```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans

# Load the microstructure image
image_path = '/content/Fe-C-3-2-Si-2-5-Mg-0-05-wt-_jpg.rf.c0f
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Apply Canny edge detection to highlight boundaries
edges = cv2.Canny(gray, threshold1=50, threshold2=150)

# Step 2: Thresholding to separate phases
_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + c

# Step 3: Apply morphological operations to clean up
kernel = np.ones((3, 3), np.uint8)
morph = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)

# Step 4: Use K-means clustering for segmentation
pixel_values = gray.reshape((-1, 1))  # Reshape the image to
pixel_values = np.float32(pixel_values)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(pixel_values)
segmented_image = kmeans.labels_.reshape(gray.shape)

# Visualize results
plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)
plt.title("Original Image")
plt.imshow(image_rgb)

plt.subplot(2, 3, 2)
plt.title("Grayscale Image")
plt.imshow(gray, cmap='gray')

plt.subplot(2, 3, 3)
plt.title("Canny Edge Detection")
plt.imshow(edges, cmap='gray')

plt.subplot(2, 3, 4)
plt.title("Thresholded Image")
plt.imshow(thresh, cmap='gray')

plt.subplot(2, 3, 5)
plt.title("Morphological Cleanup")
plt.imshow(morph, cmap='gray')

plt.subplot(2, 3, 6)
plt.title("K-means Segmentation")
plt.imshow(segmented_image, cmap='gray')

plt.tight_layout()
plt.show()
```

DisabledFunctionError ✕ •••

You

Please explain the error:

```
DisabledFunctionError: cv2.imshow(
to crash; see https://github.com/j
As a substitution, consider using
  from google.colab.patches import
```

Gemini

Hypothesis

The error `DisabledFunctionError: cv2.imshow() is disabled in Colab` arises because you are trying to use the `cv2.imshow()` function within a Google Colab environment. This function is known to cause crashes in Colab's Jupyter sessions, so it has been disabled.

The traceback clearly shows the issue originating from this line:

```
cv2.imshow('Ferrite Mask', ferri
```

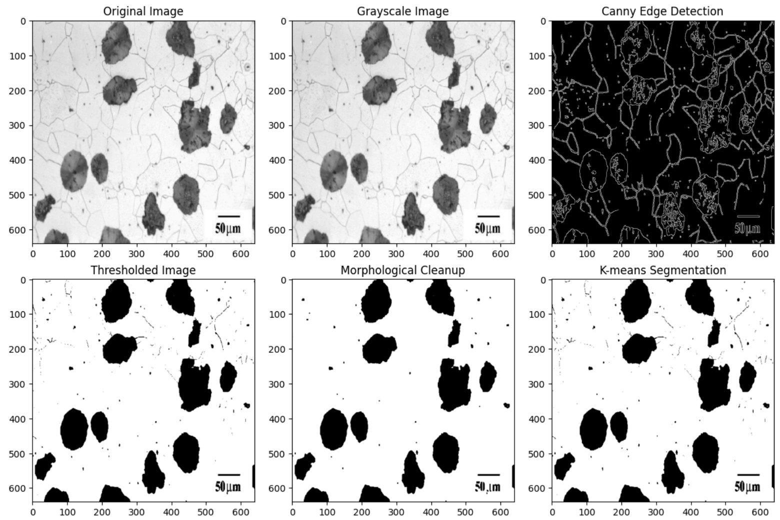
Use code with caution

This is because `cv2.imshow()` which is used for displaying images in a separate window, is not supported in Google Colab.

Suggested Changes

To display images within Google Colab, you should use the `cv2_imshow()` function provided by the `google.colab.patches` module. Here's how you can modify your code:

```
import cv2
import numpy as np
from sklearn.cluster import KMeai
from google.colab.patches import

# ... (rest of your code remains
```




```
import cv2
import numpy as np
from sklearn.cluster import KMeans

# Load the microstructure image
image_path = '/content/Fe-C-3-2-Si-2-5-Mg-0-05-wt-_jpg.rf.c
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Apply K-means Clustering
pixel_values = gray.reshape((-1, 1))
pixel_values = np.float32(pixel_values)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(pixel_values)
segmented_image = kmeans.labels_.reshape(gray.shape)

# Identify ferrite and pearlite clusters
ferrite_mask = (segmented_image == segmented_image.max()).a
pearlite_mask = (segmented_image == segmented_image.min()).

# Step 2: Quantify Phase Content
total_pixels = gray.size
ferrite_pixels = np.sum(ferrite_mask == 255)
pearlite_pixels = np.sum(pearlite_mask == 255)

ferrite_content = (ferrite_pixels / total_pixels) * 100
pearlite_content = (pearlite_pixels / total_pixels) * 100
```

```
# Step 3: Display Masks (optiona
cv2_imshow(ferrite_mask) # Use c
cv2_imshow(pearlite_mask) # Use

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Use code with caution

Explanation of Changes:

1. **Import `cv2_imshow`:** The line `from google.colab.patches import cv2_imshow` imports the necessary function for displaying images in Colab.
2. **Replace `cv2.imshow`:** The calls to `cv2.imshow` have been replaced with `cv2_imshow`. This function will display the images directly within the Colab notebook output.

▶ Sources

Rate this answer

```
# Print Results
print(f"Ferrite Content: {ferrite_content:.2f}%")
print(f"Pearlite Content: {pearlite_content:.2f}%")

# Step 3: Display Masks (optional visualization)
cv2.imshow(ferrite_mask)
cv2.imshow( pearlite_mask)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

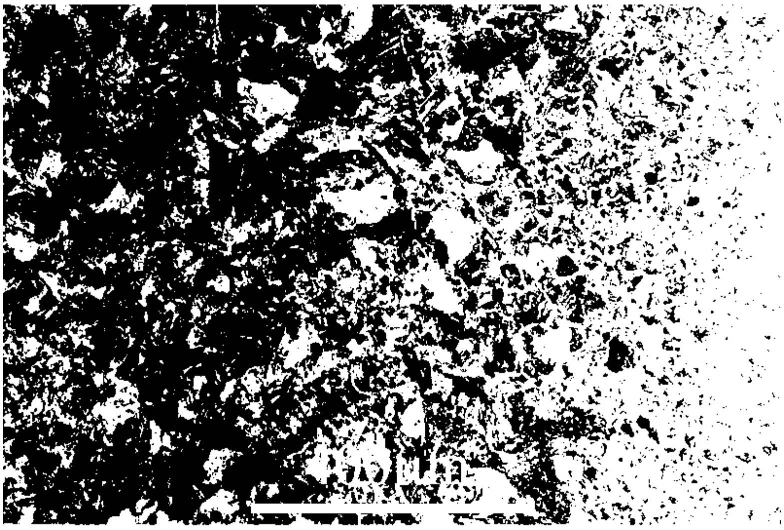
```
Ferrite Content: 82.60%
Pearlite Content: 17.40%
---------------------------------------------------------------------------
DisabledFunctionError                     Traceback (most recent call last)
<ipython-input-6-54bf0ff7c944> in <cell line: 35>()
     33
     34 # Step 3: Display Masks (optional visualization)
---> 35 cv2.imshow('Ferrite Mask', ferrite_mask)
     36 cv2.imshow('Pearlite Mask', pearlite_mask)
     37 cv2.waitKey(0)

/usr/local/lib/python3.10/dist-packages/google/colab/_import_hooks/_cv2.py in wrapped(*args, **kwargs)
     46   def wrapped(*args, **kwargs):
     47     if not os.environ.get(env_var, False):
---> 48       raise DisabledFunctionError(message, name or func.__name__)
     49     return func(*args, **kwargs)
     50

DisabledFunctionError: cv2.imshow() is disabled in Colab, because it causes Jupyter sessions
to crash; see
```

Next steps: Explain error

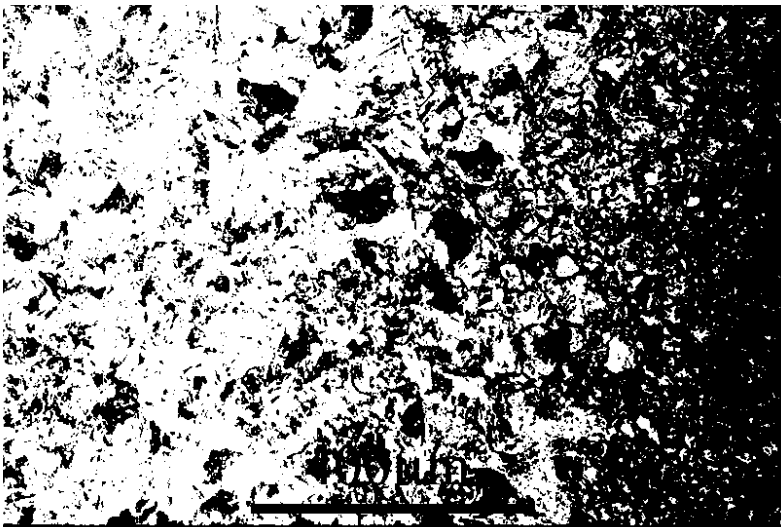
```
import cv2
import numpy as np
from sklearn.cluster import KMeans
from google.colab.patches import cv2_imshow  # Colab-compatit

# Load the microstructure image
image_path = '/content/Fe, C 0.07, Mn 2.2 (wt%) (4).jpg'  # U
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Apply K-means Clustering
pixel_values = gray.reshape((-1, 1))
pixel_values = np.float32(pixel_values)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(pixel_values)
segmented_image = kmeans.labels_.reshape(gray.shape)

# Step 2: Assign phases correctly
# Ferrite is darker, Pearlite is lighter
ferrite_mask = (segmented_image == segmented_image.min()).ast
```

```
    pearlite_mask = (segmented_image == segmented_image.max()).as

    # Step 3: Quantify Phase Content
    total_pixels = gray.size
    ferrite_pixels = np.sum(ferrite_mask == 255)
    pearlite_pixels = np.sum(pearlite_mask == 255)

    ferrite_content = (ferrite_pixels / total_pixels) * 100
    pearlite_content = (pearlite_pixels / total_pixels) * 100

    # Print Results
    print(f"Ferrite Content: {ferrite_content:.2f}%")
    print(f"Pearlite Content: {pearlite_content:.2f}%")

    # Step 4: Display Masks
    print("Ferrite Mask:")
    cv2_imshow(ferrite_mask)

    print("Pearlite Mask:")
    cv2_imshow(pearlite_mask)
```

```
Ferrite Content: 47.55%
Pearlite Content: 52.45%
Ferrite Mask:
```



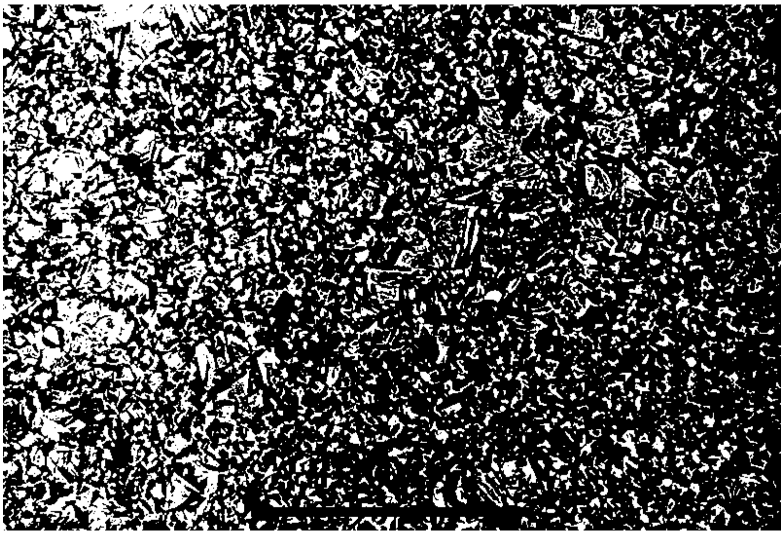
```
Pearlite Mask:
```



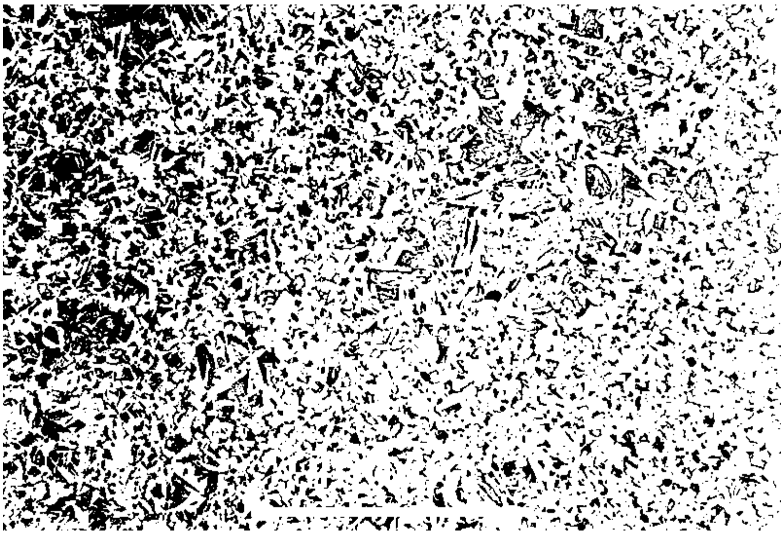
Generate a slider using jupyter widgets Close

```
# Load the microstructure image
image_path = '/content/Fe, C 0.07, Mn 2.2 (wt%) (2).jpg'  #
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Apply K-means Clustering
pixel_values = gray.reshape((-1, 1))
pixel_values = np.float32(pixel_values)

kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(pixel_values)
segmented_image = kmeans.labels_.reshape(gray.shape)

# Step 2: Assign phases correctly
# Ferrite is darker, Pearlite is lighter
ferrite_mask = (segmented_image == segmented_image.min()).a
pearlite_mask = (segmented_image == segmented_image.max()).

# Step 3: Quantify Phase Content
total_pixels = gray.size
ferrite_pixels = np.sum(ferrite_mask == 255)
pearlite_pixels = np.sum(pearlite_mask == 255)

ferrite_content = (ferrite_pixels / total_pixels) * 100
pearlite_content = (pearlite_pixels / total_pixels) * 100

# Print Results
print(f"Ferrite Content: {ferrite_content:.2f}%")
print(f"Pearlite Content: {pearlite_content:.2f}%")

# Step 4: Display Masks
print("Ferrite Mask:")
cv2_imshow(ferrite_mask)

print("Pearlite Mask:")
cv2_imshow(pearlite_mask)
```

```
Ferrite Content: 29.20%
Pearlite Content: 70.80%
Ferrite Mask:
```

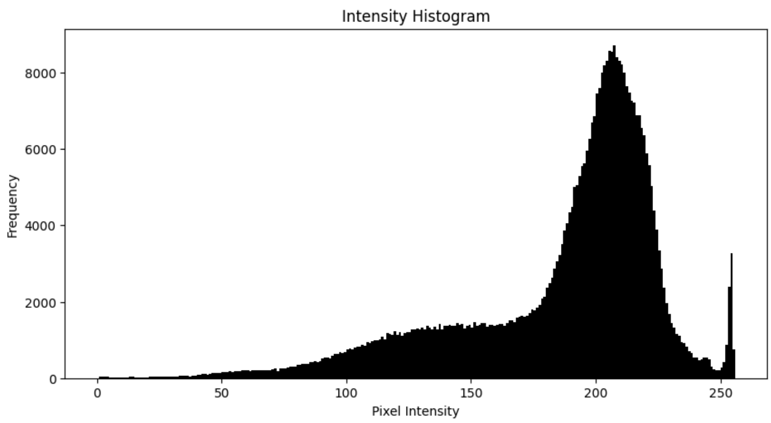


```
Pearlite Mask:
```



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

# Load the microstructure image
image_path = '/content/Fe, C 0.07, Mn 2.2 (wt%)'  # File path
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Histogram Analysis to Find Intensity Distribution
plt.figure(figsize=(10, 5))
plt.hist(gray.ravel(), bins=256, range=(0, 256), color='black
plt.title("Intensity Histogram")
plt.xlabel("Pixel Intensity")
plt.ylabel("Frequency")
plt.show()

# Step 2: Thresholding (Otsu's Method)
_, ferrite_pearlite_mask = cv2.threshold(gray, 0, 255, cv2.TH

# Ferrite = dark (binary 0), Pearlite = light (binary 255)
```

```
ferrite_mask = (ferrite_pearlite_mask == 0).astype(np.uint8)
pearlite_mask = (ferrite_pearlite_mask == 255).astype(np.uint

# Step 3: Percentage Calculation
total_pixels = gray.size
ferrite_pixels = np.sum(ferrite_mask == 255)
pearlite_pixels = np.sum(pearlite_mask == 255)

ferrite_percentage = (ferrite_pixels / total_pixels) * 100
pearlite_percentage = (pearlite_pixels / total_pixels) * 100

# Print Results
print(f"Ferrite Content (Actual): {ferrite_percentage:.2f}%")
print(f"Pearlite Content (Actual): {pearlite_percentage:.2f}%

# Display the Masks
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title("Ferrite Mask")
plt.imshow(ferrite_mask, cmap='gray')

plt.subplot(1, 2, 2)
plt.title("Pearlite Mask")
plt.imshow(pearlite_mask, cmap='gray')
plt.show()
```




```
Ferrite Content (Actual): 23.03%
Pearlite Content (Actual): 76.97%
```