Bridge Cheating Report

Haopeng Li, Zinuo Li

August 2023

1 Introduction

This project aims to provide technical support for bridge card games by detecting the suit, rank, position, and orientation of playing cards. The project is based on the YOLOv7 framework and collects data from various scenarios. We have also developed methods to synthesize data to improve the model's accuracy. In the end, we achieved an impressive mAP@0.5 score of 0.959, indicating that our model exhibits excellent accuracy.

2 Card Detection

2.1 Data Annotation

In this work, we have two types of dataset that are used in training process: Manually Annotated Dataset and Automatically Generated Dataset.

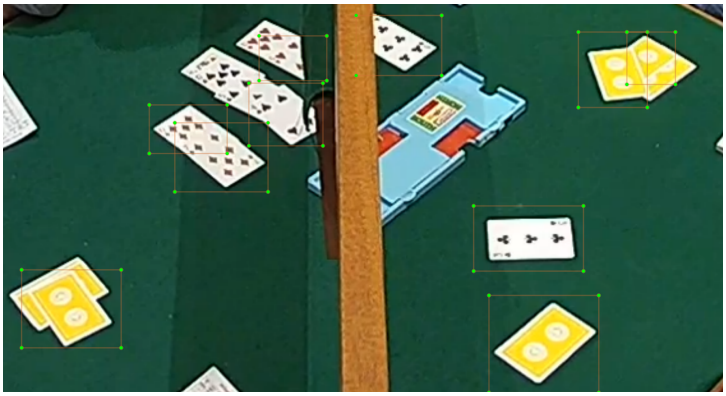


Figure 1: Manually annotated example.

As shown in Figure 1, for the manually annotated dataset, we collected approximately 20 videos from YouTube that contain diverse scenes. We used LabelImg to annotate the data, where the annotation standard was to only

annotate poker cards that are clearly visible and blocked less than 30%. For poker cards whose upper right corner numbers are unclear, we can determine the number of those poker cards by counting the number of patterns in the middle of the poker cards. We annotated a total of 54 classes, including 52 poker suit and number classes, 1 back class and 1 unknown class. The back class indicates that the poker card is facing down. The unknown class represents poker cards that are visible but their identities cannot be determined by human eyes, or some heavily occluded poker cards.

2.2 Synthetic Data Generation

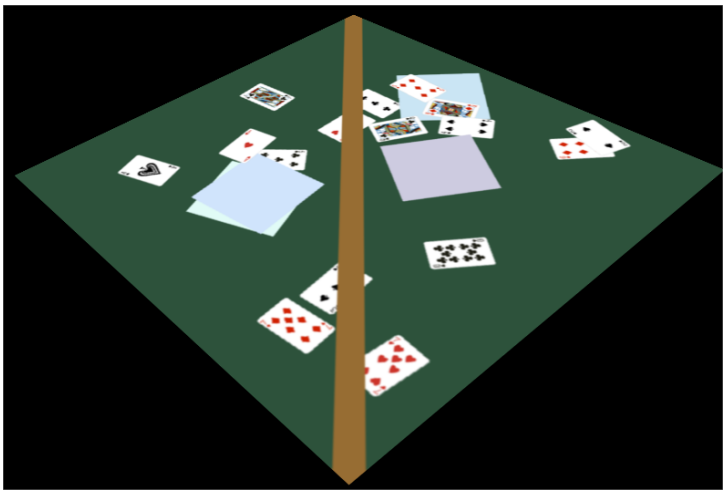


Figure 2: Generated data example.

As shown in Figure 2, the generated image is a simple scenario with random poker cards and scoreboards. Gaussian blur is applied to every card for simulating the real scenario. The whole steps to generate the data can be simply summarized as follows, note that this is just a brief summaryfor detailed parameters please refer to the code:

1. First, create an empty image I with dimensions of 800×800 pixels, where the pixel values are set to 0, and convert I to an Image object.

2. Invoke the patch_img() function to process I, adding random number of poker cards and scoreboards to I, and returning the processed image I along with card coordinates and the labels.

3. Utilize the cv2.fillPoly() function to draw a polygon on I representing a gameboard, with the color (56, 102, 154).

4. Add Gaussian blur to I.

5. Retrieve the shape of I and define a variable 'pts1' to represent the coordinates of the four corners of the original image. Define a variable 'pts2_mean' to represent the average coordinates of the four corners in the final perspective view. Calculate the transformation matrix between 'pts1' and 'pts2_mean', and apply the it to I.

6. Calculate the final coordinates 'label_final' based on the matrix obtained from the previous step. Finally, return the transformed image I and the transformed coordinates.

2.3 Card Detection by Deep Neural Network

In this step, we utilize the YOLOv7 framework for object detection. During the training phase, we employ approximately 4000 generated images and 6000 manually annotated images, with an epoch value set to 300 to obtain the final results. We conduct several different experiments using the same test set, and the results indicate that the mixup training approach yields the best performance, as shown in Table 1:

1. **Directly training:** Directly utilize the mannual annotated dataset for training without any pre-precessing steps.

2. **Pretrained training:** Use the generated dataset as pre-trained weight and then fintune the weight based on the mannual annotated dataset.

3. **Mix training:** Mix the generated dataset and mannual annotated dataset in a ratio of approximately 4:6.

The higher the values of each metric, the better. Typically, we tend to prioritize the mAP@0.5 metric to evaluate the model's performance.

Method	Precession	Recall	mAP@0.5	mAP@0.95
Direct	**0.939**	0.886	0.947	0.782
Pretrained	0.915	0.855	0.925	0.758
Mixup	0.925	**0.921**	**0.959**	**0.797**

Table 1: Quantitative analysis of different training methods on the same test set.

3 Card Orientation Detection

3.1 Table Detection

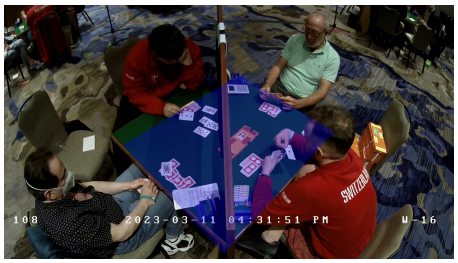
3.1.1 Pseudo Table Mask Generation

We use traditional computer vision techniques to generate the table mask.

(a) Good Example

(b) Good Example

(c) Good Example



(d) Bad Example

Figure 3: Pseudo Table Mask

1. Convert the image from RGB color space to HSV color space.
2. Find the regions that have the similar color to the table.
3. Perform dilation, area closing, and erosion to the mask.
4. Find the largest region in the mask, which is considered as the table mask.
5. Use Canny edge detector and Hough line transformation to acquire the edges of the table.
6. Filter out the noisy lines according to the direction (degree) of the table edge.
7. Combine the lines that indicating the same table edge. Four lines remain in total, representing four table edges.
8. Calculate all intersection points among the above four lines. Only preserve the points indicating the table corners.

Figure 3a–3c show good results of the generated masks, while Figure 3d shows the bad result.

3.1.2 Table Edge Detection Based on Deep Neural Network

The tradition method above is not robust enough to obtain satisfactory table masks/corners. However, we can use the generated masks as pseudo labels for

the task of table segmentation. Specifically, we can obtain plenty of table masks given images using the method above, and construct a large-scale training set for table segmentation. Then, we train YOLO-v7 (segmentation version) using the image-mask pairs. Finally, we obtain a robust table segmentation model that takes as input an image and outputs the table mask.

The model above outputs a binary mask for an image (as illustrated in Figure 4), and we use Step 5–8 in Section 3.1.1 to obtain the final table edges.

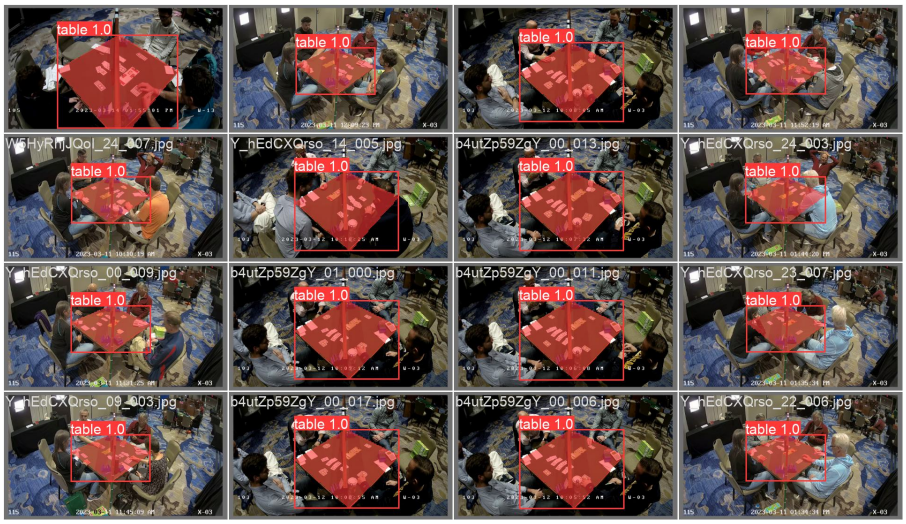


Figure 4: Table Masks Predicted by Deep Neural Network

3.2 Temporal Smoothing

Small disturbance of the bounding box would lead to the changes of the computed angle for a static card. Therefore, we perform temporal smoothing to the bounding boxes across different frames, and make it fixed for a static card. The temporal smoothing is achieved by Kalman filtering that tracks the bounding box of the same card and makes the changing smooth. Finally, we use the temporally-smoothed bounding boxes for the following process.

3.3 Card Edge Detection

The card edge detection relies on the detected bounding boxes of cards from Section 2. Given an image and the bounding boxes of cards, we detect all the card edges by the following steps.

1. Use Canny edge detector to obtain the edge binary mask of the image.
2. Crop the edge mask using the bounding boxes of cards.

3. For each cropped mask, perform Hough line transformation to obtain all the lines. Choose the longest line as the edge of the card.

3.4 Orientation Computation

We compute the angle between the card edge and its closest table edge, as illustrated in Figure 5. However, the computed angle is not reliable because there is always a strong perspective effect. Therefore, we first need to transform the table surface into a square, and apply the same transformation to the table edges and card edges. Finally, we obtain the true angle between the card edge and its closest table edge.

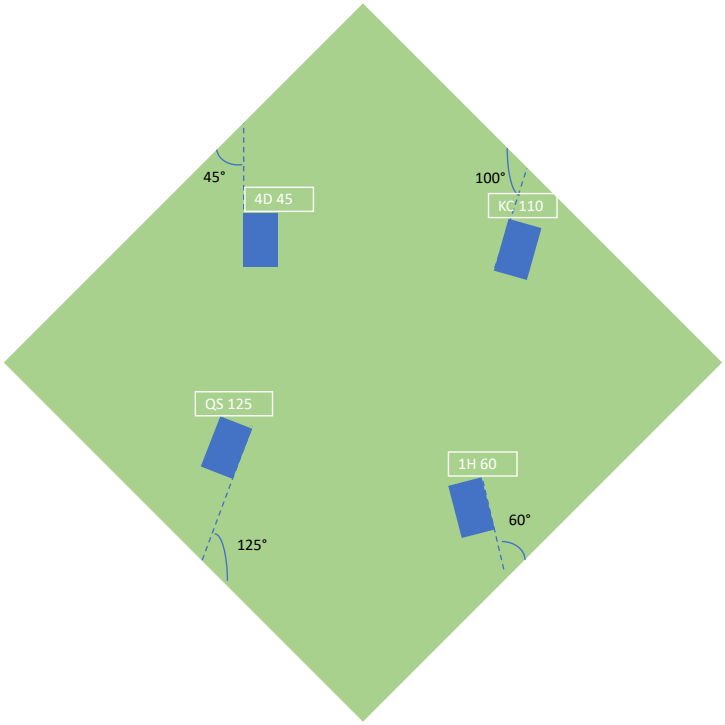


Figure 5: Orientation Definition

4 Usage

This project utilizes two different models to obtain the final results. The usage is as follows:

1. In **card_detect** project, specify **save_dir** in **video_split.py**, run it to gain the screenshots of video in a directory $\langle D \rangle$.

2. In **card_detect** project, run **python detect.py --weights card_detection.pt --source** $\langle D \rangle$ **--name** $\langle N \rangle$ **--save-txt --nosave --save-conf**

3. Find corresponding txt results in **card_detect/runs/detect/**$\langle N \rangle$**/labels**.

4. Copy $\langle D \rangle$ **and** $\langle N \rangle$**/labels** above to **orientation_detect project**, and rename **/labels** to $\langle D \rangle\_res$. For example, $\langle D \rangle$ is **test_video** so the $\langle D \rangle\_res$ will be **test_video_res**.

5. In **orientation_detect** project, run **python -u segment/predict.py --weights last.pt --source** $\langle D \rangle$ **--name** $\langle \hat{N} \rangle$.

6. In **orientation_detect** project, find image results in **orientation_detect/runs/predict-seg/**$\langle \hat{N} \rangle$**/vis**.

7. In **orientation_detect** project, specify **image_folder** in **merge_images.py** and run it to obtain final video result.