NHL STENDEN
computer vision & data science

Object Detection Models for Detecting Apples in Orchards

NHL Stenden Centre of Expertise in Computer Vision & Data Science

Dennis de Graaf, Dylan Hiemstra and Wouter Welling

Supervisors: Willem Dijkstra and Jaap van de Loosdrecht

Abstract—To build a system that helps reduce the working hours of human apple pickers, an object detection model for detecting apples is essential. In this project, we create an apple detection benchmark with three different object detection models. Although data augmentation generally improves results, our research shows that not all augmentations do. It also shows that using a model that has been trained on the MinneApple dataset and using it the RIWO dataset results in a decrease in performance. This research can be used as a stepping stone to continue the search of finding a usable and reliable object detection model for use in orchards. The YOLOv5X shows the best F1-score of 71%, and the second best, EfficientDet D3, shows a F1-score of 70% on the MinneApple dataset.

Index Terms—Object Detection, Picking Apples, Transfer Learning, Deep Learning

1 Introduction

In 2019 The Netherlands produced over 273 million kilograms of apples [1]. A human worker can pick between 130 and 150 kilograms of apples per hour [1]. This translates to an estimate of 1.95 million working hours each year. A system that is able to detect the apples could help to reduce the working hours of human pickers. For such a system to work, it needs to detect the location of the apples in the scene. This is where automatic fruit detection comes into play.

At the beginning of fruit detection, researchers used traditional computer vision algorithms to detect the fruits [2]. Since detecting apples in orchards comes with a few difficulties, like different light sources and apples behind branches, these algorithms were not performing well enough for usage in the real world. This all changed when computer vision in combination with deep learning became popular. Currently, there are multiple deep learning models that can be used for fruit detection. They are called object detection models. Object detection models are much more capable of solving the difficulties of detecting apples in orchards [2].

The goal of our research is to find a set of object detection models, that can be used for apple detection, and benchmark them using a dataset that consists of RGB images from an apple orchard. We have chosen to find a set of object detection models because there might be multiple object detection models that can be used for apple detection.

1.1 Research questions

In this paper we aim to answer the following main question: **What object detection models are best suited for detection apples in orchards?** Asking the following sub-questions will help answer the research question:

- What is the performance of the object detection models?
- What is the inference time of the object detection models? This is relevant because this research might be used for selecting hardware for an apple picking solution.
- What influence has data augmentation on the performance of object detection models?
- Is it possible to use a detection model that is trained on the MinneApple dataset and to use that model on the RIWO dataset without extra training? This is relevant because we can see how robust a model is on another dataset.

- *Dennis de Graaf is an Electrical Engineering student at the NHL Stenden University of Applied Sciences, E-mail: dennis.de.graaf@cyacs.nl*
- *Dylan Hiemstra is a Software Engineering student at the Hanze University of Applied Sciences, E-mail: p.d.p.hiemstra@st.hanze.nl*
- *Wouter Welling is an Electrical Engineering student at the NHL Stenden University of Applied Sciences, E-mail: wouter.welling@student.nhlstenden.com*
- *Willem Dijkstra is a researcher at the NHL Stenden Centre of Expertise in Computer Vision & Data Science, E-mail: willem.dijkstra@nhlstenden.com*
- *Jaap van de Loosdrecht is a professor at the NHL Stenden Centre of Expertise in Computer Vision & Data Science, E-mail: jaap.van.de.loosdrecht@nhlstenden.com*

2 State of the art

Previously, there was no unified dataset containing images of orchards for benchmarking object detection models. The MinneApple paper [2] describes how the researchers created a high quality dataset containing images of orchards that can be used for comparing object detection models.

In our research, we will use this dataset for benchmarking a set of object detection models. The dataset contains images of apples in an apple orchard, with a total of 41.000 annotated object instances in 1.000 images. The MinneApple paper describes the process of creating a dataset for doing object detection in orchards and provides test results of three object detection models. The authors of [2] did not include the inference time of these tested object detection models and also not (all) the models that we would like to test:

Faster R-CNN [3]. The MinneApple paper claims that Faster R-CNN is the state-of-the-art solution for automatic apple detection. We want to confirm this and see how it performs compared to the other mentioned models. We will use a different backbone called VVG-16 instead of ResNet 512 as used in the MinneApple paper. This is because we want to check if this model is still the state-of-the-art with different backbone.

EfficientDet [4]. EfficientDet has not been in a comparison using the MinneApple dataset yet while it does have great results on the COCO 2017 dataset [4]. It has multiple scaling levels, meaning different sizes of the network are available. This makes it very interesting to see how it will perform using the MinneApple dataset.

YOLO v5 [5]. YOLOv5 is a one-stage detector. Just like EfficientDet, it also has not been in a comparison using the MinneApple dataset. It also has multiple scaling levels like EfficientDet. Because it has so many similarities with EfficientDet it will be interesting to see the difference.

With the results from our benchmark, we can compare every model based on the performance and inference time to determine which of the models will be best suitable for the detection of apples in orchards.

3 Materials and Methods

In this section, we will explain shortly the selected models, datasets and explain how to convert segmentation's to bounding boxes and provide basic information about metrics which we will be using for the experiments.

3.1 Faster R-CNN

In [3], Faster-RCNN is being introduced. Faster R-CNN stands for region-based convolutional neural network. It is a detection system that is composed of two modules. The first module is the region proposal network (RPN) its purpose is to propose multiple objects that are identifiable within an image. The second module is Fast R-CNN which is used for object detection. We use the VVG-16 backbone instead of the ResNet512 backbone because we would like to see the difference between the two. The modules combined is the Faster R-CNN network. See Figure 1 for a visual representation of Faster R-CNN. In our research, we use the simple-Faster-RCNN repository[6].

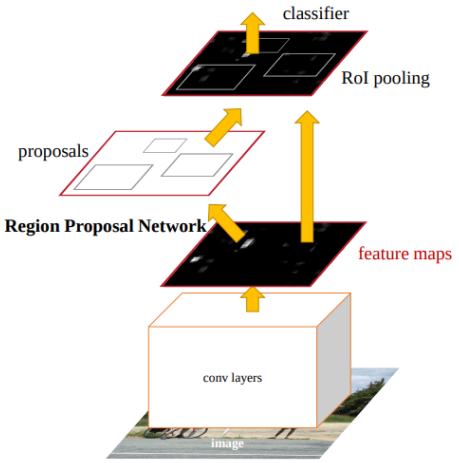


Fig. 1: Faster-RCNN architecture [3]

3.2 EfficientDet

In [4], a weighted bi-directional feature pyramid network (BiFPN) is introduced. A feature pyramid network (FPN) is a network where multi-scaled features are being fused using a top-down pathway. It sacrifices accuracy for efficiency. A BiFPN, which uses a bi-directional pathway instead, has a better trade-off in that regard. In the paper, a new object detection model called EfficientDet is being introduced as well. It uses ImageNet-pretrained EfficientNets as its backbone and a BiFPN as a feature network. A feature network takes different scaling levels of features from the backbone and feeds it to a class and bounding box prediction network to detect the object of interest. In Figure 2, the architecture of EfficientDet is shown. The EfficientDet repository that we use is from Rwightman [7].
For EfficientDet, we use tiling. This is because EfficientDet has a fixed input size. With tiling, the image will be split into multiple tiles and the tile will be resized to the appropriate input size. We did not choose for resizing because our assumption is that it will make the apples too small to detect.

It is worth to note that due to hardware limitations related to memory usages of the networks from this repository we cannot use EfficientDet D4 through D7x for the experiments.

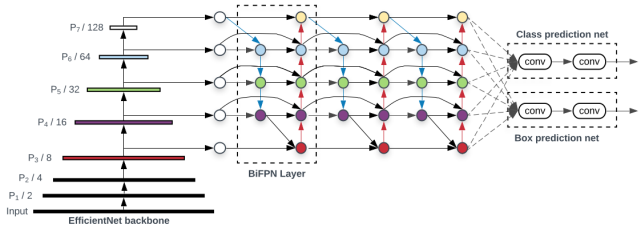


Fig. 2: EfficientDet architecture [4]

3.3 YOLO v5

According to [5], the majority of convolution neural network-based models are slow and cost-inefficient. This led to the creation of YOLO. YOLO stands for You Only Look Once. It is a real-time object recognition system that can recognize multiple objects in a single frame. YOLO is based on a single convolutional neural network (CNN). The CNN divides an image into regions and predicts the bounding boxes. Since the birth of YOLO, different versions of YOLO has been introduced. The most popular YOLO version is v3 introduced by Joseph Redmon [8]. YOLOv5 is one of the latest releases by another company named: Ultralytics, this is the repository that we are using [9]. YOLOv5 in comparison to YOLOv3 differs in the anchor techniques. YOLOv5 automatically learns the bounding box anchors. This is important for the use of custom datasets. The standard bounding box anchors in v3 and v4 are optimized for the COCO dataset, not for a custom dataset. YOLOv5 has four versions: S, M, L, X. YOLOv5X is the largest model and YOLOv5S is the smallest. YOLOv5 versions do not differ in layers. The models differ in the scaling multipliers of the width and depth of the network. In order to ensure a fair comparison with other object detection models (Faster R-CNN and Effucuebtdet), we did not use the build-in augmentation, TTA (Test-Time Augmentation), or pretrained weights. The input image size has to be a power of 32, which is why we use an image size of 1024x576 for training and testing.

3.4 MinneApple Dataset

The MinneApple dataset has been released to enable direct comparisons of networks by providing a large variety of high-resolution (720 by 1280 pixels) images acquired in orchards. The image collection has been done by using a standard Samsung Galaxy S4 cell phone. Video footage was used by facing the camera horizontally moving through an orchard with a speed of approximately 1 m/s. This was done to avoid motion blur. Every 5th frame of the video sequences is extracted to create the dataset and was annotated manually in the form of segmentations. The main dataset consists of a total of 1000 images. Within these images, there are a total of 41.000 annotated object instances. During the creation of the dataset variety of different colors, lighting conditions, and different stages of the ripening cycle were taken into account. See Figure 3 for an example of images taken with different lighting conditions.

3.5 RIWO Dataset

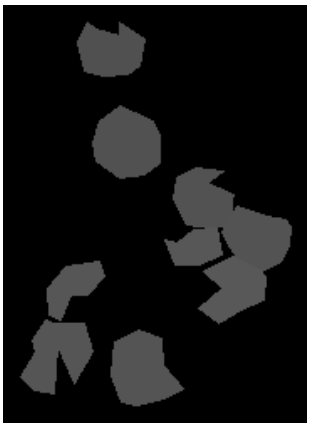
In addition to the MinneApple dataset, our client provided us with the RIWO dataset. However, this dataset is relatively small and only contains 32 pictures with varying resolutions. These are 720x720, 720x960 and 720x1280. Hence, we opt for using this dataset to test the generalization ability of the object detection models. See Figure 9 for an example image from the dataset.


Fig. 3: Example of different lighting conditions [2]

Fig. 4: Picture from RIWO dataset

3.6 Segmentations to bounding box

Since The MinneApple dataset has annotated segmentations instead of bounding boxes, we had to convert the segmentations to bounding boxes. In Figure 5, the before and after-effects are shown.


(a) Segmentations

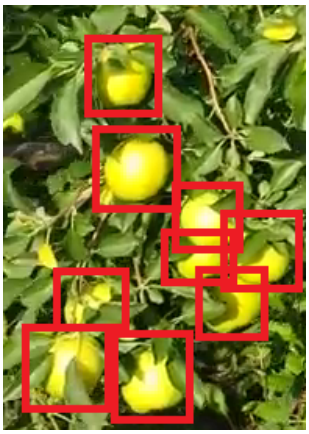

(b) Bounding Boxes

Fig. 5: Visualization of converting segmentations to bounding boxes

3.7 Metrics and Thresholds

To compare each model, we need to determine how each model performs. To determine this we will use the following metrics and thresholds:

Precision. Precision is the indicator of the percentage of the number of correctly detected apples compared to the total number of predicted apples. See Equation 1.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{1}$$

Recall. Recall is the indicator of the percentage of number of correctly detected apples compared to the number of apples in the ground truth. See Equation 2.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2}$$

F1-score. F1-score combines precision and recall into a single value. See Equation 3.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

Mean Average Precision. Mean Average Precision (mAP) indicates the area under the precision-recall curve. The higher its value the more confident the model predicts. Note that since we have only one class we are actually using the average precision.
Inference time. The inference time is measured in milliseconds. It indicates the average time it takes to retrieve the predictions out of one image.
Intersection over Union. Intersection over Union (IoU) is a threshold used to determine if a bounding box is positive or negative. If the IoU has a value of 0.5, it means that the predicted bounding boxes need to overlap at least 50% with the ground-truth bounding box in order to be considered a positive bounding box.
Confidence threshold. Confidence threshold is used to determine when a bounding box is a true positive. The value is the class confidence score of the highest F1-score on the validation set.

Both Intersection over Union and confidence threshold are used to calculate precision and recall. This means it also indirectly is used for F1-score and Mean Average Precision.

4 Experiments & Results

In this subsection, we will explain which and how we will run our experiments. To filter out random advantages, like a lucky seed (a coincidence of generating the best weights possible for a model), we will run each experiment three times and take the average of each metric per network. We will note the following metrics for each experiment: precision, recall, F1-score, mean average precision (mAP) and inference time. An IoU of 0.5 is used because it is the industry standard. All experiments are using the MinneApple dataset for training, validation and testing. For every experiment we will use the same split of the dataset. The test dataset contains 161 images, the train set 403, and the validation set 106 images.

To find the best confidence threshold, we calculate the F1 score for each confidence level using the validation set. Then, we pick the

confidence threshold with the highest F1 score and use that confidence threshold with the testing set for the final results.

4.1 Hardware specifications

In Table 1 the hardware specifications of our testing VMs is shown.

Table 1: Virtual machines and the corresponding hardware specifications

Computer name	GPU	GPU-RAM	CPU	RAM
VM-DF	Tesla P100-SXM2	16GB	E5-2699 v4 @ 2.20GHz	32GB
VM-CVDS	GeForce RTX 2070	8GB	i9-7960X @ 2.80GHz	16GB
VM-SN	GeForce RTX 2070	8GB	AMD EPYC 7452 32-Core Processor @ 2.40GHz	32GB

4.2 Experiment A

Experiment A will be used to make a baseline for the other experiments. We will use the networks that we discussed before: Faster R-CNN, EfficientDet (with tiling) D0 through D3 and YOLOV5 models S, M, L, and X. The data is not augmented meaning that no augmentations will be used, even built-in augmentations will be disabled. We expect that each scaling level of EfficientDet will perform better than the previous one. We also expect that bigger versions of YOLOv5 will better perform than smaller ones, due to network sizes. Faster R-CNN is expected to perform the same (even though a different backbone is used) as in the MinneApple paper[2] (mAP@0.5 of 0.775).

4.2.1 Results from Experiment A

In Table 2, the average metrics of each run from experiment A is shown. In Appendix A.1, the full results are shown.

As seen in the inference column in Table 2, the inference times of EfficientDet is much higher than YOLOv5 and Faster-RCNN. This is caused by the tiling process. We chose to include the time it takes to stitch together the tiles into one image. This is also the reason why D2 and D3 are faster than D0 and D1, while D2 and D3 are bigger networks. D2 and D3 have bigger input sizes. Meaning that fewer tiles have to be generated for D2 and D3.
One more interesting phenomenon in these results is the confidence values of YOLOv5. Those values are very low, indicating that YOLOv5 is underfitting. This is probably caused by removing the built-in augmentations from our implementation of YOLOv5.
EffcientDet D3 is the best performing model in this experiment, because the F1 score is the highest.

It is worth to note that each model has a different input size, except YOLOv5. EfficientDet D2 and D3 cannot be used on an 8GB GPU. Results from MinneApple paper used a different split ratio and used another backbone with more network layers.

4.3 Experiment B

With experiment B, we would like to answer the question about the influence of augmentations. We will take the same approach as in experiment A but add augmentations into the mix. We expect to see an increase in each metric except for inference time. The following augmentations will be used in this experiment:

- ISO noise (probability 0.30). See Figure 7a.
- Median blur (probability 0.30, limit 3). See Figure 7b.

Table 2: Experiment A

Expt. A	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,49	0,64	0,56	0,48	0,33	382
EfficientDet D1	0,47	0,61	0,53	0,44	**0,45**	474
EfficientDet D2	0,67	0,65	0,66	0,59	0,27	168
EfficientDet D3	0,70	**0,70**	**0,70**	0,65	0,36	194
YOLOv5 S	0,76	0,56	0,64	0,53	0,03	**9**
YOLOv5 M	**0,78**	0,57	0,66	0,54	0,09	15
YOLOv5 L	0,76	0,57	0,65	0,51	0,09	21
YOLOv5 X	0,76	0,51	0,61	0,48	0,03	38
Faster R-CNN	0,47	0,29	0,35	0,25	0,36	68
Tiled FR-CNN (MinneApple)	NA	NA	NA	0.639	NA	NA
Faster R-CNN (MinneApple)	NA	NA	NA	**0,775**	NA	NA
Masked R-CNN (MinneApple)	NA	NA	NA	0.763	NA	NA

Expt = Experiment, Prc = Precision, Rel = Recall, mAP = mean Average Precision, Th = Confidence threshold used for precision and recall, Inf = Inference time

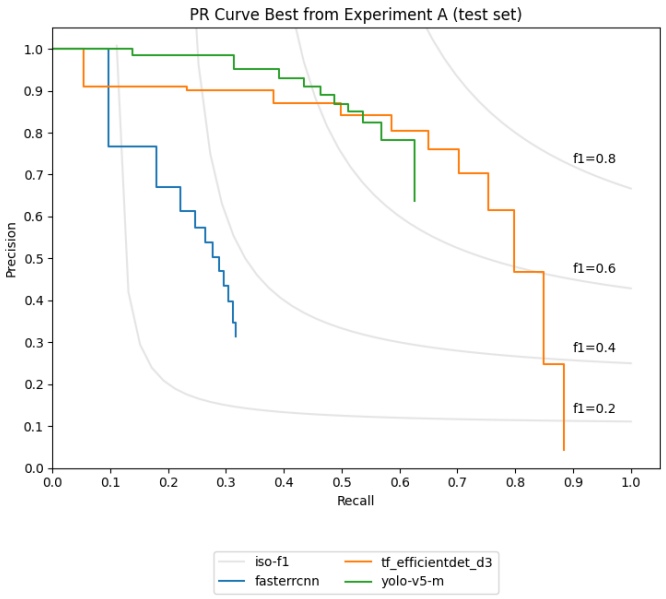


Fig. 6: PR curve of the best models from Experiment A

- Random brightness contrast (probability 0.30, limit 0.20). See Figure 7c.
- Hue saturation (probability 0.30, shift limit 3). See Figure 7d.
- Horizontal flip (probability 0.25). See Figure 7e.
- Shift + rotate (Limit 0.0625, probability 0.25, between -10°and +10°) See Figure 7f.

(a) ISO noise (b) Median blur (c) Random brightness contrast

(d) Hue saturation (e) Horizontal flip (f) Shift + rotate

(g) Original

Fig. 7: Visualization of different augmentations

We will also take a quick look at YOLOv5 with the built-in augmentations turned on. We want to see how different the results are with it turned on. We will still use the MinneApple dataset and we expect it to have better results since it is turned on by default.

4.3.1 Results from Experiment B

In Table 3, the average metrics of each run from experiment B is shown. In Appendix A.2, the full results and precision recall curve is shown. From Table 3 we can conclude that YOLOv5 X is the best performing model.

As seen in Table 3, the same phenomenons from experiment A are occurring in experiment B.

In Table 4 we can see that YOLOv5 performs much better with the built-in augmentations.

4.4 Experiment C

In experiment C we will be using the models with the highest F1 score of experiment A and B. These models are trained on the MinneApple dataset. We want to find out how the performance will change if we test the object detection models on a dataset from another orchard.

Table 3: Experiment B

Expt. B	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,52	0,64	0,57	0,48	0,36	394
EfficientDet D1	0,48	0,63	0,53	0,44	0,42	460
EfficientDet D2	0,68	0,68	0,68	0,61	0,27	170
EfficientDet D3	0,69	**0,72**	0,70	**0,64**	0,33	198
YOLOv5 S	0,75	0,64	0,69	0,6	0,06	**8,33**
YOLOv5 M	0,77	0,62	0,69	0,6	0,06	15
YOLOv5 L	**0,80**	0,56	0,66	0,54	0,09	21
YOLOv5 X	0,78	0,65	**0,71**	0,60	0,09	39
Faster R-CNN	0,46	0,29	0,35	0,25	0,33	70

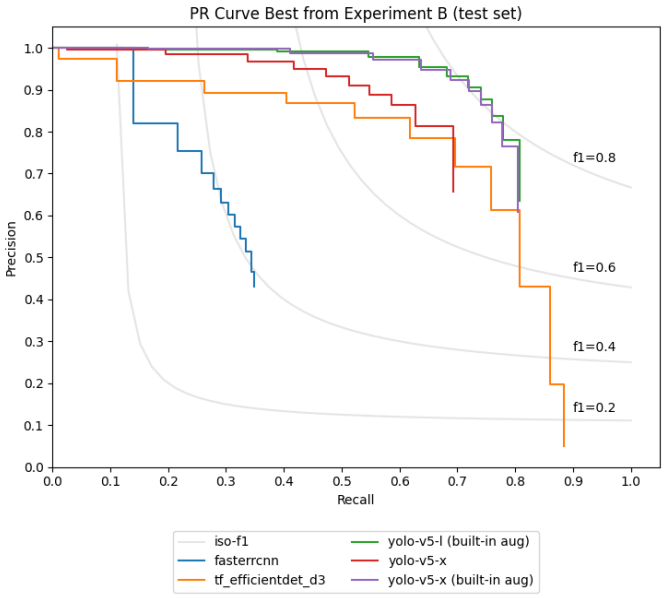


Fig. 8: PR curve of the best models from Experiment B

Table 4: Experiment B, with YOLOv5 built-in augmentations.

Expt. D	Prc.	Rcl.	F1	mAP	Th.	Inf.
YOLOv5 S	0,81	0,74	0,77	0,7	0,18	8
YOLOv5 M	0,83	0,75	0,79	0,71	0,18	15
YOLOv5 L	**0,84**	**0,76**	**0,80**	**0,71**	**0,18**	**21**
YOLOv5 X	0,86	0,74	0,80	0,71	0,27	32

The dataset which we will be testing is the RIWO dataset. This set has a variety of resolutions that the networks are not trained on. Our expectation to see a slight deviation in each metric.

4.4.1 Results from Experiment C

In Table 5, the average metrics of each run from experiment C is shown. We have chosen to run this expirment three times as well to

avoid detection noise. In Appendix A.3, the full results and precision recall curve is shown. Table 5 shows that EfficientDet D3 has achieved the highest F1 score.

We have used the following best performing models of experiment A and B:

- EfficientDet D0: experiment B, run 2
- EfficientDet D1: experiment B, run 1
- EfficientDet D2: experiment B, run 2
- EfficientDet D3: experiment B, run 3
- YOLOv5 S: experiment B, run 1
- YOLOv5 M: experiment B, run 3
- YOLOv5 L: experiment B, run 3
- YOLOv5 X: experiment B, run 3
- Faster R-CNN: experiment A, run 2

Table 5: Experiment C

Expt. C	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,25	0,55	0,34	0,26	0,36	339
EfficientDet D1	0,32	0,55	0,41	0,32	0,42	363
EfficientDet D2	0,36	0,55	0,44	0,37	0,27	140
EfficientDet D3	0,37	**0,61**	**0,46**	0,42	0,33	163
YOLOv5 S	0,34	0,57	0,42	0,4	0,09	**10**
YOLOv5 M	0,36	0,57	0,44	0,41	0,06	19
YOLOv5 L	0,35	0,58	0,44	**0,44**	0,06	25
YOLOv5 X	**0,41**	0,49	0,45	0,39	0,09	43
Faster R-CNN	0,38	0,38	0,38	0,27	0,36	62
YOLOv5 L with built-in augmentations	0,36	0,6	0,45	**0,44**	0,44	25

4.5 Example image

In Figure 10 you can see the same image that went through YOLOv5 L, EfficientDet D0 and Faster R-CNN.

4.6 PR curves

To visualize the performances, we generated precision recall curves. They are located in Appendix B.

4.7 Used Hardware

In Table 6 we show on which hardware each model has been used to gather the metrics. It is important to note that this does not influence the performance but it does influence the inference time.

5 Discussion, Conclusion & Future Work

In this section we will discuss the results, conclude the research and provide suggestions on future work.

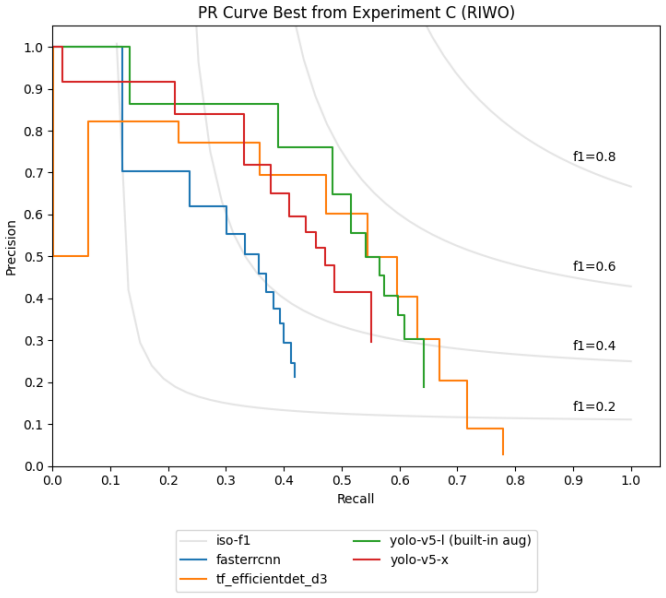


Fig. 9: PR curve of the best models from Experiment C

Fig. 10: Example image. Faster R-CNN, EfficientDet, YOLOv5 L

Table 6: Used hardware for experiments

Network	Hardware
EfficientDet D0	VM_DF
EfficientDet D1	VM_DF
EfficientDet D2	VM_DF
EfficientDet D3	VM_DF
YOLOv5 S	VM_CVDS
YOLOv5 M	VM_CVDS
YOLOv5 L	VM_CVDS
YOLOv5 X	VM_CVDS
Faster R-CNN	VM_SN

5.1 Discussion and Conclusion

In our research, we compared three object detection models: EfficientDet, YOLOv5, and Faster RCNN. We have done three experiments with each of the models, and with YOLOv5 one extra experiment.

A lesson we have learned is that it is really important to write down each condition for every model. For example, we have disabled the built-in augmentations of YOLOv5. This resulted in YOLOv5 underfitting, which is not a good scenario. However it is fairer to do that if we are comparing models without the use of any augmentation. We need to know how a model performs without any additions, from

there we can start to try out different augmentations and see what will increase the performance of each model separately.

One of our findings is that adding tiling adds a lot of overhead to inference time. It is however necessary to add tiling for EfficientDet. This however does limits our research since we did not test EfficientDet without tiling. We also did not try tiling for YOLOv5 and Faster RCNN, but since those networks can be fed through (almost) without resizing, we are not sure if this will improve performance. The overhead could be solved with processing the tiles in parallel, this will improve inference times.

Another finding is that using our chosen augmentations does not impact the results that much. The reason is that the augmentations we apply are already (partly) included in the variation of the dataset. We do see that YOLOv5 performs a little bit better with our chosen augmentations. However, it still underfits since we disabled the built-in augmentations. This is why we did another small experiment with YOLOv5 with the built-in augmentations enabled. This improves the results largely. We did not however try out those augmentations on EfficientDet and Faster RCNN due to time constrains.

Our last finding is that it is possible to use a model trained on the MinneApple dataset and use it on the RIWO dataset. However, it will result in a decrease in performance. Faster RCNN, though, is more robust with both datasets. We are not sure why this is happening. Looking at the results from Faster R-CNN in the MinneApple paper we can see that they achieved a mAP of 0,775 while we achieved a mAP of 0,25 in both experiment A and B. We can conclude that Faster RCNN should perform better with ResNet 512 backbone, instead of the VVG-16 backbone we are using. This might happen due to the fact that VVG-16 has less layers than Resnet-512.

Based on the results from the experiments we can conclude that, without using any augmentations, EfficientDet D3 (F1 score: 0,70, mAP: 0,65, inference time: 194ms) has the potential for being the best-suited model for apple detection. With augmentations though, YOLOv5 X (F1-score: 0,71, mAP: 0,60, inference time: 39ms) has better potential for being the best- suited model for apple detection. The differences between the two models are very slim though and EfficientDet D3 has to be run on a bigger machine than YOLOv5 X.

The goal of the research has been accomplished. After this research new questions are ready to be answered. This research can be taken further to the next step in determining which model is best suited for apple detection.

5.2 Future Work

We think that the next steps in this research is to research tiling more. Our expectations are that tiling provides a better accuracy on smaller objects. What are the benefits in using tiling for YOLOv5 and Faster R-CNN? Augmentations from the original repository seem to really improve the results of YOLOv5. Those augmentations might also help in increasing the results of EfficientDet and Faster R-CNN. For Faster R-CNN, we used the VVG-16 backbone instead of ResNet 512. It seems that Faster R-CNN is robust on both the MinneApple dataset and RIWO. Our suspicion matters that using the ResNet 512 backbone, like from the MinneApple paper will increase the performance of Faster R-CNN and still be robust on both datasets. An interesting thing to experiment with is increasing the number of epochs. We saw examples of YOLOv5 being trained for 3000 epochs, the hypothesis is that YOLOv5 can have a significant increase of performance around the 2000 epochs. We think that more data would also have an effect on the performances of the models.

ACKNOWLEDGEMENTS

- This project is financially supported by SIA RAAK-mkb under the Focus op Vision project.

- The work is supported by NHL-Stenden Computer Vision Data Science and Hogeschool Saxion.

- All authors have contributed equally.

REFERENCES

[1] Fruitconsult.com, "Fruitconsult pluktijdstippenadvies 2016," p. 2, 2016.

[2] P. R. N. Häni and V. Isler, "Minneapple: A benchmark dataset for apple detection and segmentation." University of Minnesota, 2020.

[3] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[4] B. T. Mingxing Tan Ruoming Pang Quoc V. Le Google Research, "Efficientdet: Scalable and efficient object detection," 07 2020.

[5] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv e-prints*, p. arXiv:2004.10934, Apr. 2020.

[6] Y. Chen. A simple and fast implementation of faster r-cnn. [Online]. Available: https://github.com/chenyuntc/simple-faster-rcnn-pytorch

[7] rwightman. Efficientdet (pytorch). [Online]. Available: https://github.com/rwightman/efficientdet-pytorch

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[9] G. Jocher. Ultralytics (yolov5). [Online]. Available: https://github.com/ultralytics/yolov5

A EXPERIMENT RUNS

Every experiment has been ran 3 times to exclude train/evaluation/test noise. Here we show the all data we collected of each run.

A.1 Experiment A

Results from experiment A can be seen below.

Table 7: Experiment A Run 1

Expt. A R1	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,49	0,63	0,55	0,49	0,33	382
EfficientDet D1	0,45	0,61	0,52	0,43	0,45	476
EfficientDet D2	0,67	0,66	0,66	0,6	0,27	167
EfficientDet D3	0,7	0,69	0,7	0,65	0,36	192
YOLOv5 S	0,78	0,56	0,65	0,51	0,09	9
YOLOv5 M	0,78	0,57	0,66	0,51	0,09	15
YOLOv5 L	**0,78**	**0,56**	**0,66**	**0,52**	**0,03**	**21**
YOLOv5 X	0,77	0,54	0,63	0,51	0,03	37
Faster R-CNN	0,45	0,28	0,34	0,24	0,36	67,91

Table 8: Experiment A Run 2

Expt. A R2	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,51	0,64	0,57	0,48	0,33	378
EfficientDet D1	0,51	0,57	0,54	0,44	0,45	474
EfficientDet D2	0,71	0,63	0,66	0,59	0,27	170
EfficientDet D3	**0,72**	**0,69**	**0,71**	**0,66**	**0,36**	**195**
YOLOv5 S	0,72	0,58	0,64	0,57	0,09	9
YOLOv5 M	0,77	0,58	0,66	0,59	0,09	15
YOLOv5 L	0,74	0,56	0,64	0,51	0,03	21
YOLOv5 X	0,73	0,56	0,63	0,51	0,03	39
Faster R-CNN	0,48	0,29	0,36	0,26	0,36	69

Table 9: Experiment A Run 3

Expt. A R3	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,51	0,65	0,57	0,48	0,33	385
EfficientDet D1	0,45	0,64	0,53	0,44	0,45	472
EfficientDet D2	0,63	0,67	0,65	0,57	0,27	168
EfficientDet D3	**0,68**	**0,72**	**0,70**	**0,65**	**0,36**	**194**
YOLOv5 S	0,77	0,55	0,64	0,51	0,09	9
YOLOv5 M	0,8	0,55	0,65	0,51	0,09	15
YOLOv5 L	0,75	0,58	0,65	0,51	0,03	21
YOLOv5 X	0,77	0,44	0,56	0,43	0,03	39
Faster R-CNN	0,48	0,29	0,36	0,25	0,36	67,49

A.2 Experiment B

Results from experiment B can be seen below.

Table 10: Experiment B Run 1

Expt. B R1	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,51	0,64	0,57	0,47	0,36	392
EfficientDet D1	0,52	0,6	0,55	0,45	0,42	483
EfficientDet D2	0,66	0,66	0,66	0,59	0,27	167
EfficientDet D3	0,68	0,72	0,70	0,64	0,33	196
YOLOv5 S	**0,75**	**0,66**	**0,70**	**0,60**	**0,09**	**8**
YOLOv5 M	0,80	0,59	0,68	0,60	0,06	15
YOLOv5 L	0,78	0,53	0,64	0,51	0,06	21
YOLOv5 X	0,79	0,64	0,71	0,60	0,09	39
Faster R-CNN	0,44	0,28	0,34	0,24	0,33	69

Table 11: Experiment B Run 2

Expt. B R2	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,52	0,65	0,58	0,48	0,36	393
EfficientDet D1	0,51	0,56	0,53	0,42	0,42	446
EfficientDet D2	0,69	0,69	0,69	0,62	0,27	171
EfficientDet D3	0,66	0,73	0,69	0,64	0,33	198
YOLOv5 S	0,73	0,62	0,67	0,59	0,09	8
YOLOv5 M	**0,76**	**0,62**	**0,69**	**0,59**	**0,06**	**15**
YOLOv5 L	0,82	0,53	0,64	0,51	0,06	21
YOLOv5 X	0,76	0,65	0,7	0,59	0,09	39
Faster R-CNN	0,46	0,30	0,36	0,26	0,33	70

Table 12: Experiment B Run 3

Expt. B R3	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,53	0,63	0,57	0,48	0,36	396
EfficientDet D1	0,4	0,72	0,51	0,44	0,42	451
EfficientDet D2	0,70	0,68	0,69	0,62	0,27	172
EfficientDet D3	**0,72**	**0,70**	**0,71**	**0,65**	**0,33**	**201**
YOLOv5 S	0,77	0,63	0,69	0,60	0,09	9
YOLOv5 M	0,76	0,64	0,69	0,60	0,06	15
YOLOv5 L	0,79	0,61	0,69	0,60	0,06	21
YOLOv5 X	0,79	0,65	0,72	0,61	0,09	39
Faster R-CNN	0,47	0,29	0,36	0,25	0,33	70

A.3 Experiment C

Results from experiment C can be seen below.

Table 13: Experiment C Run 1

Expt. C R1	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,25	0,54	0,34	0,26	0,36	340
EfficientDet D1	0,32	0,56	0,41	0,32	0,42	352
EfficientDet D2	0,36	0,54	0,44	0,37	0,27	146
EfficientDet D3	**0,37**	**0,61**	**0,46**	**0,42**	**0,33**	**166**
YOLOv5 S	0,34	0,57	0,42	0,40	0,09	10
YOLOv5 M	0,36	0,57	0,44	0,41	0,06	19
YOLOv5 L	0,35	0,58	0,44	0,44	0,06	25
YOLOv5 X	0,41	0,49	0,45	0,39	0,09	42
Faster R-CNN	0,37	0,38	0,38	0,26	0,36	61

Table 14: Experiment C Run 2

Expt. C R2	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,25	0,55	0,35	0,26	0,36	345
EfficientDet D1	0,32	0,55	0,41	0,32	0,42	358
EfficientDet D2	0,37	0,56	0,45	0,37	0,27	137
EfficientDet D3	**0,37**	**0,61**	**0,46**	**0,42**	**0,33**	**159**
YOLOv5 S	0,34	0,57	0,42	0,4	0,09	10
YOLOv5 M	0,36	0,57	0,44	0,41	0,06	19
YOLOv5 L	0,35	0,58	0,44	0,44	0,06	26
YOLOv5 X	0,41	0,49	0,45	0,39	0,09	42
Faster R-CNN	0,37	0,38	0,38	0,28	0,36	64

Table 15: Experiment C Run 3

Expt. C R3	Prc.	Rcl.	F1	mAP	Th.	Inf.
EfficientDet D0	0,25	0,55	0,34	0,25	0,36	333
EfficientDet D1	0,32	0,55	0,41	0,32	0,42	378
EfficientDet D2	0,36	0,54	0,43	0,37	0,27	137
EfficientDet D3	**0,37**	**0,60**	**0,45**	**0,41**	**0,33**	**163**
YOLOv5 S	0,34	0,57	0,42	0,4	0,09	10
YOLOv5 M	0,36	0,57	0,44	0,41	0,06	19
YOLOv5 L	0,35	0,58	0,44	0,44	0,06	25
YOLOv5 X	0,41	0,49	0,45	0,39	0,09	44
Faster R-CNN	0,39	0,38	0,38	0,28	0,36	62

B PR-Curve

B.1 YOLOv5

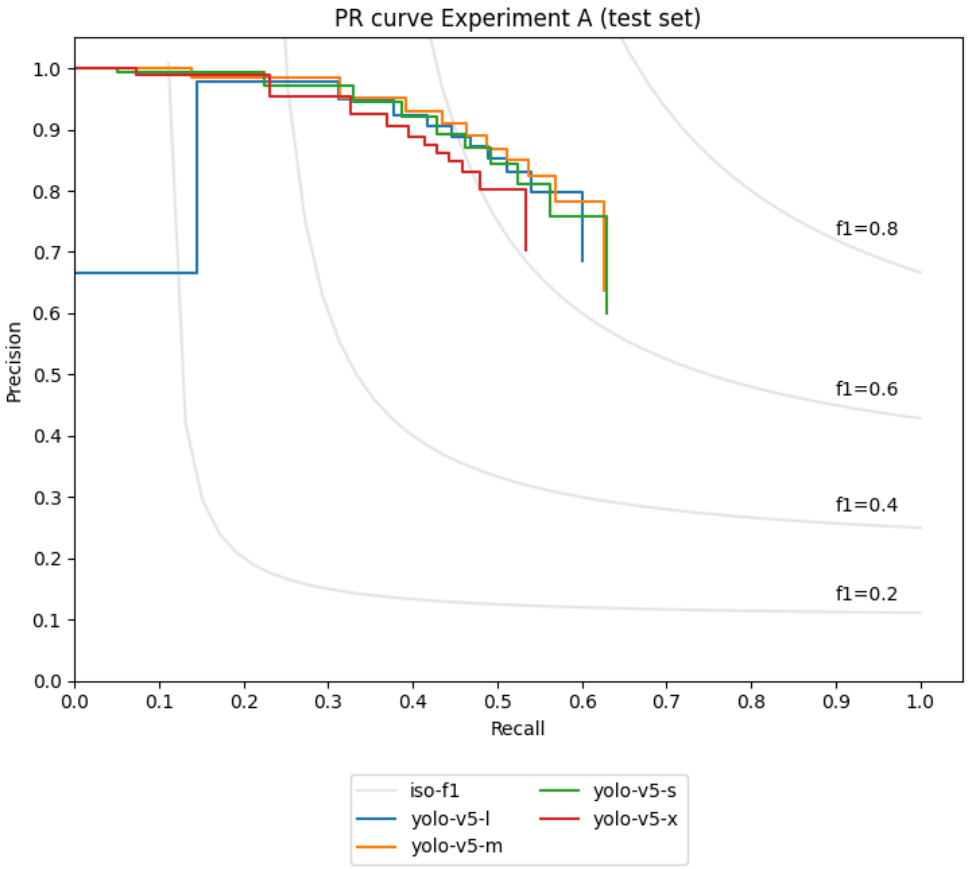


Fig. 11: YOLOv5 PR curve experiment A on the test set.

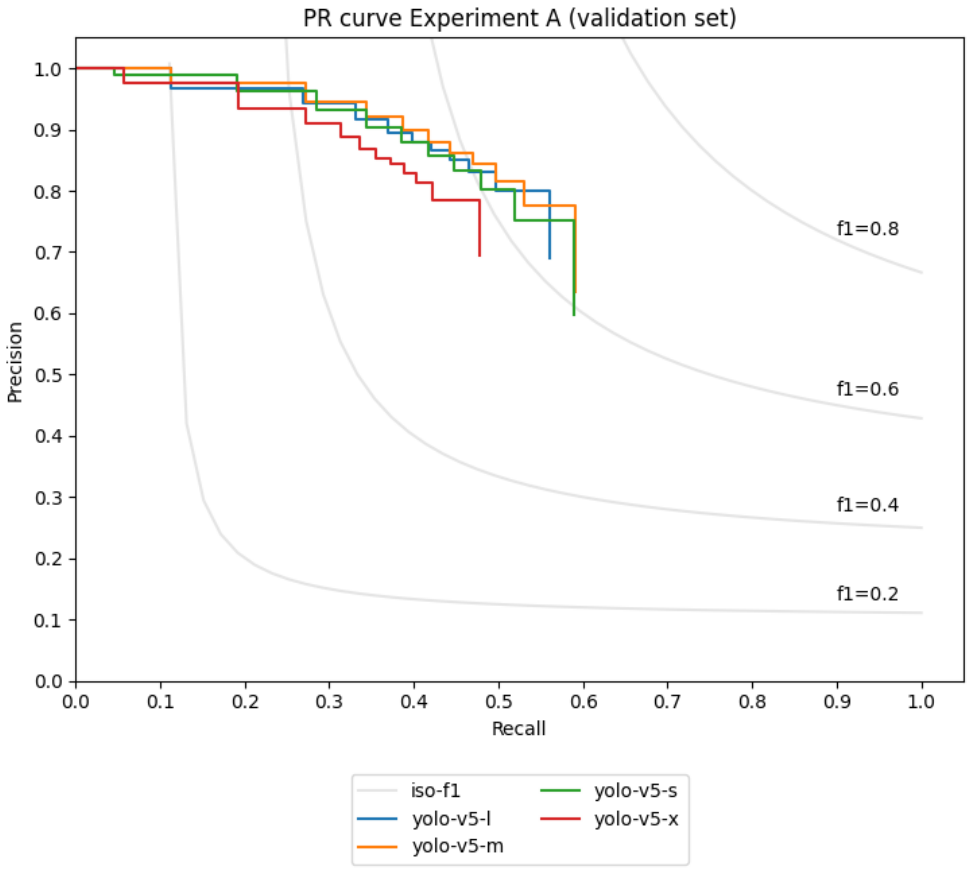


Fig. 12: YOLOv5 PR curve experiment A on the validation set.

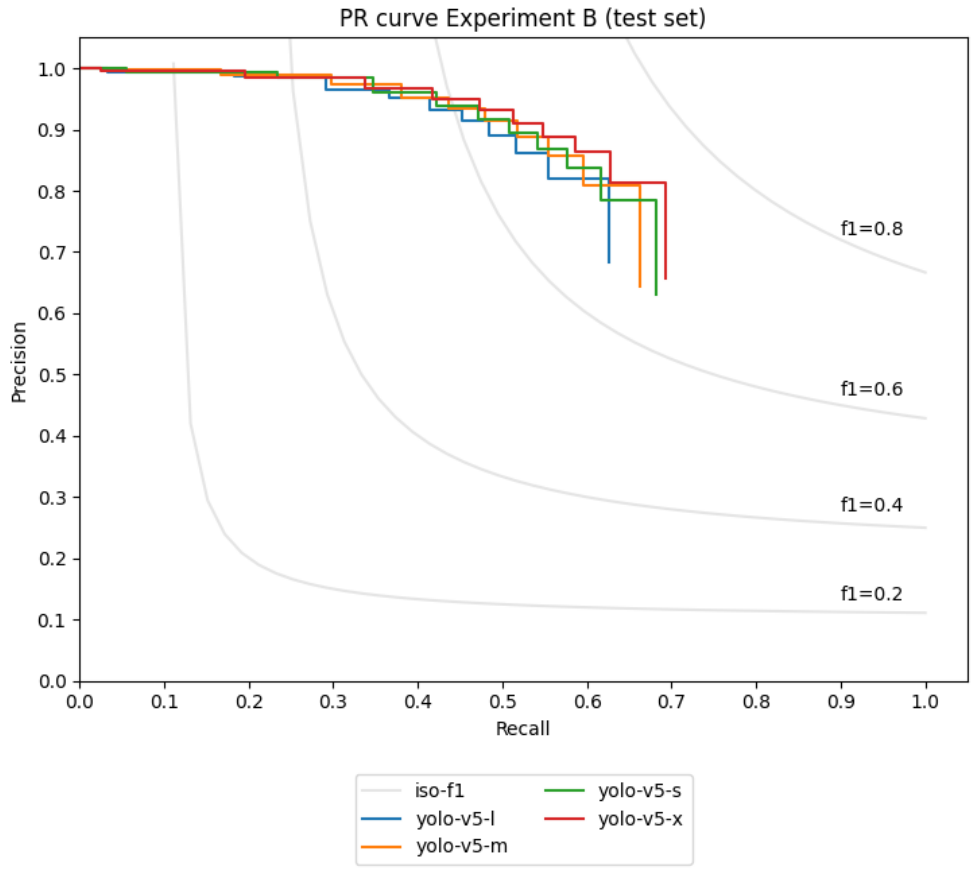


Fig. 13: YOLOv5 PR curve experiment B on the test set.

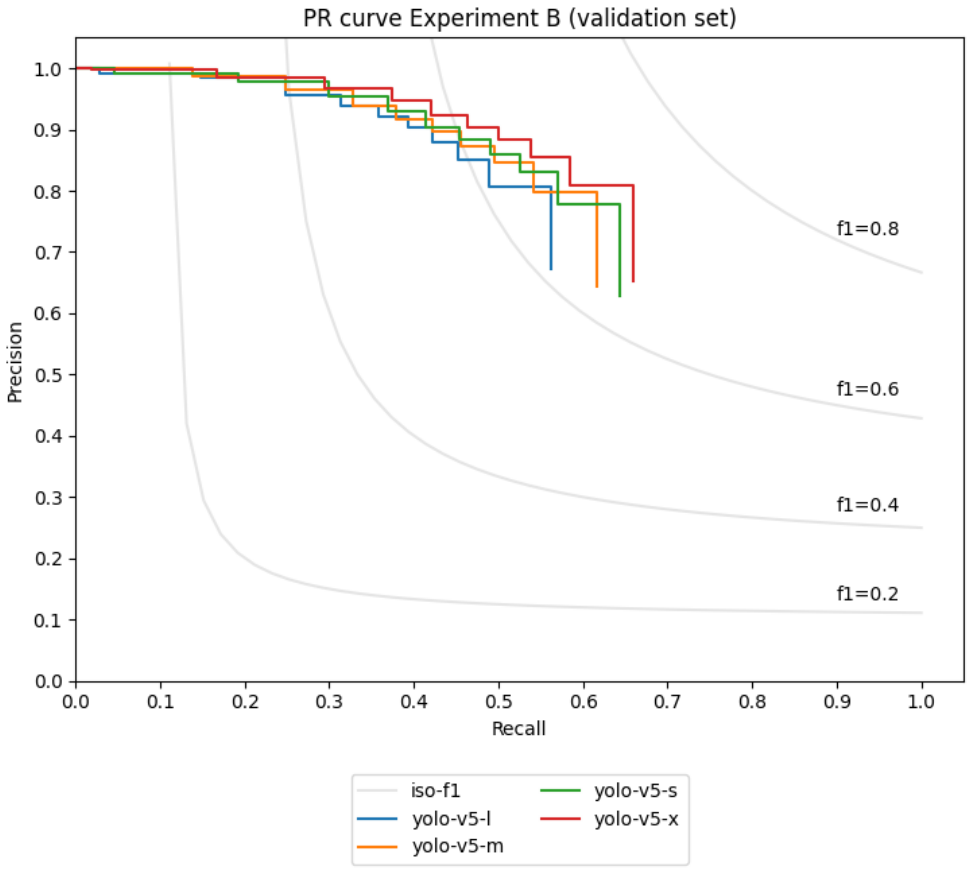


Fig. 14: YOLOv5 PR curve experiment B on the validation set.

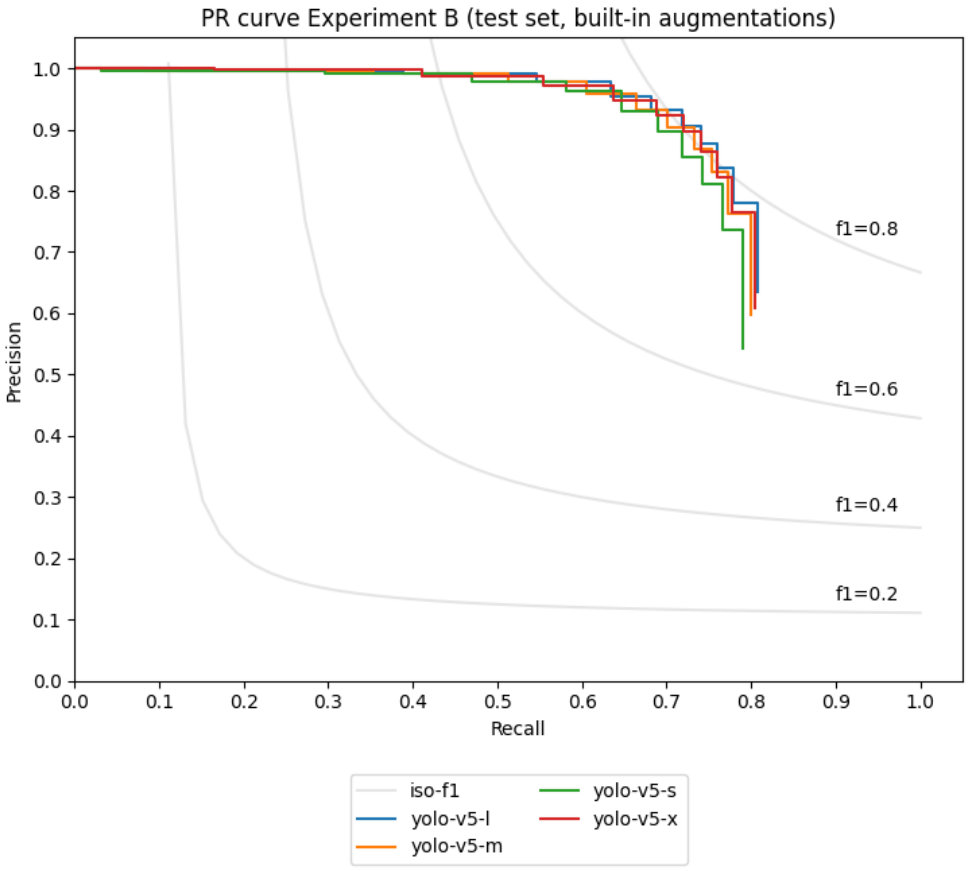


Fig. 15: YOLOv5 PR curve experiment B, with built-in augmentations, on the test set.

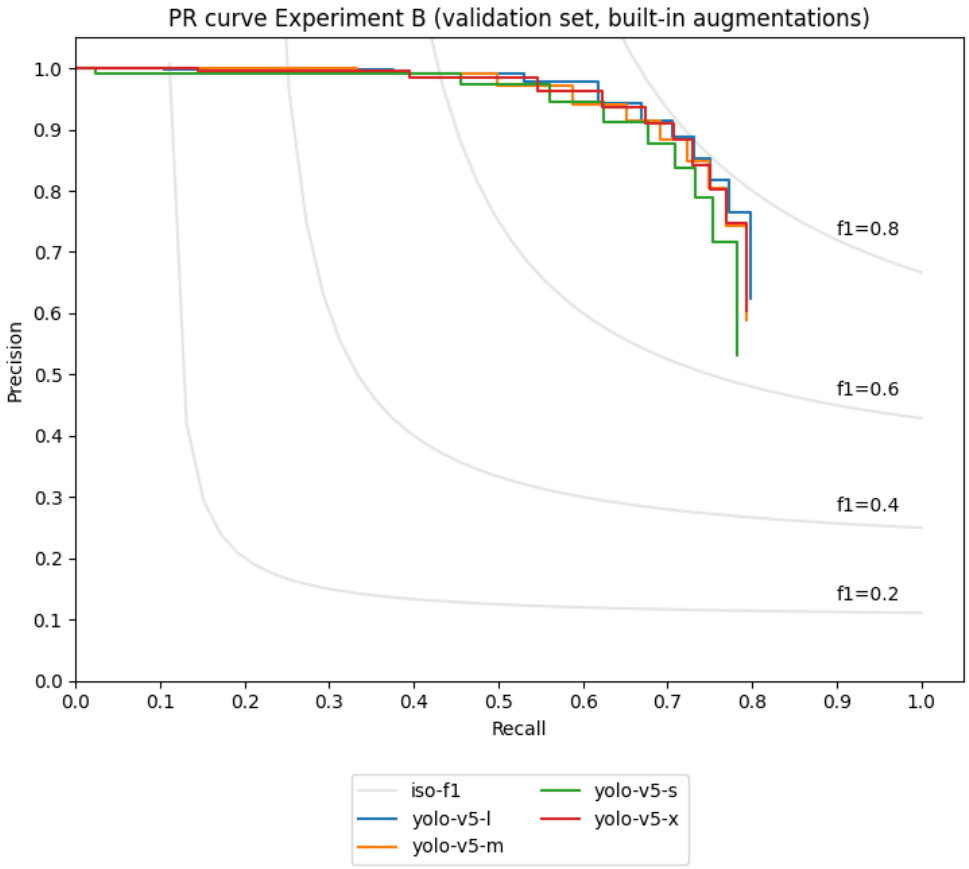


Fig. 16: YOLOv5 PR curve experiment B, with built-in augmentations, on the validation set.

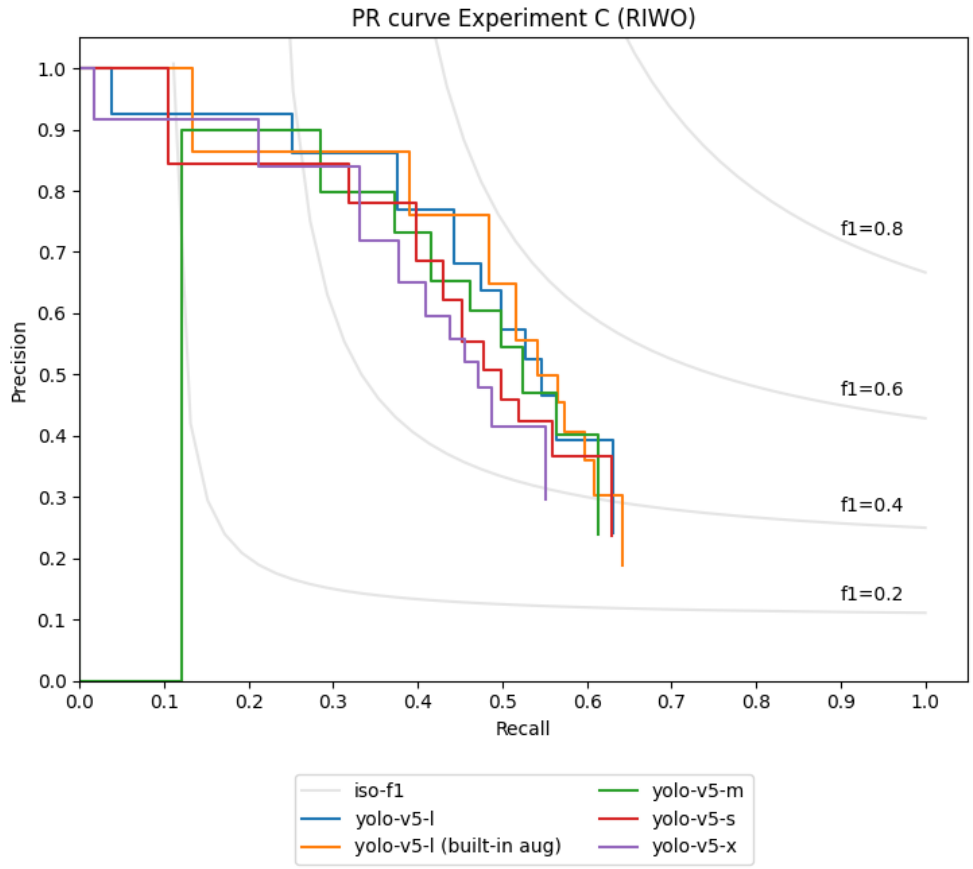


Fig. 17: YOLOv5 PR curve experiment C on the RIWO dataset.

B.2 EfficientDet

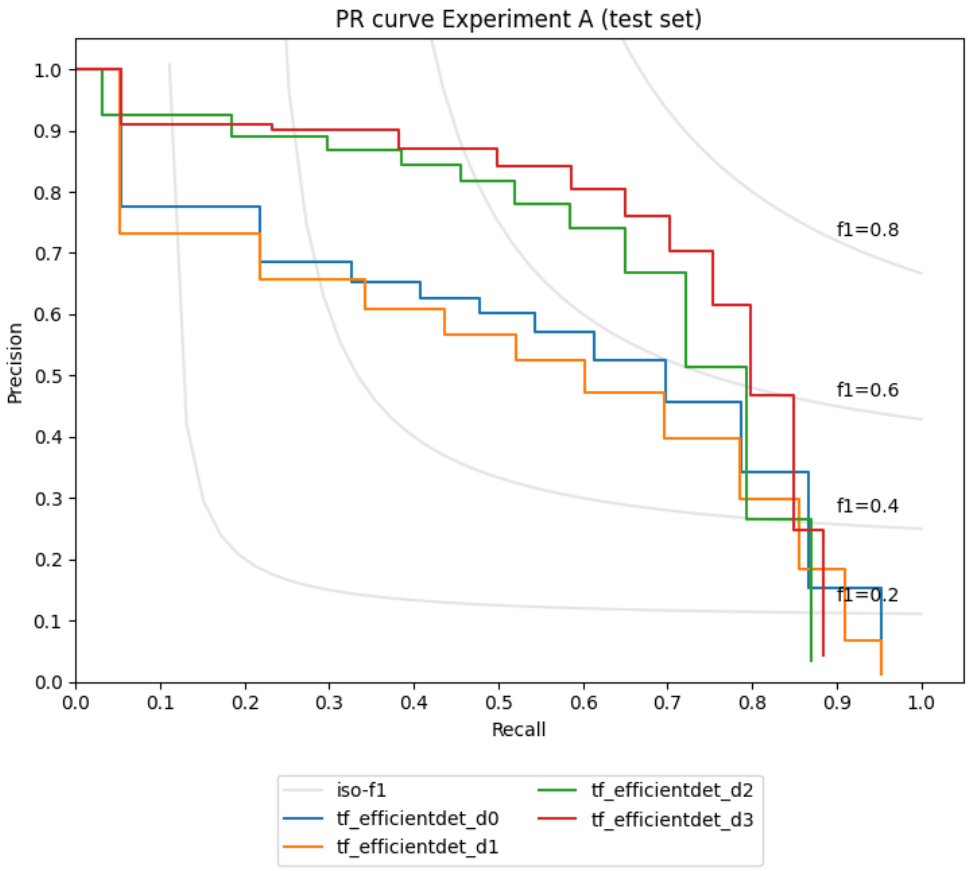


Fig. 18: EfficientDet PR curve experiment A on the test set.

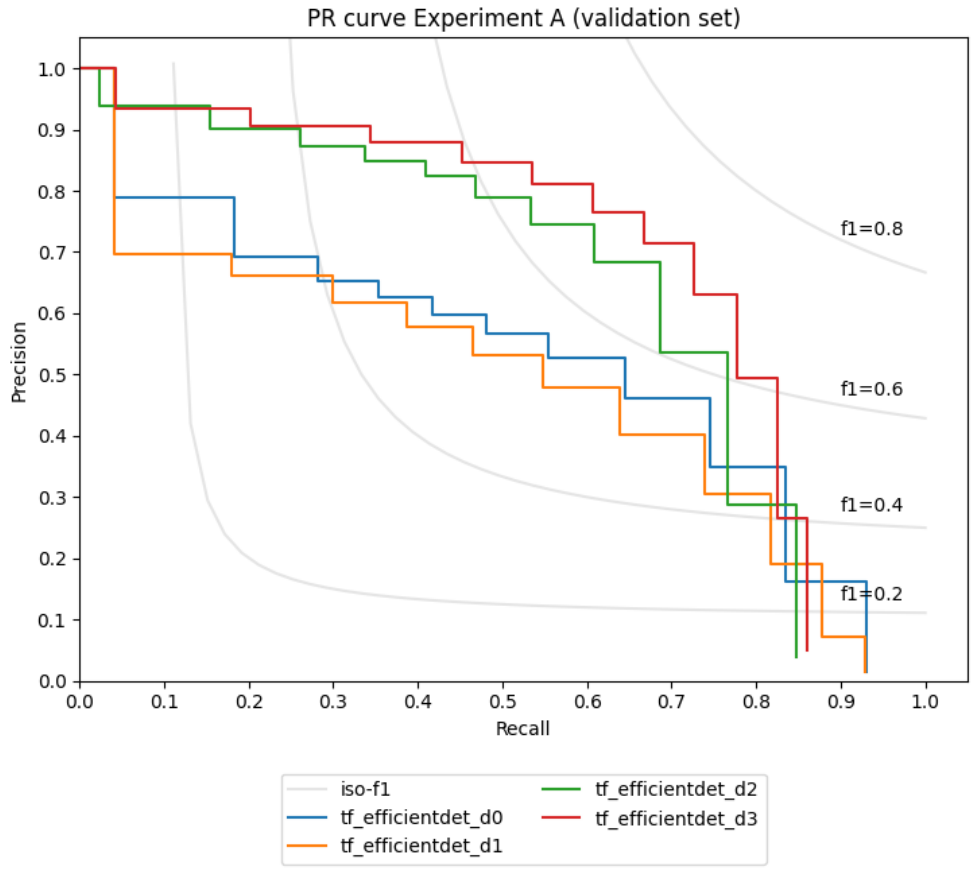


Fig. 19: EfficientDet PR curve experiment A on the validation set.

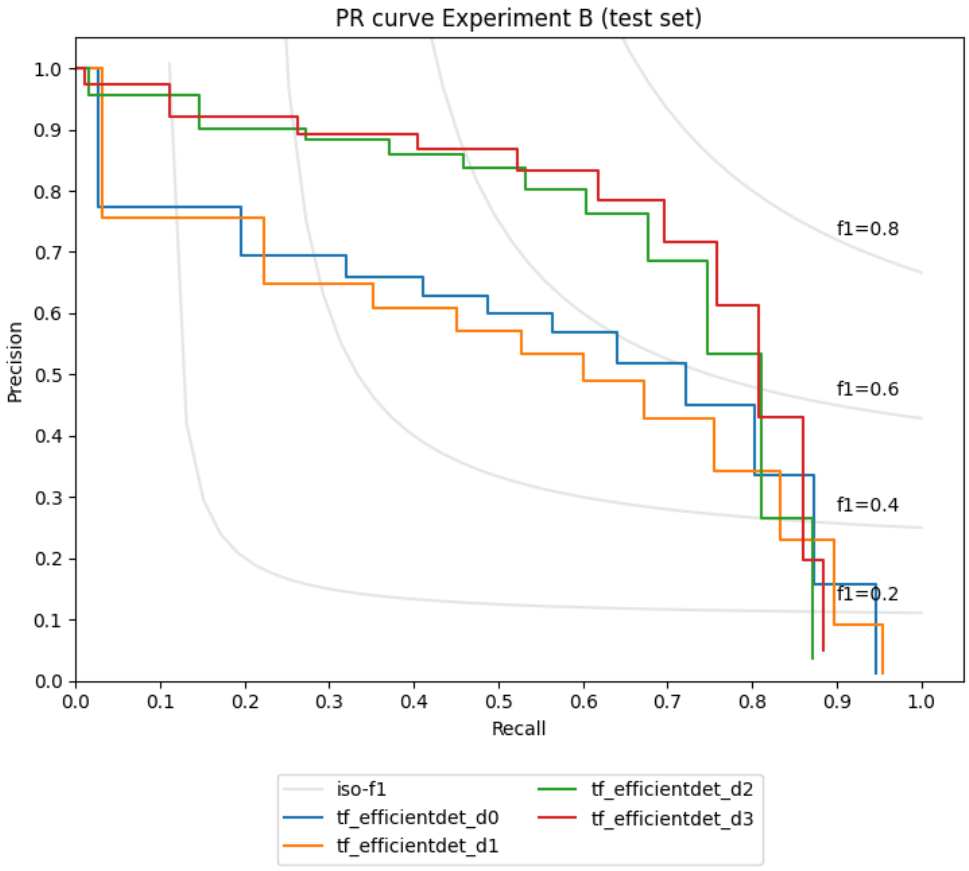


Fig. 20: EfficientDet PR curve experiment B on the test set.

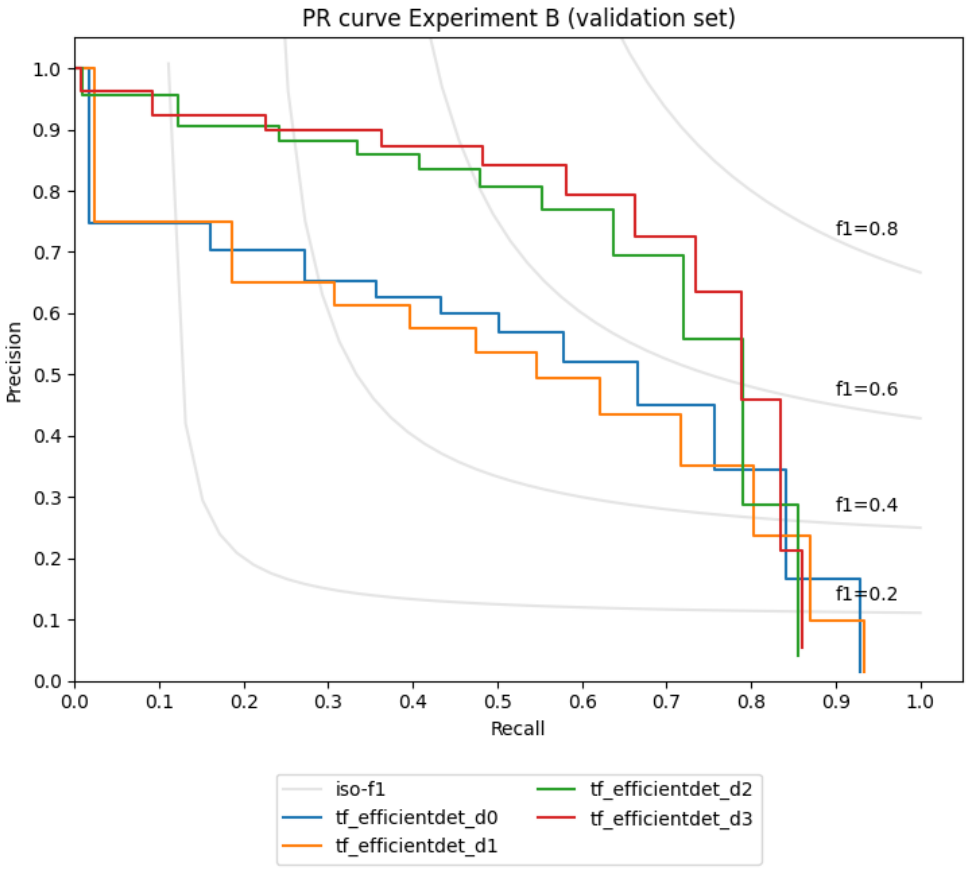


Fig. 21: EfficientDet PR curve experiment B on the validation set.

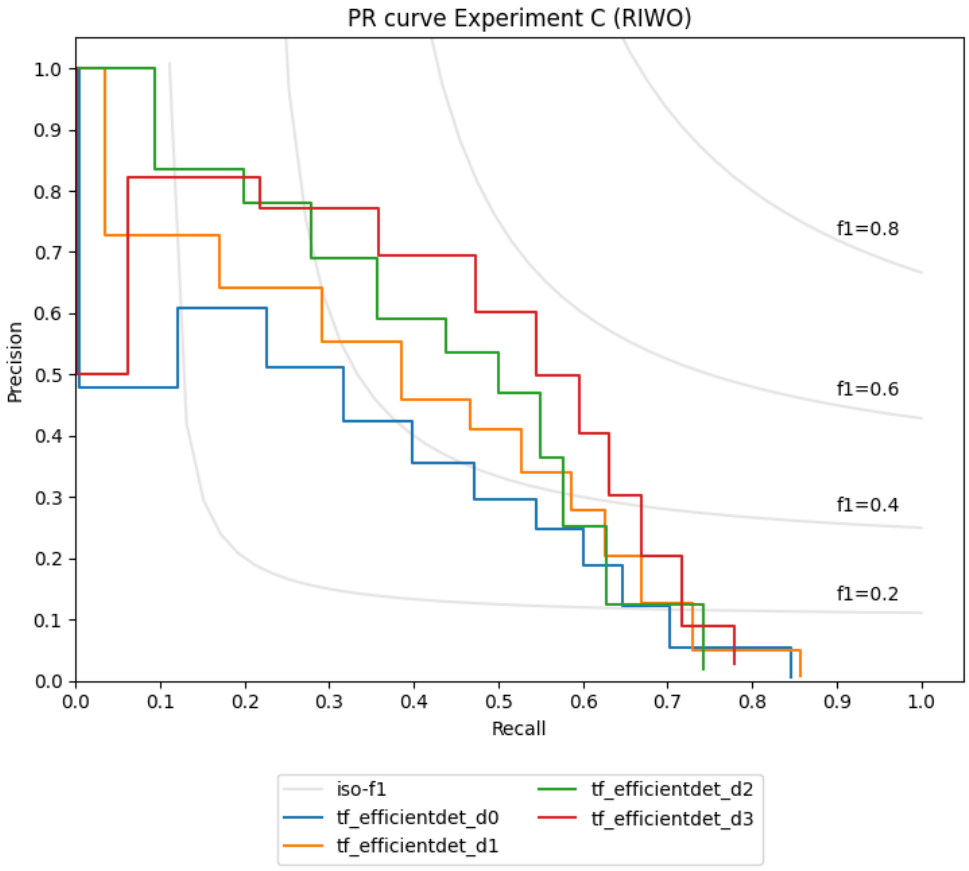


Fig. 22: EfficientDet PR curve experiment C on the RIWO dataset.

B.3 Faster-RCNN

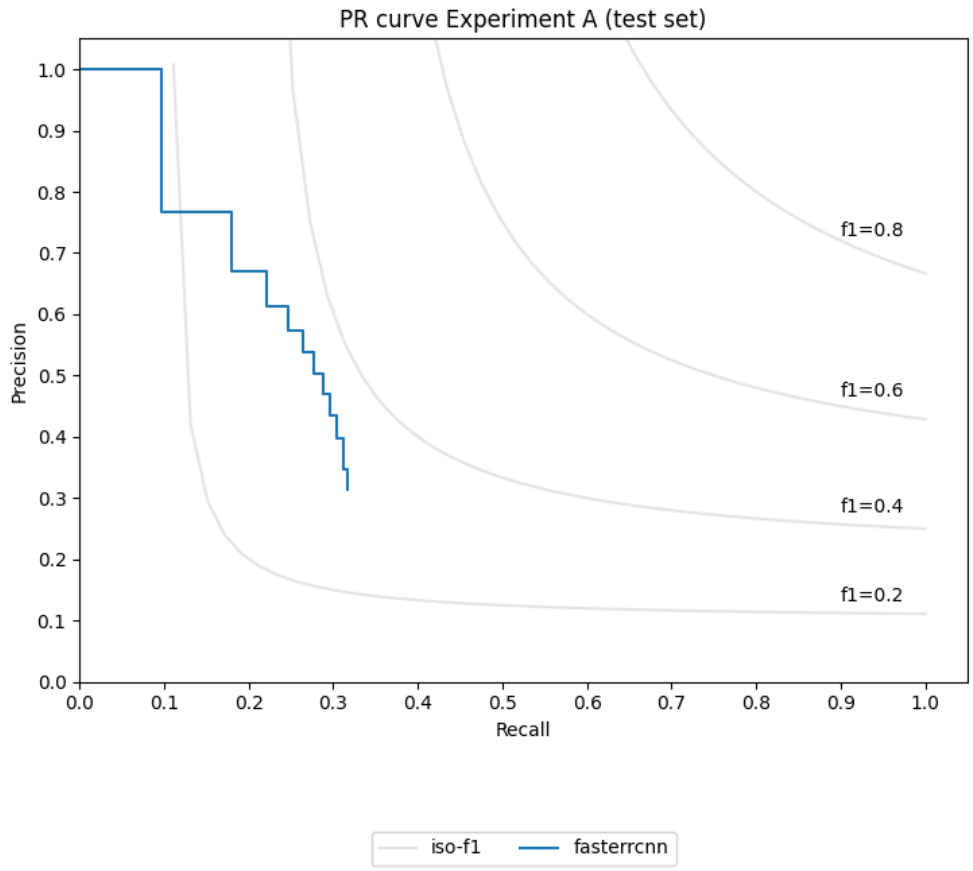


Fig. 23: Faster-RCNN PR curve experiment A on the test set.

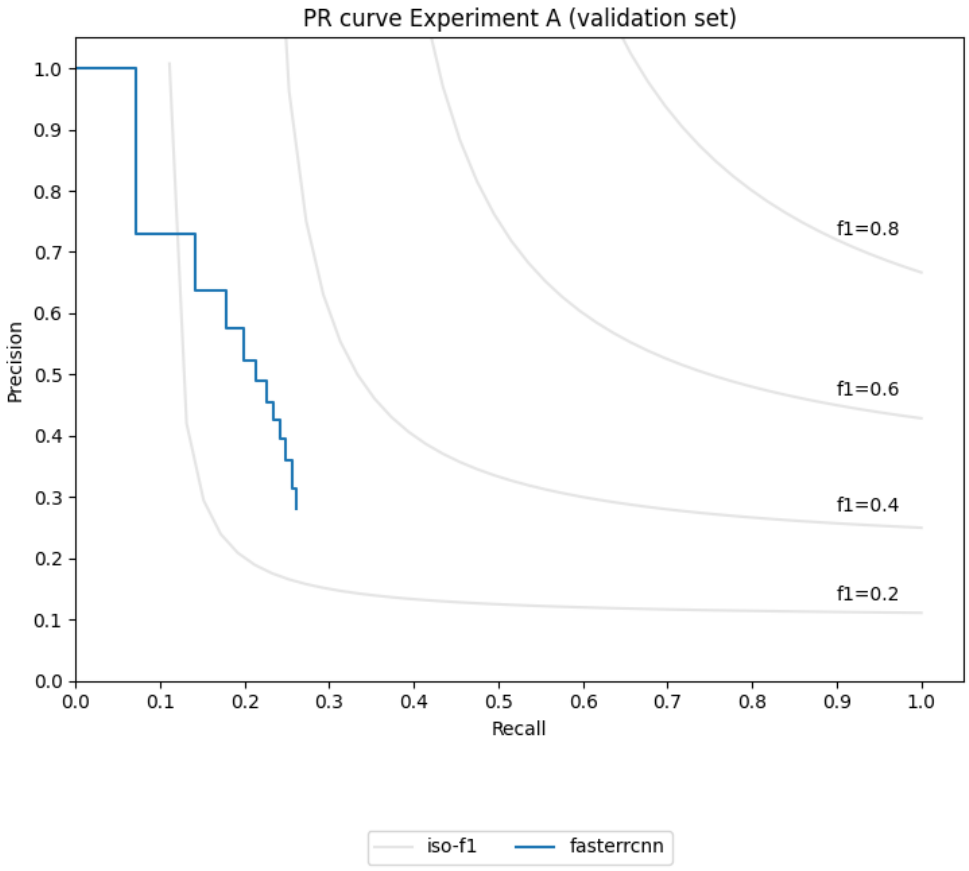


Fig. 24: Faster-RCNN PR curve experiment A on the validation set.

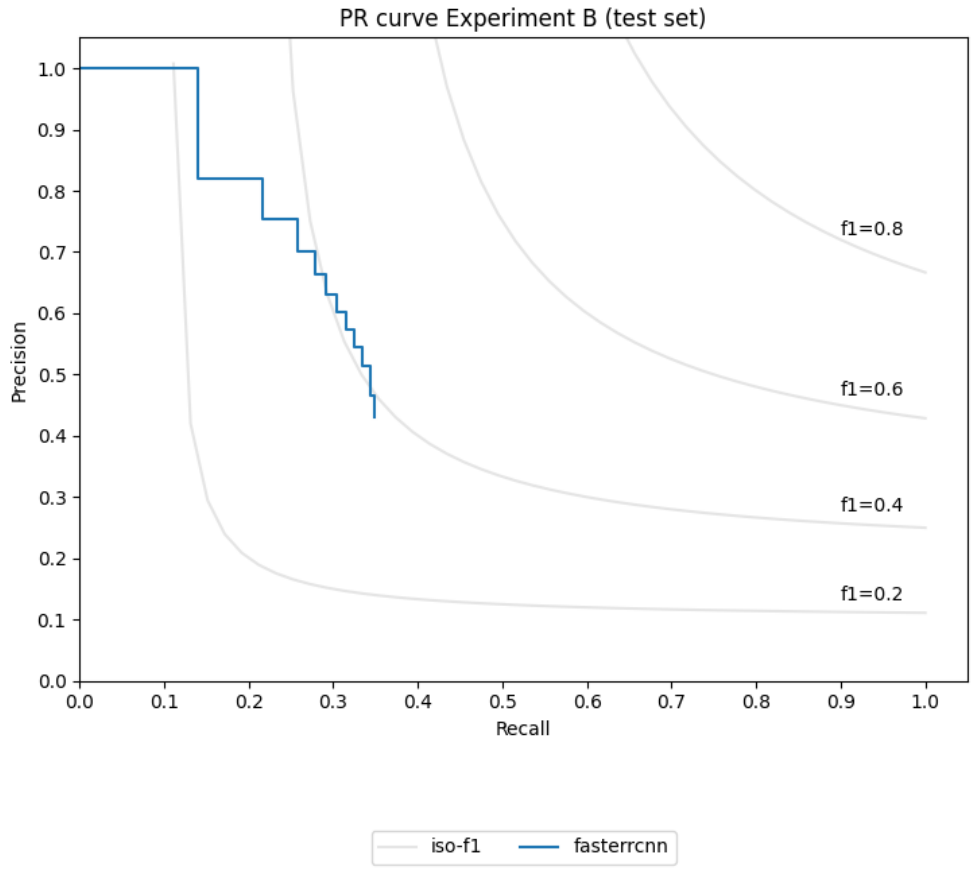


Fig. 25: Faster-RCNN PR curve experiment B on the test set.

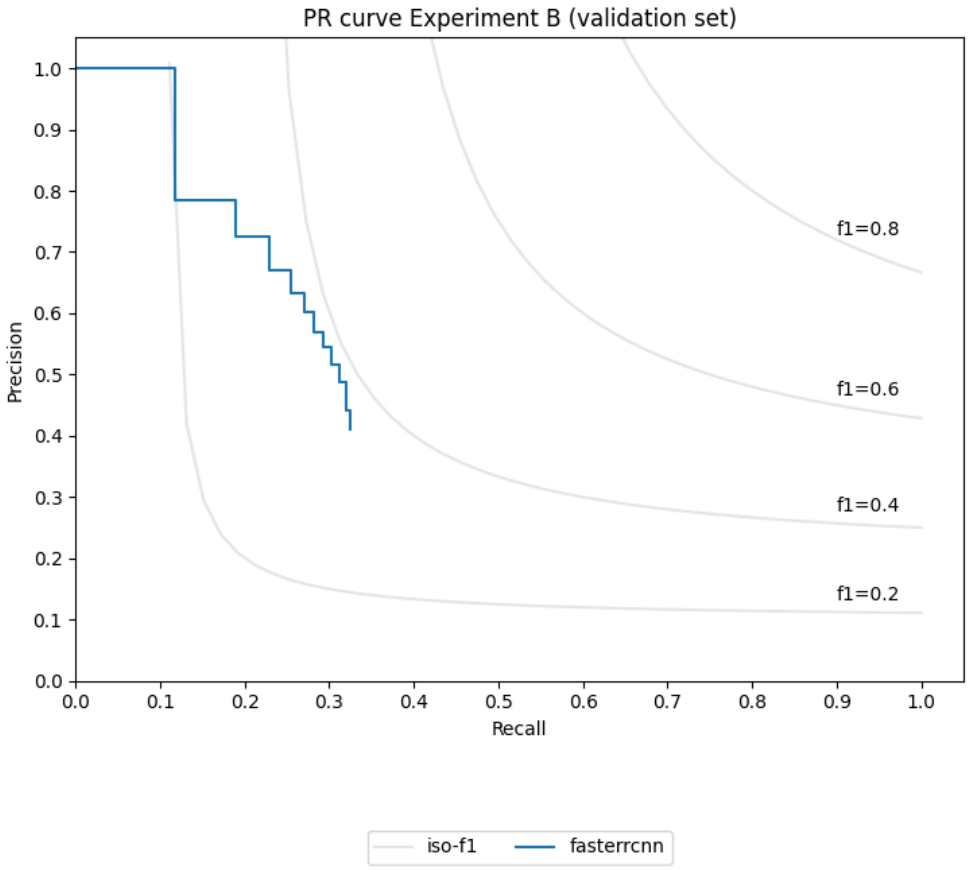


Fig. 26: Faster-RCNN PR curve experiment B on the validation set.

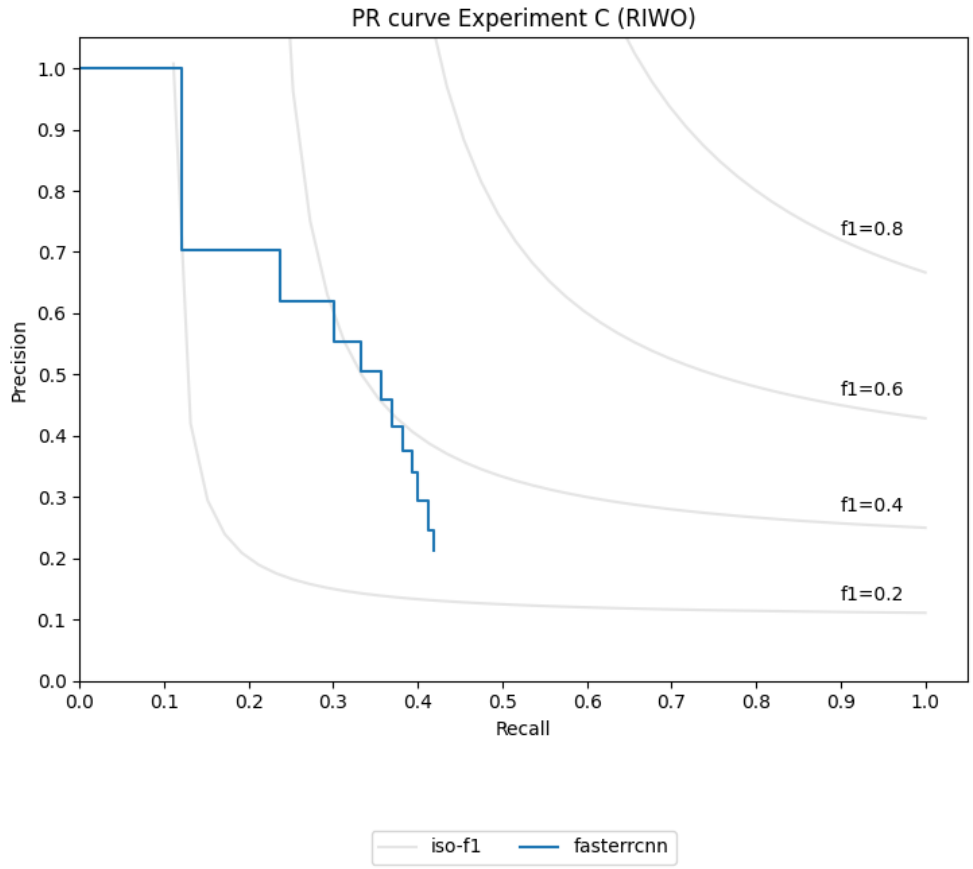


Fig. 27: Faster-RCNN PR curve experiment C on the RIWO dataset.