Making Sense of Images

By

Srikar Devarakonda(sd1454)

For

Big Data & Visual Analytics

(CSE-8990-01)

Due date: 30th March 2016

Assignment-2

Submitted to the Faculty of

Mississippi State University

in Partial Fulfillment of the Requirements

for the Degree of Master of Science

in Computer Science

in the Department of Computer Science and Engineering

Mississippi State, Mississippi

March 2016

Making Sense of Images

**Data acquisition**. Read the data and the labels into the computer.

The CIFAR-10 python version dataset is downloaded from the university of Toronto website.The dataset chose for this assignment is CIFAR-10 labeled dataset which consists of 60000 images divided into 10 classes each containing 6000 each. The data is stored as 5 training batches containing 50000 images and 1 test batch containing 10000 images chosen randomly from each class.

The archive contains the files data\_batch\_1, data\_batch\_2, ..., data\_batch\_5, as well as test\_batch. Each of these files is a Python "pickled" object produced with [cPickle](http://www.python.org/doc/2.5/lib/module-cPickle.html). Here is a Python routine which will open such a file and return a dictionary:

Loaded in this way, each of the batch files contains a dictionary with the following elements:

1. **data** -- a 10000x3072 [numpy](http://numpy.scipy.org/) array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
2. **labels** -- a list of 10000 numbers in the range 0-9. The number at index *i* indicates the label of the *i*th image in the array **data**.

The dataset contains another file, called batches.meta. It too contains a Python dictionary object. It has the following entries:

**label\_names** -- a 10-element list which gives meaningful names to the numeric labels in the **labels** array described above. For example, label\_names[0] == "airplane", label\_names[1] == "automobile", etc.

The tools used for this assignment is Python along with its inbuilt packages.

The data is unpickled into disctionary using the following code :

def unpickle(file):

import cPickle

fo = open(file, 'rb')

dict = cPickle.load(fo)

fo.close()

return dict

my\_data = {'data': unpickle("data\_batch\_1")['data'], 'labels': unpickle("data\_batch\_1")['labels']}

for each in range(2, 6):

my\_data['data'] = np.concatenate((my\_data['data'], unpickle("data\_batch\_" + str(each))['data']), axis=0)

my\_data['labels'] = np.concatenate((my\_data['labels'], unpickle("data\_batch\_" + str(each))['labels']), axis=0)

data = my\_data['data']

labels = my\_data['labels']

label\_names = unpickle("batches.meta")['label\_names']

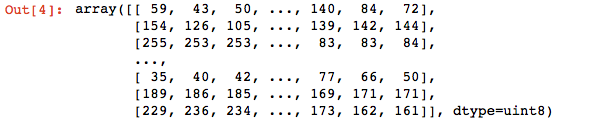


Fig1: The image data after unpickling it.

my\_data is the name of the dictionary that contains entire CIFAR-10 data. Data contains the image data. labels contain the labels of the images. label\_names contains the class names of the images.

**Feature extraction**. A feature summarizes a certain aspect of an image. Each images can then be represented by its feature vector.

the entire cifar-10 dataset can be taken as a features vectors as it contains RGB values of each image. The features chosen for the CIFAR-10 dataset are the average color of all pixels, the maximum and minimum intensity of all pixels, segmenting each images into 4-parts and finding the average color of all pixels, the maximum and minimum intensity of all pixels. Applied prewritten canny edge detection for each image which identifies the edges in the images and gives the key-point which represents edges in the image and also applied ORB(Oriented FAST and Rotated BRIEF) algorithm which gives the key-points in the image based on the intensity weighted centroid of the patch with located corner at center.

for i in range(0,50000):

imgdt= data[i]

lind= labels[i]

average\_rgb = []

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(0, 1024):

redcount += imgdt[x]

for x in range(1024, 2048):

greencount += imgdt[x]

for x in range(2048, 3072):

bluecount += imgdt[x]

average\_rgb.append((redcount/3072, greencount/3072, bluecount/3072))

#print redcount/3072, greencount/3072, bluecount/3072,label\_names[lind]

X\_train.append([redcount/3072, greencount/3072, bluecount/3072,max(imgdt[:1024]),min(imgdt[:1024]),max(imgdt[1024:2048]),min(imgdt[1024:2048]),max(imgdt[2048:3072]),min(imgdt[2048:3072])])

#y\_train.append(label\_names[lind])

#print X\_train, y\_train

p1dr = data[each][0:256]

p1dg = data[each][1024:1280]

p1db = data[each][2048:2304]

p2dr = data[each][256:512]

p2dg = data[each][1280:1536]

p2db = data[each][2304:2560]

p3dr = data[each][512:768]

p3dg = data[each][1536:1792]

p3db = data[each][2560:2816]

p4dr = data[each][768:1024]

p4dg = data[each][1972:2048]

p4db = data[each][2816:3072]

feat = []

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(0, 256):

redcount += data[each][x]

for x in range(1024, 1280):

greencount += data[each][x]

for x in range(2048, 2304):

bluecount += data[each][x]

feat.append(redcount/768)

feat.append(greencount/768)

feat.append(bluecount/768)

feat.append(np.amax(p1dr))

feat.append(np.amax(p1dg))

feat.append(np.amax(p1db))

feat.append(np.amin(p1dr))

feat.append(np.amin(p1dg))

feat.append(np.amin(p1db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(256,512):

redcount += data[each][x]

for x in range(1280,1536):

greencount += data[each][x]

for x in range(2304,2560):

bluecount += data[each][x]

feat.append(redcount/768)

feat.append(greencount/768)

feat.append(bluecount/768)

feat.append(np.amax(p2dr))

feat.append(np.amax(p2dg))

feat.append(np.amax(p2db))

feat.append(np.amin(p2dr))

feat.append(np.amin(p2dg))

feat.append(np.amin(p2db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(512,768):

redcount += data[each][x]

for x in range(1536,1792):

greencount += data[each][x]

for x in range(2560,2816):

bluecount += data[each][x]

feat.append(redcount/768)

feat.append(greencount/768)

feat.append(bluecount/768)

feat.append(np.amax(p3dr))

feat.append(np.amax(p3dg))

feat.append(np.amax(p3db))

feat.append(np.amin(p3dr))

feat.append(np.amin(p3dg))

feat.append(np.amin(p3db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(768,1024):

redcount += data[each][x]

for x in range(1792,2048):

greencount += data[each][x]

for x in range(2816,3072):

bluecount += data[each][x]

feat.append(redcount/768)

feat.append(greencount/768)

feat.append(bluecount/768)

feat.append(np.amax(p4dr))

feat.append(np.amax(p4dg))

feat.append(np.amax(p4db))

feat.append(np.amin(p4dr))

feat.append(np.amin(p4dg))

feat.append(np.amin(p4db))

X\_train.append(feat)

edges ={}

edges = cv2.Canny(data,100,200)

edges

# Initiate STAR detector

orb = cv2.ORB()

# find the keypoints with ORB

kp = orb.detect(data,None)

# compute the descriptors with ORB

kp, des = orb.compute(data, kp)

kp, des

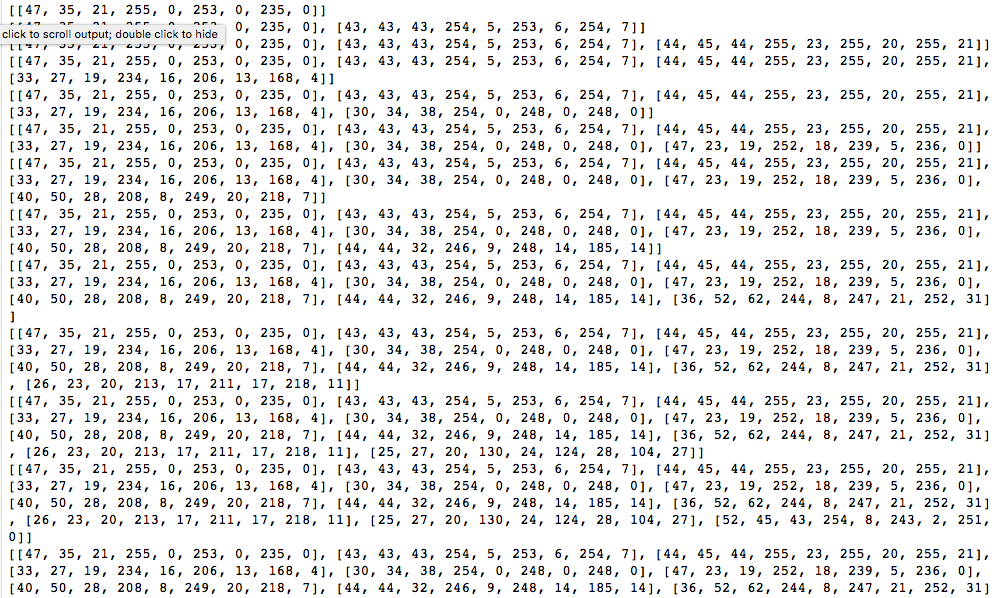


Fig 2: Average Intensities of Pixels Along with Maximum and Minimum Intensities (Training)

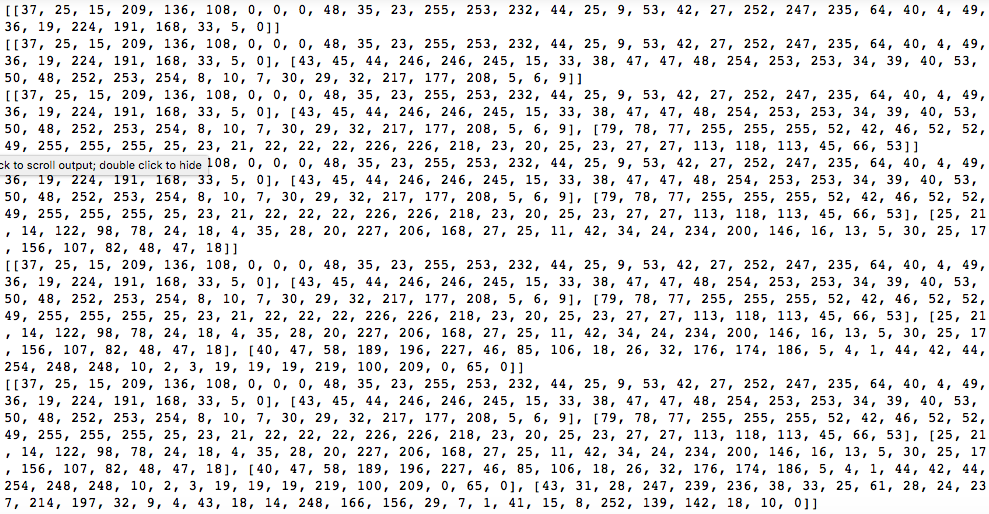


Fig3: Feature Vectors When image is segmented into 4 parts.

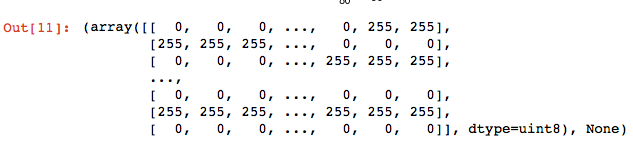


Fig4: Feature Vectors for canny edge detection

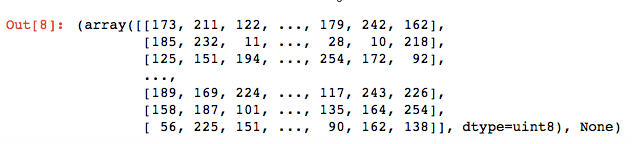


Fig 5: Feature Vectors for ORB

After calculating all the features all the features vectors are stored as list of lists in X\_train list. The above procedure is for the training part of the dataset. The same process of feature extraction is repeated for the test set.

for i in range(0,10000):

imgdt= test\_data[i]

lind= test\_labels[i]

average\_rgb = []

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(0, 1024):

redcount += imgdt[x]

for x in range(1024, 2048):

greencount += imgdt[x]

for x in range(2048, 3072):

bluecount += imgdt[x]

average\_rgb.append((redcount/3072, greencount/3072, bluecount/3072))

#print redcount/3072, greencount/3072, bluecount/3072,label\_names[lind]

X\_test.append([redcount/3072, greencount/3072, bluecount/3072,max(imgdt[:1024]),min(imgdt[:1024]),max(imgdt[1024:2048]),min(imgdt[1024:2048]),max(imgdt[2048:3072]),min(imgdt[2048:3072])])

#y\_test.append(label\_names[lind])

#print X\_test,y\_test

p1dr = test\_data[each][0:256]

p1dg = test\_data[each][1024:1280]

p1db = test\_data[each][2048:2304]

p2dr = test\_data[each][256:512]

p2dg = test\_data[each][1280:1536]

p2db = test\_data[each][2304:2560]

p3dr = test\_data[each][512:768]

p3dg = test\_data[each][1536:1792]

p3db = test\_data[each][2560:2816]

p4dr = test\_data[each][768:1024]

p4dg = test\_data[each][1972:2048]

p4db = test\_data[each][2816:3072]

feat1 = []

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(0, 256):

redcount += test\_data[each][x]

for x in range(1024, 1280):

greencount += test\_data[each][x]

for x in range(2048, 2304):

bluecount += test\_data[each][x]

feat1.append(redcount/768)

feat1.append(greencount/768)

feat1.append(bluecount/768)

feat1.append(np.amax(p1dr))

feat1.append(np.amax(p1dg))

feat1.append(np.amax(p1db))

feat1.append(np.amin(p1dr))

feat1.append(np.amin(p1dg))

feat1.append(np.amin(p1db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(256,512):

redcount += test\_data[each][x]

for x in range(1280,1536):

greencount += test\_data[each][x]

for x in range(2304,2560):

bluecount += test\_data[each][x]

feat1.append(redcount/768)

feat1.append(greencount/768)

feat1.append(bluecount/768)

feat1.append(np.amax(p2dr))

feat1.append(np.amax(p2dg))

feat1.append(np.amax(p2db))

feat1.append(np.amin(p2dr))

feat1.append(np.amin(p2dg))

feat1.append(np.amin(p2db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(512,768):

redcount += test\_data[each][x]

for x in range(1536,1792):

greencount += test\_data[each][x]

for x in range(2560,2816):

bluecount += test\_data[each][x]

feat1.append(redcount/768)

feat1.append(greencount/768)

feat1.append(bluecount/768)

feat1.append(np.amax(p3dr))

feat1.append(np.amax(p3dg))

feat1.append(np.amax(p3db))

feat1.append(np.amin(p3dr))

feat1.append(np.amin(p3dg))

feat1.append(np.amin(p3db))

#count number of rgb pixels and find average

redcount = 0;

greencount = 0;

bluecount = 0;

for x in range(768,1024):

redcount += test\_data[each][x]

for x in range(1792,2048):

greencount += test\_data[each][x]

for x in range(2816,3072):

bluecount += test\_data[each][x]

feat1.append(redcount/768)

feat1.append(greencount/768)

feat1.append(bluecount/768)

feat1.append(np.amax(p4dr))

feat1.append(np.amax(p4dg))

feat1.append(np.amax(p4db))

feat1.append(np.amin(p4dr))

feat1.append(np.amin(p4dg))

feat1.append(np.amin(p4db))

X\_test.append(feat1)

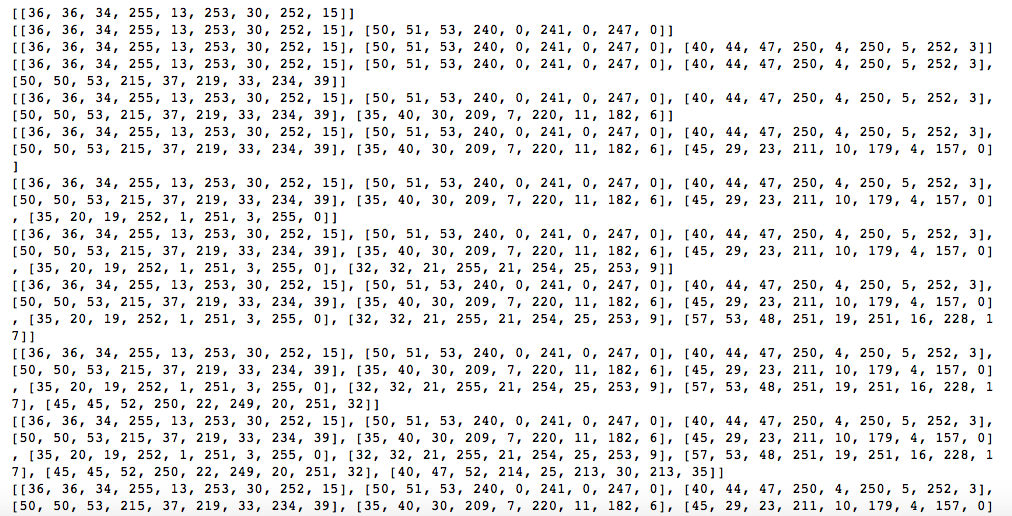


Fig 6: Average Intensities of Pixels Along with Maximum and Minimum Intensities(TEST)

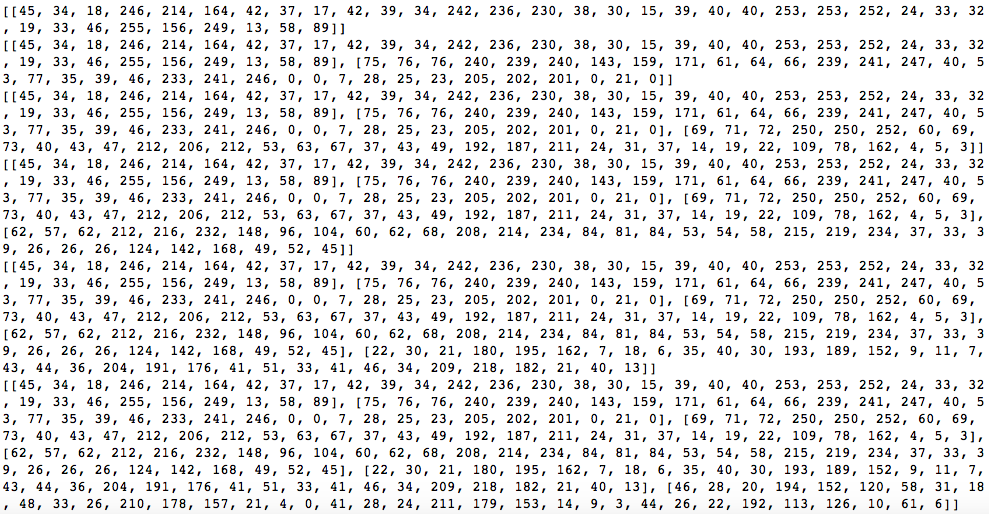


Fig 7: Feature Vectors When image is segmented into 4 parts.

The test dataset features are combined and are stored in X\_test.

**Principal components.** Compute the principal components of the images. Explain your findings. You may use what you find to construct better feature vectors.

If you take the entire data as feature vectors then it will give many almost 3072 feature points which are too many. The entire data is plotted using the following code:

x1 = []

y1 = []

z1 = []

for item in abc:

x1.append(item[0])

y1.append(item[1])

z1.append(item[2])

fig1 = plt.figure()

ax = Axes3D(fig1)

pltData = [x1,y1,z1]

ax.scatter(pltData[0], pltData[1], pltData[2], 'bo')

xLine = ((min(pltData[0]), max(pltData[0])), (0, 0), (0,0))

ax.plot(xLine[0], xLine[1], xLine[2], 'r')

yLine = ((0, 0), (min(pltData[1]), max(pltData[1])), (0,0))

ax.plot(yLine[0], yLine[1], yLine[2], 'r')

zLine = ((0, 0), (0,0), (min(pltData[2]), max(pltData[2])))

ax.plot(zLine[0], zLine[1], zLine[2], 'r')

ax.set\_xlabel("x-axis")

ax.set\_ylabel("y-axis")

ax.set\_zlabel("Z-axis")

ax.set\_title("CIFAR-10 PCA")

plt.show()

X = np.array(X\_train)

pca = PCA(n\_components=36)

abc = pca.fit\_transform(X)

pca1 = pca.score(X)

abc,pca1

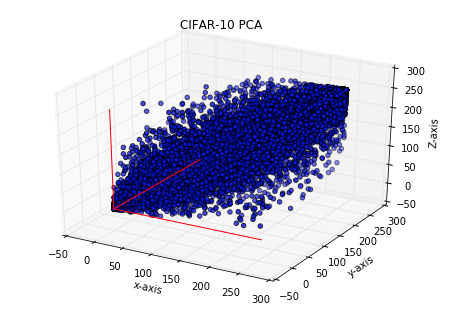


Fig 8: Image Data

The principle components of the images are calculated using the above code by directly implementing the PCA package from sklearn.

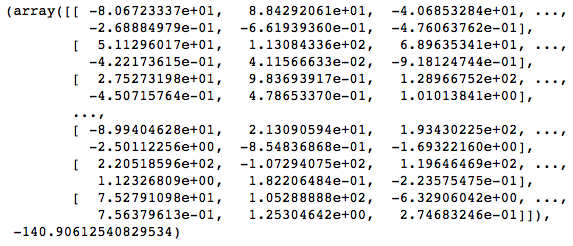


Fig 9: Principal Components of Images

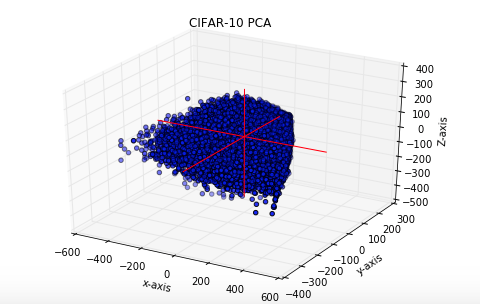


Fig 10: PCA of Feature Vectors

After calculating the PCA for feature vectors the number of feature points are reduced.

**Statistical analysis of the image feature vectors**. Compute mean, deviation, and other statistics for each class of images. Form hypotheses and test them.  For example, you can hypothesize that the feature vectors of the cat images are significantly different than the car images. Test the hypotheses with Student’s T test or other tests.

Images under each class are identified with the help of label names and created a dictionary for each type of class. The following function returns the mean, variance, and standard deviation for each class of image under training set.

def get\_stats(class\_name):

mean = 0

variance = 0

std\_deviation = 0

size = len(class\_data[class\_name])

for each in class\_data[class\_name]:

mean += np.mean(each)

variance += np.var(each)

std\_deviation += np.std(each)

mean = mean / size

variance = variance / size

std\_deviation = std\_deviation / size

return mean, variance, std\_deviation, size

get\_stats('airplane'), get\_stats('automobile'), get\_stats('bird'),get\_stats('cat'), get\_stats('deer'), get\_stats('dog'), get\_stats('frog'), get\_stats('horse'),get\_stats('ship'),get\_stats('truck')

We have 3 feature vectors, so the mean for all the feature vectors are as follows:

Case 1: When the dataset as whole is considered as feature vectors:

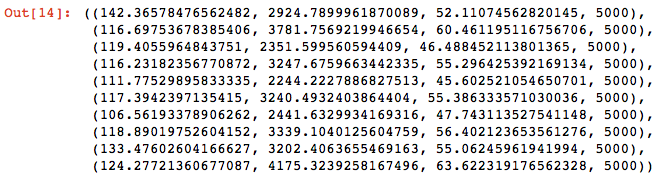


Fig 11: Mean, Variance, Standard Deviation and the Number of Images in Each Class.

Case 2: When the Average Intensities of Pixels Along with Maximum and Minimum Intensities for an image is calculated.

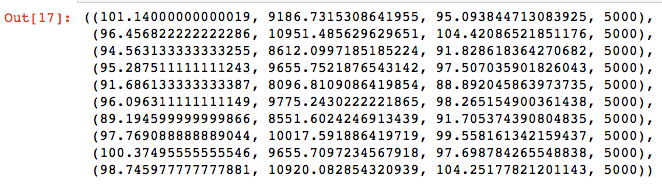


Fig 12: Mean, Variance, Standard Deviation and the Number of Images in Each Class.

Case 3: When images is segmented into 4 parts:

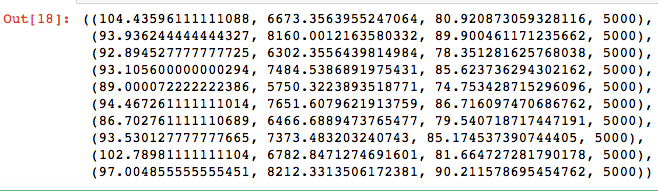
****

Fig 13: Mean, Variance, Standard Deviation and the Number of Images in Each Class.

Hypothesis Testing:

Welch’s t-test was used for the hypothesis testing which involves calculation of t value for two independent samples with unequal variances.

What percentage of features of deer and airplane are similar?

Null hypothesis: feature vectors of deer and airplane are similar

Ho : U1= U2

Research hypothesis: feature vectors of deer and airplane are different.

H1 : U1 ~~=/~~ U2

def tstat(class\_name1, class\_name2):

X1 = get\_stats(class\_name1)[0]

X2 = get\_stats(class\_name2)[0]

S1 = get\_stats(class\_name1)[1]

S2 = get\_stats(class\_name2)[1]

N1=5000

N2=5000

t1= X1-X2

t2= sqrt((S1/N1+S2/N2))

t11=t1/t2

S1 = get\_stats(class\_name1)[1]

S2 = get\_stats(class\_name2)[1]

sum1 =((get\_stats(class\_name1)[1]+get\_stats(class\_name2)[1]))\*((get\_stats(class\_name1)[1]+get\_stats(class\_name2)[1]))

s14 =(get\_stats(class\_name1)[1]\*get\_stats(class\_name1)[1])

s24=(get\_stats(class\_name2)[1]\*get\_stats(class\_name2)[1])

n1 = 5000\*5000

v1= 4999

n2 = 5000\*5000

v2=4999

sum2= (s14/n1\*v1 + s24/n2\*v2)

df1 = (sum1/sum2)

df= df1/10000

p\_value = 2\*st.t.cdf(-np.abs(t11),df)

return t11, df, p\_value

if p\_value<= 0.5:

print(False)

else:

print (True)

tstat('airplane','deer')

The above code gives the t value, degree of freedom and p-value required to reject or accept the hypothesis.

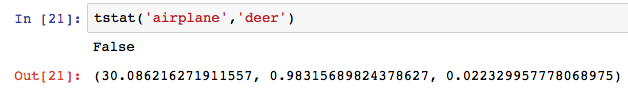
****

Fig 14: Case1, Raw dataset.

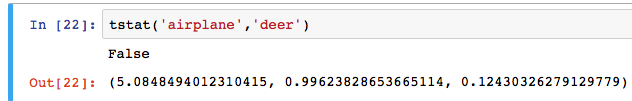
****

Fig 15: Case 2, Average Intensities of Pixels Along with Maximum and Minimum Intensities

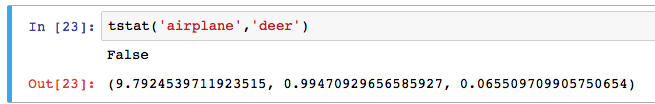


Fig 16: Case 3, Feature Vectors When Image is Segmented into 4 parts

In our case, the t-values are 30.0, 5.08, 9.97, degree of freedom are .983, 0.996, 0.9947 and p-values are 0.0223, 0.124, 0.0655. We consider our significance level to be 0.5. Since p-value turns out to be less than the .05 significance level in all the cases, returns false and wereject the null hypothesis which means that feature vectors of deer and airplane are not similar.

**Cluster the images based on feature vectors**. Use k-means, hierarchical clustering, etc. to cluster the images into groups. Compare your results with the labels of the images and see if clustering with your feature vectors can put the images into the right classes.

Used K-means clustering algorithm to cluster the images into 10 clusters as there are different classes of images.

K-means was imported from sk-learn package.

k\_means = KMeans(n\_clusters=10, n\_jobs=8)

k\_means.fit(X\_train)

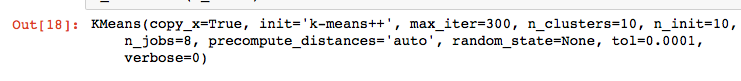


Fig 17: Output of K-means Clustering

Net we used predict function to get the images from a particular class.

k\_means.predict(class\_data\_test['airplane'])

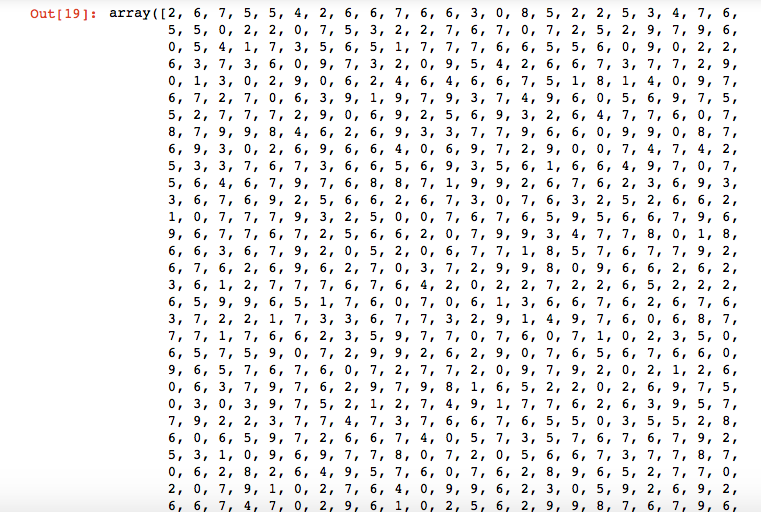


Fig 18: Output of predict function.

def get\_count(class\_name):

return Counter(k\_means.predict(class\_data\_test[class\_name]))

get\_count('truck')

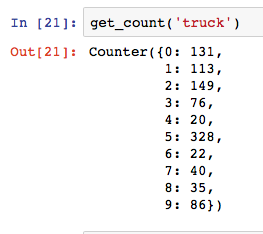


Fig 19: Cluster and number of images

This predict function gives the number of images of class truck in each cluster. The prediction says that’s the truck belongs to cluster 6 it has more number of images of truck class.

Now we wrote probability function to predict whether the given

from collections import Counter

def get\_prob(class\_name):

c = Counter(k\_means.predict(class\_data\_test['truck']))

clusternumber = c.most\_common(1)[0][0]

val = (c.most\_common(1)[0][1])

return clusternumber, val

get\_prob('truck')

Case 1: Raw Dataset is used as feature vector.

Screen%20Shot%202016-03-27%20at%202.55.01%20PM.png

Fig 20: Cluster number and Probability

The above function gives the probability of the truck class is in cluster 5 is 32.8 % and it shows that the k-means prediction function also says that cluster 5 is truck class as it has 328 truck class images.

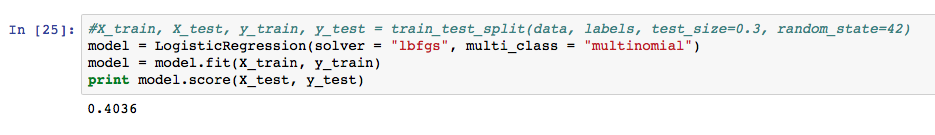
Next we used multinomial logistic regression to compares the training sets and test set for finding out the accuracy of the classification of images into clusters by k-means.

Multinomial Logistic Regression is the linear regression analysis to conduct when the dependent variable is nominal with more than two levels. Thus it is an extension of logistic regression, which analyzes dichotomous (binary) dependents.

model = LogisticRegression(solver = "lbfgs", multi\_class = "multinomial")

model = model.fit(X\_train, y\_train)

print model.score(X\_test, y\_test)



The above function gives the probability of the truck class is in cluster 7 is 34.4% and it shows that the k-means prediction function also says that cluster 7 is truck class as it has 346 truck class images.

Multinomial Logistic Regression accuracy is 26.67%



Fig 22: Case 2, Average Intensities of Pixels Along with Maximum and Minimum Intensities

The below function gives the probability of the truck class is in cluster 3 is 27% and it shows that the k-means prediction function also says that cluster 3 is truck class as it has 270 truck class images.

Multinomial Logistic Regression accuracy is 38.07%

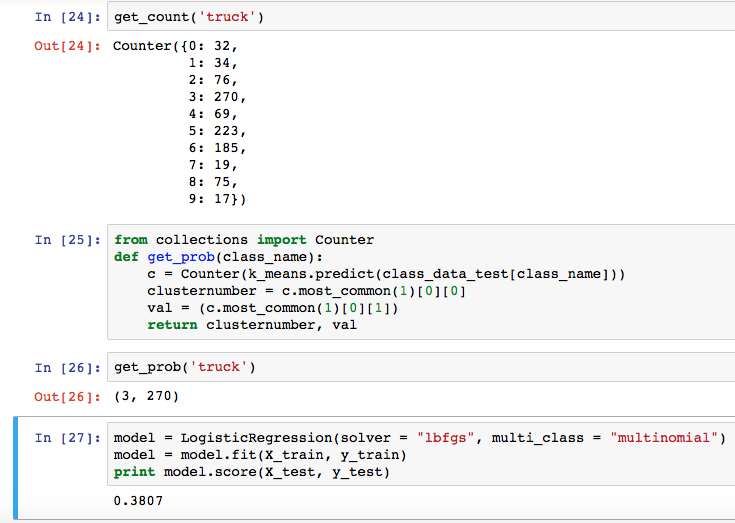


Fig 23: Case 3, Feature Vectors When Image is Segmented into 4 parts.