Introduction to Image Processing

Group Project Final Report

Group 5

Topic: Counting the value of coins in a scene

Group members:

- Seyfullah Özdemir
- Vusal Alakbarov
- Mustafa Arınmış
- Toghrul Bakhishov
- Ulaş Yurttaş

Introduction to the problem

Our daily lives require highly accurate and effective automatic coin identification systems. Coin sorting devices and coin identification systems have become important in our daily lives. They can be found in places like banks, grocery stores, and vending machines. Even though they are utilized on a daily basis, coin identification systems can also be employed for research by institutions or organizations that work with historical coins. Based on the various techniques they employ, there are three major types of coin recognition systems on the market that use mechanical processes, electromagnetic techniques, and image processing.

Image-based systems have emerged in recent years, where coin images are processed using techniques like FFT (Fast Fourier Transform), DCT (Discrete Cosine Transform), edge detection, and segmentation. Various features are extracted from the images to recognize different coins. This summary highlights the existing systems and techniques proposed by researchers in image-based coin recognition.

Challenges we expect to overcome are overlapping coins, determining coin value out of its size when its tail, lighting, noise, and size of the data set might be problems that take time to solve. We restricted the problem to only Turkish Lira to limit the data set.

High-level steps:

1. Detect coins
2. Determine their counts (values)
3. Sum them up and output the result

Survey:

In recent years coin recognition systems based on images became popular. These images are processed by using various techniques of image processing like, Gabor Wavelets, DCT, edge detection, segmentation, image subtraction, and various features are extracted from the images.

The following table gives example techniques, coin types they were used on and their accuracy rates:

Year	Authors	Approach	Coins Tested	Accuracy
1993	Minoru Fukumi et al.	BP and GA for neural network-based coin recognition	500 yen, 500 won	100%
1996	Paul Davidsson	Learning characteristic decision trees for coin classification	Canadian, Hong-Kong	99.7%, 98.3%
2003	Michael Nolle et al.	Coin recognition and sorting system called Dagobert	Modern coins	99.24%
2004	Seth McNeill et al.	Coin recognition system based on vector quantization and histogram modeling	US coins	94%
2005	Reinhold Huber et al.	Multistage approach using Eigenspace and Bayesian fusion for coin classification	Coins from 30 countries	93.23%
2006	Adnan Khashman et al.	Intelligent Coin Identification System (ICIS)	Turkish 1 Lira and 2 Euro coin	96.3%
2010	Hussein R. Al-Zoubi	Statistical approach for Jordanian coin recognition	Jordanian coins	97%
2010	Huahua Chen	Chinese coin recognition based on unwrapped image	144 variably rotated coins	80.6%

Method:

Before getting into details, let’s see our input image to see and analyze differences clearly:

1. **Clear noises: Apply Gaussian Blur**

It is necessary to use noise reduction to improve image quality, enhance visual interpretation, and enable accurate analysis.

We have our self-implemented gaussianblur.py file below and next to it there is the code snippet to show the blurred image as an output, and the output itself:

```python
import cv2
import numpy as np

def gkernel(length, sigma):
    """
    Gaussian Kernel Creator via given length and sigma
    """

    ax = np.linspace(-(length - 1) / 2., (length - 1) / 2., length)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sigma))

    return kernel / np.sum(kernel)


def gaussian_blur(image, kernel_size, sigma):
    # Create the Gaussian kernel
    kernel = gkernel(kernel_size, sigma)

    # Apply the Gaussian kernel to the image
    blurred = cv2.filter2D(image, -1, kernel)

    return blurred
```

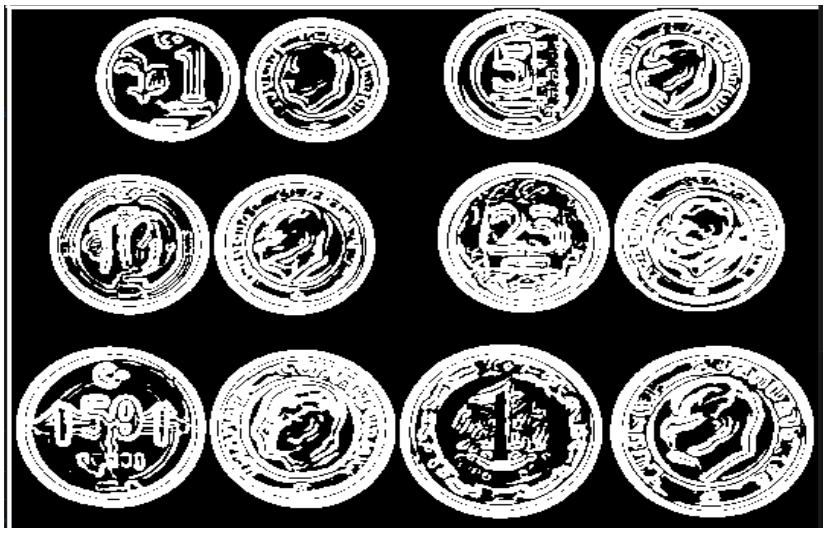
```python
blurred_image = gaussian_blur(image, 15, 1.5)
cv2.imshow('Gaussian', blurred_image)
```

2. Apply Edge Detection: Canny edge detector

With edge detection we will be able to identify edges of coins. Let's see the code we implemented for canny.py, how we run it and the corresponding output:

```python
import cv2
import numpy as np

def canny_edge_detection(image):

    # Calculate gradient in x and y directions
    gradient_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    gradient_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)

    # Calculate gradient magnitude and direction
    gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)

    high_threshold = 30
    low_threshold = 150

    strong_edges = np.where(gradient_magnitude > high_threshold, 255, 0).astype(np.uint8)
    weak_edges = np.where((gradient_magnitude <= high_threshold) & (gradient_magnitude > low_threshold), 255, 0).astyp

    edges = np.zeros_like(strong_edges)
    edges[strong_edges > 0] = 255

    # Define the neighborhood kernel
    kernel = np.ones((3, 3), dtype=np.uint8)

    # Perform dilation to connect weak edges to strong edges
    dilated_weak_edges = cv2.dilate(weak_edges, kernel, iterations=1)

    # Determine final edges by suppressing weak edges not connected to strong edges
    final_edges = np.where((edges > 0) | (dilated_weak_edges > 0), 255, 0).astype(np.uint8)

    return final_edges
```

```python
edges = canny_edge_detection(blurred_image)
edged = cv2.Canny(blurred_image, 40, 100)
cv2.imshow('Canny Edges', edges)
```



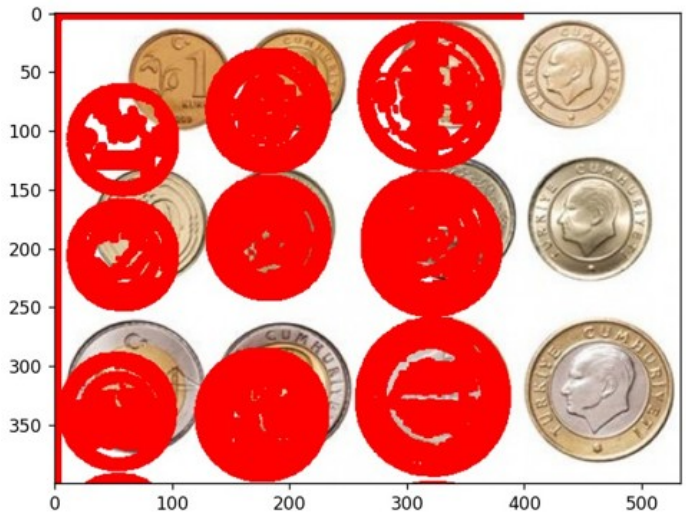
3. Contour the coins:

Before detecting coins curves should join all the continuous points (along the boundary), and having same color or intensity.

3.1 Find Contours:

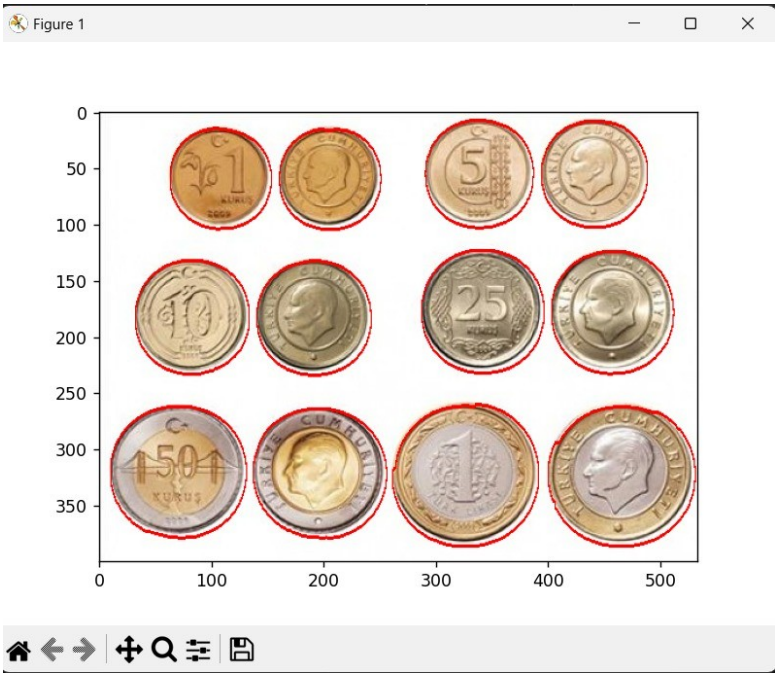
We tried to implement findContours() method, but we get unsuccessful results, unfortunately. Let's see the code and its output:

```python
def find_contours(binary):
    contours = []  # List to store all contours found
    # Array to track visited pixels
    visited = np.zeros(binary.shape, dtype=bool)

    # Iterate over each pixel in the binary image
    for i in range(1, binary.shape[0] - 1):
        for j in range(1, binary.shape[1] - 1):
            if binary[i, j] == 255 and not visited[i, j]:
                # Start a new contour if the current pixel is white and not visited
                contour = []
                stack = [(i, j)]  # Stack to store pixels of the contour

                while stack:
                    x, y = stack.pop()
                    visited[x, y] = True  # Mark the pixel as visited

                    neighbors = get_neighbors(binary, visited, x, y)
                    for neighbor in neighbors:
                        nx, ny = neighbor
                        stack.append((nx, ny))
                        visited[nx, ny] = True

                    if not any(get_neighbors(binary, visited, x, y)):
                        # If the current pixel has no unvisited neighbors, add it to the contour
                        contour.append((x, y))

                contours.append(contour)  # Add the contour to the list

    return contours
```



3.2 Draw Contours:

```python
def draw_contours(image, contours, color=(0, 255, 0)):
    for contour in contours:
        contour = np.array(contour)
        contour = contour.squeeze().reshape(-1, 2)

        # Iterate over each point in the contour
        for i in range(len(contour)):
            pt1 = tuple(contour[i])  # Get the current point as a tuple
            # Get the next point (wrap around to the beginning if it's the last point)
            pt2 = tuple(contour[(i + 1) % len(contour)])
            # Draw a line segment between pt1 and pt2 in the image
            draw_line(image, pt1, pt2, color)
```



4. Convolutional Neural Network (CNN)

We used the Keras library to train a Convolutional Neural Network (CNN) model, which is a popular deep learning technique, in order to detect coins. The CNN model learns to identify different types of coins based on their visual features, allowing for automatic coin detection in images.

We implemented predict() method to split corresponding quantities of our coins as you see on the snippet below:

```python
def predict(img, model):
    new_gray = convert_to_grayscale(img)
    new_resized = cv2.resize(new_gray, (416, 416))
    new_input = np.expand_dims(new_resized, axis=0)
    new_input = np.expand_dims(new_input, axis=-1)
    cnn_model = model

    (images, labels, classes) = prepare_images()
    X_train, X_test, y_train, y_test = train_test_split(images, labels, test_size=0.2, random_state=42)
    label_encoder = LabelEncoder()
    label_encoder.fit_transform(y_train)
    label_encoder.transform(y_test)

    # Predict the label
    predictions = cnn_model.predict(new_input)
    predicted_class_index = np.argmax(predictions)
    predicted_class = label_encoder.inverse_transform([predicted_class_index])[0]
    if predicted_class == 0:
        return ('5kr', 0.05)
    elif predicted_class == 1:
        return ('10kr', 0.1)
    elif predicted_class == 2:
        return ('25kr', 0.25)
    elif predicted_class == 3:
        return ('50kr', 0.5)
    else:
        return ('1tl', 1.0)
```

Dataset:

The dataset includes all types of Turkish coins in various angles including head and tails. The coins are: 0.01 TRY, 0.1 TRY, 0.25 TRY, 0.5 TRY, 1 TRY. You can see the images from the link below.

Data set: https://www.kaggle.com/datasets/nyahmet/turkish-coin-dataset

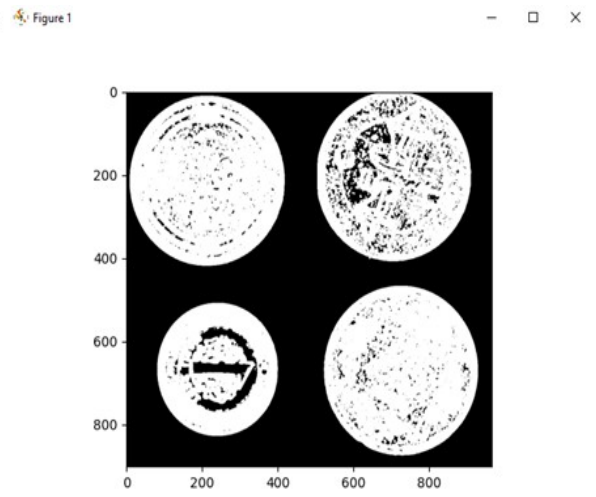
Experimental Results:

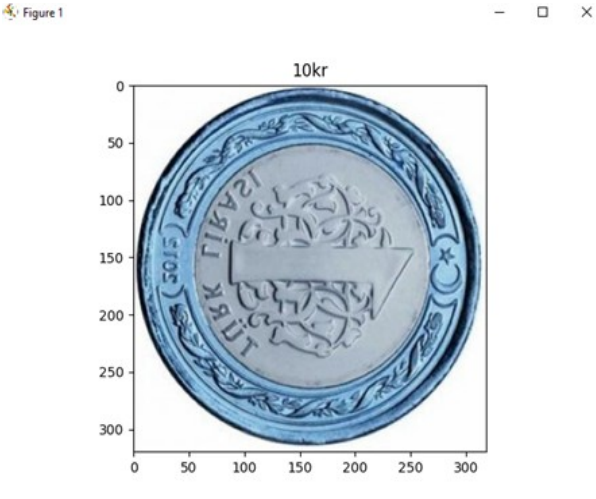
For the experiment section, we set 80 percent of the images in the data set as trains and the remaining 20 percent as tests. Below is the success of the model for the train and test set as a result of the 10-step epochs. As a result, we achieved an 86% success rate.

Test Loss: 1.7188084125518799

Test Accuracy: 0.8636363744735718

First of all, the program aims to find all the circles in the picture. Then, it stores the pixel values of these circles as an image in the array. The model we have created for each value in the Array makes an estimation.
An example is given below. In this example, our model detected all 4 coins first. But It was able to estimate the value of only 3 of them, these are 0.1 lira, 0.5 lira, 0.25 lira, . It guessed the value of the remaining 1 incorrectly, that is 1.0 lira. As a result, the model returned 0.95 lira, while it should have returned 1.85 lira.





Source Code:

Our source code performs coin counting on an input image. It applies image processing techniques such as Gaussian blur, Canny edge detection, and contouring, which are implemented from scratch by ourselves, to identify and extract the coins from the image. Then, a pre-trained CNN model is used to predict the coin type and associated denomination. The total amount is calculated by summing up the denominations of all detected coins. You can visit our Github repository to view the full source code.

Github repository of source:

https://github.com/arinmis/coin-detection

Resources:

- Image Processing Based Systems and Techniques for the Recognition of Ancient and Modern Coins: https://arxiv.org/ftp/arxiv/papers/1312/1312.6599.pdf
- Automated Coin Recognition and Counting by Image Processing Techniques: https://www.ijert.org/automated-coin-recognition-and-counting-by-image-processing-techniques
- Automated Coin Recognition System using ANN: https://arxiv.org/ftp/arxiv/papers/1312/1312.6615.pdf