Gr-ResQ – Graphene Recipes for Synthesis of high-Quality materials
Week 1 - Image Segmentation Laboratory

by Aagam Shah, Mitisha Surana, Sameh Tawfick, and Elif Ertekin

This week, we will explore three different approaches to image segmentation: template matching, k-means clustering, and U-Net Neural networks. Our goal is to be able to take an SEM image of graphene on a substrate, and quickly determine which of the pixels have graphene and which do not. Separating one set of pixels from another is a type of classification known as image segmentation. In projects like Gr-ResQ it is important to be able to rapidly segment images in a uniform and automated way.

The SEM images we will be using are using were generated in the group of Sam Tawfick by Mitisha Surana, on samples grown in Sam's group. Here is one example:

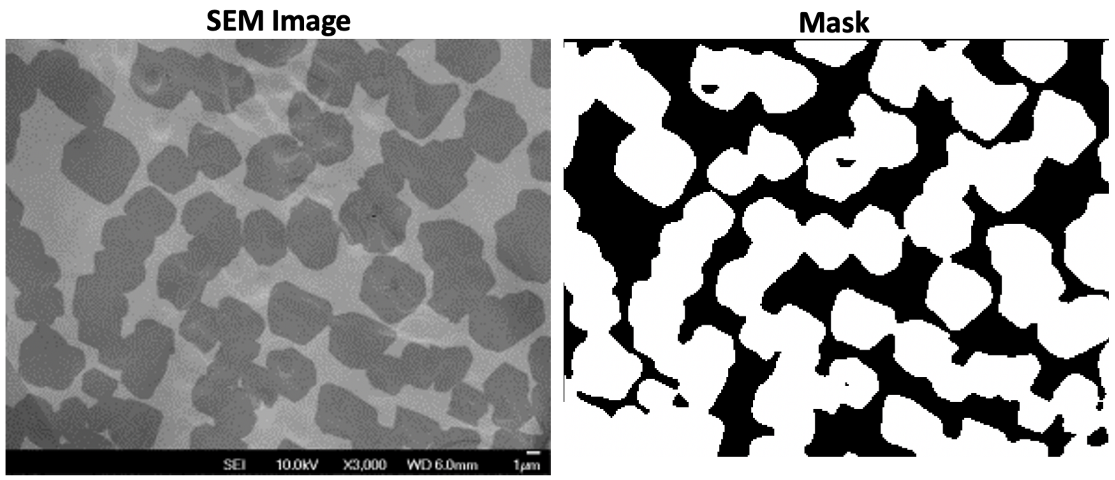


Figure 1: SEM image of graphene on copper, and mask.

- *Question 1. What percent of the field shown above is covered by graphene? Estimate to the nearest integer, and write down your estimated value:* ________________

The image on the left is from the SEM, and the image on the right is called a *mask*. On the mask, black and white indicate graphene and not-graphene (and not necessarily in any particular order!).

If you didn't have the mask, could you tell which pixels correspond to graphene and which pixels correspond to the substrate? Are there any other pixels that are not obviously graphene or substrate (contaminants, etc.)? Once we've seen one or two images, we begin to understand how to classify the pixels. We can recognize the darker contrast, and we look for dark blobs that often look hexagonal.

Human classification remains the best in terms of accuracy and precision – but human classification is slow. Also, how accurately do you think you can estimate the *coverage*, i.e. the proportion of the field of view contains graphene? This week we are going to look at a few ways to rapidly segment SEM images and measure quantities like coverage using image processing methods based on machine learning.

Before turning the page, answer the following question.

- *Question 2. If you had to come up with a computer algorithm to segment SEM images like the one above, what is the simplest way that you can think of to do it?*

Color Masking – The easiest way to segment an SEM Image*

* at least in our experience

Color masking refers to classifying the pixels in an image simply based on their intensity or hue. For example:

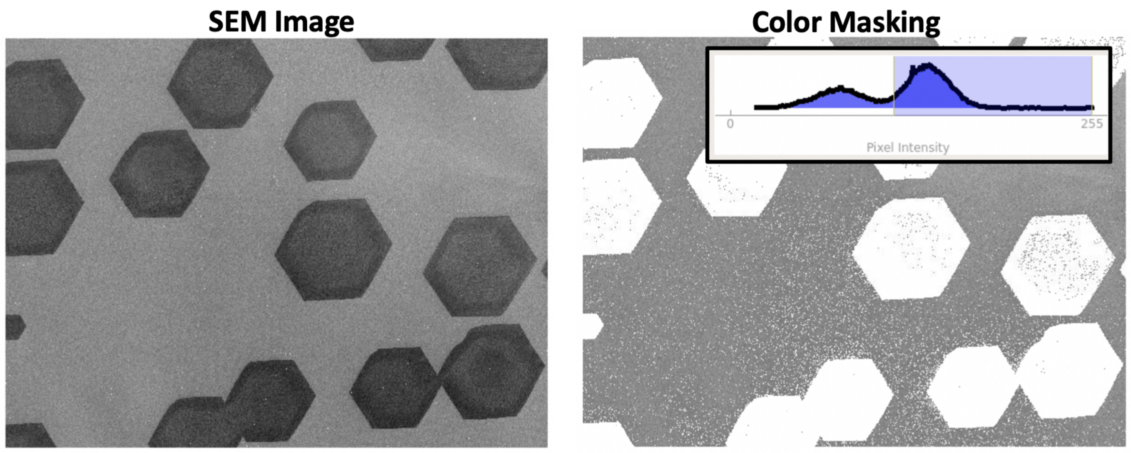


Figure 2: Eaxmple of color masking to segment SEM images.

Each pixel's intensity is determined. And then the pixels whose intensity exceed a selected threshold are selected. The histogram above shows the distribution of pixel intensities, and the red line shows· where the threshold has been set.

- *Question 3. What could be some shortcomings of color masking as a classification scheme? Look closely at the Figure above. Do you see any issues?*

Part I. Template Matching

Now we will have a chance to play with some jupyter notebooks that illustrate template matching and k-means clustering. These notebooks rely heavily on the cv2 python library (Computer Vision 2). Open the following notebook in your colab workspace:

https://github.com/ertekin-research-group/image-segment/blob/main/bin/Template_Matching.ipynb (paste this link in the GitHub tab on Google Colab)

Create a new code cell and enter the following command at the beginning of the notebook:
!git clone https://github.com/ertekin-research-group/image-segment.git

Change the file paths everyewhere in the notebook from *../data/* to *image-segment/data/*

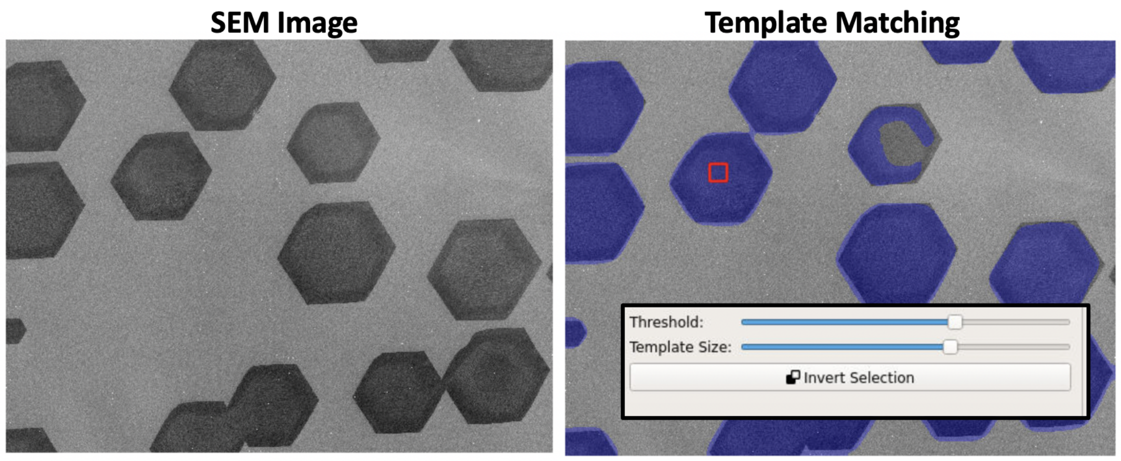


Figure 3: Example of template matching.

Template matching works by measuring how similar a regions of the image is to a selected template. The main parameters to set are the template position and size, and the threshold for similarity. Explore the notebook and run the cells.

- *Question 4. What do the cell blocks copied below do?*

- *Question 5. What is the effect of using a smaller or larger template? Why might you want a small template? A big one? What is the effect of changing the threshold?*

- *Question 6. Now have each person in your group choose a different image (change the path in the notebook to select an image!). Choose different types of images from each other. Play*

```
res = cv2.matchTemplate(img_gray, template, cv2.TM_SQDIFF_NORMED)

res_plot = plt.imshow(res, cmap='gray') # display the match percentages as a heat map
```

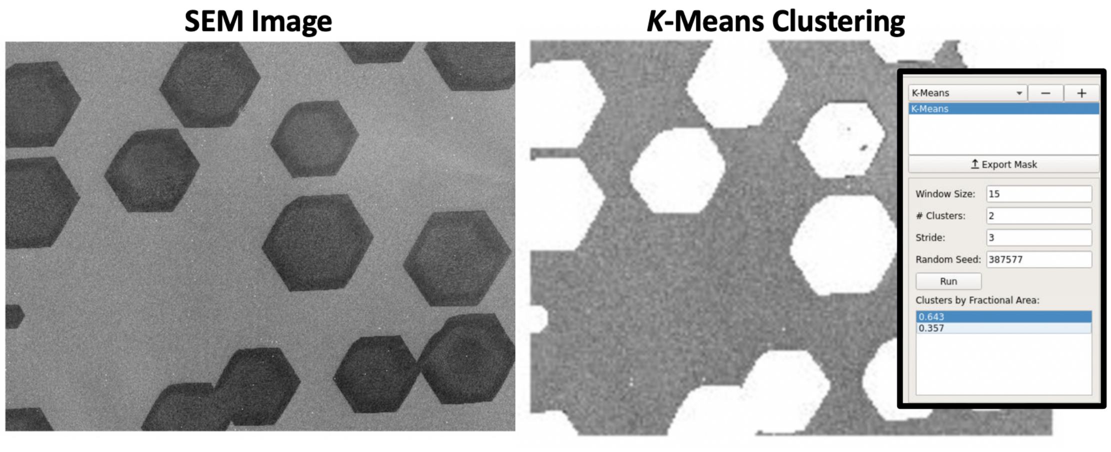
```
simple_mask = np.where(res<=threshold, 1, 0) # apply a binary filter

simple_plt = plt.imshow(simple_mask, cmap='gray') # display the masked image
```

with the parameters until you have made as good of a mask as possible. What are some of the shortcomings of template matching?

Part II. k-means clustering

Now open the k-means clustering notebook. k-means is an unsupervised learning method that aims to group all the pixels into a pre-selected number of clusters. All the pixels in a given cluster are more similar to each other than to the pixels in the other clusters. An example is shown below.



Window Size: 15
# Clusters: 2
Stride: 3
Random Seed: 387577
Clusters by Fractional Area:
0.643
0.357

Figure 4: Example of k-means clustering.

To open the notebook, follow the same steps as you did for Template Matching with the following notebook link: *https://github.com/ertekin-research-group/image-segment/blob/main/bin/K-Means.ipynb* Explore the notebook and run the cells. Note that k-means may take a few minutes to run, but mini-batch k-means will be faster. The main parameters are the window size, the number of clusters, and the stride length.

- *Question 7. What do the cell blocks copied below do?*

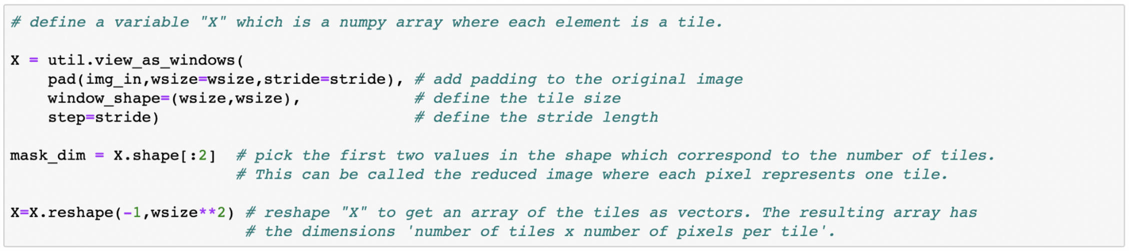
  ____
  ____
  ____

- *Question 8. Why is k-means considered to be a machine learning method? Why is it unsupervised?*

  ____
  ____
  ____
  ____
  ____

- *Question 9. What is the effect of varying the stride length? Window size? Number of clusters?*

  ____
  ____

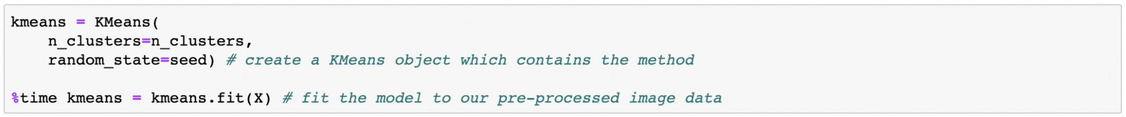


```
# define a variable "X" which is a numpy array where each element is a tile.

X = util.view_as_windows(
    pad(img_in,wsize=wsize,stride=stride), # add padding to the original image
    window_shape=(wsize,wsize),            # define the tile size
    step=stride)                           # define the stride length

mask_dim = X.shape[:2]  # pick the first two values in the shape which correspond to the number of tiles.
                        # This can be called the reduced image where each pixel represents one tile.

X=X.reshape(-1,wsize**2) # reshape "X" to get an array of the tiles as vectors. The resulting array has
                         # the dimensions 'number of tiles x number of pixels per tile'.
```



```
kmeans = KMeans(
    n_clusters=n_clusters,
    random_state=seed) # create a KMeans object which contains the method

%time kmeans = kmeans.fit(X) # fit the model to our pre-processed image data
```

- *Question 10. Now have each person in your group choose a different image (change the path in the notebook to select an image!). Choose different types of images from each other. Play with the parameters until you have made as good of a mask as possible. What are some of the shortcomings of k-means clustering?*

Part III. U-Net Neural Network

To be filled in by Elif ….