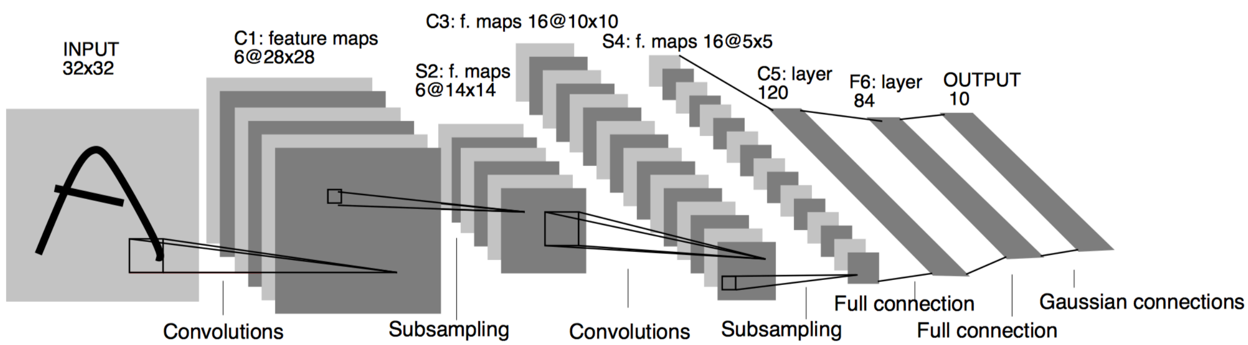
This is a programming assignment to create, train and test a CNN for the task of object classification. To keep the task manageable, we will use a small dataset and small network. With 7 main parts:

1. We will use the CIFAR-10 dataset. (https://www.cs.toronto.edu/~kriz/cifar.html). It consists of 10 mutually exclusive classes with 50,000 training images and 10000 test images, evenly distributed across the 10 classes. Each image is a 32x32 RGB image. Please use [data_batch_1, data_batch_2, data_batch_3, data_batch_4] for training, and data_batch_5 for validation, and test_batch for test.

2. Construct a LeNet-5 style CNN network, using Tensorflow functions. LeNet-5 is shown graphically below. Additional details of LeNet-5 can be found in http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf



Our our goal is to not exactly duplicate the architecture of the original LeNet-5 architecture. You may use the following hyper-parameters as a guide to start with.

> First layer has six, 5x5 convolution filters, stride =1, each followed by a max pooling layer of 2x2 with stride = 2. Second convolution layer has 16, 5x5 convolution filters, stride =1, each followed by 2x2 max pooling with stride = 2.
> Next comes a fully connected layer of dimensions 120 followed by another fully connected layer of dimension 84.
> A softmax layer of dimension 10 provides the classification probabilities (note that this is labeled "Gaussian connections" in the diagram but you can ignore this distinction).
> All activation units should be ReLU.

3. Train the network using the given training data. We suggest using a mini-batch size of 64, learning rate of 0.001 and the ADAM optimizer, though you are free to experiment with other values. Note that you need not write your own backpropagation algorithm, it is implemented in Tensorflow. However, you will need to write your own sampling program for constructing mini-batches; Python provides a number of "Random" functions that can generate a number in [0, 1] or return a sequence of desired size.

Record the error after each step so you can monitor it and plot it to show results.

During training, you should test on the validation set at some regular intervals; say every 0.5 epochs, to check whether the model is overfitting.

4. You should subtract mean of the images in the test set for pre-processing. An easy way to do this is to compute a mean image (per channel) and subtract from each train and test image. This replaces earlier guidance to use “per_image_standardization” which is more complex. Initialize network parameters by using the Xavier function.

5. You should augment the training data. A possible way to augment data is: enlarge the image by 10%, crop the 90% of the image in left-top right-top, left-bottom, right-bottom and central, and then flip the image and do this again. You may use “resize” functions in OpenCV or in TensorFlow to enlarge the image.

6. Test the trained network on the test data to obtain classification and show the results in the form of confusion matrix and classification accuracy for each class. Additionally, also show accuracy at different ranks (say rank-1 and rank-5).

7. Experiment with variations of the network with the aim of improving performance or ease of training the network without losing accuracy. Some suggestions are provided below but these are not requirements nor guaranteed to improve performance. Instead, you should be guided by the results of variations you obtain and also by study of other methods that will have been discussed in class. A suggested list follows.

- i) Change the filter size (make them smaller) followed possibly by additional layer.
- ii) Change the number of filters in the early layers.
- iii) Change the size of the fully connected layers.
- iv) Use weight regularizer (weight-decay).
- v) For inference, take multiple crops of expanded test image and average the scores.

As the variations can be combined, the number of options to try can be very large, it is not expected that you would try all combinations but, instead, choose a small number of combinations that you expect will result in improved performance.

Question 1. A brief description of the programs you write, including the source listing.

```
# some parameters
# check points dir
# calculate and subtract mean
# load the data
# calculate and subtract mean
# get random batch
# build preprocess architecture
# do the preprocess in tf
```

build the CNN
cov layer 1 with max pooling
conv layer 2 with max pooling
2 fc layers
fc out layer
create the network with pre process and scope name
get the prediction
run the network
plot the confusion matrix
plot images from https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/01_Simple_Linear_Model.ipynb
plot example errors
plot example correct
get the top 3 acc
print acc
main function
get the class name
place holder to hold train or test data, labels and class
to record step
create the network
set optimizer
get predict labels
get top-1 acc
get top-3 acc
initialize the saver
manage memory usage
try to restore the checkpoints
run the session
print test acc
close the session

Question 2. In general, show intermediate results that illustrate the workings of your program. No specific list of items to show is provided, general guidance is to show what you would find useful on your own to better understand how your method is working.
Question 3. Show evolution of loss function with multiple steps.

Question 2 and 3 TOGETHER :

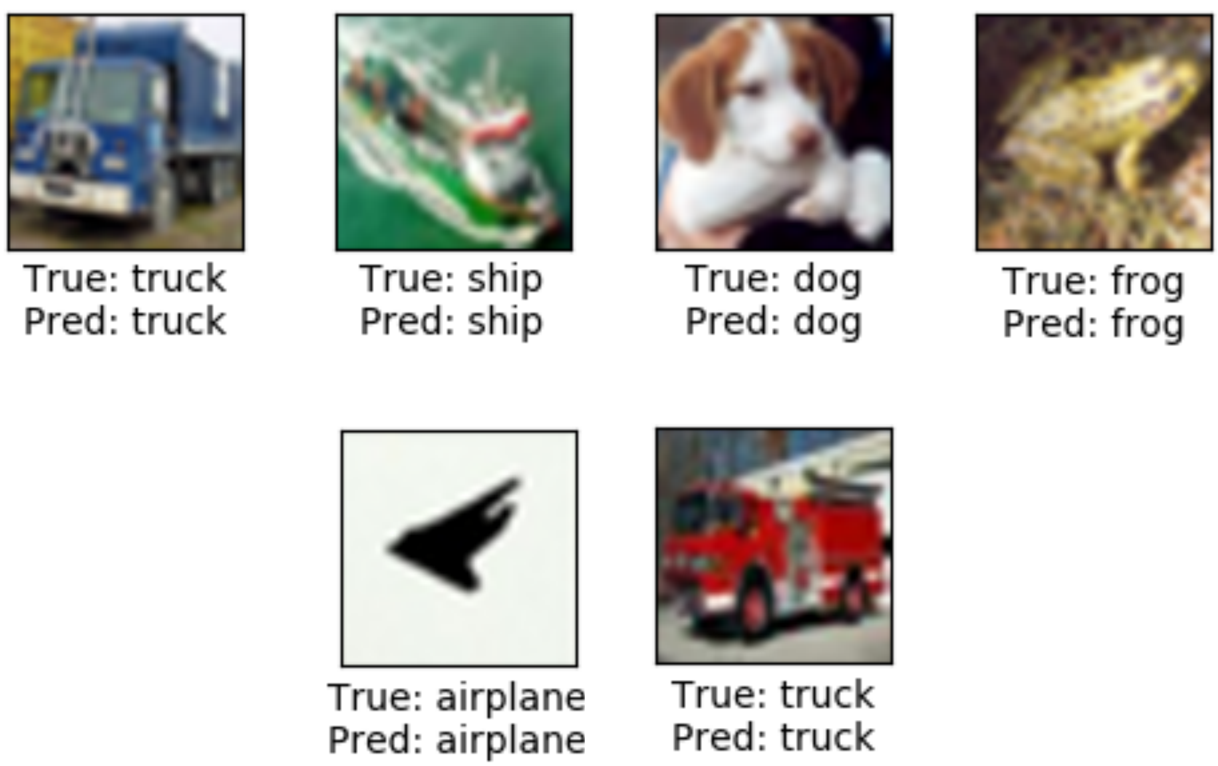
Inistial Step up:

The top1 accuracy is 66.7% and the top3 accuracy is 89.9%

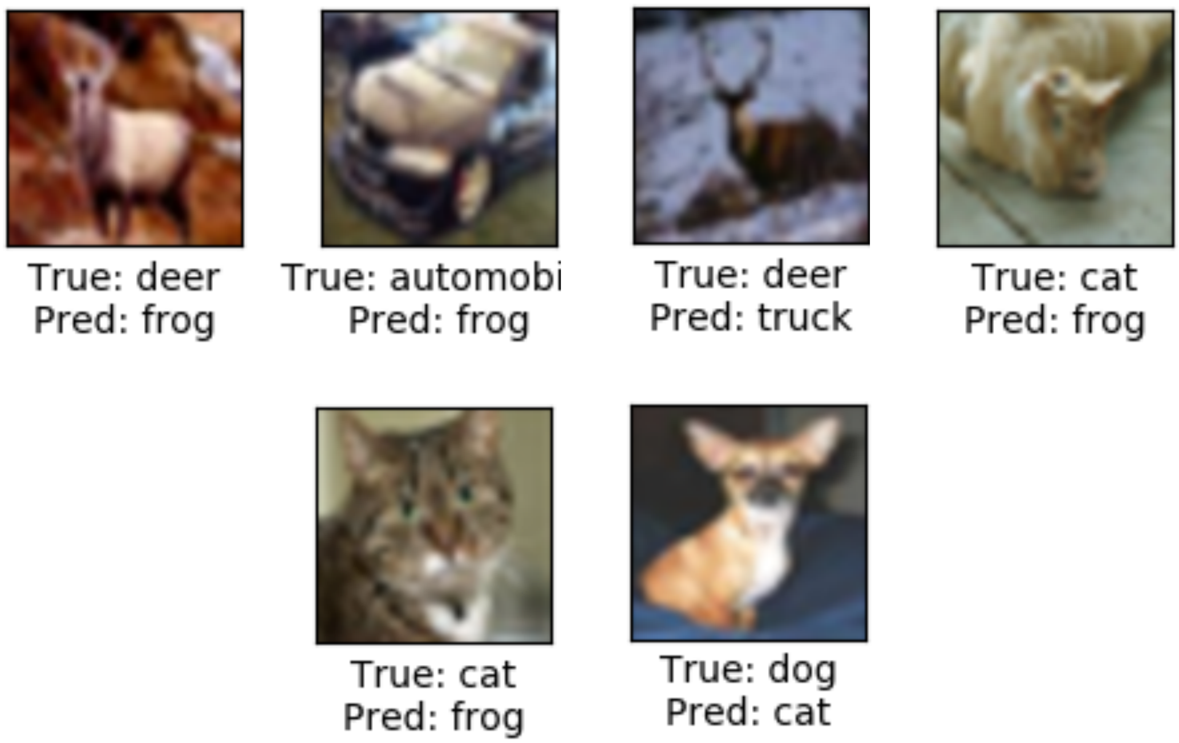
```
airplane:  77.2%    [772  35  26  16   7   5  14   7  55  63]
automobile:  84.8%  [ 17 848   5   3   0   3  13   2  11  98]
bird:  44.1%        [ 78  12 441  60  56  64 154  71  22  42]
cat:  42.2%         [ 28  32  33 422  33 160 164  54  19  55]
deer:  46.0%        [ 34  18  44  65 460  39 165 145  14  16]
dog:  53.2%         [ 14   7  20 165  19 532 101  98  11  33]
frog:  88.0%        [  8   8  11  41   9  21 880   5   4  13]
horse:  77.6%       [ 22   8  15  36  23  48  33 776   4  35]
ship:  73.9%        [ 93  66   2  16   4   3  13   7 739  57]
truck:  80.4%       [ 46  96   5   5   1   3  12  13  15 804]
```

Above shows the accuracy for each class and the confusion matrix.

Correct Samples



Incorrect Samples



Step	Loss
2500	1.475
12500	1.042
22500	0.563
32500	1.238

With padding
The top1 accuracy is 68.3% and the top3 accuracy is 90.7%

```
[783  43  30   9   6   7  13   5  58  46]
[ 12 882   4   3   0   0   6   2  12  79]
[ 80  26 559  41  53  56 103  41  17  24]
[ 37  47  62 473  46 125  86  58  22  44]
[ 44  15  88  73 537  26 114  81  11  11]
[ 19   7  56 190  37 527  52  67  16  29]
[ 14  20  28  44  14  22 833   4   6  15]
[ 32  21  40  34  28  43  24 742   7  29]
[ 78  77   8   7   6   1   4   3 780  36]
[ 45 164   7   6   2   4   9   8  21 734]
```

Step	Loss
2500	1.075
12500	0.734
22500	0.813
32500	0.437

With Regularizer to be l2_regularizer with 0.1 scale
 The top1 accuracy is 63.3% and the top3 accuracy is 87.1%

```
[742  33  48   2  13   5  15   7  85  50]
[ 29 782   3   1   0   2  13   1  34 135]
[ 74  29 522  13  54  66 141  35  18  48]
[ 37  34  79 249  52 206 180  51  29  83]
[ 40  21  99  24 450  40 192  77  20  37]
[ 25  20  62  85  44 546  96  72   9  41]
[ 16  21  32  17  17  23 844   5   6  19]
[ 44   8  43  19  37  54  44 668   6  77]
[107  72  16   7   4   2   7   5 741  39]
[ 53 134   7   6   1   1  13   7  41 737]
```

Step	Loss
2500	1.430
12500	1.263
22500	1.054
32500	0.801

With different FC size from 120 to 200 and from 84 to 100

The top1 accuracy is 68.4% and the top3 accuracy is 90.9%

```
[757  35  35  11  13   8  16  13  59  53]
[ 12 849   3   2   4   2  14   4  14  96]
[ 66  14 555  38  80  54 105  36  20  32]
[ 33  26  55 366  64 170 137  48  22  79]
[ 20  10  56  40 601  33 114  79  16  31]
[ 12   6  37 123  53 578  78  71   9  33]
[ 11  11  24  32  15  13 858   7   7  22]
[ 22  11  23  28  44  36  15 757   3  61]
[ 85  61   5  10   4   3   8   4 756  64]
[ 39  80   5   8   4   0  16   5  27 816]
```

Step	Loss
2500	1.201
12500	0.802
22500	0.794
32500	0.547

With Dropout

The top1 accuracy is 65.9% and the top3 accuracy is 89.3%

```
[814  40  33   6   5   4  21   4  50  23]
[ 31 862   3   2   0   5  16   5  14  62]
[ 80  17 485  41 104  69 140  32  10  22]
[ 54  39  56 404  59 155 147  45  13  28]
[ 51  11  49  47 563  24 160  70   9  16]
[ 34  11  42 153  47 547  82  57  14  13]
[ 23  19  27  34  15  14 851   6   0  11]
[ 54  12  29  34  44  46  46 701   1  33]
[153  93  14  12   6   2  11   4 674  31]
[ 80 156   5   8   1   9  14  11  10 706]
```

Step	Loss
2500	1.263
12500	0.962
22500	0.918
32500	0.794

Question 4. A summary and discussion of the results, including effects of parameter choices. Include visualization of results; show some examples of successful and some failure examples.

By changing the parameters for the CNN network, the best accuracy I can get for top1 and top3 are 68.4% and 90.9%, respectively. For each of my experiment, I show losses for different steps and confusion matrix. I only show the correct and incorrect samples for the initial model.

I did four different experiments, they are adding paddings, adding regularizer, change FC sizes, and adding dropouts. The experimental results indicates that:

> With paddings, the accuracy can go up. I think it is because with padding I could include the boundary information of each image. This can help the model to see a full picture and therefore leads to a better result.
>
> With regularizer, the accuracy is similar to the initial step up. The losses of this experiment decrease in a much slower rate than others. I think it is because with a regularizer, the training process become slow. I should wait longer time (increase training steps).
>
> Changing the size of FC layers, the accuracy goes up again. Actually, it gives the best results among all my experiments. I think it is because more information are passed to this layer, and therefore a better prediction.
>
> With a dropout, the accuracy is slightly better than initial step up. Again, the losses drop in a slower rate. I think it is because the training process is slow with a dropout. I should increase the training steps.

```python
import time
from datetime import timedelta
import math
import os
from sklearn.metrics import confusion_matrix
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2

import dataset


# some parameters
IMG_SIZE = 32 ; NUM_CHANNELS = 3 ; NUM_CLASSES = 10
TRAIN_BATCH_SIZE = 64 ; ENLARGED_IMG_SIZE = 35
TEST_BATCH_SIZE = 250 ; IMG_SIZE_CROPPED = 32
IMAGES_PER_TEST_CLASS = 1000 ; IMAGES_IN_TEST = 10000

# check points dir
SAVE_DIR = 'checkpoints/'

# load the data
images_train_only, cls_train_only, labels_train_only = dataset.load_training_data_only()
images_train_and_val, cls_train_and_val, labels_train_and_val = dataset.load_training_and_val_data()
images_val, cls_val, labels_val = dataset.load_val_data()
images_test, cls_test, labels_test = dataset.load_test_data()


# calculate and subtract mean
mean_train_and_val = np.mean(images_train_and_val, axis=0)
mean_train_only = np.mean(images_train_only, axis=0)
images_train_and_val = images_train_and_val - mean_train_and_val
images_train_only = images_train_only - mean_train_only
images_test = images_test - mean_train_and_val

# get random batch
def random_batch():
    num_images = len(images_train_and_val)
    idex = np.random.choice(num_images,
                            size=TRAIN_BATCH_SIZE,
                            replace=False)
    x_batch = images_train_and_val[idex, :, :, :]
    y_batch = labels_train_and_val[idex, :]
```

```python
    return x_batch, y_batch


# build preprocess architecture
def find_pre_process(image, training):
    if training:
        image = tf.image.resize_images(image, size=[ENLARGED_IMG_SIZE, ENLARGED_IMG_SIZE])
        image = tf.random_crop(image, size=[IMG_SIZE_CROPPED, IMG_SIZE_CROPPED, NUM_CHANNELS])
        image = tf.image.random_flip_left_right(image)
    return image


# do the pre process in tf
def pre_process(images, training):
    images = tf.map_fn(lambda image: find_pre_process(image, training), images)
    return images

# build the CNN
def network(images, training):
    # cov layer 1 with max pooling
    cnn = images
    cnn = tf.layers.conv2d(inputs=cnn, filters=6, kernel_size=5, strides=(1, 1),
                           padding='valid', activation=tf.nn.relu,
                           kernel_initializer=tf.contrib.layers.xavier_initializer(),
                           bias_initializer=tf.contrib.layers.xavier_initializer(),
                           name='conv1')
    conv1 = cnn
    cnn = tf.layers.max_pooling2d(inputs=cnn, pool_size=2, strides=2)

    # conv layer 2 with max pooling
    cnn = tf.layers.conv2d(inputs=cnn, filters=16, kernel_size=5, strides=(1, 1),
                           padding='valid', activation=tf.nn.relu,
                           kernel_initializer=tf.contrib.layers.xavier_initializer(),
                           bias_initializer=tf.contrib.layers.xavier_initializer(),
                           name='conv2')
    conv2 = cnn
    cnn = tf.layers.max_pooling2d(inputs=cnn, pool_size=2, strides=2)

    # 2 fc layers
    cnn = tf.contrib.layers.flatten(cnn)
    cnn = tf.layers.dense(inputs=cnn, units=120, activation=tf.nn.relu,
                          kernel_initializer=tf.contrib.layers.xavier_initializer(),
                          bias_initializer=tf.contrib.layers.xavier_initializer(),
                          name='fc1')
```

```python
                                      name='fc1')
    cnn = tf.layers.dense(inputs=cnn, units=84, activation=tf.nn.relu,
                                      kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                      bias_initializer=tf.contrib.layers.xavier_initializer(),
                                      name='fc2')

    # fc out layer
    cnn = tf.layers.dense(inputs=cnn, units=NUM_CLASSES, activation=None,
                                      kernel_initializer=tf.contrib.layers.xavier_initializer(),
                                      bias_initializer=tf.contrib.layers.xavier_initializer(),
                                      name='fc_out')
    logits = cnn
    y_pred = tf.nn.softmax(logits=logits)

    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_true, logits=logits)
    loss = tf.reduce_mean(cross_entropy)

    return y_pred, loss


# create the network with pre process and scope name
def create_network(training):
    with tf.variable_scope('network', reuse=not training):
        images = x
        images = pre_process(images=images, training=training)
        y_pred, loss = network(images=images, training=training)
    return y_pred, loss


# get the prediction
def predict_class(images, labels, cls_true):
    num_images = len(images)
    class_pred = np.zeros(shape=num_images, dtype=np.int)
    i = 0
    while i < num_images:
        j = min(i + TEST_BATCH_SIZE, num_images)
        feed_dict = {x: images[i:j, :],
                     y_true: labels[i:j, :]}
        class_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)
        i = j
    correct = (cls_true == class_pred)
    return correct, class_pred

# run the network
def optimize(num_iterations):
    start_time = time.time()
    for i in range(num_iterations):
        x_batch, y_true_batch = random_batch()
        feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}
        i_global, _ = session.run([global_step, optimizer],
                                  feed_dict=feed_dict_train)
        if (i_global % 1000 == 0) or (i == num_iterations - 1):
            batch_acc = session.run(accuracy,
                                    feed_dict=feed_dict_train)
            batch_loss = session.run(loss,
                                    feed_dict=feed_dict_train)
            msg = "Step: {0:>6}, Training top-1 accuracy: {1:>6.1%}, Training loss: {2:4}"
            print msg.format(i_global, batch_acc, batch_loss)

        if (i_global % 1000 == 0) or (i == num_iterations - 1):
            saver.save(session,
                       save_path=save_path,
                       global_step=global_step)
    end_time = time.time()
    time_dif = end_time - start_time
    print "Time usage: " + str(timedelta(seconds=int(round(time_dif))))


# plot the confusion matrix
def plot_confusion_matrix_and_acc(class_pred):
    cm = confusion_matrix(y_true=cls_test,  # True class for test-set.
                          y_pred=class_pred)  # Predicted class.
    for i in range(NUM_CLASSES):
        print cm[i, :], class_names[i]
    # print class_names
    print ('Accuracy for each class:\n')
    for i in range(NUM_CLASSES):
        # print class_names[i]+":" , float(cm[i, i])/float(IMAGES_PER_TEST_CLASS)
        acc_tmp  = float(cm[i, i])/float(IMAGES_PER_TEST_CLASS)
        msg = "{0}: {1:>6.1%}"
        print msg.format(class_names[i], acc_tmp)


# plot images from https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/01_Simple_Linear_Model.ipynb
def plot_images(images, cls_true, class_pred=None, smooth=True):
    assert len(images) == len(cls_true) == 9
    fig, axes = plt.subplots(3, 3)
    if class_pred is None:
```

```python
        hspace = 0.3
    else:
        hspace = 0.6
    fig.subplots_adjust(hspace=hspace, wspace=0.3)

    for i, ax in enumerate(axes.flat):
        if smooth:
            interpolation = 'spline16'
        else:
            interpolation = 'nearest'

        ax.imshow(images[i, :, :, :],
                  interpolation=interpolation)

        cls_true_name = class_names[cls_true[i]]

        if class_pred is None:
            xlabel = "True: {0}".format(cls_true_name)
        else:
            cls_pred_name = class_names[class_pred[i]]

            xlabel = "True: {0}\nPred: {1}".format(cls_true_name, cls_pred_name)

        ax.set_xlabel(xlabel)
        ax.set_xticks([])
        ax.set_yticks([])

    plt.show()


# plot example errors
def plot_example_errors(class_pred, correct):
    incorrect = (correct == False)
    images = images_test[incorrect]
    class_pred = class_pred[incorrect]
    cls_true = cls_test[incorrect]
    images = images + mean_train_only
    plot_images(images=images[100:109],
                cls_true=cls_true[100:109],
                class_pred=class_pred[100:109])


# plot example correct
def plot_example_correct(class_pred, correct):
    correct_inx = (correct == True)
    images = images_test[correct_inx]
    class_pred = class_pred[correct_inx]
    cls_true = cls_test[correct_inx]
    images = images + mean_train_only
    plot_images(images=images[10:19],
                cls_true=cls_true[10:19],
                class_pred=class_pred[10:19])


# get the top 3 acc
def get_top3_acc(images, labels):
    num_images = len(images)
    test_batch_num = IMAGES_IN_TEST/TEST_BATCH_SIZE
    top3_acc_array = np.zeros(shape=test_batch_num, dtype=np.float)
    i = 0
    top3_acc_array_idx = 0
    while i < num_images:
        j = i + TEST_BATCH_SIZE
        feed_dict = {x: images[i:j, :],
                     y_true: labels[i:j, :]}
        top3_acc_array[top3_acc_array_idx] = session.run(top3_accuracy, feed_dict=feed_dict)
        i = j
        top3_acc_array_idx += 1
    return top3_acc_array.mean()


# print acc
def print_test_accuracy(show_example_errors=False,
                        show_example_correct=False,
                        show_confusion_matrix=False):

    correct, class_pred = predict_class(images=images_test,
                                        labels=labels_test, cls_true=cls_test)

    top3_acc_final = get_top3_acc(images=images_test, labels=labels_test)

    acc = correct.mean()
    num_correct = correct.sum()
    num_images = len(correct)
```

```python
    msg = "Top-1 accuracy on test set: {0:.1%}"
    print msg.format(acc)

    msg = "Top-3 accuracy on test set: {0:.1%}"
    print msg.format(top3_acc_final)

    if show_confusion_matrix:
        print "Confusion matrix:"
        print '\n\n'
        plot_confusion_matrix_and_acc(class_pred=class_pred)
        print '\n\n'

    if show_example_correct:
        plot_example_correct(class_pred=class_pred, correct=correct)

    if show_example_errors:
        plot_example_errors(class_pred=class_pred, correct=correct)


#################################################################################################################
# main function starts here


# main function
# get the class name
class_names = dataset.load_class_names()

# place holder to hold train or test data, labels and class
x = tf.placeholder(tf.float32, shape=[None, IMG_SIZE, IMG_SIZE, NUM_CHANNELS], name='x')
y_true = tf.placeholder(tf.float32, shape=[None, NUM_CLASSES], name='y_true')
y_true_cls = tf.argmax(y_true, axis=1)

# to record step
global_step = tf.Variable(initial_value=0, name='global_step', trainable=False)

# create the network
_, loss = create_network(training=True)

# set optimizer
optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss, global_step=global_step)

# get predict labels
y_pred, _ = create_network(training=False)
```

```python
# get predict labels
y_pred, _ = create_network(training=False)
y_pred_cls = tf.argmax(y_pred, axis=1)

# get top-1 acc
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

# get top-3 acc
targets = tf.cast(y_true_cls, tf.int32)
top3_accuracy = tf.reduce_mean(tf.cast(tf.nn.in_top_k(predictions=y_pred, targets=targets, k=3), tf.float32))

# initialize the saver
saver = tf.train.Saver()

# manage memory usage
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.45
session = tf.Session(config=config)

if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)

save_path = os.path.join(SAVE_DIR, 'cifar10_cnn')

# try to restore the checkpoints
try:
    print "Restoreing last checkpoint ..."
    last_chk_path = tf.train.latest_checkpoint(checkpoint_dir=SAVE_DIR)
    saver.restore(session, save_path=last_chk_path)
    print "Restored checkpoint from:", last_chk_path
except:
    print "Failed to restore checkpoint."
    print "Start to train from scratch."
    session.run(tf.global_variables_initializer())

# run the session
optimize(num_iterations=39000)

# print test acc
print_test_accuracy(show_example_errors=True,
                    show_example_correct=True,
                    show_confusion_matrix=True)
# close the session
session.close()
```