In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
from scipy.ndimage import convolve
from skimage import data, color, img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.filters import sobel, laplace, gaussian
from scipy.ndimage import convolve
import glob
import random
from collections import defaultdict
import cv2
```

In [ ]:
```
import os
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Natural Data

As natural image, I have partially downloaded a Kaggle dataset. These are mostly images of different animals, human beings and vehicles. I have added some other images collected from online as well. Access my dataset here.

Linear Filter

A linear filter in image processing operates by taking a weighted sum of the pixel values in a neighborhood around each pixel. There are several types of linear filters:

- **Sobel filter** calculates the gradient (rate of change) of intensity in an image, highlighting regions where the intensity changes rapidly, which typically corresponds to edges.

$$\text{Sobel} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- **Laplacian filter** calculates the rate at which the gradient itself changes, making it more sensitive to finer details, such as corners and points where the intensity changes abruptly in multiple directions.

$$\text{Laplacian} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- **Prewitt Filter** is similar to Sobel filter, used for edge detection.

$$\text{Prewitt} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

- **Sharpen Filter** enhances edges by amplifying high-frequency components and also works as edge detectors.

$$\text{Sharpen} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- **Gaussian Filter** smoothens an image by applying a Gaussian function to weight the neighborhood pixels. It is applied directly using a sigma value.

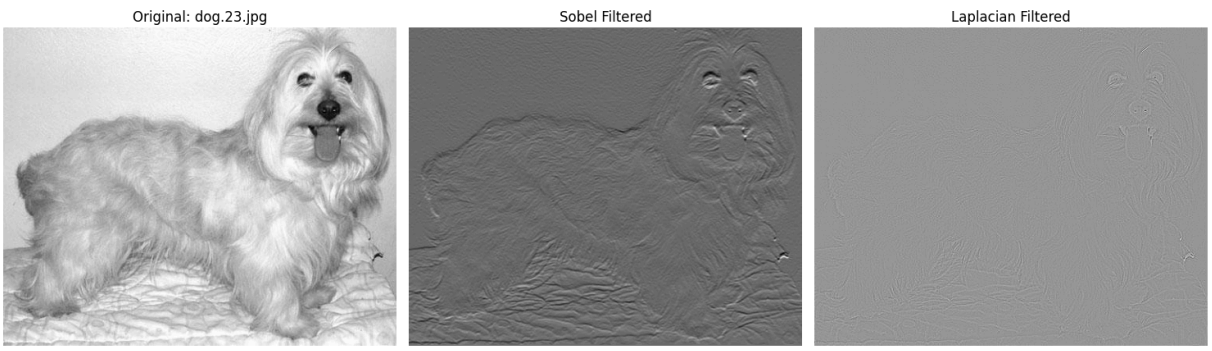
- **Mean Filter** smooths an image by replacing each pixel with the average of its neighbors.

$$\text{Mean} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Let's apply ***Sobel and Laplacian*** filter and visualize their effects

In [ ]:
```
data_dir = '/content/drive/MyDrive/CS216 Files/HW1/data'
image_extensions = ['*.jpg', '*.jpeg', '*.png', '*.bmp', '*.webp']
image_paths = []
for ext in image_extensions:
    image_paths.extend(glob.glob(os.path.join(data_dir, '**', ext), recursiv


# Group images by subfolder
subfolder_images = defaultdict(list)
for image_path in image_paths:
    subfolder = os.path.dirname(image_path)
    subfolder_images[subfolder].append(image_path)


Sobel = np.array([[-1, -2, -1],
                  [ 0,  0,  0],
                  [ 1,  2,  1]])
Laplacian = np.array([[0,  1,  0],
```

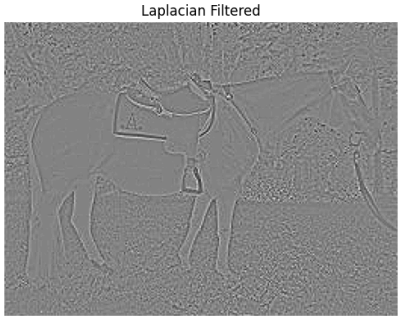
```
                        [1, -4,  1],
                        [0,  1,  0]])


selected_images = [random.choice(images) for images in subfolder_images.valu

for image_path in selected_images:
    try:
        image = imread(image_path)
        if image.ndim == 3:
            image = color.rgb2gray(img_as_float(image))
    except Exception as e:
        print(f"Error loading image {image_path}: {e}")
        continue

    sobel_filtered = cv2.filter2D(image, -1, Sobel)

    laplacian_filtered = cv2.filter2D(image, -1, Laplacian)

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Show original image
    axes[0].imshow(image, cmap='gray')
    axes[0].set_title(f'Original: {os.path.basename(image_path)}')
    axes[0].axis('off')

    # Show Sobel filtered image
    axes[1].imshow(sobel_filtered, cmap='gray')
    axes[1].set_title('Sobel Filtered')
    axes[1].axis('off')

    # Show Laplacian filtered image
    axes[2].imshow(laplacian_filtered, cmap='gray')
    axes[2].set_title('Laplacian Filtered')
    axes[2].axis('off')

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



Original: horse-19.jpg

Sobel Filtered

Laplacian Filtered



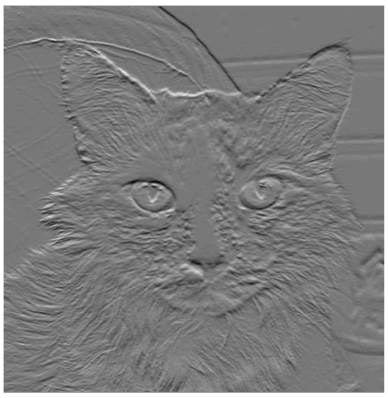
Original: rider-8.jpg

Sobel Filtered

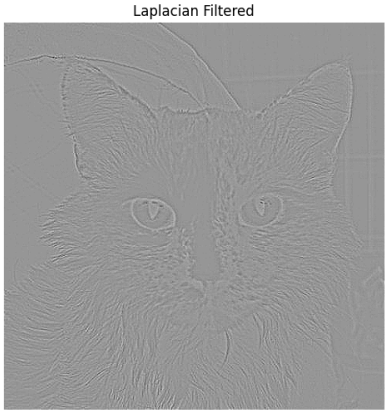
Laplacian Filtered

Original: cat.10.jpg

Sobel Filtered



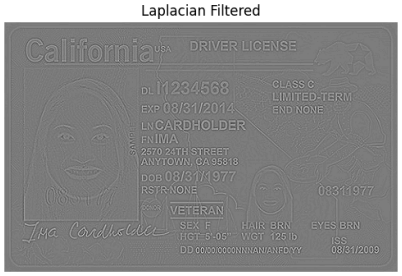
Laplacian Filtered

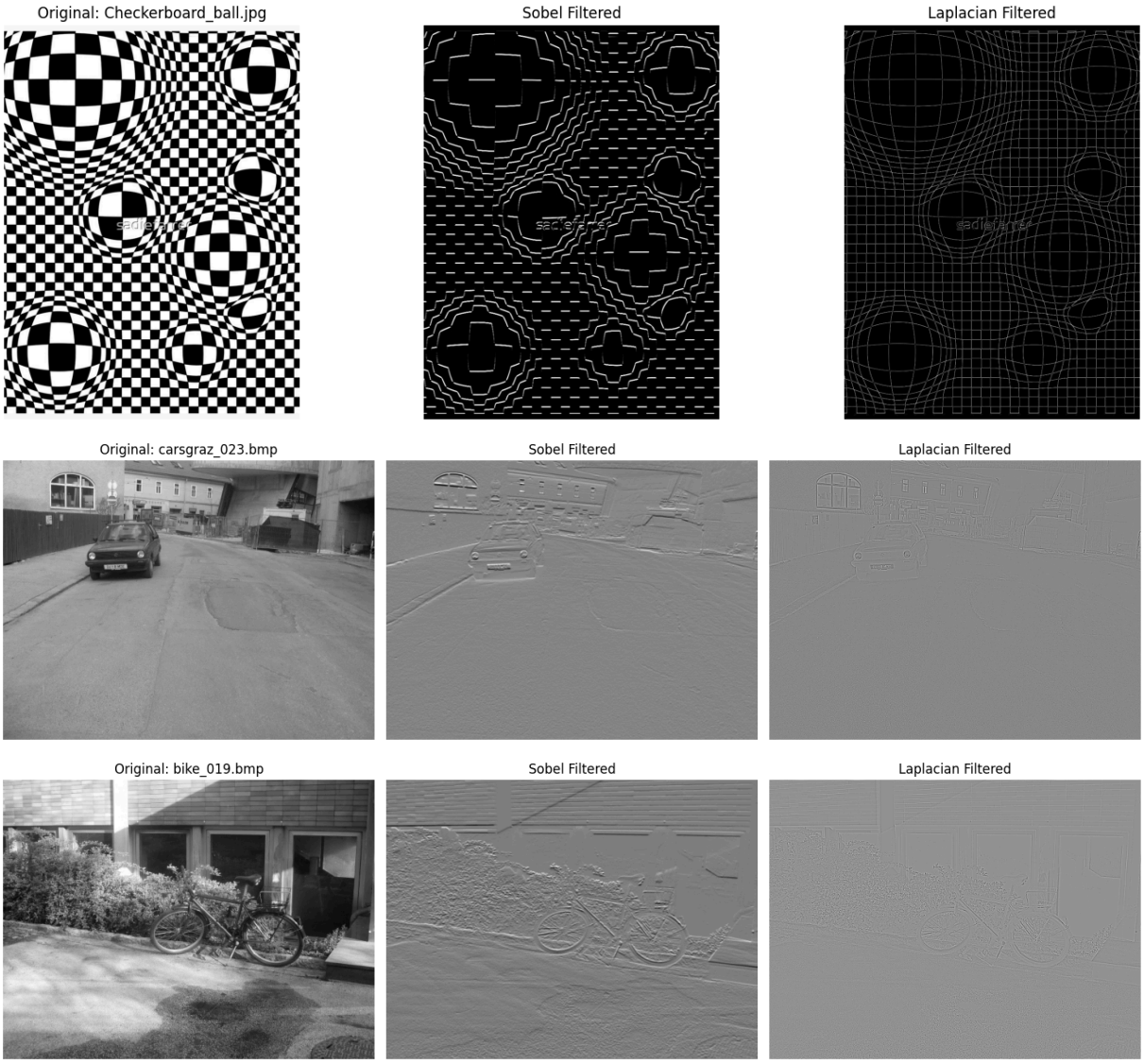


Original: driving-license.webp

Sobel Filtered

Laplacian Filtered





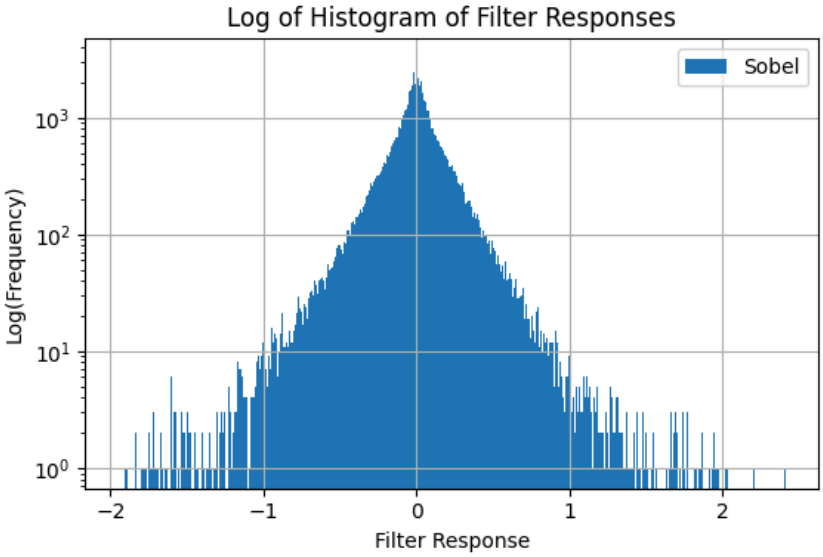
Statistics of Filter Responses

In [ ]:
```
filter_kernels = {
    'Sobel': np.array([[-1, -2, -1],
                       [ 0,  0,  0],
                       [ 1,  2,  1]]),

    'Laplacian': np.array([[ 0,  1,  0],
                           [ 1, -4,  1],
                           [ 0,  1,  0]]),
    'Prewitt': np.array([[-1, -1, -1],
                         [ 0,  0,  0],
                         [ 1,  1,  1]]),
    'Sharpen': np.array([[ 0, -1,  0],
                         [-1,  5, -1],
                         [ 0, -1,  0]]),

    'Gaussian': None,  # Gaussian will be applied directly with a specified

    'Mean': np.ones((3, 3)) / 9,
```

```
}

def filter_responses(image, kernel_name, draw = False):

    if image.ndim == 3:
        image = color.rgb2gray(img_as_float(image))

    filtered_image = None
    if filter_kernels[kernel_name] is not None:
        filtered_image = convolve(image, filter_kernels[kernel_name])
    else:
        filtered_image = gaussian(image, sigma=0.5)

    if draw:

        filter_responses = filtered_image.flatten()

        fig, axes = plt.subplots(1, 2, figsize=(10, 4))

        axes[0].imshow(image, cmap='gray')
        axes[0].set_title('Original Image')
        axes[0].axis('off')

        axes[1].hist(filter_responses, bins=1000, log=True, label=kernel_nam
        axes[1].set_title('Log of Histogram of Filter Responses')
        axes[1].set_xlabel('Filter Response')
        axes[1].set_ylabel('Log(Frequency)')
        axes[1].legend()
        axes[1].grid(True)

        plt.tight_layout()
        plt.show()

    return filtered_image
```

A filter on an image

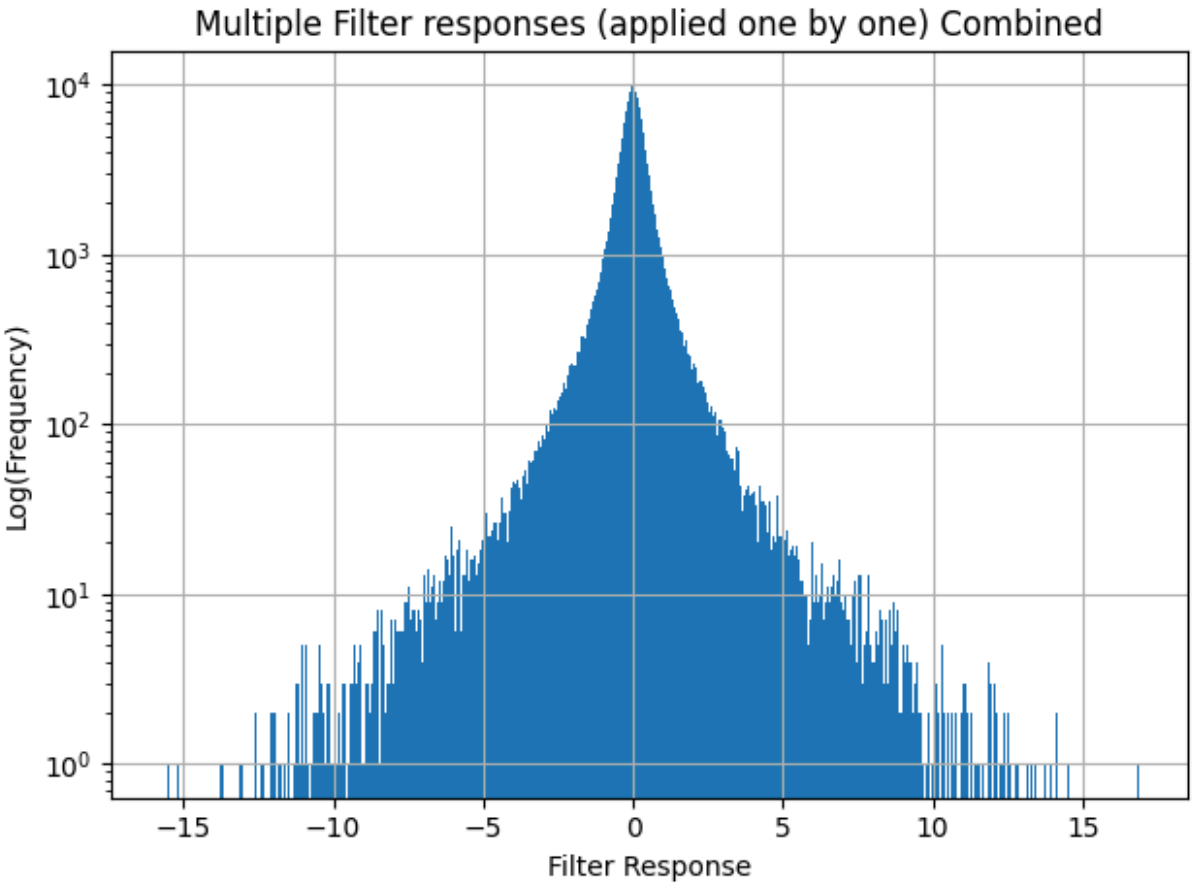
In [ ]:
```
image_paths = glob.glob(os.path.join(data_dir, '**', '*.*'), recursive=True)
image = imread(random.choice(image_paths))

filtered_image = filter_responses(image, 'Sobel' , True)
```



A set of filters on an image

In [ ]:

```
image = imread(random.choice(image_paths))

#show image
if image.ndim == 3:
    image = color.rgb2gray(img_as_float(image))
plt.figure(figsize=(4, 4))
plt.imshow(image, cmap='gray')
plt.title(f'Original Image')
plt.axis('off')
plt.show()

#apply multiple filters one by one
for filter in filter_kernels:
    filtered_image = filter_responses(image , filter)
    image = filtered_image

response = image.flatten()

#show histogram
plt.hist(response, bins=1000, log=True)
plt.title(f'Multiple Filter responses (applied one by one) Combined')
plt.xlabel('Filter Response')
plt.ylabel('Log(Frequency)')
plt.grid(True)
plt.tight_layout()
plt.show()
```

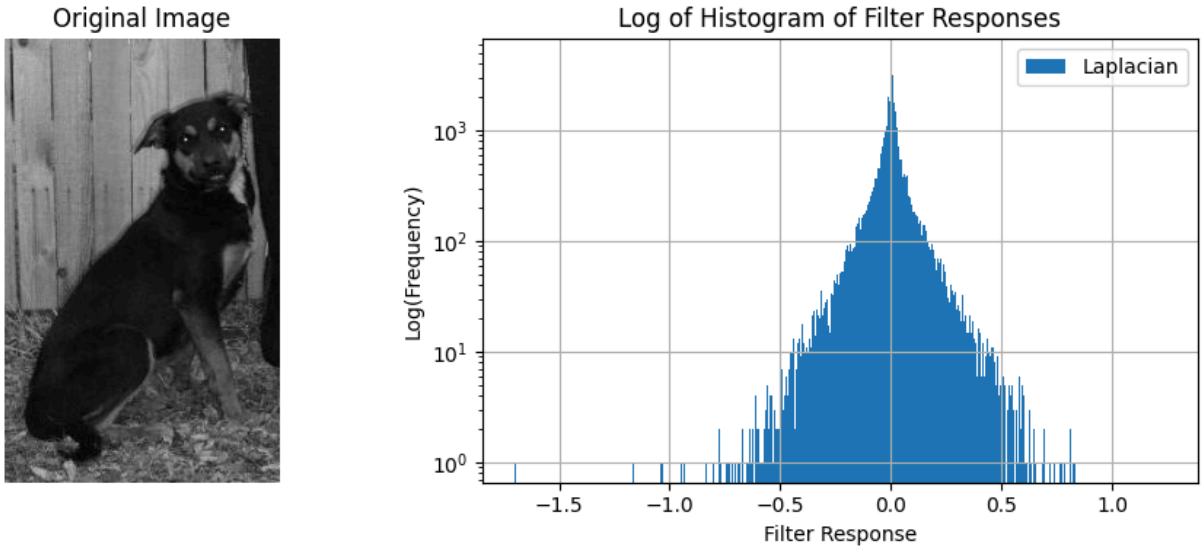


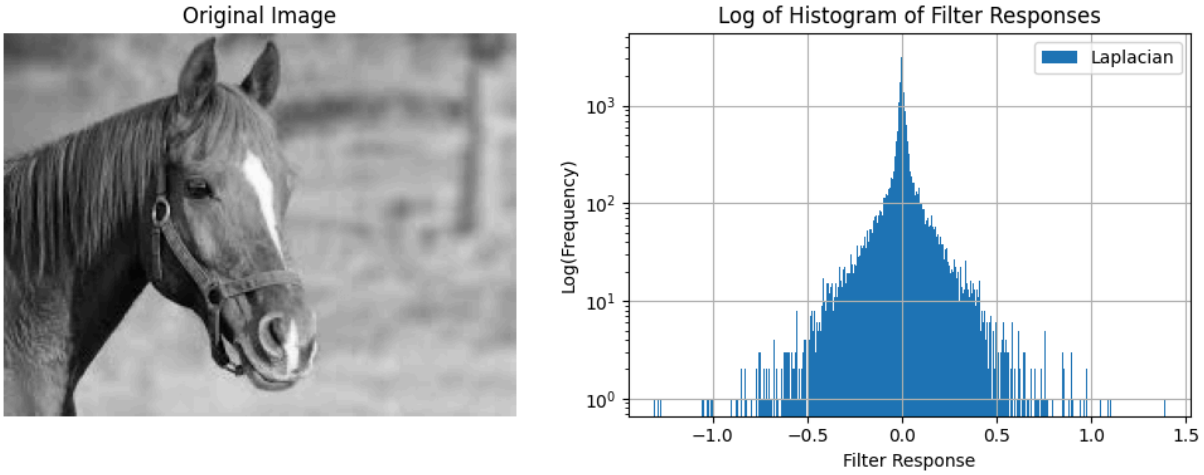
The six linear filters (Sobel, Laplacian, Gaussian, Prewitt, Mean and Sharpen) were applied sequentially to a randomly selected image, and the resulting histogram of the final filter responses is displayed in the diagram above.

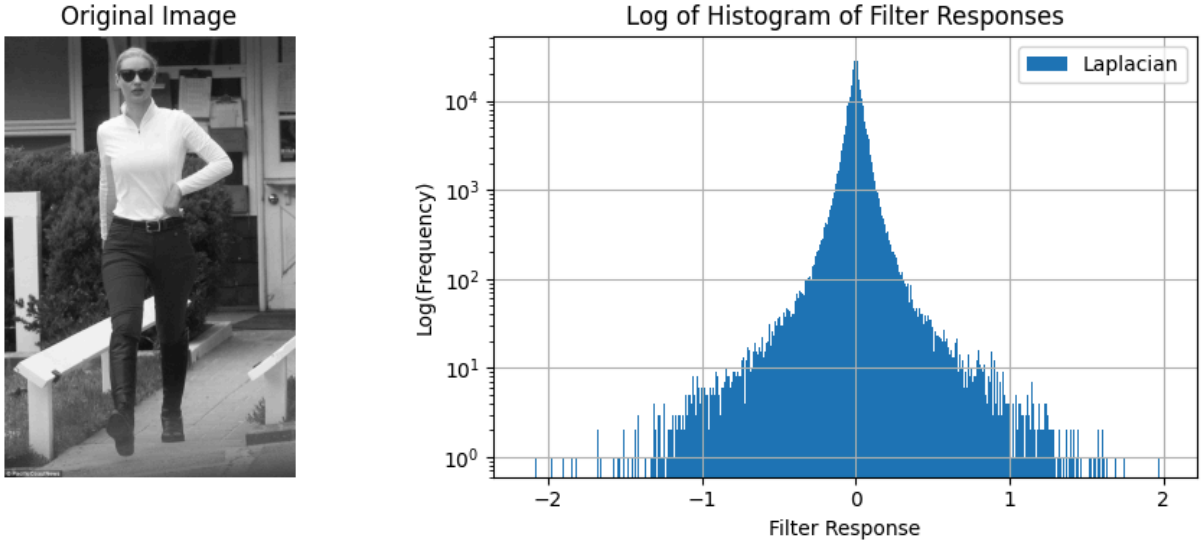
A filter on a set of images

A single image from each type was randomly selected, and the **Laplacian filter** was applied to each of them.

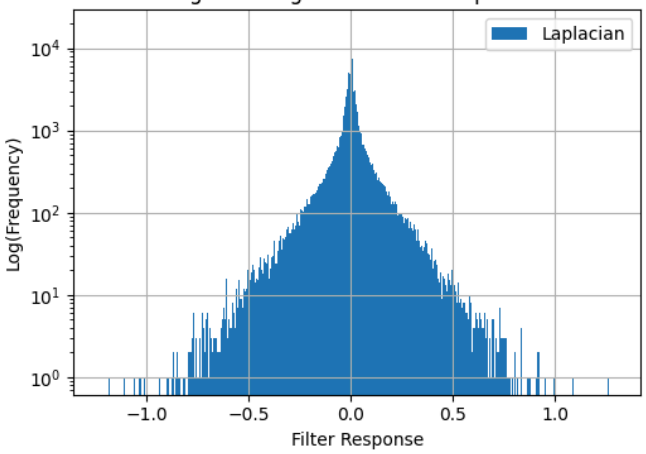
In [ ]:
```
selected_images = [random.choice(images) for images in subfolder_images.valu

for image_path in selected_images:
    image = imread(image_path)
    filter_responses(image, 'Laplacian', draw=True)
```



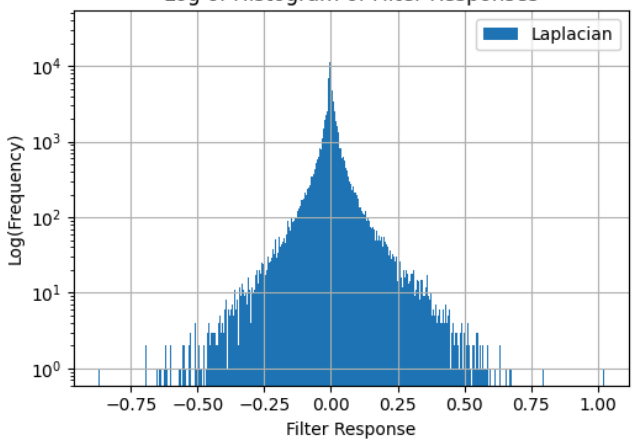




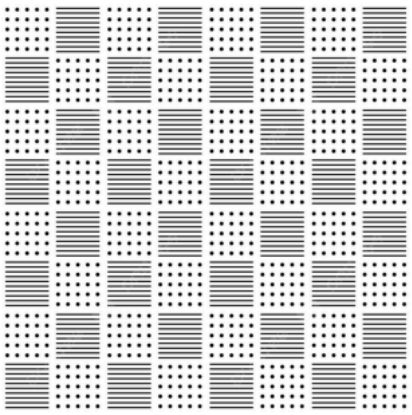
Original Image

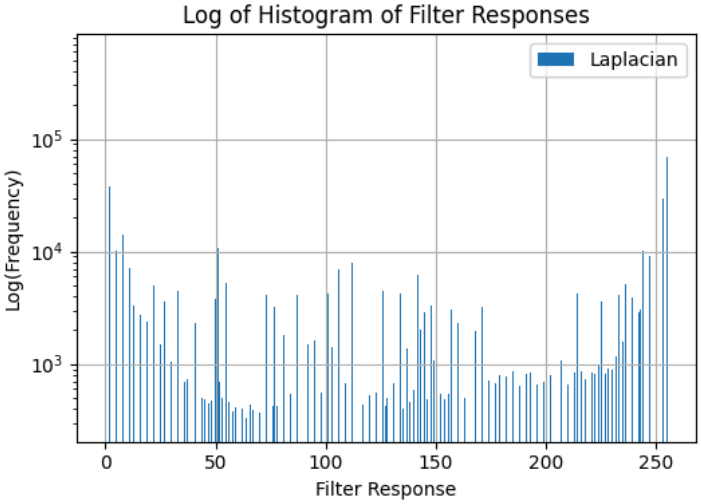


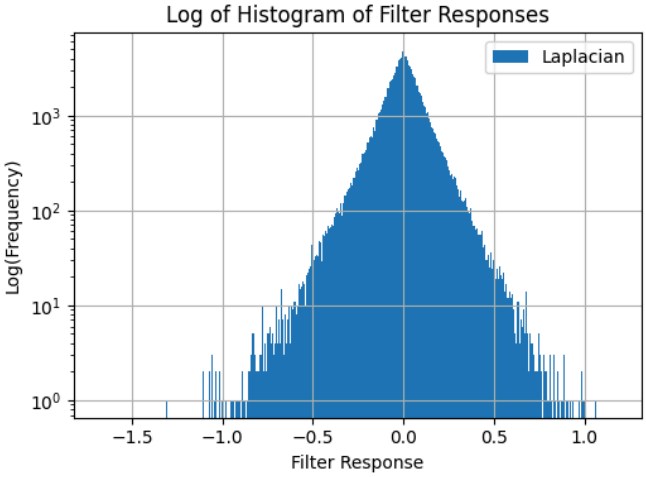
Original Image

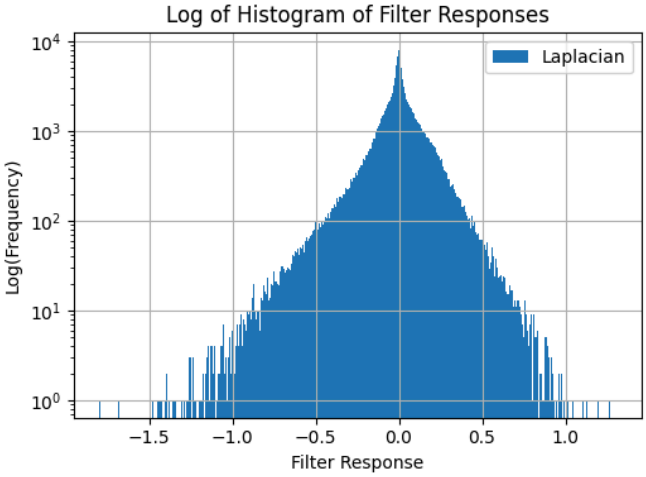


Original Image









A set of filters on a set of images

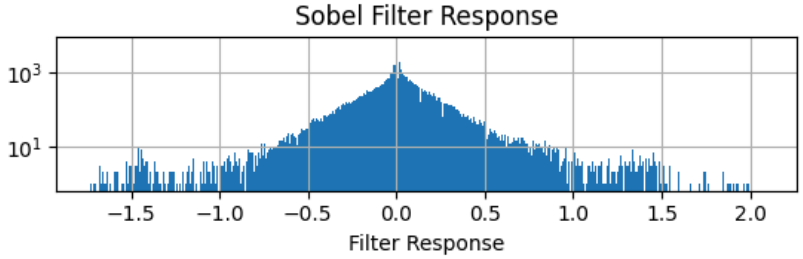
Six images were randomly selected from each type, and if a type contained fewer than six images, all available images were selected. Each of the selected images from all types was then processed using all six filters (Sobel, Laplacian, Gaussian, Prewitt, Mean, and Complete).

In [ ]:
```
# Method to process a single subfolder
def process_image_set(subfolder_path):

    image_paths = glob.glob(os.path.join(subfolder_path, '*.*'))
    selected_images = random.sample(image_paths, min(6, len(image_paths)))

    num_images = len(selected_images)
    fig, axes = plt.subplots(num_images, 2, figsize=(10, num_images * 2))
    fig.suptitle(f'Images of: {os.path.basename(subfolder_path)}', fontsize=

    for i, image_path in enumerate(selected_images):
        try:
            image = imread(image_path)
            if image.ndim == 3:
                image = color.rgb2gray(img_as_float(image))
```

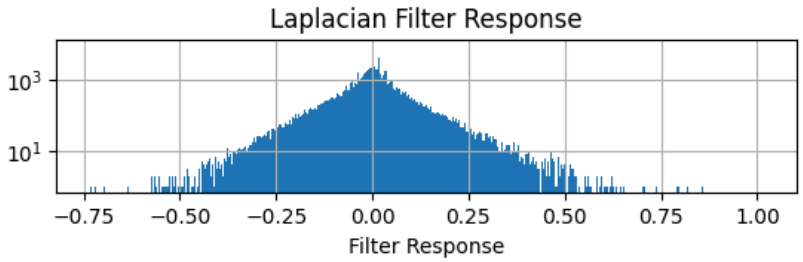
```
            # image = cv2.cvtColor(imread(image_path), cv2.COLOR_BGR2GRAY)
        except Exception as e:
            print(f"Error loading image {image_path}: {e}")
            continue

        filter_name = list(filter_kernels.keys())[i]
        kernel = filter_kernels[filter_name]
        if kernel is None:  # Apply Gaussian filter directly
            filtered_image = gaussian(image, sigma=1)
        else:
            filtered_image = convolve(image, kernel)

        axes[i, 0].imshow(image, cmap='gray')
        axes[i, 0].set_title(f'Original: {os.path.basename(image_path)}')
        axes[i, 0].axis('off')

        filter_responses = filtered_image.flatten()
        axes[i, 1].hist(filter_responses, bins=1000, log=True)
        axes[i, 1].set_title(f'{filter_name} Filter Response')
        axes[i, 1].set_xlabel('Filter Response')
        axes[i, 1].grid(True)

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.show()

for subfolder in [f.path for f in os.scandir(data_dir) if f.is_dir()]:
    process_image_set(subfolder)
```

Images of: dogs

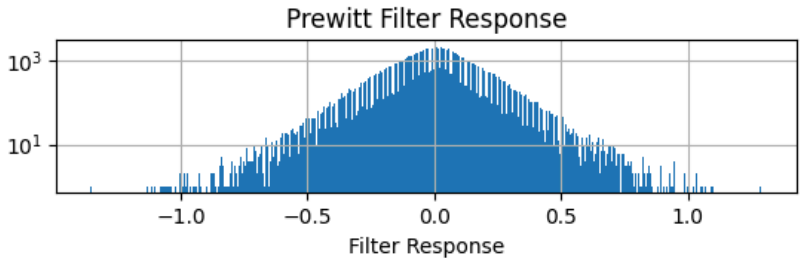
Original: dog.17.jpg



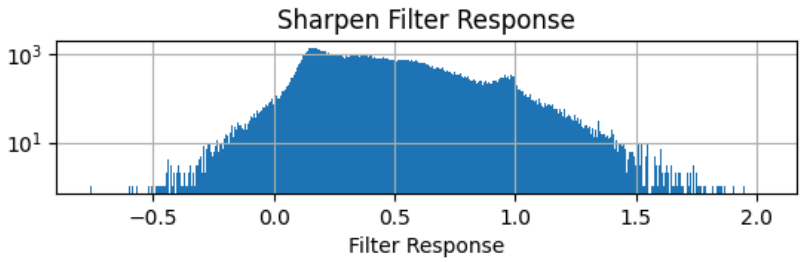
Original: dog.3.jpg



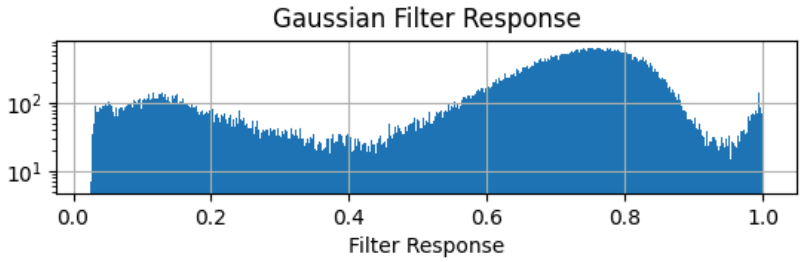
Original: dog.14.jpg



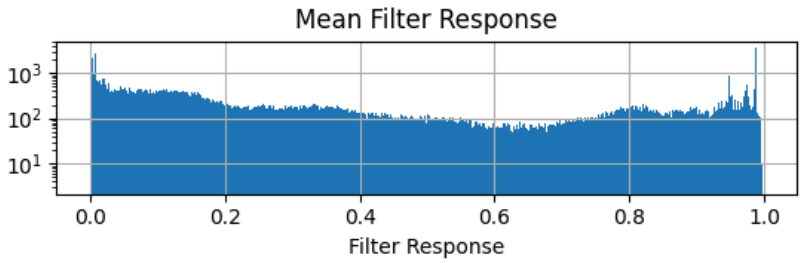
Original: dog.8.jpg



Original: dog.20.jpg

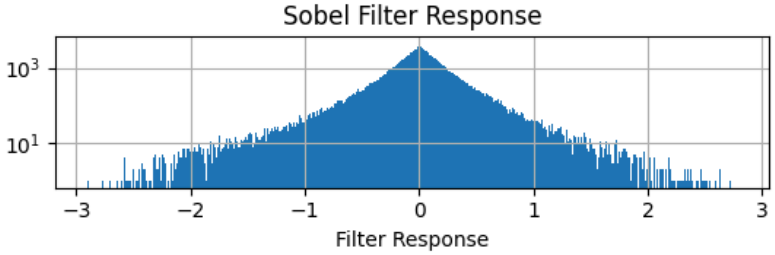


Original: dog.21.jpg

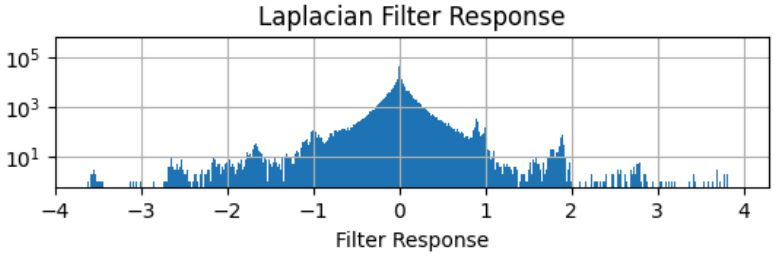


Images of: horses

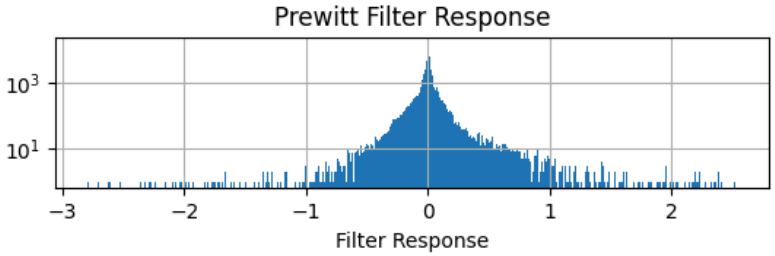
Original: horse-9.jpg



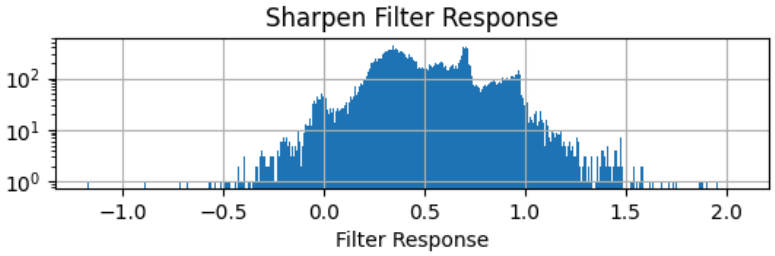
Original: horse-13.jpg



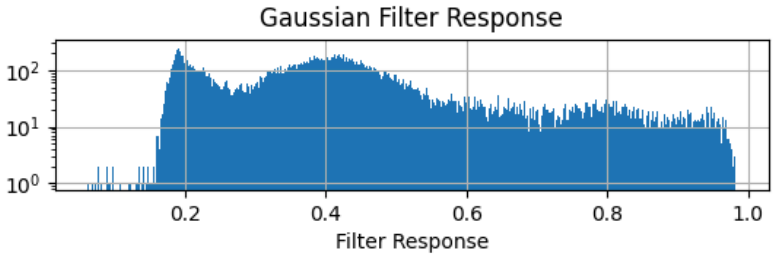
Original: horse-3.jpg



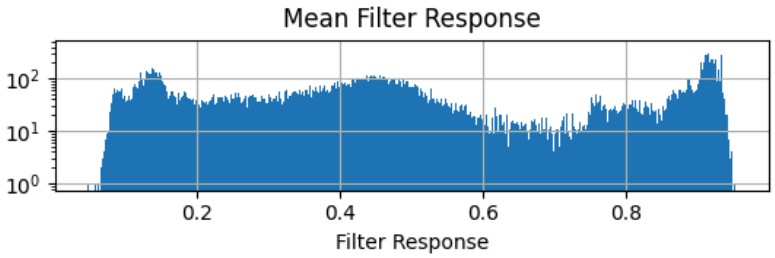
Original: horse-22.jpg



Original: horse-20.jpg

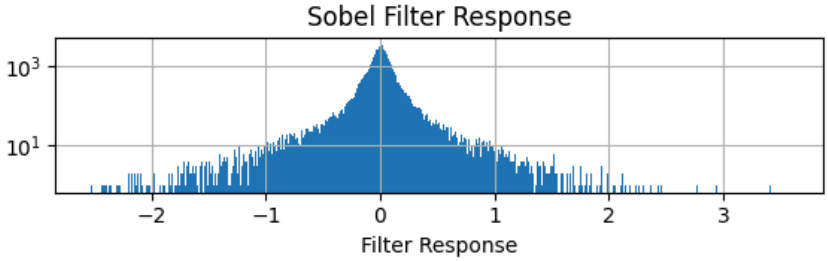


Original: horse-8.jpg

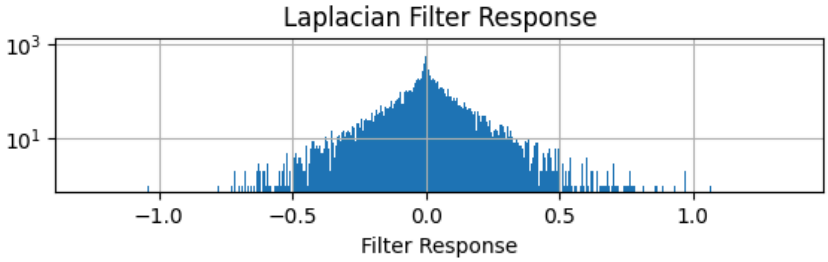


Images of: human

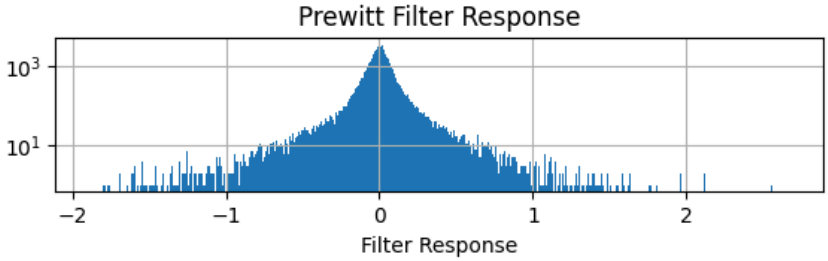
Original: rider-7.jpg



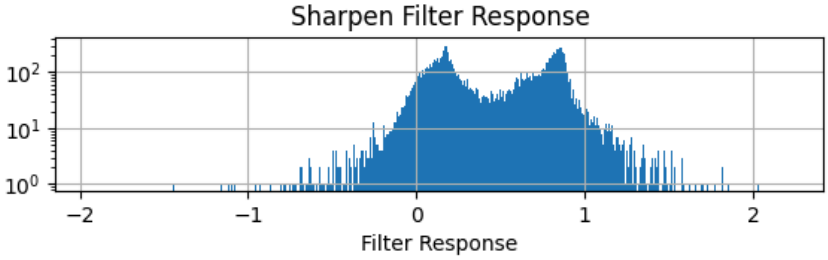
Original: rider-13.jpg



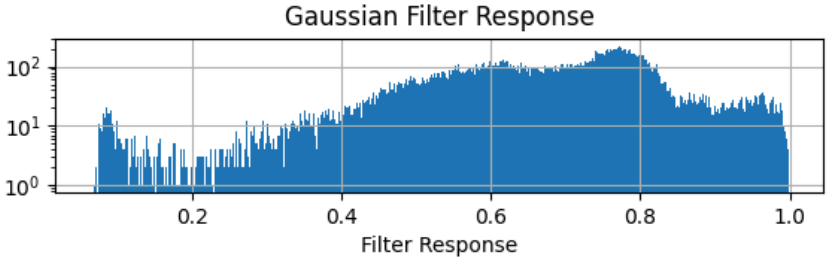
Original: rider-6.jpg



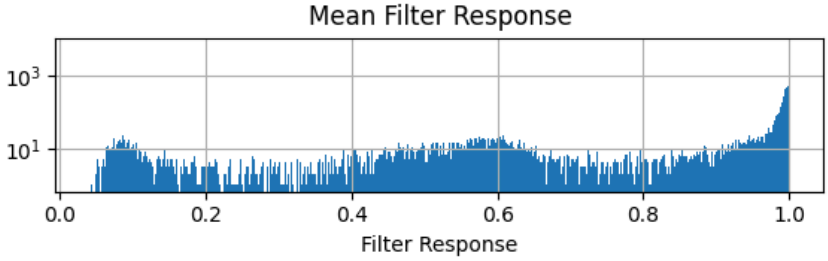
Original: rider-20.jpg



Original: rider-10.jpg

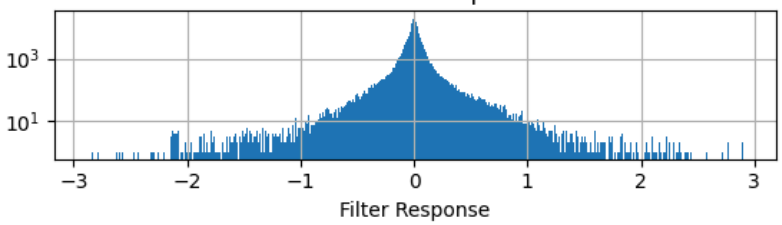


Original: rider-23.jpg

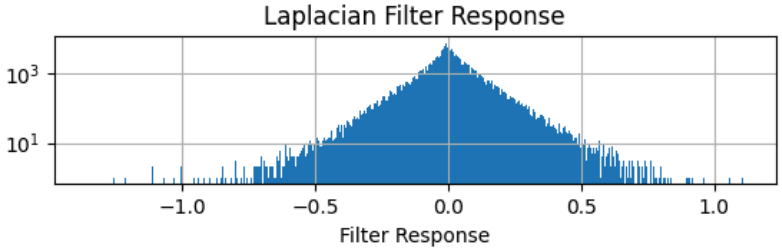


Images of: cars

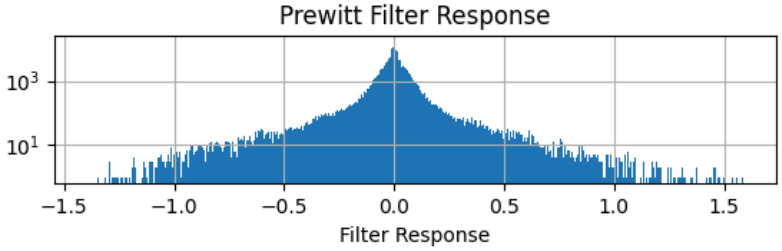
Original: carsgraz_010.bmp



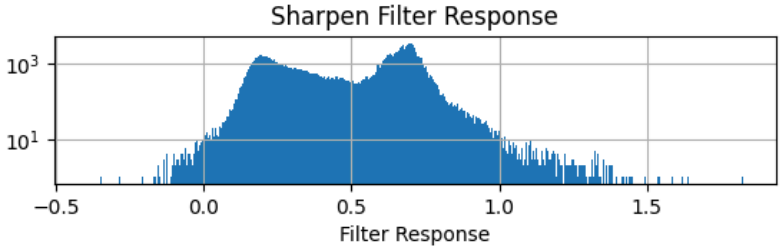
Original: carsgraz_012.bmp



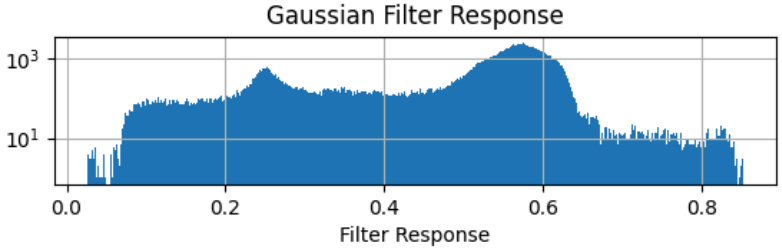
Original: carsgraz_024.bmp



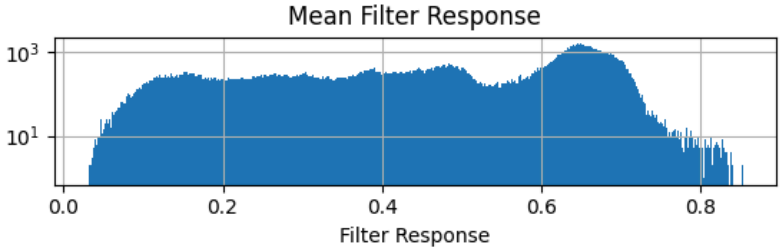
Original: carsgraz_014.bmp



Original: carsgraz_019.bmp

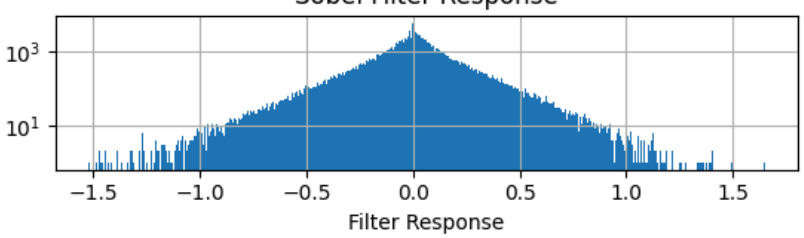


Original: carsgraz_021.bmp

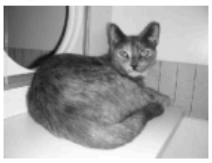


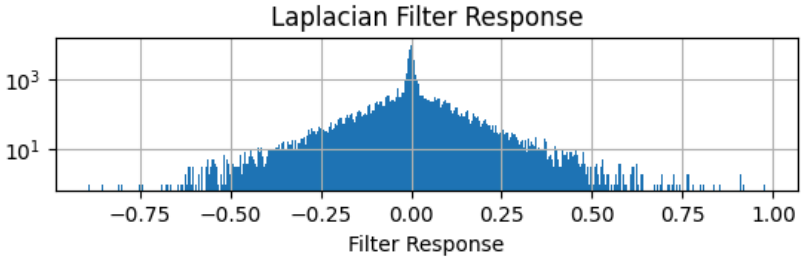
Images of: cats

Original: cat.7.jpg

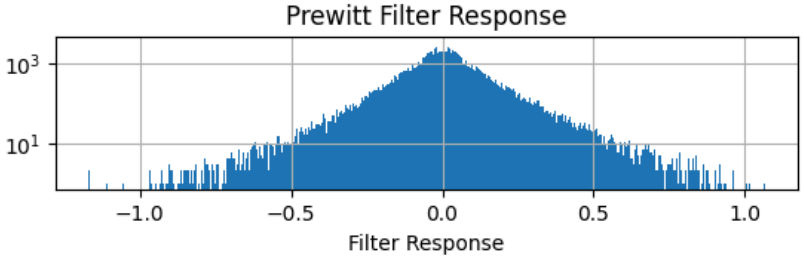


Original: cat.30.jpg

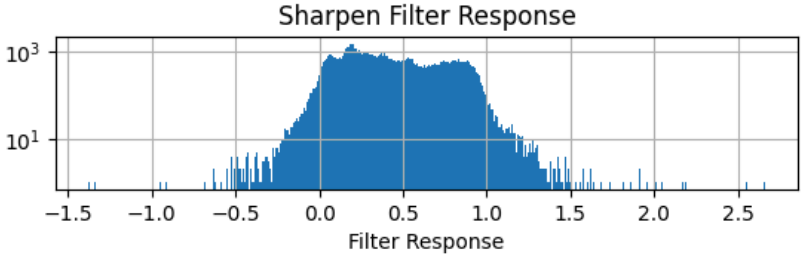




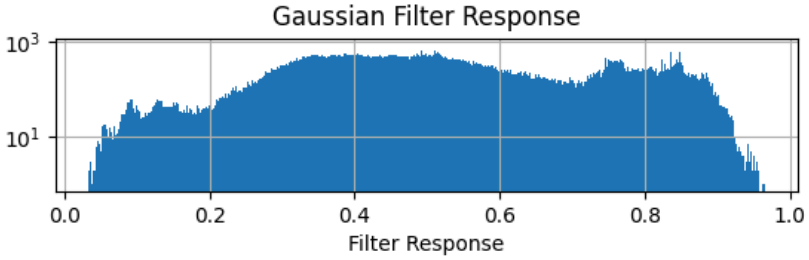
Original: cat.21.jpg



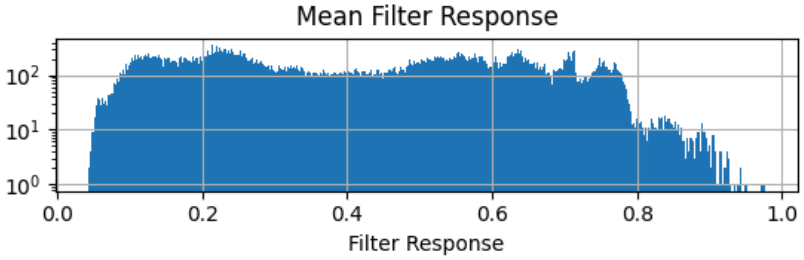
Original: cat.25.jpg



Original: cat.173.jpg

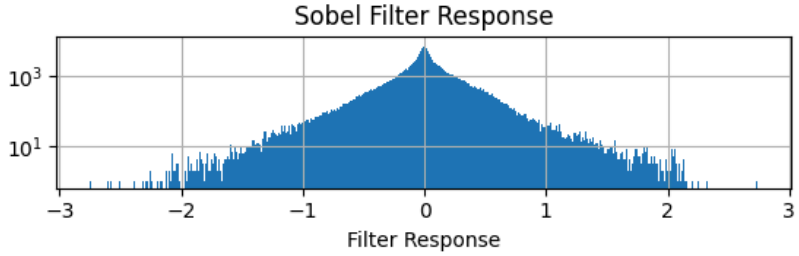


Original: cat.166.jpg

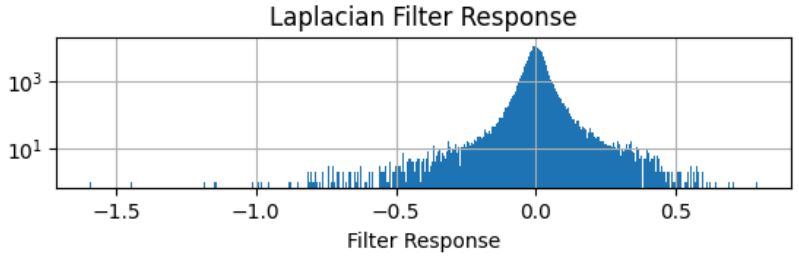


Images of: bike

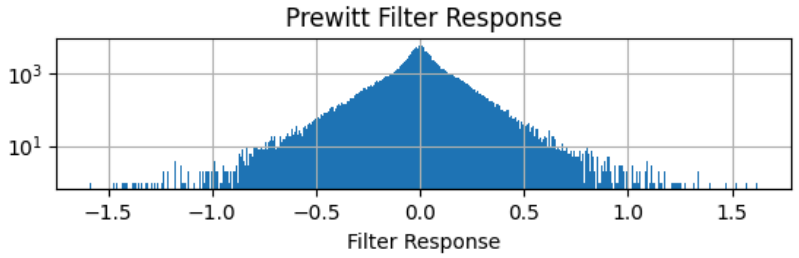
Original: bike_019.bmp



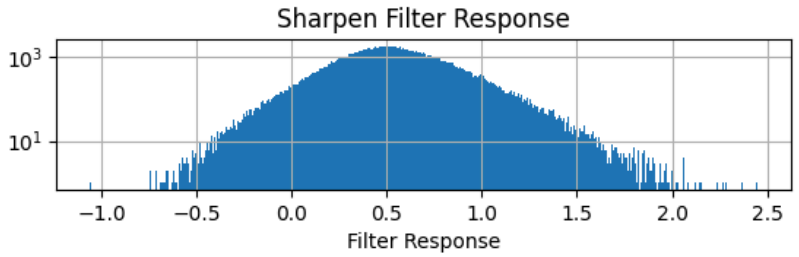
Original: bike_014.bmp



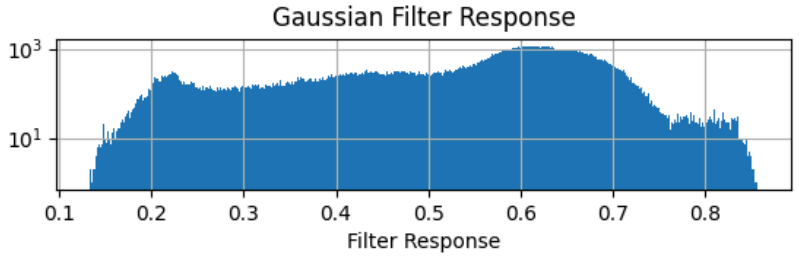
Original: bike_005.bmp



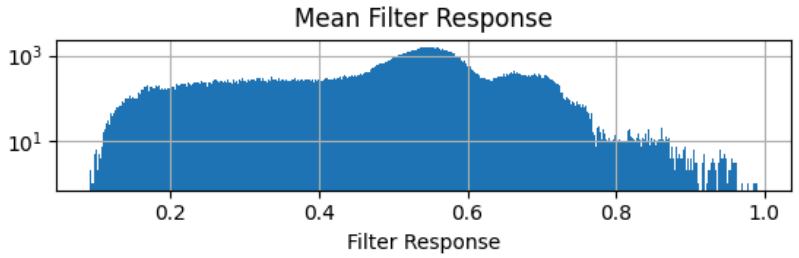
Original: bike_021.bmp



Original: bike_017.bmp

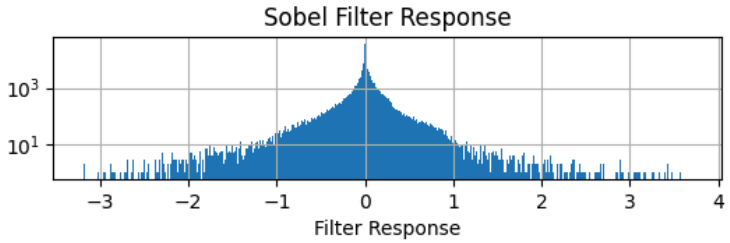


Original: bike_008.bmp

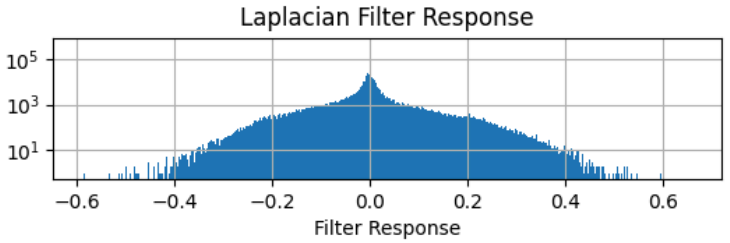


Images of: documents

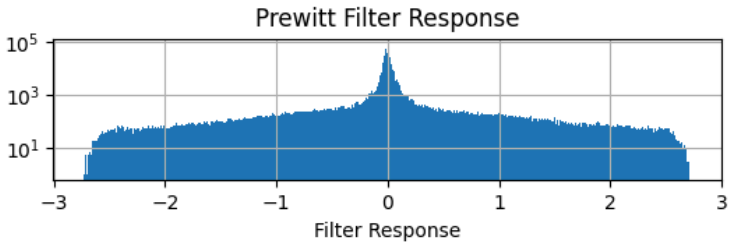
Original: NJ-license.jpg



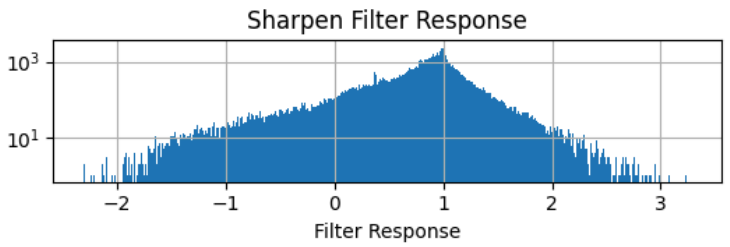
Original: nid.webp



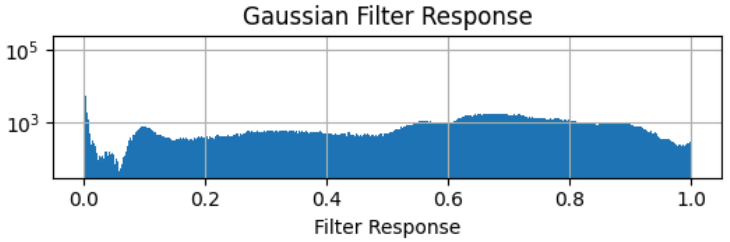
Original: texas.jpeg

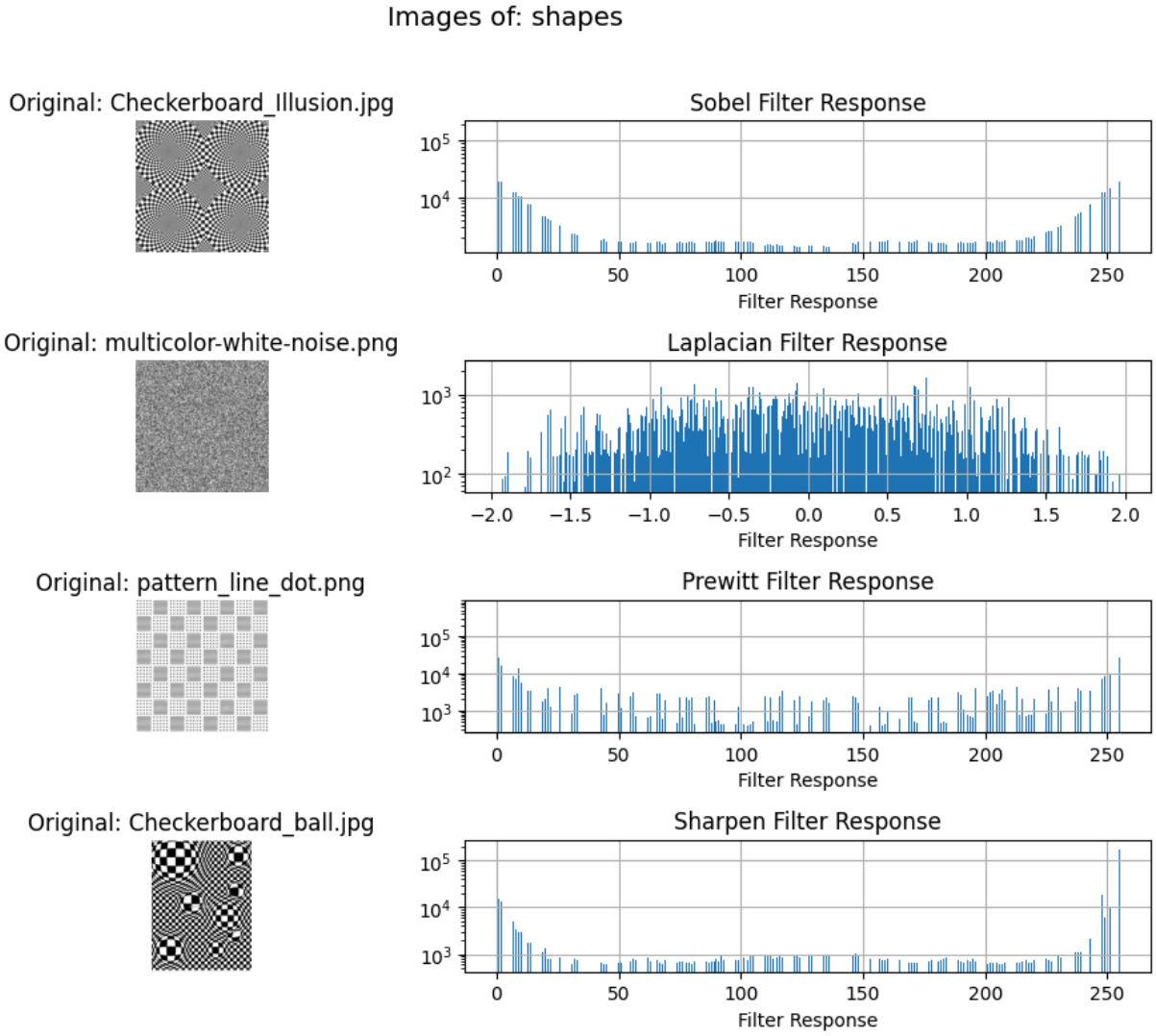


Original: driving-license.webp



Original: Visa_usa.jpg





Filter Types and Reasoning Behind Their Response

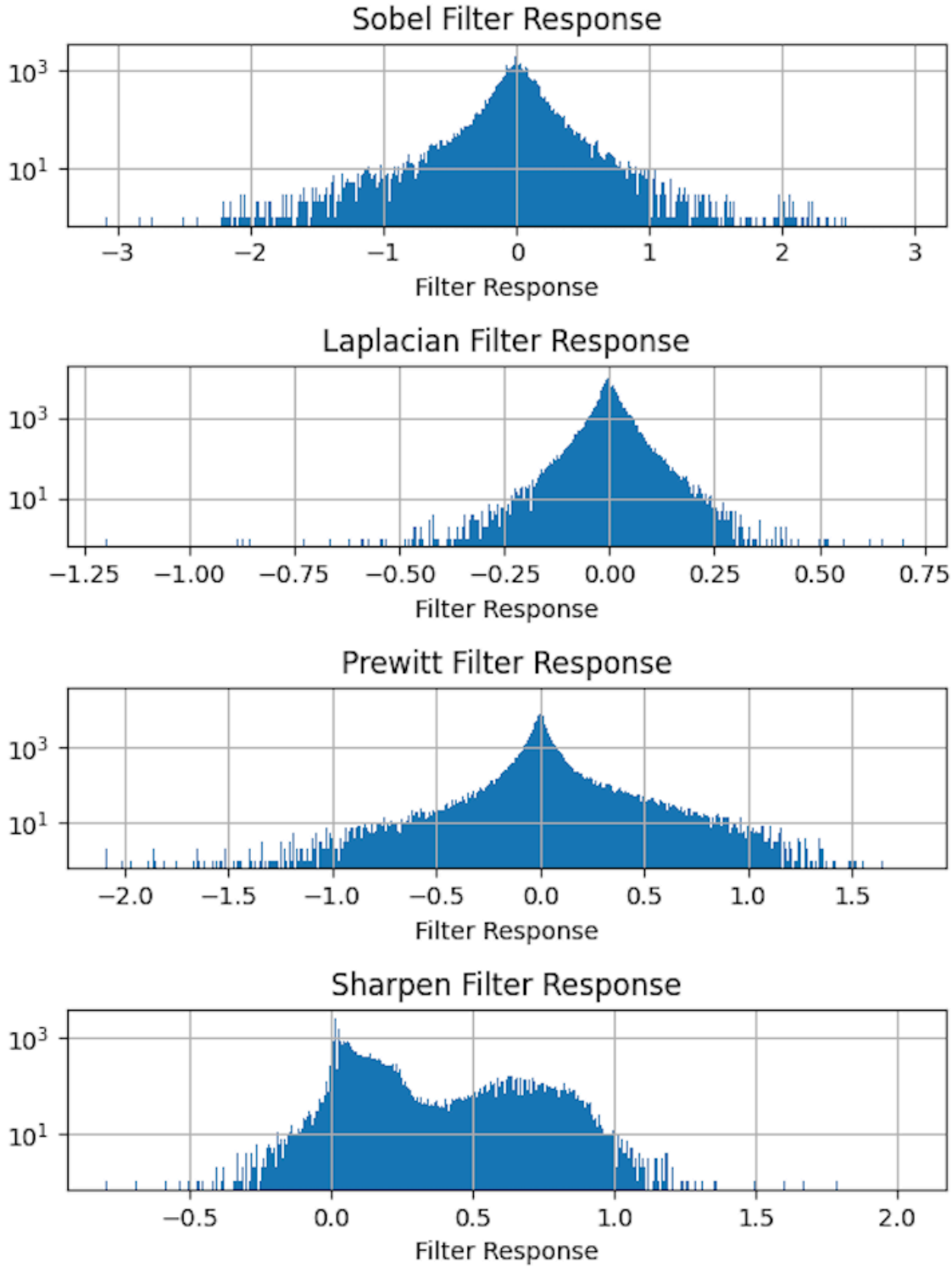
Edge Detection and Enhancement Filters

These filters are designed to detect or enhance edges by focusing on areas where pixel intensity changes rapidly.

- **Sobel and Prewitt filters** detect edges by approximating the gradient of an image. Sobel is slightly more sensitive to strong edges because it assigns higher weights to nearby pixels, while Prewitt uses uniform weights, resulting in smoother responses.
- **Laplacian filter** computes the second derivative of an image, detecting edges in all directions simultaneously. It’s particularly good at finding finer details.

- **Sharpen filter** enhances edges by amplifying high-frequency components. Unlike pure edge detectors, it also enhances fine details in the image.

Why Their Histograms Look This Way



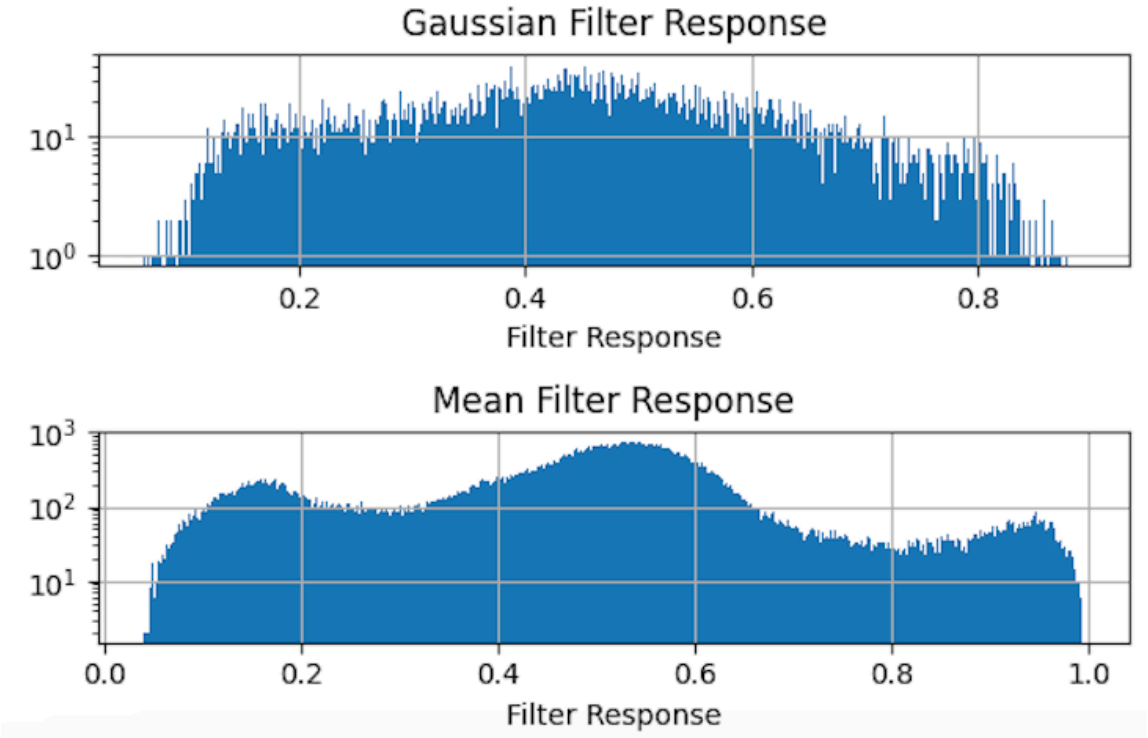
These filters highlight edges and sharp transitions, leading to histograms with a sharp peak at zero (from smooth regions) and heavy tails (from large responses

at edges). The sharpen filter, however, can occasionally produce multiple peaks or shifted peaks due to its effect on fine details and noise.

Smoothing Filters

- **Gaussian filter** applies a weighted average, where nearby pixels contribute more, creating a smooth blur. The degree of smoothing depends on the parameter σ (standard deviation). Mean filter computes a simple average of all pixels in a neighborhood, resulting in uniform blurring.
- **Mean filter** computes a simple average of all pixels in a neighborhood, resulting in uniform blurring.

Why Their Histograms Look Different

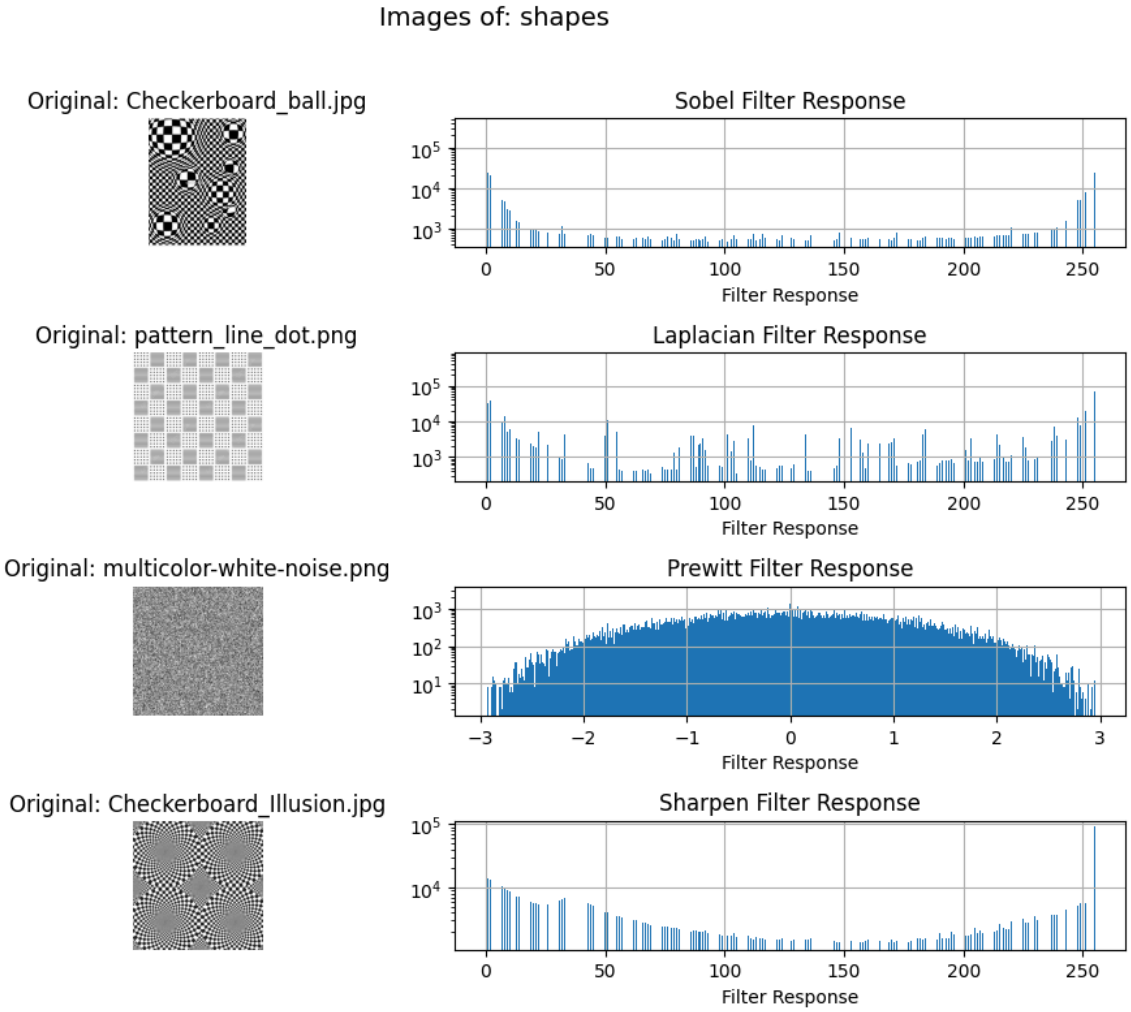


Since these filters blur the image, they reduce sharp transitions and spread values over a range near the average intensity. As a result, their histograms shift to the right of zero, with values more evenly distributed and without distinct peaks.

Images that don't Generate Expected Filter Response

I have found 4 type of images that doesn't generate high peak at zero with heavy tail histogram plot of linear filter responses. The images can be found here. The types are

- Checkerboard variant
- Checkerboard illusion with a pattern
- Noise image
- Pattern of lines & dots



Using Dead Leaves model to replicate the result

The code generates a synthetic image using the dead leaves model by randomly placing circular shapes of different sizes and intensities on a blank canvas to simulate natural image-like textures. It then applies three different filters, Gaussian, Sobel, and Laplacian to the generated image. For each filter, the code plots three subplots: the original image, the filtered image, and the histogram of

filter responses, which typically shows a high peak at zero and heavy tails, replicating the statistical properties of natural images.

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter, sobel, laplace

def generate_dead_leaves_image(image_size, num_shapes, max_shape_size):
    image = np.zeros((image_size, image_size))

    for _ in range(num_shapes):
        radius = np.random.randint(5, max_shape_size)
        intensity = np.random.rand()
        x, y = np.random.randint(0, image_size, size=2)

        # Create a circular mask
        yy, xx = np.ogrid[:image_size, :image_size]
        mask = (xx - x) ** 2 + (yy - y) ** 2 <= radius ** 2

        image[mask] = intensity
    return image

def apply_filter_and_plot(image, filter_type):
    if filter_type == 'gaussian':
        filtered_image = gaussian_filter(image, sigma=1, order=1)
    elif filter_type == 'sobel':
        filtered_image = sobel(image)
    elif filter_type == 'laplacian':
        filtered_image = laplace(image)
    else:
        raise ValueError("Unknown filter type.")


    plt.figure(figsize=(15, 5))

    # Original image
    plt.subplot(1, 3, 1)
    plt.imshow(image, cmap='gray')
    plt.title("Original Image")
    plt.colorbar()

    # Filtered image
    plt.subplot(1, 3, 2)
    plt.imshow(filtered_image, cmap='gray')
    plt.title(f"Filtered Image ({filter_type.capitalize()})")
    plt.colorbar()

    # Histogram of filter responses
    plt.subplot(1, 3, 3)
    plt.hist(filtered_image.ravel(), bins=100, density=True, alpha=0.7)
    plt.title(f"Histogram of Filter Responses ({filter_type.capitalize()})")
    plt.xlabel("Response")
    plt.ylabel("Density")

    plt.tight_layout()
```

```
    plt.show()

image_size = 256
num_shapes = 1000
max_shape_size = 50

dead_leaves_image = generate_dead_leaves_image(image_size, num_shapes, max_s

apply_filter_and_plot(dead_leaves_image, filter_type='sobel')
apply_filter_and_plot(dead_leaves_image, filter_type='laplacian')
apply_filter_and_plot(dead_leaves_image, filter_type='gaussian')
```