

On the scalability of CNNs for apple detection using RGBD data

NHL Stenden Lectoraat in Computer Vision & Data Science
This concept paper is under review by the supervisors.

Koen Molenaar

Supervisors: Maya Aghaei Gavari, Willem Dijkstra

Abstract—Picking apples is a highly repetitive task that can be automatized using AI. In these tasks, the importance of portable, and low cost devices becomes present. Because the robot should be able to cover large distances outside, carrying big and expensive machines would not be productive and would create dangers. This robot requires the ability to locate the apples, which can be done using machine learning algorithms like Neural Networks. Convolutional Neural Networks are often used for these tasks involving images. Among portable devices specially made for these tasks, there is the Jetson Nano, a device created by Nvidia. We trained multiple models using a dataset of apples including depth values. We also augmented the data so the model would generalize better. We found that using depth data improves the F1-score of the predictions of the models by a few percent consistently. Using augmentations, however, did not result in such conclusive results. Overall, the difference in performance between the models trained on RGB data, and augmented RGB data was negligible. When testing the performance of the models on the Jetson Nano, we found that the difference in inference time was often about 15x longer than on more powerful, non-portable machines. The usability of the Jetson Nano fully depends on the inference time required. This paper can be used as a basis for deciding the usability of the Jetson Nano on certain object detection tasks.

Index Terms—Object Detection, Neural Networks, Jetson Nano

1 INTRODUCTION

In the Netherlands, apple production is about 220 thousand tonnes per year [1]. Given that the average apple is 100 grams, this equates to 2,2 billion apples per year. Picking these is a highly repetitive and labor-intensive job, which gives a good opportunity for automation. During the automation process, the apples will be picked by a robot. To do that, the robot has to be able to detect and localize the fruit. This process can be done by using machine learning algorithms, for example, Neural Networks. There are different types of Neural Networks like MLP (multi-layer perceptrons), RNN (Recurrent Neural Networks), and CNN (Convolutional Neural networks). CNNs are often used for problems with images because they specifically extract image features like the shape of the objects. CNN's typically are comprised of a few types of layers: convolutional layers which extract features from the image, max-pooling layers which reduce the dimensions of the feature maps, and fully connected layers which learn from the features extracted. Depending on the final layer, there will be given a probability for each class of image it could be.

Detecting fruits in real-time with CNNs is often a computationally expensive task. To make the application of this task as accessible as possible, these networks should be tested on more portable and cheaper machines, like the Jetson Nano. This device by Nvidia can run multiple Neural Networks in parallel for applications like image classification, object detection, segmentation, and speech processing [2].

Besides normal cameras that save images in RGB format, there are 3D cameras, like the Microsoft Kinect Sensor [3], which also captures the depth data. In practice, when the fruit picking robot extends its arm, this depth data can be important in knowing how far the robot has to reach. Because of this, it would be interesting to see how the

additional depth data influences the quality of predictions using CNNs.

The goal of this paper is to test to what extent object detection models can be exported to cheaper, more portable devices like the Jetson Nano. This way processes like apple picking can be automated and be more widely used due to the accessibility.

1.1 Research Questions

In this paper, we aim to answer the question: **How well do object detection algorithms using CNNs perform on the Jetson Nano?** Besides, we want to find out whether the extra depth data or using augmented data will give better metrics in object detection.

More detailed, we want to answer the following questions:

- Which object detection model is the most accurate at detecting and localizing apples?
- Does using the available depth information give better results for detecting the apples?
- Does using augmentations on the images give better results for detecting the apples?
- To what extent can object detection models be exported to the Jetson Nano?

2 STATE OF THE ART

In this section, we will look at the existing papers on apple detection with neural networks. Here, we also look at existing papers on RGB-D data as the input to the selected object detection algorithms.

2.1 Object Detection for Apples

There have been a few research papers on the detection and localization of fruits. In this paper, [4] the authors suggest that the improved YoloV5 model gives higher scores and recognition speeds than other well-known models like EfficientDet-D0. What is missing in their work is testing the performance on smaller, less powerful devices.

The KFuji paper [5] describes how the researchers created a high-quality dataset containing images of orchards, including the extra depth dimension that can be used for object detection models.

- *Koen Molenaar is a Computing Science student at the NHL Stenden University of Applied Sciences, E-mail: koen.molenaar@student.nhlstenden.com.*
- *Maya Aghaei Gavari is a researcher at the NHL Stenden Lectoraat in Computer Vision & Data Science, E-mail: maya.aghaei.gavari@nhlstenden.com.*
- *Willem Dijkstra is a researcher at the NHL Stenden Lectoraat in Computer Vision & Data Science, E-mail: willem.dijkstra@nhlstenden.com.*

In our research, we will use this dataset for benchmarking a set of object detection models using different image modalities. The dataset contains images of apples in an apple orchard, with a total of 12,839 manually annotated apples and 967 multi-modal images, namely RGB, depth, and IR intensity. The KFuji paper describes the process of creating a dataset for doing object detection in orchards and provides test results on a Faster R-CNN network. The authors did not use the following object detection models that we would like to use:

EfficientDet [6]. EfficientDet is an algorithm that detects and recognizes various objects in an image. It has multiple scaling levels, meaning different sizes of the network are available.

YOLOv5 [7]. YOLO stands for You Only Look Once, this is also an algorithm that detects and recognizes various objects in a picture. It is similar to EfficientDet in that it has multiple scaling levels and thus different sizes.

3 MATERIALS & METHODS

This section describes the materials and methods used throughout the experiments, such as the dataset, hardware, and evaluation metrics

3.1 Dataset

This project makes use of the publicly available dataset KFuji RGB-DS [5]. The KFuji RGB-DS dataset contains multi-modal images of Fuji apples on trees collected with Microsoft Kinect v2. Since the performance of the depth sensor degrades when exposed to direct sunlight, all the images were taken at night under artificial lighting. Furthermore, the weather was clear at the time the images were taken. Each image includes information from three different modalities: color (RGB), depth (D), and range corrected intensity (S). All images were manually labeled with rectangular bounding boxes, resulting in 12,839 apples labeled across the dataset.

The dataset contains two types of data: raw data and pre-processed data. There are a total of 110 raw RGB images with sizes of 1080 x 1920 pixels. Each raw image is separated into nine tiles due to too many apples per image, and the fruit size is relatively small to the image size. A 3 x 3 grid of the original raw image is used to generate the tiles. After slicing, the pre-processed images have a size of 373 x 548 pixels (h x w). There is a 20-pixel overlap between the sliced tiles and the raw images to avoid partially split apples at the boundaries of different tiles. Images captured while slicing where there are no apples will not be saved. Eventually, there were 967 pre-processed images. The manually annotated bounding boxes for the 967 images are calculated. There are a total of 13,385 annotations for the pre-processed images. The difference with the 12,839 annotations in the raw images is the overlapping apples in the pre-processed images.

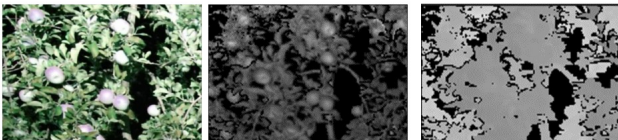


Fig. 1: RGB, depth and IR intensity respectively

An example of an RGB and depth image is shown in Figure. The color image with apples in orchards has three channels: red, green, and blue. The depth image has a single channel that only contains information about the distances to the camera. The values of the depth data were normalized between values of 0 and 255. This normalization is desirable to ensure fast convergence of the neural network

In total there were 110 multi-modal images which the authors tiled into 9 images of 373 x 548. If there were no apples in that section of the image, they left it out. After the tiling, there are a total of 967 images. Each of these images has been annotated with the bounding boxes of the apples.

In [8] the same authors of [5] used a split ratio for training, validating, and testing their models of 64:16:20, which results in 619,

155, and 193 images respectively. Because we want to compare our models with theirs, we use the same training, validation, and testing set.

3.2 Detection Algorithms

This section describes the different detection algorithms which we will be using and their architectures.

3.2.1 YOLOv5

YOLO is an abbreviation for "You Only Look Once". It is a family of deep learning models designed for fast object detection using CNNs. The YOLOv5 model architecture is based on YOLOv3 with some differences. YOLOv3 [9] exists of 53 CNN layers (Darknet-53). On top of that, there are 53 other layers for the detection part. On the other hand, the YOLOv5 architecture [7] consists of three parts: (1) backbone: CSPDarknet, (2) neck, PANet, and (3) head: YOLO layers. The backbone is used for feature extraction, the neck is used for feature fusion, and the head is used for the detection outputs.

YOLO uses anchor techniques to decide whether predictions are actual objects. Both YOLOv3 and YOLOv5 predict off-sets from a pre-determined set of boxes with specific height-width ratios. These anchor boxes are optimized on the COCO dataset. However, in the repositories, there is the option 'auto-anchor', which optimizes the anchor boxes for the specific dataset used.

The YOLOv5 repository contains four pre-defined models to choose from, namely YOLOv5s, m, l, and x with YOLOv5s being the smallest and YOLOv5x being the largest. The differences between these have to do with the scaling multipliers of the model, which scales the network's width and depth. The YOLOv5 models will downsample the input data with a factor of 32, 16, and 8 at three different scaling levels. Therefore, the input image of these models must be divisible by 32 pixels.

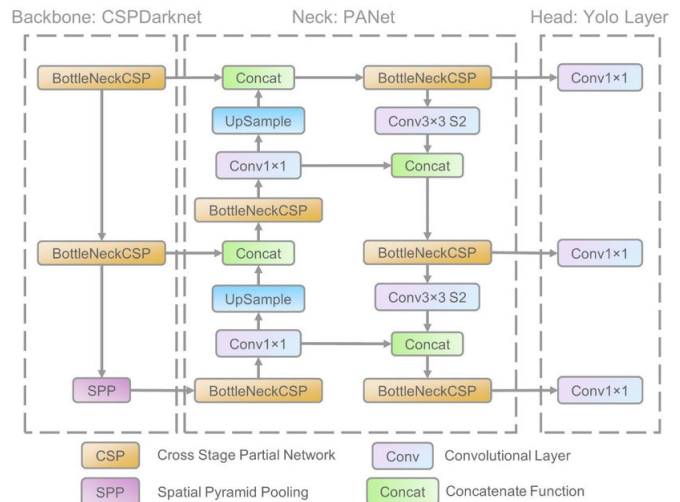


Fig. 2: YOLOv5 architecture [10]

3.2.2 EfficientDet

EfficientDet is a family of object detection models. In [6] the authors introduced a systematic way of model scaling which affects the network depth, width, and resolution. This can lead to better performance.

In figure 3 the architecture of EfficientDet is shown. It consists of three parts: (1) the backbone, (2) the feature network, and (3) the detection head.

BIFPN serves as a feature network for the EfficientDet models. It takes level 3-7 features (P3, P4, P5, P6, P7) from the backbone network which will be fused using a top-down and bottom-up bidirectional feature fusion. These fused features are then passed to

the head of the model. The head generates coordinates for the bounding boxes and the predictions for the object classes.

EfficientDet can use compound coefficients from 0 to 7, which decides the level of scaling. The higher the number, the bigger the model.

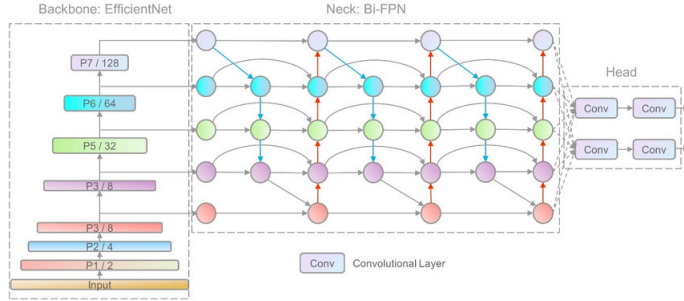


Fig. 3: EfficientDet architecture [10]

3.3 Metrics

To compare each network, we need to determine how each model performs. For this, we need quantitative metrics. In this section, we explain which metrics we used, and what they mean. They all build on top of TP, FP, FN, and FN.

- **True Positive (TP)** are the cases that have been predicted as an apple, and there is an apple.
- **False Positive (FP)** are the cases that have been predicted as an apple, and there is no apple in reality.
- **False Negative (FN)** are the cases that have been predicted as not an apple, and there is no apple in reality.
- **False Negative (FN)** are the cases that have been predicted as not an apple, and there is an apple.

Precision is a measure that tells, from the current predictions, how many of them are correct. It is calculated by dividing the correct positive predictions by the total of positive predictions. See equation 1.

$$Precision = \frac{Correct\ positive\ predictions}{Total\ positive\ predictions} = \frac{TP}{TP + FP} \quad (1)$$

Recall is an indicator that represents the percentage of correct positive predictions over all the positive labels. It is calculated by dividing the correct positive predictions by the total of positive labels. See equation 2.

$$Recall = \frac{Correct\ positive\ predictions}{All\ positive\ labels} = \frac{TP}{TP + FN} \quad (2)$$

F1-score is the weighted average of precision and recall. It is calculated by multiplying the product of precision and recall times 2 and dividing that by the sum of precision and recall. See equation 3.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \quad (3)$$

Average Precision. Mean Average Precision (mAP) is the metric summarizing the precision-recall curve into a single value that represents the average of all precisions. The precision-recall curve shows the trade-off between the two metrics for different thresholds. In the case of our dataset, which only has one class, the average precision is calculated.

3.4 Software and Hardware Specifications

In Table 1 the hardware specification used in the experiments is shown. VM-CVDS was used for training and testing the object detection models. The Jetson Nano is solely used for testing the models.

Computer name	GPU	GPU RAM	RAM	CPU
VM-CVDS	GeForce RTX 2070	8 GB	16 GB	Intel i9-7960X
Jetson Nano	128-core Maxwell™	-	4 GB	Quad-core A57

Table 1: Hardware specifications

On VM-CVDS all programming was done with Python 3.8.5 whereas, on the Jetson Nano, Python 3.6.9 was used. Table 2 provides an overview of the essential Python packages. Data processing was primarily done with OpenCV and NumPy. PyTorch 1.8.1 is used for the handling of the object detection models including training, validating, and testing.

Package	Version VM-CVDS	Version Jetson Nano	URL
NumPy	1.19.5	1.19.4	numpy.org
OpenCV-python	4.5.1.48	4.1.1	opencv.org
SciPy	1.6.0	1.5.4	scipy.org
Matplotlib	3.3.3	3.3.4	matplotlib.org
PyTorch	1.7.1	1.9.0	pytorch.org

Table 2: Relevant Python packages

4 EXPERIMENTS & RESULTS

In this section, we will explain which experiments we ran. The experiments will compare different object detection models discussed in section 3.2. With these models, we intend to answer the question of whether using the extra depth input results in better metrics. The other question is whether using augmentations within this dataset is helpful for getting better detection results. After this, we want to find out to what extent these models can be exported to the Jetson Nano.

4.1 Experiment A

In experiment A, we intended to find the best object detection model on our dataset. The utilized CNN architectures include YOLOv5 with sizes S, M, and L, and EfficientDet with compound coefficients D0, D1, D3, and D4. For all the EfficientDet models, we use EfficientNet-B0 as the backbone of the model. The data will not be augmented, meaning that apart from a resize and normalization, the images will not change. We expect that bigger models perform better than the models with smaller sizes in both networks, as they did in their respective papers [9, 6]. For YOLOv5, the input size was 608x608, and the input size for EfficientDet was 512x512.

4.1.1 Results from Experiment A

The best models per learning rates are shown in Table 3. Highlighted are the best model of each type of object detection model. YOLOv5s with a learning rate 0,001 achieved the highest F1-score of 0,853. For EfficientDet the best model was EfficientDet D0 with a learning rate of 0,001. Our expectation was that the larger versions of models would achieve better results. But since our object detection problem is relatively easy, only one class is present in very similar images, a more complex model was not necessary. The same can be found for EfficientDet, the smaller models slightly outperformed the larger ones. In [8] the authors trained this model on Faster R-CNN with optimized hyperparameters and custom anchor aspect ratios and got an F1-score of 0,867. Thus, Faster R-CNN outperformed both YOLOv5 and EfficientDet on this dataset. The complete results of experiment 1 can be found in Appendix A.

Model	Size	mAP	Recall	Precision	F1
YOLOv5	s	0.796	0.798	0.916	0.853
YOLOv5	m	0.786	0.798	0.899	0.845
YOLOv5	l	0.798	0.742	0.935	0.827
EfficientDet	D0	0.713	0.784	0.896	0.836
EfficientDet	D1	0.713	0.778	0.894	0.832
EfficientDet	D2	0.712	0.784	0.865	0.823
EfficientDet	D3	0.716	0.774	0.9	0.832
EfficientDet	D4	0.712	0.788	0.871	0.827
Faster R-CNN [8]	-	0.927	0.888	0.847	0.867

Table 3: Results of training object detection models on the RGB dataset. (the learning rates used, were all 0.001)

4.2 Experiment B

With experiment B, we intended to answer the question about the influence of augmentations. Here we trained and tested object detection models on the Kfuji dataset, but this time with augmentations. Because object detection models generalize better with augmentations, we expect the results to be slightly better than in experiment 1 without augmentations. The following augmentations will be used in this experiment:

- MedianBlur (blur-limit 3, probability 0.3) (See Figure 4c)
- RandomBrightnessContrast (brightness-limit 0.2) (See Figure 4c)
- HueSaturationValue (hue-shift-limit 3, probability 0.3) (See Figure 4c)
- HorizontalFlip (probability 0.5, shared base probability of 0.3 with ShiftScaleRotate) (See Figure 4c)
- ShiftScaleRotate (rotate-limit -15 15, probability 0.5, shared base probability of 0.3 with HorizontalFlip) (See Figure 4f)

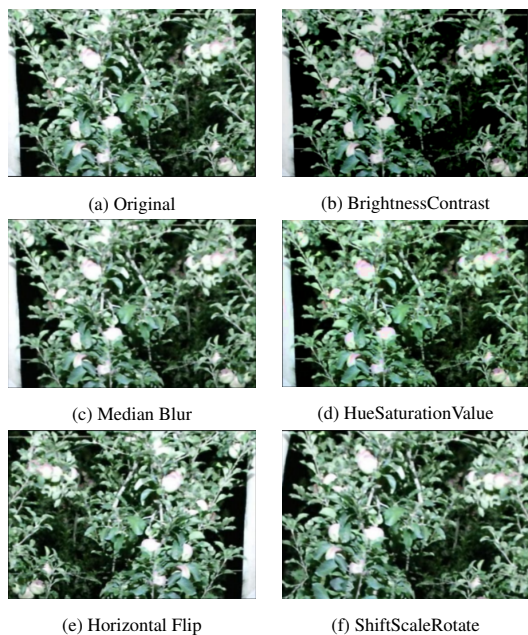


Fig. 4: Augmentations used in Experiment B

4.2.1 Results from Experiment B

The best object detection models trained on the augmented dataset are shown in Table 4. Highlighted are the best models per type of model. EfficientDet D2 gives in this case the best results with an

F1-score of 0.836. The model trained on just RGB with the same parameters scored an F1-score of 0.823 in experiment A. This would indicate that training the model on the augmented version of this dataset gives better results. However, all the other models performed slightly better when only trained on RGB data, though the difference between performance is practically negligible. The best YOLOv5 model here gives an F1-score of 0.846 whereas the same model got an F1-score of 0.853 in experiment A. The full results can be found in Appendix A.

Model	Size	Learning Rate	mAP	Recall	Precision	F1
YOLOv5	s	0.0001	0.795	0.778	0.926	0.846
YOLOv5	m	0.0001	0.779	0.764	0.932	0.839
YOLOv5	l	0.0001	0.793	0.716	0.95	0.816
EfficientDet	D0	0.001	0.705	0.805	0.856	0.83
EfficientDet	D1	0.001	0.697	0.803	0.859	0.83
EfficientDet	D2	0.001	0.695	0.807	0.868	0.836
EfficientDet	D3	0.001	0.712	0.775	0.896	0.831
EfficientDet	D4	0.001	0.611	0.736	0.909	0.814

Table 4: Results of training the object detection models on the augmented dataset

4.3 Experiment C

In experiment C, we wanted to see whether using RGB-D data instead of just RGB data causes an increase in metrics in object detection. We used the models described in Experiment A, but then changed the input shape in the model from 3 to 4. We expected a slight increase in metrics since the model has more data per image to learn from.

4.3.1 Results from Experiment C

The best object models trained on the RGBD dataset can be found in Table 5. The best YOLOv5 model was made with the size small and a learning rate of 0.0001. That model got an F1-score of 0.867. In Experiment A, the best YOLOv5 model got an F1-score of 0.853. This indicates that using the extra depth data with training the model, leads to better predictions. The same can be found for EfficientDet. The best EfficientDet model in experiment A got an F1-score of 0.836, whereas with this experiment it got an F1-score of 0.856. The complete results for experiment C can be found in Appendix A.

Model	Size	Learning Rate	mAP	Recall	Precision	F1
YOLOv5	s	0.0001	0.801	0.812	0.931	0.867
YOLOv5	m	0.0001	0.801	0.811	0.917	0.861
YOLOv5	l	0.001	0.804	0.795	0.939	0.861
EfficientDet	D0	0.001	0.798	0.85	0.859	0.854
EfficientDet	D1	0.001	0.8	0.833	0.881	0.856
EfficientDet	D2	0.001	0.714	0.819	0.868	0.843
EfficientDet	D3	0.001	0.714	0.789	0.908	0.845
EfficientDet	D4	0.001	0.714	0.818	0.876	0.846

Table 5: Results of training the object detection models on the RGB-D dataset

4.4 Experiment D

In experiment D, we wanted to measure the performance of the best models from experiments A, B, and C. on the various machines and their respective GPUs. The goal here was to see to what extent object detection models can be exported to the Jetson Nano. The metric we are looking for is inference time, which answers the question: how long does inference on one image take? Because the Jetson Nano's hardware is much less powerful than the hardware in the VM-CVDS (see Table 1), we expect the Jetson Nano to have worse inference times than on the VM-CVDS.

4.4.1 Results from Experiment D

In Table 6 the performance results of the object detection models on the different devices are shown. The inference times were collected

after the first 5 images since it would give both devices time to start their processes. As shown in Appendix B, the metrics are almost identical compared to Experiment A, indicating the predictions remain of the same quality when running the models on different machines. Highlighted are the best models from experiment A. The highlighted YOLOv5 model shows a mean inference time of 7ms on VM-CVDS with a standard deviation of 0.26. The mean inference time of that model was on the Jetson Nano 159ms with a standard deviation of 1,75ms indicating stable results over the full run, given that this is only 1.1% of its mean and on VM-CVDS it is 3.7% of its mean. Some EfficientDet models like `efficientdet0_3_0.001_4` had relatively high standard deviations of, in this case, 6ms, which would result in more than 20% higher inference times than the average. The difference between inference times on both devices is considerably large, on the Jetson Nano, inference time is often 15x larger than the inference time on VM-CVDS. For tasks where fast inference times are needed, the Jetson Nano would not be suitable, considering the fastest YOLOv5 model would give 6,2 frames per second. The complete results of experiment D can be found in Appendix B.

Model	Inf. VM	Std VM	Inf. JN	Std JN
efficientdet0_3_0.001	24	1.6	298	22.32
<code>efficientdet0_3_0.001_a</code>	25	6.03	300	9.78
<code>efficientdet0_4_0.001</code>	24	1.58	298	24.77
<code>efficientdet1_3_0.001</code>	25	1.34	315	21.43
<code>efficientdet1_3_0.001_a</code>	26	1.38	320	8.04
<code>efficientdet1_4_0.001</code>	27	1.44	317	11.95
<code>efficientdet2_3_0.001</code>	28	1.74	370	10.09
<code>efficientdet2_3_0.001_a</code>	60	1.74	603	11.54
<code>efficientdet2_4_0.001</code>	29	1.71	342	9.28
<code>efficientdet3_3_0.001</code>	31	1.28	433	15.46
<code>efficientdet3_4_0.001</code>	31	4.76	429	12.91
<code>efficientdet4_3_0.001</code>	34	1.56	552	14.15
<code>efficientdet4_4_0.001</code>	35	1.87	578	16.07
<code>yolov5l_3_0.001</code>	26	0.46	723	13.67
<code>yolov5l_3_0.001_a</code>	26	0.36	862	5.1
<code>yolov5l_4_0.001</code>	25	0.45	770	11.32
<code>yolov5m_3_0.001</code>	13	0.54	388	24.84
<code>yolov5m_3_0.001_a</code>	13	0.21	388	28.24
<code>yolov5m_4_0.001</code>	13	0.38	393	19.65
yolov5s_3_0.001	7	0.26	159	1.75
<code>yolov5s_3_0.001_a</code>	8	0.34	159	4.42
<code>yolov5s_4_0.001</code>	8	0.4	163	15.7

Table 6: Results of testing the object models on the Jetson Nano on inference time per image (values in ms). The model name follows the notion `ab_c_d_e` where a implies the type of model, b implies the model size, c implies the number of channels used (3 for RGB, 4 for RGBD), d implies the learning rate, and e implies whether the data is augmented or not

5 CONCLUSION & DISCUSSION

The goal of this research was to see how well object detection algorithms using CNNs perform on the Jetson Nano, a portable device designed for running Neural Networks. For this, we formulated the following questions:

- Which object detection model is the most accurate at detecting and localizing apples?
- Does using the available depth information give better results for detecting the apples?
- Does using augmentations on the images give better results for detecting the apples?
- To what extent can object detection models be exported to the Jetson Nano?

To answer these, we set up four experiments, A to D. First, we wanted to answer the question to which object detection model

performed the best on our dataset. In this case, that was YOLOv5s with a learning rate of 0,001 (see Table 3). For each type of model, YOLOv5, and EfficientDet, we ran using each of their respective sizes, and the learning rates 0,001 and 0,0001. These results set a baseline for the coming experiments.

The next question was whether altering the dataset with augmentations would improve the predictions of the models. The augmentations we chose are listed in section 4.2. This resulted in the best YOLOv5 model having a slightly lower F1-score compared to the baseline we set in Experiment A. The best EfficientDet model however had a slightly higher score than in Experiment A, indicating that augmentations have some effect on the predictions. But when we take into account the magnitude of difference between these results and the results from RGB, we can conclude that the augmentations did not affect the quality of predictions. This could be because the dataset did not have enough variety for the augmentations to have any effect. The YOLOv5 algorithm included the default built-in augmentations, which could have affected these results. This could further have been investigated by turning those off and comparing the results.

In the dataset, extra data was available, namely depth and IR intensity values. In this project, we only looked at the effect of using the extra depth data in combination with the regular RGB data. On all the tested models found in Table 5, the usage of depth data caused an increase in metrics. This can be understood by knowing some apples were not visible using only the RGB data. But if you look at the depth data in a greyscale form of the same picture, the outlining of some apples becomes visible (see Figure 1).

The last question was regarding the performance in terms of inference speeds. We made a performance script, loading the images one by one and feeding them into each object detection model (See Table 6). As we see there was a large difference between average inferences times between the machines, on the Jetson Nano, the inference process often took 15x longer. The reason behind this is that the machine VM-CVDS has much more resources available such as memory, and has more CUDA cores to utilize (See Table 1).

6 FUTURE WORK

We think that the next steps in this research are to use smaller models and optimize hyperparameters. As shown in Experiment A, using smaller models, does not result in worse results. In this case, it results in quite the opposite. Now, this does not mean any smaller model would work, but there are options like YOLOv5n, which is 4 times smaller than YOLOv5s but remains with the same YOLOv5 architecture. It just uses a smaller level of scaling. The fact that in all cases, the best model has been found within the first 10 epochs, only strengthens the suggestion that lesser complex and smaller models can be used in order to get lower inference times and get similar predictions.

As stated before, the authors in the KFuji paper [8] achieved better predictions using Faster R-CNN. However, they had to do several things to achieve this. One thing they did was tuning the hyperparameters. The next step in this research is to do the same with the hyperparameters of the YOLOv5 and EfficientDet models.

The augmentations used were decided qualitatively. This means looking at the dataset seeing how the images differ from each other and looking for transformations to make these differences as small as possible. Potentially there are better combinations of augmentations to use on this dataset which could cause an increase in performance.

We used the additional depth data in a naive way, namely adding an extra input channel. There may be better ways to process this depth data, as shown in this paper [11]. There the authors compare the naive way of using depth data, with using a geocentric embedding derived from the depth data in addition to RGB data. This geocentric embedding contains disparity, height, and angle values. In their paper, they suggest that this is a more effective method of using depth data.

Finally, during the performance testing on the Jetson Nano, the device got very hot. This is what you could expect when running neural networks on the GPU, but this could lead to thermal throttling.

This means the GPU's performance is limited by the high temperatures to not overheat and damage components. For the Jetson Nano, there are fans available to cool down the device when turned on. In order for better performance on the Jetson Nano, this is an important step in the research on the applications of the Jetson Nano.

REFERENCES

- [1] Koen van Gelder. Netherlands: apple production 2020, Mar 2021.
- [2] Jetson nano developer kit, Apr 2021.
- [3] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE Multimedia*, 19(2):4–10, 2012.
- [4] Bin Yan, Pan Fan, Xiaoyan Lei, Zhijie Liu, and Fuzeng Yang, Apr 2021.
- [5] Jordi Gené-Mola, Verónica Vilaplana, Joan R. Rosell-Polo, Josep-Ramon Morros, Javier Ruiz-Hidalgo, and Eduard Gregorio. Kfuji rgb-ds database: Fuji apple multi-modal images for fruit detection with color, depth and range-corrected ir data. *Data in Brief*, 25:104289, 2019.
- [6] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection, Jul 2020.
- [7] Ultralytics. ultralytics/yolov5: Yolov5 in pytorch $\dot{\iota}$ onnx $\dot{\iota}$ coreml $\dot{\iota}$ tflite.
- [8] Jordi Gené-Mola, Verónica Vilaplana, Joan R. Rosell-Polo, Josep-Ramon Morros, Javier Ruiz-Hidalgo, and Eduard Gregorio. Multi-modal deep learning for fuji apple detection using rgb-d cameras and their radiometric capabilities. *Computers and Electronics in Agriculture*, 162, 2019.
- [9] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.
- [10] Renjie Xu, Haifeng Lin, Kangjie Lu, Lin Cao, and Yunfei Liu. A forest fire detection system based on ensemble learning. *Forests*, 12:217, 02 2021.
- [11] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, 2014.

A EXPERIMENT A-C FULL RESULTS

Below are the results for the Experiments A through C.

A.1 Models trained on RGB Data Results

In Tables 7 and 8, the results of Experiment A are shown. This includes which size of the model was used, the used learning rate, and the corresponding metrics.

Model	Size	Learning Rate	mAP	Recall	Precision	F1
YOLOv5	s	0.0001	0.8	0.772	0.929	0.843
YOLOv5	s	0.001	0.796	0.798	0.916	0.853
YOLOv5	m	0.0001	0.79	0.766	0.92	0.836
YOLOv5	m	0.001	0.786	0.798	0.899	0.845
YOLOv5	l	0.0001	0.785	0.832	0.854	0.843
YOLOv5	l	0.001	0.798	0.742	0.935	0.827

Table 7: Results of training YOLOv5 models with different parameters

Model	Size	Learning Rate	mAP	Recall	Precision	F1
EfficientDet	D0	0.0001	0.708	0.821	0.823	0.822
EfficientDet	D0	0.001	0.713	0.784	0.896	0.836
EfficientDet	D1	0.0001	0.706	0.775	0.854	0.813
EfficientDet	D1	0.001	0.713	0.778	0.894	0.832
EfficientDet	D2	0.0001	0.706	0.756	0.876	0.812
EfficientDet	D2	0.001	0.712	0.784	0.865	0.823
EfficientDet	D3	0.0001	0.706	0.754	0.877	0.811
EfficientDet	D3	0.001	0.716	0.774	0.9	0.832
EfficientDet	D4	0.0001	0.625	0.719	0.897	0.798
EfficientDet	D4	0.001	0.712	0.788	0.871	0.827

Table 8: Results of training EfficientDet models with different parameters

A.2 Models Trained on Augmented Data Results

In Table 9 and 10, the results of Experiment B are shown. This includes which size of the model was used, the used learning rate, and the corresponding metrics. The augmentations used can be found in Section 4.2.

Model	Size	Learning Rate	mAP	Recall	Precision	F1
YOLOv5	s	0.001	0.791	0.746	0.94	0.832
YOLOv5	s	0.0001	0.795	0.778	0.926	0.846
YOLOv5	m	0.001	0.716	0.666	0.967	0.788
YOLOv5	m	0.0001	0.779	0.764	0.932	0.839
YOLOv5	l	0.001	0.794	0.698	0.953	0.806
YOLOv5	l	0.0001	0.793	0.716	0.95	0.816

Table 9: Results of training the YOLOv5 models on the RGB-D dataset

Model	Size	Learning Rate	mAP	Recall	Precision	F1
EfficientDet	D0	0.001	0.705	0.805	0.856	0.83
EfficientDet	D0	0.0001	0.696	0.805	0.845	0.825
EfficientDet	D1	0.001	0.697	0.803	0.859	0.83
EfficientDet	D1	0.0001	0.61	0.758	0.892	0.82
EfficientDet	D2	0.001	0.695	0.807	0.868	0.836
EfficientDet	D2	0.0001	0.617	0.739	0.91	0.816
EfficientDet	D3	0.001	0.712	0.775	0.896	0.831
EfficientDet	D3	0.0001	0.706	0.756	0.9	0.822
EfficientDet	D4	0.001	0.611	0.736	0.909	0.814
EfficientDet	D4	0.0001	0.62	0.731	0.916	0.813

Table 10: Results of training the YOLOv5 models on the RGB-D dataset

A.3 Models trained on RGB-D Data Results

In Table 11 and 12, the results of Experiment C are shown. This includes which size of the model was used, the used learning rate, and the corresponding metrics.

Model	Size	Learning Rate	mAP	Recall	Precision	F1
YOLOv5	s	0.0001	0.801	0.812	0.931	0.867
YOLOv5	s	0.001	0.801	0.81	0.923	0.863
YOLOv5	m	0.0001	0.801	0.811	0.917	0.861
YOLOv5	m	0.001	0.802	0.766	0.955	0.85
YOLOv5	l	0.0001	0.806	0.741	0.962	0.837
YOLOv5	l	0.001	0.804	0.795	0.939	0.861

Table 11: Results of training the YOLOv5 models on the RGB-D dataset

Model	Size	Learning Rate	mAP	Recall	Precision	F1
EfficientDet	D0	0.0001	0.714	0.803	0.877	0.838
EfficientDet	D0	0.001	0.798	0.85	0.859	0.854
EfficientDet	D1	0.0001	0.71	0.794	0.849	0.82
EfficientDet	D1	0.001	0.8	0.833	0.881	0.856
EfficientDet	D2	0.0001	0.713	0.811	0.848	0.829
EfficientDet	D2	0.001	0.714	0.819	0.868	0.843
EfficientDet	D3	0.0001	0.714	0.753	0.905	0.822
EfficientDet	D3	0.001	0.714	0.789	0.908	0.845
EfficientDet	D4	0.0001	0.711	0.763	0.891	0.822
EfficientDet	D4	0.001	0.714	0.818	0.876	0.846

Table 12: Results of training the YOLOv5 models on the RGB-D dataset

B EXPERIMENT 4 FULL RESULTS

In Table 13 and 14, the performance results of the models on the machines are shown. This includes the model used, the inference time, the standard deviation, and the corresponding metrics. The name of the models are built as follows: type of model, size, 3 for RGB and 4 for RGB-D, learning rate, and a for augmented or not.

B.1 Performance on the Jetson Nano

In Table 13 the performance results on the Jetson Nano are shown.

Model	Inf.	Std	Recall	Precision	AP	F1-score
efficientdet0_3.0.001	298	22.32	0.773	0.891	0.712	0.828
efficientdet0_3.0.001_a	300	9.78	0.806	0.848	0.702	0.827
efficientdet0_4.0.001	298	24.77	0.824	0.841	0.789	0.832
efficientdet1_3.0.001	315	21.43	0.756	0.886	0.706	0.816
efficientdet1_3.0.001_a	320	8.04	0.771	0.873	0.699	0.819
efficientdet1_4.0.001	317	11.95	0.824	0.862	0.713	0.842
efficientdet2_3.0.001	370	10.09	0.77	0.853	0.707	0.809
efficientdet2_3.0.001_a	603	11.54	0.768	0.885	0.695	0.823
efficientdet2_4.0.001	342	9.28	0.795	0.849	0.712	0.821
efficientdet3_3.0.001	433	15.46	0.758	0.891	0.71	0.819
efficientdet3_4.0.001	429	12.91	0.768	0.885	0.711	0.823
efficientdet4_3.0.001	552	14.15	0.781	0.844	0.705	0.811
efficientdet4_4.0.001	578	16.07	0.81	0.846	0.707	0.828
yolov5l_3.0.001	723	13.67	0.739	0.936	0.799	0.826
yolov5l_3.0.001_a	862	5.1	0.75	0.941	0.796	0.835
yolov5l_4.0.001	770	10.32	0.795	0.939	0.804	0.861
yolov5m_3.0.001	388	24.84	0.795	0.9	0.787	0.844
yolov5m_3.0.001_a	388	28.24	0.741	0.929	0.788	0.824
yolov5m_4.0.001	393	19.65	0.766	0.955	0.802	0.85
yolov5s_3.0.001	159	1.75	0.797	0.918	0.796	0.854
yolov5s_3.0.001_a	159	4.42	0.724	0.947	0.712	0.821
yolov5s_4.0.001	163	15.7	0.81	0.923	0.801	0.863

Table 13: Results of testing the object models on Jetson Nano on inference time per image (Std and Inf. in ms)

B.2 Performance on VM-CVDS

In Table 13 the performance results on VM-CVDS are shown.

Model	Inf.	Std	Recall	Precision	AP	F1-score
efficientdet0_3.0.001	24	1.6	0.776	0.892	0.713	0.83
efficientdet0_3.0.001_a	25	6.03	0.809	0.85	0.702	0.829
efficientdet0_4.0.001	24	1.58	0.833	0.84	0.789	0.836
efficientdet1_3.0.001	25	1.34	0.754	0.889	0.706	0.816
efficientdet1_3.0.001_a	26	1.38	0.775	0.871	0.701	0.82
efficientdet1_4.0.001	27	1.44	0.819	0.865	0.713	0.841
efficientdet2_3.0.001	28	1.74	0.768	0.853	0.705	0.808
efficientdet2_3.0.001_a	60	1.74	0.771	0.88	0.694	0.822
efficientdet2_4.0.001	29	1.71	0.795	0.847	0.712	0.82
efficientdet3_3.0.001	31	1.28	0.757	0.898	0.71	0.822
efficientdet3_4.0.001	31	4.76	0.771	0.883	0.707	0.823
efficientdet4_3.0.001	34	1.56	0.783	0.851	0.706	0.815
efficientdet4_4.0.001	35	1.87	0.806	0.847	0.708	0.826
yolov5l_3.0.001	26	0.46	0.742	0.935	0.798	0.827
yolov5l_3.0.001_a	26	0.36	0.753	0.94	0.795	0.836
yolov5l_4.0.001	25	0.45	0.795	0.939	0.804	0.861
yolov5m_3.0.001	13	0.54	0.798	0.899	0.786	0.845
yolov5m_3.0.001_a	13	0.21	0.741	0.927	0.787	0.824
yolov5m_4.0.001	13	0.38	0.766	0.955	0.802	0.85
yolov5s_3.0.001	7	0.26	0.798	0.916	0.796	0.853
yolov5s_3.0.001_a	8	0.34	0.726	0.945	0.712	0.822
yolov5s_4.0.001	8	0.4	0.81	0.923	0.801	0.863

Table 14: Results of testing the object models on VM-CVDS on inference time per image (Std and Inf. in ms)