Project Monopoly

Hubert Skibiński

1.Goal:

The goal of the project was to identify banknotes from the Monopoly game and count their amount.

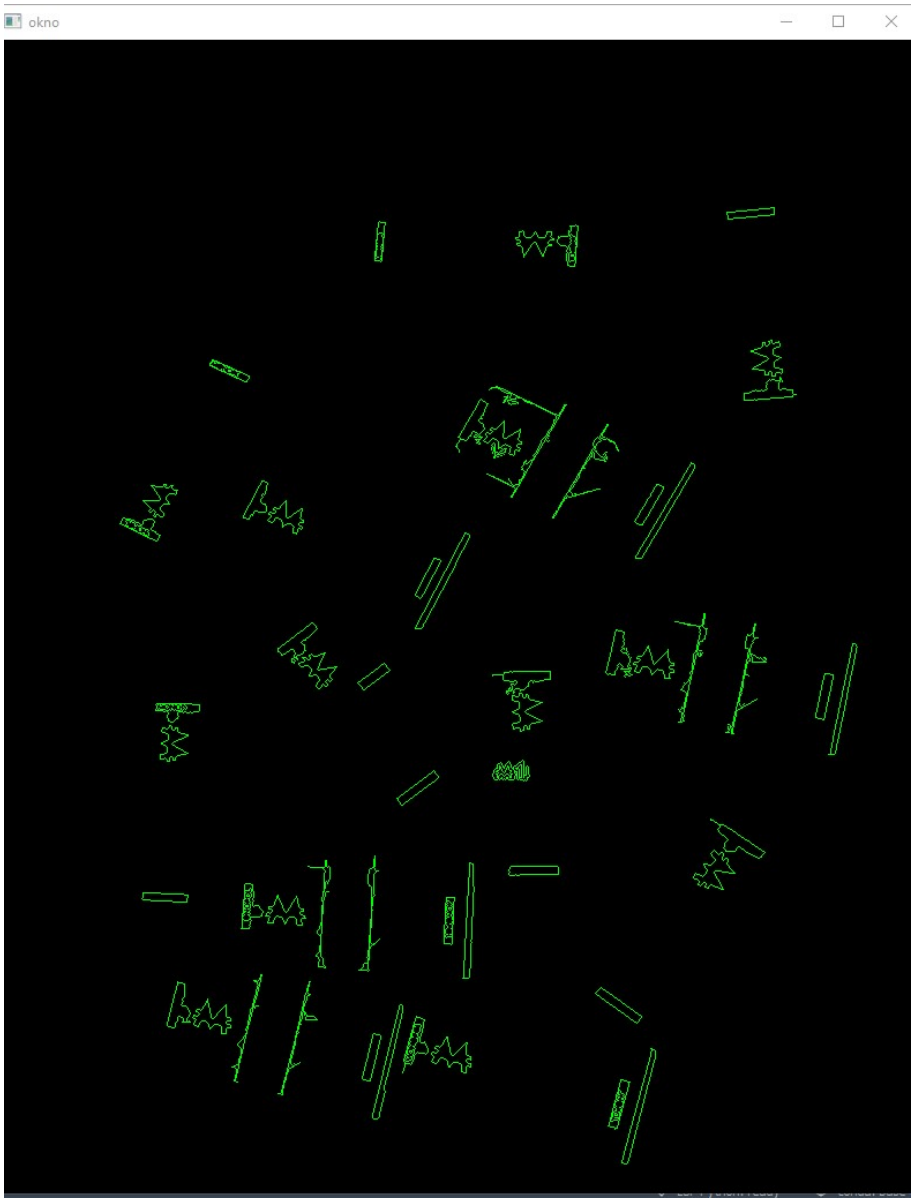
2. Sample photos for analysis:

3. Alghoritm Steps

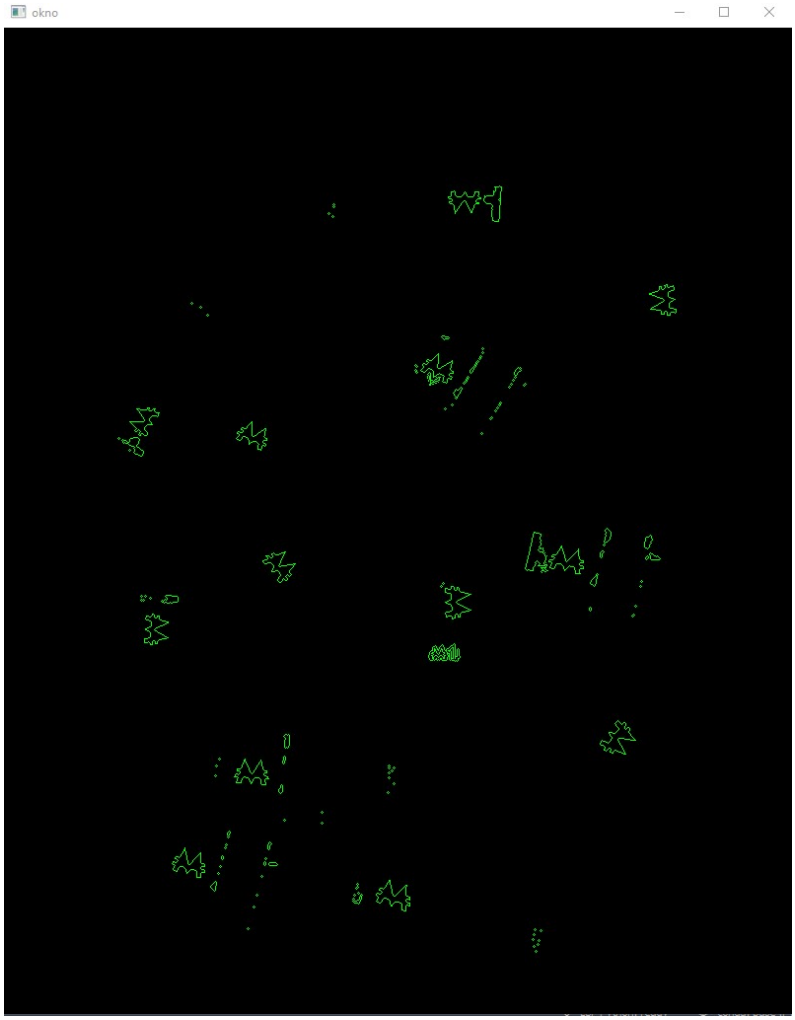
3.1 Loading the photo:

3.2 At the beginning, subtracting a constant value (80,80,80) and multiplying the resulting value by 12. (clearly separated black)

3.3 Thresholding and obtaining contours:

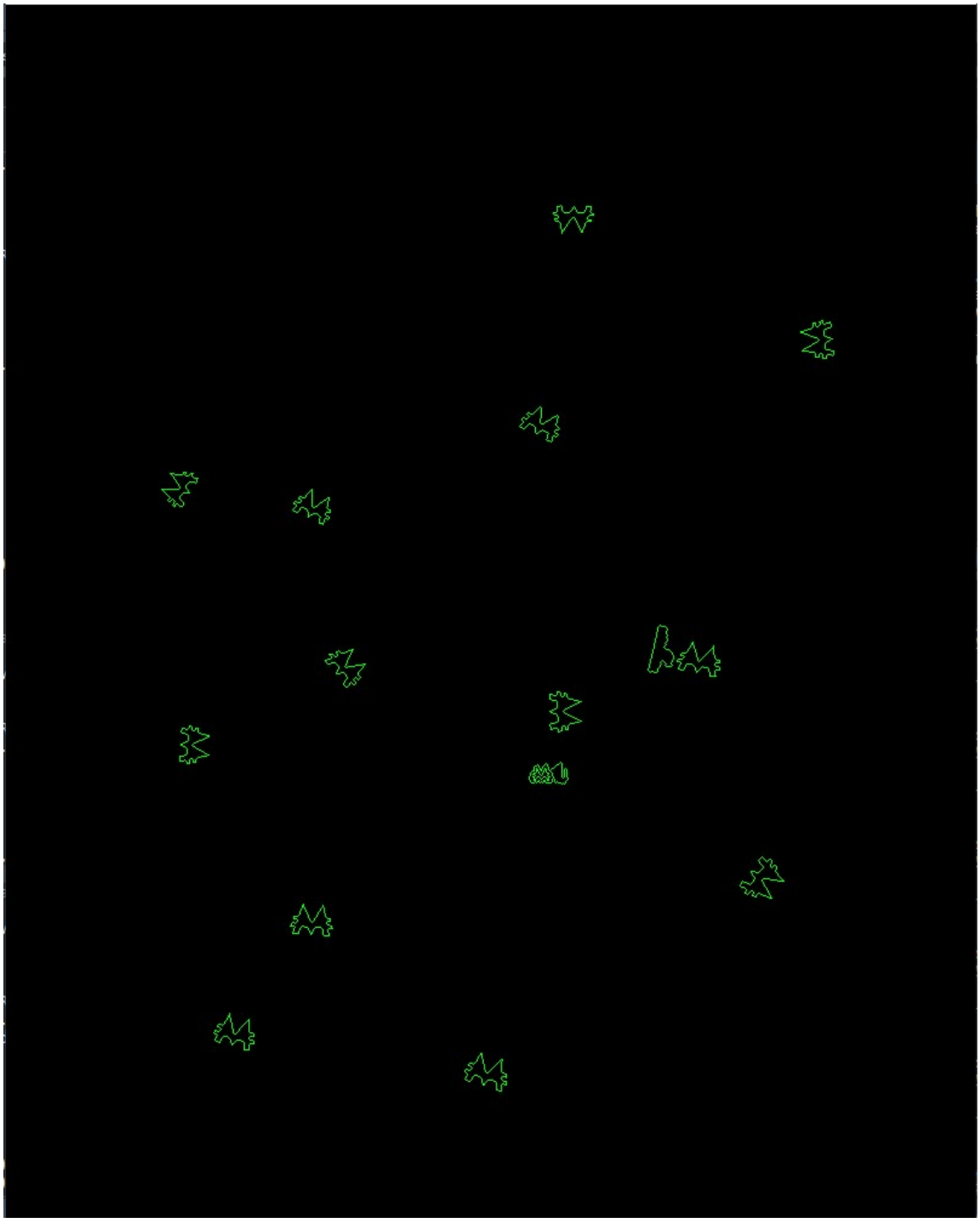


3.4 Filtration due to similarity to the sampler:

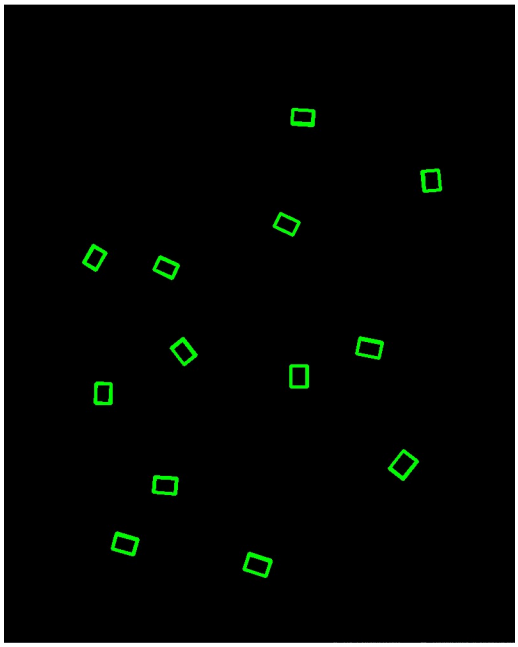


(In the program, I made drawings created by me in green. In order to compare them, e.g. using Matchshapes() I used threshold.)

3.5 Filtration due to the length of the contour:

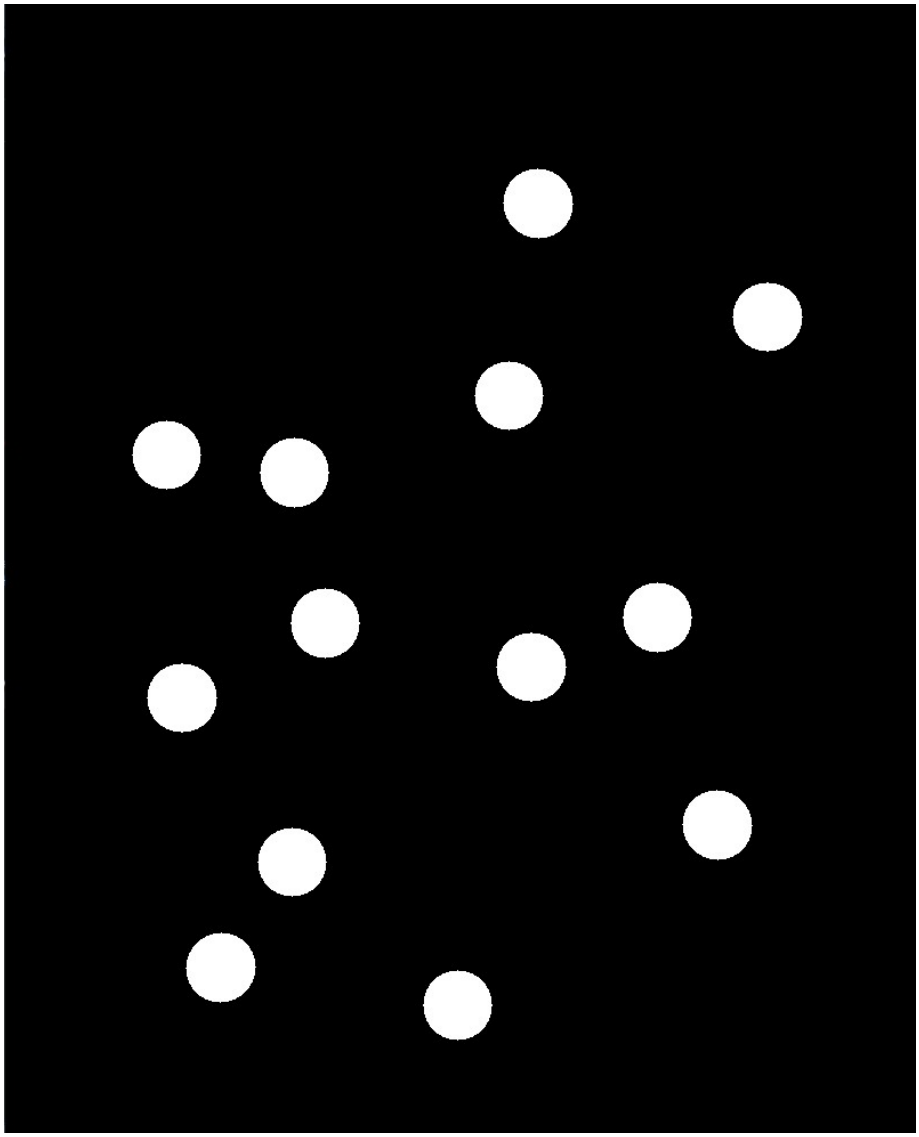


3.6 Filtration due to the ratio of length and width and the area of rectangles circumscribing the contour from point 3.5. Saving the coordinates of the centers of rectangles (some rectangles were double, so I set the thickness of the line to 3 so that they merge with each other.

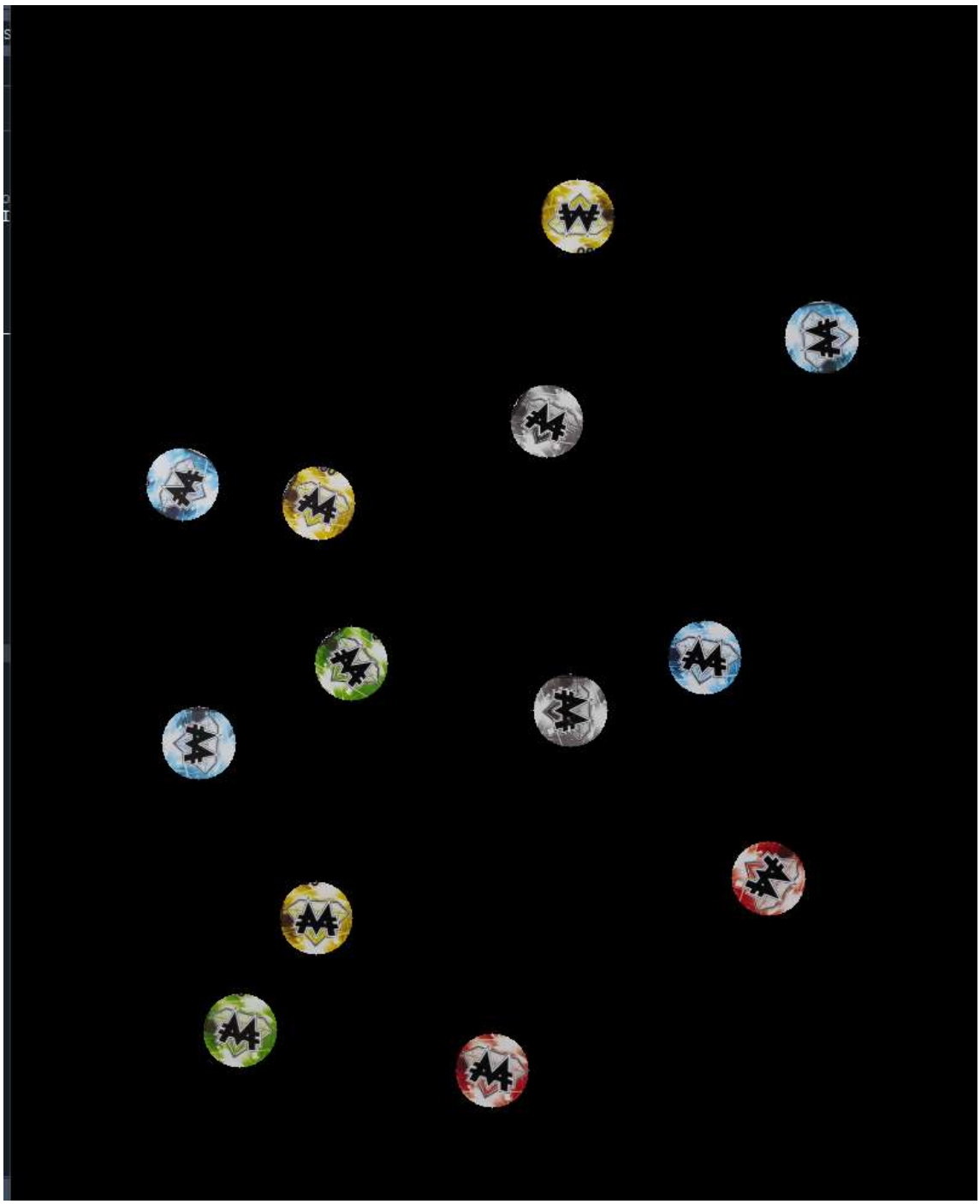


For the visualization purposes, I added rectangles with the initial image:

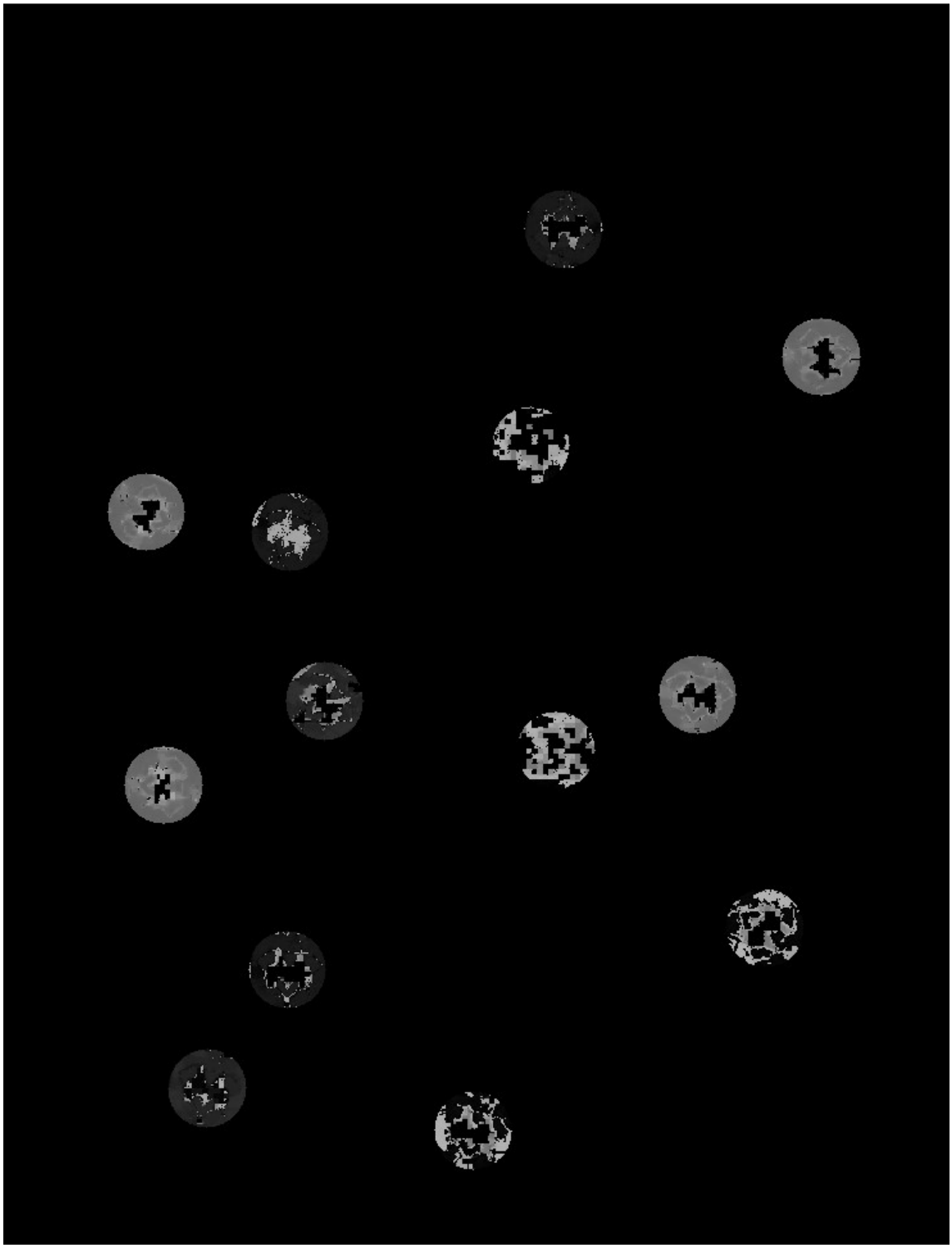
3.7 Drawing circles from the centers of rectangles designated in point 3.6 (this will be used as a mask and additionally means that double means will not be taken into account.



3.8 Obtaining the contour measures from point 3.7 and then applying the mask to the initial image:



3.9 Convert to HSV and taking only the Hue layer:



3.10 conducting a classification according to the number of pixels from the passed range in HUE. Draw surrounding circles with the color resulting from the classification into the original image:

3.11 Classification based on the Network and drawing circles with a larger radius:

4. Description of the neural network:

I created a separate database of 20 photos for learning. Using the classical algorithm, I tagged the photos(saving them to the appropriate folders). It turned out that the accuracy of the network is higher for color in the entire HSV than for Hue itself.

The dataset had 421 items. I entered 80% of all of them in the teaching set and 20% in the validation set.

Below I present the architecture of the model.

Learning took very little , I didn't have to do a lot of changes and tests because the networks quickly jumped to around 100% accuracy.

```
print(x_train.shape[0]+x_test.shape[0])
# model=keras.models.Sequential()
# model.add(keras.layers.Conv2D(32, (3, 3), activation='leaky_relu', input_shape=(32, 32, 3)))
# model.add(keras.layers.MaxPooling2D((1, 1)))
# model.add(keras.layers.Conv2D(40, (10, 10), activation='leaky_relu'))
# model.add(keras.layers.MaxPooling2D((1, 1)))
# model.add(keras.layers.Conv2D(45, (3, 3), activation='leaky_relu'))
# model.add(keras.layers.MaxPooling2D((1, 1)))
# model.add(keras.layers.Conv2D(50, (3, 3), activation='leaky_relu'))
# model.add(keras.layers.Flatten())
# model.add(keras.layers.Dense(6, activation='softmax'))

# model.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
# model.fit(x = x_train, y = y_train, epochs = 50, batch_size = 1000)
# model.save('cnn2.h5')
```