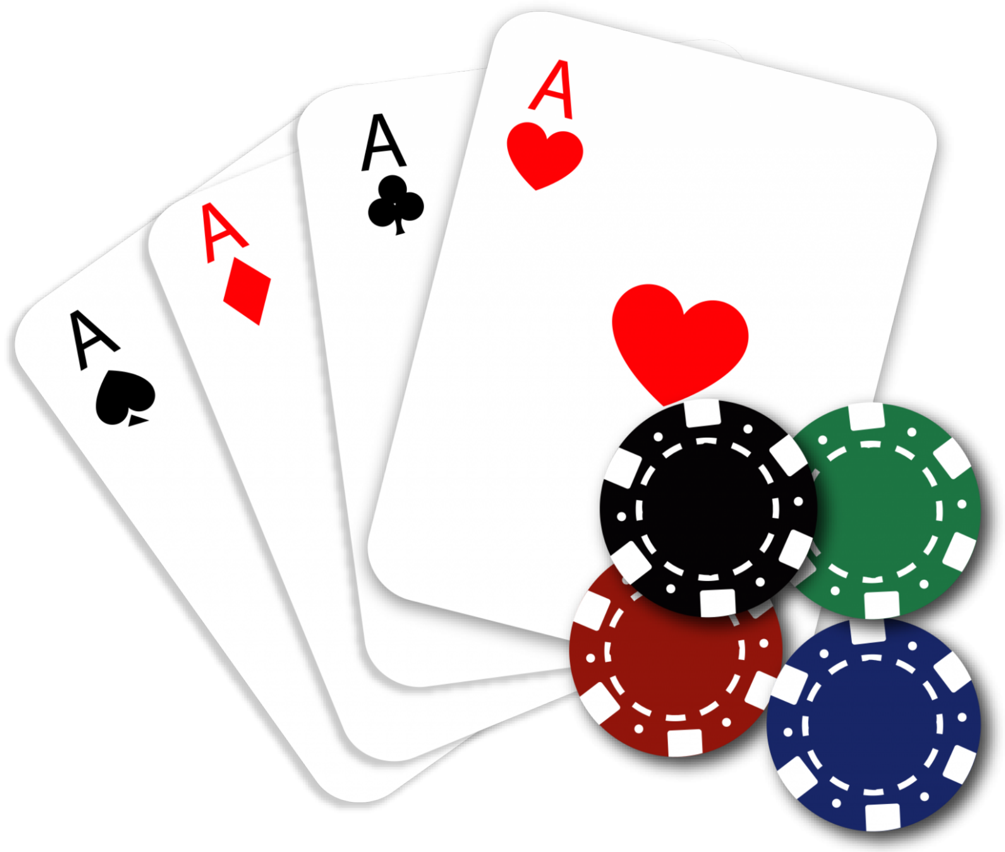
Final project presentation

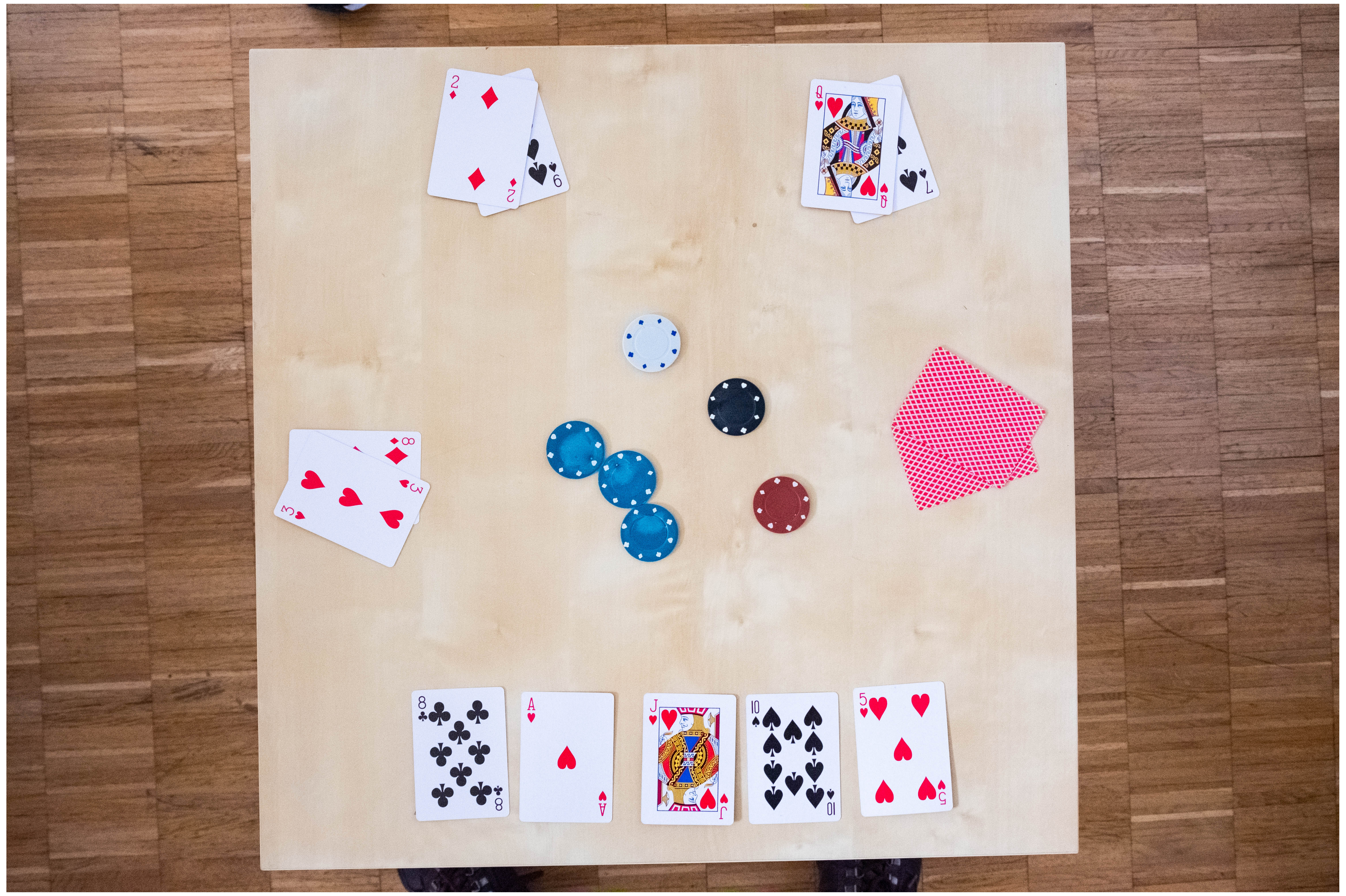
EE-451 - Image Analysis and Pattern Recognition

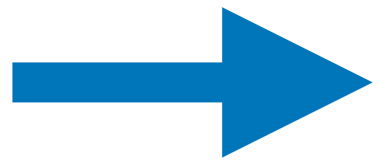


Group 8
Zunino Luca (337560)
Ceraolo Roberto (343261)
Minini Roberto (336157)

Final goal of the project

Starting point, and results to be obtained





{'T1': '8C', 'T2': 'AH', 'T3': 'JH', 'T4': '10S', 'T5': '5H', 'P11': '0', 'P12': '0', 'P21': '7S', 'P22': 'QH', 'P31': '9S', 'P32': '2D', 'P41': '8D', 'P42': '3H', 'CR': 1, 'CG': 0, 'CB': 3, 'CK': 1, 'CW': 1}

Outline of the image processing algorithm

Main steps followed to obtain the results

Preprocessing

- **Extraction** of the **corners** of the **table** from the starting image;
- **Warping** and **cutting** of the image to obtain a frontal sub-image containing only the table;
- **Splitting** of the **sub-image** containing the table in different sub-regions (four players table regions, chips table region, individual cards table region);
- **Further splitting** of the region of the table containing the **five cards** in five sub-images containing the individual cards.

Outline of the image processing algorithm

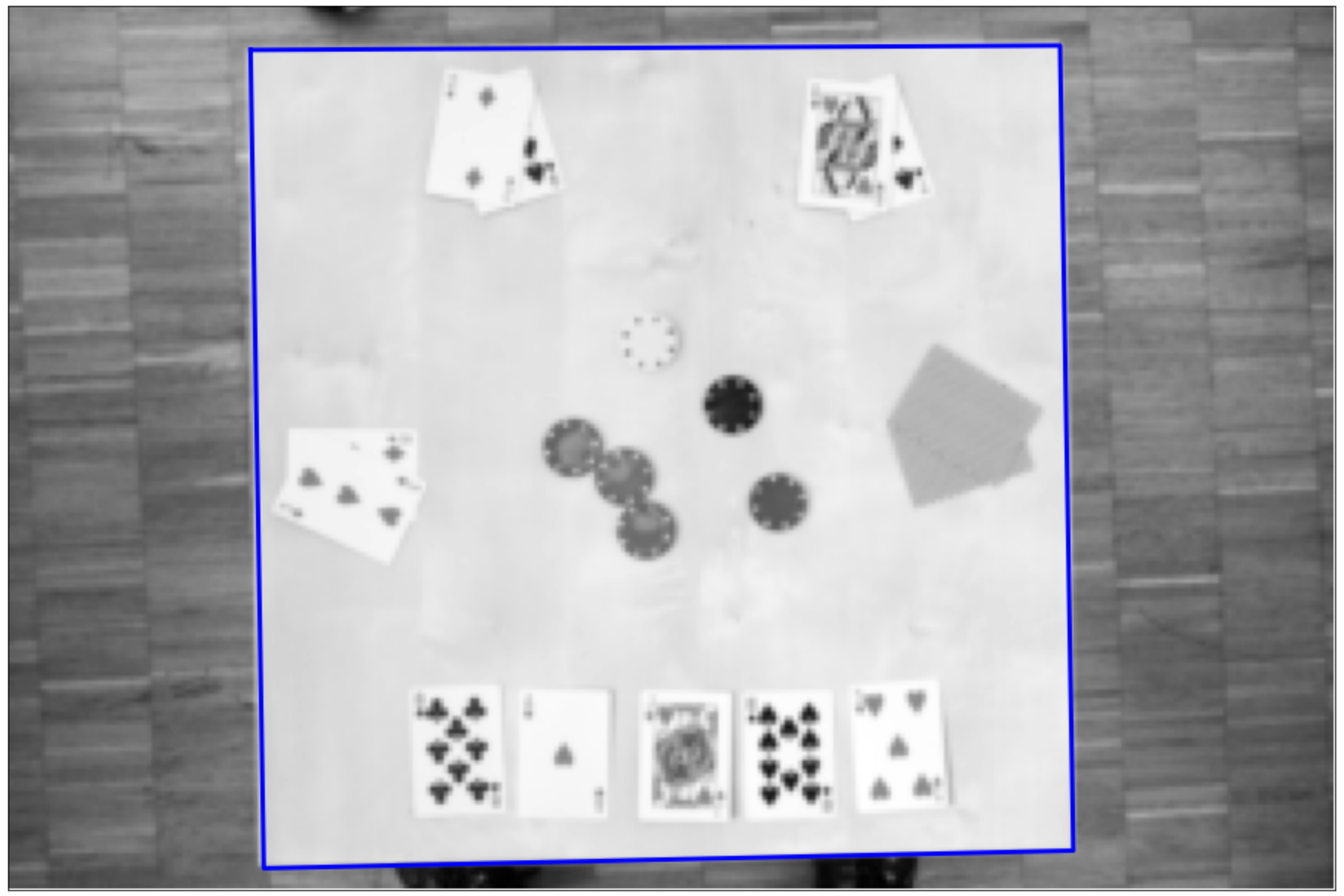
Main steps followed to obtain the results

Classification

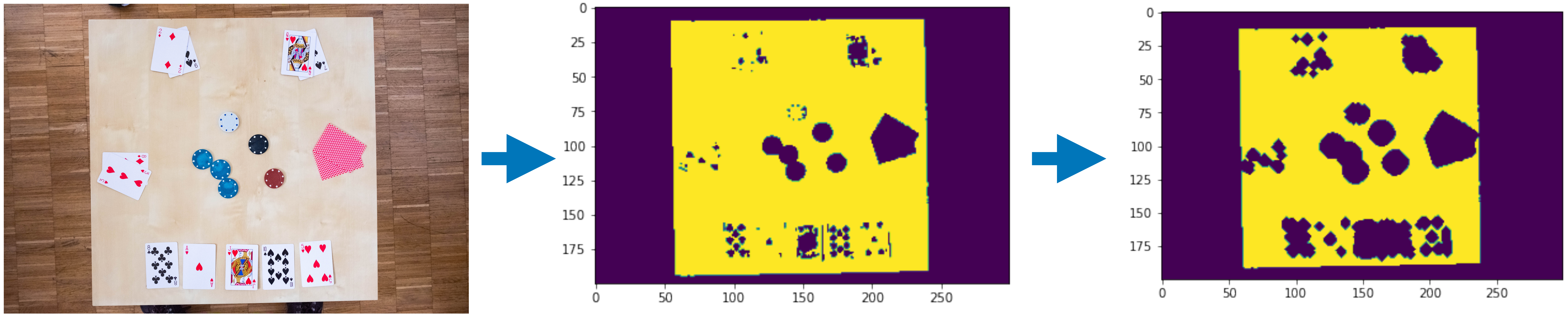
- **Detection** and **classification** of the **chips**;
- **Classification** of the table **cards** and the player cards;
- **Results** returned in a dictionary in the proper format.

1. Extraction of the table's corners from the starting image

1. Extraction of the table's corners from the starting image

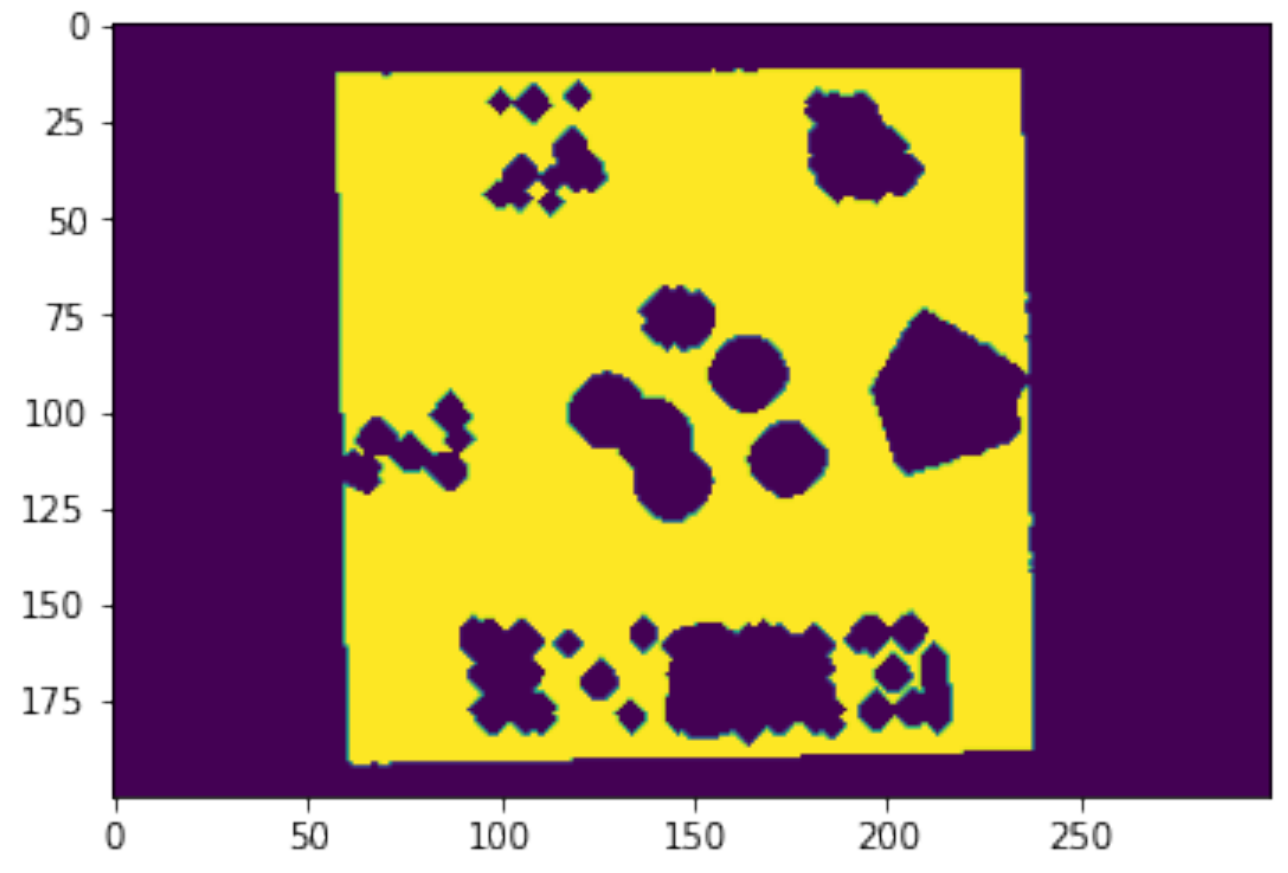


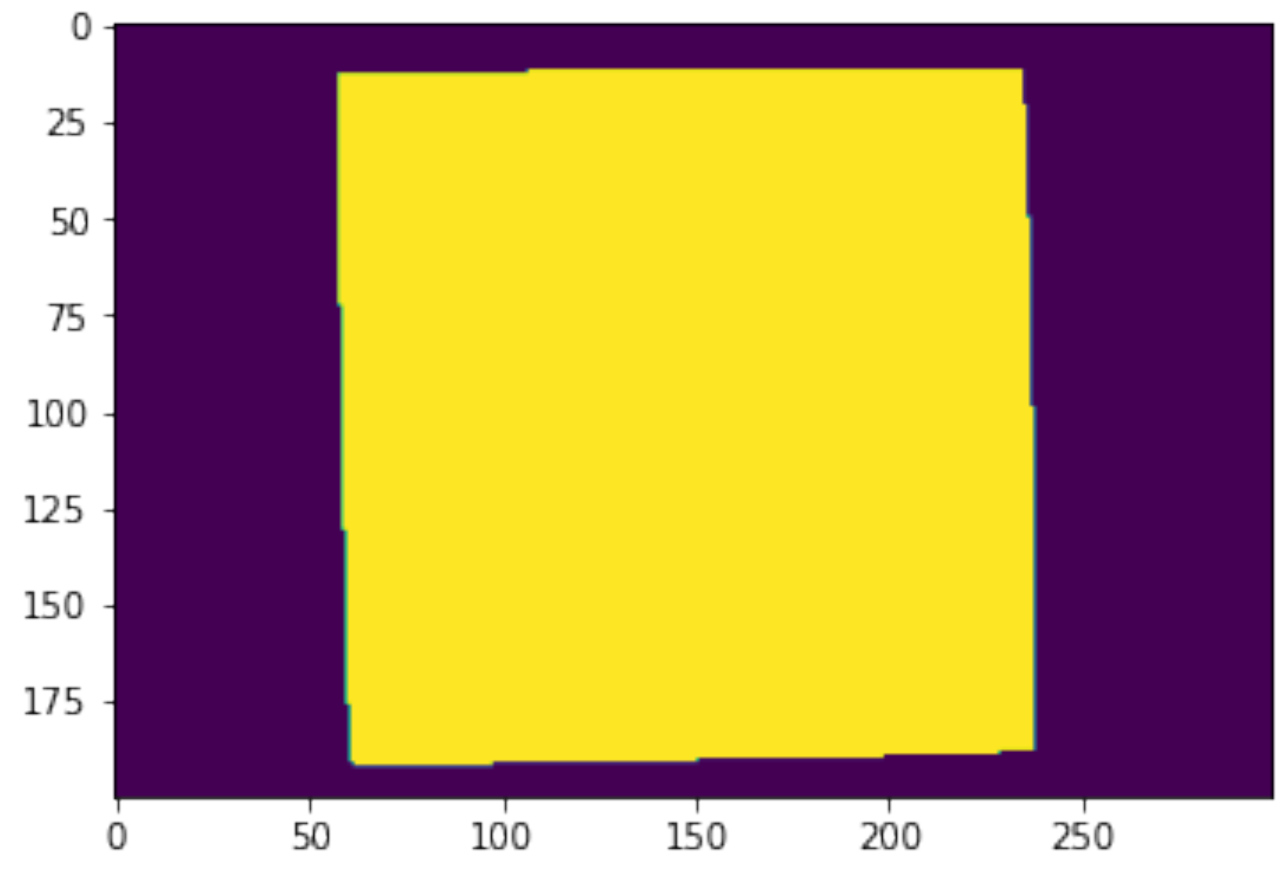
1. Extraction of the table's corners from the starting image

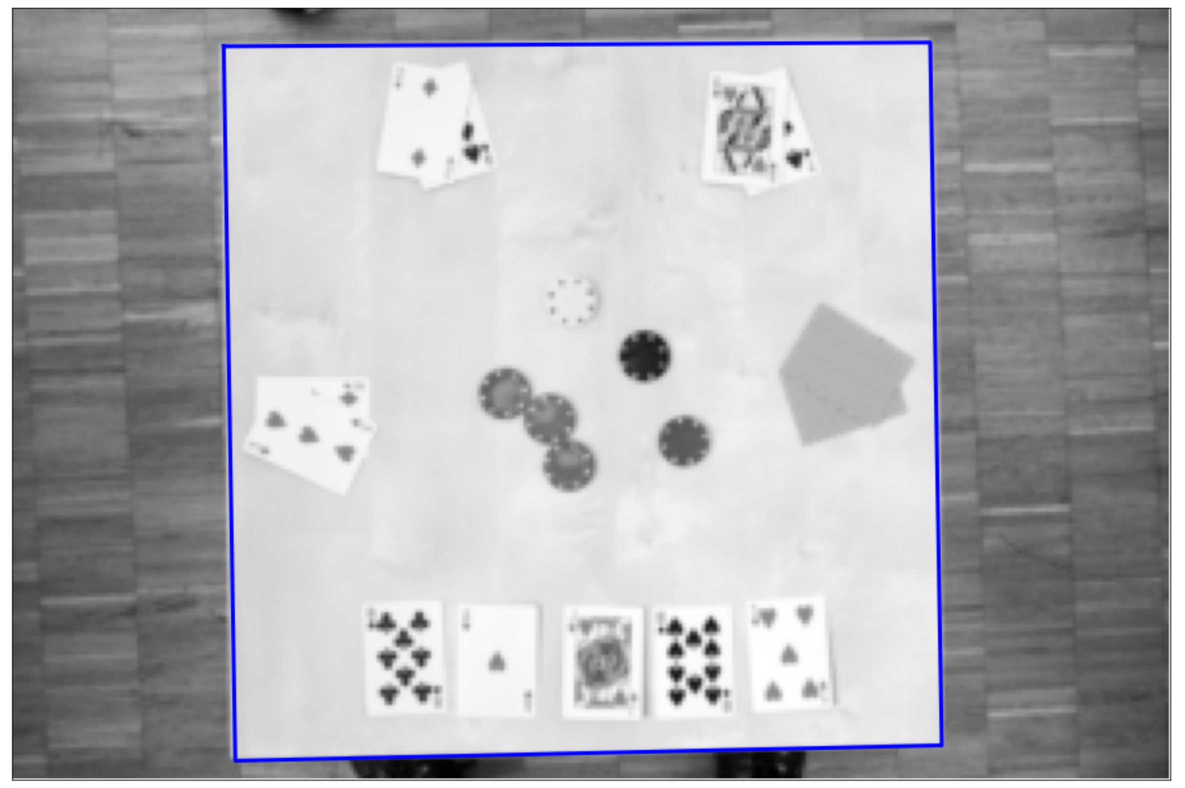


- Conversion to LAB colour space. The **L channel** was used to obtain a **grayscale image**;
- **Image rescaled** to more tractable dimensions (200 x 300 pixels);
- **Five** different **thresholds** (Triangle, Otsu, Mean, Li, Isodata) used to obtain five **binary images**. This adds robustness to the procedure.
- **Object labelling** used to assign IDs to the various objects present in the binary images;
- Detection of the most frequent ID in each image to **identify** which object is the **table**. This allows detecting each of the pixels belonging to the table;
- Each **image** is **eroded** and **opened** to separate and remove parts of the background erroneously recognized as the table.

1. Extraction of the table's corners from the starting image





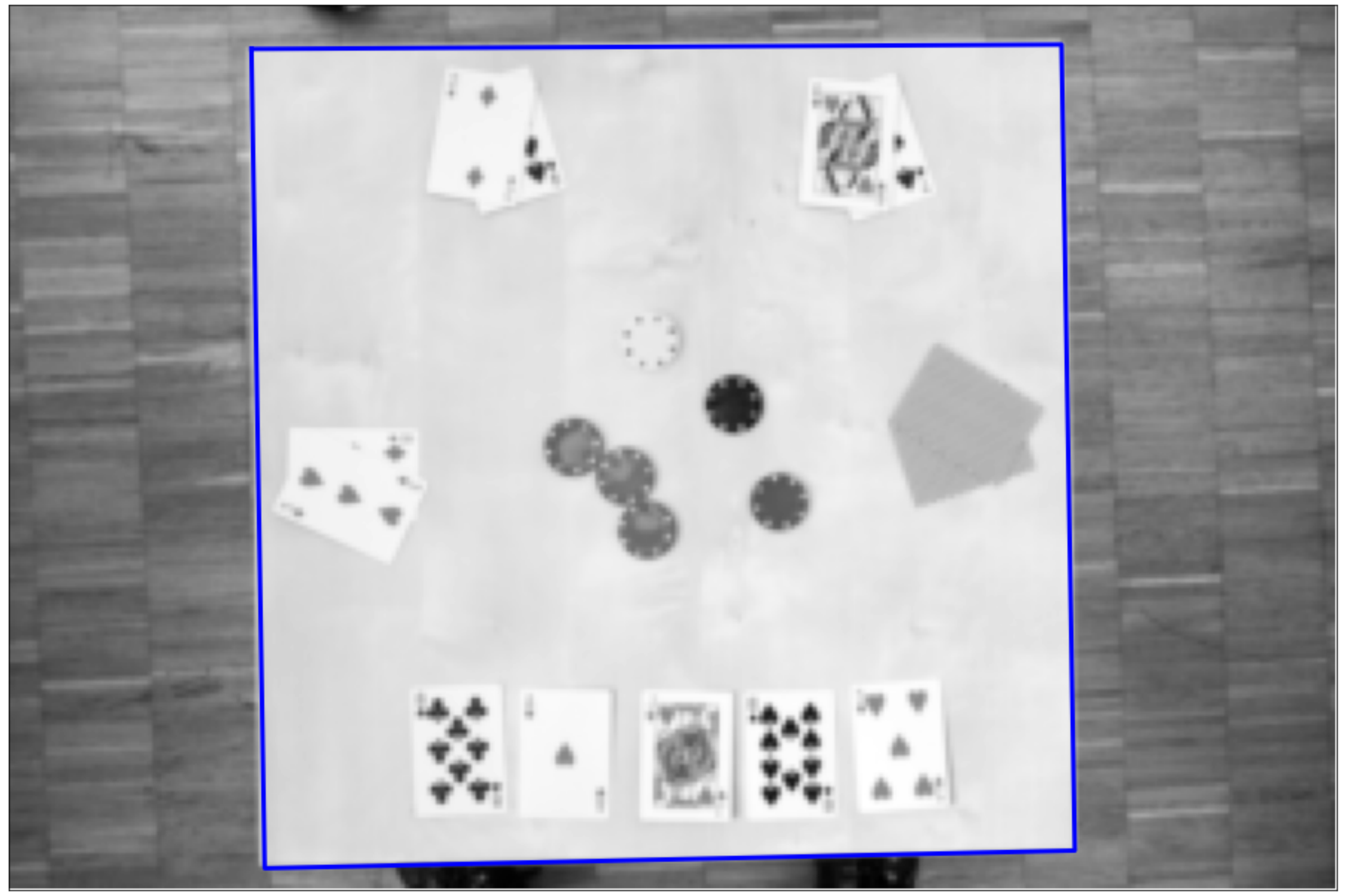


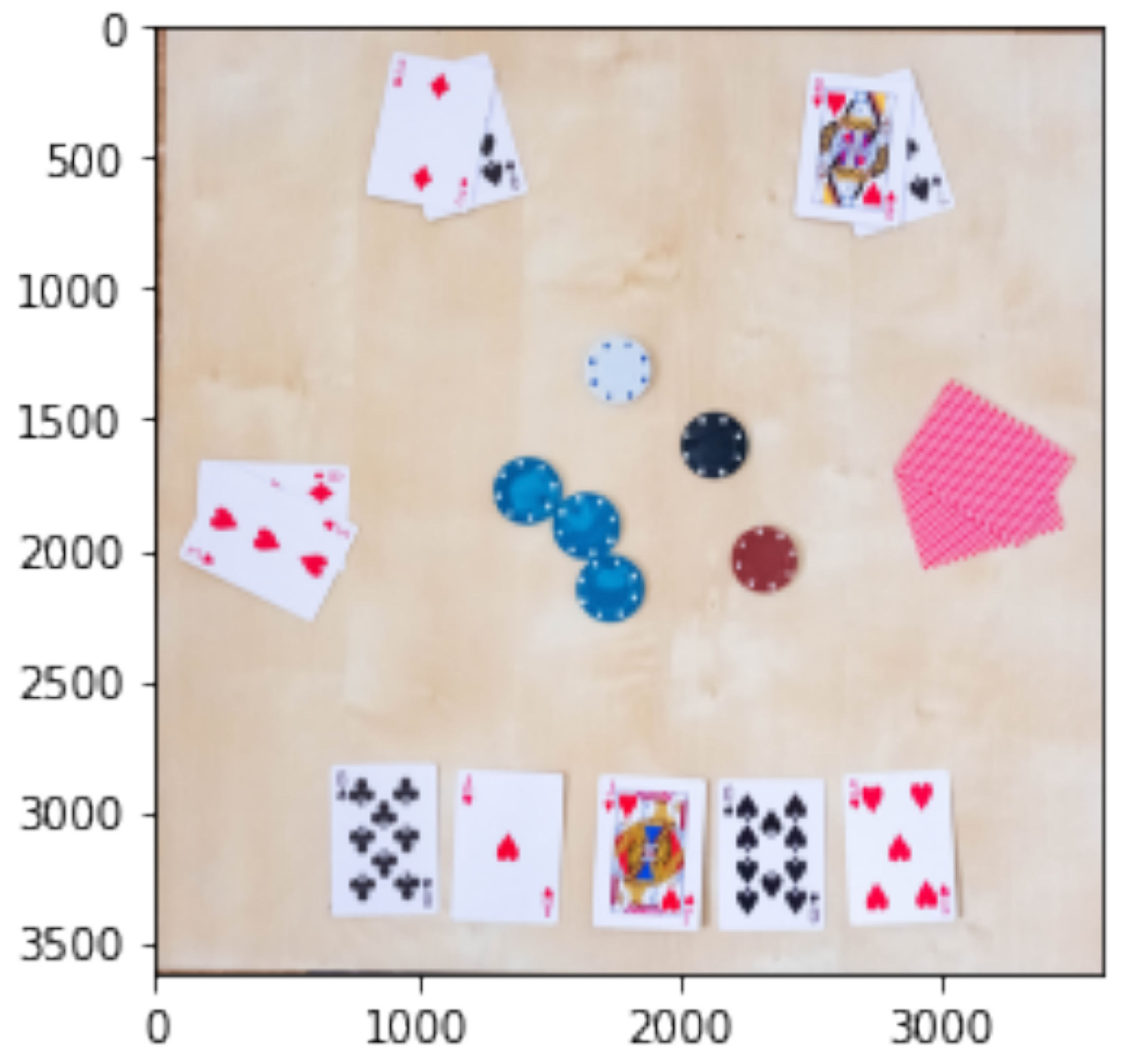
- The **convex hull enclosing the table** for each of the five binary images is found.

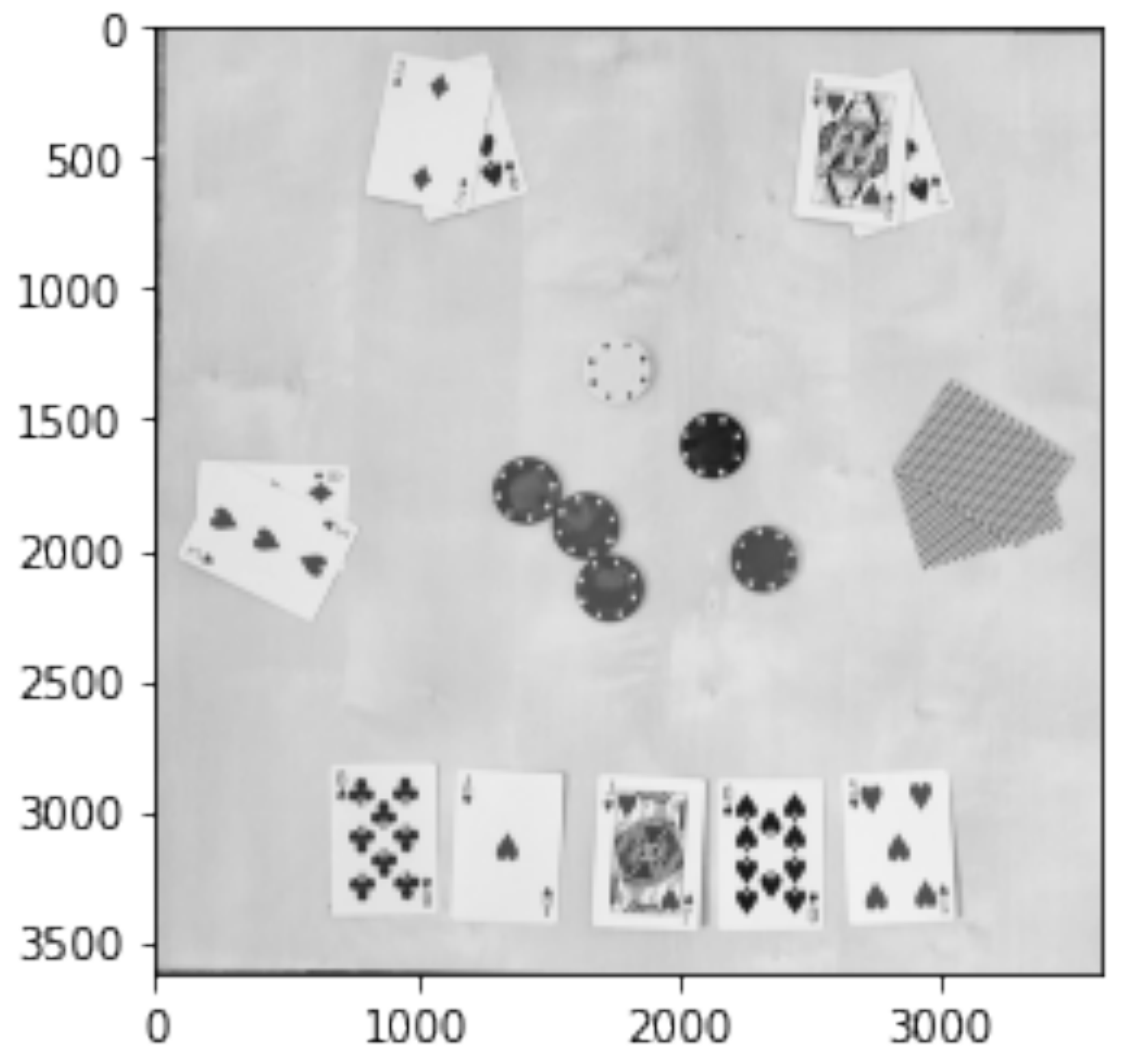
- The four **table's corners** are found by **Harris corner detection**. If the detection fails, the table is identified starting from the **minimum bounding rectangle** enclosing the points composing the convex hull;
- To choose the coordinates of the four corners starting from the five sets of the candidate coordinates, a **majority vote** is performed;
- The **coordinates** are **reordered** (if necessary) and **upsized**.

2. Warping and cutting of the image

2. Warping and cutting of the image



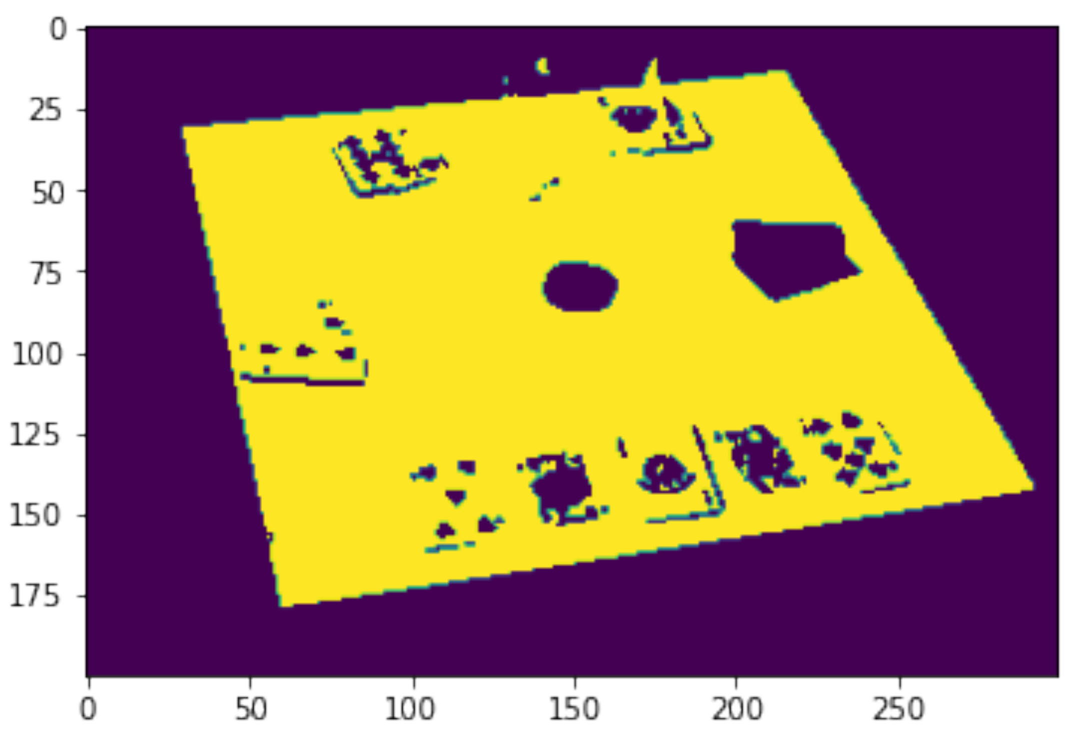


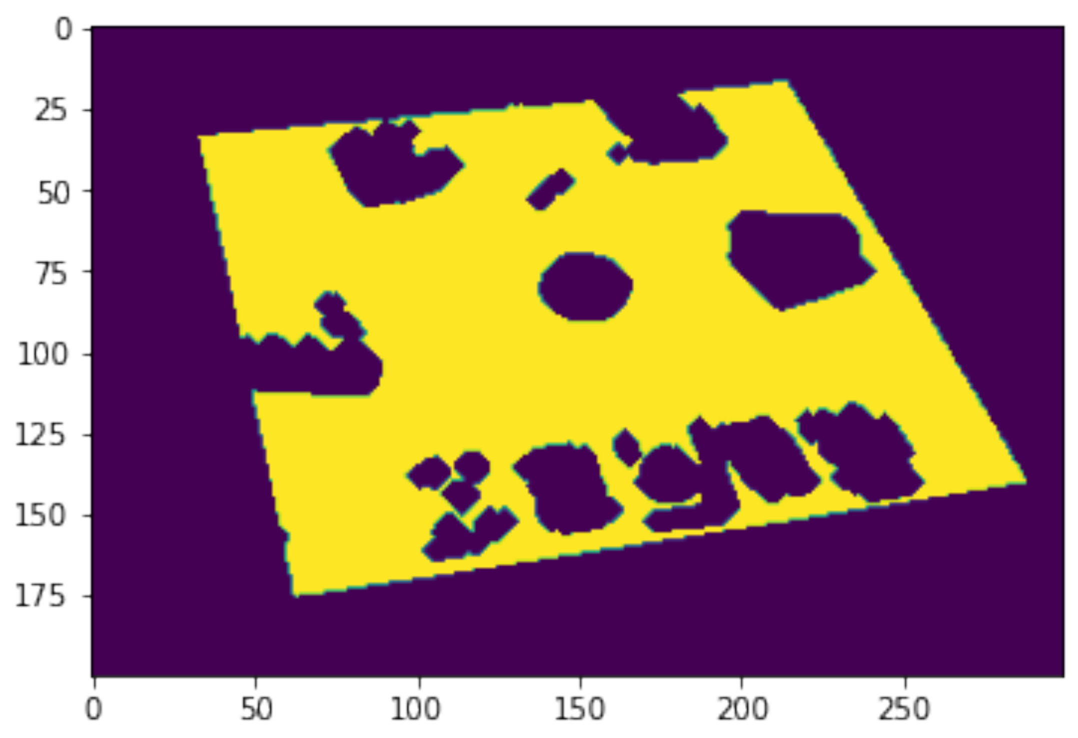


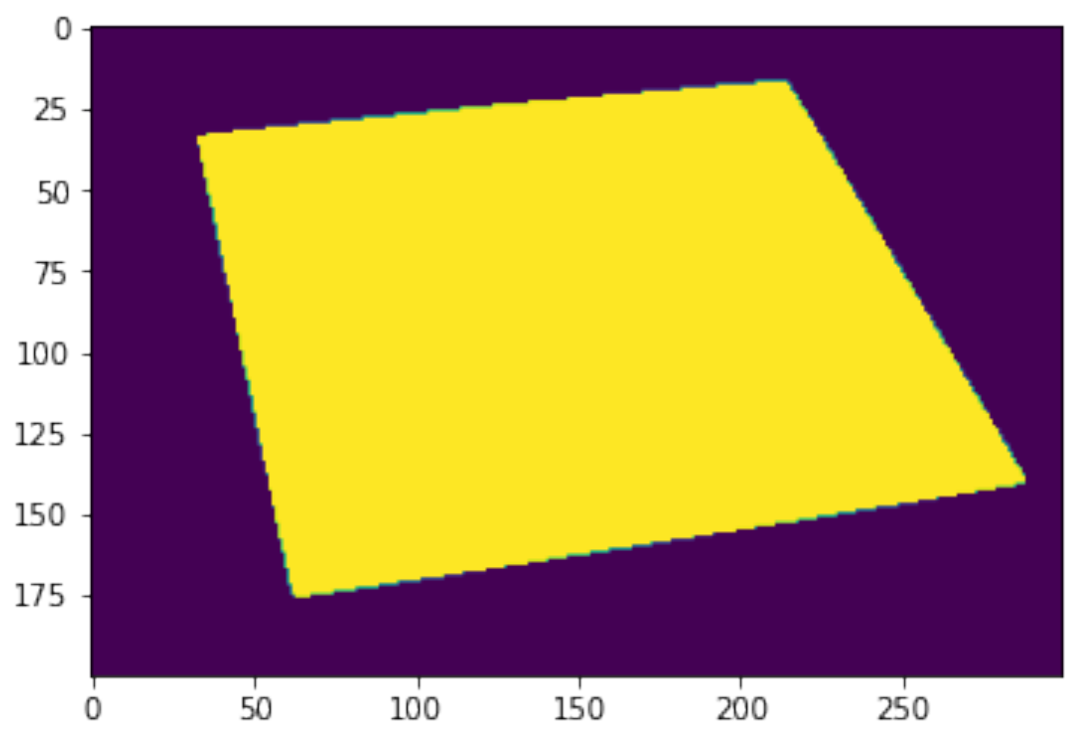
Given a set of **starting corners** (identified corners of the table) and **destination corners** (corners of a table whose sides are parallel to the sides of the image), the **warped** version of the input image is obtained. It is also **cut** so that only the table is present.

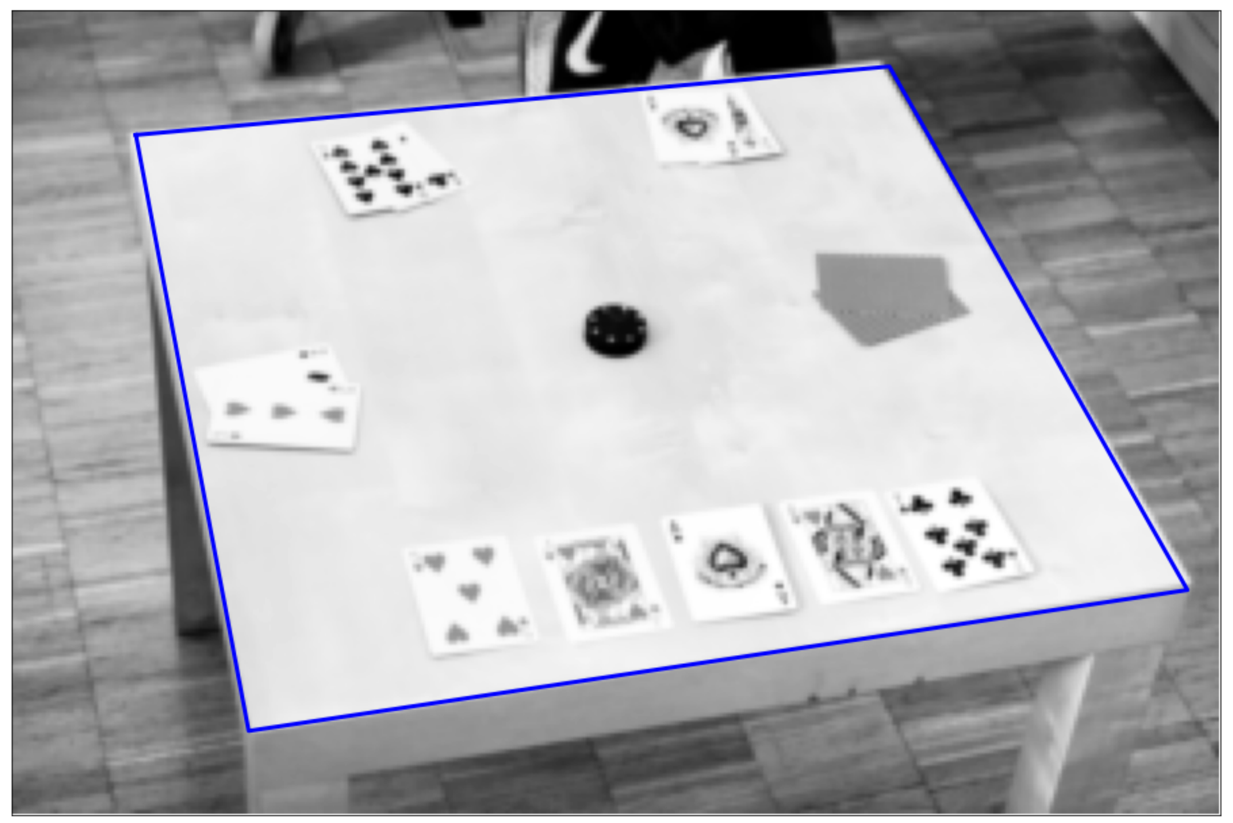
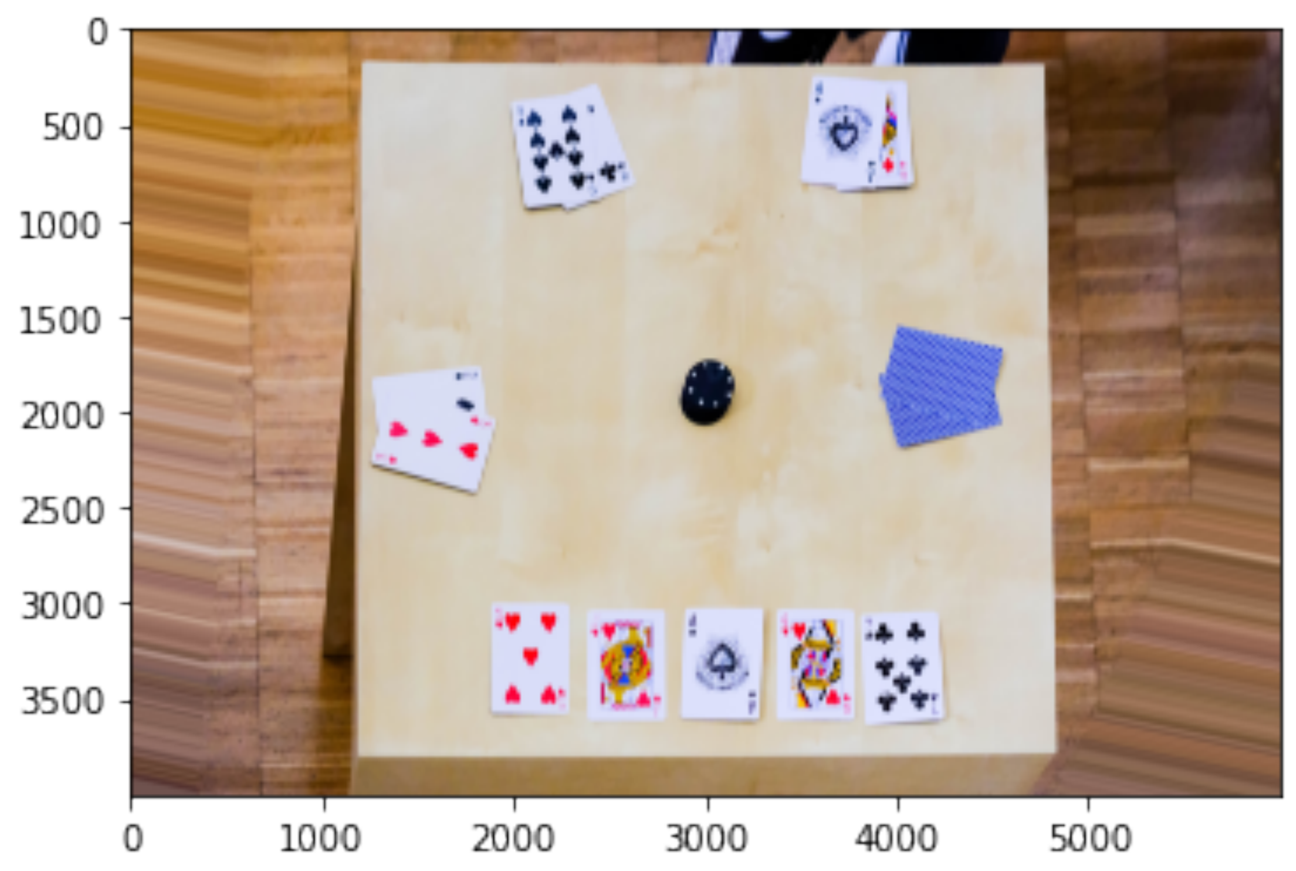
2. Warping and cutting of the image

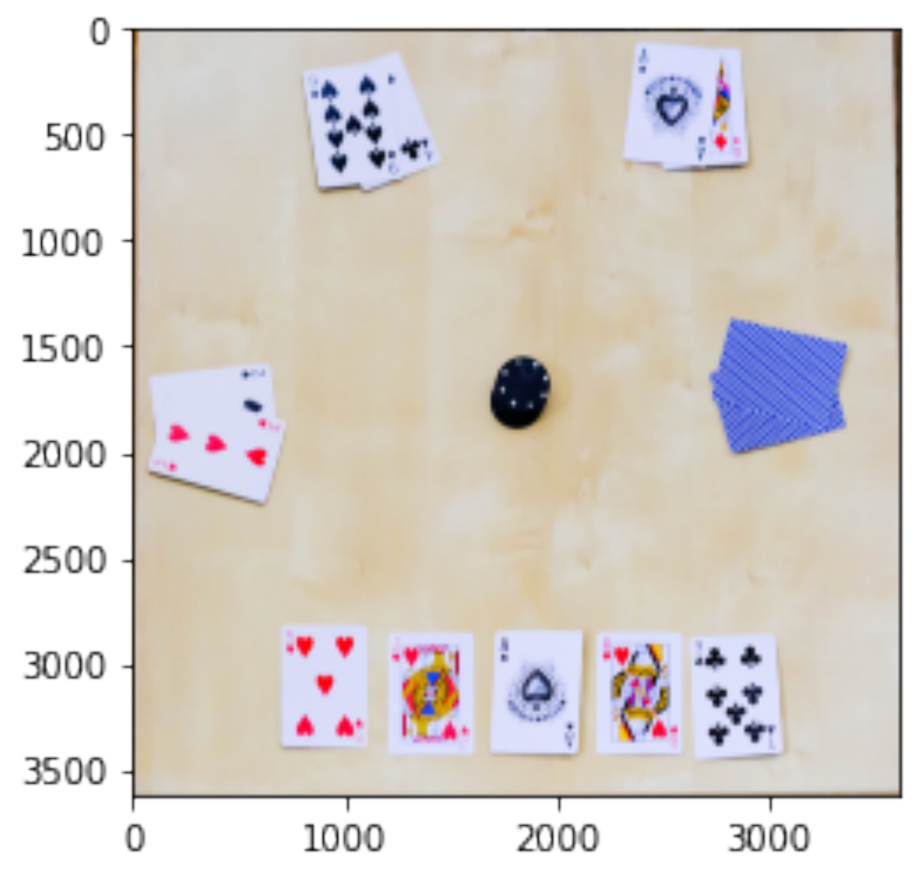
Example on the 'ultimate_test'





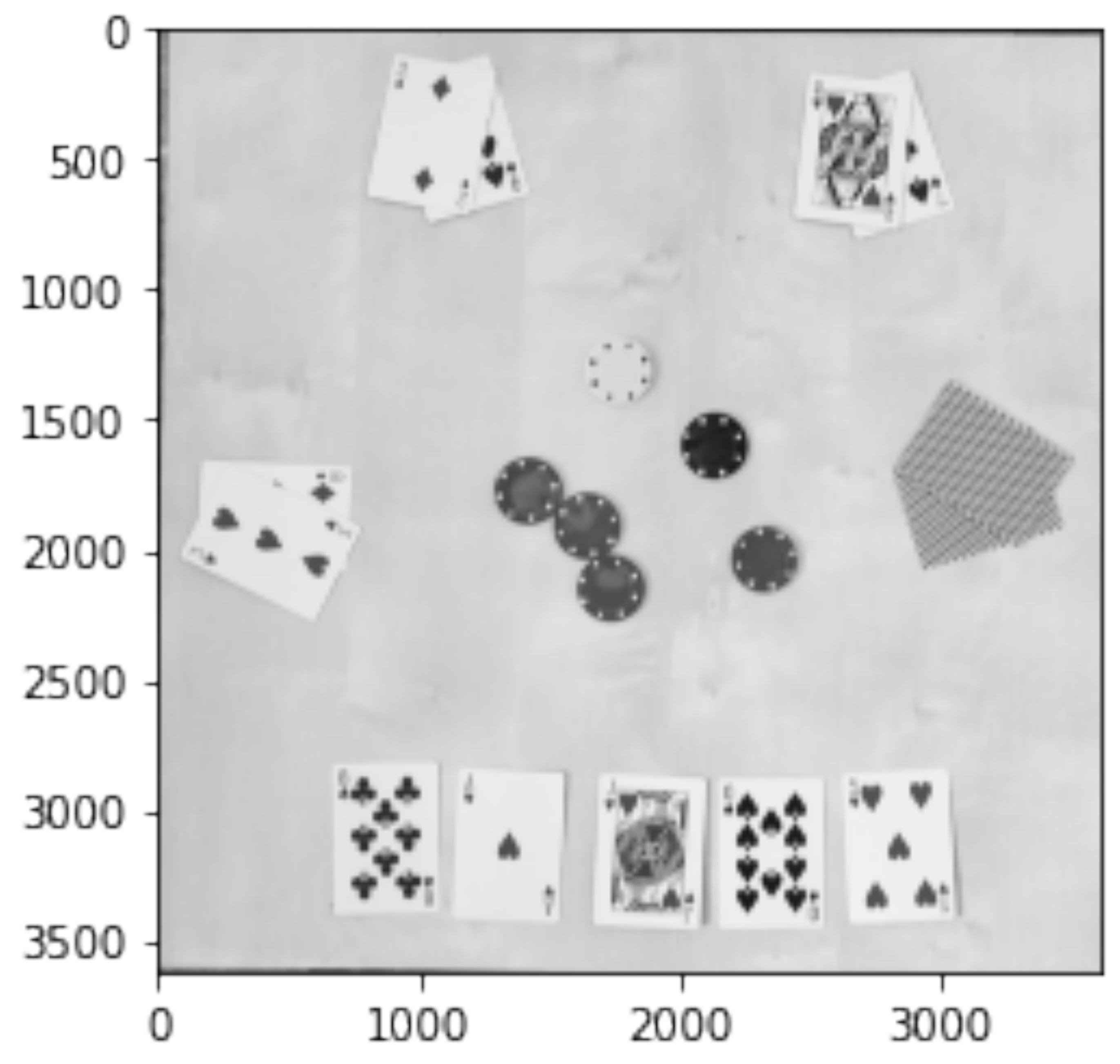


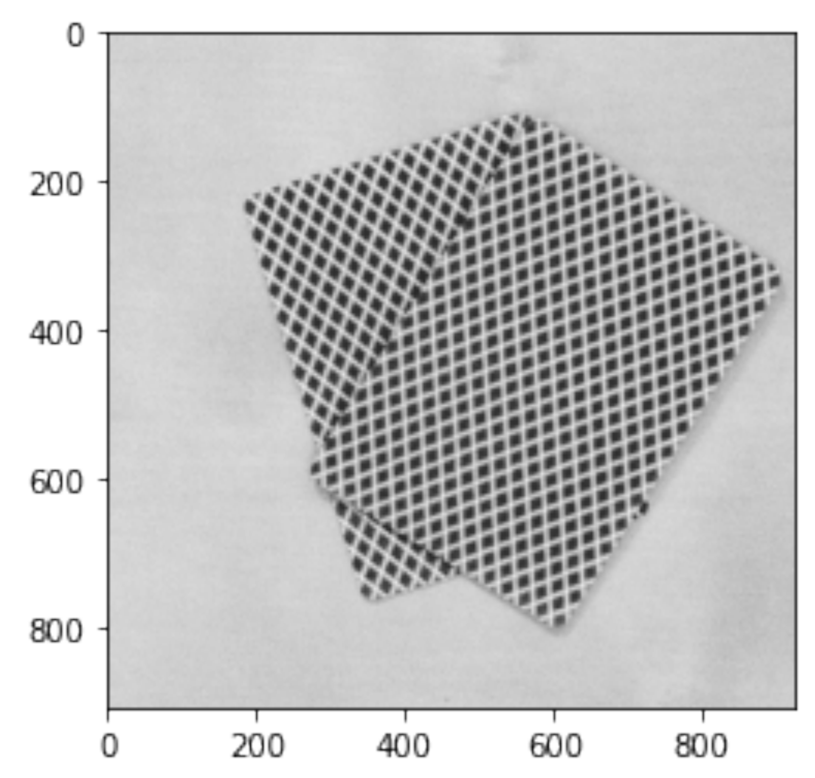



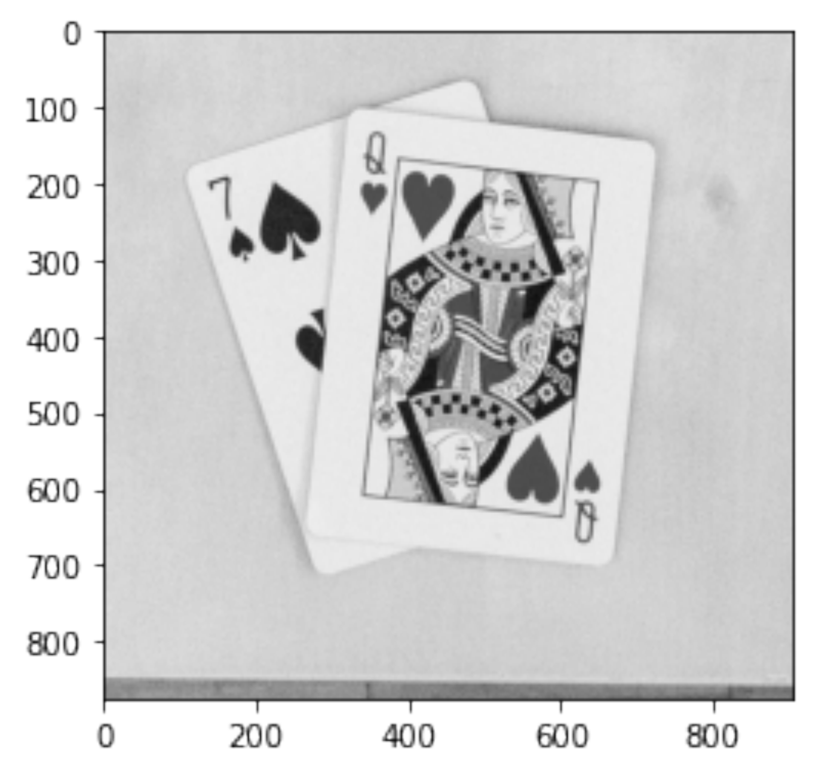


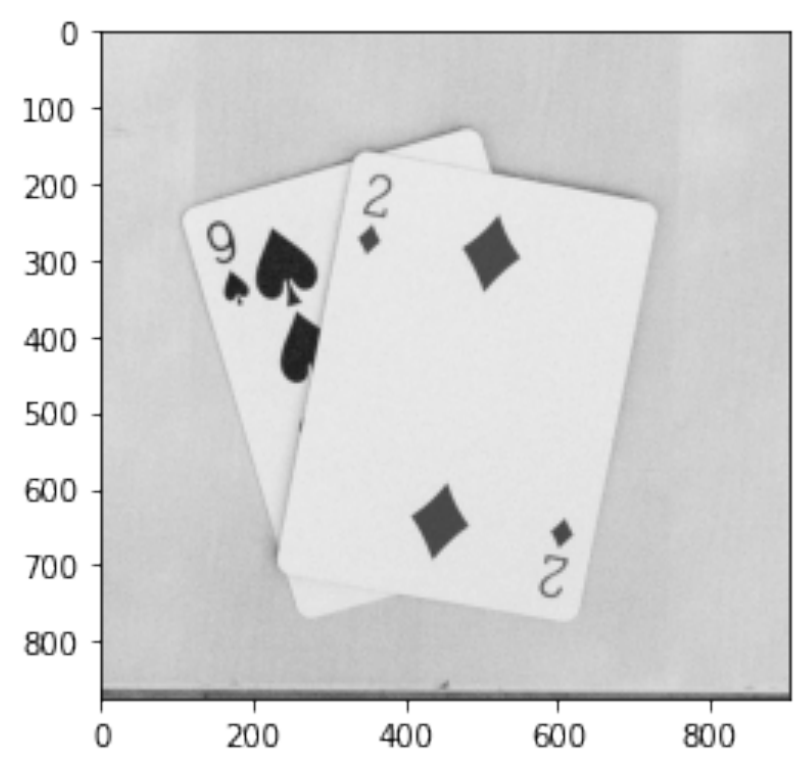
3. Splitting of the table sub-image in different sub-regions

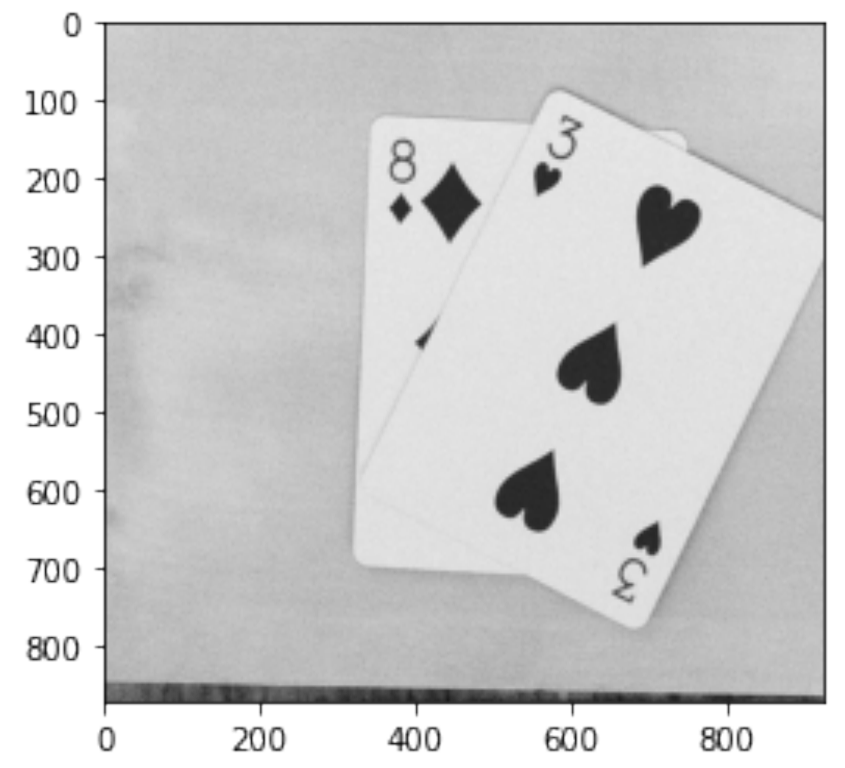
3. Splitting of the table sub-image in different sub-regions

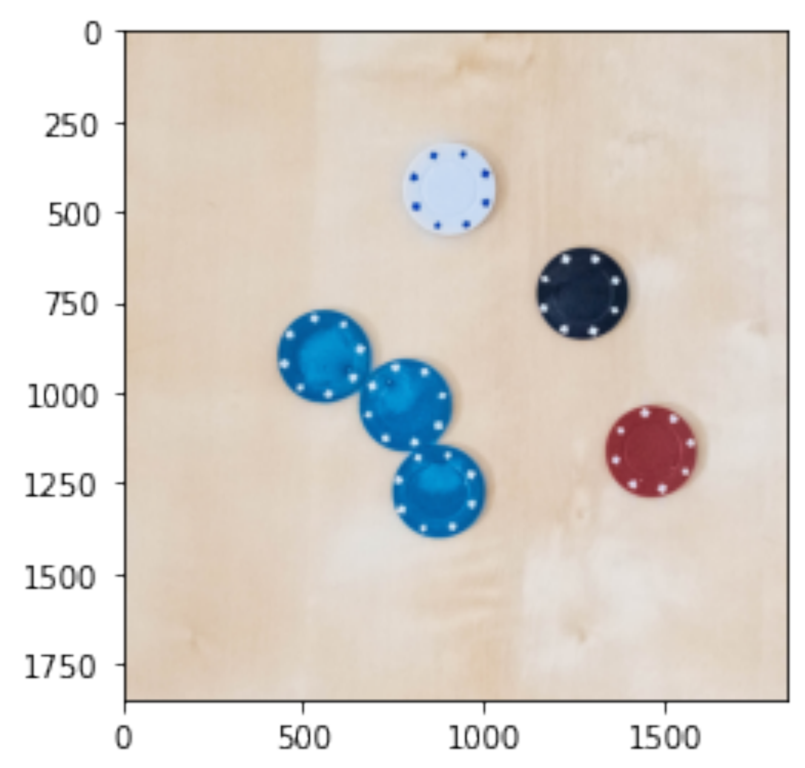








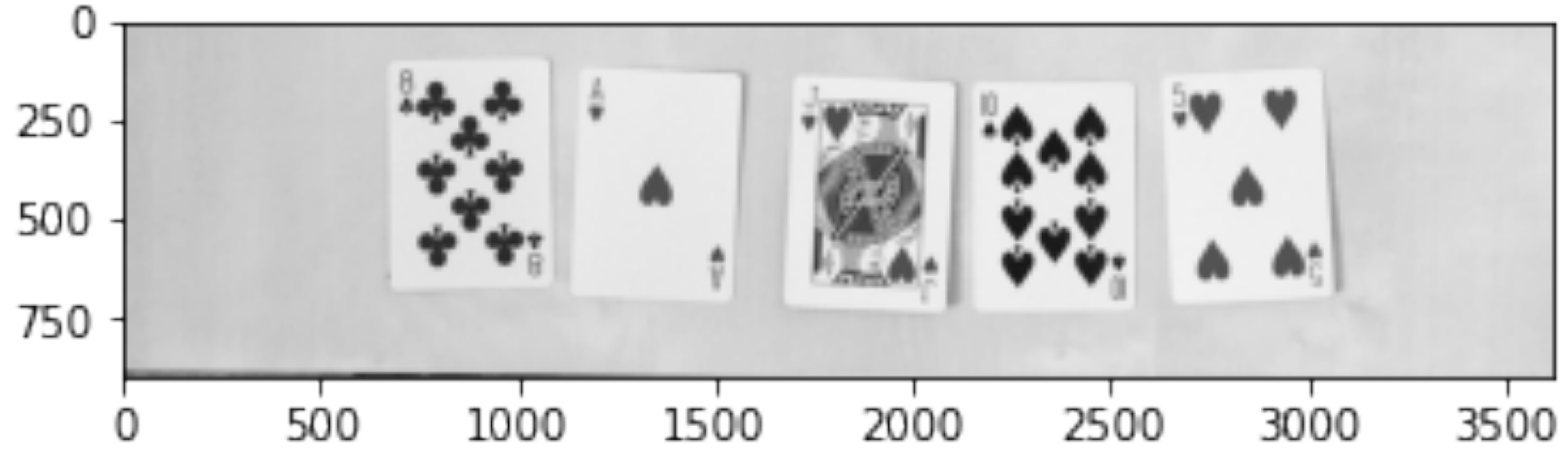


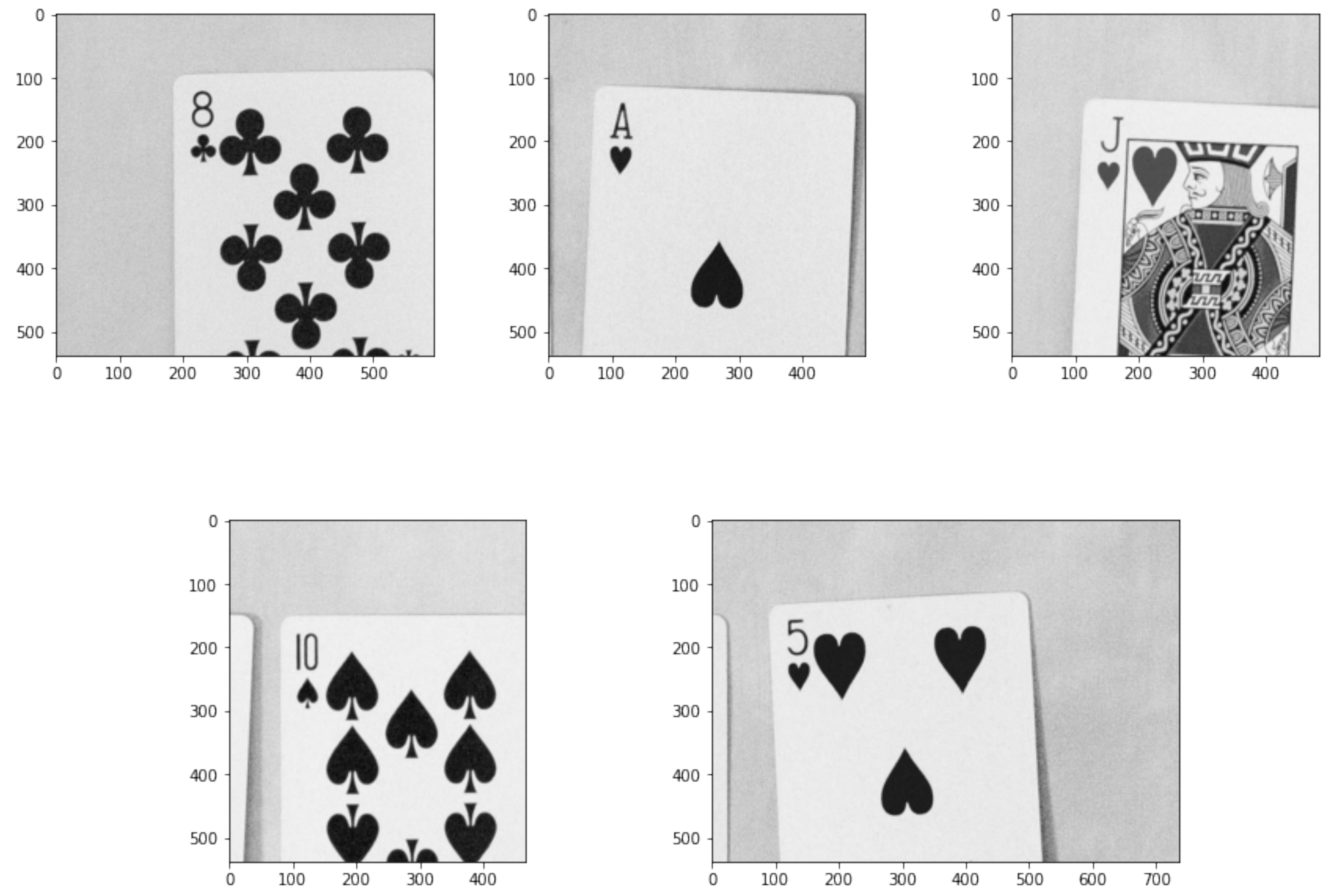


The **image** containing the entire table is **split** into different **sub-images**. The **split** into regions is **hard-coded** since it is possible to assume that the positions of the cards and the chips are approximately equal for the various images.

4. Splitting of the table cards region in five sub-images

4. Splitting of the table cards sub-image in five sub-images



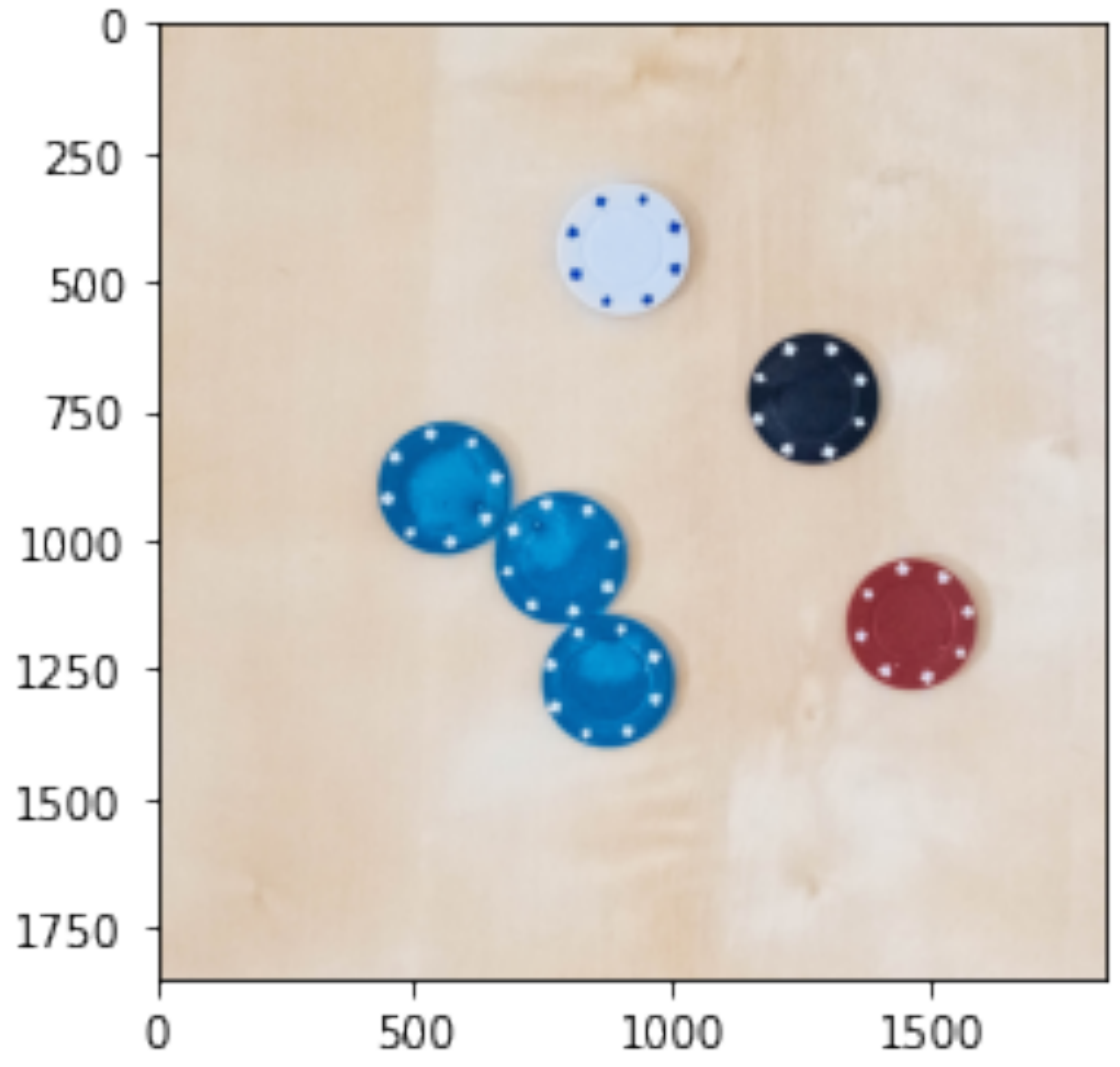


The **sub-image** containing part of the table and the **five individual cards** is further **split** into five sub-images containing the individual cards. Once again, the **split** into different regions is **hard-coded**.

5. Detection and classification of chips

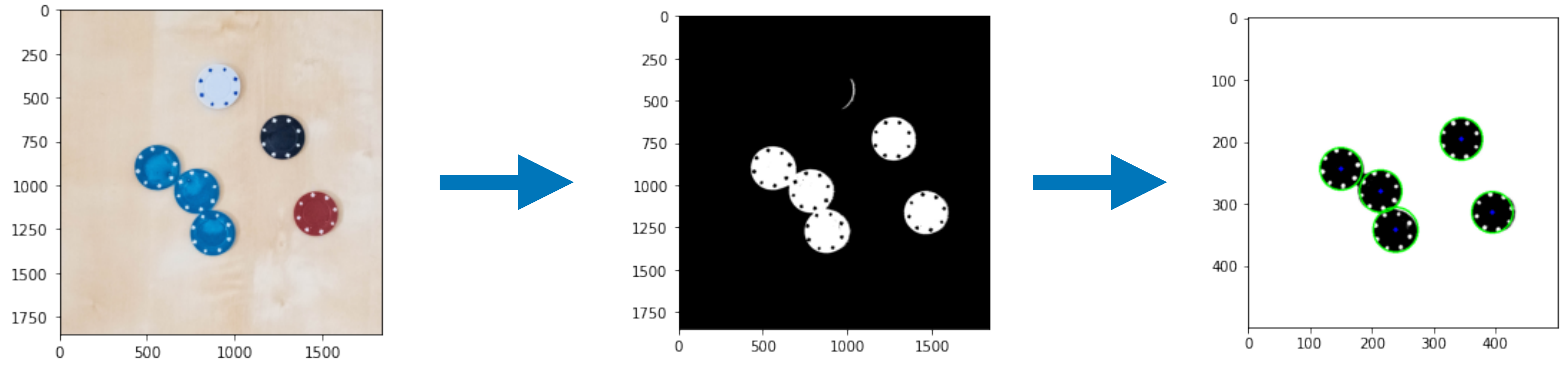
5. Detection and classification of chips

Goal



{'CR': 1, 'CG': 0, 'CB': 3, 'CK': 1, 'CW': 1}

5. Detection and classification of chips



- The sub-image is converted to **grayscale** by extracting the **L channel** of the image in the LAB colour space;
- The black, red, green, and blue **chips** are **separated from the table**. To do so, an **average grayscale colour of the table** is computed by taking into account the pixels at the border of the image.
- The **image** is **downsized**, and an **area closing** and a **dilation** are applied to remove eventual parts of the table/white chips;
- The **chips** are **detected** thanks to the **Hough circle transform**, whose parameters have been tuned so that the detection algorithm is very sensitive and can detect also partially occluded chips. This causes some false detections that are handled in the next steps of the algorithm.

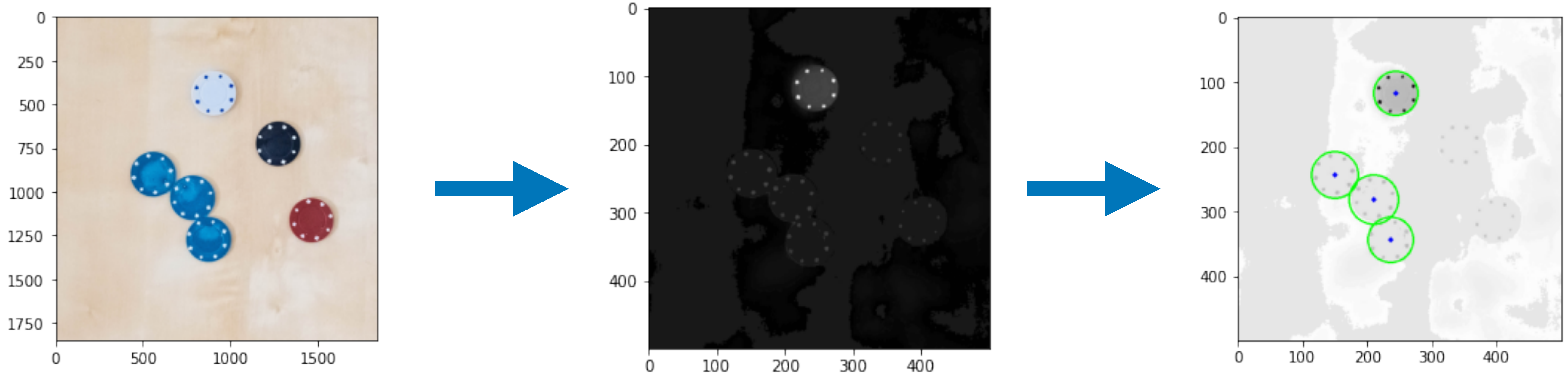
5. Detection and classification of chips

- Using masks and considering the coloured version of the sub-image, the **average colour of the inner part of each chip** is computed and is used to classify the chips.
- The **classification** is performed thanks to a **k-NN classifier**. The **features** taken into account during classification are four (four-dimensional feature vectors):
    1. The **average value of the R channel** of the inner part of the chip;
    2. The **average value of the G channel** of the inner part of the chip;
    3. The **average value of the B channel** of the inner part of the chip;
    4. The **average value of the L channel** computed over the full sub-image.

5. Detection and classification of chips

- In the k-NN classifier, we considered **k = 3**, and we obtained the labelled train set by extracting the chips from the train images and manually indicating the colour of each chip.
- Thanks to the classifier, we can count the number of black, red, green, and blue chips; we are also **able to discriminate false detections**, as we also inserted in the train set white chips and parts of the table erroneously classified as chips by the Hough circle transform.

5. Detection and classification of chips



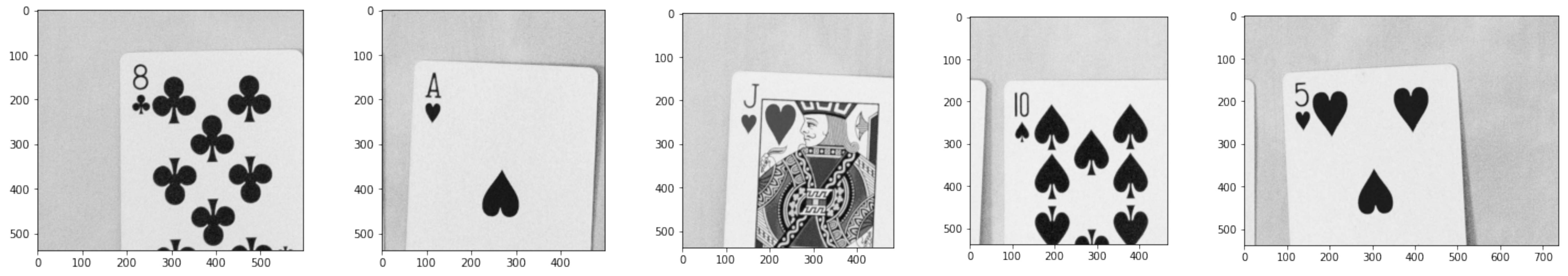
- To **detect** and **count** the number of **white chips**, the **B channel** of the sub-image containing the chips, converted into the LAB colour space, is considered;
- After a **resizing** and **masking** procedure, the white chips are much more visible than in the previous step and can be easily spotted using the **Hough circle transform** again.

- Once the chips are detected, only those that have not been identified before are considered. The **k-NN classifier** (k = 3) is used again to distinguish between the white chips and the table (mis-detected chips).

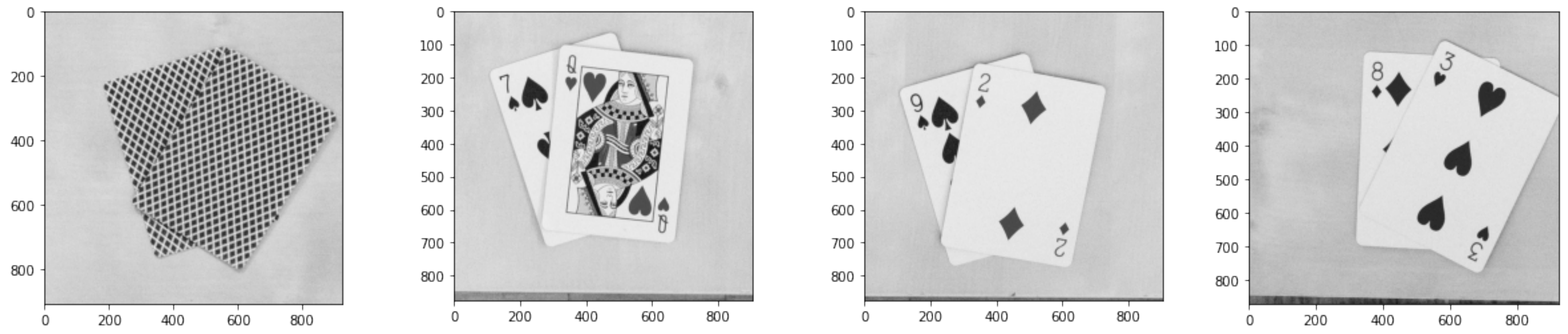
6. Classification of player and table cards

6. Classification of player and table cards

Goal



{'T1': '8C', 'T2': 'AH', 'T3': 'JH', 'T4': '10S', 'T5': '5H'}



{'P11': '0', 'P12': '0', 'P21': '7S', 'P22': 'QH', 'P31': '9S', 'P32': '2D', 'P41': '8D', 'P42': '3H'}

Classification of table cards

Classification of individual cards

- Starting from a sub-image containing a single card from the table, we aim at classifying the sub-image, identifying the **most probable number and suit present**;
- Basic idea: we would like to **avoid** any kind of **pre-processing** on the sub-image received, as the difference in lighting conditions and the similarity between the colours of the table and the cards would decrease the **robustness** of the card detection;
- Building block of the procedure: 'classify_card' function, called multiple times. This function returns the most probable number and suite present in the card displayed by the sub-image.

Classification of individual cards

- The algorithm at the basis of 'classify_card' is **template matching**, that allows to **match a series of templates** (extracted by hand from the 'spades_suits' and 'kings' images present in the 'image_setup' folder) **with the sub-image containing the card** and to return the best number match and the best suit match.

Classification of individual cards

- The metric used is '**cv.TM_CCOEFF_NORMED**', and each of the images received is compared with the templates containing the numbers A, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K (and the back of the cards, to be able to detect flipped cards) and the suits D, H, S, C;
- At the end of the comparisons, the function selects the **number template**, and the **suite template** identified with **higher confidence**. Furthermore, the function also identifies the **positions** in which the **best matches** have been detected, as this will be useful in the classification of the players' cards.

A
2
3
4
5
6
7
8
9
10
J
Q
K

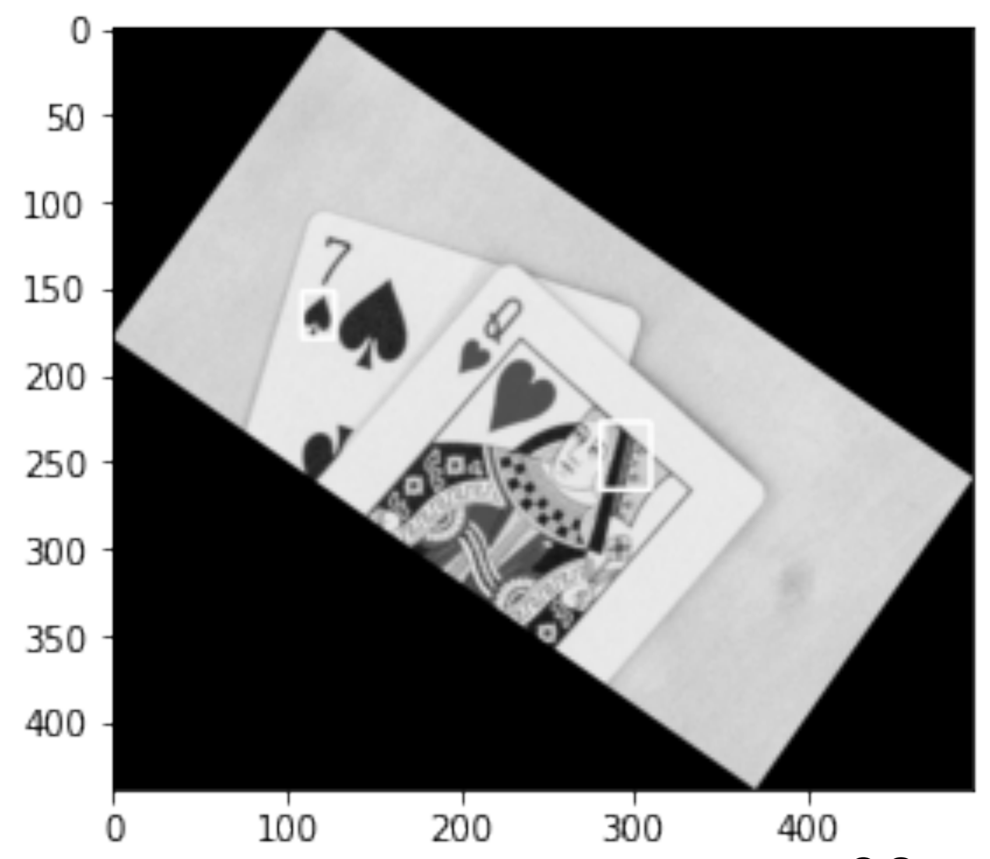
Classification of individual cards

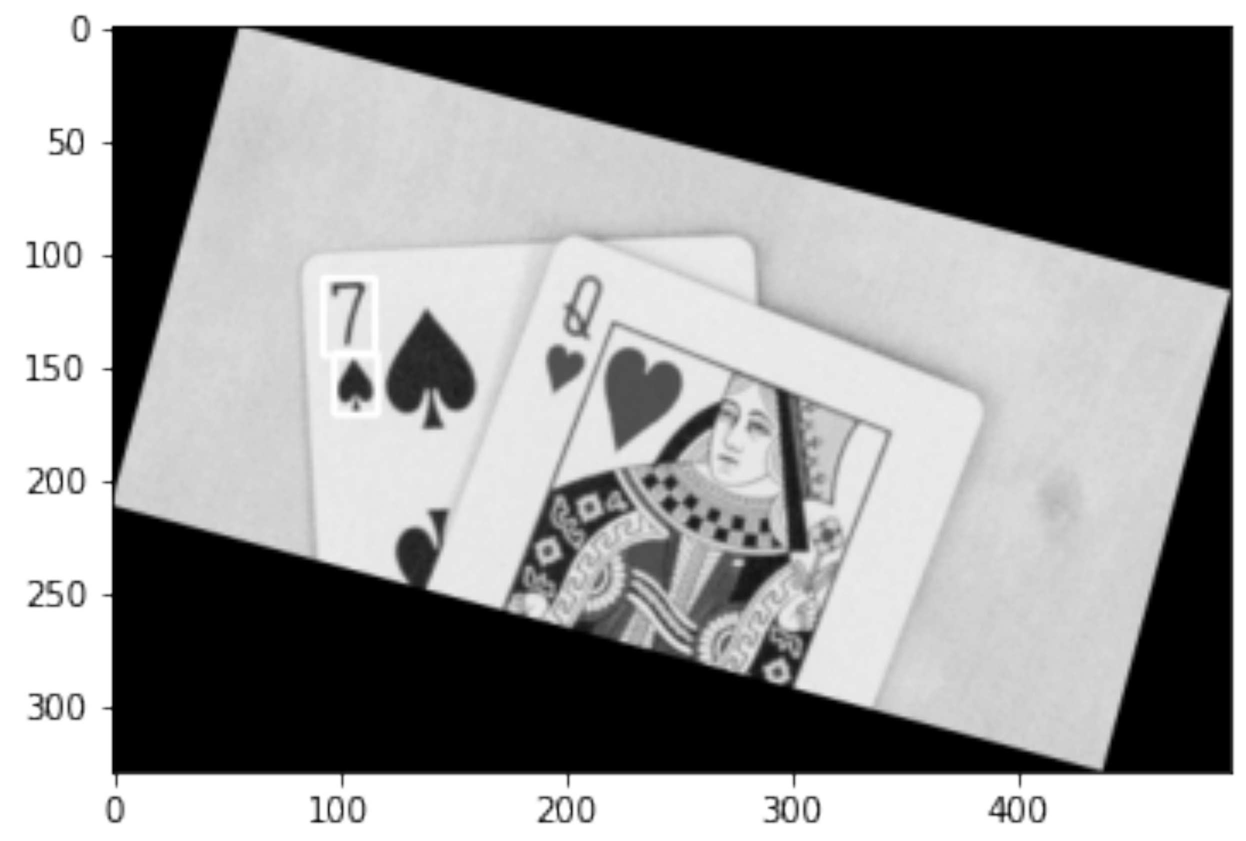
- In considering individual cards, the only difficulty is that the **card may be slightly rotated**, and the template match is not invariant to rotations;
- To solve the issue, the received **sub-image is rotated from -6° to 6° in steps of 2°**, calling each time the function 'classify_card', to which the rotated image is passed;
- **The rotation that yielded the higher confidence of the detected number is selected**, and the function 'classify_card' is called again, considering that rotation.

Classification of players’ cards

Classification of players' cards

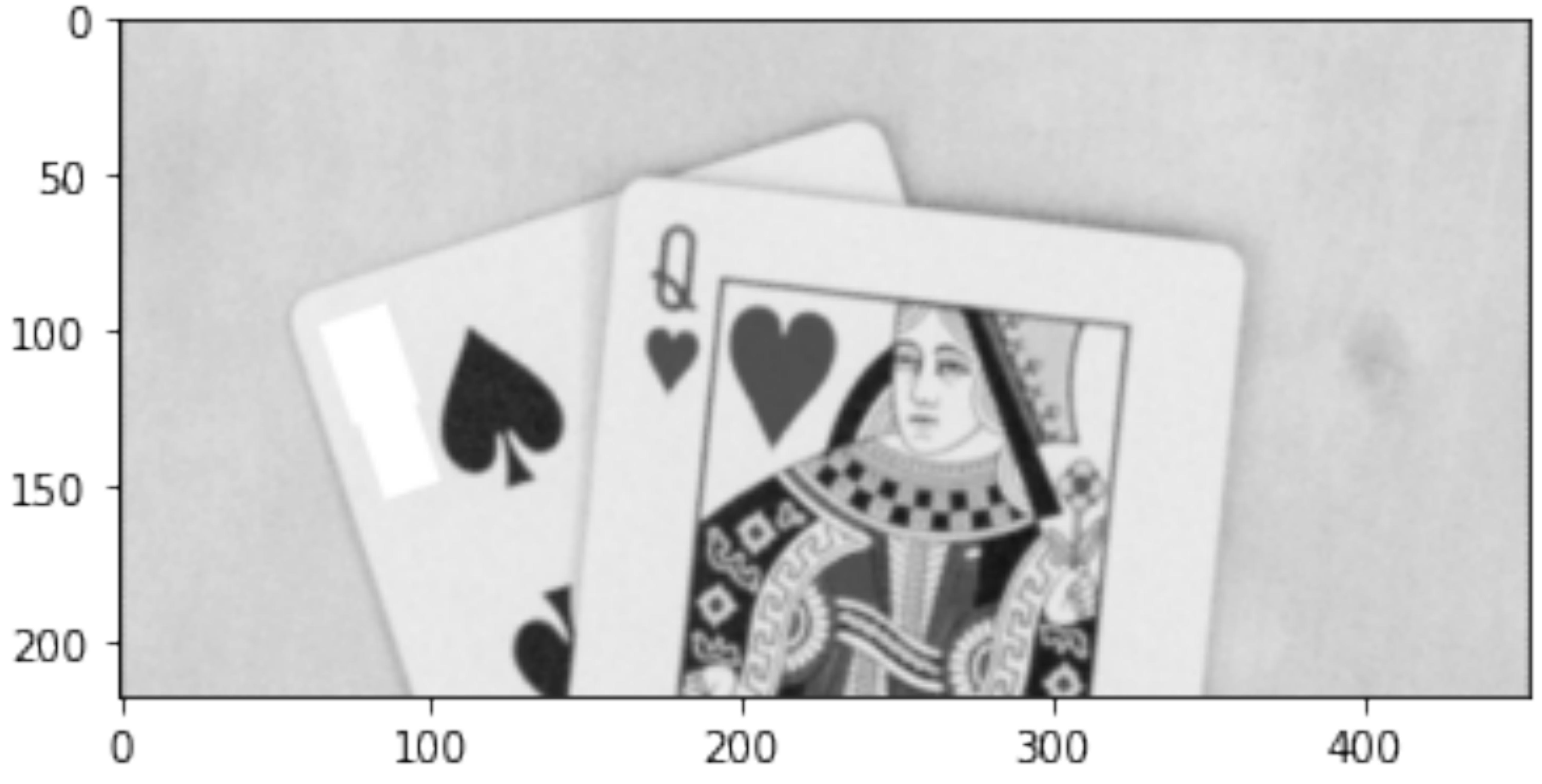
- The procedure followed for the individual cards is not sufficient in the case in which the sub-image contains the **two player cards**, as they are **rotated differently**, and as the **template match** should be **repeated** for each of the cards;
- To solve the first issue, we proceed in a way similar to the one used for the single card: the **sub-image is rotated from -35° to 35° in steps of 4°**, calling each time the function 'classify_double_card', to which the rotated image is passed.





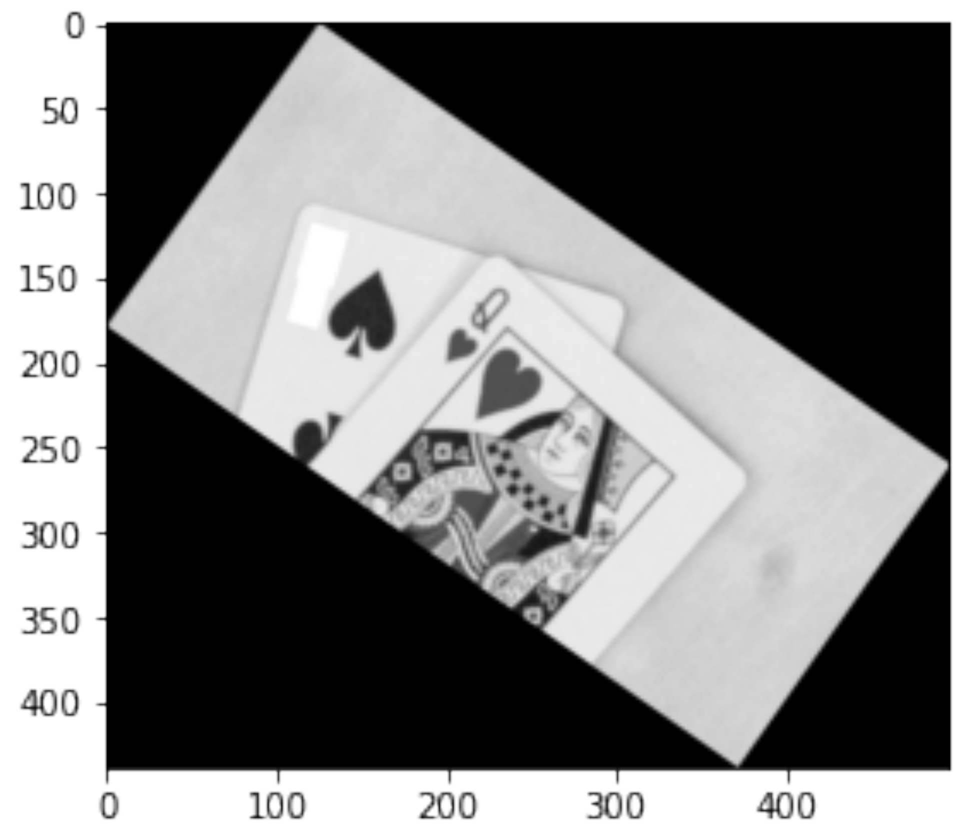
Classification of players’ cards

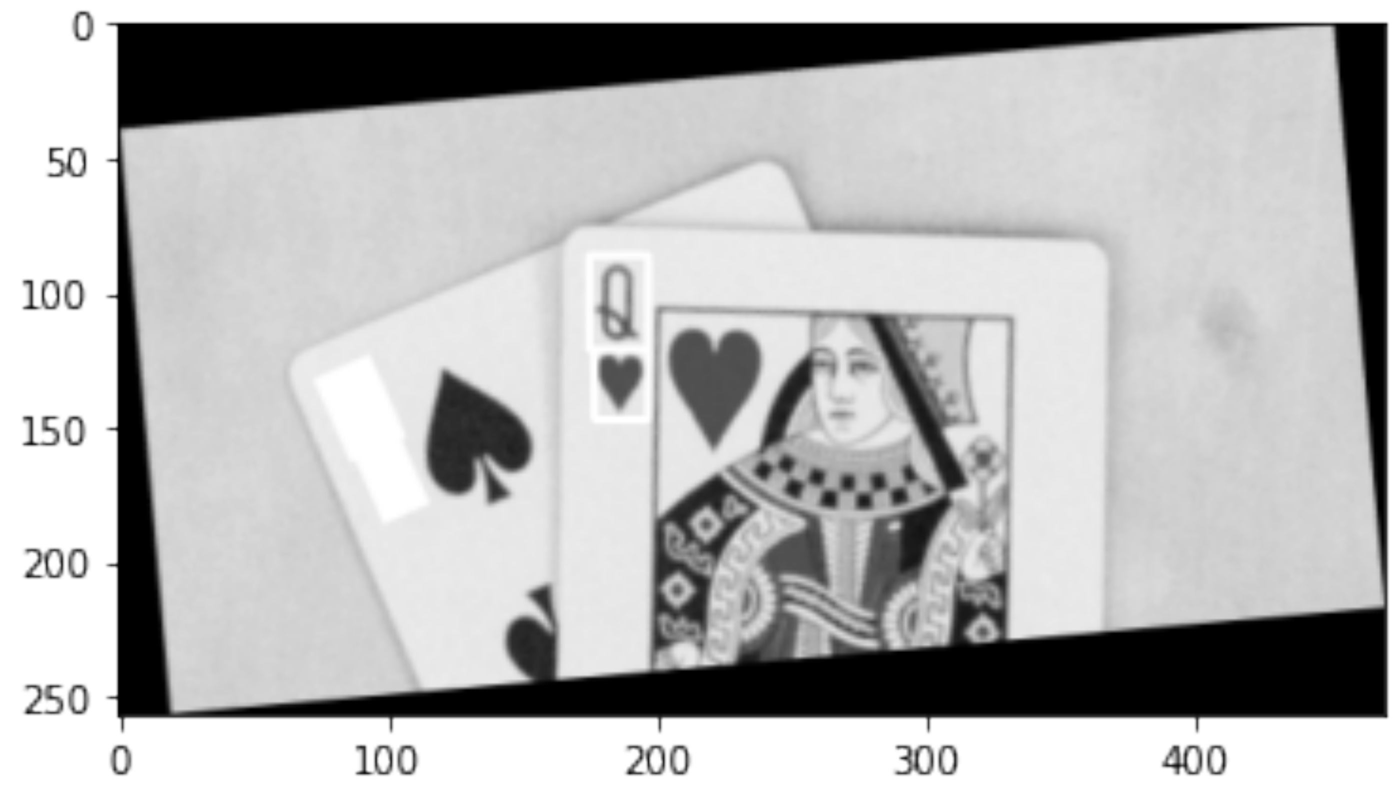
- This time, the function 'classify_double_card' is called twice for each rotation, and between the two calls, the **number and the suit identified in the first call are masked so that they are not identified again** in the second call.



Classification of players' cards

- Then, the **rotation** that yielded the **higher confidence of the detected number is selected**, and the function 'classify_double_card' is called again considering that rotation;
- It is also checked whether the number of a card and the suit of the other card have been identified. If this is the case, the **results** are **swapped**. Furthermore, the procedure returns as the **first result** the **leftmost card**, and as the **second result** the **rightmost card**, as this is the format expected as output.





7. Results are returned in a dictionary

```
{'T1': '8C', 'T2':
'AH', 'T3': 'JH', 'T4':
'10S', 'T5': '5H'}
```

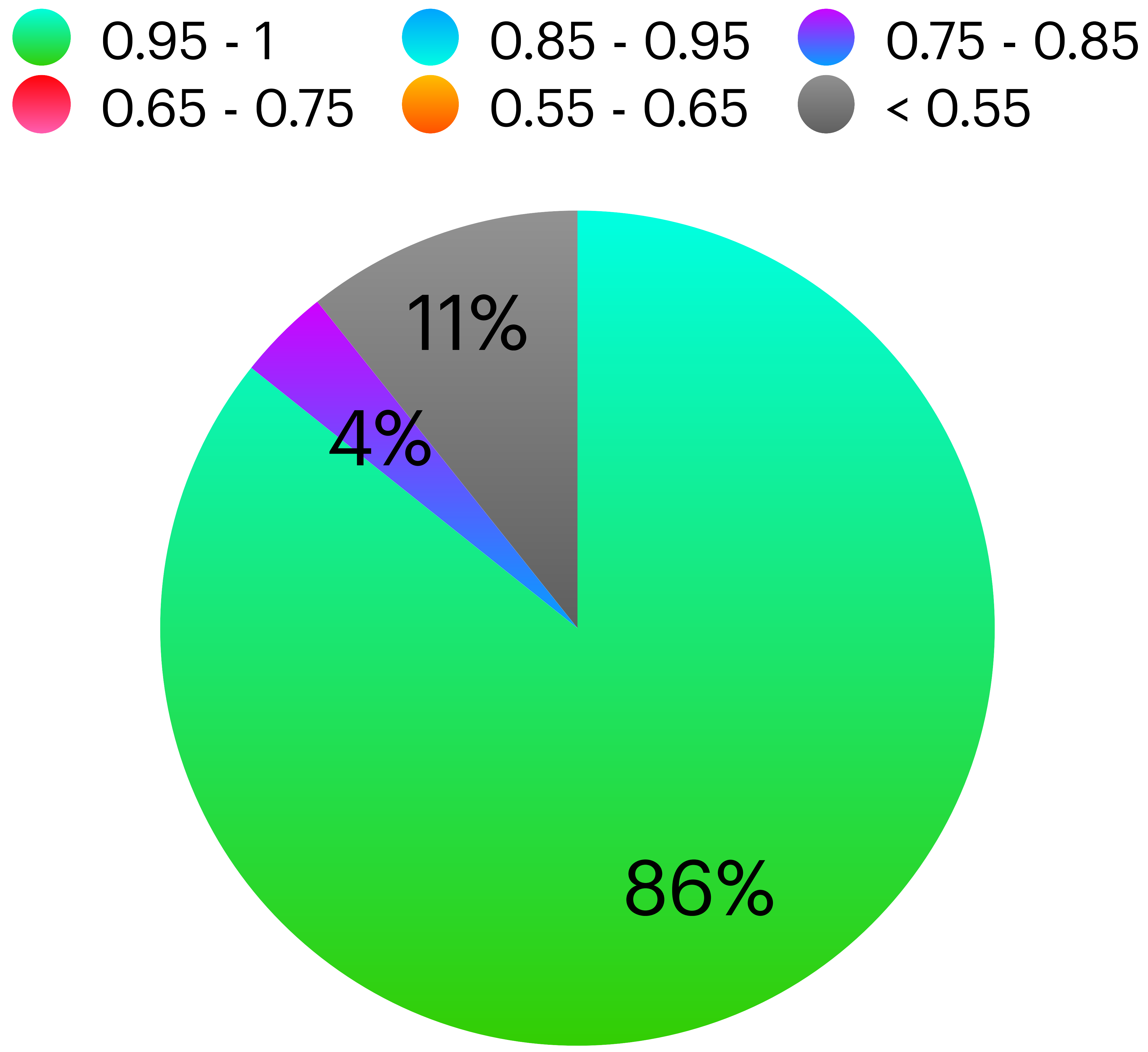
```
{'P11': '0', 'P12':
'0', 'P21': '7S',
'P22': 'QH', 'P31':
'9S', 'P32': '2D',
'P41': '8D', 'P42':
'3H'}
```

→

```
{'T1': '8C', 'T2': 'AH',
'T3': 'JH', 'T4': '10S',
'T5': '5H', 'P11': '0',
'P12': '0', 'P21': '7S',
'P22': 'QH', 'P31': '9S',
'P32': '2D', 'P41': '8D',
'P42': '3H', 'CR': 1,
'CG': 0, 'CB': 3, 'CK': 1,
'CW': 1}
```

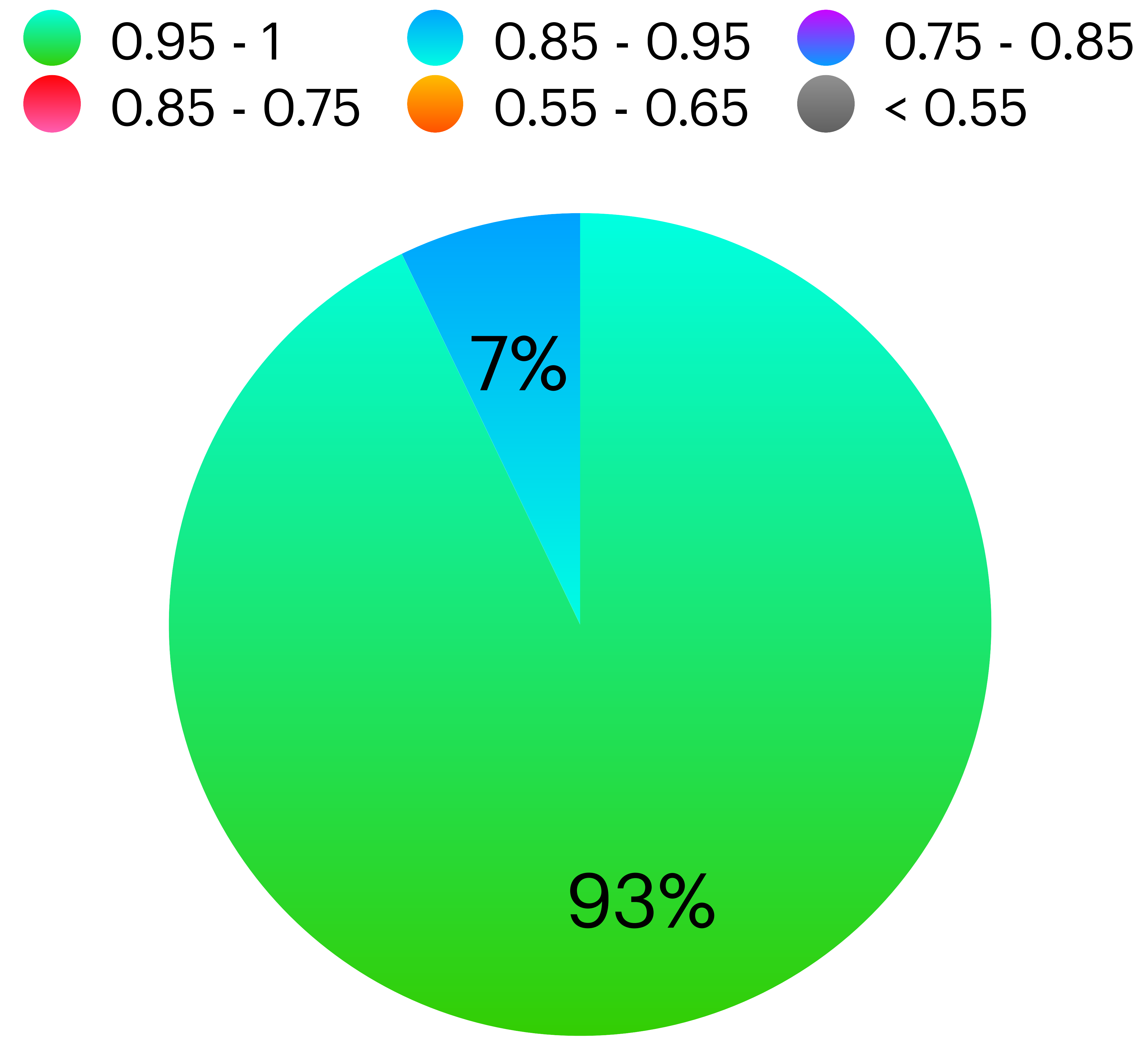
Statistics (on the train set)

Chips accuracy score



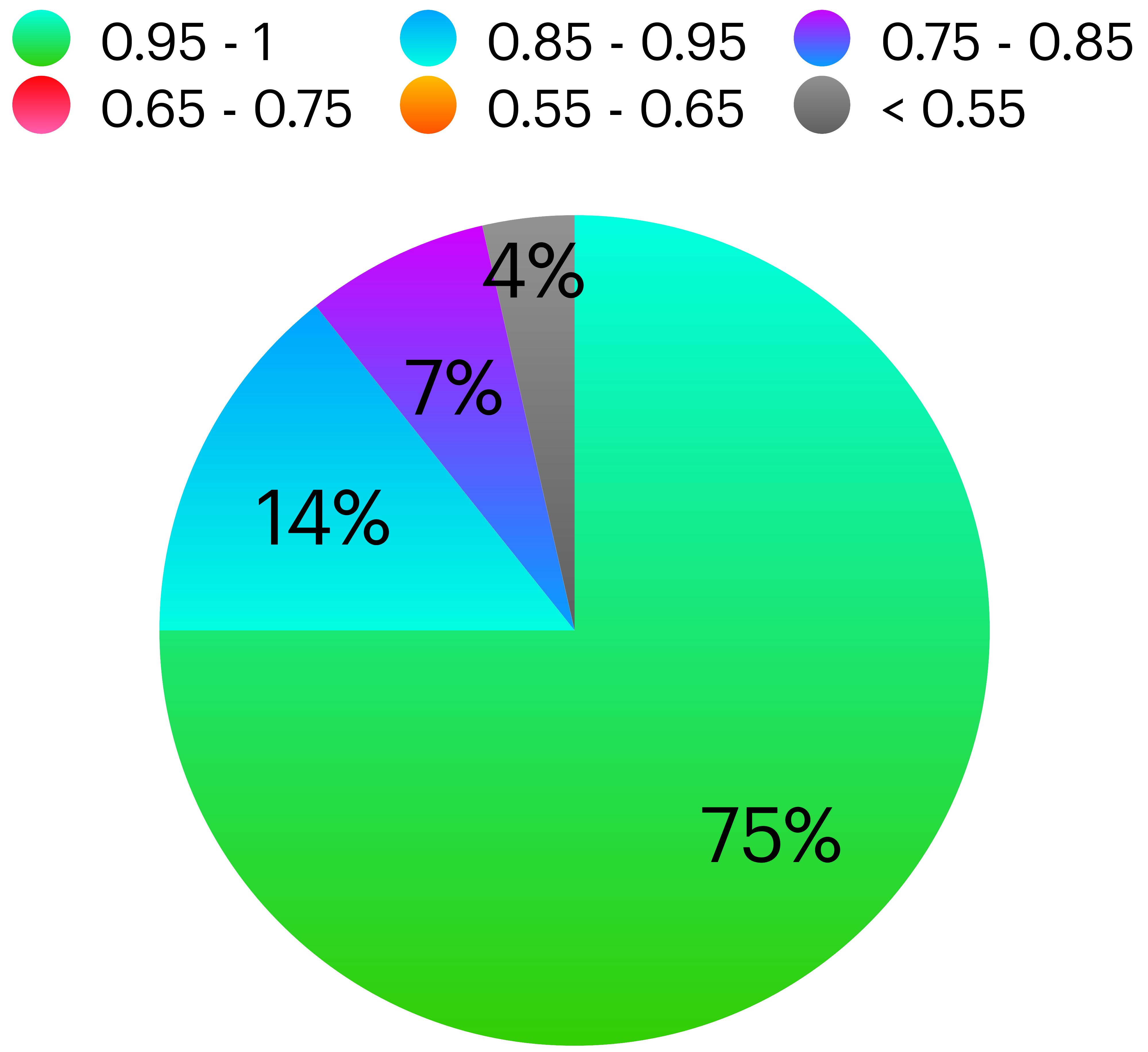
Statistics (on the train set)

Cards number accuracy score



Statistics (on the train set)

Cards suit accuracy score



Statistics (on the train set)

Average accuracy scores

Chips average accuracy score (on train set)	0.8929
Cards numbers average accuracy score (on train set)	0.9918
Cards suits average accuracy score (on train set)	0.9614
Average score (on train set)	0.9487

Thanks for your attention!