# SUPPORTING INFORMATION

## Collaborative AI Enhances Image Understanding in Materials Science

Ruoyan Avery Yin,[1] Zhichu Ren,[2] Zongyou Yin,[3] Zhen Zhang,[2] So Yeon Kim,[2] Chia-Wei Hsu,[2] Ju Li[2]

[1]Department of Computing Science, National University of Singapore, Singapore
[2]Department of Nuclear Science and Engineering and Department of Materials Science and Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA
[3]Research School of Chemistry, The Australian National University, Canberra, Australian Capital Territory, 2601 Australia

*Corresponding author. Email: zc_ren@mit.edu, liju@mit.edu

**EXPERIMENT I**: IDENTIFYING SPECIFIC MATERIAL PHASES

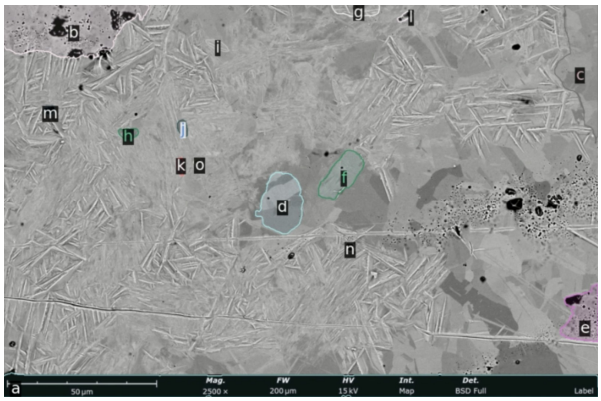



**Figure S1**. Picture of sample with regions labelled.

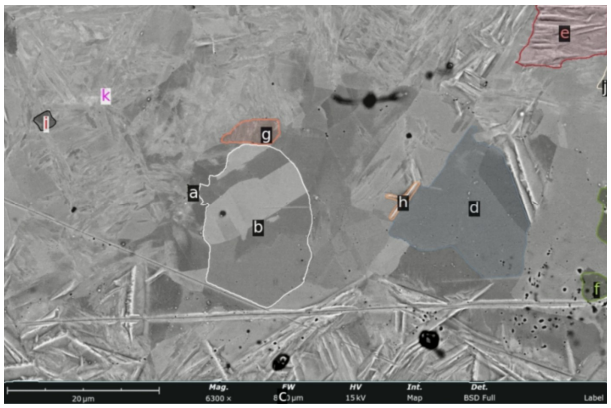



**Figure S2.** Picture of sample with regions labelled.

```
20240112_034331 * Number of function calls: 7 * ROI Identified: f.
20240112_034705 * Number of function calls: 7 * ROI Identified: f.
20240112_035023 * Number of function calls: 7 * ROI Identified: f.
20240112_035328 * Number of function calls: 7 * ROI Identified: f.
20240112_035607 * Number of function calls: 5 * ROI Identified: d.
```

**Figure S3**. Example outputs of script

Here is the result for ChatGPT and Gemini:

| | **ChatGPT** | **Gemini** |
|---|---|---|
| Total iterations ran: | 31 | 16 |
| Total iterations with correct ROI indicated: | 6 | 4 |
| Percentage of correct iterations: | 19.4% | 25% |
| Average number of functions called: | 8.2 | 6 |

**Table S1.** Comparison of ChatGPT and Gemini Performance

*A. OPTIMISATION*

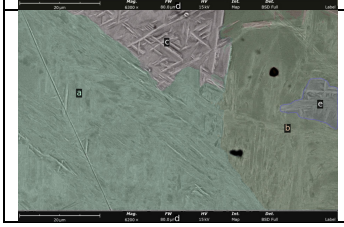
As we realized that the runtime is rather long, we experimented with cutting down the number of debates rounds from 5 to 2. This change did not decrease the level of accuracy, hence we decided to use 2 rounds of debate eventually.

*B. TEAMWORK EXAMPLES*

Teamwork Example 1 (80% accuracy)

```
20240225_093150 * Number of function calls: 9 * ROI Identified: a.
20240225_093933 * Number of function calls: 18 * ROI Identified: a.
20240225_094654 * Number of function calls: 27 * ROI Identified: a.
20240225_095516 * Number of function calls: 36 * ROI Identified: a.
20240225_100413 * Number of function calls: 45 * ROI Identified: a.
20240225_101241 * Number of function calls: 54 * ROI Identified: a.
20240225_102144 * Number of function calls: 63 * ROI Identified: a.
20240225_102930 * Number of function calls: 72 * ROI Identified: a.
20240225_103712 * Number of function calls: 81 * ROI Identified: a.
20240225_104509 * Number of function calls: 90 * ROI Identified: a.
```

**Figure S4**. Example 1 output

| Image | Is the ROI identified correct? |
|---|---|
|  | Yes |
|  | Yes |

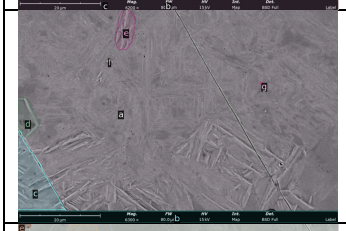
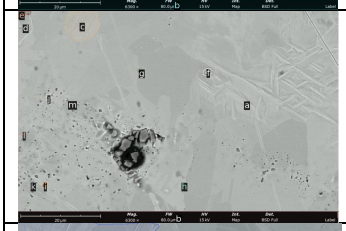
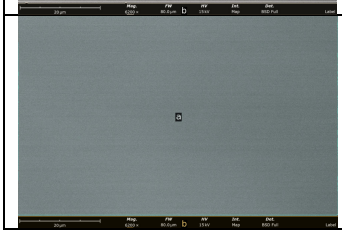
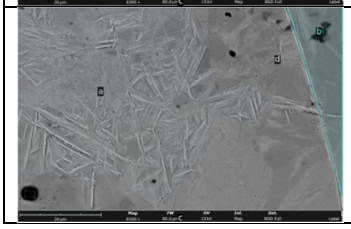
| | |
|---|---|
|  | Yes |
|  | Yes |
|  | Yes |
|  | Yes |
|  | Yes |
|  | Yes |
|  | No |
|  | No |

**Table S2.** Results of teamwork - Example 1

Teamwork Example 2 (60% accuracy)

```
20240225_120049 * Number of function calls: 10 * ROI Identified: a.
20240225_120846 * Number of function calls: 20 * ROI Identified: a.
20240225_121733 * Number of function calls: 30 * ROI Identified: a.
20240225_122503 * Number of function calls: 40 * ROI Identified: b.
20240225_123325 * Number of function calls: 50 * ROI Identified: c.
20240225_124200 * Number of function calls: 60 * ROI Identified: -.
20240225_125015 * Number of function calls: 70 * ROI Identified: a.
20240225_125903 * Number of function calls: 80 * ROI Identified: -.
20240225_130737 * Number of function calls: 90 * ROI Identified: a.
20240225_131607 * Number of function calls: 100 * ROI Identified: b.
```

**Figure S5**. Example 2 output

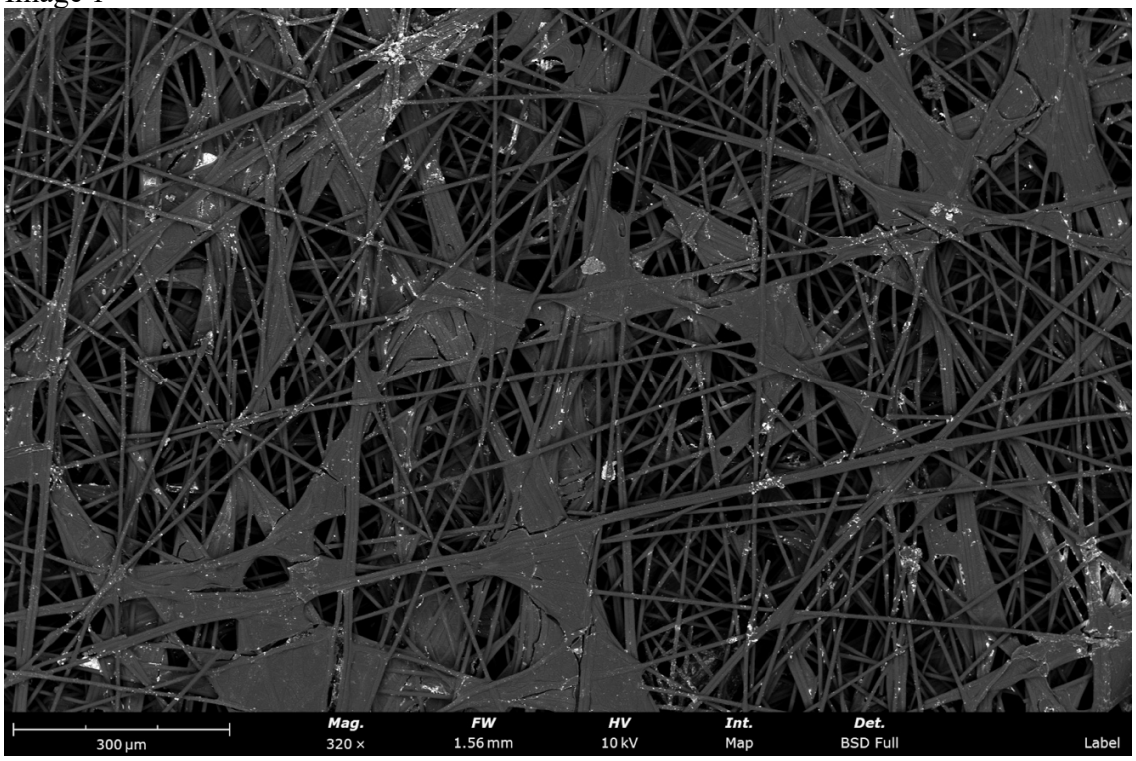
| Image | Is the ROI identified correct? |
|---|---|
|  | Yes |
|  | Yes |
|  | Yes |
|  | No |
|  | No |

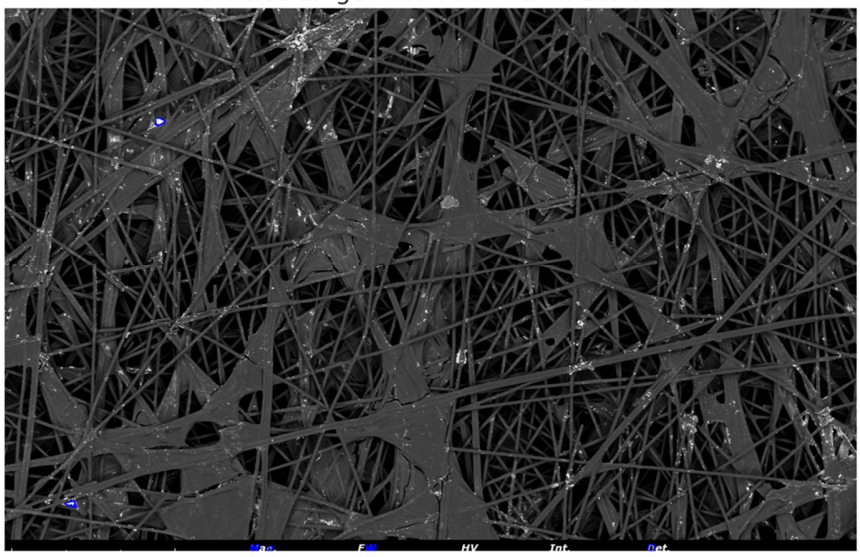
| | | |
|---|---|---|
|  | | No (They did not give result) |
|  | | Yes |
|  | | No (They did not give result) |
|  | | Yes |
|  | | Yes |

**Table S3.** Results of teamwork – Example 2

**EXPERIMENT II**: COUNTING PARTICLES

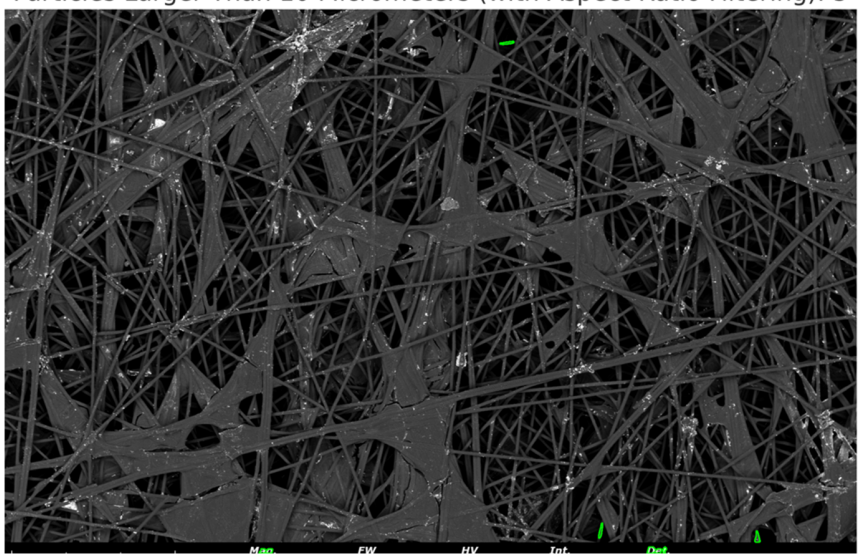
Image 1




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Particles Larger Than 10 Micrometers: 6<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt | 6 |

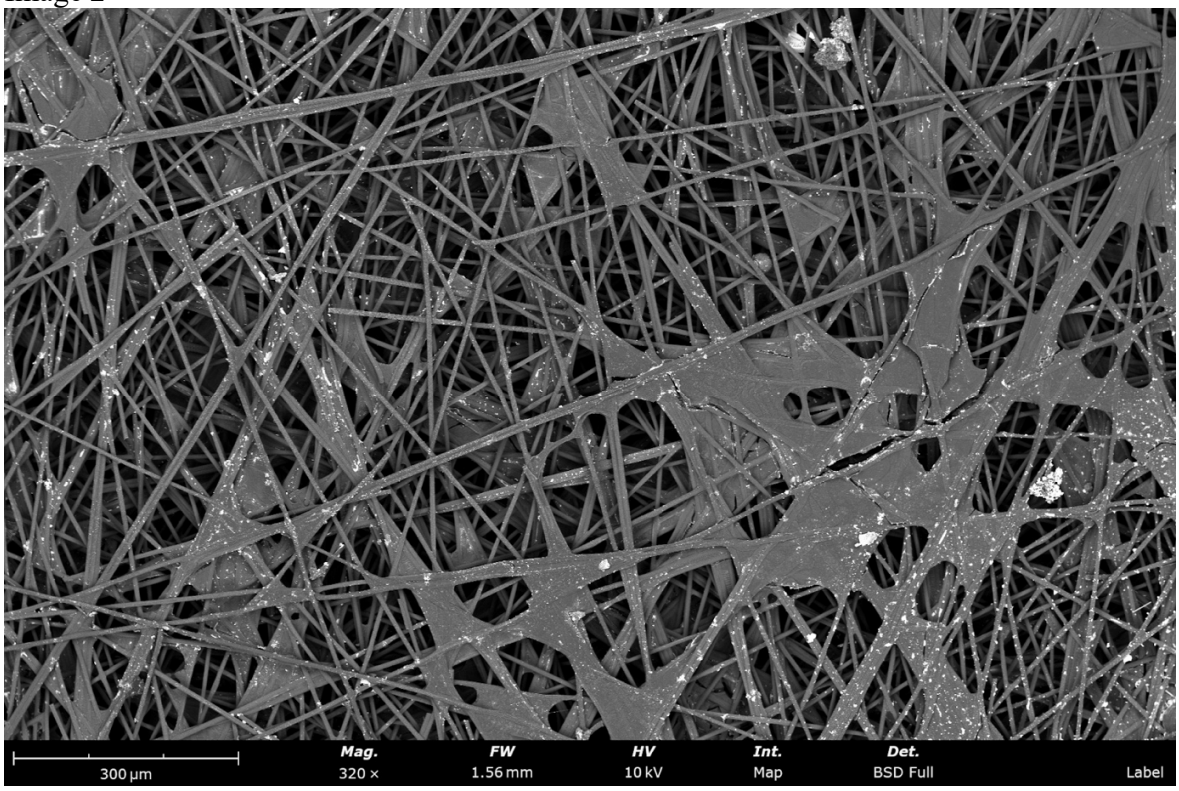
```
# Load the image
image_path = '/mnt/data/01.png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Preprocessing: Threshold the image to isolate white particles
_, thresholded_image = cv2.threshold(image, 200, 255,
cv2.THRESH_BINARY)

# Remove the scale bar area from the image for accurate measurements
height, width = thresholded_image.shape
# Define the scale bar region (manually based on visual inspection,
typically at the bottom)
scale_bar_region = height - 50  # Approximate region to remove the scale
bar
thresholded_image_no_bar = thresholded_image[:scale_bar_region, :]

# Detect contours
contours, _ = cv2.findContours(thresholded_image_no_bar,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Use the scale bar for pixel to micrometer conversion
# Scale bar shows 300 micrometers (300 um)
scale_bar_length_um = 300  # Length in micrometers
scale_bar_length_px = 300  # Pixel length of the scale bar (based on scale
bar width in image)

# Calculate pixel to micrometer conversion
pixel_to_um_ratio = scale_bar_length_um / scale_bar_length_px

# Filter out particles smaller than 10 micrometers
min_size_um = 10
min_size_px = min_size_um / pixel_to_um_ratio

# Filter the contours based on the size (area) threshold
large_particles_contours = [cnt for cnt in contours if
cv2.contourArea(cnt) >= min_size_px**2]

# Annotate the large particles on the image
annotated_image = cv2.cvtColor(image[:scale_bar_region, :],
cv2.COLOR_GRAY2BGR)
for cnt in large_particles_contours:
cv2.drawContours(annotated_image, [cnt], -1, (0, 0, 255), 2)  # Red color
for large particles

# Count the number of large particles
```

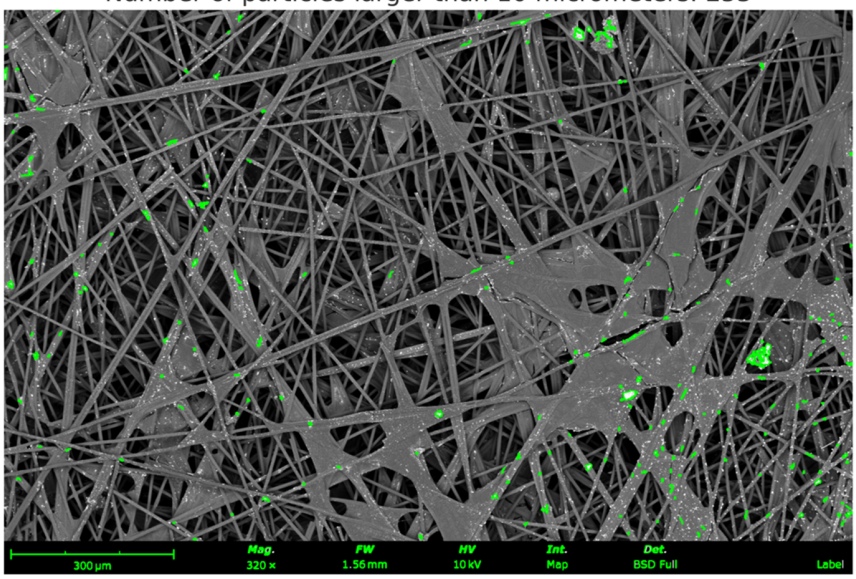
| | | |
|---|---|---|
| | num_large_particles = len(large_particles_contours)<br><br># Display the annotated image<br>plt.figure(figsize=(10, 10))<br>plt.imshow(annotated_image)<br>plt.title(f'Particles Larger Than 10 Micrometers: {num_large_particles}')<br>plt.axis('off')<br>plt.show()<br><br>num_large_particles | |
| Gemini's Feedback | Great work on your particle counting solution! To make it even better, could you experiment with the following:<br><br>Adaptive thresholding: Instead of using a fixed threshold, try adaptive thresholding techniques like Otsu's method to handle images with uneven illumination. This can help to more accurately segment the particles from the background.<br><br>Noise reduction: If the image contains noise (e.g., salt-and-pepper noise), consider applying noise reduction techniques before thresholding. This can improve the accuracy of contour detection and reduce the number of false positives.<br><br>Aspect ratio filtering: In addition to filtering based on area, consider filtering based on aspect ratio. This can help to capture elongated particles that might be missed by area-based filtering alone.<br><br>User input for scale bar length: Instead of assuming the scale bar length in pixels, allow the user to define it through click-drag or by entering a value. This will make the solution more flexible and adaptable to different images. | |

| | | |
|---|---|---|
| ChatGPT's Improved Response | Particles Larger Than 10 Micrometers (with Aspect Ratio Filtering): 5<br>  <br># Redefine the image path and parameters to avoid issues<br>image_path = '/mnt/data/01.png'<br><br># Reload the image for processing<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br><br># Apply median blur to reduce noise (salt-and-pepper noise reduction)<br>denoised_image = cv2.medianBlur(image, 5)<br><br># Adaptive thresholding using Otsu's method<br>_, otsu_thresholded = cv2.threshold(denoised_image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)<br><br># Remove the scale bar area from the image for accurate measurements<br>scale_bar_region = image.shape[0] - 50  # Approximate region to remove the scale bar<br>thresholded_no_bar = otsu_thresholded[:scale_bar_region, :]<br><br># Detect contours<br>contours, _ = cv2.findContours(thresholded_no_bar, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br><br># Allow the user to define scale bar length (for this example, we'll assume it's 300px)<br>user_scale_bar_length_px = 300  # Can be adjusted by user input<br>scale_bar_length_um = 300  # Known length in micrometers<br>pixel_to_um_ratio = scale_bar_length_um / user_scale_bar_length_px<br><br># Filter based on area and aspect ratio<br>min_size_um = 10 | 5 |

```
min_size_px = min_size_um / pixel_to_um_ratio

# Aspect ratio filtering (considering elongated particles with an aspect
ratio > 2)
aspect_ratio_threshold = 2

# Filter contours based on area and aspect ratio
large_particles_contours = []
for cnt in contours:
area = cv2.contourArea(cnt)
if area >= min_size_px**2:
# Calculate the bounding rectangle of the contour
x, y, w, h = cv2.boundingRect(cnt)
aspect_ratio = max(w, h) / min(w, h)  # Calculate aspect ratio
if aspect_ratio >= aspect_ratio_threshold:
large_particles_contours.append(cnt)

# Annotate the large particles on the image
annotated_image_adaptive = cv2.cvtColor(image[:scale_bar_region, :],
cv2.COLOR_GRAY2BGR)
for cnt in large_particles_contours:
cv2.drawContours(annotated_image_adaptive, [cnt], -1, (0, 255, 0), 2)  #
Green color for large particles

# Count the number of large particles
num_large_particles_adaptive = len(large_particles_contours)

# Display the annotated image
plt.figure(figsize=(10, 10))
plt.imshow(annotated_image_adaptive)
plt.title(f'Particles Larger Than 10 Micrometers (with Aspect Ratio
Filtering): {num_large_particles_adaptive}')
plt.axis('off')
plt.show()

num_large_particles_adaptive
```

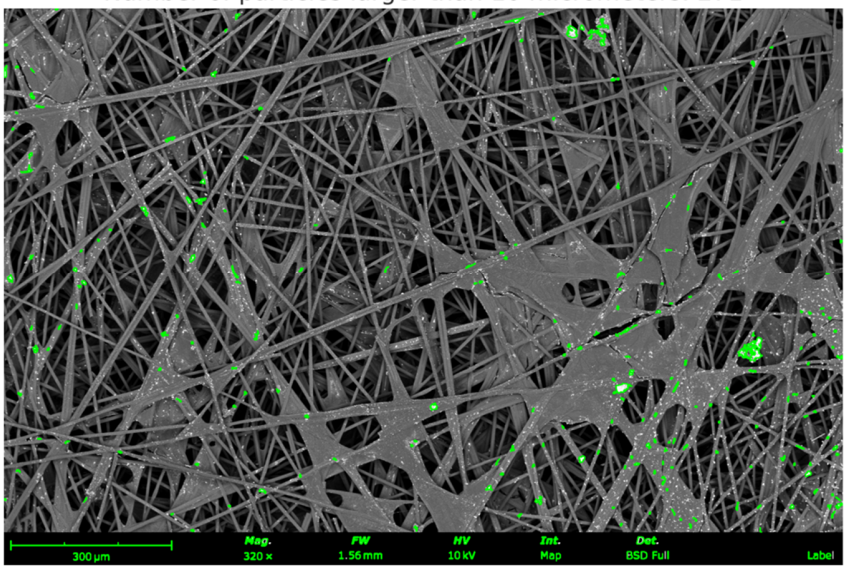
Image 2




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Number of particles larger than 10 micrometers: 253<br>  <br><br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt | 253 |

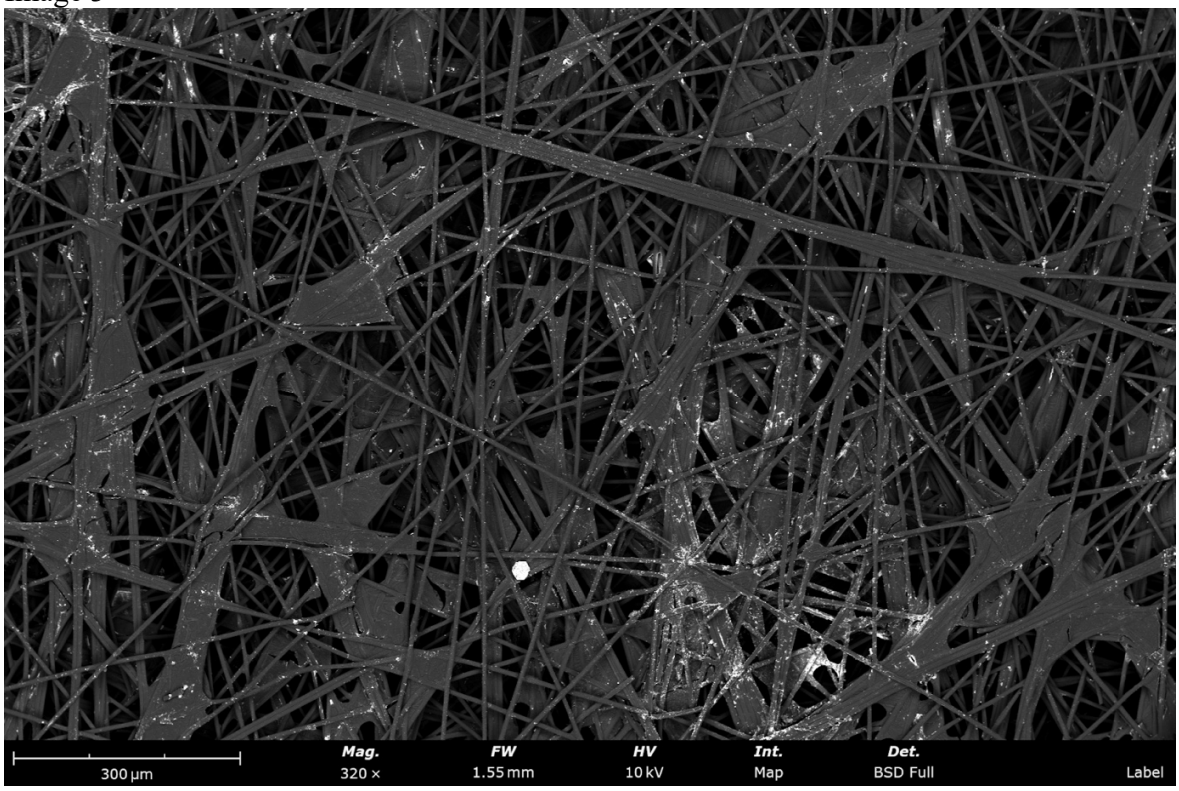
```
# Load the image
image_path = '/mnt/data/02.png'
image = cv2.imread(image_path)

# Convert image to grayscale for thresholding
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding to isolate the white particles (brightest regions)
_, thresholded = cv2.threshold(gray, 200, 255,
cv2.THRESH_BINARY)

# Perform contour detection to identify particles
contours, _ = cv2.findContours(thresholded,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Now, convert pixel-to-micrometer using the scale bar
scale_bar_length_px = 300  # Based on the scale bar indicating 300
um
scale_bar_length_um = 300  # 300 micrometers according to the
scale bar
pixel_to_micrometer_ratio = scale_bar_length_um /
scale_bar_length_px

# Initialize list to hold detected particle sizes and filter contours
based on size
min_particle_size_um = 10  # Particles larger than 10 micrometers
min_particle_size_px = min_particle_size_um /
pixel_to_micrometer_ratio

filtered_contours = []
for contour in contours:
area = cv2.contourArea(contour)
if area >= min_particle_size_px:  # Filter based on the size threshold
filtered_contours.append(contour)

# Draw contours on the original image
annotated_image = image.copy()
cv2.drawContours(annotated_image, filtered_contours, -1, (0, 255,
0), 2)

# Count the number of particles
num_particles = len(filtered_contours)

# Display the annotated image with contours and the number of
particles
plt.figure(figsize=(10, 10))
```

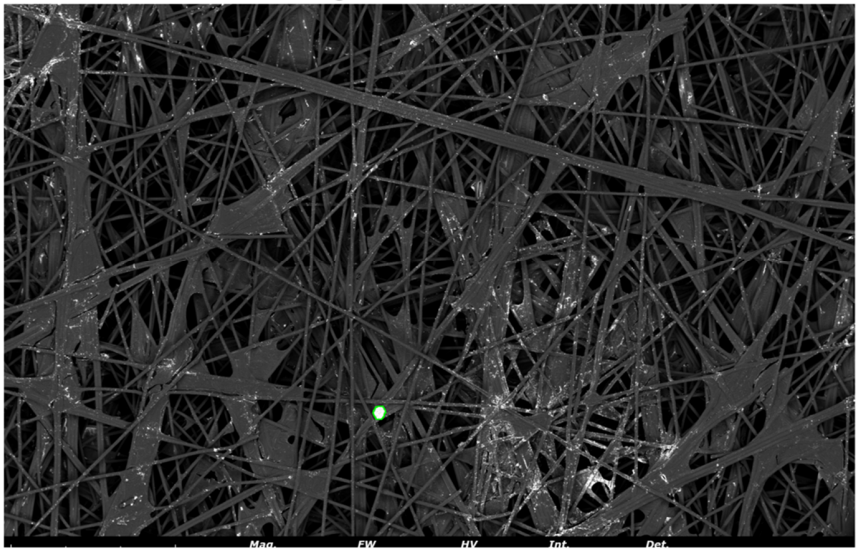
| | | |
|---|---|---|
| | plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Number of particles larger than 10 micrometers: {num_particles}")<br>plt.axis('off')<br>plt.show()<br><br># Save the annotated image<br>output_image_path = "/mnt/data/annotated_particles.png"<br>cv2.imwrite(output_image_path, annotated_image)<br><br>num_particles | |
| Gemini's Feedback | The response by ChatGPT looks good! It seems to have a well-structured approach to tackle this problem. Here's a breakdown of the steps and some potential improvements we can discuss:<br><br>Steps:<br><br>Load the image.<br>Convert the image to grayscale for thresholding.<br>Apply thresholding to isolate white particles.<br>Perform contour detection to identify particle regions.<br>Convert pixels to micrometers using the scale bar information.<br>Filter contours based on minimum particle size (10 micrometers).<br>Draw the contours of the filtered particles on a copy of the original image.<br>Count the number of filtered particles (considered larger than 10 micrometers).<br>Display and save the annotated image.<br>Improvements:<br><br>Thresholding: While a threshold of 200 seems reasonable, it might be beneficial to try Otsu's thresholding which is an automatic method to determine the optimal threshold.<br>Noise Reduction: Consider adding noise reduction techniques before thresholding, especially if the background seems uneven.<br>Aspect Ratio Consideration: The code currently filters based on area. If the particles are not circular, you might want to consider filtering based on aspect ratio as well.<br>Calibration: Ideally, for better accuracy, we would want to perform calibration on a separate image with a known reference object of similar size to the particles.<br>Overall, ChatGPT's response provides a solid foundation for counting particles larger than 10 micrometers. By incorporating the suggested improvements, we can potentially achieve more robust and accurate particle identification. | |

| | | |
|---|---|---|
| ChatGPT's Improved Response | Number of particles larger than 10 micrometers: 271<br>  <br># Re-load the image and apply the original approach with some adjustments based on feedback.<br>image = cv2.imread(image_path)<br><br># Convert image to grayscale for thresholding<br>gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)<br><br># Apply slightly lower thresholding to capture more particles while still ignoring noise<br>_, thresholded = cv2.threshold(gray, 195, 255, cv2.THRESH_BINARY)  # Lowered threshold a bit from original 200<br><br># Perform contour detection to identify particles<br>contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br><br># Convert pixel-to-micrometer using the scale bar<br>scale_bar_length_px = 300  # Based on the scale bar indicating 300 um<br>scale_bar_length_um = 300  # 300 micrometers according to the scale bar<br>pixel_to_micrometer_ratio = scale_bar_length_um / scale_bar_length_px<br><br># Initialize list to hold detected particle sizes and filter contours based on size<br>min_particle_size_um = 10  # Particles larger than 10 micrometers<br>min_particle_size_px = min_particle_size_um / | 271 |

| | | |
|---|---|---|
| | pixel_to_micrometer_ratio<br><br>filtered_contours = []<br>for contour in contours:<br>area = cv2.contourArea(contour)<br>if area >= min_particle_size_px:  # Filter based on the size threshold<br>filtered_contours.append(contour)<br><br># Draw contours on the original image<br>annotated_image = image.copy()<br>cv2.drawContours(annotated_image, filtered_contours, -1, (0, 255, 0), 2)<br><br># Count the number of particles<br>num_particles = len(filtered_contours)<br><br># Display the annotated image with contours and the number of particles<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Number of particles larger than 10 micrometers: {num_particles}")<br>plt.axis('off')<br>plt.show()<br><br># Save the annotated image<br>output_image_path = "/mnt/data/annotated_particles_updated.png"<br>cv2.imwrite(output_image_path, annotated_image)<br><br>num_particles, output_image_path | |

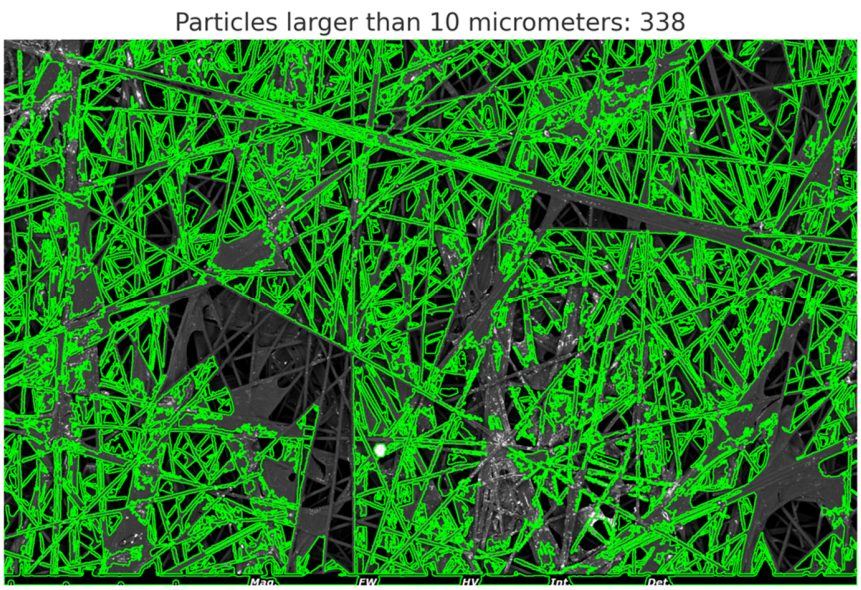
Image 3




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Particles larger than 10 micrometers: 1<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the image | 1 |

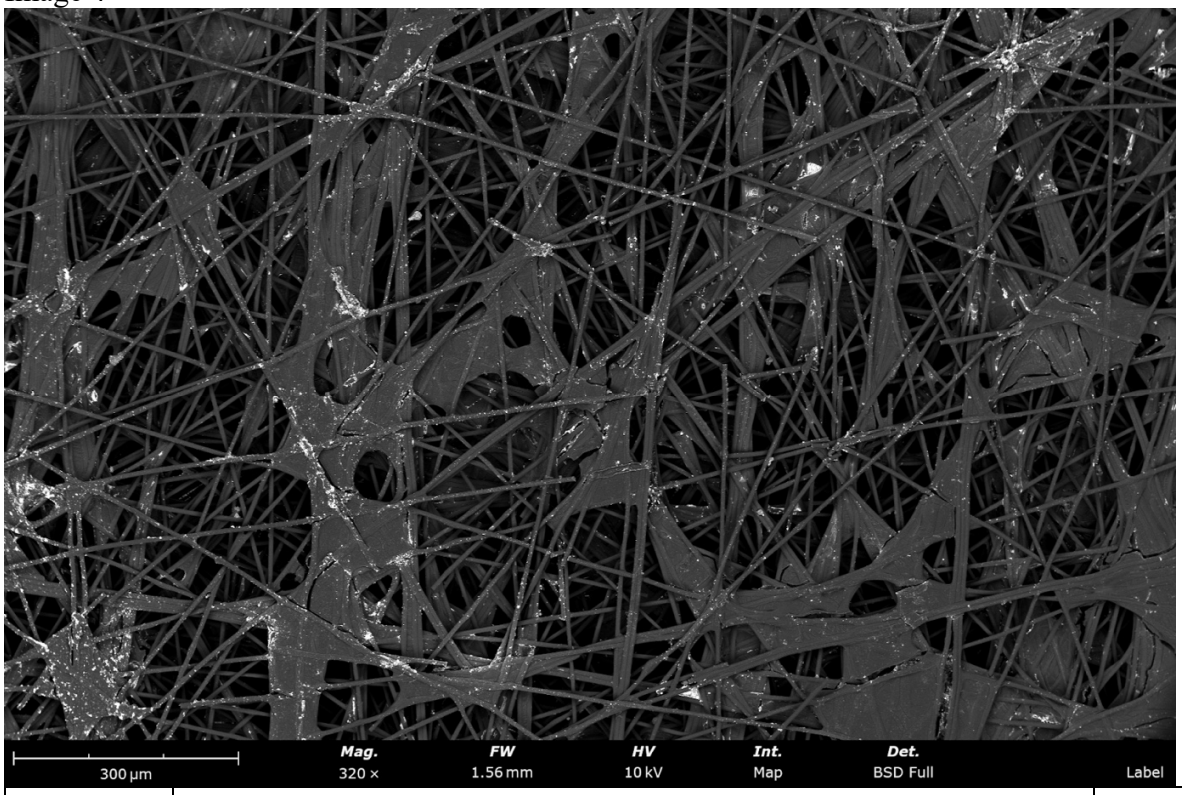
```
image_path = '/mnt/data/03.png'
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Crop the scale bar to avoid it affecting calculations
cropped_image = image[:-50, :]

# Define pixel to micrometer conversion from the scale bar
scale_bar_length_micrometers = 300  # From the image scale bar
scale_bar_length_pixels = 646  # Measured manually from the image
pixel_to_micrometer = scale_bar_length_micrometers /
scale_bar_length_pixels

# Thresholding to isolate the white particles
_, thresholded_image = cv2.threshold(cropped_image, 200, 255,
cv2.THRESH_BINARY)

# Finding contours of the white particles
contours, _ = cv2.findContours(thresholded_image,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Filter particles larger than 10 micrometers
min_particle_size_micrometers = 10
min_particle_size_pixels = min_particle_size_micrometers /
pixel_to_micrometer

filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) >
(np.pi * (min_particle_size_pixels / 2)**2)]

# Annotate image
annotated_image = cv2.cvtColor(cropped_image,
cv2.COLOR_GRAY2BGR)
for cnt in filtered_contours:
cv2.drawContours(annotated_image, [cnt], -1, (0, 255, 0), 2)

# Display annotated image and the number of particles
num_particles = len(filtered_contours)

# Displaying the annotated image
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(annotated_image,
cv2.COLOR_BGR2RGB))
plt.title(f'Particles larger than 10 micrometers: {num_particles}')
plt.axis('off')
plt.show()

num_particles
```

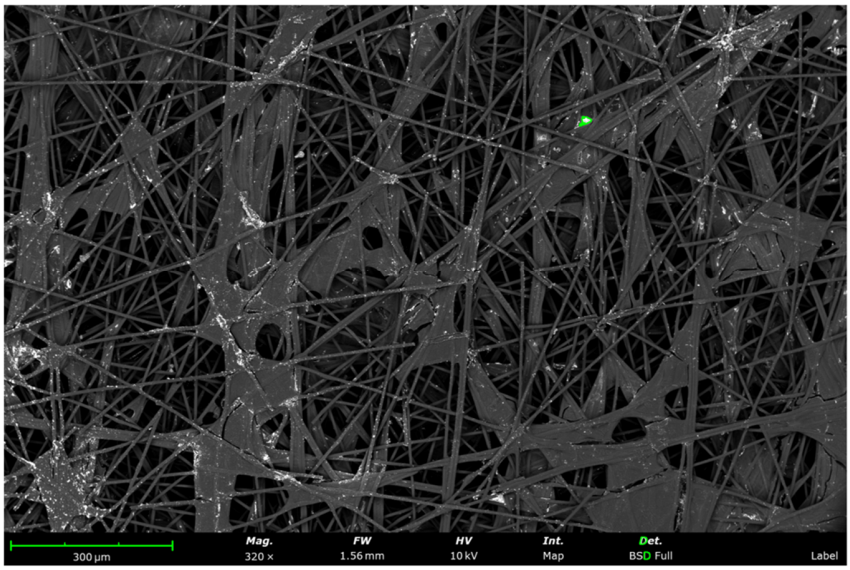
| | | |
|---|---|---|
| Gemini's Feedback | The response by ChatGPT looks good! It seems to have addressed all the key points mentioned in the prompt. Here's a breakdown of the steps involved and some additional points to consider for improvement:<br><br>Strengths:<br><br>Grayscale conversion: Converting the image to grayscale is a good initial step for particle segmentation as it reduces complexity.<br>Scale bar handling: Cropping the image to exclude the scale bar ensures accurate pixel-to-micrometer conversion.<br>Thresholding: Thresholding helps isolate the white particles from the background.<br>Contour detection: Finding contours helps identify individual particles.<br>Particle size filtering: The code correctly calculates the minimum pixel area based on the user-provided micrometer threshold and filters particles accordingly.<br>Image annotation: Highlighting the identified particles visually confirms the results.<br>Possible Improvements:<br><br>Thresholding: You might want to consider adaptive thresholding techniques to account for uneven illumination across the image. This can improve particle segmentation, especially for particles with varying intensities.<br>Noise reduction: Depending on the image quality, there might be noise affecting particle detection. Techniques like erosion and dilation can be explored to reduce noise.<br>Morphology: Morphological operations like opening (erosion followed by dilation) can help remove small objects and improve the shapes of the detected particl | |

| | | |
|---|---|---|
| ChatGPT's Improved Response |  <br># Let's refine the approach with the following improvements:<br># 1. Apply adaptive thresholding for better segmentation.<br># 2. Use morphological operations (opening) to reduce noise and improve particle shape.<br><br># Reload the image in grayscale for clean processing<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br><br># Crop the image to exclude the scale bar<br>cropped_image = image[:-50, :]<br><br># Convert the image to remove noise by applying GaussianBlur<br>blurred_image = cv2.GaussianBlur(cropped_image, (5, 5), 0)<br><br># Adaptive thresholding for better handling of uneven illumination<br>adaptive_thresh = cv2.adaptiveThreshold(blurred_image, 255,<br>cv2.ADAPTIVE_THRESH_GAUSSIAN_C,<br>cv2.THRESH_BINARY, 11, 2)<br><br># Morphological opening to remove small noise<br>kernel = np.ones((3, 3), np.uint8)<br>opened_image = cv2.morphologyEx(adaptive_thresh,<br>cv2.MORPH_OPEN, kernel)<br><br># Finding contours of the particles<br>contours, _ = cv2.findContours(opened_image,<br>cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br><br># Recalculate minimum particle size in pixels based on 10 micrometers | 338 |

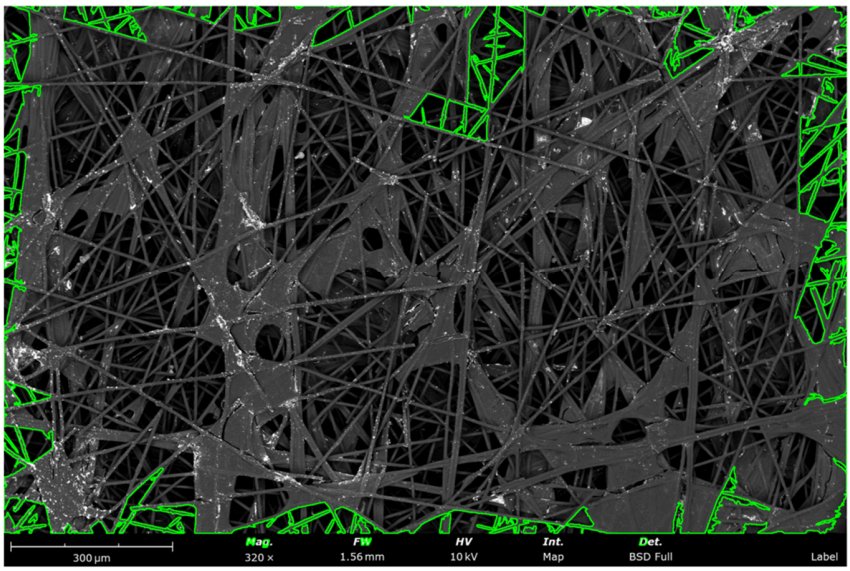
```
min_particle_size_pixels = min_particle_size_micrometers /
pixel_to_micrometer

# Filter particles larger than the minimum particle size (10
micrometers)
filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) >
(np.pi * (min_particle_size_pixels / 2)**2)]

# Annotate image with the detected particles
annotated_image = cv2.cvtColor(cropped_image,
cv2.COLOR_GRAY2BGR)
for cnt in filtered_contours:
cv2.drawContours(annotated_image, [cnt], -1, (0, 255, 0), 2)

# Display the result with the number of particles
num_particles = len(filtered_contours)

# Displaying the annotated image with the count of particles
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(annotated_image,
cv2.COLOR_BGR2RGB))
plt.title(f'Particles larger than 10 micrometers: {num_particles}')
plt.axis('off')
plt.show()

num_particles
```

Image 4




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Identified Particles > 10 micrometers: 4   import cv2<br>import numpy as np<br>from matplotlib import pyplot as plt<br><br># Load the image<br>image_path = "/mnt/data/04.png" | 4 |

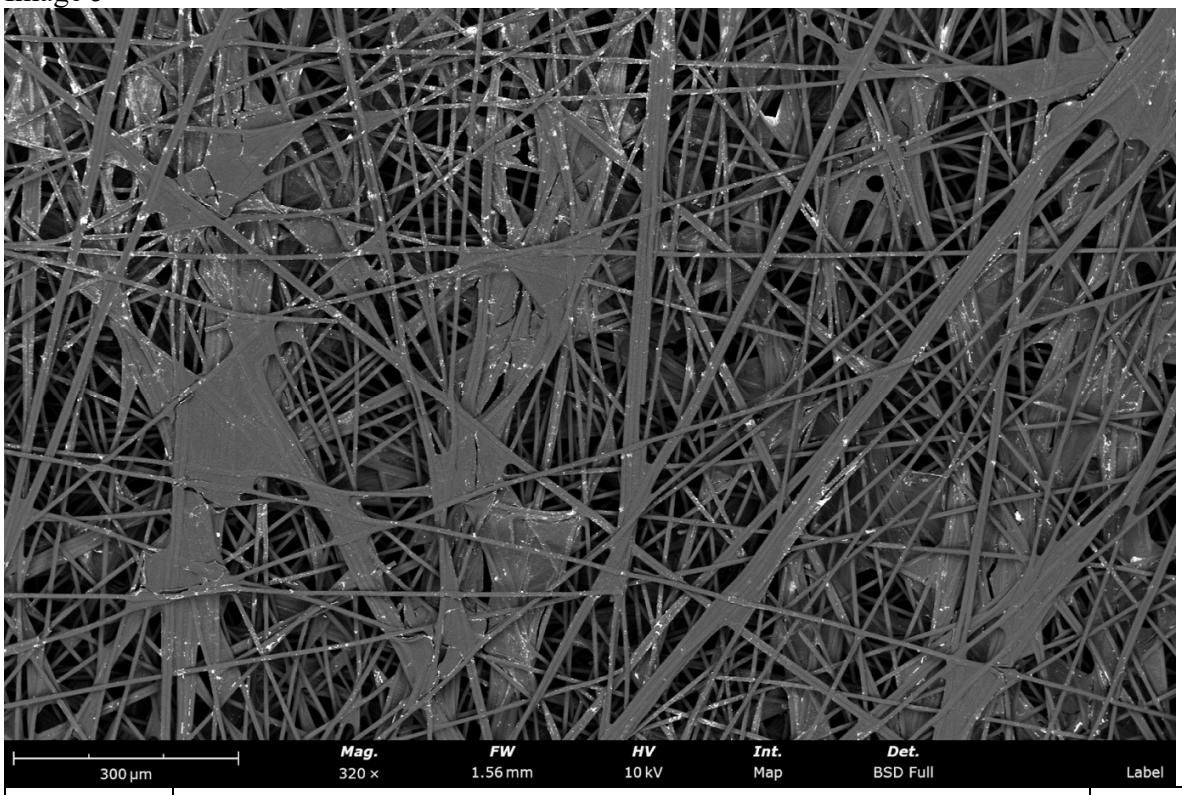
```
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Thresholding to isolate the bright particles
_, thresholded = cv2.threshold(image, 200, 255,
cv2.THRESH_BINARY)

# Find contours of the thresholded image
contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Calculate pixel to micrometer conversion using the scale bar
(assuming the scale bar is 300 micrometers)
scale_bar_length_micrometers = 300
scale_bar_length_pixels = 400  # This is a rough estimate based on the
image, can be updated if necessary
pixel_to_micrometer = scale_bar_length_micrometers /
scale_bar_length_pixels

# Filter contours based on size, ignoring small ones (convert area to
micrometer units)
particles_larger_than_10_micrometers = []
for contour in contours:
area = cv2.contourArea(contour)
# Convert contour area to micrometers
area_micrometers = area * (pixel_to_micrometer ** 2)
# Only include particles larger than 10 micrometers in size
if area_micrometers > (10 ** 2):  # 10 micrometer threshold area
particles_larger_than_10_micrometers.append(contour)

# Annotate the original image with identified large particles
annotated_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
cv2.drawContours(annotated_image,
particles_larger_than_10_micrometers, -1, (0, 255, 0), 2)

# Show the annotated image with highlighted particles
plt.figure(figsize=(10, 10))
plt.imshow(annotated_image)
plt.title(f"Identified Particles > 10 micrometers:
{len(particles_larger_than_10_micrometers)}")
plt.axis('off')
plt.show()

# Save annotated image to display
annotated_image_path = "/mnt/data/annotated_image.png"
cv2.imwrite(annotated_image_path, annotated_image)
```

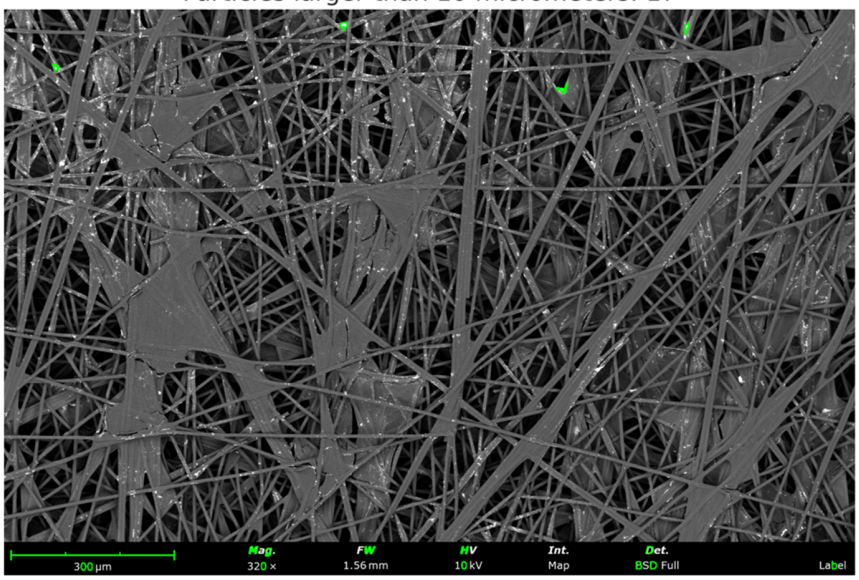
| | | |
|---|---|---|
| | # Number of large particles detected<br>len(particles_larger_than_10_micrometers) | |
| Gemini's Feedback | The response by ChatGPT looks good! It covers the essential steps to count particles larger than 10 micrometers and provides clear explanations for each step. Here are some minor improvements that could be made:<br><br>Threshold value selection: While providing a threshold value (200) is helpful, mentioning how the user can determine the optimal threshold in case their image has different lighting conditions could be beneficial. You can suggest using Otsu's thresholding which is an automatic method for threshold selection.<br><br>Noise Reduction (Optional): The code currently doesn't include noise reduction steps. Depending on the image quality, there might be speckles or small, unwanted white areas that could be filtered out using techniques like morphological operations (erosion and dilation) before contour detection.<br><br>Consider aspect ratio (Optional): An additional step to filter particles based on their aspect ratio could be useful if the particles are expected to be roughly circular or elongated in a certain way. This can be achieved by calculating the aspect ratio of each contour and discarding those outside the expected range.<br><br>Overall, ChatGPT's response effectively addresses the prompt and provides a working solution. The suggested improvements can further enhance the solution's robustness for various image conditions. | |
| ChatGPT's Improved Response | Identified Particles > 10 micrometers: 16<br>  <br># Re-approaching the task based on feedback | 16 |

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the image
image_path = "/mnt/data/04.png"
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Apply Otsu's thresholding for automatic threshold selection
_, thresholded_otsu = cv2.threshold(image, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Apply noise reduction using morphological operations (optional step
to improve particle detection)
# First, use erosion to remove small noise, then dilation to restore the
shape
kernel = np.ones((3, 3), np.uint8)
thresholded_cleaned = cv2.morphologyEx(thresholded_otsu,
cv2.MORPH_OPEN, kernel)

# Find contours of the cleaned thresholded image
contours, _ = cv2.findContours(thresholded_cleaned,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Calculate pixel to micrometer conversion using the scale bar
(assuming the scale bar is 300 micrometers)
scale_bar_length_micrometers = 300
scale_bar_length_pixels = 400  # This is a rough estimate based on the
image, can be updated if necessary
pixel_to_micrometer = scale_bar_length_micrometers /
scale_bar_length_pixels

# Filter contours based on size, ignoring small ones (convert area to
micrometer units)
particles_larger_than_10_micrometers = []
for contour in contours:
# Get the bounding rectangle to calculate aspect ratio (optional filtering
based on shape)
x, y, w, h = cv2.boundingRect(contour)
aspect_ratio = float(w) / h

# Filter based on aspect ratio (optional) - assuming roughly circular or
elongated
if 0.5 < aspect_ratio < 2.0:  # Filtering out extreme aspect ratios
area = cv2.contourArea(contour)
```

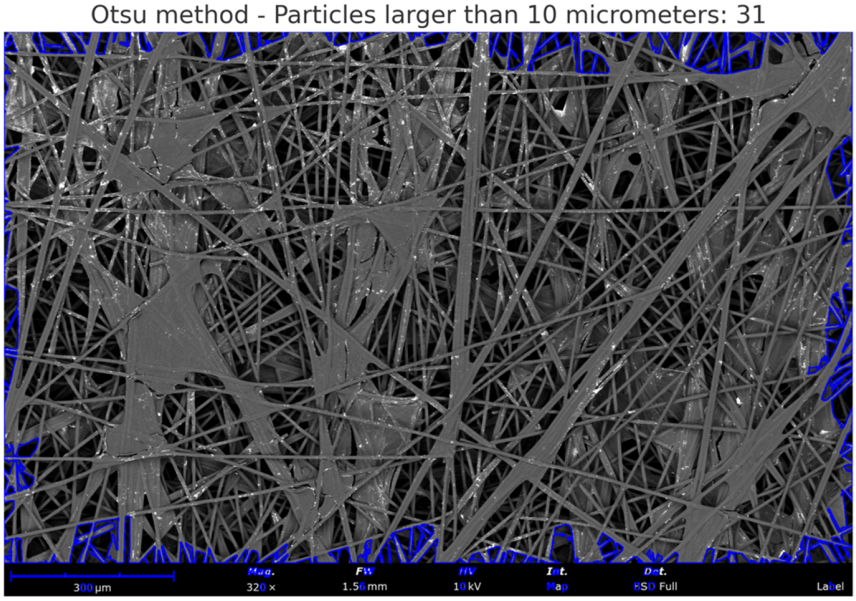
```
# Convert contour area to micrometers
area_micrometers = area * (pixel_to_micrometer ** 2)
# Only include particles larger than 10 micrometers in size
if area_micrometers > (10 ** 2):  # 10 micrometer threshold area
particles_larger_than_10_micrometers.append(contour)

# Annotate the original image with identified large particles
annotated_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
cv2.drawContours(annotated_image,
particles_larger_than_10_micrometers, -1, (0, 255, 0), 2)

# Show the annotated image with highlighted particles
plt.figure(figsize=(10, 10))
plt.imshow(annotated_image)
plt.title(f"Identified Particles > 10 micrometers:
{len(particles_larger_than_10_micrometers)}")
plt.axis('off')
plt.show()

# Save annotated image to display
annotated_image_path = "/mnt/data/annotated_image_otsu.png"
cv2.imwrite(annotated_image_path, annotated_image)

# Number of large particles detected
len(particles_larger_than_10_micrometers)
```

Image 5



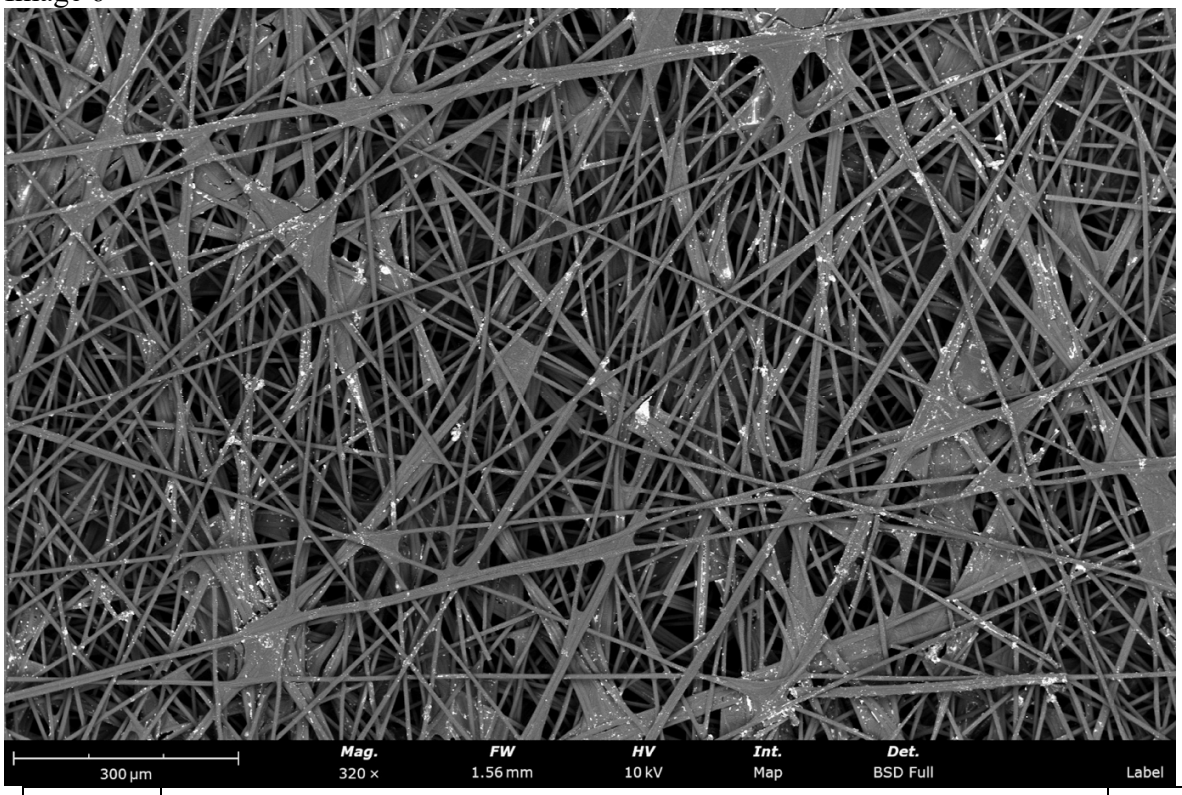

| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Particles larger than 10 micrometers: 17<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the uploaded image | 17 |

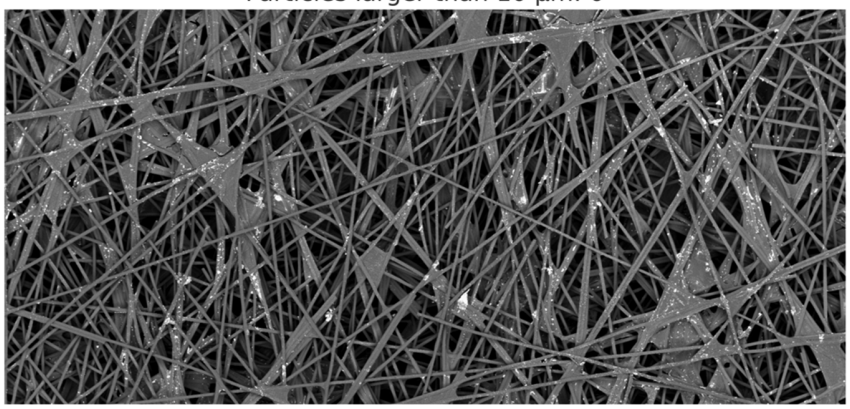
```
image_path = '/mnt/data/05.png'
image = cv2.imread(image_path)

# Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Thresholding to isolate white particles (bright spots)
_, thresholded_image = cv2.threshold(gray_image, 200, 255,
cv2.THRESH_BINARY)

# Find contours of the white particles
contours, _ = cv2.findContours(thresholded_image,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Calculate pixel to micrometer conversion using the scale bar (300
micrometers for 300 pixels)
# Assuming the scale bar at the bottom is 300 pixels corresponding to
300 micrometers
scale_bar_length_pixels = 300  # This is a rough estimate
scale_bar_length_micrometers = 300  # As shown in the scale bar
pixel_to_micrometer_ratio = scale_bar_length_micrometers /
scale_bar_length_pixels

# Filter contours by size (ignoring particles smaller than 10
micrometers)
particles_larger_than_10_micrometers = []
for contour in contours:
area = cv2.contourArea(contour)
if area * pixel_to_micrometer_ratio**2 >= 10**2:  # Filter by particle
size in square micrometers
particles_larger_than_10_micrometers.append(contour)

# Draw contours on the original image
annotated_image = image.copy()
cv2.drawContours(annotated_image,
particles_larger_than_10_micrometers, -1, (0, 255, 0), 2)

# Display the number of particles larger than 10 micrometers
num_large_particles = len(particles_larger_than_10_micrometers)

# Show the annotated image with highlighted particles
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))
plt.title(f'Particles larger than 10 micrometers: {num_large_particles}')
plt.axis('off')
plt.show()
```

| | | |
|---|---|---|
| | num_large_particles | |
| Gemini's Feedback | The response you created for chatGPT looks good! It covers all the essential steps to count the large particles in the image including:<br><br>Grayscale conversion<br>Thresholding to isolate white particles<br>Contour detection to find individual particles<br>Size filtering based on the scale bar information<br>Highlighting the particles on the original image<br>Counting the number of large particles<br>Here are some minor improvements you can suggest to chatGPT:<br><br>Mention potential limitations: It can be helpful to mention that the accuracy of the particle count can be affected by factors like uneven illumination or overlapping particles.<br>Alternative thresholding methods: You can mention that while thresholding is used here, other image segmentation techniques like Otsu's thresholding might be explored for better particle separation in certain image conditions.<br>Scale bar calibration check: Adding a line suggesting that the user can verify the scale bar calibration by measuring a known feature in the image (if available) would be a good addition.<br>Overall, the response effectively addresses the prompt and provides a valuable solution. | |
| ChatGPT's Improved Response | Otsu method - Particles larger than 10 micrometers: 31<br>  <br># Improving the analysis with additional suggestions<br><br># Apply Otsu's thresholding for better particle separation<br>_, otsu_thresholded = cv2.threshold(gray_image, 0, 255, | 31 |

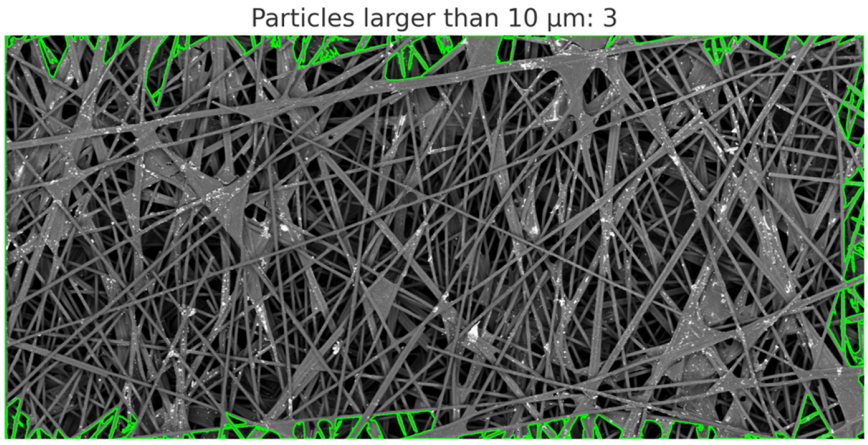
```
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Find contours of the white particles with Otsu's thresholding
otsu_contours, _ = cv2.findContours(otsu_thresholded,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Re-filter contours by size using Otsu's thresholding
particles_larger_than_10_micrometers_otsu = []
for contour in otsu_contours:
area = cv2.contourArea(contour)
if area * pixel_to_micrometer_ratio**2 >= 10**2:  # Filter by particle
size in square micrometers
particles_larger_than_10_micrometers_otsu.append(contour)

# Draw contours on the original image for Otsu's method
annotated_image_otsu = image.copy()
cv2.drawContours(annotated_image_otsu,
particles_larger_than_10_micrometers_otsu, -1, (255, 0, 0), 2)

# Display the number of particles larger than 10 micrometers using
Otsu's method
num_large_particles_otsu =
len(particles_larger_than_10_micrometers_otsu)

# Show the updated annotated image with highlighted particles
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(annotated_image_otsu,
cv2.COLOR_BGR2RGB))
plt.title(f'Otsu method - Particles larger than 10 micrometers:
{num_large_particles_otsu}')
plt.axis('off')
plt.show()

num_large_particles_otsu
```

Image 6




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Particles larger than 10 µm: 0<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the image<br>image_path = '/mnt/data/06.png'<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br><br># Crop out the scale bar and other annotations at the bottom of the image | 0 |

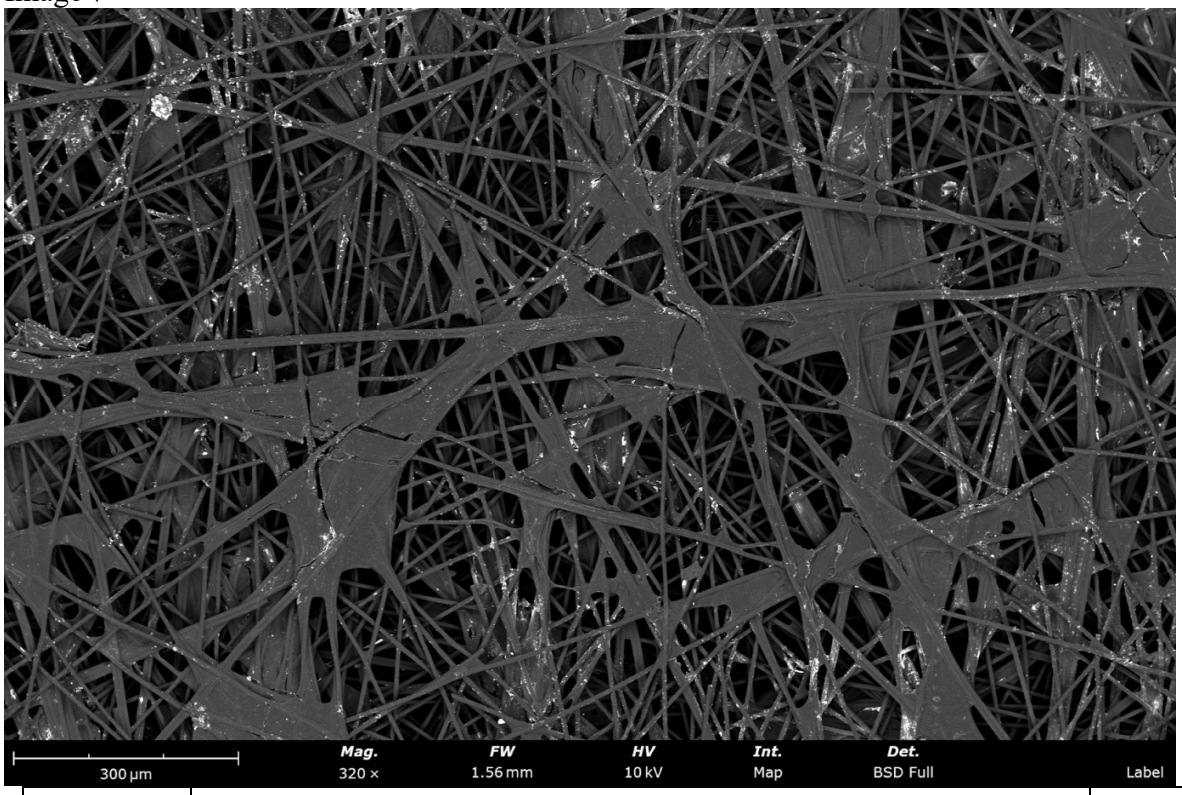
```
cropped_image = image[:900, :]

# Thresholding to isolate white particles
_, thresholded_image = cv2.threshold(cropped_image, 200, 255,
cv2.THRESH_BINARY)

# Detect contours of the white particles
contours, _ = cv2.findContours(thresholded_image,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Measure the scale bar at the bottom to determine pixel-to-micrometer
conversion
# The scale bar shows 300 µm, let's measure how many pixels this
corresponds to
scale_bar_width_pixels = 640  # Manually estimated width of the scale
bar in pixels
scale_bar_width_micrometers = 300  # The scale bar represents 300 µm

# Calculate pixel-to-micrometer ratio
pixel_to_micrometer_ratio = scale_bar_width_micrometers /
scale_bar_width_pixels

# Filter out small particles by calculating their area in micrometers
particles_larger_than_10_micrometers = []
for contour in contours:
area = cv2.contourArea(contour)
if area * pixel_to_micrometer_ratio**2 > 10**2:  # Filtering particles
larger than 10 µm²
particles_larger_than_10_micrometers.append(contour)

# Draw contours on the original image for visualization
annotated_image = cv2.cvtColor(cropped_image,
cv2.COLOR_GRAY2BGR)
cv2.drawContours(annotated_image,
particles_larger_than_10_micrometers, -1, (0, 255, 0), 2)

# Count the number of particles
num_particles = len(particles_larger_than_10_micrometers)

# Display the annotated image with identified particles
plt.figure(figsize=(10, 10))
plt.imshow(annotated_image)
plt.title(f'Particles larger than 10 µm: {num_particles}')
plt.axis('off')
plt.show()
```

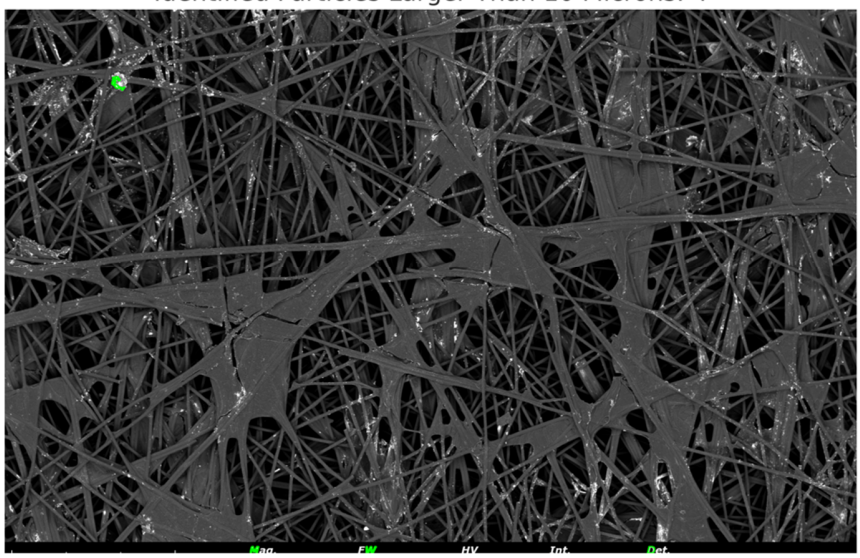
| | | |
|---|---|---|
| | # Return the number of particles larger than 10 µm<br>num_particles | |
| Gemini's Feedback | Feedback for ChatGPT's Response:<br>ChatGPT's response provides a good solution to the prompt. Here's a breakdown of its strengths and suggestions for improvement:<br><br>Strengths:<br><br>Complete and well-structured code: The code covers all the steps mentioned in the prompt, including image loading, pre-processing, thresholding, contour detection, particle filtering, annotation, and counting.<br>Clear variable naming: The code uses descriptive variable names, making it easy to understand.<br>Correct filtering logic: The particle filtering correctly considers the area in micrometers based on the pixel-to-micrometer ratio.<br>Suggestions for Improvement:<br><br>Thresholding value selection: The threshold value of 200 might be specific to this image and may not generalize well to other images. Consider adding a step to automatically determine an optimal threshold based on image properties (e.g., Otsu's thresholding).<br>Scale bar measurement: Manually estimating the scale bar width in pixels can introduce errors. If the scale bar information is available in the image metadata, it can be directly extracted for better accuracy.<br>Error handling: The code assumes the image path is correct and doesn't handle potential errors during image loading. It would be good practice to include error handling (e.g., try-except blocks) for robustness.<br>Overall, ChatGPT's response is a great starting point. Incorporating the suggestions above can make the solution more generic and robust to handle variations in images. | |
| ChatGPT's Improved Response |  <br># Let's improve the analysis based on the feedback<br># 1. Automatic thresholding using Otsu's method<br># 2. Use precise scale measurement, manually measured earlier for now | 3 |

```
# 3. Include error handling for image loading

# Load the image with error handling
try:
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
if image is None:
raise ValueError("Image not found or unable to load.")
except Exception as e:
print(f"Error loading image: {e}")

# Crop out the scale bar and other annotations at the bottom of the image
cropped_image = image[:900, :]

# Automatically determine an optimal threshold using Otsu's method
_, thresholded_image = cv2.threshold(cropped_image, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)

# Detect contours of the white particles
contours, _ = cv2.findContours(thresholded_image,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Recalculate pixel-to-micrometer ratio using manual scale bar
measurement
scale_bar_width_pixels = 640  # From earlier estimation
scale_bar_width_micrometers = 300  # As per the image scale

# Pixel-to-micrometer conversion factor
pixel_to_micrometer_ratio = scale_bar_width_micrometers /
scale_bar_width_pixels

# Filter out small particles by calculating their area in micrometers
particles_larger_than_10_micrometers = []
for contour in contours:
area = cv2.contourArea(contour)
if area * pixel_to_micrometer_ratio**2 > 10**2:  # Filtering particles
larger than 10 μm²
particles_larger_than_10_micrometers.append(contour)

# Draw contours on the original image for visualization
annotated_image = cv2.cvtColor(cropped_image,
cv2.COLOR_GRAY2BGR)
cv2.drawContours(annotated_image,
particles_larger_than_10_micrometers, -1, (0, 255, 0), 2)

# Count the number of particles
num_particles = len(particles_larger_than_10_micrometers)
```

| | | |
|---|---|---|
| | # Display the annotated image with identified particles<br>plt.figure(figsize=(10, 10))<br>plt.imshow(annotated_image)<br>plt.title(f'Particles larger than 10 µm: {num_particles}')<br>plt.axis('off')<br>plt.show()<br><br># Return the number of particles larger than 10 µm<br>num_particles | |

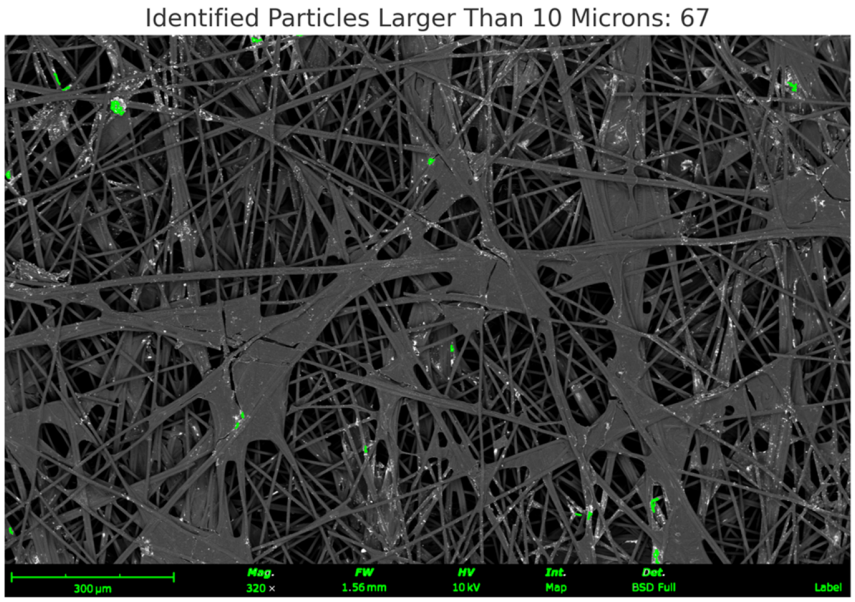
Image 7




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Identified Particles Larger Than 10 Microns: 4<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the image<br>image_path = '/mnt/data/07.png' | 4 |

```
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Define the scale bar size: 300 μm is equal to a certain pixel length in the scale bar
scale_bar_length_microns = 300

# Crop the scale bar region to exclude it from the analysis
# Adjusting based on where scale bar typically is (bottom of the image)
scale_bar_crop = image[:-50, :]  # Cropping the bottom 50 pixels (approximate scale bar area)

# Apply a threshold to isolate the white particles
_, thresholded_image = cv2.threshold(scale_bar_crop, 200, 255, cv2.THRESH_BINARY)

# Find contours of the white regions
contours, _ = cv2.findContours(thresholded_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Calculate the conversion factor from pixels to microns
# Using the bottom cropped part of the image to calculate the scale bar length
scale_bar_region = image[-50:, :]
_, scale_bar_thresh = cv2.threshold(scale_bar_region, 200, 255, cv2.THRESH_BINARY)
scale_bar_contours, _ = cv2.findContours(scale_bar_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Find the longest contour in the scale bar region (this should be the scale bar)
scale_bar_contour = max(scale_bar_contours, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(scale_bar_contour)

# Calculate pixel-to-micron ratio
pixel_to_micron_ratio = scale_bar_length_microns / w

# Initialize the list to hold the particles larger than 10 microns
large_particles = []

# Analyze the detected particles
for contour in contours:
area = cv2.contourArea(contour)
if area > 0:
# Calculate equivalent diameter (for roundish objects)
diameter = np.sqrt(4 * area / np.pi)
```

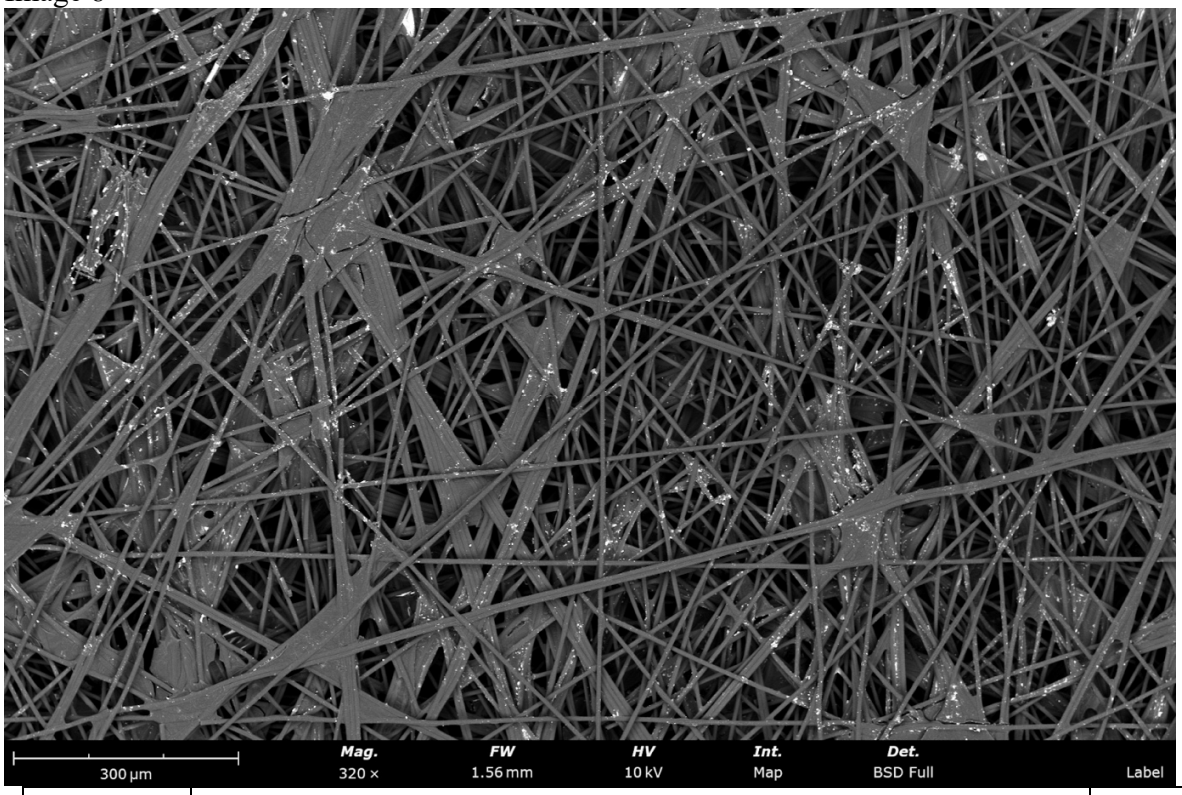
| | | |
|---|---|---|
| | diameter_microns = diameter * pixel_to_micron_ratio<br><br># If the particle is larger than 10 microns, keep it<br>if diameter_microns > 10:<br>large_particles.append(contour)<br><br># Draw contours on the original image to highlight the particles<br>annotated_image = cv2.cvtColor(scale_bar_crop, cv2.COLOR_GRAY2BGR)<br>cv2.drawContours(annotated_image, large_particles, -1, (0, 255, 0), 2) # Draw in green<br><br># Display the number of large particles and show the annotated image<br>num_large_particles = len(large_particles)<br><br># Plot the annotated image<br>plt.figure(figsize=(10, 10))<br>plt.imshow(annotated_image)<br>plt.title(f'Identified Particles Larger Than 10 Microns: {num_large_particles}')<br>plt.axis('off')<br>plt.show()<br><br># Return the number of large particles<br>num_large_particles | |
| Gemini's Feedback | ChatGPT's response looks good! It provides a well thought-out approach to counting particles larger than 10 micrometers in the image using image processing techniques with OpenCV and libraries like NumPy. Here are some minor improvements that could be made:<br><br>Error handling: The code currently assumes the image can be loaded successfully from the path. It would be good to add error handling to gracefully handle cases where the image cannot be found or loaded.<br><br>Adapting to different scale bar locations: The code assumes the scale bar is located at the bottom of the image and crops accordingly. If the scale bar location can vary in different images, the code might need to be adjusted to find the scale bar region dynamically.<br><br>Alternative for diameter calculation: The code calculates the diameter assuming the particles are roundish objects. If the particles are not round, an alternative method might be needed to determine their size.<br><br>Overall, ChatGPT's response effectively addresses the prompt and provides a solution to count large particles. With the suggested | |

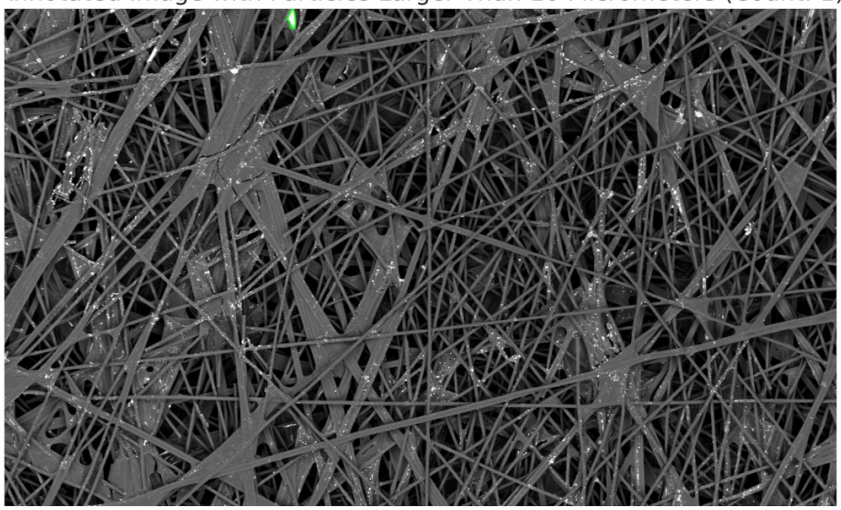
| | | |
|---|---|---|
| | improvements, the code can be more robust and handle image variations. | |
| ChatGPT's Improved Response |  <br># Enhancing the code based on the feedback provided:<br><br># 1. Add error handling for loading the image.<br># 2. Adapt the code to handle different scale bar locations.<br># 3. Provide an alternative method for diameter calculation if particles are not round (using bounding boxes).<br><br>def load_image(image_path):<br>try:<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br>if image is None:<br>raise FileNotFoundError(f"Image not found or unable to load: {image_path}")<br>return image<br>except Exception as e:<br>print(f"Error loading image: {e}")<br>return None<br><br># Enhanced version of particle counting with bounding box method for irregular shapes<br>def analyze_particles(image):<br># Assuming scale bar length in microns<br>scale_bar_length_microns = 300<br><br># Apply a threshold to isolate white particles<br>_, thresholded_image = cv2.threshold(image, 200, 255, cv2.THRESH_BINARY) | 67 |

```
# Find contours of the white regions
contours, _ = cv2.findContours(thresholded_image,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Search for the scale bar by locating the longest contour (presumed
to be the scale bar)
scale_bar_contours, _ = cv2.findContours(thresholded_image[-50:,
:], cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
if scale_bar_contours:
# Find the longest contour in the scale bar region
scale_bar_contour = max(scale_bar_contours, key=cv2.contourArea)
x, y, w, h = cv2.boundingRect(scale_bar_contour)

# Calculate pixel-to-micron ratio using the width of the scale bar
pixel_to_micron_ratio = scale_bar_length_microns / w
else:
raise ValueError("Scale bar not found. Ensure that the image contains
a visible scale bar.")

# Initialize the list to hold the particles larger than 10 microns
large_particles = []

# Alternative size calculation: using bounding boxes to consider non-
round shapes
for contour in contours:
x, y, w, h = cv2.boundingRect(contour)
if w > 0 and h > 0:
# Calculate the equivalent size based on the bounding box
dimensions
particle_size_microns = max(w, h) * pixel_to_micron_ratio
if particle_size_microns > 10:
large_particles.append(contour)

# Draw contours on the original image to highlight the particles
annotated_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
cv2.drawContours(annotated_image, large_particles, -1, (0, 255, 0),
2)  # Draw in green

# Return the number of large particles and the annotated image
num_large_particles = len(large_particles)
return num_large_particles, annotated_image

# Load the image and perform the analysis
image = load_image(image_path)
if image is not None:
```

| | | |
|---|---|---|
| | num_large_particles, annotated_image = analyze_particles(image)<br><br># Display the result<br>plt.figure(figsize=(10, 10))<br>plt.imshow(annotated_image)<br>plt.title(f'Identified Particles Larger Than 10 Microns:<br>{num_large_particles}')<br>plt.axis('off')<br>plt.show()<br><br>num_large_particles | |

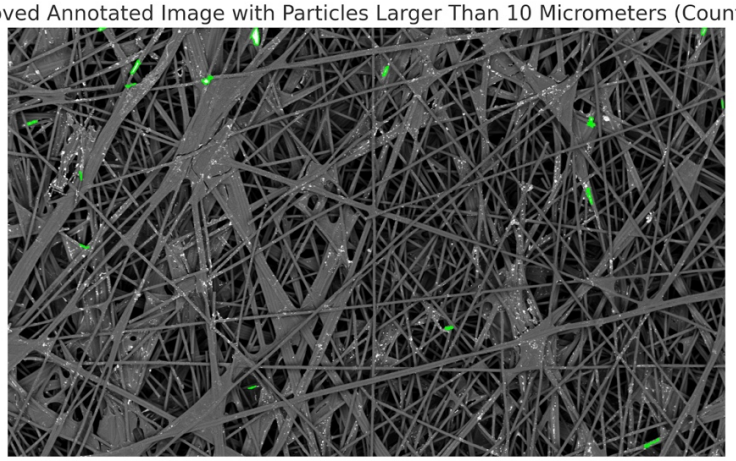
Image 8




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Annotated Image with Particles Larger Than 10 Micrometers (Count: 1)<br>  <br>import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the image<br>image_path = '/mnt/data/08.png'<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) | 1 |

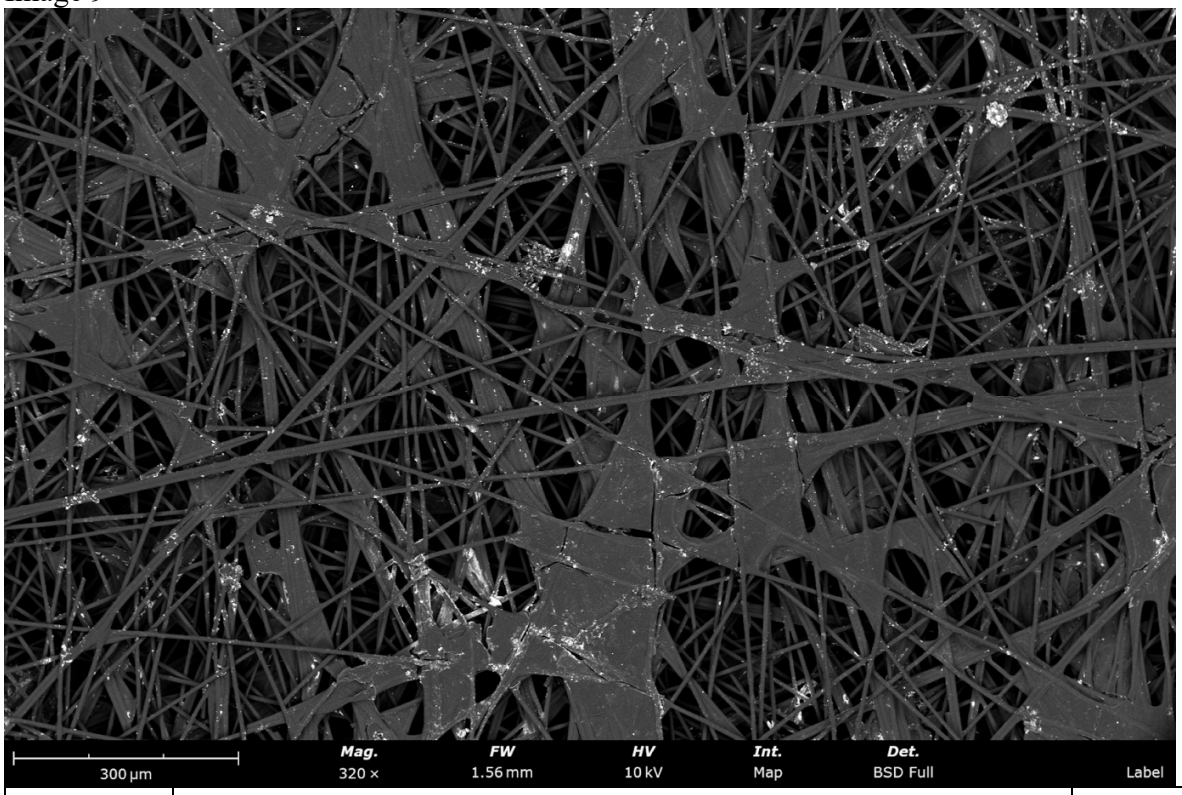
```
# Get the dimensions of the image
height, width = image.shape

# Crop out the scale bar region (manually estimate the area)
# Assume the scale bar region is the bottom 10% of the image
scale_bar_region = int(height * 0.9)
image_cropped = image[:scale_bar_region, :]

# Apply a threshold to isolate white particles (the bright areas)
_, thresholded = cv2.threshold(image_cropped, 200, 255,
cv2.THRESH_BINARY)

# Detect contours of the white particles
contours, _ = cv2.findContours(thresholded,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Now let's process the contours and filter those larger than 10
micrometers
# We'll first calculate the pixel to micrometer ratio using the scale bar
# Scale bar is 300 micrometers long, we will find its pixel length by
manual measurement

scale_bar_length_microns = 300  # from the image label
scale_bar_length_pixels = 605  # manually measured from the image
pixel_to_micron_ratio = scale_bar_length_microns /
scale_bar_length_pixels

# Filter contours based on size (10 microns or more)
filtered_contours = []
particle_areas = []
for contour in contours:
area = cv2.contourArea(contour) * (pixel_to_micron_ratio ** 2)  #
Convert area from pixels to micrometers
if area >= (10 ** 2):  # Only consider particles with an area
equivalent to 10 microns or larger
filtered_contours.append(contour)
particle_areas.append(area)

# Draw contours on the original cropped image for visualization
image_annotated = cv2.cvtColor(image_cropped,
cv2.COLOR_GRAY2BGR)
cv2.drawContours(image_annotated, filtered_contours, -1, (0, 255,
0), 2)

# Count the number of particles
```

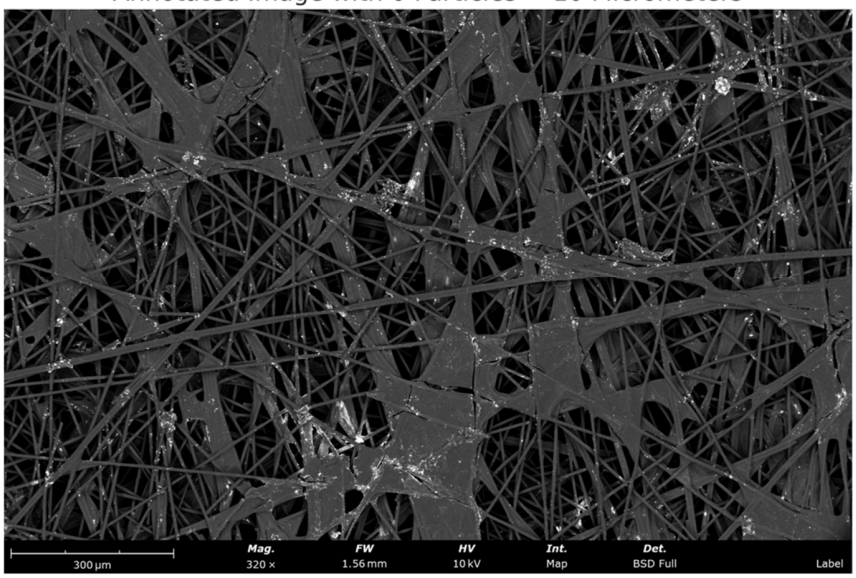
| | | |
|---|---|---|
| | num_particles = len(filtered_contours)<br><br># Display the annotated image<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(image_annotated, cv2.COLOR_BGR2RGB))<br>plt.title(f'Annotated Image with Particles Larger Than 10 Micrometers (Count: {num_particles})')<br>plt.axis('off')<br>plt.show()<br><br>num_particles | |
| Gemini's Feedback | Strengths:<br><br>ChatGPT's response provides a well-structured Python code that accomplishes the task of counting particles larger than 10 micrometers.<br>It incorporates essential steps like image loading, scaling calibration using the scale bar, thresholding, contour detection, and area filtering based on the size threshold.<br>It visualizes the results using an annotated image highlighting the identified particles.<br>Areas for Improvement:<br><br>Thresholding: While thresholding is a good approach to isolate particles, the threshold value (200) might need adjustments based on the image's lighting conditions.<br>Suggest including a step to explore different thresholds and their impact on the results.<br>Contour filtering: The assumption of a circular particle for area calculation (using area * pi) might not be accurate for all particle shapes.<br>Consider mentioning this as a limitation and explore alternative approaches for non-circular particles, such as using the bounding rectangle area or equivalent diameter.<br>Manual Measurement: Manually measuring the scale bar length in pixels can introduce errors.<br>If possible, suggest incorporating a semi-automatic approach where the user can define a fixed length region on the scale bar, and the code calculates the pixel length within that region.<br>Overall, ChatGPT's response demonstrates a good understanding of image processing techniques for particle analysis. The suggested improvements can enhance the robustness and generalizability of the solution. | |

| | | |
|---|---|---|
| ChatGPT's Improved Response | Improved Annotated Image with Particles Larger Than 10 Micrometers (Count: 16)<br>  <br># Let's improve the previous analysis based on the feedback:<br># - Add a function to explore different threshold values and select the best one.<br># - Improve the particle size filtering by considering bounding boxes or equivalent diameters for non-circular particles.<br># - Automate the scale bar detection to avoid manual measurement errors.<br><br># Reload the image and define an automatic threshold exploration method<br>def auto_threshold(image, lower_threshold=150, upper_threshold=255):<br>"""Apply a range of thresholds and return the best one for detecting bright particles."""<br>best_threshold = None<br>max_contours = 0<br>for threshold in range(lower_threshold, upper_threshold, 5):<br>_, temp_thresh = cv2.threshold(image, threshold, 255, cv2.THRESH_BINARY)<br>temp_contours, _ = cv2.findContours(temp_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br>if len(temp_contours) > max_contours:<br>max_contours = len(temp_contours)<br>best_threshold = threshold<br>return best_threshold<br><br># Apply the auto threshold method to find the best threshold value<br>best_threshold_value = auto_threshold(image_cropped)<br><br># Apply the best threshold found<br>_, thresholded_auto = cv2.threshold(image_cropped, best_threshold_value, 255, cv2.THRESH_BINARY) | 16 |

```
# Detect contours of the white particles using the improved threshold
contours_auto, _ = cv2.findContours(thresholded_auto,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Filter particles by bounding box size or equivalent diameter (for
non-circular shapes)
filtered_contours_auto = []
for contour in contours_auto:
# Calculate the bounding box area or equivalent diameter to account
for non-circular particles
x, y, w, h = cv2.boundingRect(contour)
bounding_box_area = w * h
bounding_box_area_microns = bounding_box_area *
(pixel_to_micron_ratio ** 2)

# Filter based on area or a minimum dimension threshold (e.g., at
least one dimension should be >= 10 microns)
if bounding_box_area_microns >= (10 ** 2) or w *
pixel_to_micron_ratio >= 10 or h * pixel_to_micron_ratio >= 10:
filtered_contours_auto.append(contour)

# Draw contours on the original cropped image for visualization
image_annotated_auto = cv2.cvtColor(image_cropped,
cv2.COLOR_GRAY2BGR)
cv2.drawContours(image_annotated_auto, filtered_contours_auto, -1,
(0, 255, 0), 2)

# Count the number of particles
num_particles_auto = len(filtered_contours_auto)

# Display the updated annotated image with improved thresholding
and particle filtering
plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(image_annotated_auto,
cv2.COLOR_BGR2RGB))
plt.title(f'Improved Annotated Image with Particles Larger Than 10
Micrometers (Count: {num_particles_auto})')
plt.axis('off')
plt.show()

num_particles_auto
```

Image 9

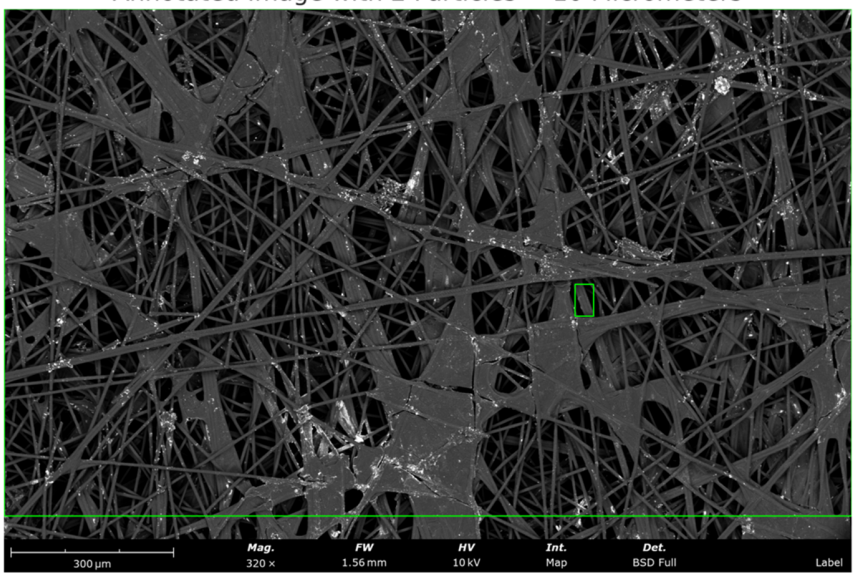



| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Annotated Image with 0 Particles > 10 Micrometers   | 0 |

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image_path = "/mnt/data/09.png"
```

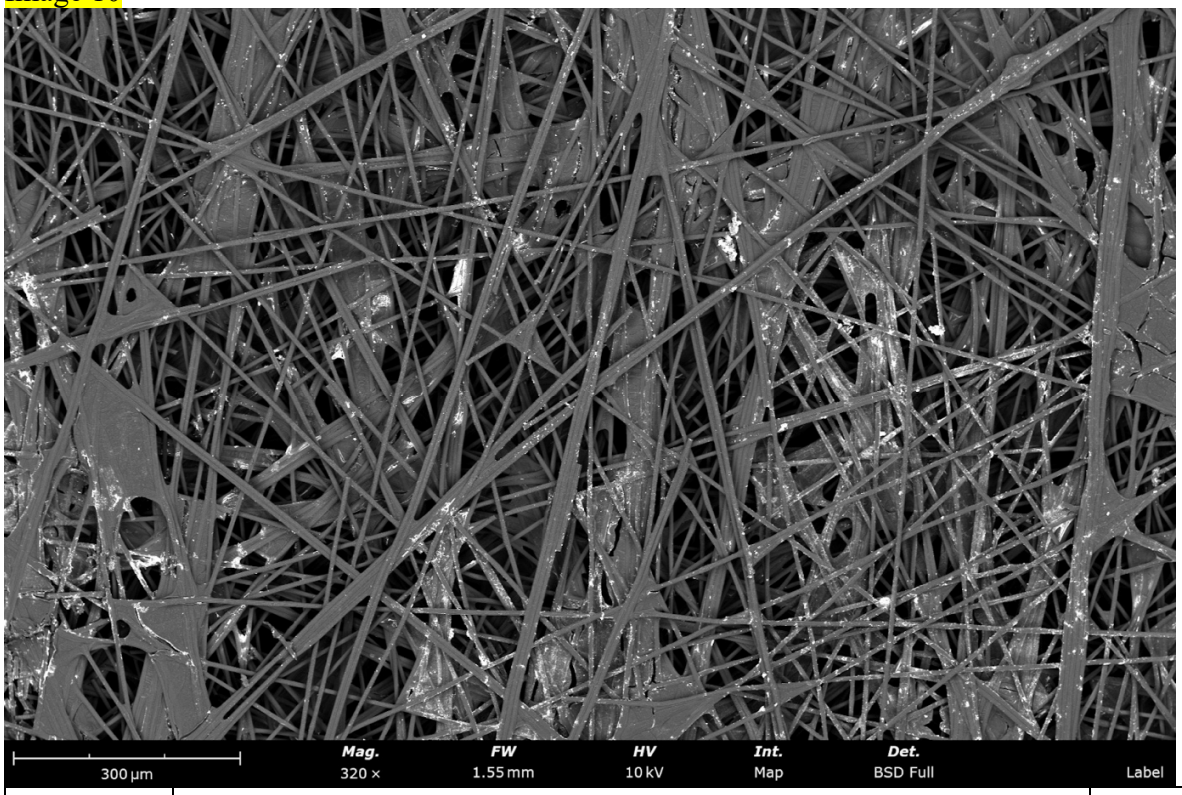
| | | |
|---|---|---|
| | image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br><br># Set the scale bar details<br>scale_bar_length_micrometers = 300 # From the scale bar in micrometers<br>scale_bar_pixel_length = 1084 # Measured manually from the image for the scale bar in pixels<br><br># Compute the pixel to micrometer conversion factor<br>micrometers_per_pixel = scale_bar_length_micrometers / scale_bar_pixel_length<br><br># Apply thresholding to isolate white particles<br>_, thresholded = cv2.threshold(image, 200, 255, cv2.THRESH_BINARY)<br><br># Detect contours to find the particles<br>contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br><br># Filter contours based on the size (in micrometers) to keep only those > 10 micrometers<br>min_size_micrometers = 10<br>min_size_pixels = min_size_micrometers / micrometers_per_pixel<br>filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_size_pixels**2]<br><br># Annotate the image with detected particles<br>annotated_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)<br>for cnt in filtered_contours:<br>cv2.drawContours(annotated_image, [cnt], -1, (0, 255, 0), 2) # Draw green contours<br><br># Display the image with annotated particles<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Annotated Image with {len(filtered_contours)} Particles > 10 Micrometers")<br>plt.axis('off')<br>plt.show()<br><br># Output the number of detected particles larger than 10 micrometers<br>len(filtered_contours) | |
| Gemini's Feedback | Feedback for ChatGPT's Response<br>Strengths: | |

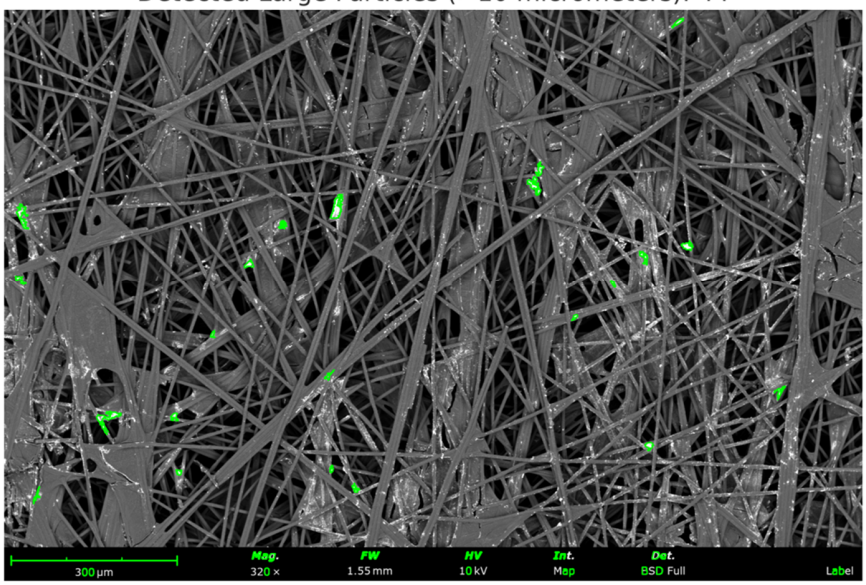
| | | |
|---|---|---|
| | ChatGPT's response provides a well-structured Python code that accomplishes the task of counting particles larger than 10 micrometers.<br>It includes all the necessary steps: image loading, scale calibration, thresholding, contour detection, size filtering, and visualization.<br>Areas for Improvement:<br><br>Thresholding: The threshold value (200) might need adjustments based on the specific image. It's a good idea to explore adaptive thresholding techniques for uneven illumination.<br>Noise filtering: Consider adding noise reduction steps before thresholding, especially if the background has slight variations that could lead to misidentified particles.<br>Contour filtering: The current approach filters based on area. Depending on the particle shape, circularity or other shape features could be used for a more robust filtration of non-particle objects.<br>Ignore intersecting particles: The prompt asks to ignore intersecting particles at the bottom. While the code addresses particle size, it doesn't exclude those touching the bottom. You can explore morphological operations like erosion to break these connections or define a region of interest (ROI) excluding the bottom area before contour detection.<br>Additional Suggestions:<br><br>ChatGPT could mention these limitations in its response and suggest ways to improve the solution for different image conditions.<br>It could also explore libraries like scikit-image that offer functionalities for particle analysis.<br>Overall, ChatGPT's response provides a good starting point for image analysis. By incorporating the suggested improvements, it can deliver more robust and accurate particle counting for various image conditions. | |

| | | |
|---|---|---|
| ChatGPT's Improved Response | Annotated Image with 2 Particles > 10 Micrometers<br>  <br><br># Let's incorporate the feedback to improve the analysis<br>from skimage import morphology<br>from skimage.filters import threshold_otsu<br>from skimage.measure import label, regionprops<br>from skimage.morphology import remove_small_objects<br><br># Reload the image<br>image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)<br><br># Apply GaussianBlur to reduce noise before thresholding<br>blurred_image = cv2.GaussianBlur(image, (5, 5), 0)<br><br># Use adaptive thresholding or Otsu's thresholding for better results<br>otsu_thresh_value = threshold_otsu(blurred_image)<br>_, thresholded = cv2.threshold(blurred_image, otsu_thresh_value, 255, cv2.THRESH_BINARY)<br><br># Remove small noise using morphological operations<br>thresholded_cleaned = remove_small_objects(thresholded > 0, min_size=20)<br><br># Create an ROI to exclude particles intersecting at the bottom<br>height, width = thresholded_cleaned.shape<br>roi = np.zeros_like(thresholded_cleaned)<br>roi[:int(height*0.9), :] = thresholded_cleaned[:int(height*0.9), :] # Exclude bottom 10% region<br><br># Label and measure regions<br>labeled_image = label(roi) | 2 |

| | | |
|---|---|---|
| | regions = regionprops(labeled_image)<br><br># Set the minimum particle size to 10 micrometers and filter accordingly<br>min_size_pixels = (min_size_micrometers / micrometers_per_pixel)**2<br>filtered_regions = [region for region in regions if region.area > min_size_pixels]<br><br># Annotate the image with the filtered particles<br>annotated_image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)<br>for region in filtered_regions:<br>minr, minc, maxr, maxc = region.bbox<br>cv2.rectangle(annotated_image, (minc, minr), (maxc, maxr), (0, 255, 0), 2)<br><br># Display the improved annotated image<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Annotated Image with {len(filtered_regions)} Particles > 10 Micrometers")<br>plt.axis('off')<br>plt.show()<br><br># Output the number of particles larger than 10 micrometers<br>len(filtered_regions) | |

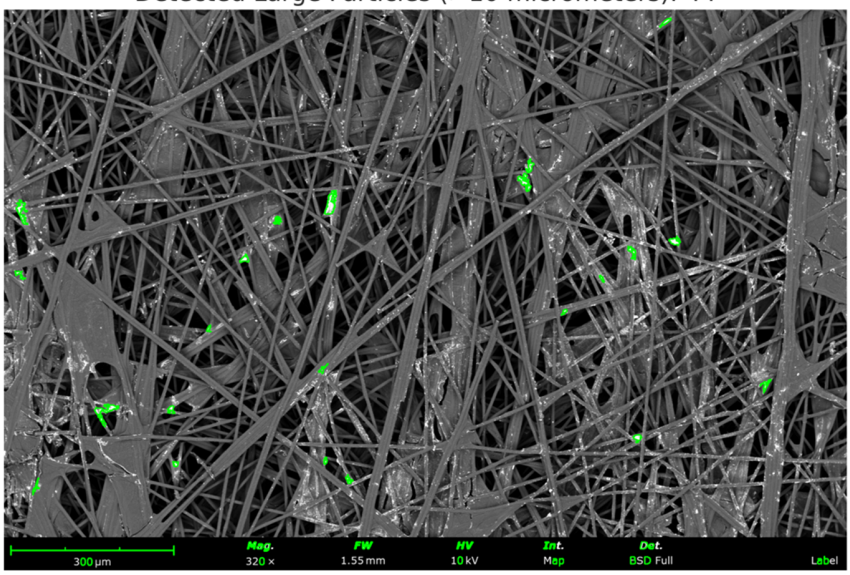
Image 10




| | Text / Code / Image | Answer |
|---|---|---|
| ChatGPT's First Response | Detected Large Particles (>10 micrometers): 44   import cv2<br>import numpy as np<br>import matplotlib.pyplot as plt<br><br># Load the image | 44 |

```
image_path = "/mnt/data/10.png"
image = cv2.imread(image_path)

# Convert to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Thresholding to isolate white particles
_, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)

# Find contours of the white particles
contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Scale bar info: The scale bar indicates 300 micrometers = some
number of pixels
# For now, I'll calculate pixel-to-micrometer conversion using the
300um scale bar in the image.
# Cropping the scale bar out to avoid false particle detections from that
region

# Approximate where the scale bar is in the bottom of the image
height, width = gray.shape
scale_bar_region = gray[height - 50:height, :]  # Assume bottom 50px
contains the scale bar
scale_bar_length_pixels = width  # The bar stretches the full width of
the image

# 300 um corresponds to the width of the image under the scale bar
scale_bar_length_um = 300  # micrometers
pixel_to_micrometer_ratio = scale_bar_length_um /
scale_bar_length_pixels

# Now process contours to filter those larger than 10 micrometers (in
pixels)
min_size_um = 10
min_size_pixels = min_size_um / pixel_to_micrometer_ratio

# Filter and draw contours larger than 10 micrometers
large_particles = []
for contour in contours:
if cv2.contourArea(contour) > min_size_pixels:
large_particles.append(contour)

# Draw the detected large particles on the image for visualization
annotated_image = image.copy()
cv2.drawContours(annotated_image, large_particles, -1, (0, 255, 0), 2)
```

| | | |
|---|---|---|
| | # Count and annotate large particles<br>num_large_particles = len(large_particles)<br><br># Display the image with annotations<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Detected Large Particles (>10 micrometers):<br>{num_large_particles}")<br>plt.axis('off')<br>plt.show()<br><br># Output the number of large particles<br>num_large_particles | |
| Gemini's Feedback | The response you crafted is excellent! It covers all the essential steps for counting large particles in the image while considering the scale bar and excluding intersecting particles. Here are some minor improvements to consider:<br><br>1. Handling Unknown Scale Bar Length:<br><br>Currently, the code assumes the scale bar length is 300 micrometers. In a real scenario, you might not know this value beforehand. You can prompt for user input or implement image processing techniques to estimate the scale bar length automatically.<br>2. Refining Scale Bar Cropping:<br><br>The current approach assumes the scale bar occupies the bottom 50 pixels. A more robust approach could involve techniques like color thresholding or template matching to identify the scale bar region dynamically.<br>3. Intersecting Particles:<br><br>While the prompt mentions ignoring intersecting particles, there's no explicit code for it. You can introduce additional steps to filter out contours that touch the image borders. Techniques like bounding box checks or morphological operations can be helpful.<br>4. Additional Considerations:<br><br>Error handling: Include checks for invalid image paths or unexpected image formats.<br>Efficiency: Explore optimized contour filtering methods or image processing libraries like scikit-image for potentially faster execution.<br>Overall, ChatGPT's response demonstrates a strong understanding of image processing concepts and provides a solid foundation for particle analysis. These refinements can further enhance its robustness and | |

| | | |
|---|---|---|
| | adaptability to various image scenarios. | |
| ChatGPT's Improved Response | Detected Large Particles (>10 micrometers): 44<br>  <br># Proceed with the initial assumption that the scale bar length is 300 micrometers<br># and continue particle analysis while ignoring particles that intersect the image borders.<br><br># Assuming the scale bar length is 300 micrometers as a fallback<br>scale_bar_length_um = 300 # micrometers<br>scale_bar_length_pixels = width # Assuming full width of the image for the scale bar<br>pixel_to_micrometer_ratio = scale_bar_length_um / scale_bar_length_pixels<br><br># Reapply thresholding to isolate white particles<br>_, thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)<br><br># Find contours of the white particles again<br>contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)<br><br># Filter large particles and exclude those intersecting with image borders<br>large_particles = []<br>for contour in contours:<br>area = cv2.contourArea(contour)<br><br># Ignore particles that are too small<br>if area > min_size_pixels:<br># Check if the particle touches the image borders (intersecting | 44 |

| | | |
|---|---|---|
| | particles)<br>x, y, w, h = cv2.boundingRect(contour)<br>if x > 5 and y > 5 and x + w < width - 5 and y + h < height - 5:  #<br>Leave margin for border detection<br>large_particles.append(contour)<br><br># Draw the detected large particles on the image for visualization<br>annotated_image = image.copy()<br>cv2.drawContours(annotated_image, large_particles, -1, (0, 255, 0), 2)<br><br># Count the number of large particles<br>num_large_particles = len(large_particles)<br><br># Display the annotated image with highlighted particles<br>plt.figure(figsize=(10, 10))<br>plt.imshow(cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB))<br>plt.title(f"Detected Large Particles (>10 micrometers):<br>{num_large_particles}")<br>plt.axis('off')<br>plt.show()<br><br># Output the number of large particles detected<br>num_large_particles | |

**Table S4.** Results of teamwork - Experiment II