hw6

January 29, 2021

1 Machine Learning and Computer Vision

1.1 Assigment 6

This assignment contains Tensorflow programming exercises.

1.2 Problem 1: Install Tensorflow

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 instalation.

[1]:
```python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
tf.print(hello)
```

```
Hello, TensorFlow!
```

1.3 Problem 2: Downloading CIFAR10

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.

[2]:
```python
import numpy as np

# unpickles raw data files
def unpickle(file):
    import pickle
    import sys
```

```python
    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.

[14]:

```python
# 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))
```

```
Random classifier accuracy: 9.900000
```

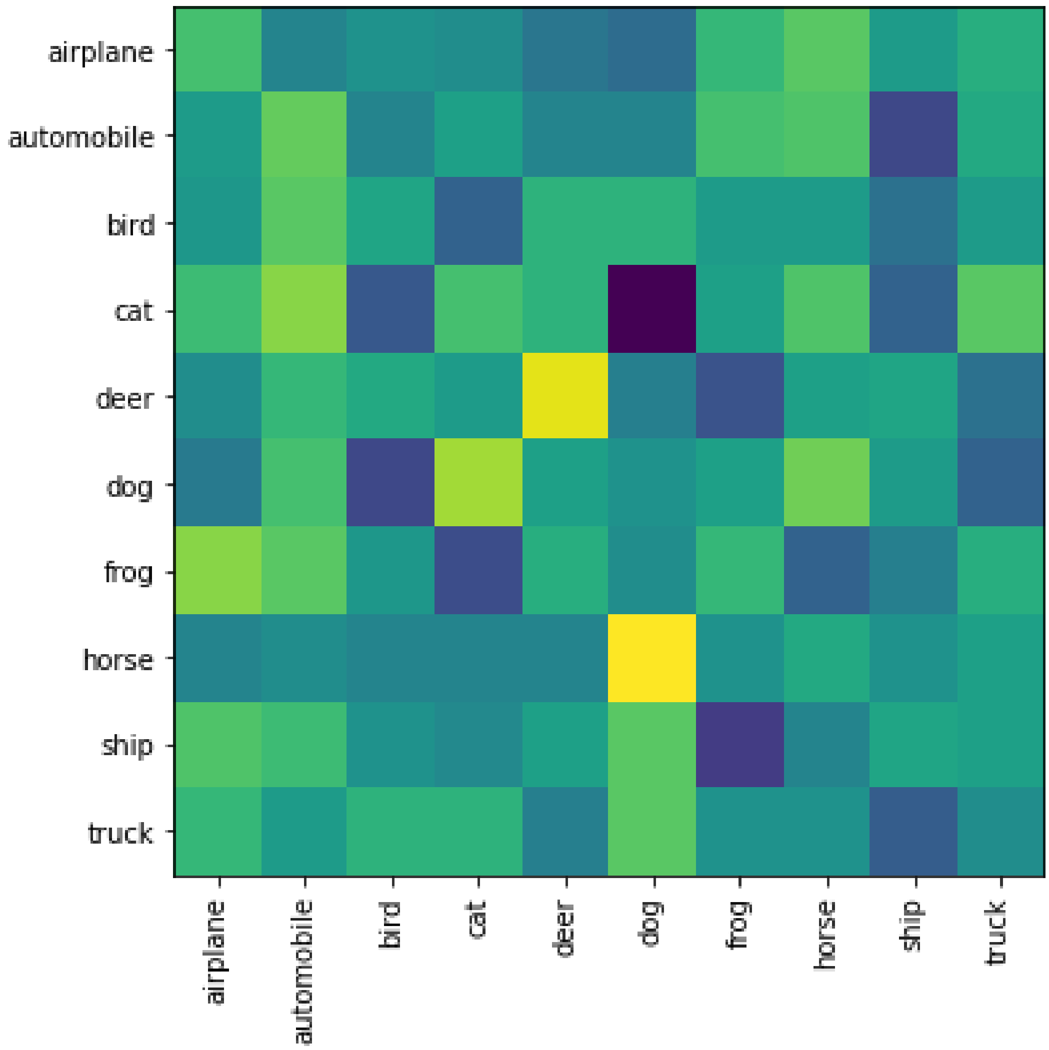
1.4 Problem 3: Confusion Matirx

Here you will implement a test script that computes the confussion matrix for a classifier. The matrix should be nxn 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.

[15]:
```
def confusion(testData, testLabels, classifier):
    '''
    Edition 1
    batchsize=50
    M = np.zeros((len(classes), len(classes)))

    for data,label in DataBatch(testData,testLabels,batchsize):

        prediction = classifier(data)

        for i in range(label[0]):
            for j in range(label[0]):
                M[i,j] += (prediction[j] == label[i])
        for i in range(M.shape[0]):
            n = np.sum(M[i, :])
            if (n != 0 ):
                M = np.divide(M, n)
    '''
    n = len(set(testLabels))
    prediction = classifier(testData)
    M = np.zeros((n,n))
    for i,j in zip(testLabels,prediction):
        M[i,j]+=1
    M=M/1000

    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)

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

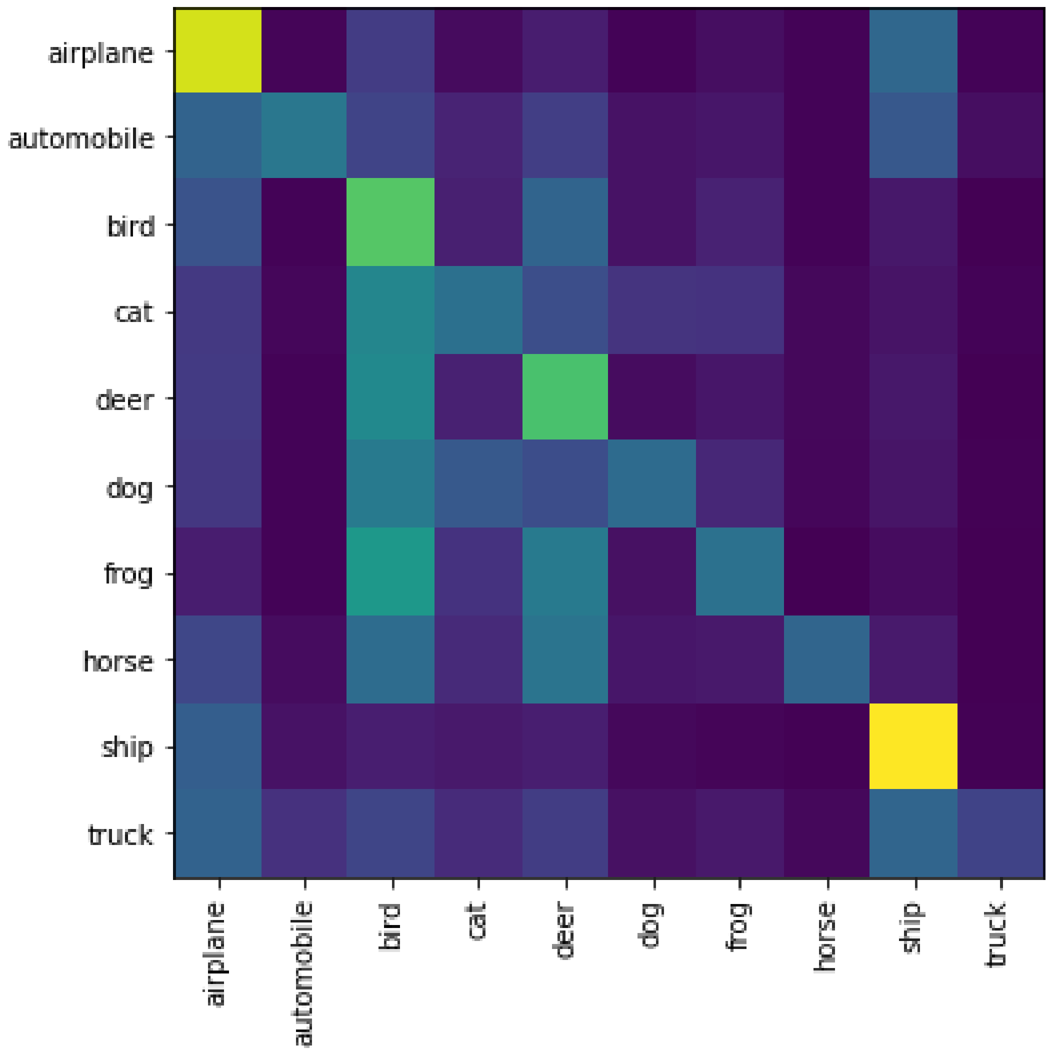
Hint: you may want to use: sklearn.neighbors.KNeighborsClassifier

[21]:
```python
from sklearn.neighbors import KNeighborsClassifier
class KNNClassifer():
    def __init__(self, k=3):
        # k is the number of neighbors involved in voting
        """your code here"""
        self.k = k
        self.clf = KNeighborsClassifier(n_neighbors=self.k)

    def train(self, trainData, trainLabels):
        """your code here"""
        batchsize=50

        data_pack = np.reshape(trainData, (-1,trainData.shape[1]*trainData.
 ↪shape[2]*trainData.shape[3]))
        self.clf = self.clf.fit(data_pack, trainLabels)


    def __call__(self, x):
        # this method should take a batch of images (batchsize, 32, 32, 3) and␣
 ↪return a batch of prediction (batchsize)
        # predictions should be int64 values in the range [0,9] corrisponding␣
 ↪to the class that the image belongs to
        x = np.reshape(x,(-1,x.shape[1]*x.shape[2]*x.shape[3]))
        prediction = self.clf.predict(x)
        return prediction


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

```
KNN classifier accuracy: 16.600000
```

[22]:
```python
# test your classifier with all the training examples (This may take a while)
# note you should get around 30% accuracy
knnClassifer = KNNClassifer()
knnClassifer.train(trainData, trainLabels)
print ('KNN classifier accuracy: %f'%test(testData, testLabels, knnClassifer))

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

```
KNN classifier accuracy: 33.030000
```



1.6 Problem 5: Principal Component Analysis (PCA) K-Nearest Neighbors (KNN)

Here you will implemnet a simple knn classifer in PCA space. You should implement PCA yourself 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.

[27]:
```
from sklearn.decomposition import PCA
class PCAKNNClassifer():
    def __init__(self, components=25, k=3):
        self.components = components
```

```
        self.k = k
        self.model = KNeighborsClassifier(n_neighbors=self.k)

    def train(self, trainData, trainLabels):
        data_pack = np.reshape(trainData, (-1,trainData.shape[1]*trainData.
 ↪shape[2]*trainData.shape[3]))
        data_mat = np.array(data_pack, dtype='float64')
        p,n = np.shape(data_mat)
        t = np.mean(data_mat, 0)
        data_mat = data_mat - t
        cov_mat = np.dot(data_mat.T, data_mat)/(p-1)
        u,s,vt = np.linalg.svd(cov_mat)
        self.u = u
        T2 = np.dot(data_mat, u[:,:self.components])
        self.model.fit(T2, trainLabels)

    def __call__(self, x):
        x = np.reshape(x, (-1, x.shape[1]*x.shape[2]*x.shape[3]))

        x_mat = np.array(x, dtype='float64')
        p,n = np.shape(x_mat)
        t = np.mean(x_mat, 0)
        x_mat = x_mat - t

        cov_mat = np.dot(x_mat.T, x_mat)/(p-1)
        T = np.dot(x_mat, self.u[:,:self.components])
        prediction = self.model.predict(T)
        return prediction



# test your classifier with only the first 100 training examples (use this␣
 ↪while debugging)
pcaknnClassiferX = PCAKNNClassifer()
pcaknnClassiferX.train(trainData[:100], trainLabels[:100])
print ('PCA-KNN classifier accuracy: %f'%test(testData, testLabels,␣
 ↪pcaknnClassiferX))
```

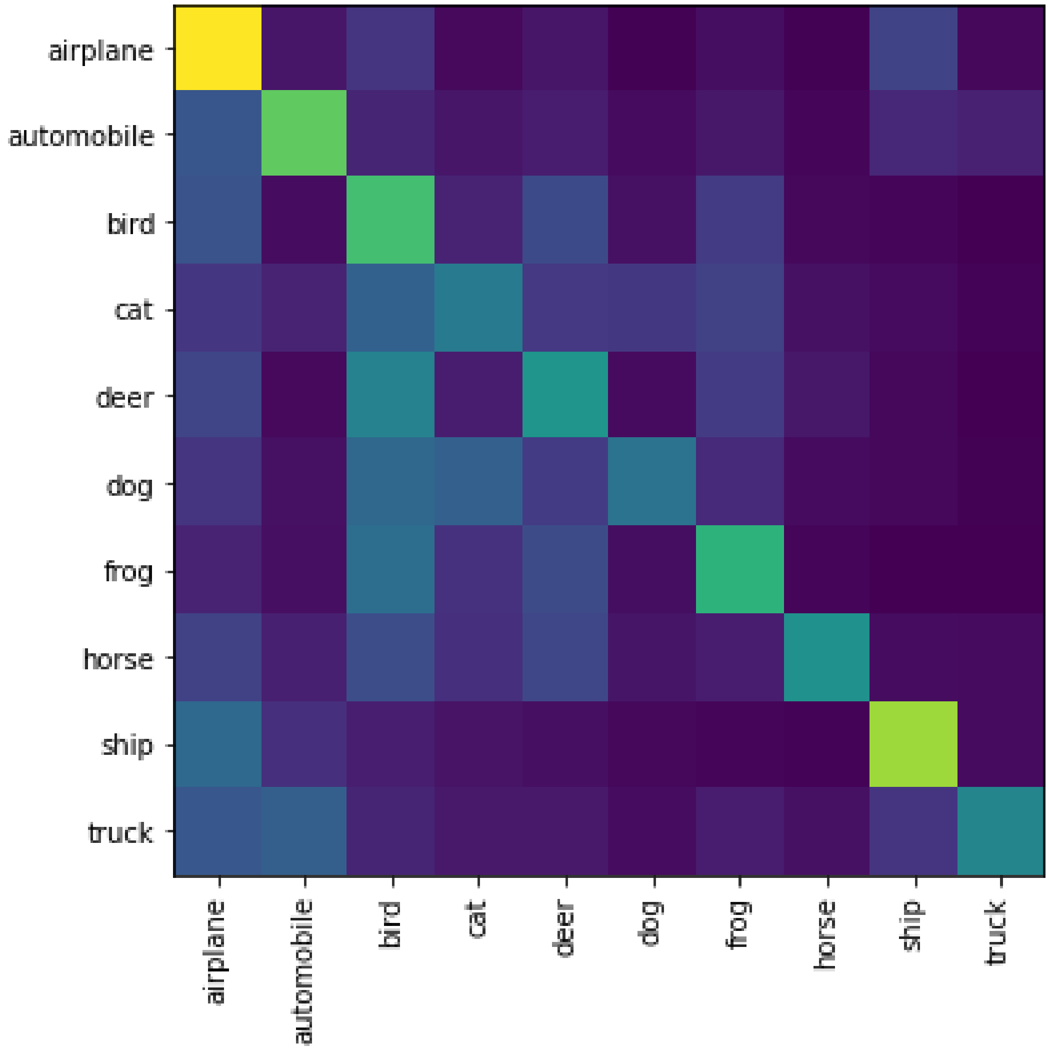
```
PCA-KNN classifier accuracy: 16.320000
```

[28]:

```
# test your classifier with all the training examples (This may take a few␣
 ↪minutes)
pcaknnClassifer = PCAKNNClassifer()
pcaknnClassifer.train(trainData, trainLabels)
print ('KNN classifier accuracy: %f'%test(testData, testLabels,␣
 ↪pcaknnClassifer))
```

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

```
KNN classifier accuracy: 37.300000
```



1.7 Deep learning

Below is some helper code to train your deep networks

Hint: see https://www.tensorflow.org/get_started/mnist/pros or https://www.tensorflow.org/get_started/mnist/beginners for reference

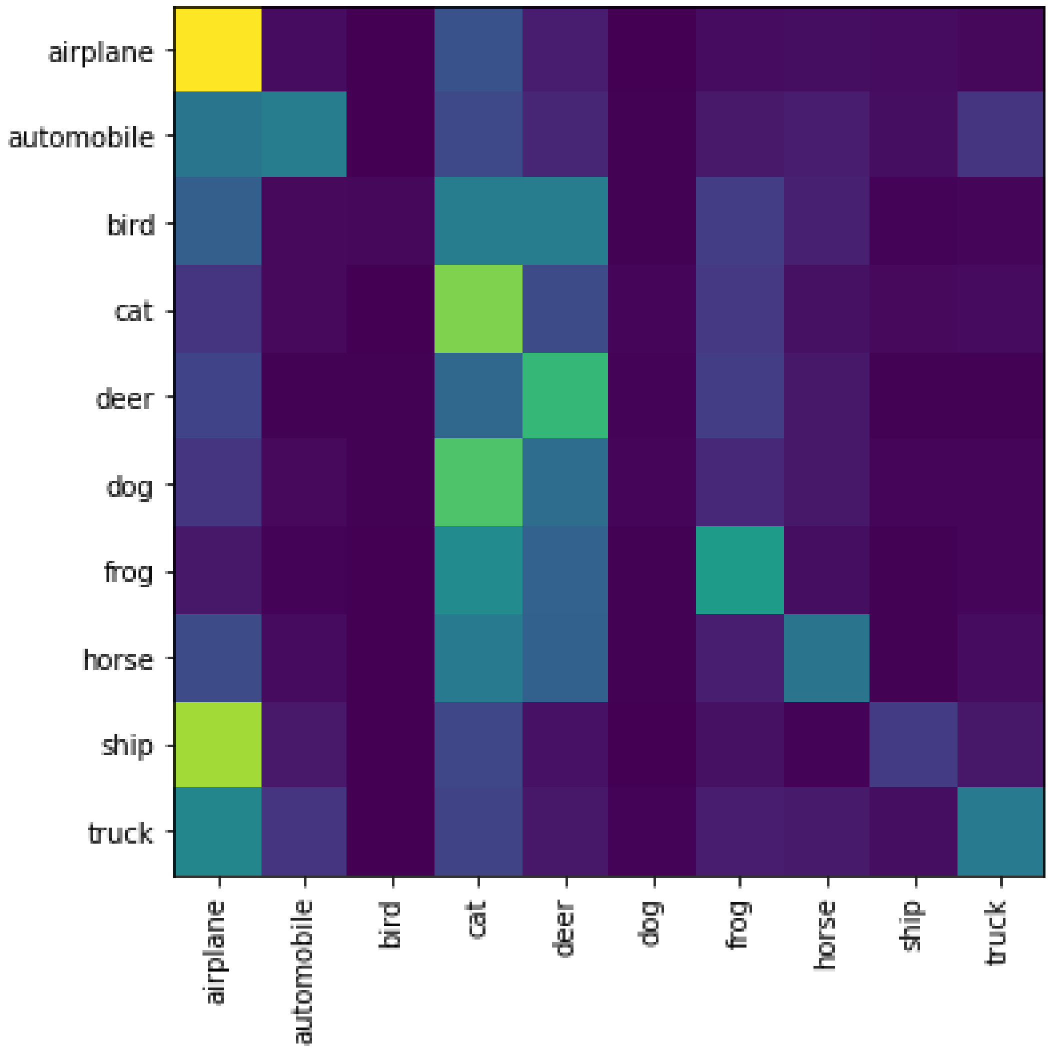
[40]:
```
# base class for your Tensorflow networks. It implements the training loop␣
↪(train) and prediction(__call__)  for you.
```

```
# You will need to implement the __init__ function to define the networks␣
↪structures in the following problems
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

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(labels=self.y_, logits=self.y))
        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, shuffle=True)):
                _, acc = self.sess.run([self.train_step, self.accuracy],␣
↪feed_dict={self.x: data, self.y_: label})
                #if i%100==99:
                #    print ('%d/%d %d %f'%(epoch, epochs, i, acc))

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

    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.compat.v1.random.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
```

```python
class LinearClassifer(TFClassifier):
    def __init__(self, classes=10):
        self.sess = tf.compat.v1.Session()

        self.x = tf.compat.v1.placeholder(tf.float32, shape=[None,32,32,3]) #␣
 ↪input batch of images
        self.y_ = tf.compat.v1.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

tf.compat.v1.disable_eager_execution()

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

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

```
WARNING:tensorflow:From C:\Users\Xirui Li\anaconda3\lib\site-
packages\tensorflow\python\compat\v2_compat.py:96: disable_resource_variables
(from tensorflow.python.ops.variable_scope) is deprecated and will be removed in
a future version.
Instructions for updating:
non-resource variables are not supported in the long term
testing epoch:1 accuracy: 25.390000
testing epoch:2 accuracy: 25.960000
testing epoch:3 accuracy: 25.530000
testing epoch:4 accuracy: 29.350000
testing epoch:5 accuracy: 25.720000
testing epoch:6 accuracy: 25.650000
testing epoch:7 accuracy: 27.290000
testing epoch:8 accuracy: 24.210000
testing epoch:9 accuracy: 27.050000
testing epoch:10 accuracy: 26.390000
testing epoch:11 accuracy: 24.840000
testing epoch:12 accuracy: 27.120000
testing epoch:13 accuracy: 27.580000
testing epoch:14 accuracy: 26.330000
```

```
testing epoch:15 accuracy: 25.880000
testing epoch:16 accuracy: 28.570000
testing epoch:17 accuracy: 23.240000
testing epoch:18 accuracy: 30.270000
testing epoch:19 accuracy: 27.280000
testing epoch:20 accuracy: 29.870000
```



1.8 Problem 6: Multi Layer Perceptron (MLP)

Here you will implement an MLP. The MLP shoud consist of 3 linear layers (matrix multiplcation 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.

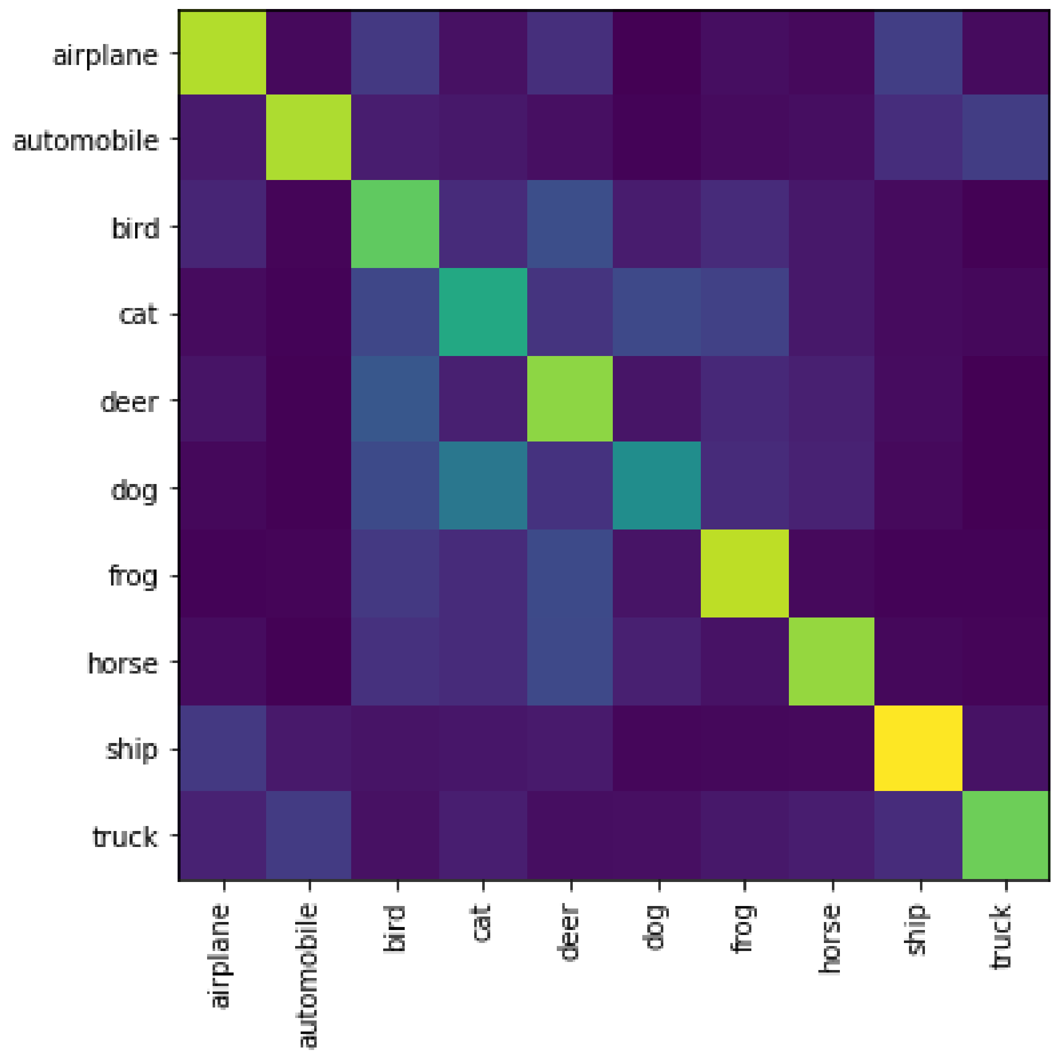
Hint: see the example linear classifier

Note: you should get around 50% accuracy

[41]:
```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

class MLPClassifer(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"""
        self.W1 = weight_variable([32*32*3,hidden*10])
        self.b1 = bias_variable([hidden*10])

        self.W2 = weight_variable([hidden*10,hidden])
        self.b2 = bias_variable([hidden])

        self.W3 = weight_variable([hidden,classes])
        self.b3 = bias_variable([classes])

        self.hidden1 = tf.nn.relu(tf.matmul(tf.reshape(self.x,(-1,32*32*3)),␣
 ↪self.W1) + self.b1)
        self.hidden2 = tf.nn.relu(tf.matmul(self.hidden1, self.W2) + self.b2)

        self.y = tf.matmul(self.hidden2, self.W3) + self.b3

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

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

```
testing epoch:1 accuracy: 39.270000
testing epoch:2 accuracy: 42.160000
```

```
testing epoch:3 accuracy: 43.770000
testing epoch:4 accuracy: 46.770000
testing epoch:5 accuracy: 48.170000
testing epoch:6 accuracy: 48.350000
testing epoch:7 accuracy: 48.450000
testing epoch:8 accuracy: 49.020000
testing epoch:9 accuracy: 49.690000
testing epoch:10 accuracy: 49.130000
testing epoch:11 accuracy: 50.050000
testing epoch:12 accuracy: 50.440000
testing epoch:13 accuracy: 50.880000
testing epoch:14 accuracy: 52.330000
testing epoch:15 accuracy: 51.510000
testing epoch:16 accuracy: 51.110000
testing epoch:17 accuracy: 50.770000
testing epoch:18 accuracy: 50.850000
testing epoch:19 accuracy: 51.730000
testing epoch:20 accuracy: 51.140000
```



1.9 Problem 7: Convolutional Neural Netork (CNN)

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))

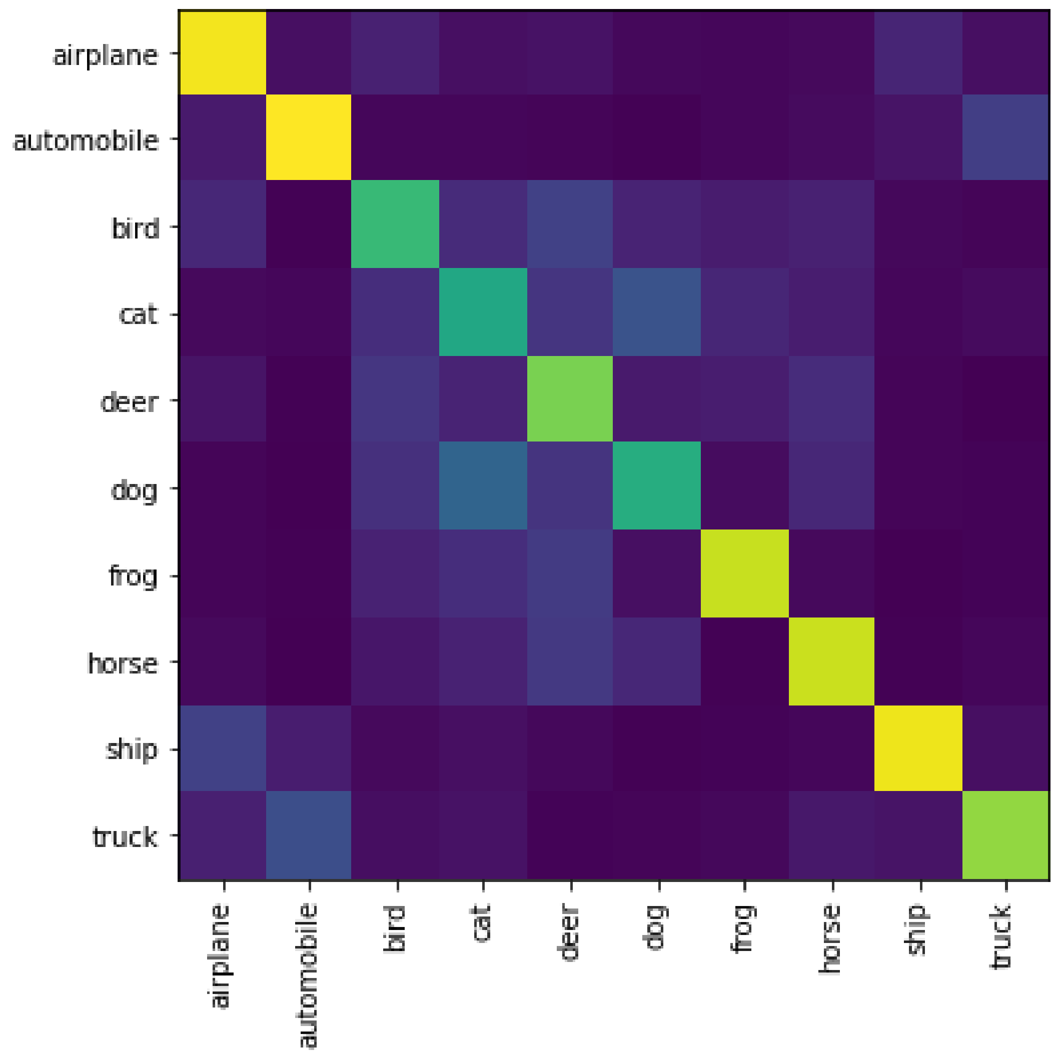
Linear(output_features=classes)

[44]:
```
def conv2d(x, W, stride=2):
    return tf.nn.conv2d(x, W, strides=[1, stride, stride, 1], padding='SAME')

class CNNClassifer(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 images
        self.y_ = tf.placeholder(tf.int64, shape=[None]) # input labels

        conv_layer_1_w = tf.Variable(tf.truncated_normal([4,4,3,n],stddev=0.
↪05,dtype=tf.float32))
        conv_layer_1_b = tf.Variable(tf.truncated_normal([n],stddev=0.
↪05,dtype=tf.float32))

        conv_layer_2_w = tf.Variable(tf.truncated_normal([4,4,n,n*2],stddev=0.
↪05,dtype=tf.float32))
        conv_layer_2_b = tf.Variable(tf.truncated_normal([n*2],stddev=0.
↪05,dtype=tf.float32))

        conv_layer_3_w = tf.Variable(tf.truncated_normal([4,4,n*2,n*4],stddev=0.
↪05,dtype=tf.float32))
        conv_layer_3_b = tf.Variable(tf.truncated_normal([n*4],stddev=0.
↪05,dtype=tf.float32))

        stride = 2
        conv_1 = conv2d(self.x, conv_layer_1_w)
        relu_1 = tf.nn.relu(conv_1 + conv_layer_1_b)
        max_pool_1 = tf.nn.max_pool(relu_1, ksize=[1,stride,stride,1],␣
↪strides=[1,stride,stride,1], padding='SAME')

        conv_2 = conv2d(max_pool_1, conv_layer_2_w)
        relu_2 = tf.nn.relu(conv_2 + conv_layer_2_b)
        max_pool_2 = tf.nn.max_pool(relu_2, ksize=[1,stride,stride,1],␣
↪strides=[1,stride,stride,1], padding='SAME')

        conv_3 = conv2d(max_pool_2,conv_layer_3_w)
        relu_3 = tf.nn.relu(conv_3 + conv_layer_3_b)

        self.W = weight_variable([n*4,classes])
        self.b = bias_variable([classes])
        self.y = tf.matmul(tf.reshape(relu_3,(-1,n*4)),self.W) + self.b


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

# display confusion matrix
```

```
M = confusion(testData, testLabels, cnnClassifer)
VisualizeConfussion(M)
```

```
testing epoch:1 accuracy: 41.220000
testing epoch:2 accuracy: 44.380000
testing epoch:3 accuracy: 48.700000
testing epoch:4 accuracy: 49.520000
testing epoch:5 accuracy: 52.010000
testing epoch:6 accuracy: 52.750000
testing epoch:7 accuracy: 54.140000
testing epoch:8 accuracy: 54.560000
testing epoch:9 accuracy: 55.590000
testing epoch:10 accuracy: 54.670000
testing epoch:11 accuracy: 55.940000
testing epoch:12 accuracy: 56.740000
testing epoch:13 accuracy: 56.280000
testing epoch:14 accuracy: 57.350000
testing epoch:15 accuracy: 58.040000
testing epoch:16 accuracy: 58.000000
testing epoch:17 accuracy: 58.050000
testing epoch:18 accuracy: 57.680000
testing epoch:19 accuracy: 58.530000
testing epoch:20 accuracy: 58.400000
```



1.10 Conclusion

Have you accomplished all parts of your assignment? What concepts did you used or learned in this assignment? What difficulties have you encountered? Explain your result for each section. Please wirte one or two short paragraph in the below Markdown window (double click to edit).

**** Your Conclusion: ****

–

1.11 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