

# Assignment4

November 29, 2017

## 1 CSE 252A Computer Vision I Fall 2017

### 1.1 Assignment 4

#### 1.2 Problem 1: Install Tensorflow [2 pts]

Follow the directions on <https://www.tensorflow.org/install/> to install Tensorflow on your computer.

Note: You will not need GPU support for this assignment so don't worry if you don't have one. Furthermore, installing with GPU support is often more difficult to configure so it is suggested that you install the CPU only version. However, if you have a GPU and would like to install GPU support feel free to do so at your own risk :)

Note: On windows, Tensorflow is only supported in python3, so you will need to install python3 for this assignment.

Run the following cell to verify your installation.

```
In [ ]: import tensorflow as tf
        hello = tf.constant('Hello, TensorFlow!')
        sess = tf.Session()
        print(sess.run(hello))
```

#### 1.3 Problem 2: Downloading CIFAR10 [1 pts]

Download the CIFAR10 dataset (<http://www.cs.toronto.edu/~kriz/cifar.html>). You will need the python version: <http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>

Extract the data to ./data Once extracted run the following cell to view a few example images.

```
In [1]: import numpy as np

        # unpickles raw data files
        def unpickle(file):
            import pickle
            import sys
            with open(file, 'rb') as fo:
                if sys.version_info[0] < 3:
                    dict = pickle.load(fo)
                else:
                    dict = pickle.load(fo, encoding='bytes')
```

```

    return dict

# loads data from a single file
def getBatch(file):
    dict = unpickle(file)
    data = dict[b'data'].reshape(-1,3,32,32).transpose(0,2,3,1)
    labels = np.asarray(dict[b'labels'], dtype=np.int64)
    return data,labels

# loads all training and testing data
def getData(path='./data'):
    classes = [s.decode('UTF-8') for s in unpickle(path+'/batches.meta')[b'label_names']]

    trainData, trainLabels = [], []
    for i in range(5):
        data, labels = getBatch(path+'/data_batch_%d'%(i+1))
        trainData.append(data)
        trainLabels.append(labels)
    trainData = np.concatenate(trainData)
    trainLabels = np.concatenate(trainLabels)

    testData, testLabels = getBatch(path+'/test_batch')
    return classes, trainData, trainLabels, testData, testLabels

# training and testing data that will be used in the following problems
classes, trainData, trainLabels, testData, testLabels = getData()

# display some example images
import matplotlib.pyplot as plt
%matplotlib inline

plt.figure(figsize=(14, 6))
for i in range(14):
    plt.subplot(2,7,i+1)
    plt.imshow(trainData[i])
    plt.title(classes[trainLabels[i]])
plt.show()

print ('train shape: ' + str(trainData.shape) + ', ' + str(trainLabels.shape))
print ('test shape : ' + str(testData.shape) + ', ' + str(testLabels.shape))

```



```
train shape: (50000, 32, 32, 3), (50000,)
test shape : (10000, 32, 32, 3), (10000,)
```

Below are some helper functions that will be used in the following problems.

```
In [ ]: # a generator for batches of data
# yields data (batchsize, 3, 32, 32) and labels (batchsize)
# if shuffle, it will load batches in a random order
def DataBatch(data, label, batchsize, shuffle=True):
    n = data.shape[0]
    if shuffle:
        index = np.random.permutation(n)
    else:
        index = np.arange(n)
    for i in range(int(np.ceil(n/batchsize))):
        inds = index[i*batchsize : min(n,(i+1)*batchsize)]
        yield data[inds], label[inds]

# tests the accuracy of a classifier
def test(testData, testLabels, classifier):
    batchsize=50
    correct=0.
    for data,label in DataBatch(testData,testLabels,batchsize):
        prediction = classifier(data)
        #print (prediction)
        correct += np.sum(prediction==label)
    return correct/testData.shape[0]*100

# a sample classifier
# given an input it outputs a random class
class RandomClassifier():
```

```

def __init__(self, classes=10):
    self.classes=classes
def __call__(self, x):
    return np.random.randint(self.classes, size=x.shape[0])

randomClassifier = RandomClassifier()
print ('Random classifier accuracy: %f'%test(testData, testLabels, randomClassifier))

```

#### 1.4 Problem 3: Confusion Matirx [5 pts]

Here you will implement a test script that computes the confusion matrix for a classifier. The matrix should be  $n \times n$  where  $n$  is the number of classes. Entry  $M[i,j]$  should contain the number of times an image of class  $i$  was classified as class  $j$ .  $M$  should be normalized such that each row sums to 1.

Hint: see the function `test()` above for reference.

```

In [ ]: def confusion(testData, testLabels, classifier):
        """your code here"""
        return M

def VisualizeConfussion(M):
    plt.figure(figsize=(14, 6))
    plt.imshow(M#, vmin=0, vmax=1)
    plt.xticks(np.arange(len(classes)), classes, rotation='vertical')
    plt.yticks(np.arange(len(classes)), classes)
    plt.show()

M = confusion(testData, testLabels, randomClassifier)
VisualizeConfussion(M)

```

#### 1.5 Problem 4: K-Nearest Neighbors (KNN) [5 pts]

Here you will implemnet a simple knn classifier. The distance metric is euclidian in pixel space.  $k$  refers to the number of neighbors involved in voting on the class.

Hint: you may want to use: `sklearn.neighbors.KNeighborsClassifier`

```

In [ ]: from sklearn.neighbors import KNeighborsClassifier
class KNNClassifier():
    def __init__(self, k=3):
        # k is the number of neighbors involved in voting
        """your code here"""

    def train(self, trainData, trainLabels):
        """your code here"""

    def __call__(self, x):
        # this method should take a batch of images (batchsize, 32, 32, 3) and return a

```

```

        # predictions should be int64 values in the range [0,9] corresponding to the cla
        """your code here"""

# test your classifier with only the first 100 training examples (use this while debugg
# note you should get around 10-20% accuracy
knnClassifierX = KNNClassifier()
knnClassifierX.train(trainData[:100], trainLabels[:100])
print ('KNN classifier accuracy: %f'%test(testData, testLabels, knnClassifierX))

```

```

In [ ]: # test your classifier with all the training examples (This may take a while)
# note you should get around 30% accuracy
knnClassifier = KNNClassifier()
knnClassifier.train(trainData, trainLabels)
print ('KNN classifier accuracy: %f'%test(testData, testLabels, knnClassifier))

# display confusion matrix for your KNN classifier with all the training examples
M = confusion(testData, testLabels, knnClassifier)
VisualizeConfussion(M)

```

## 1.6 Problem 5: Principal Component Analysis (PCA) K-Nearest Neighbors (KNN) [5 pts]

Here you will implemnet a simple knn classifier in PCA space. You should implement PCA your- self using svd (you may not use sklearn.decomposition.PCA or any other package that directly implements PCA transofrmations

Hint: Don't forget to apply the same normalization at test time.

Note: you should get similar accuracy to above, but it should run faster.

```

In [ ]: from sklearn.decomposition import PCA
class PCAKNNClassifier():
    def __init__(self, components=25, k=3):
        """your code here"""

    def train(self, trainData, trainLabels):
        """your code here"""

    def __call__(self, x):
        """your code here"""

# test your classifier with only the first 100 training examples (use this while debuggi
pcaknnClassifierX = PCAKNNClassifier()
pcaknnClassifierX.train(trainData[:100], trainLabels[:100])
print ('PCA-KNN classifier accuracy: %f'%test(testData, testLabels, pcaknnClassifierX))

```

```
In [ ]: # test your classifier with all the training examples (This may take a few minutes)
pcaknnClassifier = PCAKNNClassifier()
pcaknnClassifier.train(trainData, trainLabels)
print ('KNN classifier accuracy: %f'%test(testData, testLabels, pcaknnClassifier))

# display the confusion matrix
M = confusion(testData, testLabels, pcaknnClassifier)
VisualizeConfussion(M)
```

## 1.7 Deep learning

Below is some helper code to train your deep networks

Hint: see [https://www.tensorflow.org/get\\_started/mnist/pros](https://www.tensorflow.org/get_started/mnist/pros) or [https://www.tensorflow.org/get\\_started/mnist/beginners](https://www.tensorflow.org/get_started/mnist/beginners) for reference

```
In [ ]: # base class for your Tensorflow networks. It implements the training loop (train) and p
# You will need to implement the __init__ function to define the networks structures in
class TFClassifier():
    def __init__(self):
        pass

    def train(self, trainData, trainLabels, epochs=1, batchsize=50):
        self.prediction = tf.argmax(self.y,1)
        self.cross_entropy = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits)
        self.train_step = tf.train.AdamOptimizer(1e-4).minimize(self.cross_entropy)
        self.correct_prediction = tf.equal(self.prediction, self.y_)
        self.accuracy = tf.reduce_mean(tf.cast(self.correct_prediction, tf.float32))

        self.sess.run(tf.global_variables_initializer())

        for epoch in range(epochs):
            for i, (data,label) in enumerate(DataBatch(trainData, trainLabels, batchsize
                _, acc = self.sess.run([self.train_step, self.accuracy], feed_dict={self
                    #if i%100==99:
                    #    print ('%d/%d %d %f'%(epoch, epochs, i, acc))

            print ('testing epoch:%d accuracy: %f'%(epoch+1, test(testData, testLabels,

    def __call__(self, x):
        return self.sess.run(self.prediction, feed_dict={self.x: x})

# helper function to get weight variable
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.01)
    return tf.Variable(initial)

# helper function to get bias variable
def bias_variable(shape):
```

```

    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# example linear classifier
class LinearClassifier(TFClassifier):
    def __init__(self, classes=10):
        self.sess = tf.Session()

        self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of images
        self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels

        # model variables
        self.W = weight_variable([32*32*3,classes])
        self.b = bias_variable([classes])

        # linear operation
        self.y = tf.matmul(tf.reshape(self.x,(-1,32*32*3)),self.W) + self.b

# test the example linear classifier (note you should get around 20-30% accuracy)
linearClassifier = LinearClassifier()
linearClassifier.train(trainData, trainLabels, epochs=20)

# display confusion matrix
M = confusion(testData, testLabels, linearClassifier)
VisualizeConfusion(M)

```

## 1.8 Problem 6: Multi Layer Perceptron (MLP) [5 pts]

Here you will implement an MLP. The MLP should consist of 3 linear layers (matrix multiplication and bias offset) that map to the following feature dimensions:

```

32x32x3 -> hidden
hidden -> hidden
hidden -> classes

```

The first two linear layers should be followed with a ReLU nonlinearity. The final layer should not have a nonlinearity applied as we desire the raw logits output (see: the documentation for `tf.nn.sparse_softmax_cross_entropy_with_logits` used in the training)

The final output of the computation graph should be stored in `self.y` as that will be used in the training.

Hint: see the example linear classifier

Note: you should get around 50% accuracy

```

In [ ]: class MLPClassifier(TFClassifier):
        def __init__(self, classes=10, hidden=100):
            self.sess = tf.Session()

            self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of images
            self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels

```

```

        """your code here"""

# test your MLP classifier (note you should get around 50% accuracy)
mlpClassifier = MLPClassifier()
mlpClassifier.train(trainData, trainLabels, epochs=20)

# display confusion matrix
M = confusion(testData, testLabels, mlpClassifier)
VisualizeConfussion(M)

```

## 1.9 Problem 7: Convolutional Neural Netork (CNN) [7 pts]

Here you will implement a CNN with the following architecture:

```

ReLU( Conv(kernel_size=4x4 stride=2, output_features=n) )
ReLU( Conv(kernel_size=4x4 stride=2, output_features=n*2) )
ReLU( Conv(kernel_size=4x4 stride=2, output_features=n*4) )
Linear(output_features=classes)

```

```

In [ ]: def conv2d(x, W, stride=2):
        return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='SAME')

class CNNClassifier(TFClassifier):
    def __init__(self, classes=10, n=16):
        self.sess = tf.Session()

        self.x = tf.placeholder(tf.float32, shape=[None,32,32,3]) # input batch of image
        self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels

        """your code here"""

# test your CNN classifier (note you should get around 65% accuracy)
cnnClassifier = CNNClassifier()
cnnClassifier.train(trainData, trainLabels, epochs=20)

# display confusion matrix
M = confusion(testData, testLabels, cnnClassifier)
VisualizeConfussion(M)

```

## 1.10 Further reference

To see how state of the art deep networks do on this dataset see:  
<https://github.com/tensorflow/models/tree/master/research/resnet>