ณสิต ผลัญชัย 65010273

วิวัตร เตชะโกศล 65011001

Lab 3

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

numImg = 400
path = r'C:\Users\Nickv\Documents\ImageProcessing\Week3\asset\buildings'

fnames = []

for filename in os.listdir(path):
    file_path = os.path.join(path, filename)
    fnames.append(file_path)

fnames_mini = fnames[:numImg]

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

ทำการ Loop เพื่อเก็บ path ของภาพที่เตรียมไว้โดยนำมาเก็บใน List fnames แล้วเลือกมา 400 ภาพ เก็บไว้ใน List fnames_mini

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

tiles = []

tile_size = (16, 16)

for fname in fnames_mini:
    image = cv2.cvtColor(cv2.resize(cv2.imread(fname), tile_size), cv2.COLOR_BGR2RGB)
    tiles.append(image)

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

ทำการกำหนดขนาดของ tile แล้วโหลดภาพและปรับขนาดและเปลี่ยนสีจาก BGR เป็น RGB ก่อนนำมาเก็บไว้ใน List tiles

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

sample = 5
fig, axs = plt.subplots(1, sample, figsize=(15, 5))

random_img = random.sample(tiles, sample)

for ax, img in zip(axs, random_img):
    ax.imshow(img)
    ax.axis('off')

plt.show()

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

ทำการสุ่มตัวอย่างภาพ tile มา 5 ภาพและแสดงผล

```
### START CODE HERE ###
colors = []

for image in tiles:
    avg_color = np.mean(image, axis=(0, 1))
    colors.append(avg_color)

colors = np.array(colors)

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

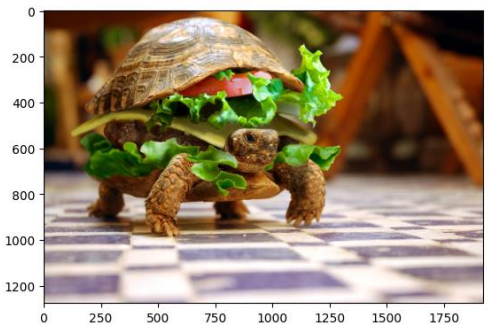
ทำการหาค่าเฉลี่ยสีของแต่ละภาพ tile ด้วย Function np.mean() แล้วเก็บค่าไว้ใน List colors

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

num_colors = 10
fig, axs = plt.subplots(1, num_colors, figsize=(20, 5), gridspec_kw={'wspace': 0, 'hspace': 0})

for i, color in enumerate(colors[:num_colors]):
    color_norm = color / 255
    axs[i].imshow([[color_norm]])
    axs[i].axis('off')

num_colors = 400
fig, axs = plt.subplots(1, num_colors, figsize=(40, 20), gridspec_kw={'wspace': 0, 'hspace': 0})

for i, color in enumerate(colors[:num_colors]):
    color_norm = color / 255
    axs[i].imshow([[color_norm]])
    axs[i].axis('off')

plt.show()

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

Loop เพื่อทำการแสดงตัวอย่างสีที่ได้จากการหาค่าเฉลี่ยก่อนหน้านี้ โดยเลือกมา 10 ภาพและ 400 ภาพ

```
### START CODE HERE ###
og_img = cv2.cvtColor(cv2.imread(r'C:\Users\Nickv\Documents\ImageProcessing\Week3\asset\main\Turtle.jpg'), cv2.COLOR_BGR2RGB)

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

ทำการโหลดภาพที่จะนำมาทำ mosaic และเปลี่ยนสีจาก RGB เป็น BGR แล้ว copy ภาพไว้ในชื่อ main_img ก่อนนำมาแสดงผล



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

height = 80
width = 120
dim = (height, width)

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

กำหนด dim โดยกำหนด height และ width ที่จะเป็น resolution ของภาพ mosaic

```
### START CODE HERE ###
resize_img = cv2.resize(main_img, (width, height))
plt.imshow(resize_img)
plt.show()
### END CODE HERE ###
```

ทำการ resize ภาพตาม dim ที่กำหนดไว้โดยภาพที่ได้จะมีความละเอียดลดลงตามจำนวน pixel ที่กำหนดไว้ (120, 80)

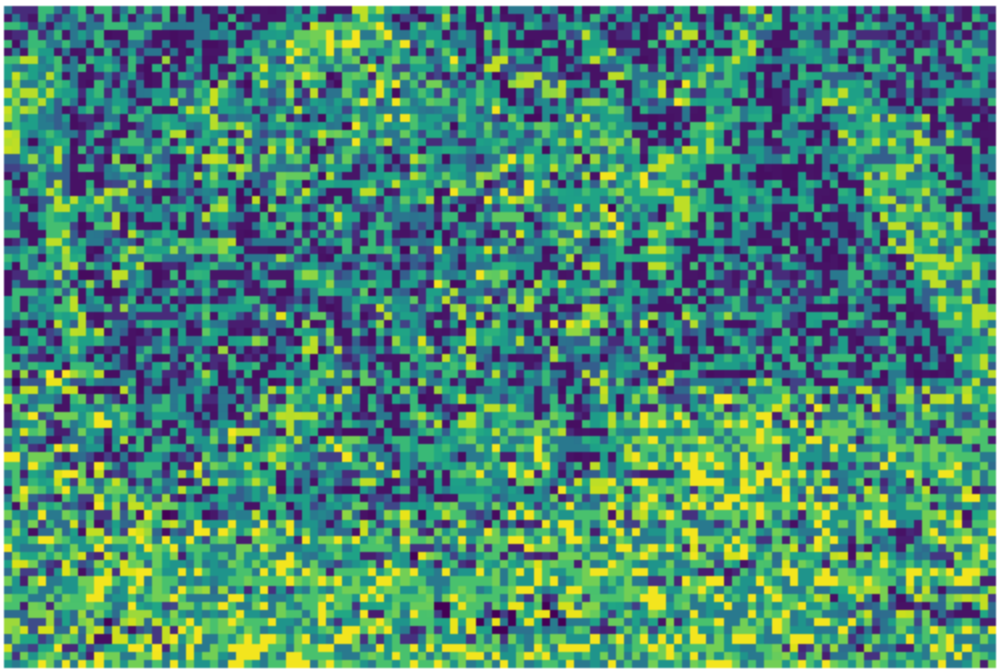
```
### START CODE HERE ###

closest_tiles = np.zeros(dim, dtype=np.uint32)

tree = spatial.KDTree(colors)

for i in range(dim[0]):
    for j in range(dim[1]):
        template = resize_img[i, j]
        match = tree.query(template, k=5)
        pick = random.randint(0, 4)
        closest_tiles[i, j] = match[1][pick]

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

ใช้ np.zeros() สร้าง array ที่มีค่า 0 และมิติตาม dim ที่กำหนดไว้โดยให้ dtype เป็น np.uint32 และสร้าง tree ขึ้นมาจาก List colors

ทำการ Loop ทุกๆ pixel ในรูปที่จะทำ mosaic โดยเก็บค่าของ pixel นั้นๆ ไว้ใน template แล้วทำการหารูปที่มีสีใกล้เคียงด้วย tree.query() จาก template ซึ่ง tree query จะสามารถกำหนดค่า k ได้ โดยค่า k หมายถึงจำนวน node ใน tree ที่มีค่าเฉลี่ยสีใกล้เคียงกับ template โดยยิ่ง k น้อยหรือ k เป็น 1 ภาพที่ได้จากการทำ mosaic ก็จะมีความใกล้เคียงมากขึ้น และยิ่งค่า k มากก็จะมีความหลากหลายของภาพ tile มากขึ้นแต่สีจะมีความใกล้เคียงภาพจริงน้อยลง โดยเรากำหนดให้ k = 5 เพื่อเลือกภาพ tile มา 5 ภาพแล้วทำการสุ่มด้วย random.randint() ต่ออีกรอบเพื่อความหลากหลายของภาพ แล้วทำการเก็บไว้ใน closest_tiles ซึ่งจะได้ค่ามาเป็น index ของ tile ที่เก็บไว้ใน List tiles

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

plt.imshow(closest_tiles)
plt.axis('off')
plt.show()

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

ทำการแสดงผล closest_tiles



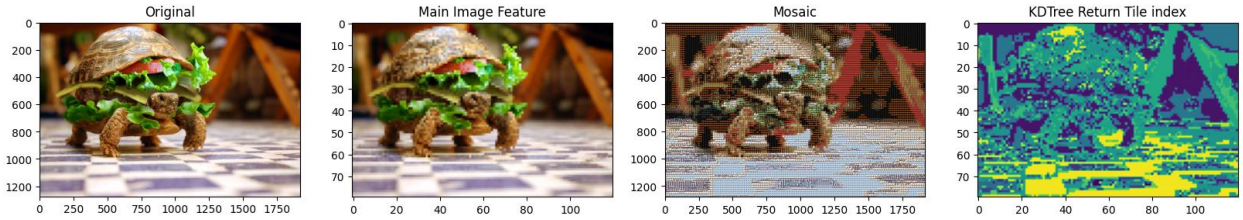
จะเห็นภาพจริงได้ลางๆ เนื่องจาก index ของ pixel ที่มีความคล้ายกันของสีจะมีโอกาสมีค่าเดียวกันแต่เนื่องจากความละเอียดที่น้อยและการกำหนดค่า K เป็น 5 ทำให้มีความคละกันเล็กน้อยจึงมองได้ไม่ชัดเจน

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

tile_height, tile_width = tile_size[0], tile_size[1]
dim = (80, 120)
mosaic_height = dim[0] * tile_height
mosaic_width = dim[1] * tile_width

output = np.zeros((mosaic_height, mosaic_width, 3), dtype=np.uint8)

for i in range(dim[0]):
    for j in range(dim[1]):
        template = resize_img[i, j]
        match = tree.query(template, k=1)
        closest_tiles[i, j] = match[1]
        tile_index = closest_tiles[i, j]
        tile = tiles[tile_index]
        output[i * tile_height:(i + 1) * tile_height, j * tile_width:(j + 1) * tile_width] = tile

plt.imshow(output)
plt.axis('off')
plt.show()

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

ทำการคำนวนขนาดของภาพที่จะนำมาทำ mosaic โดยใช้ความสูงของ tile มาคูณกับ resolution ที่ต้องการโดยเราเก็บไว้ใน dim แล้วจึงสร้าง output ซึ่งเป็น array 0 ที่มีขนาดตามที่คำนวนไว้ก่อนที่จะนำมา Loop ทำตามขั้นตอนที่แล้ว แต่ในรอบนี้นำ index ที่ได้จาก closest_tiles มาใช้เรียก tile จาก List tiles เพื่อนำมาใส่ใน output จนครบทุก pixel (ในการดูตำแหน่งของ output ที่จะนำรูปมาใส่จะใช้เป็น slicing โดยใช้ I, j คูณกับขนาดของ tile เพื่อใส่ tile แต่ละภาพ) ก่อนจะนำมาแสดงผล

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

fig, axs = plt.subplots(1, 4, figsize=(20, 15))

axs[0].imshow(og_img)
axs[0].set_title('Original')

axs[1].imshow(resize_img)
axs[1].set_title('Main Image Feature')

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

axs[3].imshow(closest_tiles)
axs[3].set_title('KDTree Return Tile index')

plt.show()

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

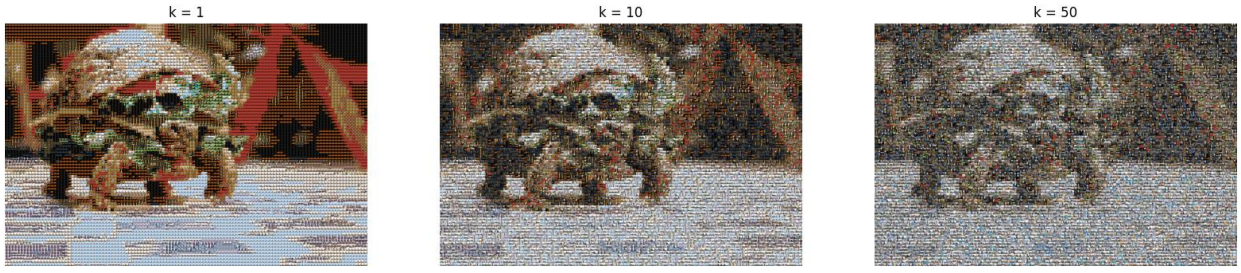
ทำการแสดงผลภาพทั้ง 4 รูปแบบ ภาพต้นแบบ ภาพที่ถูกปรับขนาด ภาพที่ทำ mosaic และ ภาพ closest_tiles



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

output_k1 = np.zeros((mosaic_height, mosaic_width, 3), dtype=np.uint8)
output_k10 = np.zeros((mosaic_height, mosaic_width, 3), dtype=np.uint8)
output_k50 = np.zeros((mosaic_height, mosaic_width, 3), dtype=np.uint8)

def fill_output(k, output):
    for i in range(dim[0]):
        for j in range(dim[1]):
            template = resize_img[i, j]
            match = tree.query(template, k=k)
            if k == 1:
                tile_index = match[1]
            else:
                pick = random.randint(0, k - 1)
                tile_index = match[1][pick]
            tile = tiles[tile_index]
            output[i * tile_height:(i + 1) * tile_height, j * tile_width:(j + 1) * tile_width] = tile

fill_output(1, output_k1)
fill_output(10, output_k10)
fill_output(50, output_k50)

fig, axs = plt.subplots(1, 3, figsize=(20, 15))
```

```
axs[0].imshow(output_k1)
axs[0].set_title('k = 1')
axs[0].axis('off')

axs[1].imshow(output_k10)
axs[1].set_title('k = 10')
axs[1].axis('off')

axs[2].imshow(output_k50)
axs[2].set_title('k = 50')
axs[2].axis('off')

plt.show()

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

ทำการ query โดยปรับค่า K เป็น 1, 10, 50 แล้วจึงแสดงผล



จะเห็นได้ว่า K = 1 ใกล้เคียงกับภาพจริงมากที่สุดแต่จะมีภาพที่ซ้ำกันเยอะโดยเฉพาะตรงที่มีเดียวเยอะๆ แต่เมื่อเพิ่มค่า K เป็น 10 และ 50 ตามลำดับก็จะมีความใกล้เคียงภาพจริงน้อยลงแต่ภาพ tile ไม่ค่อยซ้ำกัน