conv_net_gpu_p2_ZihaoWang

May 10, 2018

```
In [1]: import tensorflow as tf
        import matplotlib.pyplot as plt
        import numpy as np
        %matplotlib inline
        import skimage
        from skimage import color
        from skimage.color import rgb2hsv
        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
```

```
/software/Anaconda3-5.0.1-el7-x86_64/envs/DL_GPU_cuda_9.0/lib/python3.6/site-packages/h5py/__i
  from ._conv import register_converters as _register_converters
```

1 Get Mnist data and split into train validation and test

```
In [2]: 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 [3]: 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.np
            return (train_data, train_labels), (val_data, val_labels), (test_data, test_labels
```

3 Get transformed Mnist data

```
In [4]: 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_lab
            return (test_trans_dat, test_labels)
```

4 Convolution layer with relu

```
In [5]: 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 [6]: 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 [7]: #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 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 
            return cross_entropy, accuracy, fc2

In [8]: #tf.reset_default_graph()

        def my_create_network_deeper():
            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=250)
                    fc1r=tf.nn.relu(fc1)

            with tf.variable_scope("fc1_2"):
                    fc1_2 = fully_connected_layer(fc1r, num_features=128)
                    fc1r_2=tf.nn.relu(fc1_2)

            with tf.variable_scope("fc2"):
                    fc2 = fully_connected_layer(fc1r_2, num_features=10)

            # 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
```

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

```
In [9]: 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 [10]: # 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 [11]: 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 Plot images

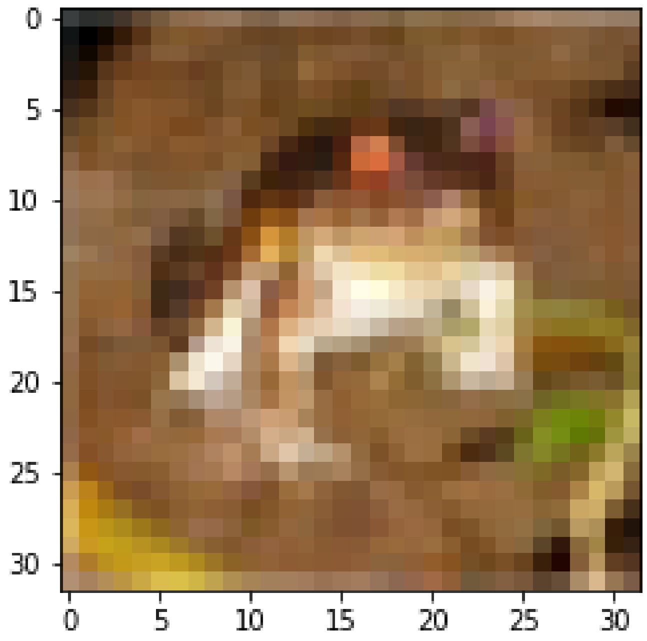
```
In [12]: ## reference https://github.com/Hvass-Labs/TensorFlow-Tutorials

         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.
```

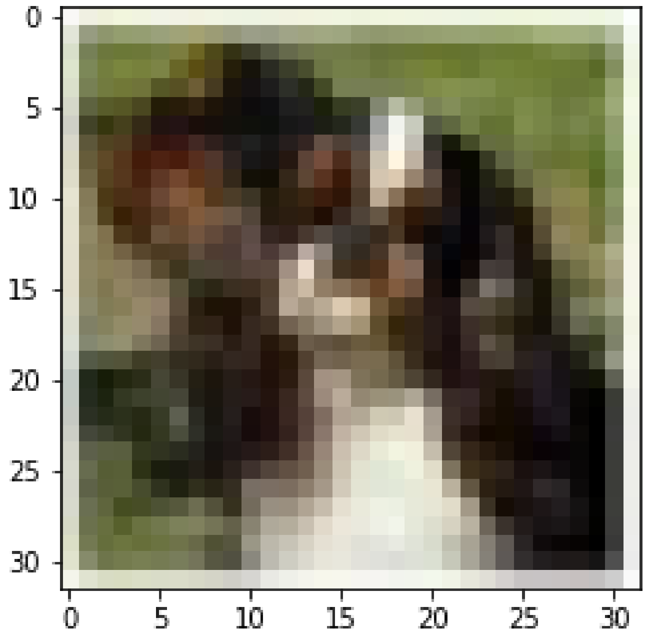
```
        values = sess.run(layer, feed_dict=feed_dict)

        # 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()
```

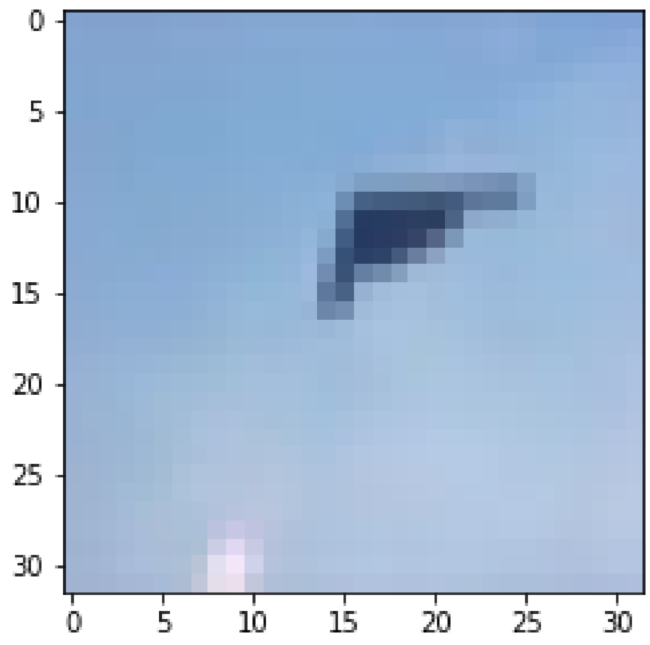
10 (a) Read in data and display

```
In [13]: train,val,test = get_cifar()
```

```
In [14]: plt.imshow(train[0][0,:].reshape(32,32,3))
         plt.show()
         plt.imshow(train[0][500,:].reshape(32,32,3))
         plt.show()
         plt.imshow(train[0][12121,:].reshape(32,32,3))
         plt.show()
```


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30



11 (b) run with original model

```
In [14]: # Run the training

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

         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_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-vac)
                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]}))

        ## 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)
```

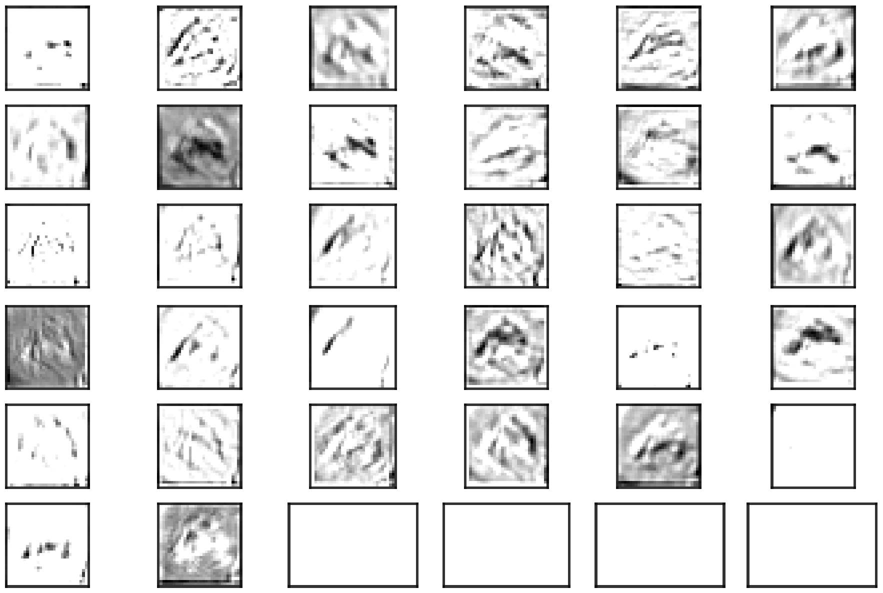
```
                plot_conv_layer(relu, image)

                # 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)
```

```
WARNING:tensorflow:From <ipython-input-7-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, 3, 32] 0.046984356
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028825352
conv2/b:0 [64] 0.0
fc1/W:0 [4096, 256] 0.021429416
fc1/b:0 [256] 0.0
fc2/W:0 [256, 10] 0.086599186
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 1.9990837574005127
get stats time 0.493161678314209
Epoch 0 Train loss, accuracy 1.863172179853916 0.3278
get stats time 0.161482572555542
EPoch 0 Validation loss, accuracy 1.8868865665435792 0.3156
Batch 0.0
Batch 10.0
Epoch time 0.87320876121521
Batch 0.0
Batch 10.0
Epoch time 0.8846406936645508
get stats time 0.2910654544830322
Epoch 2 Train loss, accuracy 1.5180832846283914 0.46369999999999995
get stats time 0.15743207931518555
EPoch 2 Validation loss, accuracy 1.5773222689151765 0.43179999999999996
Batch 0.0
Batch 10.0
Epoch time 0.8718745708465576
Batch 0.0
Batch 10.0
Epoch time 0.8689284324645996
get stats time 0.29021525382995605
```

```
Epoch 4 Train loss, accuracy 1.327112876856327 0.5317000000000001
get stats time 0.14542317390441895
EPoch 4 Validation loss, accuracy 1.406708476495743 0.5035999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8952877521514893
Batch 0.0
Batch 10.0
Epoch time 0.8787014484405518
get stats time 0.29273295402526855
Epoch 6 Train loss, accuracy 1.2335881274819376 0.5582
get stats time 0.1576552391052246
EPoch 6 Validation loss, accuracy 1.356745616197586 0.5126000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8736040592193604
Batch 0.0
Batch 10.0
Epoch time 0.8763563632965088
get stats time 0.31379199028015137
Epoch 8 Train loss, accuracy 1.1219199019908905 0.6068999999999999
get stats time 0.14529085159301758
EPoch 8 Validation loss, accuracy 1.2724182149887084 0.5526000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8805692195892334
Batch 0.0
Batch 10.0
Epoch time 0.8743829727172852
get stats time 0.2927415370941162
Epoch 10 Train loss, accuracy 1.0415601162910462 0.6352
get stats time 0.14495444297790527
EPoch 10 Validation loss, accuracy 1.225133366847038 0.5688
Batch 0.0
Batch 10.0
Epoch time 0.8815665245056152
Batch 0.0
Batch 10.0
Epoch time 0.8746922016143799
get stats time 0.2936131954193115
Epoch 12 Train loss, accuracy 0.96606361451149 0.6609
get stats time 0.15813541412353516
EPoch 12 Validation loss, accuracy 1.1963706147670745 0.5776
Batch 0.0
Batch 10.0
Epoch time 0.8737316131591797
Batch 0.0
Batch 10.0
```

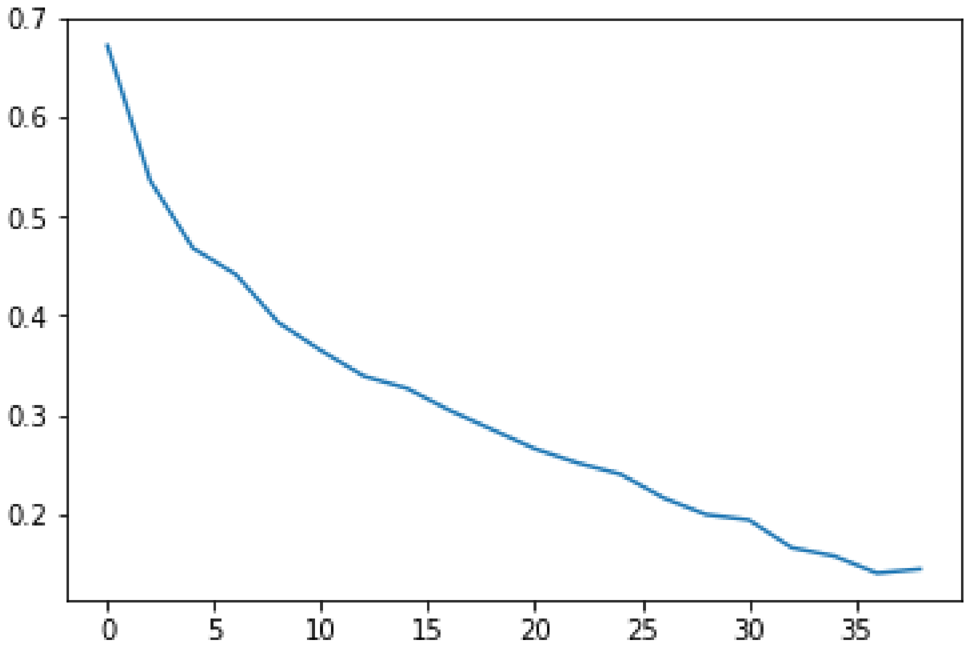
```
Epoch time 0.8742306232452393
get stats time 0.3066394329071045
Epoch 14 Train loss, accuracy 0.9227950039863586 0.6731
get stats time 0.14494705200195312
EPoch 14 Validation loss, accuracy 1.2121197356939315 0.579
Batch 0.0
Batch 10.0
Epoch time 0.8886830806732178
Batch 0.0
Batch 10.0
Epoch time 0.8725862503051758
get stats time 0.29468321800231934
Epoch 16 Train loss, accuracy 0.8650607248067856 0.6954
get stats time 0.14551353454589844
EPoch 16 Validation loss, accuracy 1.183392085647583 0.5972000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8721611499786377
Batch 0.0
Batch 10.0
Epoch time 0.8731443881988525
get stats time 0.2930135726928711
Epoch 18 Train loss, accuracy 0.8116055133461952 0.7148000000000001
get stats time 0.1583092212677002
EPoch 18 Validation loss, accuracy 1.1699283291578293 0.6042
Batch 0.0
Batch 10.0
Epoch time 0.8741278648376465
Batch 0.0
Batch 10.0
Epoch time 0.8748683929443359
get stats time 0.29314231872558594
Epoch 20 Train loss, accuracy 0.7640025320172309 0.7341999999999999
get stats time 0.1450502872467041
EPoch 20 Validation loss, accuracy 1.152965164422989 0.6083999999999999
Batch 0.0
Batch 10.0
Epoch time 0.871828556060791
Batch 0.0
Batch 10.0
Epoch time 0.8732204437255859
get stats time 0.29369330406188965
Epoch 22 Train loss, accuracy 0.7300490732431412 0.7484999999999998
get stats time 0.14539384841918945
EPoch 22 Validation loss, accuracy 1.1664470425605773 0.6113999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8718967437744141
```

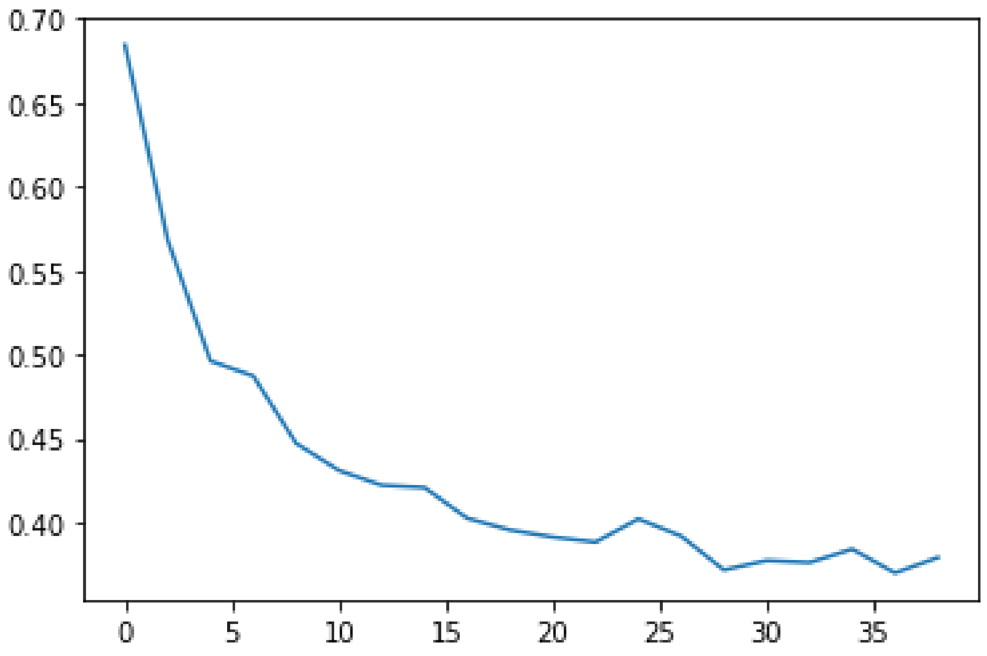
```
Batch 0.0
Batch 10.0
Epoch time 0.8728914260864258
get stats time 0.2941780090332031
Epoch 24 Train loss, accuracy 0.7027577541232108 0.7597999999999999
get stats time 0.16634416580200195
EPoch 24 Validation loss, accuracy 1.1948769769191743 0.5976
Batch 0.0
Batch 10.0
Epoch time 0.9003617763519287
Batch 0.0
Batch 10.0
Epoch time 0.8755309581756592
get stats time 0.29096078872680664
Epoch 26 Train loss, accuracy 0.6225623414874077 0.7835
get stats time 0.14500689506530762
EPoch 26 Validation loss, accuracy 1.181925915789604 0.6078
Batch 0.0
Batch 10.0
Epoch time 0.8765192031860352
Batch 0.0
Batch 10.0
Epoch time 0.8760986328125
get stats time 0.29222726821899414
Epoch 28 Train loss, accuracy 0.5829507814526558 0.8003
get stats time 0.1457357406616211
EPoch 28 Validation loss, accuracy 1.1706540600061417 0.6279999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8757503032684326
Batch 0.0
Batch 10.0
Epoch time 0.8752844333648682
get stats time 0.2952713966369629
Epoch 30 Train loss, accuracy 0.5540920521140098 0.8057000000000001
get stats time 0.14516448974609375
EPoch 30 Validation loss, accuracy 1.203604176735878 0.6224
Batch 0.0
Batch 10.0
Epoch time 0.8757984638214111
Batch 0.0
Batch 10.0
Epoch time 0.8860471248626709
get stats time 0.2925682067871094
Epoch 32 Train loss, accuracy 0.49204750835895544 0.8337
get stats time 0.15894031524658203
EPoch 32 Validation loss, accuracy 1.1909996050834657 0.6235999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8770313262939453
Batch 0.0
Batch 10.0
Epoch time 0.8736071586608887
get stats time 0.29212188720703125
Epoch 34 Train loss, accuracy 0.46666788419485095 0.8422000000000001
get stats time 0.14536309242248535
EPoch 34 Validation loss, accuracy 1.2323204615116121 0.6156
Batch 0.0
Batch 10.0
Epoch time 0.8737554550170898
Batch 0.0
Batch 10.0
Epoch time 0.873265266418457
get stats time 0.29297709465026855
Epoch 36 Train loss, accuracy 0.4157168828248977 0.8594000000000002
get stats time 0.14602351188659668
EPoch 36 Validation loss, accuracy 1.2172950077056885 0.6298
Batch 0.0
Batch 10.0
Epoch time 0.8760087490081787
Batch 0.0
Batch 10.0
Epoch time 0.8736152648925781
get stats time 0.2927098274230957
Epoch 38 Train loss, accuracy 0.41117919256687163 0.8553
get stats time 0.14602899551391602
EPoch 38 Validation loss, accuracy 1.2548944417476655 0.6205999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8737175464630127
test accuracy 0.5969
```



```
Model saved in path: tmp/model
```

```
In [18]: es = [2*i for i in range(len(err_train))]
         plt.plot(es,err_train)
         plt.show()
         plt.plot(es,err_val)
         plt.show()
```





12 (C) deeper network

I added an additional layer fc1_2 and the size of fc1 and fc1_2 are 250 and 128. From the experiment above, I cannot see obvious improvement (from 0.5843 to 0.5991)

```
In [14]: # Run the training
         import time
         batch_size=500
         step_size=.001
         num_epochs=40
         num_train=10000
         minimizer="Adam"
         data_set="cifar"
         model_name="model_deeper"
         keep_prob=.5
         dim=28
         nchannels=1
         if (data_set=="cifar"):
             dim=32
             nchannels=3

         err_train_d = []
```

```
        err_val_d = []

        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()

                # 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_).minimize
                # 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_d.append(1-ac)

                        print('Epoch',i,'Train loss, accuracy',lo,ac)
                        vlo,vac = get_stats(val[0],val[1])

                        err_val_d.append(1-vac)
                        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]}))
```

```
WARNING:tensorflow:From <ipython-input-8-ce4f47872b50>: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, 3, 32] 0.047396977
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.028806174
conv2/b:0 [64] 0.0
fc1/W:0 [4096, 250] 0.021458508
fc1/b:0 [250] 0.0
fc1_2/W:0 [250, 128] 0.07284773
fc1_2/b:0 [128] 0.0
fc2/W:0 [128, 10] 0.1187846
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 1.9196546077728271
get stats time 0.500112771987915
Epoch 0 Train loss, accuracy 1.8630122694849969 0.31980000000000003
get stats time 0.16240453720092773
EPoch 0 Validation loss, accuracy 1.8936020859718323 0.3126
Batch 0.0
Batch 10.0
Epoch time 0.8744633197784424
Batch 0.0
Batch 10.0
Epoch time 0.867276668548584
get stats time 0.2916390895843506
Epoch 2 Train loss, accuracy 1.4893339892983437 0.4631
get stats time 0.14678430557250977
EPoch 2 Validation loss, accuracy 1.530170536351204 0.4468
Batch 0.0
Batch 10.0
Epoch time 0.8688271045684814
Batch 0.0
Batch 10.0
Epoch time 0.9046802520751953
get stats time 0.29256653785705566
Epoch 4 Train loss, accuracy 1.3234235776424408 0.526
get stats time 0.15884041786193848
EPoch 4 Validation loss, accuracy 1.4017089242935181 0.49660000000000004
Batch 0.0
Batch 10.0
Epoch time 0.8717293739318848
Batch 0.0
Batch 10.0
```

```
Epoch time 0.8876552581787109
get stats time 0.2938883304595947
Epoch 6 Train loss, accuracy 1.26165203332901 0.5444
get stats time 0.15798354148864746
EPoch 6 Validation loss, accuracy 1.3924041431188583 0.5050000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8654799461364746
Batch 0.0
Batch 10.0
Epoch time 0.8620374202728271
get stats time 0.29701733589172363
Epoch 8 Train loss, accuracy 1.1399832675218582 0.5862
get stats time 0.14718914031982422
EPoch 8 Validation loss, accuracy 1.2800735924959181 0.536
Batch 0.0
Batch 10.0
Epoch time 0.8895666599273682
Batch 0.0
Batch 10.0
Epoch time 0.8648829460144043
get stats time 0.29293203353881836
Epoch 10 Train loss, accuracy 1.0888375244617463 0.6073000000000001
get stats time 0.14758753776550293
EPoch 10 Validation loss, accuracy 1.269505068230629 0.5536000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8801357746124268
Batch 0.0
Batch 10.0
Epoch time 0.8629541397094727
get stats time 0.2940328121185303
Epoch 12 Train loss, accuracy 0.9741626506090164 0.6521000000000001
get stats time 0.15912461280822754
EPoch 12 Validation loss, accuracy 1.197868437719345 0.5793999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8704791069030762
Batch 0.0
Batch 10.0
Epoch time 0.8670120239257812
get stats time 0.30141544342041016
Epoch 14 Train loss, accuracy 0.9167774410009384 0.6776
get stats time 0.14582014083862305
EPoch 14 Validation loss, accuracy 1.168568877673149 0.595
Batch 0.0
Batch 10.0
Epoch time 0.8847308158874512
```

```
Batch 0.0
Batch 10.0
Epoch time 0.8668675422668457
get stats time 0.2915458679199219
Epoch 16 Train loss, accuracy 0.8655229182362556 0.694
get stats time 0.14722156524658203
EPoch 16 Validation loss, accuracy 1.1835206122875213 0.5894
Batch 0.0
Batch 10.0
Epoch time 0.8695440292358398
Batch 0.0
Batch 10.0
Epoch time 0.8645412921905518
get stats time 0.29439663887023926
Epoch 18 Train loss, accuracy 0.793851766872406 0.7255999999999998
get stats time 0.15830779075622559
EPoch 18 Validation loss, accuracy 1.1644181807279588 0.6042
Batch 0.0
Batch 10.0
Epoch time 0.8690671920776367
Batch 0.0
Batch 10.0
Epoch time 0.8653333187103271
get stats time 0.29299187660217285
Epoch 20 Train loss, accuracy 0.7434632746934889 0.7372
get stats time 0.14639043807983398
EPoch 20 Validation loss, accuracy 1.1644230477809907 0.598
Batch 0.0
Batch 10.0
Epoch time 0.8711421489715576
Batch 0.0
Batch 10.0
Epoch time 0.8654382228851318
get stats time 0.29375624656677246
Epoch 22 Train loss, accuracy 0.6737018588066102 0.7581
get stats time 0.14647221565246582
EPoch 22 Validation loss, accuracy 1.1739661108255386 0.613
Batch 0.0
Batch 10.0
Epoch time 0.8725829124450684
Batch 0.0
Batch 10.0
Epoch time 0.8633792400360107
get stats time 0.3009495735168457
Epoch 24 Train loss, accuracy 0.6273052023887635 0.7816
get stats time 0.14694690704345703
EPoch 24 Validation loss, accuracy 1.1719592448472977 0.6154
Batch 0.0
Batch 10.0
Epoch time 0.8888516426086426
Batch 0.0
Batch 10.0
Epoch time 0.8687708377838135
get stats time 0.29523611068725586
Epoch 26 Train loss, accuracy 0.578309802877903 0.8
get stats time 0.14721941947937012
EPoch 26 Validation loss, accuracy 1.190641682744026 0.6075999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8702008724212646
Batch 0.0
Batch 10.0
Epoch time 0.8659641742706299
get stats time 0.29200029373168945
Epoch 28 Train loss, accuracy 0.5403030438005925 0.8106000000000002
get stats time 0.1472163200378418
EPoch 28 Validation loss, accuracy 1.212728726243973 0.6142
Batch 0.0
Batch 10.0
Epoch time 0.8709876537322998
Batch 0.0
Batch 10.0
Epoch time 0.8675510883331299
get stats time 0.2923905849456787
Epoch 30 Train loss, accuracy 0.47223591870069515 0.835
get stats time 0.14725184440612793
EPoch 30 Validation loss, accuracy 1.1915774752378465 0.6308
Batch 0.0
Batch 10.0
Epoch time 0.8772025108337402
Batch 0.0
Batch 10.0
Epoch time 0.8682107925415039
get stats time 0.2927236557006836
Epoch 32 Train loss, accuracy 0.44606016927957537 0.8437999999999999
get stats time 0.15893840789794922
EPoch 32 Validation loss, accuracy 1.2431449763536453 0.6252
Batch 0.0
Batch 10.0
Epoch time 0.8706822395324707
Batch 0.0
Batch 10.0
Epoch time 0.8663339614868164
get stats time 0.29186320304870605
Epoch 34 Train loss, accuracy 0.4055279045343399 0.8577999999999999
get stats time 0.14813828468322754
```

```
EPoch 34 Validation loss, accuracy 1.3148659232139588 0.6162
Batch 0.0
Batch 10.0
Epoch time 0.8716936111450195
Batch 0.0
Batch 10.0
Epoch time 0.8686792850494385
get stats time 0.2938055992126465
Epoch 36 Train loss, accuracy 0.36409663974046713 0.8752000000000001
get stats time 0.1473701000213623
EPoch 36 Validation loss, accuracy 1.310774439740181 0.6174
Batch 0.0
Batch 10.0
Epoch time 0.8722119331359863
Batch 0.0
Batch 10.0
Epoch time 0.8656387329101562
get stats time 0.2940211296081543
Epoch 38 Train loss, accuracy 0.32687467352151867 0.8922000000000001
get stats time 0.1480855941772461
EPoch 38 Validation loss, accuracy 1.359535463809967 0.619
Batch 0.0
Batch 10.0
Epoch time 0.8727898597717285
test accuracy 0.6013
```

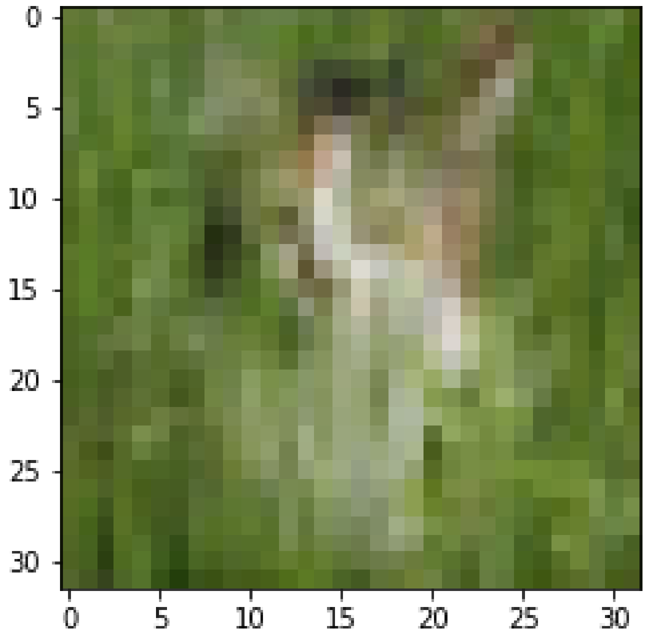
12.1 Comment

There is improvement when I add a layer between fc1 and fc2, from 0.5843 to 0.6013

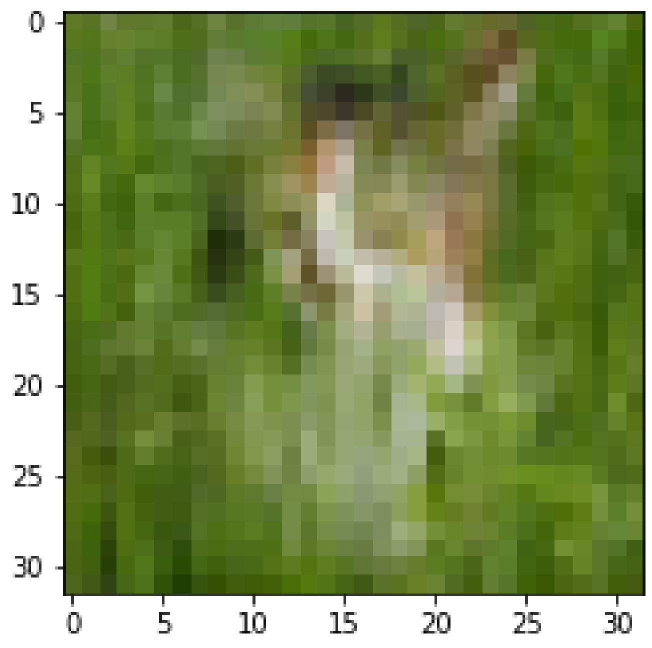
13 (d) Variability

```
In [14]: from skimage import color
         from skimage.color import rgb2hsv

In [15]: train, val,  test = get_cifar()

In [16]: import random
         random.seed(12345)
         test_image = test[0].copy()
         test_image_hsv = [skimage.color.rgb2hsv(x.reshape(32,32,3)) for x in test_image]

         def change_h(x):
             x[:,:,1] = random.uniform(0.75,1.25)*x[:,:,1]
             return x
         test_image_hsv_t = list(map(change_h,test_image_hsv))
         test_image_rgb_t = [skimage.color.hsv2rgb(x) for x in test_image_hsv_t]
```

```
In [18]: pick = 456
         plt.imshow(test[0][pick,:].reshape(32,32,3))
         plt.show()
         plt.imshow(test_image_rgb_t[pick])
         plt.show()
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```



```
In [14]: # Run the training

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

         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_).minimize
                 # 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-vac)
                         print('EPoch',i,'Validation loss, accuracy',vlo,vac)
                         # Test set accuracy

                 import random
                 random.seed(12345)
                 test_image = test[0].copy()
                 test_image_hsv = [skimage.color.rgb2hsv(x.reshape(32,32,3)) for x in test_imag

                 def change_h(x):
                     x[:,:,1] = random.uniform(0.75,1.25)*x[:,:,1]
                     return x
                 test_image_hsv_t = list(map(change_h,test_image_hsv))
                 test_image_rgb_t = [skimage.color.hsv2rgb(x) for x in test_image_hsv_t]
                 test_t = [x.reshape(3072) for x in test_image_rgb_t]
                 test_t = np.array(test_t)
```

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

```
WARNING:tensorflow:From <ipython-input-7-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, 3, 32] 0.047007147
conv1/b:0 [32] 0.0
conv2/W:0 [5, 5, 32, 64] 0.0289418
conv2/b:0 [64] 0.0
fc1/W:0 [4096, 256] 0.021446623
fc1/b:0 [256] 0.0
fc2/W:0 [256, 10] 0.08665953
fc2/b:0 [10] 0.0
Batch 0.0
Batch 10.0
Epoch time 1.9462459087371826
get stats time 0.49600839614868164
Epoch 0 Train loss, accuracy 1.856021707868576 0.33330000000000004
get stats time 0.15366435050964355
EPoch 0 Validation loss, accuracy 1.8720796416521073 0.3266
Batch 0.0
Batch 10.0
Epoch time 0.8878448009490967
Batch 0.0
Batch 10.0
Epoch time 0.8631460666656494
get stats time 0.3151071071624756
Epoch 2 Train loss, accuracy 1.5241077013254167 0.457
get stats time 0.15688180923461914
EPoch 2 Validation loss, accuracy 1.5786763971090316 0.4298
Batch 0.0
Batch 10.0
Epoch time 0.860346794128418
Batch 0.0
Batch 10.0
Epoch time 0.8807461261749268
get stats time 0.2922646999359131
Epoch 4 Train loss, accuracy 1.3751582521677015 0.5046999999999999
get stats time 0.15831637382507324
EPoch 4 Validation loss, accuracy 1.4415810370206834 0.48019999999999996
Batch 0.0
Batch 10.0
Epoch time 0.8605136871337891
Batch 0.0
Batch 10.0
Epoch time 0.8563997745513916
get stats time 0.29670071601867676
Epoch 6 Train loss, accuracy 1.2364521255612373 0.5648
get stats time 0.14582109451293945
EPoch 6 Validation loss, accuracy 1.3626994460344313 0.5148
Batch 0.0
Batch 10.0
Epoch time 0.8771209716796875
Batch 0.0
Batch 10.0
Epoch time 0.8616776466369629
get stats time 0.2941622734069824
Epoch 8 Train loss, accuracy 1.1204081438064575 0.602
get stats time 0.14689159393310547
EPoch 8 Validation loss, accuracy 1.2880287616491315 0.5471999999999999
Batch 0.0
Batch 10.0
Epoch time 0.868755578994751
Batch 0.0
Batch 10.0
Epoch time 0.8580150604248047
get stats time 0.2929966449737549
Epoch 10 Train loss, accuracy 1.0680138318657877 0.6152999999999998
get stats time 0.15915727615356445
EPoch 10 Validation loss, accuracy 1.2717671947956086 0.5516000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8696944713592529
Batch 0.0
Batch 10.0
Epoch time 0.8623621463775635
get stats time 0.3002181053161621
Epoch 12 Train loss, accuracy 0.9702171021580697 0.6618
get stats time 0.14658403396606445
EPoch 12 Validation loss, accuracy 1.208362540817261 0.5763999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8809394836425781
Batch 0.0
Batch 10.0
Epoch time 0.8600709438323975
get stats time 0.29320764541625977
Epoch 14 Train loss, accuracy 0.9084199856758118 0.6776
get stats time 0.14651203155517578
```

```
EPoch 14 Validation loss, accuracy 1.1997952248096466 0.5923999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8659214973449707
Batch 0.0
Batch 10.0
Epoch time 0.8648200035095215
get stats time 0.29303479194641113
Epoch 16 Train loss, accuracy 0.8380883897185326 0.7098
get stats time 0.15759873390197754
EPoch 16 Validation loss, accuracy 1.1822237486362457 0.5913999999999999
Batch 0.0
Batch 10.0
Epoch time 0.864891767501831
Batch 0.0
Batch 10.0
Epoch time 0.8584306240081787
get stats time 0.2937438488006592
Epoch 18 Train loss, accuracy 0.7931444250226021 0.7214
get stats time 0.14727091789245605
EPoch 18 Validation loss, accuracy 1.1796606083869934 0.6014
Batch 0.0
Batch 10.0
Epoch time 0.862440824508667
Batch 0.0
Batch 10.0
Epoch time 0.8610169887542725
get stats time 0.29401230812072754
Epoch 20 Train loss, accuracy 0.727279320716858 0.7443
get stats time 0.1461038589477539
EPoch 20 Validation loss, accuracy 1.1703539198875426 0.6102000000000001
Batch 0.0
Batch 10.0
Epoch time 0.8633122444152832
Batch 0.0
Batch 10.0
Epoch time 0.8590598106384277
get stats time 0.294769287109375
Epoch 22 Train loss, accuracy 0.6867102633953095 0.7634000000000001
get stats time 0.14583182334899902
EPoch 22 Validation loss, accuracy 1.202523431587219 0.6026
Batch 0.0
Batch 10.0
Epoch time 0.8837893009185791
Batch 0.0
Batch 10.0
Epoch time 0.862234354019165
get stats time 0.29305052757263184
```

```
Epoch 24 Train loss, accuracy 0.6258232118487358 0.7834000000000001
get stats time 0.14745712280273438
EPoch 24 Validation loss, accuracy 1.1855518007278445 0.6106
Batch 0.0
Batch 10.0
Epoch time 0.8681864738464355
Batch 0.0
Batch 10.0
Epoch time 0.8642861843109131
get stats time 0.2929973602294922
Epoch 26 Train loss, accuracy 0.5773102808713914 0.8018000000000001
get stats time 0.14739274978637695
EPoch 26 Validation loss, accuracy 1.183936979007721 0.6121999999999999
Batch 0.0
Batch 10.0
Epoch time 0.8646657466888428
Batch 0.0
Batch 10.0
Epoch time 0.8644530773162842
get stats time 0.29441094398498535
Epoch 28 Train loss, accuracy 0.5480798717379569 0.813
get stats time 0.14740729331970215
EPoch 28 Validation loss, accuracy 1.2386078662157058 0.6072
Batch 0.0
Batch 10.0
Epoch time 0.8707497119903564
Batch 0.0
Batch 10.0
Epoch time 0.8650946617126465
get stats time 0.29235291481018066
Epoch 30 Train loss, accuracy 0.49083272221088403 0.8343999999999999
get stats time 0.15881776809692383
EPoch 30 Validation loss, accuracy 1.2200721614122392 0.6226
Batch 0.0
Batch 10.0
Epoch time 0.8668134212493896
Batch 0.0
Batch 10.0
Epoch time 0.8666508197784424
get stats time 0.29369163513183594
Epoch 32 Train loss, accuracy 0.44524661531448356 0.8522000000000001
get stats time 0.14701247215270996
EPoch 32 Validation loss, accuracy 1.217758526802063 0.6106
Batch 0.0
Batch 10.0
Epoch time 0.8713250160217285
Batch 0.0
Batch 10.0
```

```
Epoch time 0.8617472648620605
get stats time 0.2938683032989502
Epoch 34 Train loss, accuracy 0.4180358002424239 0.8629
get stats time 0.1472170352935791
EPoch 34 Validation loss, accuracy 1.2668621389150618 0.6172
Batch 0.0
Batch 10.0
Epoch time 0.86692214012146
Batch 0.0
Batch 10.0
Epoch time 0.867497444152832
get stats time 0.29341626167297363
Epoch 36 Train loss, accuracy 0.38145843294858933 0.8727
get stats time 0.1473984718322754
EPoch 36 Validation loss, accuracy 1.2886421985626222 0.6208
Batch 0.0
Batch 10.0
Epoch time 0.8676199913024902
Batch 0.0
Batch 10.0
Epoch time 0.8642239570617676
get stats time 0.2920067310333252
Epoch 38 Train loss, accuracy 0.3487257687807083 0.8867999999999998
get stats time 0.1485152244567871
EPoch 38 Validation loss, accuracy 1.331723384475708 0.6076
Batch 0.0
Batch 10.0
Epoch time 0.8691592216491699
test accuracy 0.6018
```

13.1 Comment

The test accuracy on the transformed data does not change much. This time it even performs better, with a 0.6018 accuracy rate compared with 0.5969 in unchanged data.