CS484 Project Final Report
Number Detection from UNO®Cards using YOLOv5

Efe Tarhan
22002840
Electrical and Electronics Engineering Department
Bilkent University
Ankara
efe.tarhan@ug.bilkent.edu.tr

Ege Yüceel
22002840
Electrical and Electronics Engineering Department
Bilkent University
Ankara
ege.yuceel@ug.bilkent.edu.tr

***Abstract*—In this project, we explore the application of YOLOv5 (You Only Look Once version 5) for detecting and classifying UNO®cards from images with varied backgrounds. By using the "Uno Cards Dataset" and focusing on optimizing model hyper-parameters, we achieved significant accuracy in identifying card numbers and symbols. The dataset comprises images annotated with values with precise object boundary boxes, facilitating the model's training on detecting and classifying among 14 card classes. Our results demonstrate YOLOv5's effectiveness in real-time object detection tasks within complex image environments, presenting an application of computer vision implementation for in the domain of games.**

***Index Terms*—computer vision, deep learning, YOLOv5, detection, classification**

I. Introduction

Object detection and classification remain central challenges in the field of computer vision, with applications ranging from surveillance and security to aiding in the development of autonomous vehicles. In recent years, the advancement of deep learning algorithms has significantly enhanced the capability of computer vision systems to recognize and classify objects within images and real-time video feeds accurately. Among these advancements, the YOLO (You Only Look Once) series has emerged as an important architecture due to its balance between speed and accuracy, making it particularly suitable for real-time applications.

The YOLO architecture has undergone several iterations, each improving upon its predecessor in terms of detection accuracy, speed, and model efficiency. A version of these iterations, YOLOv5, introduces a range of enhancements, including architectural optimizations and training techniques, which further improve its performance. Given YOLOv5's capabilities, this project explores its application in a niche yet practical scenario: detecting and classifying UNO cards from images. UNO, a popular card game with a diverse set of cards, presents a unique challenge for object detection systems due to the variety of numbers, and symbols that must be accurately identified and classified.

This project aims to leverage YOLOv5's deep learning framework to develop a robust model capable of detecting and classifying UNO cards under varying conditions, including different backgrounds and card orientations. By using a custom "Uno Cards Dataset", this study focuses on optimizing model hyperparameters to achieve high accuracy in card detection and classification. The choice of UNO cards as the object of detection allows us to demonstrate YOLOv5's versatility and efficiency in handling complex image environments, thereby showcasing its potential applicability across a broader range of computer vision tasks.

In doing so, this research not only contributes to the ongoing development and understanding of YOLO architectures but also provides insights into the practical considerations and challenges involved in adapting deep learning models for specific, real-world applications. The following sections will detail the problem definition, dataset characteristics, model architecture, training process, and the results achieved through this project, offering a detailed overview of the application of YOLOv5 in the nuanced task of UNO card detection and classification.

II. Problem Definition

Our aim in this project is to detect and classify UNO®cards by the numbers and symbols on their faces on different background images. For this task we decided to use YOLOv5 [1] which is a powerful deep learning based tool for computer vision tasks. Given the input image, the problem is finding the bounding boxes for the labels with their corresponding number or symbol labels. The possible labels range from 0 to 14 where the first 0 to 9 labels represent the numbers while the remaining labels encode the special cards that do not have a distinct number. Our problem is not only to train such a model but also find the optimal set of parameters in terms of learning rate, momentum, number of epochs and batch size that gives the most accurate results. We will be using a custom dataset called The "Uno Cards Dataset" [1] which can be found in the website Roboflow throughout the project.

III. Dataset

The ”Uno Cards Dataset” [2] used for the project contains and there are in total 8992 raw images of UNO®cards on various backgrounds. These images contain not only 1 card but around 3-5 cards next to each other or stacked. The input features of the dataset are raw RGB images that may or may not contain an UNO®card, whereas the outputs consist of a 5-dimensional value vector for each card in an image. This vector contains the categorical label of the card, the horizontal and vertical starting corner pixel of the image, and the horizontal and vertical pixel width of the image, which allows the algorithm to create a bounding box around the region of interest and assign a label to the card with a probability. It must also be noted that there can be multiple UNO®cards in an image, which will result in several feature vectors for each image. All images in the dataset have the size of 416x416 with RGB pixels. A sample image and its corresponding value object is shown in Figure 1.

Fig. 1. A sample image from the UNO®Cards Dataset

```
0 0.32211538461538464 0.23677884615384615
    0.07692307692307693 0.08413461538461539
0 0.4026442307692308 0.2860576923076923
    0.08413461538461539 0.08653846153846154
13 0.48197115384615385 0.37259615384615385
    0.0889423076923077 0.08653846153846154
```

From the value vector presented in Figure 1, it is evident that all three UNO®cards depicted in the image have been successfully detected. The value ”13” represents the class label of the marked with the number ”8” whereas the value ”0” represents the cards with value ”0”. This value mismatch is due to the alphanumeric order of the labels. The subsequent four numbers highlight the coordinates of the bounding box encapsulating the card’s numeral on the image. This small result was a part of an initial test training. Our over-arching goal is to fine tune the model to perform even better.

Also the percentage of the data allocated for training, testing and validating purposes can be seen from Figure 2.Throughout this project we will be dealing with detecting and classifying the images with cards in this dataset.

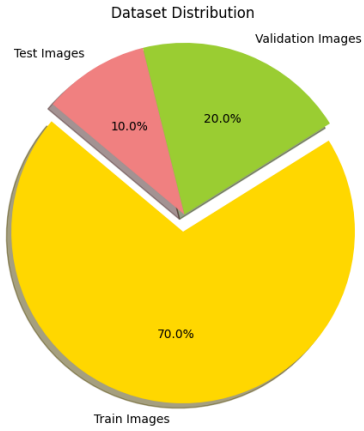

Fig. 2. Dataset Distribution

IV. Model Architecture

YOLO v5, is an advanced deep learning algorithm used for object detection tasks, part of a series of YOLO algorithms that prioritize speed and accuracy. The architecture of YOLO v5 can be divided into several key components, each with a specific function that contributes to the model’s overall performance. The model will be explained using the documentation provided by Utralytics [3]. An flow chart of the model architecture can be seen from Figure 3

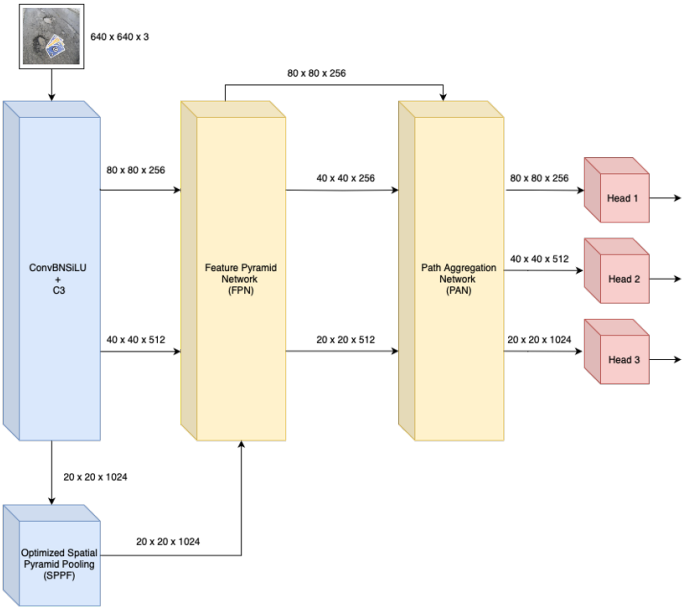

Fig. 3. Model Architecture

The backbone of YOLO v5, typically CSPDarknet53, uses a series of convolutional layers to extract features from input images. The CSP (Cross Stage Partial) structure optimizes the Darknet53 by dividing the feature map of the base layer into two parts and then merging them through a cross-stage

hierarchy. This method reduces the computational cost and enhances the learning capability of the network. The convolutional layers within this part vary in filter size and stride, enabling the extraction of features at different scales and complexities. Batch normalization and Leaky ReLU activation functions follow these convolutional layers to stabilize the learning process and introduce non-linearity. The architecture illustration of the backbone of the model can be seen from Figure 4

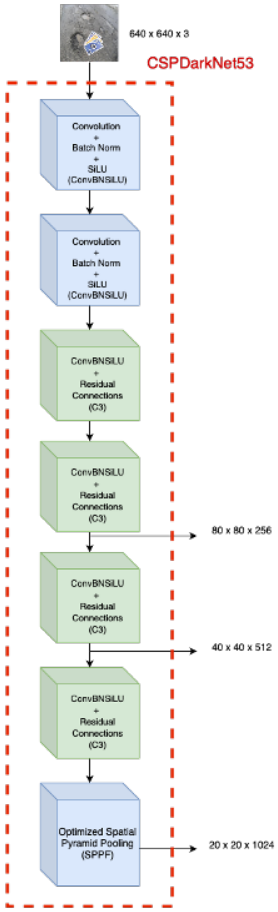

Fig. 4. Backbone of the Model Architecture

Following the Backbone is the Neck, which is tasked with feature fusion and refinement. The Neck of YOLO v5 employs an innovative combination of FPN (Feature Pyramid Network) and PAN (Path Aggregation Network) for feature fusion and enhancement. FPN improves the network's ability to detect objects at different scales by creating a pyramid of feature maps at multiple levels and then upscaling and merging these maps to ensure rich semantic information at all levels. [1] PAN further refines this process by enhancing the flow of information in both bottom-up and top-down directions, ensuring that features from all levels are effectively aggregated and utilized. This structure involves additional convolutional layers for processing the feature maps, allowing for the detailed detection of objects regardless of their size.The architecture illustration of the neck of the model consisting of the FPN and PAN blocks can be seen from Figure 5 [1]

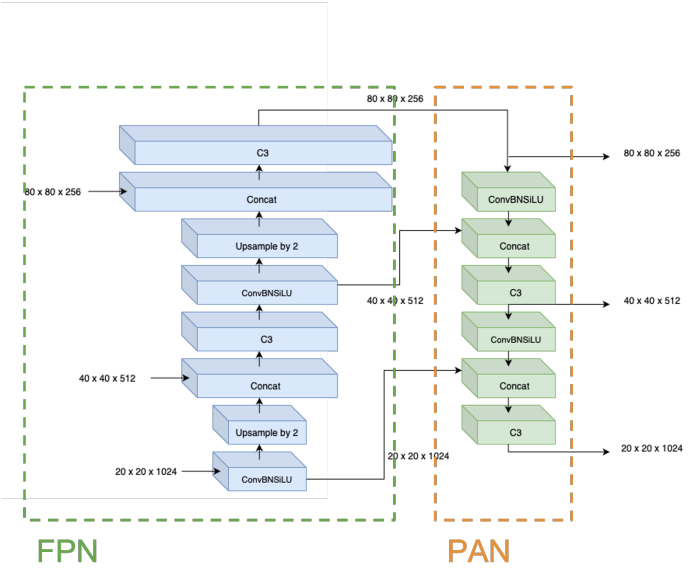

Fig. 5. Neck of the Model Architecture

The third critical component is the Head, which is the final part of the model. The detection head of YOLO v5 is responsible for the final prediction of bounding boxes, object classes, and confidence scores. It employs convolutional layers designed to map the enriched feature maps to a tensor containing the predictions. This tensor is organized such that each cell corresponds to a region in the input image and contains information about bounding boxes (coordinates and size), objectness score (the probability that an object exists within the bounding box), and class probabilities (the likelihood of each object class being present). The design allows for parallel processing of all grid cells in an image, significantly speeding up the detection process [1]. Illustration of the head part of the model can be seen from Figure 6.

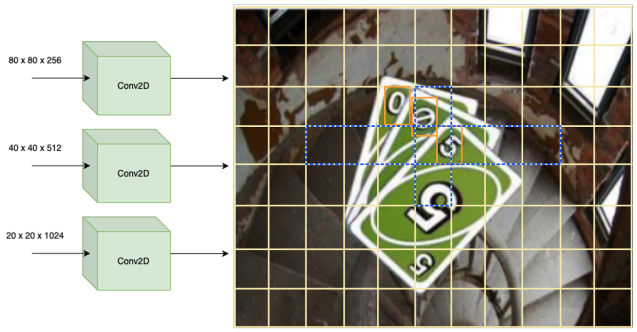

Fig. 6. Head of the Model Architecture

The YOLO v5 architecture incorporates several advancements and optimizations over its predecessors, enhancing its capabilities. One of the key capabilities is its speed, enabling real-time object detection, which is crucial for applications requiring instant feedback, such as autonomous vehicles and

surveillance systems. Additionally, YOLO v5 introduces improvements in accuracy, thanks to optimizations in model architecture and training procedures. It also offers better scalability and flexibility, allowing for customization and adaptation to various object detection tasks with different complexity levels. Furthermore, YOLO v5 supports a wide range of image sizes, enabling it to maintain high performance across different resolutions, which is particularly beneficial for applications with varying input image quality.

V. Hyperparameter Tuning

In the process of hyperparameter tuning, four optimization objectives are chosen. These are the number of epochs, batch size, learning rate and momentum. A wider range of parameters may also be chosen but these parameters are decided to be the most important ones for almost any kind of neural network training procedure. For the number of epochs, the range between 5 and 10 is scanned. The candidate options for batch size were 8, 12 and 16. Moreover, the learning rate and the momentum is searched between the ranges 0.0001-0.1 and 0.85-0.99, respectively. These ranges are decided intuitively by considering what are the hyperparameter selections are in general for such models.

After defining the set of hyperparameters to be optimized along with their search range, now an important step is to define what kind of algorithm will be used for performing the hyperparameter tuning. An initial thought would be to use the most basic "grid search" algorithm where we one by one look for all the possible combinations and decide the optimal solution. The solution of such an algorithm would definitely be the optimal solution; however, performing such a search would be computationally expensive and our computational resources are not sufficient for performing this task effectively. On the other hand, the other method, "random search", may be computationally more feasible by selecting arbitrary points from the hyperparameter space, but then the results would most likely be not optimal. To find a trade-off between these two edges, it was decided to use "Bayesian Optimization" that iteratively updates the distribution of accuracy for the hyperparameters and gradually comes up with better estimations at each attempt. To run such an algorithm, a widespread algorithm called "Optuna" is used. [4] Optuna can automatically handle the bayesian optimization over a defined number of iterations given the optimization objectives with their acceptable ranges. [4]

The optimization process with Optuna is initialized with 15 iterations (meaning 15 different models) which our computational resources was enough for. The results are displayed in 8. It is observed that the algorithm started training with a very bad selection of hyperparameters, resulting with an accuracy around 0.3. However, as the iterates continue, algorithm gradually builds it way up to more than 0.8 percent. It is also observed that between iterations, the accuracy metric sometimes drop very rapidly. This is due to the algorithms trade-off between exploration and exploitation. While Optuna

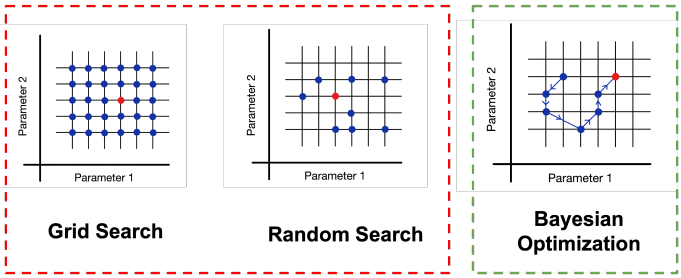

Fig. 7. Graphical comparison of search algorithms

aims to increase accuracy by valuing hyperparameter selections that give the best output, it also sometimes searches for other alternatives to scan the hyperparameter space. As we move towards the end of iterations, it can be seen that the exploration steps are over and the last 4 results are above 80% accuracy. This accuracy metric is mAp: $0.5-0.95$ which is an alternative of the standard "Mean Average Precision" but more pessimistic by setting the "Intersection over Union" threshold much higher than the standard metric. We have specifically chosen this metric because a lower and more varying score allowed the Optuna to better differentiate the better options, as the standard mAp output gave accuracy results near 98%. The results can be viewed from 8.

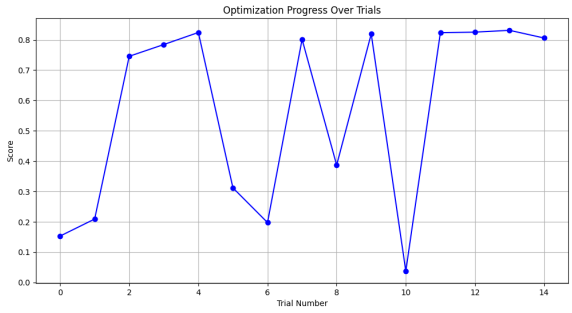

Fig. 8. Accuracy plot for different trials for optimizing the hyperparameters of the model.

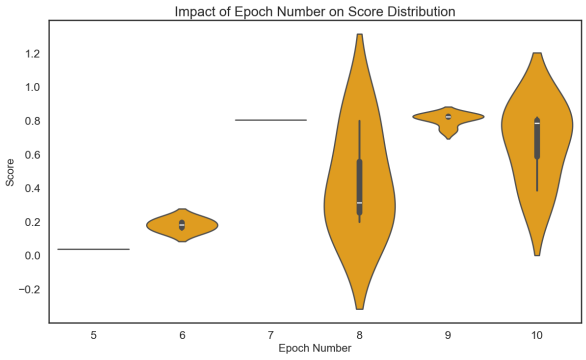

Fig. 9. Distribution of the scores for different epoch numbers

It is also interesting to observe the individual impacts of

each hyperparameter to the overall accuracy of the model. In 9, the accuracy distributions with different epochs are shown. Some of the distributions are very narrow due to the small number of trains due to the selection of parameters by algorithm. However, overall there exists a pattern where as the number of epochs increase, the mean accuracy for each distribution increase as well, showing a positive correlation between the epoch size and accuracy. On the other hand,

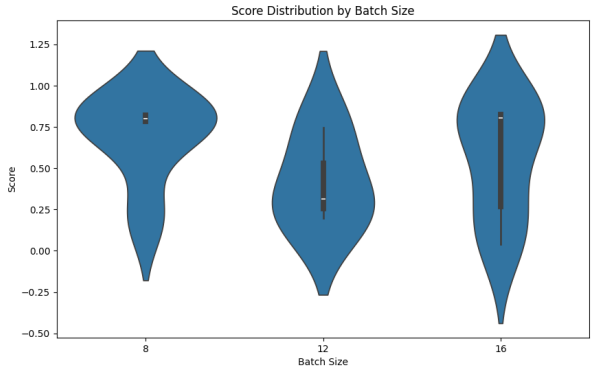

Fig. 10. Distribution of the scores for different batch sizes

such a correlation is not easily observed with 10 where the impact of batch size is investigated. There is high variance for each candidate and this allows us to conclude that there is no direct correlation (at least linear) between the batch size and accuracy, which will also be seen later in the heatmap.

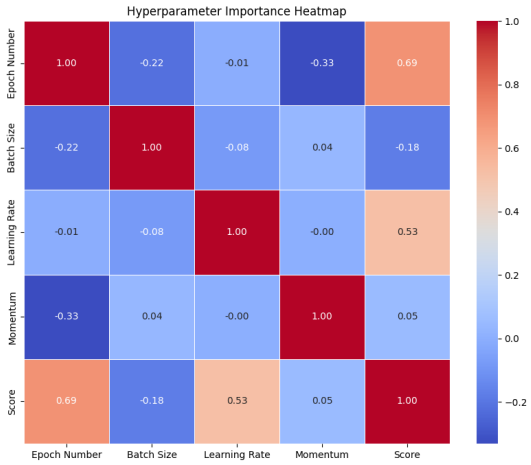

Fig. 11. Heatmap of the correlations between hyperparameters and obtained score

To take the analysis even further, it is suitable idea to not only investigate the impact of different hyperparameters on score but also their correlation with each other. From 13, our analysis about the epoch size can be verified since there is a high correlation of 0.69, while that number is -0.10 for batch size - score, indicating the uncorrelatedness interpretation that is achieved from looking at the violin plots. An interesting result is the positive correlation between the learning rate and the score with 0.53, meaning that the score increased as we increase the learning rate within the given range. A more surprising result is the negative correlation between the momentum and the number of epochs. The heatmap suggests that as the number of epochs increase, the momentum should be smaller to preserve the accuracy. After

Epoch Number	Batch Size	Learning Rate	Momentum	Score
10	16	0.014289	0.940214	0.83127

TABLE I
HYPERPARAMETERS OF THE BEST ATTEMPT

doing the Bayesian optimization over 15 iterations, the best selection of parameters comes as I, where we achieve 83.1% accuracy in terms of mAp $0.5 - 0.95$ metric. Although the

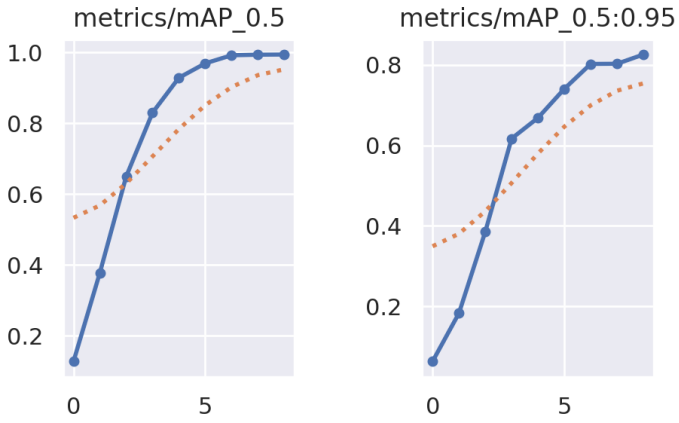

Fig. 12. Results of the selected best model for training

difference between mAp $0.5 - 0.95$ and the standard mAp$_0$.5 was previously explained, it would also be a good idea to display a quantitative difference between these metrics. In 12, we see the results of the best training for 2 different evaluation metrics. For the hyperparameter tuning mAp $0.5 - 0.95$ has been used since a lower and more variational accuracy will allow the algorithm to optimize better; however, in reality our model's accuracy is not 83.1% in terms of standard mAp, but it is actually around 99%, as shown in the first sub-figure of 12.

After the training procedure, the model is evaluated using the test set. From the obtained results a confusion matrix is created. The occurrence probabilities obtained using the frequencies can be observed from the confusion matrix given in Figure 13. The significance of these true positive rates cannot be overstated, as they serve as a clear indicator of the model's effectiveness in recognizing and categorizing each card's unique features. In the context of machine learning, especially within the realm of image classification and pattern recognition, achieving such high true positive rates is an important accomplishment. It suggests that the model is not only capable of identifying the presence of specific numbers

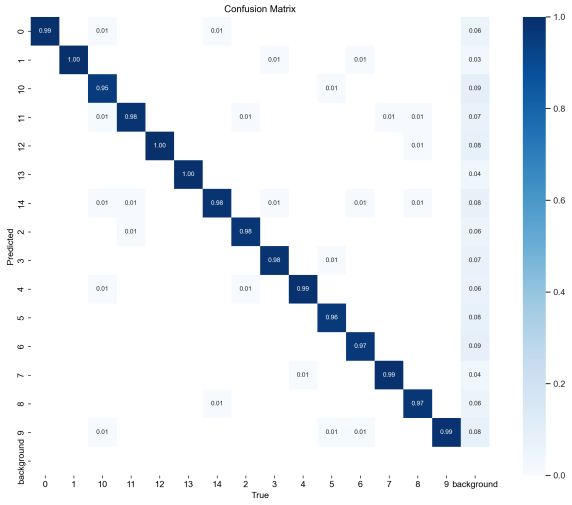

Fig. 13. Confusion matrix of the results obtained by evaluating trained model on the test set.

and symbols but also at minimizing the instances of false negatives and false positives.

Some detection and classification examples of the model on the test set can be observed from Figure 14 that are chosen randomly from the test set during training.

Fig. 14. Models detection and classification performance on random data selected from the test set created for training

VI. Baseline Comparison

Extensive research has been conducted to identify any other models trained with the same dataset, enabling us to provide a baseline and compare our results with their counterparts. However, it was found that there are no other models trained with different networks, but there was one person who trained the same YOLOV5 model with the same dataset [5]. Therefore, we decided it would be a good idea to evaluate the success of the hyperparameter tuning process to determine whether our hyperparameter search has improved the standard performance. For 10 epochs, our selection of batch size, learning rate and momentum slightly outperforms the baseline in $\text{mAp}_0.5 : 0.95$ as $83\%.1$ over $80\%.8$ while for the standard map, we see the both models are performed almost equivalently around 99%.

VII. Conclusion

In conclusion, this project showed that using the right set of hyperparameters can really improve how well a YOLOv5 model works. We used Bayesian optimization by using the Optuna library to improve the hyperparameter selection performance with small number of experiments, and this made our model not only more accurate but also more reliable. Our best model achieved a mAp accuracy of 99% and $\text{mAp\_}0.5 : 0.95$ performance of 83.1%. Going forward, we plan to try even more advanced techniques, like evolutionary algorithms, to make our approach even better. This project proves that careful tuning of these settings is key to improving how models perform, and it lays the foundation for using these techniques on other models and datasets in computer vision.

VIII. About the Code

In the code zip, code.ipynb and plotting.ipynb contains our codes that we implemented for running the model, hyperparameter tuning and optimization while the rest of the zip file is the YOLOV5 packages of [1] [3]. The dataset is not included in the code submission due it its large size, but can be accessed from [2]. The weight file is denoted as "model.pt" and it is present in the code folder.

References

[1] "Ultralytics YOLOv5 Architecture," 2024. [Online]. Available: https://docs.ultralytics.com/tr/yolov5/

[2] A. Crawshaw, "Uno cards dataset." [Online]. Available: https://public.roboflow.com/object-detection/uno-cards

[3] G. Jocher, "Comprehensive Guide to Ultralytics YOLOv5." [Online]. Available: https://github.com/ultralytics/ultralytics/blob/main/docs/en/yolov5/tutorials/architecture_description.md

[4] Optuna Team, "Optuna: A hyperparameter optimization framework," 2024, accessed on: May 16, 2024. [Online]. Available: https://github.com/optuna/optuna

[5] E. Alexiutenko, "UNO Cards Object Detection," GitHub repository, 2023. [Online]. Available: https://github.com/EvelinaAlexiutenko/UNOCards-object-detection