uOttawa

CSI 4142 - Traitement d'images

Project Part 3

Winter 2024

Due date: April 10, 2024

Detection of Persons in Images Using Histogram Comparison

Selin Kararmaz (300163876)

Kanjanokphat Kitisuwanakul (300170040)

Link to GitHub Repository

https://github.com/SelinKararmaz/CSI4533_Projet_Part_3

Description

The goal of this project is to correctly identify 5 different people in a long series of images. The 5 people will be identified using 5 test images given. The algorithm will run through 2 folders of images (cam0 and cam1) and identify each of these 5 people in the set of images.

Inputs

Here are the 5 people that we want to identify:

Figure1: 5 test images showing persons 1,2,3,4,5 respectively

The images that will be analyzed are stored in the folders cam0 and cam1, which contain a total of 831 images each.

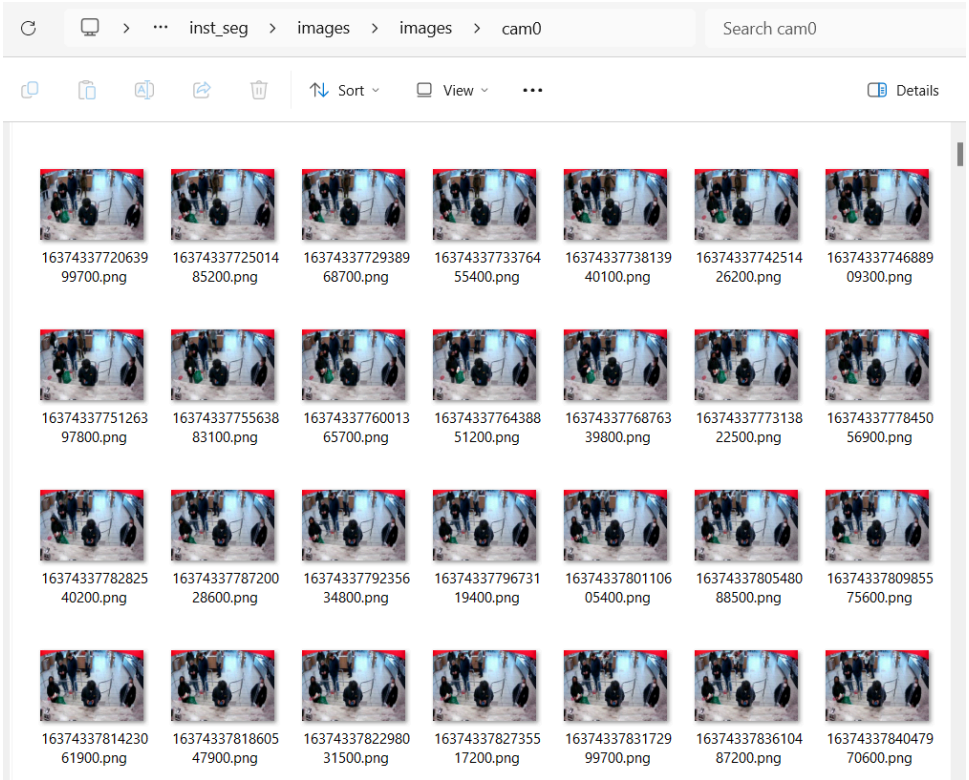


Figure 2: Screenshot showing contents of folder cam0

Methodology

In part one of this project, we used the input text file *labels.txt* to get the bounding boxes of each person in cam0 so as to compare them with our 5 sample persons.

In this part, we will be using **instance segmentation**, which has been implemented by the teaching assistant Fahed Hassanat. This method uses convolutional neural networks to effectively detect persons and segment them from the background.
The output of the algorithm are masks representing each person that has been segmented from each image.

We first run the instance segmentation on cam0 and cam1 to get a **numpy file** for each image that detects each person in the image. Note that we will have 831 numpy files for each cam folder. A total of 831*2 = 1662 numpy files for both cam0 and cam1.
These masks will be our input *labels.txt* for the detection of persons in cam0 and cam1.

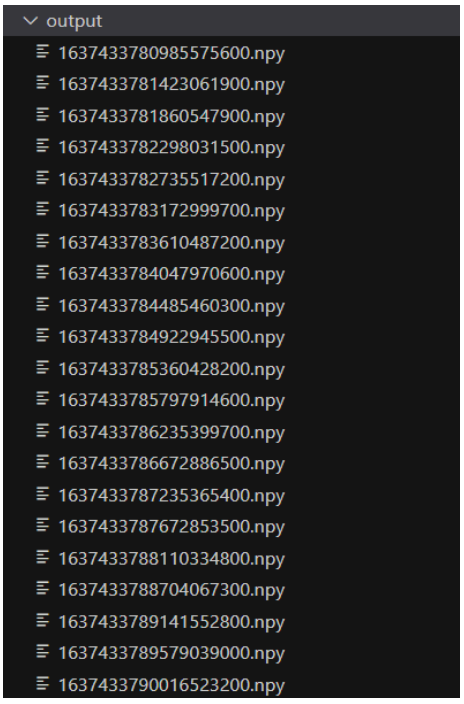


Figure 3: Example of all the numpy masks, one for each image

Below is an example of one numpy file for an image, depicting how instance segmentation works:

Figure 4: Instance segmentation detecting the people in an image and applying a red mask on it.

Once we have all the numpy masks, we have all the persons **detected**. We will be using the **Histogram Comparison** algorithm from Part 1 to correctly **identify** each of the 5 persons.

To be able to calculate the histogram of each detected person in the image, we have to apply the saved mask to each person in the image, here is an example of how a person detected from the image is extracted using the mask:

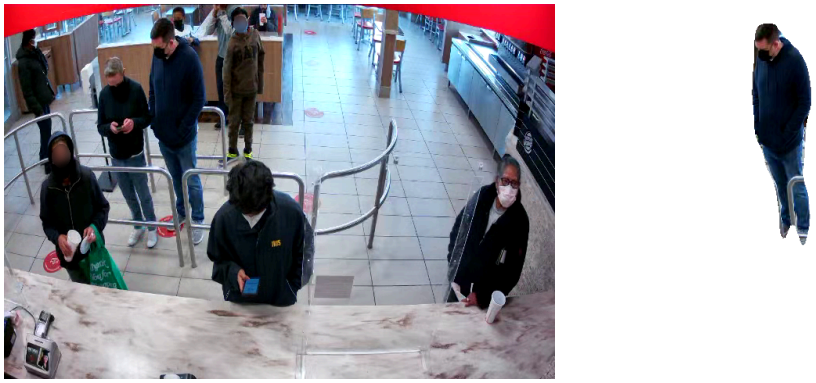


Figure 5: Extracting a person by applying the mask to the image

Implementation

The steps of the algorithm for histogram comparison is detailed below:

Pseudocode:

1. Load the image of each person we have to identify, loop starting from person 1 to person 5
2. Get the full and half histograms of the current person we are comparing. Let's say this is person 1.
3. Loop through all the images from cam0 and cam1, and for each image:
    a. Load the numpy file for this image and apply the mask to the image (reject masks that are smaller than a threshold). Note that the masks have been obtained from the instance segmentation run previously.
    b. Calculate the full and half histograms of these people
    c. Compare 4 histograms with person1 that we have to identify
    d. Keep the best match to person 1 in the image, using histogram correlation. Reject correlations that are less than a defined threshold.
    e. Draw the bounding box of that person on the image
    f. Save the image with the drawn bounding box as the best match to person 1 for this image
    g. Repeat for all images to identify person1 in all these images (loop step 3)
4. Repeat for all persons to identify all 5 persons in all images
5. This will result in 5 folders with images that identify each of the 5 persons.

This algorithm is run for both cam0 and cam1.

Code Explained

Steps 1 and 2: Load the 5 persons to identify, and calculate their full and half histograms

```
for index in range(1,6):
    image = cv.imread("../images/images/five_people" + "/person_" + str(index) + ".png")
    image = cv.cvtColor(image, cv.COLOR_BGR2RGB)
    image_half = crop_image_half(image)
    Image.fromarray(image.astype(np.uint8)).show()
    person_full = get_rgb_histogram(image)
    person_half = get_rgb_histogram(image_half)
```

Step 3a: Load the numpys for each image, and apply the mask on the image

```python
for image_name in images:

    image_path = os.path.join(source_path_dir, image_name)
    image = Image.open(image_path)

    masked = apply_saved_mask(image,1000,image_name[:-4], cam)
    result = masked[0]
    result_half = masked[1]
```

```python
def apply_saved_mask(image, threshold, image_name, cam):

    # Load mask from numpy file and apply
    masks = np.load(cam + '_npy/'+image_name+'.npy')
    img_np = np.array(image)
    people = []
    people_half = []

    # Go through all the masks
    for i, mask in enumerate(masks):

        # Only keep masks above the threshold
        if(np.count_nonzero(mask)< threshold): continue

       # create an empty array to store the result
        masked_region = np.zeros((*img_np.shape[:2], 4), dtype=np.uint8)

        # Apply the mask to the original image to retain the red mask colored region
        # this creates an image of a person only, with the background being black
        for c in range(3):
            masked_region[:, :, c] = np.where(mask, img_np[:, :, c], 0)

        # Set the alpha channel of the resulting mask to be transparent where the mask is 0 (multiplies mask by image)
        # this creates image of the person, with the background being transparent
        masked_region[:, :, 3] = (mask * 255).astype(np.uint8)
```

Here, we use a **threshold** to reject the applied masks that are too small. We first count the number of 1's in the applied mask. This represents the area that the image is in. The counts that are less than a certain threshold, here 1000, are rejected:

```python
# Only keep masks above the threshold
if(np.count_nonzero(mask)< threshold): continue
```

Note that we **apply the mask to the image** by only saving the area of the image where the mask is coloured red, and applying alpha channel to everywhere else for transparency:

```python
for i, mask in enumerate(masks):

    # Only keep masks above the threshold
    if(np.count_nonzero(mask)< threshold): continue

   # create an empty array to store the result
    masked_region = np.zeros((*img_np.shape[:2], 4), dtype=np.uint8)

    # Apply the mask to the original image to retain the red mask colored region
    # this creates an image of a person only, with the background being black
    for c in range(3):
        masked_region[:, :, c] = np.where(mask, img_np[:, :, c], 0)

    # Set the alpha channel of the resulting mask to be transparent where the mask is 0 (multiplies mask by image)
    # this creates image of the person, with the background being transparent
    masked_region[:, :, 3] = (mask * 255).astype(np.uint8)
```

Each person from an image is identified, extracted and saved in **2 arrays**: one for the full image, and another for the half of the image. Note that the bounding box of the extracted image needs to be calculated so that the half image can be found.
Return arrays *people* and *people_half* which contain the all detected people in one image.

```
    # Calculate bounding box of mask
    non_zero_indices = np.argwhere(mask)
    min_row, min_col = np.min(non_zero_indices, axis=0)
    max_row, max_col = np.max(non_zero_indices, axis=0)

    # Calculate the midpoint of the bounding box
    mid_row = (min_row + max_row) // 2

    # Get the top half of the masked region
    cropped_mask = masked_region[min_row:mid_row, :, :]

    # Append the masked image to the list
    people.append(masked_region)
    people_half.append(cropped_mask)

return people,people_half
```

Steps 3b, 3c, 3d: Using these 2 returned arrays from the previous step, we will perform **histogram comparison** to identify the first person among all the detected images.
The code below calculates the histograms of the detected images, compares it with the histogram of the person (person1 in the first loop), then keeps only the best match.

```
for i in range(len(result)):
    person_in_image = result[i]
    person_in_image_half = result_half[i]

    person_in_image_histogram = get_rgba_histogram(person_in_image)
    person_in_image_histogram_half = get_rgba_histogram(person_in_image_half)

    difference_1 = compare_correlation(person_full,person_in_image_histogram)
    difference_2 = compare_correlation(person_half,person_in_image_histogram)
    difference_3 = compare_correlation(person_full,person_in_image_histogram_half)
    difference_4 = compare_correlation(person_half,person_in_image_histogram_half)

    local_max = max(difference_1,difference_2,difference_3,difference_4)

    if(local_max == difference_1 or local_max == difference_2):
        if(smallest_dif < local_max):
            smallest_dif = local_max
            closest_person = person_in_image

    if(local_max == difference_3 or local_max == difference_4):
        if(smallest_dif < local_max):
            smallest_dif = local_max
            closest_person = person_in_image
```

Note that we use a **threshold** to reject identifications that have a correlation less than a defined value. Custom threshold is used for each person to ensure optimum results.

```
thresholds_0 = {
    1: 0.92,
    2: 0.992,
    3: 0.91,
    4: 0.70,
    5: 0.98
}
thresholds_1 = {
    1: 0.92,
    2: 0.993,
    3: 0.88,
    4: 0.65,
    5: 0.977
}
```

```
if(smallest_dif < thresholds[index]):
    continue
```

Step 3e, 3f: Draw the bounding box of the identified person in the image and save the resulting image in folder of outputs:

```
bounding_box_image = get_bounding_box(closest_person, image)

output_folder_path = "../output/person_" + str(index) +"/"+image_name

bounding_box_image.save(output_folder_path)
```

Tools and Helper functions

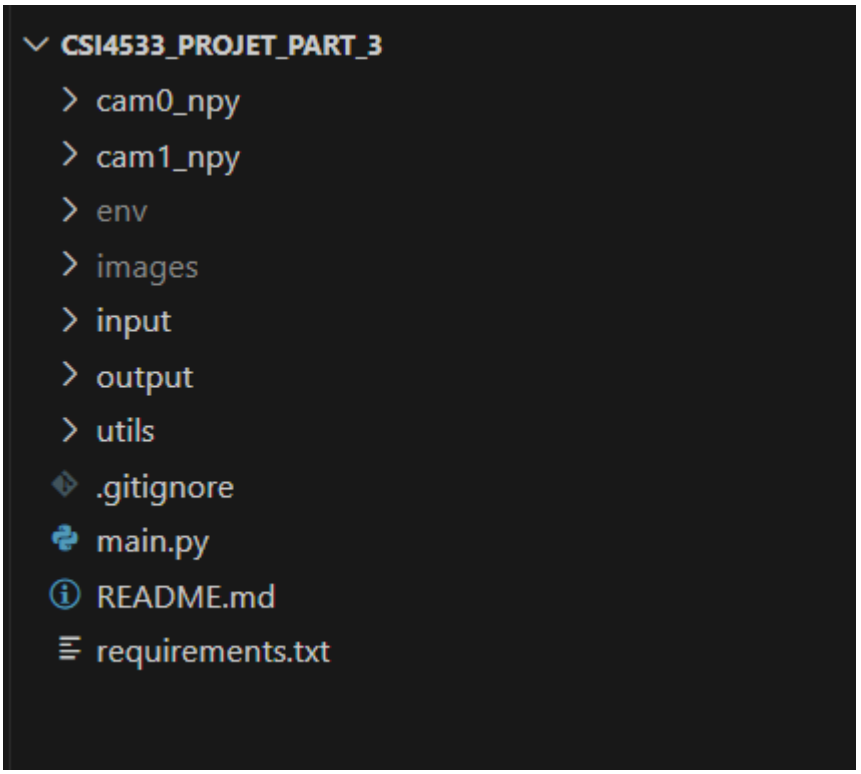
Function to calculate histogram of an image with alpha channel values:

```python
import cv2 as cv

# Number of bins for each channel
b_bins = 8
g_bins = 8
r_bins = 8

# Range for each channel
b_range = [0, 256]
g_range = [0, 256]
r_range = [0, 256]
ranges = b_range + g_range + r_range  # Concatenate lists

def get_rgb_histogram(image):···

def get_rgba_histogram(image):

    ranges = [0, 256, 0, 256, 0, 256]
    b, g, r, a = cv.split(image)

    # Create a mask for opaque regions (where alpha is not 0)
    mask = (a > 0)

    # Apply the mask to the RGB channels
    b = b[mask]
    g = g[mask]
    r = r[mask]

    # Combine the histograms for each channel using the predefined bins and ranges
    hist_rgb = cv.calcHist([b, g, r], [0, 1, 2], None, [b_bins, g_bins, r_bins], ranges, accumulate=False)

    # Normalize the histogram
    cv.normalize(hist_rgb, hist_rgb, alpha=0, beta=1, norm_type=cv.NORM_MINMAX)

    return hist_rgb
```

Function to draw the bounding box of a mask on an image:

```python
def get_bounding_box(mask, image):
    image = np.asarray(image)

    # Extract the alpha channel from the mask
    alpha_channel = mask[:, :, 3]

    # Find non-zero indices in the alpha channel
    non_zero_indices = np.argwhere(alpha_channel)

    # Calculate bounding box coordinates
    min_row, min_col = np.min(non_zero_indices, axis=0)
    max_row, max_col = np.max(non_zero_indices, axis=0)

    # Draw the bounding box on the image
    bounding_box_image = cv.rectangle(image.copy(), (min_col, min_row), (max_col, max_row), (0, 255, 0), 2)

    return Image.fromarray(bounding_box_image)
```

Folder Structure

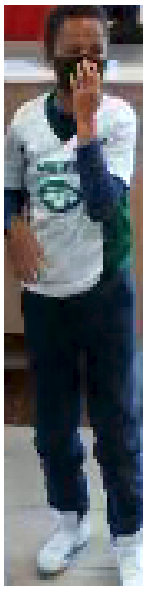


- cam0_npy and cam1_npy contain numpy mask outputs of the function save_numpy(). Each folder has 831 numpy masks corresponding to all images in cam0 and cam1
- env: python virtual environment
- images: folder unzipped from google drive link that contains all the images in cam0 and cam1 for identification
- Input: contains the 5 images of people to identify
- Output: will contain the outputs of best matches identifying the 5 people in cam0 and cam1
- Utils: helper functions
- main.py: entrypoint of our program

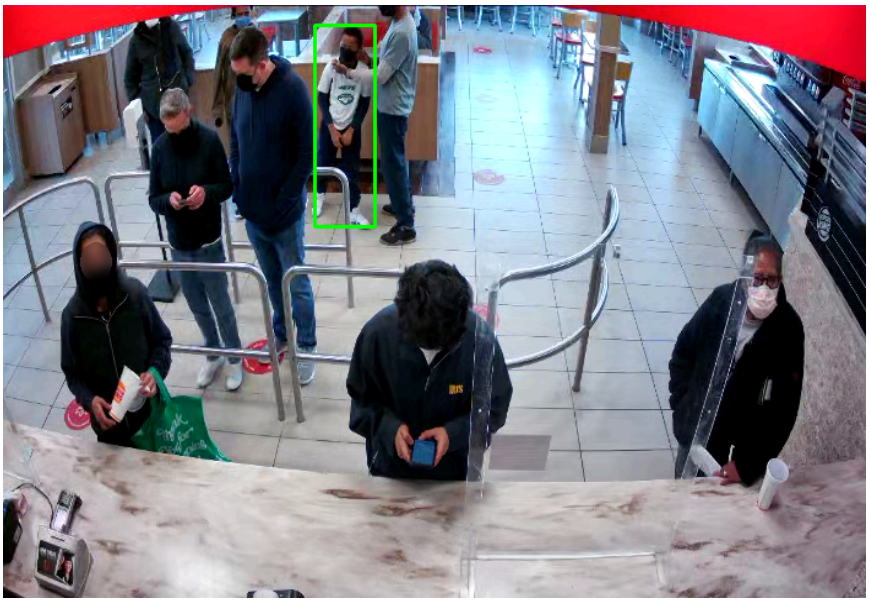
Results & Analysis

After running the algorithm as explained above, here are our results based on running the algorithm on cam0. Very similar results can also be seen for the sequence cam1, for this reason we will not be documenting them here. The full results can be seen in *output\cam1_output*

Person 1



Person 1



Person 1 identified in image 1637433783172999700.png

Figure 6: Sequence of images where person 1 is being identified for cam 0

Person 2

Note that person 2 only appears very late in the sequence of pictures.

Person 2

Person 2 identified in image 1637434125308203000.png

1637434123995747000.png

1637434124433230700.png

1637434124870715800.png

1637434125308203000.png

1637434125745684100.png

1637434126183171000.png

1637434126698781600.png

1637434127136266900.png

1637434127573749800.png

Figure 7: Sequence of images where person 2 is being identified for cam 0

1637434134136031400.png

1637434134573516500.png

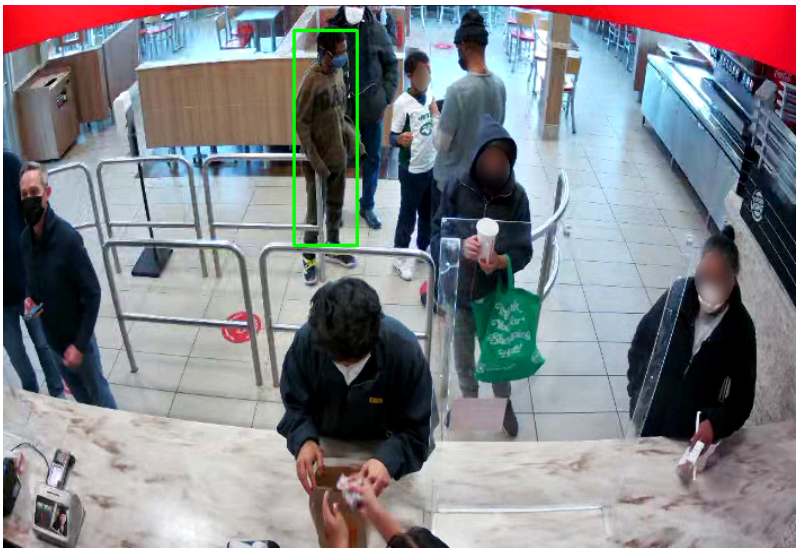
1637434151010466800.png

1637434153197895000.png

Some wrong inputs…person 2 was not correctly identified in the last 2 images for example

Person 3

Person 3



Person 3 identified in image 1637433818968679700.png

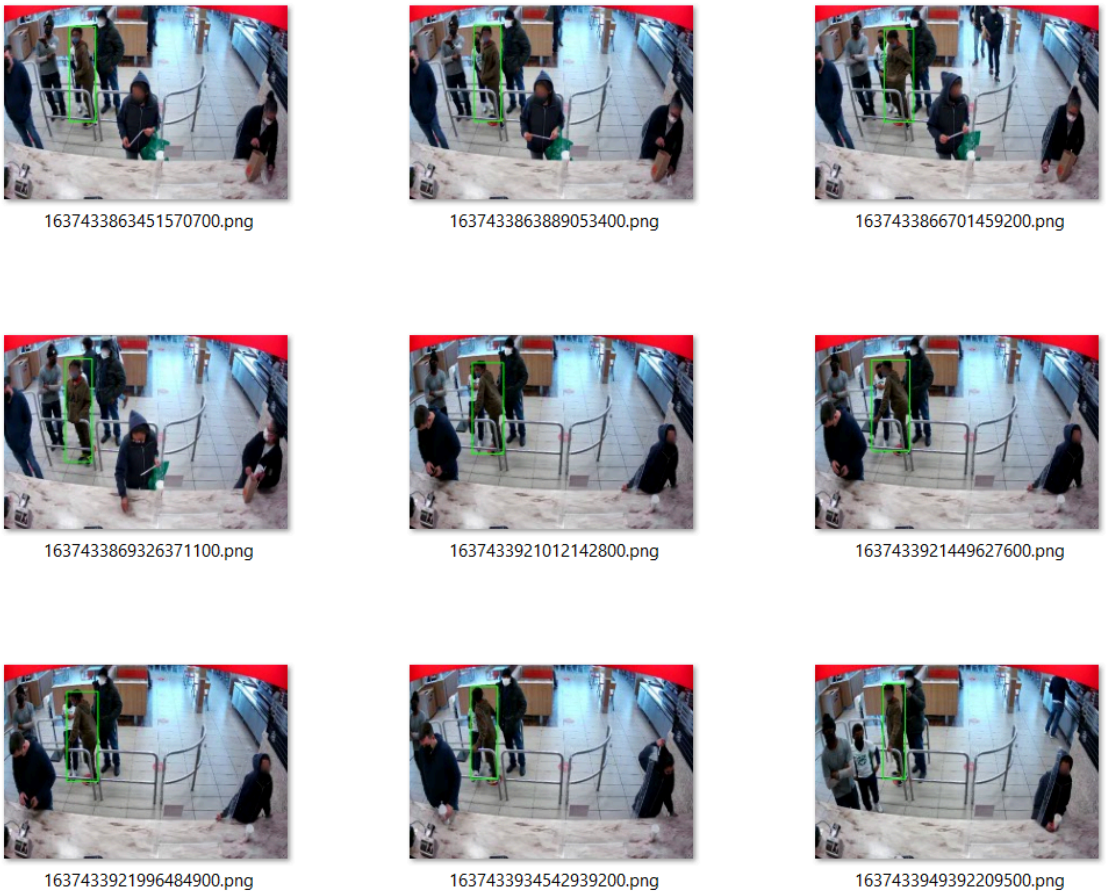
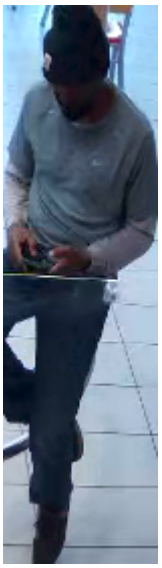
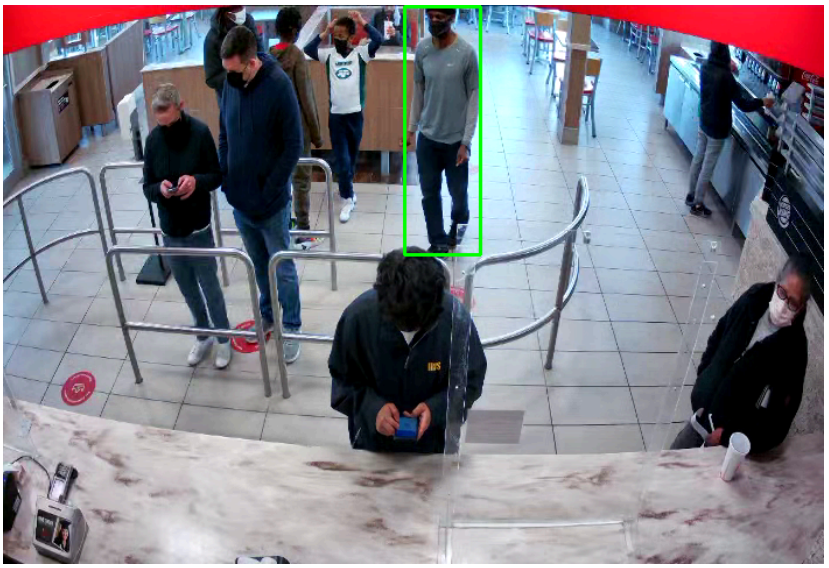


Figure 8: Sequence of images where person 3 is being identified for cam 0

Person 4



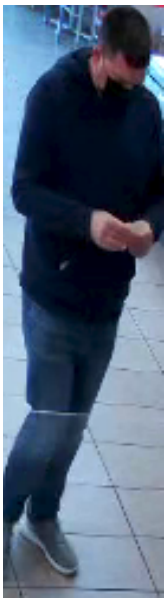
Person 4



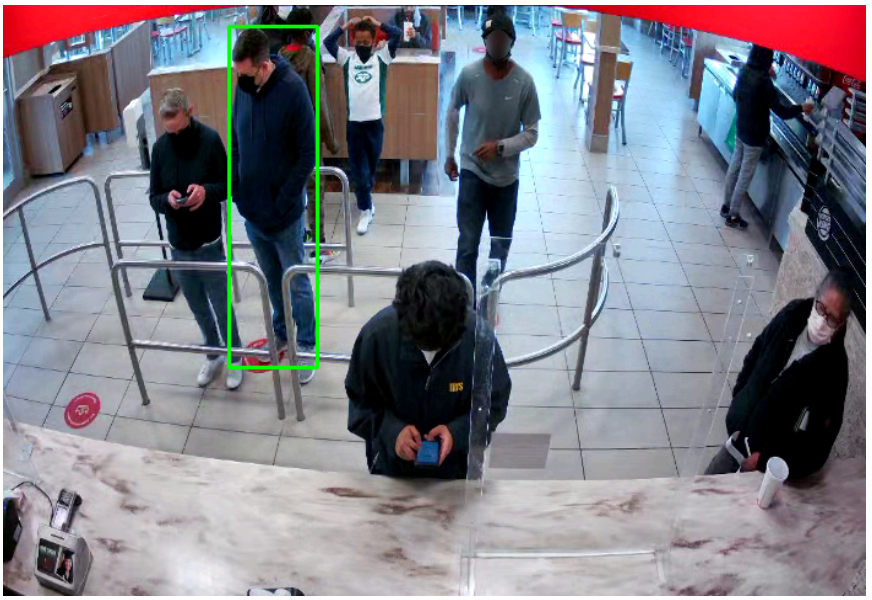
Person 4 identified in image 1637433800938033000.png

Figure 9: Sequence of images where person 4 is being identified for cam 0

Person 5



Person 5



Person 5 identified in image 1637433802266113900.png

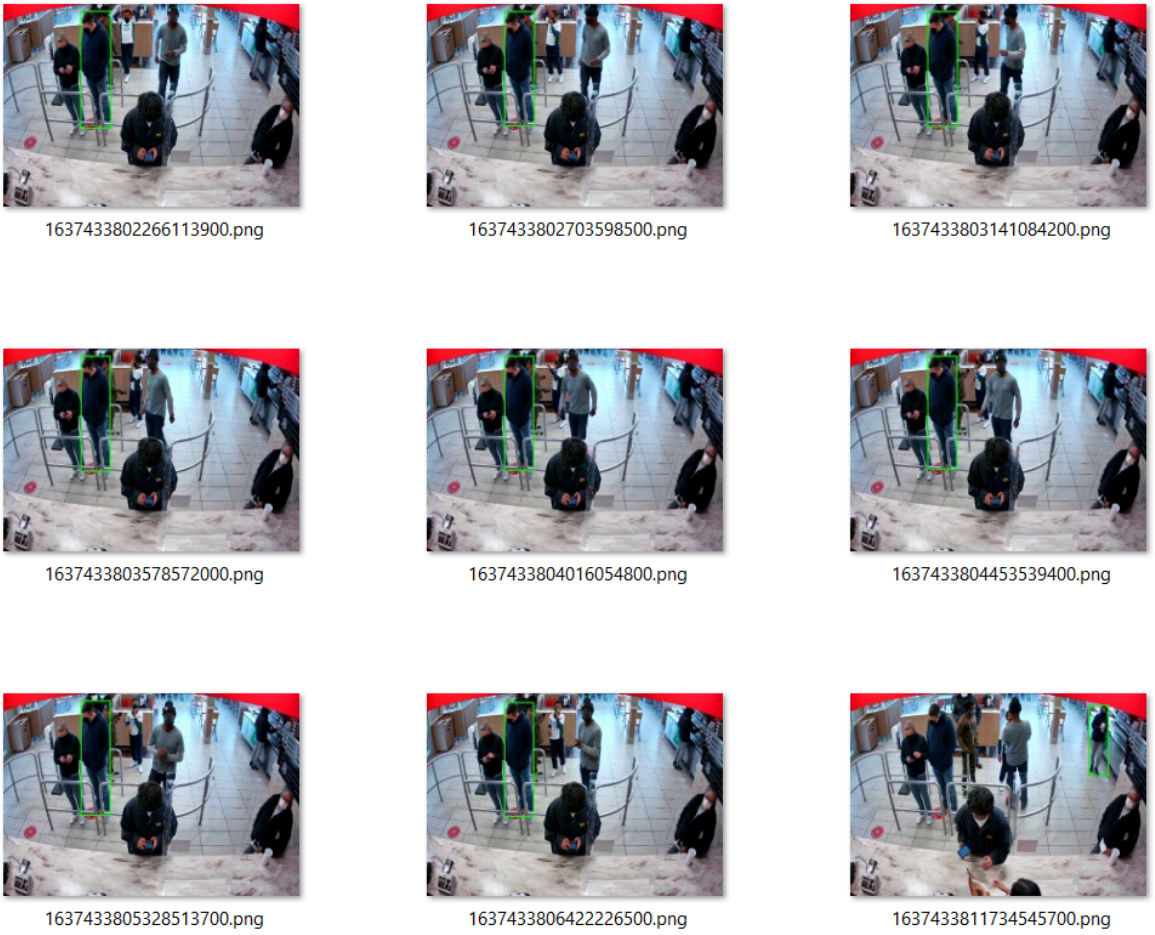


Figure 10: Sequence of images where person 5 is being identified for cam 0

Conclusion

After analyzing the results, we have concluded that **person 1** and **person 3** are the most well identified. This is due to them wearing contrasting colors from the other persons (person1 is mostly in white and person 3 is in brown). The other people have a lot of black and blue, which makes them harder to identify, as a lot of people wear black in the images.

One possible improvement would be to use the mask of the 5 people to compare to all the images, instead of using the whole image as shown in Inputs.

We have summarized the success rate of our algorithm below:

Success rate of algorithm

The table below shows the percentage of images that correctly identified the 5 people. This counting process has been done manually.

Cam0

Person	Correlation threshold kept	Total number of output images	% Success	Notes
1	0.92	237	100%	Person 1 appears in all the 237 images
2	0.992	321	34%	Person 2 only appears in 121 images, and only 116 of these images are correctly identified
3	0.91	91	100%	Person 3 appears in all the 91 images
4	0.70	74	85%	Person 4 appears in all 74 images
5	0.98	136	94%	Person 5 appears in almost all the 136 images. In some, he is a bit cut off on the side.

Table1: Percentage success for each person in cam0

Cam1

Person	Correlation threshold kept	Total number of output images	% Success	Notes
1	0.92	125	98.4%	Person 1 appears in all the 125 images
2	0.993	375	86.3%	Person 2 appears towards the end (in 158 images) since he comes into the footage later on.
3	0.88	72	98.6%	Person 3 appears in all 72 images
4	0.65	98	75.5%	Person 4 appears in all 98 images but often gets confused with the person next to him
5	0.977	99	71.3%	Person 5 appears in almost all the 99 images. In some, he is a bit cut off on the side.

Table2: Percentage success for each person in cam1