

Lab 4

Image Segmentation

```
[1]: import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage import measure

import glob

from sklearn.cluster import KMeans

from scipy import spatial

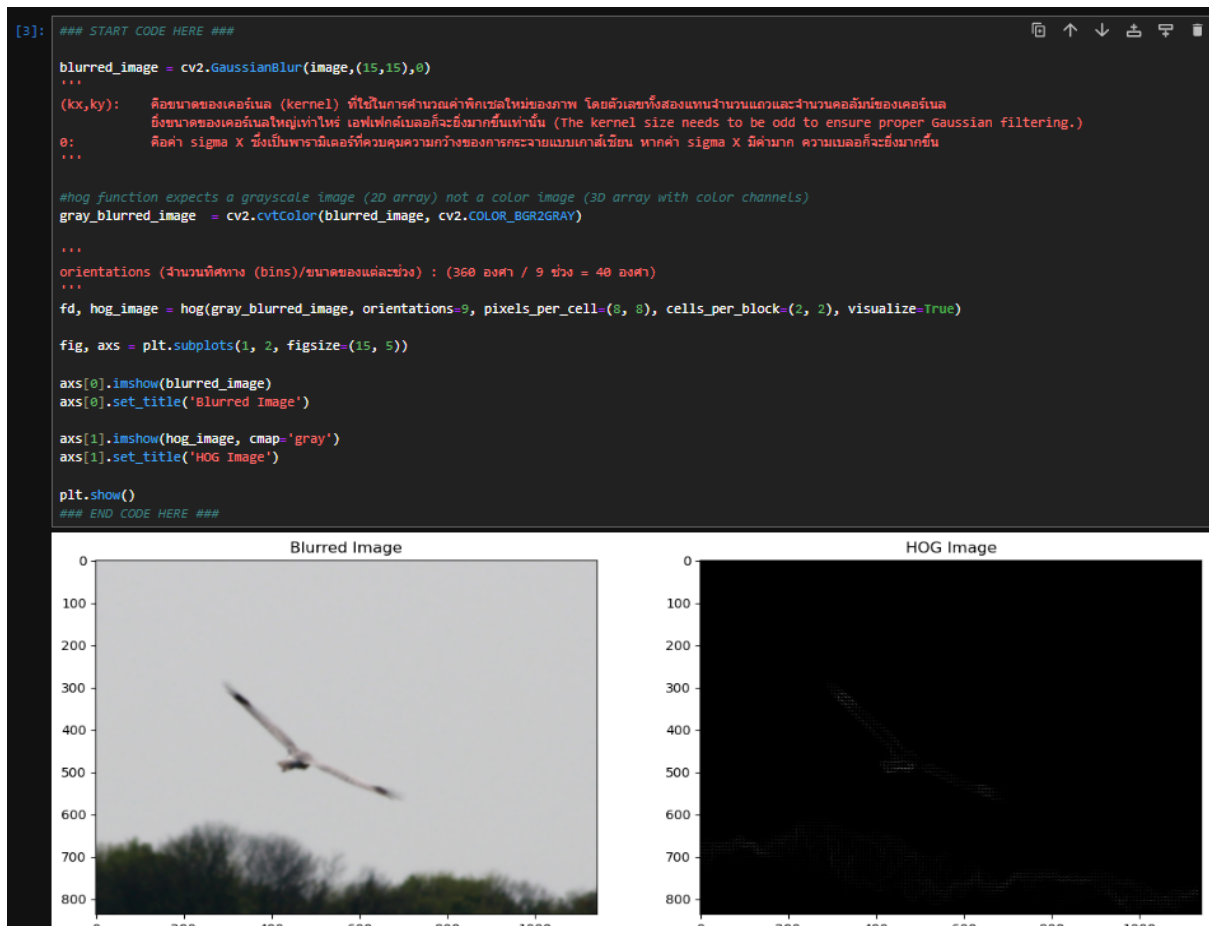
from tqdm import tqdm
import cv2
import os
import random
```

[1] จะเป็นการ Import Library ที่จำเป็นที่ต้องใช้ใน Lab 4 นี้

```
[2]: ### START CODE HERE ###
image = cv2.imread('images/bird.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.show()
### END CODE HERE ###
```



- [2]
- จะเป็นการโหลดรูปมาใช้งาน โดยใช้การ Read bird.jpg using cv2.imread
 - Convert BGR to RGB using cv2.cvtColor
 - และสุดท้ายเป็นการแสดงรูปภาพ show image using plt ที่ได้หลังจากการ convert



[3]

- `blurred_image = cv2.GaussianBlur(image, (15, 15), 0)`
 - apply Gaussian blur to the input image
 - `(kx,ky):` คือขนาดของเคอร์เนล (kernel) ที่ใช้ในการคำนวณค่าพิกเซลใหม่ของภาพ โดยตัวเลขทั้งสองแทนจำนวนแถวและจำนวนคอลัมน์ของเคอร์เนล ยิ่งขนาดของเคอร์เนลใหญ่เท่าไร เอฟเฟกต์เบลอก็จะยิ่งมากขึ้นเท่านั้น (The kernel size needs to be odd to ensure proper Gaussian filtering.)
 - `0:` คือค่า sigma X ซึ่งเป็นพารามิเตอร์ที่ควบคุมความกว้างของการกระจายแบบเกาส์เซียน หากค่า sigma X มีค่ามาก ความเบลอก็จะยิ่งมากขึ้น
- `gray_blurred_image = cv2.cvtColor(blurred_image, cv2.COLOR_BGR2GRAY)`
 - เป็นการ convert blurred image from BGR to grayscale
 - โดยที่ HOG feature extraction จะทำงานกับ grayscale images, which are 2D arrays, rather than 3D color images.

- `fd, hog_image = hog(gray_blurred_image, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)`
 - เป็น command ให้ : compute HOG feature เพื่อสำหรับภาพ grayscale ที่ blurred
 - `orientations=9` : (จำนวนทิศทาง (bins)/ขนาดของแต่ละช่วง) : (360 องศา / 9 ช่วง = 40 องศา)
 - `pixels_per_cell=(8, 8)` : size of cell that the histogram is computed.
 - `cells_per_block=(2, 2)` : The number of cells in each block used to normalize the histograms.
 - `visualize=True` : return image that visualizes the HOG features.
- โดยที่ Histogram of Oriented Gradients (HOG): เป็นเทคนิคที่ใช้ในการสกัดลักษณะของวัตถุในภาพ โดยอาศัยการวิเคราะห์การไล่ระดับสี (gradient) ของภาพ ซึ่งการไล่ระดับสีจะบอกถึงทิศทางและความเข้มของการเปลี่ยนแปลงของความสว่างในภาพ ซึ่งผลลัพธ์ที่ออกมาจาก command นี้คือ :
 - **fd (feature descriptor)**: คือ เวกเตอร์ลักษณะที่ได้จากการประมวลผล HOG ซึ่งเป็นตัวแทนของภาพนั้นๆ โดยเวกเตอร์นี้จะเก็บข้อมูลเกี่ยวกับทิศทางและความเข้มของการไล่ระดับสีของภาพ
 - **hog_image**: คือ ภาพที่แสดงผลการคำนวณ HOG โดยจะแสดงทิศทางของการไล่ระดับสีในแต่ละเซลล์ของภาพ ซึ่งจะช่วยให้เห็นภาพรวมของการเปลี่ยนแปลงของความสว่างในภาพ
- ซึ่งต่อมาก็จะเป็นการแสดงผลภาพต้นฉบับ คู่กับภาพที่ได้หลังจากการทำ hog เพื่อดูการเปลี่ยนแปลงความสว่างของภาพที่ได้

```
•[4]: ### START CODE HERE ###
class HOGSubImageExtractor:
    def __init__(self, image, tile_size, stride):
        # Process the image: convert to grayscale and apply Gaussian blur
        processed_image = self.preprocess_image(image)

        # Resize the image so that dimensions are divisible by both tile_size and stride
        # self.image = self.resize_image_to_divisible(self.processed_image, tile_size, stride)
        self.image = processed_image

        self.tile_size = tile_size
        self.stride = stride
        self.h, self.w = self.image.shape[:2]
        self.hGrid, self.wGrid = self.compute_grid_indices()
        self.hog_features = []
        self.hog_images = []
        self.extract_hog_features()

    def preprocess_image(self, image):
        gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred_image = cv2.GaussianBlur(gray_image, (15, 15), 0)
        return blurred_image

    def compute_grid_indices(self):
        # Compute grid indices for rows and columns
        hGrid = range(0, self.h - self.tile_size + 1, self.stride)
        wGrid = range(0, self.w - self.tile_size + 1, self.stride)
        return hGrid, wGrid
```

```
def extract_hog_features(self):
    orientations = 9
    pixels_per_cell = (8, 8)
    cells_per_block = (2, 2)

    for y in self.hGrid:
        for x in self.wGrid:
            tile = self.image[y:y+self.tile_size, x:x+self.tile_size]
            fd, hog_image = hog(tile, orientations, pixels_per_cell, cells_per_block, visualize=True, channel_axis=None)
            self.hog_features.append(fd)
            self.hog_images.append(hog_image)

def plot_hog_images(self):
    num_rows = len(self.hGrid)
    num_cols = len(self.wGrid)

    fig, axes = plt.subplots(num_rows, num_cols, figsize=(num_cols, num_rows))
    axes = axes.flatten()

    for idx, ax in enumerate(axes):
        if idx < len(self.hog_images):
            ax.imshow(self.hog_images[idx], cmap='gray')
            ax.axis('off') # Turn off axis
        else:
            ax.axis('off') # Hide extra subplots if any
    plt.subplots_adjust(wspace=0, hspace=0)
    plt.tight_layout(pad=0)
    plt.show()

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

[4] ต่อมาจะเป็นการสร้าง class **HOGSubimageExtractor**:


เพื่อเป็นการแบ่งภาพออกเป็นส่วนย่อยๆ และคำนวณ HOG สำหรับแต่ละส่วน ซึ่งจะนำไปประยุกต์กับการใช้กับงาน การตรวจจับวัตถุในภาพในลำดับถัดไป

- **def preprocess_image(self, image):**
 - Converts the image to grayscale and applies a Gaussian blur. This prepares the image for HOG feature extraction. (ฟังก์ชันนี้จะทำการเตรียมภาพก่อนการคำนวณ HOG)
- **def compute_grid_indices(self):**
 - Calculates the grid for tiling the image (ฟังก์ชันนี้คำนวณตำแหน่งที่จะแบ่งภาพ ซึ่งแนวตั้ง: hGrid และ แนวนอน: wGrid โดยเว้นระยะห่างตาม stride)
- **def extract_hog_features(self):**
 - ฟังก์ชันคำนวณ HOG สำหรับแต่ละส่วนของภาพ
 - Iterates over each tile in the image and extracts the HOG features
 - วน loop ไปตามตำแหน่งที่ได้จาก compute_grid_indices
 - ตัดส่วนของภาพ (subimage) ตามขนาด tile_size
 - ใช้ฟังก์ชัน hog เพื่อคำนวณ HOG ของแต่ละส่วน
 - เก็บผลลัพธ์ซึ่งเป็น feature vector (fd) และ HOG image ไว้ในลิสต์ hog_features และ hog_images ตามลำดับ
- **def plot_hog_images(self):**
 - Visualizes the HOG images for all tiles in a grid layout
- **def get_num_grid(self):**

- ฟังก์ชันนี้จะแสดงผล HOG image ในรูปแบบของ plot โดยจัดเรียง HOG image เป็นตารางตามจำนวน row และ column ที่แบ่งภาพ
- Returns the number of tiles in the vertical and horizontal directions.

```
[5]: ### START CODE HERE ###
tile_size = 64
stride = 32
hog_extractor = HOGSubimageExtractor(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 ###
```

Number of grids: (25, 34)



[5] แบ่งภาพออกเป็นส่วนย่อยขนาด 64x64 pixels โดยมีระยะห่างระหว่างส่วนย่อย 32 pixels คำนวณ HOG สำหรับแต่ละส่วนย่อย และแสดงภาพ HOG ทั้งหมดออกมา

- Create HOGSubimageExtractor
- Get the Number of Grids : ขนาดที่ได้จากการแบ่งภาพด้วย tile และ stride บนภาพต้นฉบับ
- Plot HOG Image

```
[6]: ### START CODE HERE ###
class KMeansCluster:
    def __init__(self, hog_extractor, n_clusters, random_state=42):
        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 = []
        self.perform_clustering()

    def perform_clustering(self):
        # Perform K-means clustering on the HOG features
        kmeans = KMeans(n_clusters=self.n_clusters, n_init=10, random_state=self.random_state)
        kmeans.fit(self.hog_extractor.hog_features)

        # Reshape the cluster labels to the grid shape
        self.cluster_array = kmeans.labels_.reshape(len(self.hog_extractor.hGrid), len(self.hog_extractor.wGrid))

        # Identify connected components (objects) in the cluster array
        self.all_labels = measure.label(self.cluster_array)

    def plot_cluster_and_labels(self):
        # Plot cluster assignments
        fig, axes = plt.subplots(1, 2, figsize=(12, 5))

        # Cluster assignments
        im1 = axes[0].imshow(self.cluster_array)
        axes[0].set_title('Cluster Assignments')

        # Connected components
        im2 = axes[1].imshow(self.all_labels)
        axes[1].set_title('Connected Components')
        fig.colorbar(im2, ax=axes[1])

        plt.show()

    def get_bounding_boxes(self):
        # Extract bounding boxes for each detected object
        for region in measure.regionprops(self.all_labels):
            min_row, min_col, max_row, max_col = region.bbox
            self.bounding_boxes.append((region.label, (min_col, min_row), (max_col, max_row)))
        return self.bounding_boxes
### END CODE HERE ###
```

[6] **class KMeansCluster:**

class: KMeansCluster ใช้สำหรับการแบ่งกลุ่มข้อมูล HOG features ด้วย K-means clustering เพื่อหา bounding boxes ของกลุ่มที่เชื่อมต่อกัน คล้ายการตรวจจับวัตถุในภาพ

- **def perform_clustering(self):**
 - สร้างวัตถุ kmeans - Initializes a K-means clustering model
 - Fit model to the HOG features extracted from the image tiles โดยแบ่งเป็น n_clusters กลุ่ม
 - reshaped to match the grid layout of the image ซึ่งคือการจัดเรียงผลลัพธ์ (cluster labels) ให้ตรงกับโครงสร้างของการแบ่งภาพ (grid) เก็บไว้ใน self.cluster_array
 - Identify connected components in the cluster array - เป็นการใส่ฟังก์ชัน label จากไลบรารี scikit-image เพื่อหากลุ่มของจุดที่เชื่อมต่อกันใน self.cluster_array เก็บผลลัพธ์ไว้ใน self.all_labels
- **def plot_cluster_and_labels(self):**
 - visualizes the results of clustering ซึ่ง
 - Plot อันแรกจะแสดงผลการแบ่งกลุ่ม (cluster assignments)
 - Plot ที่สองจะแสดงผลกลุ่มของจุดที่เชื่อมต่อกัน (connected components)
- **def get_bounding_boxes(self):**
 - ใช้ฟังก์ชัน regionprops จากไลบรารี scikit-image เพื่อหา bounding boxes ของแต่ละกลุ่มที่เชื่อมต่อกันใน self.all_labels

- เก็บผลลัพธ์ (bounding box และ label) ไว้ในลิสต์ self.bounding_boxes
- Extracts the bounding boxes for each detected object in the image.
 - Returns the bounding box of the region as a tuple

```
[7]: ## START CODE HERE ##
def draw_bbox(image, bboxes, original_shape, grid_shape):
    # สร้างสำเนาของภาพเพื่อไม่ให้ภาพต้นฉบับเสียหาย
    image_copy = image.copy()

    # Scale factors based on original and grid shapes
    scale_x = original_shape[1] / grid_shape[1]
    scale_y = original_shape[0] / grid_shape[0]

    # วาด bounding boxes บนภาพสำเนา
    for bbox in bboxes:
        label, start, end = bbox

        # Scale bounding box coordinates
        start_x = int(start[0] * scale_x)
        start_y = int(start[1] * scale_y)
        end_x = int(end[0] * scale_x)
        end_y = int(end[1] * scale_y)

        # วาด rectangle ด้วยสีแดง (BGR format)
        cv2.rectangle(image_copy, (start_x, start_y), (end_x, end_y), color=(255, 0, 0), thickness=2)

    return image_copy

## END CODE HERE ##
```

[7] **def draw_bbox(image, bboxes, original_shape, grid_shape):**

- Creating a Copy of the Image
- Calculating Scale Factors
 - scales the bounding box coordinates from the grid (tile) size to the original image size (คำนวณค่า scale factor เพื่อปรับขนาดของ bounding box ให้ตรงกับขนาดของภาพเดิม เนื่องจาก bounding box ที่ได้จากการทำ clustering จะอยู่ในระบบพิกัดของ grid ไม่ใช่พิกัดของภาพเดิม)
- Drawing Bounding Boxes
- Returning the Modified Image

```
[8]: kmeans_cluster = KMeansCluster(hog_extractor, n_clusters=3, random_state=42)

# Plot the clusters and labels
kmeans_cluster.plot_cluster_and_labels()

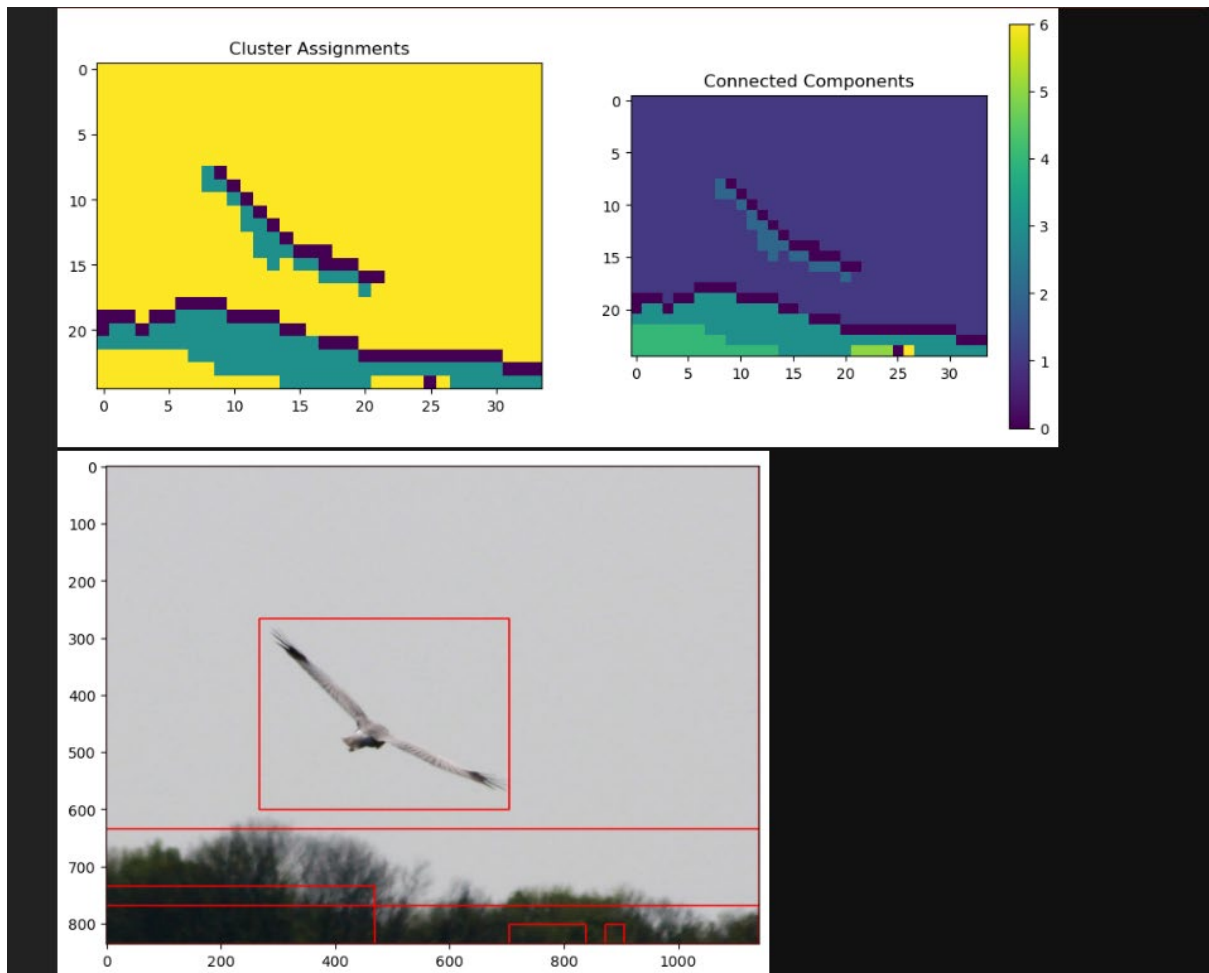
# Get the bounding boxes of detected objects
bboxes = kmeans_cluster.get_bounding_boxes()

|
grid_shape = (len(kmeans_cluster.hog_extractor.hGrid), len(kmeans_cluster.hog_extractor.wGrid))

# ฟังก์ชันที่ไม่ใช้ในการรันโค้ด
def reset_image(image):
    # Return a copy of the original image
    return image.copy()

# Draw the bounding boxes on the original image
image_copy = reset_image(image)
image_with_bboxes = draw_bbox(image_copy, bboxes, image_copy.shape, grid_shape)

# Display the image with bounding boxes
plt.figure(figsize=(8, 8))
plt.imshow(image_with_bboxes)
plt.show()
```

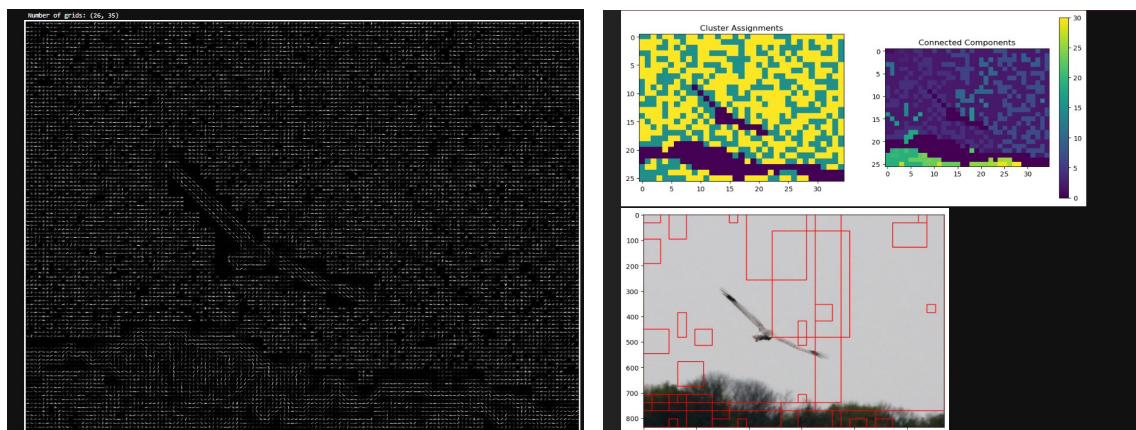



[8] performs K-means clustering on the HOG features extracted from an image, identifies distinct objects within the image, and then draws bounding boxes around these objects

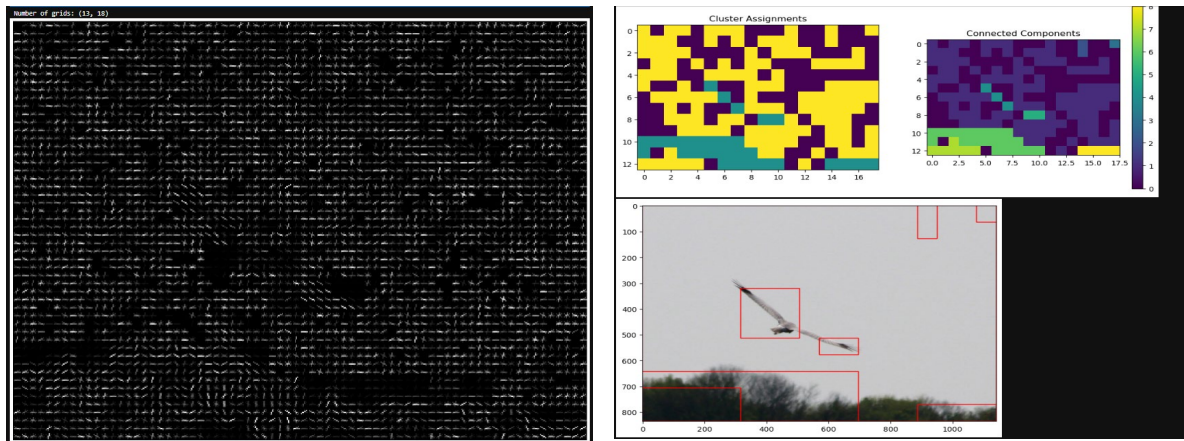
Experiment

- Do the experiment to identify 3 best parameters that produce the perfectly fitting bounding box of the object of interest in the image.

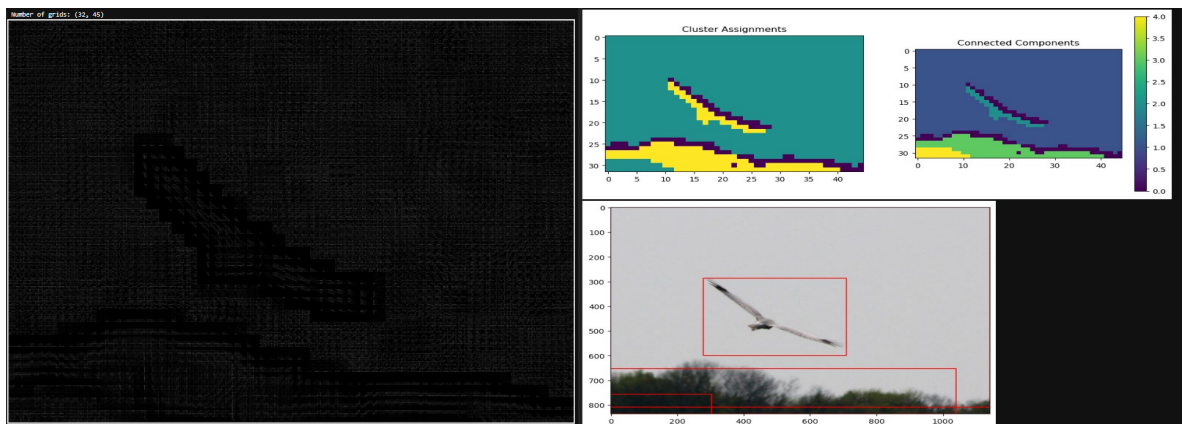
1) `tile_size = 32`, `stride = 32`



2) `tile_size = 32`, `stride = 64`



3) `tile_size = 72`, `stride = 24`



โดยโค้ดทั้ง 3 ส่วนการทดลองก็จะมีส่วนที่คล้ายกันยกเว้นแค่ `tile_size` และ `stride` แต่มีใจความสำคัญดังนี้:

1. การเตรียมข้อมูล:

- กำหนดขนาดของส่วนย่อย (`tile_size`) และระยะห่างระหว่างส่วนย่อย (`stride`)
 - สร้างวัตถุ `HOGSubimageExtractor`
 - คำนวณ Number of grids ที่ได้จาก `tile_size` และ `stride`
 - แสดงผล HOG images
- ```
hog_extractor.plot_hog_images()
```

### 2. การแบ่งกลุ่มข้อมูล:

- สร้างวัตถุ `KMeansCluster`
  - แสดงผลการแบ่งกลุ่ม
- ```
kmeans_cluster.plot_cluster_and_labels()
```

3. การวาด bounding box:

- คึงข้อมูล bounding boxes

- bboxes = kmeans_cluster.get_bounding_boxes()

- คำนวณขนาดของ grid (grid_shape)

- สร้างสำเนาของภาพต้นฉบับ (image_copy) เพื่อจะได้ไม่กระทบกับภาพต้นฉบับในการวาดสี่เหลี่ยมกรอบทับภาพนั้นๆไป

- เรียกฟังก์ชัน draw_bbox เพื่อวาด bounding box ลงบนสำเนาของภาพ

- แสดงผลภาพต้นฉบับที่มี bounding box