TEXAS HOLD'EM HANDS DETECTION USING SYNTHETIC DATA

Martino Mattia **Pranzo Alessandro**

Academic Year 2024-2025

ABSTRACT

Texas Hold'em is a globally popular poker variant characterized by strategic decision-making under uncertainty, involving elements of probability, game theory, and psychology. Despite the widespread application of deep learning techniques in computer vision, research on poker card recognition remains limited, primarily relying on classical computer vision methods. These approaches often struggle with generalization and lack publicly available datasets tailored for this task.

In this study, we propose a deep learning framework to detect poker hands from a first-person perspective, focusing on classifying hole cards and community cards. Due to the absence of suitable datasets, we generated a synthetic dataset using geometrical transformations, scanned card decks, and the MS COCO dataset for realistic backgrounds. This dataset simulates various gameplay scenarios and includes automatically generated bounding boxes for card identification.

We fine-tuned the YOLOv11 model, a state-of-the-art object detection framework, on this dataset to perform robust card detection and classification. The model demonstrates strong performance within the synthetic dataset, successfully identifying both individual cards and their roles in gameplay. However, its generalization to real-world images is limited, suggesting the need for mixed datasets combining synthetic and manually annotated real-life images.

Our findings highlight the potential of synthetic data in training deep learning models for poker card recognition while emphasizing areas for future improvement, including enhanced dataset diversity and advanced warping techniques to better simulate real-world conditions.

1 Introduction

Poker is arguably the most famous card game in the world. It's origins date back to the 19th century in the United States but its most famous variant, Texas Hold'em, gained popularity only in the 1990s. This version is played with french cards with a number of players that varies from 2 to 10. Each individual is dealt two cards, the "hole cards", and shares five community cards revealed in stages. The stages are the "flop" in which three cards are revealed, followed by the "turn" and the "river", both of which reveal one card. The goal is to construct the best poker hand using any combination of hole and community cards. At each stage the players engage in rounds of betting to form the "pot"-the amount of money gained by the winner-where they can choose to exit the game (fold), match the bet of another player (call) or raise the bet (raise). A player wins if all other players folded or if he has the best hand after the last stage, where the hands of all playing individuals are compared. The decision making under incomplete information setting makes Texas Hold'em a skill based strategic card game with an heavy influence of probability, game theory, and psychology.

Despite the popularity of deep learning approaches applied to a broad range of applications in the computer vision field in recent years, very few publications focus on the topic of cards recognition for the game of poker. Most of such attempts have been made using classic (non-learning) computer vision techniques, resulting in a lack of reliable and labeled data for potential training of deep learning models.

In this work we examine various approaches and later create a deep-learning-based framework to detect poker hands from a first person view perspective. The goal of the model will not just be the one of detecting single cards, but also detecting which cards belong to the hand of the player and those belonging to the table. Due to the lack of public datasets giving an appropriate point of view we generate a synthetic dataset using various geometrical techniques. These, together with automatically generated bounding boxes for the cards are then used to fine-tune the latest version of the You Only Look Once (YOLO)[13] object detection model.

2 Data Preparation

The methods used throughout our analysis rely heavily on the OpenCV library [11] used with Python 3.12.

2.1 Card Segmentation

Our first attempt was to use purely geometrical approach for card identification. The pipeline we tried to implement was: transform the images from RGB to gray-scale, apply a Gaussian blur, perform edge detection, and then find the polygon containing the cards. From a mathematical perspective, the transformation from RGB to gray is done by reducing the three color channel to one intensity channel this is achieved through a weighed sum:

$$I = w_R R + w_G G + w_B B \tag{1}$$

where R, G, B are the intensities of the red, green, and blue, and the weights are taken to be $w_R = 0.299, w_G = 0.587, w_B = 0.114$. Gaussian blurring is a technique used to reduce noise and detail by applying a Gaussian function as a filter. This is done by convolving the image with a Gaussian kernel of size $n \times n$, and the value of σ which determines how much neighboring pixels influence the central pixel during the convolution. So, given x, y the coordinates of a pixel relative to the center of the filter, and the Gaussian function $G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$ the blurred intensity $I'(x, y)$ is given by

$$I'(x, y) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} G(u, v) I(x + u, y + v) \tag{2}$$

Here $k = (n - 1)/2$. Edge detection is performed using Canny Edge Detection[3]. To find the polygon containing the cards, we first find the contour of the card using the *findContours*[14] function from OpenCV and then approximate the polygon containing the card.

After careful fine-tuning of the parameters for thresholding, unfortunately, this approach still lacks of generalizability, thus resulting in weak performance for our task when applied on images that fall out of the initial context. For this purpose, we decide to rely on a deep learning approach to overcome generalization problems (please refer to Section 6 for more details).

2.2 Dataset generation

As mentioned before, the lack of research on the topic slows down any attempt to using machine-learning-based approaches. However, inspired by [7], in this work we embarked on the journey of devising an automated method to generate synthetic images to train our model on. To do so, we took a set of images from the MS COCO dataset to be used as backgrounds, and two scanned decks of cards ([1], [2], check Figure 1 for a sampled scanned image) to ensure that the model doesn't learn to detect only cards on a certain background or from a specific deck. From these baselines, we than perform a series of geometric transformations in order to simulate different POVs (Point Of View) of someone playing poker in first person at the table.

To begin, we preprocessed all the images in the deck to identify the bounding boxes containing the corners of each card. For each card, we annotated two specific bounding boxes—one around the rank and suit (covering the upper-left and lower-right corners)—and another encompassing the entire card, as illustrated in Figure 5.

Next, we designed a pipeline to generate stochastic synthetic images, simulating realistic scenarios of card distributions. In each iteration of the dataset creation process,

the pipeline randomly selects one of the two decks, shuffles it, and draws 2, 5, 6, or 7 cards. This simulates different stages of poker play: "pre-flop" (where only your hand is visible), "flop" (your hand and 3 community cards are visible), "turn" (your hand and 4 community cards), and "river" (your hand and 5 community cards). The selection process follows a probability distribution that reflects the frequency of these scenarios in online poker tournaments, ensuring realistic representation.

The pipeline then progresses through the following stages:

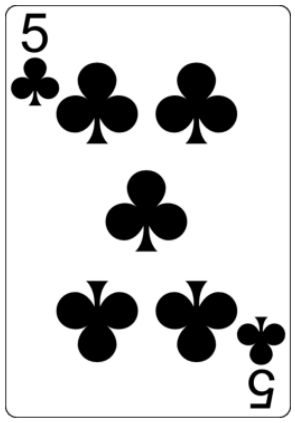

Figure 1: Scan image of 5 ♣ from deck.

Rotation: at this stage each card is rotated around its center with a random angle drawn as a normal variable centered at 0 and a standard deviation of 20 (degrees) or centered at 0 and standard deviation of 5 for the cards belonging to the hand and flop respectively. For each card, its pixels are accordingly transformed through the rotation

$$M = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & (1-\cos(\theta)) \cdot \frac{W}{2} + \sin(\theta) \cdot \frac{H}{2} \\ \sin(\theta) & \cos(\theta) & (1-\cos(\theta)) \cdot \frac{H}{2} - \sin(\theta) \cdot \frac{W}{2} \end{bmatrix}$$

where θ is the angle of rotation, W is the width, and H is the height of the image.

The matrix is adjusted to account for the new image dimensions W_{new} and H_{new}:

$$M[0,2] += \frac{W_{\text{new}} - W}{2}, \quad M[1,2] += \frac{H_{\text{new}} - H}{2}$$

Now, each point in the image is represented in homogeneous coordinates as $[x, y, 1]$. The points are then transformed using the rotation matrix M (result in Fig. 2):

$$\text{rotated\_points} = [x \quad y \quad 1] \cdot M^T$$

Translation/Merging: At this stage of the pipeline, the cards assigned to the hand and those assigned to the flop are merged into two separate canvases (see Fig. 2 for reference). This step is based on a key assumption: all cards within the same group—either "hand" or "flop"—lie on the same plane. This assumption allows us to apply the same transformation to all cards in a given group later in the pipeline, enabling us to realistically simulate real-world conditions. For example, cards on a table naturally lie flat on the same surface, while cards in a hand are held together and viewed from a single perspective.

Shading and Blurring: Each canvas coming from the previous stage is now shaded and blurred independently (result in Fig. 2). Each image is modulated by a normally distributed coefficient $s \sim \mathcal{N}(\mu = 1, \sigma = 0.3)$ clipped to $[0.3, 2.0]$ (the shading part), followed by Gaussian smoothing with a 5×5 kernel (the blurring part). This operation, defined as $I_{augmented} = \text{GaussianBlur}(\text{clip}(s \cdot I_{original}, 0, 255))$, introduces natural lighting variations while maintaining image integrity through value normalization and edge preservation.

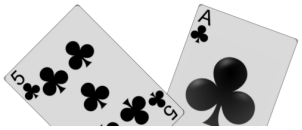

Figure 2: 5♣ and A♣ after rotation, merging and shading is applied.

3D Rotation and Warping: Each group of cards undergoes a 3D rotation around its center using specified pitch and yaw angles, ensuring that the center of the image remains stable and the entire rotated image is visible (Fig. 3). The rotation is defined by combining the pitch rotation matrix R_{pitch} and yaw rotation matrix R_{yaw}:

$$R_{\text{pitch}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_p) & -\sin(\theta_p) \\ 0 & \sin(\theta_p) & \cos(\theta_p) \end{bmatrix}$$

$$R_{\text{yaw}} = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix}$$

$$R = R_{\text{yaw}} R_{\text{pitch}}$$

After applying the rotation matrix R, the corners of the image, represented in 3D as (X, Y, Z), are projected back to 2D using the pinhole camera model:

$$x_{\text{proj}} = \frac{f \cdot X}{Z}, \quad y_{\text{proj}} = \frac{f \cdot Y}{Z}$$

where f is the focal length. To transform the entire image, a homography matrix H is calculated to map the original 2D corners to their projected 2D positions. The projected image center may shift due to the rotation, so an additional translation matrix is applied to stabilize the center:

$$T_{\text{center}} = \begin{bmatrix} 1 & 0 & c_x - c'_x \\ 0 & 1 & c_y - c'_y \\ 0 & 0 & 1 \end{bmatrix}$$

where c_x, c_y are the original center coordinates, and c'_x, c'_y are the projected center coordinates. Finally, the canvas size is adjusted to enclose the rotated image fully, ensuring no cropping, and the final transformation is represented by an adjusted homography matrix:

$$H_{\text{final}} = T_{\text{box}} \cdot T_{\text{center}} \cdot H$$

where T_{box} translates the bounding box to start at $(0, 0)$. This final matrix H_{final} is applied to warp the image and its bounding boxes, ensuring proper alignment and visualization on a resized canvas that accommodates the rotated image.

Figure 3: 5♣ and A♣ after being warped

Applying on Background In the final step, the two canvases are placed onto a random background sampled from the previously mentioned dataset (refer to Fig. 4). The hand is positioned in the lower part of the image, scaled larger than the flop, which is placed in the upper part of the image. To better simulate a first-person point of view, we apply a negative pitch to the hand and a positive pitch to the flop. Additionally, if either the hand or flop is positioned on the left or right side of the image, we apply a strong or slight negative yaw (for the left) or positive yaw (for the right), respectively.

Figure 4: Final result on background (both hand and flop)

Eventually, we repeat the process above We repeated the process described above to generate 15,000 images for training, 2,500 for testing, and 2,500 for validation, ultimately creating the full dataset. Regarding the labels, we defined bounding boxes within each image, associating each box with a label corresponding to its content (e.g., the rank and suit of a card or whether it belongs to the hand or flop).

For the bounding boxes assigned to individual cards, we adopted two distinct labeling approaches (illustrated in Figure 5). In the first approach, the label of a single card is associated with two bounding boxes, one for each of the two opposing corners (as previously described). In the second approach, the label is associated with a single bounding box encompassing the entire card. These two labeling strategies were subsequently used to train two different versions of the same model, and their performances were compared.

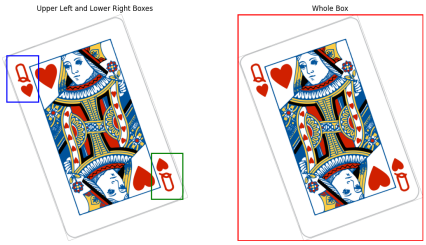

Figure 5: On the left: labeling system 1. On the right: labeling system 2.

3 Model

To perform the card classification task we used the latest YOLO version *v11* from Ultralytics. The reason why we decided to use this version is because it achieves a better performance than the previous benchmark model YOLOv8, with 22% less parameters[15], allowing us to fine tune it effectively on our machines. More in the specific we use the smallest model from this version, the nano version which has 2.6 million parameters and requires 640x640 images. This architecture is divided in three main parts: the **backbone**, the **neck**, and the **head**[12].

In order to understand how the backbone is structured we start from its most basic structure: a convolutional block. In this model a **Convolutional Block** is composed a 2D Convolutional Layer with 2D Batch Normalization[10] and SiLU activation function[6]. Each sequence of Convolutional Blocks is part of a **BottleNeck**, which behaves like a ResNet Block[9] if a *shortcut* parameter is set to true. BottleNeck blocks then compose a **C3K** block, that is a Convolutional Block followed by a sequence of BottleNeck blocks with concatenations at the end, with a final Convolutional Block. Lastly, the **C3K2** block is formed by a Convolutional block, a series of C3K blocks, and concatenation with a final Convolutional block. All this enables to handle feature extraction in a fast and computationally efficient manner.

On this line, the neck, is composed of the **SPFF** module[8] (Spatial Pyramid Pooling Fast) which uses several max-pooling operations to put together the contextual information. This enables the model to detect elements across different scales. This is coupled with the attention block **C2PSA** (Cross Stage Partial with Spatial Attention), which allows the model to focus on important regions of the image. In our case this is particularly useful since it improves the classification accuracy of partially occluded objects. Finally, the last component is a multi-scale prediction head, which uses the feature maps of the backbone and neck.

The model was trained in 70 epochs, with a batch size of 8, using a NVIDIA RTX A4000 GPU.

4 Results

As we mentioned in Section 2 the classical computer vision approach for geometrical segmentation of the images fails to generalize. In Figure 6 we can see that after tuning the parameters of the functions mentioned in 2.1 we managed to find the segmentation masks of the cards on the table. Despite this, segmenting both groups of cards resulted impossible for two reasons: the different dimensions of the cards in each group required different area conditions which retained irrelevant polygons, and the overlapping and the presence of the hand on the hole cards didn't make possible the formation of closed polygons to work with.

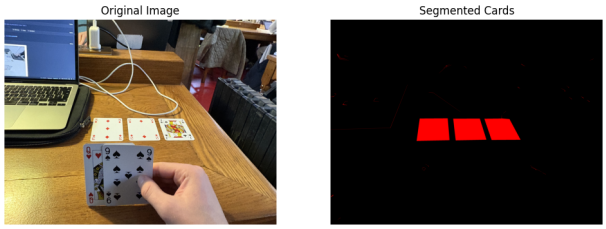

Figure 6: Demonstration of the geometrical segmentation model

On the other hand, the deep learning model trained on our synthetic dataset performs well on both classification and detection tasks, and does not give signs of overfitting.

box_loss	cls_loss	dfl_loss
0.496	0.377	0.818

Table 1: Metrics on the training dataset at the 70th epoch.

box_loss	cls_loss	dfl_loss
0.456	0.318	0.801

Table 2: Metrics on the validation dataset at the 70th epoch.

mAP50	mAP50-95
0.986	0.857

Table 3: Object detection metrics at the 70th epoch.

In Tables 1,2 we can see the three different YOLO losses at the end of training. The **Box Loss** is the difference between the predicted bounding box and the ground truth, so it ensures an accurate localization of the objects. The **Classification Loss** evaluates the model's classification capabilities. Finally the **Distribution Focal Loss** uses a regression model to predict a distribution of possible positions for the bounding box, ensuring more accurate predictions. These values are lower for the validation set, indicating that our model is generalizing well and it is not overfitting. We can notice that the classification loss is lower than the other two, which is common in these tasks.

To talk about the results in Table 3 we need to introduce the concept of Intersection over Union (IoU). This is simply the ratio of the overlap between the predicted bounding box and the ground truth to their union

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

We also define the IoU threshold that is the value over which a prediction is considered a true positive. The **mean Average Precision** is a common metric in object detection tasks computed as the mean of the area under the precision-recall curve across all classes. So, mAP50 means an mAP computed with a single IoU threshold of 0.5, and mAP50-95 an mAP computed with multiple IoU thresholds from 0.5 to 0.95 at 0.05 steps. Usually the former is enough approximate estimations of object positions, while the second is more strict. The results are excellent, indicating an almost perfect detection with a lower IoU threshold. For the stricter metric the value is lower, but it shows that the model maintains a strong performance under more demanding requirements. Figure 7 shows the trend of the metrics in Table 1, 2, whilst Figure 8 shows the behavior of the metrics in Table 3.

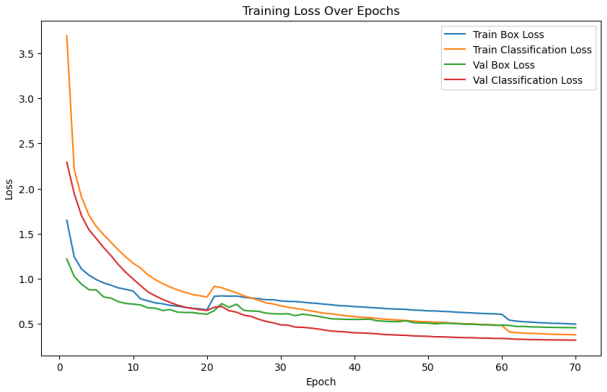

Figure 7: Behavior of the losses (both training and validation) over the 70 epochs

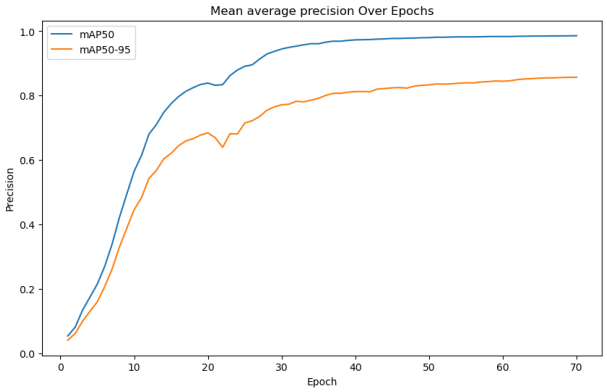

Figure 8: Behavior of the mean Average Precision metrics over the 70 epochs

We can see that in both cases the trends follow the expectations. For both images the curves are steeper in the first 20 epochs, and get close to plateauing at around 40 epochs.

5 Conclusion and Future Improvements

From our work we concluded that the "purely geometric" card matching approach works in some specific situations, but does not generalize very well. There are indeed various sources ([5], [4]) implement pipelines that work, but just from a fixed point of view-usually right above the cards-and on a fixed background. Indeed, for these systems to work, a lot of hyperparameters that depend on POV, focal distance, and external conditions need to be tuned, leading to a suboptimal method for our task. Moreover, if the all four sides of a card are not fully visible, a fairly common case with hole cards, we cannot build a closed polygon enclosing its contour, leading to a missed detection.

On the other hand, using the synthetic dataset that we have generated seems to work well inside its framework. What this means is that if training, validation, and test set are all synthetically generated, then, as seen in the previous section, the model performs well. Despite this, if we try to perform inference on a real-life image both the object detection and the classification performances drop drastically. This is because of the introduction of factors that the model has never seen, as for example decks of cards with different designs, complex shading or bending of the cards, and distance relations. Figure 9 clearly displays the generalization capabilities of our approach in an environment and on a deck never seen, clearly the bounding boxes around the corners of the cards (rather than on the whole picture) resulted to be more effective. The *hand* and *flop* labels and positions seem to be consistently classified correctly also in external pictures. For these reasons we think that one main area of improvement could be to use a mix of hand labeled and synthetic images for training so to enhance the performance. Finding a good trade-off between human effort and model improvement could lead to significant advancements.

Focusing more on possible improvements on the data generation, adding different points of view for the flop, more warping functions could lead to better generalization capabilities for real-life scenarios. For the former, one can combine rotation and pitching of the community cards so that is resembles an individual seated at different points of the table. For the latter one big improvement to better recognize private cards is implementing Bezier warping on cut cards, to simulate the bending of the cards.

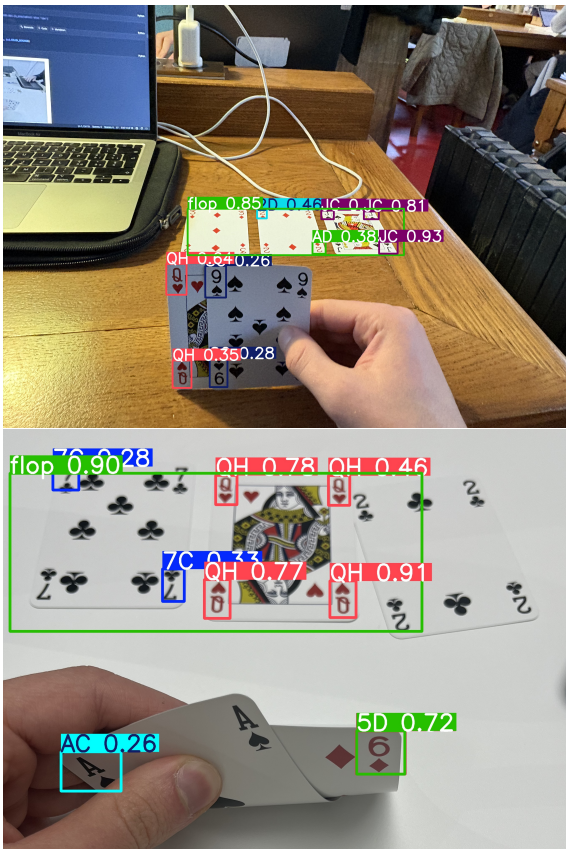

Figure 9: Examples of generalization.

References

[1] URL: https://huggingface.co/datasets/F1nn21/playing-cards/discussions/new?title=Request:%20DOI.

[2] URL: https://code.google.com/archive/p/vector-playing-cards/.

[3] John F. Canny. *Canny Edge Detection*. URL: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html.

[4] dofire. *OpenCV Playing Card Detection*. URL: https://github.com/dofire/OpenCV-Playing-Card-Detection.

[5] EdjeElectronics. *OpenCV Playing Card Detection*. URL: https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector.

[6] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning". In: *CoRR* abs/1702.03118 (2017). arXiv: 1702.03118. URL: http://arxiv.org/abs/1702.03118.

[7] geaxgx. *Playing Card Detection*. URL: https://github.com/geaxgx/playing-card-detection.

[8] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *CoRR* abs/1406.4729 (2014). arXiv: 1406.4729. URL: http://arxiv.org/abs/1406.4729.

[9] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[10] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[11] opencv. *OpenCV*. URL: https://github.com/opencv/opencv.

[12] S. Nikhileswara Rao. *YOLOv11 Architecture Explained: Next-Level Object Detection with Enhanced Speed and Accuracy*. URL: https://medium.com/@nikhil-rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71.

[13] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[14] Satoshi Suzuki and Keiichi Abe. "Topological structural analysis of digitized binary images by border following". In: *Comput. Vis. Graph. Image Process.* 30 (1985), pp. 32–46. URL: https://api.semanticscholar.org/CorpusID:205113350.

[15] Ultralytics. *Ultralytics YOLO11*. URL: https://docs.ultralytics.com/models/yolo11/#overview.