19D100011

Assignment 2

ME 793

In [28]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

Q1: Convert RGB image to Greyscale image

In [18]:
```
# Load the RGB image
rgb_image = cv2.imread('Dog.jpg')

# Convert RGB to grayscale
gray_image = cv2.cvtColor(rgb_image, cv2.COLOR_BGR2GRAY)

# Save the grayscale image
cv2.imwrite('Dog_gray.jpg', gray_image)
```

Out[18]:
```
True
```

In [27]:
```
# Show the converted image
plt.figure()

# Read the image file
orignal_img = plt.imread("Dog.jpg")
grey_img = plt.imread("Dog_gray.jpg")

plt.subplot(1, 2, 1)
plt.imshow(orignal_img)
plt.title("Orignal Image")
plt.axis('off')
```

```
plt.subplot(1, 2, 2)
plt.imshow(grey_img)
plt.title("Greyscale Image")
plt.axis('off')

# Show the plot
plt.show()
```

Q2: Perform SVD image compression with varying numbers of singular values and plot the reconstructed images

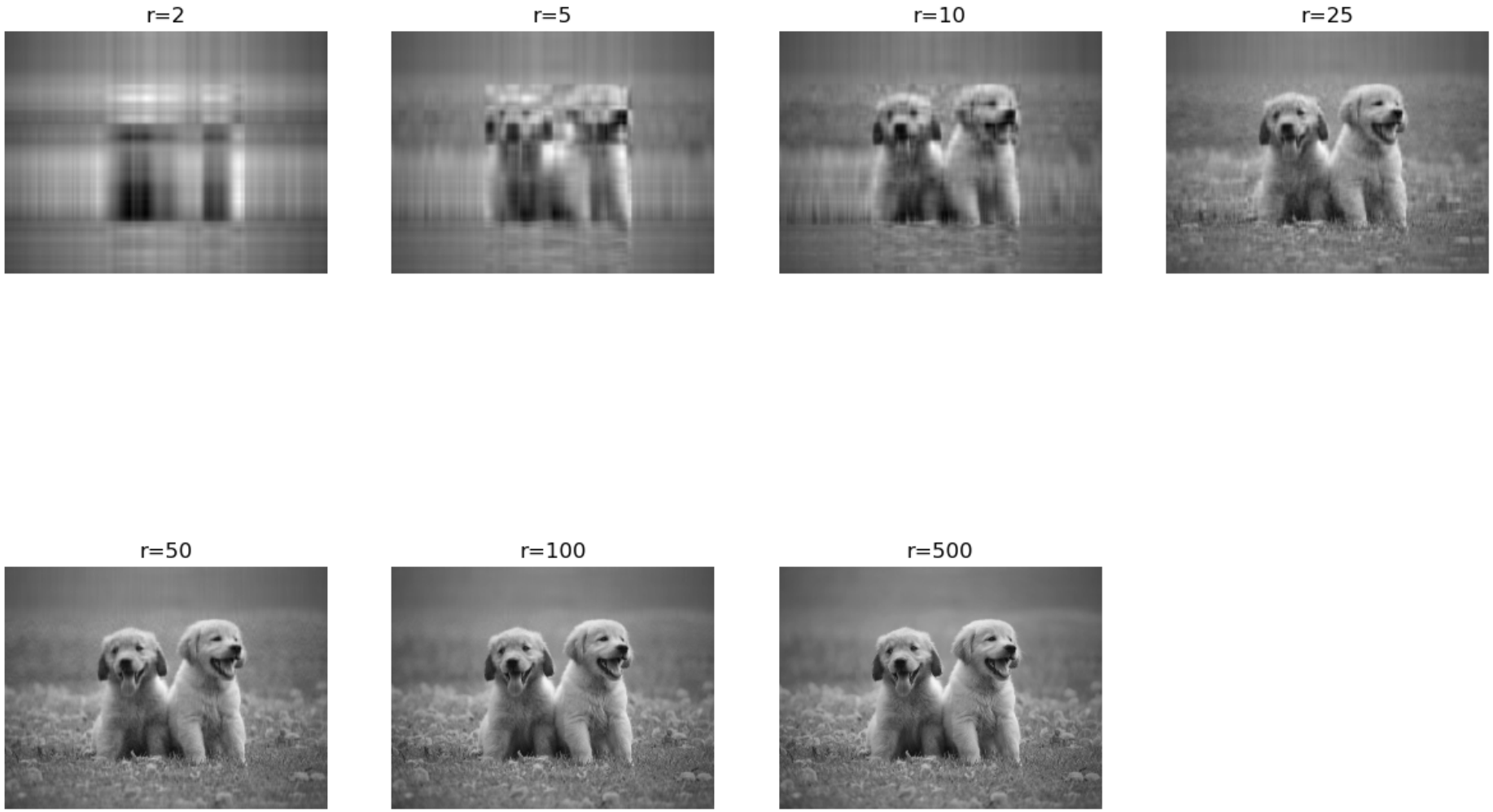
In [19]:
```
# Load the grayscale image
gray_image = cv2.imread('Dog_gray.jpg', cv2.IMREAD_GRAYSCALE)

# Perform SVD
U, S, Vt = np.linalg.svd(gray_image)

# Singular values to test
singular_values = [2, 5, 10, 25, 50, 100, 500]

# Plot reconstructed images
plt.figure(figsize=(15, 10))
for i, r in enumerate(singular_values, 1):
    reconstructed_image = np.dot(U[:, :r], np.dot(np.diag(S[:r]), Vt[:r, :]))
    plt.subplot(2, 4, i)
    plt.imshow(reconstructed_image, cmap='gray')
    plt.title(f'r={r}')
```

```
    plt.axis('off')
plt.show()
```



Q3: Memory space required for storage change with different values of 'r'

The memory required for storage can be calculated based on the dimensions of the matrices involved in the SVD process. For a grayscale image, the size can be calculated as:

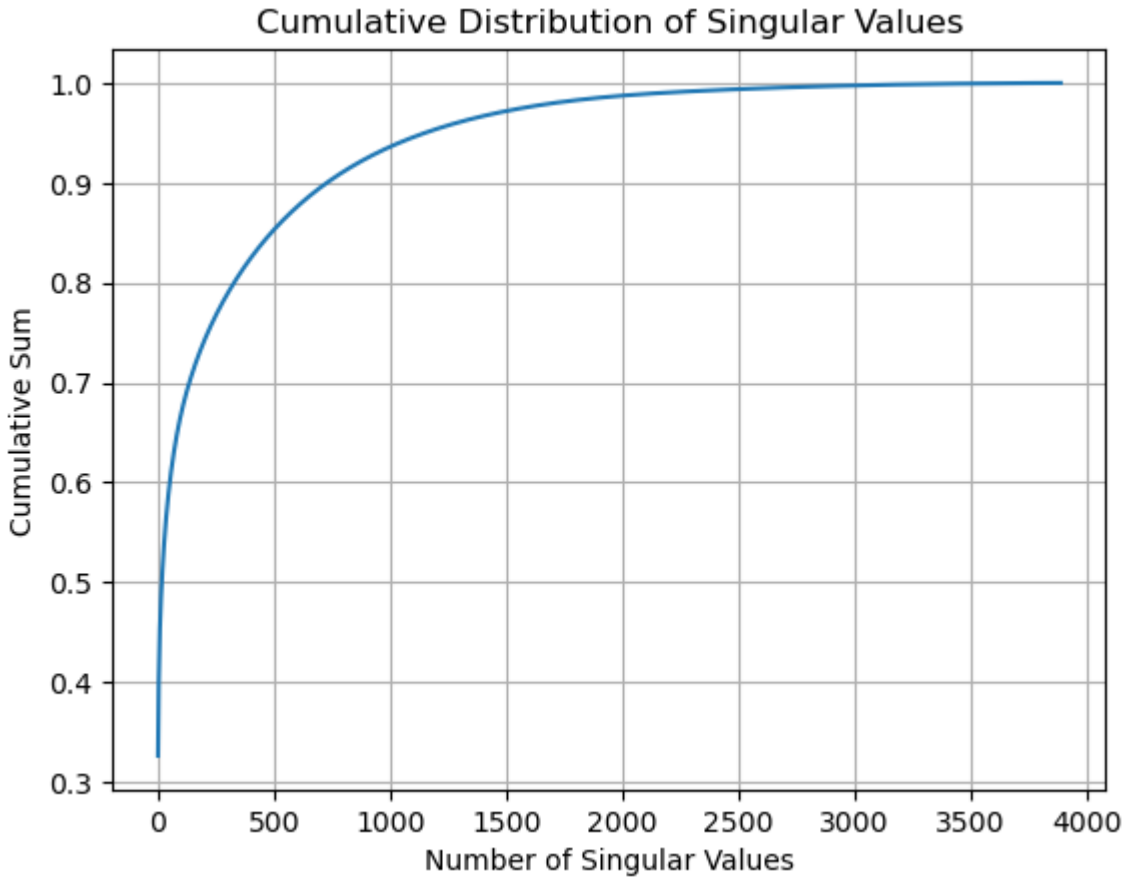
Size = (M * N) + (M * r) + (r * N)

where: M and N are the dimensions of the image (height and width) r is the number of singular values used for compression

Q4: Plot cumulative distribution curve change with different values of singular values

In [20]:
```
# Calculate cumulative distribution
cumulative = np.cumsum(S) / np.sum(S)

# Plot cumulative distribution curve
plt.plot(cumulative)
plt.title('Cumulative Distribution of Singular Values')
plt.xlabel('Number of Singular Values')
plt.ylabel('Cumulative Sum')
plt.grid(True)
plt.show()
```

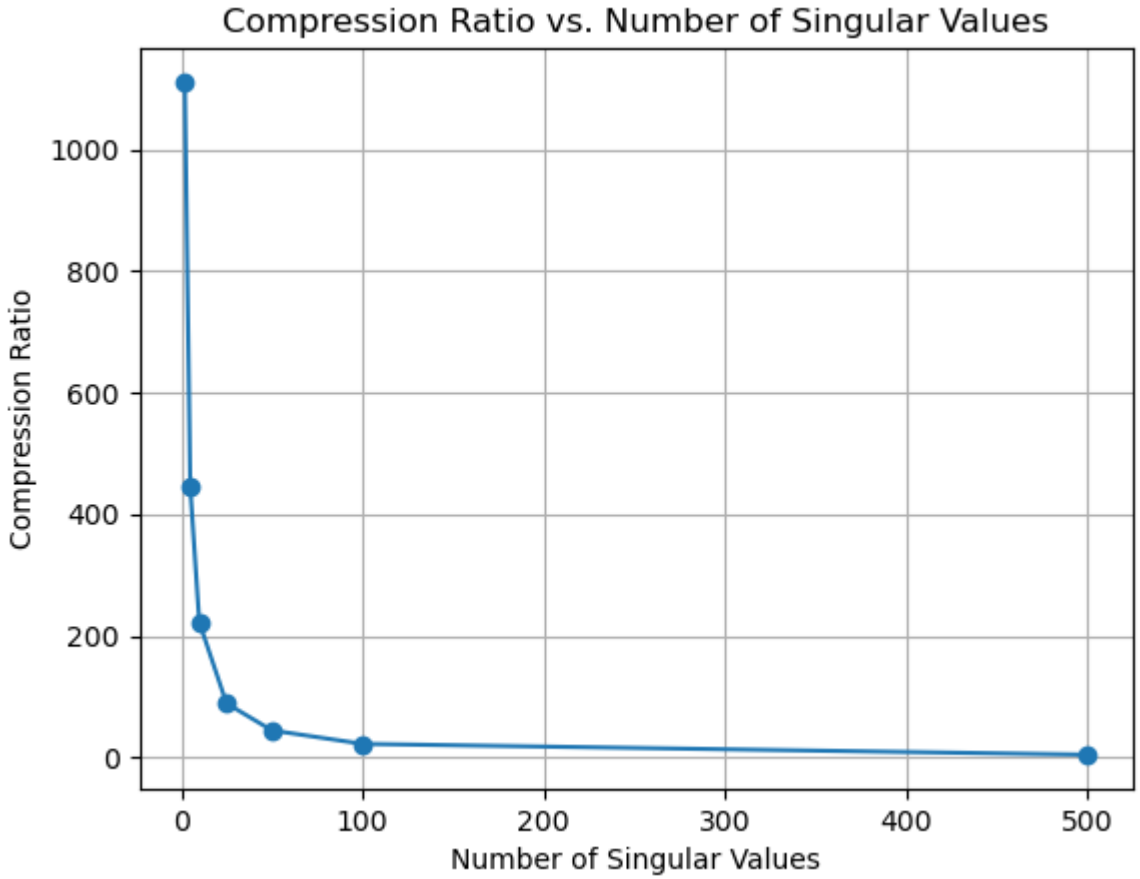


Q5: Calculate the compression ratio and plot with respect to singular values

In [21]:
```
# Calculate compression ratios
original_size = gray_image.size
compression_ratios = [original_size / ((U[:, :r].size + r + Vt[:r, :].size)) for r in singular_values]

# Plot compression ratios
plt.plot(singular_values, compression_ratios, marker='o')
plt.title('Compression Ratio vs. Number of Singular Values')
plt.xlabel('Number of Singular Values')
plt.ylabel('Compression Ratio')
```

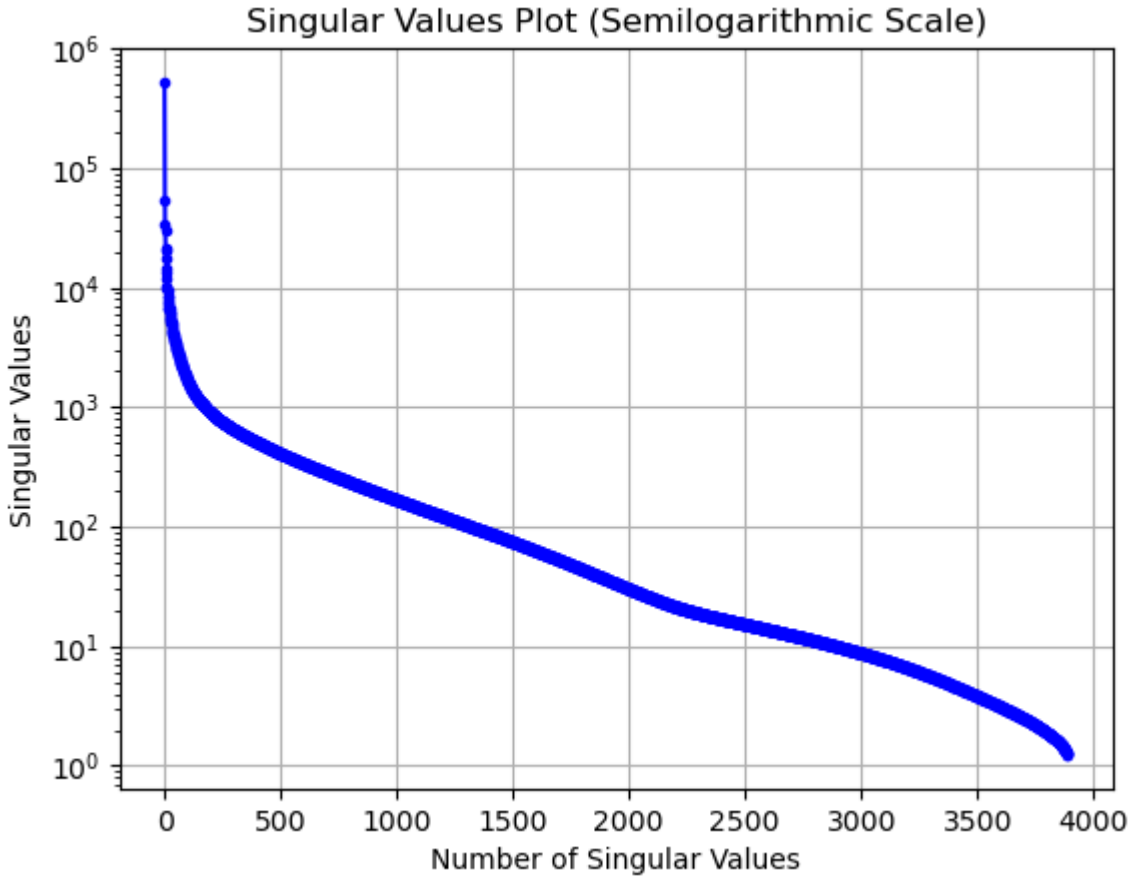
```
plt.grid(True)
plt.show()
```



Q6: Plot the Singular Values Plot (Semilogarithmic Scale) between Number of singular values and Singular values

In [22]:
```
# Plot Singular Values Plot (Semilogarithmic Scale)
plt.plot(S, 'b.-')
plt.title('Singular Values Plot (Semilogarithmic Scale)')
plt.xlabel('Number of Singular Values')
plt.ylabel('Singular Values')
plt.grid(True)
```

```
plt.yscale('log')
plt.show()
```



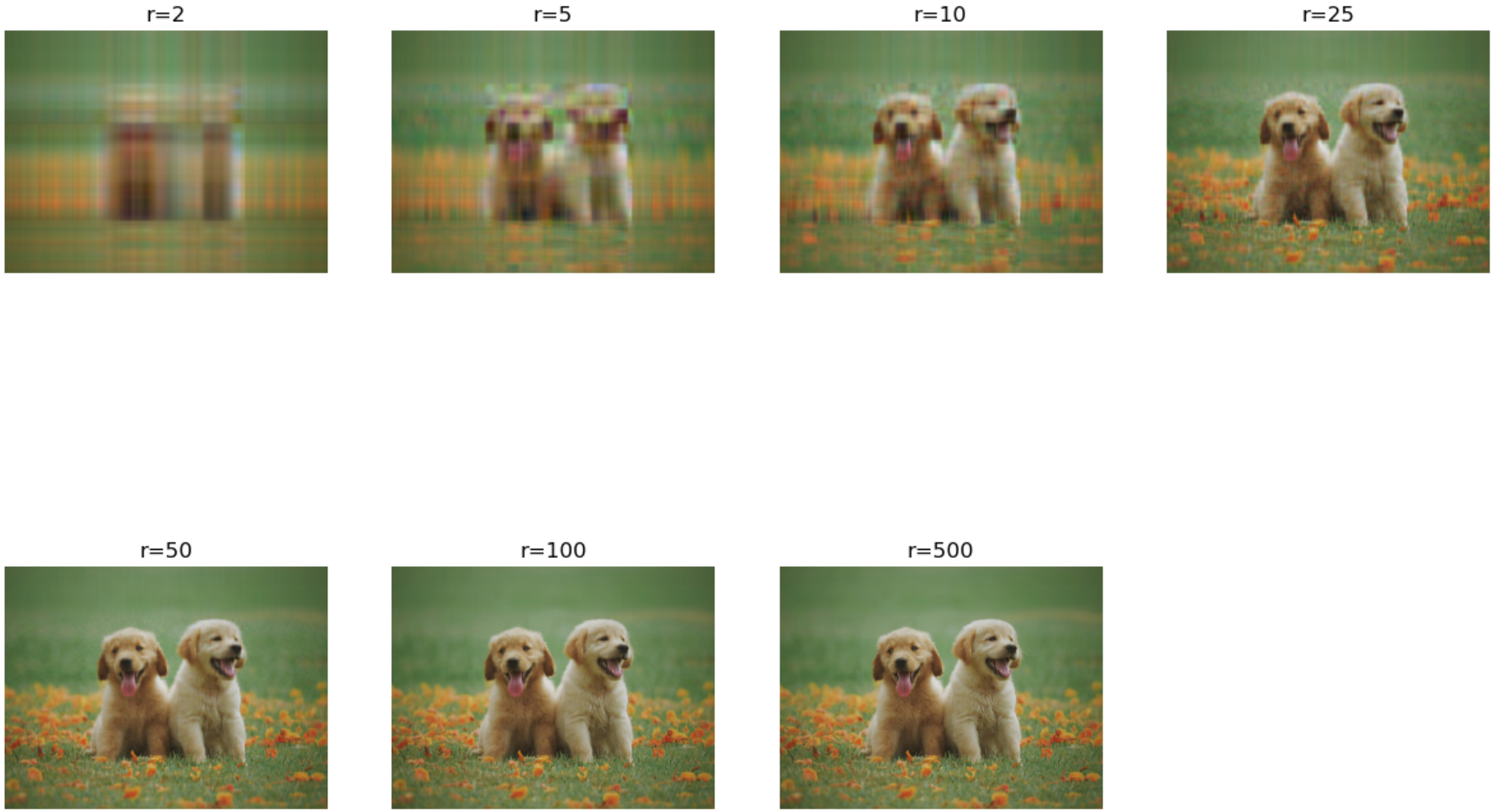
Q7: Repeat Q2 for the original RGB image. Show three channels of RGB image along with the original images.

In [23]:
```
# Perform SVD on each channel separately
U_r, S_r, Vt_r = np.linalg.svd(rgb_image[:, :, 0])
U_g, S_g, Vt_g = np.linalg.svd(rgb_image[:, :, 1])
U_b, S_b, Vt_b = np.linalg.svd(rgb_image[:, :, 2])

# Plot reconstructed images for each channel
```

```
plt.figure(figsize=(15, 10))
for i, r in enumerate(singular_values, 1):
    reconstructed_r = np.dot(U_r[:, :r], np.dot(np.diag(S_r[:r]), Vt_r[:r, :]))
    reconstructed_g = np.dot(U_g[:, :r], np.dot(np.diag(S_g[:r]), Vt_g[:r, :]))
    reconstructed_b = np.dot(U_b[:, :r], np.dot(np.diag(S_b[:r]), Vt_b[:r, :]))
    reconstructed_image = np.stack([reconstructed_r, reconstructed_g, reconstructed_b], axis=-1).astype(np.uint8)

    plt.subplot(2, 4, i)
    plt.imshow(cv2.cvtColor(reconstructed_image, cv2.COLOR_BGR2RGB))
    plt.title(f'r={r}')
    plt.axis('off')
plt.show()
```

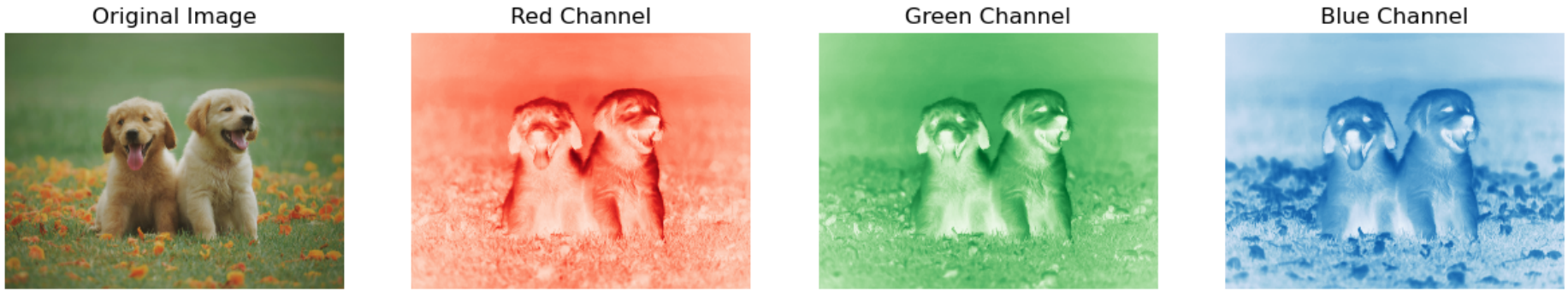


In [24]:
```
# Show original images
plt.figure(figsize=(15, 5))
plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(rgb_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.axis('off')

# Show individual channels
for i, (channel, color) in enumerate(zip(['Red', 'Green', 'Blue'], ['Reds', 'Greens', 'Blues']), 2):
    plt.subplot(1, 4, i)
    plt.imshow(rgb_image[:, :, i-2], cmap=color)
```

```
    plt.title(f'{channel} Channel')
    plt.axis('off')

plt.show()
```



In [ ]: