CSCI677 Homework5

1 Introduction

This homework is about CNN. We are going to create a CNN network to train and test object classification. The dataset we are going to use is CIFAR-100. CIFAR-100 contains 100 mutually exclusive classes. Each one has 32*32 RGB image. The network we are going to use is LeNet-5 CNN network. The package we are going to use is TenserFlow.

2 Architecture

According to the instruction. We should create a new function to make the CNN network. Follows are codes to create the CNN network.

First layer has six, 5x5 convolution filters, stride =1, each followed by a max pooling layer of 2x2 with stride = 2.

```
## create convolution 1
conv1_w = tf.Variable(tf.truncated_normal(shape= [5,5,3,6], mean= mu, stddev=sigma))
conv1_b = tf.Variable(tf.zeros(6))
conv1 = tf.nn.conv2d(x, conv1_w, strides= [1,1,1,1], padding= 'VALID') + conv1_b

conv1 = tf.nn.relu(conv1)

pool_1 = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides= [1,2,2,1], padding="VALID")
```

Second convolution layer has 16, 5x5 convolution filters, stride =1, each followed by 2x2 max pooling with stride = 2.

```
## create convolution 2
conv2_w = tf.Variable(tf.truncated_normal(shape=[5, 5, 6, 16], mean=mu, stddev=sigma))
conv2_b = tf.Variable(tf.zeros(16))
conv2 = tf.nn.conv2d(pool_1, conv2_w, strides=[1, 1, 1, 1], padding='VALID') + conv2_b

conv2 = tf.nn.relu(conv2)

pool_2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding="VALID")
```

Next comes a fully connected layer of dimensions 120 followed by another fully connected layer of dimension 84. A softmax layer of dimension 100 provides the classification probabilities. All activation units should be ReLU.

```
## create fully complete:
fc1 = flatten(pool_2)

## fc 1
fc1_w = tf.Variable(tf.truncated_normal(shape=(400, 120), mean=mu, stddev=sigma))
fc1_b = tf.Variable(tf.zeros(120))
fc1 = tf.matmul(fc1, fc1_w) + fc1_b
fc1 = tf.nn.relu(fc1)

## fc 2
fc2_w = tf.Variable(tf.truncated_normal(shape=(120, 84), mean=mu, stddev=sigma))
fc2_b = tf.Variable(tf.zeros(84))
fc2 = tf.matmul(fc1, fc2_w) + fc2_b
```

```
fc2 = tf.nn.relu(fc2)


## fc 3
fc3_w = tf.Variable(tf.truncated_normal(shape=(84, CLASS), mean=mu, stddev=sigma))
fc3_b = tf.Variable(tf.zeros(CLASS))
logits = tf.matmul(fc2, fc3_w) + fc3_b
```

3 Training

In this section, we want to know how to train the network using the given training data. I used a mini-batch size of 128, learning rate of 0.001 and the ADAM optimizer. EPUCHs are set as 50. The operation is shown as follows:

```
x = tf.placeholder(tf.float32, (None, 32,32,3))
y = tf.placeholder(tf.int32, (None))

# ??????????????????????????
one_hot_y = tf.one_hot(y,CLASS)

rate = 0.001

logits = LeNet(x)

# Create a training pipline that uses the model to classify data

loss_operation = tf.losses.softmax_cross_entropy(one_hot_y, logits)
optimizer = tf.train.AdamOptimizer(learning_rate=rate)          # adam optimizer
training_operation = optimizer.minimize(loss_operation)

# Evaluate how well the loss and accuracy of the model for a given dataset
correct_prediction = tf.equal(tf.argmax(logits,1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# yPred = tf.nn.softmax(tf.matmul(x,W) + b)
# prediction = tf.argmax(cross_entropy, 1)
prediction=tf.argmax(logits,1)
top_k_values, top_k_indices = tf.nn.top_k(logits, k=K)

names_to_values, names_to_updates = slim.metrics.aggregate_metric_map({
    'Accuracy': slim.metrics.streaming_accuracy(prediction, y),
    'Recall_5': slim.metrics.streaming_recall_at_k(
        logits, y, 5), })
```

When we have these operations, we can train our CNN network and test the validation data and test data. Figure 1 is the loss function plotting of training set and validation set. Figure 2 is the accuracy of training dataset and validation dataset. Blue lines are training dataset, and yellow lines are validation dataset. You can see that the training dataset had higher and higher accuracy, but the validation dataset was stable and had a lower accuracy. This experiment was overfitting. After I got this result, I found that another reason was that the validation set is not extracted randomly.

```
X_validation = X_train[:10000]
y_validation = y_train[:10000]
```

The validation set is extracted as top 10000 of training dataset.

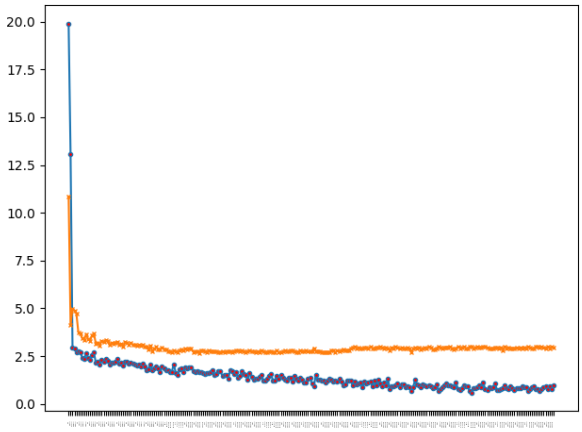


Figure 1

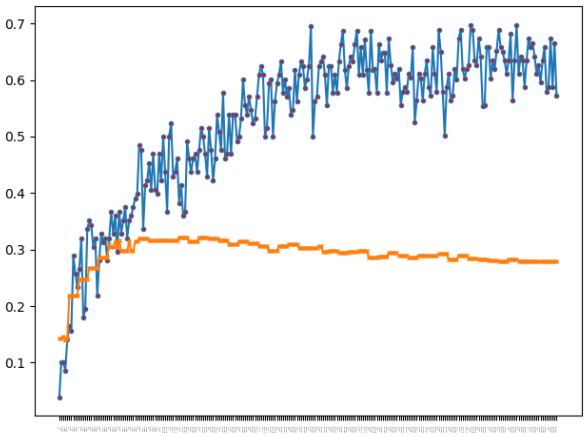


Figure 2

Note: the x labels are “Epoch--Steps”. Because there are too many x labels, so it doesn’t show well. I printed them out, they are like followes:

['0--0', '0--9600', '0--19200', '0--28800', '0--38400', '1--0', '1--9600', '1--19200', '1--28800', '1--38400', '2--0', '2--9600', …… ….'48--19200', '48--28800', '48--38400', '49--0', '49--9600', '49--19200', '49--28800', '49--38400']

For example, '*49--28800*' means 49 epoches, 28800 steps.

4 Preprocessing

In this section, we should subtract mean of the images in the test set for pre-processing. Here are the codes:

```
def imageProcessing(list):
    list = np.mean(np.array(list),axis=0)
    print list.shape
    list = np.reshape(list,(32,32,3),(0, 1, 2))
    print list.shape
    return list
def unboxImg(list):
    mean = imageProcessing(list)
    def reshapeImg(a):
        xx = np.array(a)
        xx = np.reshape(xx,(32,32,3),(0, 1, 2))
        xx = xx - mean
        return xx
    list = map(reshapeImg,list)
    return list
```

I used the “map” function to reshape every image. Every row in this dataset is (3072,1), We should calculate the mean value using np.mean, and than use reshape to change the row from (3072,1) to (32,32,3). Finally, every row should subtract the mean value.

5 Augmentation

In this section, we should process the image to make the progress more accurate. What I did was different from the instructions. I did as follows: enlarge the image by 10%, crop the 90% of the image, and then flip the image and do this again.

Here are the codes:

```
def augmentation(X_data):
    X_return = []

    for i in range(len(X_data)):
        img = cv2.resize(X_data[i], None, fx=1.1, fy=1.1)
        crop_img = img[0:32, 0:32]
        # cv2.imshow("cropped", crop_img)
        img = cv2.flip(crop_img, -1)

        # again
        img = cv2.resize(img, None, fx=1.1, fy=1.1)
        crop_img = img[0:32, 0:32]
        X_return.append(crop_img)

    return X_return
```

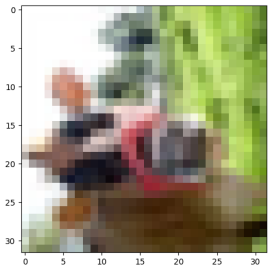


Figure a

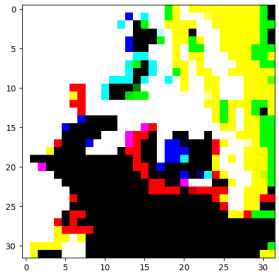


Figure b

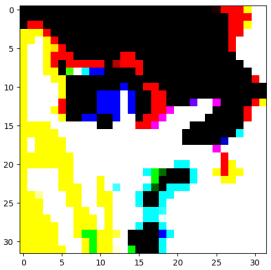


Figure c

Figure a is the original picture, it's an ox. Figure b is the picture subtracting the mean value, and the figure c is the image after augmentation.

6 Test Results

In this section, we should show the confusion matrix.

Figure 3 is confusion matrix of super class. Figure 4 is the confusion matrix of fine class.

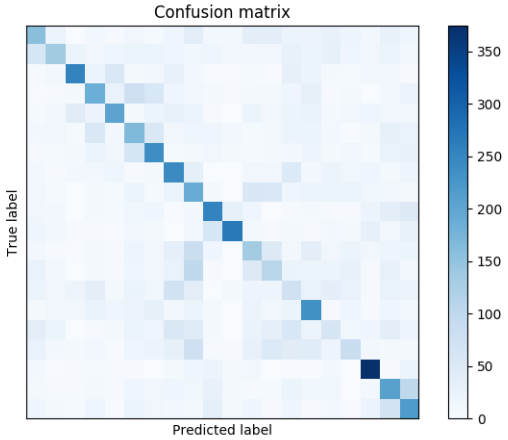


Figure 3

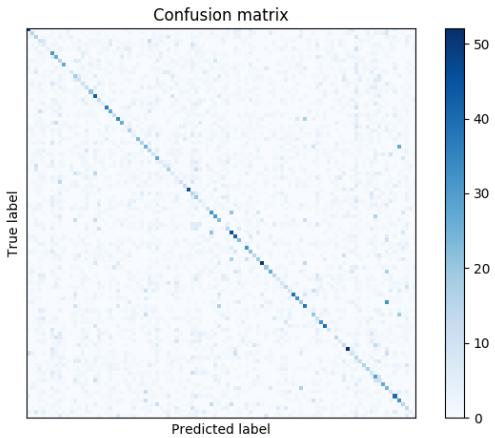


Figure 4

As the instruction said, we also should not only test the rank 1 but also the rank 5. Tensorflow has the operations to predict 5 labels. What we should do is to see whether the real label was in the predict 5 labels. Here are the codes.

```
def compareTop5 (y_data, tkv):
    print "len"
    print len(y_data)
    print len(tkv)
    num=0
    for x in range(0,len(y_data)):
        if y_data[x] in tkv[x]:
            num = num +1
    return num

tkv, tki = sess.run([top_k_indices, top_k_values], feed_dict={x: X_data, y: y_data})
num = compareTop5(y_data,tkv)
accuracyTop5 = float(num)/num_example
```

The rank-1 and rank-5 for fine class and super class were tested and the accuracy is shown in table 1. (NOTE: mini-batch size of 128, learning rate of 0.001 and the ADAM optimizer. EPUCHs are set as 50)

Table 1

Accuracy	Rank-1	Rank-5
Fine class	18.3%	38.0%
Super class	29.3%	43.4%

Calculate the classification accuracy for each class and super class. We can plot the number of predictions on the confusion matrix. However, for the fine class, it is hard to read. I printed them out. We can know how many predictions it has, so that we can calculate the accuracy of each class.

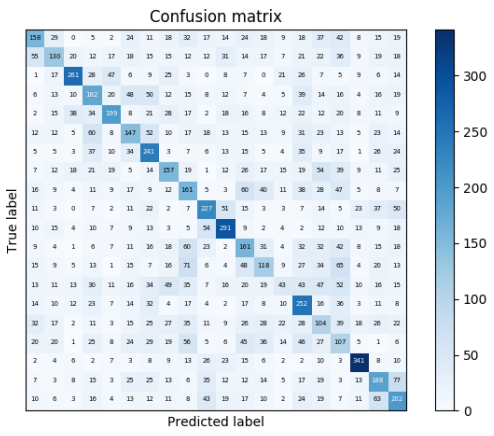


Figure 5

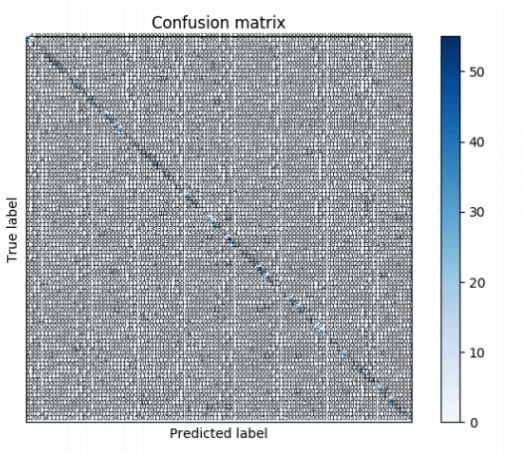
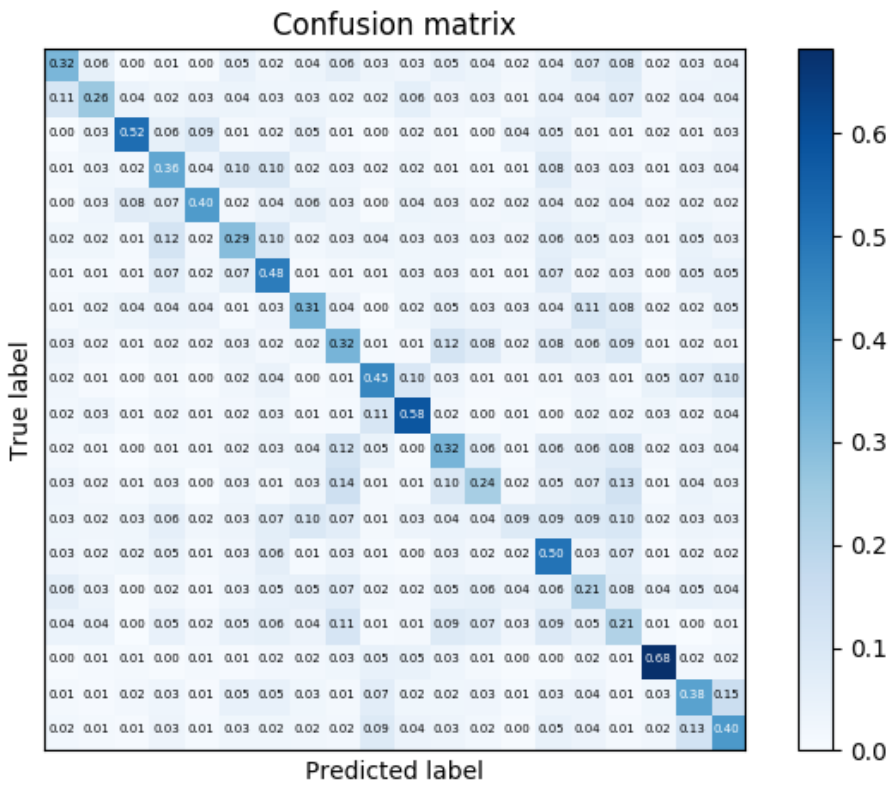


Figure6

Plot confusion matrix of super class
For example, for the class No. 18. The accuracy is 68%



Plot confusion matrix of fine class

Number:

```
[[65  2  1 ...  0  0  0]
 [ 2 30  0 ...  0  0  1]
 [ 6  1 16 ...  1  5  0]
 ...
 [ 0  0  1 ... 13  1  1]
 [ 2  0  7 ...  1  9  1]
 [ 1  0  0 ...  0  0  8]]
```

Accuracy:

```
[[0.65 0.02 0.01 ... 0.   0.   0.  ]
 [0.02 0.3  0.   ... 0.   0.   0.01]
 [0.06 0.01 0.16 ... 0.01 0.05 0.  ]
 ...
 [0.   0.   0.01 ... 0.13 0.01 0.01]
 [0.02 0.   0.07 ... 0.01 0.09 0.01]
 [0.01 0.   0.   ... 0.   0.   0.08]]
```

Accuracy, only show the number on the diagonal:

[0.65, 0.3, 0.16, 0.04, 0.03, 0.16, 0.2, 0.32, 0.25, 0.36, 0.22, 0.02, 0.23, 0.16, 0.14, 0.05, 0.17, 0.37, 0.04, 0.12, 0.43, 0.48, 0.1, 0.45, 0.34, 0.17, 0.05, 0.09, 0.34, 0.21, 0.34, 0.24, 0.16, 0.15, 0.04, 0.08, 0.13, 0.24, 0.1, 0.21, 0.09, 0.43, 0.2, 0.15, 0.04, 0.13, 0.12, 0.3, 0.39, 0.35, 0.05, 0.14, 0.7, 0.33, 0.3, 0.04, 0.19, 0.12, 0.31, 0.3, 0.67, 0.22, 0.36, 0.16, 0.06, 0.05, 0.11, 0.15, 0.51, 0.52,

0.29, 0.31, 0.12, 0.11, 0.15, 0.46, 0.54, 0.13, 0.13, 0.21, 0.03, 0.21, 0.41, 0.12, 0.11, 0.41, 0.38, 0.22, 0.13, 0.18, 0.11, 0.34, 0.14, 0.08, 0.59, 0.39, 0.19, 0.13, 0.09, 0.08]

7 System variation

In this section, we should try to change the variations to check if the performance of the CNN can be improved.

I tried two different parameters

1. Change the number of filters in the early layers
2. Change the size of the fully connected layers.

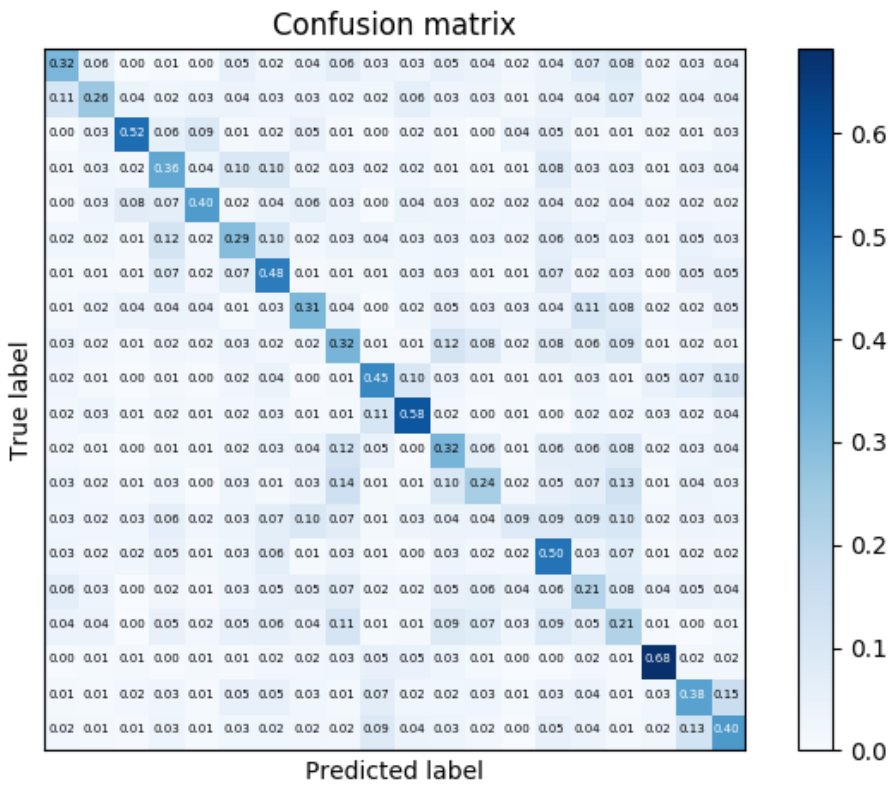
First of all, we need a baseline. I used the original parameters. Fully connected layers are 120, 84. Filters are 6, and 16.

	Baseline	Fully connected (240, 168) Filter = (6, 16)	Fully connected (60, 42) Filter = (6, 16)	Filter = (12, 32) Fully connected (120, 84)	Filter = (3, 8) Fully connected (120, 84)
Accuracy	31.3%	25.0%	10.0%	38.0%	27.2%

First, fix Filter as the original parameters. Two times the fully connection will not get a higher accuracy. Dividing the fully connection is a disaster, the accuracy will be 10%. Second, fix fully connected layers. If we double the filter layers, it really can increase the accuracy. However, if we divide the filter layers, it can decrease the accuracy.

8 Analysis and Discussion

Here are some interesting observations. Show as the confusion matrix. Class No. 18 has the highest accuracy. The class accuracy is 68%. It means this class is much easier to be distinguished from other classes.



However, there are also some images are hard to be distinguished from each other. For example, shown as below. You cannot tell the figure a is fruit or flower. If its fruit, it is hard to say it is an apple or pear. For figure b, you cannot tell if it is a fish or flower. For figure c, you cannot tell it is tree or vegetable.

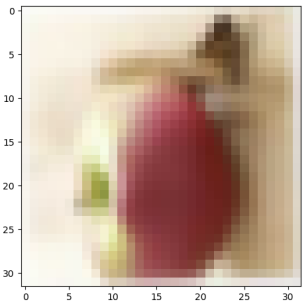


Figure a

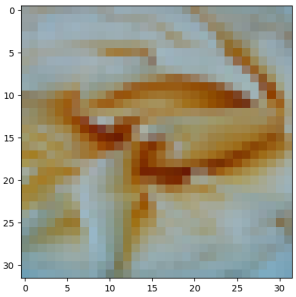


Figure b

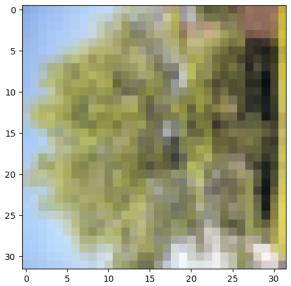


Figure c

According to the system variations, if we want to improve the parameters, we can change the filter layers. This parameter should be paid more attention to.

9 CODE

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
import numpy as np
import cv2
import random
import numpy as np
```

```
import matplotlib.pyplot as plt
from tensorflow.contrib.layers import flatten
from sklearn.metrics import confusion_matrix
import itertools
import tensorflow.contrib.slim as slim

########################################################################
# Reading Dataset:
def unpickle(file):
    import cPickle
    with open(file, 'rb') as fo:
        dict = cPickle.load(fo)
    return dict

def imageProcessing(list):
    list = np.mean(np.array(list),axis=0)
    print list.shape
    list = np.reshape(list,(32,32,3),(0, 1, 2))
    print list.shape
    return list

def unboxImg(list):
    mean = imageProcessing(list)
    def reshapeImg(a):
        xx = np.array(a)
        xx = np.reshape(xx,(32,32,3),(0, 1, 2))
        xx = xx - mean
        return xx
    list = map(reshapeImg,list)
    return list

dictionary1 = unpickle("./cifar-100-python/train")        # keys: print
dictionary['data']; dictionary['batch_label']; print dictionary['fine_labels']; print
dictionary['coarse_labels']
X_train = unboxImg(dictionary1['data'])
y_train = np.array(dictionary1['fine_labels'])



X_validation = X_train[:10000]
y_validation = y_train[:10000]

X_train = X_train[10000:]
y_train = y_train[10000:]

dictionary2 = unpickle("./cifar-100-python/test")
X_test = unboxImg(dictionary2["data"])
y_test = np.array(dictionary2['fine_labels'])


########################################################################

# check the data size
assert (len(X_train) == len(y_train))
# assert (len(X_validation) == len(y_validation))
assert (len(X_test) == len(y_test))

# check the image size and data set size
print("image shape = {}".format(X_train[0].shape))
print("Training set numbers= {}".format(len(X_train)))
print("Validation set numbers = {}".format(len(X_validation)))
```

```
print("Test set numbers = {}".format(len(X_test)))




print("updated size of train images = {}".format(X_train[0].shape))
print("updated size of validation images = {}".format(X_validation[0].shape))
print("updated size of test images = {}".format(X_test[0].shape))



#
# # visulizat data
# index = random.randint(0, len(X_train))
# image = X_train[index].squeeze()
#
# plt.figure(figsize = (1,1))
# plt.imshow(image, cmap = "gray")
# print (y_train[index])


# shuffle the training data
from sklearn.utils import shuffle
X_train,y_train = shuffle(X_train, y_train)

import tensorflow as tf

EPOCHS = 50
BATCH_SIZE = 128
K=5
CLASS=100
LOSS_X = []
LOSS_Y = []
LOSS_LABEL = []
ACC_X = []
ACC_Y = []
ACC_LABEL = []
COUNT =0

##
def augmentation(X_data):
    X_return = []

    for i in range(len(X_data)):
        img = cv2.resize(X_data[i], None, fx=1.1, fy=1.1)
        crop_img = img[0:32, 0:32]
        # cv2.imshow("cropped", crop_img)
        img = cv2.flip(crop_img, -1)

        # again
        img = cv2.resize(img, None, fx=1.1, fy=1.1)
        crop_img = img[0:32, 0:32]
        X_return.append(crop_img)

    return X_return




# Define the LeNet-5 neural network architecture
```

```
def LeNet(x):
    mu = 0
    sigma = 0.1


    # set the weight : w, b

    ## create convolution 1
    conv1_w = tf.Variable(tf.truncated_normal(shape= [5,5,3,6], mean= mu,
stddev=sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1 = tf.nn.conv2d(x, conv1_w, strides= [1,1,1,1], padding= 'VALID') + conv1_b

    conv1 = tf.nn.relu(conv1)

    pool_1 = tf.nn.max_pool(conv1, ksize=[1,2,2,1], strides= [1,2,2,1],
padding="VALID")



    ## create convolution 2
    conv2_w = tf.Variable(tf.truncated_normal(shape=[5, 5, 6, 16], mean=mu,
stddev=sigma))
    conv2_b = tf.Variable(tf.zeros(16))
    conv2 = tf.nn.conv2d(pool_1, conv2_w, strides=[1, 1, 1, 1], padding='VALID') +
conv2_b

    conv2 = tf.nn.relu(conv2)

    pool_2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding="VALID")



    ## create fully complete:
    fc1 = flatten(pool_2)

    ## fc 1
    fc1_w = tf.Variable(tf.truncated_normal(shape=(400, 120), mean=mu, stddev=sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1 = tf.matmul(fc1, fc1_w) + fc1_b
    fc1 = tf.nn.relu(fc1)

    ## fc 2
    fc2_w = tf.Variable(tf.truncated_normal(shape=(120, 84), mean=mu, stddev=sigma))
    fc2_b = tf.Variable(tf.zeros(84))
    fc2 = tf.matmul(fc1, fc2_w) + fc2_b
    fc2 = tf.nn.relu(fc2)


    ## fc 3
    fc3_w = tf.Variable(tf.truncated_normal(shape=(84, CLASS), mean=mu, stddev=sigma))
    fc3_b = tf.Variable(tf.zeros(CLASS))
    logits = tf.matmul(fc2, fc3_w) + fc3_b

    return logits


"""
```

```
Create a confusion matrix
"""
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        # print("Normalized confusion matrix")
    # else:
    #     print('Confusion matrix, without normalization')

    # print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),fontsize=5,
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

def confusionM(y_data, y_pred):

    # compute CM
    cm = confusion_matrix(y_data, y_pred)
    plt.figure()
    # plot_confusion_matrix(cm,[0,1,2,3,4,5,6,7,8,9],False)
    plot_confusion_matrix(cm,[],False)

    plt.figure()
    # plot_confusion_matrix(cm,[0,1,2,3,4,5,6,7,8,9],True)
    plot_confusion_matrix(cm,[],True)



'''
x is a placeholder for a batch of input images
y is a placeholder for a batch of output labels
'''
x = tf.placeholder(tf.float32, (None, 32,32,3))
y = tf.placeholder(tf.int32, (None))


one_hot_y = tf.one_hot(y,CLASS)

rate = 0.001
```

```
logits = LeNet(x)

# Create a training pipline that uses the model to classify data

loss_operation = tf.losses.softmax_cross_entropy(one_hot_y, logits)
optimizer = tf.train.AdamOptimizer(learning_rate=rate)          # adam optimizer
training_operation = optimizer.minimize(loss_operation)

# Evaluate how well the loss and accuracy of the model for a given dataset
correct_prediction = tf.equal(tf.argmax(logits,1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# yPred = tf.nn.softmax(tf.matmul(x,W) + b)
# prediction = tf.argmax(cross_entropy, 1)
prediction=tf.argmax(logits,1)
top_k_values, top_k_indices = tf.nn.top_k(logits, k=K)

names_to_values, names_to_updates = slim.metrics.aggregate_metric_map({
    'Accuracy': slim.metrics.streaming_accuracy(prediction, y),
    'Recall_5': slim.metrics.streaming_recall_at_k(
        logits, y, 5), })


saver = tf.train.Saver()

def compare (y_data,y_pred):
    print "len"
    print len(y_pred)
    print len(y_data)
    num = 0
    for x in range(0,len(y_pred)):
        if y_data[x] == y_pred[x]:
            num = num +1
    return num

def compareTop5 (y_data, tkv):
    print "len"
    print len(y_data)
    print len(tkv)
    num=0
    for x in range(0,len(y_data)):
        if y_data[x] in tkv[x]:
            num = num +1
    return num

def evaluate(X_data, y_data, test):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.get_default_session()

    accuracyTop1 = sess.run(accuracy_operation, feed_dict={x: X_data, y: y_data})

    # y_k_probs, y_k_pred = sess.run(tf.nn.top_k(y_, k=K), feed_dict={x: X_data})


    tkv, tki = sess.run([top_k_indices, top_k_values], feed_dict={x: X_data, y:
y_data})
    print tkv, tki

    num = compareTop5(y_data,tkv)
```

```
    accuracyTop5 = float(num)/num_example


    print acc

    if test == True:
        y_pred = sess.run(prediction, feed_dict={x: X_data})
        confusionM(y_data, y_pred)


    return accuracyTop1,accuracyTop5

# Run the training data through the training pipeline to train the model
with tf.Session() as sess:

    sess.run(tf.global_variables_initializer())
    num_example = len(X_train)

    print("Training...")
    print("")

    for i in range(EPOCHS):
        X_train, y_train = shuffle(X_train, y_train)
        for offset in range(0, num_example, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train[offset:end], y_train[offset:end]
            sess.run(training_operation, feed_dict={x:batch_x, y:batch_y})
            loss, acc = sess.run([loss_operation, accuracy_operation], feed_dict={x:
batch_x, y: batch_y})

            # log loss and acc once in a while
            if offset % 300 == 0:
                # print('[Epoch %3d - Step %5d] Loss=%.3f Accuracy=%.3f' % (i, offset,
loss, acc))

                LOSS_X.append(COUNT)
                ACC_X.append(COUNT)

                LOSS_Y.append(loss)
                ACC_Y.append(acc)

                ACC_LABEL.append(str(i)+"--"+str(offset))
                LOSS_LABEL.append(str(i)+"--"+str(offset))

                COUNT = COUNT +1



        validation_accuracy,top5 = evaluate(X_validation, y_validation, False)

        print ("EPOCH {} ...".format(i+1))
        print ("Validation accuracy = {:.3f}".format(validation_accuracy))
        print ("Validation accuracy Top5 = {:.3f}".format(top5))

        print ("")

    saver.save(sess, "./lenet")
    print ("Model saved")

with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))
```

```
    test_accuray, test_acc_top5 =evaluate (X_test, y_test, True)
    print ("Test_accuray = {:.3f}".format(test_accuray))

# confusionM([],[1],[1])
# y_data = [0,0,3,2,2,1,1,2,3]
# y_pred = [0,0,3,2,2,1,1,1,1]
#
# confusionM([],y_data,y_pred)

## Plot loss
plt.figure()
plt.plot(LOSS_X, LOSS_Y, 'ro')
# You can specify a rotation for the tick labels in degrees or with keywords.
plt.xticks(LOSS_X, LOSS_LABEL, rotation='vertical')
# Pad margins so that markers don't get clipped by the axes
plt.margins(0.2)
# Tweak spacing to prevent clipping of tick-labels
plt.subplots_adjust(bottom=0.15)
plt.show()

print LOSS_X, LOSS_Y
print ACC_X, ACC_Y
print LOSS_LABEL,ACC_LABEL
## Plot acc

plt.figure()
plt.plot(ACC_X, ACC_Y, 'ro')
# You can specify a rotation for the tick labels in degrees or with keywords.
plt.xticks(ACC_X, ACC_LABEL, rotation='vertical')
# Pad margins so that markers don't get clipped by the axes
plt.margins(0.2)
# Tweak spacing to prevent clipping of tick-labels
plt.subplots_adjust(bottom=0.15)
plt.show()




plt.show()
```

10 REFERENCE

1. https://medium.com/ymedialabs-innovation/data-augmentation-techniques-in-cnn-using-tensorflow-371ae43d5be9
2. https://www.youtube.com/watch?v=ixkpBmKRZDw&t=598s
3. https://stackoverflow.com/questions/50640687/get-top-k-predictions-from-tensorflow
4. https://stackoverflow.com/questions/39305174/what-does-x-tf-placeholdertf-float32-none-784-means
5. https://stackoverflow.com/questions/36467328/converting-3x32x32-size-numpy-image-array-to-32x32x3-and-saving-to-actual-image
6. https://stackoverflow.com/questions/44799516/tensorflow-is-there-a-metric-to-calculate-and-update-top-k-accuracy

7. https://matplotlib.org/gallery/ticks_and_spines/ticklabels_rotation.html#sphx-glr-gallery-ticks-and-spines-ticklabels-rotation-py