Lab3 Photo Mosaic KNN

Preparation

```python
### START CODE HERE ###
from google.colab import drive

# Mount the drive (adjust the path as necessary)
drive.mount('/content/drive')

# Directory containing your tile images (update this path to your image directory)
image_dir = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic'
fnames = [f for f in os.listdir(image_dir) if os.path.isfile(os.path.join(image_dir, f))]

numImg = len(fnames)
if numImg >= 100:
    fnames_mini = fnames[:100]
else:
    raise ValueError("Less than 100 images")
print(fnames_mini)
### END CODE HERE ###
```

Python

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive",
['☆.jpg', 'blkob_PP.jpg', '8gWJqzxy.jpg', 'ww8E7C0G.jpg', '2T_ccEbb.jpg', 'xJuadFWl.jpg', '02GoHCEF.jpg', 'y
```

- กำหนด dataset directory แล้วใช้ function os.listdir() ในการ list ไฟล์ทั้งหมดใน directory นั้น เพื่อเอาชื่อไฟล์ของแต่ละรูปเก็บไปใน fnames
- ให้ fnames_mini เก็บ 100 element แรกของ fnames

```python
### START CODE HERE ###
from PIL import Image
tiles = []
tile_size = (50,50)
for i in fnames_mini:
  image_path = os.path.join(image_dir, i)
  with Image.open(image_path) as img:
        img_resized = img.resize(tile_size)
        tiles.append(img_resized)
### END CODE HERE ###
```

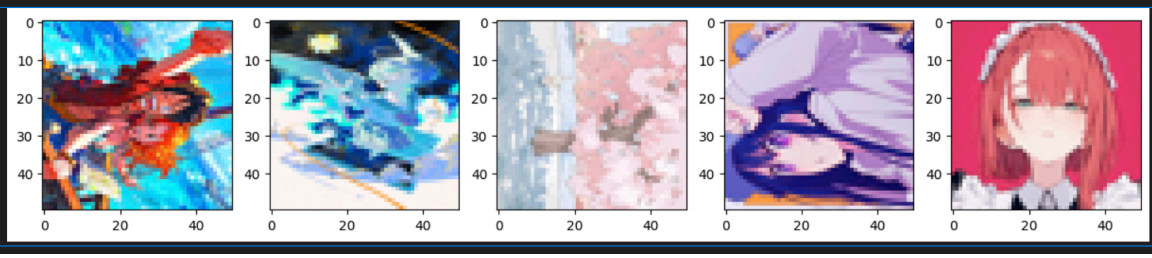
Python

```
Avatar.jpg
```

- กำหนด tile size เป็น 50 x 50 pixel แล้วนำรูปแต่ละรูปมา resize แล้วเก็บไปใน list tiles

```python
### START CODE HERE ###
number_pic = 5
if isinstance(tiles, list) and len(tiles) >= number_pic:
    random_tiles = random.sample(tiles, number_pic)

fig, ax = plt.subplots(1, number_pic, figsize=(15,15))

for i, tiles in enumerate(random_tiles):
   ax[i].imshow(tiles)
plt.show()

### END CODE HERE ###
```

Python



- สุ่มรูปมาจาก tiles จำนวน 5 รูป แล้วเอามา plot

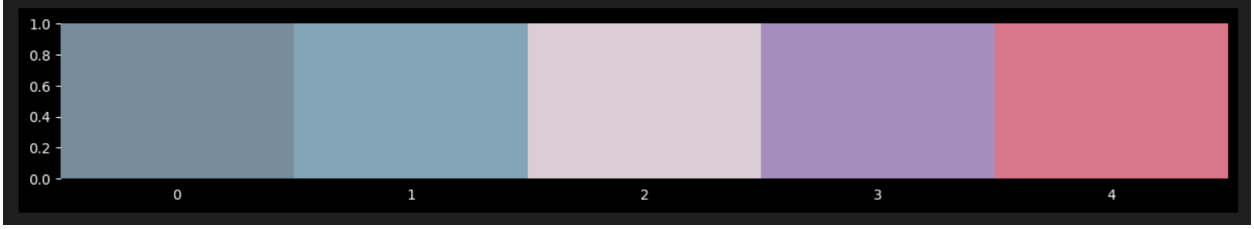
```python
### START CODE HERE ###
colors = []
for i, tiles in enumerate(random_tiles):
    tile_array = np.array(tiles)
    avg_color_per_tile = np.average(tile_array, axis=(0,1))
    colors.append(avg_color_per_tile)
print(f"Calculated colors for {len(colors)} tiles.")
print(type(colors[0]))
print(colors[1].shape)
print(colors[0])
print(colors[1])
print(colors[2])
print(colors[3])
print(colors[4])
avg = np.average(colors[0])
print(f'Test for avarage (color[0]) = {avg}')
### END CODE HERE ###
```

Python

```
Calculated colors for 5 tiles.
<class 'numpy.ndarray'>
(3,)
[117.4924 140.1412 154.3692]
[130.67   163.0888 182.2476]
[217.9556 204.8544 212.24  ]
[165.3724 140.7684 187.9316]
[216.5684 118.8212 138.1228]
Test for avarage (color[0]) = 137.33426666666665
```

- นำทั้ง 5. รูปมาหาสีเฉลี่ย โดยใช้ฟังก์ชัน np.average() แล้วเก็บใส่ list colors โดยผลลัพธ์จะออกมาเป็นค่าสีของแต่ละ channel RGB

```python
### START CODE HERE ###
from matplotlib.patches import Rectangle
meancolor_list =[ ]

fig, ax = plt.subplots(1, len(colors) ,figsize=(15, 2))
fig.patch.set_facecolor('black')
for i in range(len(colors)):
    redChannel = colors[i][0]
    BlueChannel = colors[i][1]
    greenChannel = colors[i][2]
    average_color = np.array([redChannel, BlueChannel, greenChannel]) / 255
    meancolor_list.append(average_color)
    # Print the average color
    print("The average color is:", average_color )
    print(redChannel)
    print(BlueChannel)
    print(greenChannel)
    rect = Rectangle((0, 0), 1, 1, color=average_color, edgecolor='white')
    ax[i].add_patch(rect)
    ax[i].set_xlim(0, 1)
    ax[i].set_ylim(0, 1)
   # set x in center
    ax[i].set_xticks([0.5])
    ax[i].set_xticklabels([str(i)], color='white')
   # Remove y-axis ticks
    if i == 0:
      ax[i].tick_params(axis='y', colors='white')  # Set color of y-axis ticks to white
    else:
      ax[i].set_yticks([])
    ax[i].spines['top'].set_color('none')
    ax[i].spines['right'].set_color('none')
    ax[i].spines['left'].set_color('none')
    ax[i].spines['bottom'].set_color('none')
plt.subplots_adjust(wspace=0, hspace=0)
plt.show()
### END CODE HERE ###
```

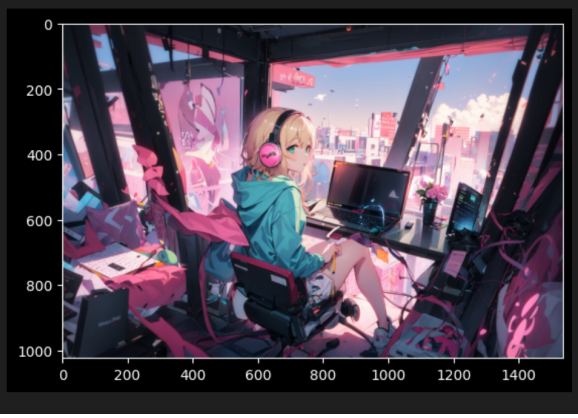


- นำสีเฉลี่ยจากข้อที่แล้วมา normalize เพื่อให้อยู่ระหว่าง 0 กับ 1 เนื่องจาก contructor ของ Retangle รับแค่สี 0 -1 จากนั้นใช้ฟังชั่น add_patch() เพื่อนำ Retangle หลายๆอันมาต่อกันอยู่กราฟเดียว และแสดงออกมาดังรูปผลลัพธ์

```python
### START CODE HERE ###
image_dir = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic/7f291d2207f694b75332e6ee36aa59da.png'
img_main = cv2.imread(image_dir)
og_img = cv2.cvtColor(img_main, cv2.COLOR_BGR2RGB)
fig, ax = plt.subplots()
fig.patch.set_facecolor('black')
ax.tick_params(axis='y', colors='white')
ax.tick_params(axis='x', colors='white')
ax.spines['top'].set_color('white')
ax.spines['right'].set_color('white')
ax.spines['left'].set_color('white')
ax.spines['bottom'].set_color('white')

main_img = og_img.copy()
plt.imshow(og_img)
### END CODE HERE ###
```

Python

```
<matplotlib.image.AxesImage at 0x7dc93c79c1c0>
```

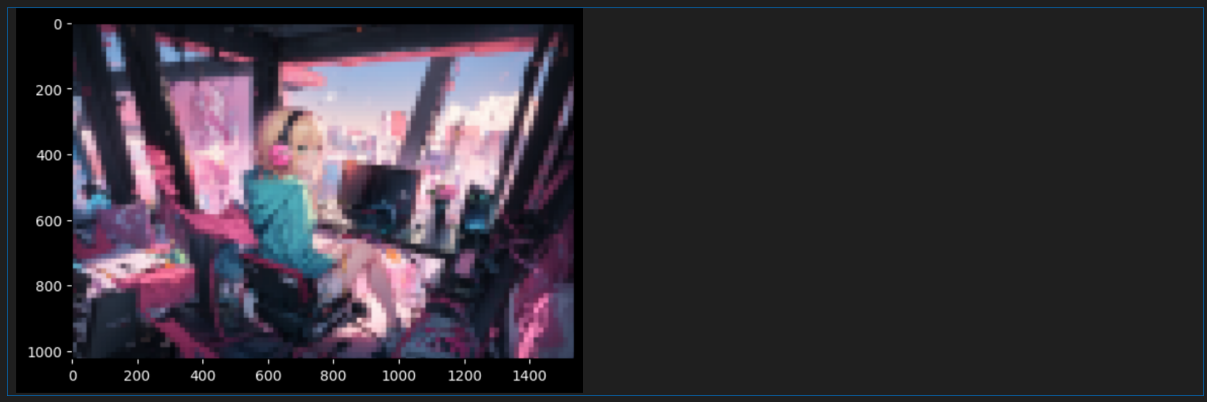


- เอารูปมาพล็อตเฉยๆครับ ไม่มีอะไร

```python
### START CODE HERE ###
tile_size = 15
resize_img = cv2.resize(og_img, dim, interpolation=cv2.INTER_AREA)
mosaic_img = np.zeros_like(resize_img)
for i in range(0, height, tile_size):
    for j in range(0, width, tile_size):
        tile = resize_img[i : i + tile_size, j : j + tile_size]
        avg_color = tile.mean(axis=(0, 1))
        mosaic_img[i : i + tile_size, j : j + tile_size] = avg_color
fig, ax = plt.subplots()
fig.patch.set_facecolor('black')
ax.tick_params(axis='y', colors='white')
ax.tick_params(axis='x', colors='white')
plt.imshow(mosaic_img)
print(mosaic_img.shape)
plt.show()
### END CODE HERE ###
```

Python

```
(1024, 1536, 3)
```



- เรานำรูปจากข้อก่อนหน้ามา resize โดยใส่ interpolation=cv2.INTER_AREA เพื่อป้องกันไม่รูปพังตอนใส่ tile
- สร้าง mosaic_img ให้มีขนาดเท่ากับรูปต้นฉบับแล้ว fill ด้วย 0 ทุกๆ channel ทุกๆ pixel
- Loop ที่ความกว้างและความสูงของรูปภาพ โดยมี step เป็น tile size (ข้ามไปทีละ tile) และทำการเฉลี่ยสีแต่ละกล่องย่อยๆ (ในที่นี่จะเป็นกล่องขนาด 15 x 15 pixel)
- นำรูปที่ได้มาทำการ plot

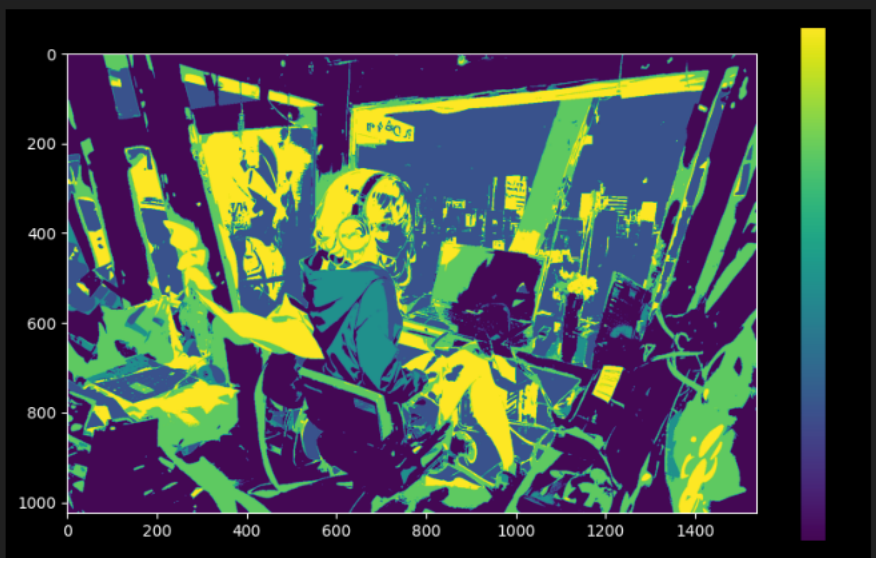
```python
from scipy.spatial import KDTree
### START CODE HERE ###
tile_colors = [color / 255 for color in colors]
tree = KDTree(tile_colors)
closest_tiles = np.zeros((height, width), dtype=int)
for i in range(height):
    for j in range(width):
        pixel_color = resize_img[i, j] / 255
        _, index = tree.query(pixel_color)
        closest_tiles[i, j] = index


### END CODE HERE ##
```

[88] Python

- ใช้ฟังก์ชัน query ของ KDTree เพื่อหา index ที่ใกล้ที่สุดของสีในแต่ละ pixel แล้วเก็บใส่ closest_tiles

```python
### START CODE HERE ###

fig, ax = plt.subplots(figsize=(10, 6))
fig.patch.set_facecolor('black')
ax.tick_params(axis='y', colors='white')
ax.tick_params(axis='x', colors='white')
ax.spines['top'].set_color('white')
ax.spines['right'].set_color('white')
ax.spines['left'].set_color('white')
ax.spines['bottom'].set_color('white')
plt.imshow(closest_tiles, cmap='viridis')
plt.colorbar()
plt.title('Closest Tiles')
plt.show()


### END CODE HERE ###
```

[105] Python



- พล็อตโดยใช้ cmap = viridis

```python
### START CODE HERE ###
tile_dir = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic'
tile_size = 10
tiles = []

for filename in os.listdir(tile_dir):
    if filename.endswith(('.png', '.jpg', '.jpeg')):
        tile_path = os.path.join(tile_dir, filename)
        tile_img = Image.open(tile_path)
        tile_img = tile_img.resize((tile_size, tile_size))
        tiles.append(np.array(tile_img))

tiles = np.array(tiles)
tile_colors = [tile.mean(axis=(0, 1)) for tile in tiles]


output = np.zeros((height, width, 3), dtype=np.uint8)
# Accessing size correctly from the first tile
tile_h, tile_w = tiles[0].shape[:2]

main_image_path = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic/7f291d2207f694b75332e6ee36aa59da.png'
main_img = Image.open(main_image_path)
main_img = main_img.resize((width, height))
main_img = np.array(main_img)

def closest_tile_index(color, tile_colors):
    distances = np.linalg.norm(tile_colors - color, axis=1)
    return np.argmin(distances)

for i in range(0, height, tile_h):
    for j in range(0, width, tile_w):
        region = main_img[i:i+tile_h, j:j+tile_w]
        region_color = region.mean(axis=(0, 1))
        tile_idx = closest_tile_index(region_color, tile_colors)
        tile = tiles[tile_idx]

        # size boundaries
        end_i = min(i + tile_h, height)
        end_j = min(j + tile_w, width)
        output[i:end_i, j:end_j] = tile[:end_i - i, :end_j - j]


### END CODE HERE ###
```

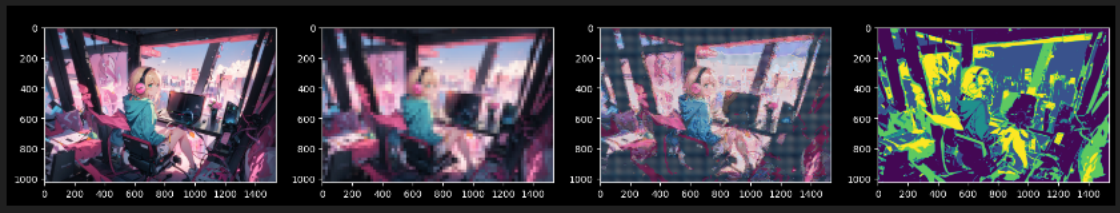
[135] Python

- List tiles directory และอ่านไฟล์ image และ นำ tile images นั้น ใส่ไปใน list ชื่อ tiles
- ลูปที่ละกล่องย่อย -> หาสีเฉลี่ย -> ไปหารูปภาพ tile ที่มีสีใกล้เคียงกับสีเฉลี่ยมากที่สุด

```python
### START CODE HERE ###
plt.figure(figsize=(10, 6))

fig, axs = plt.subplots(1, 4, figsize=(20, 10))
fig.patch.set_facecolor('black')
for ax in axs:
    ax.tick_params(axis='y', colors='white')
    ax.tick_params(axis='x', colors='white')
    ax.spines['top'].set_color('white')
    ax.spines['bottom'].set_color('white')
    ax.spines['left'].set_color('white')
    ax.spines['right'].set_color('white')
axs[0].imshow(main_img)
axs[0].set_title('Original Image')

axs[1].imshow(mosaic_img)
axs[1].set_title('Main image feature')

axs[2].imshow(output)
axs[2].set_title('Mosaic')

axs[3].imshow(closest_tiles, cmap='viridis')
axs[3].set_title('KDTree Return Tile Index ')
plt.show()


### END CODE HERE ###
```

[136] Python

```
<Figure size 1000x600 with 0 Axes>
```



- นำรูปจากข้อก่อนหน้ามา plot โดยมีรูป original , mosaic, รูปที่มี tile ข้างใน, รูป closest_tile

```python
from sklearn.neighbors import NearestNeighbors

def load_tiles(tile_dir, tile_size):
    tiles = []
    for filename in os.listdir(tile_dir):
        if filename.endswith(('.png', '.jpg', '.jpeg')):
            tile_path = os.path.join(tile_dir, filename)
            tile_img = Image.open(tile_path)
            tile_img = tile_img.resize((tile_size, tile_size))
            tiles.append(np.array(tile_img))
    tiles = np.array(tiles)
    tile_colors = [tile.mean(axis=(0, 1)) for tile in tiles]
    return tiles, tile_colors

def closest_tile_index(color, tile_colors):
    distances = np.linalg.norm(tile_colors - color, axis=1)
    return np.argmin(distances)

def create_mosaic(main_img, tiles, tile_colors, k, output_path):
    tile_h, tile_w = tiles[0].shape[:2]
    height, width = main_img.shape[:2]
    output = np.zeros((height, width, 3), dtype=np.uint8)

    # Create KDTree for tile matching
    neighbors = NearestNeighbors(n_neighbors=k, algorithm='auto').fit(tile_colors)

    for i in range(0, height, tile_h):
        for j in range(0, width, tile_w):
            region = main_img[i:i+tile_h, j:j+tile_w]
            region_color = region.mean(axis=(0, 1)).reshape(1, -1)
            distances, indices = neighbors.kneighbors(region_color)

            # average tile based on nearest neighbors
            selected_tiles = tiles[indices[0]]
            average_tile = np.mean(selected_tiles, axis=0).astype(np.uint8)

            # size boundaries
            end_i = min(i + tile_h, height)
            end_j = min(j + tile_w, width)
            output[i:end_i, j:end_j] = average_tile[:end_i - i, :end_j - j]

    # display result
    plt.xlabel(f'k = {k}')
    plt.xticks([])
    plt.yticks([])
    for spine in plt.gca().spines.values():
        spine.set_visible(False)
    plt.imshow(output)
    plt.show()

    output_image = Image.fromarray(output)
    output_image.save(output_path)
```

- นำสิ่งที่ทำไปจากข้อก่อนหน้ามาทำเป็น function ได้แก้ load_tiles() เพื่ออ่านไฟล์ tiles, closest_tile_index() เพื่อหา tile ที่มีสีใกล้กับ color ที่สุด, create_mosaic() เพื่อนำ tile ไป filled กับรูปต้นฉบับเหมือนโค้ดด้านบน แต่ในนี้จะใช้วิธีการ KNN โดยจะรับ parameter k เข้ามาด้วย เพื่อเป็นจำนวน neighbor ที่ต้องการเทรนใน KNN และใช้ฟังก์ชัน Kneighbors ในการ indicate region color ของแต่ละ pixel แทน

```python

if __name__ == "__main__":
    tile_dir = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic'
    main_image_path = '/content/drive/MyDrive/ImageColab/Lab3_Image-Clustering/playlist-image100pic/7f291d2207f694b75332e6ee36aa59da.
    output_dir = '/content'
    tile_size = 10
    k_values = [1, 10, 50]

    main_img = Image.open(main_image_path)
    width, height = main_img.size
    main_img = main_img.resize((width, height))
    main_img = np.array(main_img)

    # tile colors
    tiles, tile_colors = load_tiles(tile_dir, tile_size)

    for k in k_values:
        output_path = os.path.join(output_dir, f'mosaic_k{k}.png')
        create_mosaic(main_img, tiles, tile_colors, k, output_path)

### END CODE HERE ###
```

[147] Python

- เอาฟังก์ชันมาเรียกใช้ โดยปรับ k เป็น 1, 10, 50 ตามลำดับ โดยผลลัพธ์ดังรูปข้างล่าง