UNIVERSITY OF AMSTERDAM

Optimising YOLOv8n using Maximum Class Separation and Data Augmentation for NAO robots

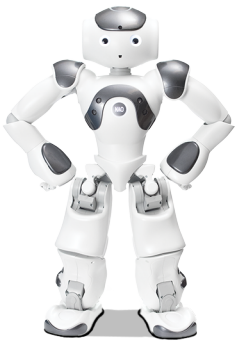
February 4, 2024

Students:
Joost Weerheim
13769758

Stephan Visser
13977571

Kim Koomen
14621312

Ross Geurts
14599996



Tutor:
Bo Flachs

Group:
F.1

Lecturer:
Iris Groen

Course:
Leren en Beslissen

Course code:
5082LEBE5Y

Abstract

Object detection is a challenging problem in both robotics and, consequently, robot soccer. The Standard Platform League (SPL) is composed exclusively of NAO robots aiming to compete in robot soccer. It is crucial that the robots perform object detection in near real time in order to play competitive robot soccer. Object detection refers to a machine learning task where the goal is to predict the location of one or multiple objects in an image, draw a well-fitting bounding box around them, and classify the object within each bounding box. YOLOv8 is a lightweight model, primarily used for object detection, often in embedded systems with limited computational resources, such as the NAO robot. In this report we aim to optimise the YOLOv8n model using two different optimisations: data augmentation and maximum class separation. The former aims to mimic real world camera scenarios that the robots must handle during a soccer match, while the latter is achieved by a technique that was introduced in the 2022 paper, *"Maximum Class Separation as Inductive Bias in One Matrix"*. We applied the maximum separation technique on the YOLOv8n model which led to poor performance due to the YOLOv8 architecture design. This led to the conclusion that YOLOv8 is not well suited for maximum class separation. Data augmentation involved using three augmentations; gaussian blurring, brightness and colour jitter. Although only marginal improvements up to 2% were observed, the research indicates potential for future endeavors.

Contents

1 Problem Statement **3**

2 Dataset **4**
2.1 Cleaning of the dataset 4
2.2 Labels in the dataset 4

3 Approach **6**
3.1 Exploring Maximum Class Separation and YOLOv8n 6
3.1.1 Maximum Class Separation as Inductive Bias in One Matrix 6
3.1.2 Implementation for YOLOv8n model 7
3.2 Data Augmentation 10

4 Results **11**
4.1 Splits and Metrics 11
4.2 Maximum Class Separation in YOLOv8n 11
4.3 Data Augmentation 13

5 Discussion **15**
5.1 Maximum Class Separation in YOLOv8n 15
5.2 Data Augmentation 15

6 Conclusion **17**

1 Problem Statement

The Intelligent Robotics Lab at the University of Amsterdam has several NAOv6 robots. Currently the NAOv6 is the most recent NAO model developed by Softbank Robotics and the most common model in the Standard Platform League (SPL). Each robot is equipped with a 1.91 GHz Intel Atom E3845 CPU, along with a quad-core processor with a single thread per core. The limited computational resources of the NAOv6 have led teams participating in the SPL to create their own custom lightweight frameworks that aim to autonomously play robot soccer. One crucial task these frameworks aim to accomplish is utilising the images from both cameras on the head of the NAOv6 to perform object detection, a computationally intensive task.

Object detection is a modern image processing technique responsible not only, for detecting which objects of a certain class are present in an image, but more importantly, provide exact coordinates of the location of the objects, relative to the image. Traditional object detection models have achieved this by processing an image multiple times. However, since 2015 a new model has entered the marketplace that provided revolutionary speed improvements in object detection and classification tasks, namely You Only Look Once (YOLO), developed by Ultralytics (Redmon et al. 2016). The YOLO architecture processes an image only once to predict bounding boxes and their corresponding class probabilities for each object in the image. This approach has a significant speed improvement over other object detection models, such as Faster R-CNN (Yin, H. Li, and Fu 2020, p. 8; Hussain 2023, p. 3).

One approach to optimise classification tasks in deep neural networks involves applying a fixed matrix to the class logits of the network before they get forwarded to the softmax function, as outlined by Kasarla et al. (2022). This matrix enhances model accuracy by introducing maximum separation as an inductive bias which has been implemented on AlexNet and ResNet models by the authors of the paper, allowing other researchers to apply the technique to other models. In our research, the primary goal is to apply this optimisation technique, as detailed in Kasarla et al. (2022) and elaborated in section 3, to improve the performance of the YOLOv8 model tasked with object detection. The combination of the speed of YOLOv8 and the limited computational resources of the NAOv6 have led us to selecting YOLOv8 as our object detection model. Our aim is to find out to what degree we can optimise the YOLOv8 model without causing a noticeable decrease in runtime speed.

2 Dataset

To train our model, we made use of a labelled image dataset made by the Irish robot soccer ball team RoboEireann[1], combined with a dataset from the Italian robotics team Unibas[2]. These datasets where obtained from the RoboCup SPL website[3]. The first dataset contains 4906 images, with a resolution of 1280x960[4], captured from both the top and bottom cameras mounted inside the head of a NAOv6 robot, paired up with text files which denote the different bounding boxes of objects present in the images adhering to the YOLO format specifications. The different labels that are present in the dataset include balls, robots, goal posts and penalty spots. These labels appear regularly to the robots during a game.

The second dataset contains 3693 images which were also taken from both cameras in the head of the NAOv6. The dataset contains the same labels as the first dataset with an additional fifth label, namely the center spot. This is the spot in the center of the soccer field that the ball is placed on during kickoff. For this dataset, the label files are in a Pascal/VOC formatted XML file.

2.1 Cleaning of the dataset

Since both of the previously mentioned datasets differ in data structure, partial pre-processing tasks were needed for the data to match the YOLO format specifications. The XML files in the second dataset were converted through a custom program in order to adhere to these specifications. Furthermore, the additional fifth label in the second dataset had to be removed, as it was the only label being absent in at least one of the datasets and often labelled incorrectly. Finally, both datasets were merged into a single directory containing a total of 7583 images.

2.2 Labels in the dataset

The dataset contains the following four classes:

- *ball*: the soccer ball used to play the game
- *robot*: one of the players
- *goal_post*: referring to the two pillars on the left and right side of the goal itself
- *pen_spot*: referring to the ground mark "×" penalty spot in front of the goal post

The images often contain more than one object, causing an image to have several labels. These are distinct as they are accompanied by a bounding box, which is a rectangle drawn directly around the instance. The visualisation below displays these labelled bounding boxes.

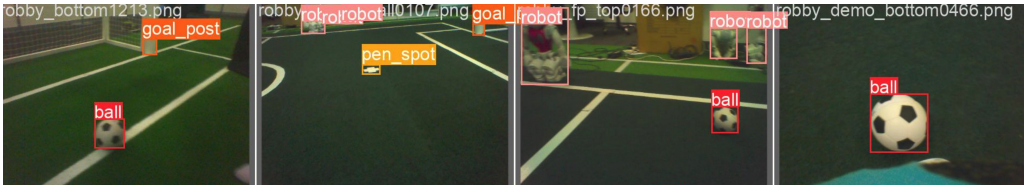
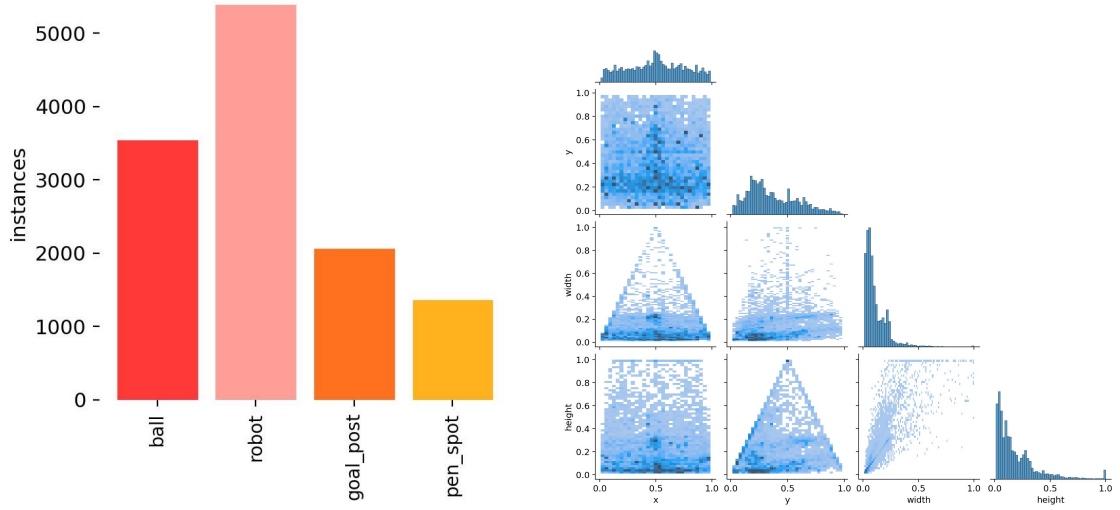


Figure 1: Bounding boxes drawn around the labelled instances.

The distribution of the labels can be seen in Figure 2. It shows that the robot instance is significantly more common to occur in the images than the other instances. This is not surprising, as there can be up to 14 robots on the field during a game according to the SPL rule book[5]. Despite the significant variation in the number of instances per class, it would not be accurate to characterise this data as long-tailed. To achieve this, the smaller or "tail" classes, such as *pen_spot*,

[1] https://roboeireann.maynoothuniversity.ie/research/SPLObjDetectDatasetV2.zip
[2] https://drive.google.com/drive/folders/1WeY1_Ql4ZJsH01DIFtnlDYj5c_jsGL59
[3] https://spl.robocup.org/datasets/
[4] http://doc.aldebaran.com/2-1/family/robots/video_robot.html
[5] https://spl.robocup.org/wp-content/uploads/SPL-Rules-2023.pdf

should ideally have only a limited number of instances. However, during training, an imbalance in class distribution may cause the model to disproportionately focus on larger or "head" classes (B. Li et al. 2022). This imbalance adversely affects the mean average precision (mAP, as detailed in subsection 3.2 and subsection 4.1) of the tail classes, as the objects are not always detected as a result of the model not having been exposed to enough samples during training. Figure 2 also shows the correlation of the labels of the bounding boxes. The figure shows that the width of the bounding boxes is often smaller than the height. This is intuitive as the bounding box for robots and goal posts is a thin and tall rectangle, whereas it is a square for the other instances. Furthermore, the x and y are values that correspond to the center of the bounding box that are evenly distributed across the data. This means that the objects are spread around the images, rather than appearing in the same spot. Thus, the model is not trained on the location of objects to determine their class.



Distribution of the labels

Correlation of the labels

Figure 2: Visualization of the labels in the dataset.

3 Approach

The goal for this project is to optimise the YOLOv8 detection model by implementing the class separation optimisation proposed by Kasarla et al. (2022), along with utilising data-augmentation. In this section, we will discuss the paper by Kasarla et al. (2022), describing the paper's findings and our reconstruction of it using ResNet. Next, we will present our findings about the YOLOv8 framework and implementation details for maximum class separation, supported using visual representations. Lastly, we will elaborate our approach for using data augmentation.

3.1 Exploring Maximum Class Separation and YOLOv8n

3.1.1 Maximum Class Separation as Inductive Bias in One Matrix

"An inductive bias allows a learning algorithm to prioritise one solution (or interpretation) over another, independent of the observed data" (Battaglia et al. 2018). Classification is a supervised machine learning task used to predict a class from a predefined number of labels given some input data. Classification in learning algorithms can be improved by maximising the class separation. A common model for this are support vector machines which try to maximise the margins between data points and the hyperplanes that separate them (Cortes and Vapnik 1995). "In deep learning-based classification, neural networks are by default not equipped with a maximum separation inductive bias. The cross-entropy loss with a softmax function is a common choice to train a classifier, aiming for discriminative power. Yet this framework does not explicitly drive maximum separation between the classes" (Kasarla et al. 2022). The authors in the study by Kasarla et al. (2022) found that instead of seeing class separation as an optimisation problem, it can be optimally solved in closed form. They propose an inductive bias that can be encoded into a deep neural network by adding a fixed matrix to the class logits before applying the softmax function. For any finite k classes, the matrix represents k maximally separated unit vectors on a $(k-1)$ dimensional hypersphere (user1551 2014). The figure below displays the recursive approach on how to reconstruct the matrix for k classes.

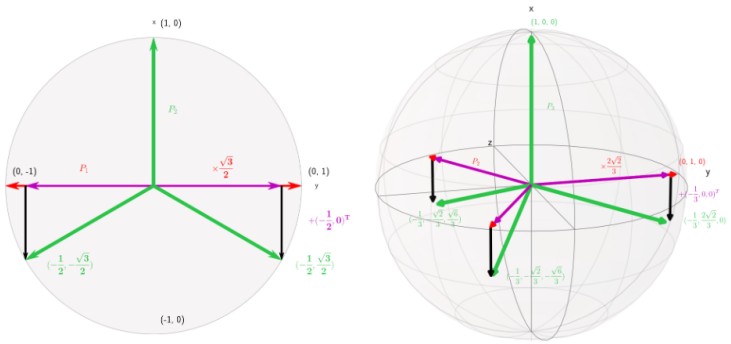


Figure 3: k-maximally separated unit length vectors on a $(k-1)$ dimensional hyperspehere. The 2D figure on the left shows a recursive update from 2 to 3 classes. The 3D figure on the right shows a recursive update from 3 to 4 classes. Red vectors depict the initial state, while green vectors illustrate the outcome post matrix multiplication. Visualisation made by the authors of Kasarla et al. (2022).

Applying a matrix multiplication on the fixed matrix with the logits layer should give back the logits embedded with a maximum separation. The resulting output can then be used as the inputs for the softmax function. This method was tested on several models such as ResNet and AlexNet using the CIFAR-10 and CIFAR-100 dataset. Authors of the paper also tested their method on long-tailed data which resulted in even better performance increases, which can be seen in the table below.

	CIFAR-100					CIFAR-10				
	-	0.2	0.1	0.02	0.01	-	0.2	0.1	0.02	0.01
ConvNet	56.70	45.97	40.34	27.35	16.59	86.68	79.47	73.90	51.40	43.67
+ This paper	**57.05**	**46.59**	**40.44**	**28.27**	**18.40**	**86.76**	**79.63**	**75.88**	**55.25**	**48.05**
	+0.35	+0.62	+0.10	+0.92	+1.81	+0.08	+0.16	+1.98	+3.85	+4.38
ResNet-32	75.77	65.74	58.98	42.71	35.02	94.63	88.17	83.10	68.64	56.98
+ This paper	**76.54**	**66.01**	**60.54**	**45.12**	**38.85**	**95.09**	**91.42**	**88.16**	**77.02**	**69.70**
	+0.77	+0.27	+1.56	+2.41	+3.83	+0.46	+3.25	+5.06	+8.38	+12.72

Figure 4: Adding maximum separation as inductive bias on CIFAR-10 and CIFAR-100 for standard and imbalanced settings. Results obtained by the authors of Kasarla et al. (2022).

We reproduced part of these results on the CIFAR-10 dataset to validate whether the results of the paper were indeed accurate. In order to do this, we used the ResNet-32 model on the CIFAR-10 dataset and trained it with and without the matrix applied. In order to obtain accurate measurements we trained both scenarios 3 times for 50 epochs each. The authors of the paper trained for 200 epochs but due to computational and time constraints, this was not feasible. As a result of this, we expected the accuracy to be lower but still indicative of whether the matrix made a difference. The results can be seen in the table below.

	CIFAR-10	CIFAR-10 With Imbalance Factor 0.1
ResNet-32	0.9434	0.6999
+ This paper	0.9443	0.7014
	+0.095%	+0.214%

Table 1: Adding maximum separation as inductive bias on CIFAR-10 for standard and imbalanced settings.

These results approximately followed the trend in the results obtained in the paper, thus concluding that adding the fixed matrix does indeed increase classification accuracy.

3.1.2 Implementation for YOLOv8n model

The YOLO version that has been used to conduct this research is YOLO version 8[6], which at this point in time is considered the state-of-the-art model of Ultralytics. Compared to other object detection models, such as Fast R-CNN or YOLOv5, the YOLOv8 boasts an improved accuracy and speed, making it excel in real-time object detection. Furthermore, YOLOv8 serves diverse models of varying sizes, out of which their most lightweight model, the nano (v8n) model, has been used for this project. The lightweight feature enhances speed, making it particularly suitable for the scope of this research as well as feature endeavors. However, YOLOv8n is a single shot detector, meaning that YOLOv8n performs both classification and localisation of bounding boxes in a single forward pass of the network, as opposed to other detection algorithms which perform localisation and classification separately. Because of this, single shot detectors boast improved speed compared to other detection algorithms, which also further complicates the task of achieving maximum class separation, as it requires to be implemented solely on the output designated for classification, which is intertwined with localization. Implementing maximum class separation for a two-shot detection model, such as Fast R-CNN, has been considered, but the possible performance increase gained from implementing this would not realistically be able to offset the decrease in performance when using such a model.

Since the optimisation method mentioned in the paper by Kasarla et al. (2022) was proposed for a classification model, applying the optimisation to an object detection model, such as YOLOv8, requires a consideration of where and how to apply it in the architecture. The YOLOv8 architecture, as shown in Figure 5, consists of three parts:

- The *backbone* (feature extractor) uses as a ResNet-50 deep convolutional neural network and is responsible for processing and extracting meaningful features from the input image.

[6] `https://docs.ultralytics.com/`

- The *neck* acts as a bridge between the backbone and the head by collecting feature maps from different stages of the backbone, reducing the size of the feature maps and increasing the resolution of the features.
- The *head* is responsible for taking the feature maps and uses them to predict the class probabilities and bounding boxes of objects in the image.

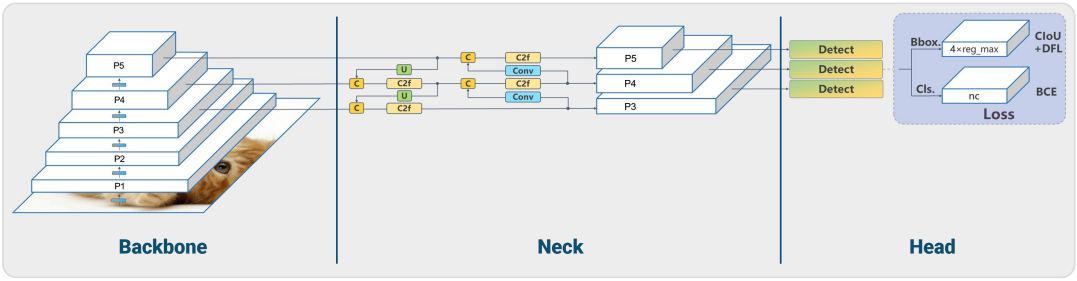


Figure 5: The YOLOv8 architecture depicting the backbone, neck and head from left to right. The three detect layers can be seen on the far right. Original visualisation made by GitHub user RangeKing.

The head of the YOLO network consists of three *detect* modules, which are used to detect three varying sizes of objects, each having a separate output. Furthermore, one of the techniques YOLO models apply for detecting objects is to divide the input image up into a grid of cells for which bounding boxes are predicted independently. These two design decisions result in the network producing an output which consists of a list of three four-dimensional tensors of shape $[B, No, H, W]$. Here, B is the batch size used during training whereas H and W are the height and width of the cell grid respectively. No is comprised of the amount of C classes along with the maximum number of bounding box predictions per grid cell (denoted as *reg_max*) multiplied by four ($reg\_max \cdot 4$). The multiplication by four is a result of each bounding box prediction consisting of x, y, *width* and *height* parameters. See Figure 6 for a visual representation.

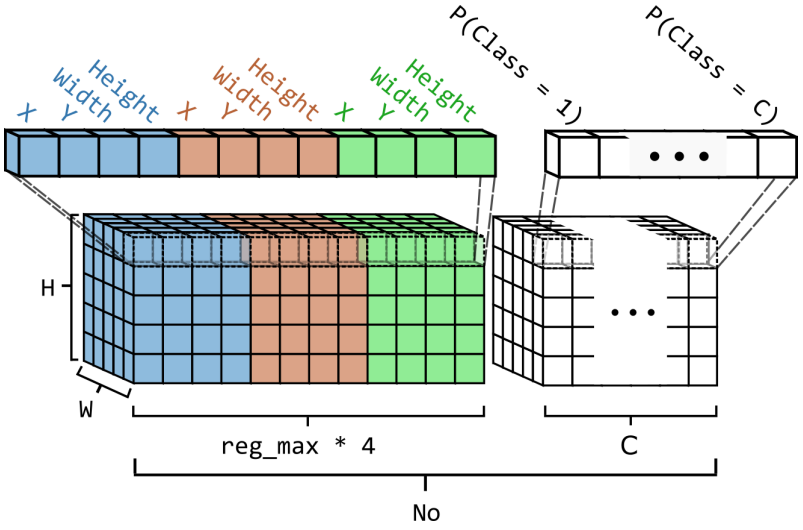


Figure 6: 3D visualisation of the output tensor of a detection module from the YOLO framework for a single batch.

After close consideration and a meeting with one of the authors of the paper, it was concluded that the application of the fixed matrix should be done on the output of the head of the network before the activation function has been applied. Nevertheless, it is theoretically possible to apply this fixed matrix in alternative parts of the architecture, albeit at the expense of no longer achieving maximum class separation.

In order to incorporate this optimisation, the Ultralytics repository[7] has been forked, allowing

[7] `https://github.com/ultralytics/ultralytics`

to modify and integrate the optimisation into our program for training the model with a custom dataset. Next, alterations have been made to the *forward* method of the *v8DetectionLoss* class, which receives the output of the network, applies an activation function and calculates the loss. During this process, the output of the network gets reshaped to a $No \times (H \cdot W)$ tensor and split to form two two-dimensional tensors per batch of size $(reg\_max \cdot 4) \times (H \cdot W)$ containing information about the bounding boxes and $C \times (H \cdot W)$ containing information about the class logits. See Figure 7 for a visual representation of the two tensors.

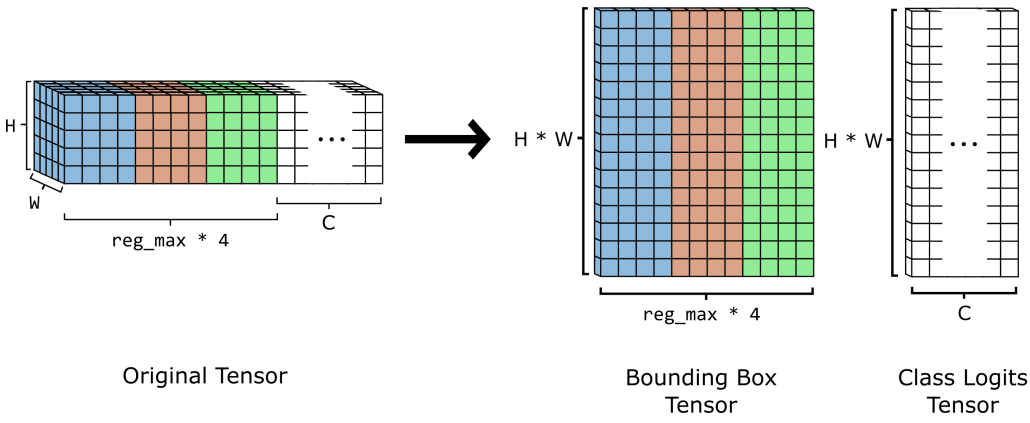


Figure 7: 3D visualization of the output for a single batch before and after the reshaping and splitting has taken place.

Here, the maximum class separation matrix should be applied to the tensor containing the class logits. However, the paper specifies that for a dataset containing C classes, the matrix used for maximum separation will be of size $C \times (C-1)$, whereas the final output of the network should be of size $(C-1) \times (W \cdot H)$. Therefore, a constraint has been put on the last layer of the network, such that it will return the correctly sized output tensor. This can be done by either removing any row of the output tensor or adjusting the size of the final convolutional layer in the detect module before applying the matrix operation. This technique proves effective because, for C classes, it suffices to maximally separate only $(C-1)$ classes, as the final class vector will be separated as a result. See Figure 8 for a visual representation of the maximum separation matrix application.

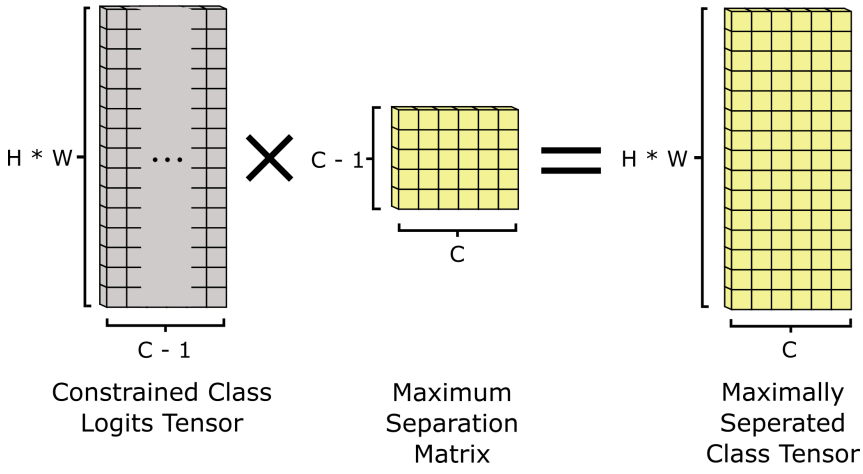


Figure 8: 3D visualization of the maximum separation matrix multiplication applied on the constrained tensor.

Alternatively, the maximum separation matrix can be applied within the forward method of the detect module. This involves intercepting the output tensor before it is returned by the function and repeating the aforementioned steps. However, this approach requires the splitting and subsequent reconstruction of the output tensor to ensure compatibility with the remaining code, potentially affecting runtime speed, especially with larger datasets.

3.2 Data Augmentation

Research into whether data augmentation would enhance the accuracy of the model was conducted alongside the application of the matrix as a secondary goal. The augmentation of data is a common machine learning technique used to increase variability in data during the training of a model, which can prevent overfitting (Guo and Gould 2015). However, data augmentation for object detection rather than classification is more complex as a result of bounding boxes. The model is not only evaluated on classification of the objects, but also on the placement of the bounding boxes. Thus, the location of the boxes has to stay consistent even when the image is distorted (Zoph et al. 2020). Even though the YOLOv8 framework does allow for built-in data augmentation by changing the configuration of the model, the following techniques are all original implementations. As a result, it allows for more control over the type and degree of augmentation done on the dataset, which is essential when researching the effects of the augmentations.

For this, three different kinds of augmentation - gaussian blurring, colour jitter and brightness - are examined. These augmentations were selected as they mimic real world scenarios that the robots occur during matches. We measure the performance of the model with 4 different metrics: precision (P), recall (R), the mean average precision taken at the intersection over union (IoU) threshold of 0.5 (mAP50) and at the IoU threshold between 0.5 and 0.95 (mAP50-95). When looking at the evaluation of the model results after applying augmentations to the data, we prioritise the mAP results, as these capture a balance between precision and recall. Gaussian blurring is applied with a maximum sigma of 5 in order to imitate slight blurring that can occur whenever the robot is moving. Brightness is adjusted with a factor of 0.5 to 1.5, modelling extreme edge cases where the robot is standing next to a window when it is either dark and cloudy or very sunny. Lastly, colour jitter is used to improve detection as all objects are greyscale such that detection should not be able to rely on background colours. It is applied by adjusting saturation with a factor between 0.5 and 1.5 and hue with a factor between -0.5 and 0.5. These different augmentations, which are shown in Figure 9, are applied to the training data with different ratios. Testing both with 10% and 50% augmented data allows us to better view the impact of the augmentations. Only applying the augmentation to 10% of the data is not commonly done, but will allow us to view gradual changes in the results. Not all data is augmented as the model still needs to be able to detect objects with regular vision.

Non-augmented | Gaussian blurring | Brightness | Colour jitter

Figure 9: Various results that can be obtained after augmenting the data with the type of augmentation being displayed below the image.

4 Results

This section presents the results of various optimisation techniques, including both data augmentation and the implementation of the closed-form matrix as described in Kasarla et al. (2022). The former illustrates performance benchmarks achieved when augmenting a chosen subset, while the latter provides more detailed insights into the loss values obtained after training.

4.1 Splits and Metrics

A 70%, 10% and 20% split has been applied on the dataset, resulting into subsets of training, validation and testing respectively using random sampling. The ratio of this split was chosen as the dataset is not very large, which would cause high variance in the training data. High variance is also known as overfitting, and can occur when the model fits too closely to the data, making it unable to generalise to new data. This phenomenon is more likely to happen when there is a split with more training data and less testing data. To combat this, a smaller percentage of the training data is taken than usual, which is then used for testing. This ensures that the model generalises well to unseen data, combatting overfitting.

As explained in subsection 3.2, we measure the model using 4 metrics: precision, recall, mAP50 and mAP50-95. The first metric, precision, measures the percentage of correct positive detections, while recall measures the ability for the model to detect all relevant cases. This means recall is affected by false negatives, rather than false positives (Padilla, Netto, and Da Silva 2020). The mAP is the average detection precision over all object categories (Diwan, Anirudh, and Tembhurne 2023). This mAP is taken over an IoU threshold measuring the overlap between the predicted and ground-truth bounding box, before classifying a detection as (in)correct (Padilla, Netto, and Da Silva 2020).

Using these metrics, a benchmark of the base YOLOv8n model can be compared against the model with changes applied, such as the matrix multiplication or data augmentation. This way, the effects of the changes on the different metrics can be viewed, which determines the in- or decrease in performance of the model. Most likely, there will be a trade-off that has to happen if one metric (i.e. precision) increases while another metric (i.e. recall) decreases. Whichever metric is more important is dependant on the problem it is trying to solve.

4.2 Maximum Class Separation in YOLOv8n

Implementing the maximum class separation matrix on the YOLOv8n model yielded lower results than expected. Table 2 displays the results obtained by training a regular YOLOv8n model on our custom dataset. Based on the findings in the paper and the distribution of our data, we expected a slight to moderate increase in the accuracy of the model.

Class	Images	Instances	P	R	mAP50	mAP50-95
all	1516	3390	0.949	0.93	0.974	0.724
ball	1516	997	0.97	0.934	0.979	0.816
robot	1516	1457	0.89	0.891	0.955	0.789
goal_post	1516	579	0.964	0.94	0.98	0.622
pen_spot	1516	357	0.97	0.955	0.98	0.672

Table 2: Average results of training YOLOv8n for 50 epochs.

Instead of an increase, implementation of the matrix resulted in a significant decrease across all performance metrics. Table 3 shows the results per class, while Table 4 shows a comparison of the results with and without applying the maximum class separation. Precision decreases the least, with around 15.3%, while recall decreased the most with nearly 50%. Lastly, both metrics that measure mAP decreased with more than 30%.

Class	Images	Instances	P	R	mAP50	mAP50-95
all	1516	3390	0.796	0.444	0.591	0.403
ball	1516	997	0.844	0.785	0.836	0.653
robot	1516	1457	0.765	0.568	0.695	0.503
goal_post	1516	579	1.0	0.0	0.387	0.186
pen_spot	1516	357	0.576	0.423	0.449	0.27

Table 3: Results of training YOLOv8n with maximum class separation applied for 200 epochs.

Class	Images	Instances	P	R	mAP50	mAP50-95
all (Without MCS)	1516	3390	0.949	0.93	0.974	0.724
all (With MCS)	1516	3390	0.796	0.444	0.591	0.403
			-15.3%	-48.6%	-38.3%	-32.1%

Table 4: Comparison of results when training YOLOv8n with or without maximum class separation applied for 200 epochs.

A possible cause for these results can be better explained through a visualisation of the loss and model metrics throughout the training process. Figure 10 displays the decrease in loss and increase in model metrics during the 50 epochs the YOLOv8n model was trained on without implementing the maximum class separation matrix. The graphs display the total elapsed epochs on the x-axis and the different losses or performance metrics on the y-axis. In just 50 epochs, the model managed to decrease the loss in validation and training to a value below 1, while also increasing the precision, recall and mAP-50 to 90% and above. The reason YOLOv8n performs so well on a custom dataset with solely 50 epochs is because the detection model is pre-trained on the COCO[8] dataset. This pre-training primes the model for object detection and contains a good foundation with regards to the features it will detect.

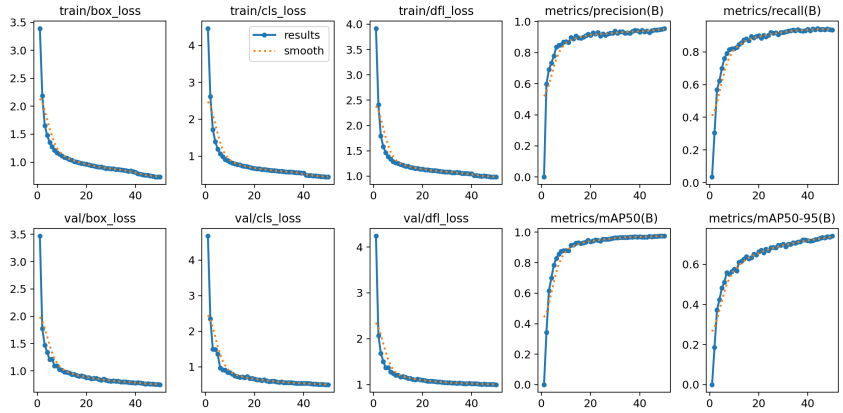


Figure 10: Average results of training YOLOv8n for 50 epochs. Displayed on the x-axis are the total elapsed epochs and displayed on the y-axis are the different losses or performance metrics.

Figure 11 shows the graphs for the training process of the YOLOv8n model with the maximum class separation matrix added. The largest change was expected to occur in the classification loss (denoted as *cls_loss* in Figure 10 and Figure 11), since the maximum class separation is expected to affect this metric the most. This indeed displayed the most diverging behaviour compared to training the base YOLOv8n, as it decreased rapidly during training. We tested this with different seeds, and consistently, the classification loss started out around 7000 to 15000, rapidly decreasing to around 2000, after which it gradually kept decreasing at a slower rate. To test the potential of the model, the model was ran for 200 epochs, resulting in a classification loss of approximately 900 after training.

[8] `https://cocodataset.org`

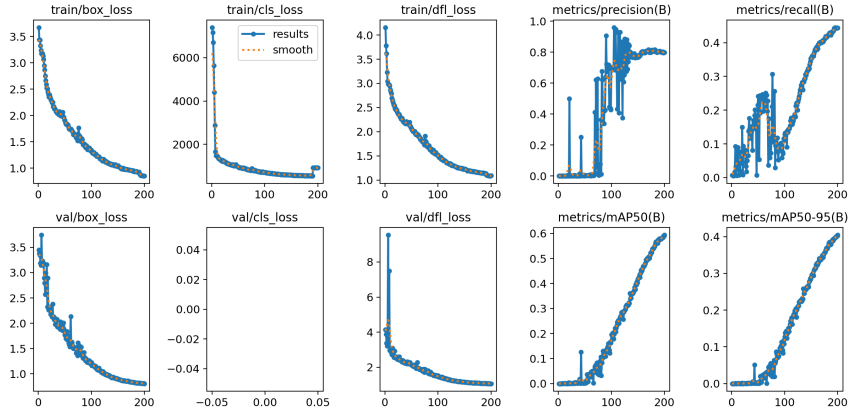


Figure 11: Results of training YOLOv8n with Maximum Separation Matrix for 200 epochs.

The question then shifts to understanding why classification loss experiences a significant decrease after applying the matrix. A possible answer lies in the specific way that maximum class separation needs to be implemented for YOLOv8. After meeting with one of the authors of the paper by Kasarla et al. (2022), it was suggested that in order to obtain the correct output dimensions from the model for applying the maximum class separation matrix, a constraint has to be put on the model. This can be done by removing any column of choice of the output tensor containing the class logits, as the model would regain this column when applying the maximum class separation matrix. While this is correct, it did not take into account that YOLOv8 detection is pre-trained on the COCO dataset. When eliminating the column, the model explicitly discarded all the pre-trained weights. This led to the huge increase in the classification loss, since the YOLO model had to be trained from scratch. Therefore, it will take significant additional research to acquire the optimal hyperparameters for training the YOLO model with the maximum class separation matrix added. Additionally, a vast amount of computational power is needed to train the model from scratch, considering the amount of epochs it will likely need in order to surpass the regular model in accuracy.

4.3 Data Augmentation

In order to improve the model performance, parts of the dataset were augmented with different augmentations. After applying the augmentations, they were tested by taking the average of 3 runs with 10 epochs. The performance of the model on each of the augmentations for a combination of all classes is displayed below, along with an explanation and evaluation of these results.

	P	R	mAP50	mAP50-95
Non-augmented	0.892	0.809	0.887	0.584
10% augmented	0.888	0.808	0.874	0.592
	-0.448%	-0.124%	-1.466%	+**1.37%**
50% augmented	0.833	0.808	0.879	0.596
	-6.614%	-0.124%	-0.902%	+**2.055%**

Table 5: Combined results of adjusting brightness with a factor between 0.5 and 1.5 on all classes.

Table 5 shows that augmenting the data with brightness only causes an increase in the mean average precision taken over the IoU threshold between 50% and 95%. This continues to increase as more data is augmented, having a 1.37% increase when 10% of the data is augmented, and a 2.055% increase when 50% of the data is augmented. However, the precision significantly decreased as a result of the data augmentations, going from a 0.448% decrease to a 6.614% decrease as the ratio of augmented data increased. While the recall and mAP50 metrics were not significantly affected, the substantial decrease in precision remains a cause for concern. As a result, it will most

likely not be a good idea to augment data with only brightness in order to improve the model performance.

	P	R	mAP50	mAP50-95
Non-augmented	0.892	0.809	0.887	0.584
10% augmented	0.889	0.786	0.869	0.579
	-0.336%	-2.843%	-2.029%	-0.856%
50% augmented	0.905	0.784	0.873	0.578
	+**1.457%**	-3.09%	-1.578%	-1.027%

Table 6: Combined results of applying colour jitter on all classes.

Table 6 shows the results of augmenting the data with colour jitter, which was an alteration of the saturation and hue of the images. When only 10% of the data is augmented, all metrics show a decrease in performance. However, when augmenting 50% of the data, the precision shows an increase of 1.457%. Then again, with a bigger decrease in the other metrics, only augmenting data with colour jitter is not a viable option to improve the model performance.

	P	R	mAP50	mAP50-95
Non-augmented	0.892	0.809	0.887	0.584
10% augmented	0.859	0.799	0.835	0.566
	-3.699%	-1.236%	-5.862%	-3.082%
50% augmented	0.905	0.803	0.882	0.593
	+**1.457%**	-0.741%	-0.564%	+**1.541%**

Table 7: Combined results of adding gaussian blurring on all classes.

Lastly, Table 7 shows the performance of the model after images are augmented using gaussian blurring. As with the previous augmentation, all metrics show a decrease in performance when only 10% of the data is augmented. When 50% of the data is affected by gaussian blurring, the performance of both precision and mAP50-95 increase with 1.457% and 1.541% respectively. Since the recall and mAP50 do not decrease with more than 1%, it is to say that augmenting using gaussian blurring is a good option to increase performance of the model. A further elaboration and reflection on the results will be discussed in the next section.

5 Discussion

5.1 Maximum Class Separation in YOLOv8n

Application of the maximum separation optimisation discussed in the paper by Kasarla et al. (2022) has not yielded satisfactory results due to a number of reasons.

First of all, the paper describes the implementation used for a network implementing classification, which differs greatly in architecture from a network implementing detection. The output of such a classification network is usually a low dimensional tensor of shape $D \times C$, with C being the amount of classes present in the dataset and D being the amount of features. The output of a YOLOv8 detection network, as described in section 3, consists of a four-dimensional tensor, which not only contains data about the class logits, but also the bounding boxes for possible detected objects. Attempting to dissect this tensor and extract the class logits that were needed proved difficult and made it uncertain whether this was done correctly. It is possible that this process is less difficult for two-stage detectors, which perform localisation and classification separately, since the separation of these tasks will simplify the process of extracting the class logits.

Secondly, both the complexity of the YOLO architecture and abstract nature of the YOLO codebase made it difficult to pinpoint the exact location where to apply the matrix for maximum separation. Attempts have been made to implement the maximum separation in alternate parts of the YOLO architecture, such as immediately after the backbone or just before the final detection layers, but these endeavours have not been successful. More experimentation could prove to useful.

Lastly, it is also possible that the pre-trained weight of YOLO models, tasked with object detection, could cause the implementation of maximum class separation to result in unsatisfactory results, since the pre-trained model might not be able to deal well with an architectural change such as adding maximum class separation.

Due to these complications, application of the maximum separation technique stated in Kasarla et al. (2022) did not yield satisfactory results within the given time frame. We acknowledge that the implementation of maximum separation for detection using YOLO is neither shown to be impossible, nor is it deemed disadvantageous for the model performance as the implementation may be possible with more research and time.

Another possible step for further development is to research methods for applying the maximum class separation matrix in alternative parts of the YOLOv8 architecture, since this could lead to better performance.

Additionally, it might be possible to acquire a YOLOv8 detection model which has no pre-trained weights. This could be beneficial for performance, as the model can be trained from scratch with a customised architecture. However, this would require finding the optimal hyperparameters for training the YOLOv8 model, which could pose to be difficult.

Finally, future detection models may lend themselves well to architectural alterations, which could make the process of implementing maximum class separation more straightforward.

5.2 Data Augmentation

In subsection 4.3, we showed the results of the data augmentation research done on the YOLOv8n model. It showed increases in accuracy for only a quarter to half of the metrics, while usually showing a higher decrease of accuracy on the different metrics. This meant we did not move forward with the dataset augmentations while doing the research on the implementation of the paper. However, gaussian blurring on a large percentage of the dataset did show promising results. Therefore, it would be worth looking into whether augmenting the data with a higher ratio would provide even better results.

Using data augmentation to optimise a YOLO model is very usable, as it can be applied to any visual dataset before training. On the other hand, a constraint of this solution is that different augmentations give very different results as ratio and type of augmentation are only applicable to certain purposes. After all, it is less desirable to use brightness augmentations with a large ratio if the goal is increasing precision, just as it is less desirable to apply gaussian blurring or colour jitter augmentations to only 10% of the data if the goal is increasing overall accuracy.

On the topic of the generalisability of the solution, using data augmentation to optimise object detection performance is very generalisable to different models and even to different problems. As

the research was only done on a YOLOv8n model, it is difficult to predict whether augmenting the data before training with a different model, such as ResNet or AlexNet, will show similar results. However, data augmentation is a very common approach to improve model performance, so it is reasonable to think that the same metrics will improve or deteriorate after augmenting the data. Furthermore, since object detection is partly a classification problem, it would be logical that the results can be reproduced in a classification model as well.

Lastly, there are a significant amount of improvements to be made to the research. First of all, since the results showed improvements in certain metrics as the ratio of augmented data increased, it could be valuable to research whether an even higher ratio, such as 100%, would warrant even better results. In the case of gaussian blurring, a decrease in recall and mAP50 becomes so minimal that augmentation becomes a very viable optimisation. Secondly, a combination of different augmentations has not been done, so it is unknown whether combining augmentations has positive or adverse effects. Since augmenting with brightness decreased precision, but gaussian blurring increased the metric, it would be interesting to know how precision changes after applying both of these augmentations. As both gaussian blurring and brightness augmentations both increased the mAP50-95 metric, a combination might see an even higher increase. Additionally, the research only covered results obtained after training the model on 10 epochs. This is a relatively low amount of epochs to train with, as training with a higher number of epochs might make the decreases in metrics disappear or make the obtained results insignificant.

6 Conclusion

This project set out to experiment with optimisation methods targeting performance improvements of the YOLOv8 model tasked with object detection. In particular, research has been conducted on the feasibility of maximum class separation on the YOLOv8n model and the effects of data augmentation as a way to imitate real world conditions. Our aim is to determine to what extent the YOLOv8 model can be enhanced without significantly impacting runtime speed.

The maximum class separation was based on the research done by Kasarla et al. (2022), where the authors achieved promising results which showed that applying a fixed maximum class separation matrix to the class logits of a classification model could improve precision and recall. In order to apply this technique to YOLOv8, research has been conducted into the inner workings of the YOLOv8 architecture, before deciding to apply the fixed matrix in the three detection heads of the model. Testing the YOLOv8n model with maximum class separation applied, resulted in a particularly high class loss along with a below baseline accuracy, compared to the base YOLOv8 model. This could be due to the pre-trained weights of YOLO models, as the pre-trained model might not be able to deal well with architectural change such as embedding maximum class separation.

The data augmentation included augmenting a portion of the input images with either gaussian blurring, brightness adjustments or color jitter, simulating different extreme environmental factors during gameplay. Outcomes have shown that it is only a viable optimisation when applying gaussian blurring to more than half of the data, even though not all performance metrics increased following this augmentation. Both colour jitter and brightness adjustments showed a decrease in accuracy of the model regardless of the ratio of data that was augmented.

Further research could explore finding the optimal hyperparameters for training the YOLOv8 model from scratch with the fixed matrix applied, and comparing it to a base YOLOv8 model that has not been pre-trained. Moreover, supplementary research regarding data augmentation in the context of YOLO is necessary to uncover strategies for optimisation without compromising accuracy in any of the performance metrics.

To conclude, the work that has been documented in this report shows that YOLOv8, being an object detection model, does not lend itself well for easy integration of the proposed optimisation technique detailed in the aforementioned paper. Due to the previously mentioned complications, it is therefore neither shown nor disproven that YOLO can be optimised with maximum class separation.

References

Battaglia, Peter et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: *arXiv*. URL: https://arxiv.org/pdf/1806.01261.pdf.

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine learning* 20, pp. 273–297.

Diwan, Tausif, G Anirudh, and Jitendra V Tembhurne (2023). "Object detection using YOLO: Challenges, architectural successors, datasets and applications". In: *multimedia Tools and Applications* 82.6, pp. 9243–9275.

Guo, Jian and Stephen Gould (2015). "Deep CNN ensemble with data augmentation for object detection". In: *arXiv preprint arXiv:1506.07224*.

Hussain, Muhammad (2023). "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection". In: *Machines* 11.7, p. 677.

Kasarla, Tejaswi et al. (2022). "Maximum class separation as inductive bias in one matrix". In: *Advances in Neural Information Processing Systems* 35, pp. 19553–19566.

Li, Bolian et al. (June 2022). "Trustworthy Long-Tailed Classification". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6970–6979.

Padilla, Rafael, Sergio L Netto, and Eduardo AB Da Silva (2020). "A survey on performance metrics for object-detection algorithms". In: *2020 international conference on systems, signals and image processing (IWSSIP)*. IEEE, pp. 237–242.

Redmon, Joseph et al. (2016). "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788.

user1551, (https://math.stackexchange.com/users/1551/user1551) (2014). *How to find $n+1$ equidistant vectors on an n-sphere?* Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/714781.

Yin, Yunhua, Huifang Li, and Wei Fu (2020). "Faster-YOLO: An accurate and faster object detection method". In: *Digital Signal Processing* 102, p. 102756. ISSN: 1051-2004. DOI: https://doi.org/10.1016/j.dsp.2020.102756. URL: https://www.sciencedirect.com/science/article/pii/S1051200420301019.

Zoph, Barret et al. (2020). "Learning data augmentation strategies for object detection". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVII 16*. Springer, pp. 566–583.