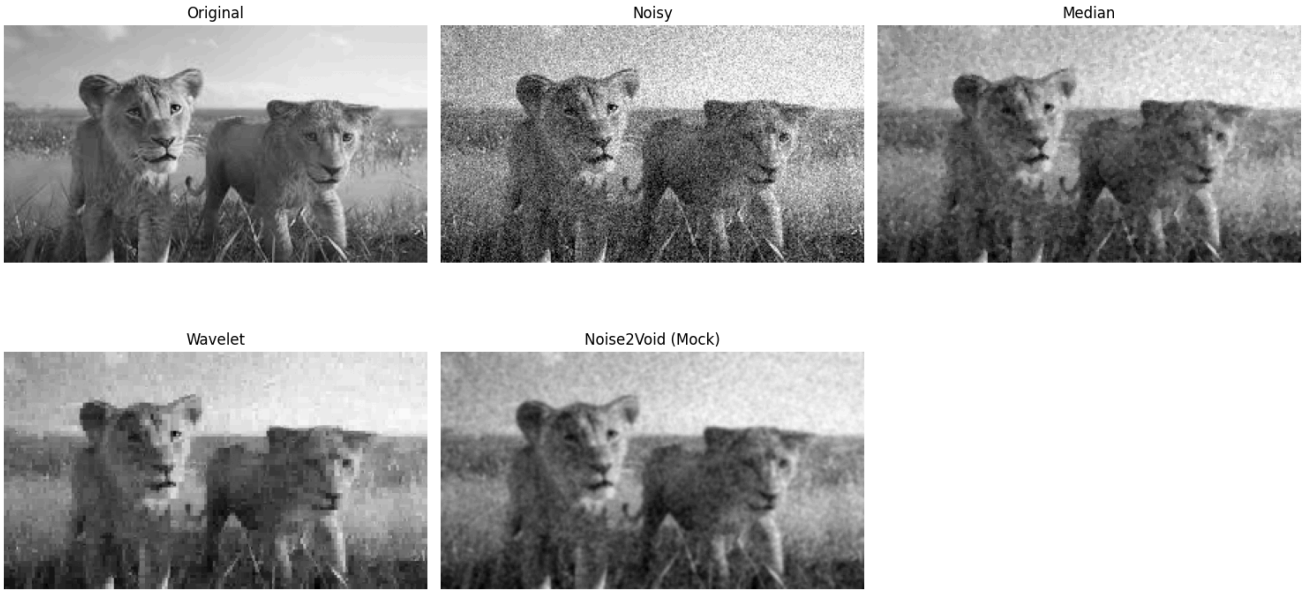
```
!pip install scikit-image opencv-python noise2void --quiet
!pip install pywavelets
```

```
ERROR: Could not find a version that satisfies the requirement noise2void (from versions: none
ERROR: No matching distribution found for noise2void
Requirement already satisfied: pywavelets in /usr/local/lib/python3.11/dist-packages (1.8.0)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from
```

```
# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io, img_as_float, restoration, metrics, color
from skimage.util import random_noise
import pywt

# Load image and convert to grayscale
image = img_as_float(io.imread('/content/image.jpeg'))  # Use your own if needed
if image.ndim == 3:
    # If image has an alpha channel (4 channels), remove it before conversion to grayscale
    if image.shape[2] == 4:
        image = image[:, :, :3]  # Select only the first 3 channels (RGB)
    image = color.rgb2gray(image)

# Add Gaussian noise
noisy_image = random_noise(image, mode='gaussian', var=0.01)

# --- 1. Median Filter Denoising ---
median_denoised = cv2.medianBlur((noisy_image * 255).astype(np.uint8), 3) / 255.0

# --- 2. Wavelet Denoising ---
# Replace 'multichannel' with 'channel_axis' and set to None for grayscale images
wavelet_denoised = restoration.denoise_wavelet(noisy_image, channel_axis=None)


from scipy.ndimage import gaussian_filter
denoised_n2v = gaussian_filter(noisy_image, sigma=1)  # Replace with actual Noise2Void prediction

# --- Metric Calculation Function ---
def evaluate(original, denoised):
    psnr = metrics.peak_signal_noise_ratio(original, denoised)
    # Specify data_range for structural_similarity
    ssim = metrics.structural_similarity(original, denoised, data_range=1.0)
    mse = metrics.mean_squared_error(original, denoised)
    return psnr, ssim, mse

metrics_median = evaluate(image, median_denoised)
metrics_wavelet = evaluate(image, wavelet_denoised)
metrics_n2v = evaluate(image, denoised_n2v)

# --- Print Metrics ---
print("Median Filter:  PSNR={:.2f}, SSIM={:.4f}, MSE={:.6f}".format(*metrics_median))
print("Wavelet Denoise: PSNR={:.2f}, SSIM={:.4f}, MSE={:.6f}".format(*metrics_wavelet))
print("Noise2Void (Mock): PSNR={:.2f}, SSIM={:.4f}, MSE={:.6f}".format(*metrics_n2v))

# --- Visualization ---
titles = ['Original', 'Noisy', 'Median', 'Wavelet', 'Noise2Void (Mock)']
images = [image, noisy_image, median_denoised, wavelet_denoised, denoised_n2v]
```

```
plt.figure(figsize=(15, 8))
for i in range(5):
    plt.subplot(2, 3, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```

```
Median Filter:  PSNR=25.29, SSIM=0.5697, MSE=0.002960
Wavelet Denoise: PSNR=26.21, SSIM=0.6781, MSE=0.002392
Noise2Void (Mock): PSNR=26.89, SSIM=0.6938, MSE=0.002049
```




```
# Upload a .mp4 video
from google.colab import files
uploaded = files.upload()  # Upload your video file here

import cv2
import os

# Create directory to store frames
video_path = next(iter(uploaded))  # Use the uploaded video filename
output_dir = "extracted_frames"
os.makedirs(output_dir, exist_ok=True)

cap = cv2.VideoCapture(video_path)
frame_count = 0

while True:
    ret, frame = cap.read()
    if not ret:
        break
    frame_path = os.path.join(output_dir, f"frame_{frame_count:04d}.png")
    cv2.imwrite(frame_path, frame)
    frame_count += 1

cap.release()
print(f"{frame_count} frames extracted to '{output_dir}'")
```

Choose Files bowtime-bow-time.gif

- **bowtime-bow-time.gif**(image/gif) - 156157 bytes, last modified: 4/29/2025 - 100% done

```
Saving bowtime-bow-time.gif to bowtime-bow-time.gif
25 frames extracted to 'extracted_frames'
```

```
# Create directories for each processing step
os.makedirs("processed/threshold", exist_ok=True)
os.makedirs("processed/gaussian", exist_ok=True)
os.makedirs("processed/canny", exist_ok=True)
os.makedirs("processed/invert", exist_ok=True)

# Reload frames
frame_files = sorted(os.listdir(output_dir))

for fname in frame_files:
    path = os.path.join(output_dir, fname)
    frame = cv2.imread(path)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Adaptive threshold
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)

    # Gaussian smoothing
    blur = cv2.GaussianBlur(gray, (5, 5), 0)

    # Canny edge detection
    edges = cv2.Canny(gray, 100, 200)

    # Bitwise not
    inverted = cv2.bitwise_not(gray)

    # Save processed frames
    cv2.imwrite(f"processed/threshold/{fname}", thresh)
    cv2.imwrite(f"processed/gaussian/{fname}", blur)
    cv2.imwrite(f"processed/canny/{fname}", edges)
    cv2.imwrite(f"processed/invert/{fname}", inverted)


import matplotlib.pyplot as plt

# Select a few sample frames to display
sample_frames = frame_files[:5]  # Change this to show more

for fname in sample_frames:
    path = os.path.join(output_dir, fname)
    frame = cv2.imread(path)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Apply transformations
    thresh = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY, 11, 2)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(gray, 100, 200)
    inverted = cv2.bitwise_not(gray)

    # Plot
    plt.figure(figsize=(12, 6))
    plt.suptitle(f"Processed Versions of Frame: {fname}", fontsize=16)
```

```
        plt.figure(figsize=(12, 6))
        plt.suptitle(f"Processed Versions of Frame: {fname}\n", fontsize=16)

        plt.subplot(1, 5, 1)
        plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        plt.title("Original")
        plt.axis('off')

        plt.subplot(1, 5, 2)
        plt.imshow(thresh, cmap='gray')
        plt.title("Adaptive Threshold")
        plt.axis('off')

        plt.subplot(1, 5, 3)
        plt.imshow(blur, cmap='gray')
        plt.title("Gaussian Blur")
        plt.axis('off')

        plt.subplot(1, 5, 4)
        plt.imshow(edges, cmap='gray')
        plt.title("Canny Edge")
        plt.axis('off')

        plt.subplot(1, 5, 5)
        plt.imshow(inverted, cmap='gray')
        plt.title("Bitwise Not")
        plt.axis('off')

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

<Figure size 1200x600 with 0 Axes>

Processed Versions of Frame: frame_0000.png

<Figure size 1200x600 with 0 Axes>

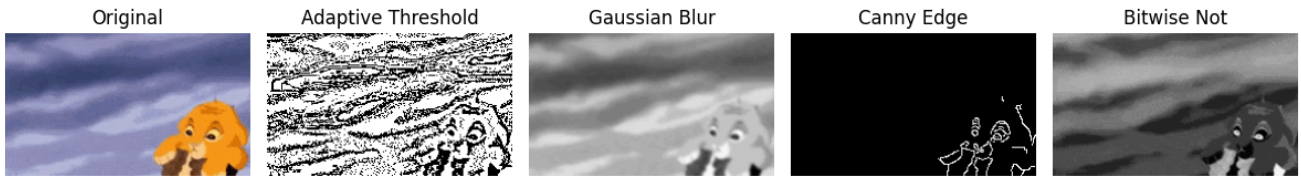
Processed Versions of Frame: frame_0001.png

<Figure size 1200x600 with 0 Axes>

Processed Versions of Frame: frame_0002.png

<Figure size 1200x600 with 0 Axes>

Processed Versions of Frame: frame_0003.png



<Figure size 1200x600 with 0 Axes>

Processed Versions of Frame: frame_0004.png


```
def create_video(folder, output_name, fps=20):
    files = sorted(os.listdir(folder))
    sample = cv2.imread(os.path.join(folder, files[0]))
    height, width = sample.shape[:2]
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_name, fourcc, fps, (width, height), isColor=False)

    for fname in files:
        img = cv2.imread(os.path.join(folder, fname), cv2.IMREAD_GRAYSCALE)
        out.write(cv2.cvtColor(img, cv2.COLOR_GRAY2BGR))

    out.release()

create_video("processed/threshold", "threshold_video.mp4")
create_video("processed/gaussian", "gaussian_video.mp4")
create_video("processed/canny", "canny_video.mp4")
create_video("processed/invert", "invert_video.mp4")
print("Videos created.")
```

```
Videos created.
```

```
from PIL import Image

# Create a 3x3 collage of the first 9 frames
collage_frames = frame_files[:9]
imgs = [Image.open(os.path.join(output_dir, f)) for f in collage_frames]

width, height = imgs[0].size
collage = Image.new('RGB', (3 * width, 3 * height))

for idx, img in enumerate(imgs):
    x = (idx % 3) * width
    y = (idx // 3) * height
    collage.paste(img, (x, y))

collage.save("video_collage.png")
collage.show()

import matplotlib.pyplot as plt
import cv2

# Read and convert the collage image for display
collage_img = cv2.imread("video_collage.png")
```

```
collage_rgb = cv2.cvtColor(collage_img, cv2.COLOR_BGR2RGB)

# Display using matplotlib
plt.figure(figsize=(8, 8))
plt.imshow(collage_rgb)
plt.axis('off')
plt.title("")
plt.show()
```