CVIA - Assignment Report
Monza e Romagna

Filippo Focaccia, Giulio Pirotta, Tommaso Vezzoli

3160552, 3160155, 3144207

1 Introduction to the Problem

Automatic card detection is pivotal in computer vision applications like virtual blackjack or augmented reality gaming, but traditional object detectors struggle in scenarios where cards are partially occluded or overlapping (common situations in card games).

In this project, we robustly detect playing cards in realistic blackjack-like scenarios using deep learning with YOLO object detection architectures.

2 Problem Formulation

The goal is to detect all playing cards visible in an image, even when cards are stacked or overlapping, as long as one corner is visible. This poses challenges in object detection, including occlusion handling or generalization from synthetic to real-world data, and a robust solution could enable digital blackjack assistants or card game digitalization in general.

Notably, in another project for our deep-learning course, we developed a blackjack-playing agent using reinforcement learning; this computer vision pipeline could provide the perceptual input to such an agent, enabling a fully end-to-end system that plays blackjack directly from visual input.

3 Background and related work

Object detection has evolved significantly with the development of one-stage detectors, among which the YOLO (You Only Look Once) family is one of the most influential.

The YOLO architecture reframes object detection as a single regression problem, directly predicting bounding boxes and class probabilities from an input image in one forward pass, trading off some accuracy for speed and simplicity.

3.1 YOLOv1

YOLOv1, introduced in 2016 by Redmon et al., divides the input image into an $S \times S$ grid, where each cell is responsible for detecting objects whose center falls within it. Each cell predicts a fixed number of bounding boxes (typically two), along with a confidence score and class probabilities. These predictions are made through a fully convolutional network consisting of 24 convolutional layers followed by 2 fully connected layers.

In our project, implementing YOLOv1 from scratch is an opportunity to learn and delve into some key topics treated in the course. However, we observe that training it from scratch on a synthetic dataset is particularly challenging, and that the architecture is not well-suited to our occlusion-heavy scenarios.

3.2 YOLOv5

YOLOv5 retains the core principle of one-shot detection while introducing a modular and extensible architecture, being popoular for a combination of strong empirical performance and efficiency.

Its architecture includes several key improvements over the first version:

- Backbone: YOLOv5 uses a CSPDarknet53 backbone, which incorporates Cross Stage Partial connections to to reduce redundant information and complexity.
- Neck and Head: It adopts a PANet-based neck and anchor-based detection head to better fuse multi-scale features, which is especially important for detecting small or partially visible objects like our playing cards.

3.3 YOLOv8

YOLOv8 adopts further improvements, including an anchor-free detection head and enabling more flexible object localization.

Additionally, it supports dynamic input shapes and improved post-processing, further enhancing accuracy and adaptability across varied detection tasks.

4 Methodology

We initially try to implement YOLOv1 from scratch to understand the full pipeline and its applicability to our task. However, the model fails to learn effectively on our data, and we hypothesize that jointly learning low-level features and bounding box regression is too complex to be done from scratch, especially for overlapping objects.

To mitigate this, we split the training in:

1. Pre-training: train the backbone as a card classifier using a dedicated dataset.

2. Fine-tuning and transfer learning: focus on the detection task.

We also experiment with replacing the YOLO backbone with ResNet-50 and initializing its weights on ImageNet. Despite reaching nearly perfect accuracy in classification, the end-to-end detection performance remains poor, possibly due to issues in the loss function or final bounding box regression layers. Eventually, we transition to YOLOv5, due to its solid performance and extensive documentation.

Leveraging the YOLOv5s model and its predefined loss function from the Ultralytics library, we are able to build a fully custom pipeline around the core model to maintain control over the training dynamics and dataset handling. Specifically, we implement the Pytorch Dataset and Dataloader classes, develop a custom training loop with learning rate scheduling and model checkpointing, write the evaluation and inference logic from scratch.

This modular approach give us a deeper understanding of the detection pipeline and allow us to adapt each component to the requirements of our task.

4.1 Benchmarking on YOLOv8

As a benchmark, we test YOLOv8 using the default Ultralytics implementation without any manual tuning: we apply transfer learning by freezing the backbone and training only the final classification layers on our dataset.

Performances are be similar to the ones for YOLOv5, supporting the effectiveness of our chosen baseline.

5 The Dataset

To fine-tune a YOLO-based object detection model on playing cards in a blackjack setting, we create a synthetic dataset using clean, pre-cropped images of individual cards, whose most informative visual areas (top-left and bottom-right corners) displaying the rank and suit are isolated from each card using manually defined regions.

Contours are extracted from these corner patches, and their union is used to compute a convex hull, which serves as a robust geometric envelope to account for small variations or rotations. A tight, axis-aligned bounding box is then derived from the convex hull and used as the annotation target.

The annotated cards are composited onto random background images at varied positions and scales to simulate realistic blackjack scenarios. Corresponding bounding boxes are adjusted and normalized relative to the full image dimensions and saved in YOLO format: class ID followed by center coordinates (x, y) and dimensions (w, h), all in relative terms.

For the classification dataset instead, we simply superimpose and augment the cards, placing them over random backgrounds from the DTD dataset.

5.1 Labels

To train our models, we need to adjust the original annotations to express the bounding box position relative to the boundaries of the corresponding grid cell (i.e., converting global image coordinates into local cell-level coordinates).

For this reason, we first identify the cell containing the box center and rescale its coordinates with respect to it, whereas the width and height measures are kept relative to the whole image dimensions.

6 Experiments and Results

6.1 YOLOv1

Our YOLOv1 model trained from scratch does not converge. Despite reaching nearly 100% accuracy on classification, detection accuracy does not improve,

confirming that localization and detection are difficult to be learnt jointly from scratch without extensive data or tuning. Literature and forum support for training YOLOv1 is lacking, making debugging difficult.

Integrating the ResNet-50 backbone initially yields substantial reduction in training loss, suggesting improved learning capacity. However, this result is misleading, as the model quickly overfits the training data and fails to generalize. Despite experimentation with dropout and learning rates, we are unable to achieve a stable balance between training and validation loss.

The struggles observed during training might also be associated to the presence of multiple corners in the same cell, causing problem during the computation of the loss, since we only retain one box per cell (the one with the highest IoU).

This is due to the fact that our dataset reflects real scenarios, where the dealt cards are close to each other.

6.2 YOLOv5

Switching to YOLOv5 significantly improves results. Fine-tuning a pre-trained small version of the model on our dataset leads to quick convergence and effective performance without extensive hyperparameters tuning.

An interesting factor determining this result is that the YOLOv5 architecture is much lighter than that of YOLOv1. By exploiting convolutions better, it requires a fraction of the parameters (from 50+M to around 10M), thus making both the forward and backward pass faster.

6.3 Evaluation

We use mean Average Precision (mAP) to evaluate performance. This metric calculates the area under the precision-recall curve for each class and then averages over classes: it is a standard for object detection and reflects both detection accuracy and localization precision.

Additionally, we build a simple live inference script using a webcam, which shows qualitative performance in controlled conditions, namely with a close-to-frontal perspective on cards. However, we observe some failure cases, including:

- Missed detections or misclassifications when the scale of cards differs significantly from the training data.
- Card count mismatch when many cards are present simultaneously, thus exceeding the distribution to which the model is exposed during training.

These are natural consequences of dataset limitations and perspective variations in live video and must be accounted for when using the tool.

7 Conclusions

This project was a valuable opportunity to apply theoretical knowledge in a practical, end-to-end computer vision task. We were able to concretely engage with many of the core concepts studied—from convolutional architectures and object detection algorithms to dataset preparation, training strategies, and performance evaluation.

Through hands-on experimentation, we learned the challenges of training detectors like YOLOv1 from scratch, highlighting the importance of transfer learning and fine-tuning of pre-trained models, which we effectively leveraged with YOLOv5 and YOLOv8.

The live inference tests and real-world deployment, while not flawless, demonstrated the practical applicability of our pipeline and helped us recognize its limitations and potential.

Overall, this assignment was not only a successful technical endeavor but also a formative educational experience. It solidified our understanding of computer vision and gave us confidence in applying these skills to real-world challenges and future projects, including the integration of perception with decision-making agents.

References

[1] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788. https://arxiv.org/abs/1506.02640

[2] Khanam, R., & Hussain, M. (2024). *What is YOLOv5: A Deep Look into the Internal Features of the Popular Object Detector*. https://arxiv.org/abs/2407.20892

[3] Jocher, G., Stoken, A., Borovec, J., et al. (2020). *YOLOv5 by Ultralytics*. GitHub Repository. https://github.com/ultralytics/yolov5

[4] Jocher, G., Chaurasia, A., Qiu, J., et al. (2023). *YOLOv8: Next-Generation Real-Time Object Detection*. Ultralytics. https://docs.ultralytics.com

[5] *Playing Card Detection*. GitHub Repository. https://github.com/geaxgx/playing-card-detection

Appendix

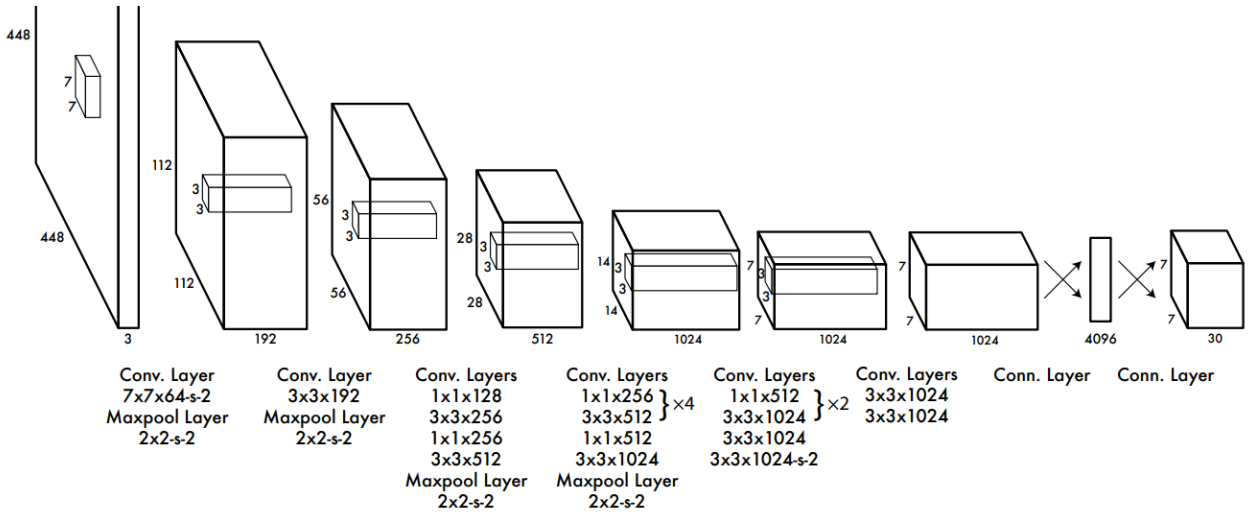


Figure 1: YOLOv1 architecture.

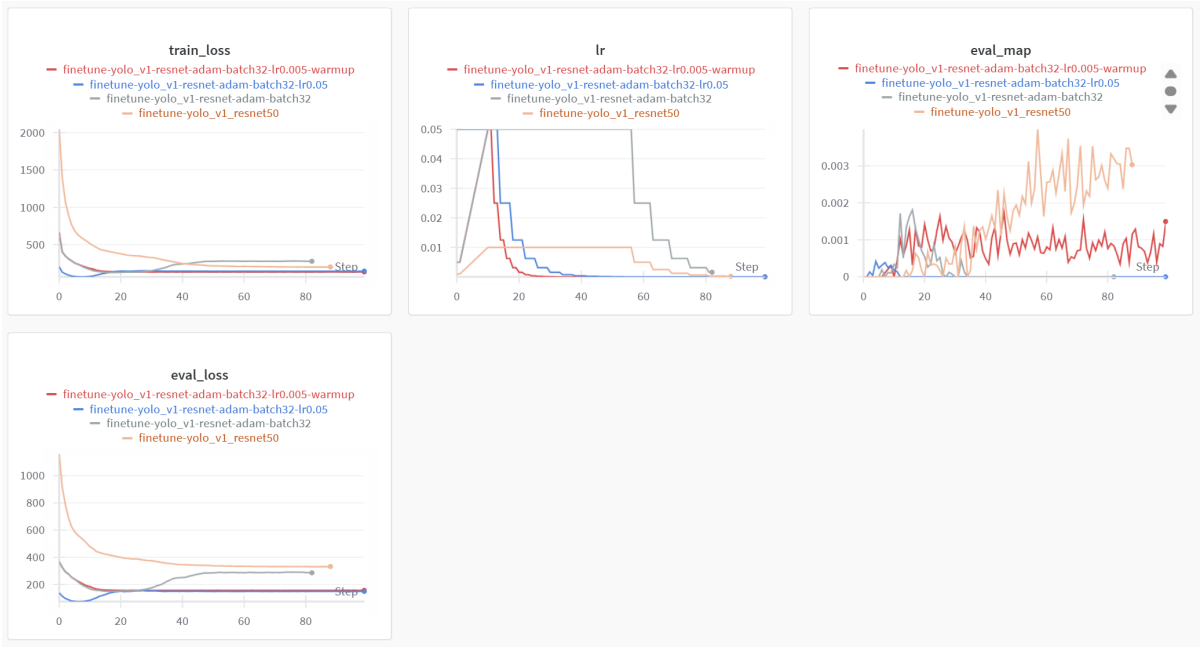


Figure 2: Sample runs for YOLOv1 training. After learning for the first epochs, the model struggles, the loss stops decreasing, and remains extremely high.

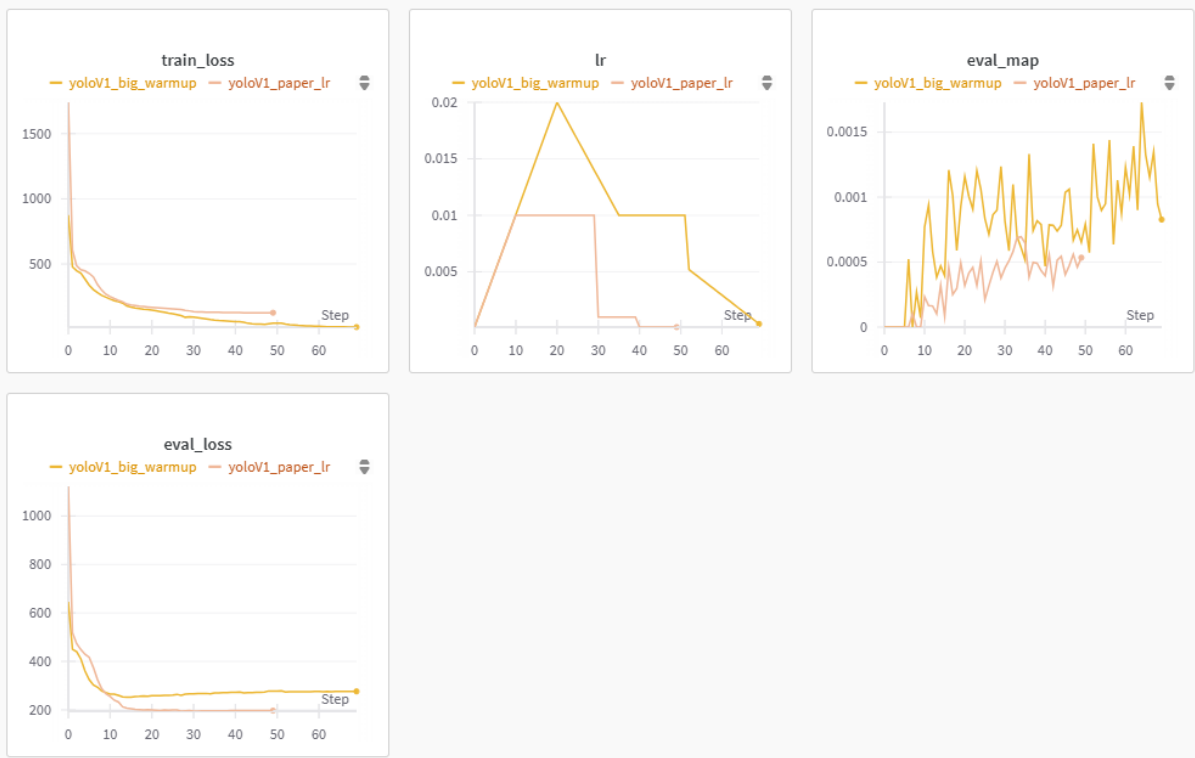


Figure 3: Sample runs for training YOLOv1 with ResNet50 backbone. The orange run follows the learning rate scheduler described in the YOLOv1 paper. The yellow run uses a learning rate scheduler with a large linear warmup in the first epochs.

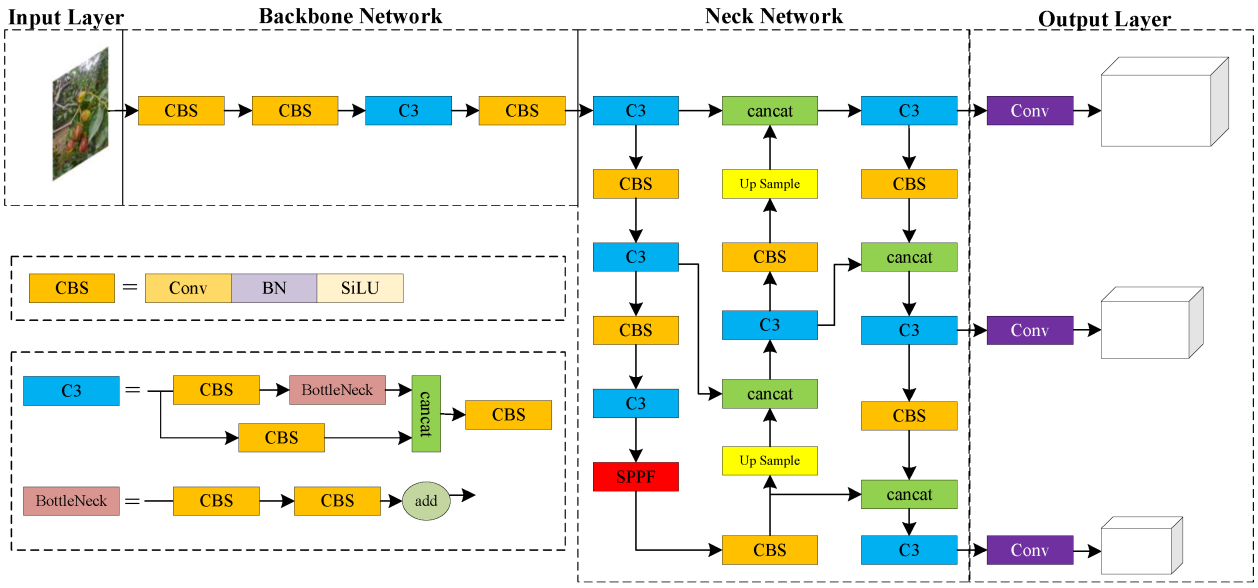
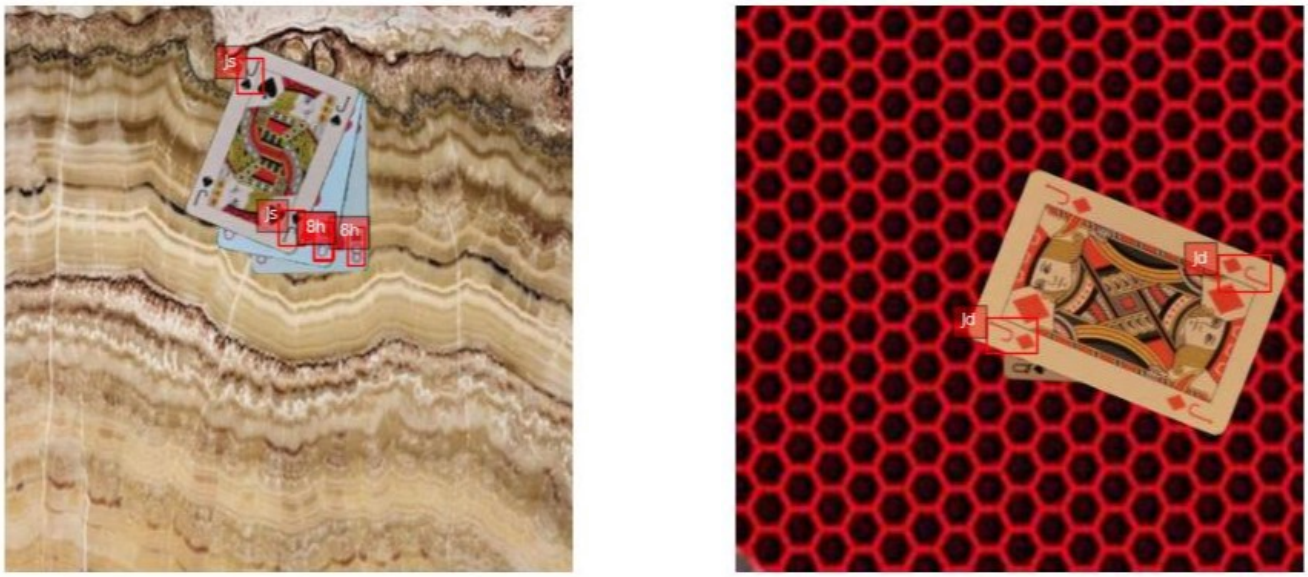


Figure 4: YOLOv5 architecture.

Figure 5: Sample runs for YOLOv5 training. The model effectively learns to detect cards.

Figure 6: Examples of well-classified images. The model effectly learns to detect and classify cards with different sizes, positions, and orientations.



(a) Example of misclassification

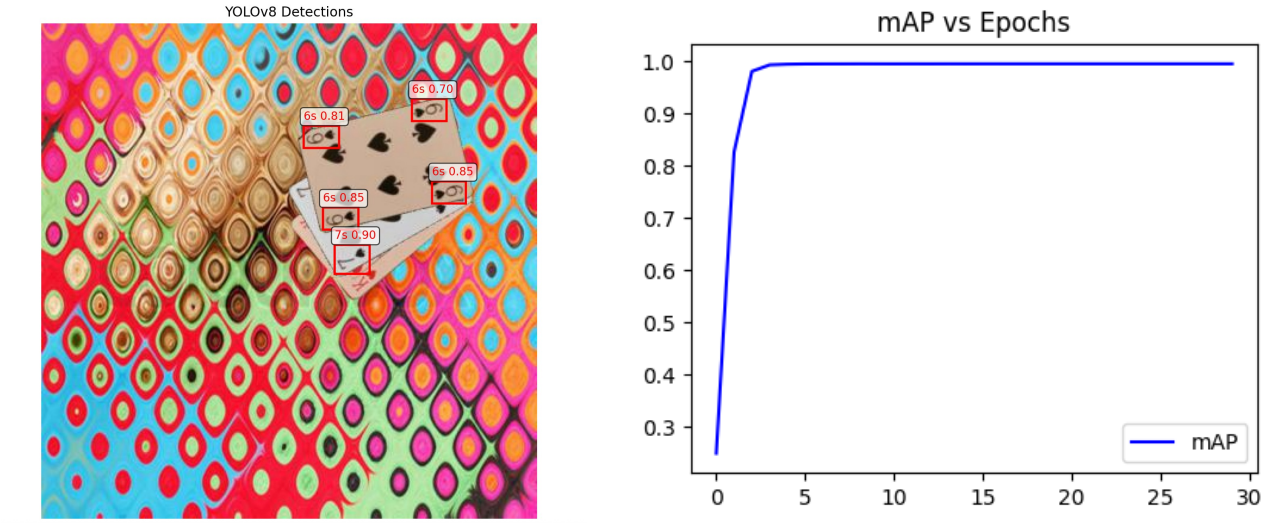
(b) Example of misdetection

Figure 7: The misdetection is likely caused by the small size of the label. The misclassification instead might be due to a combination of small size and proximity of the labels. This is a recurrent error in the test set.

(a) Real test respecting perspective

(b) Real test

Figure 8: The model generalizes well to real-life scenarios with other cards and controlled conditions, but is not robust to different perspectives.



(a) The same image as before

(b) Validation mAP

Figure 9: Results from YOLOv8. Note that although it reaches almost perfect accuracy, it sometimes fails where our model instead succeeds.