



Brief run-through of Workshop 1

In this tutorial:

- Login to nanoHUB
- Steps to follow the workshop

Aagam Shah, Darren Adams,
Sameh Tawfick, Elif Ertekin

University of Illinois at
Urbana-Champaign

Step 1: Login/Sign up to nanoHUB

The image displays two browser screenshots illustrating the process of logging in or signing up on nanoHUB.

Top Screenshot (nanoHUB.org): Shows the homepage with the navigation menu (RESOURCES, EXPLORE, NANOHUB-U, PARTNERS, COMMUNITY, ABOUT, SUPPORT, DONATE) and the 'Login Sign Up' button highlighted. The main heading reads "Serving Students, Researchers & Instructors" with statistics: "1.9 Million Annual Visitors" and "17,000 Simulation Users". Below are four service categories: "Model & Simulate", "Learn & Teach", "Develop Software", and "Share & Publish".

Bottom Screenshot (nanoHUB.org/register/): Shows the "Create New Account" page. The navigation menu is visible at the top. The main heading is "Create New Account". Below the heading, there are four social login options: "With an affiliated institution:", "Sign in with Facebook", "Sign in with Google", and "Sign in with LinkedIn". A sidebar on the right contains a note: "You can choose to log in via one of these services, and we'll help you fill in the info below!" and a link: "Already have an account? Log in here."



Step 2: View presentation slides for template matching

Unsupervised clustering methods for image segmentation: application to scanning electron microscopy images of graphene

Aagam Shah, Darren Adams, Sameh Tawfick, Elif Ertekin

University of Illinois at Urbana-Champaign



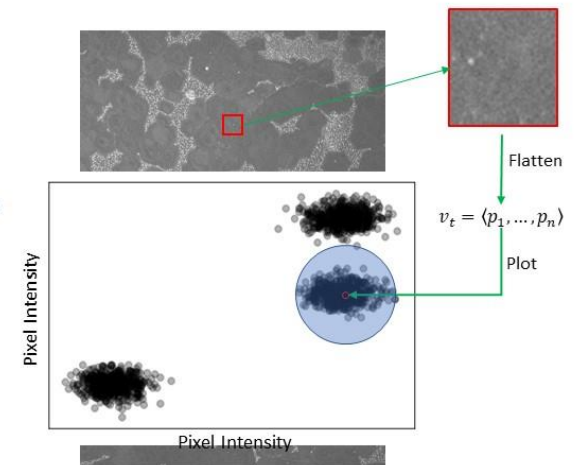
Template Matching

Idea: select area that looks like graphene and screen for similar looking areas

- Step 1: Select the “template”, flatten and vectorize it.
- Step 2: Plot it on the intensity vector plot
- Step 3: For all other parts of the image, measure how close they are to the template on the intensity vector plot
- Step 4: If the distance is within a threshold, classify as “graphene”. If not, then “not graphene”.

Parameters:

- Template position
- Template size
- Threshold (or distance)



Step 3: Launch imagesegment tool on nanoHUB

- From your browser, go to the link: <https://nanohub.org/tools/imagesegment>

SEM Image Segmentation Workshop

By [Aagam Rajeev Shah](#), [Darren K Adams](#), [Mitisha Surana](#), [Ricardo Toro](#), [Sameh H Tawfick](#), [E. Ertekin](#)


This tool introduces users to machine learning used to segment microscopy images

[Edit](#)

[Launch Tool](#)

Version 1.0 - published on 12 Jan 2021

doi:10.21981/9GM4-BM53 [cite this](#)

 This tool is closed source.

[View All Supporting Documents](#)

Click on Launch Tool to begin



Step 4: Navigate to the Template Matching Jupyter notebook

Machine Learning for SEM Image Segmentation in Materials Science

Scanning electron microscopy (SEM) images are typically used to observe the growth results of a synthesis experiment, such as areal coverage, nucleation density, and the shape, size, and quality of graphene domains. While the visual inspection of images can sometimes be sufficient to determine the quality of graphene, it is desirable to determine quantitative metrics as well. Quantitative metrics can provide for easier comparison between experimental results and are useful as response variables when attempting to predict optimal recipes. To calculate these metrics, we need to segment the image and each pixel needs to be classified as 'graphene' or 'not-graphene'.

The tutorials here will give you an insight into the usage of machine learning to segment microscopy images.

- **Get started:** Click on the links below to begin each tutorial.
- **Important:** To exit individual tutorials and return to this page, use `File -> Close and Halt`. "Terminate Session" (top right) will kill your entire Jupyter session.


[Template Matching:](#)

- Use template matching to segment a microscopy image of graphene on copper
- See the effects of changing the variables - ROI size, threshold and statistical technique

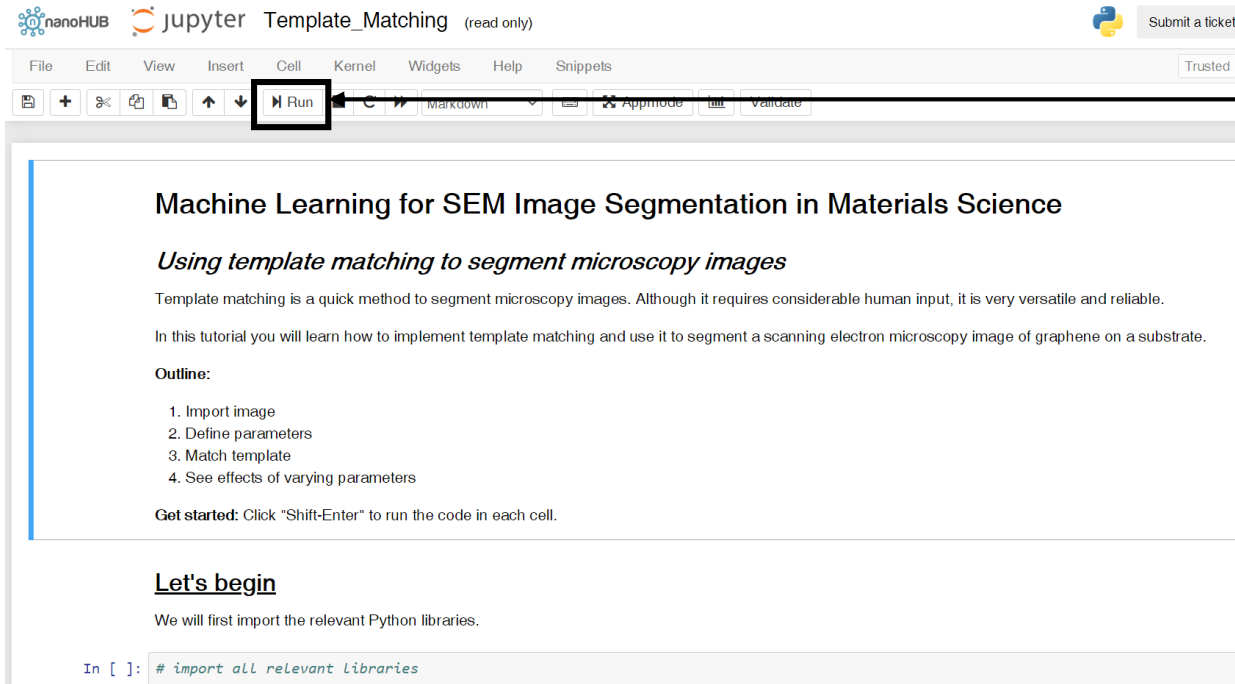
[K-Means Clustering:](#)

- Use K-Means Clustering to segment a microscopy image of graphene on copper
- See the effects of changing the variables - window size, stride and number of clusters

Click on this link



Step 5: Execute each cell in the Template Matching Jupyter notebook



nanoHUB jupyter Template_Matching (read only) Submit a ticket

File Edit View Insert Cell Kernel Widgets Help Snippets Trusted

Run

Machine Learning for SEM Image Segmentation in Materials Science

Using template matching to segment microscopy images

Template matching is a quick method to segment microscopy images. Although it requires considerable human input, it is very versatile and reliable.

In this tutorial you will learn how to implement template matching and use it to segment a scanning electron microscopy image of graphene on a substrate.

Outline:

1. Import image
2. Define parameters
3. Match template
4. See effects of varying parameters

Get started: Click "Shift-Enter" to run the code in each cell.

Let's begin

We will first import the relevant Python libraries.

```
In [ ]: # import all relevant libraries
```

Click this button to run a cell or press Shift-Enter on the keyboard

Note: Slide 7 and 8 show the steps in the Template Matching Jupyter notebook



Step 5.1: Import relevant packages and display the image

nanoHUB jupyter Template_Matching (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run Appmode Validate

Let's begin

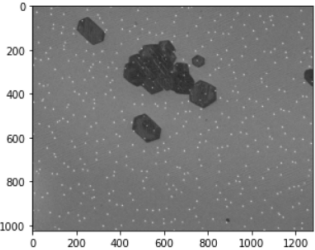
We will first import the relevant Python libraries.

```
In [1]: # import all relevant libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2 # this is the computer vision library that allows us to perform template matching
```

Import the image

Now, we import the image as a numpy n-dimensional array and display the image and its resolution.

```
In [2]: img_gray = cv2.imread('../data/test_template.tif', 0) # import the grayscale image as an ndarray
imgplot = plt.imshow(img_gray, cmap='gray') # display the raw image
```



Step 5.2: Define the parameters and view the template

nanoHUB jupyter Template_Matching (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

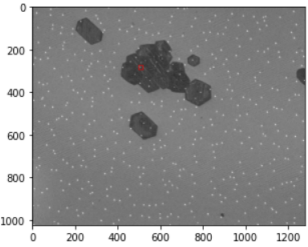
Run Appmode Validate

Define the parameters

We define all the parameters that we can change.

```
In [3]: template_size = 20 # define the side length of the template
threshold = 0.25 # define the threshold
template_posn = [275,500] # define the position of the template
```

```
In [4]: ## NOTE: THIS CELL IS NOT REQUIRED IN THE SEGMENTATION PROCESS. IT IS ONLY FOR VISUAL AID.
# View the template in the image
img_rgb = cv2.cvtColor(img_gray, cv2.COLOR_GRAY2BGR) # convert the image from grayscale to RGB
# add the rectangle to show the template
cv2.rectangle(img_rgb, (template_posn[1],template_posn[0]), (template_posn[1]+template_size,template_posn[0]+template_size), (255,0,0), 2)
imgplot = plt.imshow(img_rgb) # display the image with the template marked
```



Step 5.3: Match the template and apply a binary filter

nanoHUB jupyter Template_Matching (read only)

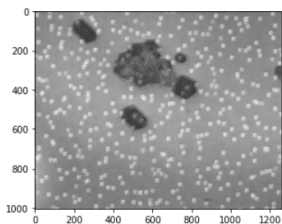
File Edit View Insert Cell Kernel Widgets Help Snippets

Run Appmode Validate

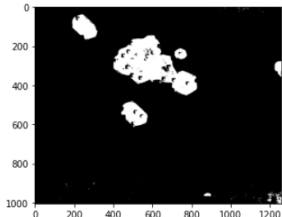
Match template

Now, we match the template to each tile in the image. The `matchTemplate` function returns an ndarray that can be interpreted as the degree of similarity.

```
In [6]: 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
```



```
In [7]: simple_mask = np.where(res<=threshold, 1, 0) # apply a binary filter
simple_plt = plt.imshow(simple_mask, cmap='gray') # display the masked image
```



Step 5.4: See the effects of varying the parameters

nanoHUB jupyter Template_Matching (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run Appmode Validate

Varying the parameters

Now, we vary the template size and the threshold and see how it affects our result.

```
In [8]: # define a function that does the template matching and returns a binary filtered array
def temp_match(roi_size, threshold, template_posn):
    template = img_gray[template_posn[0]:template_posn[0]+roi_size,template_posn[1]:template_posn[1]+roi_size]
    res = cv2.matchTemplate(img_gray, template, cv2.TM_SQDIFF_NORMED)
    simple_mask = np.where(res<=threshold, 1, 0)
    return simple_mask
```

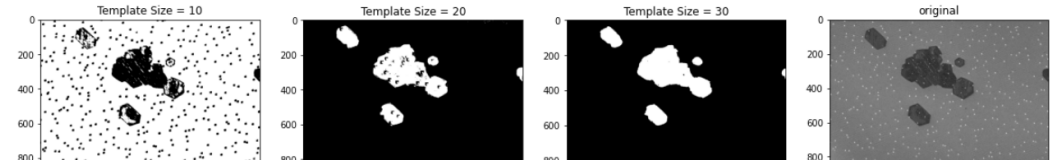
Vary the template size:

See the difference between having an template size of 10, 20 and 30 pixels.

```
In [9]: size = [10, 20, 30]
t = 0.25
posn = [275, 500]

fig, ax = plt.subplots(ncols=4, nrows=1, figsize=(20,5))
for iter, roi in enumerate(size):
    mask=temp_match(roi, t, posn)
    ax[iter].imshow(mask, cmap='gray');
    ax[iter].set_title('Template Size = %i' %roi)
ax[3].imshow(img_gray, cmap='gray')
ax[3].set_title('original')
```

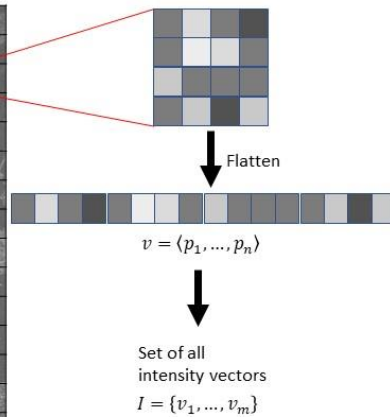
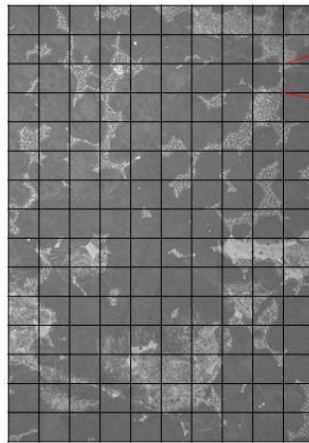
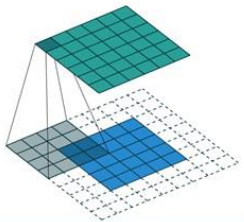
Out[9]: Text(0.5, 1.0, 'original')



Step 6: View presentation slides for template matching

K-Means: Pre-processing

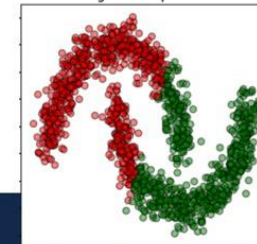
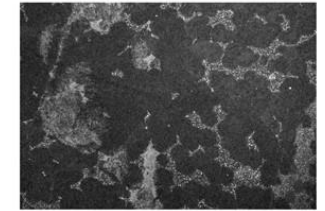
- Recall preprocessing: We divide the image into tiles, flatten them to make pixel intensity vectors and plot the vectors on high dimensional graph
- In k-means, we also control the number of pixels moved between two tiles (stride length)



I

K-Means (unsupervised clustering method)

- Advantages:
 - No need to select template or threshold
 - Fast, memory efficient
- Drawbacks:
 - Need to select number of centroids (clusters)
 - Can suffer from concave shaped blobs



I

I

Step 7: Return to the imagesegment tool on nanoHUB

- In case you closed the tool, go to the link: <https://nanohub.org/tools/imagesegment>

SEM Image Segmentation Workshop

By [Aagam Rajeev Shah](#), [Darren K Adams](#), [Mitisha Surana](#), [Ricardo Toro](#), [Sameh H Tawfick](#), [E. Ertekin](#)


This tool introduces users to machine learning used to segment microscopy images

[Edit](#)

[Launch Tool](#)

Version 1.0 - published on 12 Jan 2021

doi:10.21981/9GM4-BM53 [cite this](#)

 This tool is closed source.

[View All Supporting Documents](#)

Click on Launch Tool to begin



Step 8: Navigate to the K-Means Clustering Jupyter notebook

Machine Learning for SEM Image Segmentation in Materials Science

Scanning electron microscopy (SEM) images are typically used to observe the growth results of a synthesis experiment, such as areal coverage, nucleation density, and the shape, size, and quality of graphene domains. While the visual inspection of images can sometimes be sufficient to determine the quality of graphene, it is desirable to determine quantitative metrics as well. Quantitative metrics can provide for easier comparison between experimental results and are useful as response variables when attempting to predict optimal recipes. To calculate these metrics, we need to segment the image and each pixel needs to be classified as 'graphene' or 'not-graphene'.

The tutorials here will give you an insight into the usage of machine learning to segment microscopy images.

- **Get started:** Click on the links below to begin each tutorial.
- **Important:** To exit individual tutorials and return to this page, use `File -> Close and Halt`. "Terminate Session" (top right) will kill your entire Jupyter session.

[Template Matching:](#)

- Use template matching to segment a microscopy image of graphene on copper
- See the effects of changing the variables - ROI size, threshold and statistical technique

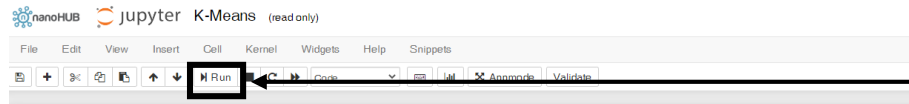
[K-Means Clustering:](#)

- Use K-Means Clustering to segment a microscopy image of graphene on copper
- See the effects of changing the variables - window size, stride and number of clusters

Click on this link



Step 9: Execute each cell in the K-Means Clustering Jupyter notebook



Click this button to run a cell or press Shift-Enter on the keyboard

Machine Learning for SEM Image Segmentation in Materials Science

Using K-Means clustering to segment microscopy images

K-Means is a relatively quick and memory efficient method to cluster images. There is no need to select any template or threshold.

In this tutorial will learn how to implement k-means clustering and use these to segment a scanning electron microscopy image of graphene on a substrate.

Outline:

1. Import image
2. Define parameters
3. Pre-process image
4. Run the method
5. Post-process the fitted data
6. See effects of varying parameters

Get started: Click "Shift-Enter" to run the code in each cell.

Let's begin

We will first import the [KMeans](#) and [MiniBatchKMeans](#) libraries, along with other relevant Python libraries.

```
In [ ]: # import all relevant libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from skimage import util
from sklearn.cluster import KMeans, MiniBatchKMeans
```

Import the image

Now, we import the image as a numpy n-dimensional array and display the image and its resolution.

Note: Slide 13 to 15 show the steps in the K-Means Clustering Jupyter notebook



Step 9.1: Import relevant packages and display the image

nanoHUB Jupyter K-Means (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run

Let's begin

We will first import the [KMeans](#) and [MiniBatchKMeans](#) libraries, along with other relevant Python libraries.

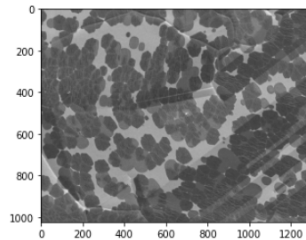
```
In [1]: # import all relevant libraries
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from skimage import util
from sklearn.cluster import KMeans, MiniBatchKMeans
```

Import the image

Now, we import the image as a numpy n-dimensional array and display the image and its resolution.

```
In [2]: img_in = np.array(Image.open('../data/test_kmeans2.tif').convert('L')) # import the grayscale image as an array
print("Resolution of Image: ", img_in.shape) # display the resolution of the image
img_plot = plt.imshow(img_in, cmap='gray') # display the image
```

Resolution of Image: (1024, 1280)



Step 9.2: Define the parameters

nanoHUB Jupyter K-Means (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Code

Define the parameters

We define all the parameters that we can change.

```
In [3]: wsize = 30 # define the window size in pixels. This is the side length of the tiles.
n_clusters = 5 # define the number of clusters. This is the "K" in "K-Means".
stride = 3 # define the stride length. This is the number of pixels moved right or down between two tiles.
seed = 197208 # define a random seed which will be used to determine the initial position of the centroids. This can be any random integer.
```

Step 9.3: Run the model and note the difference in computation time

nanoHUB jupyter K-Means (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run Code Approve Validate

Run the model

We define the standard and mini-batch k-means models and fit them to our data.

```
In [6]: 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
```

CPU times: user 12min 9s, sys: 57.4 s, total: 13min 7s
Wall time: 52.9 s

```
In [7]: mb_kmeans = MiniBatchKMeans(
        n_clusters=n_clusters,
        random_state=seed) # create a MiniBatchKMeans object which contains the method

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

# Note the difference in time between KMeans and MiniBatchKMeans
```

CPU times: user 14.4 s, sys: 2.72 s, total: 17.1 s
Wall time: 2.22 s

Step 9.4: Display the results

nanoHUB jupyter K-Means (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run Code Approve Validate

Plot the segmented images

```
In [*]: fig, ax = plt.subplots(nrows=1,ncols=3, figsize=(18,6)) # create a figure with sub-plots

# show the images segmented by KMeans and MiniBatchKMeans respectively
ax[0].imshow(clusters, cmap='gray');
ax[1].imshow(mod_mb_clusters, cmap='gray');
ax[2].imshow(img_in, cmap='gray')

# Label the images
ax[0].set_title('KMeans')
ax[1].set_title('MiniBatchKMeans')
ax[2].set_title('Original')
```

Out[10]: Text(0.5, 1.0, 'Original')



Step 9.5: See the effects of varying the parameters

nanoHUB Jupyter K-Means (read only)

File Edit View Insert Cell Kernel Widgets Help Snippets

Run Code Approve Validate

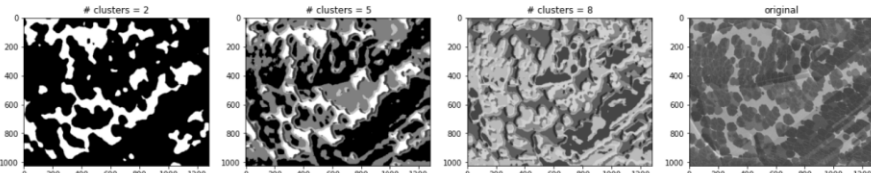
Vary the number of clusters:
See the difference between having 2, 5 and 8 clusters. Note the differences in the time taken to execute the segmentation.

```
In [*]: wsize = 30
nclusters = [2, 5, 8]
stride = 3
seed = 197208

fig, ax = plt.subplots(ncols=4, nrows=1, figsize=(20,5))
for iter, ncl in enumerate(nclusters):
    %time mb_cluster=mbmeans(ncl,seed,stride,wsize)
    ax[iter].imshow(mb_cluster, cmap='gray');
    ax[iter].set_title('# clusters = %i' %ncl)
ax[3].imshow(img_in, cmap='gray')
ax[3].set_title('original')
```

CPU times: user 11.3 s, sys: 2.06 s, total: 13.3 s
Wall time: 1.64 s
CPU times: user 12.7 s, sys: 2.29 s, total: 15 s
Wall time: 1.6 s
CPU times: user 13.6 s, sys: 2.53 s, total: 16.2 s
Wall time: 1.69 s

Out[13]: Text(0.5, 1.0, 'original')



Other resources:

- Visit <https://nanohub.org/tools/gsaimage> to use a software that performs these functions. You can upload your own image and use the same functions.
- For questions, please write to aagam2@illinois.edu

