Supplementary information on textural features

This Jupyter notebook provides an explanation of the Gray-Level Co-occurrence Matrix (GLCM) and associated textural properties or features, and their calculation, using four distinctive example patterns. The notebook can be run in Python 3 after installation of the imported libraries or in Google Collab.

Disclaimer: this notebook has been developed by D. Benet with help by ChatGPT.

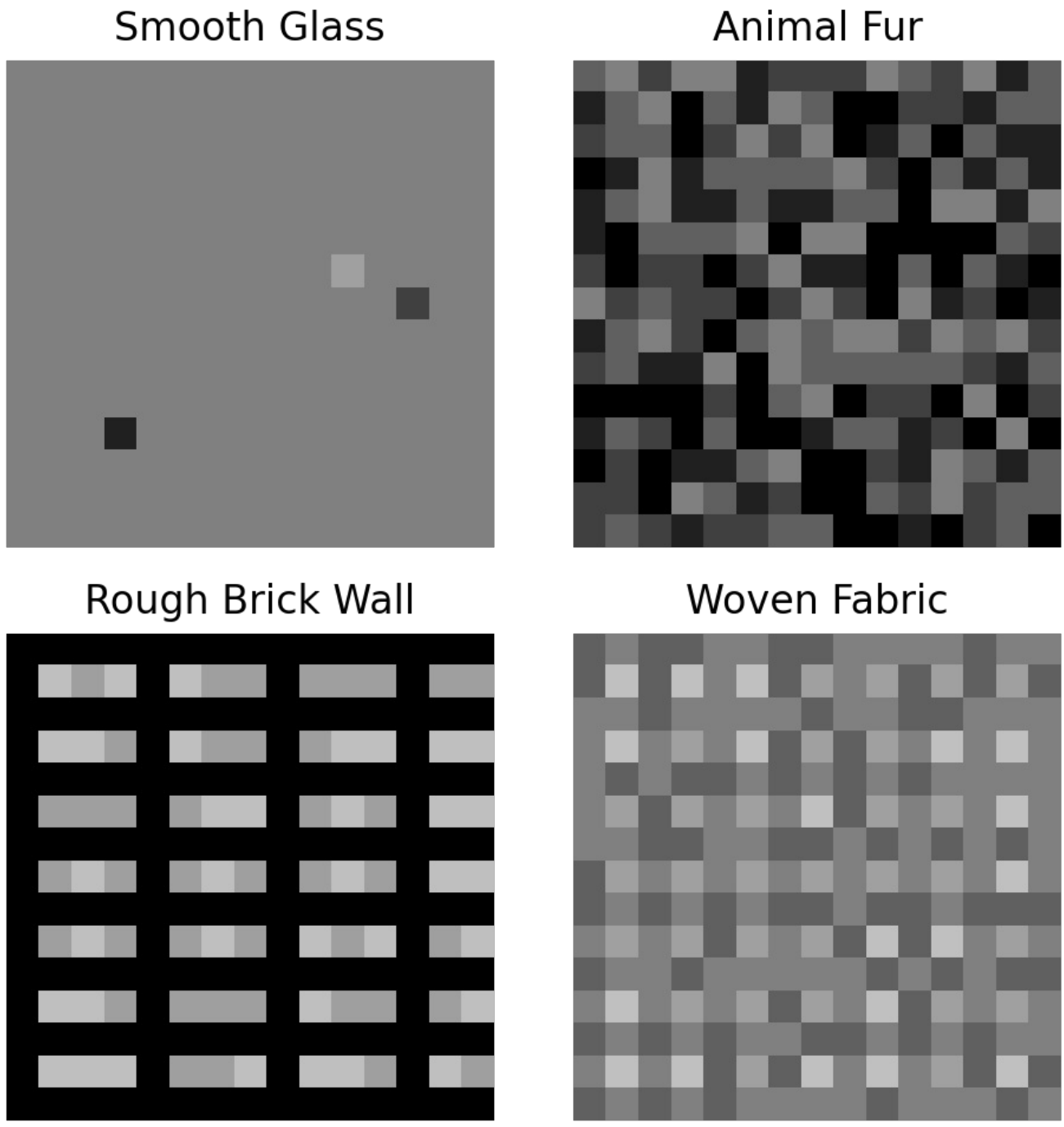
1. Haralick's Gray-Level Co-occurrence Matrix (GLCM)

The GLCM was first introduced by Haralick et al. (1973) and quantifies the spatial relationships between pairs of pixels based on their intensity values, providing valuable information about texture and patterns within an image. In this notebook the following patterns are considered:

- `smooth glass` has a uniform surface texture with few variations in color.
- `animal fur` has a complex texture with a mix of different hair lengths and patterns.
- `rough brick wall` has a coarse and irregular texture due to the rough surfaces of the bricks and mortar joints.
- `woven fabric` has a distinct texture characterized by the interlacing of threads, resulting in a patterned surface.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200
from skimage.feature import greycomatrix, greycoprops
import warnings
warnings.filterwarnings('ignore')

# Set the random seed to a specific value (e.g., 42)
np.random.seed(42)

# Create a smooth glass surface example image with slight variations
smooth_glass = np.ones((15, 15)) * 4  # Set all pixels to 4

# Add slightly different values to some pixels
smooth_glass[3, 5] = 4
smooth_glass[7, 12] = 2
smooth_glass[6, 10] = 5
smooth_glass[11, 3] = 1

# Create an animal fur example image
furry_fur = np.random.randint(0, 5, size=(15, 15))  # Random texture with lower values (0-4) for soft fur

# Generate random noise to add to images
noise = np.random.randint(1, 3, size=(15, 15))

# Create a rough brick wall example image
rough_brick_wall = np.ones((15, 15), dtype=int) * 4

for i in range(0, 15, 2):
    rough_brick_wall[i, :] = np.random.randint(-2, -1)  # Randomly add lines with values from -2 to -1

for i in range(0, 15, 4):
    rough_brick_wall[:, i] = np.random.randint(-2, -1)  # Randomly add lines with values from -2 to -1

# Add noise
rough_brick_wall = rough_brick_wall + noise

# Create a woven fabric example image
woven_fabric = np.ones((15, 15)) * 4  # Set all pixels to 4 (uniform texture)

for i in range(0, 15, 2):
    woven_fabric[i, :] = 2  # Randomly add lines with values from 5 to 8

for i in range(0, 15, 2):
    woven_fabric[:, i] = 2  # Randomly add lines with values from 5 to 8

# Add noise
woven_fabric = woven_fabric + noise

# Plot the example images
fig, axs = plt.subplots(2, 2, figsize=(5, 5))

axs[0, 0].imshow(smooth_glass, cmap='gray', vmin=0, vmax=8)
axs[0, 0].set_title('Smooth Glass')
axs[0, 0].axis('off')

axs[0, 1].imshow(furry_fur, cmap='gray', vmin=0, vmax=8)
axs[0, 1].set_title('Animal Fur')
axs[0, 1].axis('off')

axs[1, 0].imshow(rough_brick_wall, cmap='gray', vmin=0, vmax=8)
```

```
axs[1, 0].set_title('Rough Brick Wall')
axs[1, 0].axis('off')

axs[1, 1].imshow(woven_fabric, cmap='gray', vmin=0, vmax=8)
axs[1, 1].set_title('Woven Fabric')
axs[1, 1].axis('off')

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



2. Calculation of the GLCM Matrices

The GLCM is a two-dimensional matrix that represents the frequency of occurrence of pairs of pixel intensity values at a specific distance and angle in an image. The process of calculating GLCM involves the following steps:

- Choose a distance (d) and an angle (θ) to define the pixel pairs' relative positions. Common angles are 0°, 45°, 90°, and 135°, and distances typically start from 1.
- For each pixel in the image, consider its neighboring pixel at the specified distance and angle.
- Increment the corresponding entry in the GLCM matrix by 1 for each occurrence of a particular pixel pair in the image.

The GLCM is often computed using normalized values, where each entry represents the probability of finding the corresponding pixel pair in the image.

In [ ]:
```
import numpy as np, seaborn as sns
import matplotlib.pyplot as plt
from skimage.feature import graycomatrix, graycoprops
```

```python
# Compute GLCM and texture properties for each image
properties = ['contrast', 'dissimilarity', 'homogeneity', 'correlation', 'ASM', 'energy']
images = [smooth_glass, furry_fur, rough_brick_wall, woven_fabric]
image_names = ["Smooth\nglass", "Furry\nfur", "Rough\nbrick wall", "Woven\nfabric"]

# Function to display GLCM in markdown format
def display_glcm(matrix):
    print("A = ")
    print("\\begin{pmatrix}")
    for row in matrix:
        row_str = " & ".join(map(str, row))
        print(row_str + " \\\\")
    print("\\end{pmatrix}")

for i, float_image in enumerate(images):
    # Rescale float image to the range 0-8
    rescaled_image = (float_image - np.min(float_image)) / (np.max(float_image) - np.min(float_image)) * 8
    # Convert rescaled image to unsigned integer type
    image = rescaled_image.astype(np.uint8)
    #print(image.max())
    glcm = graycomatrix(image, distances=[1], angles=[0], levels=9, symmetric=True, normed=True)
    glcm_int = graycomatrix(image, distances=[1], angles=[0], levels=9, symmetric=True, normed=False)
    #display_glcm(glcm_int[:, :, 0, 0])
```

GLCMs

Smooth glass =

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 2 & 0 & 0 & 0 & 408 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0
\end{pmatrix}$$

Furry fur =

$$\begin{pmatrix}
22 & 0 & 10 & 0 & 25 & 0 & 19 & 0 & 16 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
10 & 0 & 12 & 0 & 9 & 0 & 25 & 0 & 12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
25 & 0 & 9 & 0 & 16 & 0 & 14 & 0 & 18 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
19 & 0 & 25 & 0 & 14 & 0 & 30 & 0 & 18 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
16 & 0 & 12 & 0 & 18 & 0 & 18 & 0 & 8
\end{pmatrix}$$

Rough brick wall =

$$\begin{pmatrix}
24 & 71 & 0 & 0 & 0 & 0 & 11 & 0 & 5 \\
71 & 58 & 0 & 0 & 0 & 0 & 19 & 0 & 14 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
11 & 19 & 0 & 0 & 0 & 0 & 22 & 0 & 28 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
5 & 14 & 0 & 0 & 0 & 0 & 28 & 0 & 20
\end{pmatrix}$$

Woven fabric =

$$\begin{pmatrix} 24 & 0 & 71 & 0 & 0 & 22 & 0 & 0 & 14 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 71 & 0 & 58 & 0 & 0 & 38 & 0 & 0 & 24 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 22 & 0 & 38 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 14 & 0 & 24 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

3.Calculation of Textural Features

Scikit-image's function *graycoprops* is used for calculation of contrast, dissimilarity, homogeneity, energy, correlation and ASM at a distance of 1 pixel and an angle of 0 degrees.

In [ ]:
```
import pandas as pd
import tabulate

images = [smooth_glass, furry_fur, rough_brick_wall, woven_fabric]
image_names_table = ['Smooth glass', 'Animal fur', 'Rough brick wall', 'Woven fabric']
results = []
for i, float_image in enumerate(images):
    # Rescale float image to the range 0-8
    rescaled_image = (float_image - np.min(float_image)) / (np.max(float_image) - np.min(float_image)) * 8
    # Convert rescaled image to unsigned integer type
    image = rescaled_image.astype(np.uint8)
    #print(image.max())
    glcm = graycomatrix(image, distances=[1], angles=[0], levels=9, symmetric=True, normed=True)
    glcm_int = graycomatrix(image, distances=[1], angles=[0], levels=9, symmetric=True, normed=False)
    #display_glcm(glcm_int[:, :, 0, 0])
    texture_props = [graycoprops(glcm, prop)[0, 0] for prop in properties]
    results.append(texture_props)

# Create a DataFrame from the results
df = pd.DataFrame(results, index=image_names_table, columns=[prop.capitalize() for prop in  properties])

# Display the DataFrame as a nice table in Markdown
#print(tabulate.tabulate(df, headers='keys', tablefmt='pipe', showindex=True))
```

In [ ]:
```
df
```

Out[ ]:

	Contrast	Dissimilarity	Homogeneity	Correlation	Asm	Energy
Smooth glass	0.533333	0.114286	0.974151	-0.005472	0.943810	0.971499
Animal fur	16.342857	3.276190	0.282305	-0.033747	0.044898	0.211891
Rough brick wall	9.809524	2.028571	0.497548	0.488325	0.101361	0.318372
Woven fabric	13.980952	2.961905	0.289096	-0.232967	0.110102	0.331816

The obtained textural values strongly differ from one pattern to another. Below, the obtained values are explained based on each textural feature along with the underlying mathematical concepts.

3.1 Contrast

Measures the intensity contrast between neighboring pixels. It increases with the intensity differences between pixel pairs. Higher values indicate more diverse and contrasting textures.

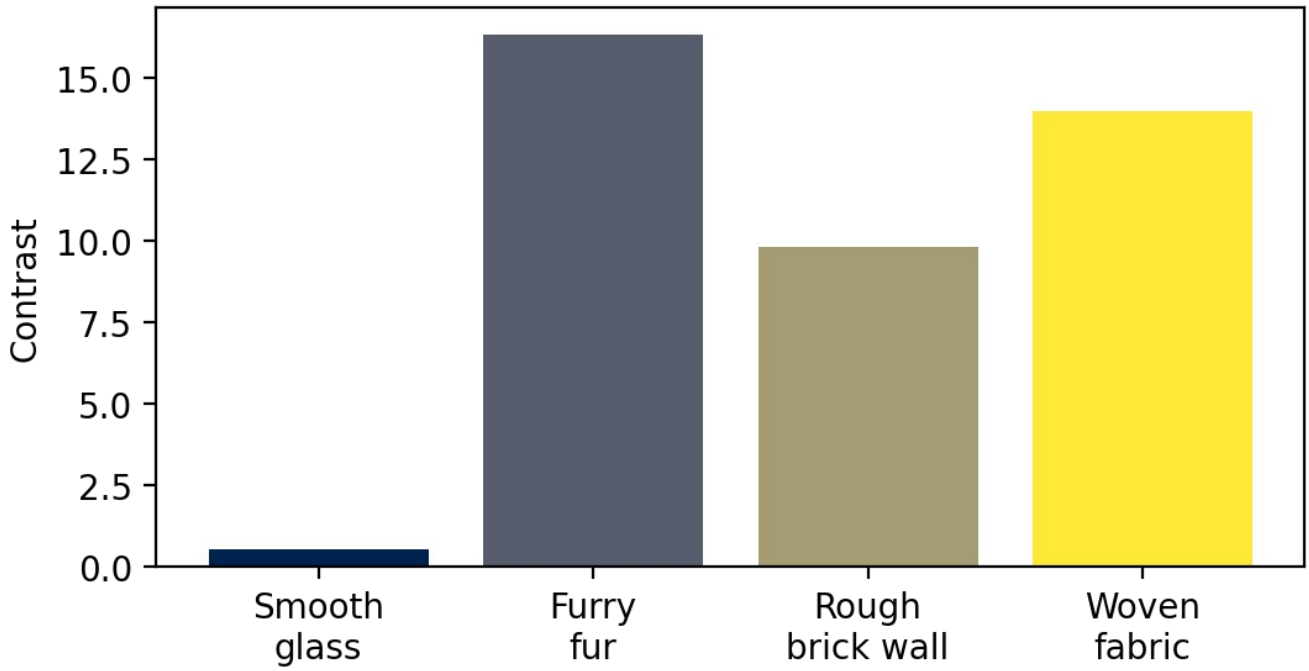
The contrast (C) is calculated using the following formula:

$$C = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P(i,j) \cdot (i-j)^2$$

Here, P(i, j) represents the normalized GLCM element at row i and column j, and N is the number of gray levels in the image. The contrast value is higher for images with sharp transitions and lower for more uniform textures.

In [ ]:
```
# Plot contrast
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Contrast'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel(column.capitalize())
```

```
plt.show()
```



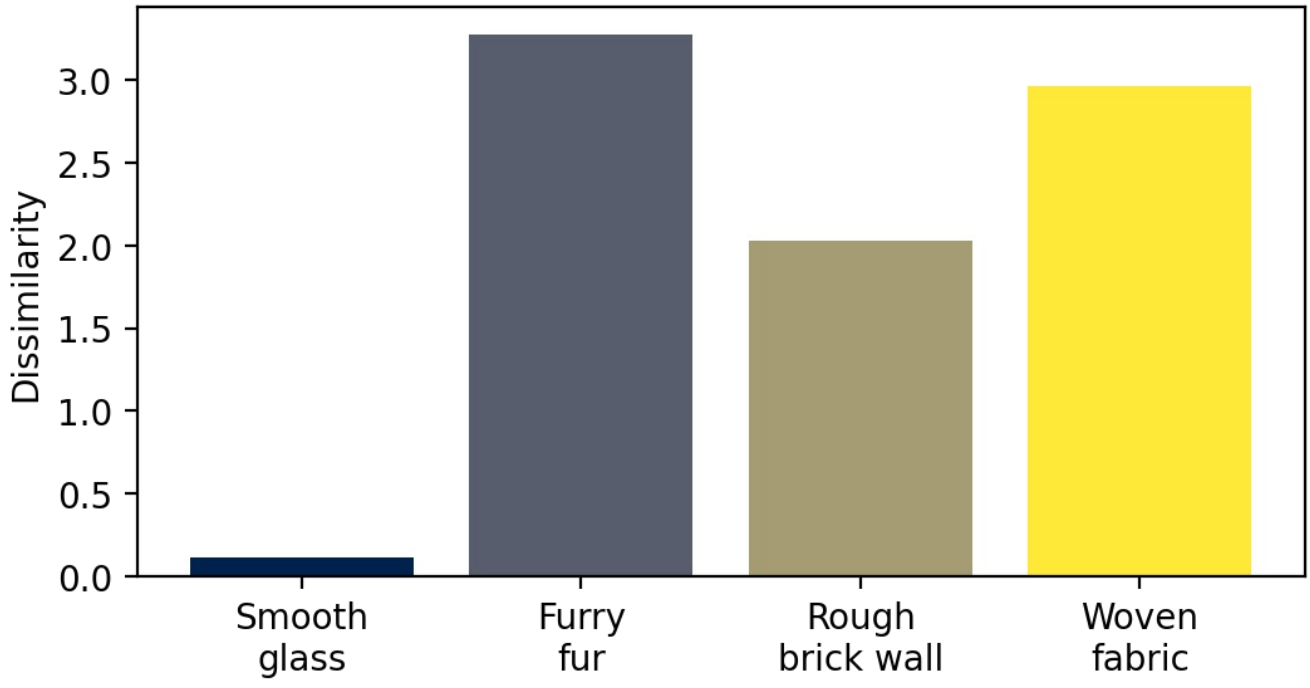
3.2 Dissimilarity

Measures the average difference between pixel pairs in the image. It is similar to contrast but emphasizes the difference between pixel intensities, irrespective of their direction. The dissimilarity value is higher for textures with varying pixel intensities and lower for more uniform textures.

The dissimilarity (D) is calculated using the following formula:

$$D = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P(i,j) \cdot |i-j|$$

In [ ]:
```
# Plot dissimilarity
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Dissimilarity'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel(column.capitalize())
plt.show()
```



3.3 Homogeneity

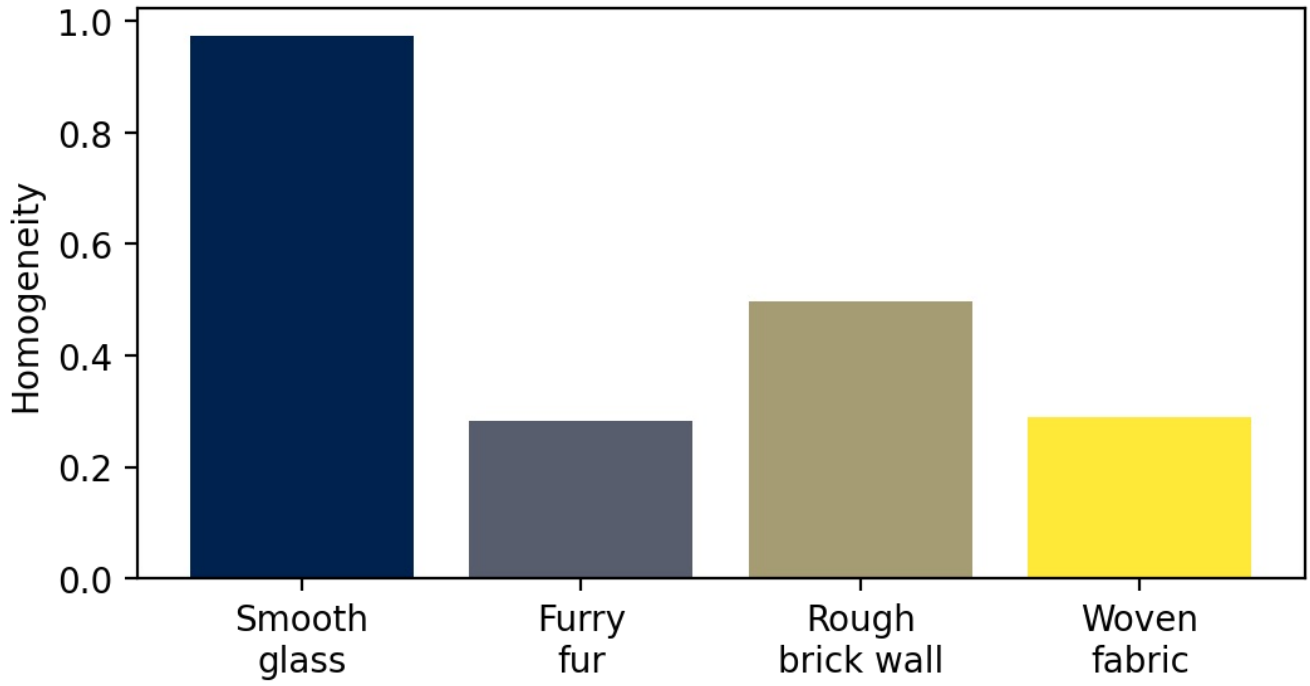
Also known as Inverse Difference Moment, measures the closeness of the GLCM elements to the matrix diagonal. It provides a measure

of how similar neighboring pixels are in intensity. The homogeneity value is higher for textures with more similar pixel intensities, indicating smoother and more regular textures.

The homogeneity (H) is calculated using the following formula:

$$H = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{P(i,j)}{1 + |i-j|}$$

In [ ]:
```
# Plot homogeneity
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Homogeneity'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel(column.capitalize())
plt.show()
```



3.4 Correlation

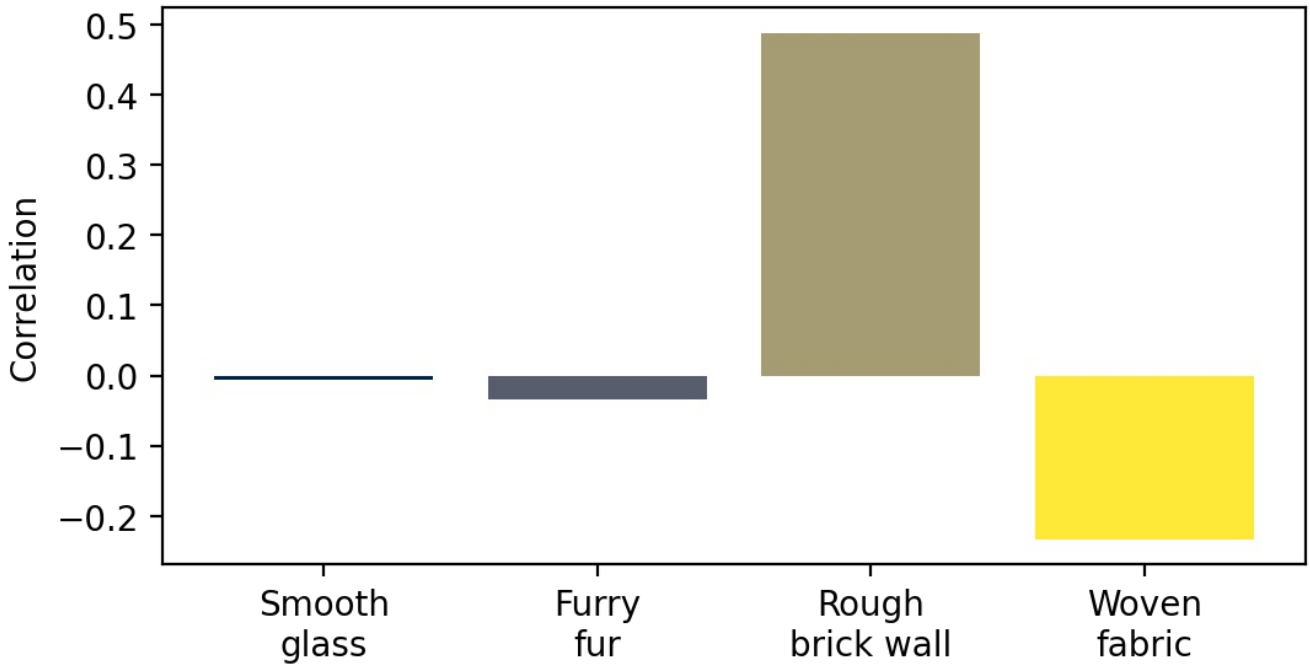
Measures the linear dependency between pixel pairs in the image. It provides an indication of how much the pixel values are correlated with their neighbors.

The correlation (R) is calculated using the following formula:

$$R = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \frac{P(i,j) \cdot (i-\mu)(j-\mu)}{\sigma_i \cdot \sigma_j}$$

Here, μ represents the mean of the GLCM, and σ_i and σ_j represent the standard deviations along the rows and columns of the GLCM, respectively. The correlation value is higher for textures with a strong linear relationship between pixel values.

In [ ]:
```
# Plot homogeneity
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Correlation'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel(column.capitalize())
plt.show()
```



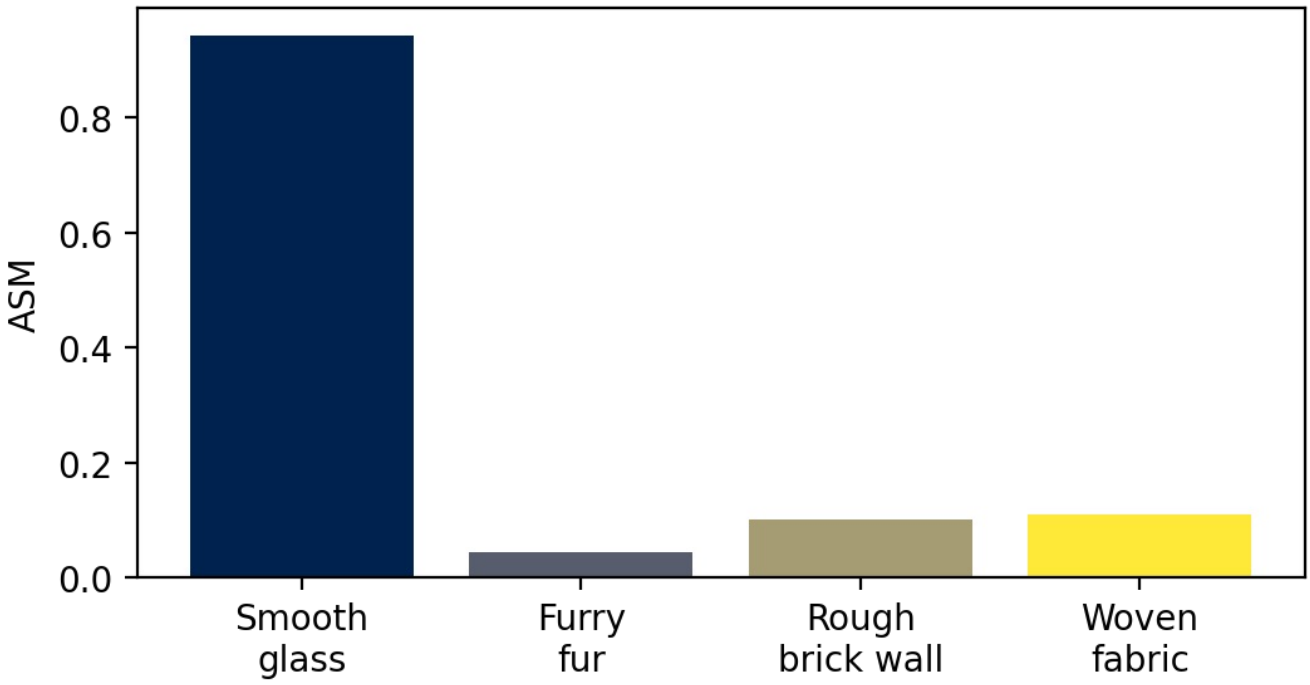
3.5 Angular Second Moment (ASM)

Measures the uniformity or regularity of the image texture. It is directly related to the GLCM's entropy and is a measure of the texture's orderliness. The ASM value is higher for images with more homogenous and regular textures and lower for images with more complex and disordered textures.

The ASM is calculated using the following formula:

$$ASM = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} P(i,j)^2$$

In [ ]:
```
# Plot ASM
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Asm'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel('ASM')
plt.show()
```



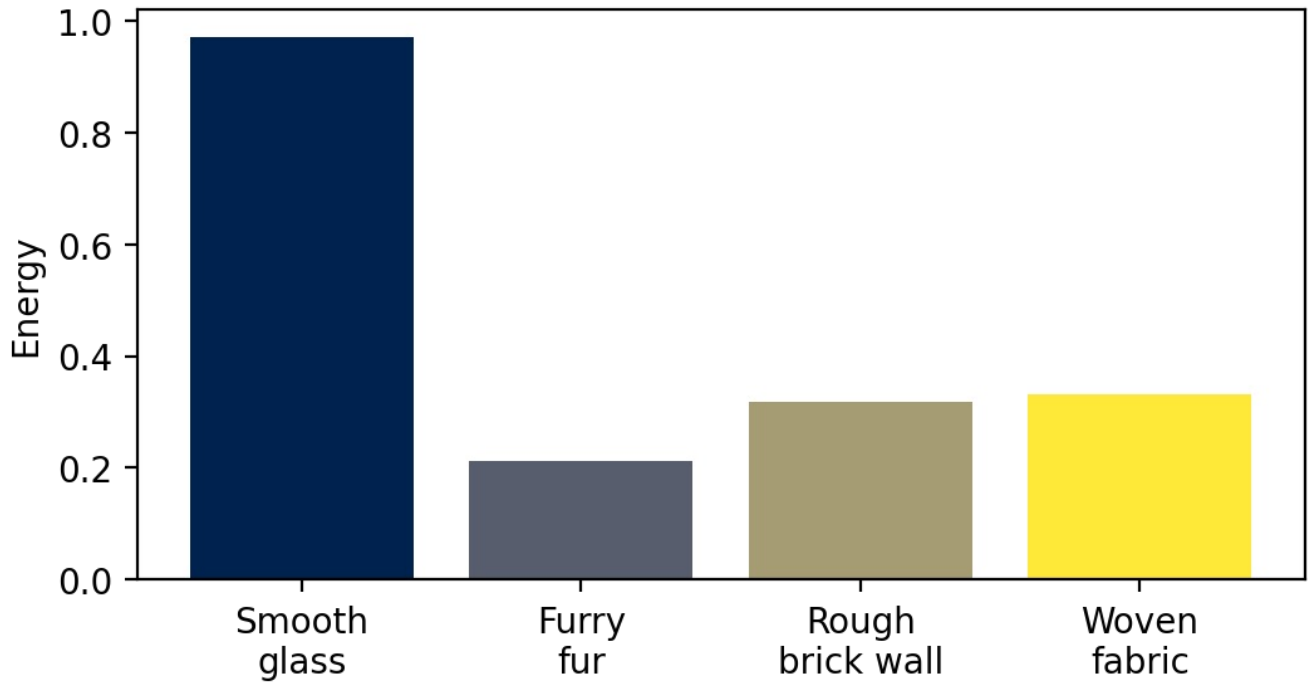
3.6 Energy

It is simply the square root of the Angular Second Moment (ASM). It provides the same information as ASM but is more intuitive because it is on the same scale as the pixel intensities. Energy ranges between 0 and 1, with 1 indicating a very uniform and smooth texture.

The energy (E) is calculated using the following formula:

$$E = \sqrt{ASM}$$

In [ ]:
```
# Plot energy
fig, axs = plt.subplots(figsize=(6, 3))
column = 'Energy'
color = plt.cm.get_cmap('cividis')(np.linspace(0, 1, 4))
axs.bar(image_names, df[column], color=color, linewidth = 2)
axs.set_xticklabels(image_names)
axs.set_ylabel(column)
plt.show()
```



4. Take-away message

In the GLCM, diagonal elements ($(i = j)$) represent the occurrence of pixel pairs with the same intensity value. These elements capture the self-similarity or homogeneity of the texture within an image. Higher diagonal elements indicate a more uniform texture, while lower diagonal elements represent variations in the intensity values, indicating a more complex or heterogeneous texture.

The diagonal elements contribute significantly to textural properties like Homogeneity, Energy, and Correlation, as they emphasize the spatial relationships between pixels with the same intensity. In contrast, off-diagonal elements ($(i \neq j)$) represent pixel pairs with different intensity values and contribute to properties like Contrast and Dissimilarity, reflecting the variation and differences in texture.

In summary, GLCMs are crucial in quantifying and analyzing texture in images. They provide a valuable set of textural properties that help in various image processing tasks, such as feature extraction. The calculation of these properties involves analyzing the frequency of pixel pairs at specific distances and angles and applying mathematical formulas to extract meaningful information about texture from the GLCM. The diagonal elements play a key role in describing the homogeneity and self-similarity of texture, while off-diagonal elements indicate the variations and differences between pixel pairs.