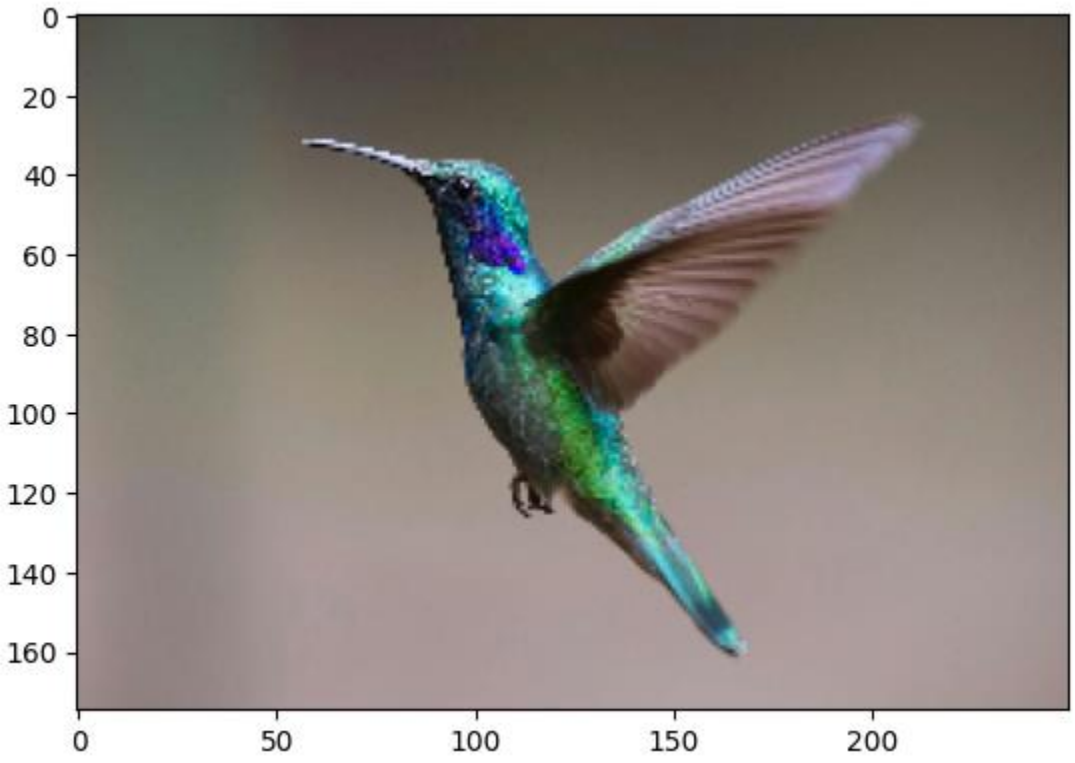
ณสิต ผลัญชัย 65010273

วิวัตร เตชะโกศล 65011001

Lab 4

```
### START CODE HERE ###
image = cv2.resize(cv2.cvtColor(cv2.imread(r'C:\Users\Nickv\Documents\ImageProcessing
plt.imshow(image)
plt.show()
### END CODE HERE ###
```

ทำการโหลดรูปแล้วเปลี่ยนสีจาก BGR เป็น RGB รวมถึงทำการ resize เป็น (250, 175)



```
### START CODE HERE ###
blurry_image = cv2.GaussianBlur(image, (11, 11), 10)
fd, hog_image = hog(blurry_image, orientations=9, pixels_per_cell=(4, 4),
                    cells_per_block=(2, 2), visualize=True, channel_axis=-1)

fig, axs = plt.subplots(1, 2, figsize=(15, 15))

axs[0].imshow(blurry_image)
axs[1].imshow(hog_image, cmap='gray')

plt.show()
### END CODE HERE ###
```

ทำการใช้ function GaussianBlur โดยให้ kernel มีขนาดเป็น (11, 11) และ SD เป็น 10

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

ซึ่งจะนำขนาดของ kernel และ SD คำนวณด้วยสมการด้านบนแล้วนำ matrix ที่ได้ขนาด 11 x 11 นั้นมาทำการ convolution กับ image ที่เรา input เข้าไปแล้วเก็บไว้ใน blurry_image

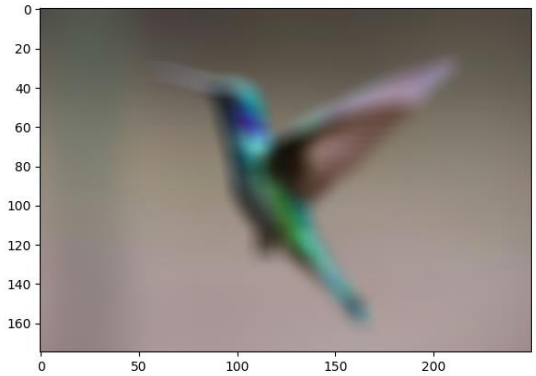
ทำการใช้ function hog เพื่อหา Histogram of Oriented Gradients โดยมี parameters ที่สำคัญเป็น

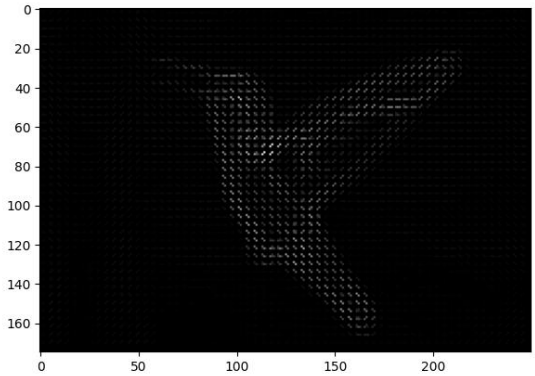
orientations คือจำนวน bin ที่จะใช้แบ่งมุม 0 – 180 องศา เช่นถ้าเป็น 9 มุม 0 – 180 ก็จะถูกแบ่งเป็น 9 ช่วงเท่าๆกัน

pixels_per_cell คือจำนวน pixel ต่อ 1 cell หรือก็คือขนาดของ cell ที่เราจะนำมาคำนวณ

cells_per_block คือจำนวณของ cell ต่อ 1 block โดยจะนำ block มาทำ histogram normalization ซึ่งจะมีผลต่อผลลัพธ์หลังการคำนวณโดยเฉพาะการเปลี่ยนแปลงของสีและคอนทราส

จากนั้นก็ทำการแสดงผลรูปที่ได้จากการ Gaussian Blur และ รูปที่ได้จากการทำ HOG





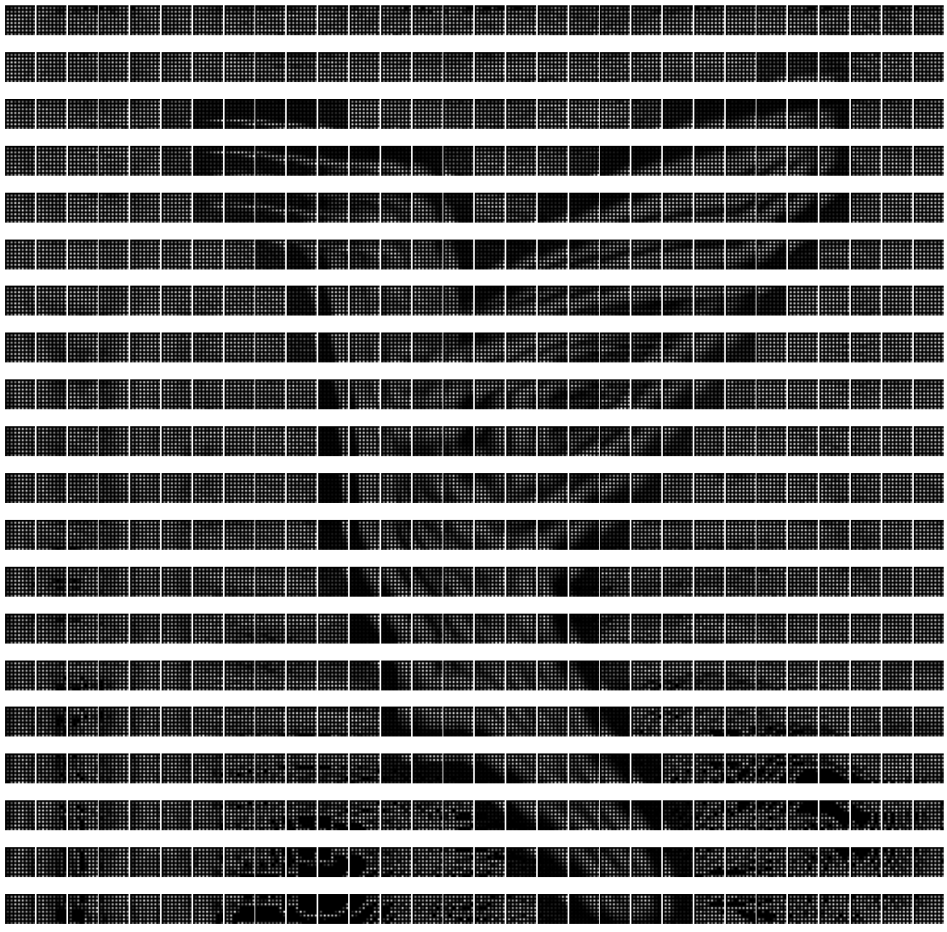
```python
### START CODE HERE ###
class HOGSubimageExtractor:
    def __init__(self, image, tile_size, stride):
        self.image = image
        self.tile_size = tile_size
        self.stride = stride
        self.h, self.w, _ = image.shape
        self.hGrid = []
        self.wGrid = []
        self.hog_features = []
        self.hog_images = []
        self.extract_hog_features()

    def extract_hog_features(self):
        self.hGrid = []
        self.wGrid = []
        self.hog_features = []
        self.hog_images = []

        for y in range(0, self.h - self.tile_size + 1, self.stride):
            for x in range(0, self.w - self.tile_size + 1, self.stride):
                sub_img = self.image[y:y+self.tile_size, x:x+self.tile_size]
                hog_feature, hog_image = hog(
                    sub_img,
                    pixels_per_cell=(self.tile_size//8, self.tile_size//8),
                    cells_per_block=(2, 2),
                    visualize=True,
                    channel_axis=-1
                )
                self.hog_features.append(hog_feature)
                self.hog_images.append(hog_image)
                self.hGrid.append(y)
                self.wGrid.append(x)

        return self.hog_features
```

```python
    def plot_hog_images(self):
        num_tiles = len(self.hog_images)
        num_cols = (self.w - self.tile_size) // self.stride + 1
        num_rows = (self.h - self.tile_size) // self.stride + 1
        fig, axes = plt.subplots(num_rows, num_cols, figsize=(15, 15))
        if num_rows == 1:
            axes = [axes]

        for idx, ax in enumerate(axes.flat):
            if idx < num_tiles:
                ax.imshow(self.hog_images[idx], cmap='gray')
                ax.axis('off')
            else:
                ax.remove()
        plt.subplots_adjust(wspace=0.05, hspace=0.05)
        plt.show()

    def get_num_grid(self):
        return len(self.hGrid), len(self.wGrid)
### END CODE HERE ###
```

ทำการเติมโค้ดใน Class HOGSubimageExtractor โดยจะมี function extract_hog_features() ซึ่งจะทำงานเหมือนกับ hog ด้านบนแต่จะแบ่งรูปออกเป็นรูปเล็กๆ ก่อนนำมาหา HOG โดยเราจะต้องกำหนด tile_size และ stride โดยรูปย่อยที่ถูกแบ่งจะมีขนาด tile_size * tile_size และถูกเลื่อนไปทีละ stride โดยรูปผลลัพธ์จากการทำ hog ของแต่ละ tile จะถูกเก็บไว้ใน list hog_image เพื่อใช้แสดงผลใน function plot_hog_images()

```
### START CODE HERE ###
tile_size = 16
stride = 8
hog_extractor = HOGSubimageExtractor(blurry_image, tile_size, stride)
num_grid = hog_extractor.get_num_grid()
print(f'Number of grids: {num_grid}')
hog_extractor.plot_hog_images()
### END CODE HERE ###
```

ทดลอง HOG sub-image extract ด้วย tile_size = 16, stride = 8 จะได้จำนวน grids: (600, 600) และมีผลลัพธ์ดังรูปด้านล่าง



```python
### START CODE HERE ###
class KMeansCluster:
    def __init__(self, hog_extractor, n_clusters, random_state):
        self.hog_extractor = hog_extractor
        self.n_clusters = n_clusters
        self.random_state = random_state
        self.cluster_array = None
        self.all_labels = None
        self.bounding_boxes = None
        self.perform_clustering()

    def perform_clustering(self):
        hog_features = self.hog_extractor.extract_hog_features()

        kmeans = KMeans(n_clusters=self.n_clusters, random_state=self.random_state)
        kmeans.fit(hog_features)
        cluster_labels = kmeans.labels_

        num_rows = (self.hog_extractor.h - self.hog_extractor.tile_size) // self.hog_extractor.stride + 1
        num_cols = (self.hog_extractor.w - self.hog_extractor.tile_size) // self.hog_extractor.stride + 1

        self.cluster_array = cluster_labels.reshape((num_rows, num_cols))

        self.all_labels = measure.label(self.cluster_array)
```

```python
    def plot_cluster_and_labels(self):
        plt.figure(figsize=(12, 6))

        plt.subplot(1, 2, 1)
        plt.title("Cluster Assignments")
        plt.imshow(self.cluster_array, cmap='viridis')

        plt.subplot(1, 2, 2)
        plt.title("Connected Components")
        plt.imshow(self.all_labels, cmap='nipy_spectral')

        plt.show()



    def get_bounding_boxes(self, scale_x, scale_y, min_area=100):
        self.bounding_boxes = []

        regions = measure.regionprops(self.all_labels)

        for region in regions:
            if region.area >= min_area:
                start_x = int(region.bbox[1] * scale_x)
                start_y = int(region.bbox[0] * scale_y)
                end_x = int(region.bbox[3] * scale_x)
                end_y = int(region.bbox[2] * scale_y)

                self.bounding_boxes.append((region.label, (start_x, start_y), (end_x, end_y)))

        return self.bounding_boxes
```

สร้าง class KMeansCluster เผื่อนำผล HOG มาทำ K-means clustering แล้วหา connected components แล้วหาของเขตของ bounding boxes โดย class นี้จะรับตัวแปร 3 ตัว ประกอบด้วย hog_extractor จะรับ object hog_extractor จากโค้ดด้านบนมาใช้ต่อ n_clusters คือจำนวน clusters ที่เราจะใช้ใน K-means และ random_state คือ seed ในการสุ่มทำให้ run แต่ละครั้งจะ random เหมือนเดิม

Function perform_clustering() จะนำ hog features มาจัดกลุ่มด้วย model KMeans โดยใช้คำสั้ง kmeans.fit() ซึ่งจะได้ผลลัพท์ในการจับกลุ่มมาแล้วเก็บไว้ใน cluster_labels จากนั้นทำการ reshape cluster_labels ให้เป็น 2มิติ แล้วใช้ measure.label() เพื่อหา connected components แล้วเก็บไว้ใน all_labels

Function plot_cluster_and_labels() ใช้เพื่อแสดงผลภาพของการจัดกลุ่ม clustering และ connected components

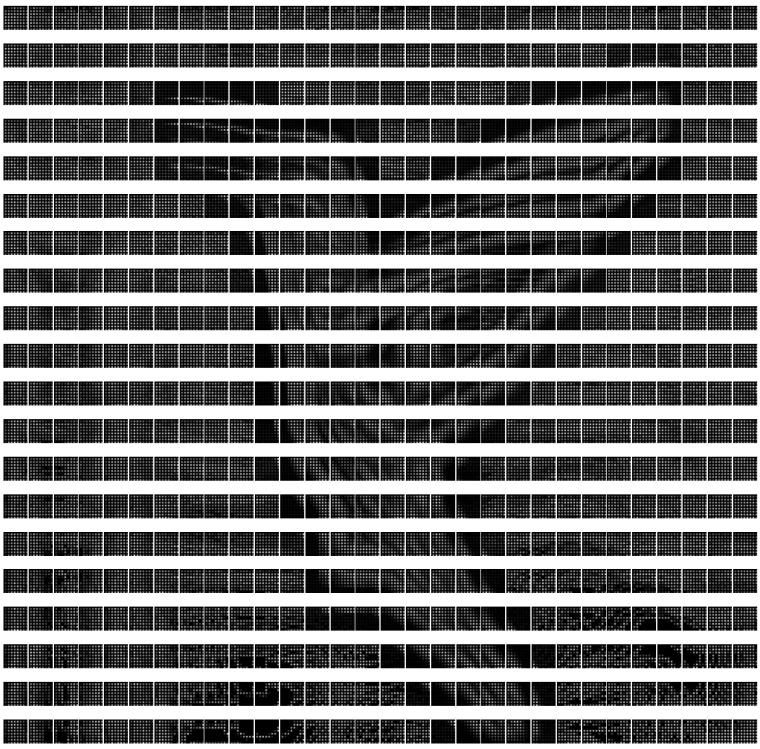
Function get_bounding_boxes() ใช้เพื่อหาขอบเขตของวัตถุที่เราจะตีกรอบโดยใช้ measure.regionprops() ซึ่งจะได้พื้นที่ที่ถูก label ทั้งขนาดและพิกัด จากนั้นทำการกรองพื้นที่ขนาดเล็กเกินไปที่เราไม่ต้องการออกด้วยการลูปเพื่อให้เหลือเฉพาะพิกัดของวัตถุที่ต้องการ จากนั้นนำพิกัดมาปรับ scale ให้ตรงกับรูปจริงแล้วเก็บไว้ใน list bounding_boxes ก่อน return ค่า bounding_boxes

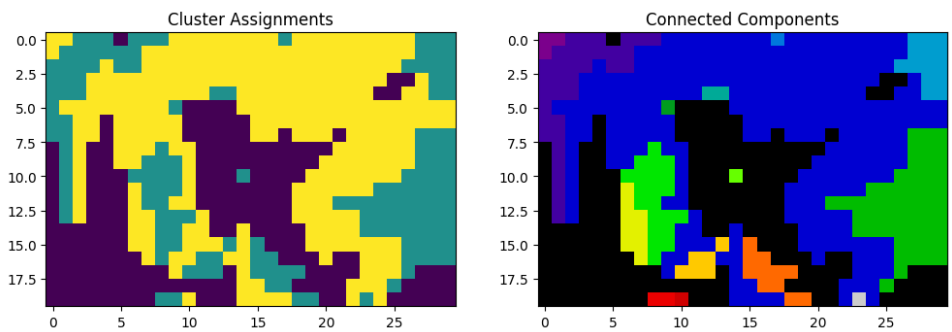
```
def draw_bbox(image, bboxes):
    for object_id, start_coords, end_coords in bboxes:
        start_x, start_y = start_coords
        end_x, end_y = end_coords

        cv2.rectangle(image, (start_x, start_y), (end_x, end_y), color=(0, 255, 0), thickness=2)

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    plt.figure(figsize=(6, 6))
    plt.imshow(image)
    plt.show()
```

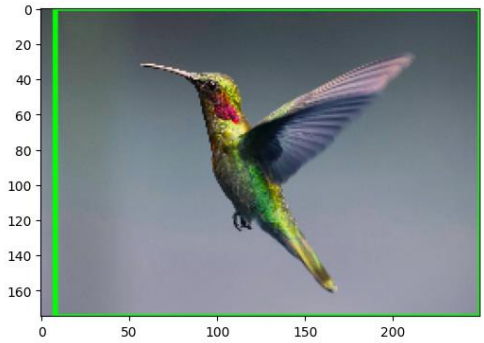
Function draw_bbox() ใช้เพื่อวาดกรอบสี่เหลี่ยมครอบ connected components จากพิกัดที่เราได้มาด้วย cv2.rectangle แล้วแสดงผล

```
tile_size = 16
stride = 8
hog_extractor = HOGSubimageExtractor(blurry_image, tile_size, stride)
num_grid = hog_extractor.get_num_grid()
print(f'Number of grids: {num_grid}')
hog_extractor.plot_hog_images()

kmeans_cluster = KMeansCluster(hog_extractor, 3, 32)
kmeans_cluster.plot_cluster_and_labels()
cluster_rows, cluster_cols = kmeans_cluster.cluster_array.shape
scale_x = image.shape[1] / cluster_cols
scale_y = image.shape[0] / cluster_rows
bboxes = kmeans_cluster.get_bounding_boxes(scale_x, scale_y)
image = cv2.resize(cv2.cvtColor(cv2.imread(r'C:\Users\Nickv\Documents\ImageProcessing\Week4\asset\Bird.jpg')
draw_bbox(image, bboxes)
```

ทำการทดลองโดยใช้ tile_size = 16, stride = 8 จำนวน cluster = 3, random state = 32 ได้ผลลัพท์ด้านล่าง

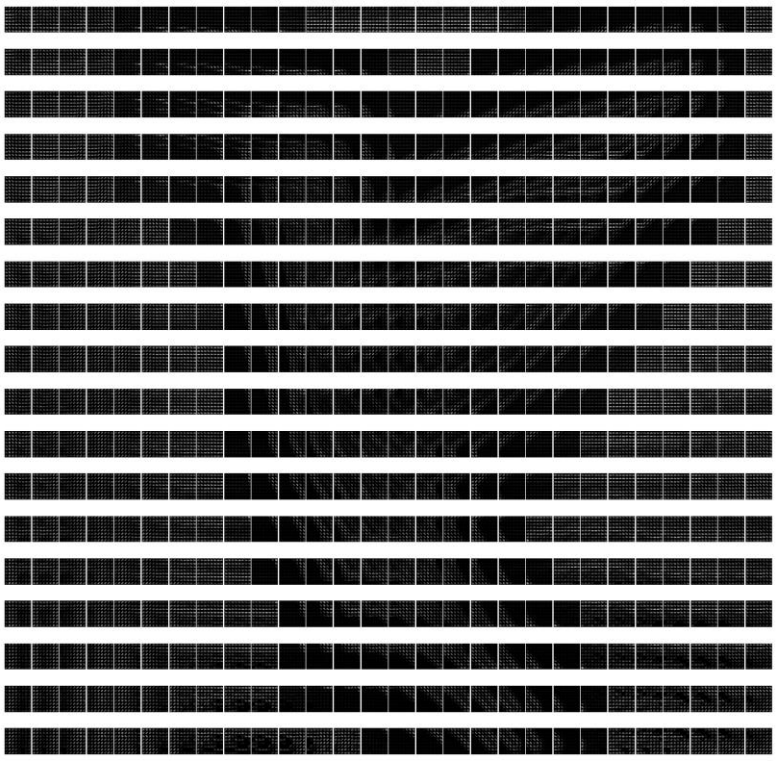


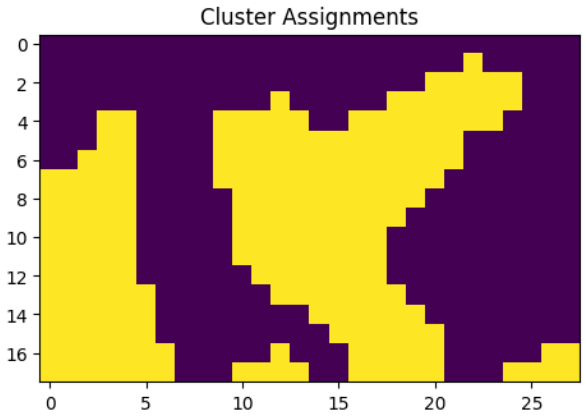


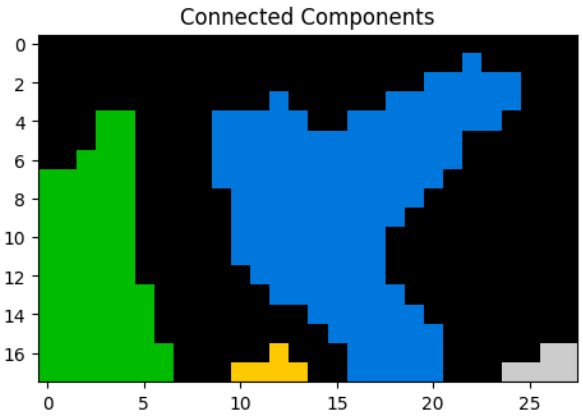


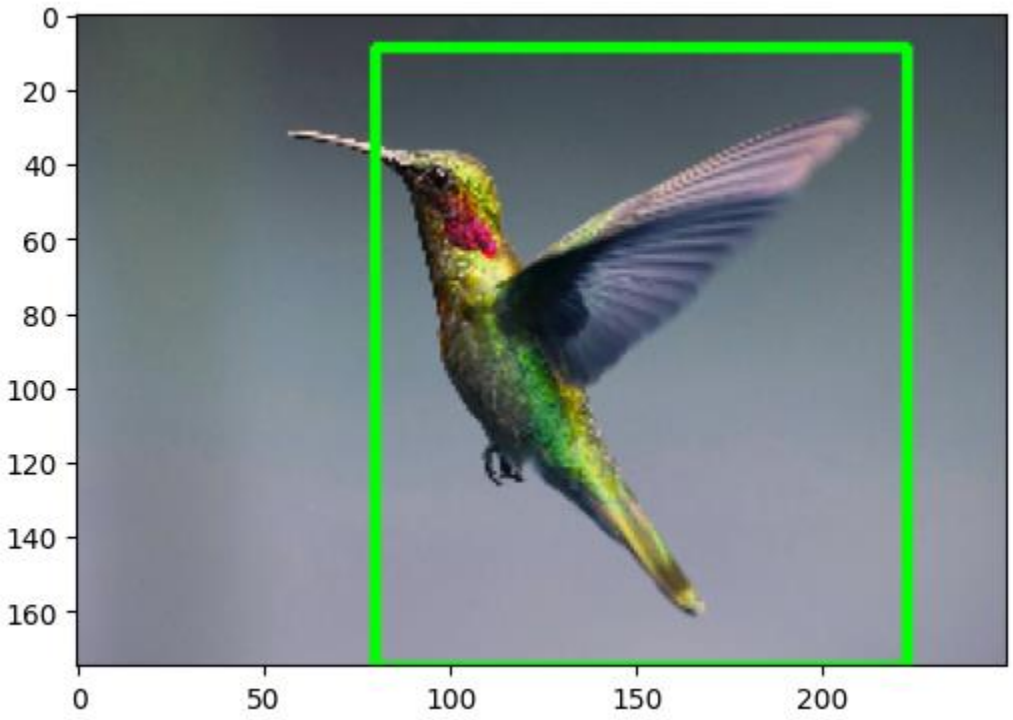

```
tile_size = 32
stride = 8

hog_extractor = HOGSubimageExtractor(blurry_image, tile_size, stride)
num_grid = hog_extractor.get_num_grid()
print(f'Number of grids: {num_grid}')
hog_extractor.plot_hog_images()

kmeans_cluster = KMeansCluster(hog_extractor, 2, 32)
kmeans_cluster.plot_cluster_and_labels()
cluster_rows, cluster_cols = kmeans_cluster.cluster_array.shape
scale_x = image.shape[1] / cluster_cols
scale_y = image.shape[0] / cluster_rows
bboxes = kmeans_cluster.get_bounding_boxes(scale_x, scale_y)
image = cv2.resize(cv2.cvtColor(cv2.imread(r'C:\Users\Nickv\Documents\ImageProcessing\Week4\asset\Bird.jpg')
draw_bbox(image, bboxes)
```

ผลลัพท์ครั้งที่ดีที่สุด tile_size = 32, stride = 8 จำนวน cluster = 2, random state = 32 ได้ผลลัพท์ด้านล่าง




Cluster Assignments
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25


Connected Components
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25


0
20
40
60
80
100
120
140
160
0
50
100
150
200