Materials Informatics – Fall 2020

Computer Project 2

Due on: Oct 31

We will use the Carnegie Mellon University Ultrahigh Carbon Steel (CMU-UHCS) dataset in

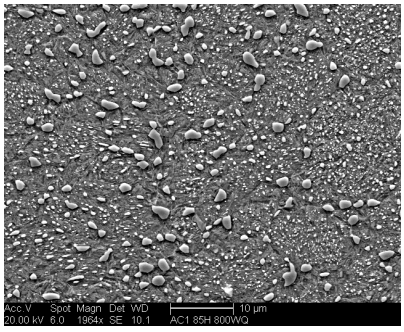
> B. DeCost, T. Francis and E. Holm (2017), "Exploring the microstructure manifold: image texture representations applied to ultrahigh carbon steel microstructures." arXiv:1702.01117v2.

The data set is available from https://braganeto.engr.tamu.edu/book-website/. (Download the zip file with all data sets.) In the folder `CMU-UHCS_Dataset` there are two excel files containing the microstructure labels and sample preparation information and a folder containing the raw images. Please read DeCost's paper to learn more about the data set.

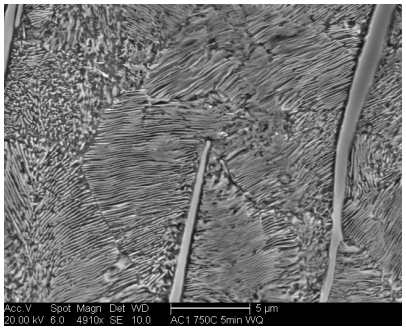
We will classify the micrographs according to **primary microconstituent**. There are a total of seven different labels, corresponding to different phases of steel resulting from different thermal processing (number of images in parenthesis): spheroidite (374), network (212), pearlite (124), pearlite + spheroidite (107), spheroidite+widmanstatten (81), martensite (36), and pearlite + widmanstatten (27). We will use the spherodite, network, pearlite, and spheroidite+widmanstatten categories for training. The training data will be **the first 100 data points** in the spherodite, network, pearlite categories and **the first 60 points** in the spheroidite+widmanstatten category. The remaining data points will compose the various test sets (more below).

Micrographs 2, 5, 9, and 58 in the database, corresponding to each of the four classes, are plotted below, from the top left clockwise. These materials are sufficiently different that classifying their micrographs should be easy, *provided that* one has the right features. In this computer project, we use the pre-trained convolutional layers of the VGG16 to provide the featurization.

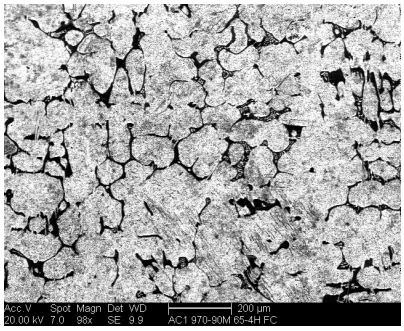
Spheroidite



Pearlite

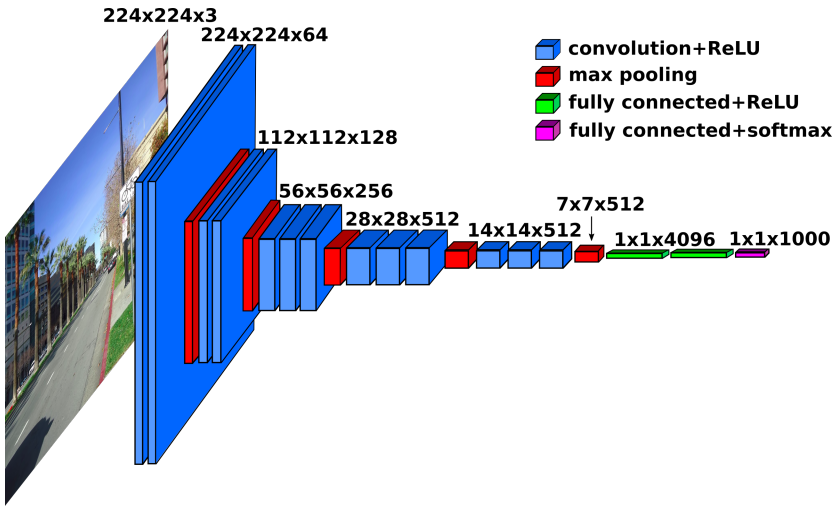


Carbide Network



Spheroidite+Widmanstätten

To featurize the images, we will use the pre-trained VGG16 deep convolutional neural network (CNN), discussed in the class, which has the architecture:



The classification rule to be used is a Radial Basis Function (RBF) nonlinear SVM. We will use a *one-vs-one* approach to deal with the multiple labels, where each of 4 choose $2 = 6$ classification problems for each pair of labels are carried out. Given a new image, each of the six classifiers is applied and then a vote is taken to achieve a consensus for the most often predicted label.

We will ignore the fully connected layers, and take the features from the input to the max-pooling layers only, using the "channels" mean value as the feature vector (each channel is a 2D image corresponding to the output of a different filter). This results in feature vectors of length 64, 128, 256, 512, 512, respectively (these lengths correspond to the number of filters in each layer and are fixed, having nothing to do with the image size). In each pairwise classification experiment, we will select one of the five layers according to the best 10-fold cross-validation error estimate (cross-validation estimators will be discussed in detail in Chapter 7).

You are supposed to record the following:

(a) The convolution layer used and the cross-validated error estimate for each of the six pairwise two-label classifiers.

(b) Separate test error rates on the unused micrographs of each of the four categories, for the pairwise two-label classifiers and the multilabel one-vs-one voting classifier described previously. For the pairwise classifiers use only the test micrographs with the two labels used to train the classifier. For the multilabel classifier, use the test micrographs with the corresponding four labels.

(c) For the mixed pearlite + spheroidite test micrographs, apply the trained pairwise classifier for pearlite vs. spheroidite and the multilabel voting classifier. Print the predicted labels by these two classifiers side by side (one row for each test micrograph). Comment your results.

(d) Now apply the multilabel classifier on the pearlite + widmanstatten and martensite micrographs and print the predicted labels. Compare to the results in part (c).

In each case above, interpret your results. Implementation should use the Scikit-Learn and Keras python libraries.

Some coding hints are given below.

1. The first step is to read in and preprocess each micrograph and featurize it. This will take most of the computation time. First read in the images using the Keras `image` utility:
   ```
   img = image.load_img('image file name')
   x = image.img_to_array(img)
   ```
   Next, crop the images to remove the subtitles:
   ```
   x = x[0:484,:,:]
   ```
   add an artificial dimension to specify a batch of one image (since Keras works with batches of images):
   ```
   x = np.expand_dims(x,axis=0)
   ```
   and use the Keras function `preprocess_input` to remove the image mean and perform other conditioning:
   ```
   x = preprocess_input(x)
   ```
   Notice that no image size reduction is necessary, as Keras can accept any input image size when in featurization mode.

2. The features are computed by first specifying as base model VGG16 with the pretrained ImageNet weights in featurization mode (do not include the top fully-connected layers):
   ```
   base_model = VGG16(weights='imagenet', include_top=False)
   ```
   extracting the desired feature map (e.g. for the first layer):
   ```
   model = Model(inputs=base_model.input,
                 outputs=base_model.get_layer('block1_pool').input)
   xb = model.predict(x)
   ```
   and computing its mean
   ```
   F = np.mean(xb,axis=(0,1,2))
   ```
   Notice that steps 1 and 2 have to be repeated for each of the micrographs.

3. The next step is to separate the generated feature vectors by label and by training/testing status, using standard python code. You should save the features to disk since they are expensive to compute.

4. You should use the scikit-learn functions `svm.SVC` and `cross_val_score` to train the SVM classifiers and calculate the cross-validation error rates, respectively, and record which layer produces the best results. When calling `svm.SVC`, set the kernel option to `'rbf'` (this corresponds to the Gaussian RBF) and the C and gamma parameters to 1 and `'scale'`, respectively.

5. Obtain test sets using standard python code. Compute the test set errors for each pairwise classifier and for the multiclass classifier, using the correspond best featurization (layer) for each pairwise classifier, obtained in the previous item. For the multiclass classifier, you should use the scikit-learn function `OnevsOneClassifier`, which has its own internal tie-breaking procedure.

6. The remaining classifiers and error rates for parts (c) and (d) can be obtained similarly.