In [ ]:

```
import tensorflow as tf
import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import scipy
from PIL import Image
from scipy import ndimage

%matplotlib inline
np.random.seed(1)
```

In [ ]:

```
def init_placeholders(n_H0, n_W0, n_C0, n_y):

    X = tf.placeholder(tf.float32, [None, n_H0, n_W0, n_C0])
    Y = tf.placeholder(tf.float32, [None, n_y])

    return X, Y

X, Y = create_placeholders(64, 64, 3, 6)
print ("X = " + str(X))
print ("Y = " + str(Y))
```

In [ ]:

```
def init_parameters(n_C0, n_C1):
    # tf.set_random_seed(1)

    W1 = tf.get_variable("W1", [4, 4, n_C0, n_C1], initializer=tf.contrib.layers.xavier_initializer(
    W2 = tf.get_variable("W2", [2, 2, n_C1, 16], initializer=tf.contrib.layers.xavier_initializer(se

    parameters = {"W1": W1,
                  "W2": W2}

    return parameters
```

In [ ]:

```
def forward_propagation(X, parameters):

    # Retrieve the parameters from the dictionary "parameters"
    W1 = parameters['W1']
    W2 = parameters['W2']

    # 1. cnn layer ----------------------
    # CONV2D: stride of 1, padding 'SAME'
    Z1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')

    # RELU
    A1 = tf.nn.relu(Z1)

    # MAXPOOL: window 8x8, sride 8, padding 'SAME'
    P1 = tf.nn.max_pool(A1, ksize=[1, 8, 8, 1], strides=[1, 8, 8, 1], padding='SAME')

    # 2. cnn layer ----------------------
    # CONV2D: filters W2, stride 1, padding 'SAME'
    Z2 = tf.nn.conv2d(P1, W2, strides=[1, 1, 1, 1], padding='SAME')
    A2 = tf.nn.relu(Z2)
    P2 = tf.nn.max_pool(A2, ksize=[1, 4, 4, 1],strides=[1, 4, 4, 1], padding='SAME')

    # FLATTEN
    P2 = tf.contrib.layers.flatten(P2)

    # FULLY-CONNECTED without non-linear activation function (not not call softmax).
    # 6 neurons in output layer. Hint: one of the arguments should be "activation_fn=None"
    Z3 = tf.contrib.layers.fully_connected(P2, 6, activation_fn=None)

    return Z3
```

In [ ]:

```
def fit(learning_rate=0.009,
        num_epochs=400,
        minibatch_size=64,
        print_cost=True):

    tf.reset_default_graph()                          # to be able to rerun the model without overwri
    tf.set_random_seed(1)                              # to keep results consistent (tensorflow seed)
    seed = 3                                           # to keep results consistent (numpy seed)

    # get the dimensions by shape of training input
    (m, n_H0, n_W0, n_C0) = mnist.train.images
    n_y = Y_train.shape[1]

    costs = []                                         # To keep track of the cost

    # Create Placeholders of the correct shape
    X, Y       = init_placeholders(n_H0, n_W0, n_C0, n_y)
    parameters = init_parameters(n_C0, 16)

    # Forward propagation: Build the forward propagation in the tensorflow graph
    Z3 = forward_propagation(X, parameters)

    # Cost function: Add cost function to tensorflow graph
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=Z3, labels=Y))
    optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)

        # Do the training loop
        for epoch in range(num_epochs):

            minibatch_cost = 0.
            num_minibatches = int(m / minibatch_size) # number of minibatches of size minibatch_size
            seed = seed + 1
            batch_xs, batch_ys = mnist.train.next_batch(minibatch_size)

            for _ in num_minibatches:
                # Run the session to execute the optimizer and the cost, the feedict should contain
                _ , temp_cost = sess.run([optimizer, cost], feed_dict={X: batch_xs.reshape(-1, 32, 3
                                                                       Y: batch_ys.reshape(-1, 10)})

                minibatch_cost += temp_cost / num_minibatches


            # Print the cost every epoch
            if print_cost == True and epoch % 100 == 0:
                print ("Cost after epoch %i: %f" % (epoch, minibatch_cost))
            if print_cost == True and epoch % 1 == 0:
                costs.append(minibatch_cost)


        # plot the cost
        plt.plot(np.squeeze(costs))
        plt.ylabel('cost')
        plt.xlabel('iterations (per tens)')
        plt.title("Learning rate =" + str(learning_rate))
```

```
        plt.show()

        # Calculate the correct predictions
        predict_op = tf.argmax(Z3, 1)
        correct_prediction = tf.equal(predict_op, tf.argmax(Y, 1))

        # Calculate accuracy on the test set
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
        print(accuracy)

        train_accuracy = accuracy.eval({X: mnist.train.images, Y: mnist.train.labels})
        # sess.run(accuracy, feed_dict={X: mnist.train.images, Y: mnist.train.labels})

        test_accuracy =  accuracy.eval({X: mnist.test.images, Y: mnist.test.images})

        print("Train Accuracy:", train_accuracy)
        print("Test Accuracy:", test_accuracy)

        return train_accuracy, test_accuracy, parameters
```

In [ ]:

```
fit()
```