Test 2 Take Home

To solve this problem, I decided to divide my solution into three phases, Segment the image, get features, and classify the image. To test the image, I created a test file in main

**Segmentation:**

To segment different object in the image I decided to use 9x9 Morphological Gradient on the grayscale image, then I took Threshold of the gradient, then I applied dilation to the threshold image, then I found contours in the image. I created a function to get segments of the image.

**Get Features:**

I decided to use object area, contour boundary shape, number of corners, corner/area, and circularity. I created a separate function for each and get feature to get all the features as array

**Classify the Image:**

I created a csv file with features I listed above for different images, then I manually filled the classified column for predetermined data with Nut, Bolt, and Washer, then I divided the data into training and test using StratifiedShuffleSplit, then I created train model using RandomForestClassifier, then I improved the tune the model using gridSearchCV. Then I saved the model.

**Main:**

Segment the image located at the image location specified by user, predict using the model saved, and increment the count of prediction detected.

**Reference:**

https://www.pyimagesearch.com/2014/11/03/display-matplotlib-rgb-image/

http://www.mathworks.com/matlabcentral/fileexchange/25157-image-segmentation-tutorial

http://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_feature2d/py\_features\_harris/py\_features\_harris.html

https://stackoverflow.com/questions/35472712/how-to-split-data-on-balanced-training-set-and-test-set-on-sklearn

http://scikit-learn.org/stable/modules/generated/sklearn.model\_selection.cross\_val\_score.html

https://docs.opencv.org/3.3.1/d3/dc0/group\_\_imgproc\_\_shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

**Segmentation:**

**import** cv2  
**def** segmentation(img):  
 *#location* imageLoc = img  
  
 *#read image* img = cv2.imread(imageLoc , 1)  
  
 *#OpenCV represents RGB images as multi-dimensional NumPy arrays…but in reverse order!  
 #This means that images are actually represented in BGR order rather than RGB!  
  
 #Plotting numpy arrays as images  
 #pyplot.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))  
  
 #define kernel for morphological* kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(9,9))  
  
 *#grayscale* gray = cv2.imread(imageLoc,0)  
  
 *#morphological gradient* gradient = cv2.morphologyEx(gray, cv2.MORPH\_GRADIENT, kernel)  
  
 *#pyplot.imshow(gradient,cmap='gray')  
 #plt.show()  
 #get the threshold of morphological gradient* retval,out = cv2.threshold(gradient,50,200,cv2.THRESH\_BINARY)  
  
 *#apply dilation to output so broken pixels are fixed* out = cv2.dilate(out,kernel,iterations=1)  
  
 *#pyplot.imshow(out,cmap='gray')  
 #pyplot.show()  
  
 #  
 #find obj* rects, contours, h = cv2.findContours(out, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 *#duplicate to test detection* imgCopy=img  
  
 *#define min and max area for objects* min\_a = .001  
 max\_a = 1  
 *#* segm = []  
 *#draw rectangle and add object in segm array as individual obj  
 #print("contours detected:",len(contours))* obj = 0  
 **for** i **in** range(len(contours)):  
 x,y,w,h = cv2.boundingRect(contours[i])  
 a\_ratio = (w\*h) / (imgCopy.shape[0] \* imgCopy.shape[1])  
 **if** (a\_ratio >= min\_a **and** a\_ratio <= max\_a):  
 obj+=1  
 segm.append(tuple([x,y,w,h]))  
 imgCopy = cv2.rectangle(imgCopy,(x,y),(x+w,y+h),(0,255,0),10)  
  
 *#print("Objects detected:",obj)*

**return** segm,img,out

**Features**:

**import** cv2  
**import** numpy **as** np  
**def** get\_object\_area\_ratio(img):  
 box\_area=img.shape[0]\*img.shape[1]  
 object\_area=np.count\_nonzero(img)  
 ratio=object\_area/box\_area  
 **return** ratio  
**def** get\_boundary\_shape(img):  
 height=img.shape[0]  
 width=img.shape[1]  
 **if**(width<=height):  
 **return** width/height  
 **else**:  
 **return** height/width  
**def** get\_corner(img):  
 dst=cv2.cornerHarris(img,2,3,0.04)  
 corners = np.count\_nonzero(dst[dst > 0.01 \* dst.max()])  
 **return** corners,dst  
  
**def** get\_circularity\_error(img):  
 b, object\_conts, h = cv2.findContours(img, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 contour\_areas = [cv2.contourArea(object\_conts[i]) **for** i **in** range(len(object\_conts))]  
 max\_contour\_idx = np.argmax(contour\_areas)  
 area\_object = contour\_areas[max\_contour\_idx]  
 radius=0  
 **if** img.shape[0]>img.shape[1]:  
 radius=img.shape[0] / 2.0  
 **else**:  
 radius=img.shape[1] / 2.0  
 area\_circle = np.pi \* radius \*\* 2  
  
 circularity\_error = abs(area\_object - area\_circle) / ((2 \* radius) \*\* 2)  
 **return** circularity\_error  
  
**def** get\_features(img):  
 a\_ratio = get\_object\_area\_ratio(img)  
 box\_scale = get\_boundary\_shape(img)  
 total\_corners, corner\_img = get\_corner(img)  
 area=img.shape[0]\*img.shape[1]  
 cornerPerArea=total\_corners\*100/area  
 circularity=get\_circularity\_error(img)  
 **return** [a\_ratio, box\_scale, total\_corners,circularity,cornerPerArea]

**Build\_Model:**

**import** csv  
**import** glob  
**from** segmentation **import** segmentation  
**from** features **import** get\_features  
**import** cv2  
**import** pandas **as** pd  
**from** sklearn.model\_selection **import** StratifiedShuffleSplit  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.model\_selection **import** cross\_val\_score  
**from** sklearn.model\_selection **import** GridSearchCV  
**from** sklearn.metrics **import** mean\_squared\_error  
**import** numpy **as** np  
**import** pickle  
  
**def** save\_model(cls, filename):  
 pickle.dump(cls, open(filename, **'wb'**))  
  
  
**def** load\_model(filename):  
 **return** pickle.load(open(filename, **'rb'**))  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
  
 *#UNCOMMENT TO BUILD DATASET  
 # images = glob.glob("./img/\*.JPG")  
 #  
 # imgId=0  
 # with open('dataset/data.csv','w') as f:  
 # writer=csv.writer(f)  
 # writer.writerow(['id','width', 'height', 'area\_ratio', 'box\_shape', 'total\_corners', 'circularity','corner/area','classified'])  
 # for img in images:  
 # segm,input, out = segmentation(img)  
 #  
 # for segment in segm:  
 # x, y, w, h = segment  
 # seg\_img = out[y:y+h, x:x+w]  
 # features = get\_features(seg\_img)  
 # name='./dataset/img/'+str(imgId)+'.png'  
 # cv2.imwrite(name, input[y:y+h, x:x+w]) # write segmented image  
 # writer.writerow([imgId,seg\_img.shape[1],seg\_img.shape[0],features[0],features[1],features[2],features[3],features[4],'NONE'])  
 # imgId+=1* trained\_data = pd.read\_csv(**'./dataset/final\_data.csv'**)  
 *#print(trained\_data.head())* trained\_data[**'classified'**] = trained\_data[**'classified'**].str.replace(**'NUT'**, **'1'**)  
 trained\_data[**'classified'**] = trained\_data[**'classified'**].str.replace(**'BOLT'**, **'2'**)  
 trained\_data[**'classified'**] = trained\_data[**'classified'**].str.replace(**'WASHER'**, **'3'**)  
  
 *#dropped the column no longer needed, DO NOT WRITE* **for** col **in** [**'id'**,**'width'**,**'height'**]:  
 trained\_data = trained\_data.drop(col, axis=1)  
 trained\_data[**'classified'**] = trained\_data[**'classified'**].apply(pd.to\_numeric)  
 *#print(trained\_data)* split = StratifiedShuffleSplit(n\_splits=1, test\_size=0.2, random\_state=42)  
 **for** train\_index, test\_index **in** split.split(trained\_data, trained\_data[**'classified'**]):  
 strat\_train\_set = trained\_data.loc[train\_index]  
 strat\_test\_set = trained\_data.loc[test\_index]  
 train\_set\_y = strat\_train\_set[**"classified"**]  
 train\_set\_x = strat\_train\_set.drop(**'classified'**, axis=1)  
 *# print("-----------SET-------------")  
 # print(train\_set\_y.head())  
 # print(train\_set\_x.head())  
  
 #train model* forest\_clf = RandomForestClassifier(random\_state=42)  
 score = cross\_val\_score(forest\_clf, train\_set\_x, train\_set\_y, cv=3, scoring=**"accuracy"**)  
 print(score)  
 params = [  
 {**'n\_estimators'**: [5,10,30], **'max\_features'**: [2,4,5], **'min\_samples\_split'**:[0.1,0.5,1.0]}  
 ]  
 *#* forest\_clf = RandomForestClassifier(random\_state=42)  
 *#* grid\_search = GridSearchCV(forest\_clf, params, cv=2, scoring=**'accuracy'**)  
 *#* grid\_search.fit(train\_set\_x, train\_set\_y)  
 *#* grid\_search.best\_params\_  
  
 features = grid\_search.best\_estimator\_.feature\_importances\_  
 attibs = list(trained\_data)  
 print(**"-----------FEATURES UTILIZED-------------"**)  
 print(sorted(zip(features, attibs), reverse=**True**))  
  
  
 final\_model = grid\_search.best\_estimator\_  
  
 test\_set\_y = strat\_test\_set[**"classified"**]  
 test\_set\_X = strat\_test\_set.drop(**'classified'**, axis=1)  
  
 final\_predictions = final\_model.predict(test\_set\_X)  
  
 final\_mse = mean\_squared\_error(test\_set\_y, final\_predictions)  
 final\_rmse = np.sqrt(final\_mse)  
  
 print(**"FINAL RNSE:"**, final\_rmse)  
  
 save\_model(final\_model, **'./models/nut\_bolt\_test.sav'**)

**Main:**

**from** segmentation **import** segmentation  
**from** features **import** get\_features  
**from** build\_model **import** load\_model  
  
  
*#segments the image*img=**'./Parts2.jpg'**segm,image,out=segmentation(img)  
  
*#load model*model = load\_model(**'./models/nut\_bolt\_test.sav'**)  
nut=0  
bolt=0  
washer=0  
*#extract the feature***for** segment **in** segm:  
 x,y,w,h=segment  
 seg\_img = out[y:y + h, x:x + w]  
 features = get\_features(seg\_img)  
 print(features)  
 pred = model.predict([features])  
 **if** pred == 1:  
 nut+=1  
 **elif** pred==2:  
 bolt+=1  
 **else**:  
 washer+=1  
print(**"Nuts:"**+str(nut)+**" Bolts:"**+str(bolt)+**" Washer:"**+str(washer))