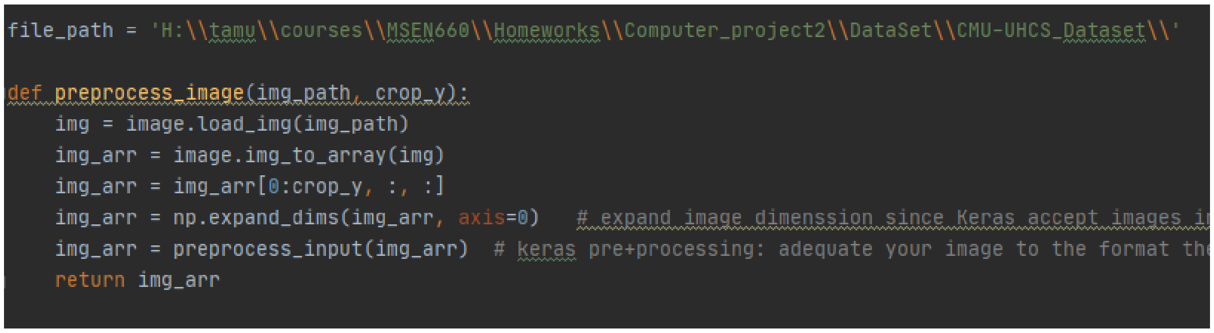
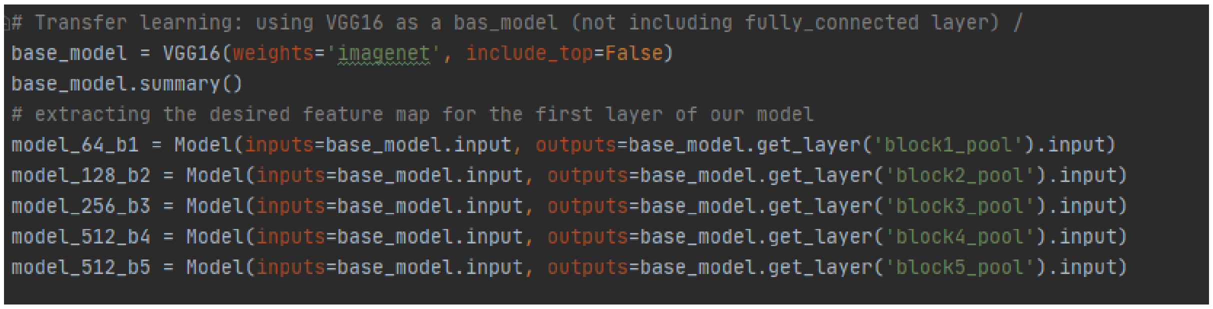
ComputerProject2: Classifying micrographs according to **Primary microconstituent**

Preprocessing: Using Pretrained model for transfer learning to get featurization; Reducing dimensionality by huge amount



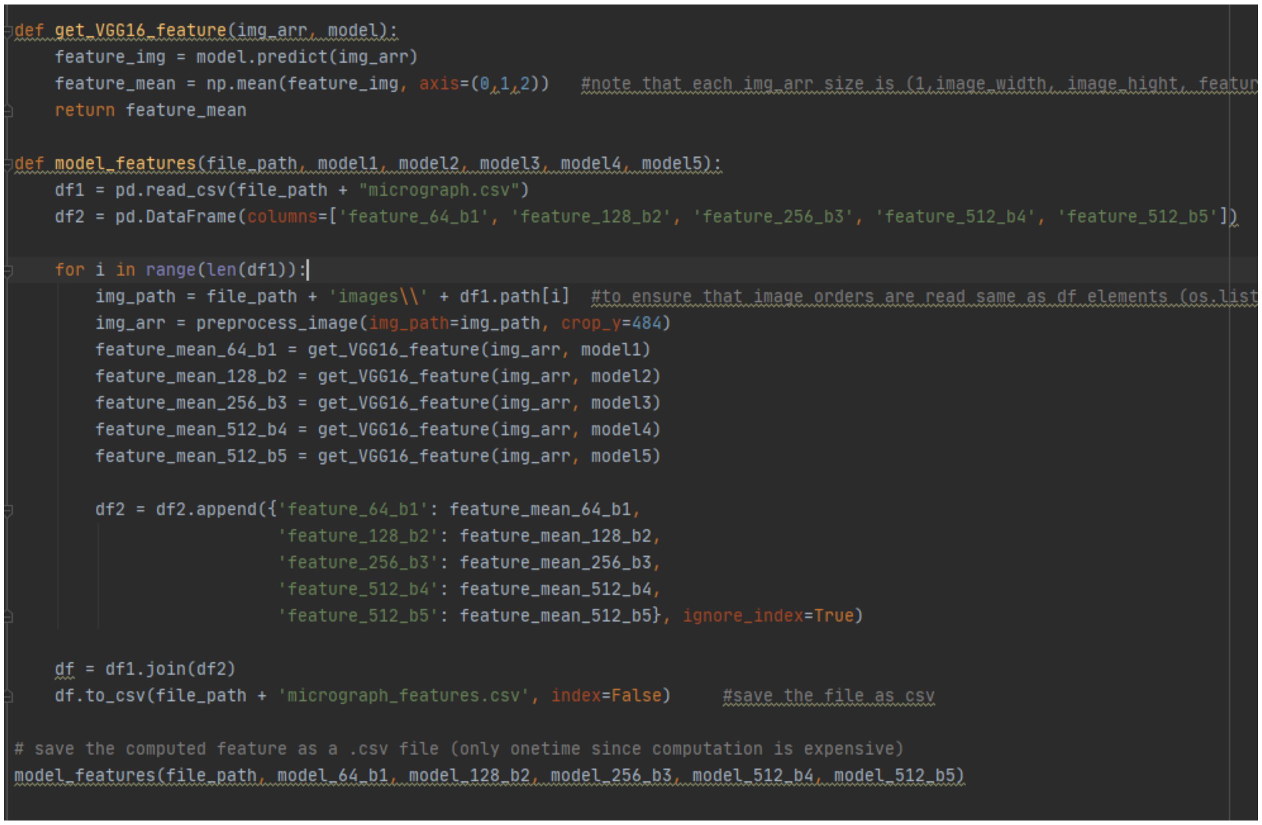
```
file_path = 'H:\\tamu\\courses\\MSEN660\\Homeworks\\Computer_project2\\DataSet\\CMU-UHCS_Dataset\\'

def preprocess_image(img_path, crop_y):
    img = image.load_img(img_path)
    img_arr = image.img_to_array(img)
    img_arr = img_arr[0:crop_y, :, :]
    img_arr = np.expand_dims(img_arr, axis=0)   # expand image dimenssion since Keras accept images i
    img_arr = preprocess_input(img_arr)  # keras pre+processing: adequate your image to the format the
    return img_arr
```

Using different layers of the model for feature extraction:



```
# Transfer learning: using VGG16 as a bas_model (not including fully_connected layer) /
base_model = VGG16(weights='imagenet', include_top=False)
base_model.summary()
# extracting the desired feature map for the first layer of our model
model_64_b1 = Model(inputs=base_model.input, outputs=base_model.get_layer('block1_pool').input)
model_128_b2 = Model(inputs=base_model.input, outputs=base_model.get_layer('block2_pool').input)
model_256_b3 = Model(inputs=base_model.input, outputs=base_model.get_layer('block3_pool').input)
model_512_b4 = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').input)
model_512_b5 = Model(inputs=base_model.input, outputs=base_model.get_layer('block5_pool').input)
```

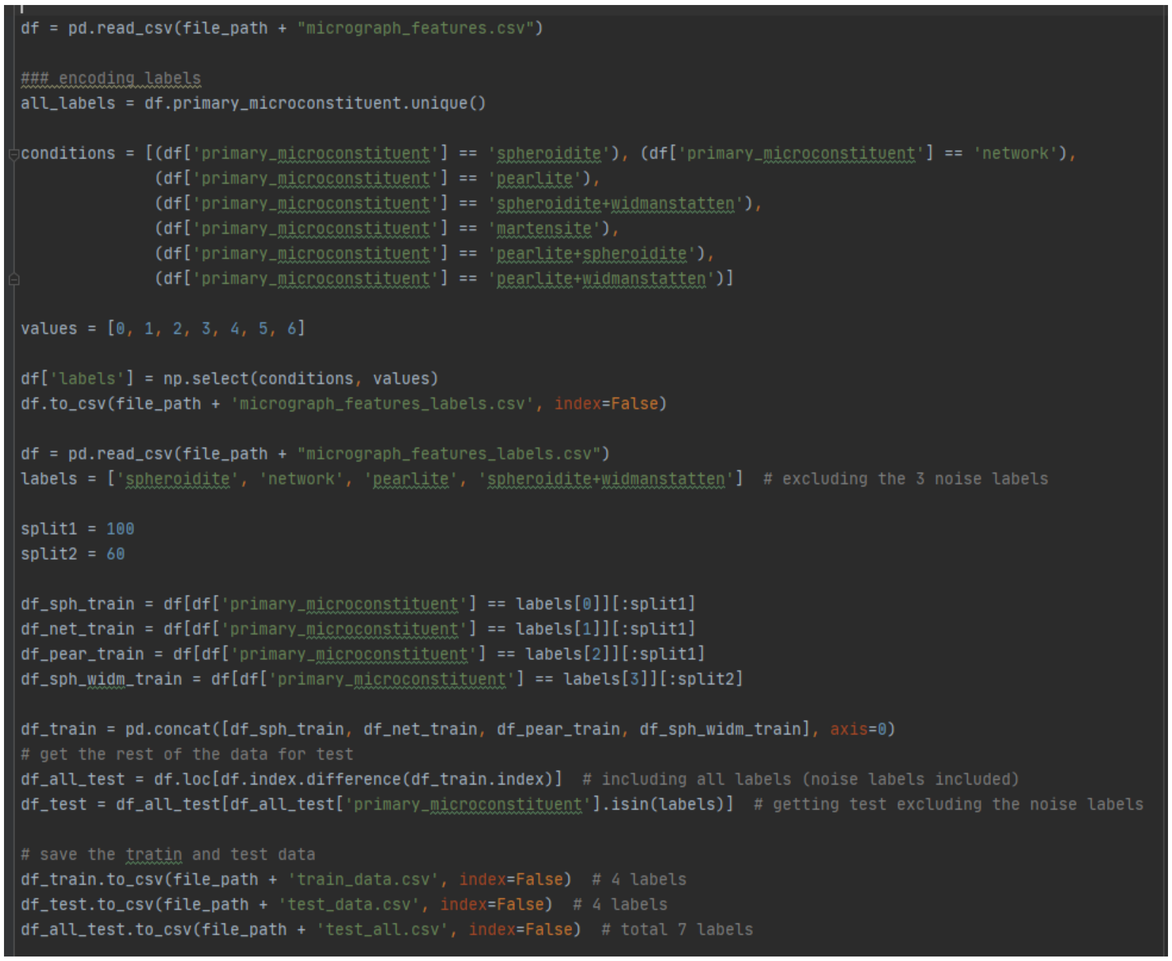
Use the VGG models for the training data and calculate the weights (the result is saved as .csv file)



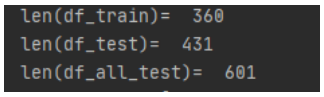
```
def get_VGG16_feature(img_arr, model):
    feature_img = model.predict(img_arr)
    feature_mean = np.mean(feature_img, axis=(0,1,2))   #note that each img_arr size is (1,image_width, image_hight, featur
    return feature_mean

def model_features(file_path, model1, model2, model3, model4, model5):
    df1 = pd.read_csv(file_path + "micrograph.csv")
    df2 = pd.DataFrame(columns=['feature_64_b1', 'feature_128_b2', 'feature_256_b3', 'feature_512_b4', 'feature_512_b5'])

    for i in range(len(df1)):
        img_path = file_path + 'images\\' + df1.path[i]  #to ensure that image orders are read same as df elements (os.list
        img_arr = preprocess_image(img_path=img_path, crop_y=484)
        feature_mean_64_b1 = get_VGG16_feature(img_arr, model1)
        feature_mean_128_b2 = get_VGG16_feature(img_arr, model2)
        feature_mean_256_b3 = get_VGG16_feature(img_arr, model3)
        feature_mean_512_b4 = get_VGG16_feature(img_arr, model4)
        feature_mean_512_b5 = get_VGG16_feature(img_arr, model5)

        df2 = df2.append({'feature_64_b1': feature_mean_64_b1,
                          'feature_128_b2': feature_mean_128_b2,
                          'feature_256_b3': feature_mean_256_b3,
                          'feature_512_b4': feature_mean_512_b4,
                          'feature_512_b5': feature_mean_512_b5}, ignore_index=True)

    df = df1.join(df2)
    df.to_csv(file_path + 'micrograph_features.csv', index=False)     #save the file as csv

# save the computed feature as a .csv file (only onetime since computation is expensive)
model_features(file_path, model_64_b1, model_128_b2, model_256_b3, model_512_b4, model_512_b5)
```

Split the data to train and test and save the .csv files

Only 4 labels are used to train data, the rest is considered as noise

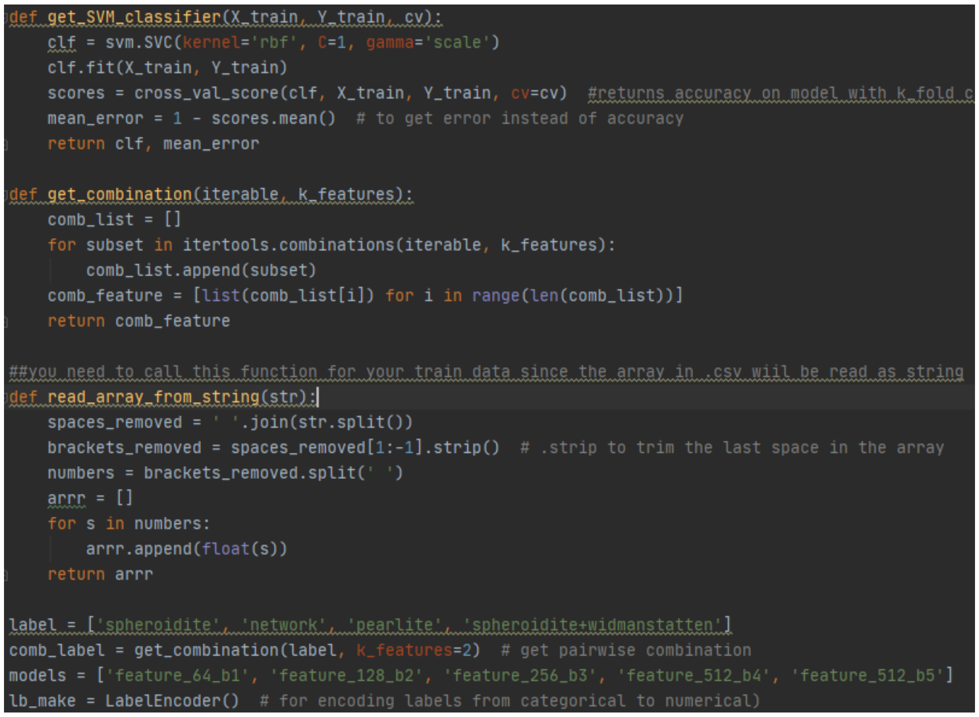


```
df = pd.read_csv(file_path + "micrograph_features.csv")

### encoding labels
all_labels = df.primary_microconstituent.unique()

conditions = [(df['primary_microconstituent'] == 'spheroidite'), (df['primary_microconstituent'] == 'network'),
              (df['primary_microconstituent'] == 'pearlite'),
              (df['primary_microconstituent'] == 'spheroidite+widmanstatten'),
              (df['primary_microconstituent'] == 'martensite'),
              (df['primary_microconstituent'] == 'pearlite+spheroidite'),
              (df['primary_microconstituent'] == 'pearlite+widmanstatten')]

values = [0, 1, 2, 3, 4, 5, 6]

df['labels'] = np.select(conditions, values)
df.to_csv(file_path + 'micrograph_features_labels.csv', index=False)

df = pd.read_csv(file_path + "micrograph_features_labels.csv")
labels = ['spheroidite', 'network', 'pearlite', 'spheroidite+widmanstatten']  # excluding the 3 noise labels

split1 = 100
split2 = 60

df_sph_train = df[df['primary_microconstituent'] == labels[0]][:split1]
df_net_train = df[df['primary_microconstituent'] == labels[1]][:split1]
df_pear_train = df[df['primary_microconstituent'] == labels[2]][:split1]
df_sph_widm_train = df[df['primary_microconstituent'] == labels[3]][:split2]

df_train = pd.concat([df_sph_train, df_net_train, df_pear_train, df_sph_widm_train], axis=0)
# get the rest of the data for test
df_all_test = df.loc[df.index.difference(df_train.index)]  # including all labels (noise labels included)
df_test = df_all_test[df_all_test['primary_microconstituent'].isin(labels)]  # getting test excluding the noise labels

# save the tratin and test data
df_train.to_csv(file_path + 'train_data.csv', index=False)  # 4 labels
df_test.to_csv(file_path + 'test_data.csv', index=False)  # 4 labels
df_all_test.to_csv(file_path + 'test_all.csv', index=False)  # total 7 labels
```



```
len(df_train)=  360
len(df_test)=  431
len(df_all_test)=  601
```

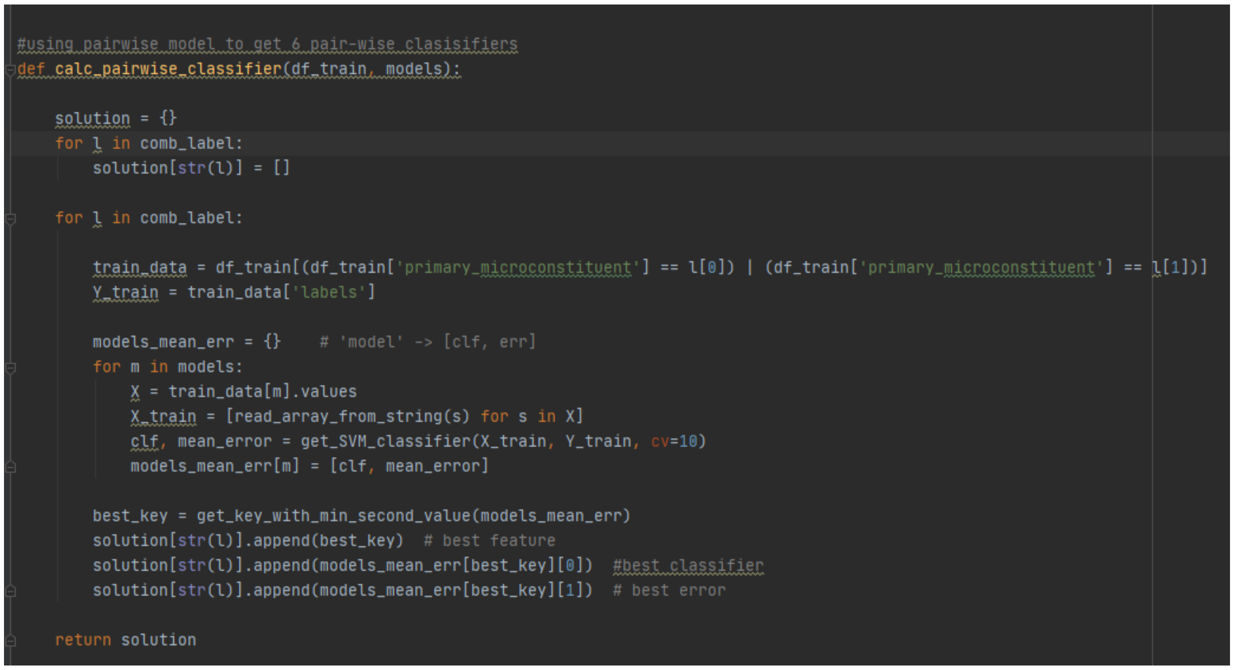
Required functions for pairwise and multilabel classifier:



```
def get_SVM_classifier(X_train, Y_train, cv):
    clf = svm.SVC(kernel='rbf', C=1, gamma='scale')
    clf.fit(X_train, Y_train)
    scores = cross_val_score(clf, X_train, Y_train, cv=cv)  #returns accuracy on model with k_fold c
    mean_error = 1 - scores.mean()  # to get error instead of accuracy
    return clf, mean_error

def get_combination(iterable, k_features):
    comb_list = []
    for subset in itertools.combinations(iterable, k_features):
        comb_list.append(subset)
    comb_feature = [list(comb_list[i]) for i in range(len(comb_list))]
    return comb_feature

##you need to call this function for your train data since the array in .csv will be read as string
def read_array_from_string(str):
    spaces_removed = ' '.join(str.split())
    brackets_removed = spaces_removed[1:-1].strip()  # .strip to trim the last space in the array
    numbers = brackets_removed.split(' ')
    arrr = []
    for s in numbers:
        arrr.append(float(s))
    return arrr

label = ['spheroidite', 'network', 'pearlite', 'spheroidite+widmanstatten']
comb_label = get_combination(label, k_features=2)  # get pairwise combination
models = ['feature_64_b1', 'feature_128_b2', 'feature_256_b3', 'feature_512_b4', 'feature_512_b5']
lb_make = LabelEncoder()  # for encoding labels from categorical to numerical)
```

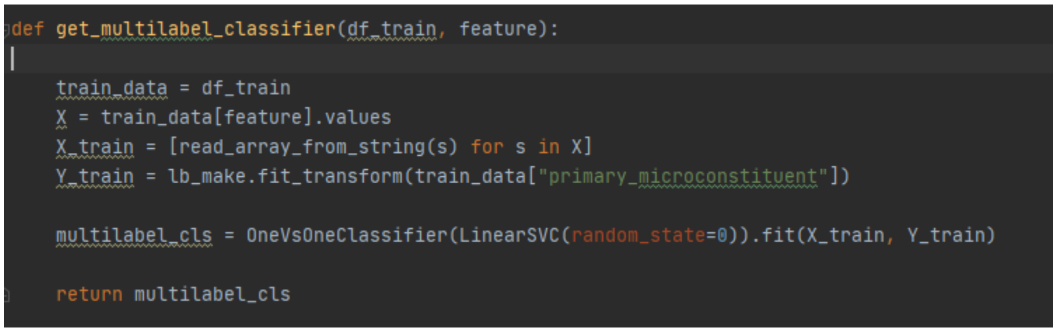
The last function is to get array from strings of array. When I read feature vectors from .csv file they are read as str rather than vector, hence I needed to bring them back to vector.

pairwise classifier:



```python
#using pairwise model to get 6 pair-wise clasisifiers
def calc_pairwise_classifier(df_train, models):

    solution = {}
    for l in comb_label:
        solution[str(l)] = []

    for l in comb_label:

        train_data = df_train[(df_train['primary_microconstituent'] == l[0]) | (df_train['primary_microconstituent'] == l[1])]
        Y_train = train_data['labels']

        models_mean_err = {}    # 'model' -> [clf, err]
        for m in models:
            X = train_data[m].values
            X_train = [read_array_from_string(s) for s in X]
            clf, mean_error = get_SVM_classifier(X_train, Y_train, cv=10)
            models_mean_err[m] = [clf, mean_error]

        best_key = get_key_with_min_second_value(models_mean_err)
        solution[str(l)].append(best_key)  # best feature
        solution[str(l)].append(models_mean_err[best_key][0])  #best classifier
        solution[str(l)].append(models_mean_err[best_key][1])  # best error

    return solution
```

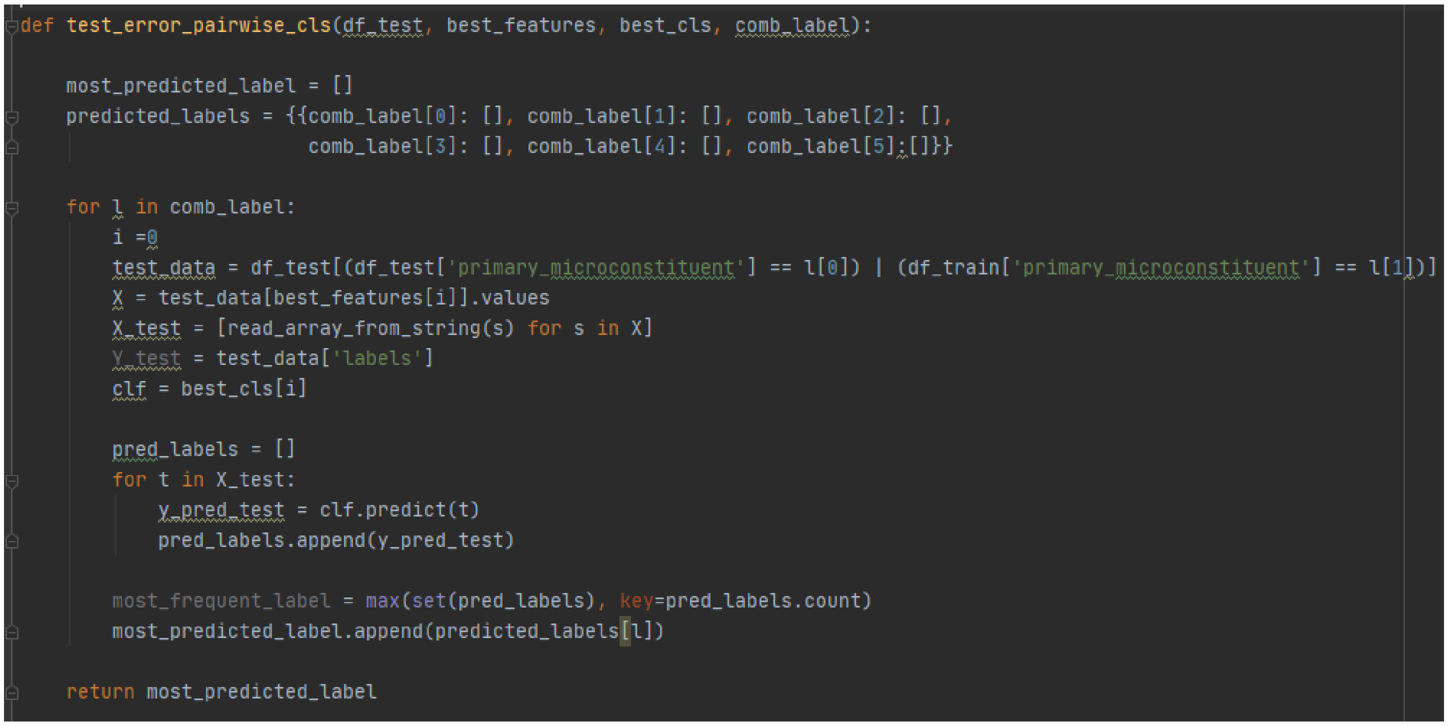
Get multilabel classifier using OneVsOneClassifier



```python
def get_multilabel_classifier(df_train, feature):

    train_data = df_train
    X = train_data[feature].values
    X_train = [read_array_from_string(s) for s in X]
    Y_train = lb_make.fit_transform(train_data["primary_microconstituent"])

    multilabel_cls = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X_train, Y_train)

    return multilabel_cls
```

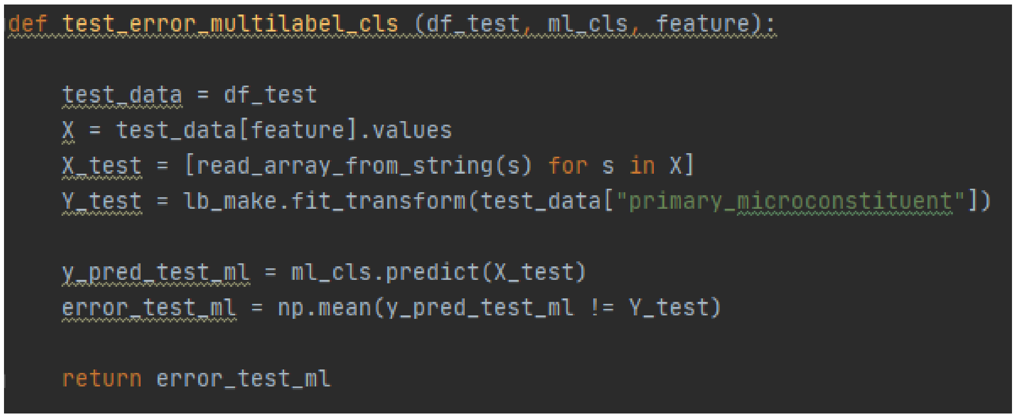
for new image (test data) the selected 6 pairwise classifier and 1 multi_label classifier will be used:

testing on pairwise classifier:



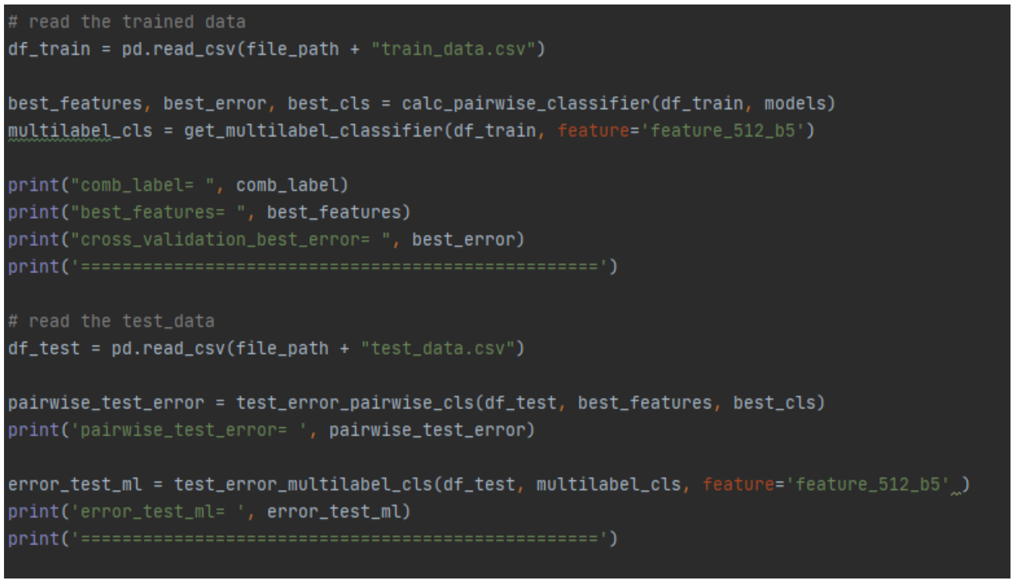
```python
def test_error_pairwise_cls(df_test, best_features, best_cls, comb_label):

    most_predicted_label = []
    predicted_labels = {{comb_label[0]: [], comb_label[1]: [], comb_label[2]: [],
                         comb_label[3]: [], comb_label[4]: [], comb_label[5]:[]}}

    for l in comb_label:
        i =0
        test_data = df_test[(df_test['primary_microconstituent'] == l[0]) | (df_train['primary_microconstituent'] == l[1])]
        X = test_data[best_features[i]].values
        X_test = [read_array_from_string(s) for s in X]
        Y_test = test_data['labels']
        clf = best_cls[i]

        pred_labels = []
        for t in X_test:
            y_pred_test = clf.predict(t)
            pred_labels.append(y_pred_test)

        most_frequent_label = max(set(pred_labels), key=pred_labels.count)
        most_predicted_label.append(predicted_labels[l])

    return most_predicted_label
```

Testing on multilabel classifier:



```python
def test_error_multilabel_cls (df_test, ml_cls, feature):

    test_data = df_test
    X = test_data[feature].values
    X_test = [read_array_from_string(s) for s in X]
    Y_test = lb_make.fit_transform(test_data["primary_microconstituent"])

    y_pred_test_ml = ml_cls.predict(X_test)
    error_test_ml = np.mean(y_pred_test_ml != Y_test)

    return error_test_ml
```

Calling functions:



```python
# read the trained data
df_train = pd.read_csv(file_path + "train_data.csv")

best_features, best_error, best_cls = calc_pairwise_classifier(df_train, models)
multilabel_cls = get_multilabel_classifier(df_train, feature='feature_512_b5')

print("comb_label= ", comb_label)
print("best_features= ", best_features)
print("cross_validation_best_error= ", best_error)
print('================================================')

# read the test_data
df_test = pd.read_csv(file_path + "test_data.csv")

pairwise_test_error = test_error_pairwise_cls(df_test, best_features, best_cls)
print('pairwise_test_error= ', pairwise_test_error)

error_test_ml = test_error_multilabel_cls(df_test, multilabel_cls, feature='feature_512_b5' )
print('error_test_ml= ', error_test_ml)
print('================================================')
```

Results:

comb_labels= [['spheroidite', 'network'], ['spheroidite', 'pearlite'], ['spheroidite', 'spheroidite+widmanstatten'], ['network', 'pearlite'], ['network', 'spheroidite+widmanstatten'], ['pearlite', 'spheroidite+widmanstatten']]

a) Crosvalidation score for each feature-layer per every pairwise classifier:

Pairwise Classifier		feature_64_ b1	feature_128_ b2	feature_256_ b3	feature_512_ b4	feature_512_ b5
spheroidite	network	0.72	0.92	0.985	0.985	0.989
spheroidite	pearlite	0.845	0.875	0.97	0.99	0.995
spheroidite	spheroidite+widmanstat ten	0.681	0.7	0.775	0.813	0.845
network	pearlite	0.87	0.93	0.96	0.99	1
network	spheroidite+widmanstat ten	0.744	0.938	0.994	1	0.981
pearlite	spheroidite+widmanstat ten	0.836	0.888	0.925	0.969	0.975

Discussion: the results show that The fifth feature layer (vector-size: 512) has the best result for all six pairwise classifiers. This could be justified because of the size of the feature and also being later out put of the VGG (block 5) so more process is done in the later outputs compare to the primary outputs. Hence the b-512 is also used for the multilabel classifier.

b) Based on the question instruction, we will get 326 samples for train and 431 samples for tests with 4 categories defined in the question. Pairwise classifier is used on the test micrographs with the two labels used to train the corresponding classifier. For multiple classifier the whole test with all labels is used. The table below, shows the number of errors for each category and its classifiers separately and then the test error rate is the average of all classifiers errors for a specific category.

Pairwise classifier	Spheroidite(274)	Network(112)	Pearlite(24)	Sph+wid(21)
Clf1	3	6		
Clf2	4		0	
Clf3	16			5
Clf4	-	5	0	
Clf5	-	2		0
Clf6	-		0	1
Average	2.8	0.59	0	6.35

multi-label	Spheroidite(274)	Network(112)	Pearlite(24)	Sph+wid(21)
one-vs-one	19	1	0	4
%	6.9	0.89	0	19.04

c) There are 107 test points:

Pairwise	Multi-classs
spheroidite	spheroidite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
spheroidite	spheroidite
spheroidite	spheroidite
spheroidite	spheroidite
spheroidite	network
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	spheroidite+widmanstatten
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite

pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite+widmanstatten
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite+widmanstatten
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
pearlite	spheroidite+widmanstatten
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	network
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite+widmanstatten
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite

pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	spheroidite
pearlite	pearlite
pearlite	pearlite
spheroidite	spheroidite
spheroidite	network

The results are the same for most of the test points with few exceptions. The reason is that the multi-label classifier is not trained for other multi-labels and hence may not yield the correct result.

d)

Pearlite+wid

pearlite
spheroidite+widmanstatten
pearlite
spheroidite+widmanstatten
spheroidite+widmanstatten
pearlite
pearlite
spheroidite+widmanstatten
spheroidite+widmanstatten
pearlite
pearlite
spheroidite+widmanstatten
pearlite
spheroidite+widmanstatten
spheroidite+widmanstatten
pearlite
spheroidite+widmanstatten
pearlite
spheroidite+widmanstatten
spheroidite+widmanstatten

pearlite
spheroidite+widmanstatten
spheroidite+widmanstatten
spheroidite+widmanstatten
spheroidite+widmanstatten
pearlite
pearlite

Martensite:

pearlite
network
pearlite
pearlite
pearlite
network
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
pearlite
network

pearlite
network
pearlite
pearlite

Discussion:

The classifier is trained for Pearlite and also for Spherodite+wid although it has not been trained specifically for Pearlite+Wid. So the results show that the tets data is predicted as either Pearlite or Spherodite+wid.

The classifier is not trained with Martensite label so it cannot classify it correctly however it mostly assign perlite to this category which could be interpreted that martensite feature is close to pearlite feature rather than the other 4 labels trained through the classifier.