ISCTE – Instituto Universitário de Lisboa

Object Detection: Rock, Paper, Scissors

Joao Almeida nº87583
Joao Brandao nº105113
Sara Ataíde Santos nº131804

Curricular Unit: *Aprendizagem Profunda para Visão por Computador*
Lisboa, January 11, 2026

Abstract

Context: In the following report we showcase the combination between image processing with Convolutional Neural Networks (CNN) and a gamified application to create a real-time Rock-Paper-Scissors game using standard webcam hardware.
Objective: Develop a two-player game that detects hand gestures (Rock, Paper, Scissors, Gun, Thumb Up, Thumb Down) in real time using the YOLOv11 model trained on a custom dataset, with game logic, scoring, and an immersive HUD.
Method: Four-phase approach:

1. Generate and label a custom dataset with bounding boxes using Roboflow;
2. Train a YOLO11s model with data augmentation;
3. Implement real-time detection with OpenCV and YOLO tracking;
4. Implement the final gamified app that identifies hand signs for the popular game: ”Rock, Paper Scissors”

Results The trained model achieved precision 0.965, recall 0.966, and mAP50 0.982. The system runs in real time, tracks players accross time, and provides an immersive gaming experience with a functional HUD and robust edge-case handling.

Contents

1 Introduction

1.1 Context

Gesture recognition is currently a fundamental area in Computer Vision, with growing applications in human–computer interaction (HCI), augmented reality, robot control, and the interpretation of sign language [1]. Convolutional Neural Networks (CNNs) stand out as the predominant architecture for gesture recognition, demonstrating a superior ability to extract relevant spatial features in images and videos [2].

1.2 Motivation

This project arose from the idea of implementing an object detection project after looking at an old project on sign language. This presented an opportunity to apply the fundamental concepts of the course unit in a practical application, making it possible to understand the development pipeline of a Computer Vision system, from the dataset generation to implementing a real-time classification system.

The choice of theme for this project, gesture recognition for the Rock, Paper, Scissors game with real-time gameplay using a webcam—is justified for several reasons:

- Relevance: It helps consolidate knowledge about convolutional architectures, model optimization, and real-time image processing;
- Challenge: Processing video frames in real time and detecting gestures via the webcam introduces additional challenges, such as lighting variations and different backgrounds, among others;
- Familiarity: Rock–Paper–Scissors (RPS) is universally known and played since childhood. This familiarity makes it easier to communicate the results and demonstrate the system;
- Gamification: Implementing an interactive application in which the user can actually play against another user turns this project into a fun experience

1.3 Objectives

The main objective of this project is to create a new way to play the classic Rock Paper Scissors (RPS) game. On the way to reaching this main goal we set sub goals for ourselves such as training a CNN model capable of detecting the signs required for both playing the game and also interacting with the interface, symbols for start and restart of the game. Training a model for our custom symbols entails a custom dataset, which we generated through a python script. Evaluating the model's performance and therefore tweaking the dataset and other parameters. Finaly implementing the actual app that is fed from the live webcam feed and is able to track the players hands, correctly classify their hand signs and keep track of each player's score.

2 Datasets

The core of the system's intelligence lies in its ability to recognize specific hand gestures. We wanted the model to be accurate to our specific signs for interacting with the application, therefore we decided early on that we would need to create our own dataset.

2.1 Dataset Generation

The dataset was designed to go beyond the standard Rock-Paper-Scissors signs, incorporating control gestures for system interaction and an extra sign to make the game more interesting.

The system recognizes seven distinct classes:

- **Game Signs:** Rock, Paper and Scissors, used for the main game logic.
- **Control Signs:** ”Thumbs Up” and ”Thumbs Down”, allow players to navigate throught the game phases.
- **Special Signs:** ”Gun”, was included as a custom gesture designed by the team to test possible alternate rulesets for the game such as Rock-Paper-Scissors-Gun.

2.1.1 Image Capture

To ensure robustness, a dataset generation python script from an original Kaggle source [3] was adapted and modified to be used to capture images. The code took consecutive image captures of the computers webcam until it reached 200 images, each group member used the script in their own surrounding and at varying times of day to create a varied combination of backgrounds and lighting. The methods for image capture varied from capturing each of the hand signs at varying angles and distances to capturing real games of RPS.

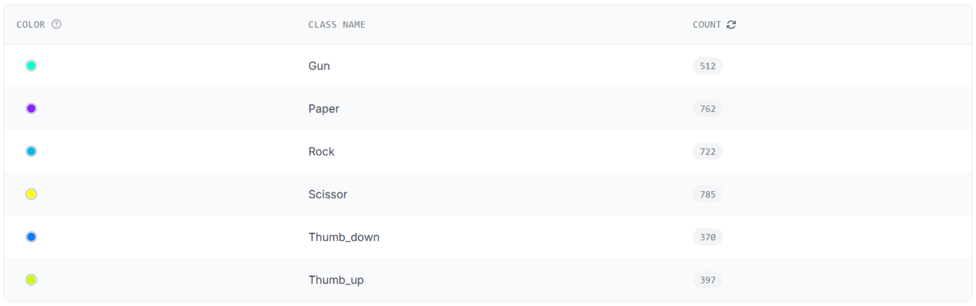
Later in the process when it was deemed that certain hand signs were underepresented in the datasat additional image captures were done in order to balance the dataset and avoid bias in the classification.

2.1.2 Annotation

The annotation was done through Roboflow [4]. When annotating images where the hands were too blurry due to rapid motion or with hand gestures in between the aformentioned signs, the hand sign was left purposefully unannotated. This was to avoid false positives where the algorithm would attempt to classify the hand signs before any of the players finished the motions that precede throwing out Rock Paper or Scissors.

This decision was done early in the process of creating the system before the details of the classification logic were fully thought out and would later cause complications to be addressed in the results.

At the end of the annotation process, after aditional images were added to compensate for the least represented classes this was the end result. After having our dataset complete and labeled, each symbols representation can be seen in fig1 with around 700 images for the sings Rock, Paper and Scissor, around 500 for the Gun symbols and around 300 for the Thumbs up and Thumbs down symbols.

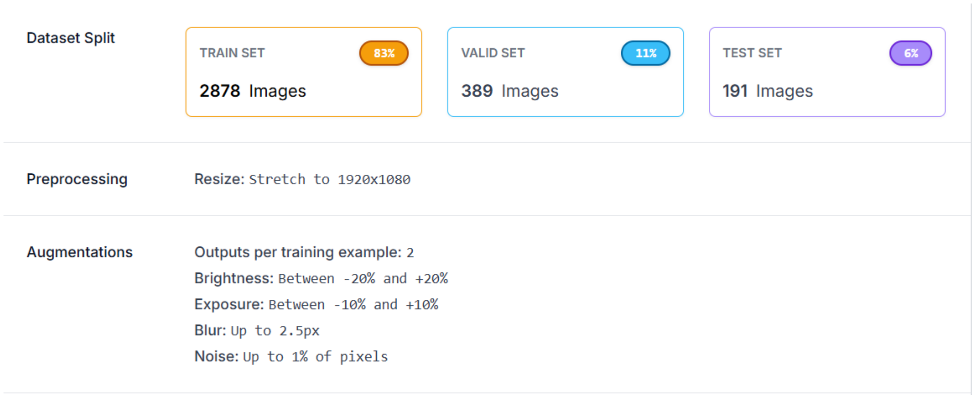


COLOR	CLASS NAME	COUNT
●	Gun	512
●	Paper	762
●	Rock	722
●	Scissor	785
●	Thumb_down	370
●	Thumb_up	397

Figure 1: Dataset - Annotations Roboflow

2.1.3 Data augmentation

For data augmentation roboflow was also used. Among the preprocessing and data augmentation options we used **Scale** to 1920x1080 along with **Brightness, Exposure, Noise** and **Blur** as shown below applied to random images until the train set was 2x bigger than the original this helped remove any reaining biases due to diferences in our respective environments and lighting. The Training, Validation and Test split was at 83%, 11%, 6% respectively after augmentation.



Dataset Split

TRAIN SET 83% 2878 Images

VALID SET 11% 389 Images

TEST SET 6% 191 Images

Preprocessing

Resize: Stretch to 1920x1080

Augmentations

Outputs per training example: 2
Brightness: Between -20% and +20%
Exposure: Between -10% and +10%
Blur: Up to 2.5px
Noise: Up to 1% of pixels

Figure 2: Dataset - Data Augmentation Roboflow

3 Description

The system developed in this course unit consists of an interactive “Rock, Paper, Scissors” (RPS) game application. The goal of the system is to correctly identify the labels from the webcam. This system incorporates gamification logic, using a visual interface (Heads-Up Display) that overlays relevant information on the video stream, such as the players’ scores.

3.1 Model Training

Before actually developing any system we needed to have a model capable of detecting our hand signs. Training custom YOLO model involves a systematic process that begins with gathering and labeling a diverse dataset. These images are organized into a specific directory structure, consisting of ”train” and ”validation” folders, each containing separate subfolders for images and their corresponding label files. Once the data is prepared, a (`data.yaml`) configuration file is generated to define the paths to these folders and specify the object classes the model will lean to detect. A deep dive into how we got our dataset can be found in section 2.

During training, the algorithm iteratively adjusts the network weights based on the training images and evaluates performance on the validation set at the end of each epoch, reporting metrics like mean Average Precision (mAP), precision, and recall.

The Ultralytics framework automatically manages critical hyperparameters including learning rate scheduling, batch size, and data aumentation techniques. Upon completion, the best-performing model checkpoint is saved on disk.

Following training, the best model is stored and its location is referenced in the code of the main application to user and perform the predictions using the live feed from the webcam.

3.2 Main Game

Once we open the main game the players are greeted with a live view of their webcam and some information drawn on top of the frame (the HUD). This information prompts them to show their hands and perform a Thumbs up sign in order to be registered as players, this phase is what we call the **detection phase**

Once both players are registered, the system initiates the **game phase**. The system waits for both players to hold a hand sign for a set ammount of time (lock) before calculating who is the winner of the round.

3.2.1 Architecture

To ensure readability, the application was developed into multiple modules, each with a single responsibility:

- **Detection Module (`detection/`):** Uses the YOLO (You Only Look Once) model for real-time location and classification of hand gestures. This module is responsible for processing camera frames and returning coordinates (bounding boxes) and detected classes (e.g., Rock, Paper, Scissors, OK).

- **Game Management Module (`game/`):** Contains the rules for the RPS game and manages the transition between the player detection phases and the active game phase.
- **Interface and Visualization Module (`ui/`):** Responsible for all visual feedback. Includes the rendering of custom bouding boxes, complete with progress bars for ”confirming” (locking) a gesture, the confidence of the model, the tracking id and the player number.
- **State Module (`game_state.py`):** Centralizes current game information, such as the duration of the game, the ID of detected players, their accumulated score, their locked in gesture and more.

1. **Frame Acquisition**: Frames are captured from the native webcam of the device.
2. **Object Tracking**: Rather than performing independent detections on every frame, the system utilizes the `model.track()` function. This assigns persistent unique IDs to each hand, allowing the system to maintain player consistency even if a hand briefly exits the frame or crosses the central boundary.

3.2.2 Detection Phase

The detection phase is the system's entry point, handled primarily by the `yolo_handler.py` and `hand_tracking.py` modules. During this stage, the YOLO model performs real-time inference to identify hand gestures and assign unique tracking IDs to every detected hand. To transition from a ”pending hand” to a registered player, a user must perform a `THUMB_UP` gesture and maintain it for a specific duration. This temporal validation, or ”locking” mechanism, is tracked via a progress percentage bar drawn as part of our custom bounding box. Once two distinct hands have successfully locked with a thumbs up gesture, their tracking IDs are assigned to player slots, and the system transitions to the active game phase.

3.2.3 Game Phase

Once players are registered, the `Game Module` orchestrates the competition logic through the `phases.py` and `rules.py` files. In this phase, the system continuously monitors the tracking IDs of both players to update their current signs. The competitive flow relies on the same locking mechanism mentioned before: both players must hold a playable gesture (Rock, Paper, Scissors or Gun) for a set duration.

Rules When both gestures are locked, the winner is determined based on classic RPS rules: Rock beats Scissors, Paper beats Rock, and Scissors beats Paper and some additional rules for our 4th sign.

The gun wins to every other sign. Like they say, never bring a sword to a gun fight. We wanted to add more symbols to the game and this secret symbol brings an element of surprise to the game. We know this 4th symbol is much more powerfull than all symbols and brings a lot of unbalance to the classic game, in the future we can work on limiting its power implementing a limit to how many times it can be used within a game.

Edge cases To maintain accurate feedback to the players, the `PlayerTimeoutManager` monitors player visibility. If players become invisible a timeout progress bar will be displayed at the bottom center of the screen warning the players to bring the hand back to the visible area. If a hand is invisible for more than 60 seconds, the module triggers a state reset back to the detection phase to allow for re-registration.

3.2.4 User Interface and Heads-Up Display (HUD)

The Interface and Visualization module, centered in `hud.py` and `bounding_boxes.py`, provides critical real-time feedback to the users. The HUD overlays several layers of information onto the video feed:

- **Detection Overlays**: Custom bounding boxes are rendered around hands, with colors and thicknesses that change dynamically based on the lock state.
- **Progress Indicators**: Both registration and round-locking progress are visualized through character-based progress bars drawn directly beneath the bounding boxes.
- **Game Statistics**: During the game phase, the HUD displays the current score for each player, their tracking IDs, and the results of the previous round.
- **Safety Warnings**: If the timeout manager becomes active due to missing players, a red warning bar and a countdown timer appear at the bottom-center of the screen to notify users of an impending session reset.
- **Help System**: A toggleable help UI provides users with keyboard shortcuts, such as 'R' for manual restart or 'Q' to quit.

For more information about the project and how it is implemented you can visit the github repository at `https://github.com/Jonny-Brandony/APVC-RPS-Project` [5] and on youtube we host a video demonstration of two players playing a game of rock paper scissors using our project: `https://www.youtube.com/watch?v=5xVqWNhR5iI` [6].

4 Results

The system's performance was evaluated based on the training metrics of the YOLOv11 model and the real-time operational efficiency of the game logic. This chapter presents comprehensive analysis of the model's convergence behaviour, detection accuracy, and classification performance across all gesture classes.

4.1 Dataset Distribution

The training dataset consisted of 6,046 instances distributed across six classes, as illustrated in Figure 3. The distribution shows Scissor with the highest representation (1,178 instances), followed by Paper (1 104), Rock (1 018), Gun (700), Thumb_up (526), and Thumb_down (520).

Initially we thought that this disparity would bring some inconsistencies to the model's behaviour while using the app, but overall the model gave surprising results. The only weakness was with the Gun being confused by the Scissor and Thumbs up symbols, we attribute this to their inhate similarity of having fingers pointing to one direction (Gun and Scissor) and the thumbs pointing up (Gun and Thumbs Up).

We hope that improving the representation of the gun would solve this issue. Surprisingly the thumb up and thumb down symbols were consistently correctly predicted despite being the most underrepresented symbols.

4.2 Loss Curves

The Figure 4 presents the comprehensive training and validation metrics over approximately 60 epochs. The training exhibited healthy convergence patterns across all loss components, demonstrating stable learning without overfitting.

The bounding box loss (box_loss) decreased from approximately 1.1 to 0.45, indicating improved localization accuracy. Classification loss (cls_loss) reduced from 1.8 to 0.25, demostrating the model's enhanced ability to correctly identify object classes. Validation losses followed similar trajectories, stabilizing around epoch 50, with validation box loss at approximately 0.77, classification loss at 0.32.

4.3 Overall Performance Metrics

The trained model achieved exceptional performance metrics on the validation set:

- **Precision:** 0.99 (99%) - Near perfect accuracy in positive predictions, with minimal false positives.
- **Recall:** 0.99 (99%) - Excellent detection rate, capturing almost all true instances.
- **mAP50:** 0.987 (98.7%) - Outstanding performance at IoU threshold of 0.5.
- **mAP50-95:** 0.89 (89%) - Strong performance across multiple IoU thresholds, demonstrating robust localization.

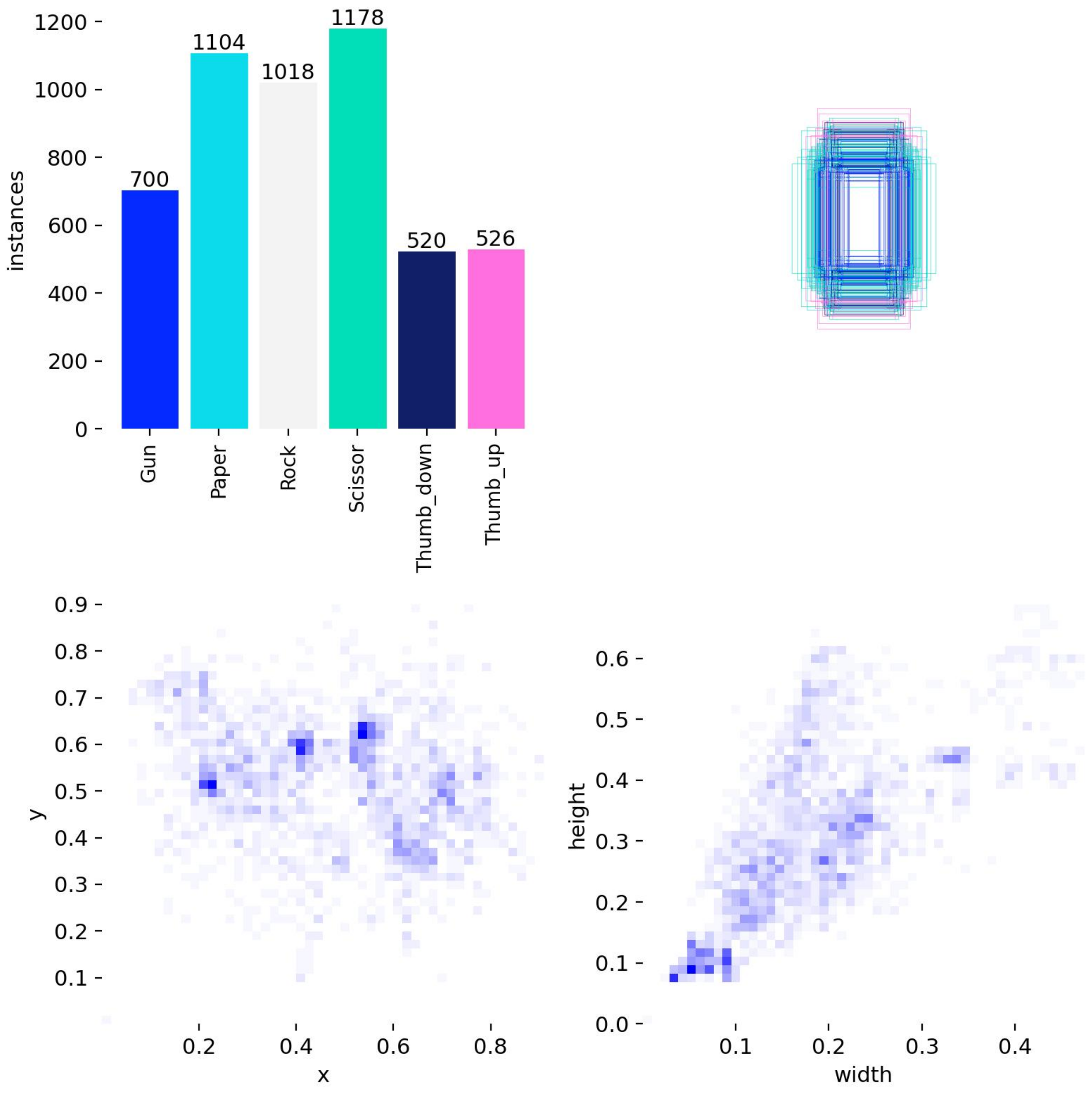


Figure 3: Dataset Distribution

These metrics indicate that the model achieves both high precision and high recall simultaneously, representating an optimal balance for real-world deployment.

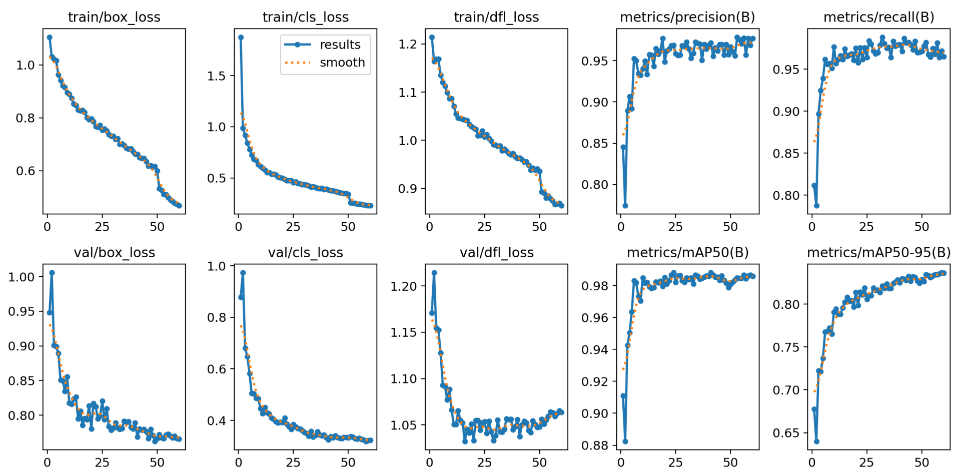


Figure 4: Results - Metrics

4.3.1 Precision-Recall Analysis

The Precision-Recall curve in Figure 5 illustrates the trade-off between precision and recall across different confidenve thresholds. The curve demonstrates excellent performance width all classes achieving mAP50 values above 97.9%. The overall mAP50 values above 0.987 at a confidence threshold of 0.987 indicates that the model maintains high precision while capturing nearly all positive instances.

4.3.2 Confusion Matrix

As shown in Figure 6, the Confusion Matrix provides a granular look at the classification performance across all five hand gesture classes.

- Diagonal Accuracy: High values along the main diagonal confirm that most gestures are correctly classified by the YOLO model.
- Common Misclassifications: The matrix helps identify specific ”confusion” points, such as the system occasionally mistaking a ”Rock” for a ”Thumb-up”.
- Background False Positives: The ”Background” class shows whether non-hand objects in the enviroment are being incorrectly identified as gestures.

Minimal confusion between gesture classes, indicating strong discriminative features learned by the model. The primary source of error is confusion with background rather than between gesture classes.

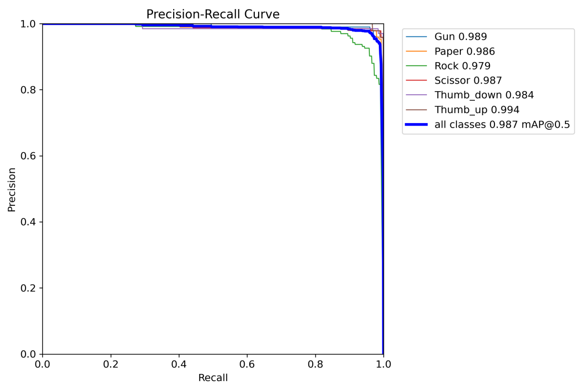
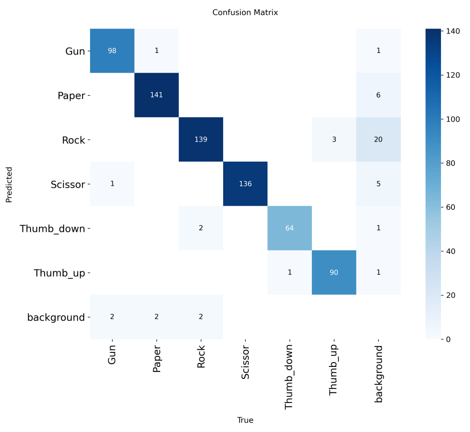


Figure 5: Precision-Recall Curve



Predicted \ True	Gun	Paper	Rock	Scissor	Thumb_down	Thumb_up	background
Gun	98	1					1
Paper		141					6
Rock			139			3	20
Scissor	1			136			5
Thumb_down			2		64		1
Thumb_up					1	90	1
background	2	2	2				

Figure 6: Confusion Matrix for the five gesture classes

5 Conclusion

5.1 Performance

The comprehensive evaluation demonstrates that the YOLOv11 model achieves exceptional performance for hand gesture recognition, with near-perfect precision and recall metrics, minimal inter-class confusion, and robust performance across varying confidence thresholds when used to classify images.

In terms of usage for the application the model showed increasing difficulty classifying hand signs and keeping tracking the further away they were from the camera, the best performance being when the user was visible from the torso up. This is expected as most of the images from the dataset were taken at that distance the furthest ones being taken with users seen from the knees up and will not be a problem as users are expected to use the application from around that distance.

5.2 Future Work

When testing the code there was an issue where any player that shook their hand too much like in the begining of any game of RPS, the script would lose the player id for that hand, this was because when creating the dataset hands that were in between signs or hands that were too blurry were deliberately left unannotated to avoid false positives.

A possible solution to this would be to create a ”Blur” or ”Hand” class for annotation and to add sufficient images to the dataset for it to be recognized, this way when a player shakes their hand or any other sudden movement the tracking wouldnt be lost.

In addition the current version of the project only supports one rule set but new gamemodes can be added in the future to support other variations os Rock-Paper-Scissors or other hand sign based games, along with this it would need new UI to represent the gamemode choice and new handsigns added to the dataset for both the new games and to navegate the UI. Alongside these changes, we would need to add these additional signs to our dataset in the same proportion as our current signs to avoid bias.

Another future improvement would be to train with a larger base YOLO model, as the current project was trained on top of YOLO small. We tried to train the YOLO Medium but ran into errors related with inssufficient memory in our computers, as such we are curious how the model would improve tracking and recognition by using a larger base model.

5.3 Closing Statement

The project has succeeded in its objective of creating a game that detects hand gestures in order to score games of Rock-Paper-Scissors, achieving high accuracy but limited tracking an issue that is understood and can be addressed with simple changes.

Overall its design and implementation demonstrates the practical application of concepts and skills covered in the course from dataset generation to model training and evalutation.

References

[1] M. Oudah, A. Al-Naji, and J. Chahl, “Hand gesture recognition based on computer vision: A review of techniques,” *Journal of Imaging*, vol. 6, no. 8, p. 73, 2020. DOI: 10.3390/jimaging6080073 [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC8321080/

[2] P. Molchanov, S. Gupta, K. Kim, and J. Kautz, “Hand gesture recognition with 3d convolutional neural networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 1–7. DOI: 10.1109/CVPRW.2015.7301342 [Online]. Available: https://ieeexplore.ieee.org/document/7301342

[3] A. Donciu-Julin, *Rockpaperscissorscnn*, https://github.com/alexdjulin/RockPaperScissorsCNN, GitHub repository, 2024.

[4] *Roboflow*, https://roboflow.com/, Website used to perform data labbelling.

[5] J. Brandão, J. Almeida, and S. Santos. “APVC-RPS.” GitHub repository, Accessed: Jan. 11, 2026. [Online]. Available: https://github.com/Jonny-Brandony/APVC-RPS-Project

[6] J. Almeida, *Demo: Rps*, Youtube video, https://youtu.be/5xVqWNhR5iI, 2026.

6 Annex

In figure 7 we can see the batch predictions made by the custom model on the validation subset. With this image the reader can have a visual understanding of how the images in the dataset were taken and how the model behaved in predicting the hand signs within them.

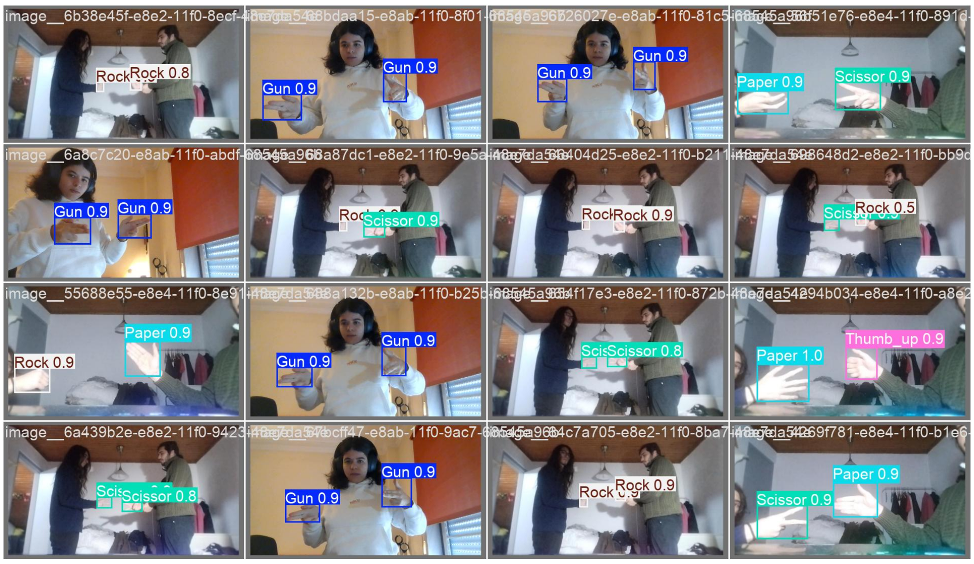


Figure 7: Batch predictions on the validation subset