Using Geometric Gradient Analysis to detect Out of Distribution Data for YOLO

Abel J. Oakley

Layout: typeset by the author using LaTeX.
Cover illustration: Abel J. Oakley

Using Geometric Gradient Analysis to detect Out of Distribution Data for YOLO

a smaller subtitle

Abel J. Oakley
10653333

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
Hidde

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

Semester 2, 2022

Contents

Chapter 1

Introduction

1.1 Artificial Intelligence and Object Detection

We are swiftly heading towards a world where our daily lives are run by Artificial Intelligent systems. Instead of humans regulating and securing safety-related systems (e.g. life support systems, railway signals or air traffic control), computer algorithms have taken over these tasks (Knight, 2002). Moreover, new critical systems have emerged with a certain complexity only a computer can efficiently operate (e.g. a nuclear reactor fail-safe, car brake systems). These *safety-critical systems* are also called *life-critical systems* because when they fail or malfunction, the consequence will most likely be death or serious injury to people. Therefore, such a system must be highly reliable.

Imagine an object detection algorithm of a self driving car that detects objects that appear in front of the car. We can teach this algorithm to detect whether the object in front is a horse or seagull [1] (figure 1.1 has an example of objects detected by an object detection algorithm for cars). Both objects require a different choice of operations. In case of the horse it most definitely must brake to avoid damage to the car and driver, in contrast to the seagull which will either fly away or get hit, causing no damage to the driver or car. In this case the *critical system* is the instruction set the algorithm has for each specific detected object. However, what if the algorithm is confident it has detected a specific object, but this prediction is wrong? This might be because the algorithm is not correctly trained, or the detection algorithm detects out of distribution data.

[1] https://i.ytimg.com/vi/WZmSMkK9VuA/hqdefault.jpg

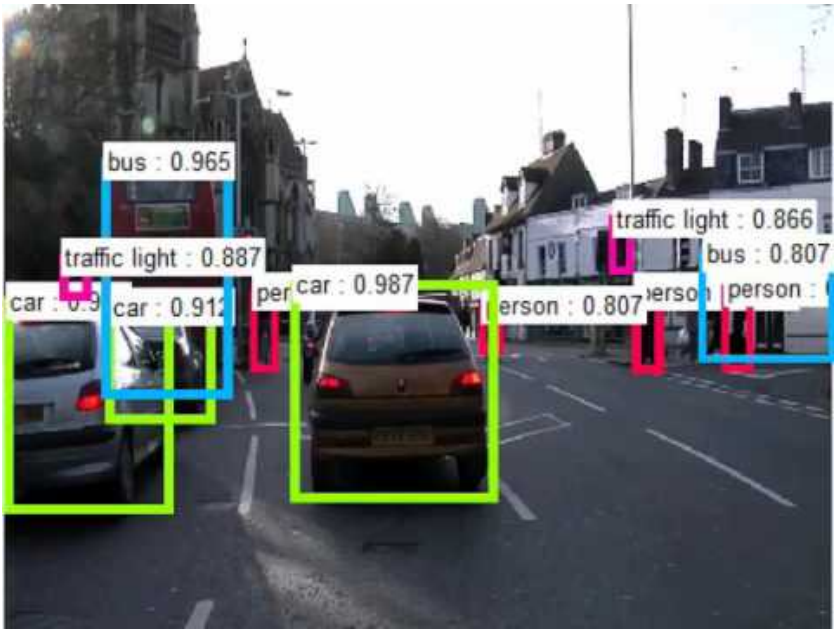


Figure 1.1: Object detection for cars

These object detection problems are easy to solve by standard AI systems. However, there is a significant problem when it comes to out of distribution data. Imagine that the car detects an object that it has never seen before, for example a panda. Now the detection algorithm will refer to all the object classes it has seen (trained on), and tries to fit the new object into one of these classes. This could have problematic implications towards it's decision making capabilities. The new detected object could be classified as a seagull. The decision the car will make is now, just drive ahead because the seagull will fly away. However, the panda will not fly away and the car will crash, causing serious harm to the driver. To circumvent this, a system is needed that can detect whenever the object detection algorithm detects an object that it has never seen before. And with this implement a new instruction set that has causes less harm to the driver.

1.2 Real world Critical System failures

In 2018 a critical system failed where a self driving car wasn't able to recognise a female pedestrian who was pushing a bicycle across a four-lane road, who revealed by sources, was jay-walking [2]. The car was not able to recognise the pedestrian as a person, but instead was switching between the classifications "vehicle, bicycle and an other". Therefore the car wasn't able to predict a correct path. The driver who was present for safety also couldn't circumvent the crash, which resulted the death of the pedestrian. Not only can failures of these systems harm others, it can also result in the death of the driver itself.

A recent article in the New York Times describes that nearly 400 crashes have occured in the United States in 10 months involving driver assisted cars. Accumulating six deaths and five serious injuries[3]. A popular point of discussion is that in Arizona, driverless taxis are tested, which means that all the control is in the hands of an AI.

In most cases, the court ruled the drivers as responsible for the accident, and not the *critical-systems* that failed. However, it is highly likely that in the future these drivers will not be present, leaving the self driving car alone in it's decision making. These accidents give rise to ethical questions who to blame in this situation. Therefore it is important to prevent these accidents by improving the reliability of these *critical-systems*. The next section discusses a probable solution.

[2]https://www.nbcnews.com/tech/tech-news/self-driving-uber-car-hit-killed-woman-did-not-recognize-n1079281

[3]https://www.nytimes.com/2022/06/15/business/self-driving-car-nhtsa-crash-data.html

1.3 The Geometric Gradient Analysis

Neural Networks have been used to achieve great results and accuracy when the distribution of the training and test data are similar. Nonetheless, when the model is used in real-world scenarios, different distributions of data may be observed that are not contained in the training set.

A study in the trustworthyness of object detection using geometric gradient analysis (Schwinn et al., 2021), has positive results regarding the detection of out of distribution data. Using *saliency maps* and *correlation matrices*, similarity between certain object classes can be measured. Thus, when an out of distribution object is detected, a low similarity across all known object classes will be measured. This can be used as a red flag to let an object detection algorithm know that it is observing an object class that is not yet learned. Moreover, most other methods used, require a retraining of the used model for more robust classification of out of distribution data. This is not the case with the geometric gradient analysis, which only looks at the geometry of the loss landscape based on the saliency maps of the input.

This work focuses on the object detection of robots playing football in the dutch Robocop, using the object detection algorithm You Only Look Once, or YOLO. Is it possible to use geometric gradient analysis described in the paper to detect out of distribution data and therefore, improve the confidence of the Yolo algorithm?

Not only OOD can form a problem for correctly detecting objects. Objects can have other complications i.e. the image can be an adversial example or the image can have random noise. A relevant sub question can therefore be stated as; can GGA be used to improve object classification on adversial examples or images with noise?

In the next chapter we will go in depth on all the tools needed to answer these questions. Additionally we will explain the functionality of the YOLO algorithm.

Chapter 2

Background

2.1 Robocup

The object detection algorithm used for this thesis focuses on the RoboCup. The Robot Soccer World Cup is a yearly international competition founded by a group of university professors to create more interest around robotics and AI. Moreover, the competition pushes advancements in technology surrounding AI and robotics. The University of Amsterdam has it's own Robot Soccer team named, Dutch NAO Team (see figure 2.1).

The robots playing soccer are fully autonomous and therefore require specific AI. They are able to detect sound (e.g. to hear the referee whistling for a foul) and to detect objects (i.e. to be aware of the location of the ball, or to avoid another player). This thesis focuses on the object detection which is done by YOLO.

Figure 2.1: Match of the Dutch NAO Team

2.2 You Only look Once

To understand the geometric gradient analysis and how it is applied to YOLO, we first need to explain a few important subjects. We briefly explain YOLO, basic convolution and final the geometric gradient analysis.

This section discusses real-time object detection with the use of, You Only Look Once, or YOLO. Within the field of computer vision, object detection has been one of the classical problems. Object detection focuses on two things; the detection of an object and the classification of the detected object. We follow the Thesis Thuan (2021), which goes in detail on how YOLO works and has evolved in the years.

The aim of YOLO is to segment an image into a grid of size NxN (in figure 2.2 a 3 by 3 grid is shown). Then, if the centre of an object falls into one of these cells, that grid will focus on detecting that object. This implies that objects with overlapping bounding boxes will still be detected by different grid cells.

Figure 2.2: 3 x 3 Grid Cells

Object detection is realised by having each cell in the grid predict the bounding boxes with their related confident scores. The bounding box contains 4 parameters $(x_{cood}, y_{cood}, w, h)$, as seen in figure 2.3. Where the values x_{cood}, y_{cood} (in red) are the coordinates of the centre and w, h (in blue) are the width and height of the bounding box. Each class in a grid also has a predicted probability described as $p(Class_i|Object)$.

Figure 2.3: Detected Robot with corresponding vector values

All of these parameters create the following vector,

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \end{bmatrix} \tag{2.1}$$

Here p_c is the confident score, b_x, b_y, b_h, b_w describe the bounding box and c_1 and c_2 are the class predictions.

For the YOLO algorithm, in the case of this thesis, the classes are the robot or the ball. Since each grid cell is responsible for detecting an object with it's corresponding center that is in that specific grid cell, multiple objects and different objects can have a center inside that grid cell. Therefore, multiple vectors are created each with a corresponding $(x_{cood}, y_{cood}, w, h)$, and a classification c_1, c_2. In figure 2.3 an image is shown with multiple objects and their corresponding grid cell will create three vectors.

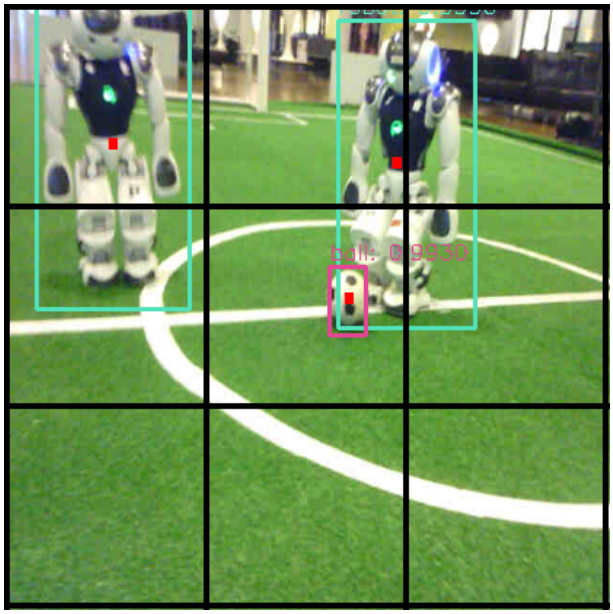


Figure 2.4: Two robots and a Ball

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \end{bmatrix} \tag{2.2}$$

In this situation the three vectors have $p_c = 1$ which corresponds to the confidence score. The first and the last vector have $(c_1 = 0)$ and $(c_2 = 1)$, which means that the predicted class is a robot. For the middle vector $c_1 = 1$ and $c_2 = 0$, where the predicted class is a ball.

2.2.1 The Neural Network Architecture

Yolo uses the DarkNet Architecture to process all image features,(Thuan, 2021), followed by fully connected layers performing bounding predictions for objects.

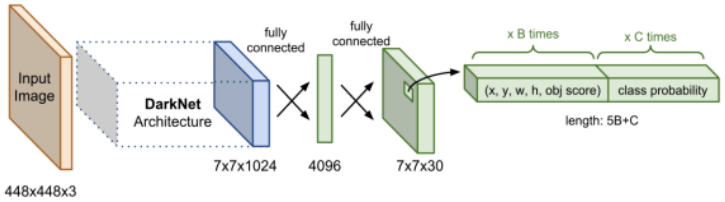


Figure 2.5: YOLOv1 architecture

2.2.2 Convolutions with anchor box

To predict more than one object inside the same grid cell, Yolo uses an architecture to predict bounding boxes instead of using fully connected layers. An anchor box is a list of predefined boxes matching the detected objects. These bounding boxes are predicted based on ground truth boxes and predefined k anchor boxes.

2.2.3 Loss Function of the YOLO algorithm

In this section we will briefly explain the loss function used for training networks in Yolo. During training of a neural net, the model needs to determine the error based on the current predictions, such that it can perform a gradient descent. Therefor a loss function is needed. Important to note is, that there will be images where their respective grid cells don't contain any objects with their corresponding confidence score of zero. Avoiding this, which can lead to training divergence and model instability (Thuan, 2021), Yolo places a high penalty for predictions for predictions conainting objects and a low penalty for predictions with no object. The loss function is calculated by taking the sum of the bounding box parameters from the vector $(x_{cood}, y_{cood}, w, h, confidencescore, classprobability)$

$$
\begin{aligned}
&[1]\lambda_{cood}\sum_{i=0}^{s^2}\sum_{j=0}^{B}\mathbb{I}_{i,j}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]+\\
&[2]\lambda_{cood}\sum_{i=0}^{s^2}\sum_{j=0}^{B}\mathbb{I}_{i,j}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]+\\
&[3]\sum_{i=0}^{s^2}\sum_{j=0}^{B}\mathbb{I}_{i,j}^{obj}(C_i-\hat{C}_i)^2+\\
&[4]\lambda_{noobj}\sum_{i=0}^{s^2}\sum_{j=0}^{B}\mathbb{I}_{i,j}^{obj}(C_i-\hat{C}_i)^2+\\
&[5]\sum_{i=0}^{s^2}\mathbb{I}_{i}^{obj}\sum_{c\epsilon classes}(p_i(c)-\hat{p}_i(c))^2
\end{aligned}
\tag{2.3}
$$

1. Computes the loss related to the predicted and ground truth bounding box position with their respective center (x_c, y_c). $\mathbb{I}_{i,j}^{obj}$, where this is 1 if the object is inside the j^{th} predicted bounding box in i^{th} cell, else it is 0.

2. The predicted bounding box will focus on prediction an object based on the prediction with the highest IOU with the ground truth.

3. This computes the loss of confidence score combining both cases where the object is either in the bounding box or not.

4. Combined with [4]

5. This computes the loss of class probability.

2.3 General Machine Learning Methods

2.3.1 Convolutional Neural Networks (CNN)

The architecture of neural networks used for object detection usually are Convolutional Neural Networks. These neural networks utilises convolutions inside convolutional layers (Albawi et al., 2017). A basic 2D convolution between an image and a filter is defined as follows.

$$L[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} h[m,n] * x[i-m, j-n]$$

2.3.2 Forward Pass

Each unit inside a neural network computes two operations: the computation of the weighted sum and to process the sum through an activation function(Liu et al., 2015). The most general case of a forward pass is given in the equation below, with a single layered, densely connected network. x is defined as the single sample input with corresponding features $[X_1, ..., X_n]$ and w_i are the weights matrices of the layers. Lastly f denotes the activation function.

$$y = f(x * w_1) * w_2$$

2.3.3 Max Pooling

Max pooling is a technique used to reduce the dimension of images, by reducing the amount of pixels from the previous layer. This is achieved by calculating the maximum value for regions in a feature map, followed by creating a pooled feature map. The figure below shows how a 4 x 4 matrix is reduced to a 2 x 2 matrix with the use of max pooling. The function calculates the maximum value in each square, denoted by colour, and puts those values inside a new reduced matrix.

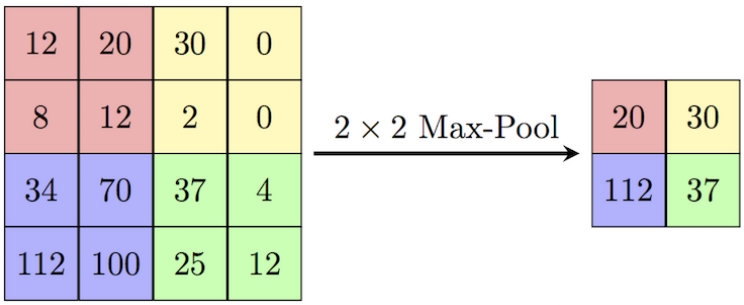


Figure 2.6: 2 x 2 Max Pool

2.3.4 Bounding Boxes and Intersection over Union

An image that contains objects will have corresponding bounding boxes for those objects, see figure 2.7. Since the same object can have multiple bounding boxes, a method is introduced to generate the optimal bounding box for that object.

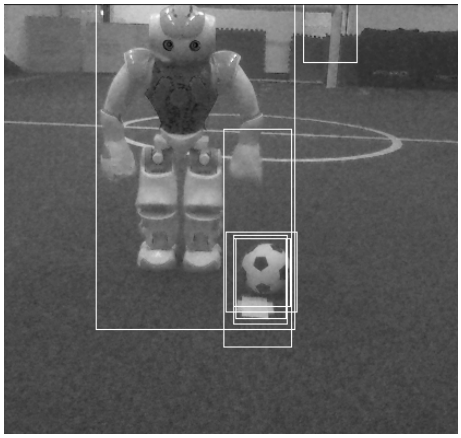


Figure 2.7: Bounding Boxes

For this we use the IoU. The IoU has two functions here:

- When trying to find the accuracy of a model, the IoU can see how well the predicted bounding box overlaps with the ground truth bounding box.
- When there are multiple predicted bounding boxes in an image, the most accurate bounding box can be found while removing the other bounding boxes.

A better IoU correlates to a better predicted bounding box. Figure 2.8 shows three scenarios of IoU with respect to their accuracy.

Figure 2.8: IoU

2.4 The Geometric Gradient Analysis

Here we describe the Geometric Gradient Analysis and it's vital components. Where the general idea is reconstructed such that it works with the Yolo model used for detecting the Robot and Ball. The GGA has two major components, the saliency map and the cosine similarity matrix. We explain both and in final use that to explain the GGA.

Saliency Maps

A saliency is a way of showing what is important in an image (Simonyan et al., 2013). A neural network can put more weight on specific pixels in an image, which therefor has a large influence on the prediction of the neural network.

To generate a saliency map, we calculate the gradient of output category with respect to an input image. We can use this information to see how the output category value changes with respect to small changes in the input image pixels. The positive values in the gradients give information that a small change to that specific pixel will increase the output value. When doing this for every pixel, we can plot an image where the pixels who have a big influence are drawn with a light colour, in contrast to pixels with no influence, seen in figure 2.9.

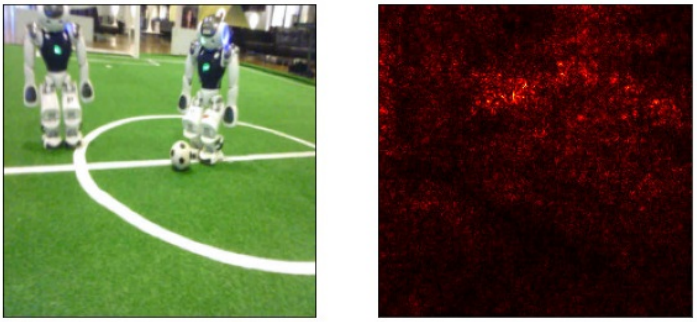


Figure 2.9: Saliency Map

To create a saliency map, we compute the gradient of output category in respect to an input image,

$$\frac{\delta output}{\delta input}$$

This can be used to highlight salient image regions that influence the output class the most, removing the 'black box' of the model. The saliency map with respective output class will be used in the next segment, the cosine similarity matrix.

2.4.1 Cosine Similarity Matrix

In this segment we use saliency maps with their corresponding output class to create cosine similarity matrices. A cosine similarity matrix $CSM = CSM(x)$ for any given sample x is defined as,

$$CSM = (c_{ij}) \epsilon R^{CxC}, c_{ij} = \frac{s_i * s_j}{|s_i||s_j|}$$

here $i, j \ \epsilon \ \{1, ..., C\}$ and c_{ij} produce the cosine similarity between two saliency maps s_i and s_j. The figure below shows the result of two saliency maps with their corresponding CSM. In figure a CSM is created for two classes, Ball and Robot.

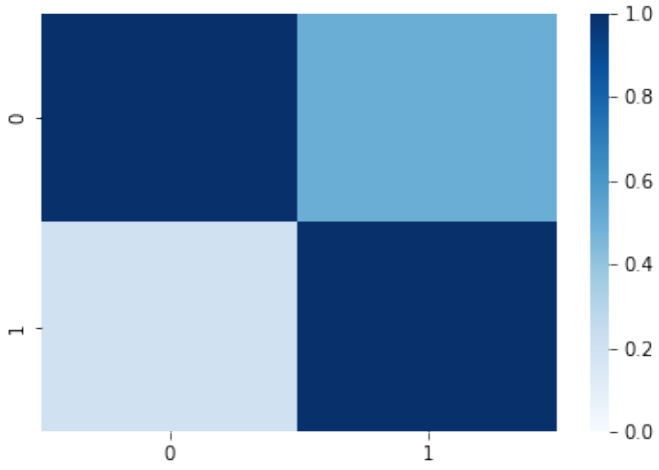


Figure 2.10: Cosine Similarity Matrix

2.4.2 Geometric Gradient

In this section we describe the geometric gradient analysis and incorporate it with the yolo model. The geometric gradient analysis uses a pair (x, y) consisting of an input sample $x, \in R^d$ with the corresponding class label $y \in \{1, ..., C\}$ in a supervised classification task (Schwinn et al., 2021). Since YOLO is able to detect multiple objects in one image, this needs some rewriting. Now the input sample x will be the same, however y may contain multiple detected objects in the form,

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \end{bmatrix} \tag{2.4}$$

Moreover, there are only two classes that can be detected, either a robot or a ball. This implies that $y \in \{0, 1\}$ where 0 stands for the class ball and 1 for the class robot. Now we have the neural net F_θ with the parameter vector $\theta \in \Theta$ and with k the class predicted by the neural network given the image sample x. The loss function for each predicted class in the image is defined as $L(F_\theta(x), y)$. The saliency map $s_i(x) \in R^d$, with ith class for a given sample x and detected object $k \in \{1, .., Objects\}$, we define the cosine similarity matrix as,

$$CSM = (c_{ij}) \in R^{CxC}, c_{ij} = \frac{s_i * s_j}{|s_i||s_j|}$$

where $i, j \in \{1, ..., C\}$ and c_ij represent the cosine similarity maps between two saliency maps s_i and s_j.

Whenever YOLO observes OOD data and wrongly classifies it, the GGA can observe the geometric relations between the saliency maps of the output classes. The geometric relation of the OOD will be different than in-distribution data, which then can be used to classify the detected object as OOD.

Chapter 3

Method

3.1 Preprocessing

3.1.1 Transform Images to Tensor

The GGA algorithm requires saliency maps of input images with respect to their output class. To use images we use a function that resize a given image to the desired size 416 x 416. Then it is transformed into a Torch Tensor. The GGA accepts a Tensor containing all input image tensors, therefore we stack the tensors of multiple images into a new tensor with dimension $(X, 3, 416, 416)$. Here the size X are all the images that are included into the tensor, 3 are the channels and 416,416 is the input image size.

3.1.2 Datasets

To test the Geometric Gradient Analysis, we created several different datasets to test on.

- Out of Distribution
- Normal
- Noise Filter
- Rotation
- Gaussian Blur

Each of the datasets have a different impact on how the yolo algorithm classifies objects. In short we explain each method with corresponding impact on the classification.

Out of Distribution

The most important dataset is the Out of Distribution dataset. For this we used three seperate datasets containing images of three different sports; American Football, baseball and women figure skating. The American Football dataset contains 191 images, the baseball dataset 174 and lastly the women figure skating dataset contains 159 images. In figure 3.1 we see a couple of examples of these datasets. These datasets contains no objects of balls or robots so any predicted object class will be incorrect. The dataset has been screened to make sure no robot or ball is present. We suspect that YOLO will perform correctly and will not detect any object as ball or robot in these datasets. However, if YOLO does detect a ball or robot in these images, then those images can be used as OOD for testing the GGA. In the latter case, we can backtrack which images in the OOD dataset have miss classifications, and create a new dataset from these images.

Figure 3.1: OOD dataset containing three different sports

Normal, Noise Filter, Rotation and Gaussian Blur

For these datasets we used a dataset from the paper Yao et al. (2022). This dataset contains 491 images of robots playing football. To create more data we created multiple algorithms that transformed the image into four different images; the original, with noise filter, rotated 90 degrees and a gaussian blur. Examples are shown in the figure below. We tested the YOLO algorithm on all the different datasets to see if the accuracy and confidence changed. This was the case for all datasets, where the noise filter completely removed any classification.

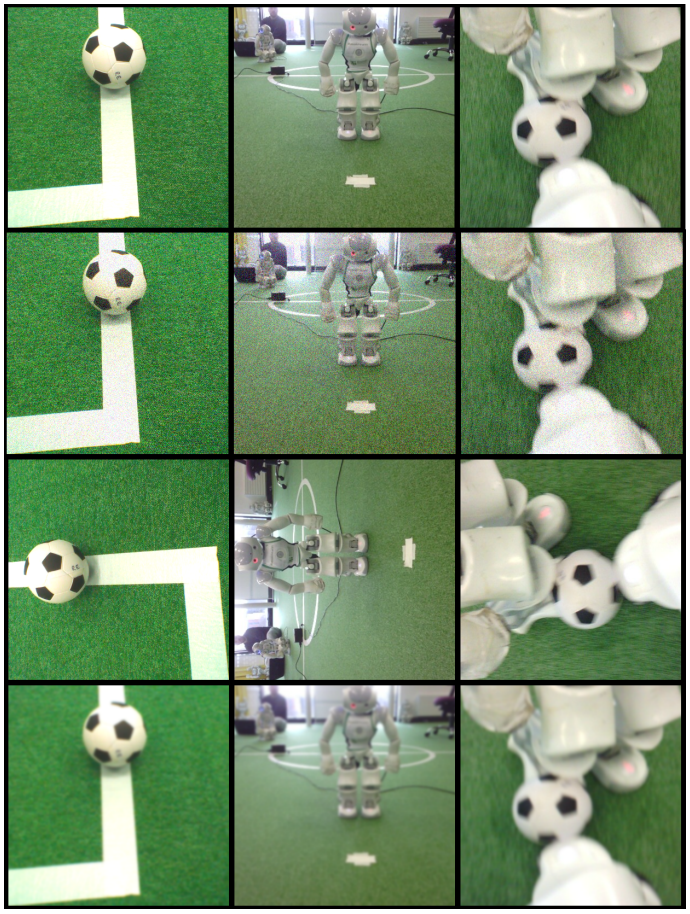


Figure 3.2: From top to bottem, Normal, Noise Filter, Rotation and Gaussian Blur

3.2 Pipeline

In this section we will shortly explain how we are going to use geometric gradient analysis to see if an detected object is out of distribution. In figure 6 we see the pipeline for out experiment. When the input is correctly classified by the model,

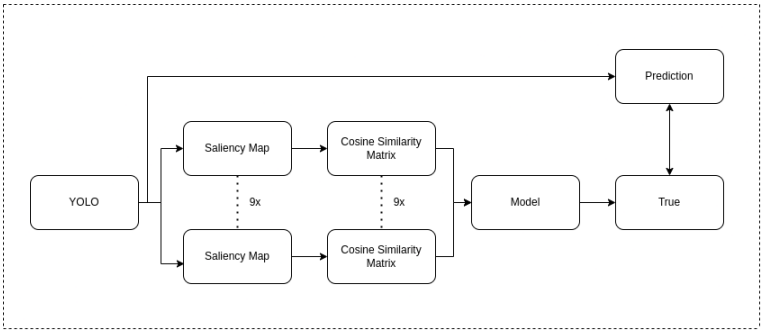


Figure 3.3: The Pipeline for the experiment

the saliency map of a predicted class will point in a direction which is opposite to the saliency maps of all the other classes (Schwinn et al., 2021), in our case the ball or the robot. This will result in an low average cosine similarity between these saliency maps in the rows an columns of the predicted class label seen as in figure 7.

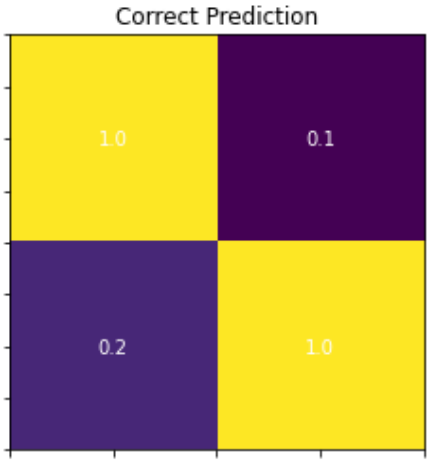


Figure 3.4: correct predicted class

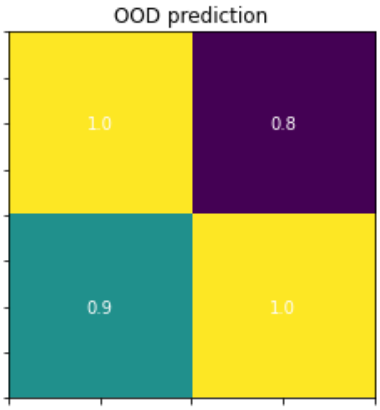


Figure 3.5: Out of Distribution prediction

In contrast to the OOD samples, the saliency maps of the non predicted classes point towards different directions. Therefore the cosine similarity will be lower than average, seen in figure 8. We will use this on the OOD dataset explained in the previous section. We will compare the dataset of the Robocup with the OOD with their respective cosine similarity. If our hypothesis is correct, the GGA detector will observe a difference in it's cosine similarity.

3.3 YOLO Model

We have printed the model of YOLO and added it to the Appendix. Since we did not change anything of the model, or retrained the model, we conclude that the print of the model will suffice.

Chapter 4

Results

4.1 Out of Distribution

To examine if the OOD dataset will suffice to test the GGA, we used the YOLO algorithm to make predictions. For the dataset of each sport, we have observed different miss classifications. Moreover, the amount of miss classifications differs per sport. We will examine each dataset seperatly. In Table 1.1 we see the differences in miss classifications of the datasets.

	Size Dataset	Classification Ball	Classification Robot	Total Images
American F.	191	20	31	35
Baseball	174	10	29	29
Female S. K.	159	0	0	0

4.1.1 American Football OOD

In the American Football dataset we have seen the most miss classifications. In figure 4.1 we clearly see that YOLO incorrectly classifies objects in this dataset as either ball or robot.

Figure 4.1: OOD detection in the American Football dataset

A few observations can be made by reviewing the detected objects in this dataset, and the most important observations can be seen in the previous image. In every miss classification, the ball and robot have two opposite characteristics, the bounding box of the ball is in all the images in this dataset smaller than the bounding box of the robot. When comparing the average width and height using part of the vector, $(x_{cood}, y_{cood}, w, h)$, we see a that the average bounding box of the robot higher. Another interesting observation is that most of the detected miss classified balls, only contain the colours black and white. This is clearly seen in figure 4.1, where four objects classified as balls, are all small and have a black and white colour. This clearly exposes the weakness of training an object detection algorithm in a closed environment, whereas the real world contains numerous objects or part of objects describing the same features as the ball.

4.1.2 Baseball OOD

The results of this dataset has the same observations as the American Football dataset, see figure 4.2. Again we see the relative size between the bounding boxes of each detected class. The only anomaly is the image in the upper left corner, where it classified a human, far away as a ball.

Figure 4.2: OOD detection in the Baseball dataset

4.1.3 Female Figure Skating

This sport was chosen because it had female players, did not have grass in the background and in all the images there was no ball present. This resulted, as seen in the previous table, that no miss classifications occurred.

4.1.4 GGA on OOD

Using the miss classified images we were able to create an OOD dataset of 64 images. However, this was not enough to train the GGA detector. The cosine similarity matrices abstracted from the in distribution set together with the OOD was had no difference from each other. We will further elaborate this in the discussion.

4.2 Normal, Noise Filter, Rotation and Gaussian Blur

We have tested the performance of YOLO on the dataset with Noise Filter, Rotation and a Gaussian Blur. In the table below we see that the noise filter completely removes the ability of YOLO to detect any object. When looking with a human eye, the rotation and Gaussian blur are harder to detect than a simple noise filter. This might indicate that an object detection algorithm is dependent on pixel values and the human eye uses a more continuous observation of the image. Changing pixel values of nearby structures might have a large influence of the detection of objects.

	Classification Ball	Classification Robot	Total Images
Normal	51	64	107
Noise Filter	0	0	0
Rotation	42	10	50
Gaussian Blur	40	33	72

Chapter 5

Discussion

In this section we delve into the importance and relevance of the results found in this paper.

5.1 Susceptibility of YOLO on OOD

As earlier studies have shown, YOLO is great at detecting objects. Therefore we had the expectation that YOLO would perform well with OOD. Moreover we predicted that YOLO would not detect anything, or wouldn't miss classify any objects in the OOD dataset. However, the results have shown the exact opposite. When using the OOD dataset of an American Football match or Baseball match, YOLO was inclined to falsely detect objects and classify them wrongly. Even though, YOLO works well in a closed environment, when introduced to different data, it's effectiveness reduces greatly.

This result should get more attention in the future. Most research is focused on improving YOLO in detecting objects on training data without taking in account the possibility of OOD data. As mentioned in the first chapter, a lot of object detecting algorithms are trained in a closed environment, which makes them more susceptible for out OOD data.

5.2 YOLO, adversial examples and noise data

We have seen that the performance of YOLO reduces when introduced to adversarial examples and noise data. Moreover, adding noise resulted in the YOLO algorithm to detect no objects and the adversarial examples also performed worse. When we take the example from chapter one with the self driving car we can see that this a vital problem for object detection. For example, the probability that a self driving car learns to identify pedestrians who walk upwards, is very high. However when a person falls, it's orientation is rotated and as we have seen in the results, causes the object detecting algorithm to perform less. This can result in a miss classification of the pedestrian, and cause a crash. Another example could be that the observing camera is dirty or wet, which results in a blurred image. Here too, the algorithm performed less which could also have disastrous consequences.

This can be extended towards the Robocup, where two teams of robots play football against each other. There are many situations where a robot might have a different angle, i.e. when a robot is laying on the ground. If the robot isn't detected by YOLO, it can trip over it, which can result in multiple robots falling on the ground.

5.3 Geometric Gradient Analysis

As the results have shown, we can conclude that the Geometric Gradient Analysis was not very effective. This is where the limitations of this research is met. For the GGA to work, a large dataset of OOD is needed. In contrast to the findings of the susceptibility of YOLO on OOD, where only a small dataset can suffice. Finding OOD data was also a large obstacle. After using several datasets, ranging from animals, different plants and in the end, different sports, only the team sports played on grass seemed effective in creating OOD. However these datasets themselves had a small size.

5.4 Revising the hypothesis

In the final section of this thesis we revise our hypothesis from our introduction: Is it possible to use geometric gradient analysis described in the paper to detect out of distribution data and therefore, improve the confidence of the Yolo algorithm? This question can not be answered after this thesis. However the pieces to answer the question are there. The article Schwinn et al. (2021) confirms that GGA can be used to detect OOD, and this thesis has proven it is possible to create the OOD

dataset based on the YOLO model, to test the GGA. The next step to answer this question lies in the creation of a larger dataset.

References

Albawi, S., Mohammed, T. A. and Al-Zawi, S. (2017), Understanding of a convolutional neural network, *in* '2017 international conference on engineering and technology (ICET)', Ieee, pp. 1–6.

Knight, J. C. (2002), Safety critical systems: challenges and directions, *in* 'Proceedings of the 24th international conference on software engineering', pp. 547–550.

Liu, T., Fang, S., Zhao, Y., Wang, P. and Zhang, J. (2015), 'Implementation of training convolutional neural networks', *arXiv preprint arXiv:1506.01195* .

Schwinn, L., Nguyen, A., Raab, R., Bungert, L., Tenbrinck, D., Zanca, D., Burger, M. and Eskofier, B. (2021), 'Identifying untrustworthy predictions in neural networks by geometric gradient analysis', *arXiv preprint arXiv:2102.12196* .

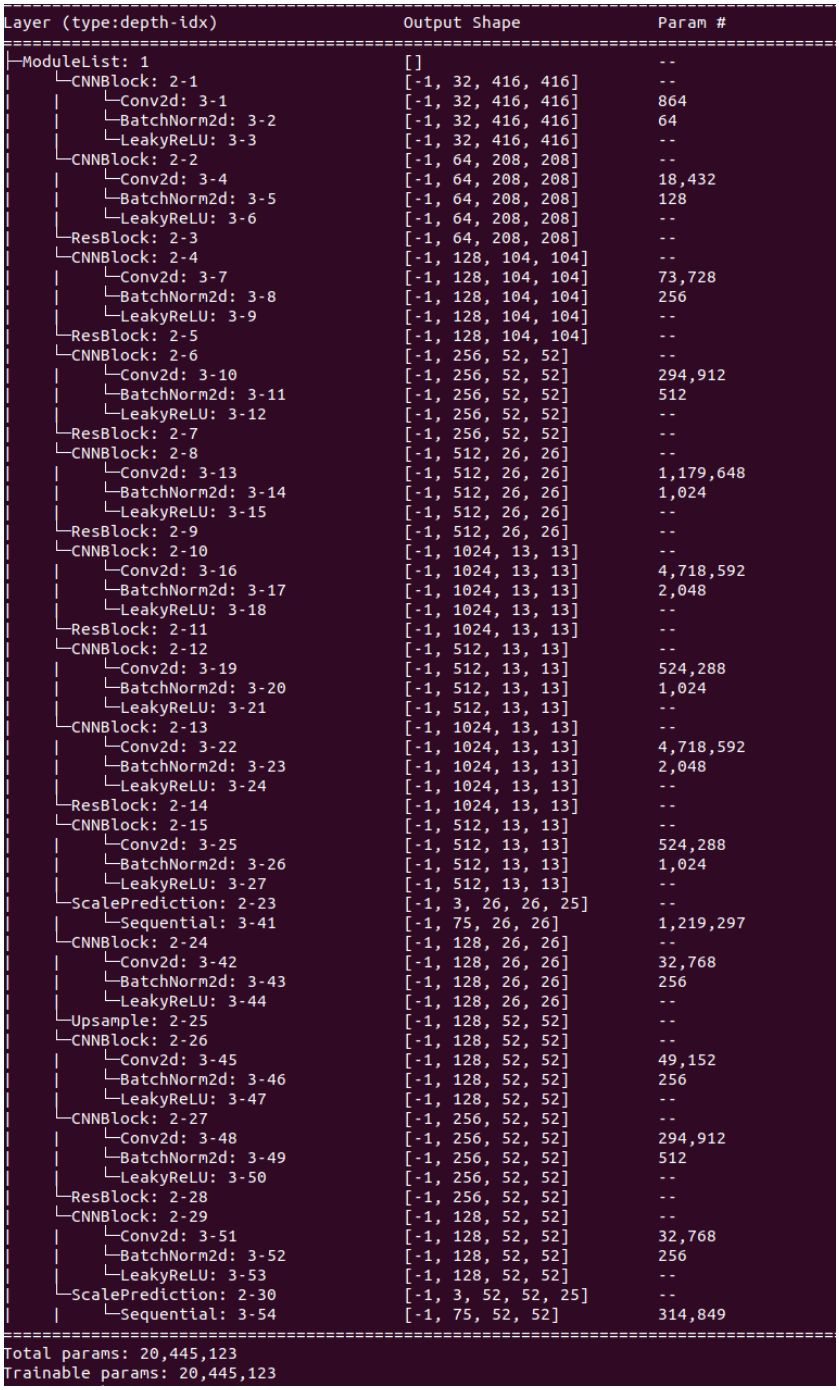
Simonyan, K., Vedaldi, A. and Zisserman, A. (2013), 'Deep inside convolutional networks: Visualising image classification models and saliency maps', *arXiv preprint arXiv:1312.6034* .

Thuan, D. (2021), 'Evolution of yolo algorithm and yolov5: the state-of-the-art object detection algorithm'.

Yao, Z., Douglas, W., O'Keeffe, S. and Villing, R. (2022), Faster yolo-lite: Faster object detection on robot and edge devices, *in* 'Robot World Cup', Springer, pp. 226–237.

Chapter 6

Apendix



```
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─ModuleList: 1                          []                        --
|    └─CNNBlock: 2-1                     [-1, 32, 416, 416]        --
|    |    └─Conv2d: 3-1                  [-1, 32, 416, 416]        864
|    |    └─BatchNorm2d: 3-2             [-1, 32, 416, 416]        64
|    |    └─LeakyReLU: 3-3               [-1, 32, 416, 416]        --
|    └─CNNBlock: 2-2                     [-1, 64, 208, 208]        --
|    |    └─Conv2d: 3-4                  [-1, 64, 208, 208]        18,432
|    |    └─BatchNorm2d: 3-5             [-1, 64, 208, 208]        128
|    |    └─LeakyReLU: 3-6               [-1, 64, 208, 208]        --
|    └─ResBlock: 2-3                     [-1, 64, 208, 208]        --
|    └─CNNBlock: 2-4                     [-1, 128, 104, 104]       --
|    |    └─Conv2d: 3-7                  [-1, 128, 104, 104]       73,728
|    |    └─BatchNorm2d: 3-8             [-1, 128, 104, 104]       256
|    |    └─LeakyReLU: 3-9               [-1, 128, 104, 104]       --
|    └─ResBlock: 2-5                     [-1, 128, 104, 104]       --
|    └─CNNBlock: 2-6                     [-1, 256, 52, 52]         --
|    |    └─Conv2d: 3-10                 [-1, 256, 52, 52]         294,912
|    |    └─BatchNorm2d: 3-11            [-1, 256, 52, 52]         512
|    |    └─LeakyReLU: 3-12              [-1, 256, 52, 52]         --
|    └─ResBlock: 2-7                     [-1, 256, 52, 52]         --
|    └─CNNBlock: 2-8                     [-1, 512, 26, 26]         --
|    |    └─Conv2d: 3-13                 [-1, 512, 26, 26]         1,179,648
|    |    └─BatchNorm2d: 3-14            [-1, 512, 26, 26]         1,024
|    |    └─LeakyReLU: 3-15              [-1, 512, 26, 26]         --
|    └─ResBlock: 2-9                     [-1, 512, 26, 26]         --
|    └─CNNBlock: 2-10                    [-1, 1024, 13, 13]        --
|    |    └─Conv2d: 3-16                 [-1, 1024, 13, 13]        4,718,592
|    |    └─BatchNorm2d: 3-17            [-1, 1024, 13, 13]        2,048
|    |    └─LeakyReLU: 3-18              [-1, 1024, 13, 13]        --
|    └─ResBlock: 2-11                    [-1, 1024, 13, 13]        --
|    └─CNNBlock: 2-12                    [-1, 512, 13, 13]         --
|    |    └─Conv2d: 3-19                 [-1, 512, 13, 13]         524,288
|    |    └─BatchNorm2d: 3-20            [-1, 512, 13, 13]         1,024
|    |    └─LeakyReLU: 3-21              [-1, 512, 13, 13]         --
|    └─CNNBlock: 2-13                    [-1, 1024, 13, 13]        --
|    |    └─Conv2d: 3-22                 [-1, 1024, 13, 13]        4,718,592
|    |    └─BatchNorm2d: 3-23            [-1, 1024, 13, 13]        2,048
|    |    └─LeakyReLU: 3-24              [-1, 1024, 13, 13]        --
|    └─ResBlock: 2-14                    [-1, 1024, 13, 13]        --
|    └─CNNBlock: 2-15                    [-1, 512, 13, 13]         --
|    |    └─Conv2d: 3-25                 [-1, 512, 13, 13]         524,288
|    |    └─BatchNorm2d: 3-26            [-1, 512, 13, 13]         1,024
|    |    └─LeakyReLU: 3-27              [-1, 512, 13, 13]         --
|    └─ScalePrediction: 2-23             [-1, 3, 26, 26, 25]       --
|    |    └─Sequential: 3-41             [-1, 75, 26, 26]          1,219,297
|    └─CNNBlock: 2-24                    [-1, 128, 26, 26]         --
|    |    └─Conv2d: 3-42                 [-1, 128, 26, 26]         32,768
|    |    └─BatchNorm2d: 3-43            [-1, 128, 26, 26]         256
|    |    └─LeakyReLU: 3-44              [-1, 128, 26, 26]         --
|    └─Upsample: 2-25                    [-1, 128, 52, 52]         --
|    └─CNNBlock: 2-26                    [-1, 128, 52, 52]         --
|    |    └─Conv2d: 3-45                 [-1, 128, 52, 52]         49,152
|    |    └─BatchNorm2d: 3-46            [-1, 128, 52, 52]         256
|    |    └─LeakyReLU: 3-47              [-1, 128, 52, 52]         --
|    └─CNNBlock: 2-27                    [-1, 256, 52, 52]         --
|    |    └─Conv2d: 3-48                 [-1, 256, 52, 52]         294,912
|    |    └─BatchNorm2d: 3-49            [-1, 256, 52, 52]         512
|    |    └─LeakyReLU: 3-50              [-1, 256, 52, 52]         --
|    └─ResBlock: 2-28                    [-1, 256, 52, 52]         --
|    └─CNNBlock: 2-29                    [-1, 128, 52, 52]         --
|    |    └─Conv2d: 3-51                 [-1, 128, 52, 52]         32,768
|    |    └─BatchNorm2d: 3-52            [-1, 128, 52, 52]         256
|    |    └─LeakyReLU: 3-53              [-1, 128, 52, 52]         --
|    └─ScalePrediction: 2-30             [-1, 3, 52, 52, 25]       --
|    |    └─Sequential: 3-54             [-1, 75, 52, 52]          314,849
==========================================================================================
Total params: 20,445,123
Trainable params: 20,445,123
```

Figure 6.1: Print of Model