

ECSE 415 Project Report

McGill University, Fall 2020

Group #: *No Group*

Group Members:

- Anas Deis 260684605

Description of Person Detector (part 1):

Describe the method used to detect people in part 1. State from where the method was obtained (e.g. website or github repository) and briefly describe how the method works. Provide an image showing the detections obtained on at least one image from the dataset.

The method used in part 1 is detectron2 obtained from Detectron2 Beginner's Tutorial – [Detectron2 Tutorial.ipynb - Colaboratory \(google.com\)](#) from Facebook AI Research. The Pytorch-based modular object detection library currently does not support CPU computations due to the computational power required to run the neural networks, hence why the need to run the code on the GPU. The pre-trained detectron2 model that is run in the implementation of part 1 is based on COCO dataset that uses the Mask RCNN with ResNet-50 backbone and Feature Pyramid Networks (FPN) for object detections.



Figure 1. Detectron2 object detections part 1

Description of Person Detector (part 2):

Describe the method used to detect people in part 2. Briefly describe how the method works. Describe what features you used to feed into the SVM classifier. Explain your choice of window size. Outline the process used to get the training data (positive and negative examples). Explain your choice of the number of training examples.

The method used to detect people in part 2 uses Histograms of Oriented Gradients (HoG) in combination with linear Support Vector Machines (SVM). SVMs are used to train a model to classify if an image contains a certain object or not, and to predict or detect objects after training. HoG is a feature/image descriptor used in image processing and other visual areas for the purpose of object detection. The HoG is capable to distinguish the target and the background with HoG visualization and the technique counts occurrences of gradient orientation in localized portions of an image [1]. The edges are represented by the gradient of an image and is a good representation for a person as they are invariant to color and partially invariant to texture. The edges can also become invariant to brightness if properly normalized. HoG build on the gradients of an image and despite occlusion being a common problem with object detections, the HoG and SVM provide accurate results and an efficient system.

I extract positive and negative samples, join them into *features_train* array, and assign labels (0 for negative and 1 for positive) to them. I extract suitable features using HoG to feed them into an SVM classifier for training as shown in Figure 4. The positive samples are extracted by using the person detector implemented in part 1 as can be seen in Figure 3 to identify the person patches in each image through the bounding boxes coordinates. As for the negative patches, it is a longer process that uses the positive samples coordinates as seen in Figure 5. I display some examples of the positive and negative samples in Figure 6. I compute the negative patches through a sliding window of the image and use *get_overlap* method in a loop that iterates through the positive coordinates of every positive patch in the image at every step of the sliding window. The *overlap* is initialized to 0, then the output returned by the *get_overlap* is added to it at every iteration of the loop. The positive coordinates inputted in the *get_neg_patches* come from the output returned by *get_pos_patches* as seen in Figure 4. The *get_overlap* function takes as input the image in question, the negative coordinates of the sliding window, and the positive coordinates for a positive patch as seen in Figure 2. It works by initializing the image pixels to False after cloning it, then setting to True the area that is bounded by the negative coordinates corresponding to the sliding window coordinates. Then, the same is done for the positive patch coordinates. Therefore, I have two images that are set to False everywhere except for the area corresponding to the patches, either negative or positive. A bitwise AND numpy function is then used to compare the two images for overlap. The numpy sum is used on the returned array to compute the number of pixels overlapped. The result is then divided by the sum of pixels of the positive patch to get the proportion of overlap. If overlap is 0, then the sliding window is determined to be a valid negative patch.

ECSE 415 Introduction to Computer Vision

The window size chosen is (64,128) because it fits the best or is the most common when it comes to the matching physical size of human beings on images in the literature of object detections. The step size chosen is 16, as it gives best results with testing, so that the window slides 16 pixels at a time. The training data consists of roughly 12,000 features with about 10,000 being negative samples. The number of training samples required depend on the complexity of the model. As the model is not too complex, and the images are all similar, a great number of samples was not required. I tested with extracting features from 100 images, and got back an accuracy score of 99.28% when testing with features from 5 random images as seen in Figure 12.

```
# Check overlap between two detections of an image function
def get_overlap(img, pred, gt):
    """
    @brief Calculates the pixel overlap between two detections in an image
    @param img: ndarray. The input image where that has the two detections
    @param pred: [x1, y1, x2, y2]. Coordinates for the prediction
    @param gt: [x1, y1, x2, y2]. Coordinates for the ground truth
    @return overlap: Overlap between 0 and 1, where 0 shows no overlap and 1 complete overlap
    """

    # create binary images with pixels set to False/True
    img_pred = np.full((img.shape[0], img.shape[1]), False)
    img_pred[pred[1]:pred[3],pred[0]:pred[2]] = True

    img_gt = np.full((img.shape[0], img.shape[1]), False)
    img_gt[gt[1]:gt[3],gt[0]:gt[2]] = True

    # perform bitwise AND operation to compute overlap
    result = np.bitwise_and(img_gt, img_pred)
    overlap = np.sum(result) / np.sum(img_gt)

    # return the sum of array (number of 1s)
    return overlap
```

Figure 2. *get_overlap* function

```
# Get positive patches function
def get_pos_patches(img):
    """
    @brief Extract the negative samples from an image
    @param img: ndarray. The image to extract positive patches from
    @return pos_patches: list. The negative patches
    @return img_copy: ndarray. Used to display the image with rectangles
    """

    pos_patches = []
    pos_coordinates = []
    outputs,_ = detectron(img) # use the person detector implemented in part 1 to identify the person patches in each image.
    boxes = outputs["instances"].pred_boxes.tensor.tolist() # get list of boundary boxes for person with format array([x1,y1,x2,y2],...)
    for i,person in enumerate(outputs["instances"].pred_classes.tolist()):
        if(person == 0): # if boundary is person (label = 0)
            box = boxes[i] # get boundary box coordinates for person
            x1=round(box[0])
            y1=round(box[1])
            x2=round(box[2])
            y2=round(box[3])
            patch = img[y1:y2, x1:x2] # extract positive patch
            pos_coordinates.append([x1,y1,x2,y2]) # append coordinates to use for negative patches
            pos_patches.append(patch) # append patch to the array of positive patches

    return pos_patches, pos_coordinates
```

Figure 3. *get_pos_patches* function

```
# Iterate to get training samples
for image in images:
    pos_patches, pos_coordinates = get_pos_patches(image)
    pos_patches_list.extend(pos_patches) # append patch to the array of positive patches
    neg_patches, img_disp = get_neg_patches(image, pos_coordinates, step_size=32)
    neg_patches_list.extend(neg_patches) # append patch to the array of negative patches
```

Figure 4. Extract positive and negative patches for images

ECSE 415 Introduction to Computer Vision

```
# Get negative patches function
def get_neg_patches(img, pos_coordinates, step_size=16, window_size=(64, 128)):
    """
    @brief Extract the negative samples from an image
    @param img: ndarray. The image to extract negative patches from
    @param step_size: number. Step size for sliding window. Default: 16
    @param window_size: (w,h). Window size (width,height) for sliding window. Default: (64,128)
    @param pos_coordinates: [x1, y1, x2, y2] list. Positive coordinates to check for overlap
    @return neg_patches: list. The negative patches
    @return img_copy: ndarray. Optional. Used to display the image with rectangles
    """
    neg_patches = []
    img_copy = img.copy() # for display
    for (x, y, window) in sliding_window(img, step_size=step_size, window_size=window_size):
        neg_patch = img[y:y+window_size[1], x:x+window_size[0]]

        overlap = 0 #initialize
        for pos_coordinate in pos_coordinates: #check for every window if there is overlap with positive samples
            overlap += get_overlap(img, [x,y,x+window_size[0],y+window_size[1]], pos_coordinate)

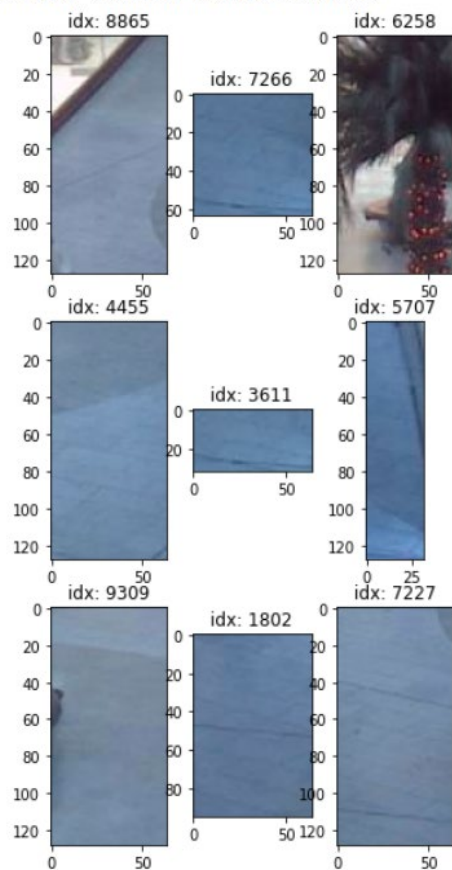
        if(overlap == 0): # no overlap, then extract negative patch and append
            neg_patches.append(neg_patch)

    # for display
    img_copy = cv2.rectangle(img_copy, (x, y), (x + window_size[0], y + window_size[1]), (255, 0, 0), 2)

    return neg_patches, img_copy
```

Figure 5. *get_neg_patches* function

Display 9 random negative samples:



Display 9 random positive samples:

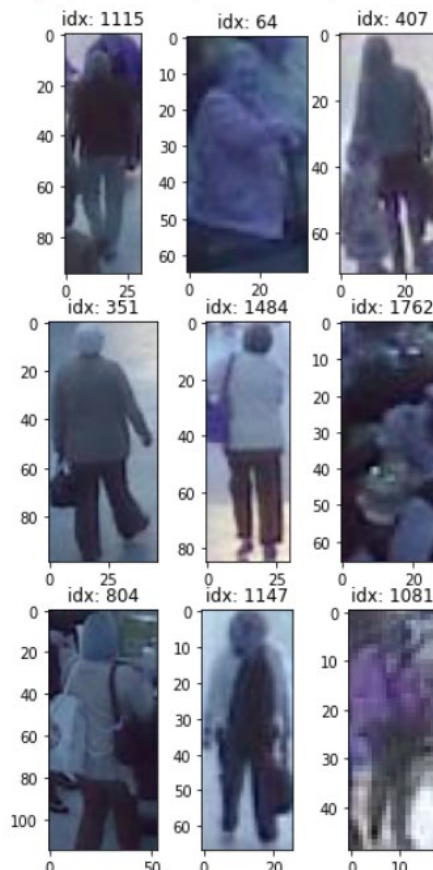


Figure 6. Some examples of positive and negative patches

ECSE 415 Introduction to Computer Vision

Description of the duplicate detection removal and person counting:

Describe the method used to remove duplicate detections in part 2. Briefly describe how the method works. Also describe the code used to count the detections.

The method used to remove the duplicate detections is based on the *nms* function which makes use of *get_iou* function. Before using the Non-Maximum Suppression function (*nms*), I filter out the bounding boxes that have a score less than the confidence level or minimum score required as shown in Figure 7 to improve the duplicate removal process. I compute the IoU metrics for each bounding box compared to the ground truth to get the *scores* array as I will explain below in the next section. I then set an overlap threshold to be used by the *nms* function named *overlap_th*. The *nms* function also takes the *boxes* and *scores* as inputs. If the boxes are empty, then return empty. Otherwise, I compute the indices that would sort an array using *np.argsort* as shown in Figure 9. For example, an array of [5,2,4,3] would return [1,3,2,0]. This is done to keep track of the *boxes* indices and their corresponding *scores* while sorting the array to know the highest scores. Then, I loop through the indices, while appending to *boxes_list* the highest score that is the last index at each iteration. I then compute the IoU between the box with the highest score and the remaining boxes in the list to get the IoU overlap. I then delete all indices from the *indices* list where the overlap is higher than the set threshold for allowed overlap as shown in Figure 9. For example, if there are four bounding boxes with different scores like 80%, 60%, 40%, 20%, then the first step is the filtering step where I disqualify any score that is less than 30% let's say, so we are left with 3 bounding boxes with scores 80%, 60%, and 40%. Now, we take the highest score, that is the 80%, and compare it with the others. If the overlap is more than the threshold set, then the bounding box is deemed to be a duplicate. In that case, let's say the 60% does not overlap enough and the 40% has an overlap that is beyond the threshold set with *overlap_th*, then the bounding box that has the 40% would be deemed a duplicate and would be deleted. Otherwise, the 60% is deemed not to be a duplicate and would correspond to another person.

Finally, once there are no more indices in the *indices* list, that means we have sifted through all scores and bounding boxes and removed duplicates. The *boxes_list* would then contain all the indices of the bounding boxes that are not duplicates and have the one with the highest score for each person. Therefore, as we now have the list of indices of our bounding boxes, it suffices to compute the length of the *boxes_idx* to get *person_count* as shown in Figure 8 to get the count of persons in an image.

```
if (score_max > cfd_lvl): # filter before NMS by setting a confidence level for better results
    pred_boxes_list.append([x,y,x+window_size[0],y+window_size[1]])
    pred_scores_list.append(score_max)
```

Figure 7. Filtering bounding boxes by score from *detect_svm* function

```
# Compute NMS to remove duplicates
boxes_idx = nms(np.asarray(pred_boxes_list), np.asarray(pred_scores_list), overlap_th)
person_count.append(len(boxes_idx))
```

Figure 8. Use of *nms* function in *detect_svm* function

ECSE 415 Introduction to Computer Vision

```
# Compute non-maximum suppression using IoU function
def nms(bboxes, scores, overlap_th=0.3):
    """
    @brief Implement the Non-Maximum Suppression technique to clean up prediction from duplicates.
    @param scores: ndarray. List of scores for boxes
    @param boxes: N x [x1, y1, x2, y2] ndarray. Coordinates for boxes
    @param overlap_th: number. Overlap threshold to suppress the boxes with high IoU with respect to the previously selected one. Default: 0.3
    @return indices of boundary boxes
    """

    # If boxes is empty, return empty list
    if len(bboxes) == 0:
        return []

    # Initialize
    boxes_idx = []

    # Sort the indices by the score
    indices = np.argsort(scores)

    # loop so long as there are still indices in the list
    while len(indices) > 0:

        # Select the index with highest score and append to
        last = len(indices) - 1
        idx = indices[last]
        boxes_idx.append(idx)

        # Get the IoU between the box with highest score and the boxes remained in the indices list
        overlap = get_iou(bboxes[idx], bboxes[indices[:last]])

        # delete all indices from the index list with overlap > overlap_th
        indices = np.delete(indices, np.concatenate([last, np.where(overlap > overlap_th)[0])))

    # return indices of selected boundary boxes
    return boxes_idx
```

Figure 9. NMS function

Evaluation of Person Detector:

Describe how you evaluated the performance of your person detector. Take the ground truth to be the bounding boxes detected in part 1. Use the IoU metric to quantify the performance of the detector. Provide an image showing the detections obtained on at least one image from the dataset.

I evaluated the performance of my person detector system using testing positive and negative patches. They were extracted the same way the training samples were extracted but from new randomly generated images. The accuracy is then calculated as shown in Figure 12, which is a nearly perfect score of 99.28%. When it comes to test the person detections, the size of the sliding window might be larger or smaller than positive and negative patches, so it will not give a perfect prediction. The IoU metrics that quantify the performance of detecting persons using sliding windows can be shown on top of each boundary box as seen in Figure 13. The IoU compares the sliding window to the ground truth, that is the positive patch detected by detectron2. The IoU will not give a nearly perfect match as I use a fixed sliding window of (64,128) whereas the ground truth patches are of different sizes. The IoU function *get_iou* takes as inputs *box* and *boxes* as seen in Figure 11. It measures the intersection over union of one box relative to the other *boxes* based on $\text{Union}(A,B) = A + B - \text{Inter}(A,B)$. In order to get a perfect IoU, the sliding window must perfectly fit or match the ground truth, which does not happen due to the nature of sliding windows and not multi-scaling. As can be seen in Figure 10, the score is computed by inputting the sliding window coordinates as *box* and the ground truth computed by detectron2 named as *boxes_list*. I then take the maximum value of the array

ECSE 415 Introduction to Computer Vision

returned bearing in mind most values inside the array will be 0 except for a few low score values representing neighboring ground truths that might have overlapped with the sliding window and need to be discarded.

```
if(person[0] == 1): # if prediction is person
    score = get_iou([x,y,x+window_size[0],y+window_size[1]], np.asarray(boxes_list))*100 # use IoU metric to score how well detector works compared to ground truth
    score_max = int(round(np.amax(score))) # get the maximum score to get relevant bounding box of ground truth, i.e. discard overlaps from other bounding boxes
```

Figure 10. IoU scores for persons

```
# Get IoU function
def get_iou(box, boxes):
    """
    @brief Implement the Intersection-over-Union (IoU) between box and boxes
    @param box: [x1, y1, x2, y2] ndarray. Coordinates for the first box
    @param boxes: N x [x1, y1, x2, y2] ndarray. Coordinates for the second box
    @return IoU
    """

    # Calculate the [x1, y1, x2, y2] coordinates of the intersection of rectangles
    x1 = np.maximum(box[0], boxes[:, 0])
    y1 = np.maximum(box[1], boxes[:, 1])
    x2 = np.minimum(box[2], boxes[:, 2])
    y2 = np.minimum(box[3], boxes[:, 3])
    inter_areas = np.maximum(x2 - x1, 0) * np.maximum(y2 - y1, 0)

    # Calculate the union area by using the formula: Union(A,B) = A + B - Inter(A,B)
    box_area = (box[2] - box[0]) * (box[3] - box[1])
    boxes_area = (boxes[:, 2] - boxes[:, 0]) * (boxes[:, 3] - boxes[:, 1])
    union_areas = box_area + boxes_area - inter_areas

    # compute the IoU
    iou = inter_areas / union_areas

    return iou
```

Figure 11. IoU function

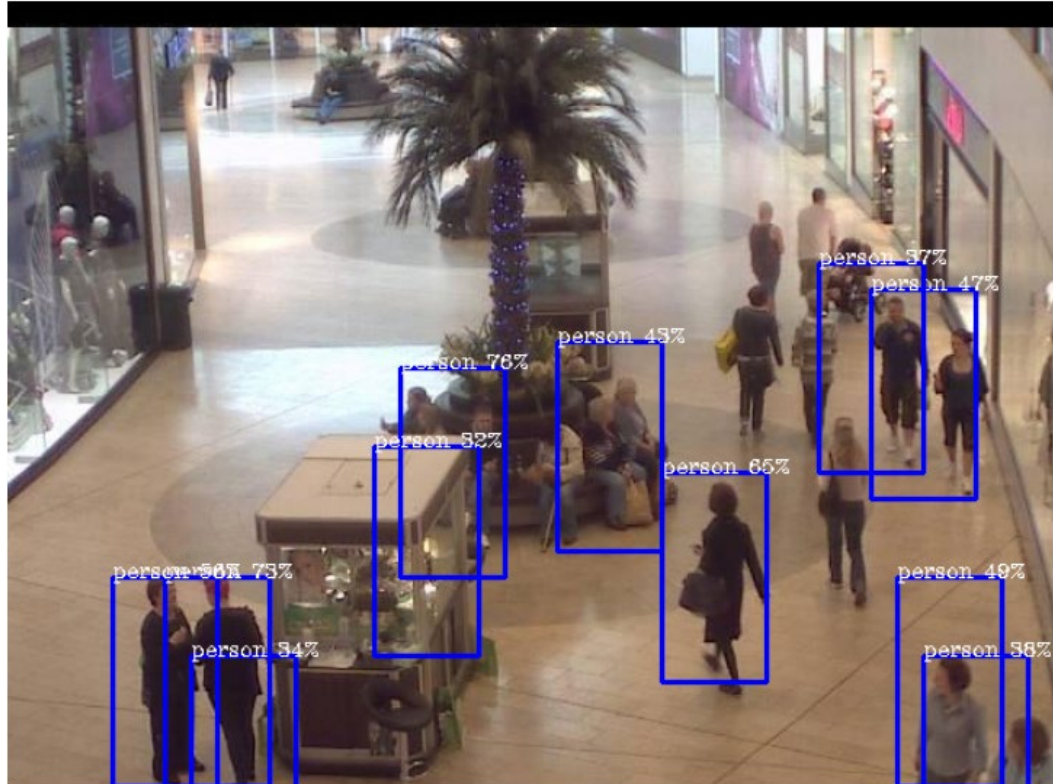
[illegible]

Accuracy: 99.28 %

Figure 12. SVM classifier accuracy

ECSE 415 Introduction to Computer Vision

1 - Display after NMS:



1 - Display before NMS:

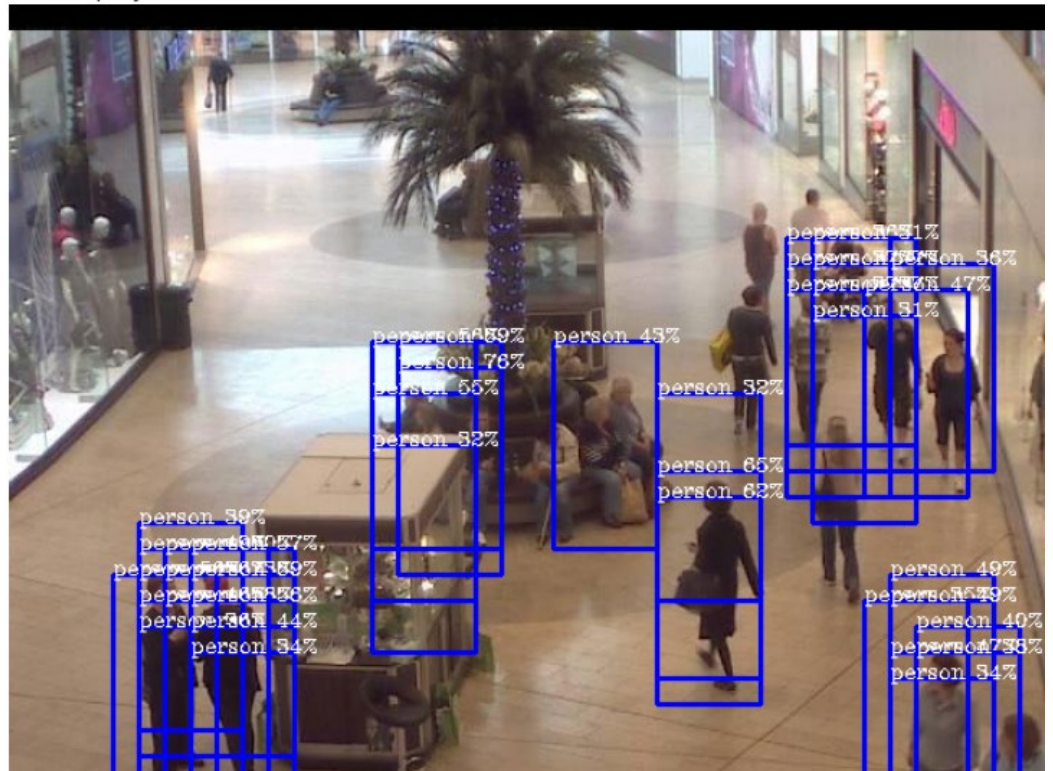


Figure 13. seq_000859.jpg with SVM detections before and after NMS

ECSE 415 Introduction to Computer Vision

Evaluation of Person Counting:

Run the person detector of part 2 on all 1000 images of the dataset. Construct the response spreadsheet and upload it to the Kaggle competition web site. List the score (ranking and metric) reported on the Kaggle web site leaderboard. (note: this will not be the *final* score/ranking, as that will only be reported after the end of the competition period).

I do not have a ranking on the leadership board as I submitted after the competition closed on December 13, 2020 at 6PM, however my rank can be estimated to be 22 using Figure 14. My public score is 0.69097 as shown in Figure 15.































Overview	Data	Notebooks	Discussion	Leaderboard	Rules	Team	My Submissions	Make Submission
#	Δpub	Team Name	Notebook	Team Members	Score	Entries	Last	
1	▲1	Ethical-AI		 	0.00352	2	6d	
2	▲1	Group 12		  	0.10454	8	2d	
3	▲1	Group18- Insert "Team Name"		  	0.11381	11	2d	
4	▲2	Group #04: Les Nordiques		  	0.12917	10	1d	
5	▲2	Group 15		  	0.13371	6	1d	
6	▼1	Group #01 - Paul Blart: Mail Cop		 	0.13866	6	5d	
7	▲1	Group #20 - Stop The Count!		  	0.13901	4	4d	
8	▲2	Group 25 - I've Got My Eye On ...		  	0.16545	5	1d	
9	—	constant - all values=32			0.19820			
9	—	G10 - The engFUNeers		  	0.19889	4	1d	
10	▼9	Group #24 - Astroworld Shop...		  	0.24627	6		
11	—	21-Savage		  	0.25010	1	1d	
12	—	Group 26		  	0.26821	6	1d	
13	—	Group 16		  	0.27760	2	1d	
14	—	Group#7		  	0.29048	1	1d	
15	—	uniform random values			0.33860			
15	—	Group 8		  	0.36083	5	1d	
16	—	Group #02 - Cuatro Vision		  	0.41405	5	2d	
17	—	Group 14			0.49255	1	1d	
18	—	Group 13		  	0.50244	1	1d	
19	—	Group #19 - cv2.imread("tea...		   	0.52304	1	1d	
20	—	G23		  	0.55407	1	5d	
21	—	Group 5		   	0.57939	1	1d	
22	—	Group #32		   	0.71706	1	1d	
23	—	Group 27 - Modern Turing		   	0.75244	2		

Figure 14. Kaggle Leaderboard

1 submissions for [Anas Deis](#)

Sort by

Most recent

All

Successful

Selected

Submission and Description	Private Score	Public Score	Use for Final Score
<div>DeisAnas_submission.csv</div> <div>4 minutes ago by Anas Deis</div> <div>No Group - Anas Deis</div>	0.69549	0.69097	<div><input type="checkbox"/></div> <div>Submission is after deadline</div>
No more submissions to show			

Figure 15. Kaggle Score

ECSE 415 Introduction to Computer Vision

Discussion:

Discuss the lessons learned during this project. How could you improve upon your results? Are there better ways to estimate how many people are in an image than the approach used in this project? Describe any difficulties you faced in the project. Discuss the suitability of different types of features (e.g. HoG vs Haar or LBP or even Deep Features provided by a CNN).

I learnt how to properly use a classifier to train data while using NMS to remove duplicates, and the effect of tuning the variables can be significant, namely the 'C' parameter for SVM, the score and overlap thresholds. The problem with my current code implementation is that I did not optimize it and could be improved significantly as I was short on time. I would have really liked to explore more and implement a scale-space pyramid to my system able to localize and detect objects in images at multiple scales and locations. The system has a hard time detecting people that were a bit far (on top of the images) or children due to the sliding window being fixed at (64,128) without rescaling. I faced difficulties when it comes to determining the appropriate sizes for the windows size, step size, the HoG parameters, etc. for best results. In a comparative study dealing with frontal face detection using video sequences, it proved that the hog face detector was the most accurate out of the three methods: (1) Haar-like cascade, (2) Histogram of Oriented Gradients (HoG) with Support Vector Machine and (3) Linear Binary Pattern cascade (LBP). It achieved the highest detection rate (92.68%). The Haar face detector achieved the second-best detection rate almost 14% inferior to the HoG face detector. The LBP face detector achieved the worst detection rate which was 32% inferior to the HOG face detector. This approach was very fast and not accurate. It detected many false positives during the test [2]. Therefore, HoG+SVM would be the most accurate and robust method when it comes to the detecting persons, yet slower than LBP.

References

- [1] S. Joseph, and A. Pradeep. "Object Tracking using HOG and SVM," in International Journal of Engineering Trends and Technology (IJETT), vol. 48, no. 6, 2017. [Accessed Dec. 14, 2020]
- [2] A. Adouani, W. M. Ben Henia and Z. Lachiri, "Comparison of Haar-like, HOG and LBP approaches for face detection in video sequences," *2019 16th International Multi-Conference on Systems, Signals & Devices (SSD)*, Istanbul, Turkey, 2019, pp. 266-271, doi: 10.1109/SSD.2019.8893214.[Accessed Dec. 14, 2020]