conv_net_gpu_p1_ZihaoWang

May 10, 2018

```
In [2]: import tensorflow as tf
        import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib inline
        import math

        # Convert labels to one-hot vectors

        # Convert classes to indicator vectors
        def one_hot(values,n_values=10):
            n_v = np.maximum(n_values,np.max(values) + 1)
            oh=np.eye(n_v)[values]
            return oh
```

1 Get Mnist data and split into train validation and test

```
In [3]: def get_mnist():

            data=np.float64(np.load('/project/cmsc25025/mnist/MNIST.npy'))
            labels=np.float32(np.load('/project/cmsc25025/mnist/MNIST_labels.npy'))
            print(data.shape)
            data=np.float32(data)/255.
            train_dat=data[0:50000]
            train_labels=one_hot(np.int32(labels[0:50000]))
            val_dat=data[50000:60000]
            val_labels=one_hot(np.int32(labels[50000:60000]))
            test_dat=data[60000:70000]
            test_labels=one_hot(np.int32(labels[60000:70000]))

            return (train_dat, train_labels), (val_dat, val_labels), (test_dat, test_labels)
```

2 Get CIFAR10 data and split into train validation and test

```
In [4]: def get_cifar():
            tr=np.float32(np.load('/project/cmsc25025/mnist/CIFAR_10.npy'))
            tr_lb=np.int32(np.load('/project/cmsc25025/mnist/CIFAR_labels.npy'))
```

```
        tr=tr.reshape((-1,np.prod(np.array(tr.shape)[1:4])))
        train_data=tr[0:45000]/255.
        train_labels=one_hot(tr_lb[0:45000])
        val_data=tr[45000:]/255.
        val_labels=one_hot(tr_lb[45000:])
        test_data=np.float32(np.load('/project/cmsc25025/mnist/CIFAR_10_test.npy'))
        test_data=test_data.reshape((-1,np.prod(np.array(test_data.shape)[1:4])))
        test_data=test_data/255.
        test_labels=one_hot(np.int32(np.load('/project/cmsc25025/mnist/CIFAR_labels_test.n
        return (train_data, train_labels), (val_data, val_labels), (test_data, test_labels
```

3 Get transformed Mnist data

```
In [5]: def get_mnist_trans():
            test_trans_dat=np.float32(np.load('/project/cmsc25025/mnist/MNIST_TEST_TRANS.npy')
            test_labels=one_hot(np.int32(np.float32(np.load('/project/cmsc25025/mnist/MNIST_la
            return (test_trans_dat, test_labels)
```

4 Convolution layer with relu

```
In [6]: def conv_relu_layer(input,filter_size=[3,3],num_features=[1]):

            # Get number of input features from input and add to shape of new layer
            shape=filter_size+[input.get_shape().as_list()[-1],num_features]
            W = tf.get_variable('W',shape=shape) # Default initialization is Glorot (the one e
            b = tf.get_variable('b',shape=[num_features],initializer=tf.zeros_initializer)
            conv = tf.nn.conv2d(input, W, strides=[1, 1, 1, 1], padding='SAME')
            relu = tf.nn.relu(conv + b)
            return(relu)
```

5 Fully connected layer

```
In [7]: def fully_connected_layer(input,num_features):
            # Make sure input is flattened.
            flat_dim=np.int32(np.array(input.get_shape().as_list())[1:].prod())
            input_flattened = tf.reshape(input, shape=[-1,flat_dim])
            shape=[flat_dim,num_features]
            W_fc = tf.get_variable('W',shape=shape)
            b_fc = tf.get_variable('b',shape=[num_features],initializer=tf.zeros_initializer)
            fc = tf.matmul(input_flattened, W_fc) + b_fc
            return(fc)
```

6 The network

```
In [8]: #tf.reset_default_graph()
```

```
def create_network():
    pool_ksize=[1,2,2,1]
    pool_strides=[1,2,2,1]
    # The network:
    with tf.variable_scope("conv1"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu1 = conv_relu_layer(x_image, filter_size=[5, 5],num_features=32)
            pool1 = tf.nn.max_pool(relu1, ksize=pool_ksize, strides=pool_strides, paddi
    with tf.variable_scope("conv2"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu2 = conv_relu_layer(pool1, filter_size=[5, 5],num_features=64)
            pool2 = tf.nn.max_pool(relu2, ksize=pool_ksize, strides=pool_strides, paddi
    with tf.variable_scope('dropout2'):
            drop2=tf.nn.dropout(pool2,keep_prob)
    with tf.variable_scope("fc1"):
            fc1 = fully_connected_layer(drop2, num_features=256)
            fc1r=tf.nn.relu(fc1)

    with tf.variable_scope("fc2"):
            fc2 = fully_connected_layer(fc1r, num_features=10)

    # Names (OUT,LOSS, ACC) below added to make it easier to use this tensor when rest
    fc2 = tf.identity(fc2, name="OUT")
    # The loss computation
    with tf.variable_scope('cross_entropy_loss'):
        cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y

    # Accuracy computation
    with tf.variable_scope('helpers'):
        correct_prediction = tf.equal(tf.argmax(fc2, 1), tf.argmax(y_, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="ACC")
    # We return the final functions (they contain all the information about the graph
    return cross_entropy, accuracy, fc2
```

In [9]:
```
#tf.reset_default_graph()

def my_create_network0(nums_features = [32,64,256,10]):
    pool_ksize=[1,2,2,1]
    pool_strides=[1,2,2,1]
    # The network:
    with tf.variable_scope("conv1"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu1 = conv_relu_layer(x_image, filter_size=[5, 5],num_features=nums_featu
            pool1 = tf.nn.max_pool(relu1, ksize=pool_ksize, strides=pool_strides, paddi
    with tf.variable_scope("conv2"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu2 = conv_relu_layer(pool1, filter_size=[5, 5],num_features=nums_feature
            pool2 = tf.nn.max_pool(relu2, ksize=pool_ksize, strides=pool_strides, paddi
    with tf.variable_scope('dropout2'):
            drop2=tf.nn.dropout(pool2,keep_prob)
    with tf.variable_scope("fc1"):
            fc1 = fully_connected_layer(drop2, num_features=nums_features[2])
            fc1r=tf.nn.relu(fc1)

    with tf.variable_scope("fc2"):
            fc2 = fully_connected_layer(fc1r, num_features=nums_features[3])

    # Names (OUT,LOSS, ACC) below added to make it easier to use this tensor when resto
    fc2 = tf.identity(fc2, name="OUT")
    # The loss computation
    with tf.variable_scope('cross_entropy_loss'):
        cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y

    # Accuracy computation
    with tf.variable_scope('helpers'):
        correct_prediction = tf.equal(tf.argmax(fc2, 1), tf.argmax(y_, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="ACC")
    # We return the final functions (they contain all the information about the graph o
    return cross_entropy, accuracy, fc2
```

In [10]:
```
#tf.reset_default_graph()

def my_create_network_deeper(nums_features = [32,64,128,128,10]):
    pool_ksize=[1,2,2,1]
    pool_strides=[1,2,2,1]
    # The network:
    with tf.variable_scope("conv1"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu1 = conv_relu_layer(x_image, filter_size=[5, 5],num_features=nums_feat
            pool1 = tf.nn.max_pool(relu1, ksize=pool_ksize, strides=pool_strides, padd
    with tf.variable_scope("conv2"):
            # Variables created here will be named "conv1/weights", "conv1/biases".
            relu2 = conv_relu_layer(pool1, filter_size=[5, 5],num_features=nums_featur
            pool2 = tf.nn.max_pool(relu2, ksize=pool_ksize, strides=pool_strides, padd
    with tf.variable_scope('dropout2'):
            drop2=tf.nn.dropout(pool2,keep_prob)
    with tf.variable_scope("fc1"):
            fc1 = fully_connected_layer(drop2, num_features=nums_features[2])
            fc1r=tf.nn.relu(fc1)

    with tf.variable_scope("fc2"):
            fc2 = fully_connected_layer(fc1r, num_features=nums_features[3])
            fc2r=tf.nn.relu(fc2)

    with tf.variable_scope("fc3"):
            fc3 = fully_connected_layer(fc2r, num_features=nums_features[4])
```

```
        # Names (OUT,LOSS, ACC) below added to make it easier to use this tensor when res
        fc3 = tf.identity(fc3, name="OUT")
        # The loss computation
        with tf.variable_scope('cross_entropy_loss'):
            cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=

        # Accuracy computation
        with tf.variable_scope('helpers'):
            correct_prediction = tf.equal(tf.argmax(fc3, 1), tf.argmax(y_, 1))
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="ACC")
        # We return the final functions (they contain all the information about the graph
        return cross_entropy, accuracy, fc3
```

```
In [11]: #tf.reset_default_graph()

    def my_create_network_deeper2(nums_features = [32,64,224,224,224,10]):
        pool_ksize=[1,2,2,1]
        pool_strides=[1,2,2,1]
        # The network:
        with tf.variable_scope("conv1"):
                # Variables created here will be named "conv1/weights", "conv1/biases".
                relu1 = conv_relu_layer(x_image, filter_size=[5, 5],num_features=nums_fea
                pool1 = tf.nn.max_pool(relu1, ksize=pool_ksize, strides=pool_strides, padd
        with tf.variable_scope("conv2"):
                # Variables created here will be named "conv1/weights", "conv1/biases".
                relu2 = conv_relu_layer(pool1, filter_size=[5, 5],num_features=nums_featu
                pool2 = tf.nn.max_pool(relu2, ksize=pool_ksize, strides=pool_strides, padd
        with tf.variable_scope('dropout2'):
                drop2=tf.nn.dropout(pool2,keep_prob)
        with tf.variable_scope("fc1"):
                fc1 = fully_connected_layer(drop2, num_features=nums_features[2])
                fc1r=tf.nn.relu(fc1)

        with tf.variable_scope("fc2"):
                fc2 = fully_connected_layer(fc1r, num_features=nums_features[3])
                fc2r=tf.nn.relu(fc2)

        with tf.variable_scope("fc2_2"):
                fc2_2 = fully_connected_layer(fc2r, num_features=nums_features[3])
                fc2r_2=tf.nn.relu(fc2_2)

        with tf.variable_scope("fc3"):
                fc3 = fully_connected_layer(fc2r_2, num_features=nums_features[5])

        # Names (OUT,LOSS, ACC) below added to make it easier to use this tensor when res
        fc3 = tf.identity(fc3, name="OUT")
        # The loss computation
```

```
            with tf.variable_scope('cross_entropy_loss'):
                cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=

            # Accuracy computation
            with tf.variable_scope('helpers'):
                correct_prediction = tf.equal(tf.argmax(fc3, 1), tf.argmax(y_, 1))
                accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="ACC")
            # We return the final functions (they contain all the information about the graph
            return cross_entropy, accuracy, fc3
```

```
In [12]: #tf.reset_default_graph()

        def my_create_network_t0(nums_features = [32,64,256,10],\
                                 pool_ksize=[1,6,6,1], pool_strides=[1,2,2,1]):
            #pool_ksize=[1,2,2,1]
            pool_ksize = pool_ksize
            #pool_ksize2 = pool_ksize2
            pool_strides=pool_strides
            # The network:
            with tf.variable_scope("conv1"):
                    # Variables created here will be named "conv1/weights", "conv1/biases".
                    relu1 = conv_relu_layer(x_image, filter_size=[5, 5],num_features=nums_fea
                    pool1 = tf.nn.max_pool(relu1, ksize=pool_ksize, strides=pool_strides, padd
            with tf.variable_scope("conv2"):
                    # Variables created here will be named "conv1/weights", "conv1/biases".
                    relu2 = conv_relu_layer(pool1, filter_size=[5, 5],num_features=nums_featu
                    pool2 = tf.nn.max_pool(relu2, ksize=pool_ksize, strides=pool_strides, padd
            with tf.variable_scope('dropout2'):
                    drop2=tf.nn.dropout(pool2,keep_prob)
            with tf.variable_scope("fc1"):
                    fc1 = fully_connected_layer(drop2, num_features=nums_features[2])
                    fc1r=tf.nn.relu(fc1)

            with tf.variable_scope("fc2"):
                    fc2 = fully_connected_layer(fc1r, num_features=nums_features[3])

            # Names (OUT,LOSS, ACC) below added to make it easier to use this tensor when res
            fc2 = tf.identity(fc2, name="OUT")
            # The loss computation
            with tf.variable_scope('cross_entropy_loss'):
                cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=

            # Accuracy computation
            with tf.variable_scope('helpers'):
                correct_prediction = tf.equal(tf.argmax(fc2, 1), tf.argmax(y_, 1))
                accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32),name="ACC")
            # We return the final functions (they contain all the information about the graph
            return cross_entropy, accuracy, fc2
```

7 Get loss and accuracy on a data set with output from final layer fc2

```
In [13]: from scipy.special import logsumexp

         def get_stats(data,labels):
             t1=time.time()
             lo=0.
             acc=0.
             delta=1000
             rr=np.arange(0,data.shape[0],delta)
             for i in rr:
                 fc2_out=fc2.eval(feed_dict={x: data[i:i+delta], y_:labels[i:i+delta]})
                 log_sf=logsumexp(fc2_out,axis=1).reshape((fc2_out.shape[0],1))-fc2_out
                 lo+=np.mean(np.sum(labels[i:i+delta]*log_sf, axis=1))
                 acc += np.mean(np.equal(np.argmax(fc2_out, axis=1),np.argmax(labels[i:i+delta]
             acc=acc/np.float32(len(rr))
             lo=lo/np.float32(len(rr))
             print('get stats time',time.time()-t1)
             # We return the final functions (they contain all the information about the graph
             return lo, acc
```

8 Run one epoch

```
In [14]: # Run the iterations of one epoch
         def run_epoch(train,val,ii,batch_size,train_step_new):
                 t1=time.time()
                 # Randomly shuffle the training data
                 np.random.shuffle(ii)
                 tr=train[0][ii]
                 y=train[1][ii]
                 lo=0.
                 acc=0.
                 # Run disjoint batches on shuffled data
                 for j in np.arange(0,len(y),batch_size):
                     if (np.mod(j,5000)==0):
                         print('Batch',j/batch_size)
                     batch=(tr[j:j+batch_size],y[j:j+batch_size])
                     train_step_new.run(feed_dict={x: batch[0], y_: batch[1], lr_: step_size,ke
                 print('Epoch time',time.time()-t1)

In [15]: def get_data(data_set):
             if (data_set=="cifar"):
                 return(get_cifar())
             elif (data_set=="mnist"):
                 return(get_mnist())
             elif (data_set=="mnist_transform"):
                 return(get_mnist_trans())
```

9 (a)

9.1 (i)

The number of parameters (only considering weights in the ANN) are (conv1/W + conv1/b + conv2/W + conv2/b + fc1/W +fc1/b + fc2/W + fc2/b) = *55132* + 32 + *553264* + *64* + *(7764)*256 + 256 + 256*10 + 10

9.2 (ii)

```
In [17]: ## reference https://github.com/Hvass-Labs/TensorFlow-Tutorials

         def plot_conv_layer(layer, image):
             # Assume layer is a TensorFlow op that outputs a 4-dim tensor
             # which is the output of a convolutional layer,
             # e.g. layer_conv1 or layer_conv2.

             # Create a feed-dict containing just one image.
             # Note that we don't need to feed y_true because it is
             # not used in this calculation.
             feed_dict = {x: [image]}

             # Calculate and retrieve the output values of the layer
             # when inputting that image.
             values = sess.run(layer, feed_dict=feed_dict)

             # Number of filters used in the conv. layer.
             num_filters = values.shape[3]

             # Number of grids to plot.
             # Rounded-up, square-root of the number of filters.
             num_grids = math.ceil(math.sqrt(num_filters))

             # Create figure with a grid of sub-plots.
             fig, axes = plt.subplots(num_grids, num_grids)

             # Plot the output images of all the filters.
             for i, ax in enumerate(axes.flat):
                 # Only plot the images for valid filters.
                 if i<num_filters:
                     # Get the output image of using the i'th filter.
                     # See new_conv_layer() for details on the format
                     # of this 4-dim tensor.
                     img = values[0, :, :, i]

                     # Plot image.
                     ax.imshow(img, interpolation='nearest', cmap='binary')
```

```
            # Remove ticks from the plot.
            ax.set_xticks([])
            ax.set_yticks([])

        # Ensure the plot is shown correctly with multiple plots
        # in a single Notebook cell.
        plt.show()
```

```
In [18]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         model_name="model"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3


         ## collect error
         err_train = []
         err_val = []

         tf.reset_default_graph()

         x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
         x_image = tf.reshape(x, [-1, dim, dim, nchannels])
         # Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
         # The number of incoming channels, for example, will be 3 if the image is color: RGB
         # We will slide filter over this 2d picture with conv2d function.
         y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
         # Allows you to control the time step during the iterations
         lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
         keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

         with tf.Session() as sess:
                 train,val,test=get_data(data_set=data_set)
                 # Create the network architecture with the above placeholdes as the inputs.
                 cross_entropy, accuracy, fc2 =create_network()

                 # Define the miminization method
                 if (minimizer=="Adam"):
                     train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entrop
                 elif (minimizer=="SGD"):
                     train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
                 # Initialize variables
                 sess.run(tf.global_variables_initializer())
                 # Show trainable variables
                 for v in tf.trainable_variables():
                     print(v.name,v.get_shape().as_list(),np.std(v.eval()))
                 ii=np.arange(0,num_train,1) #len(train_data),1)
                 # Run epochs
                 for i in range(num_epochs):  # number of epochs
                     run_epoch(train,val,ii,batch_size,train_step)
                     if (np.mod(i,2)==0):
                         lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                         err_train.append(1-ac)
                         print('Epoch',i,'Train loss, accuracy',lo,ac)

                         vlo,vac = get_stats(val[0],val[1])

                         err_val.append(1-ac)
                         print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                         # Test set accuracy

                 ## my show images
                 image = train[0][0]
                 w_out = tf.trainable_variables()[0]
                 b_out = tf.trainable_variables()[1]
                 conv = tf.nn.conv2d(x_image, w_out, strides=[1, 1, 1, 1], padding='SAME')
                 ## conv2d : arbitrary filters that can mix channels together
                 relu = tf.nn.relu(conv + b_out)
                 plot_conv_layer(relu, image)


                 print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

                 # Save model
                 tf.add_to_collection("optimizer", train_step)
                 saver = tf.train.Saver()
                 save_path = saver.save(sess, "tmp/"+model_name)
                 print("Model saved in path: %s" % save_path)
```

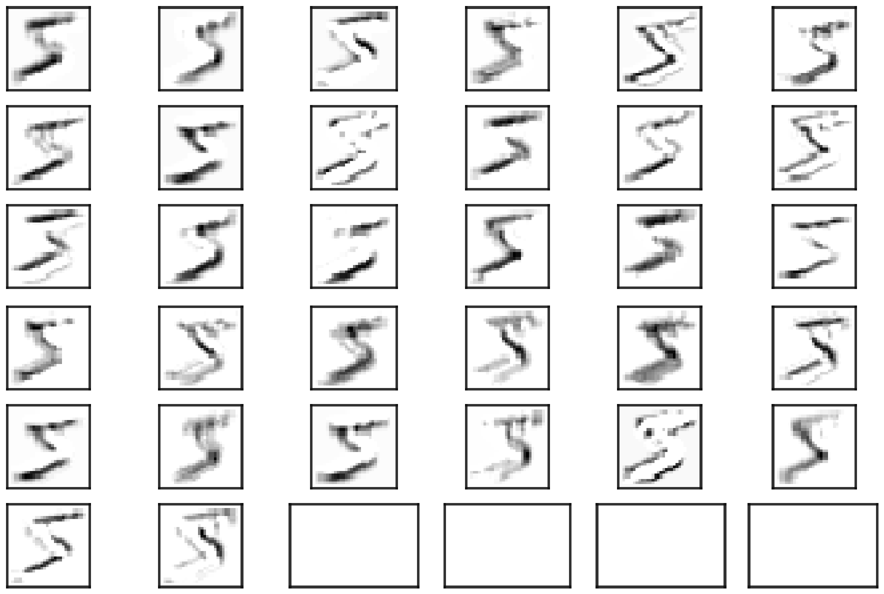
```
(70000, 784)
WARNING:tensorflow:From <ipython-input-8-db2e7185f21f>:28: softmax_cross_entropy_with_logits (
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.
```

```
See @{tf.nn.softmax_cross_entropy_with_logits_v2}.

conv1/W:0 [5, 5, 1, 32] 0.04984897
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028907983
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 256] 0.024255468
fc1/b:0 [256] 0.0
fc2/W:0 [256, 10] 0.08639823
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 1.6918435096740723
get stats time 0.353147029876709
Epoch 0 Train loss, accuracy 0.4482795285344124 0.8669
get stats time 0.2297039031982422
EPoch 0 Validation loss, accuracy 0.45575756679773327 0.8659000000000001
Batch 0.0
Batch 10.0
Epoch time 0.674262523651123
Batch 0.0
Batch 10.0
Epoch time 0.6614406108856201
get stats time 0.22439861297607422
Epoch 2 Train loss, accuracy 0.1562100731611252 0.9526000000000001
get stats time 0.21896934509277344
EPoch 2 Validation loss, accuracy 0.1684809594154358 0.9479
Batch 0.0
Batch 10.0
Epoch time 0.6527078151702881
Batch 0.0
Batch 10.0
Epoch time 0.652632474899292
get stats time 0.21477603912353516
Epoch 4 Train loss, accuracy 0.08945461488962174 0.9726000000000001
get stats time 0.21258974075317383
EPoch 4 Validation loss, accuracy 0.10038899763822554 0.9685999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6512701511383057
Batch 0.0
Batch 10.0
Epoch time 0.6593914031982422
get stats time 0.2151932716369629
Epoch 6 Train loss, accuracy 0.06329983332157134 0.9814
get stats time 0.2195911407470703
EPoch 6 Validation loss, accuracy 0.08774313745498656 0.9724999999999999
```

```
Batch 0.0
Batch 10.0
Epoch time 0.6510219573974609
Batch 0.0
Batch 10.0
Epoch time 0.6514744758605957
get stats time 0.21529626846313477
Epoch 8 Train loss, accuracy 0.05512021901607513 0.9831000000000001
get stats time 0.21350383758544922
EPoch 8 Validation loss, accuracy 0.07881852829456329 0.9753999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6511971950531006
Batch 0.0
Batch 10.0
Epoch time 0.6525740623474121
get stats time 0.2159733772277832
Epoch 10 Train loss, accuracy 0.04678899030685425 0.985
get stats time 0.21184659004211426
EPoch 10 Validation loss, accuracy 0.07548435647487639 0.9759
Batch 0.0
Batch 10.0
Epoch time 0.6535007953643799
Batch 0.0
Batch 10.0
Epoch time 0.6584107875823975
get stats time 0.21932768821716309
Epoch 12 Train loss, accuracy 0.03358711137771606 0.9899999999999999
get stats time 0.220048189163208
EPoch 12 Validation loss, accuracy 0.06388084797859192 0.9796999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6543817520141602
Batch 0.0
Batch 10.0
Epoch time 0.6529603004455566
get stats time 0.21616840362548828
Epoch 14 Train loss, accuracy 0.030431817173957827 0.9894999999999999
get stats time 0.2126619815826416
EPoch 14 Validation loss, accuracy 0.06590138413906096 0.9776000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6555366516113281
Batch 0.0
Batch 10.0
Epoch time 0.6563677787780762
get stats time 0.2152266502380371
Epoch 16 Train loss, accuracy 0.022563250541687008 0.9929
```

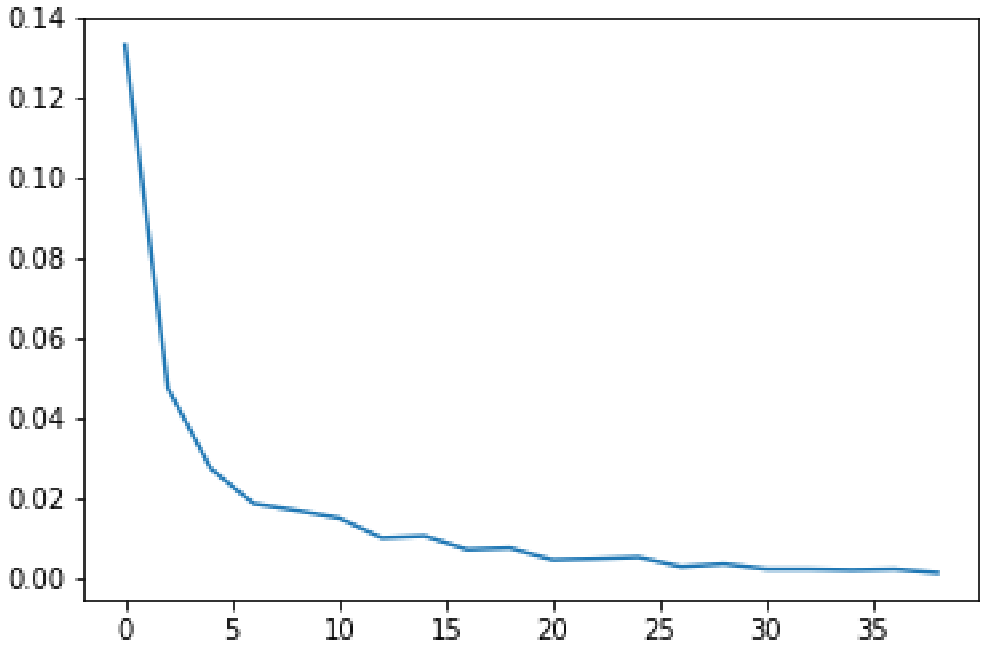
```
get stats time 0.21196556091308594
EPoch 16 Validation loss, accuracy 0.06553765685558319 0.9812000000000001
Batch 0.0
Batch 10.0
Epoch time 0.653789758682251
Batch 0.0
Batch 10.0
Epoch time 0.6543099880218506
get stats time 0.21435976028442383
Epoch 18 Train loss, accuracy 0.02600747392177582 0.9924999999999999
get stats time 0.2129073143005371
EPoch 18 Validation loss, accuracy 0.06765929424762726 0.9788
Batch 0.0
Batch 10.0
Epoch time 0.6514456272125244
Batch 0.0
Batch 10.0
Epoch time 0.6521327495574951
get stats time 0.21645402908325195
Epoch 20 Train loss, accuracy 0.01727199788093567 0.9955
get stats time 0.21244525909423828
EPoch 20 Validation loss, accuracy 0.06361159577369689 0.9817
Batch 0.0
Batch 10.0
Epoch time 0.6542301177978516
Batch 0.0
Batch 10.0
Epoch time 0.6583967208862305
get stats time 0.21743512153625488
Epoch 22 Train loss, accuracy 0.014254060769081117 0.9952
get stats time 0.2161858081817627
EPoch 22 Validation loss, accuracy 0.05893779640197755 0.9824999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6522088050842285
Batch 0.0
Batch 10.0
Epoch time 0.6521532535552979
get stats time 0.21802043914794922
Epoch 24 Train loss, accuracy 0.014752464842796326 0.9948
get stats time 0.21552467346191406
EPoch 24 Validation loss, accuracy 0.06857980673313142 0.9808999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6521406173706055
Batch 0.0
Batch 10.0
Epoch time 0.6521823406219482
```

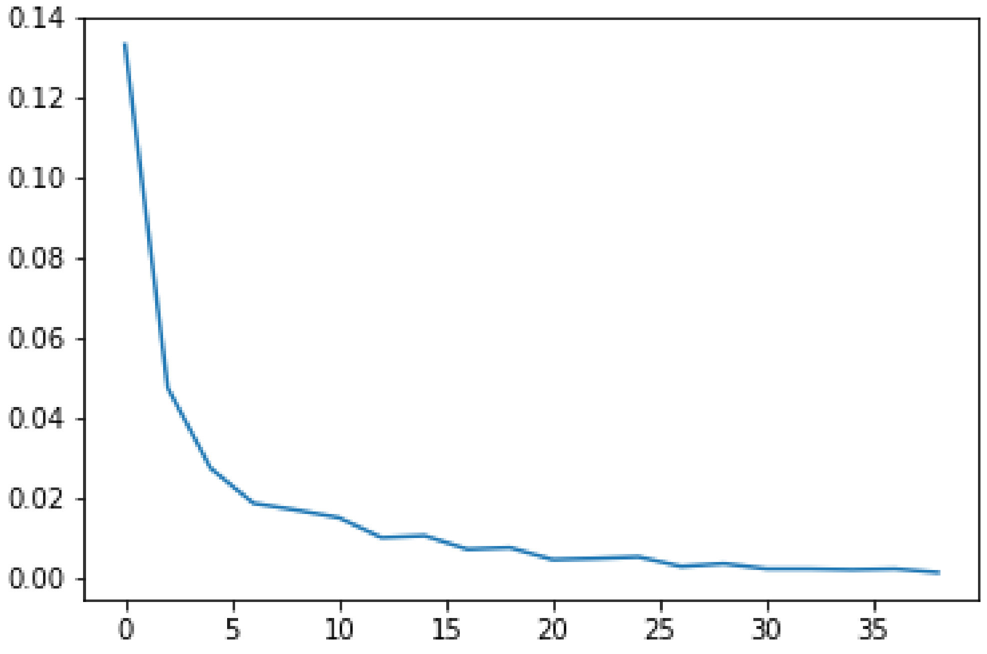
```
get stats time 0.21790528297424316
Epoch 26 Train loss, accuracy 0.00954787073135376 0.9972000000000001
get stats time 0.2140963077545166
EPoch 26 Validation loss, accuracy 0.06076204619407655 0.9815999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6509859561920166
Batch 0.0
Batch 10.0
Epoch time 0.6535987854003906
get stats time 0.21794748306274414
Epoch 28 Train loss, accuracy 0.011592423391342163 0.9964999999999999
get stats time 0.21320891380310059
EPoch 28 Validation loss, accuracy 0.058441766834259036 0.9816
Batch 0.0
Batch 10.0
Epoch time 0.6540348529815674
Batch 0.0
Batch 10.0
Epoch time 0.65488600730896
get stats time 0.21843385696411133
Epoch 30 Train loss, accuracy 0.007272921037673949 0.9977999999999998
get stats time 0.2127070426940918
EPoch 30 Validation loss, accuracy 0.06840771062374117 0.9824999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6519591808319092
Batch 0.0
Batch 10.0
Epoch time 0.6523458957672119
get stats time 0.21827197074890137
Epoch 32 Train loss, accuracy 0.007694808077812196 0.9977999999999998
get stats time 0.21379756927490234
EPoch 32 Validation loss, accuracy 0.06454321665763854 0.9828000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6524994373321533
Batch 0.0
Batch 10.0
Epoch time 0.6563489437103271
get stats time 0.21724867820739746
Epoch 34 Train loss, accuracy 0.007407559347152709 0.998
get stats time 0.21322274208068848
EPoch 34 Validation loss, accuracy 0.058300833487510695 0.9842000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6524338722229004
Batch 0.0
Batch 10.0
Epoch time 0.6532430648803711
get stats time 0.21862149238586426
Epoch 36 Train loss, accuracy 0.005976331520080567 0.9977999999999998
get stats time 0.21489405632019043
EPoch 36 Validation loss, accuracy 0.06577790241241456 0.9823000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6497631072998047
Batch 0.0
Batch 10.0
Epoch time 0.6586387157440186
get stats time 0.22445201873779297
Epoch 38 Train loss, accuracy 0.004472441387176513 0.9987
get stats time 0.2164771556854248
EPoch 38 Validation loss, accuracy 0.06159228494167328 0.9835999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6568303108215332
```



```
test accuracy 0.9834
Model saved in path: tmp/model
```

```
In [19]: times = [2*i for i in range(int(0.5*num_epochs))]
         plt.plot(times,err_train)
         plt.show()
         plt.plot(times,err_val)
         plt.show()
```





10 (b)

10.1 (i)

Since the number of features at different stages are [32,64,256,10], I reduce them by half or multiply them by two in the two models below (model_1_0 and model_1_1)

```
In [20]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         model_name="modelb_1_0"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3


         ## collect error
         err_train = []
         err_val = []

         tf.reset_default_graph()

         x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
         x_image = tf.reshape(x, [-1, dim, dim, nchannels])
         # Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
         # The number of incoming channels, for example, will be 3 if the image is color: RGB
         # We will slide filter over this 2d picture with conv2d function.
         y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
         # Allows you to control the time step during the iterations
         lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
         keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

         with tf.Session() as sess:
                 train,val,test=get_data(data_set=data_set)
                 # Create the network architecture with the above placeholdes as the inputs.
                 cross_entropy, accuracy, fc2 =my_create_network0([16,32,128,10])

                 # Define the miminization method
                 if (minimizer=="Adam"):
                     train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
                 elif (minimizer=="SGD"):
```

```
                train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
            # Initialize variables
            sess.run(tf.global_variables_initializer())
            # Show trainable variables
            for v in tf.trainable_variables():
                print(v.name,v.get_shape().as_list(),np.std(v.eval()))
            ii=np.arange(0,num_train,1) #len(train_data),1)
            # Run epochs
            for i in range(num_epochs):  # number of epochs
                run_epoch(train,val,ii,batch_size,train_step)
                if (np.mod(i,2)==0):
                    lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                    err_train.append(1-ac)
                    print('Epoch',i,'Train loss, accuracy',lo,ac)

                    vlo,vac = get_stats(val[0],val[1])

                    err_val.append(1-ac)
                    print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                    # Test set accuracy

            ## my show images
            image = train[0][0]
            w_out = tf.trainable_variables()[0]
            b_out = tf.trainable_variables()[1]
            conv = tf.nn.conv2d(x_image, w_out, strides=[1, 1, 1, 1], padding='SAME')
            ## conv2d : arbitrary filters that can mix channels together
            relu = tf.nn.relu(conv + b_out)
            plot_conv_layer(relu, image)


            print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

            # Save model
            tf.add_to_collection("optimizer", train_step)
            saver = tf.train.Saver()
            save_path = saver.save(sess, "tmp/"+model_name)
            print("Model saved in path: %s" % save_path)
```

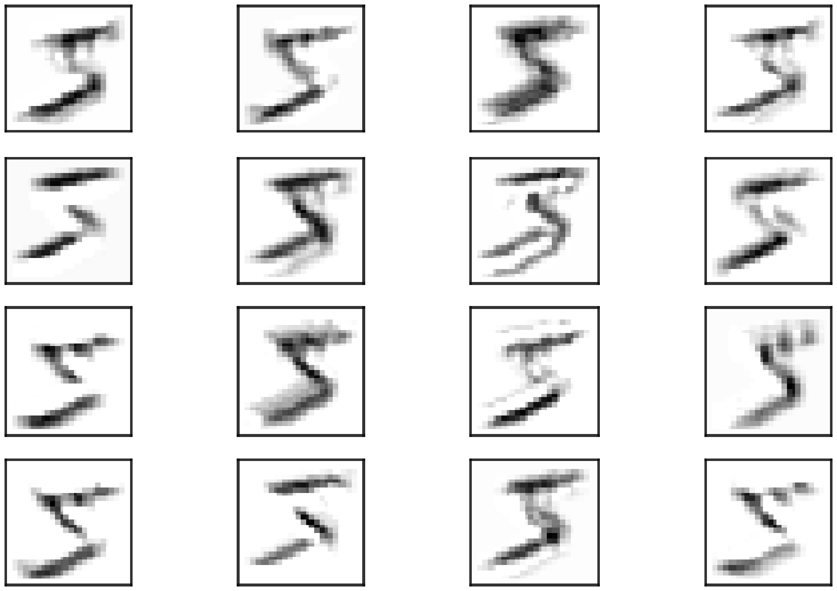
```
(70000, 784)
conv1/W:0 [5, 5, 1, 16] 0.06626024
conv1/b:0 [16] 0.0
conv2/W:0 [5, 5, 16, 32] 0.04090626
conv2/b:0 [32] 0.0
fc1/W:0 [1568, 128] 0.03430428
fc1/b:0 [128] 0.0
fc2/W:0 [128, 10] 0.11958436
```

```
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.6212716102600098
get stats time 0.19229507446289062
Epoch 0 Train loss, accuracy 0.7525206516742706 0.7529
get stats time 0.12084197998046875
EPoch 0 Validation loss, accuracy 0.7502213500976562 0.7497999999999999
Batch 0.0
Batch 10.0
Epoch time 0.3740236759185791
Batch 0.0
Batch 10.0
Epoch time 0.36102962493896484
get stats time 0.12416958808898926
Epoch 2 Train loss, accuracy 0.2909443348884583 0.9134999999999998
get stats time 0.1219792366027832
EPoch 2 Validation loss, accuracy 0.2998488209366798 0.9096
Batch 0.0
Batch 10.0
Epoch time 0.35977673530578613
Batch 0.0
Batch 10.0
Epoch time 0.3566727638244629
get stats time 0.1207122802734375
Epoch 4 Train loss, accuracy 0.1809331141233444 0.9472999999999999
get stats time 0.12103533744812012
EPoch 4 Validation loss, accuracy 0.18855550175905228 0.9448000000000001
Batch 0.0
Batch 10.0
Epoch time 0.36075782775878906
Batch 0.0
Batch 10.0
Epoch time 0.35674595832824707
get stats time 0.12051081657409668
Epoch 6 Train loss, accuracy 0.1425252720594406 0.9559
get stats time 0.12079572677612305
EPoch 6 Validation loss, accuracy 0.1627442553639412 0.9490999999999999
Batch 0.0
Batch 10.0
Epoch time 0.3552682399749756
Batch 0.0
Batch 10.0
Epoch time 0.3541452884674072
get stats time 0.11972761154174805
Epoch 8 Train loss, accuracy 0.10384687459468842 0.9665999999999999
get stats time 0.12057065963745117
EPoch 8 Validation loss, accuracy 0.12003101217746737 0.9643
```

```
Batch 0.0
Batch 10.0
Epoch time 0.3552694320678711
Batch 0.0
Batch 10.0
Epoch time 0.3572964668273926
get stats time 0.12012362480163574
Epoch 10 Train loss, accuracy 0.09017848572731019 0.9731
get stats time 0.1208798885345459
EPoch 10 Validation loss, accuracy 0.10854688972234725 0.9660999999999997
Batch 0.0
Batch 10.0
Epoch time 0.3568544387817383
Batch 0.0
Batch 10.0
Epoch time 0.35517430305480957
get stats time 0.12023687362670898
Epoch 12 Train loss, accuracy 0.07179776480197907 0.9785999999999999
get stats time 0.1203620433807373
EPoch 12 Validation loss, accuracy 0.09997841532230375 0.9680999999999997
Batch 0.0
Batch 10.0
Epoch time 0.3548412322998047
Batch 0.0
Batch 10.0
Epoch time 0.35648083686828613
get stats time 0.12100601196289062
Epoch 14 Train loss, accuracy 0.06532896279096603 0.9801
get stats time 0.12092995643615723
EPoch 14 Validation loss, accuracy 0.09229900360107421 0.9700999999999999
Batch 0.0
Batch 10.0
Epoch time 0.3585681915283203
Batch 0.0
Batch 10.0
Epoch time 0.35608506202697754
get stats time 0.1206197738647461
Epoch 16 Train loss, accuracy 0.058720227861404406 0.9802000000000002
get stats time 0.1209416389465332
EPoch 16 Validation loss, accuracy 0.08921795258522033 0.9706999999999999
Batch 0.0
Batch 10.0
Epoch time 0.3567180633544922
Batch 0.0
Batch 10.0
Epoch time 0.35553455352783203
get stats time 0.12042522430419922
Epoch 18 Train loss, accuracy 0.04750251032114029 0.9853
```

```
get stats time 0.12102484703063965
EPoch 18 Validation loss, accuracy 0.08174146409034728 0.9734
Batch 0.0
Batch 10.0
Epoch time 0.3578317165374756
Batch 0.0
Batch 10.0
Epoch time 0.3560199737548828
get stats time 0.12071752548217773
Epoch 20 Train loss, accuracy 0.049000501418113705 0.9846999999999999
get stats time 0.12111878395080566
EPoch 20 Validation loss, accuracy 0.08618371381759643 0.9714
Batch 0.0
Batch 10.0
Epoch time 0.35845208168029785
Batch 0.0
Batch 10.0
Epoch time 0.35630273818969727
get stats time 0.12025070190429688
Epoch 22 Train loss, accuracy 0.04241529889106751 0.986
get stats time 0.12088418006896973
EPoch 22 Validation loss, accuracy 0.07616112322807311 0.9728
Batch 0.0
Batch 10.0
Epoch time 0.36042118072509766
Batch 0.0
Batch 10.0
Epoch time 0.35579729080200195
get stats time 0.12055230140686035
Epoch 24 Train loss, accuracy 0.039133616447448734 0.9879000000000001
get stats time 0.12079596519470215
EPoch 24 Validation loss, accuracy 0.0770701864719391 0.9763999999999999
Batch 0.0
Batch 10.0
Epoch time 0.35527610778808594
Batch 0.0
Batch 10.0
Epoch time 0.35608935356140137
get stats time 0.12051963806152344
Epoch 26 Train loss, accuracy 0.03762646048069 0.9873
get stats time 0.12102818489074707
EPoch 26 Validation loss, accuracy 0.0774847933769226 0.9736999999999998
Batch 0.0
Batch 10.0
Epoch time 0.35822463035583496
Batch 0.0
Batch 10.0
Epoch time 0.35494565963745117
```

```
get stats time 0.12091660499572754
Epoch 28 Train loss, accuracy 0.028830539345741275 0.9898
get stats time 0.1207418441772461
EPoch 28 Validation loss, accuracy 0.07402841384410859 0.9771999999999998
Batch 0.0
Batch 10.0
Epoch time 0.3536081314086914
Batch 0.0
Batch 10.0
Epoch time 0.3546328544616699
get stats time 0.12050437927246094
Epoch 30 Train loss, accuracy 0.034174067735672 0.9895999999999999
get stats time 0.12086057662963867
EPoch 30 Validation loss, accuracy 0.08359914059638976 0.9734999999999999
Batch 0.0
Batch 10.0
Epoch time 0.35818934440612793
Batch 0.0
Batch 10.0
Epoch time 0.35802698135375977
get stats time 0.1207892894744873
Epoch 32 Train loss, accuracy 0.028259380793571477 0.9913999999999998
get stats time 0.12092113494873047
EPoch 32 Validation loss, accuracy 0.07391517834663393 0.9757
Batch 0.0
Batch 10.0
Epoch time 0.3577132225036621
Batch 0.0
Batch 10.0
Epoch time 0.35662102699279785
get stats time 0.1214597225189209
Epoch 34 Train loss, accuracy 0.022712325739860536 0.9934999999999998
get stats time 0.1244497299194336
EPoch 34 Validation loss, accuracy 0.06679591224193573 0.9794999999999998
Batch 0.0
Batch 10.0
Epoch time 0.35622262954711914
Batch 0.0
Batch 10.0
Epoch time 0.3559386730194092
get stats time 0.11994385719299316
Epoch 36 Train loss, accuracy 0.02047781512737274 0.9936
get stats time 0.12051844596862793
EPoch 36 Validation loss, accuracy 0.06796779913902282 0.9792
Batch 0.0
Batch 10.0
Epoch time 0.35582923889160156
Batch 0.0
Batch 10.0
Epoch time 0.3582286834716797
get stats time 0.12098145484924316
Epoch 38 Train loss, accuracy 0.021336458277702332 0.9935
get stats time 0.1215050220489502
EPoch 38 Validation loss, accuracy 0.0743234480381012 0.9783
Batch 0.0
Batch 10.0
Epoch time 0.355318546295166
```



```
test accuracy 0.9805
Model saved in path: tmp/modelb_1_0
```

In [21]:
```
## run the model and plot error
import time
batch_size=500
step_size=.001
num_epochs=40
num_train=10000
minimizer="Adam"
data_set="mnist"
model_name="modelb_1_1"
keep_prob=.5
dim=28
nchannels=1
```

```
if (data_set=="cifar"):
    dim=32
    nchannels=3


## collect error
err_train = []
err_val = []

tf.reset_default_graph()

x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
x_image = tf.reshape(x, [-1, dim, dim, nchannels])
# Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
# The number of incoming channels, for example, will be 3 if the image is color: RGB
# We will slide filter over this 2d picture with conv2d function.
y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
# Allows you to control the time step during the iterations
lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

with tf.Session() as sess:
        train,val,test=get_data(data_set=data_set)
        # Create the network architecture with the above placeholdes as the inputs.
        cross_entropy, accuracy, fc2 =my_create_network0([64,128,512,10])

        # Define the miminization method
        if (minimizer=="Adam"):
            train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
        elif (minimizer=="SGD"):
            train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
        # Initialize variables
        sess.run(tf.global_variables_initializer())
        # Show trainable variables
        for v in tf.trainable_variables():
            print(v.name,v.get_shape().as_list(),np.std(v.eval()))
        ii=np.arange(0,num_train,1) #len(train_data),1)
        # Run epochs
        for i in range(num_epochs):  # number of epochs
            run_epoch(train,val,ii,batch_size,train_step)
            if (np.mod(i,2)==0):
                lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                err_train.append(1-ac)
                print('Epoch',i,'Train loss, accuracy',lo,ac)

                vlo,vac = get_stats(val[0],val[1])
                err_val.append(1-ac)
                print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                # Test set accuracy



        print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

        # Save model
        tf.add_to_collection("optimizer", train_step)
        saver = tf.train.Saver()
        save_path = saver.save(sess, "tmp/"+model_name)
        print("Model saved in path: %s" % save_path)
```

```
(70000, 784)
conv1/W:0 [5, 5, 1, 64] 0.034763683
conv1/b:0 [64] 0.0
conv2/W:0 [5, 5, 64, 128] 0.020439275
conv2/b:0 [128] 0.0
fc1/W:0 [6272, 512] 0.017165015
fc1/b:0 [512] 0.0
fc2/W:0 [512, 10] 0.061925355
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 1.9501206874847412
get stats time 0.7524991035461426
Epoch 0 Train loss, accuracy 0.3176660941243171 0.9009
get stats time 0.47524309158325195
EPoch 0 Validation loss, accuracy 0.3178613252520562 0.9004
Batch 0.0
Batch 10.0
Epoch time 1.2778542041778564
Batch 0.0
Batch 10.0
Epoch time 1.2798724174499512
get stats time 0.4716768264770508
Epoch 2 Train loss, accuracy 0.09488893461227418 0.9702
get stats time 0.47082066535949707
EPoch 2 Validation loss, accuracy 0.1067925462603569 0.9654999999999999
Batch 0.0
Batch 10.0
Epoch time 1.2794301509857178
Batch 0.0
Batch 10.0
Epoch time 1.2821762561798096
get stats time 0.475297212600708
Epoch 4 Train loss, accuracy 0.0559228479385376 0.9808999999999999
```

```
get stats time 0.47009849548339844
EPoch 4 Validation loss, accuracy 0.07610997134447098 0.9757
Batch 0.0
Batch 10.0
Epoch time 1.281306266784668
Batch 0.0
Batch 10.0
Epoch time 1.2799336910247803
get stats time 0.4756169319152832
Epoch 6 Train loss, accuracy 0.03037374705076218 0.9911999999999999
get stats time 0.4731333255767822
EPoch 6 Validation loss, accuracy 0.06013475439548492 0.9800000000000001
Batch 0.0
Batch 10.0
Epoch time 1.2844631671905518
Batch 0.0
Batch 10.0
Epoch time 1.2816519737243652
get stats time 0.474515438079834
Epoch 8 Train loss, accuracy 0.01967305173873901 0.9930999999999999
get stats time 0.4732859134674072
EPoch 8 Validation loss, accuracy 0.05932434637546539 0.9802
Batch 0.0
Batch 10.0
Epoch time 1.2849233150482178
Batch 0.0
Batch 10.0
Epoch time 1.283308506011963
get stats time 0.47557687759399414
Epoch 10 Train loss, accuracy 0.019602958583831785 0.9935
get stats time 0.469637393951416
EPoch 10 Validation loss, accuracy 0.0529415201663971 0.9837
Batch 0.0
Batch 10.0
Epoch time 1.282069444656372
Batch 0.0
Batch 10.0
Epoch time 1.29374098777771
get stats time 0.47672009468078613
Epoch 12 Train loss, accuracy 0.011519347405433655 0.9974000000000001
get stats time 0.47008705139160156
EPoch 12 Validation loss, accuracy 0.05527172523736954 0.9815999999999999
Batch 0.0
Batch 10.0
Epoch time 1.2838752269744873
Batch 0.0
Batch 10.0
Epoch time 1.2854256629943848
```

```
get stats time 0.4746816158294678
Epoch 14 Train loss, accuracy 0.010237732505798338 0.9978
get stats time 0.4728524684906006
EPoch 14 Validation loss, accuracy 0.05717179877758026 0.9831000000000001
Batch 0.0
Batch 10.0
Epoch time 1.2810132503509521
Batch 0.0
Batch 10.0
Epoch time 1.2843396663665771
get stats time 0.4770998954772949
Epoch 16 Train loss, accuracy 0.00825893805027008 0.9975999999999999
get stats time 0.473494291305542
EPoch 16 Validation loss, accuracy 0.055733454680442815 0.9837
Batch 0.0
Batch 10.0
Epoch time 1.2828691005706787
Batch 0.0
Batch 10.0
Epoch time 1.2846009731292725
get stats time 0.4752688407897949
Epoch 18 Train loss, accuracy 0.007139792251586914 0.9976
get stats time 0.47396016120910645
EPoch 18 Validation loss, accuracy 0.05634791173934937 0.9846000000000001
Batch 0.0
Batch 10.0
Epoch time 1.2828283309936523
Batch 0.0
Batch 10.0
Epoch time 1.2837300300598145
get stats time 0.47623777389526367
Epoch 20 Train loss, accuracy 0.006470523977279663 0.9978
get stats time 0.47374629974365234
EPoch 20 Validation loss, accuracy 0.06254597163200379 0.9823000000000001
Batch 0.0
Batch 10.0
Epoch time 1.2864036560058594
Batch 0.0
Batch 10.0
Epoch time 1.2852261066436768
get stats time 0.4756743907928467
Epoch 22 Train loss, accuracy 0.005414007544517517 0.9981
get stats time 0.47394633293151855
EPoch 22 Validation loss, accuracy 0.057287541007995615 0.9834999999999999
Batch 0.0
Batch 10.0
Epoch time 1.282198190689087
Batch 0.0
Batch 10.0
Epoch time 1.2878408432006836
get stats time 0.4764702320098877
Epoch 24 Train loss, accuracy 0.005859805488586426 0.9981
get stats time 0.4755845069885254
EPoch 24 Validation loss, accuracy 0.06103538811206817 0.9831
Batch 0.0
Batch 10.0
Epoch time 1.2843725681304932
Batch 0.0
Batch 10.0
Epoch time 1.284433364868164
get stats time 0.47690510749816895
Epoch 26 Train loss, accuracy 0.003769154262542725 0.999
get stats time 0.4737699031829834
EPoch 26 Validation loss, accuracy 0.05711698832511901 0.9844999999999999
Batch 0.0
Batch 10.0
Epoch time 1.2845990657806396
Batch 0.0
Batch 10.0
Epoch time 1.2895429134368896
get stats time 0.47609710693359375
Epoch 28 Train loss, accuracy 0.006652981901168824 0.9975999999999999
get stats time 0.47356319427490234
EPoch 28 Validation loss, accuracy 0.06333007197380067 0.9831
Batch 0.0
Batch 10.0
Epoch time 1.2836220264434814
Batch 0.0
Batch 10.0
Epoch time 1.285470724105835
get stats time 0.4810338020324707
Epoch 30 Train loss, accuracy 0.005789741849899293 0.9982
get stats time 0.47414112091064453
EPoch 30 Validation loss, accuracy 0.06972173326015471 0.9831
Batch 0.0
Batch 10.0
Epoch time 1.2967860698699951
Batch 0.0
Batch 10.0
Epoch time 1.2859926223754883
get stats time 0.4761958122253418
Epoch 32 Train loss, accuracy 0.00491790690422058 0.9984
get stats time 0.47564172744750977
EPoch 32 Validation loss, accuracy 0.06456809659004212 0.9827000000000001
Batch 0.0
Batch 10.0
```

```
Epoch time 1.2845723628997803
Batch 0.0
Batch 10.0
Epoch time 1.2855048179626465
get stats time 0.47529101371765137
Epoch 34 Train loss, accuracy 0.0051591493129730225 0.9983000000000001
get stats time 0.47563838958740234
EPoch 34 Validation loss, accuracy 0.0646334686756134 0.9836
Batch 0.0
Batch 10.0
Epoch time 1.288863182067871
Batch 0.0
Batch 10.0
Epoch time 1.2882392406463623
get stats time 0.47644567489624023
Epoch 36 Train loss, accuracy 0.005763577222824097 0.9981
get stats time 0.4827554225921631
EPoch 36 Validation loss, accuracy 0.08363681225776672 0.9810000000000001
Batch 0.0
Batch 10.0
Epoch time 1.2866637706756592
Batch 0.0
Batch 10.0
Epoch time 1.2887599468231201
get stats time 0.47700047492980957
Epoch 38 Train loss, accuracy 0.0032255837917327885 0.9985999999999999
get stats time 0.4761943817138672
EPoch 38 Validation loss, accuracy 0.06340068440437316 0.9841999999999999
Batch 0.0
Batch 10.0
Epoch time 1.2868571281433105
test accuracy 0.9813
Model saved in path: tmp/modelb_1_1
```

10.1.1 result:

The test error rate in the reduced model (model_1_0) is 0.9783, while in the multiplied model (model_1_1) it is 0.9833. As a comparison the orginal model it is 0.9824. So the more parameters in this case improve the test accuracy a little bit. However, more parameters also means more time in each epoch.

10.2 (ii)

Below I tried the following architecture: [32,64,240,210,10], [32,64,224,224,224,10], [32,64,256,224,224,10] (the first two keep the same number of parameters, while the last one has more parameters)(the numbers are the number of features in one layer)

```
In [30]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         model_name="model_deeper_0"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3


         ## collect error
         err_train = []
         err_val = []

         tf.reset_default_graph()

         x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
         x_image = tf.reshape(x, [-1, dim, dim, nchannels])
         # Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
         # The number of incoming channels, for example, will be 3 if the image is color: RGB
         # We will slide filter over this 2d picture with conv2d function.
         y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
         # Allows you to control the time step during the iterations
         lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
         keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

         with tf.Session() as sess:
                 train,val,test=get_data(data_set=data_set)
                 # Create the network architecture with the above placeholdes as the inputs.
                 cross_entropy, accuracy, fc2 =my_create_network_deeper([32,64,240,210,10])

                 # Define the miminization method
                 if (minimizer=="Adam"):
                     train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entrop
                 elif (minimizer=="SGD"):
                     train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
                 # Initialize variables
                 sess.run(tf.global_variables_initializer())
                 # Show trainable variables
                 for v in tf.trainable_variables():
                     print(v.name,v.get_shape().as_list(),np.std(v.eval()))
                 ii=np.arange(0,num_train,1) #len(train_data),1)
                 # Run epochs
                 for i in range(num_epochs):  # number of epochs
                     run_epoch(train,val,ii,batch_size,train_step)
                     if (np.mod(i,2)==0):
                         lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                         err_train.append(1-ac)
                         print('Epoch',i,'Train loss, accuracy',lo,ac)

                         vlo,vac = get_stats(val[0],val[1])

                         err_val.append(1-ac)
                         print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                         # Test set accuracy

                 ## my show images
                 image = train[0][0]
                 w_out = tf.trainable_variables()[0]
                 b_out = tf.trainable_variables()[1]
                 conv = tf.nn.conv2d(x_image, w_out, strides=[1, 1, 1, 1], padding='SAME')
                 ## conv2d : arbitrary filters that can mix channels together
                 relu = tf.nn.relu(conv + b_out)
                 plot_conv_layer(relu, image)


                 print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

                 # Save model
                 tf.add_to_collection("optimizer", train_step)
                 saver = tf.train.Saver()
                 save_path = saver.save(sess, "tmp/"+model_name)
                 print("Model saved in path: %s" % save_path)
```

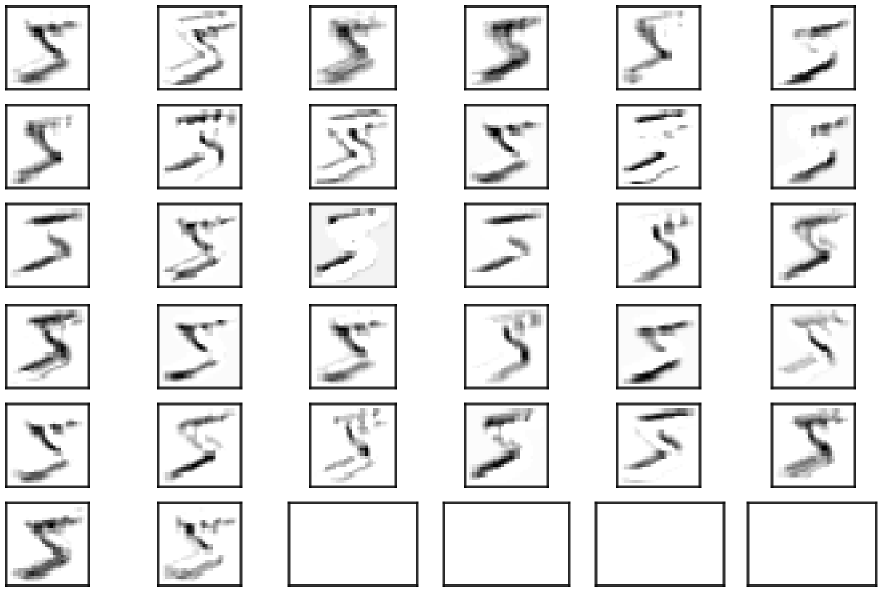
```
(70000, 784)
conv1/W:0 [5, 5, 1, 32] 0.048800137
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.02887622
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 240] 0.024336992
fc1/b:0 [240] 0.0
fc2/W:0 [240, 210] 0.066787235
fc2/b:0 [210] 0.0
fc3/W:0 [210, 10] 0.094975024
fc3/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.8936588764190674
```

```
get stats time 0.2430107593536377
Epoch 0 Train loss, accuracy 0.43547355686426165 0.8698
get stats time 0.2194356918334961
EPoch 0 Validation loss, accuracy 0.4485174379587174 0.8694999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6781699657440186
Batch 0.0
Batch 10.0
Epoch time 0.6625754833221436
get stats time 0.21741676330566406
Epoch 2 Train loss, accuracy 0.17471903624534607 0.9461999999999999
get stats time 0.2167205810546875
EPoch 2 Validation loss, accuracy 0.1832666418194771 0.9408999999999998
Batch 0.0
Batch 10.0
Epoch time 0.655937671661377
Batch 0.0
Batch 10.0
Epoch time 0.6581921577453613
get stats time 0.21680355072021484
Epoch 4 Train loss, accuracy 0.09481891448497773 0.9727999999999998
get stats time 0.21612787246704102
EPoch 4 Validation loss, accuracy 0.10538156999349595 0.9676
Batch 0.0
Batch 10.0
Epoch time 0.6520516872406006
Batch 0.0
Batch 10.0
Epoch time 0.6547667980194092
get stats time 0.21637344360351562
Epoch 6 Train loss, accuracy 0.06525173903703689 0.9796999999999999
get stats time 0.21730709075927734
EPoch 6 Validation loss, accuracy 0.08426662608385085 0.9728999999999999
Batch 0.0
Batch 10.0
Epoch time 0.656898021697998
Batch 0.0
Batch 10.0
Epoch time 0.6576991081237793
get stats time 0.216386079788208
Epoch 8 Train loss, accuracy 0.050072873687744136 0.9846999999999999
get stats time 0.21652698516845703
EPoch 8 Validation loss, accuracy 0.07369180924892427 0.9753000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6548287868499756
Batch 0.0
Batch 10.0
Epoch time 0.6537327766418457
get stats time 0.21652722358703613
Epoch 10 Train loss, accuracy 0.04219155458211899 0.9867000000000001
get stats time 0.2171492576599121
EPoch 10 Validation loss, accuracy 0.07239775682687759 0.9758999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6589059829711914
Batch 0.0
Batch 10.0
Epoch time 0.656524658203125
get stats time 0.21686887741088867
Epoch 12 Train loss, accuracy 0.04618773815631867 0.9847000000000001
get stats time 0.21713614463806152
EPoch 12 Validation loss, accuracy 0.0810712292432785 0.9747
Batch 0.0
Batch 10.0
Epoch time 0.6550590991973877
Batch 0.0
Batch 10.0
Epoch time 0.6530137062072754
get stats time 0.21737027168273926
Epoch 14 Train loss, accuracy 0.02263558249473572 0.9924999999999999
get stats time 0.21832895278930664
EPoch 14 Validation loss, accuracy 0.06217178707122804 0.9797
Batch 0.0
Batch 10.0
Epoch time 0.6584601402282715
Batch 0.0
Batch 10.0
Epoch time 0.6557984352111816
get stats time 0.21700525283813477
Epoch 16 Train loss, accuracy 0.021758823752403258 0.9932000000000001
get stats time 0.21852445602416992
EPoch 16 Validation loss, accuracy 0.06025446310043335 0.9803999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6547954082489014
Batch 0.0
Batch 10.0
Epoch time 0.6531960964202881
get stats time 0.21497273445129395
Epoch 18 Train loss, accuracy 0.018917369580268857 0.9937000000000001
get stats time 0.21449637413024902
EPoch 18 Validation loss, accuracy 0.061777390098571784 0.9811000000000002
Batch 0.0
Batch 10.0
```

```
Epoch time 0.6560194492340088
Batch 0.0
Batch 10.0
Epoch time 0.6554601192474365
get stats time 0.21814441680908203
Epoch 20 Train loss, accuracy 0.016526622819900517 0.9946000000000002
get stats time 0.21808242797851562
EPoch 20 Validation loss, accuracy 0.0679818647861481 0.9808999999999998
Batch 0.0
Batch 10.0
Epoch time 0.673001766204834
Batch 0.0
Batch 10.0
Epoch time 0.6588966846466064
get stats time 0.21759700775146484
Epoch 22 Train loss, accuracy 0.019512956309318544 0.993
get stats time 0.21780800819396973
EPoch 22 Validation loss, accuracy 0.07072707335948944 0.9791000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6592168807983398
Batch 0.0
Batch 10.0
Epoch time 0.6575291156768799
get stats time 0.21782612800598145
Epoch 24 Train loss, accuracy 0.010692953181266783 0.9969000000000001
get stats time 0.21653223037719727
EPoch 24 Validation loss, accuracy 0.06209315567016601 0.9827
Batch 0.0
Batch 10.0
Epoch time 0.655785083770752
Batch 0.0
Batch 10.0
Epoch time 0.6545262336730957
get stats time 0.21657609939575195
Epoch 26 Train loss, accuracy 0.014767927622795104 0.9946999999999999
get stats time 0.21701550483703613
EPoch 26 Validation loss, accuracy 0.07130506813526154 0.9808999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6588640213012695
Batch 0.0
Batch 10.0
Epoch time 0.6611990928649902
get stats time 0.21671581268310547
Epoch 28 Train loss, accuracy 0.009011694979667664 0.9967
get stats time 0.22576165199279785
EPoch 28 Validation loss, accuracy 0.06571407408714296 0.9808
```

```
Batch 0.0
Batch 10.0
Epoch time 0.6572566032409668
Batch 0.0
Batch 10.0
Epoch time 0.65867018699646
get stats time 0.21471285820007324
Epoch 30 Train loss, accuracy 0.008629247379302978 0.9973000000000001
get stats time 0.21500492095947266
EPoch 30 Validation loss, accuracy 0.06907319062948226 0.9817999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6553187370300293
Batch 0.0
Batch 10.0
Epoch time 0.6583287715911865
get stats time 0.21897554397583008
Epoch 32 Train loss, accuracy 0.008087055587768555 0.9971
get stats time 0.21844053268432617
EPoch 32 Validation loss, accuracy 0.06813856151103972 0.9817
Batch 0.0
Batch 10.0
Epoch time 0.6592066287994385
Batch 0.0
Batch 10.0
Epoch time 0.6603958606719971
get stats time 0.22032833099365234
Epoch 34 Train loss, accuracy 0.005399570322036743 0.998
get stats time 0.21735405921936035
EPoch 34 Validation loss, accuracy 0.059643136119842524 0.9836
Batch 0.0
Batch 10.0
Epoch time 0.6595263481140137
Batch 0.0
Batch 10.0
Epoch time 0.6586761474609375
get stats time 0.21714186668395996
Epoch 36 Train loss, accuracy 0.00684496455192566 0.9978
get stats time 0.2184309959411621
EPoch 36 Validation loss, accuracy 0.0580956463098526 0.9851999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6609392166137695
Batch 0.0
Batch 10.0
Epoch time 0.6598951816558838
get stats time 0.2172093391418457
Epoch 38 Train loss, accuracy 0.00772629177570343 0.9974999999999999
```

```
get stats time 0.21775484085083008
EPoch 38 Validation loss, accuracy 0.06203665122985841 0.9826
Batch 0.0
Batch 10.0
Epoch time 0.6552743911743164
```



```
test accuracy 0.9836
Model saved in path: tmp/model_deeper_0
```

```
In [33]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         model_name="model_deeper_1"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3
```

```python
## collect error
err_train = []
err_val = []

tf.reset_default_graph()

x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
x_image = tf.reshape(x, [-1, dim, dim, nchannels])
# Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
# The number of incoming channels, for example, will be 3 if the image is color: RGB
# We will slide filter over this 2d picture with conv2d function.
y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
# Allows you to control the time step during the iterations
lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

with tf.Session() as sess:
        train,val,test=get_data(data_set=data_set)
        # Create the network architecture with the above placeholdes as the inputs.
        cross_entropy, accuracy, fc2 =my_create_network_deeper2([32,64,224,224,224,10

        # Define the miminization method
        if (minimizer=="Adam"):
            train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
        elif (minimizer=="SGD"):
            train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
        # Initialize variables
        sess.run(tf.global_variables_initializer())
        # Show trainable variables
        for v in tf.trainable_variables():
            print(v.name,v.get_shape().as_list(),np.std(v.eval()))
        ii=np.arange(0,num_train,1) #len(train_data),1)
        # Run epochs
        for i in range(num_epochs):  # number of epochs
            run_epoch(train,val,ii,batch_size,train_step)
            if (np.mod(i,2)==0):
                lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                err_train.append(1-ac)
                print('Epoch',i,'Train loss, accuracy',lo,ac)

                vlo,vac = get_stats(val[0],val[1])

                err_val.append(1-ac)
                print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                # Test set accuracy
                print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))
```

```
(70000, 784)
conv1/W:0 [5, 5, 1, 32] 0.04998337
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028936958
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 224] 0.024404064
fc1/b:0 [224] 0.0
fc2/W:0 [224, 224] 0.06666557
fc2/b:0 [224] 0.0
fc2_2/W:0 [224, 224] 0.066686116
fc2_2/b:0 [224] 0.0
fc3/W:0 [224, 10] 0.09107214
fc3/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.8971526622772217
get stats time 0.2515430450439453
Epoch 0 Train loss, accuracy 0.49649948761463164 0.8471
get stats time 0.2220289707183838
EPoch 0 Validation loss, accuracy 0.5162094485163689 0.843
Batch 0.0
Batch 10.0
Epoch time 0.6812534332275391
Batch 0.0
Batch 10.0
Epoch time 0.6756200790405273
get stats time 0.21800589561462402
Epoch 2 Train loss, accuracy 0.16770881843566895 0.9517
get stats time 0.21880030632019043
EPoch 2 Validation loss, accuracy 0.1752091443181038 0.9479999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6683459281921387
Batch 0.0
Batch 10.0
Epoch time 0.6671011447906494
get stats time 0.2179713249206543
Epoch 4 Train loss, accuracy 0.09747653061151505 0.9705
get stats time 0.2179737091064453
EPoch 4 Validation loss, accuracy 0.11401975394487378 0.9638999999999998
Batch 0.0
Batch 10.0
Epoch time 0.666496753692627
Batch 0.0
Batch 10.0
```

```
Epoch time 0.6641213893890381
get stats time 0.2200629711151123
Epoch 6 Train loss, accuracy 0.09374170308113097 0.9705999999999998
get stats time 0.21728110313415527
EPoch 6 Validation loss, accuracy 0.11922378741502762 0.9625
Batch 0.0
Batch 10.0
Epoch time 0.6619188785552979
Batch 0.0
Batch 10.0
Epoch time 0.6636552810668945
get stats time 0.2191791534423828
Epoch 8 Train loss, accuracy 0.04981110008955002 0.9839
get stats time 0.2187175750732422
EPoch 8 Validation loss, accuracy 0.08014370274543763 0.9718
Batch 0.0
Batch 10.0
Epoch time 0.6612768173217773
Batch 0.0
Batch 10.0
Epoch time 0.662020206451416
get stats time 0.22057080268859863
Epoch 10 Train loss, accuracy 0.0374714403629303 0.9873999999999998
get stats time 0.21855831146240234
EPoch 10 Validation loss, accuracy 0.07271314685344696 0.9781000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6658120155334473
Batch 0.0
Batch 10.0
Epoch time 0.6671426296234131
get stats time 0.2211289405822754
Epoch 12 Train loss, accuracy 0.029803488087654113 0.99
get stats time 0.21700549125671387
EPoch 12 Validation loss, accuracy 0.0664085797905922 0.9794999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6613664627075195
Batch 0.0
Batch 10.0
Epoch time 0.6632969379425049
get stats time 0.21989774703979492
Epoch 14 Train loss, accuracy 0.0320343316078186 0.9888999999999999
get stats time 0.21789789199829102
EPoch 14 Validation loss, accuracy 0.07869283757209779 0.9762000000000002
Batch 0.0
Batch 10.0
Epoch time 0.6666519641876221
```

```
Batch 0.0
Batch 10.0
Epoch time 0.6692156791687012
get stats time 0.2179853916168213
Epoch 16 Train loss, accuracy 0.02110619270801544 0.9924
get stats time 0.2161853313446045
EPoch 16 Validation loss, accuracy 0.07012098166942597 0.9791000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6605215072631836
Batch 0.0
Batch 10.0
Epoch time 0.6619253158569336
get stats time 0.21547222137451172
Epoch 18 Train loss, accuracy 0.023360618424415586 0.992
get stats time 0.2150406837463379
EPoch 18 Validation loss, accuracy 0.07353776867389679 0.9765
Batch 0.0
Batch 10.0
Epoch time 0.6606240272521973
Batch 0.0
Batch 10.0
Epoch time 0.6608943939208984
get stats time 0.214646577835083
Epoch 20 Train loss, accuracy 0.017948216724395755 0.9949999999999999
get stats time 0.21534109115600586
EPoch 20 Validation loss, accuracy 0.06607638676166536 0.9792
Batch 0.0
Batch 10.0
Epoch time 0.6613764762878418
Batch 0.0
Batch 10.0
Epoch time 0.6626486778259277
get stats time 0.21741843223571777
Epoch 22 Train loss, accuracy 0.01249481918811798 0.9960000000000001
get stats time 0.21769046783447266
EPoch 22 Validation loss, accuracy 0.06091844625473022 0.9809999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6676583290100098
Batch 0.0
Batch 10.0
Epoch time 0.666736364364624
get stats time 0.21785998344421387
Epoch 24 Train loss, accuracy 0.011647874593734741 0.9962999999999997
get stats time 0.21611714363098145
EPoch 24 Validation loss, accuracy 0.07448943810462952 0.9789999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6626477241516113
Batch 0.0
Batch 10.0
Epoch time 0.6628212928771973
get stats time 0.21531939506530762
Epoch 26 Train loss, accuracy 0.010416230487823485 0.9955999999999999
get stats time 0.21535468101501465
EPoch 26 Validation loss, accuracy 0.05899771130084992 0.9818999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6628940105438232
Batch 0.0
Batch 10.0
Epoch time 0.6623966693878174
get stats time 0.21498823165893555
Epoch 28 Train loss, accuracy 0.009260671496391296 0.9968999999999999
get stats time 0.2152235507965088
EPoch 28 Validation loss, accuracy 0.06327804317474364 0.9829000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6630587577819824
Batch 0.0
Batch 10.0
Epoch time 0.6651792526245117
get stats time 0.21756958961486816
Epoch 30 Train loss, accuracy 0.012496077871322632 0.9952999999999997
get stats time 0.2186424732208252
EPoch 30 Validation loss, accuracy 0.06897561173439026 0.9800999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6760175228118896
Batch 0.0
Batch 10.0
Epoch time 0.6820359230041504
get stats time 0.2193593978881836
Epoch 32 Train loss, accuracy 0.013393122553825377 0.9949
get stats time 0.21976971626281738
EPoch 32 Validation loss, accuracy 0.07238120484352113 0.9803
Batch 0.0
Batch 10.0
Epoch time 0.6660966873168945
Batch 0.0
Batch 10.0
Epoch time 0.6644444465637207
get stats time 0.21532964706420898
Epoch 34 Train loss, accuracy 0.009112470293045044 0.9969999999999999
get stats time 0.21579837799072266
```

```
EPoch 34 Validation loss, accuracy 0.06959240887165069 0.9799999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6636583805084229
Batch 0.0
Batch 10.0
Epoch time 0.6640701293945312
get stats time 0.21812105178833008
Epoch 36 Train loss, accuracy 0.0042214189052581785 0.9989000000000001
get stats time 0.21804094314575195
EPoch 36 Validation loss, accuracy 0.05981023738384247 0.9821
Batch 0.0
Batch 10.0
Epoch time 0.6659541130065918
Batch 0.0
Batch 10.0
Epoch time 0.6648671627044678
get stats time 0.21489810943603516
Epoch 38 Train loss, accuracy 0.007949977922439575 0.9973000000000001
get stats time 0.21598196029663086
EPoch 38 Validation loss, accuracy 0.06837756078243254 0.9809000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6634368896484375
test accuracy 0.9809
```

```
In [34]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         model_name="model_deeper_2"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3


         ## collect error
         err_train = []
         err_val = []
```

```
tf.reset_default_graph()

x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
x_image = tf.reshape(x, [-1, dim, dim, nchannels])
# Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
# The number of incoming channels, for example, will be 3 if the image is color: RGB
# We will slide filter over this 2d picture with conv2d function.
y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
# Allows you to control the time step during the iterations
lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

with tf.Session() as sess:
        train,val,test=get_data(data_set=data_set)
        # Create the network architecture with the above placeholdes as the inputs.
        cross_entropy, accuracy, fc2 =my_create_network_deeper2([32,64,256,224,224,10

        # Define the miminization method
        if (minimizer=="Adam"):
            train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
        elif (minimizer=="SGD"):
            train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
        # Initialize variables
        sess.run(tf.global_variables_initializer())
        # Show trainable variables
        for v in tf.trainable_variables():
            print(v.name,v.get_shape().as_list(),np.std(v.eval()))
        ii=np.arange(0,num_train,1) #len(train_data),1)
        # Run epochs
        for i in range(num_epochs):  # number of epochs
            run_epoch(train,val,ii,batch_size,train_step)
            if (np.mod(i,2)==0):
                lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                err_train.append(1-ac)
                print('Epoch',i,'Train loss, accuracy',lo,ac)

                vlo,vac = get_stats(val[0],val[1])

                err_val.append(1-ac)
                print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                # Test set accuracy


        print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))
```

```
(70000, 784)
conv1/W:0 [5, 5, 1, 32] 0.048070334
```

```
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028969055
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 256] 0.02428934
fc1/b:0 [256] 0.0
fc2/W:0 [256, 224] 0.064581335
fc2/b:0 [224] 0.0
fc2_2/W:0 [224, 224] 0.06688118
fc2_2/b:0 [224] 0.0
fc3/W:0 [224, 10] 0.09073307
fc3/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.8937857151031494
get stats time 0.2483670711517334
Epoch 0 Train loss, accuracy 0.48833240683078766 0.8521000000000001
get stats time 0.2212073802947998
EPoch 0 Validation loss, accuracy 0.49485979948043834 0.8552
Batch 0.0
Batch 10.0
Epoch time 0.677483320236206
Batch 0.0
Batch 10.0
Epoch time 0.6702156066894531
get stats time 0.21885204315185547
Epoch 2 Train loss, accuracy 0.15362498872280123 0.9559
get stats time 0.2198801040649414
EPoch 2 Validation loss, accuracy 0.1611006445169449 0.9502
Batch 0.0
Batch 10.0
Epoch time 0.669180154800415
Batch 0.0
Batch 10.0
Epoch time 0.6667838096618652
get stats time 0.21953964233398438
Epoch 4 Train loss, accuracy 0.10033532700538636 0.9681
get stats time 0.21710944175720215
EPoch 4 Validation loss, accuracy 0.12123142404556275 0.9602999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6634998321533203
Batch 0.0
Batch 10.0
Epoch time 0.6639695167541504
get stats time 0.21882224082946777
Epoch 6 Train loss, accuracy 0.07217171021699906 0.9780000000000001
get stats time 0.21911096572875977
EPoch 6 Validation loss, accuracy 0.08781648094654083 0.9711000000000001
```

```
Batch 0.0
Batch 10.0
Epoch time 0.6685192584991455
Batch 0.0
Batch 10.0
Epoch time 0.6666231155395508
get stats time 0.21931910514831543
Epoch 8 Train loss, accuracy 0.04794078778028488 0.9849
get stats time 0.2169654369354248
EPoch 8 Validation loss, accuracy 0.07414989860057832 0.9769
Batch 0.0
Batch 10.0
Epoch time 0.6638393402099609
Batch 0.0
Batch 10.0
Epoch time 0.6631319522857666
get stats time 0.21983003616333008
Epoch 10 Train loss, accuracy 0.05632472541332244 0.9804999999999999
get stats time 0.21863532066345215
EPoch 10 Validation loss, accuracy 0.09058653893470764 0.9706999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6699581146240234
Batch 0.0
Batch 10.0
Epoch time 0.669074535369873
get stats time 0.2191786766052246
Epoch 12 Train loss, accuracy 0.03523924596309662 0.9888
get stats time 0.21667766571044922
EPoch 12 Validation loss, accuracy 0.07574239385128022 0.9760999999999997
Batch 0.0
Batch 10.0
Epoch time 0.6664600372314453
Batch 0.0
Batch 10.0
Epoch time 0.6675879955291748
get stats time 0.2188549041748047
Epoch 14 Train loss, accuracy 0.025650050425529473 0.9927999999999999
get stats time 0.2185966968536377
EPoch 14 Validation loss, accuracy 0.07048974418640137 0.9782
Batch 0.0
Batch 10.0
Epoch time 0.6649491786956787
Batch 0.0
Batch 10.0
Epoch time 0.6655523777008057
get stats time 0.21873116493225098
Epoch 16 Train loss, accuracy 0.020400077044963834 0.9926999999999999
```

```
get stats time 0.21859264373779297
EPoch 16 Validation loss, accuracy 0.06339472298622131 0.9810000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6690540313720703
Batch 0.0
Batch 10.0
Epoch time 0.6681697368621826
get stats time 0.2186751365661621
Epoch 18 Train loss, accuracy 0.02786921355724335 0.9905999999999999
get stats time 0.2183699607849121
EPoch 18 Validation loss, accuracy 0.07057451232671737 0.9773
Batch 0.0
Batch 10.0
Epoch time 0.6628048419952393
Batch 0.0
Batch 10.0
Epoch time 0.6656355857849121
get stats time 0.21829676628112793
Epoch 20 Train loss, accuracy 0.018967480659484864 0.9934999999999998
get stats time 0.2201695442199707
EPoch 20 Validation loss, accuracy 0.06726949417591095 0.9794
Batch 0.0
Batch 10.0
Epoch time 0.6702897548675537
Batch 0.0
Batch 10.0
Epoch time 0.6682770252227783
get stats time 0.21854853630065918
Epoch 22 Train loss, accuracy 0.014402190423011779 0.9955999999999999
get stats time 0.2190258502960205
EPoch 22 Validation loss, accuracy 0.062108219838142395 0.9804999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6663885116577148
Batch 0.0
Batch 10.0
Epoch time 0.6651113033294678
get stats time 0.2184445858001709
Epoch 24 Train loss, accuracy 0.012672050404548645 0.9963999999999998
get stats time 0.2226548194885254
EPoch 24 Validation loss, accuracy 0.06597253170013427 0.9805999999999997
Batch 0.0
Batch 10.0
Epoch time 0.6687486171722412
Batch 0.0
Batch 10.0
Epoch time 0.6706840991973877
```

```
get stats time 0.21858692169189453
Epoch 26 Train loss, accuracy 0.009489149737358094 0.9969000000000001
get stats time 0.2211322784423828
EPoch 26 Validation loss, accuracy 0.06578392772674561 0.9812999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6621334552764893
Batch 0.0
Batch 10.0
Epoch time 0.6675407886505127
get stats time 0.21930932998657227
Epoch 28 Train loss, accuracy 0.014443740439414978 0.9946000000000002
get stats time 0.22155404090881348
EPoch 28 Validation loss, accuracy 0.07328332490921019 0.9792
Batch 0.0
Batch 10.0
Epoch time 0.6697812080383301
Batch 0.0
Batch 10.0
Epoch time 0.6679646968841553
get stats time 0.22031688690185547
Epoch 30 Train loss, accuracy 0.010979573726654052 0.9966999999999999
get stats time 0.21994352340698242
EPoch 30 Validation loss, accuracy 0.06655089616775513 0.9815999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6636207103729248
Batch 0.0
Batch 10.0
Epoch time 0.6639397144317627
get stats time 0.21737122535705566
Epoch 32 Train loss, accuracy 0.015441707611083983 0.994
get stats time 0.21752429008483887
EPoch 32 Validation loss, accuracy 0.08168251819610596 0.9767999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6662070751190186
Batch 0.0
Batch 10.0
Epoch time 0.6711900234222412
get stats time 0.22161364555358887
Epoch 34 Train loss, accuracy 0.010430855131149292 0.9960999999999999
get stats time 0.21907353401184082
EPoch 34 Validation loss, accuracy 0.06391429097652436 0.9824999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6635990142822266
Batch 0.0
Batch 10.0
Epoch time 0.664984941482544
get stats time 0.22438335418701172
Epoch 36 Train loss, accuracy 0.017856870079040528 0.9934
get stats time 0.2185523509979248
EPoch 36 Validation loss, accuracy 0.0746577057003975 0.9773
Batch 0.0
Batch 10.0
Epoch time 0.6702709197998047
Batch 0.0
Batch 10.0
Epoch time 0.6702659130096436
get stats time 0.22120022773742676
Epoch 38 Train loss, accuracy 0.00545239806175232 0.9985999999999999
get stats time 0.21751713752746582
EPoch 38 Validation loss, accuracy 0.06151514867544174 0.9817
Batch 0.0
Batch 10.0
Epoch time 0.6637172698974609
test accuracy 0.981
```

10.3 Comment:

By simply adding one more layer with the same number of parameters, the accuracy increases from 0.98 to 0.9836. Though the increase is moderate, considering the fact that doubling the number of parameters get 0.9824 accuracy, we can see the depth of the network maybe matters more than just the number of parameters. However, as I use deeper (two more layers) network, the accuracy does not increase. I use [32,64,256,224,224,10] (increase parameters)to test if deeper network really can improves performance. The result is still almost the same. Thus, deeper network does not always improve performance.

10.4 (iii)

Use the model with one layer deeper than the original one

```
In [19]: train,val,test=get_data(data_set="mnist")
         train[0].shape
```

```
(70000, 784)
```

```
Out[19]: (50000, 784)
```

```
In [20]: ## run the model and plot error
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
```

```
num_train=50000
minimizer="Adam"
data_set="mnist"
model_name="model_deeper_0"
keep_prob=.5
dim=28
nchannels=1
if (data_set=="cifar"):
    dim=32
    nchannels=3


## collect error
err_train = []
err_val = []

tf.reset_default_graph()

x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
x_image = tf.reshape(x, [-1, dim, dim, nchannels])
# Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
# The number of incoming channels, for example, will be 3 if the image is color: RGB
# We will slide filter over this 2d picture with conv2d function.
y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
# Allows you to control the time step during the iterations
lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

with tf.Session() as sess:
        train,val,test=get_data(data_set=data_set)
        # Create the network architecture with the above placeholdes as the inputs.
        cross_entropy, accuracy, fc2 =my_create_network_deeper([32,64,240,210,10])

        # Define the miminization method
        if (minimizer=="Adam"):
            train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
        elif (minimizer=="SGD"):
            train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
        # Initialize variables
        sess.run(tf.global_variables_initializer())
        # Show trainable variables
        for v in tf.trainable_variables():
            print(v.name,v.get_shape().as_list(),np.std(v.eval()))
        ii=np.arange(0,num_train,1) #len(train_data),1)
        # Run epochs
        for i in range(num_epochs):  # number of epochs
            run_epoch(train,val,ii,batch_size,train_step)
            if (np.mod(i,2)==0):
                lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])

                err_train.append(1-ac)
                print('Epoch',i,'Train loss, accuracy',lo,ac)

                vlo,vac = get_stats(val[0],val[1])

                err_val.append(1-ac)
                print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                # Test set accuracy

        ## my show images
        image = train[0][0]
        w_out = tf.trainable_variables()[0]
        b_out = tf.trainable_variables()[1]
        conv = tf.nn.conv2d(x_image, w_out, strides=[1, 1, 1, 1], padding='SAME')
        ## conv2d : arbitrary filters that can mix channels together
        relu = tf.nn.relu(conv + b_out)
        plot_conv_layer(relu, image)


        print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

        # Save model
        tf.add_to_collection("optimizer", train_step)
        saver = tf.train.Saver()
        save_path = saver.save(sess, "tmp/"+model_name)
        print("Model saved in path: %s" % save_path)
```

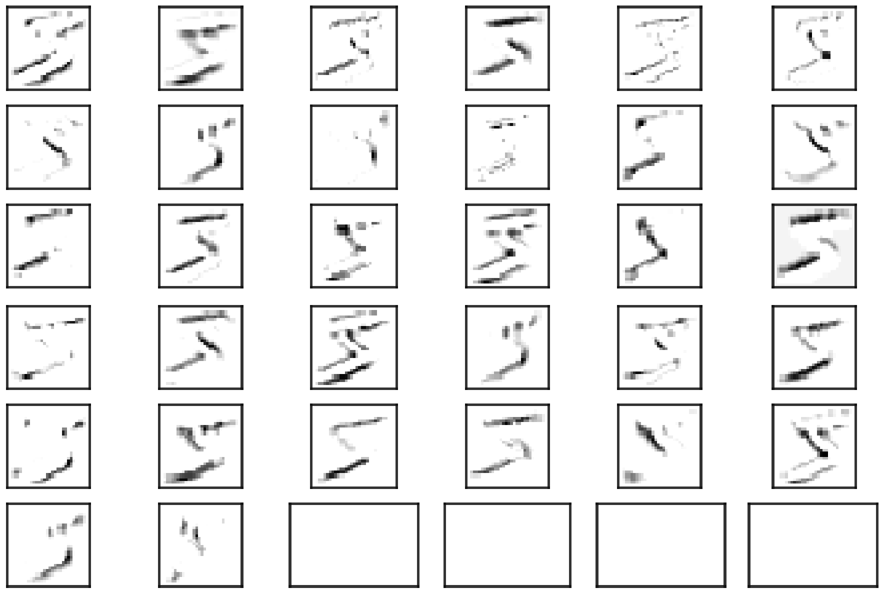
```
(70000, 784)
WARNING:tensorflow:From <ipython-input-10-32c56643641f>:32: softmax_cross_entropy_with_logits
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See @{tf.nn.softmax_cross_entropy_with_logits_v2}.

conv1/W:0 [5, 5, 1, 32] 0.04914404
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028796514
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 240] 0.024364792
fc1/b:0 [240] 0.0
fc2/W:0 [240, 210] 0.06655863
fc2/b:0 [210] 0.0
fc3/W:0 [210, 10] 0.09445546
fc3/b:0 [10] 0.0
```

```
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 11.115446090698242
get stats time 1.2272121906280518
Epoch 0 Train loss, accuracy 0.1092635355734825 0.9660400000000001
get stats time 0.21979737281799316
EPoch 0 Validation loss, accuracy 0.09752671483755113 0.9682000000000001
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.339102029800415
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3697662353515625
get stats time 1.0665414333343506
Epoch 2 Train loss, accuracy 0.04962280452489853 0.9839599999999997
get stats time 0.22019505500793457
EPoch 2 Validation loss, accuracy 0.04649085544347763 0.9853
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.330775022506714
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3754050731658936
get stats time 1.0665955543518066
Epoch 4 Train loss, accuracy 0.03572170806884766 0.9886599999999999
get stats time 0.2203972339630127
EPoch 4 Validation loss, accuracy 0.04219804396629334 0.9862
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.347752094268799
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.375649929046631
get stats time 1.0757036209106445
Epoch 6 Train loss, accuracy 0.023272879850864413 0.9926199999999997
get stats time 0.21580791473388672
EPoch 6 Validation loss, accuracy 0.032121779108047484 0.9894000000000001
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.369868516921997
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3433749675750732
get stats time 1.0671348571777344
Epoch 8 Train loss, accuracy 0.019514729087352748 0.9937399999999996
get stats time 0.21831464767456055
EPoch 8 Validation loss, accuracy 0.030057189702987673 0.9893999999999998
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3452229499816895
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.346950054168701
get stats time 1.0698890686035156
Epoch 10 Train loss, accuracy 0.016694962759017946 0.9944799999999998
get stats time 0.21500086784362793
```

```
EPoch 10 Validation loss, accuracy 0.030455158972740176 0.9903000000000001
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3499672412872314
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.353878974914551
get stats time 1.1025505065917969
Epoch 12 Train loss, accuracy 0.013907220687866212 0.9956799999999997
get stats time 0.22005534172058105
EPoch 12 Validation loss, accuracy 0.030257659506797784 0.9906
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3540587425231934
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
```

```
Epoch time 3.3691301345825195
get stats time 1.071664810180664
Epoch 14 Train loss, accuracy 0.014905199108123776 0.9952200000000001
get stats time 0.2149524688720703
EPoch 14 Validation loss, accuracy 0.03181792888641357 0.9904999999999999
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.362975835800171
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.397479295730591
get stats time 1.078669548034668
Epoch 16 Train loss, accuracy 0.00972677267551422 0.9968999999999997
get stats time 0.2143096923828125
EPoch 16 Validation loss, accuracy 0.02950509052276611 0.9907
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.360036849975586
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3749496936798096
get stats time 1.0799896717071533
Epoch 18 Train loss, accuracy 0.009315414786338808 0.9967599999999999
get stats time 0.21446847915649414
EPoch 18 Validation loss, accuracy 0.03384416577816009 0.9902000000000001
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3609132766723633
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.363194227218628
get stats time 1.0707380771636963
Epoch 20 Train loss, accuracy 0.007641263999389655 0.9973599999999998
get stats time 0.21450543403625488
EPoch 20 Validation loss, accuracy 0.029021514105796814 0.9912000000000001
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.362992525100708
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3643078804016113
get stats time 1.1032752990722656
Epoch 22 Train loss, accuracy 0.012139579744338986 0.9958999999999998
get stats time 0.22050094604492188
EPoch 22 Validation loss, accuracy 0.03719831848144531 0.9902
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3676302433013916
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.372857093811035
get stats time 1.0742201805114746
Epoch 24 Train loss, accuracy 0.006083870401382444 0.9976800000000001
get stats time 0.21413397789001465
EPoch 24 Validation loss, accuracy 0.03199668817520142 0.9907
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.364772319793701
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3682923316955566
get stats time 1.0700633525848389
Epoch 26 Train loss, accuracy 0.007175378127098084 0.9973199999999998
get stats time 0.21406292915344238
EPoch 26 Validation loss, accuracy 0.03681624135971069 0.9906
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.365229606628418
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3684465885162354
get stats time 1.0717904567718506
Epoch 28 Train loss, accuracy 0.005933764405250547 0.99796
get stats time 0.2143399715423584
EPoch 28 Validation loss, accuracy 0.03205601427555084 0.9917
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3671274185180664
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3851330280303955
get stats time 1.0784428119659424
Epoch 30 Train loss, accuracy 0.005452357075214385 0.9981800000000001
get stats time 0.21461272239685059
EPoch 30 Validation loss, accuracy 0.03212955703735352 0.9912999999999998
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.370328903198242
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3727025985717773
get stats time 1.1163678169250488
Epoch 32 Train loss, accuracy 0.007216043486595155 0.99756
get stats time 0.2142195701599121
EPoch 32 Validation loss, accuracy 0.036272793078422544 0.9905999999999999
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3670434951782227
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3842599391937256
get stats time 1.0789613723754883
Epoch 34 Train loss, accuracy 0.004096264123916626 0.9985800000000001
get stats time 0.2151317596435547
EPoch 34 Validation loss, accuracy 0.03246441831588745 0.9914999999999999
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.385563611984253
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.386343240737915
get stats time 1.0718865394592285
```

```
Epoch 36 Train loss, accuracy 0.007113953599929811 0.9975200000000004
get stats time 0.21444416046142578
EPoch 36 Validation loss, accuracy 0.03980694451332093 0.9896999999999998
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3698368072509766
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3719663619995117
get stats time 1.1027166843414307
Epoch 38 Train loss, accuracy 0.004311813049316407 0.9985399999999999
get stats time 0.22023701667785645
EPoch 38 Validation loss, accuracy 0.03158524935245514 0.9928999999999999
Batch 0.0
Batch 10.0
Batch 20.0
Batch 30.0
Batch 40.0
Batch 50.0
Batch 60.0
Batch 70.0
Batch 80.0
Batch 90.0
Epoch time 3.3697285652160645
```



```
test accuracy 0.9899
Model saved in path: tmp/model_deeper_0
```

10.5 Comment

With all the data running in my best model, the accuracy is almost 0.99

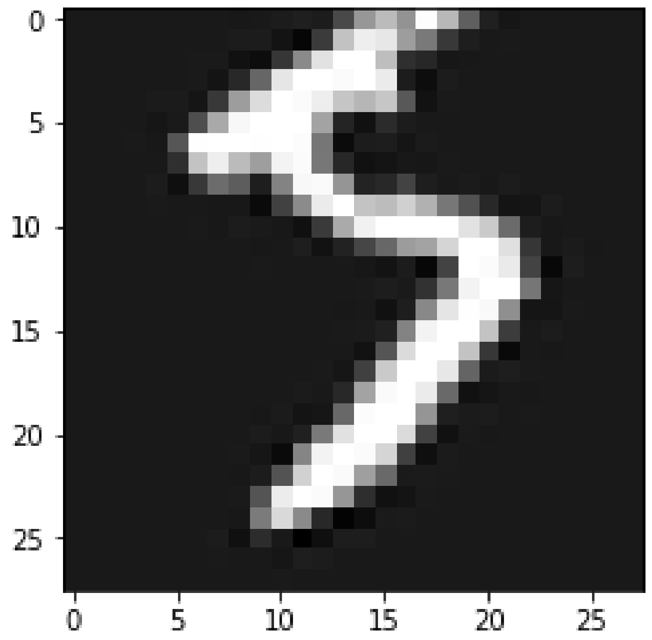
11 (C)

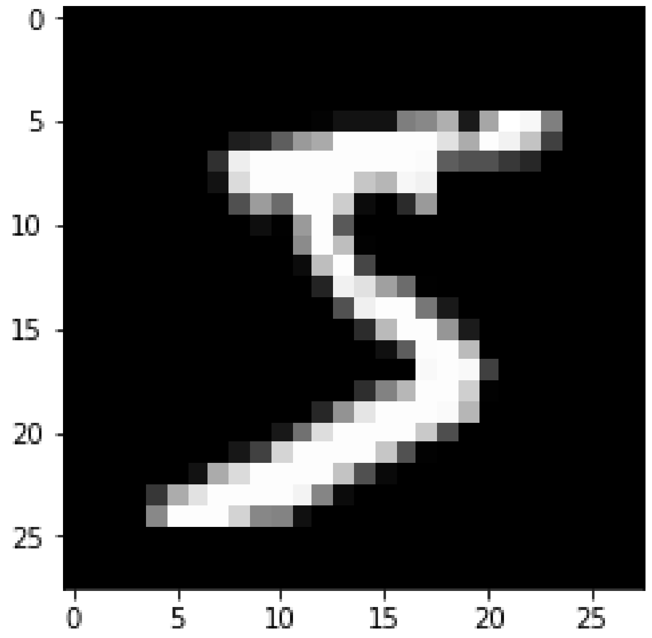
11.1 (i) Display

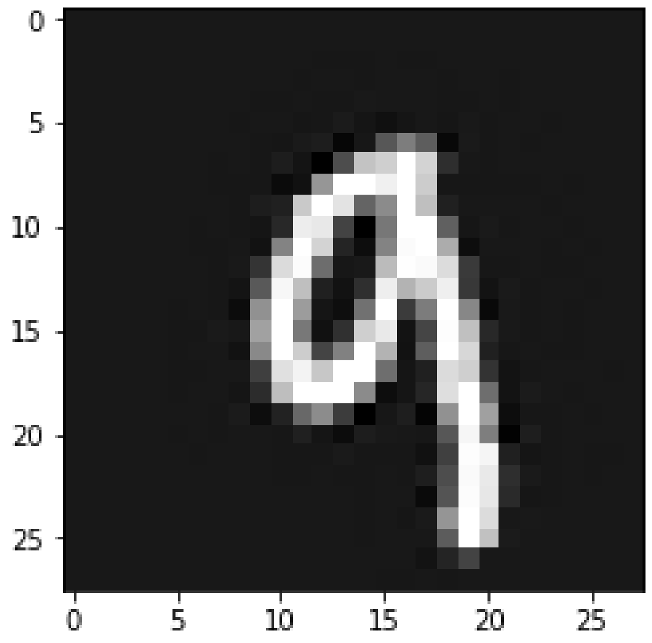
```
In [22]: test_t,label_t = get_data(data_set = "mnist_transform")
         train,val,test = get_data(data_set = "mnist")
```

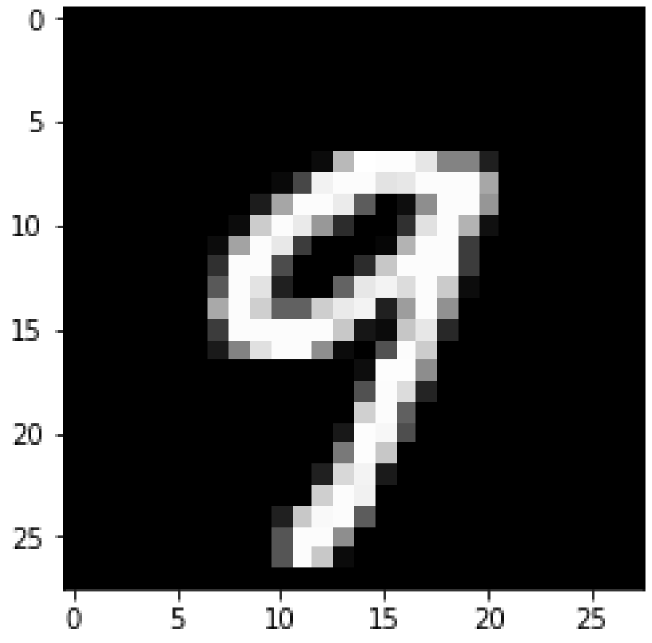
```
(70000, 784)
```

```
In [24]: plt.imshow(test_t[0,:].reshape(28,28), cmap = 'gray')
         plt.show()
         plt.imshow(train[0][0,:].reshape(28,28), cmap = 'gray')
         plt.show()
         plt.imshow(test_t[45,:].reshape(28,28), cmap = 'gray')
         plt.show()
         plt.imshow(train[0][45,:].reshape(28,28), cmap = 'gray')
         plt.show()
```


0
5
10
15
20
25
0
5
10
15
20
25


0
5
10
15
20
25
0
5
10
15
20
25





11.2 (ii) try original data

```
In [25]: # Run the training
```

```
import time
batch_size=500
step_size=.001
num_epochs=40
num_train=10000
minimizer="Adam"
data_set="mnist"
data_set2="mnist_transform"
model_name="model_org_t"
keep_prob=.5
dim=28
nchannels=1
if (data_set=="cifar"):
    dim=32
    nchannels=3


tf.reset_default_graph()

x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
x_image = tf.reshape(x, [-1, dim, dim, nchannels])
# Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
# The number of incoming channels, for example, will be 3 if the image is color: RGB
# We will slide filter over this 2d picture with conv2d function.
y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
# Allows you to control the time step during the iterations
lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

with tf.Session() as sess:
        train=get_data(data_set=data_set)[0]

        data_t,label_t=get_data(data_set=data_set2)
        #train = (data_t[:50000,:],label_t[:50000,:])
        val = (data_t[50000:60000,:],label_t[50000:60000,:])
        test = (data_t[60000:,:],label_t[60000:,:])

        # Create the network architecture with the above placeholdes as the inputs.
        cross_entropy, accuracy, fc2 =create_network()

        # Define the miminization method
        if (minimizer=="Adam"):
            train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entrop
        elif (minimizer=="SGD"):
            train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
        # Initialize variables
        sess.run(tf.global_variables_initializer())
        # Show trainable variables
```

```
                for v in tf.trainable_variables():
                    print(v.name,v.get_shape().as_list(),np.std(v.eval()))
                ii=np.arange(0,num_train,1) #len(train_data),1)
                # Run epochs
                for i in range(num_epochs):  # number of epochs
                    run_epoch(train,val,ii,batch_size,train_step)
                    if (np.mod(i,2)==0):
                        lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])
                        print('Epoch',i,'Train loss, accuracy',lo,ac)
                        vlo,vac = get_stats(val[0],val[1])
                        print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                        # Test set accuracy

                print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

                # Save model
                tf.add_to_collection("optimizer", train_step)
                saver = tf.train.Saver()
                save_path = saver.save(sess, "tmp/"+model_name)
                print("Model saved in path: %s" % save_path)
```

```
(70000, 784)
conv1/W:0 [5, 5, 1, 32] 0.049018368
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.02884209
conv2/b:0 [64] 0.0
fc1/W:0 [3136, 256] 0.02427195
fc1/b:0 [256] 0.0
fc2/W:0 [256, 10] 0.08708927
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.8851244449615479
get stats time 0.25666165351867676
Epoch 0 Train loss, accuracy 0.4225864974737168 0.8718
get stats time 0.22640061378479004
EPoch 0 Validation loss, accuracy 3.132067643332481 0.3394
Batch 0.0
Batch 10.0
Epoch time 0.6786906719207764
Batch 0.0
Batch 10.0
Epoch time 0.6763641834259033
get stats time 0.22584247589111328
Epoch 2 Train loss, accuracy 0.14590953574180604 0.9578999999999999
get stats time 0.21957921981811523
EPoch 2 Validation loss, accuracy 2.375671470534802 0.4378
Batch 0.0
Batch 10.0
Epoch time 0.6608531475067139
Batch 0.0
Batch 10.0
Epoch time 0.6600513458251953
get stats time 0.22120046615600586
Epoch 4 Train loss, accuracy 0.08874740862846375 0.9743999999999999
get stats time 0.2188727855682373
EPoch 4 Validation loss, accuracy 2.1604574501514433 0.4828
Batch 0.0
Batch 10.0
Epoch time 0.6636598110198975
Batch 0.0
Batch 10.0
Epoch time 0.66349196434021
get stats time 0.2208852767944336
Epoch 6 Train loss, accuracy 0.06830487713813782 0.9790000000000001
get stats time 0.22046709060668945
EPoch 6 Validation loss, accuracy 2.2312459975838657 0.49939999999999996
Batch 0.0
Batch 10.0
Epoch time 0.6593127250671387
Batch 0.0
Batch 10.0
Epoch time 0.6618821620941162
get stats time 0.22530603408813477
Epoch 8 Train loss, accuracy 0.054177508187294 0.9832999999999998
get stats time 0.2193613052368164
EPoch 8 Validation loss, accuracy 2.1632734020709994 0.5237
Batch 0.0
Batch 10.0
Epoch time 0.6594383716583252
Batch 0.0
Batch 10.0
Epoch time 0.6668307781219482
get stats time 0.22587275505065918
Epoch 10 Train loss, accuracy 0.042769554805755616 0.9859
get stats time 0.22075343132019043
EPoch 10 Validation loss, accuracy 2.3012826985955237 0.5339
Batch 0.0
Batch 10.0
Epoch time 0.6578500270843506
Batch 0.0
Batch 10.0
Epoch time 0.6660280227661133
get stats time 0.22347235679626465
Epoch 12 Train loss, accuracy 0.033197513079643255 0.9892
get stats time 0.21309280395507812
```

```
EPoch 12 Validation loss, accuracy 2.338957607483864 0.5353999999999999
Batch 0.0
Batch 10.0
Epoch time 0.6677038669586182
Batch 0.0
Batch 10.0
Epoch time 0.6631083488464355
get stats time 0.22409605979919434
Epoch 14 Train loss, accuracy 0.024358587145805362 0.9917000000000001
get stats time 0.2183842658996582
EPoch 14 Validation loss, accuracy 2.38737086135149 0.5519
Batch 0.0
Batch 10.0
Epoch time 0.6620562076568604
Batch 0.0
Batch 10.0
Epoch time 0.6621456146240234
get stats time 0.22176146507263184
Epoch 16 Train loss, accuracy 0.025662967824935913 0.992
get stats time 0.2198784351348877
EPoch 16 Validation loss, accuracy 2.47525295535326 0.5539
Batch 0.0
Batch 10.0
Epoch time 0.6588327884674072
Batch 0.0
Batch 10.0
Epoch time 0.6624476909637451
get stats time 0.21799588203430176
Epoch 18 Train loss, accuracy 0.018596133828163143 0.9944
get stats time 0.21789002418518066
EPoch 18 Validation loss, accuracy 2.498043253064156 0.5577
Batch 0.0
Batch 10.0
Epoch time 0.6613543033599854
Batch 0.0
Batch 10.0
Epoch time 0.6607940196990967
get stats time 0.22560453414916992
Epoch 20 Train loss, accuracy 0.014624339199066161 0.9952
get stats time 0.21902036666870117
EPoch 20 Validation loss, accuracy 2.5612464792013165 0.5698000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6649699211120605
Batch 0.0
Batch 10.0
Epoch time 0.6617405414581299
get stats time 0.22650861740112305
```

```
Epoch 22 Train loss, accuracy 0.013904175949096681 0.9953999999999998
get stats time 0.21996092796325684
EPoch 22 Validation loss, accuracy 2.5622631301641468 0.5723
Batch 0.0
Batch 10.0
Epoch time 0.6604123115539551
Batch 0.0
Batch 10.0
Epoch time 0.6642117500305176
get stats time 0.22136592864990234
Epoch 24 Train loss, accuracy 0.015300699567794798 0.9958
get stats time 0.2196354866027832
EPoch 24 Validation loss, accuracy 2.582729642283916 0.5707
Batch 0.0
Batch 10.0
Epoch time 0.6638908386230469
Batch 0.0
Batch 10.0
Epoch time 0.6681084632873535
get stats time 0.2215254306793213
Epoch 26 Train loss, accuracy 0.012199741411209105 0.9955999999999999
get stats time 0.21611356735229492
EPoch 26 Validation loss, accuracy 2.7600014631032947 0.5622
Batch 0.0
Batch 10.0
Epoch time 0.6628315448760986
Batch 0.0
Batch 10.0
Epoch time 0.6629209518432617
get stats time 0.2256619930267334
Epoch 28 Train loss, accuracy 0.009836513042449951 0.9966999999999999
get stats time 0.22047734260559082
EPoch 28 Validation loss, accuracy 2.734001048994064 0.5820000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6662454605102539
Batch 0.0
Batch 10.0
Epoch time 0.661733865737915
get stats time 0.22104477882385254
Epoch 30 Train loss, accuracy 0.010314118099212646 0.9969000000000001
get stats time 0.21979975700378418
EPoch 30 Validation loss, accuracy 2.949692815470696 0.5645
Batch 0.0
Batch 10.0
Epoch time 0.6607232093811035
Batch 0.0
Batch 10.0
```

```
Epoch time 0.666295051574707
get stats time 0.22468209266662598
Epoch 32 Train loss, accuracy 0.009451341962814332 0.9973000000000001
get stats time 0.2170572280883789
EPoch 32 Validation loss, accuracy 2.9850494269132617 0.5705
Batch 0.0
Batch 10.0
Epoch time 0.6618576049804688
Batch 0.0
Batch 10.0
Epoch time 0.6615703105926514
get stats time 0.22600030899047852
Epoch 34 Train loss, accuracy 0.00781462173461914 0.9977
get stats time 0.2181248664855957
EPoch 34 Validation loss, accuracy 2.7261554158926007 0.585
Batch 0.0
Batch 10.0
Epoch time 0.6601345539093018
Batch 0.0
Batch 10.0
Epoch time 0.6603693962097168
get stats time 0.2246532440185547
Epoch 36 Train loss, accuracy 0.013561532831192016 0.9959999999999999
get stats time 0.22052621841430664
EPoch 36 Validation loss, accuracy 2.878900481557846 0.5801000000000001
Batch 0.0
Batch 10.0
Epoch time 0.6620516777038574
Batch 0.0
Batch 10.0
Epoch time 0.6670749187469482
get stats time 0.2252974510192871
Epoch 38 Train loss, accuracy 0.006058556890487671 0.9983000000000001
get stats time 0.21940326690673828
EPoch 38 Validation loss, accuracy 3.1043067299365994 0.5769999999999998
Batch 0.0
Batch 10.0
Epoch time 0.6599924564361572
test accuracy 0.5802
Model saved in path: tmp/model_org_t
```

11.2.1 analysis of experiment of using orginal model and testing on transformed data

From the experiment we can see that the accuracy drops drammatically, to around 58%.

11.2.2 (iii) propose changes to the original model

I change the pooling ksize and stride. I performed many experiments and list the results at the end. I only show the computation for the best model below (with ksize [1,9,9,1] and stride [1,3,3,1])

```
In [26]: # Run the training

         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="mnist"
         data_set2="mnist_transform"
         model_name="model_new_t_size9_stride3"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3


         tf.reset_default_graph()

         x = tf.placeholder(tf.float32, shape=[None, dim*dim*nchannels],name="x")
         x_image = tf.reshape(x, [-1, dim, dim, nchannels])
         # Dimensions of x_image: [Batch size, Column size, Row size, Number of incoming chann
         # The number of incoming channels, for example, will be 3 if the image is color: RGB
         # We will slide filter over this 2d picture with conv2d function.
         y_ = tf.placeholder(tf.float32, shape=[None,10],name="y")
         # Allows you to control the time step during the iterations
         lr_ = tf.placeholder(tf.float32, shape=[],name="learning_rate")
         keep_prob_=tf.placeholder(tf.float32, shape=[],name="keep_prob")

         with tf.Session() as sess:
                 train=get_data(data_set=data_set)[0]

                 data_t,label_t=get_data(data_set=data_set2)
                 #train = (data_t[:50000,:],label_t[:50000,:])
                 val = (data_t[50000:60000,:],label_t[50000:60000,:])
                 test = (data_t[60000:,:],label_t[60000:,:])

                 # Create the network architecture with the above placeholdes as the inputs.
                 cross_entropy, accuracy, fc2 =my_create_network_t0(pool_ksize=[1,9,9,1],\
                                                                    pool_strides=[1,3,3,1])


                 # Define the miminization method
```

```
            if (minimizer=="Adam"):
                train_step=tf.train.AdamOptimizer(learning_rate=lr_).minimize(cross_entro
            elif (minimizer=="SGD"):
                train_step = tf.train.GradientDescentOptimizer(learning_rate=lr_).minimiz
            # Initialize variables
            sess.run(tf.global_variables_initializer())
            # Show trainable variables
            for v in tf.trainable_variables():
                print(v.name,v.get_shape().as_list(),np.std(v.eval()))
            ii=np.arange(0,num_train,1) #len(train_data),1)
            # Run epochs
            for i in range(num_epochs):  # number of epochs
                run_epoch(train,val,ii,batch_size,train_step)
                if (np.mod(i,2)==0):
                    lo,ac = get_stats(train[0][0:num_train],train[1][0:num_train])
                    print('Epoch',i,'Train loss, accuracy',lo,ac)
                    vlo,vac = get_stats(val[0],val[1])
                    print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                    # Test set accuracy

            print('test accuracy %g' % accuracy.eval(feed_dict={x: test[0], y_:test[1]}))

            # Save model
            tf.add_to_collection("optimizer", train_step)
            saver = tf.train.Saver()
            save_path = saver.save(sess, "tmp/"+model_name)
            print("Model saved in path: %s" % save_path)
```

```
(70000, 784)
conv1/W:0 [5, 5, 1, 32] 0.0485146
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028784038
conv2/b:0 [64] 0.0
fc1/W:0 [1024, 256] 0.039508898
fc1/b:0 [256] 0.0
fc2/W:0 [256, 10] 0.08771751
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 0.9228222370147705
get stats time 0.23228049278259277
Epoch 0 Train loss, accuracy 1.1338774825811386 0.655
get stats time 0.1754913330078125
EPoch 0 Validation loss, accuracy 1.9231937439203264 0.3456
Batch 0.0
Batch 10.0
Epoch time 0.5957107543945312
Batch 0.0
Batch 10.0
Epoch time 0.5920295715332031
get stats time 0.17769742012023926
Epoch 2 Train loss, accuracy 0.237403590285778 0.9327000000000002
get stats time 0.17354130744934082
EPoch 2 Validation loss, accuracy 1.8248793835282324 0.5164
Batch 0.0
Batch 10.0
Epoch time 0.5921800136566162
Batch 0.0
Batch 10.0
Epoch time 0.5894832611083984
get stats time 0.1723771095275879
Epoch 4 Train loss, accuracy 0.14670280714035033 0.9571
get stats time 0.17130208015441895
EPoch 4 Validation loss, accuracy 1.4878640374660492 0.5974000000000002
Batch 0.0
Batch 10.0
Epoch time 0.5876123905181885
Batch 0.0
Batch 10.0
Epoch time 0.5892355442047119
get stats time 0.1693415641784668
Epoch 6 Train loss, accuracy 0.12559971467256545 0.9647999999999998
get stats time 0.17029690742492676
EPoch 6 Validation loss, accuracy 1.397703254544735 0.6268999999999999
Batch 0.0
Batch 10.0
Epoch time 0.5903818607330322
Batch 0.0
Batch 10.0
Epoch time 0.5888247489929199
get stats time 0.1730809211730957
Epoch 8 Train loss, accuracy 0.08558536230325699 0.9742999999999998
get stats time 0.17149877548217773
EPoch 8 Validation loss, accuracy 1.2256485028266906 0.6456000000000001
Batch 0.0
Batch 10.0
Epoch time 0.5876176357269287
Batch 0.0
Batch 10.0
Epoch time 0.5895817279815674
get stats time 0.16869163513183594
Epoch 10 Train loss, accuracy 0.09176163569688796 0.9722999999999999
get stats time 0.17047357559204102
EPoch 10 Validation loss, accuracy 1.234152205860615 0.6522
Batch 0.0
Batch 10.0
```

```
Epoch time 0.590888500213623
Batch 0.0
Batch 10.0
Epoch time 0.5876631736755371
get stats time 0.17299604415893555
Epoch 12 Train loss, accuracy 0.060946556377410886 0.9803999999999998
get stats time 0.17263340950012207
EPoch 12 Validation loss, accuracy 1.1278719218373296 0.6877000000000001
Batch 0.0
Batch 10.0
Epoch time 0.5876765251159668
Batch 0.0
Batch 10.0
Epoch time 0.5904643535614014
get stats time 0.17032670974731445
Epoch 14 Train loss, accuracy 0.05294985411167145 0.9841
get stats time 0.17041778564453125
EPoch 14 Validation loss, accuracy 1.1129031538128853 0.6935
Batch 0.0
Batch 10.0
Epoch time 0.5894489288330078
Batch 0.0
Batch 10.0
Epoch time 0.5878448486328125
get stats time 0.17403960227966309
Epoch 16 Train loss, accuracy 0.046445943510532384 0.9865
get stats time 0.17319345474243164
EPoch 16 Validation loss, accuracy 1.0986749396324158 0.6909
Batch 0.0
Batch 10.0
Epoch time 0.5885725021362305
Batch 0.0
Batch 10.0
Epoch time 0.589440107345581
get stats time 0.1709294319152832
Epoch 18 Train loss, accuracy 0.04917809804677964 0.9857000000000001
get stats time 0.16936278343200684
EPoch 18 Validation loss, accuracy 1.223768912768364 0.6826
Batch 0.0
Batch 10.0
Epoch time 0.594062328338623
Batch 0.0
Batch 10.0
Epoch time 0.5884711742401123
get stats time 0.17186808586120605
Epoch 20 Train loss, accuracy 0.03733087284564972 0.9888999999999999
get stats time 0.17327189445495605
EPoch 20 Validation loss, accuracy 1.0997463568568229 0.7059999999999998
```

```
Batch 0.0
Batch 10.0
Epoch time 0.5924890041351318
Batch 0.0
Batch 10.0
Epoch time 0.593256950378418
get stats time 0.17346572875976562
Epoch 22 Train loss, accuracy 0.02654422838687897 0.9922000000000001
get stats time 0.17175769805908203
EPoch 22 Validation loss, accuracy 1.0369240003943445 0.7240999999999999
Batch 0.0
Batch 10.0
Epoch time 0.5932374000549316
Batch 0.0
Batch 10.0
Epoch time 0.5971131324768066
get stats time 0.17329740524291992
Epoch 24 Train loss, accuracy 0.02440968556404114 0.9927999999999999
get stats time 0.17355918884277344
EPoch 24 Validation loss, accuracy 1.1040704236149788 0.7188999999999999
Batch 0.0
Batch 10.0
Epoch time 0.5927515029907227
Batch 0.0
Batch 10.0
Epoch time 0.5934007167816162
get stats time 0.17282533645629883
Epoch 26 Train loss, accuracy 0.024350544953346254 0.9927000000000001
get stats time 0.18067169189453125
EPoch 26 Validation loss, accuracy 1.0505886253237724 0.7285999999999999
Batch 0.0
Batch 10.0
Epoch time 0.5931460857391357
Batch 0.0
Batch 10.0
Epoch time 0.5918850898742676
get stats time 0.1719377040863037
Epoch 28 Train loss, accuracy 0.017860165762901302 0.9951000000000001
get stats time 0.17331552505493164
EPoch 28 Validation loss, accuracy 1.0749715718865396 0.7257999999999999
Batch 0.0
Batch 10.0
Epoch time 0.5913476943969727
Batch 0.0
Batch 10.0
Epoch time 0.5915586948394775
get stats time 0.17345643043518066
Epoch 30 Train loss, accuracy 0.017277949357032776 0.9952
```

```
get stats time 0.17412137985229492
EPoch 30 Validation loss, accuracy 1.087869784975052 0.7331000000000001
Batch 0.0
Batch 10.0
Epoch time 0.5970580577850342
Batch 0.0
Batch 10.0
Epoch time 0.5957825183868408
get stats time 0.17339038848876953
Epoch 32 Train loss, accuracy 0.020378100252151487 0.993
get stats time 0.17351722717285156
EPoch 32 Validation loss, accuracy 1.1989693815112112 0.7162
Batch 0.0
Batch 10.0
Epoch time 0.5929660797119141
Batch 0.0
Batch 10.0
Epoch time 0.592010498046875
get stats time 0.1727437973022461
Epoch 34 Train loss, accuracy 0.018210551691055298 0.9945999999999999
get stats time 0.17383646965026855
EPoch 34 Validation loss, accuracy 1.115160995543003 0.7283
Batch 0.0
Batch 10.0
Epoch time 0.592409610748291
Batch 0.0
Batch 10.0
Epoch time 0.590416431427002
get stats time 0.2204294204711914
Epoch 36 Train loss, accuracy 0.013164818477630616 0.9962
get stats time 0.17372989654541016
EPoch 36 Validation loss, accuracy 1.1196402932882308 0.7372000000000001
Batch 0.0
Batch 10.0
Epoch time 0.5885124206542969
Batch 0.0
Batch 10.0
Epoch time 0.5904181003570557
get stats time 0.16952967643737793
Epoch 38 Train loss, accuracy 0.02994526126384735 0.9891
get stats time 0.17127656936645508
EPoch 38 Validation loss, accuracy 1.2244303511738779 0.7218000000000001
Batch 0.0
Batch 10.0
Epoch time 0.5910334587097168
test accuracy 0.728
Model saved in path: tmp/model_new_t_size9_stride3
```

11.2.3 Analysis on different experiments with different pool_ksize and pool_stride

```
In [ ]: pool_ksize     pool_stride     acc <br>
        2 2 0.58 <br>
        4 2 0.683 <br>
        6 2 0.69 <br>
        ... <br>
        9 2 0.7149 <br>
        10 2 0.7204 <br>
        11 2 0.7183 <br>
        ... <br>
        14 2 0.7102 <br>
        From above we can see the best results for fixed stride of 2 is 0.7204, achieved with k

        6 3 0.7115 <br>
        7 3 0.7137 <br>
        8 3 0.715 <br>
        9 3 0.728 <br>
        10 3 0.7197 <br>
        From above we can see the best results for fixed stride of 3 is 0.7274, achieved with k


        6 4 0.6901 <br>
        8 4 0.7035 <br>
        9 4 0.7183 <br>
        10 4 0.6924 <br>
        ... <br>
        12 4 0.6942 <br>
```

From above we can see the best results for fixed stride of 7 is 0.7183, achieved with ksize of 9.

To Conclude, to achieve better accuracy on transformed data sets, it is better to choose a larger pool window (9 or 10, bigger than that might loose too much nontrivial information). And from experiments the best way is to set ksize as 9 and stride as 3. The best accuracy is 0.728