable in the second convolution layer.
-  Deeper layers encode higher-level concepts such as single borders, corners and angles.
-  **Higher level** presentations carry less information about the visual contents of the image, but **more information related to the class** (helpful for classifying) of the image.

5. Train a CNN with L2 regularization (change to file `hw3_105061210_reg.py`)

I write another CNN with L2 regularization in file `hw3_105061210_reg.py`





- ✚ The learning curve here (with L2 regularization) seems better (less over-fitting) than part 3 (without L2 regularization).
- ✚ The weights here (with L2 regularization) are closer to 0 than part 3 (without L2 regularization). This is reasonable since L2 regularization adds another term (weights penalty) to the loss function. As a result, when we minimize the loss, we also try to **get as smaller absolute value of the weights** as possible.

6. Ways to solve over-fitting

- ✚ **dropout** (already used in all previous CNN model):

A dropout layer randomly removes some nodes in the network with all of their incoming and outgoing connections. It can be viewed as a form of ensemble models, and every node may be more insensitive to the weights of other nodes. Dropout makes each node less specialized; the overall model still learns the features.

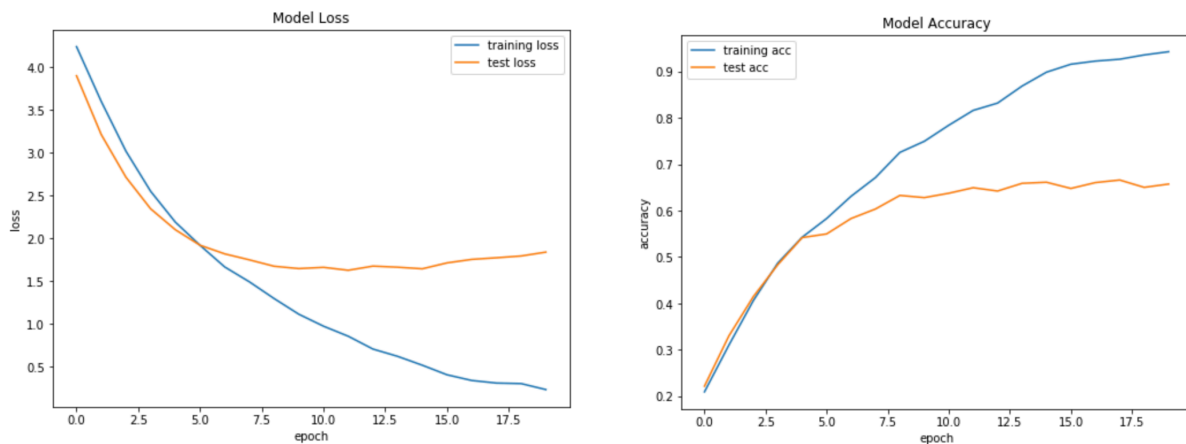
- ✚ **regularization** (already used in part 5.):

L2 regularization penalizes on the square value of all weights. It tends to make all the weights become closer to zero (some weights that do not used may be turned to zero).

- ✚ **early stopping** (I try epoch = 20):

stop before performance of the model on the validation set becomes worse.

The loss curve here (epoch = 20) on validation set is better than part 3. (epoch = 30)



data augmentation (do augmentation in file `data_augmentation.py` and training in file `hw3_105061210_aug.py`):

Data augmentation **synthesizes more training data** based on the original training dataset.

Common ways are: adding noise, flipping, shifting up, shifting down, shifting left, shifting right, rotating and cropping.

I write 'adding noise, flipping, shifting up, shifting down, shifting left, shifting right' in `data_augmentation.py` but only use adding noise and flipping when training due to the lack of computation resources.

dropout (0.5) + L2 regularization (parameter bigger than part 5.) + data augmentation (add noise and flip)

