Kaunas University of Technology

Faculty of Informatics

Research on techniques for automatic recognition and tracking of basketball shots from video

Master’s Final Degree Project

Juozas Širmenis

Project author

Asoc. prof. Mantas Lukoševičius

Supervisor

Kaunas, 2025

Kaunas University of Technology

Faculty of Informatics

Research on techniques for automatic recognition and tracking of basketball shots from video

Master's Final Degree Project

Artificial Intelligence in Computer Science (6211BX007)

Juozas Širmenis

Project author

Asoc. prof. Mantas Lukoševičius

Supervisor

Asoc. prof. Liudas Motiejūnas

Reviewer

Kaunas, 2025

Kaunas University of Technology

Faculty of Informatics

Juozas Širmenis

Research on techniques for automatic recognition and tracking of basketball shots from video

Declaration of Academic Integrity

I confirm that the final project of mine, Juozas Širmenis, on the topic „Research on techniques for automatic recognition and tracking of basketball shots from video" is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarised from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by Law) have been paid to anyone for any contribution to this project.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

(name and surname filled in by hand) (signature)

Širmenis, Juozas. Research on techniques for automatic recognition and tracking of basketball shots from video. Master's Final Degree Project / supervisor Doc. Mantas Lukoševičius; Informatics Faculty, Kaunas University of Technology.

Study field and area (study field group): Computer science, Informatics (B01).

Keywords: Computer Vision, Object Tracking, Basketball Shot Recognition, Video Processing.

Kaunas, 2025. 54 p.p.

Summary

This research focuses on the development of a system capable of detecting a basketball backboard, rim, net, and ball from a single static video stream. The system operates by processing frames extracted from the video, with the aim of achieving accurate and efficient object detection in real time. Different versions and configurations of YOLO models were tested and compared based on precision, inference speed, and model size. The ultimate objective is to identify the most effective setup to support a reliable shot recognition system.

Širmenis, Juozas. Automatinio krepšinio metimų atpažinimo ir sekimo iš vaizdo įrašų metodų tyrimas. Magistro baigiamasis projektas / vadovas doc. Mantas Lukoševičius; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos mokslai, Informatika (B01).

Reikšminiai žodžiai: kompiuterinė rega, objektų aptikimas, krepšinio metimų atpažinimas, vaizdo įrašų apdorojimas.

Kaunas, 2025. 54 p.p.

Santrauka

Šiame tyrime daugiausia dėmesio skiriama metodams, kurie iš vieno statinio vaizdo srauto aptiktų krepšinio lentą, lanką, tinklelį ir kamuolį. Sistema veikia apdorodama iš vaizdo įrašo išskirtus kadrus, siekiant tiksliai ir efektyviai aptikti objektus realiuoju laiku. Buvo išbandytos įvairios YOLO modelių versijos ir konfigūracijos, kurios buvo lyginamos pagal tikslumą, išvadų pateikimo greitį ir modelio dydį. Galutinis tikslas - nustatyti veiksmingiausią konfigūraciją, kuri padėtų sukurti patikimą krepšinio metimų atpažinimo sistemą.

Table of contents

List of tables

List of figures

List of abbreviations and terms

Abbreviations:

YOLO - You Only Look Once - real-time object detection algorithm family that predicts bounding boxes and class probabilities in a single forward pass.

YOLOv5 / v7 / v8 / v9 / v11 / v12 - Different versions of YOLO models, each with architectural enhancements for speed, accuracy, or efficiency.

mAP - Mean Average Precision - a standard metric used to evaluate the accuracy of object detection models by averaging precision across all classes and confidence thresholds; higher mAP indicates better detection performance.

Terms:

Computer Vision - It is a field of different solutions for the computer to process the images.

Confidence Score - A numeric value (0-1) assigned to each detection, indicating the model's certainty that the detected object belongs to a given class.

Inference Time - The amount of time it takes for a trained model to process a single input and return detection results; lower inference time implies faster, more real-time capable performance.

Active Frame - A selected and analyzed video frame for shot recognition.

Shooting Stage - The phase that initiates when the ball is above the net and moving closer to the rim, indicating the start of a potential shot.

Shot Stage - Triggered when the ball crosses a midpoint between its initial shooting position and the rim, confirming a shot attempt has occurred.

MoM Stage - Made or Missed - the evaluation phase activated when the ball drops below the net, during which the system determines if the shot was made or missed.

Introduction

Problem Relevance

The ubiquity of object detection and recognition is escalating, propelled by the proliferation of smartphones, surveillance cameras, AI solutions, and sophisticated algorithms. Our devices are now more computationally powerful than ever before. According to a source, contemporary smartphones possess greater computational capacity than the computer used for NASA's 1960s moon mission [1]. This surge in computational power has sparked a multitude of innovative applications, such as using camera feeds to identify objects and spot potential thieves or integrating cameras into manufacturing processes to detect production defects.

Various approaches are being explored for object detection. Some researchers employ sensors to detect and recognize the behavior patterns of different species [2], while others focus on recognizing human activities [3]. There are also efforts to use Computer Vision for detecting hands and other objects [4], and some are experimenting with different AI solutions [5, 6].

The application of object detection in sports is increasingly prevalent. The insights derived from AI analysis can significantly enhance the performance of an athlete or a team. The scope of AI in the sports domain is extensive: it spans from healthcare and skill enhancement to goal recognition in soccer or automated shot counting in basketball [7, 8].

The development of such systems requires careful consideration of numerous factors. These include the parameters of the camera, changes in object lighting over time, whether the object is moving, the possibility of other objects appearing in the frame, and whether the camera is stationary. There are countless scenarios to consider in this complex field.

Aim and Objectives

This project aims to investigate the suitability of the most promising artificial intelligence techniques for real-time basketball shot recognition and tracking from video. The main objectives are:

1. Capture and prepare an original basketball dataset for public availability.
2. Find a model to detect a basketball ball.
3. Find a model to detect basketball backboard and hoop.
4. Combine the two models to create a shot recognition system.

Additional objective:

1. Publish a paper at a conference based on the Master's project.

Document Structure

In this paper, we performed an analysis of image processing using computer vision algorithms and machine learning models. We also discussed the experiment environment, dataset, and YOLO models. Finally, we summarized the results and proposed the shot recognition system.

1. Image Processing Analysis

In this section, we will discuss various computer vision algorithms for computational simplicity, noise reduction, motion and edge detection, and various object recognition techniques. In addition, different solutions from a machine learning perspective are mentioned.

1.1. Computer Vision Algorithms

In this section, we will talk about different methods that make computation easier, help detect the movement and edges of objects, reduce noise, and find different objects in the image.

1.1.1. Simplify the Calculation

There are several methods that can simplify computer vision calculations, particularly when working with images. One common method is to convert images to grayscale, which reduces the amount of data that needs to be processed by eliminating color information. However, this can cause additional disturbances, as some information such as sharpness, shadows, contrast, or image texture may be lost [9].

Some other methods used to simplify the calculation of algorithms:

1. **Resizing images**. Reducing the size of an image can also help to simplify calculations, as it reduces the number of pixels that need to be processed.
2. **Applying filters**. Filters can be used to highlight specific features in an image and simplify calculations by reducing the amount of noise or irrelevant information in the image.
3. **Applying thresholding**. Thresholding is a technique that involves setting a threshold value for pixel intensity, and converting all pixels below that value to black and all pixels above it to white. This can simplify calculations by reducing the number of distinct intensity values in the image.

1.1.2. Movement detection

There are several computer vision techniques that can detect moving objects in video. Some of these techniques include:

1. **Frame Difference**. The reference frame is subtracted from the current frame, and the resulting absolute difference image is thresholded to identify pixels that have changed between the two frames. These pixels are then marked as representing moving objects in the video [10].
2. **Background subtraction**. Separating foreground from the background by subtracting the background of a video frame from the current frame to identify moving objects [11].
3. **Optical flow**. This involves tracking the movement of pixels between successive frames to detect moving objects.
4. **Feature matching**. This involves identifying and tracking specific features, such as corners or edges, in a video.

These techniques can be used alone or in combination to detect and track moving objects in a video.

1.1.3. Noise Reduction

Noise reduction is a common task in image processing that aims to improve the visual quality of an image by reducing the amount of noise present in it. There are various methods for reducing noise in images, including the following:

1. **Median filtering**. This method involves replacing the value of a pixel with the median value of the surrounding pixels. This can effectively reduce noise because the median is less sensitive to outlier values than the mean [12].
2. **Gaussian blur**. This method involves convolving the image with a Gaussian kernel, which has the effect of smoothing the image and reducing high-frequency noise [13].
3. **Wiener filtering**. This method involves estimating the power spectrum of the noise in the image and then applying a filter that is designed to minimize the mean square error between the original image and the filtered image [13].
4. **Total variation denoising**: This method involves minimizing the total variation of the image, which has the effect of smoothing the image while preserving edges [13].
5. **Non-local means denoising**: This method involves replacing each pixel with the average of similar pixels in the image, which can effectively reduce noise while preserving edges and other important details [13].
6. **Morphological operations**. Open, Close, and Dilate functions reduce noise around and inside the object and expand the object itself [14].

These are just a few examples of the many methods that are available for reducing noise in images. The choice of which method to use will depend on the specific characteristics of the noise present in the image and the desired properties of the resulting denoised image.

1.1.4. Edge Detection

In computer vision, there are several methods that can be used to detect edges in an image. Some of the most commonly used methods are:

1. **Sobel operator**. This method uses a set of convolution kernels to approximate the gradient of the image intensity at each pixel. The resulting gradient magnitude and direction can be used to identify edges in the image [15].
2. **Canny edge detector**. This method uses a combination of Gaussian smoothing, gradient computation, and non-maximum suppression to detect edges in the image. It is widely used due to its good performance and ability to suppress noise [16].
3. **Laplacian operator**. This method uses a second-order derivative to identify edges in the image. It is sensitive to both the intensity and the gradient of the image and can be used to detect both sharp and diffuse edges, although very sensitive to noise [17].
4. **Hough transforms**. This method is used to detect lines in an image. It works by projecting the image onto a parametric space and identifying the parameters that correspond to lines in the image [18].

There are many other methods that can be used to detect edges in images, and the best approach depends on the specific characteristics of the image and the application requirements.

1.1.5. Geometric Shapes Detection

The result of the image preprocessing leads to other algorithms, which find different geometric shapes in the image. These methods are ones of the most common:

1. **Hough Circles**. Find a circle-like object in the grayscale image by using a modified Hough transform [19].
2. **Harris Corner Detector**. Based on the algorithm's documentation [20]: "It basically finds the difference in intensity for a displacement in all directions."
3. **Hough Lines**. This function detects the line by using the standard or standard multi-scale Hough transform algorithm [19].

There are other algorithms for the detection of different shapes, but in this paper, we are focusing on these three.

1.2. Machine Learning Algorithms

In this section, we are going to talk about different machine learning algorithms to detect objects in a video and the results of experiments with them.

1.2.1. Oriented R-CNN

Firstly, we have to talk about R-CNN (Regional Convolutional Neural Networks) to fully understand Oriented R-CNN. R-CNN is a type of object detection algorithm that uses a convolutional neural network to classify object proposals. The main idea behind R-CNN is to take an entire image and then run a convolutional neural network (CNN) on each region proposal to classify the object inside the region.

Oriented R-CNN is an extension of R-CNN which also predicts the angle of the object. The main idea behind Oriented R-CNN is to predict the angle of the object along with the bounding box coordinates to improve the detection accuracy of objects that are in a rotated position.

With Oriented R-CNN, while using the DOTA dataset for the experiments, speed of 15.1 FPS and 75.9% accuracy of object detection was achieved [21].

1.2.2. AutoML (CFML)

AutoML (Automated Machine Learning) is a type of machine learning that automates the process of selecting and fine-tuning models for a particular task. It is also referred to as CFML (Code-Free Machine Learning). It is used to automate the process of selecting, training, and fine-tuning models for object detection tasks. This can include selecting the best algorithm or architecture for a given dataset and task, as well as automating the process of tuning hyperparameters and selecting the best model based on performance metrics.

AutoML for object detection can be particularly useful in situations where there are a large number of possible models or where the data is complex and difficult to work with. By automating the process, AutoML can help to save time and resources, and can also make object detection more accessible to people who are not experts in machine learning.

The main advantage of AutoML is that it allows the use of machine learning techniques without the need for a deep knowledge of algorithms. It also helps to reduce human effort and increase the accuracy of the models created.

AutoML was used in experiments with 31,000 images dataset. The two models were selected: RetinaNet and YOLOv3. With the first one, the 66.9% accuracy was achieved, and with the second one - 52.7% [22].

1.2.3. CNN

Convolutional Neural Networks (CNNs) are a type of deep neural network that are commonly used in computer vision tasks, including object detection. The main idea behind CNNs is to use convolutional layers, which are designed to learn spatial hierarchies of features from input images.

In object detection, CNNs are used to extract features from the images and generate feature maps. These feature maps are then used to detect objects in the image by identifying regions that are likely to contain objects.

With the 500 image dataset and using CNN, an accuracy of 98% is reached for table and chair detection [23].

1.2.4. RetinaNet-18

RetinaNet is a single-stage object detector, similar to YOLO and SSD, that aims to achieve a balance between accuracy and computation cost. It uses a ResNet-18 as the backbone network, which is a pre-trained CNN model that has 18 layers. The use of ResNet-18 allows RetinaNet-18 to extract robust features from the input images, which are then used to detect objects.

The main idea behind RetinaNet is the use of a Feature Pyramid Network (FPN), which is a neural network architecture that generates a pyramid of feature maps at different scales. This allows RetinaNet to detect objects at different scales, which is important for object detection tasks, because objects can appear at different sizes in an image.

Another important aspect of RetinaNet is the use of Focal Loss, which is a loss function that addresses the issue of class imbalance in object detection tasks. Class imbalance occurs when the number of positive samples (objects) is much smaller than the number of negative samples (non-objects) in the dataset. Focal Loss helps to reduce this imbalance by down-weighting the loss for well-classified samples and giving more weight to the hard-to-classify samples.

With the RetinaNet-18, on the nuScenes dataset, an accuracy of 48.8% was reached in the detection of different objects [24].

1.2.5. YOLO Models

The YOLO (You Only Look Once) algorithm takes a different approach to object detection compared to traditional methods. Instead of repurposing classifiers, YOLO treats object detection as a regression problem, allowing accurate localization of bounding boxes and associated class probabilities. The key innovation lies in the use of a single neural network, which enables end-to-end optimization and directly improves detection performance [25].

YOLO stands out for its simplicity. It employs a single convolutional network that efficiently predicts multiple bounding boxes and their corresponding class probabilities. Unlike conventional approaches, YOLO trains on complete images and directly optimizes the detection performance. This unified model offers several advantages over traditional methods in object detection.

To process images using YOLO, a straightforward three-step procedure is followed: the input image is resized, a single convolutional network is applied to the image, and the resulting detections are filtered based on the model's confidence threshold [25]. See the example in Figure 1.

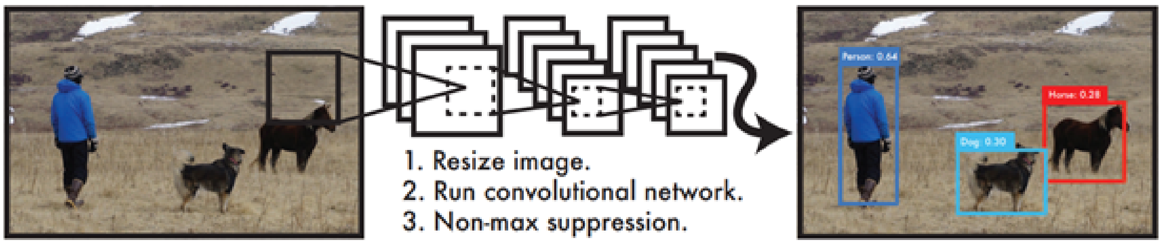


Figure 1. The YOLO detection system, pp. 779 [25]

Since its introduction in 2016, the YOLO (You Only Look Once) algorithm has spawned several variations and improved versions. Notable mutations of YOLO include YOLOv3 [26] and YOLOv4 [27], which introduced enhancements such as multi-scale predictions, feature extraction improvements, and advanced architectural modifications.

Another approach in the YOLO family is YOLOv2, which was evaluated for detecting third molar angles in dental panoramic X-rays using Winter's classification. In this application, YOLOv2 demonstrated superior performance compared to Faster R-CNN and SSD across multiple training configurations. Using ResNet-18 as the feature extractor, YOLOv2 achieved a mean average precision (mAP) of 96% on the test set, showing strong generalization and low variance. These results support YOLOv2's effectiveness for medical object detection tasks where both accuracy and speed are critical [28].

These iterations of the YOLO algorithm have made significant advancements in real-time object detection, resulting in increased accuracy and improved speed. One notable variant is YOLOv5. YOLOv5 was compared to EfficientDet on the COCO dataset, where it achieved higher mAP scores as well as faster inference speeds, demonstrating superior overall performance [29]. To gain a deeper understanding of the capabilities of YOLOv5, further investigation and analysis will be conducted [30].

The evolution and mutation of YOLO demonstrate the continuous efforts to enhance object detection performance, adapt to different application scenarios, and optimize the trade-offs between speed and accuracy. Another notable mentions include YOLOv7 [31] and YOLOv8 [32], which further contributed to the advancements in the YOLO series.

Another approach of YOLO, a more optimized version, is Fast YOLO. It is designed to be faster and more efficient, and can be used in embedded systems and other devices with limited computational resources. It uses a faster CNN model, and other techniques to accelerate the object detection process. YOLO were used to experiment with Pascal VOC dataset and the 17.85 FPS speed of object detection was achieved [33].

At the start of this study, a new YOLO algorithm method, YOLOv9, was developed [34]. It outperforms other models compared to the MS COCO dataset in terms of model size and accuracy [35]. In a comparative study on weed detection in agricultural settings, YOLOv9c achieved a mAP@0.5 of 94.0% on 960px images, outperforming YOLOv10 and RT-DETR models [36]. These results highlight YOLOv9's suitability for real-time applications requiring high precision and speed .

Another optimized YOLO model is YOLOv11, designed to deliver both high accuracy and real-time performance across complex detection tasks. In the power equipment detection domain, YOLOv11 achieved the highest mAP score among five tested models, confirming its robustness and precision [37]. Similarly, in solar panel defect detection, YOLOv11 reached a mAP_0.5 of 93.4%, outperforming YOLOv5 and YOLOv8, while maintaining a fast inference time of 7.7 ms [38]. In a comprehensive study focusing on fruitlet detection in complex orchard environments, YOLOv11-N achieved the best inference time of 2.4 ms, outperforming other YOLO models, underscoring its suitability for real-time object detection applications [39]. These results highlight YOLOv11's ability to balance detection accuracy, model size, and processing speed—making it a promising candidate for high-performance, resource-efficient applications such as defect monitoring and sports analytics.

The YOLOv12, a state-of-the-art object detection model, was published at the end of this study. The model developed to further enhance both accuracy and efficiency by integrating attention mechanisms into its architecture. On the COCO dataset, YOLOv12-L achieved a mAP of 53.7% with an inference latency of 6.77 ms, while the lightweight YOLOv12-N reached 40.6% mAP at just 1.64 ms—outperforming its YOLOv11 counterparts in both speed and accuracy for the same model sizes [40]. In a comprehensive study focusing on fruitlet detection in complex orchard environments, YOLOv12-L achieved the highest recall rate of 0.90 among all tested YOLO configurations, indicating its robustness in detecting objects in challenging real-world scenarios [39]. These results demonstrate YOLOv12's capability to handle real-time object detection in computationally constrained environments as well as complex visual conditions like those found in agricultural and industrial settings.

The literature analysis indicates that the most promising results in object detection are achieved using various versions of the YOLO architecture. YOLO models have consistently demonstrated high accuracy and efficiency across diverse domains and datasets. Given their strong performance, this study will further investigate and compare different YOLO versions to identify the most suitable configuration for the basketball ball and hoop detection task.

Description of YOLOv5

YOLOv5, an advanced variant of the YOLO (You Only Look Once) algorithm, has further refined the architecture and introduced efficient model designs, advanced augmentation techniques, and a simplified implementation. This evolution has led to YOLOv5 gaining significant popularity for its exceptional balance between accuracy and speed, making it a highly suitable choice for real-time object detection tasks across various domains. By capitalizing on these advancements in architecture and incorporating efficient model designs, YOLOv5 exhibits improved performance in terms of both accuracy and processing speed. Its ability to strike a harmonious trade-off between precise object detection and real-time processing positions YOLOv5 as a preferred algorithm for a wide range of real-world applications [41]

Variants of YOLOv5 are different versions or configurations of the YOLOv5 algorithm, each with specific characteristics and improvements. One notable variant is YOLOv5s, which represents the ”small” version of YOLOv5 [42]. We will take a deeper look into this algorithm.

In addition to YOLOv5s, other variants such as YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra-large) are available. These variants differ in their model sizes and complexity, with larger versions offering potentially higher accuracy at the cost of increased computational demands [42].

Description of YOLOv7

YOLOv7 is a real-time object detection algorithm built on convolutional neural networks (CNNs). It uses convolutional layers to extract features from images, then applies bounding box regression and class prediction to localize and classify objects. YOLOv7's strength lies in its speed and real-time processing [31].

In terms of accuracy, YOLOv7 exhibits competitive performance. It achieves a high average precision (AP) score of 56.8% on the challenging MS COCO dataset, demonstrating its ability to accurately detect and classify objects across a wide range of categories. This dataset includes a diverse range of objects, making YOLOv7 adaptable to different scenarios and domains [31].

In addition to its architecture and performance, YOLOv7 introduces innovative techniques to improve accuracy during training. It incorporates a ”trainable bag-of-freebies” approach, which combines various state-of-the-art training strategies and modules to enhance the detection capabilities of the network. This approach addresses challenges in network training and object detection, contributing to the improved accuracy and robustness of YOLOv7 [31].

Description of YOLOv8

The innovative YOLOv8 state-of-the-art (SOTA) model adds new features and enhancements to further increase performance and flexibility while building on the success of earlier YOLO versions. YOLOv8 is a great option for a variety of object detection, recognition and tracking, instance segmentation, image classification, and pose estimation tasks because of its quick, precise, and user-friendly design [43].

YOLOv8 attains high precision. On COCO, for instance, the YOLOv8m model or medium model, achieves a 50.2% mAP. YOLOv8 performed significantly better than YOLOv5 when measured against Roboflow 100, a dataset that is specifically used to assess model performance on different task-specific domains. Unlike other models, which divide tasks into numerous Python files that you can run, YOLOv8 includes a CLI that facilitates easier model training. Moreover, a Python package that offers a smoother coding experience than previous models is included [44].

With YOLOv8, a number of features should be prioritized. These are YOLOv8's significant features [45]:

1. **Better Accuracy**: By implementing novel strategies and optimizations, YOLOv8 enhances object detection accuracy in comparison to its predecessors.
2. **Enhanced Speed**: YOLOv8 maintains high accuracy while reaching faster inference speeds compared to other object detection models.
3. **Multiple Backbones**: YOLOv8 allows users to select the most appropriate model for their particular use case by supporting a number of backbones, including EfficientNet, ResNet, and CSPDarknet.
4. **Adaptive Training**: To improve model performance, YOLOv8 employs adaptive training to balance the loss function and optimize the learning rate during training.
5. **Advanced-Data Augmentation**: To increase the model's resilience and generalizability, YOLOv8 makes use of advanced data augmentation methods like MixUp and CutMix.
6. **Customizable Architecture**: YOLOv8's architecture is highly adjustable, enabling users to quickly and easily change the model's parameters and structure to meet their needs.
7. **Pre-Trained Models**: For convenience and transfer learning across a range of datasets, YOLOv8 offers pre-trained models [45].

Description of YOLOv9

YOLOv9 is a significant advancement in the field of object detection. It introduces two key concepts: Programmable Gradient Information (PGI) and a new lightweight network architecture, the Generalized Efficient Layer Aggregation Network (GELAN) [34].

PGI is designed to address the information bottleneck problem, a common issue in deep networks where input data loses information during the feedforward process. PGI generates reliable gradients through an auxiliary reversible branch, allowing deep features to maintain key characteristics for executing the target task. This ensures that the model can obtain complete input information for the target task, enabling reliable gradient information for network weight updates. GELAN, on the other hand, is a highly efficient and lightweight neural network. It uses conventional convolution to achieve higher parameter utilization than depth-wise convolution designs based on advanced technology. GELAN shows great advantages in terms of being light, fast, and accurate [34].

When combined, PGI and GELAN enhance the object detection performance of YOLOv9. The model outperforms existing real-time object detectors in all aspects. Specifically, compared to its predecessor, YOLOv8, YOLOv9 reduces the number of parameters by 49% and the amount of calculations by 43%, while still achieving a 0.6% AP improvement on the MS COCO dataset. This demonstrates that YOLOv9 is not only more efficient but also more accurate [34].

Description of YOLOv11

YOLOv11 is an evolution of the YOLO object detection framework, emphasizing improvements in both detection accuracy and computational efficiency. The model incorporates several novel components, including the C3k2 block, SPPF (Spatial Pyramid Pooling - Fast), and C2PSA (Convolutional block with Parallel Spatial Attention), which collectively contribute to enhanced feature extraction and computational efficiency [46].

The C3k2 block is a modified version of the Cross Stage Partial (CSP) architecture, utilizing a kernel size of 2 to balance depth and computational load. SPPF accelerates the pooling process, enabling faster multi-scale feature aggregation. C2PSA introduces parallel spatial attention mechanisms, allowing the model to focus on relevant features more effectively. This enables YOLOv11 to maintain high performance while remaining suitable for deployment on resource-constrained devices [46].

These architectural innovations enable YOLOv11 to perform a variety of computer vision tasks beyond object detection, including instance segmentation, pose estimation, and oriented object detection (OBB). The model is released in multiple variants - ranging from YOLOv11n to YOLOv11x - each tailored for different performance and resource requirements. The model demonstrates improved mean Average Precision (mAP) and reduced parameter count compared to its predecessors. For instance, YOLOv11m achieves a mean Average Precision (mAP) of 51.5% on the COCO dataset while using 22% fewer parameters than YOLOv8m, making it computationally efficient without compromising accuracy [47].

Empirical results show that YOLOv11 also achieves improved performance over YOLOv10 on standard benchmarks such as MS COCO. It provides higher average precision while maintaining low latency and efficient resource usage. These enhancements make YOLOv11 well-suited for a wide range of real-time object detection tasks, from cloud-based processing to mobile deployment [47].

Description of YOLOv12

YOLOv12 represents a significant advancement in real-time object detection by integrating attention mechanisms into the YOLO framework without compromising speed. This model introduces two primary innovations: the Area Attention mechanism and the Residual Efficient Layer Aggregation Network (R-ELAN) [40].

The Area Attention mechanism is designed to efficiently capture long-range dependencies by dividing feature maps into equal-sized regions along horizontal or vertical axes. This approach simplifies the computation compared to traditional self-attention methods, enabling the model to maintain a large receptive field with reduced computational overhead [40, 48].

Complementing this, R-ELAN enhances feature aggregation by introducing block-level residual connections and a bottleneck-like structure. These modifications address optimization challenges, particularly in larger-scale attention-centric models, leading to improved learning dynamics and performance [40, 48].

Similar to its predecessor, YOLOv12 is available in five scaled variants - YOLOv12-N, S, M, L, and X - designed to balance accuracy and computational efficiency across diverse deployment scenarios. Beyond object detection, these variants support a wide range of vision tasks, including instance segmentation, pose estimation, and oriented object detection (OBB). In the context of object detection, YOLOv12-N achieves a mean Average Precision (mAP) of 40.6% with an inference latency of 1.64 ms on a T4 GPU, outperforming YOLOv10-N and YOLOv11-N by 2.1% and 1.2% mAP, respectively, at equivalent speeds [40].

Overall, the introduction of Area Attention and the Residual ELAN backbone allows YOLOv12 to achieve strong detection performance while maintaining real-time efficiency. These enhancements contribute to improved accuracy, faster inference, and better generalization across object scales, as demonstrated by empirical results on standard benchmarks such as MS COCO [40, 48].

2. Preliminary System of Shot Recognition

This research is a follow-up to a previous study that used computer vision algorithms to identify and detect a basketball ball and board [49]. In this paper, we aim to achieve the same goal - detecting the ball and the basketball board, but using a different approach based on machine learning algorithms. Ultimately, the two models will be integrated to develop a shot recognition system. A basic diagram of the final algorithm, which will be further investigated as a result of this study, is shown in the Figure 2.

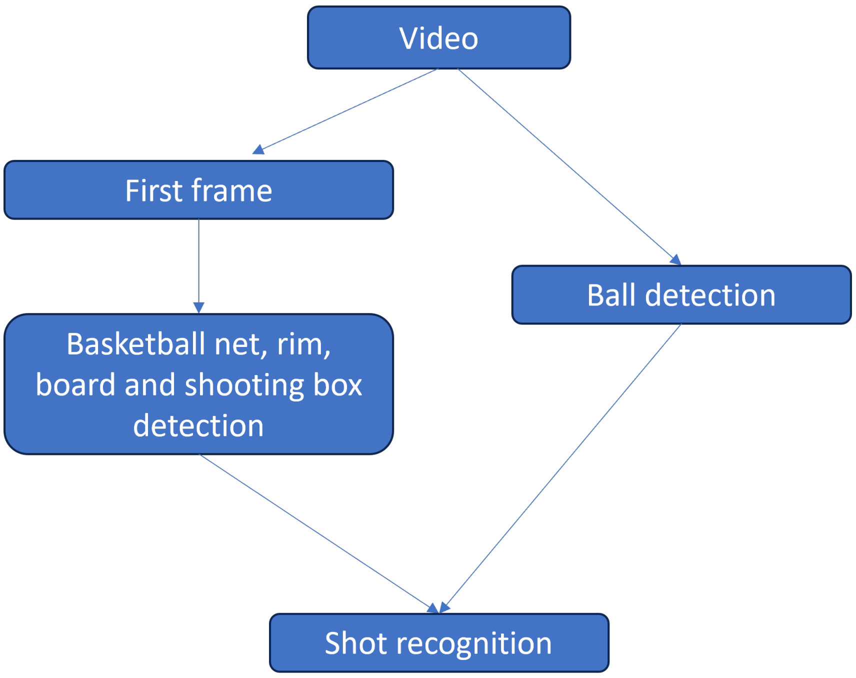


Figure 2. Preliminary plan of the final algorithm

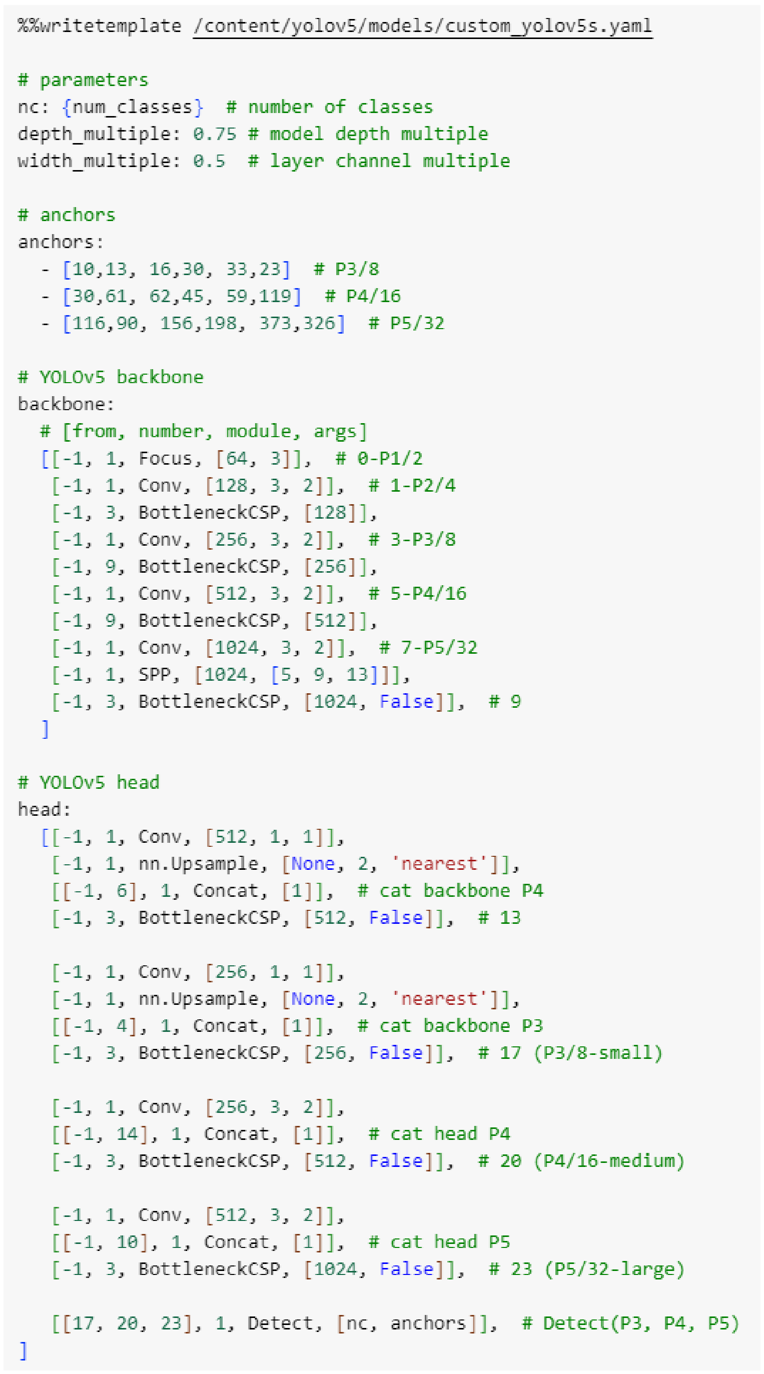
Initially, a video file of the basketball shot is captured. From the first frame of the video, the algorithm identifies key elements such as the basketball net, rim, backboard, and shooting box. Simultaneously, throughout the entire frames of the video, the algorithm identifies the moving ball. All components will be combined into a unified system, which, with the addition of custom logic, will be capable of determining whether a shot was made or missed.

3. Experiments With YOLO Algorithms

For ball recognition and detection, five YOLO versions were selected: YOLOv5, YOLOv7, YOLOv8, YOLOv11, and YOLOv12. For the detection of the basketball board, net, rim, and shooting box, another set of five YOLO models was used: YOLOv5, YOLOv7, YOLOv9, YOLOv11, and YOLOv12.

3.1. Architecture of YOLOv5

For this work, the architecture of YOLOv5s created by Ultralytics was taken from Roboflow template [43] (see Figure 3).



```
%%writetemplate /content/yolov5/models/custom_yolov5s.yaml

# parameters
nc: {num_classes}  # number of classes
depth_multiple: 0.75 # model depth multiple
width_multiple: 0.5  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Focus, [64, 3]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],  # 9
  ]

# YOLOv5 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, BottleneckCSP, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, BottleneckCSP, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, BottleneckCSP, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, BottleneckCSP, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
]
```

Figure 3. YOLOv5s architecture [42]

Firstly, we set the number of classes and the model‘s depth and width. Depth - is a multiple that determines the number of layers and the width - number of channels in the layers. These parameters allow for flexibility in adjusting the model’s complexity and resource requirements based on the specific needs of the application or dataset.

The architecture consists of three main components: anchors, backbone, and head [42]:

- **Anchors**: These are predefined bounding box sizes that are used for object detection. YOLOv5s has three sets of anchors, each associated with a specific feature map level (P3, P4, P5) in the model.
- **Backbone**: This part of the architecture extracts features from the input image. It consists of several convolutional layers, including Focus, Conv, and BottleneckCSP modules. These layers process the input image at different resolutions to capture hierarchical features.
- **Head**: The head of the model takes the features from the backbone and performs further processing. It includes additional Conv, Upsample, Concat, and BottleneckCSP modules. The head combines features from different levels (P3, P4) of the backbone using concatenation. It also performs upsampling and additional convolution operations. Finally, it utilizes the Detect module to generate the final detection output, which includes bounding box coordinates and class predictions.

3.2. Architecture of YOLOv7

The architecture of YOLOv7 algorithm looks almost the same as YOLOv5 but has many more parts in backbone and head. The algorithm shares common components, including anchors, backbone, and head. However, it is intentionally designed to be wider, deeper, and overall more complex in order to enhance its performance [50] (see Figure 4).

3.3. Architecture of YOLOv8

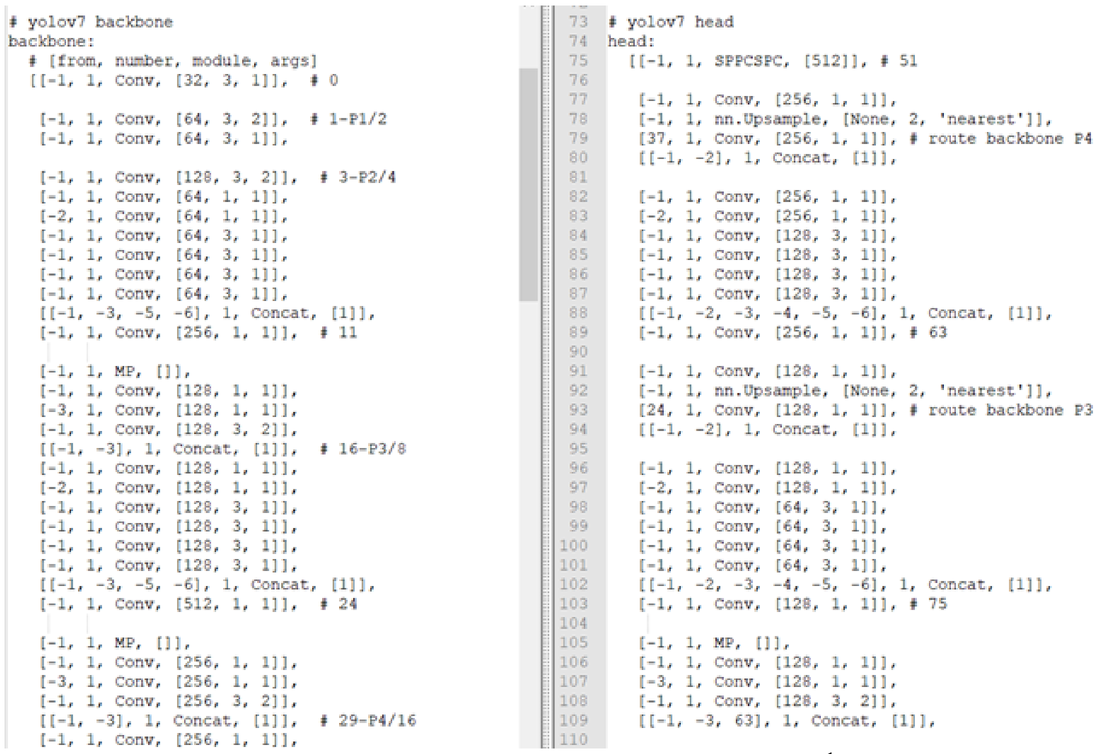
The algorithm shares common components, including backbone and head. However, it is intentionally designed to be wider, deeper, and overall more complex in order to enhance its performance.

For the YOLOv8, the two primary components of the convolutional neural network are the backbone and the head [45]:

- **The backbone** of YOLOv8 is a redesigned version of the CSPDarknet53 architecture. To enhance information flow between the various layers, this design, which has 53 convolutional layers, uses cross-stage partial connections.
- **The head** of YOLOv8 consists of numerous convolutional layers followed by a succession of fully linked layers. The bounding boxes, objectness scores, and class probabilities for the objects found in an image are predicted by these layers [45].

The implementation of a self-attention mechanism in the network’s head is one of YOLOv8’s primary features. Through this technique, the model is able to focus on different areas of the

[1]Source: https://github.com/WongKinYiu/yolov7



```
# yolov7 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [32, 3, 1]],  # 0

   [-1, 1, Conv, [64, 3, 2]],  # 1-P1/2
   [-1, 1, Conv, [64, 3, 1]],

   [-1, 1, Conv, [128, 3, 2]],  # 3-P2/4
   [-1, 1, Conv, [64, 1, 1]],
   [-2, 1, Conv, [64, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]],  # 11

   [-1, 1, MP, []],
   [-1, 1, Conv, [128, 1, 1]],
   [-3, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 16-P3/8
   [-1, 1, Conv, [128, 1, 1]],
   [-2, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -3, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [512, 1, 1]],  # 24

   [-1, 1, MP, []],
   [-1, 1, Conv, [256, 1, 1]],
   [-3, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, -3], 1, Concat, [1]],  # 29-P4/16
   [-1, 1, Conv, [256, 1, 1]],
```

```
# yolov7 head
head:
  [[-1, 1, SPPCSPC, [512]], # 51

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [37, 1, Conv, [256, 1, 1]], # route backbone P4
   [[-1, -2], 1, Concat, [1]],

   [-1, 1, Conv, [256, 1, 1]],
   [-2, 1, Conv, [256, 1, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [-1, 1, Conv, [128, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [256, 1, 1]], # 63

   [-1, 1, Conv, [128, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [24, 1, Conv, [128, 1, 1]], # route backbone P3
   [[-1, -2], 1, Concat, [1]],

   [-1, 1, Conv, [128, 1, 1]],
   [-2, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [-1, 1, Conv, [64, 3, 1]],
   [[-1, -2, -3, -4, -5, -6], 1, Concat, [1]],
   [-1, 1, Conv, [128, 1, 1]], # 75

   [-1, 1, MP, []],
   [-1, 1, Conv, [128, 1, 1]],
   [-3, 1, Conv, [128, 1, 1]],
   [-1, 1, Conv, [128, 3, 2]],
   [[-1, -3, 63], 1, Concat, [1]],
```

Figure 4. The part of the YOLOv7 architecture[1]

image and modify the weighting of certain features according to how relevant they are to the job at hand [45].

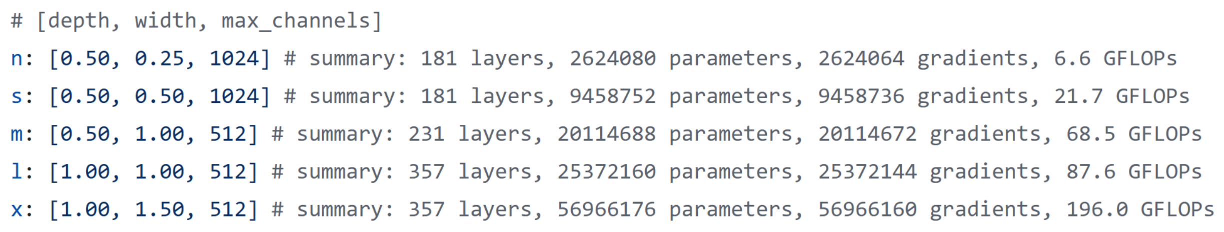
3.4. Architecture of YOLOv9

The architecture of the YOLOv9 is similar to older versions. Common elements of the algorithm are the backbone and head [35].

- **The backbone** of YOLOv9 is built upon the Generalized Efficient Layer Aggregation Network (GELAN), a streamlined and highly efficient architecture designed for rich feature extraction with minimal computational overhead. GELAN utilizes conventional convolution layers rather than depthwise separable convolutions (used in YOLOv8), which improves both the representational capacity and parameter utilization. The architecture aggregates features across different layers in a hierarchical manner, improving gradient flow and maintaining spatial detail. GELAN achieves a balance between depth and efficiency, enabling strong performance even in lightweight model variants [34] .
- **The head** of YOLOv8 OLOv9 retains the decoupled detection head design first introduced in YOLOv8. This head separates the classification and regression tasks into parallel branches, allowing the network to specialize in both object localization and class prediction. The decoupling reduces conflict between tasks and leads to improved detection accuracy. YOLOv9 also incorporates enhancements in label assignment and loss computation strategies to further refine output predictions. Importantly, the head remains anchor-free, continuing the trend toward simplified and generalizable object detectors [34].

3.5. Architecture of YOLOv11

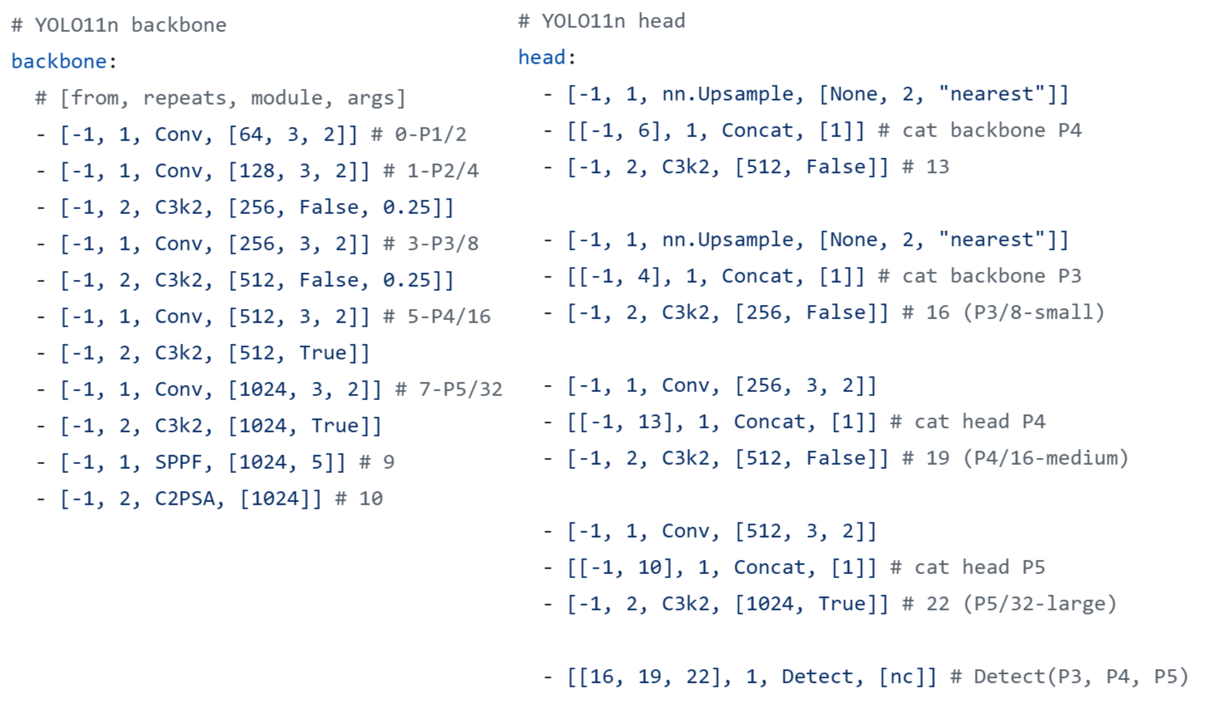
The YOLOv11 is available in five scaled versions: N, S, M, L, and X. The primary differences among these variants lie in the maximum number of channels, as well as the model's depth and width (see Figure 5).



```
# [depth, width, max_channels]
n: [0.50, 0.25, 1024] # summary: 181 layers, 2624080 parameters, 2624064 gradients, 6.6 GFLOPs
s: [0.50, 0.50, 1024] # summary: 181 layers, 9458752 parameters, 9458736 gradients, 21.7 GFLOPs
m: [0.50, 1.00, 512] # summary: 231 layers, 20114688 parameters, 20114672 gradients, 68.5 GFLOPs
l: [1.00, 1.00, 512] # summary: 357 layers, 25372160 parameters, 25372144 gradients, 87.6 GFLOPs
x: [1.00, 1.50, 512] # summary: 357 layers, 56966176 parameters, 56966160 gradients, 196.0 GFLOPs
```

Figure 5. The versions of YOLOv11 model [47]

YOLOv11 adopts an anchor-free architecture and is structured around two primary components: the backbone and the detection head (see Figure 6).



```
# YOLO11n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 2, C3k2, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 2, C3k2, [1024, True]]
  - [-1, 1, SPPF, [1024, 5]] # 9
  - [-1, 2, C2PSA, [1024]] # 10
```

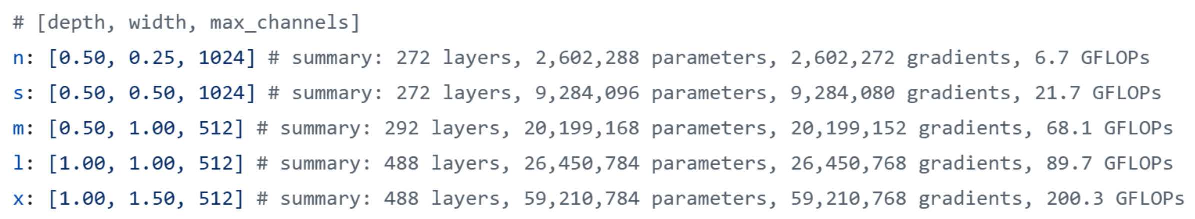
```
# YOLO11n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 2, C3k2, [512, False]] # 13

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 2, C3k2, [256, False]] # 16 (P3/8-small)

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 13], 1, Concat, [1]] # cat head P4
  - [-1, 2, C3k2, [512, False]] # 19 (P4/16-medium)

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 10], 1, Concat, [1]] # cat head P5
  - [-1, 2, C3k2, [1024, True]] # 22 (P5/32-large)

  - [[16, 19, 22], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

Figure 6. The architecture of YOLOv11 model N version [47]

The YOLOv11 backbone consists of a series of convolutional and C3k2 blocks for progressive feature extraction, followed by an SPPF module for multi-scale context aggregation. The final stage includes a C2PSA module, which applies parallel spatial attention to the high-level features, allowing the network to enhance focus on informative regions of the input image [46, 47].

3.6. Architecture of YOLOv12

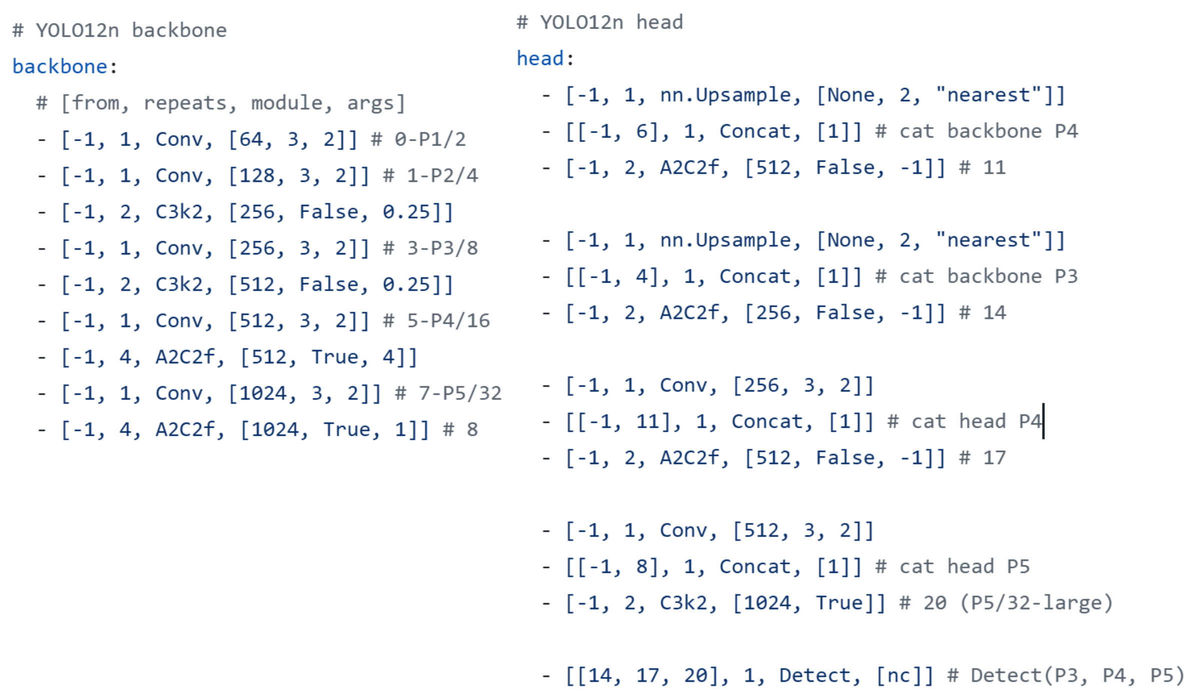
The YOLOv12, like YOLOv11, is available in five scaled versions: N, S, M, L, and X. The primary differences among these variants lie in the maximum number of channels, as well as the model's depth and width (see Figure 7).



```
# [depth, width, max_channels]
n: [0.50, 0.25, 1024] # summary: 272 layers, 2,602,288 parameters, 2,602,272 gradients, 6.7 GFLOPs
s: [0.50, 0.50, 1024] # summary: 272 layers, 9,284,096 parameters, 9,284,080 gradients, 21.7 GFLOPs
m: [0.50, 1.00, 512] # summary: 292 layers, 20,199,168 parameters, 20,199,152 gradients, 68.1 GFLOPs
l: [1.00, 1.00, 512] # summary: 488 layers, 26,450,784 parameters, 26,450,768 gradients, 89.7 GFLOPs
x: [1.00, 1.50, 512] # summary: 488 layers, 59,210,784 parameters, 59,210,768 gradients, 200.3 GFLOPs
```

Figure 7. The versions of YOLOv12 model [47]

YOLOv12 adopts an anchor-free architecture and is structured around two primary components: the backbone and the detection head (see Figure 8).



```
# YOLO12n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 4, A2C2f, [512, True, 4]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 4, A2C2f, [1024, True, 1]] # 8
```

```
# YOLO12n head
head:
  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 6], 1, Concat, [1]] # cat backbone P4
  - [-1, 2, A2C2f, [512, False, -1]] # 11

  - [-1, 1, nn.Upsample, [None, 2, "nearest"]]
  - [[-1, 4], 1, Concat, [1]] # cat backbone P3
  - [-1, 2, A2C2f, [256, False, -1]] # 14

  - [-1, 1, Conv, [256, 3, 2]]
  - [[-1, 11], 1, Concat, [1]] # cat head P4
  - [-1, 2, A2C2f, [512, False, -1]] # 17

  - [-1, 1, Conv, [512, 3, 2]]
  - [[-1, 8], 1, Concat, [1]] # cat head P5
  - [-1, 2, C3k2, [1024, True]] # 20 (P5/32-large)

  - [[14, 17, 20], 1, Detect, [nc]] # Detect(P3, P4, P5)
```

Figure 8. The architecture of YOLOv12 model N version [47]

The YOLOv12 backbone combines standard convolutional and C3k2 blocks with the newly introduced A2C2f modules to enable progressive feature extraction enriched with spatial attention. While the early stages use lightweight convolutions and C3k2 for efficient processing, the deeper stages (P4 and P5) are equipped with stacked A2C2f blocks, which enhance context modeling through cross-channel and spatial interaction. This design allows the backbone to effectively capture both local and long-range dependencies in the input features [40, 51].

The head follows a multi-scale approach by fusing features across three resolution levels (P3, P4, P5). A2C2f blocks are also employed in the head to refine features after upsampling and concatenation, strengthening the model's ability to localize and classify objects across varied scales. A final C3k2 block is applied at the largest scale (P5) before detection, maintaining a balance between attention mechanisms and computational efficiency [40, 51]

3.7. Basketball Shot Data Collection

For the experiments, self-recorded basketball shots were captured. These shots were taken with different levels of accuracy (0, 1, 2, 3 out of 3) in a single video using two different mobile devices: the Samsung Galaxy J5 and the Samsung Galaxy A52s. In total, 153 video recordings of 30 FPS were made with these specifications. In Figure 9 we can see the positions of the camera.

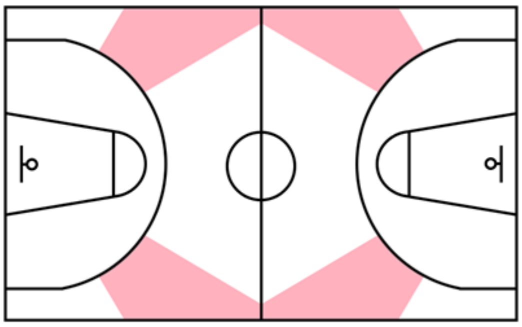


Figure 9. Positions of the camera in the basketball court

The static camera position (marked in red) was set at approximately 45 degrees, with a variation of approximately 10 degrees, relative to the basketball's location. A different spot was chosen for each video, ensuring variations in the shooting perspective.

For the experiments involving different YOLO algorithms, the first frame of each video was selected. An example of these pictures is shown in Figure 10.

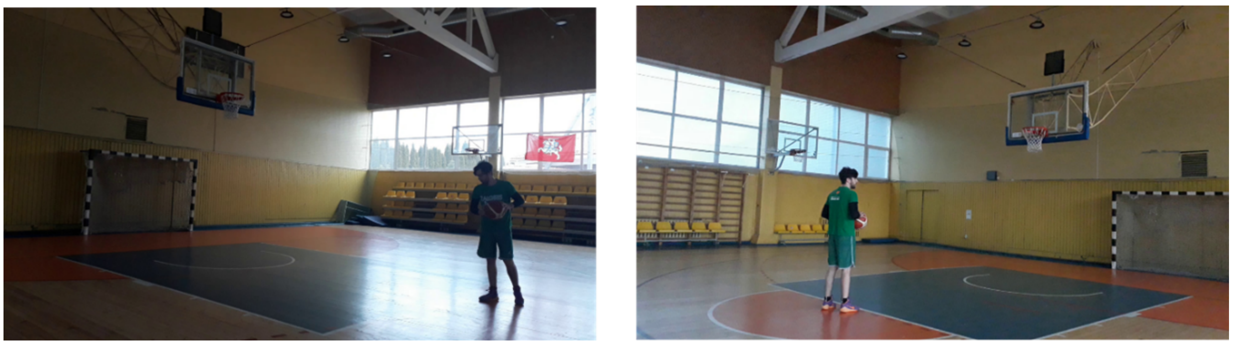


Figure 10. Examples of the dataset

3.7.1. Labels for Training

The dataset includes 161 pictures with a clearly visible basketball. The labeling process involved marking the basketball board, rim, net, and shooting box. We used the Roboflow tool to label the dataset, and the example of the labeled data on the website shown in Figure 11. The full dataset is publicly available via url: https://universe.roboflow.com/ktu-bo8mo/magister-yhpnv.

The labeling for the ball detection task was also made using Roboflow. It was simpler because the labeled data only contained visible or hidden ball (see Figure 12). The full (1000 images) dataset is publicly available via url:

https://universe.roboflow.com/ktu-magister/basketball-ball-segments-from-shooting-videos

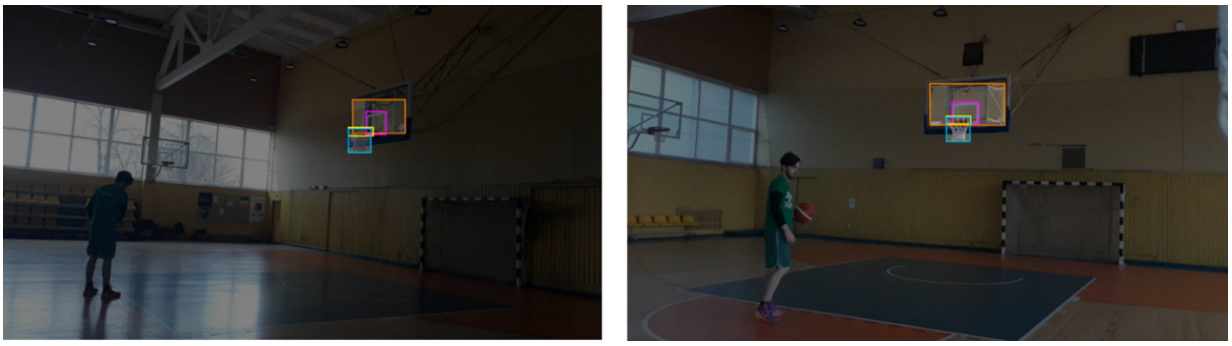


Figure 11. Example of labeled data for board detection

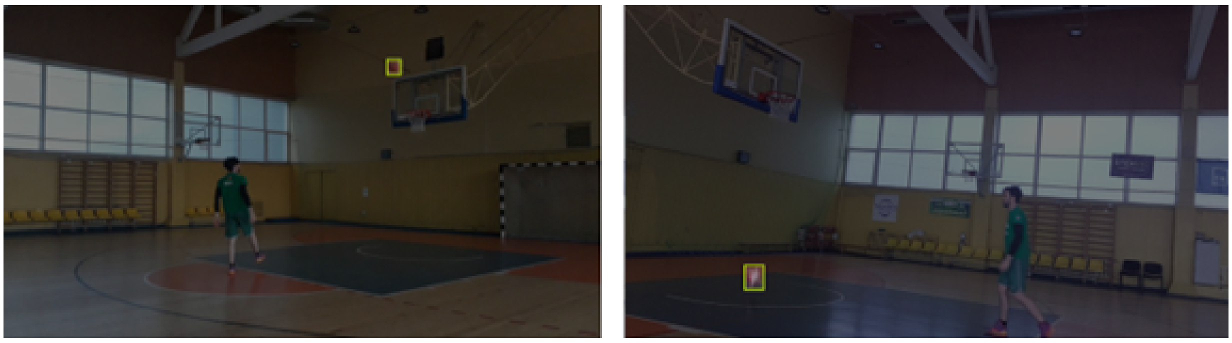


Figure 12. Example of labeled data for ball detection

3.8. Model Performance Environment

The experimental investigations with YOLO models were conducted within the Google Colab environment, leveraging Python as the primary programming language. The models were instantiated either from existing libraries or directly from the source code. Each experimental iteration was executed utilizing a virtual Graphics Processing Unit (GPU) to expedite computations and enhance model training efficiency.

3.8.1. Evaluation Metrics

To obtain the results of the board and ball recognition evaluation, four main indicators of the recognition problem were selected. These indicators were calculated based on a validation set consisting of 640x640 images to detect the board and a different size image to detect the ball. In addition, three aspects of the model itself were assessed:

- **Precision**: This metric measures the accuracy of the model's predictions by calculating the proportion of correctly predicted positive samples among all samples classified as positive.
- **Recall**: Also known as sensitivity, recall quantifies the model's ability to correctly identify positive samples by calculating the proportion of correctly predicted positive samples among all actual positive samples.
- **mAP_0.5**: The mean Average Precision at an IoU threshold of 0.5 assesses the model's performance in terms of localization accuracy and object detection at a specific intersection-over-union (IoU) threshold.
- **mAP_0.5:0.95**: The mean Average Precision across a range of IoU thresholds from 0.5 to 0.95 provides a more comprehensive evaluation of the model's detection accuracy across a spectrum of IoU values.

- **Inference Time**: This parameter measures the duration required to perform a detection task for one image.
- **Parameters**: This refers to the quantity of parameters in the YOLO model, which affects its complexity.
- **Size**: This indicates the size of the trained YOLO model in MB, which impacts storage requirements and deployment feasibility.

It is important to note that YOLOv5, YOLOv9, and YOLOv12 models were trained from scratch, while YOLOv7 and YOLOv11 were initialized with pre-trained weights. For YOLOv8, both training approaches were utilized. Also, for YOLOv12, both validation and detection were performed using the same image resolution during training.

3.8.2. Experiment Scenarios

The experiments involved exploring various scenarios, where specific aspects of the YOLO algorithms were modified and evaluated.

- **Image Size**: It was changed to observe the influence of different dimensions of input images on detection accuracy and speed.
- **Batch Size**: The number of images processed simultaneously was adjusted to assess its impact on training efficiency and model performance.
- **Model Depth**: Variations in the number of layers in the YOLOv5 model were introduced to observe changes in detection accuracy, model complexity, and computational requirements. For other YOLO algorithms the default of 1 were selected.
- **Model Width**: The number of channels within the layers was manipulated to investigate its effect on detection performance and resource utilization. For other YOLO algorithms the default of 1 were selected.
- **Epochs**: The model was trained for a specified number of iterations to evaluate how the YOLO algorithm evolves and improves over time.
- **Optimizer**: Modified for YOLOv8 algorithm. It adjusts a neural network's parameters to minimize the prediction error using algorithms like SGD, Adam or AdamW.
- **Initial Learning Rate**: Modified for YOLOv8 algorithm. It is a key hyperparameter dictating step size during optimization, affecting convergence speed; finding an optimal rate is crucial for effective model training.

4. Results of YOLO Models

In this section, we discuss the results of the different YOLO algorithms applied to the ball and board detection task.

4.1. Results of Ball Detection

In this section, we present the results of our experiments using different YOLO algorithms on the before mentioned dataset for the ball detection task.

4.1.1. Results of Initial Experiments

Among the various experimental scenarios conducted, the optimal results are summarized in Table 1 for the YOLOv5, YOLOv7, YOLOv8, YOLOv11, and YOLOv12 models. The model architectures were configured with default depth and width parameters (both set to 1), and the input image size was fixed at 640 pixels.

Table 1. Results of ball detection for each YOLO model

Model	Size	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
YOLOv5	95.3MB	140.3ms	0.967	0.894	0.956	0.725
YOLOv7	74.8MB	51.2ms	0.989	0.899	0.949	0.729
YOLOv8	6.2MB	26.8ms	0.953	0.889	0.95	0.666
YOLOv11	51.2MB	25.4ms	0.988	0.920	0.961	0.732
YOLOv12	53.7MB	28.4ms	0.969	0.905	0.956	0.679

The YOLOv5 model achieved its best performance with a batch size of 22 and 150 training epochs. Similarly, YOLOv7 performed well under the same batch size, requiring 60 epochs. The best configuration for YOLOv8 was a batch size of 32 and 50 training epochs. For YOLOv11, optimal performance was obtained with a batch size of 24 and 25 epochs, while YOLOv12 showed optimal results using a batch size of 8 and 50 epochs.

The YOLOv8 and YOLOv12 models rank among the top three in terms of lowest inference time, though this comes at the cost of reduced precision. Other three models, YOLOv5, YOLOv7 and YOLOv11 demonstrate similar performance in terms of mAP_0.5:0.95. However, YOLOv11 performs better in terms of detection speed. Considering computational efficiency and training time, further investigation will be conducted using the YOLOv5, YOLOv11 and YOLOv12 models.

4.1.2. Results of YOLOv5 Configuration

In the initial phase of our study, we conducted experiments to ascertain the optimal image shape for training purposes: square or rectangular format would yield better results. The outcomes, depicted in Figure 13, guided our subsequent decisions. For consistency, we maintained a fixed depth and width of 1 for all experimental configurations. Additionally, we adjusted the batch size and number of training epochs to accommodate the limitations of the Google Colab environment.

There is no discernible trend or clear dependence on the different image formats. However, based

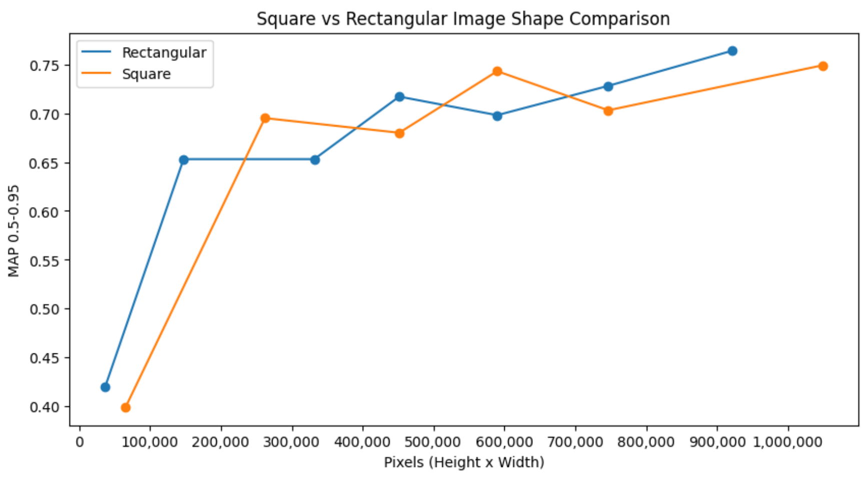


Figure 13. Square vs rectangular image size comparison

on the results, we decided to go for a square format of 768x768 pixels. This choice corresponds to the second best result and falls roughly in the middle in terms of pixel count.

In the second phase of our research, we conducted experiments using the YOLOv5 architecture. We systematically varied the depth and width of the model within the range of 0.25 to 1.25, with increments of 0.25. The outcomes of these experiments are presented in Figure 14. The colour represents a model where the first number is the depth and the second the width.

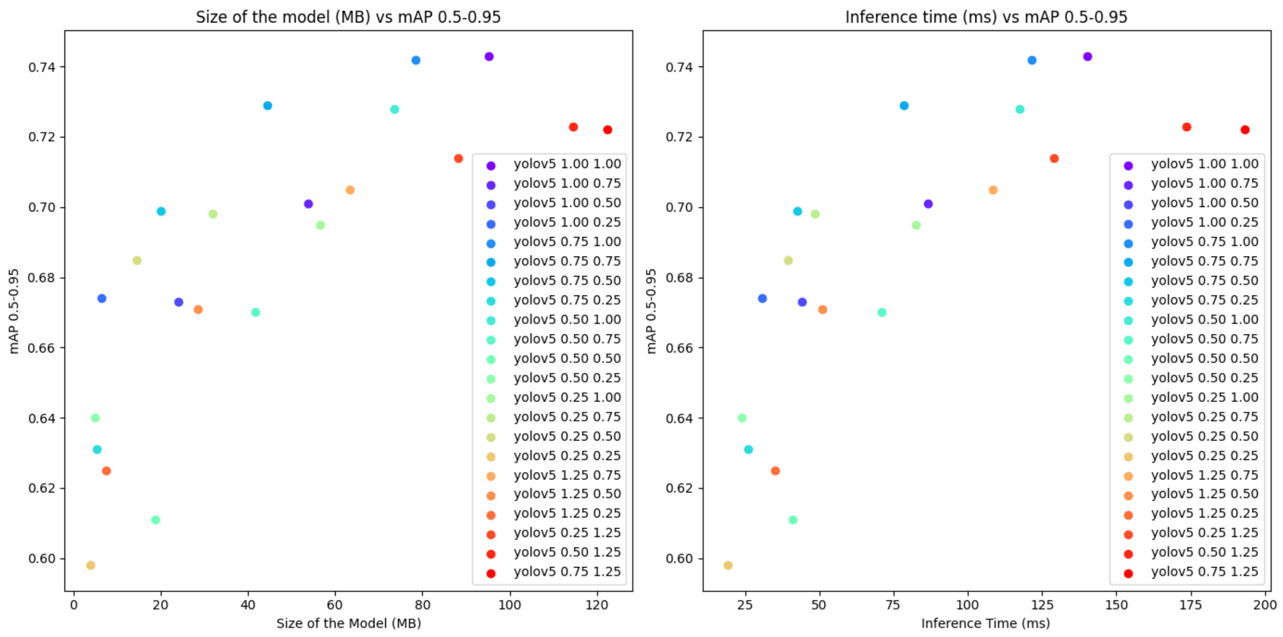


Figure 14. Relationship between model size and inference time for mAP_0.5:0.95 Score of ball detection

The results provide a clear insight: as the model size increases, so does its inference time. Additionally, an evident trend curve emerges from the data. Based on these findings, we can identify a YOLOv5 configuration that approaches optimality: a depth of 0.75 and a width of 0.5. This configuration achieved satisfactory results, with an mAP_0.5:0.95 score of 0.699, while maintaining a swift detection rate of only 42.6 milliseconds. The prediction example of this model is shown in Figure 15.

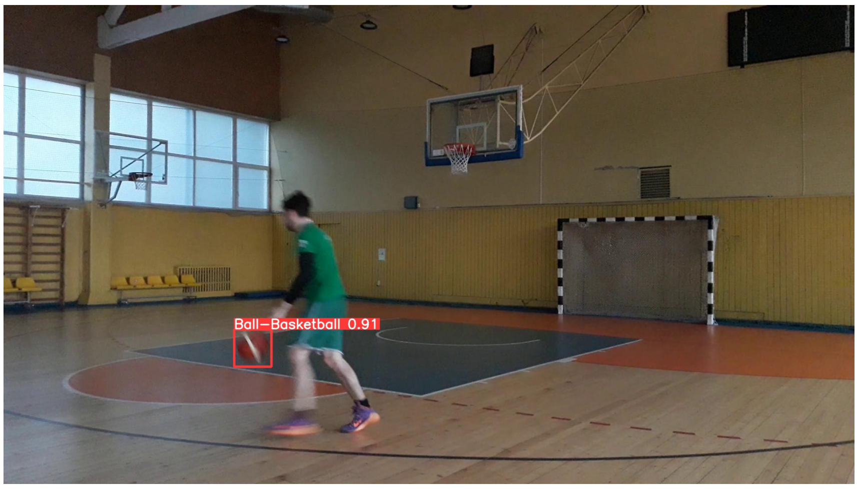


Figure 15. Prediction eesults of YOLOv5 ball detection

While the results demonstrate accurate predictions at high confidence scores, some images still lack ball detection despite the ball being visible.

4.1.3. Results of YOLOv11 Configuration

Experiments with the YOLOv11 model were conducted using its 3 smallest configurations: N, S, and M, each tested with different hyperparameter settings. The top two experiments for each configuration are presented in Table 2. The size of YOLOv11-N is 5.5 MB, YOLOv11-S is 19.2 MB, and YOLOv11-M is 40.5 MB. The corresponding number of parameters is 2.6 million for YOLOv11-N, 9.4 million for YOLOv11-S, and 20.1 million for YOLOv11-M. The number of training epochs was fixed at 25.

Table 2. Results of YOLOv11 ball detection for validation set with hyper-parameters

YOLOv11	Image Size	Batch	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
N	768	32	9.1ms	0.993	0.945	0.976	0.749
N	704	32	8.8ms	0.979	0.943	0.971	0.740
S	800	24	14.2ms	0.984	0.945	0.974	0.761
S	768	24	12.0ms	0.984	0.918	0.964	0.751
M	800	16	32.2ms	0.964	0.94	0.966	0.740
M	704	20	30.4ms	0.964	0.939	0.965	0.752

The most promising results were obtained with the smallest YOLOv11 model, version N, which demonstrated the lowest inference time (8.8-9.1 ms) and the smallest model size (5.5 MB) among all tested configurations. Despite its compact size, it achieved evaluation metrics comparable to those of larger models. A mAP_0.5:0.95 score of 0.749 was achieved with an image size of 768 and a batch size of 32. A visual representation of these results is provided in Figure 16.

The results show that the YOLOv11 model converges rapidly, with all key metrics stabilising after about 20 epochs. It achieves relatively high precision, recall, and mAP scores, with training taking approximately 9 minutes - likely due to the use of pre-trained weights.

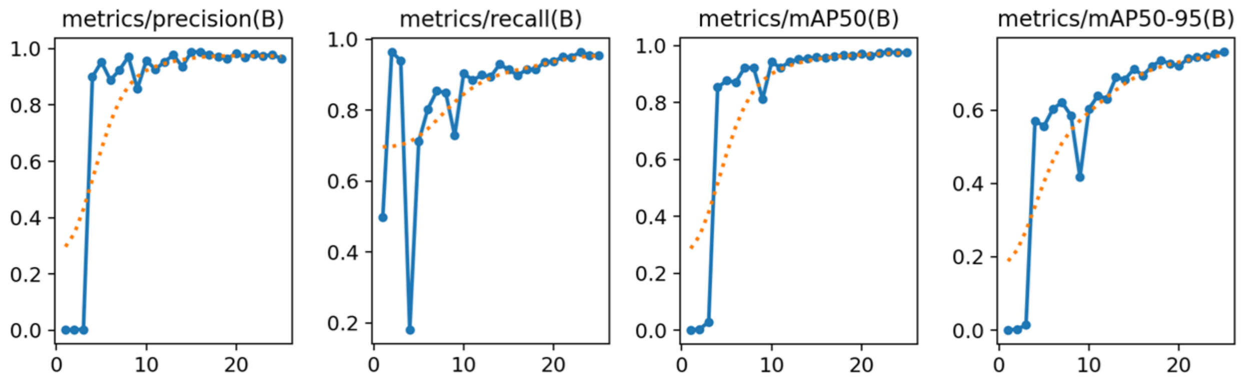


Figure 16. The results of the best YOLOv11 configuration (N)

4.1.4. Results of YOLOv12 Configuration

Experiments with the YOLOv12 model were conducted using its 3 smallest configurations: N, S, and M, each tested with different hyperparameter settings. The top two experiments for each configuration are presented in Table 3. The size of YOLOv12-N is 5.4 MB, YOLOv12-S is 18.6 MB, and YOLOv12-M is 39.7 MB. The corresponding number of parameters is 2.5 million for YOLOv12-N, 9.1 million for YOLOv12-S, and 19.6 million for YOLOv12-M. The number of training epochs ranged from 50 to 60. The detection inference time was calculated using the same image size as during training.

Table 3. Results of YOLOv12 ball detection for validation set with hyper-parameters

YOLOv12	Image Size	Batch	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
N	800	16	12.5ms	0.931	0.905	0.932	0.659
N	768	20	12.6ms	0.964	0.869	0.938	0.665
S	800	6	16.5ms	0.979	0.927	0.963	0.715
S	768	8	14.0ms	0.970	0.915	0.961	0.705
M	800	6	28.7ms	0.958	0.92	0.963	0.727
M	768	6	27.6ms	0.984	0.917	0.961	0.713

The highest mAP_0.5:0.95 score of 0.727 was achieved using the YOLOv12 M version, which is also the largest model among those tested (39.7 MB) and exhibits the highest inference time at 28.7 ms. These results were obtained using an input image size of 800 and a batch size of 6. A visual representation of these findings is provided in Figure 17.

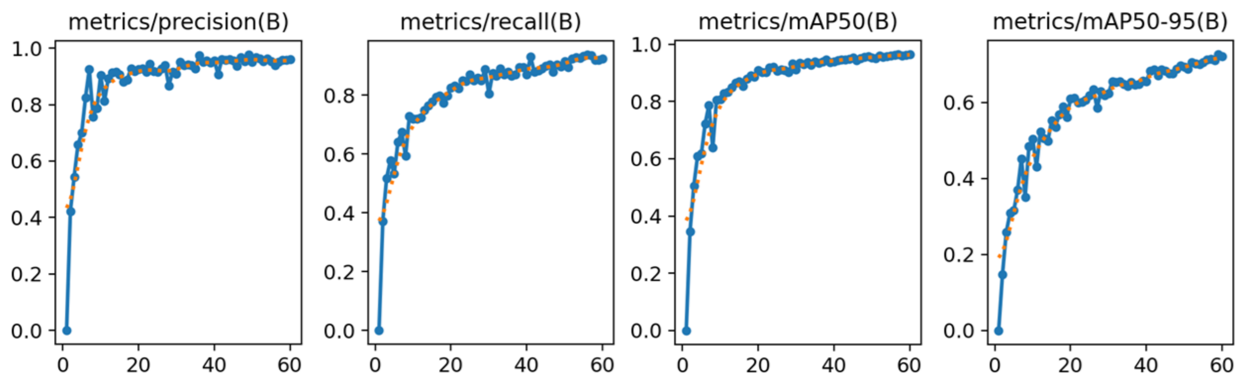


Figure 17. The results of the best YOLOv12 configuration (M)

The results show that the YOLOv12 model converges rapidly, with all key metrics stabilising after about 40 epochs. It achieves relatively high precision, recall, and mAP scores, with training taking approximately 1.5 hours - likely due to the model being trained from scratch.

4.2. Final Results of Ball Detection

Optimal configurations for each model were selected and are shown in Figure 18. For each model, a mAP_0.5:0.95 result and inference time of 768x768 image were selected.

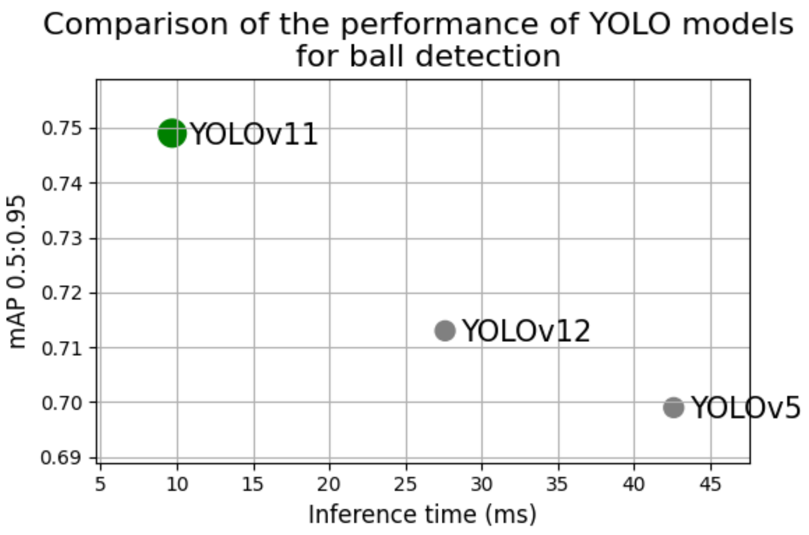


Figure 18. Comparison of the performance of YOLO models for ball detection

The comparison of the three YOLO models shows that they differ in performance, in particular in mAP_0.5:0.95 score and inference time. Although it is often assumed that these two indicators are inversely proportional, in this case such assumptions may not be entirely appropriate due to the fundamental architectural differences between the models.

YOLOv11 stands out for achieving a high mAP_0.5:0.95 score and, at the same time, the lowest inference time. In contrast, YOLOv12, with a longer inference time of, achieves a slightly lower precision. YOLOv5, although the slowest model, achieves the worst precision results. These results suggest that YOLOv11 provides the best balance between speed and precision, making it the most suitable candidate for real-time ball detection task.

For future investigations, we plan to use the YOLOv11-N model, which achieved a mAP_0.5:0.95 score of 0.749 and an inference time of 9.1 ms, with a model size of only 5.5 MB and 2.6 million parameters. These results were obtained using an image size of 768, a batch size of 32, and 25 training epochs. The prediction example of this model is shown in Figure 19.

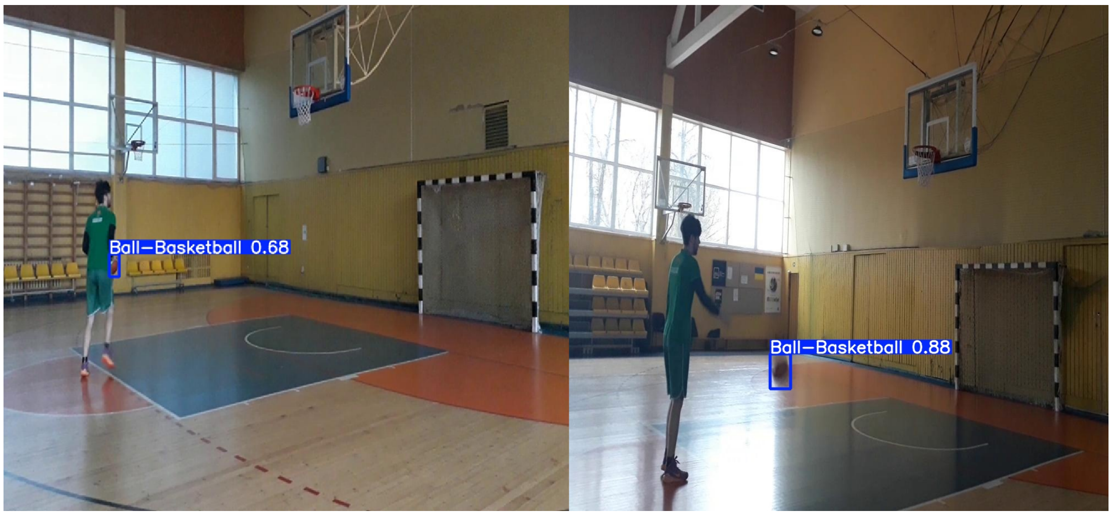


Figure 19. Prediction results of YOLOv11-N board detection

The images demonstrate that the ball is detected with high confidence, even when it is slightly blurred. Notably, the model was also able to detect the ball when it was partially covered by the shooter, but with a lower confidence score.

4.3. Results of Board Detection

In this section, we present the results of our experiments using different YOLOv5, YOLOv7, YOLOv9, YOLOv11, and YOLOv12 algorithms on the before mentioned dataset for the board detection task.

4.3.1. Results of YOLOv5

The best results of the experiments with the YOLOv5 model and hyperparameters for each image size are shown in table 4. The number of epochs was set to 200 and the batch size to 32. If the training reached the limits of Google Colab, the batch size would be reduced until it passed.

The results are as expected, the higher the training image size, the higher the precision, although the difference is not high. The other metrics have a similar value, except the mAP_0.5:0.95 score, which is the highest when the training image size was the highest - 640x640. The visual results of the model configuration that reached the highest score of mAP_0.5:0.95 are shown in Figure 20.

Obviously, 200 epochs are too many, as most of the model's results start to converge at around 120 epochs, although it should be noted that the result for mAP_0.5:0.95 is still increasing, but there is a noticeable decrease in tempo.

Table 4. Results of YOLOv5 board detection for validation set with hyper-parameters

Image Size	Depth	Width	Size	Inference Time	Precision	Recall	mAP 0.5	mAP 0.5:0.95
512	1	1	95.4MB	37.7ms	0.979	0.989	0.995	0.774
544	0.4	0.9	49.4MB	21.4ms	0.992	0.998	0.995	0.823
576	0.6	0.9	61.9MB	24.4ms	0.995	0.994	0.995	0.824
608	0.6	0.9	61.9MB	24.6ms	0.991	1	0.995	0.84
640	1	1	95.4MB	31.3ms	0.993	0.996	0.995	0.849

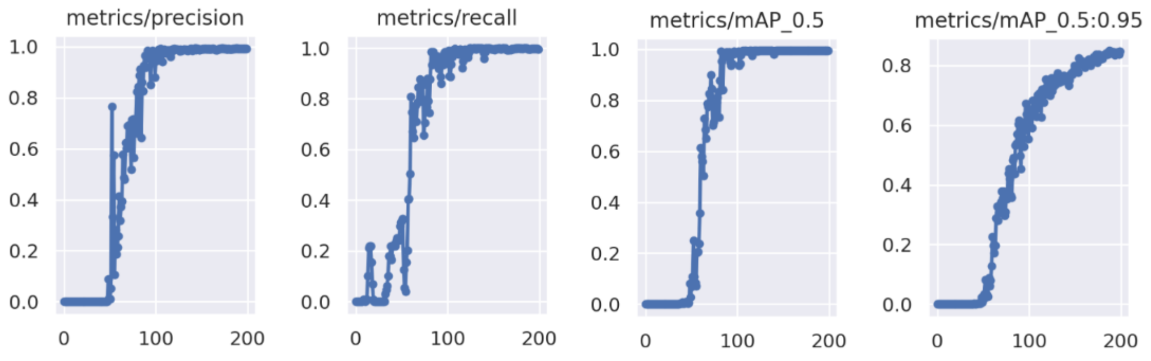


Figure 20. The results of the best YOLOv5 configuration over training epochs.

YOLOv5 architecture comparison graph shown in figure 21. In the graph, the black curve shows the default YOLOv5 architecture, with the depth and width set at 1. The other curves show different configurations of depth and width, paired with the curve style and colour.

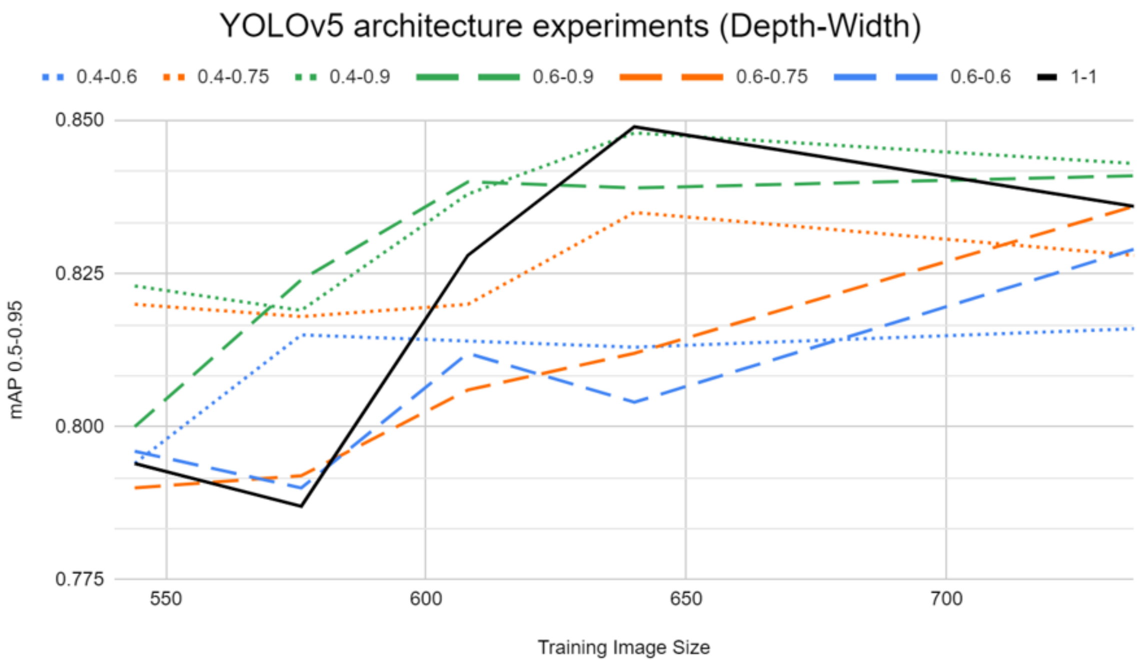


Figure 21. Architecture of YOLOv5 comparison

The most of the time, the lower depth outperformed the higher one. Another important observation is that the greater the width, the better the results in almost all cases.

4.3.2. Results of YOLOv7

The top 5 results of the experiments with YOLOv7 model and hyper-parameters are shown in Table 5. The size of the model has remained at 74.8 MB all along.

Table 5. Results of YOLOv7 board detection for validation set with hyper-parameters

Image Size	Batch Size	Epochs	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
512	32	125	15.4ms	0.989	0.996	0.995	0.775
544	32	125	15.6ms	0.991	1	0.995	0.753
576	26	125	15.3ms	0.990	1	0.995	0.765
608	26	125	15.8ms	0.997	1	0.995	0.811
640	24	125	15.8ms	0.993	1	0.995	0.793

As anticipated, the precision and recall increases with training image size, although the difference is not very great. The other metrics have a similar value, except the mAP_0.5:0.95 score, which is the highest when the training image size was equal to 608x608. The visual results of the model configuration that reached the highest score of mAP_0.5:0.95 are shown in Figure 22.

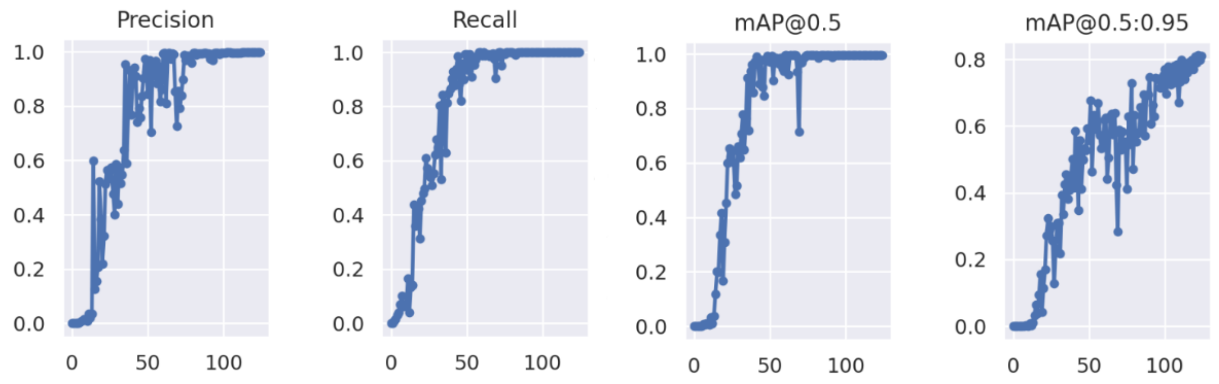


Figure 22. The results of the best YOLOv7 configuration over training epochs

The first three metrics start to flatter at about 90 epochs, but it looks like the mAP_0.5:0.95 value is still increasing.

4.3.3. Results of YOLOv9

The top 3 results for each version of YOLOv9 (C and E) and the configuration of the hyper-parameters are shown in Table 6. The size of the YOLOv9-C is 102.8 MB and the size of YOLOv9-E is 140.0 MB. The epochs for both - 20.

Table 6. Results of YOLOv9 board detection for validation set with hyper-parameters

YOLOv9	Image Size	Batch	Epochs	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
C	544	8	20	55.3ms	0.988	0.955	0.993	0.802
C	640	8	20	59.2ms	0.964	0.998	0.994	0.863
C	736	8	20	59.1ms	0.996	1	0.995	0.899
E	640	8	20	73.5ms	0.991	0.993	0.995	0.874
E	736	6	20	73.1ms	0.997	1	0.995	0.887
E	832	5	20	70.3ms	0.997	1	0.995	0.893

The results are as expected - the larger the size of the training image, the higher the precision and recall, although the difference is not high. Interestingly, changing the image size to a higher one, even than the original one, increased the results for all metrics. The other metrics have a similar value, except the mAP_0.5:0.95 score, which is the highest when the training image size is equal to 736x736 while using YOLOv9-C. The visual results of this model's configuration which reached the highest score of mAP_0.5:0.95 are shown in Figure 23.

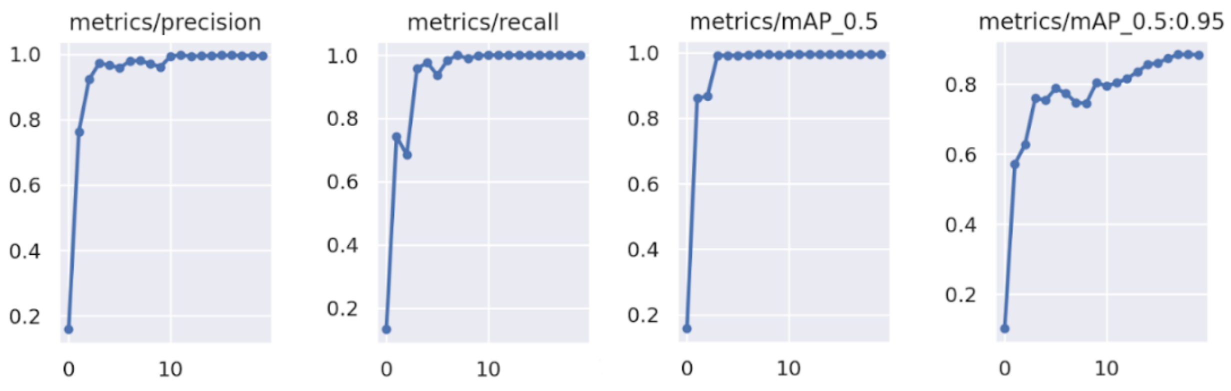


Figure 23. The results of the best YOLOv9 configuration (C)

The results show that the new approach of YOLOv9 trains very fast - at about 10 epochs. This is probably a result of using pre-trained weights for training of the model. All metrics start to flatter, except for the mAP_0.5:0.95 score. In the end, it looks like it starts to drop a bit.

4.3.4. Results of YOLOv11

The top 2 results for the 3 largest YOLOv11 versions (M, L, and X), along with their hyperparameter configurations, are summarized in Table 7. The size of YOLOv11-M is 40.5 MB, YOLOv11-L is 51.2 MB, and YOLOv11-X is 114.4 MB. The corresponding number of parameters is 20.1 million for YOLOv11-M, 25.3 million for YOLOv11-L, and 56.9 million for YOLOv11-X.

The most optimal model and configuration identified is the YOLOv11-M version, using a training image size of 736, batch size of 12, and 25 training epochs. This configuration achieved the

Table 7. Results of YOLOv11 board detection for validation set with hyper-parameters

YOLOv11	Image Size	Batch	Epochs	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
M	736	12	25	26.1ms	0.996	1	0.995	0.906
M	640	20	20	26.5ms	0.995	1	0.995	0.895
L	736	12	20	32.4ms	0.995	1	0.995	0.897
L	640	8	25	32.6ms	0.997	1	0.995	0.895
X	736	6	20	59.1ms	0.996	1	0.995	0.890
X	608	8	20	60.1ms	0.995	1	0.995	0.889

highest mAP_0.5:0.95 score of 0.906 and the lowest inference time of 26.1 ms. The model is also the smallest: a size of 51.2 MB and 20.1 million parameters, making it well-suited for our future tasks. All the relevant evaluation metrics were very similar for each model. The visual results of this configuration are presented in Figure 24.

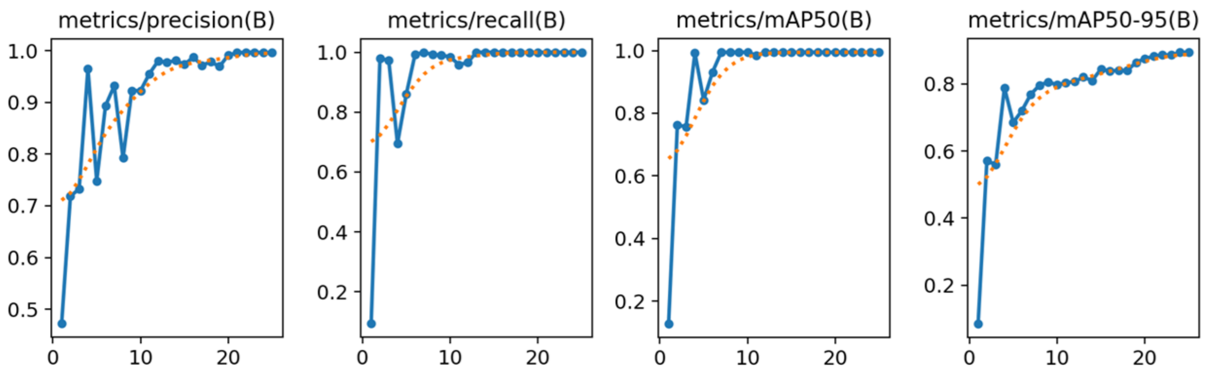


Figure 24. The results of the best YOLOv11 configuration (M)

The results indicate that YOLOv11, similar to YOLOv9, converges quickly and requires only a small number of training epochs. The model takes approximately 5 minutes to train and has shown good performance from the very beginning, most likely due to the use of well-optimised pre-trained weights. Most evaluation metrics begin to flatter after 15-20 epochs.

4.3.5. Results of YOLOv12

The top 2 results for the 3 largest YOLOv12 versions (M, L, and X), along with their hyperparameter configurations, are summarized in Table 8. The size of YOLOv12-M is 39.7 MB, YOLOv12-L is 53.7 MB MB, and YOLOv12-X is 119.6 MB. The corresponding number of parameters is 19.61 million for YOLOv12-M, 26.3 million for YOLOv12-L, and 59.2 million for YOLOv12-X.

Table 8. Results of YOLOv12 board detection for validation set with hyper-parameters

YOLOv12	Image Size	Batch	Epochs	Inference Time	Precision	Recall	mAP0.5	mAP 0.5:0.95
M	800	8	50	26.9ms	0.997	0.998	0.995	0.781
M	768	10	50	28.0ms	0.993	0.997	0.995	0.780
L	800	6	50	51.0ms	0.995	1	0.995	0.776
L	768	6	50	46.7ms	0.997	1	0.995	0.772
X	704	6	50	58.5ms	0.985	1	0.995	0.762
X	640	8	50	47.2ms	0.993	1	0.995	0.779

The most optimal model and configuration identified is the YOLOv12-M version, using a training and validation image size of 768, batch size of 10, and 50 training epochs. This configuration achieved an inference time of 26.9 ms and a model size of 39.7 MB. All the relevant evaluation metrics were very similar for each model. The visual results of this configuration are presented in Figure 25.

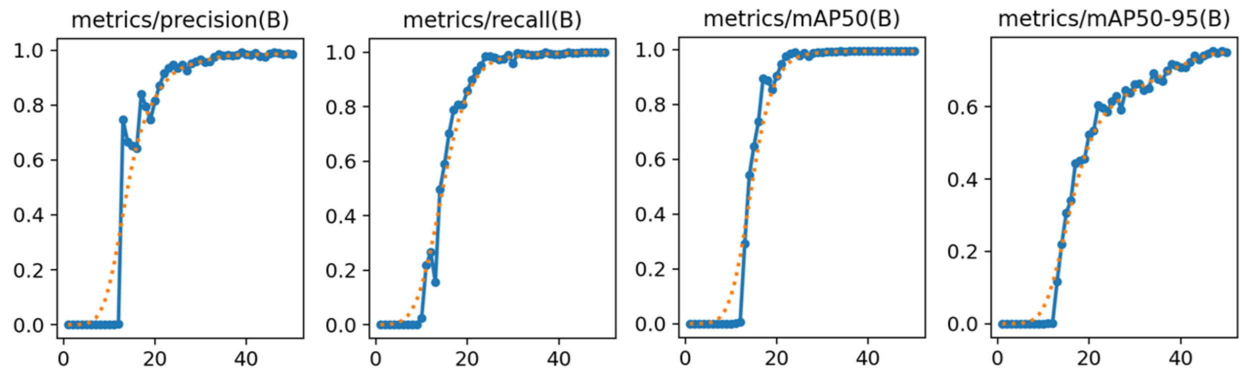


Figure 25. The results of the best YOLOv12 configuration (M)

The results indicate that YOLOv12 training is consistent, likely due to the model being trained from scratch. The total training time is approximately 20 minutes. In the early stages of training, mAP scores are close to zero, likely reflecting the model's attempts to adapt to the properties of the new dataset. Most evaluation metrics begin to flatter after 40 epochs.

4.3.6. Final Results of Board Detection

The graph in Figure 26 illustrates the comparison of YOLO models based on training image size and mAP_0.5:0.95 score. The blue curve - YOLOv5 model, the red - YOLOv7, the yellow/light orange - two versions of YOLOv9, the cyan - YOLOv11, and the dark orange - YOLOv12.

The YOLOv11 model demonstrated superior performance compared to the other models in terms of mAP_0.5:0.95, although the YOLOv9 model ultimately reached a comparable score. The performance difference between YOLOv9 configurations C and E was minimal. Meanwhile, YOLOv5 and YOLOv7 showed similar results, with YOLOv5 slightly outperforming YOLOv7. YOLOv12, on the other hand, exhibited the weakest performance overall.

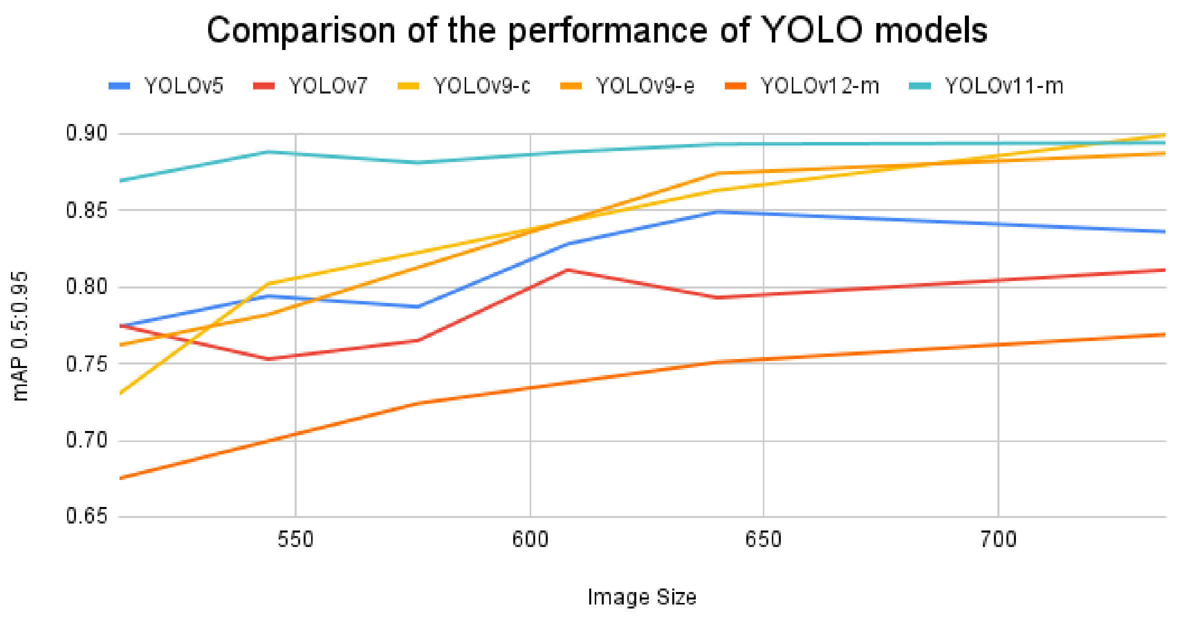


Figure 26. Comparison of the performance of YOLO models of board detection

The Figure 27 shows the best mAP_0.5:0.95 scores achieved by each model and architectural version, in relation to their respective model sizes and inference times (ms).

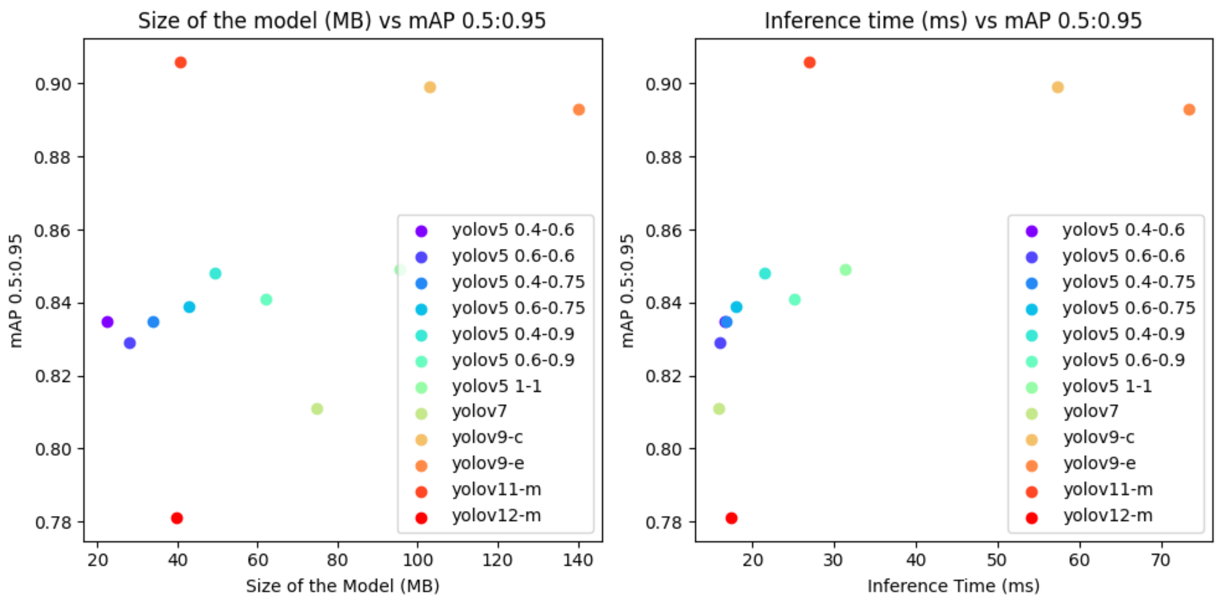


Figure 27. Relationship between model size and inference time for mAP_0.5:0.95 score of board detection

Among the YOLO models, YOLOv11 stands out as the most precise, fastest, and one of the smallest. In contrast, YOLOv7 achieves the lowest inference time but compromises precision when the default depth and width are set to 1. The results from YOLOv5 demonstrate that optimal configurations can be achieved through architectural experimentation. YOLOv9 exhibits strong precision but has the longest inference time and the largest model size. YOLOv12, overall, delivered the weakest performance.

For future investigations, we plan to use the YOLOv11-M model, which achieved a mAP_0.5:0.95 score of 0.906 and an inference time of 26.1 ms, with a model size of 40.5 MB and 20.1 million parameters. These results were obtained using an image size of 736, a batch size of 12, and 25 training epochs. The prediction results for this model are presented in Figure 28.

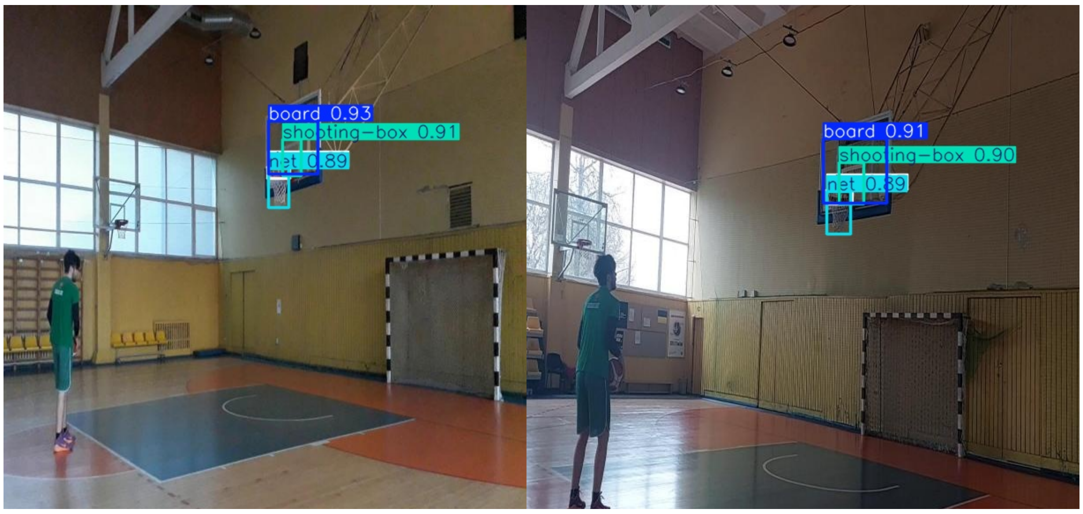


Figure 28. Prediction results of YOLOv11-M board detection

The results show great promise, with accurate predictions made at high confidence scores. However, it’s important to note that in certain cases, not all labels are visible.

5. Realization of Shot Recognition

In this section, we present an automatic basketball shot analysis system using video. The method uses the YOLOv11-N model for ball detection and the YOLOv11-M model for basketball backboard characterization. We also apply custom logic to determine shot attempts and their results.

5.1. Shot Detection System

We assume that the input video format is consistent with the dataset used in this study (see Section 3.7). To improve efficiency, the system analyzes every second frame of the video and resize frame to 640x640. For clarity, we refer to each of these analyzed frames as active frames. During the first 15 active frames, the system attempts to detect the basketball backboard and verify its position. If no significant changes are observed, the detected position is stored and used in subsequent computations. For backboard detection, only the rim and net components were considered, as including the shooting box and the board itself was considered not necessary of the initial stage of system development.

In each active frame, the system attempts to detect the basketball. Among the detected ball candidates provided by the model, only the one with the highest confidence score is selected. When a ball is successfully detected, the system evaluates its center position relative to the rim and determines whether it is sufficiently close. If the detected ball is found to be close to the rim, the system initiates ball detection and shot recognition calculations for every frame of the video, continuing this process until the ball moves away from the rim.

Shot recognition and outcome evaluation are structured into three main stages: *shooting*, *shot*, and *Made or Missed (MoM)* (see Figure 29). Each stage represents a separate phase of the ball trajectory and evaluation, forming the basic logic for determining whether a shot was successful.

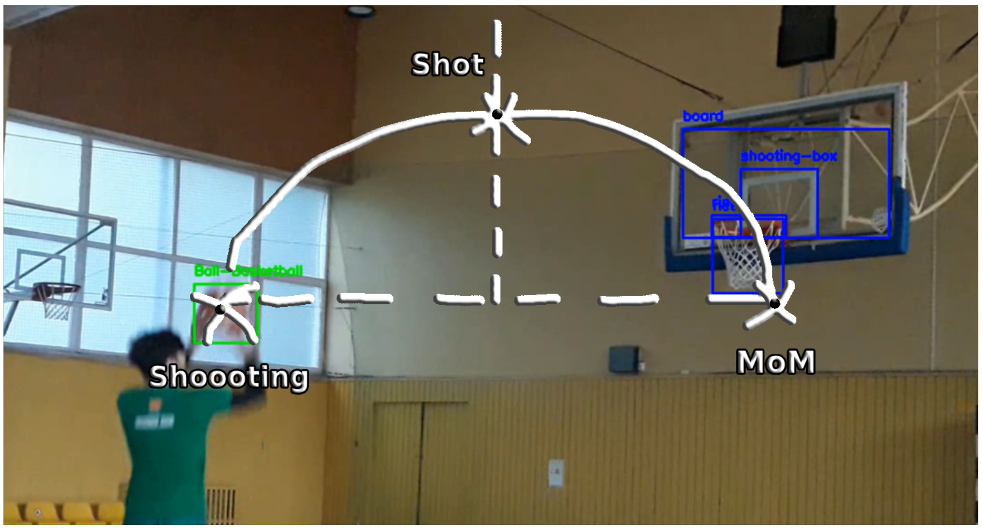


Figure 29. Stages of a shot recognition system

To initiate the *shooting* stage, the following core conditions must be met:

1. The center of the ball must be positioned higher than the lowest point of the net.
2. The three most recently detected ball positions must each show the ball moving progressively closer to the rim.
3. The *shot* stage must not have been activated yet.

If all of these conditions are satisfied, the system analyzes the distance between the rim's center and the ball's position at the moment the *shooting* stage begins.

To initiate the *shot* stage, the ball must pass the midpoint between the rim and its position at the start of the *shooting* stage. Entering the *shot* stage indicates that the shot count has increased. However, to determine whether the shot was made or missed, the system must enter the *MoM* stage. This stage is triggered when the detected center of the ball drops below the lowest point of the net.

The *MoM* stage involves a more complex evaluation. First, the system analyzes a sequence of 20 consecutive frames, recording whether the ball was detected or not in each frame. From this sequence, the algorithm attempts to identify a specific pattern consisting of three parts:

1. **Last Detected Position:** the final position of the ball before it temporarily disappears from detection.
2. **Detection Gap:** a series of frames where the ball is not detected.
3. **First Detected Position After the Gap:** the first position where the ball reappears after the gap.

This pattern of detection stages emerged during the system's implementation. We observed that when the ball enters the rim or net, it often becomes difficult for the system to detect it reliably.

Whenever an undetected gap is found between two positions of the ball, the system initiates a sequence of calculations. First, it checks whether both the last detected position before the gap and the first detected position after the gap fall within the horizontal and vertical boundaries of the rim. This suggests that the ball may have passed through the rim. To further minimize false positives in shot recognition, the system also calculates the spatial distance between these two positions. If the distance is within a reasonable threshold, indicating a realistic and continuous trajectory, the shot is classified as made. Otherwise, it is considered a missed shot.

5.2. Results of Shot Recognition

In this section, we present the evaluation of the system using both seen and unseen data, and discuss potential directions for future work and system improvements.

5.2.1. Results from Seen Data

The system were tested with the 53 videos of made and missed shots which was used to create original datasets for board and ball detection. The few examples of analyzed videos shown in Figure 30.

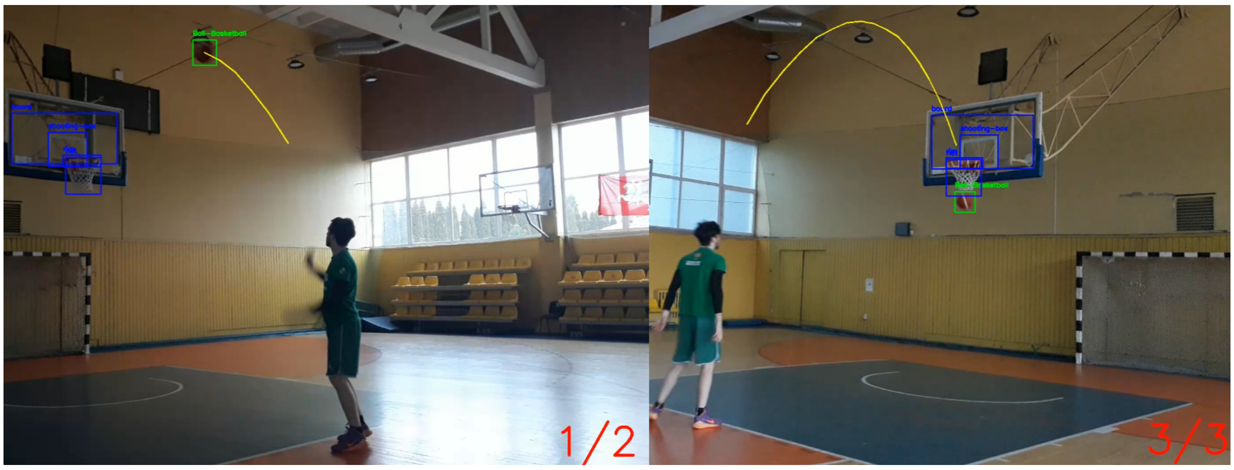


Figure 30. Results of the system from seen data

The system performed very well in terms of hoop and ball detection. Shot recognition achieved an accuracy of 96.2%, correctly identifying 153 out of 159 shots. However, the system also incorrectly identified 2 additional shots in situations where no shot had actually occurred.

5.2.2. Results from Unseen Data

In this section, we present the performance of the recognition system on seen and unseen data, including recordings from both indoor and outdoor basketball courts.

Results from Same Court

We retained several unused videos from the same court recording session specifically for evaluation purposes. Five of these videos were selected to test the shot recognition capabilities of the system. The results demonstrated excellent performance, with the system correctly identifying whether the shot was made or missed in 100% of the cases - achieving 16 correct recognitions out of 16.

Results from Unseen Indoor Court

The system was tested using two different videos recorded on unseen indoor courts, each featuring different color balls. The results of these tests are illustrated in Figure 31.

As expected, the shot recognition system did not identify the ball when it was of a different color than those present in the training data. However, the system demonstrated robustness by successfully detecting the original ball type and even recognizing an unfamiliar basketball backboard.

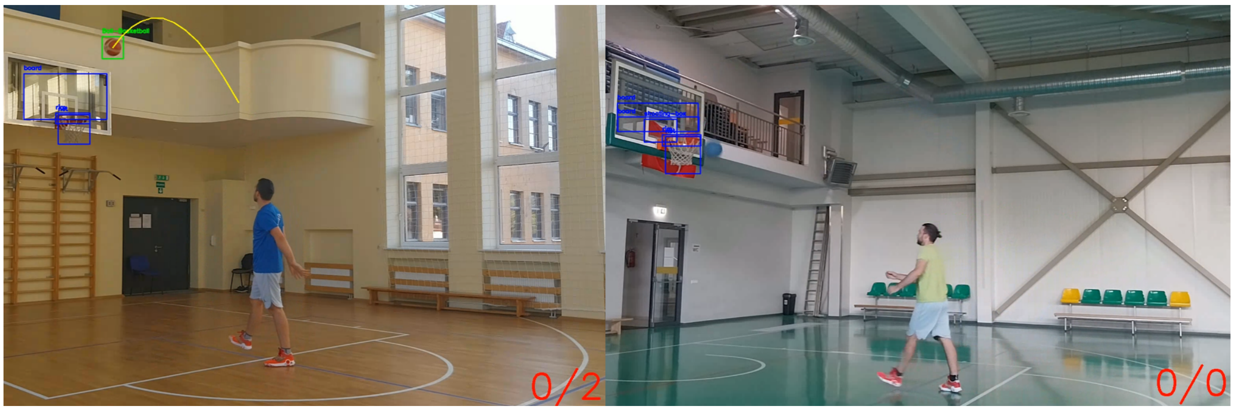


Figure 31. Results of the system from unseen indoor court data

In the right video, no shots were detected, which was consistent with the content. In the left video, the system correctly recognized 2 out of 3 shots, demonstrating promising generalization capabilities in previously unseen scenarios.

Results from Unseen Outdoor Court

The system was tested using two different videos recorded on unseen outdoor courts, each featuring different color balls. The results of one of these tests are illustrated in Figure 32.

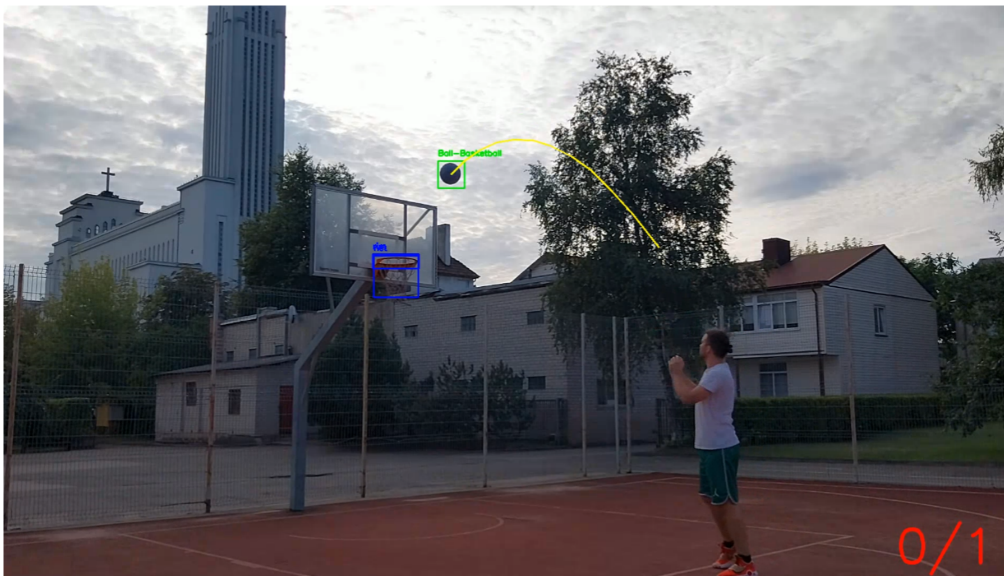


Figure 32. Results of the system from unseen outdoor court data 1

Unexpectedly, the shot recognition system was partially able to detect the ball even when its color differed from that used in the training data. Additionally, the system successfully identified the rim and net of the basketball backboard, enabling an attempt to recognize shots. Unfortunately, none of the shots were correctly recognized in this case (0 out of 3). This outcome contrasts with the results from the other video, which are presented in Figure 33.

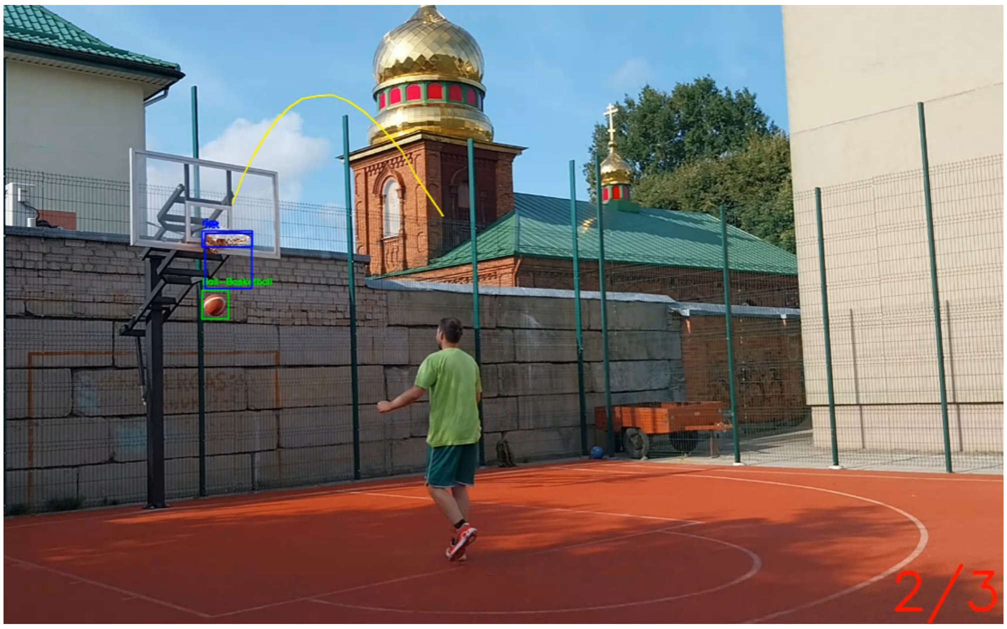


Figure 33. Results of the system from unseen outdoor court data 2

In this video, all shots were correctly recognized (3 out of 3), demonstrating perfect performance even when evaluated on previously unseen data.

5.3. Summary of Results

The shot recognition system demonstrated strong performance when evaluated on videos recorded on the same court as the training data. Surprisingly, it also performed reasonably well on unseen indoor and outdoor basketball courts. These results provide a promising outlook for future work, while also highlighting current limitations in the system's ability to generalize. Further improvements are needed to enhance its reliability across varied conditions.

5.4. Future Work

Further improvements will require access to a larger and more diverse dataset. Modifying the model architecture or training the models from scratch with more training epochs may lead to better detection performance. However, the main focus should be on improving the shot recognition process itself. One promising direction is to include additional labels in the training data, such as the ball being inside the rim or net. This could significantly enhance the system's ability to recognize successful shots.

Conclusions

1.1 Maintaining a consistent video recording environment, including the same court, camera angle, and ball, significantly contributed to high detection precision. For wider applicability, training datasets should include a variety of courts and ball appearance.

2.1 The evaluation metrics revealed no significant difference in performance when training with square versus rectangular images, when the total number of pixels remained similar.

2.2 The results indicate a clear correlation between model size and inference time, with larger models consistently requiring more time to complete inference.

2.3 The YOLOv8 model excelled in detection speed and model size for ball detection, but sacrificed precision. YOLOv7 and YOLOv5 performed comparably, with YOLOv7 being faster in detection but requiring longer training than YOLOv5.

2.4 The newly developed models YOLOv11 and YOLOv12 had smaller inference time compared to many of the previous models, reflecting a clear trend in favor of real-time performance in the latest developments.

2.5 The smallest model, YOLOv11-N, achieved the best inference time of 9.1 ms and had the most compact model sizes, while also achieved a high mAP_0.5:0.95 score of 0.749. Due to this strong balance between speed and accuracy, YOLOv11-N was selected for further development in this study.

3.1 Modifying the YOLOv5 architecture yields reduced inference time and a smaller model size, albeit with a modest impact on performance.

3.2 The greater the width of the model, the better the results in almost all cases. Most of the time, the lower depth of the model outperformed the higher one.

3.3 For board detection, YOLOv9 and YOLOv11 outperforms other models, achieving the highest mAP scores. However, YOLOv9 is also the slowest and largest model. In contrast, YOLOv7 slightly fall short YOLOv5 across most evaluation metrics, but YOLOv7 remains the fastest model. YOLOv12, on the other hand, exhibited the weakest performance overall (with Depth and Width set to 1).

3.4 The YOLOv11-M model achieved the highest mAP_0.5:0.95 score of 0.906 while maintaining a relatively small size and a low inference time of 26.1 ms. Due to the best precision and high efficiency, this configuration was chosen for further investigation in the study.

4.1 The system demonstrated excellent performance on videos recorded on the same court. When combining both seen and unseen data, it accurately recognized shots in 96.6% of cases (169 out of 175). These results suggest that consistency in the environment, such as court layout, lighting, and ball appearance, is a key factor for effective model training and reliable inference.

4.2 The shot recognition system performed surprisingly well on unseen indoor and outdoor basketball court data, correctly identifying 41.7% of shots (5 out of 12). While not perfect, these results demonstrate the model's potential and highlight promising directions for further system development and generalization.

Datasets were created and made publicly available to other researchers.

Based on the research and results of the Master's thesis, a paper was presented and published at the 29th International Conference Information Society and University Studies – IVUS 2024.

List of references

1. OFFERMANN, Eddie. *(20) "A smartphone of today has more computing power than NASA's 1960 supercomputer" | LinkedIn* [`https://www.linkedin.com/pulse/smartphone-today-has-more-computing-power-than-nasas-1960-offermann/`]. 2017. (Accessed on 05/21/2024).
2. KERR, Jeremy T; Marsha OSTROVSKY. From space to species: ecological applications for remote sensing. *Trends in ecology & evolution*. 2003, **jourvol** 18, **number** 6, **pages** 299–305.
3. LI, Kai; Wenyu YANG; Min YI; Zhigang SHEN. Graphene-based pressure sensor and strain sensor for detecting human activities. *Smart Materials and Structures*. 2021, **jourvol** 30, **number** 8, **page** 085027.
4. STARNER, Thad **andothers**. The perceptive workbench: Computer-vision-based gesture tracking, object tracking, and 3D reconstruction for augmented desks. *Machine Vision and Applications*. 2003, **jourvol** 14, **pages** 59–71.
5. WONG, Patrick. Identifying table tennis balls from real match scenes using image processing and artificial intelligence techniques. *International Journal of Simulation Systems, Science & Technology*. 2009, **jourvol** 10, **number** 7, **pages** 6–14.
6. LEE, Yong-Hwan; Youngseop KIM. Comparison of CNN and YOLO for Object Detection. *Journal of the semiconductor & display technology*. 2020, **jourvol** 19, **number** 1, **pages** 85–92.
7. LIU, Peihua; Nan YUE; Jiandong CHEN. A machine-learning-based medical imaging fast recognition of injury mechanism for athletes of winter sports. *Frontiers in public health*. 2022, **jourvol** 10, **page** 842452.
8. BARTH, Michael; Arne GÜLLICH; Christian RASCHNER; Eike EMRICH. The path to international medals: A supervised machine learning approach to explore the impact of coach-led sport-specific and non-specific practice. *PLoS One*. 2020, **jourvol** 15, **number** 9, e0239378.
9. SARAVANAN, Chandran. Color image to grayscale image conversion. **in***2010 Second International Conference on Computer Engineering and Applications*: IEEE, 2010, **volume** 2, **pages** 196–199.
10. SINGLA, Nishu. Motion detection based on frame difference method. *International Journal of Information & Computation Technology*. 2014, **jourvol** 4, **number** 15, **pages** 1559–1565.
11. GARCIA-GARCIA, Belmar; Thierry BOUWMANS; Alberto Jorge Rosales SILVA. Background subtraction in real applications: Challenges, current models and future directions. *Computer Science Review*. 2020, **jourvol** 35, **page** 100204.
12. PEI, Zhijun; Qingqiao TONG; Lina WANG; Jun ZHANG. A median filter method for image noise variance estimation. **in***2010 Second International Conference on Information Technology and Computer Science*: IEEE, 2010, **pages** 13–16.
13. BUADES, Antoni; Bartomeu COLL; Jean Michel MOREL. On image denoising methods. *CMLA Preprint*. 2004, **jourvol** 5, **pages** 19–26.

14. CHAKRABORTY, Bodhisattwa; Sukadev MEHER. A real-time trajectory-based ball detection-and-tracking framework for basketball video. *Journal of optics*. 2013, **jourvol** 42, **pages** 156–170.
15. CHAPLE, Girish; RD DARUWALA. Design of Sobel operator based image edge detection algorithm on FPGA. **in***2014 International Conference on Communication and Signal Processing*: IEEE, 2014, **pages** 788–792.
16. *OpenCV: Canny Edge Detection* [`https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html`]. **nodate** (Accessed on 05/21/2024).
17. WANG, Xin. Laplacian operator-based edge detectors. *IEEE transactions on pattern analysis and machine intelligence*. 2007, **jourvol** 29, **number** 5, **pages** 886–890.
18. CHA, J; Rufus H COFER; Samuel P KOZAITIS. Extended Hough transform for linear feature detection. *Pattern Recognition*. 2006, **jourvol** 39, **number** 6, **pages** 1034–1043.
19. *OpenCV: Feature Detection* [`https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d`]. **nodate** (Accessed on 05/21/2024).
20. *OpenCV: Harris Corner Detection* [`https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html`]. **nodate** (Accessed on 05/21/2024).
21. XIE, Xingxing **andothers**. Oriented R-CNN for object detection. **in***Proceedings of the IEEE/CVF international conference on computer vision*: 2021, **pages** 3520–3529.
22. UNADKAT, Vyom **andothers**. Code-free machine learning for object detection in surgical video: a benchmarking, feasibility, and cost study. *Neurosurgical Focus*. 2022, **jourvol** 52, **number** 4, E11.
23. DEWI, Syarifah Rosita **andothers**. Deep Learning Object Detection Pada video menggunakan tensorflow dan convolutional neural network. 2018.
24. STÄCKER, Lukas **andothers**. Deployment of deep neural networks for object detection on edge ai devices with runtime optimization. **in***Proceedings of the IEEE/CVF International Conference on Computer Vision*: 2021, **pages** 1015–1022.
25. REDMON, Joseph; Santosh DIVVALA; Ross GIRSHICK; Ali FARHADI. You only look once: Unified, real-time object detection. **in***Proceedings of the IEEE conference on computer vision and pattern recognition*: 2016, **pages** 779–788.
26. REDMON, Joseph; Ali FARHADI. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*. 2018.
27. BOCHKOVSKIY, Alexey; Chien-Yao WANG; Hong-Yuan Mark LIAO. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*. 2020.
28. VILCAPOMA, Piero **andothers**. Comparison of Faster R-CNN, YOLO, and SSD for Third Molar Angle Detection in Dental Panoramic X-rays. *Sensors*. 2024, **jourvol** 24, **number** 18. ISSN 1424-8220. **urlfrom**DOI: `10.3390/s24186053`.
29. ULTRALYTICS. *YOLOv5 vs. EfficientDet: A Detailed Comparison for Object Detection — docs.ultralytics.com* [`https://docs.ultralytics.com/compare/yolov5-vs-efficientdet/`]. **nodate** [Accessed 23-05-2025].
30. *ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite* [`https://github.com/ultralytics/yolov5`]. 2024. (Accessed on 02/05/2024).

31. WANG, Chien-Yao; Alexey BOCHKOVSKIY; Hong-Yuan Mark LIAO. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. **in***Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*: 2023, **pages** 7464–7475.
32. TERVEN, Juan; Diana CORDOVA-ESPARZA. A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. *arXiv preprint arXiv:2304.00501*. 2023.
33. SHAFIEE, Mohammad Javad; Brendan CHYWL; Francis LI; Alexander WONG. Fast YOLO: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*. 2017.
34. WANG, Chien-Yao; I-Hau YEH; Hong-Yuan Mark LIAO. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. *arXiv preprint arXiv:2402.13616*. 2024.
35. KIN-YIU, Wong. *WongKinYiu/yolov9: Implementation of paper - YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information* [`https://github.com/WongKinYiu/yolov9`]. 2024. (Accessed on 03/25/2024).
36. SALTIK, Ahmet Oğuz; Alicia ALLMENDINGER; Anthony STEIN. Comparative analysis of yolov9, yolov10 and rt-detr for real-time weed detection. *arXiv preprint arXiv:2412.13490*. 2024.
37. HE, Zijian **andothers**. Comprehensive Performance Evaluation of YOLOv11, YOLOv10, YOLOv9, YOLOv8 and YOLOv5 on Object Detection of Power Equipment. *arXiv preprint arXiv:2411.18871*. 2024.
38. KHANAM, Rahima; Tahreem ASGHAR; Muhammad HUSSAIN. Comparative performance evaluation of yolov5, yolov8, and yolov11 for solar panel defect detection. **in***Solar*: MDPI, 2025, **volume** 5, **page** 6. **number** 1.
39. SAPKOTA, Ranjan **andothers**. Comprehensive performance evaluation of yolo11, yolov10, yolov9 and yolov8 on detecting and counting fruitlet in complex orchard environments. *Authorea Preprints*. 2024.
40. TIAN, Yunjie; Qixiang YE; David DOERMANN. Yolov12: Attention-centric real-time object detectors. *arXiv preprint arXiv:2502.12524*. 2025.
41. *YOLOv5 - Ultralytics | Revolutionizing the World of Vision AI* [`https://ultralytics.com/yolov5`]. **nodate** (Accessed on 05/28/2023).
42. JOCHER, Glenn. *GitHub - ultralytics/yolov5: YOLOv5 in PyTorch > ONNX > CoreML > TFLite* [`https://github.com/ultralytics/yolov5`]. **nodate** (Accessed on 05/21/2024).
43. *YOLOv5 Object Detection Model: What is, How to Use* [`https://roboflow.com/model/yolov5`]. 2020. (Accessed on 05/21/2024).
44. JACOB SOLAWETZ, Francesco. *What is YOLOv8? The Ultimate Guide. [2024]* [`https://blog.roboflow.com/whats-new-in-yolov8/`]. 2023. (Accessed on 05/21/2024).
45. MEHRA, Akshit. *Understanding YOLOv8 Architecture, Applications & Features* [`https://www.labellerr.com/blog/understanding-yolov8-architecture-applications-features/`]. 2023. (Accessed on 05/21/2024).
46. KHANAM, Rahima; Muhammad HUSSAIN. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*. 2024.

47. JOCHER, Glenn; Jing QIU. *Ultralytics YOLO11*. 2024. **version** 11.0.0. **urlalso**: `https://github.com/ultralytics/ultralytics`.
48. TIAN, Yunjie; Qixiang YE; David DOERMANN. *YOLOv12: Attention-Centric Real-Time Object Detectors*. 2025. **urlalso**: `https://github.com/sunsmarterjie/yolov12`.
49. ŠIRMENIS, Juozas; Mantas LUKOŠEVIČIUS. Tracking basketball shots–preliminary results. **in***CEUR workshop proceedings: IVUS 2021: Information society and university studies 2021: Proceedings of the 26th international conference on information society and university studies (IVUS 2021), Kaunas, Lithuania, April 23, 2021*: CEUR-WS, 2021, **volume** 2915, **pages** 181–190.
50. *WongKinYiu/yolov7: Implementation of paper - YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors* [`https://github.com/WongKinYiu/yolov7`]. 2024. (Accessed on 02/05/2024).
51. KHANAM, Rahima; Muhammad HUSSAIN. A Review of YOLOv12: Attention-Based Enhancements vs. Previous Versions. *arXiv preprint arXiv:2504.11995*. 2025.