```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from skimage.feature import graycomatrix, graycoprops
from skimage import data
from skimage import io


image_url_s = "/content/smooth (1).jpg"

# Load the image from the URL
image_s = io.imread(image_url_s)

image_s = cv2.cvtColor(image_s, cv2.COLOR_BGR2GRAY)

# Calculate the GLCM
glcm_s = graycomatrix(image_s, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)

image_url_c = "/content/coarse (1).jpg"

# Load the image from the URL
image_c = io.imread(image_url_c)

image_c = cv2.cvtColor(image_c, cv2.COLOR_BGR2GRAY)

# Calculate the GLCM
glcm_c = graycomatrix(image_c, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)

image_url_r = "/content/random (1).jpg"

# Load the image from the URL
image_r = io.imread(image_url_r)

image_r = cv2.cvtColor(image_r, cv2.COLOR_BGR2GRAY)

# Calculate the GLCM
glcm_r = graycomatrix(image_r, distances=[1], angles=[0], levels=256, symmetric=True, normed=True)

print(glcm_s)
```

```
[[0.00014299]]

...

[[0.        ]]

[[0.        ]]

[[0.        ]]]
```

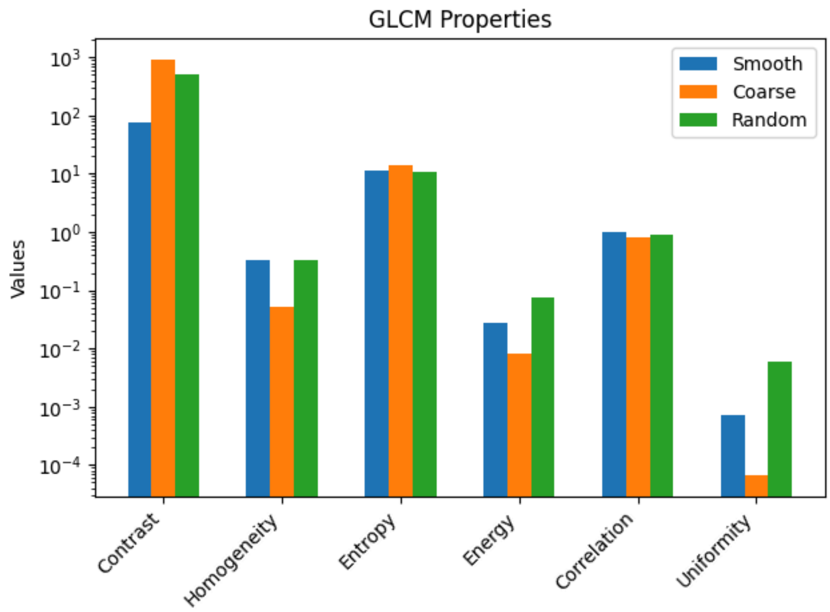
```
[[0.        ]]

[[0.        ]]

...

[[0.        ]]

[[0.        ]]

[[0.        ]]]]
```

```
print(f"Smooth Image: \n")

contrast = graycoprops(glcm_s, 'contrast')[0, 0]
print(f"Contrast: {contrast}")

homogeneity = graycoprops(glcm_s, 'homogeneity')[0, 0]
print(f"Homogeneity: {homogeneity}")

entropy = -np.sum(glcm_s * np.log2(glcm_s + np.finfo(float).eps))  # Avoid log(0)
print(f"Entropy: {entropy}")

energy = graycoprops(glcm_s, 'energy')[0, 0]
print(f"Energy: {energy}")

correlation = graycoprops(glcm_s, 'correlation')[0, 0]
print(f"Correlation: {correlation}")

uniformity = np.sum(glcm_s**2)
print(f"Uniformity: {uniformity}")

print(f"\n\n")


print(f"Coarse Image: \n")

contrast = graycoprops(glcm_c, 'contrast')[0, 0]
print(f"Contrast: {contrast}")

homogeneity = graycoprops(glcm_c, 'homogeneity')[0, 0]
print(f"Homogeneity: {homogeneity}")

entropy = -np.sum(glcm_c * np.log2(glcm_c + np.finfo(float).eps))  # Avoid log(0)
print(f"Entropy: {entropy}")

energy = graycoprops(glcm_c, 'energy')[0, 0]
print(f"Energy: {energy}")

correlation = graycoprops(glcm_c, 'correlation')[0, 0]
print(f"Correlation: {correlation}")

uniformity = np.sum(glcm_c**2)
print(f"Uniformity: {uniformity}")

print(f"\n\n")


print(f"Random Image: \n")

contrast = graycoprops(glcm_r, 'contrast')[0, 0]
print(f"Contrast: {contrast}")

homogeneity = graycoprops(glcm_r, 'homogeneity')[0, 0]
print(f"Homogeneity: {homogeneity}")

entropy = -np.sum(glcm_r * np.log2(glcm_r + np.finfo(float).eps))  # Avoid log(0)
print(f"Entropy: {entropy}")

energy = graycoprops(glcm_r, 'energy')[0, 0]
print(f"Energy: {energy}")

correlation = graycoprops(glcm_r, 'correlation')[0, 0]
print(f"Correlation: {correlation}")

uniformity = np.sum(glcm_r**2)
print(f"Uniformity: {uniformity}")

print(f"\n\n")
```

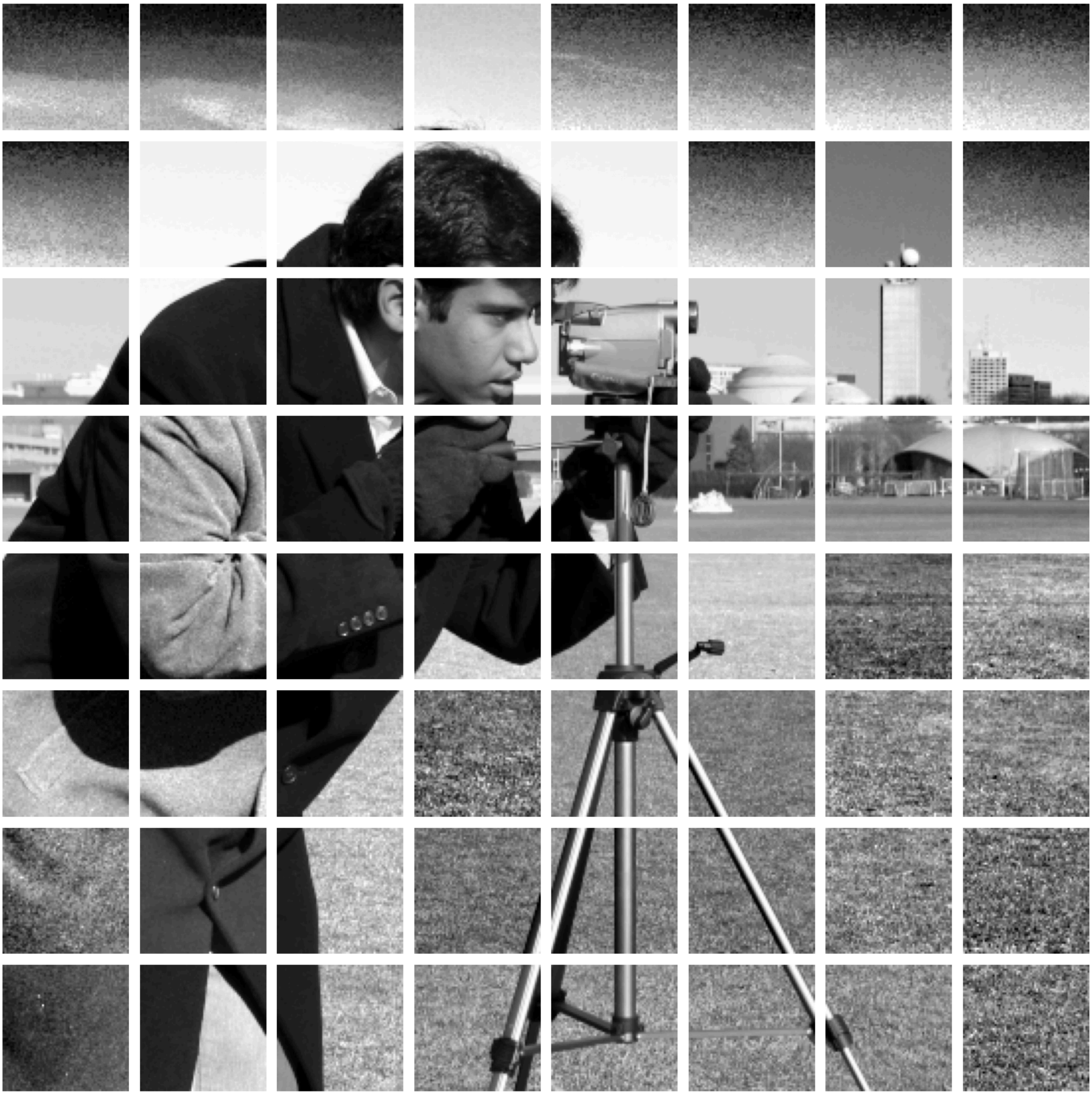
```
Smooth Image:
```

```
Contrast: 77.10747240801221
Homogeneity: 0.3223863534611505
Entropy: 11.383288246857482
Energy: 0.026945952530940533
Correlation: 0.9828275987363824
Uniformity: 0.0007260843577997003



Coarse Image:

Contrast: 920.8361701045027
Homogeneity: 0.051822932699479615
Entropy: 14.334394081805499
Energy: 0.008172530883756805
Correlation: 0.8121832368607294
Uniformity: 6.679026104595874e-05



Random Image:

Contrast: 521.4233250876109
Homogeneity: 0.32558657698305526
Entropy: 10.727158990407611
Energy: 0.07622820379559789
Correlation: 0.8939849867015447
Uniformity: 0.005810739053903202
```

```
# Data for the plot
properties = ['Contrast', 'Homogeneity', 'Entropy', 'Energy', 'Correlation', 'Uniformity']
smooth_values = [
    graycoprops(glcm_s, 'contrast')[0, 0],
    graycoprops(glcm_s, 'homogeneity')[0, 0],
    -np.sum(glcm_s * np.log2(glcm_s + np.finfo(float).eps)),
    graycoprops(glcm_s, 'energy')[0, 0],
    graycoprops(glcm_s, 'correlation')[0, 0],
    np.sum(glcm_s**2)
]
coarse_values = [
    graycoprops(glcm_c, 'contrast')[0, 0],
    graycoprops(glcm_c, 'homogeneity')[0, 0],
    -np.sum(glcm_c * np.log2(glcm_c + np.finfo(float).eps)),
    graycoprops(glcm_c, 'energy')[0, 0],
    graycoprops(glcm_c, 'correlation')[0, 0],
    np.sum(glcm_c**2)
]
random_values = [
    graycoprops(glcm_r, 'contrast')[0, 0],
    graycoprops(glcm_r, 'homogeneity')[0, 0],
    -np.sum(glcm_r * np.log2(glcm_r + np.finfo(float).eps)),
    graycoprops(glcm_r, 'energy')[0, 0],
    graycoprops(glcm_r, 'correlation')[0, 0],
    np.sum(glcm_r**2)
]

x = np.arange(len(properties))  # the label locations
width = 0.2  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width, smooth_values, width, label='Smooth')
rects2 = ax.bar(x, coarse_values, width, label='Coarse')
rects3 = ax.bar(x + width, random_values, width, label='Random')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Values')
ax.set_title('GLCM Properties')
ax.set_xticks(x)
ax.set_xticklabels(properties, rotation=45, ha='right')  # Rotate x-axis labels for better readability
ax.legend()

# Set y-axis scale to logarithmic
ax.set_yscale('log')  # or 'symlog' for handling negative values

fig.tight_layout()

plt.show()
```




```
import numpy as np
import matplotlib.pyplot as plt
from skimage import data

# Load the cameraman image
image = data.camera()

# Define the patch size
patch_size = (64, 64)  # Example: 64x64 patches

# Get image dimensions
image_height, image_width = image.shape

# Create a list to hold the image patches
patches = []

# Segment the image into patches
for i in range(0, image_height, patch_size[0]):
    for j in range(0, image_width, patch_size[1]):
        patch = image[i:i+patch_size[0], j:j+patch_size[1]]
        patches.append(patch)

# Plot all patches
num_patches = len(patches)
cols = 8  # Define the number of columns for display
rows = num_patches // cols + (num_patches % cols > 0)

fig, axes = plt.subplots(rows, cols, figsize=(15, 15))

# Flatten axes to handle them in a single loop
axes = axes.flatten()

# Loop through patches and axes to display them
for i, patch in enumerate(patches):
    axes[i].imshow(patch, cmap='gray')
    axes[i].axis('off')

# Hide any unused axes
for i in range(num_patches, len(axes)):
    axes[i].axis('off')

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



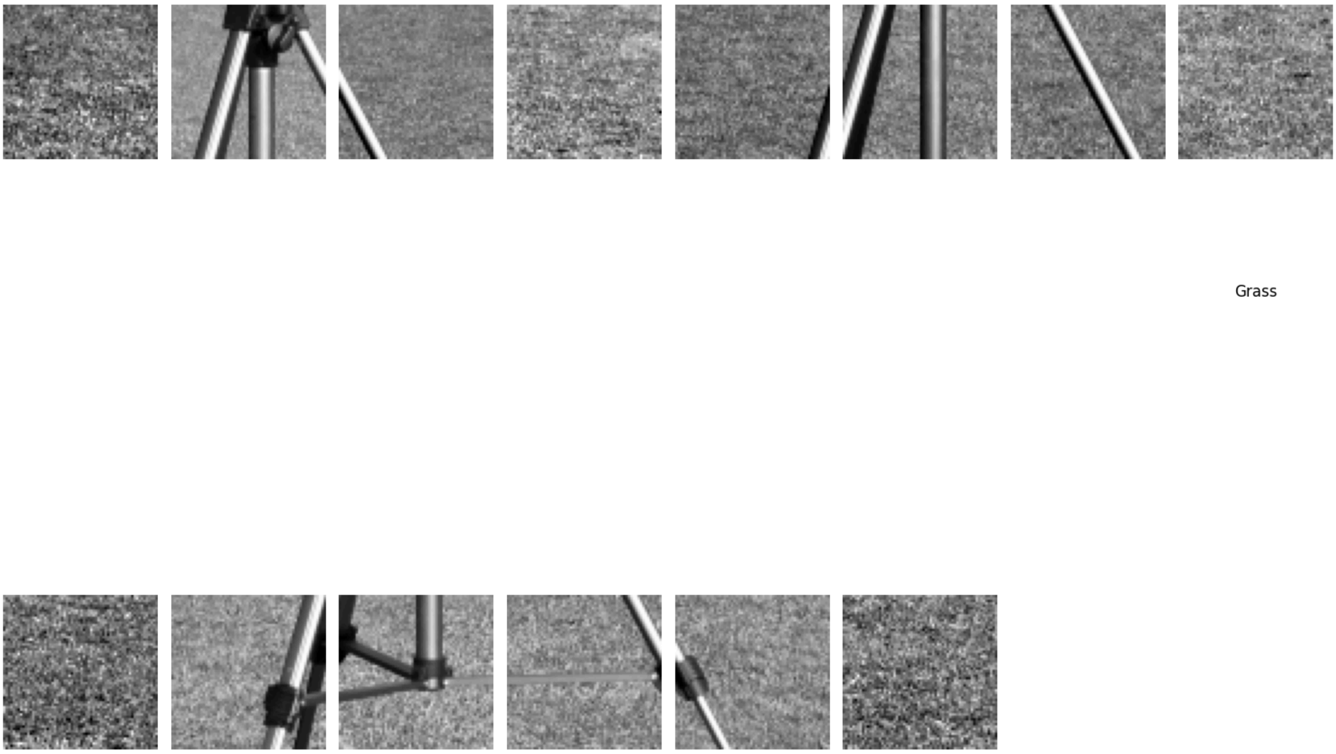
```
from skimage import data, feature, exposure
import math

# Load the cameraman image
image = data.camera()

# Define the patch size
patch_size = (64, 64)  # Example: 64x64 patches

# Get image dimensions
image_height, image_width = image.shape
```

```
# Create a list to hold the image patches
patches = []

# Function to compute GLCM features for a patch
def compute_glcm_features(patch):
    # Compute the GLCM for a patch (with 1 pixel offset in four directions)
    glcm = graycomatrix(patch, distances=[1], angles=[0, np.pi/4, np.pi/2, 3*np.pi/4], symmetric=True, normed=True)

    # Calculate texture features (uniformity and homogeneity)
    uniformity = graycoprops(glcm, 'energy')  # Energy is a proxy for uniformity
    homogeneity = graycoprops(glcm, 'homogeneity')

    # Average over all directions
    uniformity_avg = np.mean(uniformity)
    homogeneity_avg = np.mean(homogeneity)

    return uniformity_avg, homogeneity_avg

# Segment the image into patches
for i in range(0, image_height, patch_size[0]):
    for j in range(0, image_width, patch_size[1]):
        patch = image[i:i+patch_size[0], j:j+patch_size[1]]
        patches.append(patch)

# Filter patches based on texture features
selected_patches = []
threshold_uniformity = 0.2  # Example threshold for uniformity
threshold_homogeneity = 0.7  # Example threshold for homogeneity

for patch in patches:
    uniformity, homogeneity = compute_glcm_features(patch)

    # Only select patches that meet the uniformity and homogeneity criteria
    if uniformity > threshold_uniformity and homogeneity > threshold_homogeneity:
        selected_patches.append(patch)

# Plot the selected patches (likely to correspond to grass regions)
num_selected_patches = len(selected_patches)
cols = 8  # Define the number of columns for display
rows = math.ceil(num_selected_patches / cols)

fig, axes = plt.subplots(rows, cols, figsize=(15, 15))

# Flatten axes to handle them in a single loop
axes = axes.flatten()

# Loop through selected patches and axes to display them
for i, patch in enumerate(selected_patches):
    axes[i].imshow(patch, cmap='gray')
    axes[i].axis('off')

# Hide any unused axes
for i in range(num_selected_patches, len(axes)):
    axes[i].axis('off')

plt.title('Sky')
plt.tight_layout()
plt.show()
```

Sky

```
from skimage import data, feature
import math

# Load the cameraman image
image = data.camera()

# Define the patch size
patch_size = (64, 64)  # Example: 64x64 patches

# Get image dimensions
image_height, image_width = image.shape
```

```
# Create a list to hold the image patches
patches = []

# Function to compute GLCM features for a patch
def compute_glcm_features(patch):
    # Compute the GLCM for a patch (with 1 pixel offset in four directions)
    glcm = graycomatrix(patch, distances=[1], angles=[0, np.pi/4, np.pi/2, 3*np.pi/4], symmetric=True, normed=True)

    # Calculate texture features (uniformity and homogeneity)
    uniformity = graycoprops(glcm, 'energy')  # Energy is a proxy for uniformity
    homogeneity = graycoprops(glcm, 'homogeneity')

    # Average over all directions
    uniformity_avg = np.mean(uniformity)
    homogeneity_avg = np.mean(homogeneity)

    return uniformity_avg, homogeneity_avg

# Segment the image into patches
for i in range(0, image_height, patch_size[0]):
    for j in range(0, image_width, patch_size[1]):
        patch = image[i:i+patch_size[0], j:j+patch_size[1]]
        patches.append(patch)

# Filter patches based on texture features, specifically targeting rough/complex regions
selected_patches = []
threshold_uniformity = 0.03  # Lower uniformity for rougher patches (lower uniformity = rougher textures)
threshold_homogeneity = 0.1  # Lower homogeneity for rougher patches (lower homogeneity = rougher textures)

for patch in patches:
    uniformity, homogeneity = compute_glcm_features(patch)

    # Select patches that have low uniformity and homogeneity (rougher textures)
    if uniformity < threshold_uniformity and homogeneity < threshold_homogeneity:
        selected_patches.append(patch)

# Plot the selected patches (likely to correspond to rough/complex regions)
num_selected_patches = len(selected_patches)
cols = 8  # Define the number of columns for display
rows = math.ceil(num_selected_patches / cols)

fig, axes = plt.subplots(rows, cols, figsize=(15, 15))

# Flatten axes to handle them in a single loop
axes = axes.flatten()

# Loop through selected patches and axes to display them
for i, patch in enumerate(selected_patches):
    axes[i].imshow(patch, cmap='gray')
    axes[i].axis('off')

# Hide any unused axes
for i in range(num_selected_patches, len(axes)):
    axes[i].axis('off')

plt.title('Grass')
plt.tight_layout()
plt.show()
```




Double-click (or enter) to edit