VE445 INTRO. TO MACHINE LEARNING

REPORT OF
LAB 2

CNN

NAME AND ID

Sijun ZHANG 516370910155

Professor: Bo Yuan

April 19, 2019

1 Introduction

In the last lab, we have implemented another machine learning model, Convolution Neural Network. In lab 2, we have generated image identification models on two image data sets.

1.1 Convolution Layers

The convolution defined for 2-dimension space is different from the one for 1-dimension space. Given I as an image and K a matrix known as convolution kernel, the convolution between the image and the kernel is given by:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

1.2 Activation function

Like BP activation function to increase the generalization ability. The common activation function includes sigmoid function, tanh function and ReLU function.

In this Lab, I have only implement ReLU function within the CNN model layers.

1.3 Dropout

A neural network with complicated structure and plenty of full-connected layers tends to overfit. To avoid of overfitting, we will use the dropout method to reduce the connection between layers. The key idea is to randomly drop units (along with their connections) from the neural network during training.

In the AlexNet and the simple net used to process cifar10 data set, the dropout processes only exist in the two hidden layers of fully connected part to simplify the training process and actually the dense part has the most parameters.

1.4 Pooling

Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer.

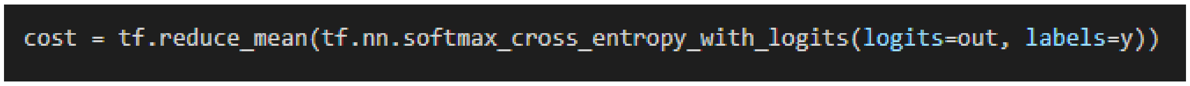
In this Lab, I have only implement *max_pool* function within the CNN model layers.

1.5 One-hot

We have transform the numeric label data in to the data table size of $[data\_num, classes\_num]$

1.6 Softmax and Loss Function

In this part we use the inner-built function of tensorflow to stimulate the cross entropy and softmax.



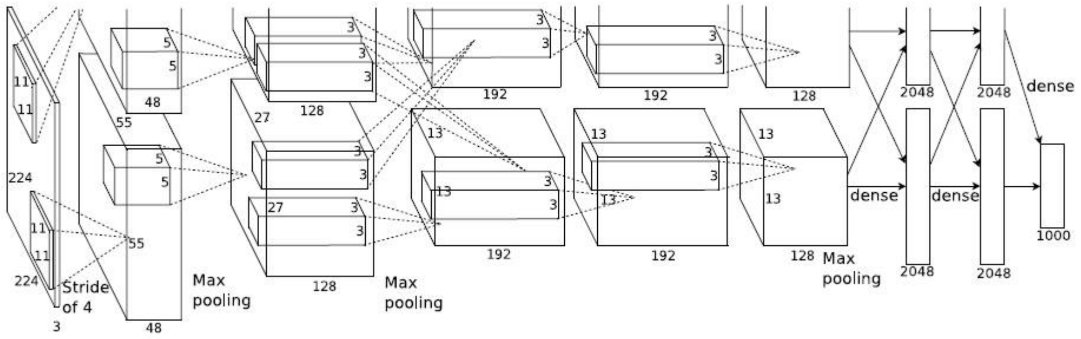
```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=y))
```

2 Structure of CNN Model

2.1 Mnist

To process the mnist data set, I have introduced the traditional CNN model - AlexNet to improve the performance of idenfication model, which is consist of 5 convolution, max-pooling and lrn layers and 3 3 fully connected layers.

The basic structure of the Alexnet is shown in the following graph

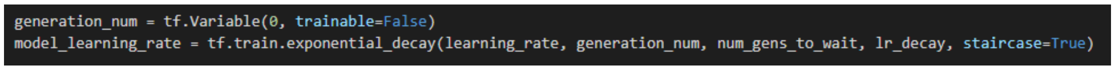


```
weights = {
    'wc1': tf.Variable(tf.random_normal([11, 11, 1, 64])),
    'wc2': tf.Variable(tf.random_normal([5, 5, 64, 192])),
    'wc3': tf.Variable(tf.random_normal([3, 3, 192, 384])),
    'wc4': tf.Variable(tf.random_normal([3, 3, 384, 384])),
    'wc5': tf.Variable(tf.random_normal([3, 3, 384, 256])),
    'wd1': tf.Variable(tf.random_normal([4*4*256, 4096])),
    'wd2': tf.Variable(tf.random_normal([4096, 4096])),
    'out': tf.Variable(tf.random_normal([4096, 10]))
}
biases = {
    'bc1': tf.Variable(tf.random_normal([64])),
    'bc2': tf.Variable(tf.random_normal([192])),
    'bc3': tf.Variable(tf.random_normal([384])),
    'bc4': tf.Variable(tf.random_normal([384])),
    'bc5': tf.Variable(tf.random_normal([256])),
    'bd1': tf.Variable(tf.random_normal([4096])),
    'bd2': tf.Variable(tf.random_normal([4096])),
    'out': tf.Variable(tf.random_normal([n_classes]))}
```

Each Max Pool select $[2, 2]$ part in 2-D image part and also has the step of $[2, 2]$, which exist in the 1st, the 2nd and the 5th convolution layers.

2.1.1 Learning Rate

In this part, we have implemented auto adjusted learning rate, which is declining according to the increasing Generation number. The inner-built tensorflow function provide this part.



```
generation_num = tf.Variable(0, trainable=False)
model_learning_rate = tf.train.exponential_decay(learning_rate, generation_num, num_gens_to_wait, lr_decay, staircase=True)
```

2.2 Cifar10

To accelerate the training speed of CNN model in Cifar10, I simplify the CNN model which can be expressed in the following part

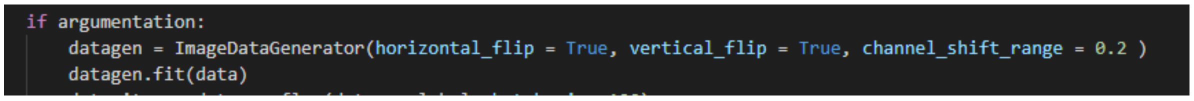
```
weights = {
    'wc1': tf.Variable(tf.random_normal([3, 3, 3, 64])),
    'wc2': tf.Variable(tf.random_normal([3, 3, 64, 128])),
    'wc3': tf.Variable(tf.random_normal([3, 3, 128, 256])),
    'wd1': tf.Variable(tf.random_normal([2*2*256, 1024])),
    'out': tf.Variable(tf.random_normal([1024, 10]))
}
biases = {
    'bc1': tf.Variable(tf.random_normal([64])),
    'bc2': tf.Variable(tf.random_normal([128])),
    'bc3': tf.Variable(tf.random_normal([256])),

    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([n_classes]))}
```

Each Max Pool select $[2, 2]$ part in 2-D image part in each channel and has the step of $[3, 3]$, which exist behind all convolution layers.

2.3 Data augmenting

As it is hard to generalize the cifar10 classifying problem within a relative small training data set, so I have introduced the keras image data generator part to augment the training data and added some variance to the training input by flipping the images vertically and horizontally and shifting the channel of the images.

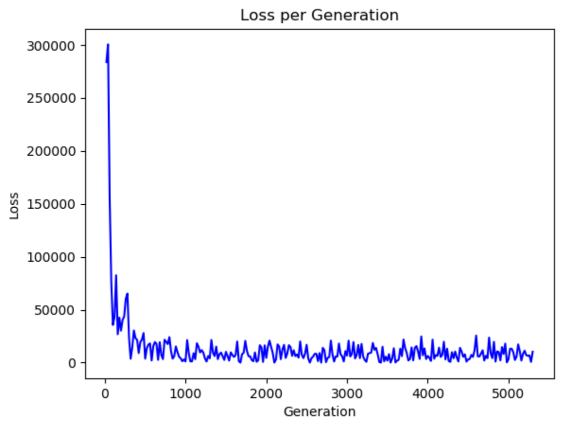


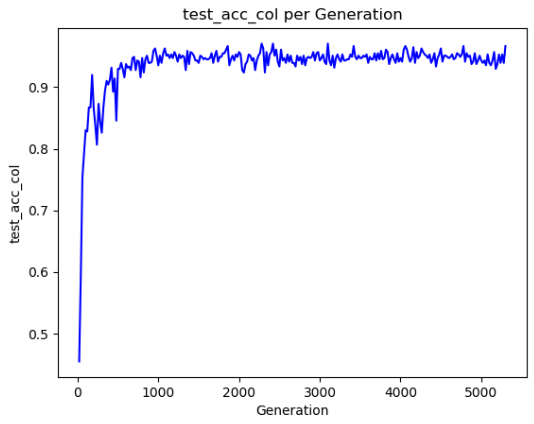
```
if argumentation:
    datagen = ImageDataGenerator(horizontal_flip = True, vertical_flip = True, channel_shift_range = 0.2 )
    datagen.fit(data)
```

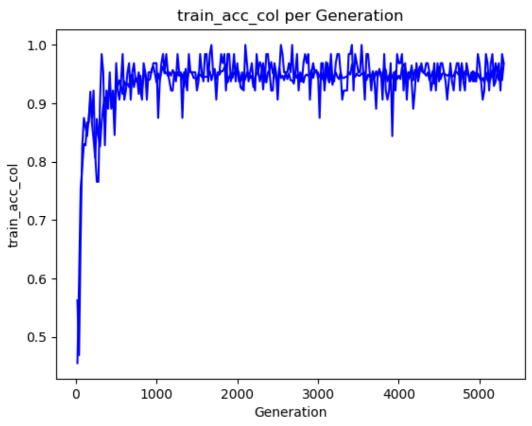
3 Result

3.1 Mnist

We finally reached 96.67% test accuracy after 5300 generations with batch size = 64.

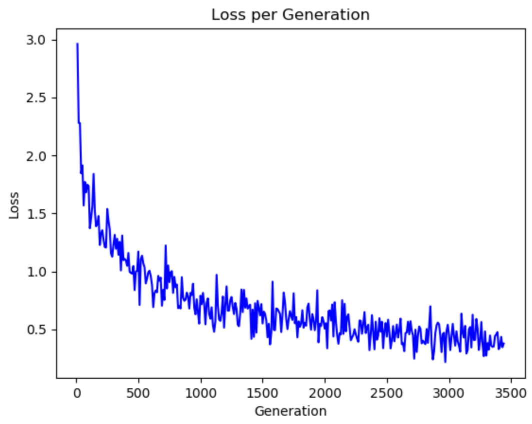


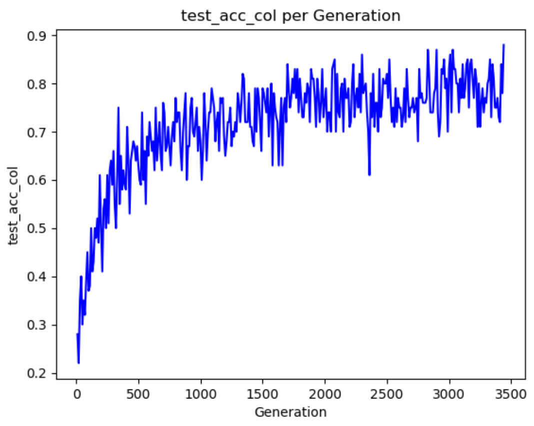


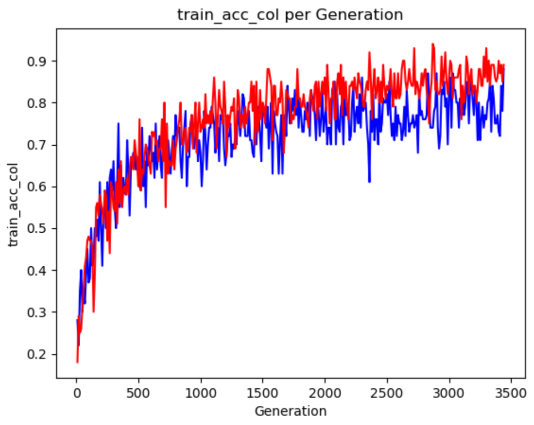


3.2 Cifar10

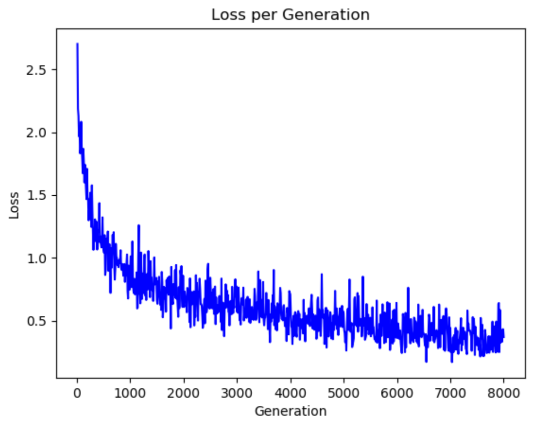
We finally reached 87.4% test accuracy after 3400 generations with batch size = 100.

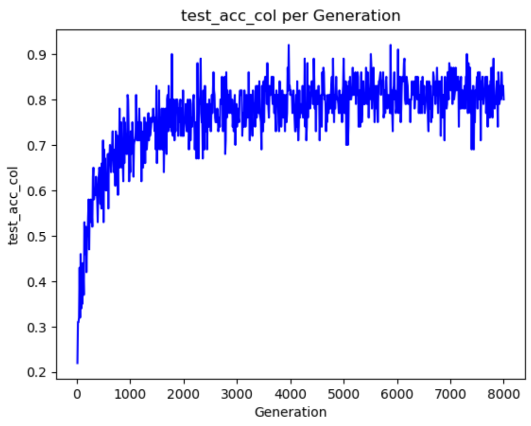


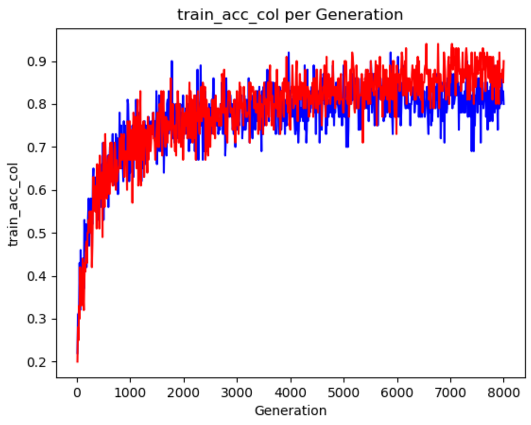




after augment the training data set, the accuracy is not significantly increased, but the rate of joggle in the curve of test accuracy is some how reduced.







4 Appendix

4.1 NN_simpleNet_cifar10.py

```
#coding = utf-8
from __future__ import print_function
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.python.framework import ops
from keras.preprocessing.image import ImageDataGenerator
ops.reset_default_graph()

batch_size = 100
step = 0
train_iter = 400000
```

```
traget_acc = 0.84
traget_step = 3000

argumentation = False

display_step = 10
def one_hot(label):
    l = len(label)
    out = np.zeros([l, 10])
    for i in range (l):
        ind = label[i]
        out[i][ind] = 1

    return out

from keras.datasets.cifar10 import load_data

(data,label),(test_data,test_label) = load_data()


data_size = len(data)

data = data / 255.0
data = data.astype(np.float32)
label = one_hot(label)
label = label.astype(np.int32)


test_size = len(test_data)
test_data = test_data / 255.0
test_data = test_data.astype(np.float32)
test_label = one_hot(test_label)

test_label = test_label.astype(np.int32)


input_x = tf.placeholder(dtype=tf.float32, shape=[None, 32, 32, 3])
y = tf.placeholder(dtype=tf.float32, shape=[None, 10])
keep_prob = tf.placeholder(tf.float32)
is_traing = tf.placeholder(tf.bool)

if argumentation:
    datagen = ImageDataGenerator(horizontal_flip = True,
                                 vertical_flip = True, channel_shift_range = 0.2 )
    datagen.fit(data)
    data_iter = datagen.flow(data, y=label, batch_size=100)
    train_iter = train_iter*2
    traget_acc += 0.03
    traget_step = traget_step*2



####conv1
W1 = tf.Variable(tf.truncated_normal([3, 3, 3, 64], dtype=tf.float32, stddev=5e-2))
conv_1 = tf.nn.conv2d(input_x, W1, strides=(1, 1, 1, 1), padding="VALID")
print(conv_1)
bn1 = tf.layers.batch_normalization(conv_1, training=is_traing)
relu_1 = tf.nn.relu(bn1)
print(relu_1)

pool_1 = tf.nn.max_pool(relu_1, strides=[1, 2, 2, 1], padding="VALID", ksize=[1, 3, 3, 1])
print(pool_1)

####conv2
W2 = tf.Variable(tf.truncated_normal(shape=[3, 3, 64, 128], dtype=tf.float32, stddev=5e-2))
conv_2 = tf.nn.conv2d(pool_1, W2, strides=[1, 1, 1, 1], padding="SAME")
print(conv_2)
bn2 = tf.layers.batch_normalization(conv_2, training=is_traing)
relu_2 = tf.nn.relu(bn2)
print(relu_2)
pool_2 = tf.nn.max_pool(relu_2, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")
print(pool_2)

####conv3

W3 = tf.Variable(tf.truncated_normal(shape=[3, 3, 128, 256], dtype=tf.float32, stddev=1e-1))
conv_3 = tf.nn.conv2d(pool_2, W3, strides=[1, 1, 1, 1], padding="SAME")
print(conv_3)
bn3 = tf.layers.batch_normalization(conv_3, training=is_traing)
relu_3 = tf.nn.relu(bn3)
print(relu_3)
pool_3 = tf.nn.max_pool(relu_3, strides=[1, 2, 2, 1], ksize=[1, 3, 3, 1], padding="VALID")
print(pool_3)

#fc1
dense_tmp = tf.reshape(pool_3, shape=[-1, 2*2*256])
print(dense_tmp)
fc1 = tf.Variable(tf.truncated_normal(shape=[2*2*256, 1024], stddev=0.04))
bn_fc1 = tf.layers.batch_normalization(tf.matmul(dense_tmp, fc1), training=is_traing)
dense1 = tf.nn.relu(bn_fc1)
dropout1 = tf.nn.dropout(dense1, keep_prob)
```

```
#fc2
fc2 = tf.Variable(tf.truncated_normal(shape=[1024, 10], stddev=0.04))
out = tf.matmul(dropout1, fc2)
print(out)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=y))
optimizer = tf.train.AdamOptimizer(0.01).minimize(cost)



correct_pred = tf.equal(tf.argmax(out, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

init = tf.initialize_all_variables()
saver = tf.train.Saver()


loss_col = []
train_acc_col = []
test_acc_col = []
step_jump_col = []
flag = False

with tf.Session() as sess:
    sess.run(init)
    # saver.restore(sess, "model_tmp/cifar10_demo.ckpt")
    step = 1
    # Keep training until reach max iterations
    while step * batch_size < train_iter:
        step += 1
        rand_index = np.random.choice(data_size, size=batch_size)
        batch_xs = data[rand_index]
        batch_ys = label[rand_index]
        if argumentation:
            batch_xs, batch_ys = data_iter.next()

        # print (batch_xs)
#

        opt, acc, loss = sess.run([optimizer, accuracy, cost],
                                  feed_dict={input_x: batch_xs, y: batch_ys,
                                  keep_prob: 0.6, is_traing: True})
        if step % display_step == 0:
            print ("Generation: " + str(step) + ", Minibatch Loss= " +
                   "{:.6f}".format(loss) + ",Training Accuracy= " + "{:.5f}".format(acc))

            # pred_out = sess.run(out[0], feed_dict={input_x: batch_xs, y: batch_ys,
                    keep_prob: 1.0, is_traing:True})
            # print ('y_out is ' + str( batch_ys[0]) )
            # print ("Pred is " + str(pred_out))

            rand_index = np.random.choice(test_size, size=batch_size)
            d = test_data[rand_index]
            l = test_label[rand_index]
            acc_test = sess.run(accuracy, feed_dict={input_x: d, y: l, keep_prob: 1.0,
                                is_traing: True})
            step_jump_col.append(step)
            loss_col.append(loss)
            test_acc_col.append(acc_test)
            train_acc_col.append(acc)
            print ("Testing Accuracy:", acc_test)
            flag = False
            if (acc_test >= traget_acc):
                rand_index = np.random.choice(test_size, size=batch_size)
                rand_x = test_data[rand_index]
                rand_y = test_label[rand_index]
                acc_test = sess.run(accuracy, feed_dict={input_x: rand_x, y: rand_y,
                keep_prob: 1.0, is_traing:True})
                flag = (acc_test >= traget_acc) and (step >= traget_step)
        if (flag):
            break
    mode = "model_tmp/cifar10_model"
    if argumentation:
        mode = "model_tmp_argumented/cifar10_model"
    saver.save(sess, mode)
    print ("Optimization Finished!")

p1 = './fig/cifar10_loss'
p2 = './fig/cifar10_test_acc'
p3 = './fig/cifar10_train_acc'
if argumentation:
    p1 = p1 + '_argumentation'
    p2 = p2 + '_argumentation'
    p3 = p3 + '_argumentation'

plt.plot(step_jump_col, loss_col, 'b-')
plt.title('Loss per Generation')
plt.xlabel('Generation')
plt.ylabel('Loss')
plt.savefig(p1)
plt.show()

plt.plot(step_jump_col, test_acc_col, 'b-')
plt.title('test_acc_col per Generation')
plt.xlabel('Generation')
plt.ylabel('test_acc_col')
plt.savefig(p2)



plt.plot(step_jump_col, train_acc_col, 'r-')
plt.title('train_acc_col per Generation')
plt.xlabel('Generation')
plt.ylabel('train_acc_col')
plt.savefig(p3)
```

4.2 NN_Alexnet_mnist.py

```
#coding = utf-8
from __future__ import print_function

import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.python.framework import ops
ops.reset_default_graph()



learning_rate = 0.002
training_iters = 400000
batch_size = 64
display_step = 20


n_input = 784
n_classes = 10
dropout = 0.75

###
from keras.datasets.mnist import load_data

(data,label),(test_data,test_label) = load_data()


data_size = len(data)
data = tf.cast(data, np.float32)
data = tf.nn.l2_normalize(data, dim = 1)
data = tf.reshape(data, [data_size, n_input])
```

```
label = tf.one_hot(label, n_classes)
label = tf.cast(label, np.int32)

test_size = len(test_data)
test_data = tf.cast(test_data, tf.float32)
test_data = tf.nn.l2_normalize(test_data, dim = 1)
test_data = tf.reshape(test_data, [test_size, n_input])
test_label = tf.one_hot(test_label, n_classes)
test_label = tf.cast(test_label, tf.int32)



###

lr_decay = 0.1
num_gens_to_wait = 480.

x = tf.placeholder(tf.float32, [None, n_input])
y = tf.placeholder(tf.float32, [None, n_classes])
keep_prob = tf.placeholder(tf.float32)

def conv2d(name, l_input, w, b):
        return tf.nn.relu(tf.nn.bias_add(tf.nn.conv2d(l_input, w,
                        strides=[1, 1, 1, 1], padding='SAME'),b), name=name)

def max_pool(name, l_input, k):
        return tf.nn.max_pool(l_input, ksize=[1, k, k, 1], strides=[1, k, k, 1],
                            padding='SAME', name=name)

def norm(name, l_input, lsize=4):
        return tf.nn.lrn(l_input, lsize, bias=1.0, alpha=0.001 / 9.0, beta=0.75, name=name)

weights = {
    'wc1': tf.Variable(tf.random_normal([11, 11, 1, 64])),
    'wc2': tf.Variable(tf.random_normal([5, 5, 64, 192])),
    'wc3': tf.Variable(tf.random_normal([3, 3, 192, 384])),
    'wc4': tf.Variable(tf.random_normal([3, 3, 384, 384])),
    'wc5': tf.Variable(tf.random_normal([3, 3, 384, 256])),
    'wd1': tf.Variable(tf.random_normal([4*4*256, 4096])),
    'wd2': tf.Variable(tf.random_normal([4096, 4096])),
    'out': tf.Variable(tf.random_normal([4096, 10]))
}
biases = {
    'bc1': tf.Variable(tf.random_normal([64])),
    'bc2': tf.Variable(tf.random_normal([192])),
    'bc3': tf.Variable(tf.random_normal([384])),
    'bc4': tf.Variable(tf.random_normal([384])),
    'bc5': tf.Variable(tf.random_normal([256])),
    'bd1': tf.Variable(tf.random_normal([4096])),
    'bd2': tf.Variable(tf.random_normal([4096])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}

def alex_net(_X, _weights, _biases, _dropout):
        _X = tf.reshape(_X, shape=[-1, 28, 28, 1])

        conv1 = conv2d('conv1', _X, _weights['wc1'], _biases['bc1'])
        pool1 = max_pool('pool1', conv1, k=2)
        norm1 = norm('norm1', pool1, lsize=4)

        conv2 = conv2d('conv2', norm1, _weights['wc2'], _biases['bc2'])
        pool2 = max_pool('pool2', conv2, k=2)
        norm2 = norm('norm2', pool2, lsize=4)

        conv3 = conv2d('conv3', norm2, _weights['wc3'], _biases['bc3'])
        norm3 = norm('norm3', conv3, lsize=4)

        conv4 = conv2d('conv4', norm3, _weights['wc4'], _biases['bc4'])
        norm4 = norm('norm4', conv4, lsize=4)

        conv5 = conv2d('conv5', norm4, _weights['wc5'], _biases['bc5'])
        pool5 = max_pool('pool5', conv5, k=2)
        norm5 = norm('norm5', pool5, lsize=4)

        dense1 = tf.reshape(norm5, [-1, _weights['wd1'].get_shape().as_list()[0]])
        dense1 = tf.nn.relu(tf.matmul(dense1, _weights['wd1']) + _biases['bd1'], name='fc1')
        dense1 = tf.nn.dropout(dense1, _dropout)

        dense2 = tf.reshape(dense1, [-1, _weights['wd2'].get_shape().as_list()[0]])
        dense2 = tf.nn.relu(tf.matmul(dense1, _weights['wd2']) +
                            _biases['bd2'], name='fc2') # Relu activation
        dense2 = tf.nn.dropout(dense2, _dropout)

        out = tf.matmul(dense2, _weights['out']) + _biases['out']
        return out

pred = alex_net(x, weights, biases, keep_prob)

# loss1 = tf.nn.softmax_cross_entropy_with_logits(logits = pred, labels = y)[0]
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = pred, labels = y))
```

```
generation_num = tf.Variable(0, trainable=False)
model_learning_rate = tf.train.exponential_decay(learning_rate,
                        generation_num, num_gens_to_wait, lr_decay, staircase=True)

optimizer = tf.train.AdamOptimizer(learning_rate=model_learning_rate).minimize(cost)

correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))


init = tf.initialize_all_variables()

saver = tf.train.Saver()

loss_col = []
train_acc_col = []
test_acc_col = []
step_jump_col = []
flag = False

with tf.Session() as sess:
        sess.run(init)
        data = sess.run(data)
        label = sess.run(label)
        test_data = sess.run(test_data)
        test_label = sess.run(test_label)
        step = 1
        # Keep training until reach max iterations
        while step * batch_size < training_iters:
                # batch_xs, batch_ys = mnist.train.next_batch(batch_size)
                rand_index = np.random.choice(data_size, size=batch_size)
                batch_xs = data[rand_index]
                batch_ys = label[rand_index]

                sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys,
                            keep_prob: dropout, generation_num:step})
                if step % display_step == 0:
                        acc = sess.run(accuracy, feed_dict={x: batch_xs, y: batch_ys,
                                    keep_prob: 1.})
                        loss = sess.run(cost, feed_dict={x: batch_xs, y: batch_ys,
                                    keep_prob: 1.})
                        print ("Generation: " + str(step) + ", Batch Loss = " +
                                    "{:.6f}".format(loss) + ", Training Accuracy = " +
                                        "{:.5f}".format(acc))
                        # pred_out = sess.run(pred[0], feed_dict={x: batch_xs, y: batch_ys,
                                        keep_prob: 1.})
```

```
                    # print ("Pred is " + str(pred_out))\
                    # pred_out = sess.run(pred[0], feed_dict={x: batch_xs, y: batch_ys,
                          keep_prob: 1.})

                    # print ('y_out is ' + str( batch_ys[0]) )
                    # print ("Pred is " + str(pred_out))
                    rand_index = np.random.choice(test_size, size=256*2)
                    rand_x = test_data[rand_index]
                    rand_y = test_label[rand_index]
                    acc_test = sess.run(accuracy, feed_dict={x: rand_x, y: rand_y,
                          keep_prob: 1.})
                    print ('Testing Accuracy: ' + str(acc_test) )

                    step_jump_col.append(step)
                    loss_col.append(loss)
                    test_acc_col.append(acc_test)
                    train_acc_col.append(acc)
                    flag = False
                    if (acc_test >= 0.965):
                        rand_index = np.random.choice(test_size, size=256*2)
                        rand_x = test_data[rand_index]
                        rand_y = test_label[rand_index]
                        acc_test = sess.run(accuracy, feed_dict={x: rand_x, y: rand_y,
                              keep_prob: 1.})
                        flag = (acc_test >= 0.965) and (step >= 2500)
                if (flag):
                    break

                step += 1
                # if (step == 2):
            saver.save(sess,'D:\model\mnist_model')

            print ("Optimization Finished!")

    plt.plot(step_jump_col, loss_col, 'b-')
    plt.title('Loss per Generation')
    plt.xlabel('Generation')
    plt.ylabel('Loss')

    plt.savefig('./fig/mnist_loss')
    plt.show()


    plt.plot(step_jump_col, test_acc_col, 'b-')
    plt.title('test_acc_col per Generation')
    plt.xlabel('Generation')
```

```
plt.ylabel('test_acc_col')
plt.savefig('./fig/mnist_test_acc')


plt.plot(step_jump_col, train_acc_col, 'b-')
plt.title('train_acc_col per Generation')
plt.xlabel('Generation')
plt.ylabel('train_acc_col')
plt.savefig('./fig/mnist_train_acc')
```