1st Step: Change image to data

Image → numpy array → pandas dataframe → csv

In [2]:
```
import os
import numpy as np
import pandas as pd
from PIL import Image
from PIL import ImageEnhance
import matplotlib.pyplot as plt
from scipy.linalg import svd
from PIL import ImageFilter
```

Healthy Potato

The case where the newly added image to the dataset is NAN is thought to be due to the fact that the ratio of the cropped photo is not 1:1, so that pixel point does not exist. In the future, we will stretch all the images to 1:1 format in order to reach harmonization with the previous images.

In [8]:
```
# Set the directory path
dir_path_healthy = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy'

# Create an empty list to store the data
Potato_healthy_data = []

# Loop through all files in the directory
for file in os.listdir(dir_path_healthy):
    # Check if the file is an image file
    if file.endswith(('.JPG', '.jpg')):
        # Load the image file as a numpy array
        img = Image.open(os.path.join(dir_path_healthy, file))
        arr = np.array(img)
        # Reshape the array to 1D
        arr_1d = np.reshape(arr, -1)
        # Append the 1D array to the list
        Potato_healthy_data.append(arr_1d)
```

```
# Convert the list to a DataFrame
Potato_healthy_1 = pd.DataFrame(Potato_healthy_data)


Potato_healthy_1
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	196598	196599	196600	196601	196602	196603	196604	196605	196606	196607
0	161	140	147	134	113	120	132	111	118	151	...	171.0	175.0	162.0	172.0	151.0	138.0	148.0	125.0	112.0	122.0
1	136	116	125	142	122	131	149	129	138	164	...	205.0	207.0	190.0	198.0	177.0	160.0	168.0	192.0	175.0	183.0
2	200	184	194	179	163	173	159	143	153	180	...	130.0	141.0	122.0	128.0	134.0	115.0	121.0	182.0	163.0	169.0
3	133	114	120	111	92	98	115	96	102	111	...	154.0	182.0	169.0	179.0	157.0	144.0	154.0	186.0	173.0	183.0
4	141	129	133	147	135	139	127	115	119	134	...	172.0	185.0	175.0	176.0	187.0	177.0	178.0	180.0	170.0	171.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
231	113	170	57	114	171	58	117	173	62	121	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
232	84	154	56	81	151	53	100	170	72	99	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
233	33	106	17	41	114	25	48	121	31	40	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
234	56	117	48	44	105	36	45	106	37	37	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
235	12	6	0	27	23	11	41	37	25	33	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

236 rows × 196608 columns

Early Blight Potato

In [9]:
```
# Set the directory path
dir_path_healthy = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight'

# Create an empty list to store the data
Potato_Early_blight_data = []

# Loop through all files in the directory
for file in os.listdir(dir_path_healthy):
```

```
    # Check if the file is an image file
    if file.endswith(('.JPG')):
        # Load the image file as a numpy array
        img = Image.open(os.path.join(dir_path_healthy, file))
        arr = np.array(img)
        # Reshape the array to 1D
        arr_1d = np.reshape(arr, -1)
        # Append the 1D array to the list
        Potato_Early_blight_data.append(arr_1d)

# Convert the list to a DataFrame
Potato_Early_blight_1 = pd.DataFrame(Potato_Early_blight_data)


Potato_Early_blight_1
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	196598	196599	196600	196601	196602	196603	196604	196605	196606	196607
0	123	120	127	161	158	165	148	145	152	141	...	173	169	167	180	172	170	183	173	171	184
1	169	162	169	170	163	171	162	154	165	161	...	187	175	171	185	178	174	188	180	176	190
2	199	196	203	194	191	198	187	184	191	184	...	136	127	122	129	132	127	134	143	138	145
3	94	97	114	123	126	143	129	132	149	131	...	202	173	181	200	171	179	198	169	177	196
4	157	156	172	163	162	178	165	164	180	161	...	161	142	140	153	137	135	148	130	128	141
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	131	129	143	111	109	123	66	64	78	129	...	187	176	175	189	174	173	187	172	171	185
996	194	184	193	187	177	186	183	173	181	186	...	138	132	126	130	130	124	128	134	128	132
997	121	119	132	126	124	137	110	108	121	120	...	204	193	191	204	195	193	206	195	193	206
998	185	183	194	181	179	190	178	176	187	179	...	159	137	135	149	136	134	148	152	150	164
999	120	117	128	153	150	161	210	207	218	150	...	210	197	196	210	189	188	202	190	189	203

1000 rows × 196608 columns

Late Blight Potato

In [11]:
```
# Set the directory path
dir_path_healthy = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'

# Create an empty list to store the data
Potato_Late_blight_data = []

# Loop through all files in the directory
for file in os.listdir(dir_path_healthy):
    # Check if the file is an image file
    if file.endswith(('.JPG')):
        # Load the image file as a numpy array
        img = Image.open(os.path.join(dir_path_healthy, file))
        arr = np.array(img)
        # Reshape the array to 1D
        arr_1d = np.reshape(arr, -1)
        # Append the 1D array to the list
        Potato_Late_blight_data.append(arr_1d)

# Convert the list to a DataFrame
Potato_Late_blight_1 = pd.DataFrame(Potato_Late_blight_data)


Potato_Late_blight_1
```

Out[11]:

	0	1	2	3	4	5	6	7	8	9	...	196598	196599	196600	196601	196602	196603	196604	196605	196606	196607
0	145	134	138	146	135	139	147	136	140	147	...	116	120	114	118	116	110	114	107	101	105
1	163	151	161	145	133	143	130	118	128	140	...	138	128	120	133	133	125	138	142	134	147
2	160	155	162	162	157	164	166	161	168	168	...	131	129	124	130	131	126	132	135	130	136
3	107	94	104	101	88	98	123	110	120	105	...	160	133	128	148	137	132	152	151	146	166
4	196	180	181	163	147	148	193	177	178	180	...	138	121	91	93	101	71	73	118	88	90
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	140	109	107	136	105	103	151	120	118	138	...	140	155	136	138	162	143	145	168	149	151
996	165	158	165	168	161	168	171	164	171	172	...	122	124	111	121	123	110	120	124	111	121
997	136	124	128	134	122	126	132	120	124	131	...	144	147	136	142	161	150	156	135	124	130
998	144	133	137	133	122	126	133	122	126	139	...	182	177	175	180	179	177	182	181	179	184
999	108	95	104	119	106	115	82	69	78	98	...	162	148	144	161	145	141	158	143	139	156

1000 rows × 196608 columns

Data Normalization

This step rescales the pixel values of an image to a specific range: [0, 255] to ensure that the image data has a consistent range. Normalizing an image can provide the following benefits:\ 1.Avoiding issues like gradient explosion or vanishing gradients during model training, making the model converge more easily.\ 2.Ensuring that different images have similar data ranges, aiding the model in generalizing to various images.\ 3.Reducing the impact of noise or outliers in the input data on the CNN model.

In [7]:
```
def normalize_image(img_path):
    normalized_data = []
    for file in os.listdir(img_path):
        if file.endswith(('.JPG', '.jpg')):
            img = Image.open(os.path.join(img_path, file))
            arr = np.array(img)
            arr_1d = arr.flatten()
            normalized_arr = arr_1d / 255.0
```

```
            normalized_data.append(normalized_arr)
    normalized_df = pd.DataFrame(normalized_data)
    return normalized_df

def save_df_to_csv(normalized_df, csv_filename):
    # Save the DataFrame to a CSV file
    normalized_df.to_csv(csv_filename, index=False)
    print(f"Data saved to {csv_filename}")

# Paths to the directories containing the image datasets
paths = [
    'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy',
    'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight',
    'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'
]

# Filenames for the CSV files to save the normalized data
csv_filenames = [
    'Potato_Healthy_1_normalized.csv',
    'Potato_Early_blight_1_normalized.csv',
    'Potato_Late_blight_1_normalized.csv'
]

# Process each dataset, normalize the images, and save the results to CSV
for path, csv_filename in zip(paths, csv_filenames):
    normalized_df = normalize_image(path)
    save_df_to_csv(normalized_df, csv_filename)
```

```
Data saved to Potato_Healthy_1_normalized.csv
Data saved to Potato_Early_blight_1_normalized.csv
Data saved to Potato_Late_blight_1_normalized.csv
```

2nd Step:Data Augmentation And Image Enhancement

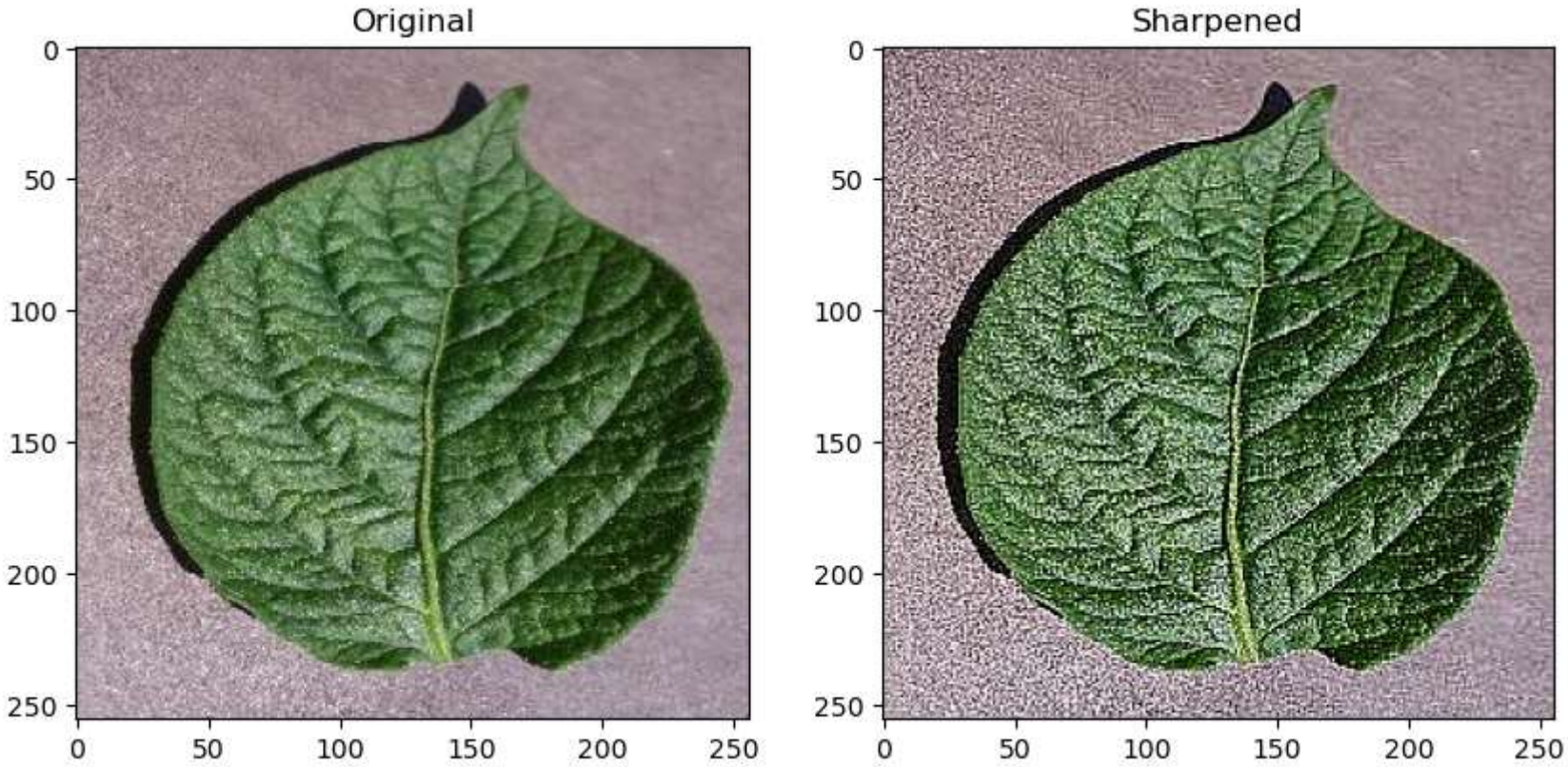
Data augmentation is a powerful method for increasing the accuracy of modern image classifiers. We applied several augmentation techniques to the raw dataset to improve our model performance and recognition capability.\ After each image augmentation is performed, the new images obtained are changed to data and the normalization of the data is also performed.

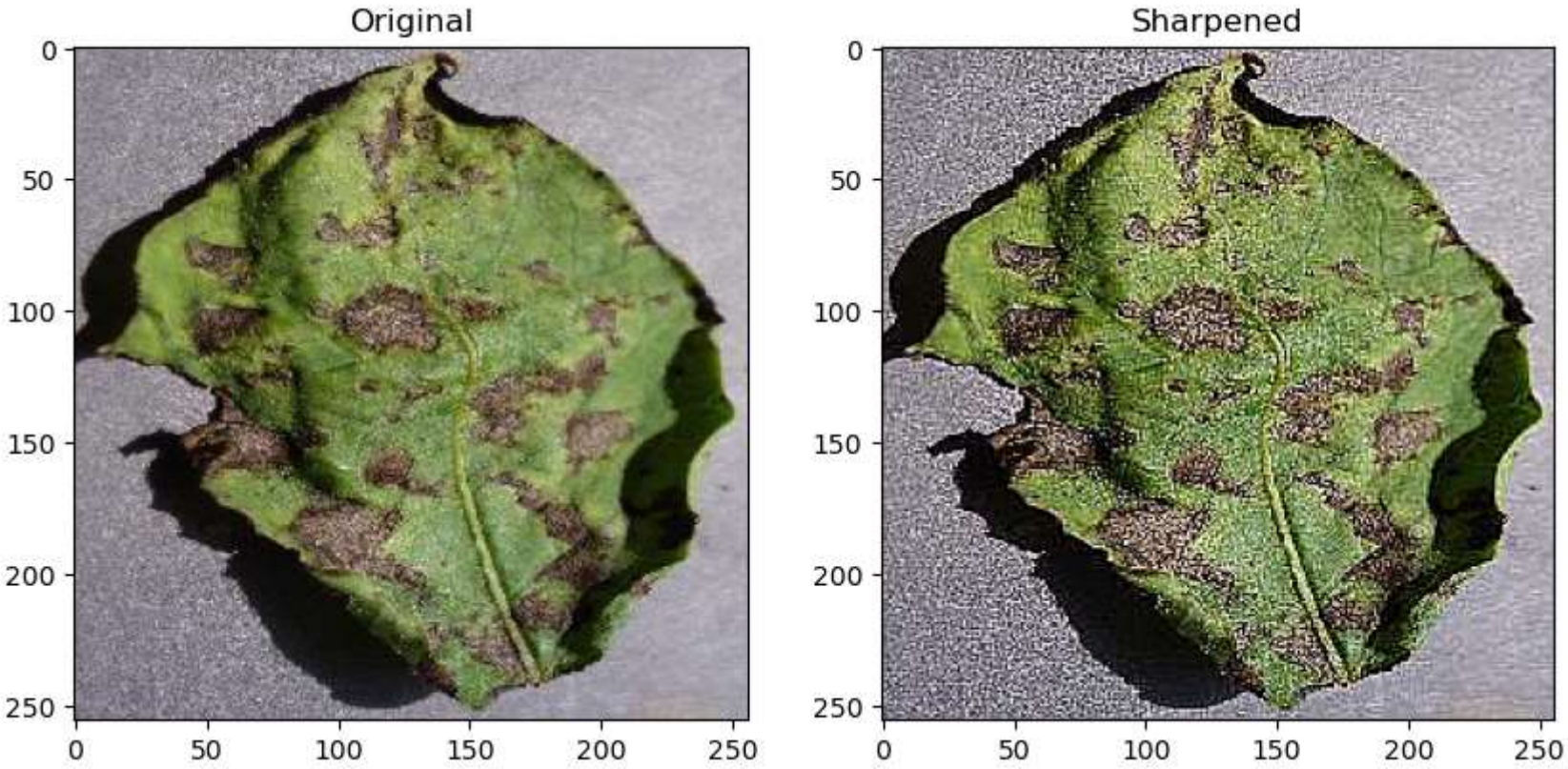
1.Image Sharpen And Data normalization

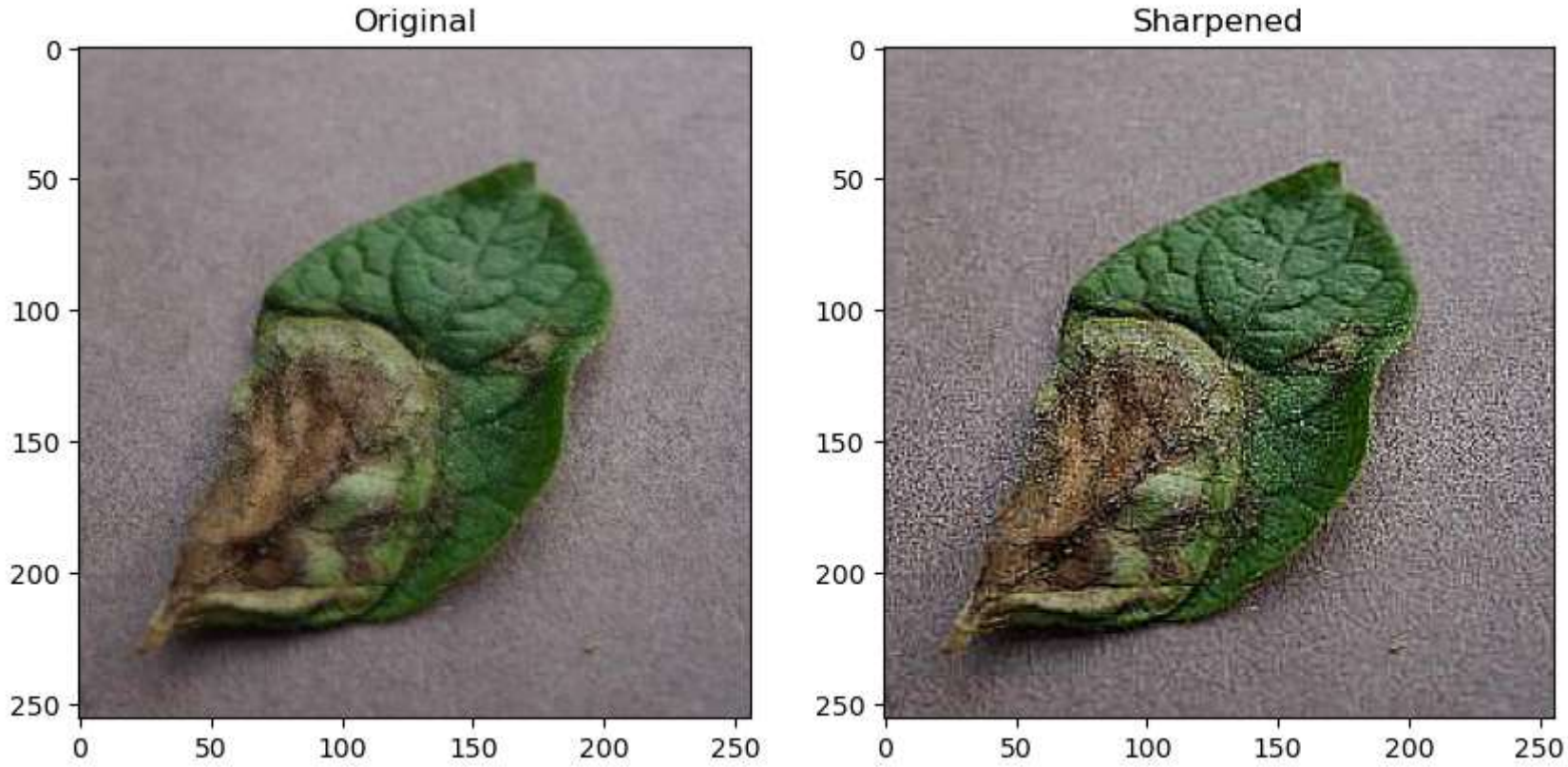
Sharpening enhances the edges of an image, making the details of the image more visible. This is very helpful for models to recognize subtle features in an image.

In [3]:
```
# Set the directory paths
dir_paths = {
    'healthy': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy',
    'early_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight',
    'late_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'
}

# Function to sharpen an image and return it without saving
def sharpen_image(image):
    enhancer = ImageEnhance.Sharpness(image)
    sharpened_img = enhancer.enhance(4.0)  # Adjust the sharpness level as needed
    return sharpened_img

# Function to display image comparisons
def plot_image_comparison(original_image, sharpened_image):
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(original_image)
    axs[0].set_title('Original')
    axs[1].imshow(sharpened_image)
    axs[1].set_title('Sharpened')
    plt.show()

# Function to normalize and flatten image pixels
def normalize_and_flatten_image(image):
    img_array = np.asarray(image) / 255.0  # Normalize pixel values to 0-1 scale
    return img_array.flatten()  # Flatten the array

# Process images, display comparison, normalize, and save to CSV
def process_images(dir_path, category):
    normalized_pixels_list = []
    for i, file in enumerate(os.listdir(dir_path)):
        if file.endswith(('.JPG', '.jpg')):
            image_path = os.path.join(dir_path, file)
            img = Image.open(image_path)
            sharpened_img = sharpen_image(img)
```

```
            if i == 0:  # Only display comparison for the first image
                plot_image_comparison(img, sharpened_img)
            flattened_data = normalize_and_flatten_image(sharpened_img)
            normalized_pixels_list.append(flattened_data)
    # Create DataFrame and save to CSV
    df = pd.DataFrame(normalized_pixels_list)
    csv_file_path = f'{category}_normalized_pixel_values.csv'
    df.to_csv(csv_file_path, index=False)

# Process each category of images
for category, dir_path in dir_paths.items():
    process_images(dir_path, category)
```




Original
Sharpened
0
50
100
150
200
250

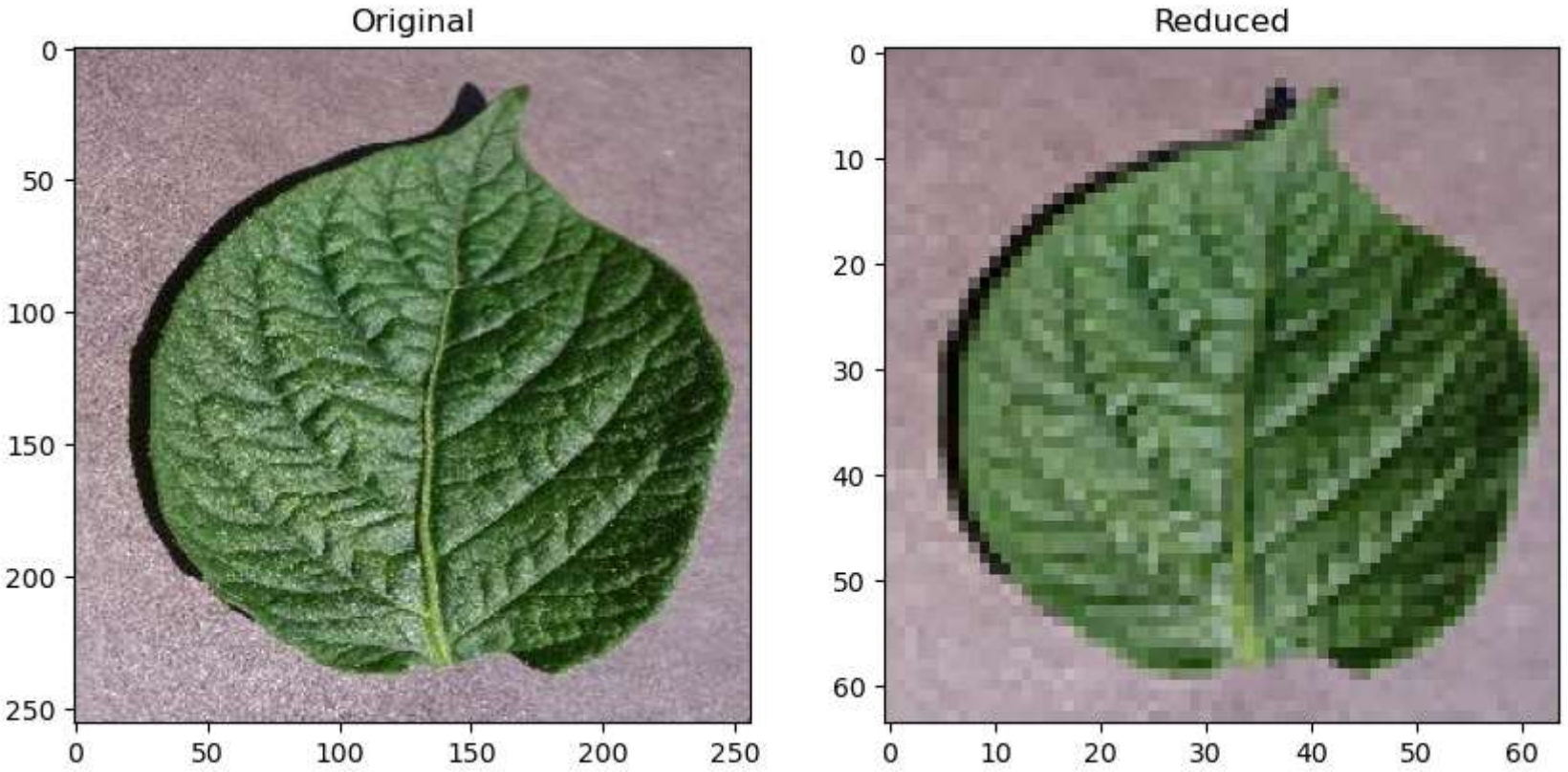


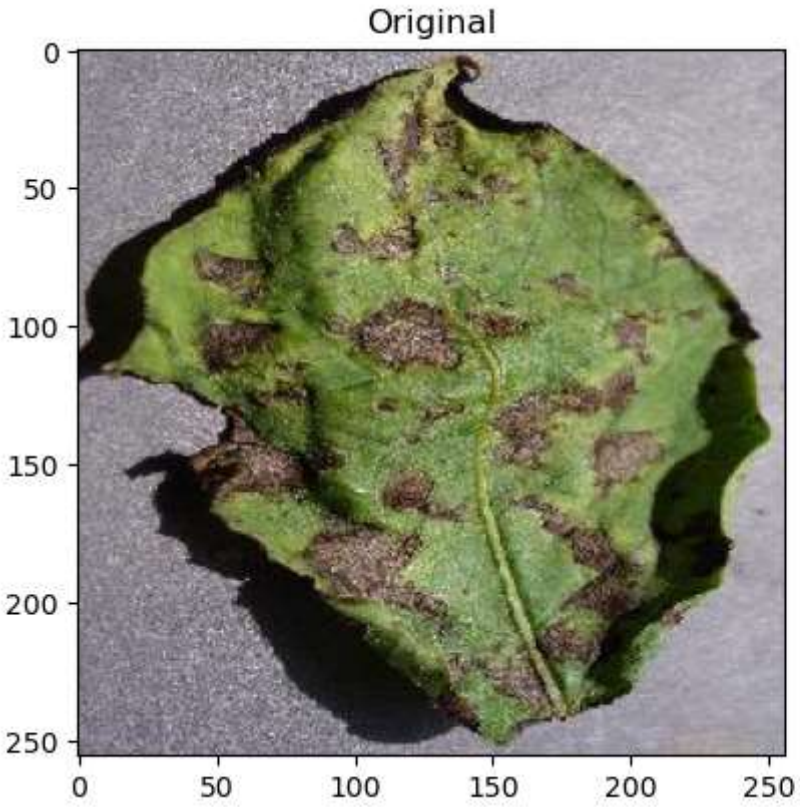
2.Image Compressed（Reduce Pixles）And Data normalization

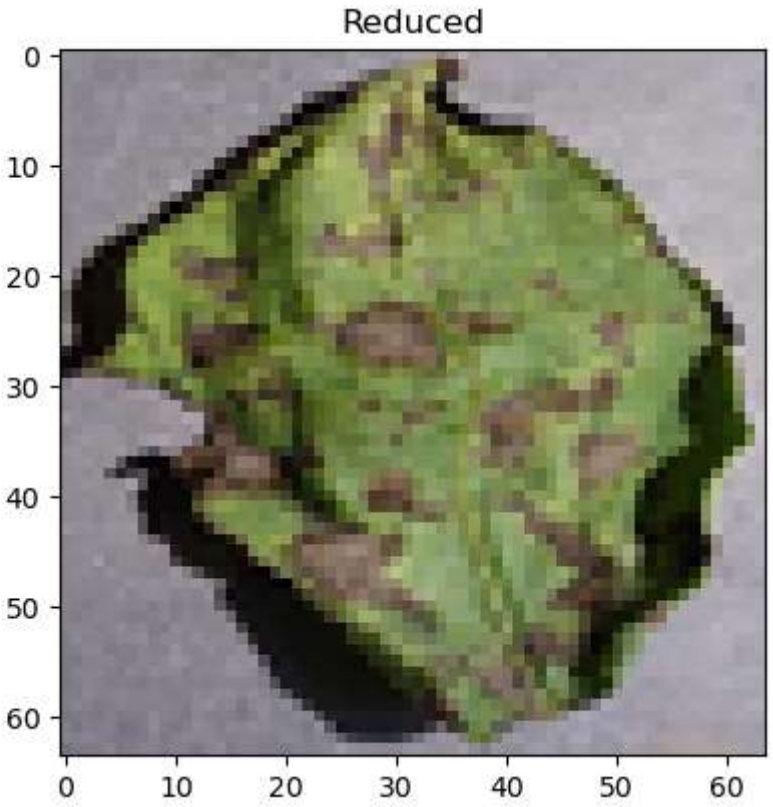
Image compression aims to reduce file size by decreasing the number of pixels to optimize storage space utilization, speed up data transfer, and improve network loading efficiency, but needs to be balanced against image quality loss.

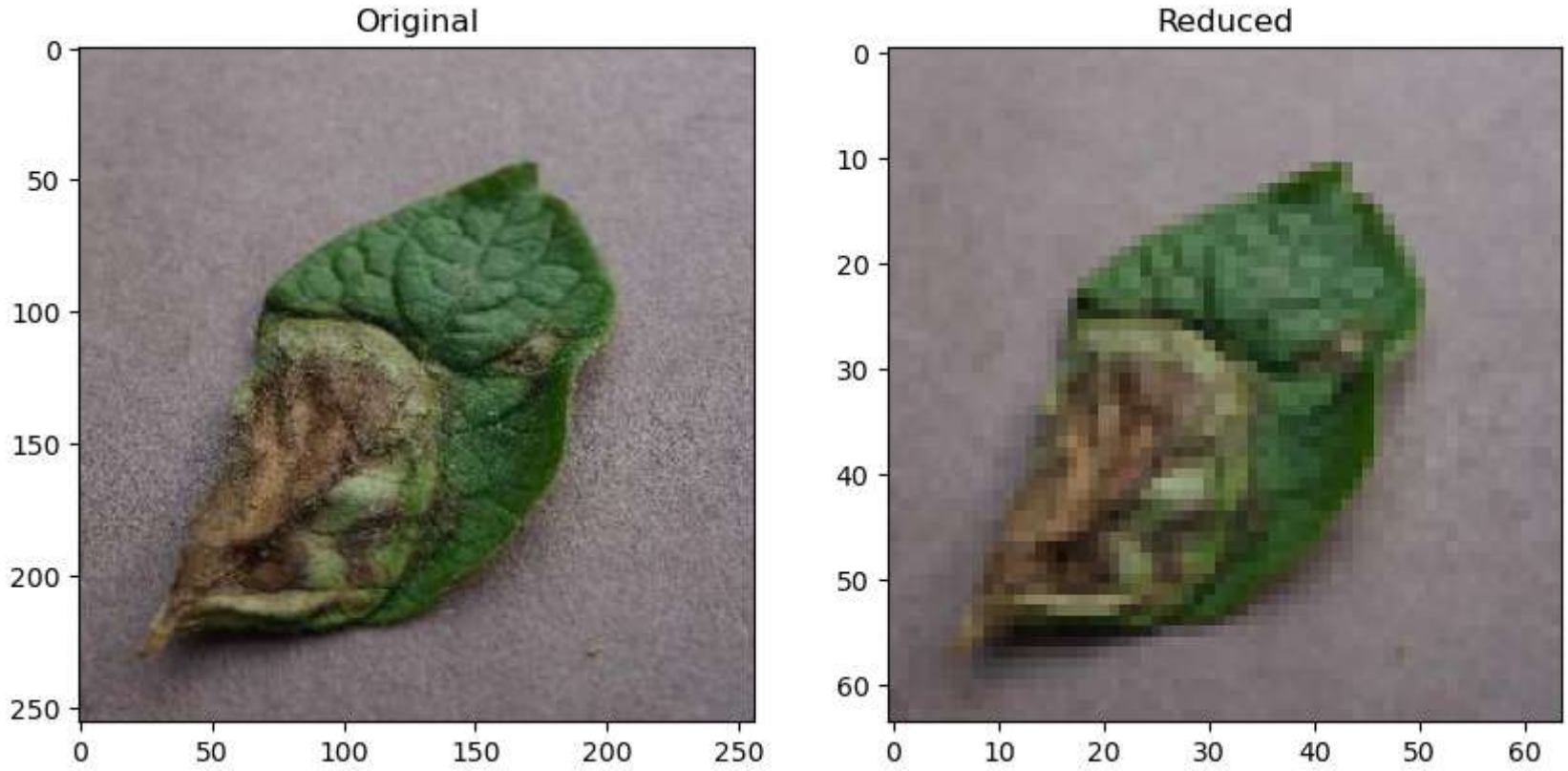
In [4]:
```
# Set the directory paths
dir_paths = {
    'healthy': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy',
    'early_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight',
    'late_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'
}

# Function to compress an image by reducing pixels and return it
def reduce_pixel_image(image, reduction_ratio=0.25):
```

```
    new_width = int(image.width * reduction_ratio)
    new_height = int(image.height * reduction_ratio)
    reduced_img = image.resize((new_width, new_height), Image.LANCZOS)
    return reduced_img

# Function to display image comparisons
def plot_image_comparison(original_image, reduced_image):
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(original_image)
    axs[0].set_title('Original')
    axs[1].imshow(reduced_image)
    axs[1].set_title('Reduced')
    plt.show()

# Function to normalize and flatten image pixels
def normalize_and_flatten_image(image):
    img_array = np.asarray(image) / 255.0
    return img_array.flatten()

# Process images, display comparison, normalize, and save to CSV
def process_images(dir_path, category):
    normalized_pixels_list = []
    for i, file in enumerate(os.listdir(dir_path)):
        if file.endswith(('.JPG', '.jpg')):
            image_path = os.path.join(dir_path, file)
            img = Image.open(image_path)
            reduced_img = reduce_pixel_image(img)
            if i == 0:  # Only display comparison for the first image
                plot_image_comparison(img, reduced_img)
            flattened_data = normalize_and_flatten_image(reduced_img)
            normalized_pixels_list.append(flattened_data)
    # Create DataFrame and save to CSV
    df = pd.DataFrame(normalized_pixels_list)
    csv_file_path = f'{category}_normalized_compressed_values.csv'
    df.to_csv(csv_file_path, index=False)

# Process each category of images
for category, dir_path in dir_paths.items():
    process_images(dir_path, category)
```


Original
0
50
100
150
200
250
0
50
100
150
200
250
Reduced
0
10
20
30
40
50
60
0
10
20
30
40
50
60


Original
0
50
100
150
200
250
0
50
100
150
200
250


Reduced
0
10
20
30
40
50
60
0
10
20
30
40
50
60



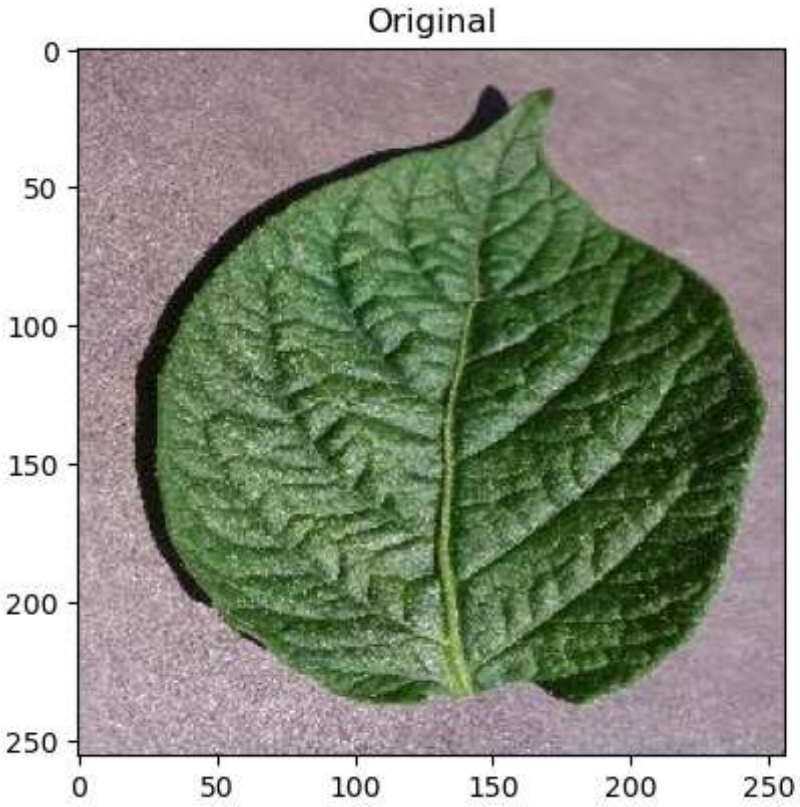
3.Image Sharpen+Image Compressed And Data normalization

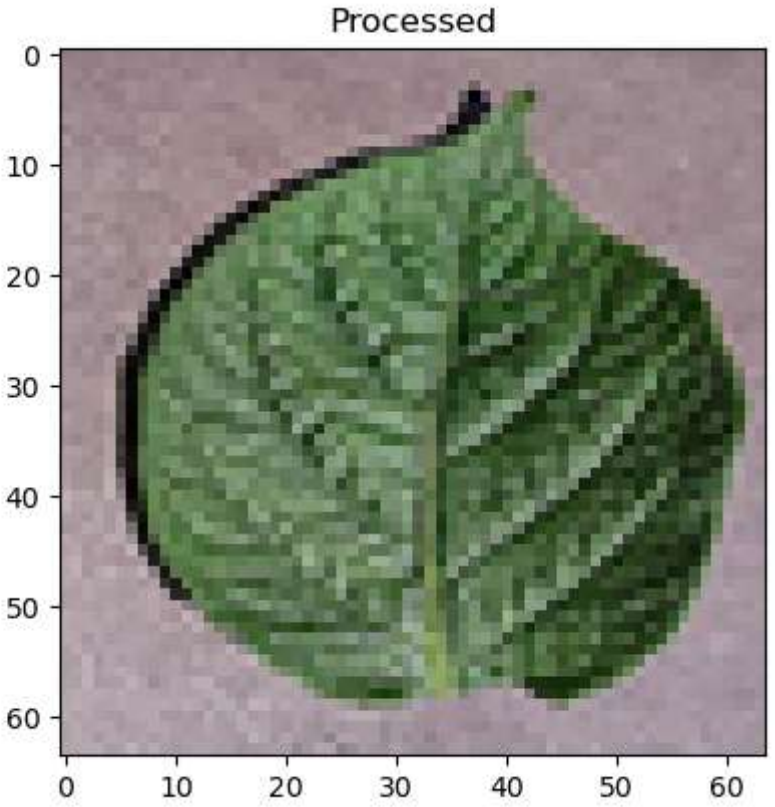
Image sharpening improves the visual quality of an image, making important features more prominent, while image compression ensures that file sizes are optimized, which is especially important for processing large-scale image datasets. Increased processing efficiency means that more images can be processed in the same amount of time or the same amount of work can be done with fewer resources.

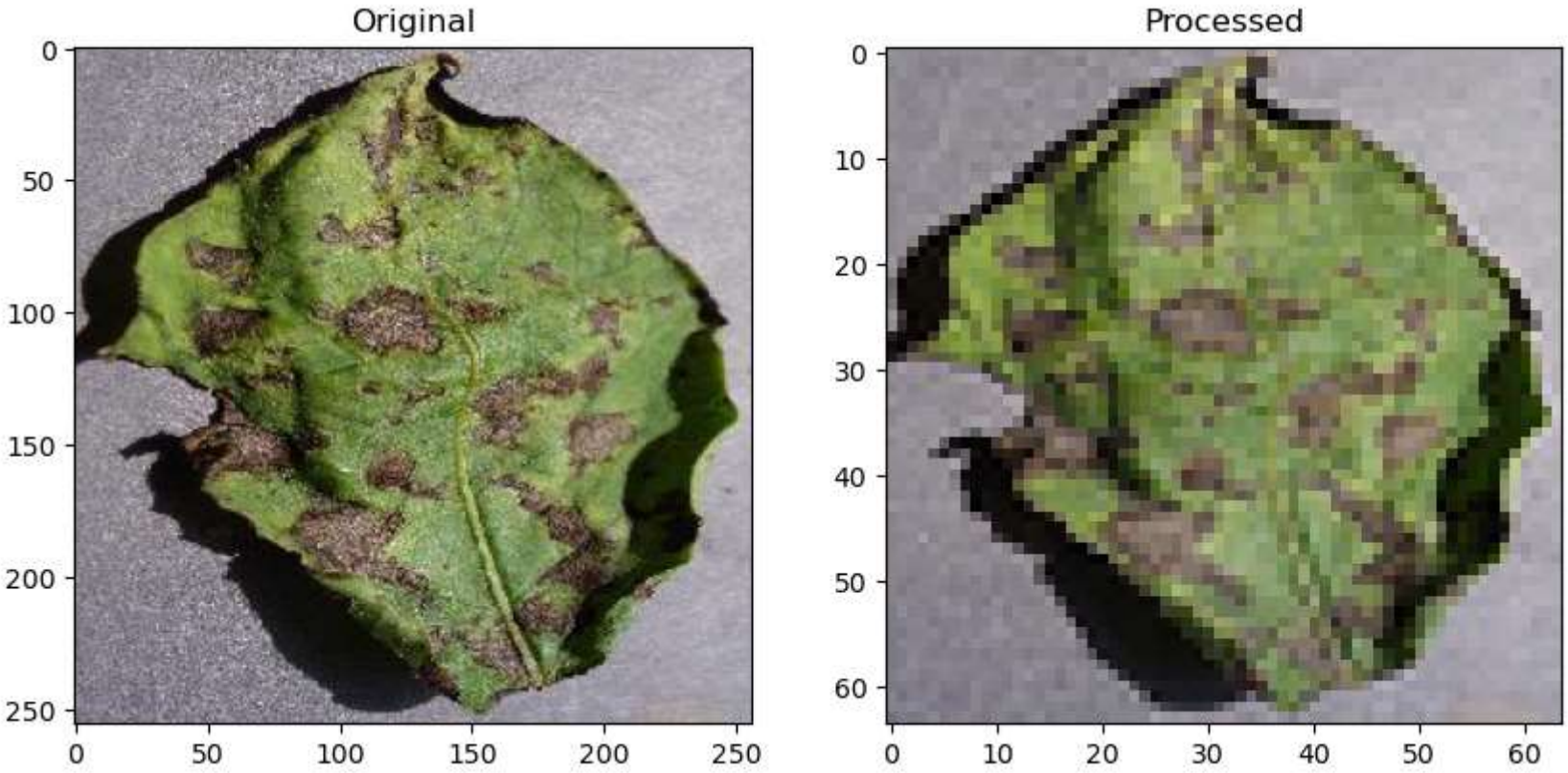
In [5]:
```
# Set the directory paths
dir_paths = {
    'healthy': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy',
    'early_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight',
    'late_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'
}

# Function to sharpen and then reduce pixels of an image, and return it
```

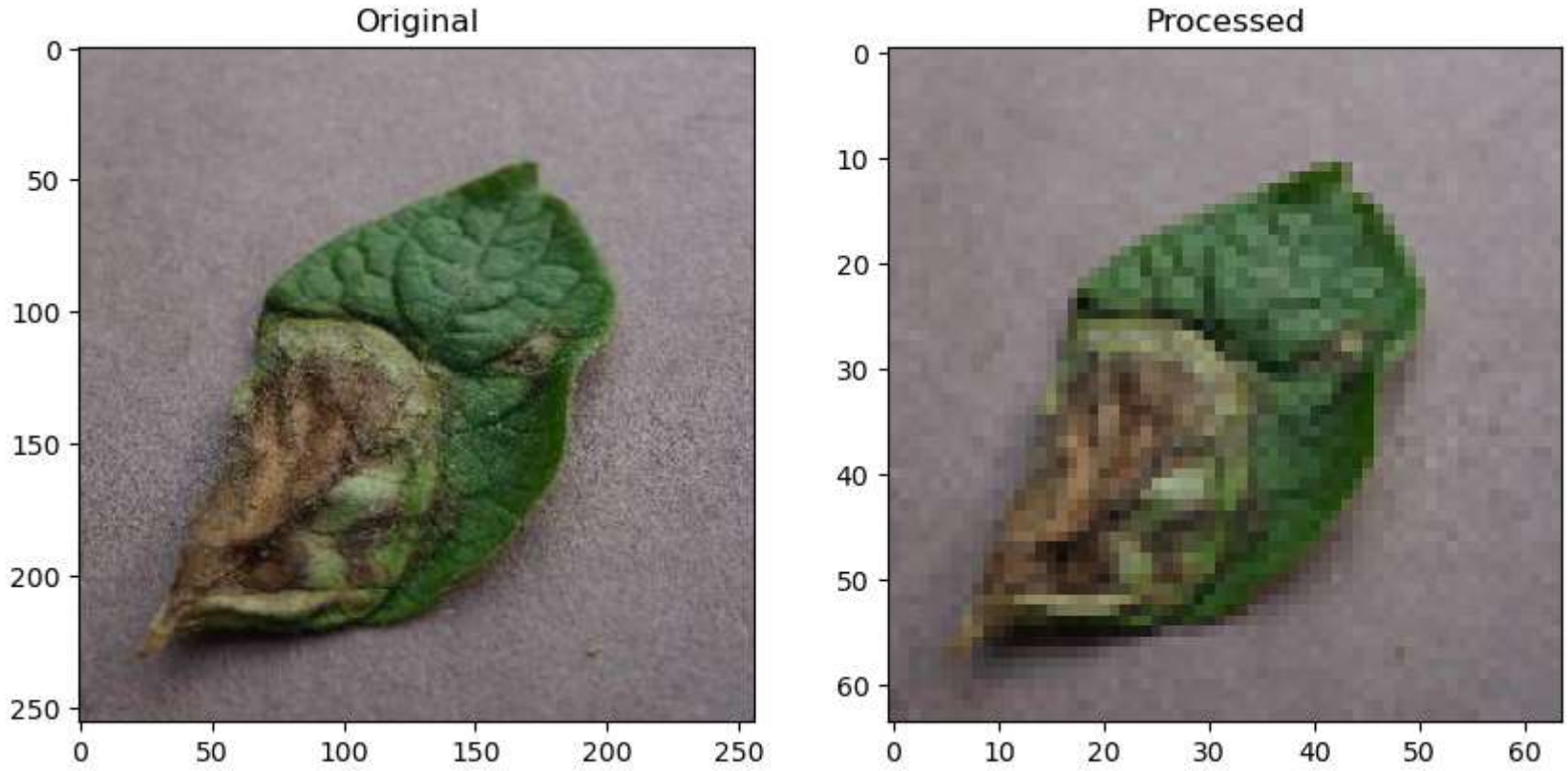
```
def sharpen_and_reduce_pixel_image(image, sharpen_ratio=4.0, reduction_ratio=0.25):
    # Sharpen the image
    enhancer = ImageEnhance.Sharpness(image)
    sharpened_img = enhancer.enhance(sharpen_ratio)

    # Reduce image size by specified reduction ratio
    new_width = int(sharpened_img.width * reduction_ratio)
    new_height = int(sharpened_img.height * reduction_ratio)
    reduced_img = sharpened_img.resize((new_width, new_height), Image.LANCZOS)

    return reduced_img

# Function to display image comparisons
def plot_image_comparison(original_image, processed_image):
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(original_image)
    axs[0].set_title('Original')
    axs[1].imshow(processed_image)
    axs[1].set_title('Processed')
    plt.show()

# Function to normalize and flatten image pixels
def normalize_and_flatten_image(image):
    img_array = np.asarray(image) / 255.0
    return img_array.flatten()

# Process images, display comparison, normalize, and save to CSV
def process_images(dir_path, category):
    normalized_pixels_list = []
    for i, file in enumerate(os.listdir(dir_path)):
        if file.endswith(('.JPG', '.jpg')):
            image_path = os.path.join(dir_path, file)
            img = Image.open(image_path)
            processed_img = sharpen_and_reduce_pixel_image(img)
            if i == 0:  # Display comparison for the first image
                plot_image_comparison(img, processed_img)
            flattened_data = normalize_and_flatten_image(processed_img)
            normalized_pixels_list.append(flattened_data)
    # Create DataFrame and save to CSV
    df = pd.DataFrame(normalized_pixels_list)
    csv_file_path = f'{category}_normalized_sharpen_compressed_values.csv'
    df.to_csv(csv_file_path, index=False)

# Process each category of images
```

```
for category, dir_path in dir_paths.items():
    process_images(dir_path, category)
```






Original
Processed
0
50
100
150
200
250
10
20
30
40
60



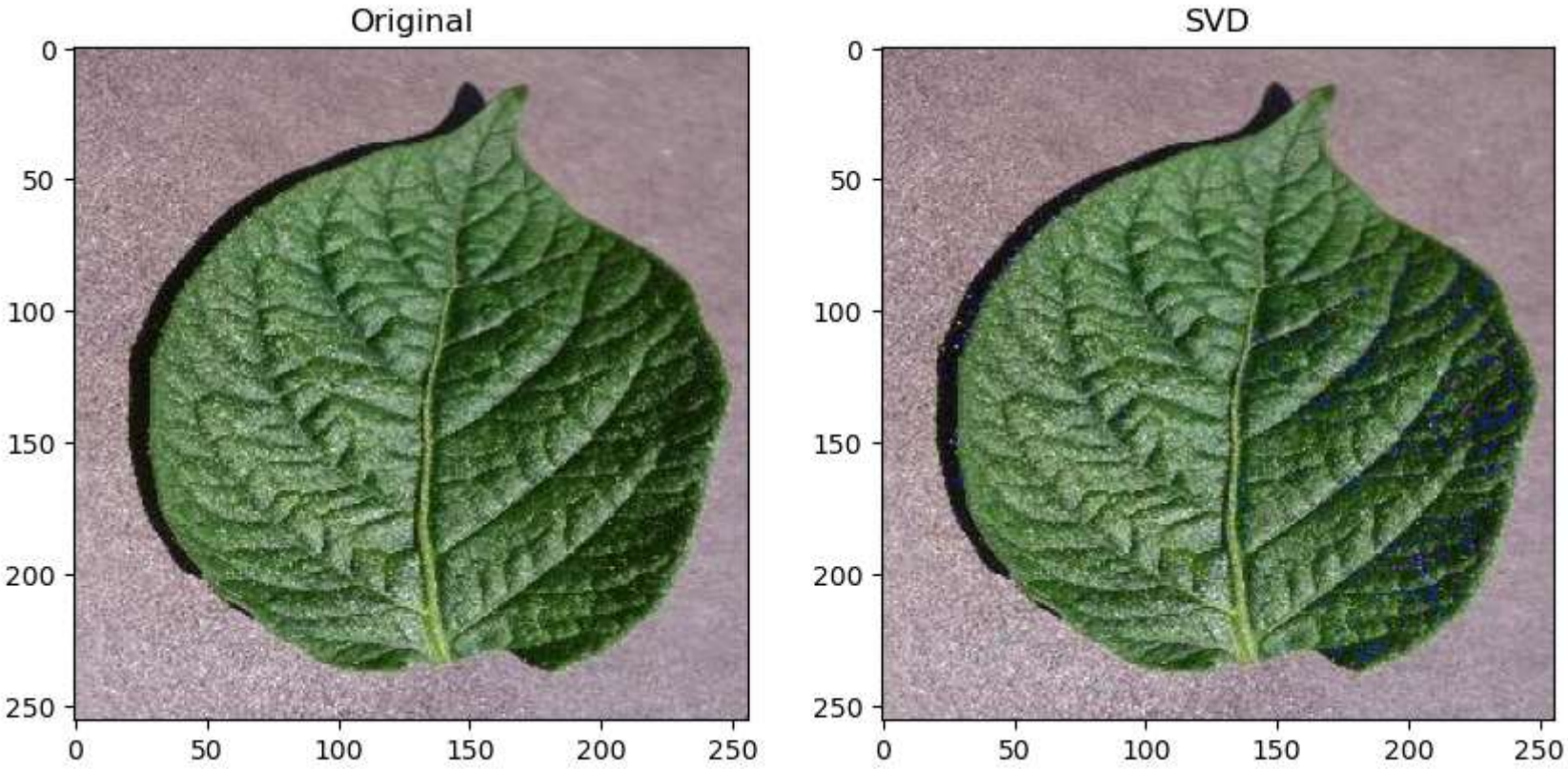
4. Singular Value Decomposition(SVD)

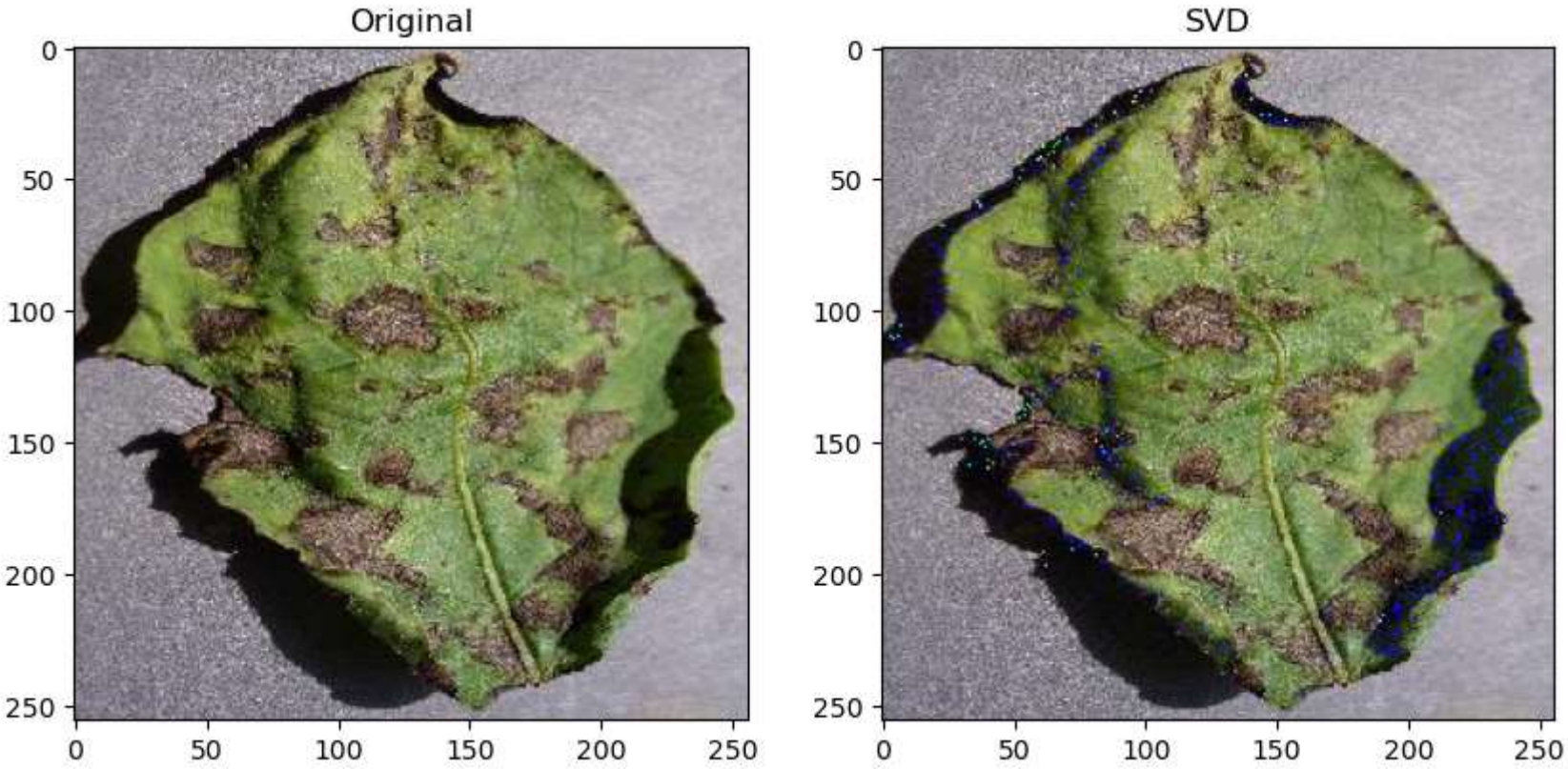
To reduce the amount of pixels, we decided to process the image with the SVD compression. Since Singular Value Decomposition (SVD) is usually used to process grayscale images, it expects the input image to be a two-dimensional matrix rather than a three-dimensional (color image) or higher dimensional image. The color image is divided into three channels, R, G, and B, which are separately compressed by SVD, and then merged into a single color image

In [14]:
```
import os
import numpy as np
from PIL import Image
from scipy.linalg import svd
import matplotlib.pyplot as plt
```

```
# Set the directory paths
dir_path_healthy = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy'
dir_path_Early_blight = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight'
dir_path_Late_blight = 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'

# Create folders for SVD images of different categories
if not os.path.exists('svd_images'):
    os.mkdir('svd_images')

# Function to perform SVD on an image, followed by normalization
def svd_image(image_path, output_path, compression_ratio=0.6):
    img = Image.open(image_path)
    img_array = np.array(img)
    img_shape = img_array.shape

    # Initialize arrays for R, G, and B channels
    R_channel = img_array[:, :, 0]
    G_channel = img_array[:, :, 1]
    B_channel = img_array[:, :, 2]

    # Function to perform SVD and compress a channel
    def svd_compress(channel):
        U, S, Vt = svd(channel, full_matrices=False)
        num_singular_values = int(compression_ratio * min(channel.shape))
        U_truncated = U[:, :num_singular_values]
        S_truncated = np.diag(S[:num_singular_values])
        Vt_truncated = Vt[:num_singular_values, :]
        compressed_channel = np.dot(U_truncated, np.dot(S_truncated, Vt_truncated))
        return compressed_channel

    # Compress each channel using SVD
    compressed_R = svd_compress(R_channel)
    compressed_G = svd_compress(G_channel)
    compressed_B = svd_compress(B_channel)

    # Normalize the compressed image data
    def normalize_data(data):
        return (data - np.min(data)) / (np.max(data) - np.min(data))

    normalized_R = normalize_data(compressed_R)
    normalized_G = normalize_data(compressed_G)
    normalized_B = normalize_data(compressed_B)

    # Stack normalized channels back into an image array
```

```
    normalized_img_array = np.stack([normalized_R, normalized_G, normalized_B], axis=-1)



# Function to display image comparisons
def plot_compressed_image(original_image, compressed_image):
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(original_image)
    axs[0].set_title('Original')
    axs[1].imshow(compressed_image)
    axs[1].set_title('SVD')
    plt.show()

# Display compressed image comparisons (Healthy potatoes)
original_healthy_img = Image.open(os.path.join(dir_path_healthy, os.listdir(dir_path_healthy)[0]))
compressed_healthy_img = Image.open(os.path.join('compressed_images', 'healthy_compressed_' + os.listdir(dir_path_healthy)[0]))
plot_compressed_image(original_healthy_img, compressed_healthy_img)

# Display compressed image comparisons (Early blight potatoes)
original_early_blight_img = Image.open(os.path.join(dir_path_Early_blight, os.listdir(dir_path_Early_blight)[0]))
compressed_early_blight_img = Image.open(os.path.join('compressed_images', 'early_blight_compressed_' + os.listdir(dir_path_Early_b
plot_compressed_image(original_early_blight_img, compressed_early_blight_img)

# Display compressed image comparisons (Late blight potatoes)
original_late_blight_img = Image.open(os.path.join(dir_path_Late_blight, os.listdir(dir_path_Late_blight)[0]))
compressed_late_blight_img = Image.open(os.path.join('compressed_images', 'late_blight_compressed_' + os.listdir(dir_path_Late_blig
plot_compressed_image(original_late_blight_img, compressed_late_blight_img)
```


Original
SVD
0
50
100
150
200
250

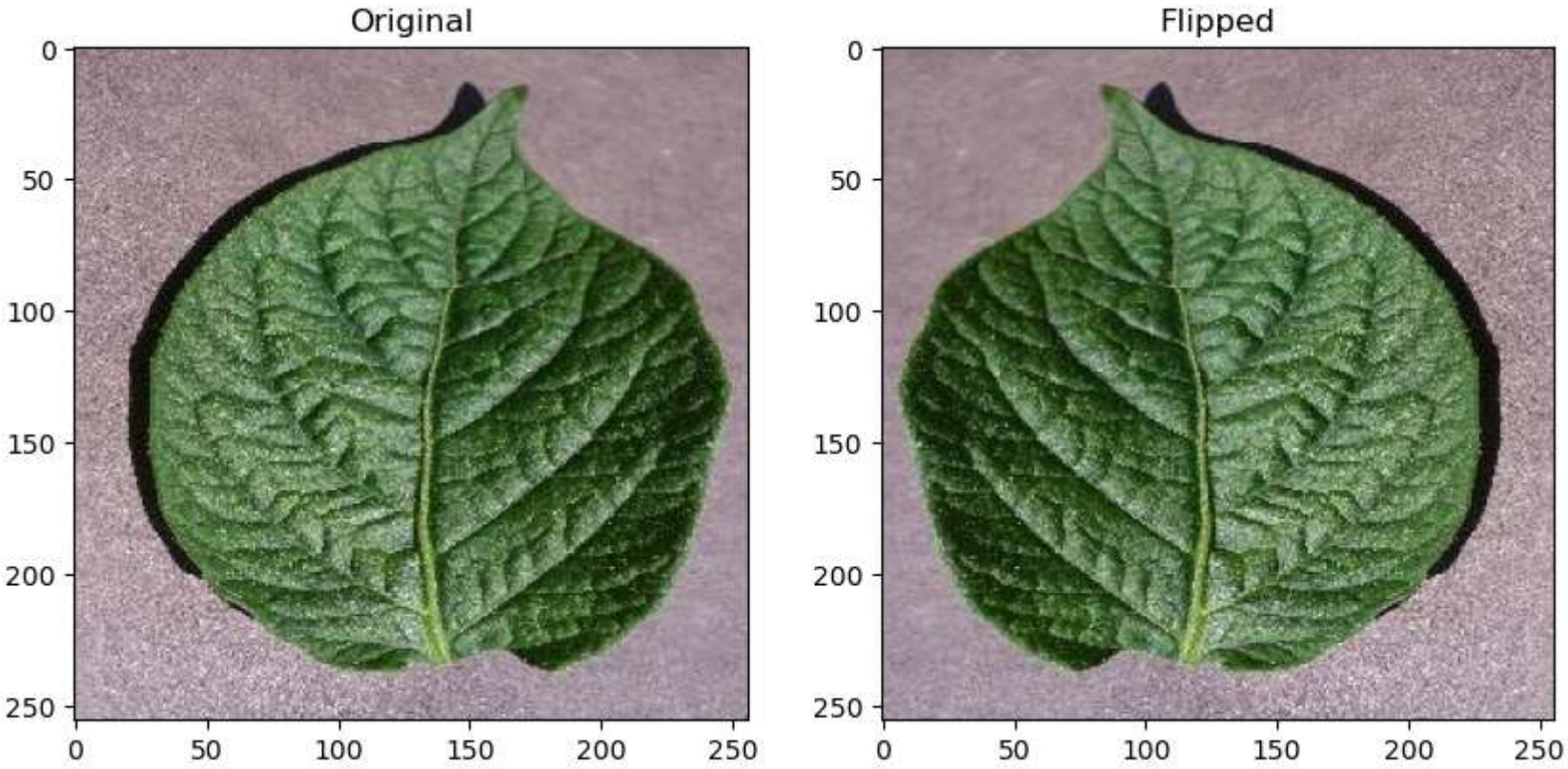

Original
SVD
0
50
100
150
200
250

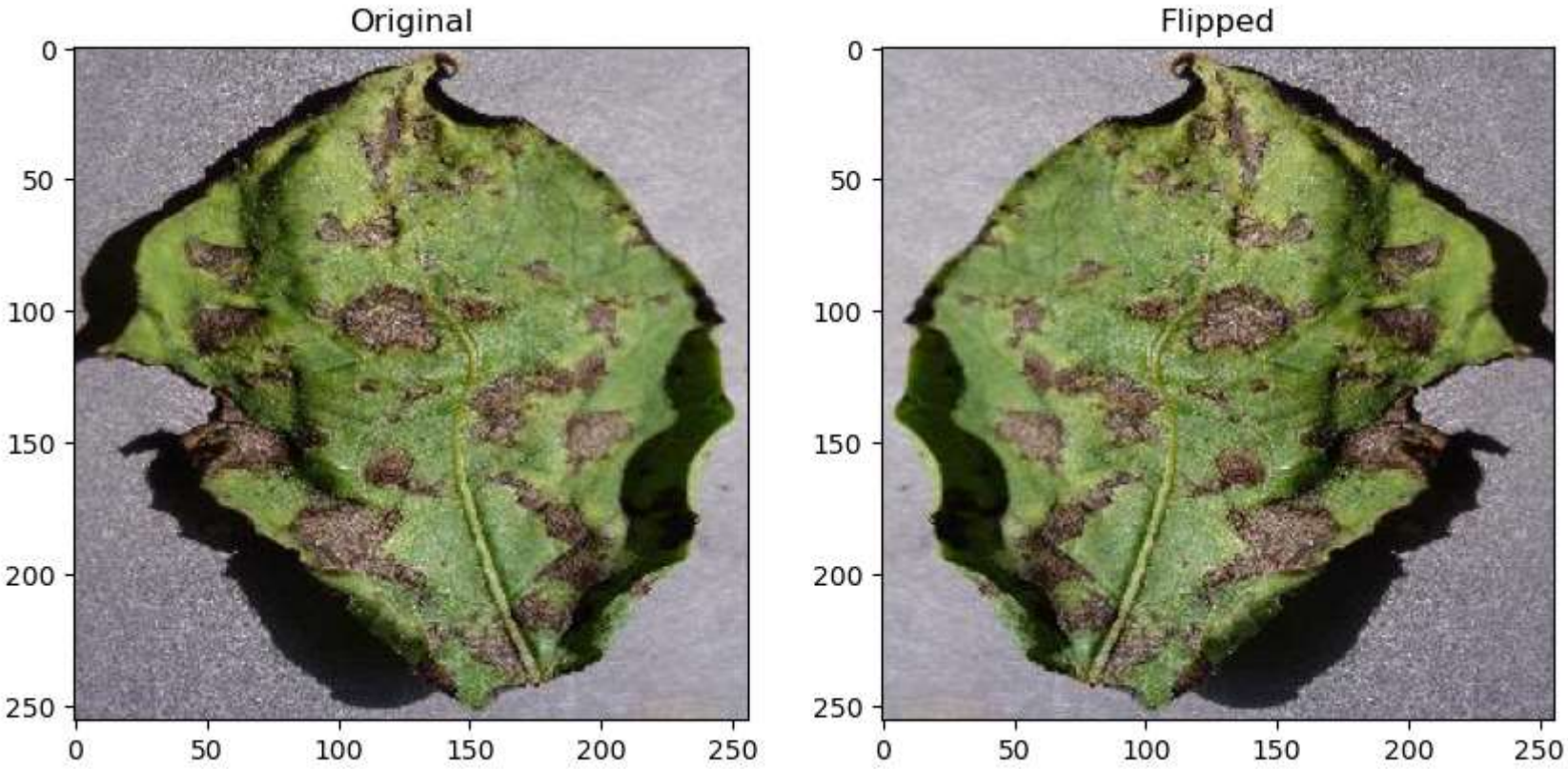
5.Horizontal Flip And Data normalization

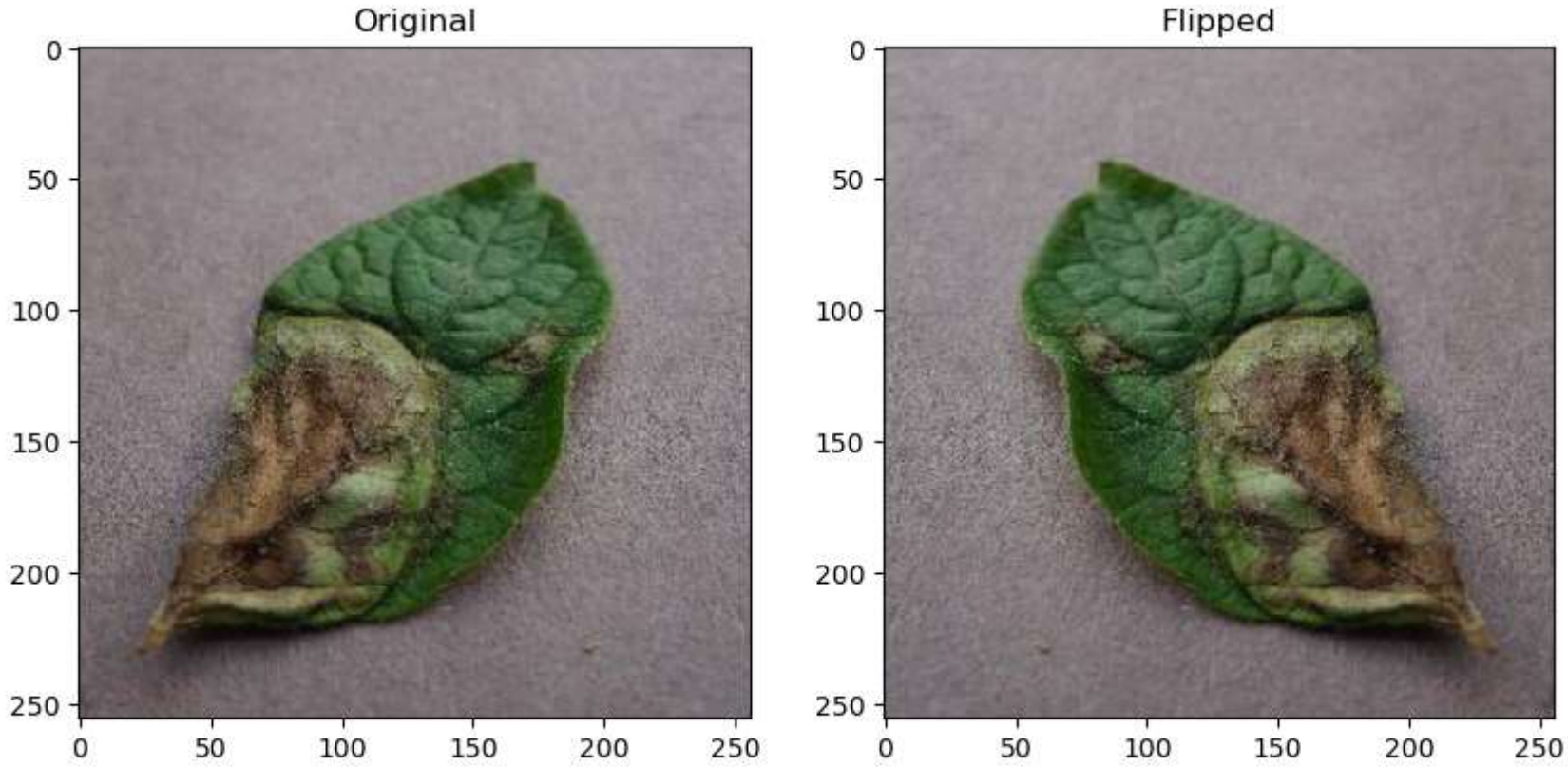
By flipping the original image horizontally, the size and diversity of the training set can be artificially increased. This diversity helps the model learn more features about the shape and texture of the object, rather than relying on a specific orientation or location.

In [6]:
```
import os
import numpy as np
import pandas as pd
from PIL import Image
import matplotlib.pyplot as plt

# Set the directory paths
dir_paths = {
    'healthy': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___healthy',
```

```
    'early_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Early_blight',
    'late_blight': 'D:\\UC BERKELEY\\243\\Module 1\\potato\\color\\Potato___Late_blight'
}

# Function to flip an image horizontally and return it
def flip_image_horizontal(image):
    flipped_img = image.transpose(Image.FLIP_LEFT_RIGHT)
    return flipped_img

# Function to display image comparisons
def plot_image_comparison(original_image, flipped_image):
    fig, axs = plt.subplots(1, 2, figsize=(10, 5))
    axs[0].imshow(original_image)
    axs[0].set_title('Original')
    axs[1].imshow(flipped_image)
    axs[1].set_title('Flipped')
    plt.show()

# Function to normalize and flatten image pixels
def normalize_and_flatten_image(image):
    img_array = np.asarray(image) / 255.0
    return img_array.flatten()

# Process images, display comparison, normalize, and save to CSV
def process_images(dir_path, category):
    normalized_pixels_list = []
    for i, file in enumerate(os.listdir(dir_path)):
        if file.endswith(('.JPG')):
            image_path = os.path.join(dir_path, file)
            img = Image.open(image_path)
            flipped_img = flip_image_horizontal(img)
            if i == 0:  # Only display comparison for the first image
                plot_image_comparison(img, flipped_img)
            flattened_data = normalize_and_flatten_image(flipped_img)
            normalized_pixels_list.append(flattened_data)
    # Create DataFrame and save to CSV
    df = pd.DataFrame(normalized_pixels_list)
    csv_file_path = f'{category}_flipped_normalized_pixel_values.csv'
    df.to_csv(csv_file_path, index=False)

# Process each category of images
for category, dir_path in dir_paths.items():
    process_images(dir_path, category)
```


Original
Flipped
0
50
100
150
200
250


Original
Flipped
0
50
100
150
200
250



In [ ]: