COUNTING CARD SYSTEM FOR BLACKJACK

COMPUTER VISION - HOMEWORK 2

Davide Baggio
Department of Computer Engineering
University of Padua

Zoren Martinez
Department of Computer Engineering
University of Padua

Francesco Pivotto
Department of Computer Engineering
University of Padua

1 Introduction

This project focuses on the development of a Computer Vision-based system for automatic card recognition in blackjack games. The primary objective is to assist casino operations by detecting unauthorized card counting activities through real-time visual analysis. The system captures and processes video frames to identify individual cards on the table, localizing their positions and extracting their face values. Suit recognition is omitted, as it is not relevant to standard card counting strategies. Each recognized card is assigned a corresponding value based on the High-Low counting system.

Figure 1: Screenshot of a video frame with the recognition displayed

Implementation Notes

- All tests and evaluations were conducted on a selected portion of the video [6]. The annotations used for performance evaluation were created manually using CVAT [2].
- A separate Python script was used exclusively for the deep learning component focused on recognizing the card numbers and characters.
- If a different video name is provided as a command-line argument, the program will perform card detection without running the evaluation. If no argument is passed, the program will run the detection on the video used to create the ground truth data and display the evaluation results.

The core pipeline of our project is structured into three main stages:

1. Preprocessing
2. Card Shape Recognition
3. Card Value Recognition

Each of these components is detailed in the following sections.

IMPORTANT: To avoid bloating the main codebase, we chose not to include certain large external libraries directly in the repository. These dependencies are automatically downloaded and installed during the execution of the build.sh script. Specifically, the system relies on libtorch [7] for character recognition and json.hpp [5] for parsing the JSON annotation files made via CVAT. Additionally, the pre-trained model used for character recognition is also downloaded automatically during the build process, as it is too large to be stored in the repository directly.

Build and Execution

To compile and download the necessary files, run the following command from the terminal:

```
sh build.sh
```

To execute the program:

```
./build/bin/cv_detection
```

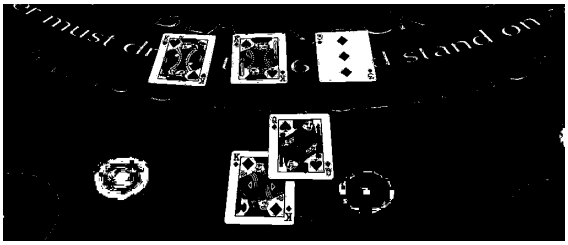
2 Preprocessing

In our computer vision project for blackjack, we use two distinct preprocessing pipelines to optimize system efficiency. The first one operates on the full frame to detect the position of the cards on the table. The second, lighter preprocessing is applied only to the individual cards already detected, in order to enhance their visibility and improve recognition accuracy (e.g., for identifying value and suit).

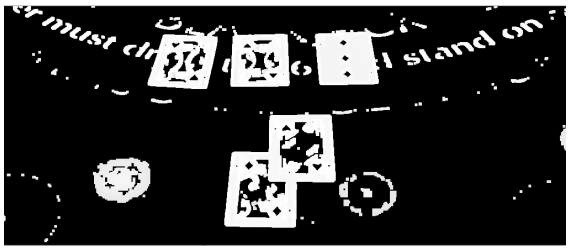
Full Image Preprocessing

This stage is designed to accurately locate the cards on the table, even under challenging conditions such as overlapping or poor lighting. It consists of the following four steps:

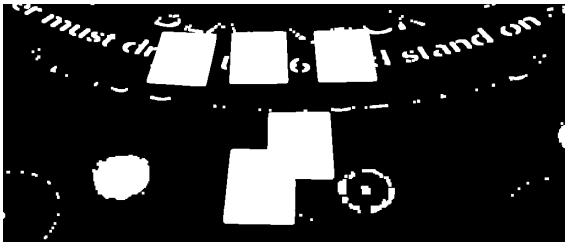
- **White pixel extraction:** Pixels with high brightness values are selected, as they typically correspond to the surface of the playing cards.
- **Morphological dilation:** Bright areas are expanded to produce clearer and more continuous contours. This is especially helpful for illustrated cards, which often contain large dark regions that can hinder consistent edge detection, particularly when cards overlap.
- **Contour detection and filling:** Connected components made of white pixels (potential card regions) are identified and filled to create solid shapes that are more robust to subsequent operations.
- **Morphological erosion:** A final erosion step is applied to smooth the contours and remove small unwanted elements, such as white letters on the table that may otherwise be misclassified as part of a card.



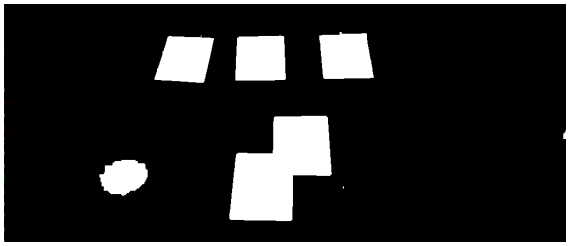
(a) Image after white pixel Extraction



(b) Image after Dilatation



(c) Image after Contour Filling



(d) Final result after Erosion

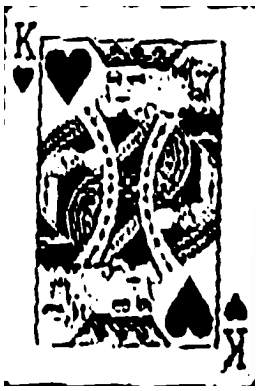
Figure 2: Preprocessing steps applied to the card detection pipeline

Individual Card Preprocessing

Once the cards have been detected, each card undergoes a lighter preprocessing routine focused on improving content visibility and recognition reliability. The following five steps are performed:

- **Grayscale conversion:** The image is simplified by removing color information, retaining only intensity values.
- **CLAHE (Contrast Limited Adaptive Histogram Equalization):** Local contrast is enhanced to highlight important details, even under uneven lighting conditions.
- **Sharpening:** A kernel is applied to increase image sharpness, enhancing edges and relevant features.
- **Gaussian blur:** A light blur reduces noise and minor imperfections while preserving essential information.
- **Otsu thresholding:** The image is automatically binarized to clearly separate card symbols from the background.

(a) Original card image



(b) Processed card

Figure 3: Comparison between a card before and after the preprocessing pipeline

3 Card Shape Recognition

This card processing approach is inspired by the work of Wesley Cooper [1], who proposed a method for detecting and analyzing playing cards in real-time gaming scenarios. Building on this idea, our method introduces additional steps for robustness under overlapping and noisy conditions.

The card processing phase begins with the detection of contours of the connected components in the preprocessed image. Once all contours are extracted, a filtering step is applied: contours with an area smaller than 3000 pixels or a perimeter shorter than 250 pixels are discarded, as they are considered noise or irrelevant objects.

For each remaining contour, two key points are identified: the top-right extreme and the bottom-left extreme, defined as the contour points closest to the top-right and bottom-left corners of the image, respectively. These points are used to draw a diagonal line across the card. The algorithm then searches for local maxima along the contour, which are points whose distance from the diagonal is greater than a fixed number of their neighboring points. These maxima approximately correspond to the card corners. Since each card should yield two such maxima, the total number of detected maxima is expected to be a multiple of two.

For each pair of detected maxima, a set of four points is defined to represent the card corners: the two local maxima and two additional extremes computed from their geometric configuration. A small pixel tolerance is applied to these points to compensate for slight contour shrinkage caused by the morphological erosion step during preprocessing.

The final result of this process is a set of quadrilaterals that accurately represent the visible cards in the image, ready to be extracted, normalized, and passed to the recognition pipeline.

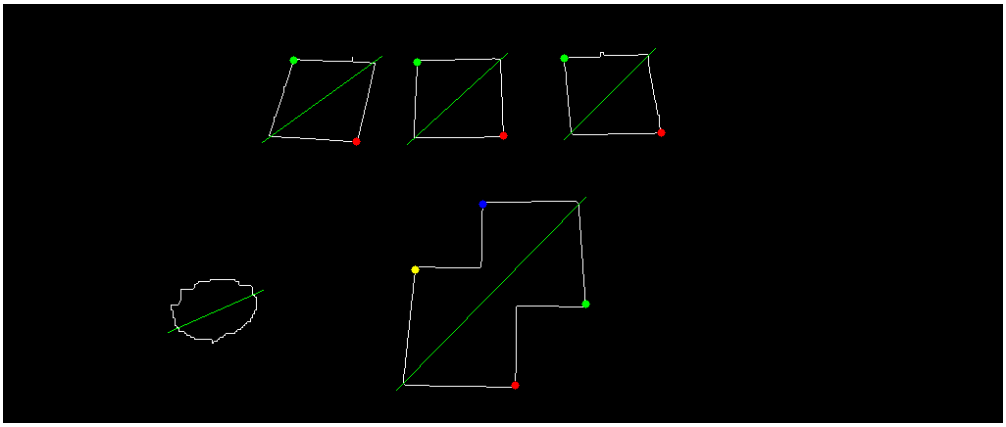


Figure 4: Example of the card contour processing result.

4 Card Value Recognition

Patch extraction

Once the processed image of an individual card is obtained, the value recognition step begins. A fixed-size patch is extracted from the top-left corner of the card, where the rank symbol is typically located. This region is then refined by filtering for the most central contours within the patch, in order to reduce the impact of inaccurate region selection, such as parts of the background or card suits, which are irrelevant for our purposes. The final output is a binarized image focused on the card's number or letter symbol.

Figure 5: Example of the extracted patches

For the recognition of card values, we tested multiple approaches. Traditional methods such as template matching and SIFT were evaluated but did not produce satisfactory results. Ultimately, we opted to train a CNN model. Since no annotated dataset containing only card values was available, and manual annotation was impractical, we decided to generate the dataset synthetically from fonts. The goal was to create a letter dataset that closely resembles the characters and symbols extracted from the image patches.

Font and Dataset Generation

The program generates a synthetic dataset by rendering images of card characters using two distinct fonts: "Khepri font" [4] for the "10" and "Card Characters Font" [3] for all the other ones. The character set includes numeric values (2–10) and face cards (A, J, Q, K). The "10" card is handled as a special case by combining two separate glyphs ('1' and '0') with geometric transformations to better approximate its typical visual representation. Each character is rendered on a fixed-size white canvas before augmentation.

Figure 6: Font used for the card values

Data Augmentation Techniques

To improve model robustness and simulate realistic capture conditions, multiple data augmentations are applied to the generated images.

These include:

- **Random scaling**: scaling the image between 1x and 1.2x.
- **Random shifting**: translation shifts up to ±15% of the image dimensions.
- **Perspective warping**: applying small random perspective distortions.
- **Random rotation**: rotation within ±20 degrees for most characters.
- **Morphological closing**: reducing noise by closing small holes in the binary images.
- **Resizing down and up**: simulating blur or pixelation effects.
- **Final binarization**: applying a threshold to convert images to binary format.

Figure 7: Example of the generated characters used to train the model

The model was trained on 800 synthetic images generated per class.

Convolutional Neural Network Architecture

The classification model is a convolutional neural network (CNN) with the following structure:

- **Feature extractor**: three convolutional layers with ReLU activation and max pooling, progressively reducing spatial dimensions while increasing feature depth.
- **Classifier**: two fully connected layers following a flattening operation, ending with a linear output layer sized to the number of card classes (13).

The model is trained using the Adam optimizer and cross-entropy loss for 10 epochs, showing steady improvements in classification accuracy. After training, the model weights are saved and a TorchScript version is exported.

5 Results and Performance Evaluation

The performance evaluation in this project is based on comparing the system's predictions with manually annotated ground truth data provided via CVAT, using the COCO JSON format. Specifically, for each predicted polygon, the Intersection over Union (IoU) is computed against all ground truth polygons in the corresponding frame. A prediction is considered correct if:

- **The IoU** exceeds a predefined threshold (default: 0.5), and
- **The predicted label** (i.e., the card value) matches the ground truth label.

Using these matches, the following standard metrics are computed:

- **Precision**: the proportion of correct predictions over the total number of predictions.
- **Recall**: the proportion of correct predictions over the total number of ground truth instances.
- **F1 Score**: the harmonic mean of precision and recall, particularly useful for imbalanced datasets.

The evaluation is performed on a frame-by-frame basis across the entire video.

Metric	Value
Precision	0.940
Recall	0.681
F1 Score	0.790

Table 1: Performance evaluation of the card value recognition.

The system achieves a high precision of **0.940**, indicating that the vast majority of recognized card values are accurate. This reflects a robust detection pipeline capable of minimizing false positives, even in a visually complex environment.

The recall of **0.681** suggests that while most visible cards are successfully identified, there is still room for improvement in detecting all instances particularly in challenging visual conditions. The overall F1 score of **0.790** confirms a solid balance between accuracy and completeness, demonstrating that the approach is well suited for reliable card recognition under real-world constraints.

6 Challenges and Limitations

Several practical challenges emerged during system development, primarily due to the visual complexity of real game environments.

- **Table Markings** The poker table features large white printed text that can be misinterpreted as card contours during preprocessing, leading to incorrect detections.
- **Low Video Resolution**: The use of 720p footage introduces noise and limits detail, further complicating recognition, especially for small characters or stylized fonts.
- **Patch Extraction Issues**: Accurate card value recognition depends on precisely extracting the card's rank region. Imperfect detection of card boundaries can result in misaligned patches and reduced recognition accuracy.
- **Chip Interference**: Poker chips with white symbols or borders can visually resemble cards and cause false positives, especially when partially overlapping the cards.
- **Synthetic Dataset Noise**: The dataset for value recognition was generated using fonts and synthetic augmentation. While effective overall, reproducing the real noise of low-resolution video, such as blur and compression artifacts remains imperfect and affects model accuracy.

7 Conclusions

Despite the challenges encountered, such as visual interference from table markings and chips, limitations in synthetic data generation, and the inherent noise of low-resolution footage, the developed system demonstrates solid performance, particularly in its ability to reliably identify playing cards with high precision. The modular pipeline and targeted

preprocessing strategies have proven effective in real-world conditions. While there is room for improvement, especially in recall and patch extraction accuracy, the current implementation provides a robust foundation for further refinement and extension.

8 Future Implementations

The system can be improved by additional preprocessing of the frames of the videos and better training of the deep learning model. In addition adding the counter would be the final goal but we need to consider the timing of the game during the cards dealing.

9 Members contributions

Zoren Martinez, **around 80h**. Card symbol recognition, deep learning model development.
Davide Baggio, **around 80h**. Performance evaluation, overall code structure.
Francesco Pivotto, **around 80h**. Card bounding box detection, preprocessing.

References

[1] Wesley Cooper. Blackjack tracking system. B.a. (mod.) computer science final year project, Trinity College Dublin, Dublin, Ireland, April 2004. Supervisor: Dr. Kenneth Dawson-Howe. URL: https://www.scss.tcd.ie/Kenneth.Dawson-Howe/Projects/Previous/FYP2004_WesleyCooper.pdf.

[2] CVAT Team. Cvat: Computer vision annotation tool. https://www.cvat.ai/, 2025. Accessed: 2025-08-02.

[3] Font.download. Card characters font. https://font.download/font/card-characters, 2025. Accessed: 2025-08-02.

[4] Fontesk. Khepri typeface. https://fontesk.com/khepri-typeface/, 2025. Accessed: 2025-08-02.

[5] Niels Lohmann. Json for modern c++. https://github.com/nlohmann/json, 2025. Version 3.11.3, accessed: 2025-08-02.

[6] Perfect Pair. Blackjack | $ 900,000 buy in | doubling up huge $200,000 bets all se... https://youtu.be/ifFlurZNYgs?si=lkVX6_q3GeWUTfpC, 2025. YouTube video, accessed: 2025-08-02.

[7] PyTorch Team. Libtorch: Pytorch c++ api. https://pytorch.org/get-started/locally/#libtorch, 2025. Accessed: 2025-08-02.