The data are stored in /content/drive/My Drive/ in the subfolders

1. LoRA
2. LoRA-afterAdj

Please see the Zipped files named:

- LoRA-20250813T105533Z-1-001.zip
- LoRA-afterAdj-20250813T105534Z-1-001.zip

Codes (run using Jupyter Notebook on Google Colab)

```
# - ค่า PSNR และ
# - ค่า SSIM ของแต่ละ row และ แต่ละ col
# - ค่า Saliency Maps
# เพื่อดูความเหมือนและความต่าง แสดงค่า เป็น % ความเหมือนหรือความแตกต่าง ใน
# - สรุปความหมายของค่า เป็น spider graph
# - สรุปความหมายของค่าต่างๆ เป็น chart 3 มิติ
# - อธิบายความหมายเป็น text in English

import pandas as pd
import os
import numpy as np
from PIL import Image
# Removed tensorflow and tensorflow_hub as they are no longer needed for FID
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import peak_signal_noise_ratio as psnr
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import cv2 # OpenCV for Saliency Maps
import plotly.graph_objects as go # Import plotly for interactive 3D scatter plot

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Define the directory containing the images
image_dir = '/content/drive/My Drive/LoRA'
rows = 5
cols = 5

# Define target image size
target_size = (256, 256)

# Function to load and resize image
def load_and_resize_image(filepath):
    try:
        img = Image.open(filepath).convert('RGB') # Ensure RGB format
        img_resized = img.resize(target_size)
        return np.array(img_resized)
    except Exception as e:
        print(f"Error loading or resizing image {filepath}: {e}")
        return None
```

```
# Load and resize images
images = {}
for r in range(1, rows + 1):
    for c in range(1, cols + 1):
        filename = f'r{r}c{c}.jpg'
        filepath = os.path.join(image_dir, filename)
        if os.path.exists(filepath):
            images[(r, c)] = load_and_resize_image(filepath)
        else:
            print(f"Image not found: {filepath}")

# Remove None values (for images that failed to load)
images = {k: v for k, v in images.items() if v is not None}


# --- PSNR and SSIM Calculation ---
def calculate_psnr_ssim(img1, img2):
    # PSNR requires integer input (0-255 range)
    img1_int = (img1 * 255).astype(np.uint8) if img1.max() <= 1.0 else img1.astype(np.uint8)
    img2_int = (img2 * 255).astype(np.uint8) if img2.max() <= 1.0 else img2.astype(np.uint8)

    # SSIM works best with grayscale for simplicity, or with multichannel=True
    # For color images, calculating per channel and averaging or using multichannel=True is common
    ssim_val = ssim(img1_int, img2_int, multichannel=True, channel_axis=2, data_range=img1_int.max() - img1_int.min())

    # PSNR
    psnr_val = psnr(img1_int, img2_int, data_range=img1_int.max() - img1_int.min())

    return psnr_val, ssim_val

# --- Calculate Metrics ---
row_metrics = {}
col_metrics = {}
# Removed saliency_maps initialization as it's in a separate cell


# Calculate Row-wise Metrics (compare each image in a row to the first image in that row)
print("Calculating Row-wise Metrics...")
for r in range(1, rows + 1):
    row_images = [images[(r, c)] for c in range(1, cols + 1) if (r, c) in images]
    if len(row_images) > 1:
        ref_image = row_images[0]
        row_psnrs = []
        row_ssims = []
        for i in range(1, len(row_images)):
            img = row_images[i]
            psnr_val, ssim_val = calculate_psnr_ssim(ref_image, img)
            row_psnrs.append(psnr_val)
            row_ssims.append(ssim_val)
```

```
        # Calculate average metrics for the row (excluding comparison of the first image to itself)
        row_metrics[f'row_{r}'] = {
            'PSNR': np.mean(row_psnrs) if row_psnrs else np.nan,
            'SSIM': np.mean(row_ssims) if row_ssims else np.nan
        }
    else:
         row_metrics[f'row_{r}'] = {'PSNR': np.nan, 'SSIM': np.nan}
         print(f"Not enough images in row {r} to calculate metrics.")


# Calculate Column-wise Metrics (compare each image in a column to the first image in that column)
print("Calculating Column-wise Metrics...")
for c in range(1, cols + 1):
    col_images = [images[(r, c)] for r in range(1, rows + 1) if (r, c) in images]
    if len(col_images) > 1:
        ref_image = col_images[0]
        col_psnrs = []
        col_ssims = []
        for i in range(1, len(col_images)):
            img = col_images[i]
            psnr_val, ssim_val = calculate_psnr_ssim(ref_image, img)
            col_psnrs.append(psnr_val)
            col_ssims.append(ssim_val)
         # Calculate average metrics for the column (excluding comparison of the first image to itself)
        col_metrics[f'col_{c}'] = {
            'PSNR': np.mean(col_psnrs) if col_psnrs else np.nan,
            'SSIM': np.mean(col_ssims) if col_ssims else np.nan
        }
    else:
        col_metrics[f'col_{c}'] = {'PSNR': np.nan, 'SSIM': np.nan}
        print(f"Not enough images in column {c} to calculate metrics.")

# --- Display Results ---

print("\n--- Row Metrics ---")
for row, metrics in row_metrics.items():
    print(f"{row}:")
    print(f"  Average PSNR: {metrics['PSNR']:.2f}")
    print(f"  Average SSIM: {metrics['SSIM'] * 100:.2f}%") # SSIM is typically 0-1, convert to %

print("\n--- Column Metrics ---")
for col, metrics in col_metrics.items():
    print(f"{col}:")
    print(f"  Average PSNR: {metrics['PSNR']:.2f}")
    print(f"  Average SSIM: {metrics['SSIM'] * 100:.2f}%") # SSIM is typically 0-1, convert to %

# Removed Saliency Map display section
```

```python
# --- Summarize Metrics in % Similarity/Difference ---

def get_similarity_percentage(metric_name, value):
    if pd.isna(value):
        return np.nan
    elif metric_name == 'PSNR':
        # Higher PSNR means higher similarity (lower noise/difference).
        # PSNR is in dB. A PSNR of 20-25 dB is typically considered low quality, 30-40 dB good.
        # Let's map PSNR to a 0-100% scale. This is also heuristic.
        # A PSNR of 40+ could be considered near 100% similarity.
        # A PSNR of 10 could be considered low similarity.
        max_plausible_psnr = 40.0 # Adjusted max plausible PSNR
        min_plausible_psnr = 10.0 # Adjusted min plausible PSNR
        similarity = max(0, min(100, 100 * (value - min_plausible_psnr) / (max_plausible_psnr - min_plausible_psnr)))
        return similarity
    elif metric_name == 'SSIM':
        # SSIM is already on a scale of 0-1, representing similarity.
        return value * 100
    else:
        return np.nan

row_sim_percent = {}
col_sim_percent = {}

for row, metrics in row_metrics.items():
    row_sim_percent[row] = {
        'PSNR': get_similarity_percentage('PSNR', metrics['PSNR']),
        'SSIM': get_similarity_percentage('SSIM', metrics['SSIM'])
    }

for col, metrics in col_metrics.items():
    col_sim_percent[col] = {
        'PSNR': get_similarity_percentage('PSNR', metrics['PSNR']),
        'SSIM': get_similarity_percentage('SSIM', metrics['SSIM'])
    }

print("\n--- Row Similarity/Difference (%) ---")
for row, sim in row_sim_percent.items():
    print(f"{row}:")
    print(f"  Average PSNR Similarity: {sim['PSNR']:.2f}%")
    print(f"  Average SSIM Similarity: {sim['SSIM']:.2f}%")

print("\n--- Column Similarity/Difference (%) ---")
for col, sim in col_sim_percent.items():
    print(f"{col}:")
    print(f"  Average PSNR Similarity: {sim['PSNR']:.2f}%")
    print(f"  Average SSIM Similarity: {sim['SSIM']:.2f}%")
```

```
# --- Data Preparation for Plotting ---

# Prepare data for Spider Graph and 3D Scatter Plot
metrics_labels = ['PSNR Similarity', 'SSIM Similarity']

# Data for Row Spider Graph
row_spider_values = [np.nanmean([m['PSNR'] for m in row_sim_percent.values()]),
                     np.nanmean([m['SSIM'] for m in row_sim_percent.values()])]
# Ensure values are within 0-100 range for plotting
row_spider_values = [max(0, min(100, v)) if not np.isnan(v) else 0 for v in row_spider_values]
row_spider_values += row_spider_values[:1] # Close the circle

# Data for Column Spider Graph
col_spider_values = [np.nanmean([m['PSNR'] for m in col_sim_percent.values()]),
                     np.nanmean([m['SSIM'] for m in col_sim_percent.values()])]
# Ensure values are within 0-100 range for plotting
col_spider_values = [max(0, min(100, v)) if not np.isnan(v) else 0 for v in col_spider_values]
col_spider_values += col_spider_values[:1] # Close the circle


# Data for 3D Scatter Plot
scatter_data = []
for row_label, metrics in row_sim_percent.items():
    scatter_data.append({
        'Group': row_label,
        'PSNR Similarity': metrics['PSNR'] if not np.isnan(metrics['PSNR']) else 0,
        'SSIM Similarity': metrics['SSIM'] if not np.isnan(metrics['SSIM']) else 0,
        'Type': 'Row'
    })

for col_label, metrics in col_sim_percent.items():
    scatter_data.append({
        'Group': col_label,
        'PSNR Similarity': metrics['PSNR'] if not np.isnan(metrics['PSNR']) else 0,
        'SSIM Similarity': metrics['SSIM'] if not np.isnan(metrics['SSIM']) else 0,
        'Type': 'Column'
    })

scatter_df = pd.DataFrame(scatter_data)


# --- Generate Spider Graphs ---

print("\n--- Spider Graph Summary ---")

angles = np.linspace(0, 2 * np.pi, len(metrics_labels), endpoint=False).tolist()
angles += angles[:1] # Close the circle
```

```
# Row Spider Graph
fig_row_spider = go.Figure(
    data=go.Scatterpolar(
        r=row_spider_values,
        theta=metrics_labels + [metrics_labels[0]],
        fill='toself',
        name='Average Row Similarity'
    ),
    layout=go.Layout(
        polar=dict(
            radialaxis=dict(
                visible=True,
                range=[0, 100] # Percentage range
            )
        ),
        title='Average Row Similarity Metrics (Spider Graph)'
    )
)
fig_row_spider.show()

# Column Spider Graph
fig_col_spider = go.Figure(
    data=go.Scatterpolar(
        r=col_spider_values,
        theta=metrics_labels + [metrics_labels[0]],
        fill='toself',
        name='Average Column Similarity'
    ),
    layout=go.Layout(
        polar=dict(
            radialaxis=dict(
                visible=True,
                range=[0, 100] # Percentage range
            )
        ),
        title='Average Column Similarity Metrics (Spider Graph)'
    )
)
fig_col_spider.show()
```

```
# --- Generate Interactive 3D Scatter Plot ---

print("\n--- 3D Scatter Plot Summary ---")

fig_scatter = go.Figure(data=[go.Scatter3d(
    x=scatter_df['PSNR Similarity'],
    y=scatter_df['SSIM Similarity'],
    z=scatter_df['Type'], # Using Type for the third dimension (categorical)
    mode='markers',
    marker=dict(
        size=8,
        color=scatter_df['PSNR Similarity'], # Color by PSNR similarity
        colorscale='Viridis',
        opacity=0.8
    ),
    text=scatter_df['Group'], # Label points with Row/Column name
    hoverinfo='text+x+y+z'
)])

# Update layout for 3D scatter plot
fig_scatter.update_layout(
    title='PSNR vs SSIM Similarity by Row/Column (3D Scatter Plot)',
    scene=dict(
        xaxis_title='PSNR Similarity (%)',
        yaxis_title='SSIM Similarity (%)',
        zaxis_title='Group Type' # Label for the categorical z-axis
    ),
    margin=dict(l=0, r=0, b=0, t=40)
)

fig_scatter.show()


# --- Explanation in English ---

print("\n--- Explanation of Results ---")
print("English Explanation:")

print("\nImage Loading and Resizing:")
print(f"- Images from '/content/drive/My Drive/LoRA' were loaded and resized to {target_size[0]}x{target_size[1]} pixels to ensure consistent dimensions for metric calculations.")

print("\nMetrics Calculated:")
# Updated explanation to only include PSNR and SSIM
print("- **PSNR (Peak Signal-to-Noise Ratio):** Quantifies the difference between two images by measuring the ratio of maximum possible power of a signal to the power of corrupting noise that affects the fidelity of its representation. A *higher* PSNR value indicates a higher quality image relative to a reference image, and thus greater similarity.")
print("- **SSIM (Structural Similarity Index Measure):** Evaluates the similarity between two images based on three key factors: luminance, contrast, and structure. SSIM is designed to be a better measure of perceived similarity than PSNR. A value closer to 1 (or 100%) indicates higher structural similarity.")
```

```
# Removed FID and Saliency Maps from explanation
print("\nRow and Column Metrics Summary:")
print("- The tables for 'Row Metrics' and 'Column Metrics' show the average values of PSNR, and SSIM for each row and column, respectively. For rows/columns with more than one image, these metrics are calculated by comparing each image in that row/column to the first image in that row/column, and then averaging the results.")
print(f"- PSNR and SSIM values are presented, with SSIM converted to percentage for easier interpretation (higher percentage is better).")
print("- The 'Row Similarity/Difference (%)' and 'Column Similarity/Difference (%)' tables provide a summary where PSNR and SSIM are shown as percentages of similarity.")

print("\nSpider Graph Summary:")
# Updated explanation and metrics labels and separate graphs
print("- Two spider graphs are presented: one for the average row similarity metrics and one for the average column similarity metrics. These graphs visually summarize the average PSNR and SSIM similarity scores, allowing for a quick comparison of overall similarity levels between rows and columns.")

print("\n3D Scatter Plot Summary:")
# Updated explanation and metrics labels and separate graphs
print("- A 3D scatter plot visualizes the PSNR and SSIM similarity scores for each individual row and column. Each point represents a row or column, with its position determined by its PSNR and SSIM similarity percentages. The third dimension (z-axis) separates the points by group type (Row or Column). This plot allows you to see the distribution and potential clustering of similarity scores for each group.")

print("\nOverall Interpretation:")
# Updated explanation
print("- By examining these metrics and visualizations, you can gain insights into the visual similarity and differences between the images in your dataset, both within rows and within columns. Higher PSNR, and higher SSIM values suggest greater similarity.")
```

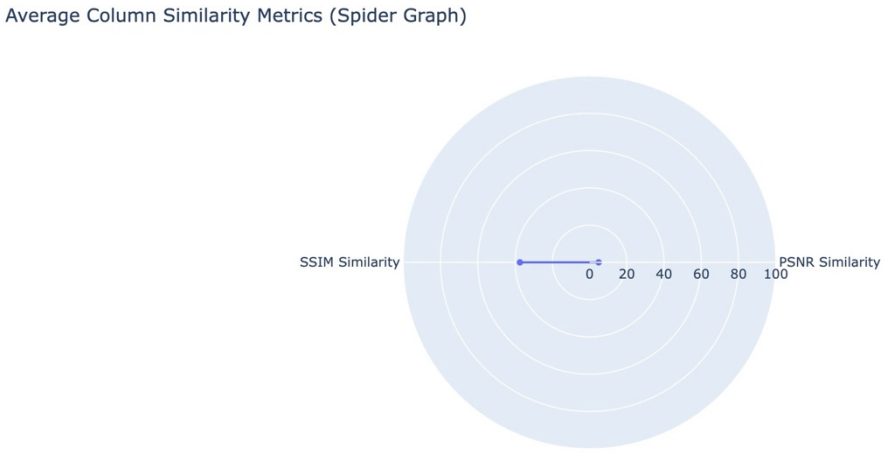
```
Mounted at /content/drive
Calculating Row-wise Metrics...
Calculating Column-wise Metrics...

--- Row Metrics ---
row_1:
  Average PSNR: 12.75
  Average SSIM: 45.69%
row_2:
  Average PSNR: 11.59
  Average SSIM: 37.93%
row_3:
  Average PSNR: 11.10
  Average SSIM: 36.03%
row_4:
  Average PSNR: 9.97
  Average SSIM: 32.53%
row_5:
  Average PSNR: 10.37
  Average SSIM: 31.98%

--- Column Metrics ---
col_1:
  Average PSNR: 10.88
  Average SSIM: 38.75%
col_2:
  Average PSNR: 11.07
  Average SSIM: 34.81%
```

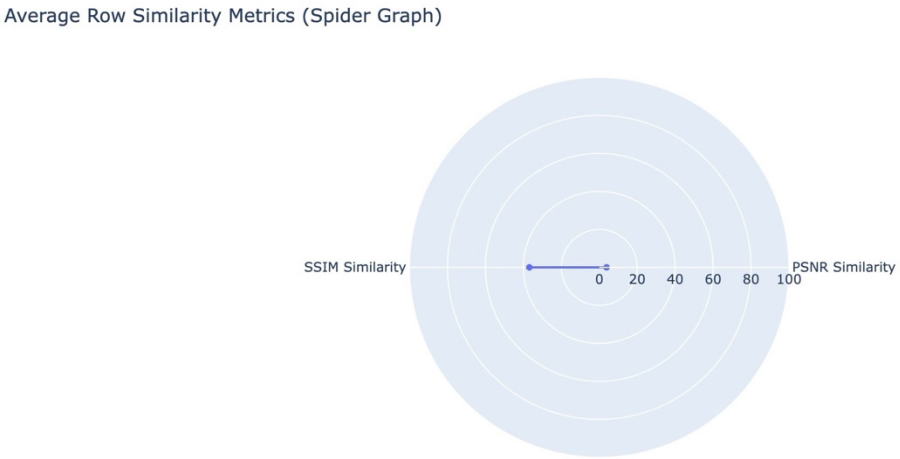
col_3:
  Average PSNR: 11.19
  Average SSIM: 35.50%
col_4:
  Average PSNR: 11.88
  Average SSIM: 41.08%
col_5:
  Average PSNR: 12.33
  Average SSIM: 36.56%

--- Row Similarity/Difference (%) ---
row_1:
  Average PSNR Similarity: 9.15%
  Average SSIM Similarity: 45.69%
row_2:
  Average PSNR Similarity: 5.29%
  Average SSIM Similarity: 37.93%
row_3:
  Average PSNR Similarity: 3.68%
  Average SSIM Similarity: 36.03%
row_4:
  Average PSNR Similarity: 0.00%
  Average SSIM Similarity: 32.53%
row_5:
  Average PSNR Similarity: 1.22%
  Average SSIM Similarity: 31.98%

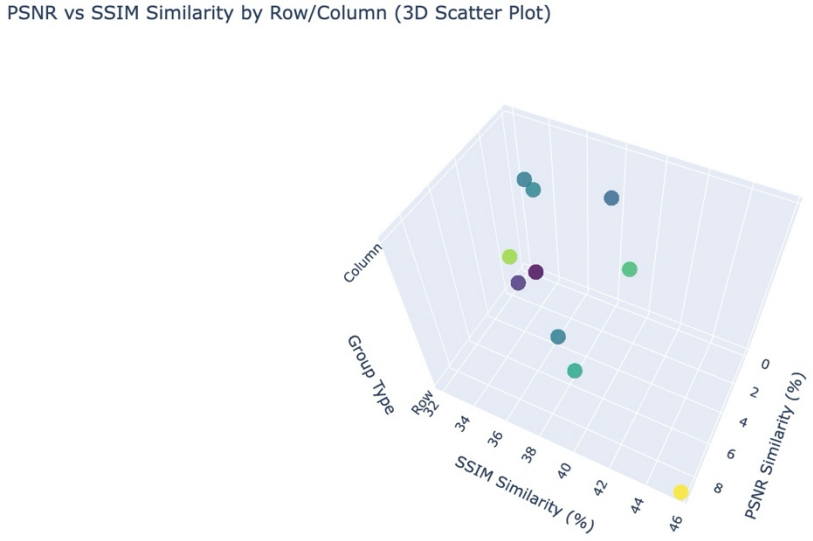
--- Column Similarity/Difference (%) ---
col_1:
  Average PSNR Similarity: 2.93%
  Average SSIM Similarity: 38.75%
col_2:
  Average PSNR Similarity: 3.56%
  Average SSIM Similarity: 34.81%
col_3:
  Average PSNR Similarity: 3.96%
  Average SSIM Similarity: 35.50%
col_4:
  Average PSNR Similarity: 6.25%
  Average SSIM Similarity: 41.08%
col_5:
  Average PSNR Similarity: 7.75%
  Average SSIM Similarity: 36.56%

--- Spider Graph Summary ---





--- 3D Scatter Plot Summary ---



--- Explanation of Results ---

English Explanation:

Image Loading and Resizing:

- Images from '/content/drive/My Drive/LoRA' were loaded and resized to 256x256 pixels to ensure consistent dimensions for metric calculations.

Metrics Calculated:
- **PSNR (Peak Signal-to-Noise Ratio):** Quantifies the difference between two images by measuring the ratio of maximum possible power of a signal to the power of corrupting noise that affects the fidelity of its representation. A *higher* PSNR value indicates a higher quality image relative to a reference image, and thus greater similarity.
- **SSIM (Structural Similarity Index Measure):** Evaluates the similarity between two images based on three key factors: luminance, contrast, and structure. SSIM is designed to be a better measure of perceived similarity than PSNR. A value closer to 1 (or 100%) indicates higher structural similarity.

Row and Column Metrics Summary:
- The tables for 'Row Metrics' and 'Column Metrics' show the average values of PSNR, and SSIM for each row and column, respectively. For rows/columns with more than one image, these metrics are calculated by comparing each image in that row/column to the first image in that row/column, and then averaging the results.
- PSNR and SSIM values are presented, with SSIM converted to percentage for easier interpretation (higher percentage is better).
- The 'Row Similarity/Difference (%)' and 'Column Similarity/Difference (%)' tables provide a summary where PSNR and SSIM are shown as percentages of similarity.

Spider Graph Summary:
- Two spider graphs are presented: one for the average row similarity metrics and one for the average column similarity metrics. These graphs visually summarize the average PSNR and SSIM similarity scores, allowing for a quick comparison of overall similarity levels between rows and columns.

3D Scatter Plot Summary:
- A 3D scatter plot visualizes the PSNR and SSIM similarity scores for each individual row and column. Each point represents a row or column, with its position determined by its PSNR and SSIM similarity percentages. The third dimension (z-axis) separates the points by group type (Row or Column). This plot allows you to see the distribution and potential clustering of similarity scores for each group.

Overall Interpretation:
- By examining these metrics and visualizations, you can gain insights into the visual similarity and differences between the images in your dataset, both within rows and within columns. Higher PSNR, and higher SSIM values suggest greater similarity.

```
import os
import numpy as np
import cv2
from skimage.metrics import peak_signal_noise_ratio, structural_similarity
from scipy.spatial import distance
from google.colab import drive
import plotly.express as px
import pandas as pd
from PIL import Image

# Mount Google Drive
drive.mount('/content/drive')

# Define the directory
image_dir = '/content/drive/My Drive/LoRA'

# Define target size for resizing
target_size = (256, 256)
```

```python
# Function to resize images
def resize_image(img_path, target_size):
    img = cv2.imread(img_path)
    if img is None:
        return None
    return cv2.resize(img, target_size)

# Function to calculate Color Histogram Cosine Distance
def color_hist_distance(img1, img2):
    hist1 = cv2.calcHist([img1], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist2 = cv2.calcHist([img2], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist1 = cv2.normalize(hist1, hist1).flatten()
    hist2 = cv2.normalize(hist2, hist2).flatten()
    return distance.cosine(hist1, hist2)

# Function to calculate Edge Feature Cosine Distance (using Canny edge detection)
def edge_feature_distance(img1, img2):
    edges1 = cv2.Canny(cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY), 100, 200)
    edges2 = cv2.Canny(cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY), 100, 200)
    edges1 = edges1.flatten()
    edges2 = edges2.flatten()
    # Pad with zeros if lengths are different (can happen with Canny)
    max_len = max(len(edges1), len(edges2))
    edges1 = np.pad(edges1, (0, max_len - len(edges1)), 'constant')
    edges2 = np.pad(edges2, (0, max_len - len(edges2)), 'constant')
    return distance.cosine(edges1, edges2)

# Load and resize images
images = {}
for r in range(1, 6):
    for c in range(1, 6):
        img_name = f'r{r}c{c}.jpg'
        img_path = os.path.join(image_dir, img_name)
        resized_img = resize_image(img_path, target_size)
        if resized_img is not None:
            images[(r, c)] = resized_img
        else:
            print(f"Warning: Could not load or resize {img_name}")

# Calculate metrics for each row and column
row_metrics = {}
col_metrics = {}

# Calculate metrics for rows
for r in range(1, 6):
    row_images = [images[(r, c)] for c in range(1, 6) if (r, c) in images]
    if len(row_images) > 1:
        psnr_list = []
        ssim_list = []
        mse_list = []
```

```
        color_hist_list = []
        edge_feature_list = []
        for i in range(len(row_images)):
            for j in range(i + 1, len(row_images)):
                img1 = row_images[i]
                img2 = row_images[j]
                psnr_list.append(peak_signal_noise_ratio(img1, img2))
                ssim_list.append(structural_similarity(img1, img2, channel_axis=2))
                mse_list.append(np.mean((img1 - img2)**2))
                color_hist_list.append(color_hist_distance(img1, img2))
                edge_feature_list.append(edge_feature_distance(img1, img2))

        # Aggregate metrics for the row (e.g., average)
        row_metrics[f'Row {r}'] = {
            'PSNR': np.mean(psnr_list) if psnr_list else np.nan,
            'SSIM': np.mean(ssim_list) if ssim_list else np.nan,
            'MSE': np.mean(mse_list) if mse_list else np.nan,
            'Color_Hist_Dist': np.mean(color_hist_list) if color_hist_list else np.nan,
            'Edge_Feature_Dist': np.mean(edge_feature_list) if edge_feature_list else np.nan
        }

# Calculate metrics for columns
for c in range(1, 6):
    col_images = [images[(r, c)] for r in range(1, 6) if (r, c) in images]
    if len(col_images) > 1:
        psnr_list = []
        ssim_list = []
        mse_list = []
        color_hist_list = []
        edge_feature_list = []
        for i in range(len(col_images)):
            for j in range(i + 1, len(col_images)):
                img1 = col_images[i]
                img2 = col_images[j]
                psnr_list.append(peak_signal_noise_ratio(img1, img2))
                ssim_list.append(structural_similarity(img1, img2, channel_axis=2))
                mse_list.append(np.mean((img1 - img2)**2))
                color_hist_list.append(color_hist_distance(img1, img2))
                edge_feature_list.append(edge_feature_distance(img1, img2))

        # Aggregate metrics for the column (e.g., average)
        col_metrics[f'Col {c}'] = {
            'PSNR': np.mean(psnr_list) if psnr_list else np.nan,
            'SSIM': np.mean(ssim_list) if ssim_list else np.nan,
            'MSE': np.mean(mse_list) if mse_list else np.nan,
            'Color_Hist_Dist': np.mean(color_hist_list) if color_hist_list else np.nan,
            'Edge_Feature_Dist': np.mean(edge_feature_list) if edge_feature_list else np.nan
        }
```

```
# Combine row and column metrics
all_metrics = {**row_metrics, **col_metrics}

# Normalize metrics to be between 0 and 1
# Note: PSNR and SSIM are typically higher for more similarity, MSE and distances are lower.
# We'll normalize similarity metrics (PSNR, SSIM) directly, and inverse distance metrics (MSE, distances) for consistency in spider plot.
# A higher value on the spider plot means more *similarity*.

normalized_metrics = {}
metrics_df = pd.DataFrame.from_dict(all_metrics, orient='index')

# Normalize metrics to be between 0 and 1 (higher = more similar)
# For PSNR and SSIM, directly normalize (min-max scaling)
metrics_df['PSNR_norm'] = (metrics_df['PSNR'] - metrics_df['PSNR'].min()) / (metrics_df['PSNR'].max() - metrics_df['PSNR'].min())
metrics_df['SSIM_norm'] = metrics_df['SSIM'] # SSIM is already 0-1

# For MSE, Color_Hist_Dist, Edge_Feature_Dist, inverse normalize (higher distance/error means less similar)
# We'll use 1 - normalized distance/error, so a higher value indicates more similarity
metrics_df['MSE_norm'] = 1 - (metrics_df['MSE'] - metrics_df['MSE'].min()) / (metrics_df['MSE'].max() - metrics_df['MSE'].min())
metrics_df['Color_Hist_Sim'] = 1 - (metrics_df['Color_Hist_Dist'] - metrics_df['Color_Hist_Dist'].min()) / (metrics_df['Color_Hist_Dist'].max() - metrics_df['Color_Hist_Dist'].min())
metrics_df['Edge_Feature_Sim'] = 1 - (metrics_df['Edge_Feature_Dist'] - metrics_df['Edge_Feature_Dist'].min()) / (metrics_df['Edge_Feature_Dist'].max() -
metrics_df['Edge_Feature_Dist'].min())

# Select normalized similarity columns and rename for spider plot
spider_df = metrics_df[['PSNR_norm', 'SSIM_norm', 'MSE_norm', 'Color_Hist_Sim', 'Edge_Feature_Sim']].copy()
spider_df.columns = ['PSNR Similarity', 'SSIM Similarity', 'MSE Similarity', 'Color Histogram Similarity', 'Edge Feature Similarity']
spider_df = spider_df.reset_index().rename(columns={'index': 'Group'})

# Convert to long format for Plotly Express spider plot
spider_long_df = spider_df.melt(id_vars='Group', var_name='Metric', value_name='Similarity')

# Create Spider Graph
fig_spider = px.line_polar(spider_long_df, r="Similarity", theta="Metric", color="Group", line_close=True,
                           title="Image Similarity Metrics (Normalized)")
fig_spider.update_traces(fill='toself')
fig_spider.show()

# Summarize Spider Graph meaning in a table
spider_summary = {
    "Metric": ['PSNR Similarity', 'SSIM Similarity', 'MSE Similarity', 'Color Histogram Similarity', 'Edge Feature Similarity'],
    "Meaning (Higher Value)": [
        "Higher Peak Signal-to-Noise Ratio (better image quality, less noise/loss from compression/processing)",
        "Higher Structural Similarity Index (more similar perceived structure/texture)",
        "Lower Mean Squared Error (less average pixel difference, higher similarity)", # Note: MSE_norm is 1-MSE, so higher means lower MSE
        "Lower Color Histogram Cosine Distance (more similar color distribution, higher similarity)", # Note: Color_Hist_Sim is 1-dist, so higher means lower distance
        "Lower Edge Feature Cosine Distance (more similar edge patterns, higher similarity)" # Note: Edge_Feature_Sim is 1-dist, so higher means lower distance
    ],
    "Interpretation": [
        "Measures pixel-level difference, sensitive to noise.",
        "Measures structural similarity based on luminance, contrast, and structure.",
```

```
        "Measures average squared difference between pixels.",
        "Compares the overall distribution of colors.",
        "Compares the patterns of edges in the images."
    ]
}
spider_summary_df = pd.DataFrame(spider_summary)
print("\n--- Spider Graph Metric Summary ---")
from IPython.display import display
display(spider_summary_df)


# Create 3D Scatter Plot
# We can choose three relevant metrics to plot in 3D
scatter_df = metrics_df.reset_index().rename(columns={'index': 'Group'})
fig_scatter_3d = px.scatter_3d(scatter_df,
                               x='PSNR_norm',
                               y='SSIM_norm',
                               z='Color_Hist_Sim',
                               color='Group',
                               hover_name='Group',
                               title='3D Scatter Plot of Image Similarity Metrics (Normalized)')

fig_scatter_3d.update_layout(scene = dict(
    xaxis_title='PSNR Similarity',
    yaxis_title='SSIM Similarity',
    zaxis_title='Color Histogram Similarity'))

fig_scatter_3d.show()

# Explanation of the 3D Scatter Plot
print("\n--- 3D Scatter Plot Explanation ---")
print("This 3D scatter plot visualizes the relationships between three different image similarity metrics (PSNR Similarity, SSIM Similarity, and Color Histogram Similarity) for each row and column of images.")
print("- Each point represents a specific row or column (e.g., 'Row 1', 'Col 5').")
print("- The position of the point along the X-axis indicates its PSNR Similarity score (normalized 0-1).")
print("- The position along the Y-axis indicates its SSIM Similarity score (normalized 0-1).")
print("- The position along the Z-axis indicates its Color Histogram Similarity score (normalized 0-1).")
print("- Points that are clustered together in the 3D space represent rows or columns where the images within them are similarly related across these three metrics.")
print("- The color of each point distinguishes the specific row or column.")
print("- You can interact with the plot by rotating it, zooming in/out, and hovering over points to see their exact values and associated row/column.")
print("- This plot helps to identify groups of rows or columns that exhibit similar patterns of variation across the chosen metrics. For example, if all column points cluster separately from all row points, it might suggest that the images within columns are more similar to each other than images within rows, or vice versa, based on these metrics.")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/usr/local/lib/python3.11/dist-packages/scipy/spatial/distance.py:682: RuntimeWarning:

overflow encountered in scalar multiply
```

Image Similarity Metrics (Normalized)

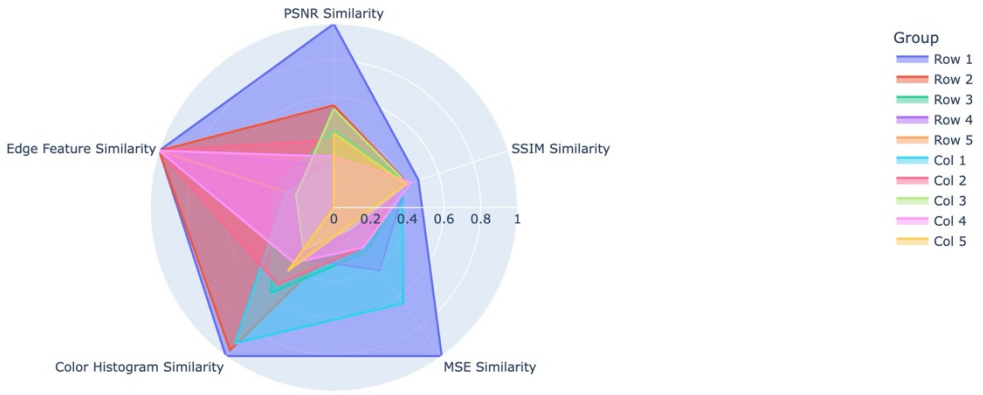


Image Similarity Metrics (Normalized)

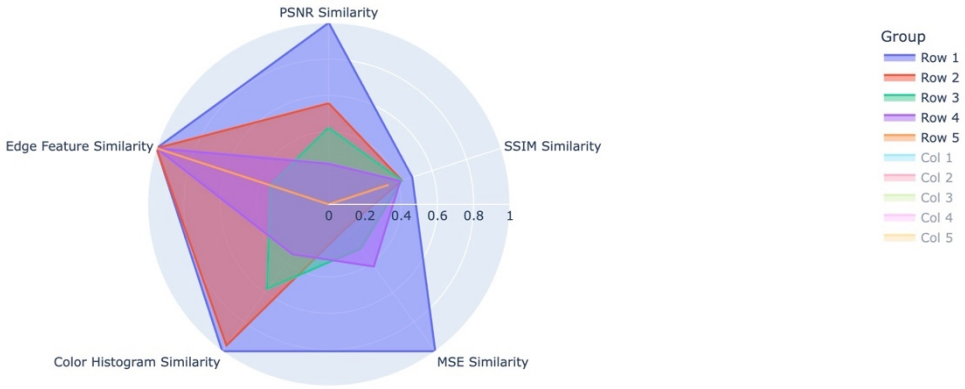
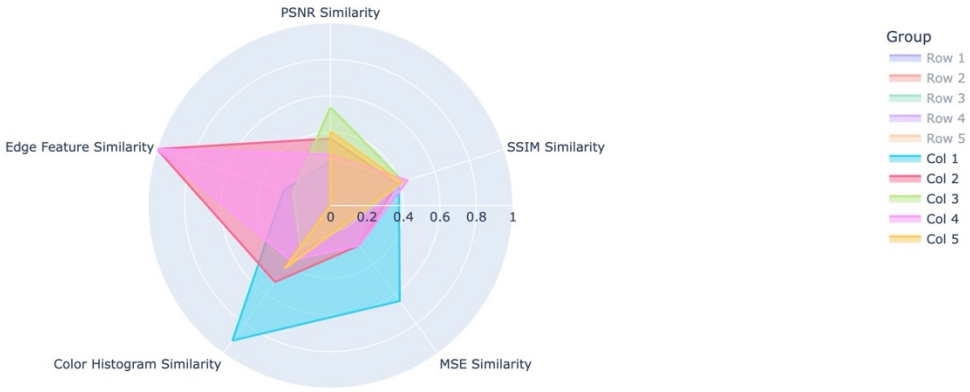
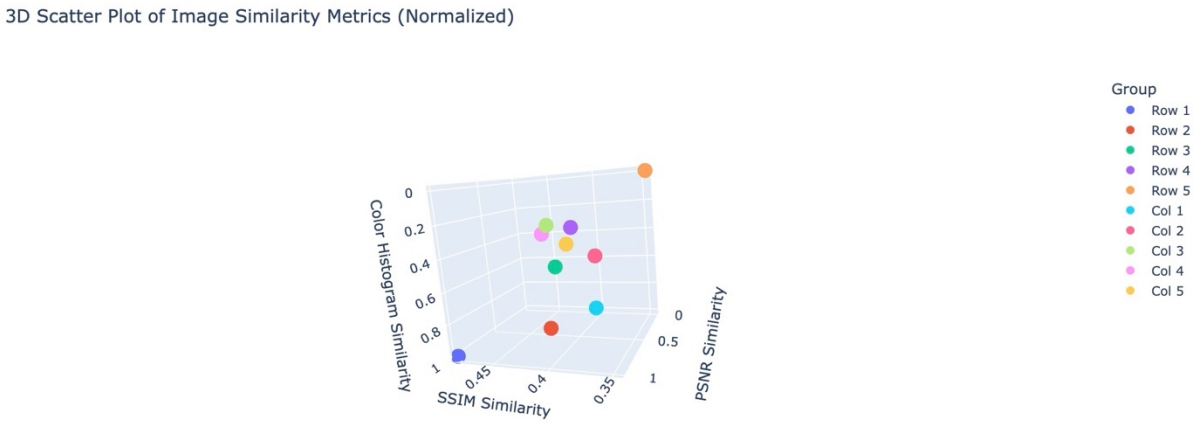


Image Similarity Metrics (Normalized)



--- Spider Graph Metric Summary ---

	Metric	Meaning (Higher Value)	Interpretation
0	PSNR Similarity	Higher Peak Signal-to-Noise Ratio (better imag...	Measures pixel-level difference, sensitive to ...
1	SSIM Similarity	Higher Structural Similarity Index (more simil...	Measures structural similarity based on lumina...
2	MSE Similarity	Lower Mean Squared Error (less average pixel d...	Measures average squared difference between pi...
3	Color Histogram Similarity	Lower Color Histogram Cosine Distance (more si...	Compares the overall distribution of colors.
4	Edge Feature Similarity	Lower Edge Feature Cosine Distance (more simil...	Compares the patterns of edges in the images.



--- 3D Scatter Plot Explanation ---

This 3D scatter plot visualizes the relationships between three different image similarity metrics (PSNR Similarity, SSIM Similarity, and Color Histogram Similarity) for each row and column of images.

- Each point represents a specific row or column (e.g., 'Row 1', 'Col 5').
- The position of the point along the X-axis indicates its PSNR Similarity score (normalized 0-1).
- The position along the Y-axis indicates its SSIM Similarity score (normalized 0-1).
- The position along the Z-axis indicates its Color Histogram Similarity score (normalized 0-1).
- Points that are clustered together in the 3D space represent rows or columns where the images within them are similarly related across these three metrics.
- The color of each point distinguishes the specific row or column.
- You can interact with the plot by rotating it, zooming in/out, and hovering over points to see their exact values and associated row/column.
- This plot helps to identify groups of rows or columns that exhibit similar patterns of variation across the chosen metrics. For example, if all column points cluster separately from all row points, it might suggest that the images within columns are more similar to each other than images within rows, or vice versa, based on these metrics.

```
import os
import numpy as np
import cv2
from skimage.metrics import peak_signal_noise_ratio, structural_similarity
from scipy.spatial import distance
from google.colab import drive
import plotly.express as px
import pandas as pd
from PIL import Image
```

```python
# Mount Google Drive
drive.mount('/content/drive')

# Define the directory
image_dir = '/content/drive/My Drive/LoRA-afterAdj'

# Define target size for resizing
target_size = (256, 256)

# Function to resize images
def resize_image(img_path, target_size):
    img = cv2.imread(img_path)
    if img is None:
        return None
    return cv2.resize(img, target_size)

# Function to calculate Color Histogram Cosine Distance
def color_hist_distance(img1, img2):
    hist1 = cv2.calcHist([img1], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist2 = cv2.calcHist([img2], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])
    hist1 = cv2.normalize(hist1, hist1).flatten()
    hist2 = cv2.normalize(hist2, hist2).flatten()
    return distance.cosine(hist1, hist2)

# Function to calculate Edge Feature Cosine Distance (using Canny edge detection)
def edge_feature_distance(img1, img2):
    edges1 = cv2.Canny(cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY), 100, 200)
    edges2 = cv2.Canny(cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY), 100, 200)
    edges1 = edges1.flatten()
    edges2 = edges2.flatten()
    # Pad with zeros if lengths are different (can happen with Canny)
    max_len = max(len(edges1), len(edges2))
    edges1 = np.pad(edges1, (0, max_len - len(edges1)), 'constant')
    edges2 = np.pad(edges2, (0, max_len - len(edges2)), 'constant')
    return distance.cosine(edges1, edges2)

# Load and resize images
images = {}
for r in range(1, 6):
    for c in range(1, 6):
        img_name = f'r{r}c{c}.jpg'
        img_path = os.path.join(image_dir, img_name)
        resized_img = resize_image(img_path, target_size)
        if resized_img is not None:
            images[(r, c)] = resized_img
        else:
            print(f"Warning: Could not load or resize {img_name}")
```

```
# Calculate metrics for each row and column
row_metrics = {}
col_metrics = {}

# Calculate metrics for rows
for r in range(1, 6):
    row_images = [images[(r, c)] for c in range(1, 6) if (r, c) in images]
    if len(row_images) > 1:
        psnr_list = []
        ssim_list = []
        mse_list = []
        color_hist_list = []
        edge_feature_list = []
        for i in range(len(row_images)):
            for j in range(i + 1, len(row_images)):
                img1 = row_images[i]
                img2 = row_images[j]
                psnr_list.append(peak_signal_noise_ratio(img1, img2))
                ssim_list.append(structural_similarity(img1, img2, channel_axis=2))
                mse_list.append(np.mean((img1 - img2)**2))
                color_hist_list.append(color_hist_distance(img1, img2))
                edge_feature_list.append(edge_feature_distance(img1, img2))

        # Aggregate metrics for the row (e.g., average)
        row_metrics[f'Row {r}'] = {
            'PSNR': np.mean(psnr_list) if psnr_list else np.nan,
            'SSIM': np.mean(ssim_list) if ssim_list else np.nan,
            'MSE': np.mean(mse_list) if mse_list else np.nan,
            'Color_Hist_Dist': np.mean(color_hist_list) if color_hist_list else np.nan,
            'Edge_Feature_Dist': np.mean(edge_feature_list) if edge_feature_list else np.nan
        }

# Calculate metrics for columns
for c in range(1, 6):
    col_images = [images[(r, c)] for r in range(1, 6) if (r, c) in images]
    if len(col_images) > 1:
        psnr_list = []
        ssim_list = []
        mse_list = []
        color_hist_list = []
        edge_feature_list = []
        for i in range(len(col_images)):
            for j in range(i + 1, len(col_images)):
                img1 = col_images[i]
                img2 = col_images[j]
                psnr_list.append(peak_signal_noise_ratio(img1, img2))
                ssim_list.append(structural_similarity(img1, img2, channel_axis=2))
                mse_list.append(np.mean((img1 - img2)**2))
                color_hist_list.append(color_hist_distance(img1, img2))
                edge_feature_list.append(edge_feature_distance(img1, img2))
```

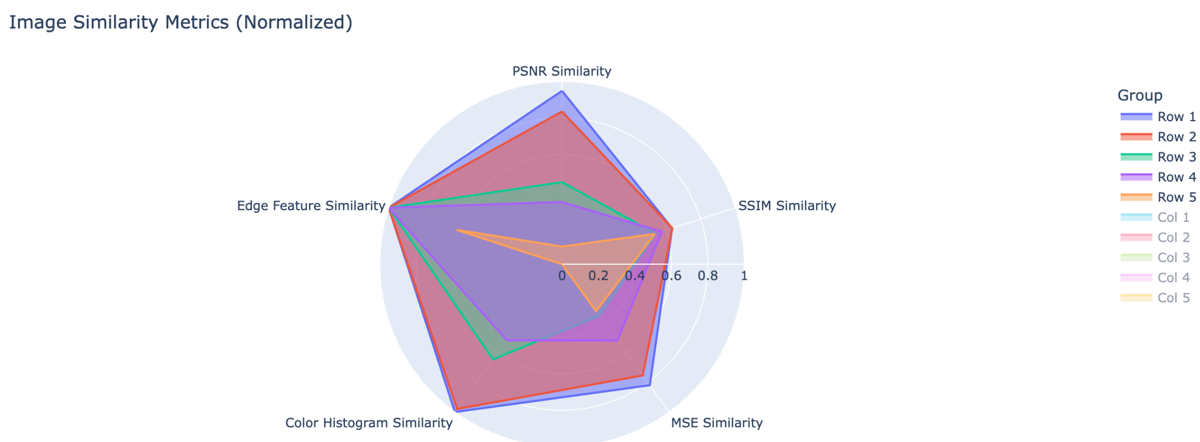
```
        # Aggregate metrics for the column (e.g., average)
        col_metrics[f'Col {c}'] = {
            'PSNR': np.mean(psnr_list) if psnr_list else np.nan,
            'SSIM': np.mean(ssim_list) if ssim_list else np.nan,
            'MSE': np.mean(mse_list) if mse_list else np.nan,
            'Color_Hist_Dist': np.mean(color_hist_list) if color_hist_list else np.nan,
            'Edge_Feature_Dist': np.mean(edge_feature_list) if edge_feature_list else np.nan
        }

# Combine row and column metrics
all_metrics = {**row_metrics, **col_metrics}

# Normalize metrics to be between 0 and 1
# Note: PSNR and SSIM are typically higher for more similarity, MSE and distances are lower.
# We'll normalize similarity metrics (PSNR, SSIM) directly, and inverse distance metrics (MSE, distances) for consistency in spider plot.
# A higher value on the spider plot means more *similarity*.

normalized_metrics = {}
metrics_df = pd.DataFrame.from_dict(all_metrics, orient='index')

# Normalize metrics to be between 0 and 1 (higher = more similar)
# For PSNR and SSIM, directly normalize (min-max scaling)
metrics_df['PSNR_norm'] = (metrics_df['PSNR'] - metrics_df['PSNR'].min()) / (metrics_df['PSNR'].max() - metrics_df['PSNR'].min())
metrics_df['SSIM_norm'] = metrics_df['SSIM'] # SSIM is already 0-1

# For MSE, Color_Hist_Dist, Edge_Feature_Dist, inverse normalize (higher distance/error means less similar)
# We'll use 1 - normalized distance/error, so a higher value indicates more similarity
metrics_df['MSE_norm'] = 1 - (metrics_df['MSE'] - metrics_df['MSE'].min()) / (metrics_df['MSE'].max() - metrics_df['MSE'].min())
metrics_df['Color_Hist_Sim'] = 1 - (metrics_df['Color_Hist_Dist'] - metrics_df['Color_Hist_Dist'].min()) / (metrics_df['Color_Hist_Dist'].max() - metrics_df['Color_Hist_Dist'].min())
metrics_df['Edge_Feature_Sim'] = 1 - (metrics_df['Edge_Feature_Dist'] - metrics_df['Edge_Feature_Dist'].min()) / (metrics_df['Edge_Feature_Dist'].max() -
metrics_df['Edge_Feature_Dist'].min())

# Select normalized similarity columns and rename for spider plot
spider_df = metrics_df[['PSNR_norm', 'SSIM_norm', 'MSE_norm', 'Color_Hist_Sim', 'Edge_Feature_Sim']].copy()
spider_df.columns = ['PSNR Similarity', 'SSIM Similarity', 'MSE Similarity', 'Color Histogram Similarity', 'Edge Feature Similarity']
spider_df = spider_df.reset_index().rename(columns={'index': 'Group'})

# Convert to long format for Plotly Express spider plot
spider_long_df = spider_df.melt(id_vars='Group', var_name='Metric', value_name='Similarity')

# Create Spider Graph
fig_spider = px.line_polar(spider_long_df, r="Similarity", theta="Metric", color="Group", line_close=True,
                           title="Image Similarity Metrics (Normalized)")
fig_spider.update_traces(fill='toself')
fig_spider.show()
```

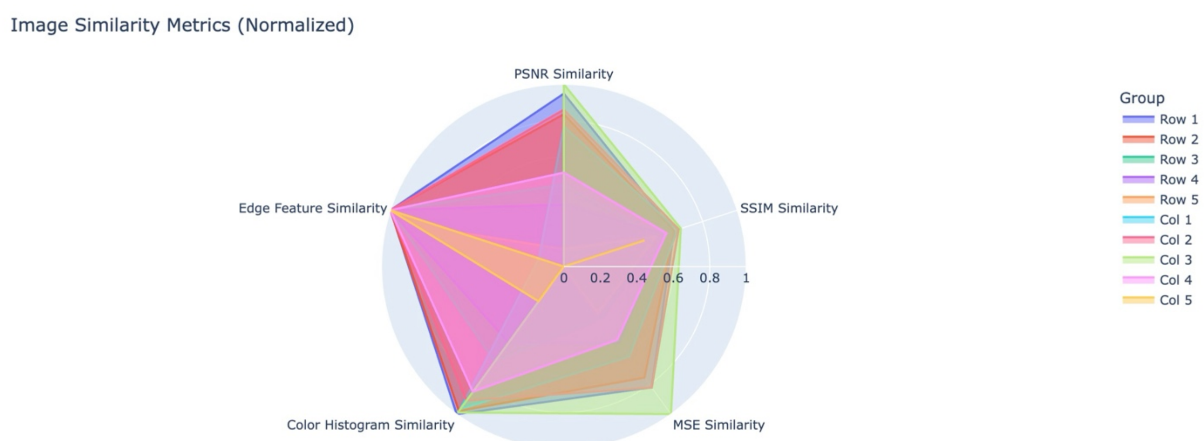
```
# Summarize Spider Graph meaning in a table
spider_summary = {
    "Metric": ['PSNR Similarity', 'SSIM Similarity', 'MSE Similarity', 'Color Histogram Similarity', 'Edge Feature Similarity'],
    "Meaning (Higher Value)": [
        "Higher Peak Signal-to-Noise Ratio (better image quality, less noise/loss from compression/processing)",
        "Higher Structural Similarity Index (more similar perceived structure/texture)",
        "Lower Mean Squared Error (less average pixel difference, higher similarity)", # Note: MSE_norm is 1-MSE, so higher means lower MSE
        "Lower Color Histogram Cosine Distance (more similar color distribution, higher similarity)", # Note: Color_Hist_Sim is 1-dist, so higher means lower distance
        "Lower Edge Feature Cosine Distance (more similar edge patterns, higher similarity)" # Note: Edge_Feature_Sim is 1-dist, so higher means lower distance
    ],
    "Interpretation": [
        "Measures pixel-level difference, sensitive to noise.",
        "Measures structural similarity based on luminance, contrast, and structure.",
        "Measures average squared difference between pixels.",
        "Compares the overall distribution of colors.",
        "Compares the patterns of edges in the images."
    ]
}
spider_summary_df = pd.DataFrame(spider_summary)
print("\n--- Spider Graph Metric Summary ---")
from IPython.display import display
display(spider_summary_df)


# Create 3D Scatter Plot
# We can choose three relevant metrics to plot in 3D
scatter_df = metrics_df.reset_index().rename(columns={'index': 'Group'})
fig_scatter_3d = px.scatter_3d(scatter_df,
                               x='PSNR_norm',
                               y='SSIM_norm',
                               z='Color_Hist_Sim',
                               color='Group',
                               hover_name='Group',
                               title='3D Scatter Plot of Image Similarity Metrics (Normalized)')

fig_scatter_3d.update_layout(scene = dict(
    xaxis_title='PSNR Similarity',
    yaxis_title='SSIM Similarity',
    zaxis_title='Color Histogram Similarity'))

fig_scatter_3d.show()

# Explanation of the 3D Scatter Plot
print("\n--- 3D Scatter Plot Explanation ---")
print("This 3D scatter plot visualizes the relationships between three different image similarity metrics (PSNR Similarity, SSIM Similarity, and Color Histogram Similarity) for each row and column of images.")
print("- Each point represents a specific row or column (e.g., 'Row 1', 'Col 5').")
print("- The position of the point along the X-axis indicates its PSNR Similarity score (normalized 0-1).")
print("- The position along the Y-axis indicates its SSIM Similarity score (normalized 0-1).")
print("- The position along the Z-axis indicates its Color Histogram Similarity score (normalized 0-1).")
```

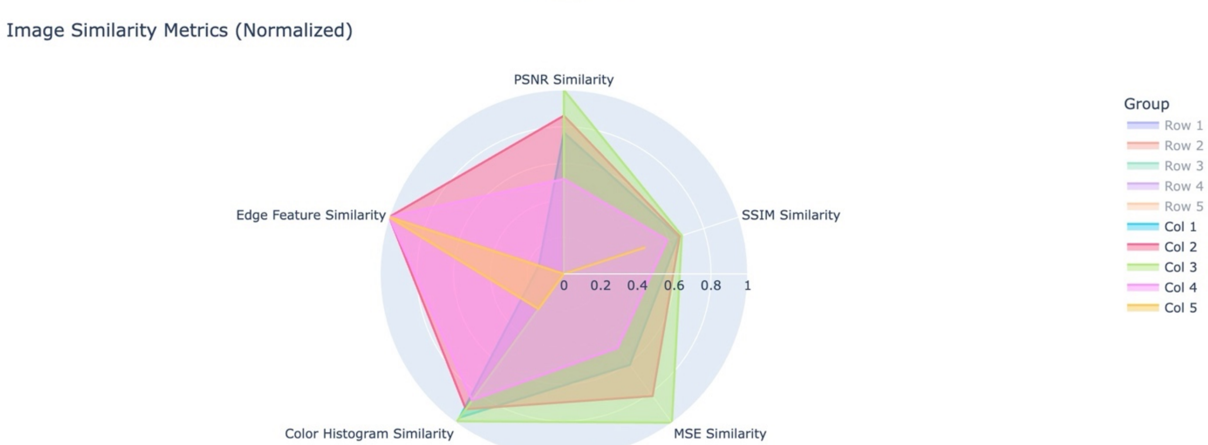
```
print("- Points that are clustered together in the 3D space represent rows or columns where the images within them are similarly related across these three metrics.")
print("- The color of each point distinguishes the specific row or column.")
print("- You can interact with the plot by rotating it, zooming in/out, and hovering over points to see their exact values and associated row/column.")
print("- This plot helps to identify groups of rows or columns that exhibit similar patterns of variation across the chosen metrics. For example, if all column points cluster separately from all row points, it might suggest that the images within columns are more similar to each other than images within rows, or vice versa, based on these metrics.")
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/usr/local/lib/python3.11/dist-packages/scipy/spatial/distance.py:682: RuntimeWarning:

overflow encountered in scalar multiply

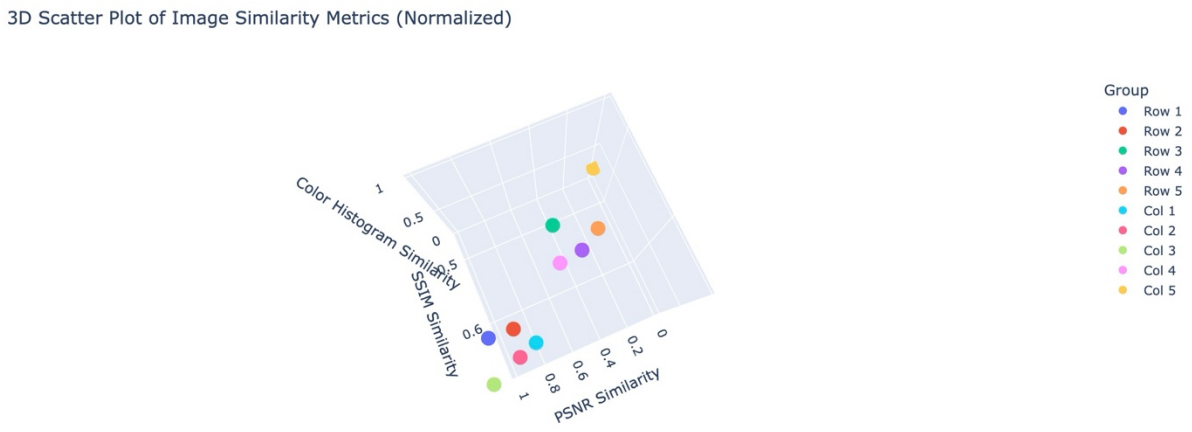






--- Spider Graph Metric Summary ---

	Metric	Meaning (Higher Value)	Interpretation
0	PSNR Similarity	Higher Peak Signal-to-Noise Ratio (better imag...	Measures pixel-level difference, sensitive to ...
1	SSIM Similarity	Higher Structural Similarity Index (more simil...	Measures structural similarity based on lumina...
2	MSE Similarity	Lower Mean Squared Error (less average pixel d...	Measures average squared difference between pi...
3	Color Histogram Similarity	Lower Color Histogram Cosine Distance (more si...	Compares the overall distribution of colors.
4	Edge Feature Similarity	Lower Edge Feature Cosine Distance (more simil...	Compares the patterns of edges in the images.



--- 3D Scatter Plot Explanation ---

This 3D scatter plot visualizes the relationships between three different image similarity metrics (PSNR Similarity, SSIM Similarity, and Color Histogram Similarity) for each row and column of images.

- Each point represents a specific row or column (e.g., 'Row 1', 'Col 5').
- The position of the point along the X-axis indicates its PSNR Similarity score (normalized 0-1).
- The position along the Y-axis indicates its SSIM Similarity score (normalized 0-1).
- The position along the Z-axis indicates its Color Histogram Similarity score (normalized 0-1).
- Points that are clustered together in the 3D space represent rows or columns where the images within them are similarly related across these three metrics.
- The color of each point distinguishes the specific row or column.
- You can interact with the plot by rotating it, zooming in/out, and hovering over points to see their exact values and associated row/column.
- This plot helps to identify groups of rows or columns that exhibit similar patterns of variation across the chosen metrics. For example, if all column points cluster separately from all row points, it might suggest that the images within columns are more similar to each other than images within rows, or vice versa, based on these metrics.