In [1]:

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import cv2
import ntpath
import keras
import sys
from sklearn import metrics
from sklearn import preprocessing
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.svm import LinearSVC,SVC,NuSVC
from sklearn.multiclass import OneVsOneClassifier
from sklearn.preprocessing import StandardScaler
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input,VGG16
from keras.models import Sequential,load_model,Model
from keras.utils import to_categorical
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [2]:

```
datadir = 'micrograph'
columns = ['micrograph_id','path','micron_bar','micron_bar_units','micron_bar_px',
           'magnification','detector','sample_key','contributor_key','primary_microconstituent']
data = pd.read_csv('micrograph.csv')
data.head()
```

Out[2]:

	micrograph_id	path	micron_bar	micron_bar_units	micron_bar_px	magnification	detector	sample_key	contributor_key
0	1	micrograph1.tif	5	um	129	4910x	SE	42.0	2
1	2	micrograph2.tif	10	um	103	1964X	SE	18.0	2
2	4	micrograph4.tif	10	um	129	NaN	SE	35.0	2
3	5	micrograph5.tif	5	um	129	4910X	SE	10.0	2
4	6	micrograph6.tif	20	um	124	1178X	SE	29.0	2

In [3]:

```
p = 0
n = 0
s = 0
Xp = [];Xpt = [];
Yp = [];Ypt = [];
Xn = [];Xnt = [];
Yn = [];Ynt = [];
Xs = [];Xst = [];
Ys = [];Yst = [];
Xmt= []
Ymt = []
data_dir = 'micrograph'
for i in range(len(data)):
    indexed_data = data.iloc[i]
    if(indexed_data[9]=='pearlite'and p < 100):
        p = p+1
        x = indexed_data[1]
        Xp.append(os.path.join(data_dir, x.strip()))
        Yp.append(indexed_data[9])

    if(indexed_data[9]=='pearlite'and p >= 100):
        x = indexed_data[1]
        Xpt.append(os.path.join(data_dir, x.strip()))
        Ypt.append(indexed_data[9])
```

```
Xp = np.asarray(Xp)
Yp = np.asarray(Yp)

for i in range(len(data)):
    indexed_data = data.iloc[i]
    if(indexed_data[9]=='network'and n < 100):
        n = n+1
        x = indexed_data[1]
        Xn.append(os.path.join(data_dir, x.strip()))
        Yn.append(indexed_data[9])
    if(indexed_data[9]=='network'and n >= 100):
        x = indexed_data[1]
        Xnt.append(os.path.join(data_dir, x.strip()))
        Ynt.append(indexed_data[9])
Xn = np.asarray(Xn)
Yn = np.asarray(Yn)

for i in range(len(data)):
    indexed_data = data.iloc[i]
    if(indexed_data[9]=='spheroidite'and s < 100):
        s = s+1
        x = indexed_data[1]
        Xs.append(os.path.join(data_dir, x.strip()))
        Ys.append(indexed_data[9])
    if(indexed_data[9]=='spheroidite'and s >= 100):
        x = indexed_data[1]
        Xst.append(os.path.join(data_dir, x.strip()))
        Yst.append(indexed_data[9])

Xs = np.asarray(Xs)
Ys = np.asarray(Ys)

for i in range(len(data)):
    indexed_data = data.iloc[i]
    if(indexed_data[9]=='spheroidite'or indexed_data[9]=='network' or
       indexed_data[9]=='pearlite' or indexed_data[9]=='martensite' or
       indexed_data[9]=='widmanstatten'):
        pass
    else:
        x = indexed_data[1]
        Xmt.append(os.path.join(data_dir, x.strip()))
        Ymt.append(indexed_data[9])
```

In [5]:

```
def datagen(Xp,Xs,Xn,Yp,Ys,Yn,Xpt,Xst,Xnt,Ypt,Yst,Ynt):
    image_path = []
    label = []


    X1 = np.concatenate((Xp,Xs))
    Y1 = np.concatenate((Yp,Ys))
    X2 = np.concatenate((Xp,Xn))
    Y2 = np.concatenate((Yp,Yn))
    X3 = np.concatenate((Xs,Xn))
    Y3 = np.concatenate((Ys,Yn))


    Xr = np.concatenate((Xp,Xs,Xn))
    Yr = np.concatenate((Yp,Ys,Yn))

    X1t = np.concatenate((Xpt,Xst))
    Y1t = np.concatenate((Ypt,Yst))
    X2t = np.concatenate((Xpt,Xnt))
    Y2t = np.concatenate((Ypt,Ynt))
    X3t = np.concatenate((Xst,Xnt))
    Y3t = np.concatenate((Yst,Ynt))

    Xe = np.concatenate((Xpt,Xst,Xnt))
    Ye = np.concatenate((Ypt,Yst,Ynt))


    return X1,Y1,X2,Y2,X3,Y3,Xr,Yr,X1t,Y1t,X2t,Y2t,X3t,Y3t,Xe,Ye

X1,Y1,X2,Y2,X3,Y3,Xr,Yr,X1t,Y1t,X2t,Y2t,X3t,Y3t,Xe,Ye = datagen(Xp,Xs,Xn,Yp,Ys,Yn,Xpt,Xst,Xnt,Ypt,Y
st,Ynt)
```

In [6]:

```
#le = preprocessing.LabelEncoder()
#le.fit(['pearlite','spheroidite','network'])
#Y1 = le.transform(Y1)
##Y1 = to_categorical(Y1)
##le.fit(['pearlite',])
#Y2 = le.transform(Y2)
##Y2 = to_categorical(Y2)
##le.fit(['spheroidite','network'])
#Y3 = le.transform(Y3)
##Y3 = to_categorical(Y3)
#Y1t_ = le.transform(Yt)
#le.fit((['pearlite+spheroidite',]))
```

In [7]:

```
def pre_processor(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = x[0:484,:,:]
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x

def generator(X):
    x = []
    for i in range(len(X)):
        x.append(pre_processor(X[i]))
    return x
```

In [8]:

```
X1 = generator(X1)
X2 = generator(X2)
X3 = generator(X3)
Xr = generator(Xr)

X1t = generator(X1t)
X2t = generator(X2t)
X3t = generator(X3t)

Xe = generator(Xe)
```

In [33]:

```
#single labels
Xpt = generator(Xpt)
Xst = generator(Xst)
Xnt = generator(Xnt)
```

In [9]:

```
base_model = VGG16(weights='imagenet', include_top=False)
```

In [10]:

```
def exctractor(X,model):
    x = []
    for i in range(len(X)):
        x.append(model.predict(X[i]))
    return np.array(x)
```

In [ ]:

```
model_M1 = Model(inputs=base_model.input, outputs=base_model.get_layer('block1_pool').output)
model_M2 = Model(inputs=base_model.input, outputs=base_model.get_layer('block2_pool').output)
model_M3 = Model(inputs=base_model.input, outputs=base_model.get_layer('block3_pool').output)
model_M4 = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').output)
```

```
model_M5 = Model(inputs=base_model.input, outputs=base_model.get_layer('block5_pool').output)
```

In [11]:

```
X1M1 = exctractor(X1,model_M1)
X1M2 = exctractor(X1,model_M2)
X1M3 = exctractor(X1,model_M3)
X1M4 = exctractor(X1,model_M4)
X1M5 = exctractor(X1,model_M5)
```

In [12]:

```
X2M1 = exctractor(X2,model_M1)
X2M2 = exctractor(X2,model_M2)
X2M3 = exctractor(X2,model_M3)
X2M4 = exctractor(X2,model_M4)
X2M5 = exctractor(X2,model_M5)
```

In [13]:

```
X3M1 = exctractor(X3,model_M1)
X3M2 = exctractor(X3,model_M2)
X3M3 = exctractor(X3,model_M3)
X3M4 = exctractor(X3,model_M4)
X3M5 = exctractor(X3,model_M5)
```

In [47]:

```
X_train1_ = exctractor(Xr,model_M1)
X_train2_ = exctractor(Xr,model_M2)
X_train3_ = exctractor(Xr,model_M3)
X_train4_ = exctractor(Xr,model_M4)
X_train5_ = exctractor(Xr,model_M5)
```

In [16]:

```
def reshapper(X):
    x = []
    for i in range(len(X)):
        v = []
        a = X[i].reshape(X.shape[4],-1)
        for c in range(len(a)):
            v.append(sum(a[c])/(X.shape[2]*X.shape[3]))
        x.append(np.array(v).T)
    return np.array(x)
```

In [17]:

```
X1M1_ = reshapper(X1M1)
X1M2_ = reshapper(X1M2)
X1M3_ = reshapper(X1M3)
X1M4_ = reshapper(X1M4)
X1M5_ = reshapper(X1M5)
```

In [18]:

```
X2M1_ = reshapper(X2M1)
X2M2_ = reshapper(X2M2)
X2M3_ = reshapper(X2M3)
X2M4_ = reshapper(X2M4)
X2M5_ = reshapper(X2M5)
```

In [19]:

```
X3M1_ = reshapper(X3M1)
X3M2_ = reshapper(X3M2)
X3M3_ = reshapper(X3M3)
X3M4_ = reshapper(X3M4)
X3M5_ = reshapper(X3M5)
```

In [48]:

```
X_train1 = reshapper(X_train1_)
X_train2 = reshapper(X_train2_)
X_train3 = reshapper(X_train3_)
X_train4 = reshapper(X_train4_)
X_train5 = reshapper(X_train5_)
```

In [52]:

```
clf1 = SVC(gamma ='auto' ,kernel ='rbf', C = 10)

scores_X1M1 = cross_val_score(clf1, X1M1_, Y1, cv=10)
scores_X1M2 = cross_val_score(clf1, X1M2_, Y1, cv=10)
scores_X1M3 = cross_val_score(clf1, X1M3_, Y1, cv=10)
scores_X1M4 = cross_val_score(clf1, X1M4_, Y1, cv=10)
scores_X1M5 = cross_val_score(clf1, X1M5_, Y1, cv=10)
print('CV error on layer 1',1-sum(scores_X1M1)/10)
print('CV error on layer 2',1-sum(scores_X1M2)/10)
print('CV error on layer 3',1-sum(scores_X1M3)/10)
print('CV error on layer 4',1-sum(scores_X1M4)/10)
print('CV error on layer 5',1-sum(scores_X1M5)/10)
```

```
CV error on layer 1 0.49
CV error on layer 2 0.49
CV error on layer 3 0.49
CV error on layer 4 0.5
CV error on layer 5 0.13000000000000012
```

In [53]:

```
clf2 = SVC(gamma ='auto' ,kernel ='rbf', C = 10)
scores_X2M1 = cross_val_score(clf2, X2M1_, Y2, cv=10)
scores_X2M2 = cross_val_score(clf2, X2M2_, Y2, cv=10)
scores_X2M3 = cross_val_score(clf2, X2M3_, Y2, cv=10)
scores_X2M4 = cross_val_score(clf2, X2M4_, Y2, cv=10)
scores_X2M5 = cross_val_score(clf2, X2M5_, Y2, cv=10)
print('CV error on layer 1',1-sum(scores_X2M1)/10)
print('CV error on layer 2',1-sum(scores_X2M2)/10)
print('CV error on layer 3',1-sum(scores_X2M3)/10)
print('CV error on layer 4',1-sum(scores_X2M4)/10)
print('CV error on layer 5',1-sum(scores_X2M5)/10)
```

```
CV error on layer 1 0.495
CV error on layer 2 0.49
CV error on layer 3 0.4850000000000001
CV error on layer 4 0.495
CV error on layer 5 0.019999999999999907
```

In [54]:

```
clf3 = SVC(gamma ='auto' ,kernel ='rbf', C = 10)
scores_X3M1 = cross_val_score(clf3, X3M1_, Y3, cv=10)
scores_X3M2 = cross_val_score(clf3, X3M2_, Y3, cv=10)
scores_X3M3 = cross_val_score(clf3, X3M3_, Y3, cv=10)
scores_X3M4 = cross_val_score(clf3, X3M4_, Y3, cv=10)
scores_X3M5 = cross_val_score(clf3, X3M5_, Y3, cv=10)
print('CV error on layer 1',1-sum(scores_X3M1)/10)
print('CV error on layer 2',1-sum(scores_X3M2)/10)
print('CV error on layer 3',1-sum(scores_X3M3)/10)
print('CV error on layer 4',1-sum(scores_X3M4)/10)
print('CV error on layer 5',1-sum(scores_X3M5)/10)
```

```
CV error on layer 1 0.5
CV error on layer 2 0.49
CV error on layer 3 0.4850000000000001
CV error on layer 4 0.4850000000000001
CV error on layer 5 0.05500000000000005
```

In [55]:

```
clf = SVC(gamma ='auto' ,kernel ='rbf', C = 10)
clfm = OneVsOneClassifier(clf)
scores_Xtest1 = cross_val_score(clfm, X_train1, Yr, cv=10)
scores_Xtest2 = cross_val_score(clfm, X_train2, Yr, cv=10)
scores_Xtest3 = cross_val_score(clfm, X_train3, Yr, cv=10)
scores_Xtest4 = cross_val_score(clfm, X_train4, Yr, cv=10)
scores_Xtest5 = cross_val_score(clfm, X_train5, Yr, cv=10)
print('CV error on layer 1',1-sum(scores_Xtest1)/10)
print('CV error on layer 2',1-sum(scores_Xtest2)/10)
print('CV error on layer 3',1-sum(scores_Xtest3)/10)
print('CV error on layer 4',1-sum(scores_Xtest4)/10)
print('CV error on layer 5',1-sum(scores_Xtest5)/10)
```

```
CV error on layer 1 0.6599999999999999
CV error on layer 2 0.6533333333333333
CV error on layer 3 0.6499999999999999
CV error on layer 4 0.6566666666666666
CV error on layer 5 0.1233333333333334
```

The final Maxpool layer gives the least error and hence we will use only the final layer i.e Model5 or block_pool5

In [15]:

```
X_test1 = exctractor(X1t,model_M5)
X_test2 = exctractor(X2t,model_M5)
X_test3 = exctractor(X3t,model_M5)
X_test = exctractor(Xe,model_M5)
```

In [23]:

```
X_test1 = reshapper(X_test1)
X_test2 = reshapper(X_test2)
X_test3 = reshapper(X_test3)
X_test = reshapper(X_test)
```

In [35]:

```
Xpt = exctractor(Xpt,model_M5)
Xst = exctractor(Xst,model_M5)
Xnt = exctractor(Xnt,model_M5)
```

In [36]:

```
Xpt = reshapper(Xpt)
Xst = reshapper(Xst)
Xnt = reshapper(Xnt)
```

In [50]:

```
#test with 1 labels test set


e11 = clf1.fit(X1M5_, Y1).score(Xpt,Ypt)
e12 = clf1.fit(X1M5_, Y1).score(Xst,Yst)

e21 = clf1.fit(X2M5_, Y2).score(Xpt,Ypt)
e22 = clf1.fit(X2M5_, Y2).score(Xnt,Ynt)


e31 = clf3.fit(X3M5_, Y3).score(Xnt,Ynt)
e32 = clf3.fit(X3M5_, Y3).score(Xst,Yst)

ept = clfm.fit(X_train5,Yr).score(Xpt,Ypt)
est = clfm.fit(X_train5,Yr).score(Xst,Yst)
ent = clfm.fit(X_train5,Yr).score(Xnt,Ynt)
```

```
print('Error on class 1 (pearlite)',1-e11)
print('Error on class 1 (spherodite)',1-e12)

print('Error on class 2 (pearlite)',1-e21)
print('Error on class 2 (network)',1-e22)

print('Error on class 3 (network)',1-e31)
print('Error on class 3 (spherodite)',1-e32)

print('Error on multiclass (pearlite)',1-ept)
print('Error on multiclass (spherodite)',1-est)
print('Error on multiclass (network)',1-ent)
```

```
Error on class 1 (pearlite) 0.07999999999999996
Error on class 1 (spherodite) 0.20727272727272728
Error on class 2 (pearlite) 0.040000000000000036
Error on class 2 (network) 0.06194690265486724
Error on class 3 (network) 0.08849557522123896
Error on class 3 (spherodite) 0.032727272727272716
Error on multiclass (pearlite) 0.07999999999999996
Error on multiclass (spherodite) 0.24
Error on multiclass (network) 0.09734513274336287
```

In [51]:

```
#test with 2 and 3 labels test set


e1 = clf1.fit(X1M5_, Y1).score(X_test1,Y1t)
e2 = clf2.fit(X2M5_, Y2).score(X_test2,Y2t)
e3 = clf3.fit(X3M5_, Y3).score(X_test3,Y3t)

et = clfm.fit(X_train5,Yr).score(X_test,Ye)

epst = clfm.fit(X_train,Yr).score(X_test1,Y1t)
epnt = clfm.fit(X_train,Yr).score(X_test2,Y2t)
esnt = clfm.fit(X_train,Yr).score(X_test3,Y3t)
print('Error on class 1 (pearlite & spherodite)',1-e1)
print('Error on class 2 (pearlite & network)',1-e2)
print('Error on class 3 (spherodite & network)',1-e3)
print('Error on multiclass (all)',1-et)
print('Error on multiclass (pearlite & spherodite)',1-epst)
print('Error on multiclass (pearlite & network)',1-epnt)
print('Error on multiclass (spherodite & network)',1-esnt)
```

```
Error on class 1 (pearlite & spherodite) 0.19666666666666666
Error on class 2 (pearlite & network) 0.05797101449275366
Error on class 3 (spherodite & network) 0.0489690721649485
Error on multiclass (all) 0.19128329297820823
Error on multiclass (pearlite & spherodite) 0.22666666666666668
Error on multiclass (pearlite & network) 0.09420289855072461
Error on multiclass (spherodite & network) 0.19845360824742264
```

VGG16 is one of the most sophisticated CNN architecture which is widely used in machine learning for various problems like object detection, image classification, etc.

It is trained on millions of images, in other words the network has weights which are ideally suited for feature extractions. When an image is passed through VGG model, with every consecutive layer, more and more image features which are crucial for the recognition of that image are highlighted.

In our case, as it is also tested with cross-validation score, we can confidently say that the 5^{th} max-pooling layer generates the features with almost ideal requirements. The test error also seems reasonable when checked with unseen images which were passed through the 5^{th} maxpool layer.

The test error can be further reduced if feature selection is carried out on the features extracted from the 5^{th} maxpool layer.

CV error on layer 1 class 1 0.49 Class1(pearlite & spherodite)

CV error on layer 2 class 1 0.49

CV error on layer 3 class 1 0.49

CV error on layer 4 class 1 0.5

CV error on layer 5 class 1 0.13

CV error on layer 1 class 2 0.495 Class2(pearlite & network)

CV error on layer 2 class 2 0.49

CV error on layer 3 class 2 0.485

CV error on layer 4 class 2 0.495

CV error on layer 5 class 2 0.019

CV error on layer 1 class 3 0.5 Class 3 (spherodite & network)

CV error on layer 2 class 3 0.49

CV error on layer 3 class 3 0.48

CV error on layer 4 class 3 0.485

CV error on layer 5 class 3 0.019

CV error on layer 1 multiclass 0.659

CV error on layer 2 multiclass 0.653

CV error on layer 3 multiclass 0.649

CV error on layer 4 multiclass 0.656

CV error on layer 5 multiclass 0.123

#Test with 1 labels test set

Error on class 1 (pearlite) 0.07999999999999996

Error on class 1 (spherodite) 0.20727272727272728

Error on class 2 (pearlite) 0.040000000000000036

Error on class 2 (network) 0.06194690265486724

Error on class 3 (network) 0.08849557522123896

Error on class 3 (spherodite) 0.032727272727272716

Error on multiclass (pearlite) 0.07999999999999996

Error on multiclass (spherodite) 0.24

Error on multiclass (network) 0.09734513274336287

#Test with 2 and 3 labels test set

Error on class 1 (pearlite & spherodite) 0.19666666666666666

Error on class 2 (pearlite & network) 0.05797101449275366

Error on class 3 (spherodite & network) 0.0489690721649485

Error on multiclass (all) 0.19128329297820823

Error on multiclass (pearlite & spherodite) 0.22666666666666668

Error on multiclass (pearlite & network) 0.09420289855072461

Error on multiclass (spherodite & network) 0.19845360824742264

The error is reasonably low for all the classifiers.