HACETTEPE UNIVERSITY

AIN433

ASSIGNMENT 2

Hüseyin Eren DOĞAN – 2210765009

INDEX

1 – Part 1: Edge Detection and Hough Circle Transform

1.1 – Edge Detection

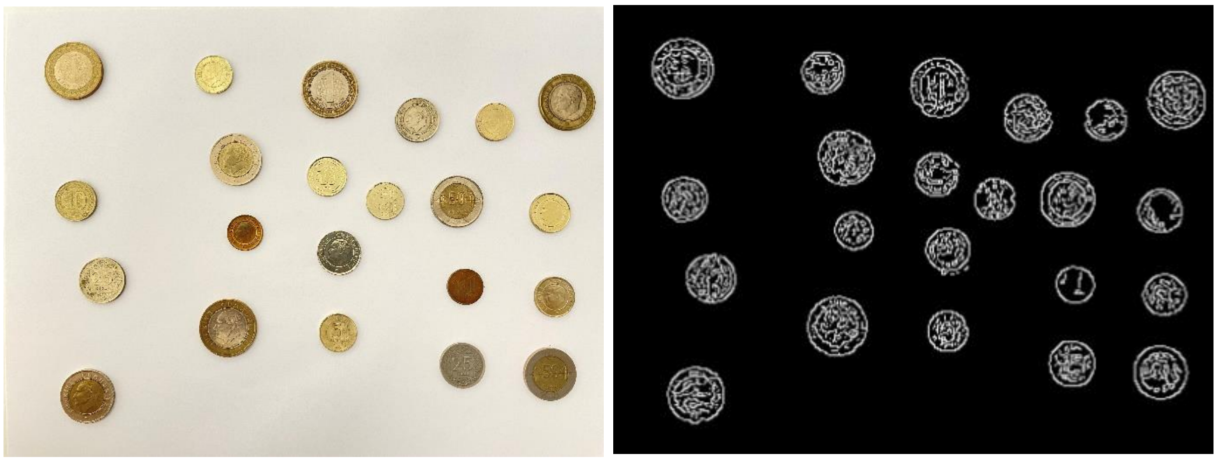
The Canny edge detector was chosen for its well-established reputation in accurately detecting edges. Its advantages include robust noise reduction, precise localization of edges, and minimal response to false edges.

The edge detection parameters were set with low Canny thresholds (20 and 60) to enhance the detection of edges, particularly in the context of identifying circles. Non-maximum suppression was employed to handle multiple edge responses effectively, ensuring an optimal outcome in detecting circles.

The code snippet where Canny edge detector applied have shown below:

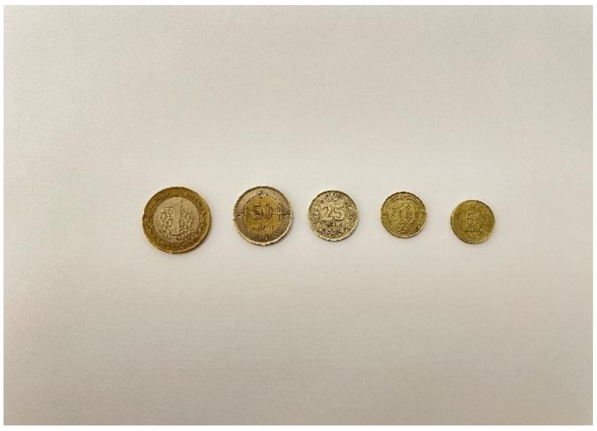
```
# Function to do all the process for images from reading to writing the output
def process_images(input_folder, output_folder, min_radius, max_radius):
    #...
    for filename in os.listdir(input_folder):
        #...
        # Applying canny edge detection
        image_gray = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)
        image_blur = cv2.GaussianBlur(image_gray, (3, 3), 0)
        image_canny = cv2.Canny(image=image_blur, threshold1=20, threshold2=60)
        #...
```

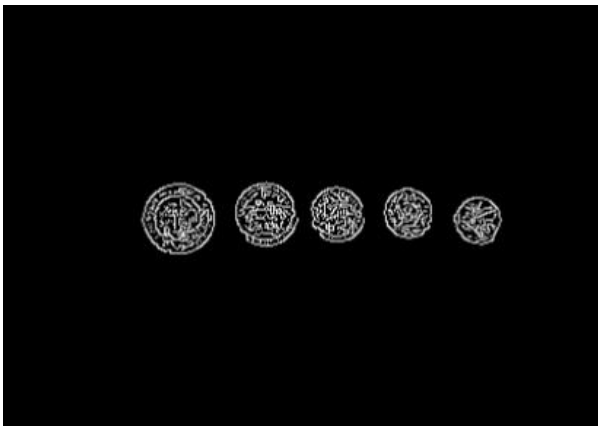
A few results of Canny edge detector:



Original image　　　　Detected circles

(Figure 1)


Original image


Detected circles

(Figure 2)

1.2 – Circle Detection with Hough Transform

The Hough transform algorithm is employed for the detection of geometric shapes, including circles. In this process, image points are transformed into a parameter space, facilitating the identification of shapes through peak detection. Circles are detected by representing them in polar coordinates, with the radius and center being the parameters of interest. The algorithm is utilized for its ability to robustly identify circular patterns, making it suitable for applications such as image analysis, computer vision, and object recognition.

Here are the steps of the circle detection using the Hough transform:

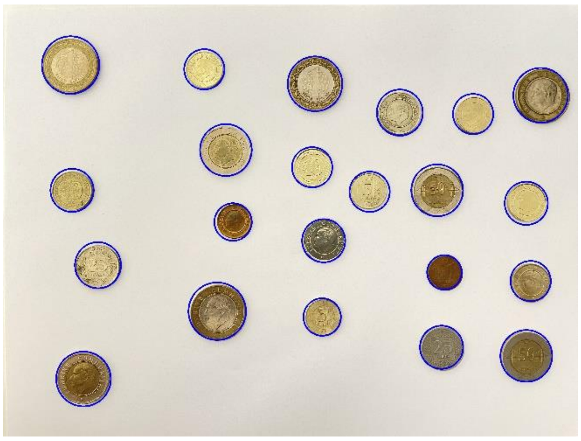
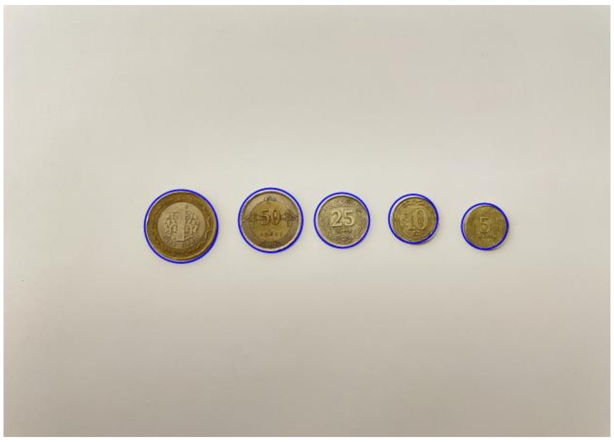
- **Edge Detection:** Begin by applying an edge detection algorithm, such as the Canny edge detector, to identify potential edge points in the image.
- **Hough Transform Accumulator Initialization:** Create an accumulator array, typically a 2D array representing the parameter space for circles. The dimensions of the array correspond to the possible circle centers and radii.
- **Voting:** For each edge point in the image, vote in the accumulator array for possible circle parameters. Increment the corresponding cells in the accumulator based on the possible circle centers and radii.
- **Peak Detection:** Identify peaks in the accumulator array. Peaks represent potential circles in the image. This step involves setting a threshold to distinguish significant peaks from noise.
- **Circle Parameter Extraction:** Extract the parameters (center coordinates and radius) corresponding to the identified peaks. These parameters represent the detected circles in the image.
- **Post-processing:** Optional post-processing steps may include filtering out duplicate or closely located circles and refining circle parameters for improved accuracy.
- **Visualization:** Visualize the detected circles on the original image to assess the performance of the algorithm.

The implementation of the Hough Circle Transform algorithm:

```python
# Function to perform hough circle transform
def hough_circle_transform(image, min_radius, max_radius, threshold):
    # Getting the height and width of the image
    height, width = image.shape

    # Initializing an accumulator array with dimensions
    accumulator = np.zeros((height, width, max_radius - min_radius + 1))

    # Precomputing the cosine and sine values for each angle
    cos_theta = np.cos(np.radians(np.arange(0, 360)))
    sin_theta = np.sin(np.radians(np.arange(0, 360)))

    # Looping for each possible radius value
    for r in range(min_radius, max_radius + 1):
        # Looping for each possible angle value
        for theta in range(0, 360):
            # Calculating the possible center coordinates of the circle for each edge pixel
            a = np.around(np.nonzero(image)[0] - r * cos_theta[theta]).astype(int)
            b = np.around(np.nonzero(image)[1] - r * sin_theta[theta]).astype(int)
            # Checking if there is any center coordinates that are outside the image
            valid = np.where((a >= 0) & (a < height) & (b >= 0) & (b < width))
            # Incrementing the accumulator values at the valid center coordinates
            accumulator[a[valid], b[valid], r - min_radius] += 1

    # Initializing a list to store the detected circles
    circles = []

    # Looping for each pixel in the accumulator array
    for x in range(height):
        for y in range(width):
            for r in range(max_radius - min_radius + 1):
                if accumulator[x, y, r] >= threshold:
                    circles.append((x, y, r + min_radius))

    # Return the list of detected circles
    return circles
```

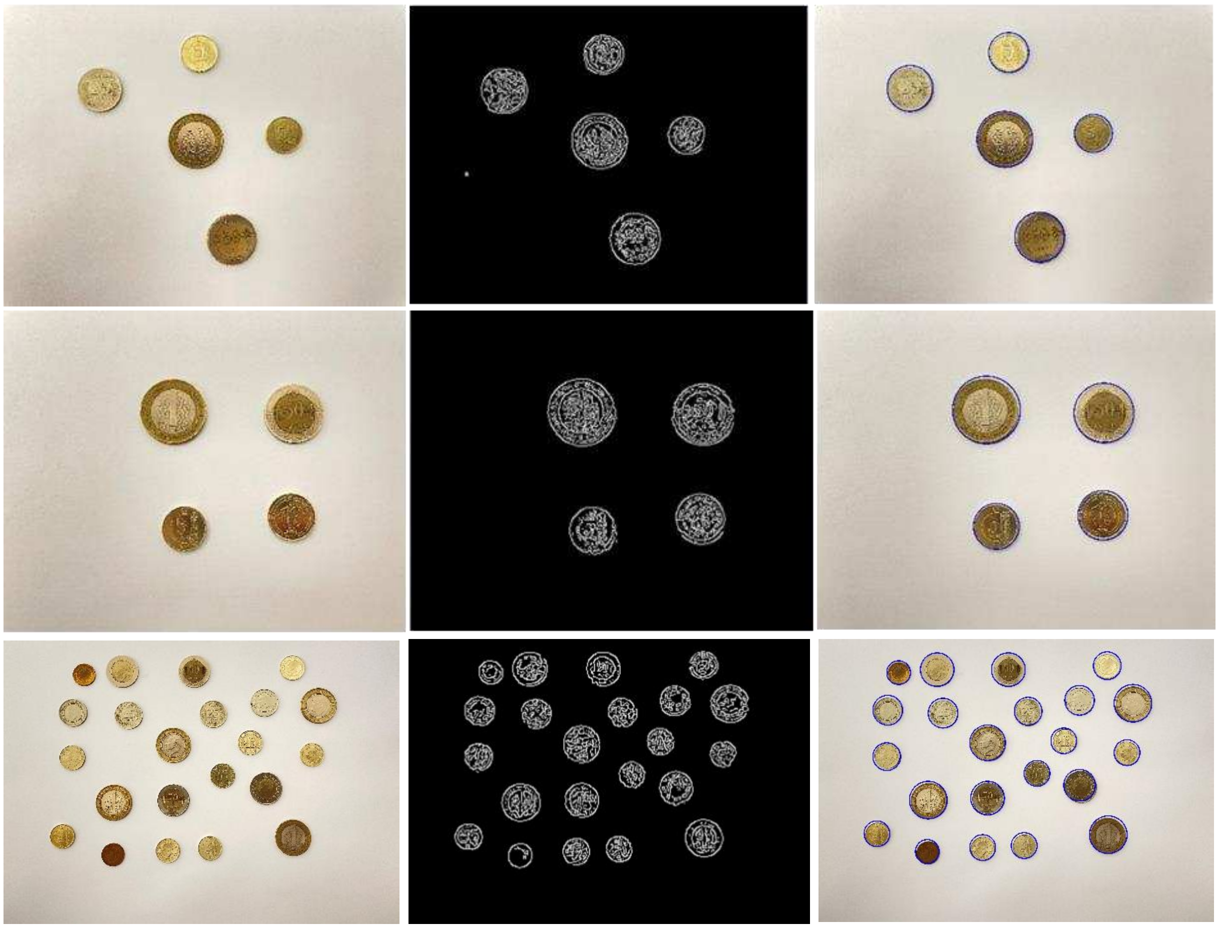
The images were subjected to the application of this algorithm, followed by the application of non-maximum suppression to precisely identify one circle for each coin. Then the circles were found successfully.

Results of images in Figure 1 and Figure 2:

Some other results of circle detection have shown below:

Original image Detected edges Detected circles



2 – Part 2: Histogram of Gradients

2.1 – Histogram of Gradients

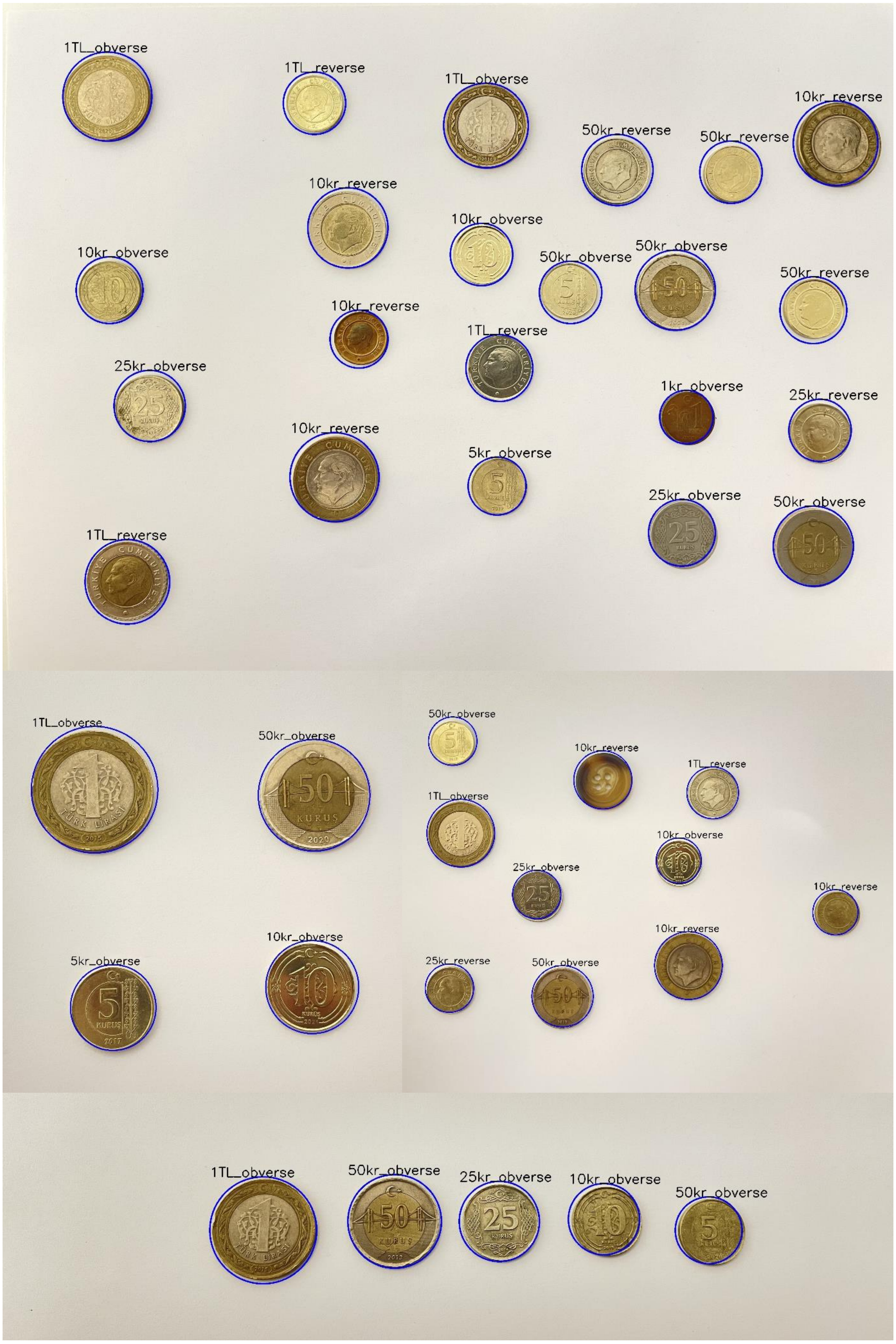
The Histogram of Gradients (HOG) algorithm is employed for feature extraction in image processing. In this method, gradients of pixel intensities are computed, and their orientations are quantized into histogram bins. The histograms serve as descriptors capturing local patterns and gradients in the image. The algorithm is widely utilized for object detection and recognition due to its effectiveness in encoding shape and texture information.

The steps of algorithm are:

- **Gradient Computation:** Gradients are computed for each pixel in the image to capture information about the intensity variations and edges.
- **Gradient Magnitude and Orientation Calculation**: The magnitude and orientation of the gradients are calculated for each pixel, providing a representation of local image structure.
- **Cell Division**: The image is divided into cells, and gradient information within each cell is accumulated to capture localized patterns.
- **Histogram Binning**: Histograms of gradient orientations are created by binning gradient orientations into predefined bins within each cell.
- **Block Normalization**: Normalization is applied to groups of cells known as blocks, facilitating robustness against variations in lighting and contrast.
- **Descriptor Formation**: The final feature descriptor is formed by concatenating the normalized block histograms, creating a representation that is invariant to changes in illumination and contrast.
- **Object Detection**: The HOG descriptors can be used for object detection by training a classifier on these features, making the algorithm valuable for tasks such as pedestrian detection in images.

In this code, the orientation calculation was omitted in the second article, considering that coins in TestV are not rotated, only in TestR. The resolution to this issue in TestR involved the application of augmentation in the training data.

Some of the results of HoG classifying:



2.2 – Discuss on Results

When predictions are observed, it is noted that the model correctly distinguishes whether a coin is observed or reserved. The underlying reason for this capability may stem from the fact that every coin exhibits a very similar reserved side. When predictions exclusively focus on observed coins, a higher accuracy is observed compared to reserved coins.

Additionally, a common false prediction is identified, where 5 kr is predicted as 50 kr. The potential reason for this misclassification is suggested to be the similarity in the visual features of the digits '5' and '50.'

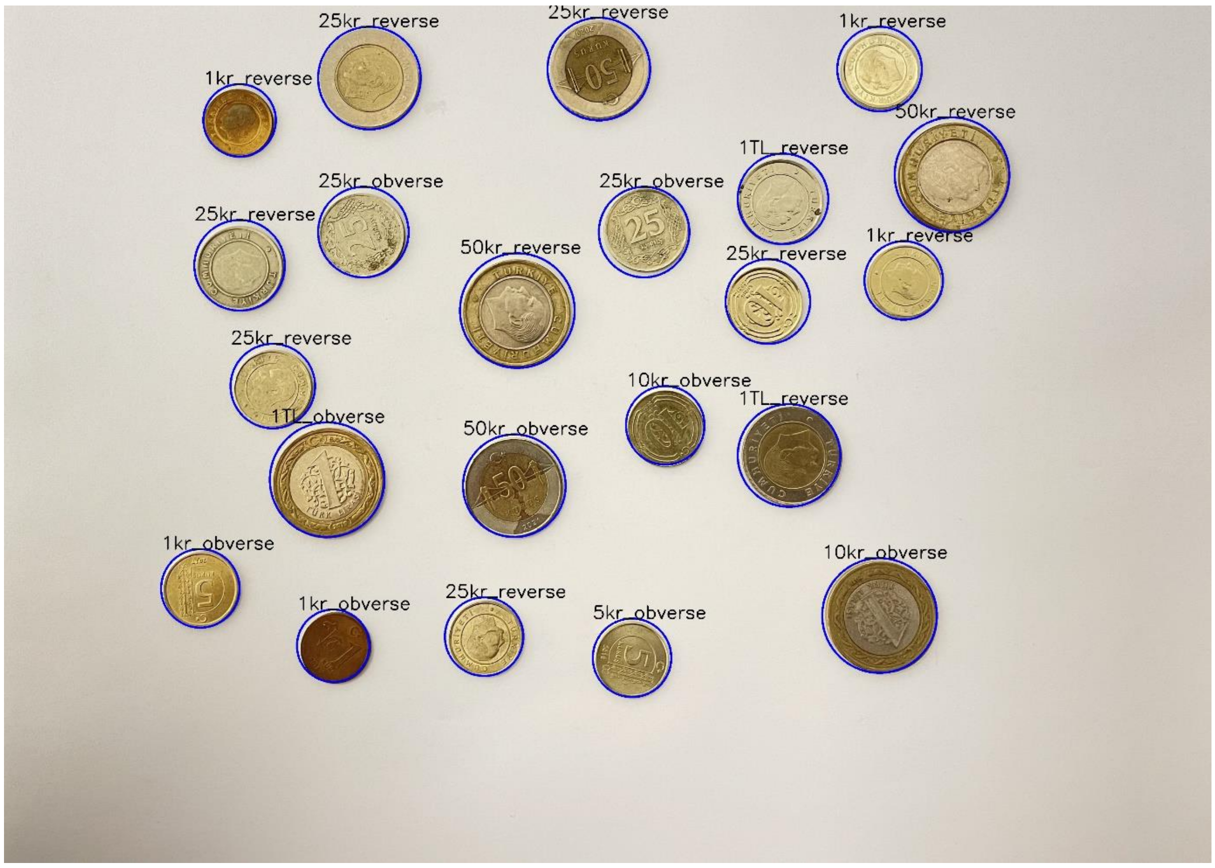
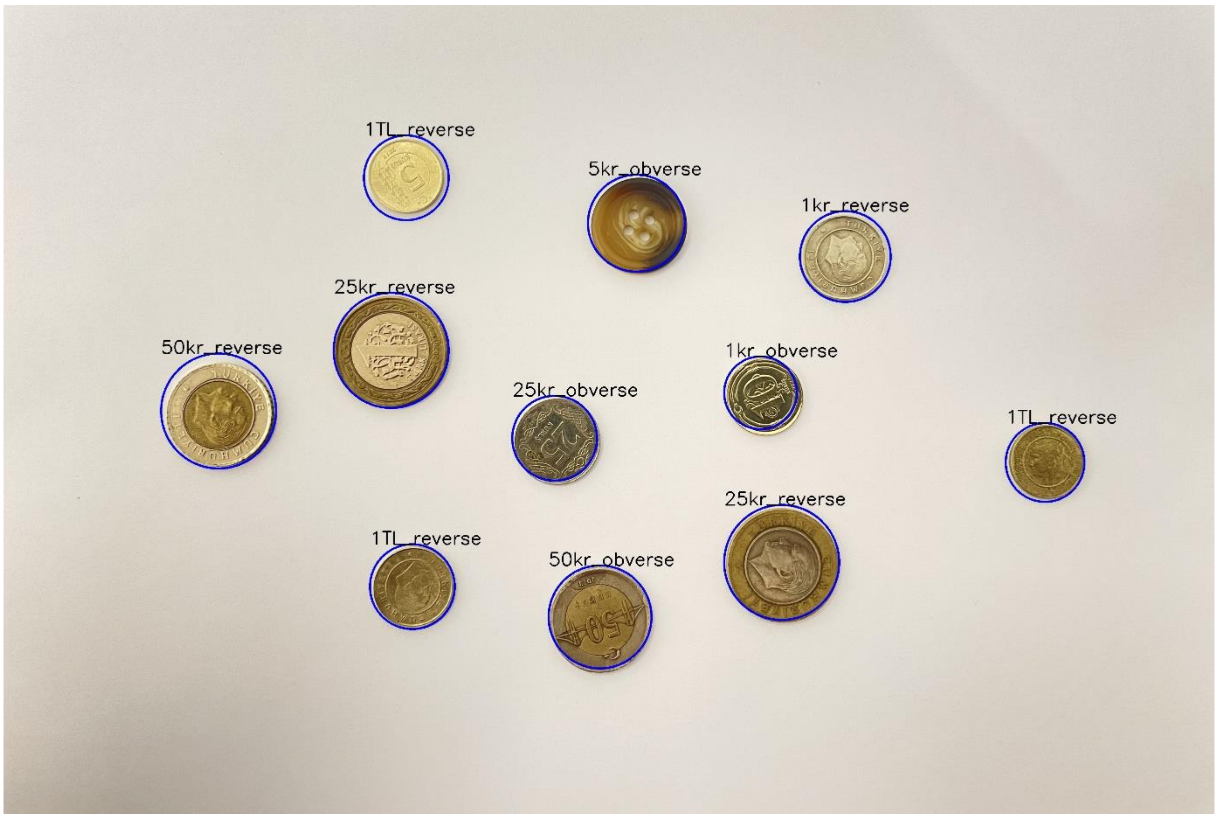
The failures in the model's performance are attributed to insufficient or limited data. In this assignment, it is contended that there was not enough training data available to adequately train the model. The provision of only 4-5 images for each class is considered inadequate, potentially leading to overfitting. It is worth noting that increasing the diversity and quantity of training data may mitigate such issues and improve the model's generalization.

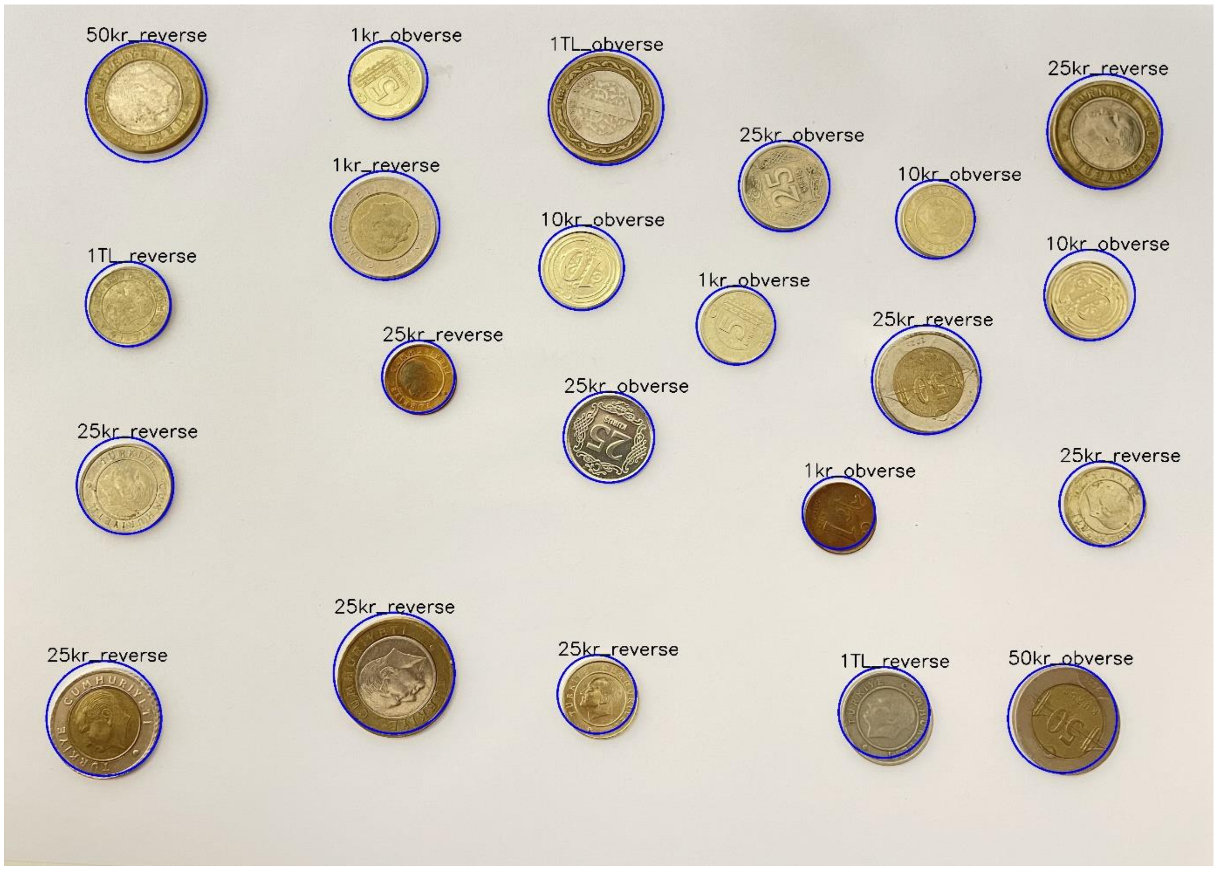
2.3 – Bonus Part (TestR)

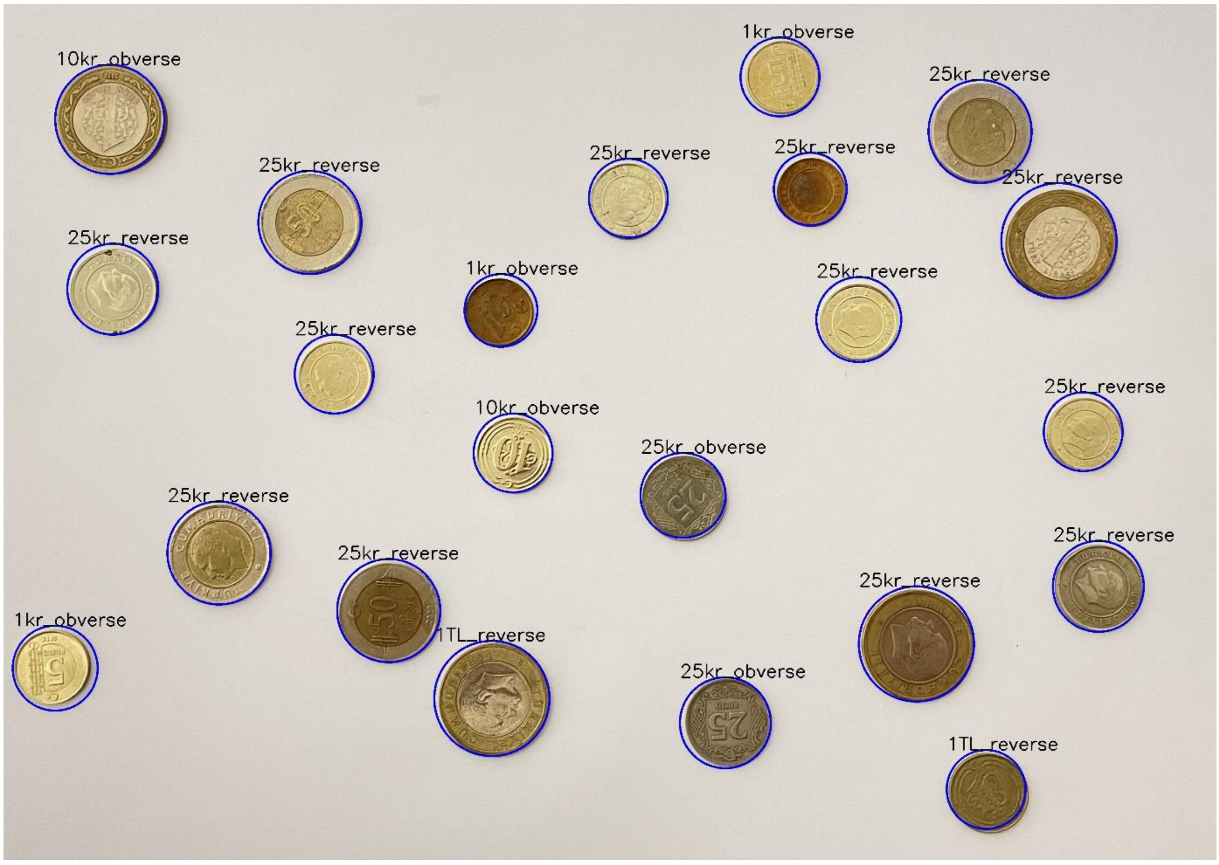
```
for path in train_image_paths:
    image = train_images(path)

      #...

# Augmenting the training set for TestR where the coins can be rotated
    for angle in range(0, 360, 15):
        # Rotating the image repeatedly by 15 degrees
        rotated_image = np.array(Image.fromarray(image).rotate(angle))

        # Extracting HoG features
        featureR = hog_features(rotated_image)

        # Getting features and labels lists
        featuresR.append(featureR)
        labelsR.append(label)
```

In this bonus part, the same methods employed in the TestV folder were utilized. The orientation problem was addressed by augmenting the training data. Prior to the extraction of HOG features from train images, each image was repeatedly rotated by 15 degrees, and the HOG features of the rotated images were extracted. The model used in TestR was subsequently trained using these features.

Results of HoG which robusts orientation:


25kr_reverse
25kr_reverse
1kr_reverse
1kr_reverse
50kr_reverse
1TL_reverse
25kr_obverse
25kr_obverse
25kr_reverse
50kr_reverse
25kr_reverse
1kr_reverse
25kr_reverse
10kr_obverse
1TL_reverse
1TL_obverse
50kr_obverse
1kr_obverse
10kr_obverse
25kr_reverse
1kr_obverse
5kr_obverse

1TL_reverse
5kr_obverse
1kr_reverse
25kr_reverse
50kr_reverse
1kr_obverse
25kr_obverse
1TL_reverse
25kr_reverse
1TL_reverse
50kr_obverse


50kr_reverse
1kr_obverse
1TL_obverse
25kr_reverse
25kr_obverse
1kr_reverse
10kr_obverse
10kr_obverse
1TL_reverse
10kr_obverse
1kr_obverse
25kr_reverse
25kr_reverse
25kr_obverse
25kr_reverse
1kr_obverse
25kr_reverse
25kr_reverse
25kr_reverse
25kr_reverse
1TL_reverse
50kr_obverse


1kr_obverse
10kr_obverse
25kr_reverse
25kr_reverse
25kr_reverse
25kr_reverse
25kr_reverse
25kr_reverse
1kr_obverse
25kr_reverse
25kr_reverse
10kr_obverse
25kr_reverse
25kr_obverse
25kr_reverse
25kr_reverse
25kr_reverse
1kr_obverse
1TL_reverse
25kr_reverse
25kr_obverse
1TL_reverse