TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Hans Miikael Uuspõld 153107IAPM

AUTOMATIC LABELING IN THE CONTEXT OF HISTORICAL IMAGES

Master’s thesis

Supervisor: Andri Riid
PhD

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Hans Miikael Uuspõld

13.05.2020

Abstract

This thesis aimed to integrate automatic labelling into a historical image archive using deep learning. While several pre-trained models exist, they have two drawbacks. First being that they are trained on modern images, and as the size and shape of objects has changed over time, then they aren’t necessarily capable of detecting the objects from historical images. Second being that they do not necessarily detect objects that are of interest in historical images.

Two tasks were performed to achieve this. First, creating an annotation solution that can be integrate into the website. Second, choosing an appropriate object detection algorithm.

Three questions were asked when choosing the object detection algorithm.

1) Which algorithm learns the fastest in terms of epochs?
2) How do the algorithms’ accuracies increase with additional data?
3) How fast is the detection of the different algorithms?

New models were trained using historical images in order to answer the questions. The algorithms used were Detectron2’s implementation of Faster R-CNN, YOLOv3, EfficientDet D0, and EfficientDet D2.

To evaluate how fast the algorithms learn, in terms of epochs, the accuracies of different models were evaluated using mAP @ 0.5 and mAP @ [0.5:0.95] after every 25 epochs, starting with 25 and ending with 150. This was done for every dataset size.

In order to see how the algorithms’ accuracies increase with additional data the following process was done. The dataset was evaluated after every 25 images. The dataset size was incremented by 25 images per class, starting from 25 until reaching 100 for a single class. Once 100 was reached for a single class, then an additional class was added and it started from 25 images per class. So, 2 classes, 25 images were 50 images total. The dataset sizes were [1…10] * {25, 50, 75, 100}.

Detection accuracies were measured in seconds. This was, however, a secondary parameter.

The algorithm which performed the best was Detectron2's implementation of Faster R-CNN. At certain dataset sizes EfficientDet D2 overtook it in detection accuracy by a small margin, but in terms of learning speed and generalizing from a smaller dataset Faster R-CNN was superior. YOLOv3 trailed close behind and EfficientDet D0 was too far behind in accuracy to consider. Thus, Detectron2's implementation of Faster R-CNN was chosen to be used in Ajapaik's environment.

This thesis is written in English and is 55 pages long, including 7 chapters, 64 figures and 5 tables.

Annotatsioon

Automaatne sildistamine ajalooliste piltide kontekstis

Antud lõputöö eesmärk oli integreerida automaatne sildistamine ajalooliste piltide arhiivi kasutades sügavõpet. Kuigi olemas on mitmeid välja õpetatud mudeleid, siis on neil kaks põhilist puudust antud kontekstis. Esitaks, kuna objektid on muutunud läbi ajaloo ning olemasolevad mudelid on treenitud tänapäevaste andmetega, siis ei ole need mudelid tingimata suutelised tuvastama neid samu objekte ajaloolistelt piltidelt. Teiseks, objektid, mida antud mudelid tuvastavad, ei ole tingimata kasulikud ajalooliste piltide kontekstis.

Edukaks integreerimiseks oli vaja täita kaks ülesannet. Esiteks oli vaja luua annoteerimissüsteem, mida saaks integreerida veebi keskkonda. Teiseks oli vaja valida sobiv algoritm antud kontekstile.

Sobiva algoritmi valimiseks püstitati kolm küsimust.

1) Milline algoritm õpib epohhe arvestades kõige kiiremini?

2) Kuidas muutuvad algoritmide täpsused kui lisatakse andmeid?

3) Kui kiiresti tuvastavad erinevad algoritmid pilte?

Uued mudelid treeniti välja kasutades ajaloolisi andmeid, et vastata esitatud küsimustele. Võrdlusteks valitud algoritmid olid Detectron2 implementatsioon Faster R-CNN algoritmist, YOLOv3, EfficientDet D0 ja EfficientDet D2.

Erinevate mudelite täpsusi hinnati iga 25 epohhi järel, et uurida kuidas muutub täpsus üle epohhide. Hindamine algas 25-st epohhist ning lõppes 150-ga. Täpsuse hindamiseks kasutati mAP @ 0.5 ja mAP @ [0.5:0.95]. Seda hindamist tehti iga andmestiku kohta.

Järgnev protsess oli kasutuses hindamaks täpsusi üle erinevate andmestike suuruste. Andmestikke hinnati iga kord kui lisati 25 pilti. Andmestiku suurust tõsteti 25 pildi võrra klassi kohta. Alustati 25-st pildist ning lõpetati 100-ga ühe klassi kohta. Peale seda, kui ühe klassi kohta oli jõutud 100 pildini, alustati protsessi uuesti 25-st, kuid nüüd lisati teine klass. Nüüd oli mõlema klassi kohta 25 pilti, ehk andmestiku suurus oli 50. Iga sammuga lisati *klasside arv * 25 pilti*. Kõik kasutatud andmestiku suurused olid [1...10] * {25, 50, 75, 100}.

Pildi töötlemiskiirust mõõdeti sekundites. See aga ei omanud sama suurt kaalu kui täpsused.

Algoritm, mis saavutas parimad tulemused, oli Detectron2 implementatsioon Faster R-CNN-st. Teatud andmestike suuruste juures sai EfficientDet D2 väikese eduga paremaid täpsuse tulemusi. Arvestades aga epohhide põhist õppimiskiirust ja võimet üldistada väikeselt andmehulgalt, siis Faster R-CNN oli parem. YOLOv3 tulemused olid lähedal Faster R-CNN ja EfficientDet D2 omadele. EfficientDet D0 jäi täpsuses aga liiga palju maha, et seda arvestada. Otsustati, arvestades eelnevalt toodud aspekte, et Detectron2 implementatsioon Faster R-CNN algoritmist oleks kõige parem valik Ajapaiga keskkonna jaoks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 55 leheküljel, 7 peatükki, 64 joonist, 5 tabelit.

List of abbreviations and terms

CNN	Convolutional neural network
YOLO	You Only Look Once
R-CNN	Regions with CNN features
RQ	Research question
mAP	Mean average precision
TP	True positive
FP	False positive
FN	False negative
TN	True negative
RoI	Region of interest
IoU	Intersection over union
BiFPN	Weighted bidirectional feature network
PANet	Path aggregation network
R101	ResNet-101
X101	RexNeXt-101
FPN	Feature pyramid network
AP	Average precision

Table of contents

List of figures

List of tables

1 Introduction

History is something that people should remember and learn from. It’s a subject of research and it’s taught in schools. Often, when discussing history in an academic environment, illustrations are provided – be it of drawings, objects, or even events. However, not everyone has their own historical images. Image archives exist for those purposes.

There are several image archives that deal with storing and labelling images. These include national ones and private ones. In the context of Estonia, an example of a national archive would be FOTIS [1], which is owned by Rahvusarhiiv [2]. On a smaller scale there is a collection of images for Saku county in their digital archive [3]. In addition to these national ones there are also private archives. For example, Vanadpildid [4] and Ajapaik [5].

The archives must have a search function in order to make navigating the large quantities of images an easier task. However, the images must get their labels somehow and labelling them manually can be a time-consuming task. As hardware has developed then certain automated approaches have become computationally feasible, namely convolutional neural networks.

Convolutional neural networks are a subset of machine learning that specializes in working with images. Several different networks and algorithms have been developed for object detection. Among them are YOLO [6], SSD [7], R-CNN [8], Faster R-CNN [9], EfficientDet [10] and many more. However, one algorithm doesn’t fit all cases, as they make trade-offs. Some are faster, while others are more accurate. Thus, the usage of these algorithms must be looked at case by case.

The context in which object detection algorithms will be applied within this thesis is the historical image archive Ajapaik [5]. It got started in 2011 and provides different features relating to historical photos, such as marking the locations on a map where the image was taken, and taking rephotos, that is taking a picture in the same location that the original image was in and then allowing a side-by-side comparison of what it looks like today and what it looked like on the original photo. CNNs could be used in order to make the grouping of its images easier. In order to integrate CNNs into the website two tasks must be done – develop an annotation system in order to have ground truths from which to train, and choosing the appropriate algorithm. As the number of historical photos is limited and as Ajapaik is a non-profit enterprise, then the hardware available is limited, thus there are two main problems for choosing the appropriate algorithm. The first is which algorithm can learn the fastest and the second is which algorithm can increase its accuracy the fastest from a limited amount of data. Finally, a less important aspect, but still considerable is the detection time of algorithms. So, the research questions would be as follow:

RQ1: Which algorithm learns the fastest in terms of epochs?

RQ2: How do the algorithms' accuracy increase with additional data?

RQ3: How fast is the detection of the different algorithms?

The following approaches will be taken in order to answer the above questions:

- Check the accuracy of the trained models after every 25 epochs, up until 150.
- Check the accuracy after every 25 images.
- Check the average time spent in seconds to process images by the algorithms.

Chapter 2 describes in more detail how the research questions will be answered. In Chapter 3 different available benchmark datasets will be described. Chapter 4 will describe the accuracy measurement metric and the algorithms chosen for the evaluation and why they were chosen. Chapter 5 will describe the dataset used for evaluations and the developed annotation solution. Chapter 6 will list the results of the evaluations.

2 Thesis outline

New models must be trained for historical images. The models that exist have two drawbacks when considering historical images. First is that they have been trained using modern images, thus they can misclassify historical images, as the shapes and sizes of objects have not remained consistent throughout history. Second is that they don’t detect objects that would be of interest in historical images.

In order to evaluate the models in the context of this thesis the following approach is used.

First, a large enough quantity of images is gathered using open image archives. FINNA [17] and FOTIS [1] are used for the data, as they are representative of Ajapaik. The open image archives are used due to release schedule conflicts with Ajapaik. In the production environment the data labelled by the users is used. The data gathered must be filtered for miscellaneous images. That is, images that are not related to a searched keyword. For example, when searching for tanks, the word might just be in the photographer’s name and have nothing to do with the image.

The filtered images must then be labelled with bounding boxes to have ground truth data for training. An appropriate annotation tool must be chosen for that.

Training of the models starts once the labelling process is finished. For the training process a total of 1000 images with labels are used, which will be incrementally used. As transfer learning is used, then a base model must be present. A pre-trained model from the COCO dataset [12] is used as the base model.

The training begins with a 25-image dataset, which contains a single class. During the training process, the model saves checkpoints after every 25 epochs. The training of a model is considered finished once 150 epochs are reached. Each of the checkpoints are then evaluated using mAP @ 0.5 and mAP @ [0.5:0.95]. The evaluation metrics are described in more detail in chapter 3.2.1. After evaluation 25 images are added to the dataset and the process is repeated until 100 images are reached.

Once 100 images of a class are reached, then it begins again from 25, but with an additional class. At the start of the cycle, 25 images per class are used. So, with 2 classes and 25 images, there are a total of 50 images in the dataset. Now the dataset gets incremented by 25 images per class, meaning that with 2 classes, the step size is 50 images, as each class contributes 25 images. This continues until 1000 images are reached as a total of 10 classes will be used. So, the dataset sizes will be [1…10] * {25, 50, 75, 100}.

Previously trained models with the historical image dataset are not reused for training. Each dataset’s training cycle begins with a fresh COCO pre-trained model.

The above process is done for every algorithm – YOLOv3, Faster R-CNN, EfficientDet D0, and EfficientDet D2.

3 Background and related work

Machine learning algorithms require a large amount of data to be as accurate as possible. In the context of CNNs that means having images and annotations for those images in order to perform a certain type of image recognition task. Some of these tasks are as follows:

- Image classification – it involves assigning a certain label to an image. An example of that would be seeing an image of a car and then assigning the car label to that image.

- Object detection – it involves assigning multiple labels to an image and then localizing them on the image by drawing a bounding box around them. An example of that would be seeing a car and then saying that the car's bounding box starts at pixels x_1, y_1 and ends at x_2, y_2.

- Object segmentation – like object detection, it assigns labels and localizes the objects. However, instead of simply drawing a bounding box, this type of detection draws an exact boundary. The segmentation boundary is described as a list of x, y point pairs.

In the context of this thesis the problem of automatically labelling images will be addressed as an object detection problem. This is because there can be several objects on the image. Additionally, users validate the annotations that have been added automatically or by other users, so for the users to clearly see what they're validating there needs to be a boundary. However, segmentation is not necessary, as an exact boundary doesn't add extra benefits to localizing the object in this context.

3.1 Evaluation datasets

Several open source data collections exist that have been developed for the training of machine learning algorithms. They consist of images and their respective annotations.

3.1.1 Common Objects in Context (COCO)

The COCO [12] dataset is one of the larger openly available datasets currently and thus a lot of projects use it to evaluate their algorithms. It has over 330000 images and 1.5 million instances of objects. It contains captions, segmentation data, bounding boxes, and person keypoints. Overall there are 80 object categories defined.

COCO also offers its own method for calculating the mean average precision. This is described in more detail in chapter 3.2.1.

3.1.2 PASCAL VOC

PASCAL VOC [14] used to be a standard dataset against which people tested their algorithms. However, as of 2012, it has been discontinued. Regardless, algorithms are still evaluated against it. The dataset is smaller than COCO with 20 classes, 11530 images, 27450 object annotations and 6929 segmentations.

3.1.3 ImageNet

This is the largest dataset of images. There are roughly 14.1 million images in the dataset. About a million of them have bounding box annotations [15].

3.1.4 Google Open Images

Google’s Open Images is the largest dataset with object location annotations. It contains roughly 9 million images annotated with different information. Specifically, there are 16 million bounding boxes with 600 object classes on 1.9 million images [16].

3.2 Evaluation metrics

There are different metrics that can be used to describe an algorithm’s performance. For example, precision and recall. However, most of the algorithms provide their performance in mean average precision (mAP), as it combines different metrics into one.

3.2.1 Mean average precision (mAP)

Several terms must be defined before defining mAP.

Intersection over union (IoU) shows how much did the bounding boxes of the ground truth and the algorithm's output match. It's calculated using the area of overlap divided by the area of union displayed in figure 1.

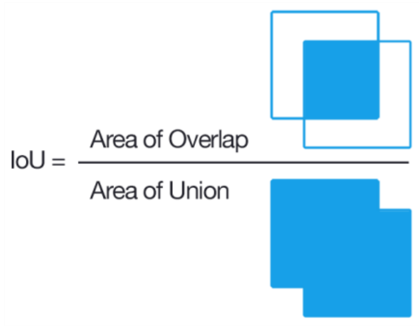


Figure 1. Intersection over Union calculation [11]

A threshold is chosen in order to classify a detection as either positive or negative based on the IoU. Anything above or equal to the threshold is considered a positive, while anything below it is considered a negative. Additionally, the type must be correct.

Positive and negative result must be defined when estimating the validity of a machine learning algorithm's results. There are four classifications – true positive (TP), false positive (FP), false negative (FN), true negative (TN).

- True positive means that a bounding box that was estimated by the algorithm overlaps with the ground truth bounding box above a certain IoU threshold.
- False positive means that a bounding box that was estimated by the algorithm was not present in the ground truth.
- False negative means that a bounding box existed in the ground truth but was not estimated by the algorithm or that the estimation was under a certain IoU threshold.
- True negative means that a bounding box was not estimated, and it did not exist in the ground truth either.

Precision (1) shows how much of the positive detections were truly positive detections.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

Recall (2) shows how much of the ground truth positives were defined as positive by the algorithm.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

Precision and recall can be calculated with the positive and negative detections. Their values will be used to calculate the average precision (AP). There are two ways to do this – using area under the curve approach or an N interpolated points. Both start with removing any zigzags from the precision/recall curve as can be seen in figure 2, where blue dots represent the recall/precision pairs and the green line represents the points without zigzags.

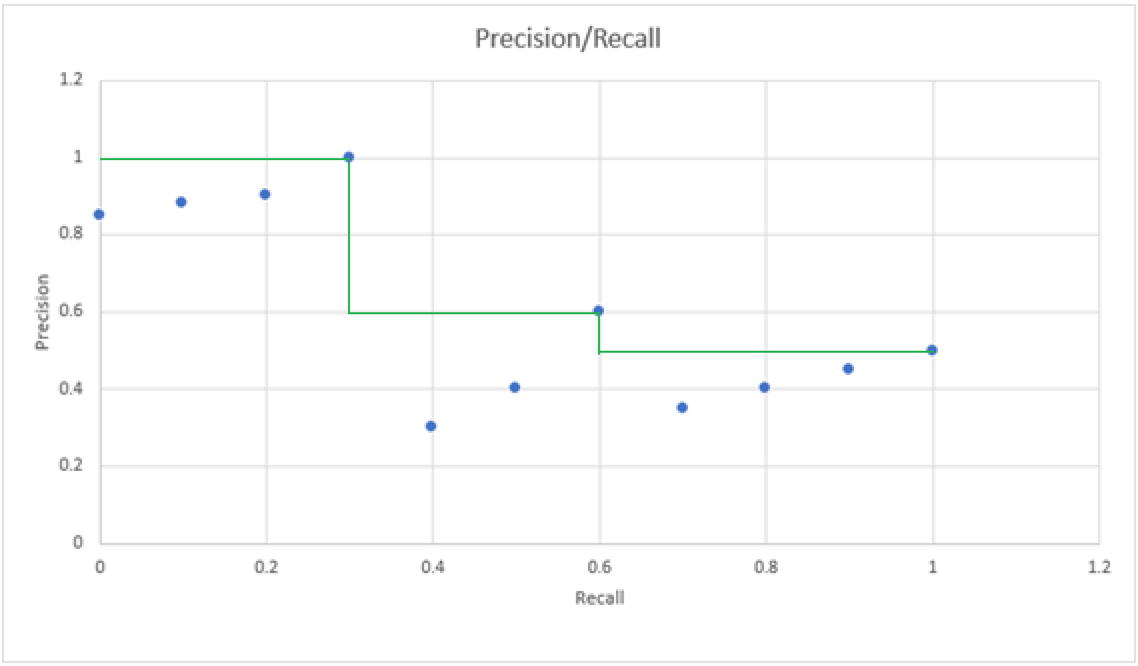


Figure 2. Recall/precision zigzag smoothing

The removal of zigzags is done by setting each recall point’s precision to be equal to the highest precision that comes after that point (3).

$$p_{interp}(r) = \max_{\hat{r} \geq r} p(\hat{r}) \tag{3}$$

Using N interpolated points is as follows. The area is distributed into N equally distant points. For each point the maximum precision value is chosen using formula 3. The precisions are summed and divided by the number of points (4).

$$AP = \frac{1}{N} \sum_{r \in \{0,\ldots,1.0\}} p_{interp}(r) \tag{4}$$

Using area under the curve approach to calculate AP is as follows. Each precision drop represents a separate rectangle. The area of each of the rectangles is taken and summed up (5).

$$AP = \sum (r_{n+1} - r_n) p_{interp}(r_{n+1}) \tag{5}$$

mAP is calculated by taking the average of APs over every class.

The most common metric that is used for evaluating the mAP of an algorithm is the COCO method. Two things are done in that. First, recall is given 101 (N=101) points in that method. Second, mAP is calculated over a range of IoU threshold values. First, mAP is calculated for an IoU threshold value of 0.5, then it gets incremented by 0.05 and the mAP is calculated again. This goes on until 0.95. The average of those mAPs is taken. It is denoted as mAP@[0.5:0.95]. COCO mAP provides insight into which algorithms localize better.

In the context of this thesis mAP at a threshold of 0.5, denoted as mAP@0.5, and mAP@[0.5:0.95] will be used to compare the accuracies of the algorithms at various steps.

4 Object detectors

Before the popularity of deep learning, object detection was done by using manually engineered feature detectors. This involved finding the appropriate values for filters instead of letting a network learn them. A sliding window would then slide over the image at different scales to try and find matches to the filters.

The major shift to deep learning happened in 2012 when AlexNet [33] won the ImageNet Large Scale Visual Recognition Challenge [44]. Ever since, a lot of object detection algorithms using CNNs have come out [6][7][8].

They can mainly be grouped into two categories – single-stage and two-stage object detectors. With two-stage object detectors, the first stage is generating regions of interest. The second being the classification of those areas and adding a bounding box. Single-stage detectors do not have a region of interest generation stage, rather it all runs directly in one operation.

Most of the object detection algorithms employ anchor boxes to localize objects. However, some newer models have tried a different approach. For example, Objects as Points [34], which represents objects as a single point in their bounding box centres from which size, dimensions etc are regressed. The algorithms used in this thesis are all based on anchor boxes.

4.1 Faster R-CNN

4.1.1 Reason for choosing

It is not the newest nor the fastest algorithm. However, it is part of the model zoo inside of Detectron2 [26]. Detectron2 is a library that has several state-of-the-art object detection algorithms implemented. It was released in October of 2019. Detectron2 was chosen due to its novelty and claim that it trains faster. Faster R-CNN was chosen from the model zoo, as, at the time of this writing, it has the highest accuracy of the bounding box detectors. Additionally, the ResNet-101 [37] with a feature pyramid network [42] (R101-FPN) backbone version was chosen over the ResNeXt-101 [43] with a feature pyramid network (X101-FPN). This is because R101 trains faster and requires less memory as seen in table 1.

Table 1. Detectron2 Model Zoo Faster R-CNN snippet [28]

Name	**Train time (s/iter)**	**Inference time (s/im)**	**Train mem (GB)**	**Box AP**
R101-FPN	0.286	0.051	4.1	42.0
X101-FPN	0.638	0.098	6.7	43.0

4.1.2 Description

Faster R-CNN is a two-stage object detector. Both its predecessors, R-CNN [8] and Fast R-CNN [9], used selective search [30] to generate the RoIs. However, selective search is a slow process, thus Faster R-CNN removed it and replaced it with a CNN solution. The structure of the Faster R-CNN model can be seen in figure 3.

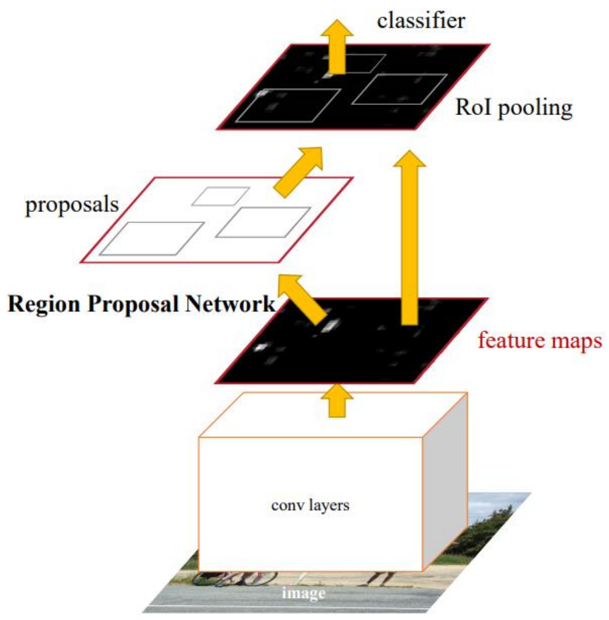


Figure 3. Faster R-CNN structure [9]

The image is fed into the chosen backbone. In the context of this thesis it’s the ResNet-101 network. A feature map is generated as the output from that network. An anchor point is generated at every point in the feature map. For every anchor point k anchor boxes are generated, which are of certain predefined width/height ratio and scales. The number of anchor boxes k depends on the amount of scales and aspect ratios chosen. For 3 scales and 3 aspect ratios, the number of boxes generated for each point would be k=9. The total number of anchor boxes generated for a feature map with dimensions width W and height H would then be *W*H*k*.

Next these anchor boxes go into the region proposal network, which starts with a 3x3 convolutional filter, followed by 2 1x1 convolutional filters in parallel. The first of the 1x1 filters is for classifying whether the anchor box is an object or background. The output size for that is *2*k*, where k is the number of anchor boxes. The second filter is for adjusting the anchor box to be more exact to the object. It outputs multipliers for the anchor box centre point x, y and for its width and height, thus its output is *4*k*.

Non-max suppression is used in order to limit the number of anchors. The highest probability boxes are found and boxes that overlap with it are removed using an IoU value. If a box overlaps with the highest probability box with an IoU larger than a set threshold, then that box is removed. An N amount of the best boxes are kept, sorted by score.

What follows is the region of interest pooling layer and actual classification. The initial feature map is used to extract each region proposal. It gets resized to a fixed size of 14x14xD, followed by max pooling with a kernel of 2x2, resulting in a feature map of 7x7xD, where D is the depth of the output from the backbone. The output gets flattened and sent into two fully connected layers of a size of 4096.

Finally, what follows are two additional fully connected layers. One for deciding the class. It has a size of N + 1, where N is the number of classes and 1 is the background class. The other one for adjusting the box dimensions to the object. This layer has a size of 4N, where N is the number of classes and 4 is the offset for centre point x, y and width and height.

4.2 YOLOv3

4.2.1 Reason for choosing

While not the newest, it is one of the fastest detection algorithms, while sacrificing some accuracy. The comparison between some models can be seen in table 2, taken from the YOLO website.

Table 2. YOLOv3 benchmark comparisons [29]

Model	**mAP @ 0.5**	**FPS**
SSD300 [7]	41.2	46
SSD500 [7]	46.5	19
YOLOv2 608x608 [40]	48.1	40
Tiny YOLO [6]	23.7	244
SSD321 [39]	45.4	16
DSSD321 [39]	46.1	12
R-FCN [41]	51.9	12
SSD513 [39]	50.4	8
DSSD513 [39]	53.3	6
FPN FRCN [42]	59.1	6
Retinanet-50-500 [36]	50.9	14
Retinanet-101-500 [36]	53.1	11
Retinanet-101-800 [36]	57.5	5
YOLOv3-320	51.5	45
YOLOv3-416	55.3	35
YOLOv3-608	57.9	20
YOLOv3-tiny	33.1	220
YOLOv3-spp	60.6	20

4.2.2 Description

YOLOv3 [35] is the third iteration in the YOLO algorithms. With it came a larger network, thus increasing detection accuracy and detections at different scales. The structure of the network can be seen in figure 4.

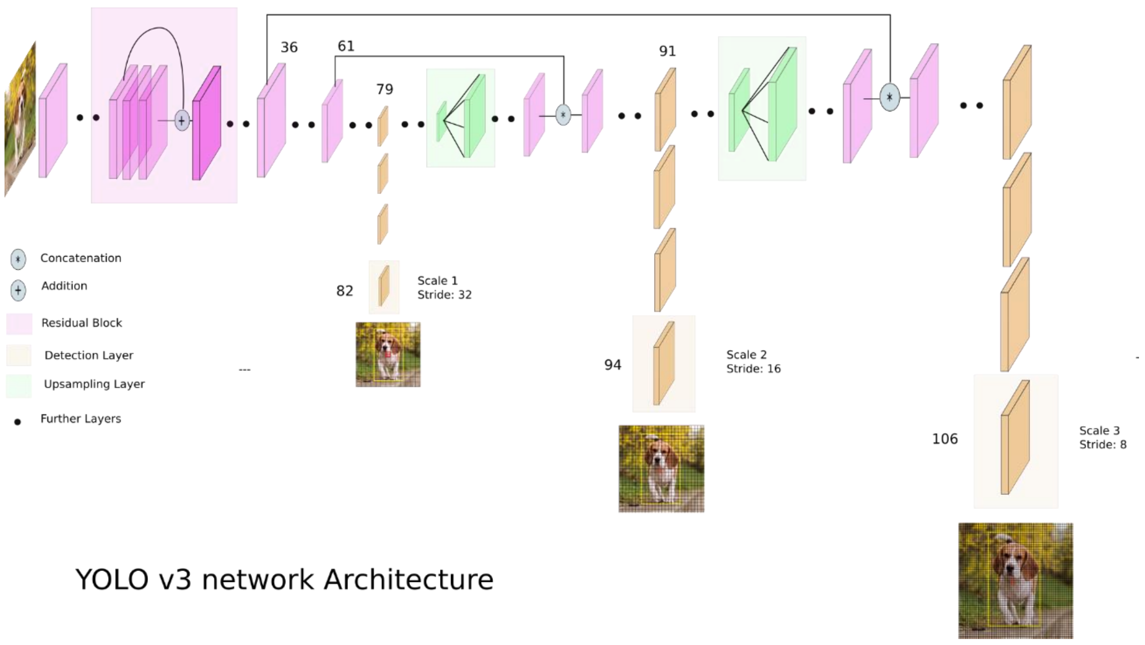


Figure 4. YOLOv3 network architecture [31]

Unlike Faster R-CNN, it is a one-stage detector. However, several implementation details are shared with Faster-RCNN. One of them being pre-defined anchor boxes. The anchor point for the boxes is the centroid of the grid. Like Faster-RCNN, it uses 3 bounding boxes at 3 different scales for a total of 9 bounding boxes. Additional ones can be added, but it increases computation considerably. The previous scales' feature maps are concatenated with the current scale in the case of medium and small scale detection, allowing the other layers to benefit from the results of the previous scale.

In the case of a 416x416 input, the dimensions of the output at different scales would be 52x52x3x(4 + 1 + N) at small scale, 26x26x3x(4 + 1 + N) at medium scale, and 13x13x3x(4 + 1 + N) at large scale, where N is the total number of classes. 3 denotes the number of anchor boxes at that scale. The final dimension is formed by similar values as can be seen in Faster-RCNN. The 4 denotes adjustments to the pre-defined anchor box for its centre x, y coordinates and its width and height. The 1 denotes objectness – whether the area has a detection or not. Finally, the number of classes, that specify what class the area has.

Non-max suppression is used to remove duplicate detections, like Faster R-CNN.

4.3 EfficientDet

4.3.1 Reason for choosing

The algorithm was chosen due to its novelty. The algorithm has varying degrees of models, where a trade-off is being made between the accuracy and model complexity. The models start from EfficientDet-D0 and go up to EfficientDet-D7. The paper [10] claims that their most basic model structure of EfficientDet-D0 is capable of matching YOLOv3 accuracy while being faster. Additionally, it is claimed that EfficientDet-D2 achieves a mAP of 43.0 on the COCO test set, which would be 1% higher than Faster R-CNN. A section of the comparisons can be seen in table 3. Both D0 and D2 were chosen for the comparisons in order to validate the accuracy claims.

Table 3. EfficientDet model comparisons snippet [10]

	COCO test set			**COCO validation set**	**FLOPs**	**Latency**	
Model	**mAP @ [0.5:0.95]**	**mAP@ 0.50**	**mAP@ 0.75**	**mAP @ [0.5:0.95]**	**FLOPs**	**GPU (ms)**	**CPU (s)**
EfficientDet-D0 (512)	33.8	52.2	35.8	33.5	2.5B	16	0.32
YOLOv3	33.0	57.9	34.4	-	71B	51	-
EfficientDet-D2 (768)	43.0	62.3	46.2	42.5	11B	24	1.2
RetinaNet-R50 (1024) [36]	40.1	-	-	-	248B	51	7.5
RetinaNet-R101 (1024) [36]	41.1	-	-	-	326B	65	9.7
ResNet-50 + NAS-FPN (640) [38]	39.9	-	-	-	141B	41	4.1

4.3.2 Description

Like YOLOv3, this is a one stage detector. It consists of an EfficientNet [13] backbone combined with a weighted bidirectional feature network (BiFPN). The main additions that this solution adds to existing ones is a new type of feature fusion, the BiFPN, and the ability to scale the network. The architecture can be seen in figure 5.

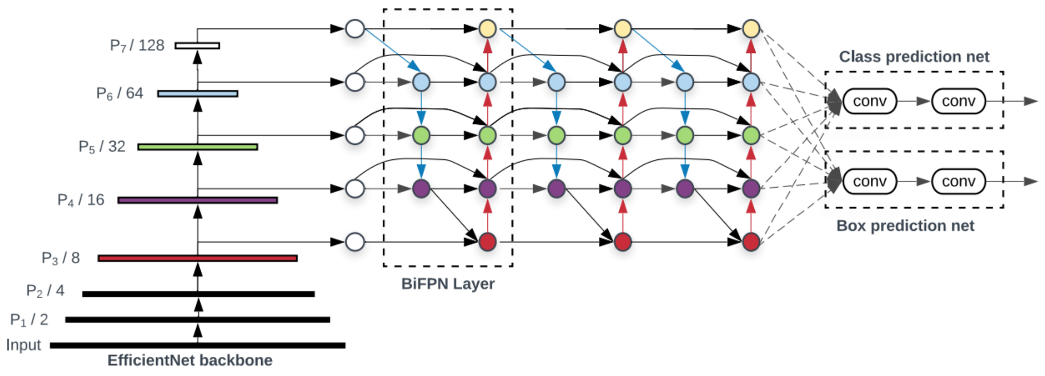


Figure 5. EfficientDet network architecture [10]

PANet [32] was used as an example in order to develop BiFPN. This was because PANet has bidirectional feature fusing, however it has additional computation costs. Thus, the approach was to optimize PANet by removing input nodes with a single edge. Additionally, an extra edge was added from the input to the output to provide more feature fusing without adding great additional cost. The top-down and bottom-up paths are considered one layer, so when the network is scaled, then the pair of those are added. What's more is that the fused feature maps have trainable weights added to it in order to negate certain layers contributing more than others.

Different sized backbones are used when scaling. These range from EfficientNet-B0 to B6. Table 4 shows the different sizes of the scaled networks.

Table 4. EfficientDet network sizes [10]

EfficientDet network	**Input size**	**EfficientNet backbone network**	**BiFPN**		**Box/Class #layers**
			#channels	**#layers**	
D0 (Φ = 0)	512	B0	64	3	3
D1 (Φ = 1)	640	B1	88	4	3
D2 (Φ = 2)	768	B2	112	5	3
D3 (Φ = 3)	896	B3	160	6	4
D4 (Φ = 4)	1024	B4	224	7	4
D5 (Φ = 5)	1280	B5	288	7	4
D6 (Φ = 6)	1280	B6	384	8	5
D7 (Φ = 6)	1536	B6	384	8	5

Input size (9), BiFPN channels (6), BiFPN layers (7), and Box/Class layers (8) are calculated based on the Φ values for D0-D6. D7 is the same as D6, except it has a larger input size.

$$W_{bifpn} = 64 * (1.35^{\Phi}) \tag{6}$$

$$D_{bifpn} = 3 + \Phi \tag{7}$$

$$D_{box} = D_{class} = 3 + \left\lfloor \frac{\Phi}{3} \right\rfloor \tag{8}$$

$$R_{input} = 512 + \Phi * 128 \tag{9}$$

5 Dataset

5.1 Gathering data

Due to the constraints of Ajapaik's system at the time of this writing it was easier to gather the data from external sites. The sites used were FINNA [17] and FOTIS [1]. FOTIS contains images mainly from Estonia, while FINNA contains images mainly from Finland. Both image archives have data similar to that which can be found on the Ajapaik website, thus the dataset gathered is representative of the data on which the algorithms will be run in production. The process of data gathering was running a JavaScript script to mass download images of a certain class and then manually filtering through the datasets to remove unrelated datasets. A considerable portion of the downloaded images had nothing to do with the objects that were searched for. Examples of the dataset can be seen in figure 6.

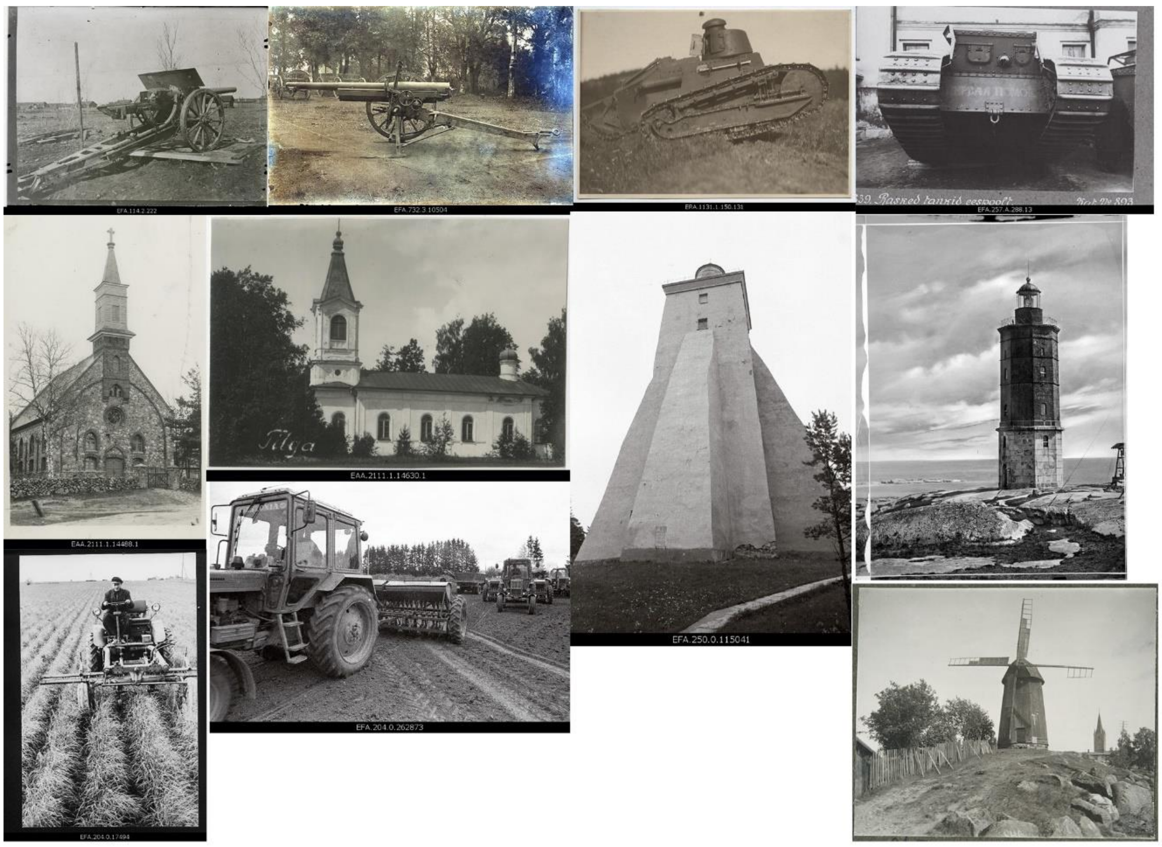


Figure 6. Examples of the dataset

5.2 Annotating

An annotation solution had to be created for the Ajapaik webpage as training data for the algorithms is crowdsourced. The users provide the ground truth labels that are used for training. Other users can then validate the labels. Labels with positive feedback get used for training, while negative ones get ignored.

There are many existing JavaScript libraries that provide this function [18][19][20]. However, all of them have some drawbacks. While a lot of them allow for customization of styling, then none of them allow for customization of event handlers, at least as far as the author could find.

There are two main ways they function. One way is by creating a box of predetermined size when the user clicks on the annotation button. Then the user must resize that as required. Another way is by holding down the mouse button and dragging. The first option can be annoying to the user and requires them to perform extra steps. The second option eliminates compatibility with touch screen devices as there's no option to hold down the mouse. In order to circumvent these problems, a separate annotation library had to be written for the webpage to have maximum control over the annotation process.

The created solution functions by clicking in the start position and then clicking at the end position to finish drawing the annotation. This allows the library to function on touch screens, in addition to mouse input devices. An example of the flow is shown in figures 7, 8, 9. In figure 7 the user has finished drawing and a popover opens asking for the user to specify the object. After the annotation has been submitted, then other users can give feedback on it as depicted in figure 8. Images with negative feedback will not be used in the training process to try and limit training on wrong data. The user can either agree with the object class or disagree and additionally specify their own object type, as shown in figure 9.

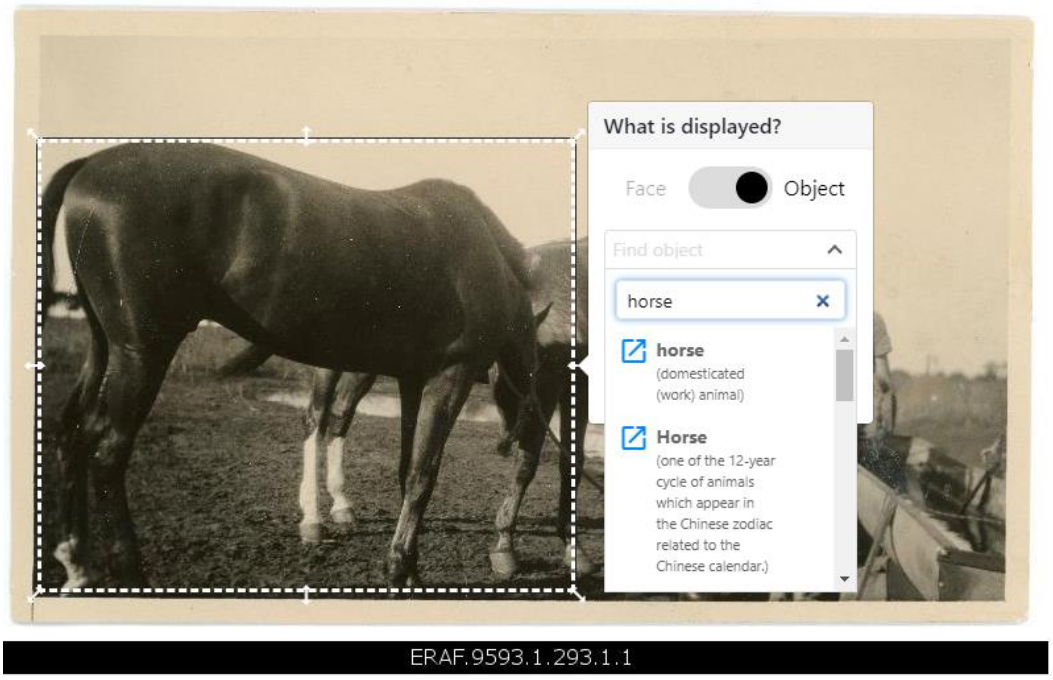

What is displayed?
Face
Object
Find object
horse
horse
(domesticated (work) animal)
Horse
(one of the 12-year cycle of animals which appear in the Chinese zodiac related to the Chinese calendar.)
ERAF.9593.1.293.1.1

Figure 7. Adding an annotation

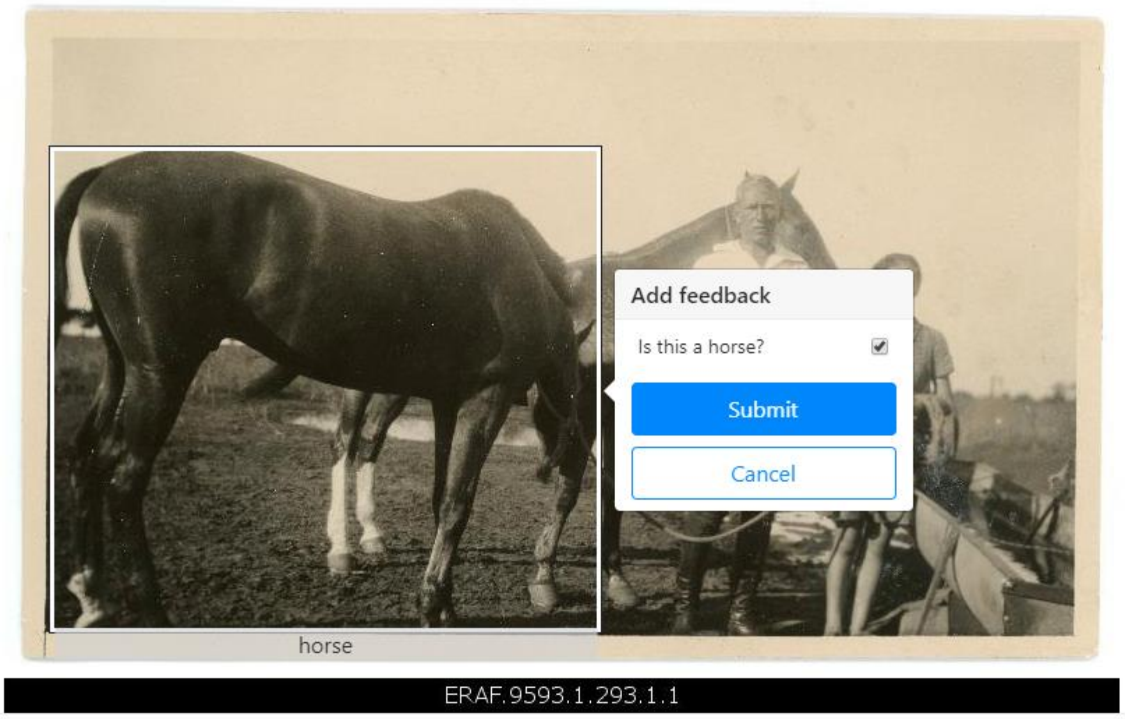


Figure 8. Adding positive feedback to the annotation

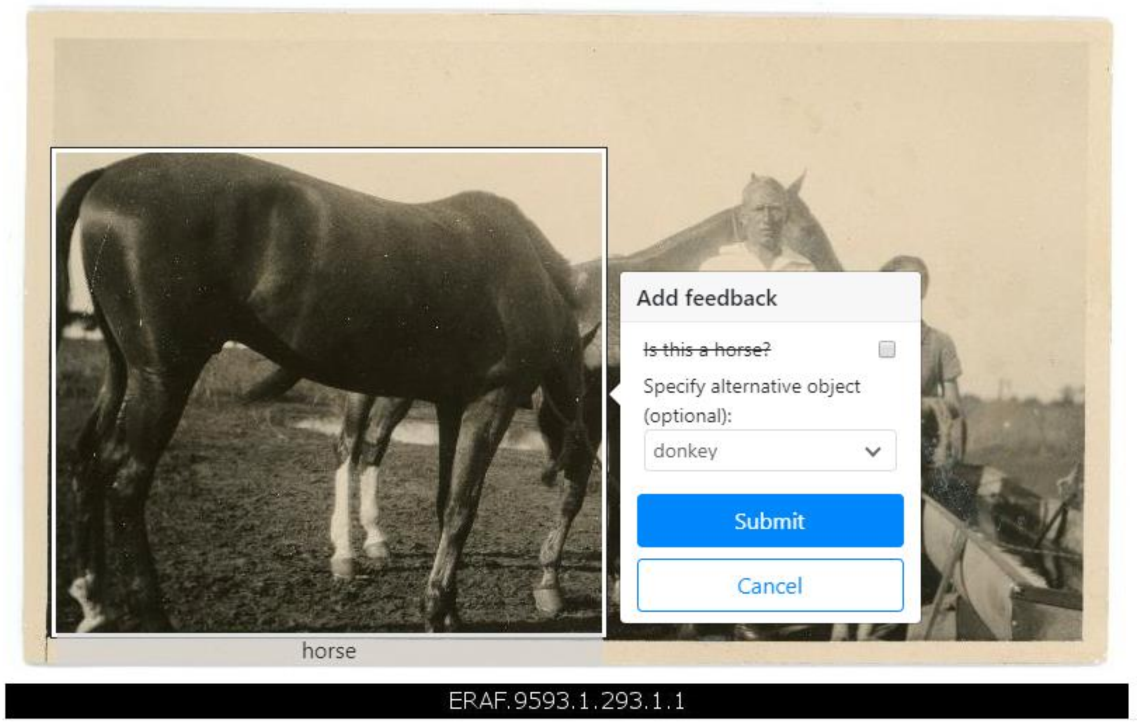


Figure 9. Giving negative feedback

The labelling solution above uses Wikidata [21] in order to give label selections. The positive side of this is that an admin doesn’t have to manually define labels that are allowed, thus requiring less manual human interaction. The negative side, however, is that while users might want to label objects as the same type, they might select an option that has the same written form but means something different. Short explanations of the objects, in addition to hyperlinks to their respective Wikipedia pages, are added in order to try and minimize that risk, which can be seen in figure 7. This does not completely eliminate the risk. However, solving that task is outside of the scope of this thesis.

The above explained process was not used to label the data used to train the algorithms in this thesis due to the limited time available and the conflicting release schedule of Ajapaik. Thus, the author had to use an existing external solution to label the data. Different annotation applications exist for images. Some are online tools, some are offline. Probably the most popular one is CVAT [21]. However, the author of this thesis decided to use LabelBox [23], due to its ease of use and online nature, meaning that annotations were easy to share between environments. But there is a drawback – the number of labels that can be exported is limited to a considerably small number when using a free license. Due to the availability of an education license, however, this was not a problem.

5.3 Running detectors on acquired data

Ajapaik’s environment is limited by hardware and funds available. As such, Ajapaik’s server cannot be used for training since there is only a CPU. Most of the time a GPU is required for training machine learning algorithms. This could be addressed by using Google Colab [24] as the training environment. As Google Colab provides 12 hours of free GPU time and the size of the data isn’t enormous, then that environment suffices. Both training and inference are performed in this environment. While Google Colab randomizes the card that a free session gets, then a Pro session gets a 16 GB Tesla P100. The inference tests are run on that card.

The training process is done incrementally to test the growth of the accuracy over epochs and increased dataset sizes. The process looks as follows:

1) 25 images of a single class are chosen.
2) Those 25 images are trained for and evaluated after every 25 epochs, up until 150 epochs.
3) The accuracy evaluation is performed using the COCO format at IoU values 0.5 (mAP@0.5) and incrementally increasing from 0.5 to 0.95 and taking the mAP (mAP@[0.5:0.95]).
4) After the completion of the epochs 25 images are added and the process is repeated until reaching 100 images.
5) Once 100 images are reached, then the process is started again from 25 images, but with an additional class. So, 25 images for each class, followed by 50 images for each class.
6) Class adding continues until reaching 10 classes. Thus, the final model is trained on 10 classes of 100 images, totalling 1000 images. The dataset sizes used are [1…10] * {25, 50, 75, 100}.

6 Analysis of the results

The process described in chapter 5.3 is used for evaluation. A total of 10 classes are used. The 10 classes are the following: church, cannon, tractor, lighthouse, monument, tank (military), bridge, windmill, altar, ruins. The classes are added to the data in that order as well. So, 1 class is church, 2 classes are church and cannon etc.

The results are presented in pairs of graphs representing mAP @ 0.5 on (a) and mAP @ [0.5: 0.95] on (b). Each pair of graphs represents the results after every 25 images. The evaluations are performed on the validation set. The validation set sizes are roughly 10% of the overall gathered data. The overall gathered data is more than the data used for training. The total dataset gathered is 4500 images. However, due to labelling being a time-consuming process, then a total of 1000 images are used for training. The validation set sizes vary by class as there are different amounts of images available per class, but the sizes range from 30 to 50 images.

The models are trained with a learning rate of 0.01. Stepwise scheduler is used for Faster R-CNN, while a cosine scheduler is used for the others. These schedulers are the defaults for the libraries.

6.1 One class

The first detection is done using the church class.

All the algorithms reach a decent score by 100 images as only a single class must be detected and that single class generalizes relatively well, since most of the churches have a similar structure as can be seen in figure 10.

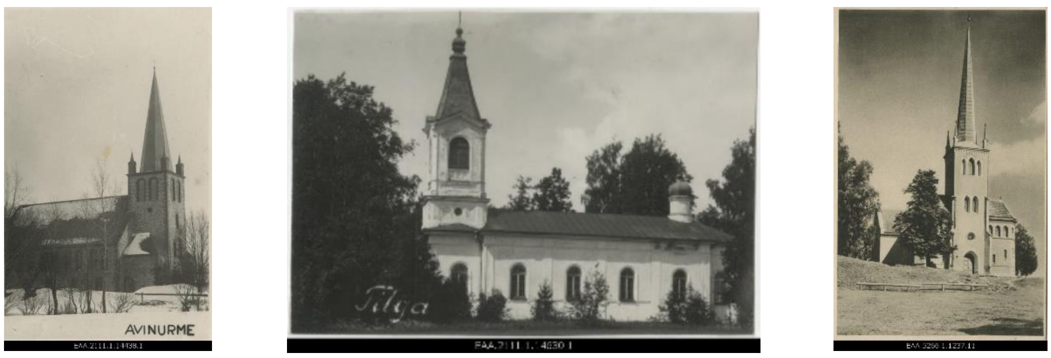
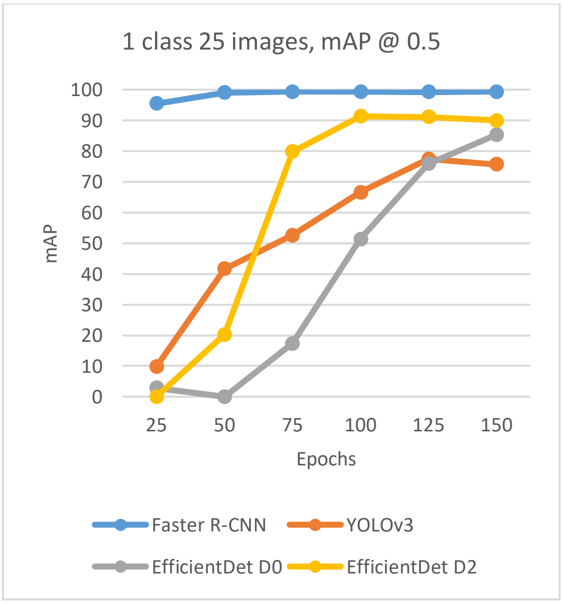


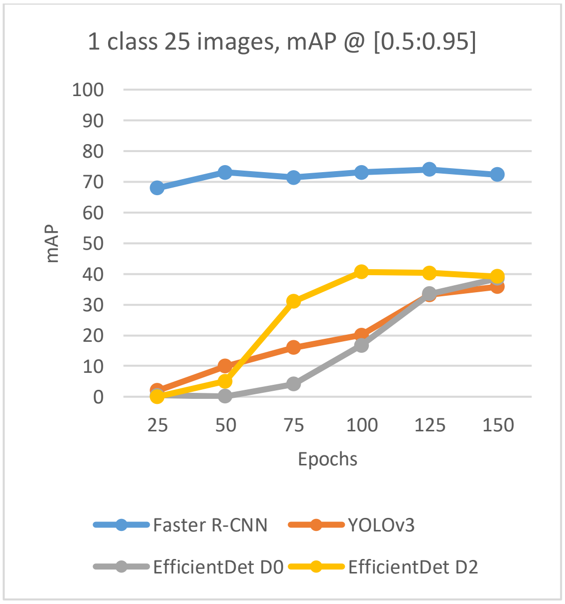
Figure 10. Church examples

What can be noted, however, is that Faster R-CNN gets to a high accuracy value already at 25 epochs, while others take a longer time to build up. This holds true for every dataset size with one class as can be seen in figures 11-14. Additionally, it reaches a near 100 mAP while training with only 25 images, as can be seen in figure 11. However, as more data gets added and epochs become longer, then the other algorithms start learning faster as well. Namely EfficientDet D2 gets close to Faster R-CNN by 100 images.

Looking at the COCO mAP score, then Faster R-CNN is the most accurate at localizing the object. However, as data gets added, then the difference between Faster R-CNN and the other algorithms reduces as can be seen in figures 11(b)-14(b).

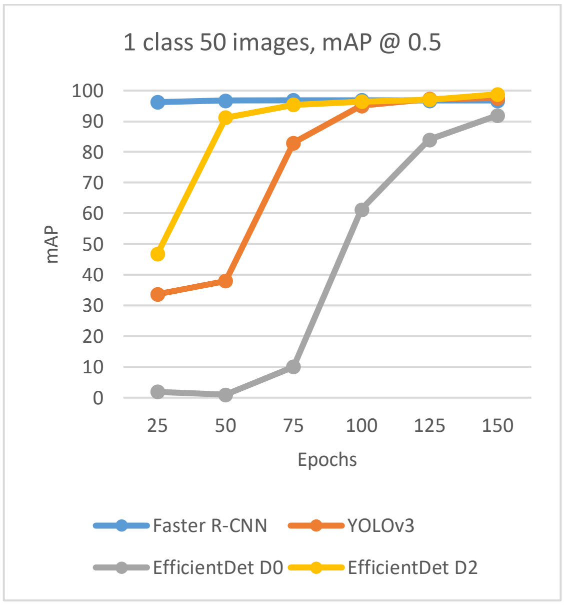


(a)

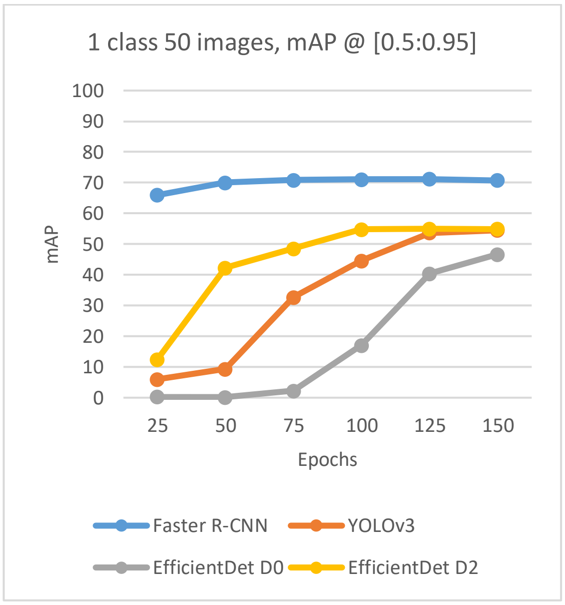


(b)

Figure 11. 1 class 25 images results

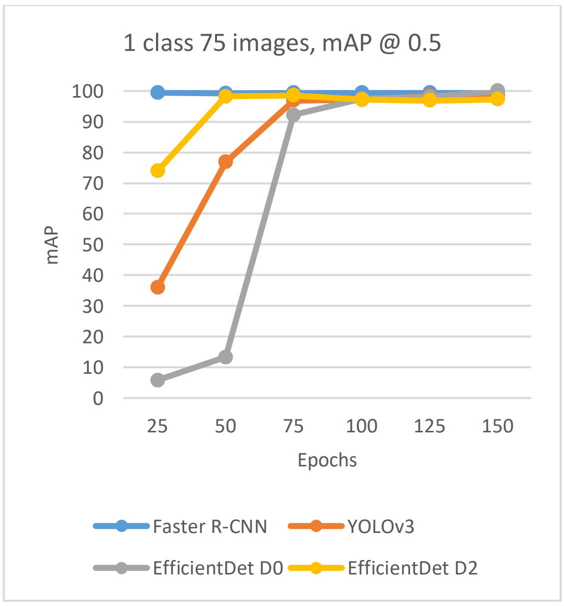


(a)

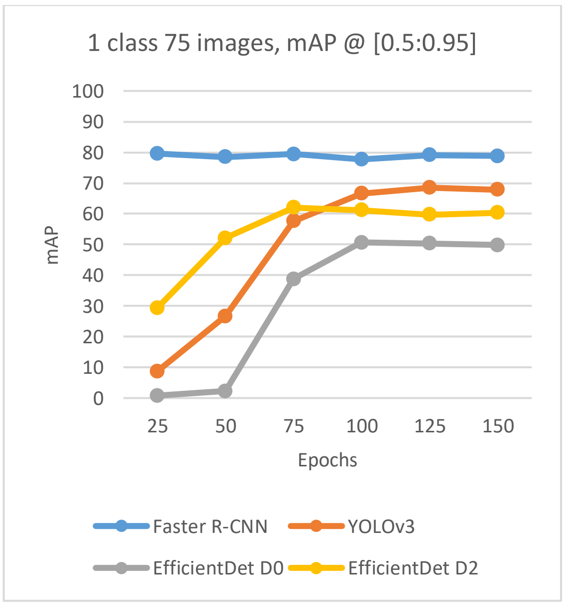


(b)

Figure 12. 1 class 50 images results

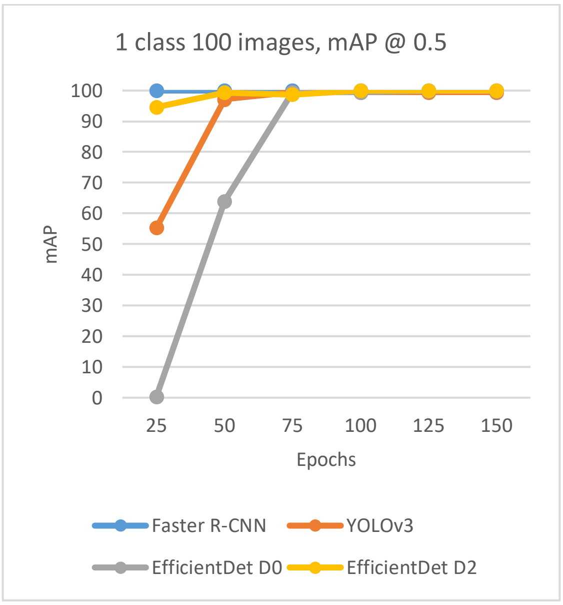


(a)

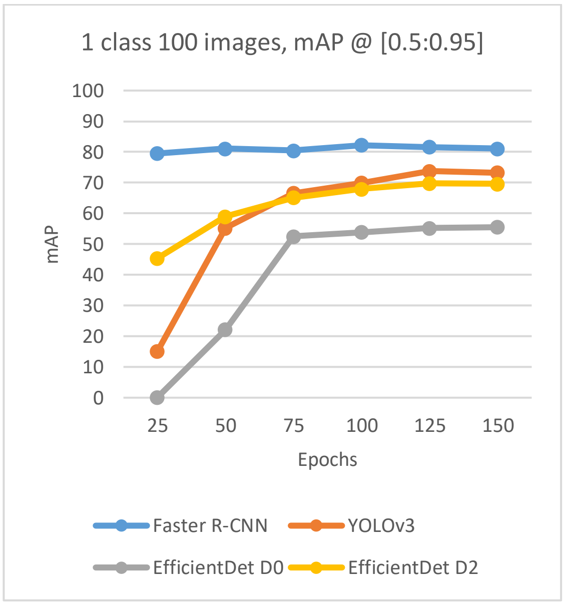


(b)

Figure 13. 1 class 75 images results



(a)



(b)

Figure 14. 1 class 100 images results

6.2 Two classes

The classes for these evaluations are church and cannon.

A drop in accuracy can be seen with the addition of a second class. While the church class is easy to generalize from, then cannon isn't. While churches of the same religion from different periods look similar, then cannons do not. An example of cannons from different periods can be seen in figure 15.

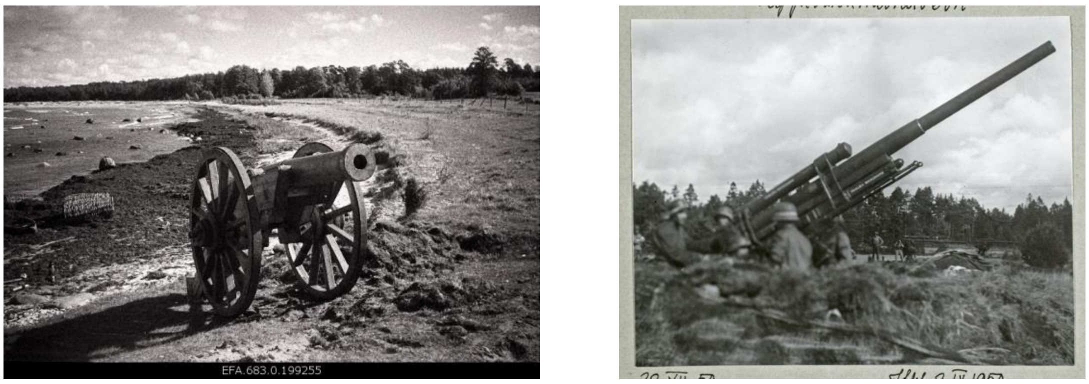
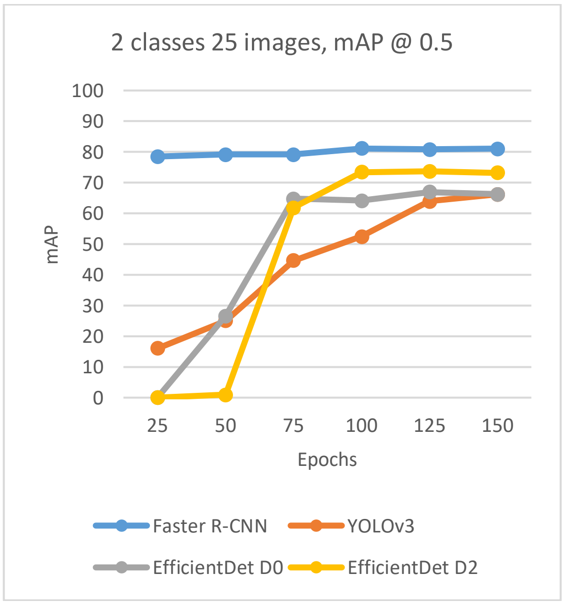


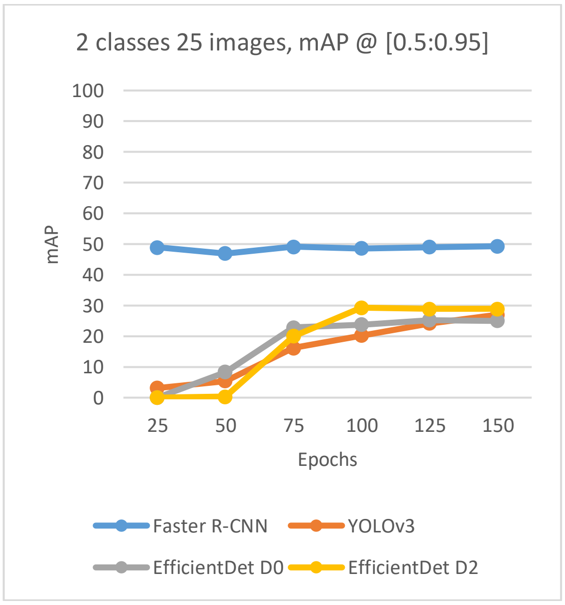
Figure 15. Cannon examples

The accuracy of the detections could be boosted by separating the different time period cannons into separate classes.

Faster R-CNN had the best detection accuracy with 1 class. However, with 2 classes it still generalizes the best from a smaller dataset as can be seen in figure 15(a), but as additional data gets added, then EfficientDet D2 catches up and even surpasses the accuracy of Faster R-CNN as described by figures 16(a) – 19(a). Even though EfficientDet D2 catches up to Faster-RCNN in detection accuracy, then the latter remains the most accurate at localizing the objects as can be seen in figures 16(b) – 19(b).

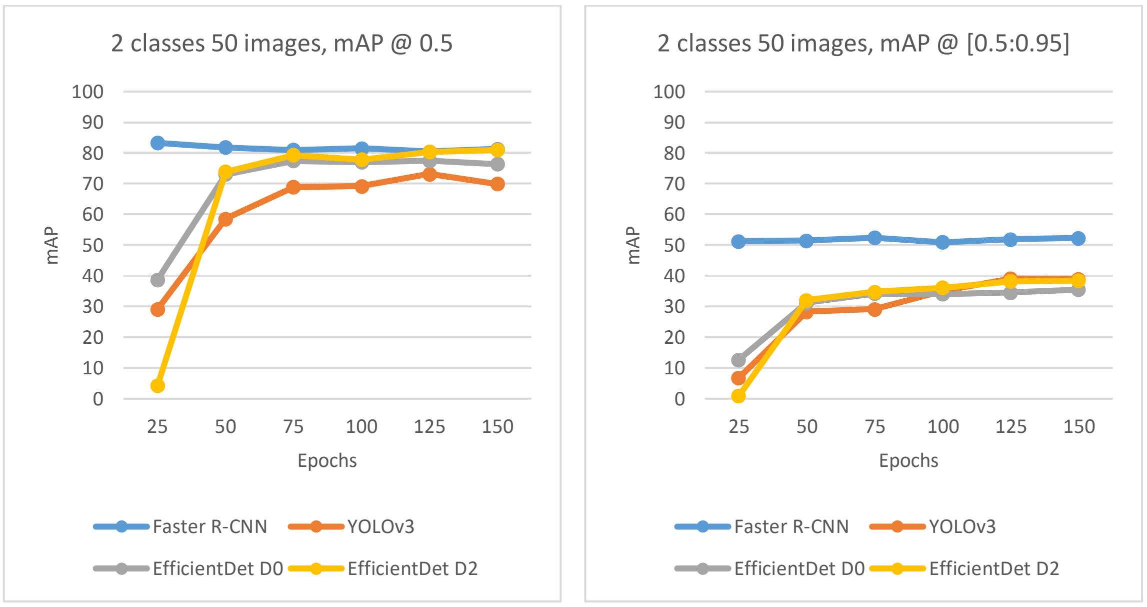
In terms of per-epoch learning performance Faster R-CNN maintains the lead with it having reached its maximum, or near maximum, value already by epoch 25.





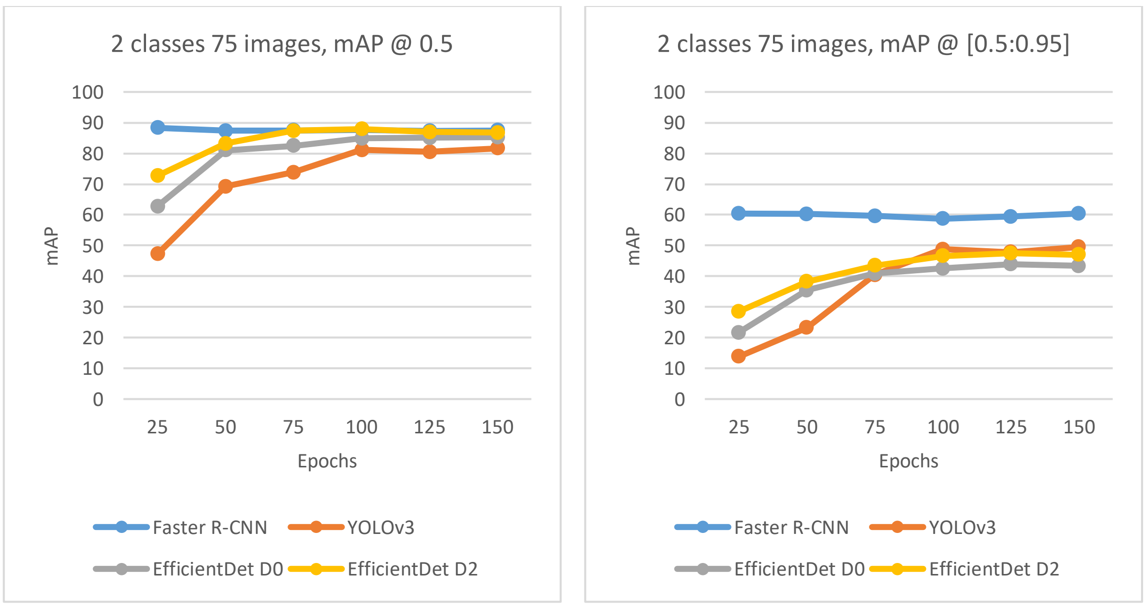
(a) (b)

Figure 16. 2 classes 25 images results



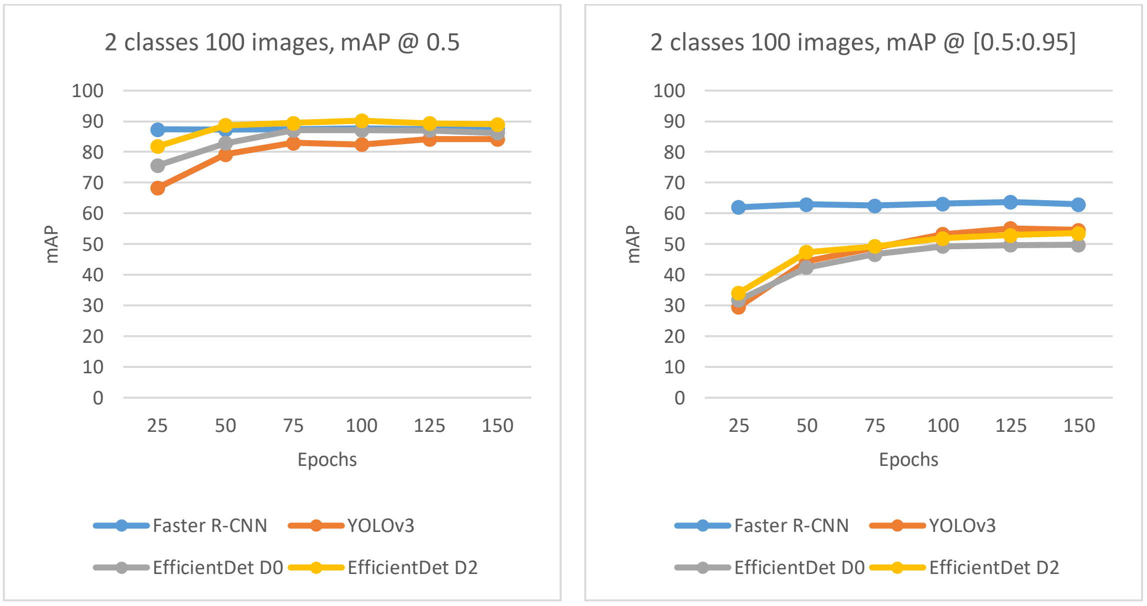
(a) (b)

Figure 17. 2 classes 50 images results



(a) (b)

Figure 18. 2 classes 75 images results



(a) (b)

Figure 19. 2 classes 100 images

6.3 Three classes

The classes for these evaluations are church, cannon, and tractor.

An additional drop in overall accuracy can be noted due to similar reasons as with the cannon class. Tractors can look different based on the period. An example of tractors can be seen in figure 20.

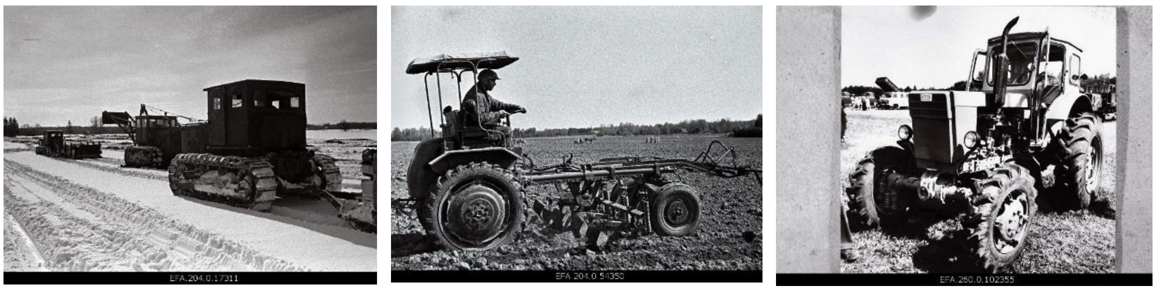
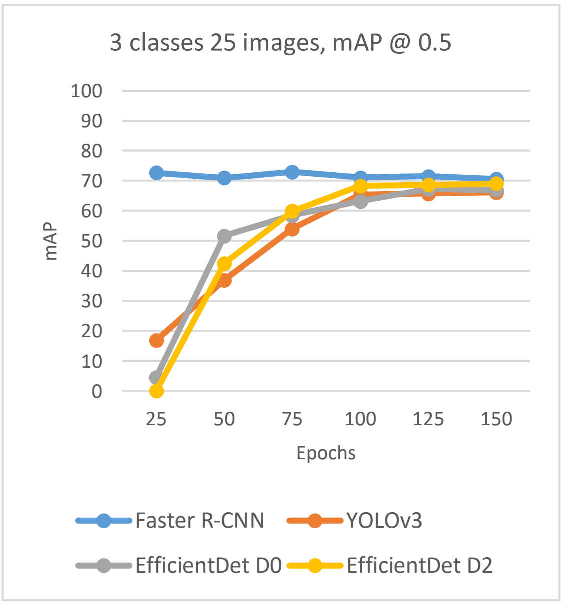


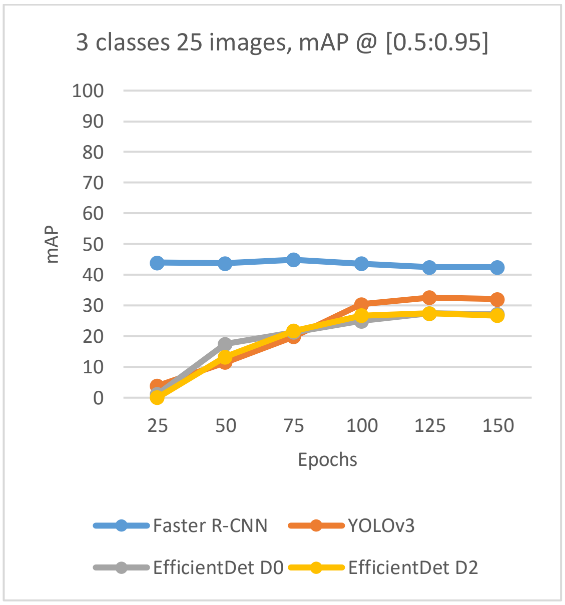
Figure 20. Tractor examples

Although the overall accuracy dropped, the same accuracy relations hold true. As more data gets added, then other algorithms catch up to Faster R-CNN as displayed in figures 21(a) – 24(a). EfficientDet D2 is even able to surpass Faster R-CNN as seen in figure 22(a).

Faster R-CNN, however, remains in the lead in terms of per-epoch learning performance and object localization accuracy described by figures 21(b) – 24(b).

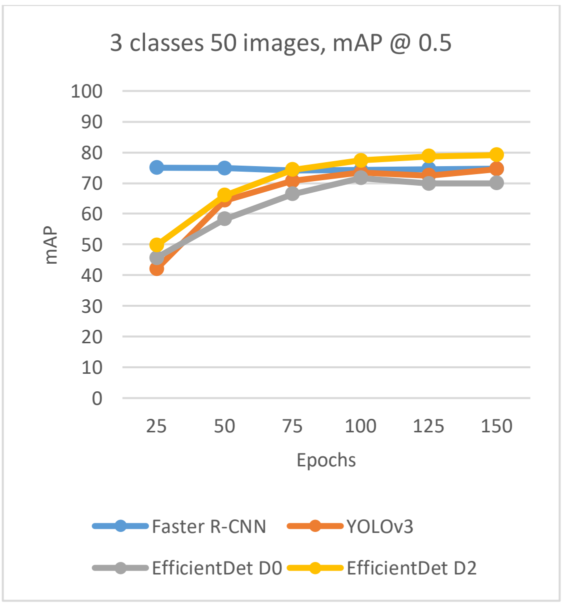


(a)

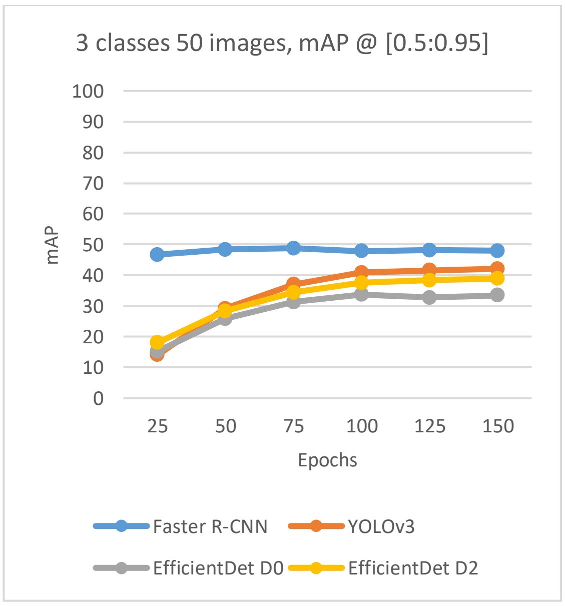


(b)

Figure 21. 3 classes 25 images

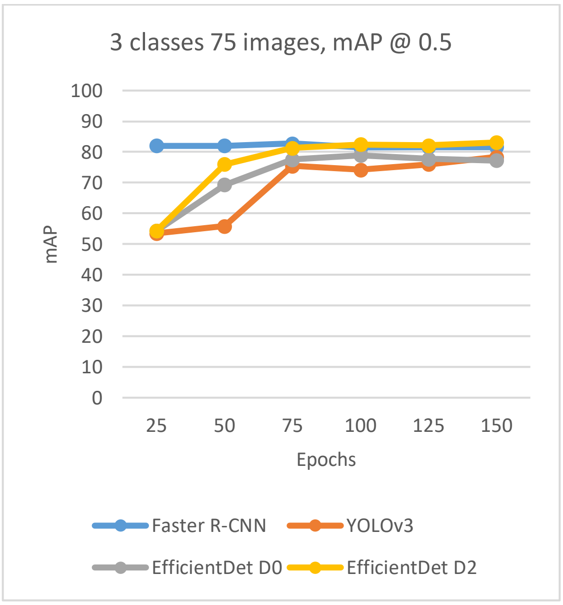


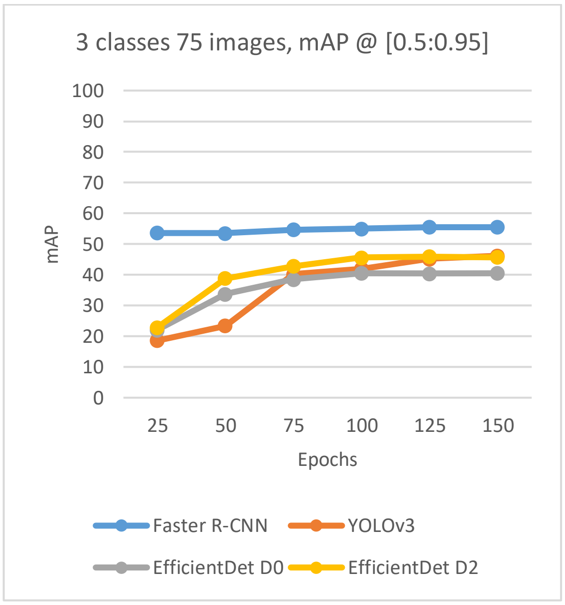
(a)



(b)

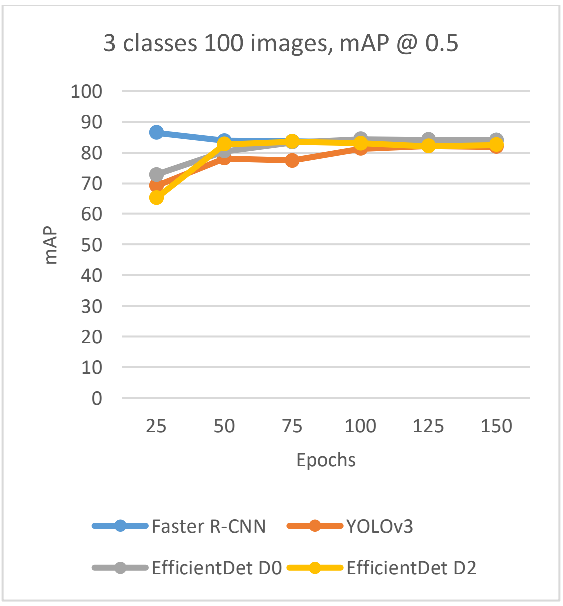
Figure 22. 3 classes 50 images results

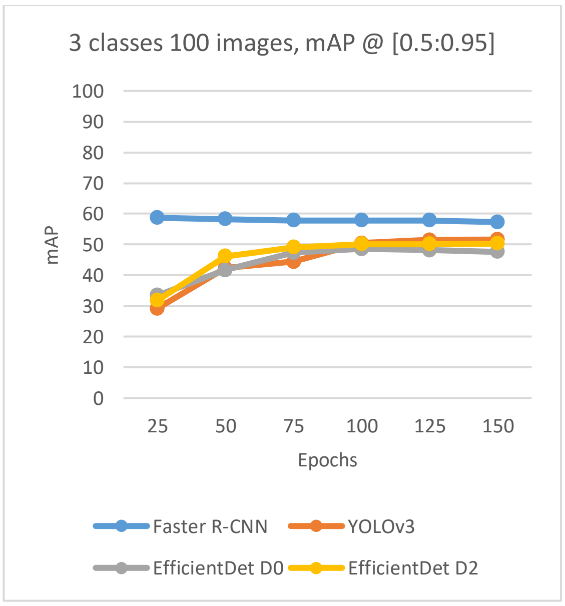




(a) (b)

Figure 23. 3 classes 75 images results





(a) (b)

Figure 24. 3 classes 100 images results

6.4 Four classes

The classes for these evaluations are church, cannon, tractor, and lighthouse.

A slight increase in overall accuracy can be noted, when compared to the results from three classes. The added lighthouse class is easier to generalize from as it doesn’t have radical differences between periods. The same tall and skinny structure is valid for most of the data. An example of the dataset can be seen in figure 25.

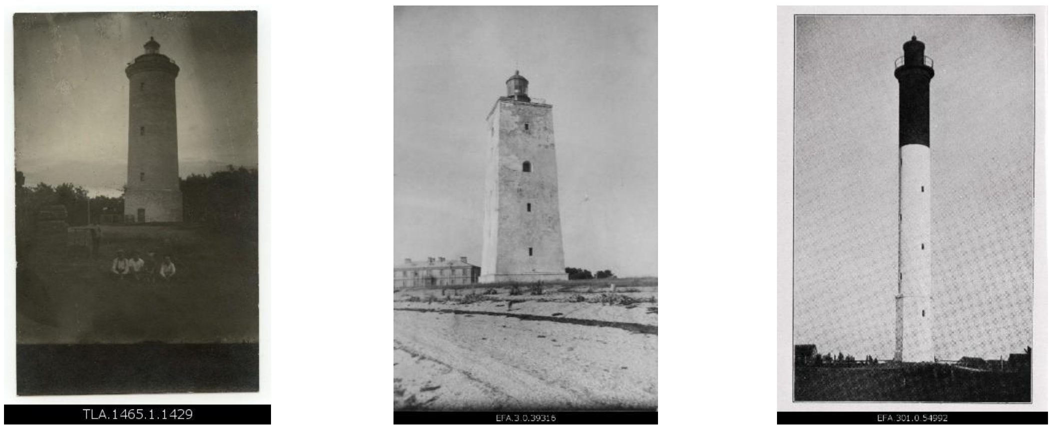
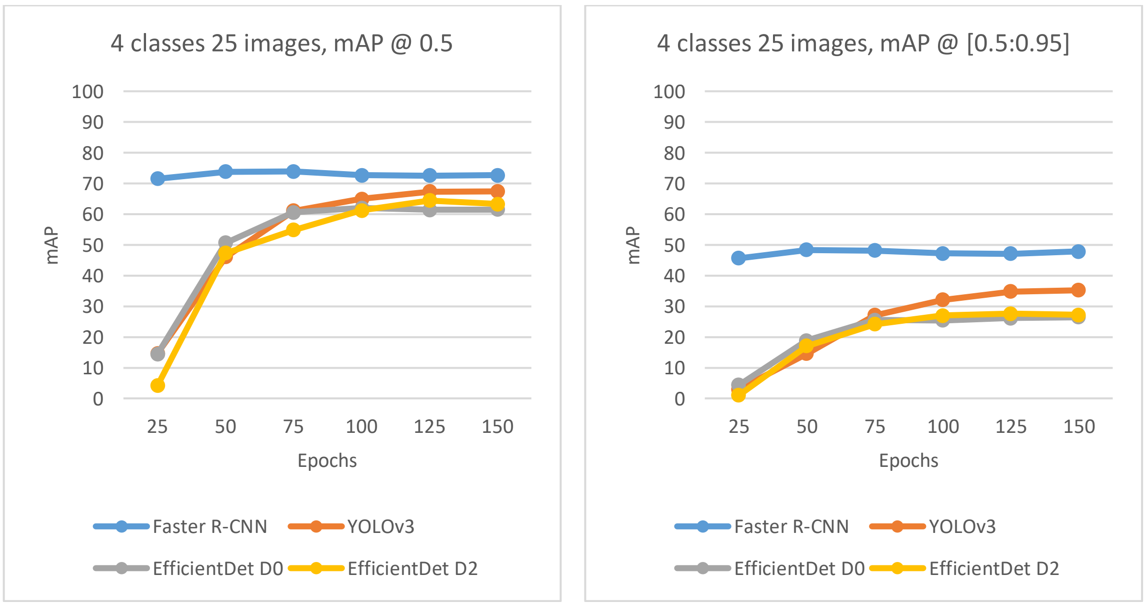


Figure 25. Lighthouse examples

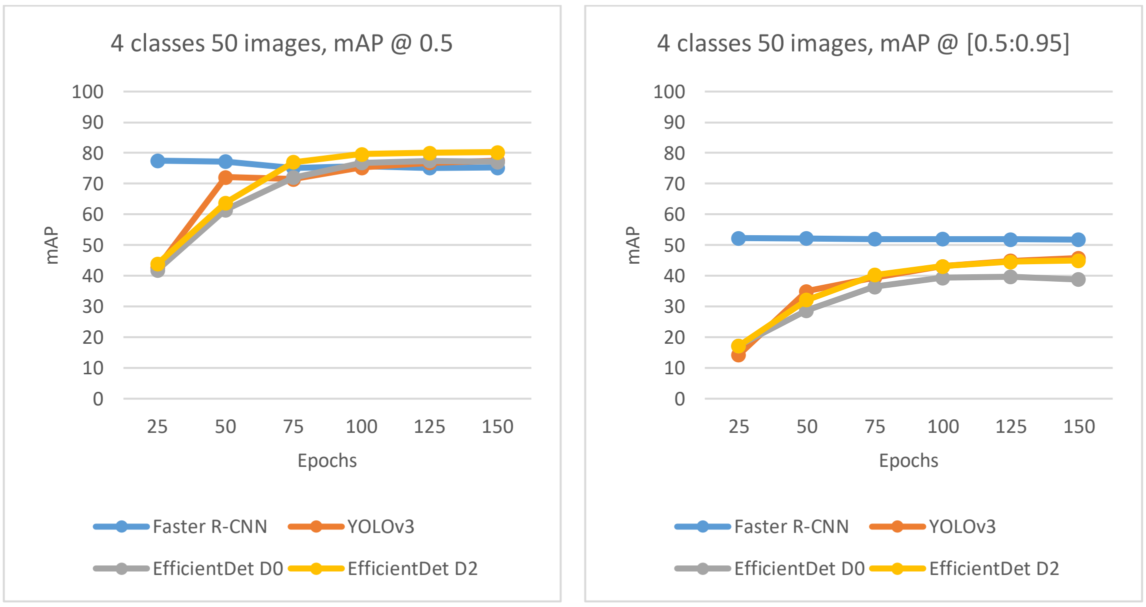
The same accuracy relations hold true as with the previous classes. Faster R-CNN can get the highest detection and localization accuracy from a small dataset, as seen in figure 26. As data gets added, then the gap starts closing, with EfficientDet D2 and YOLOv3 able to overtake at 50 images in figure 27(a).

However, regardless of the dataset size, the other algorithms cannot match Faster R-CNN's per-epoch training performance and localization accuracy as displayed in figures 26(b) – 29(b).



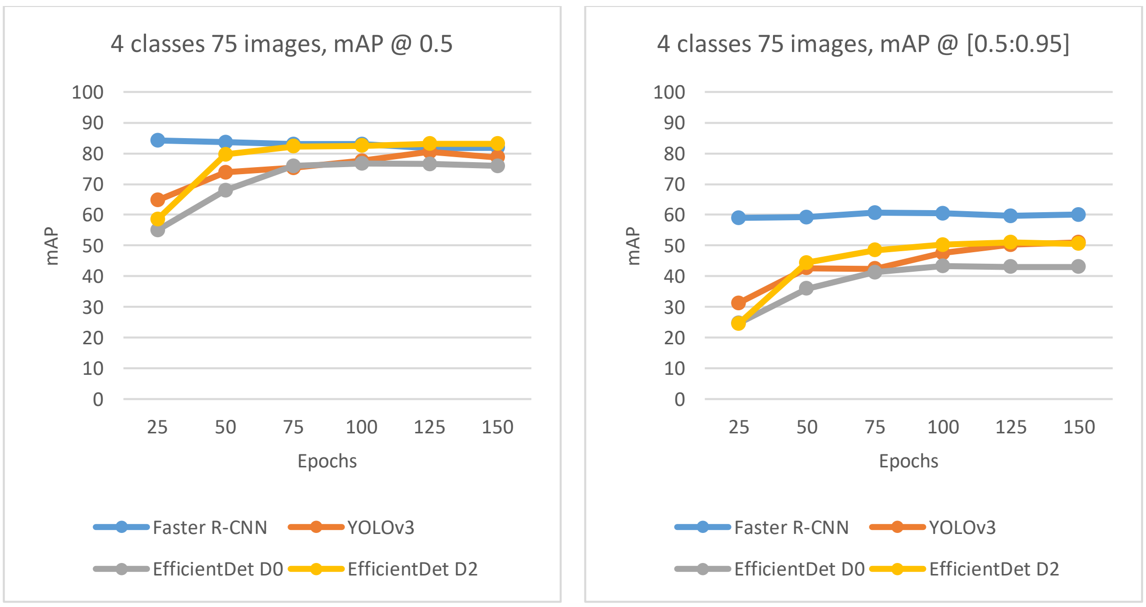
(a) (b)

Figure 26. 4 classes 25 images results



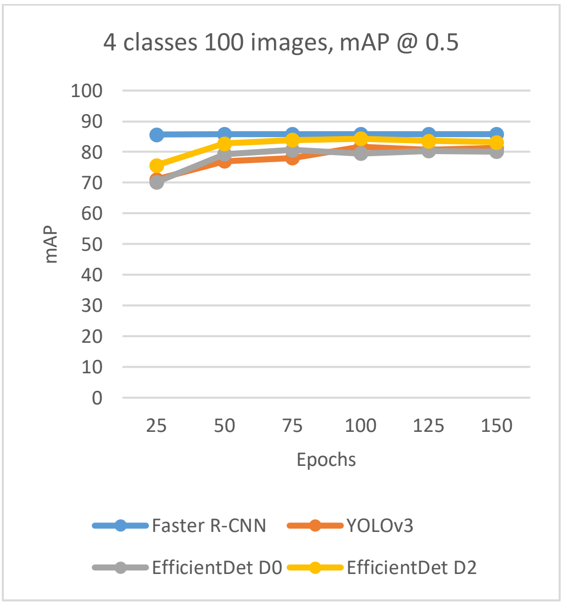
(a) (b)

Figure 27. 4 classes 50 images results

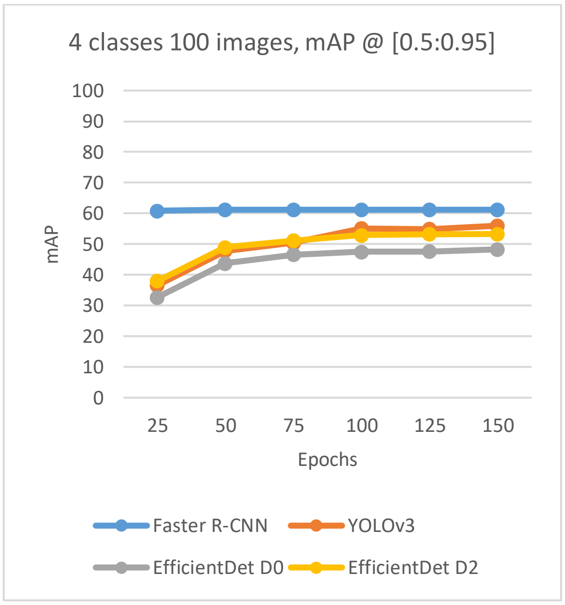


(a) (b)

Figure 28. 4 classes 75 images results



(a)



(b)

Figure 29. 4 classes 100 images results

6.5 Five classes

The classes for these evaluations are church, cannon, tractor, lighthouse, and monument.

Another drop in overall accuracy happens due to a hard to detect class. Monuments can differ greatly in shape and dimensions as seen in figure 30. Additionally, they are embedded into the environment, so their start and end aren't as clearly defined.

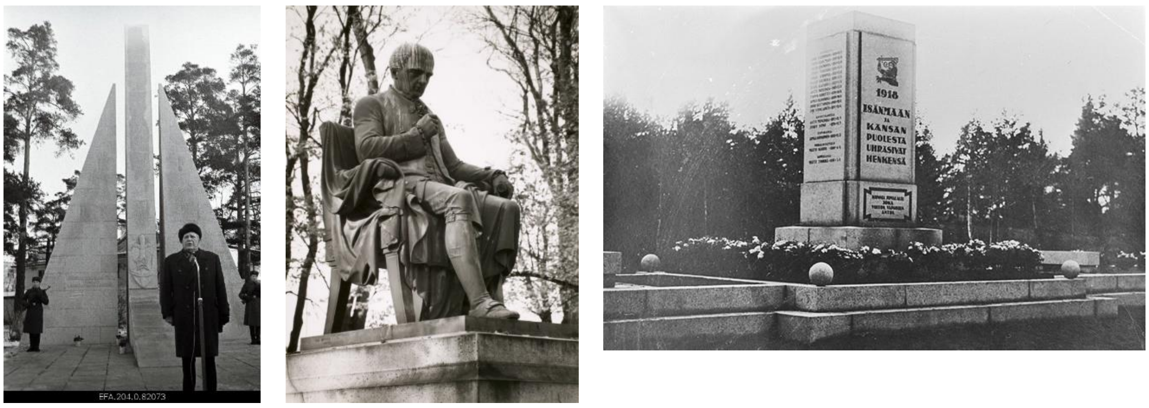
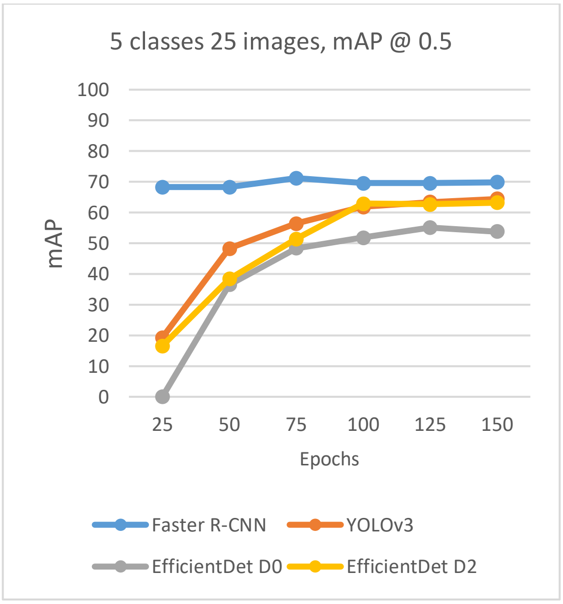


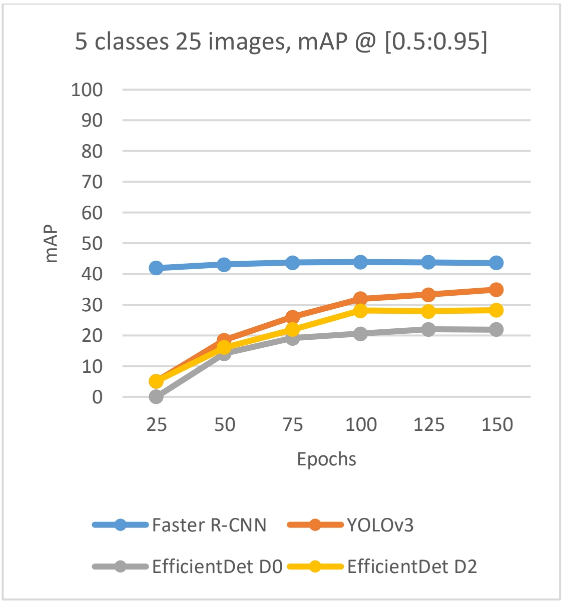
Figure 30. Monument examples

For the most part, the same relations hold true in terms of accuracy. However, with 100 images, in figure 34(a), EfficientDet D0 can match Faster R-CNN, while in previous steps it has fallen considerably behind. YOLOv3 and EfficientDet D2 have managed to pass Faster-RCNN as seen in figures 32(a) and 34(a). D2 has done so by a considerable margin.

Although Faster R-CNN is matched in detection accuracy, it still stays ahead the rest in terms of per-epoch training performance and localization accuracy. But YOLOv3 and EfficientDet D2 come close when reaching 100 images as seen in figure 34(b).

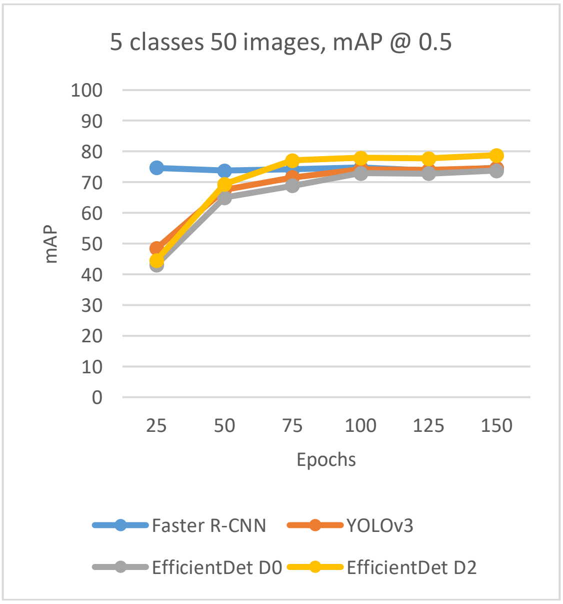


(a)

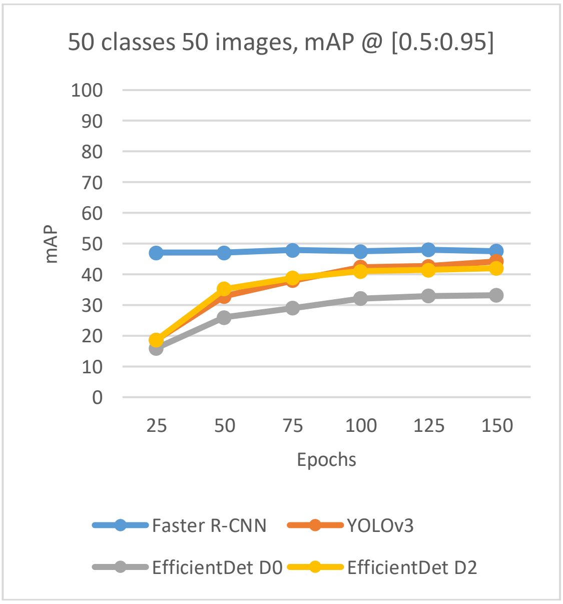


(b)

Figure 31. 5 classes 25 images results

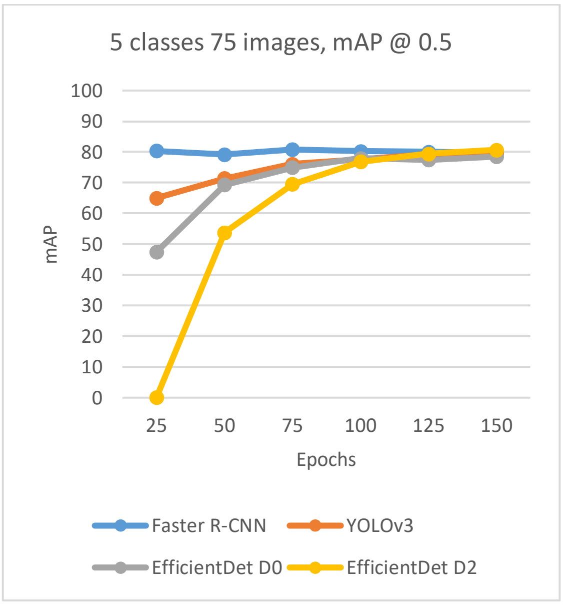


(a)

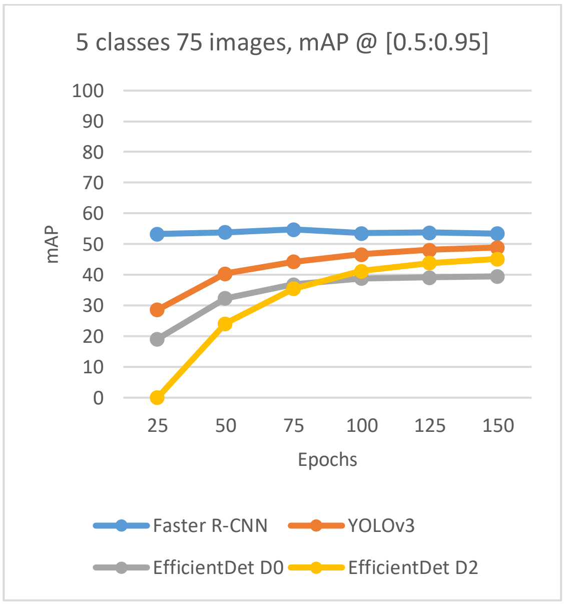


(b)

Figure 32. 5 classes 50 images results

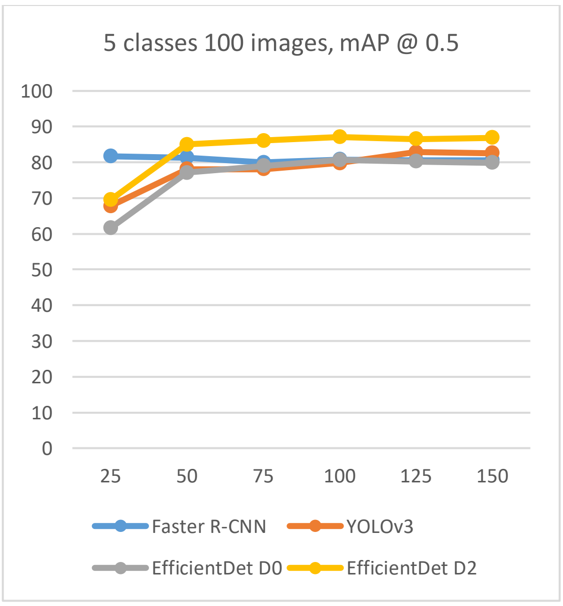


(a)

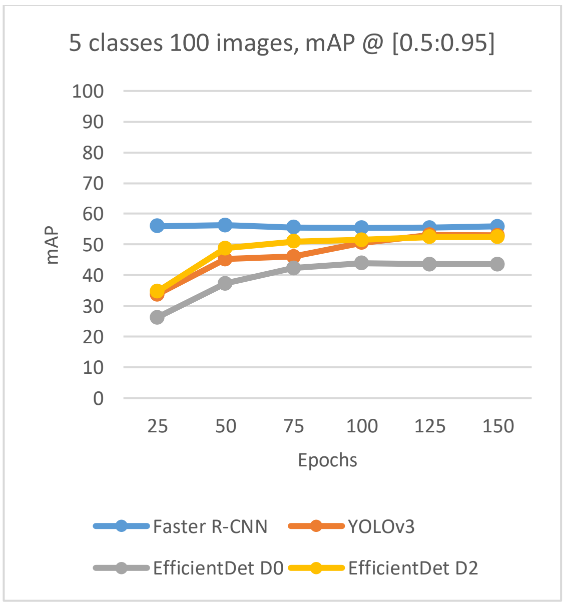


(b)

Figure 33. 5 classes 75 images results



(a)



(b)

Figure 34. 5 classes 100 images results

6.6 Six classes

The classes for these evaluations are church, cannon, tractor, lighthouse, monument, and tank.

With the addition of the tank class, the overall accuracy has stayed on a similar level to having five classes. However, the localization accuracy has risen by a bit. This could be attributed to the object being defined more clearly. That is, it's not part of the environment but a stand-alone object with a clearly defined start and end. Examples of tanks can be seen in figure 35.

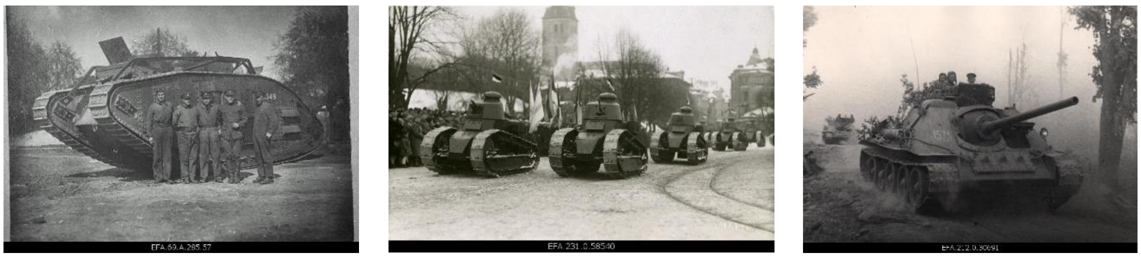


Figure 35. Tank examples

While EfficientDet D2 was able to surpass Faster R-CNN in two datasets with five classes, figures 32(a) and 34(a), then now it has surpassed it in 3 dataset sizes as seen in figures 37(a) – 39(a). YOLOv3 was able to match Faster R-CNN performance, while EfficientDet D0 lagged far behind in all the datasets, as seen in figures 36(a) – 39(a).

However, in terms of per-epoch training performance and localization accuracy, Faster R-CNN was still the best as seen in figures 36(b) – 39(b).

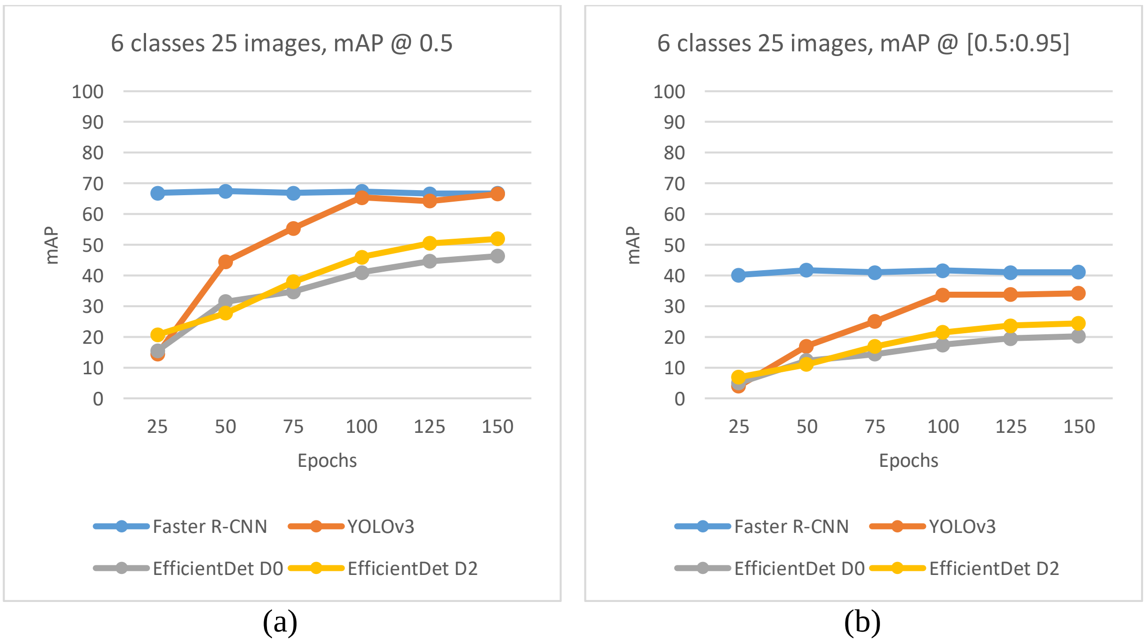
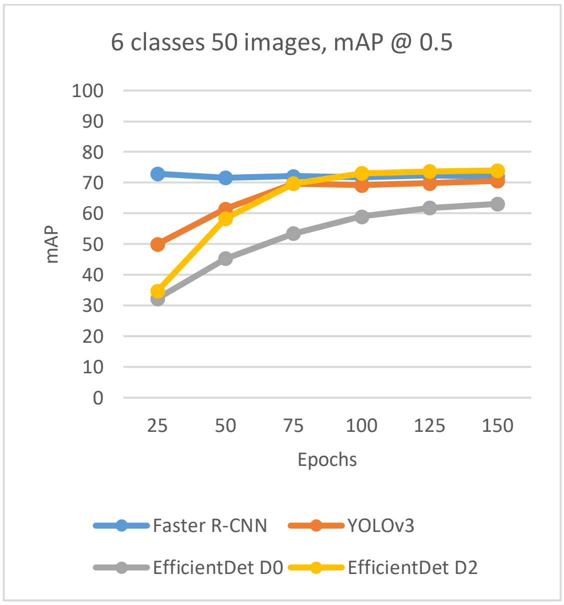
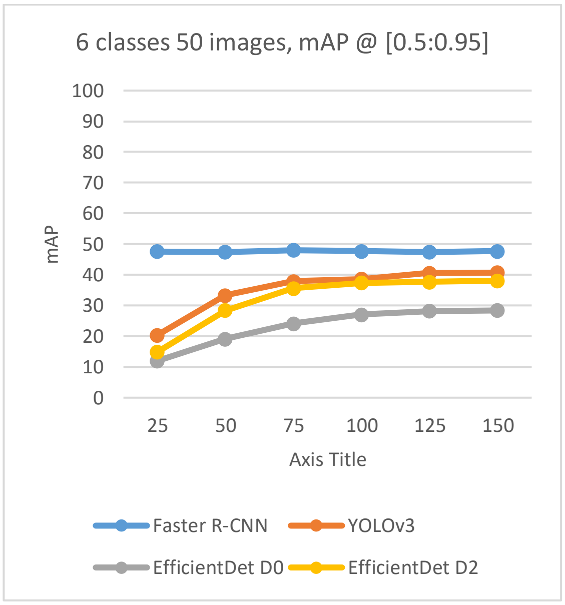


Figure 36. 6 classes 25 images results

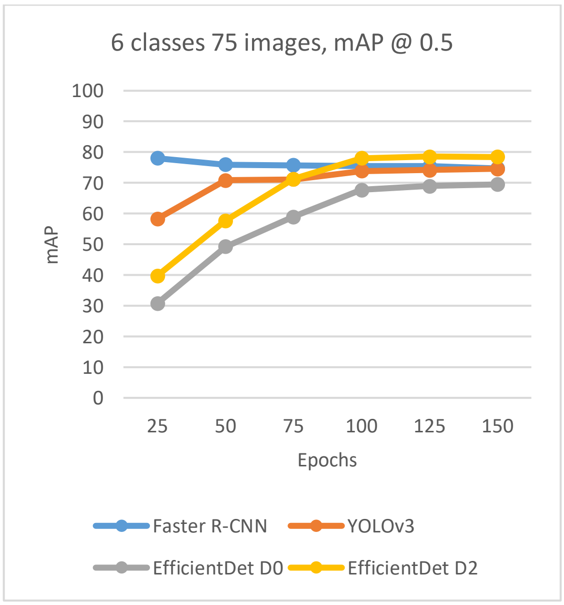


(a)

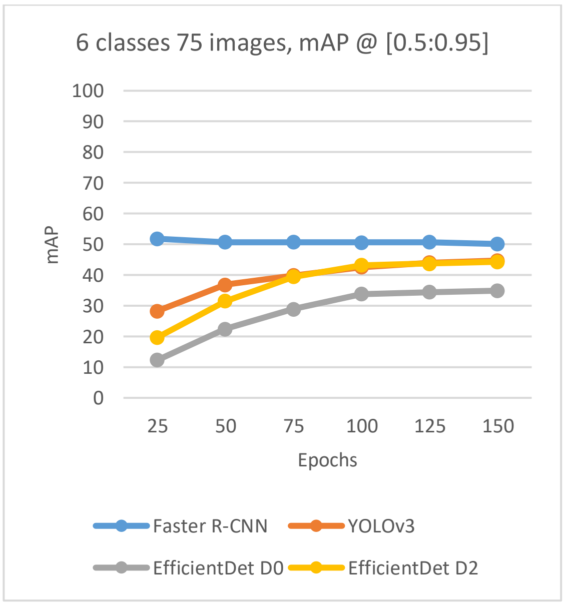


(b)

Figure 37. 6 classes 50 images results

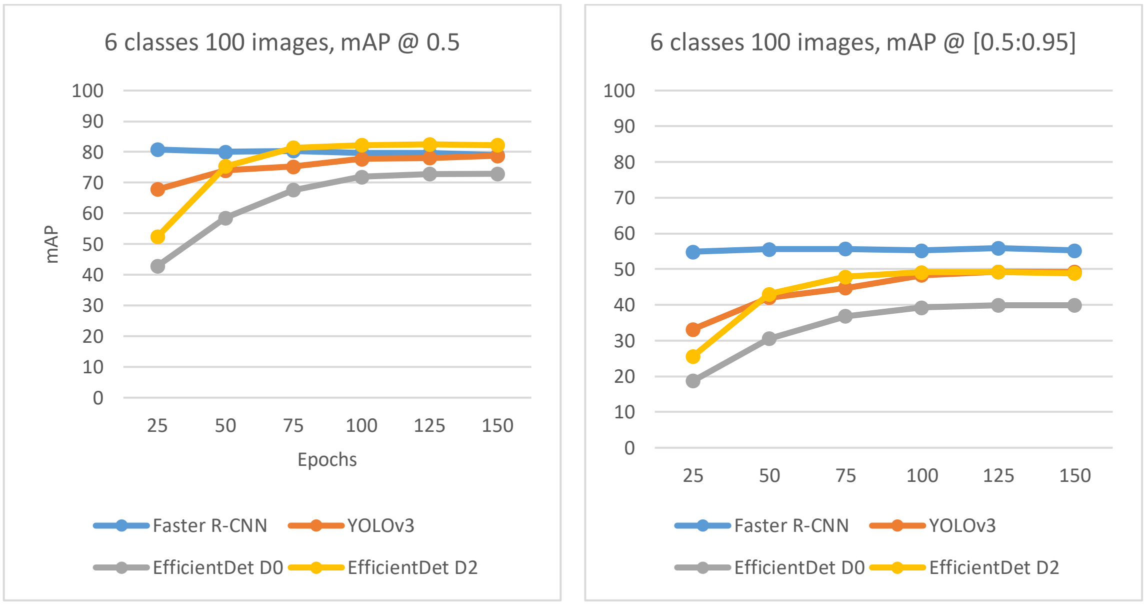


(a)



(b)

Figure 38. 6 classes 75 images results



(a) (b)

Figure 39. 6 classes 100 images results

6.7 Seven classes

The classes for these evaluations are church, cannon, tractor, lighthouse, monument, tank, and bridge.

The detection accuracy remained at a similar level as with six classes, however, localization accuracy took a slight dip. This could be due to bridges not having such clearly defined borders. Some examples of the dataset are shown in figure 40.

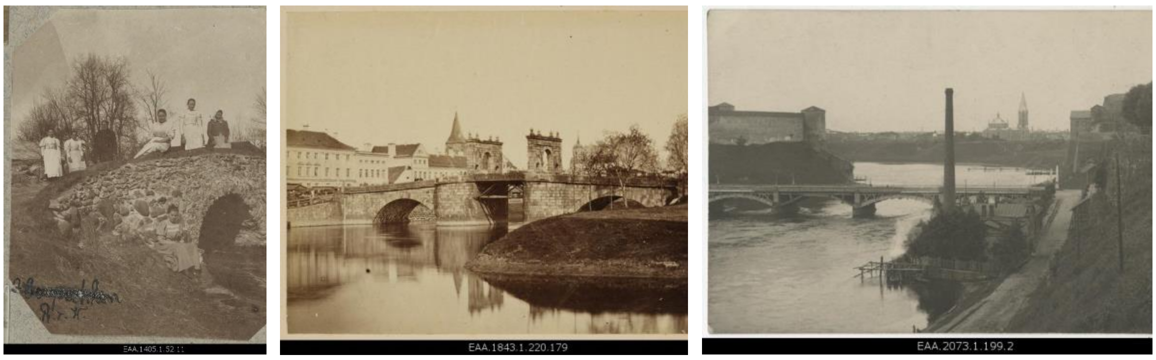
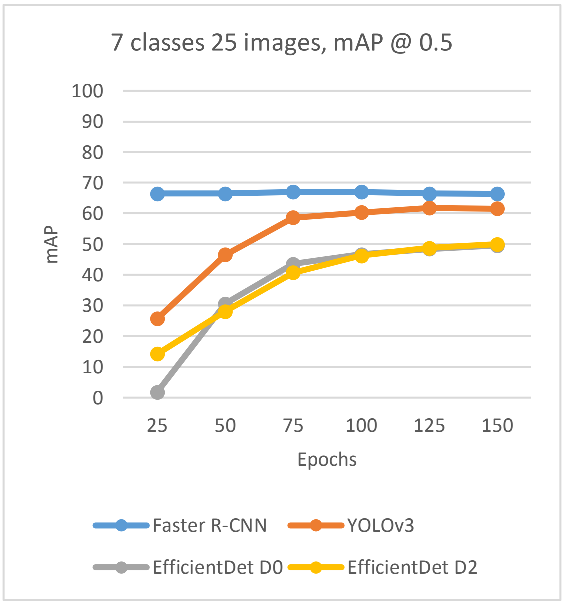
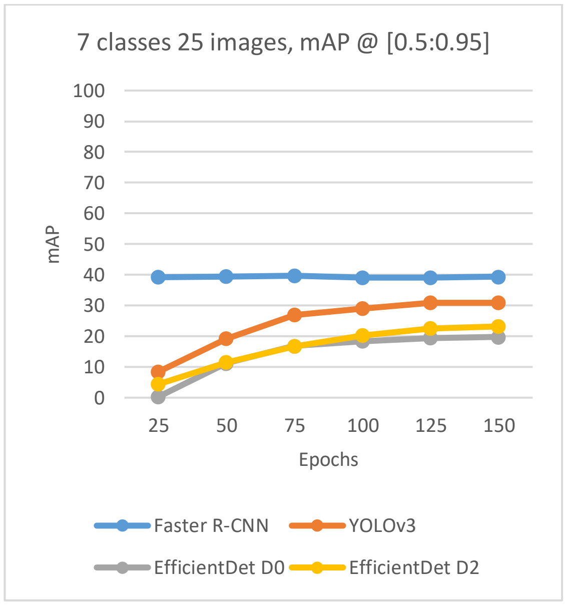


Figure 40. Bridge examples

Faster R-CNN continued being the best at object localization and per-epoch training performance, additionally, with this class, YOLOv3 and EfficientDet D2 could not overtake Faster R-CNN in any of the dataset sizes as seen in figures 41 - 44. EfficientDet D0 continued lagging behind.

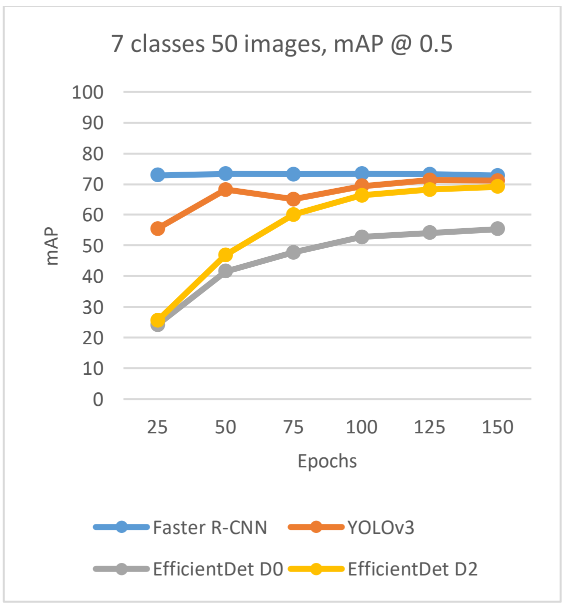


(a)

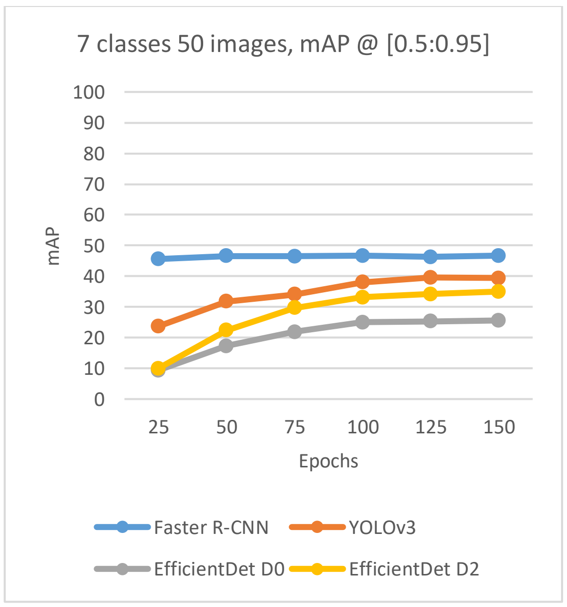


(b)

Figure 41. 7 classes 25 images results



(a)



(b)

Figure 42. 7 classes 50 images results

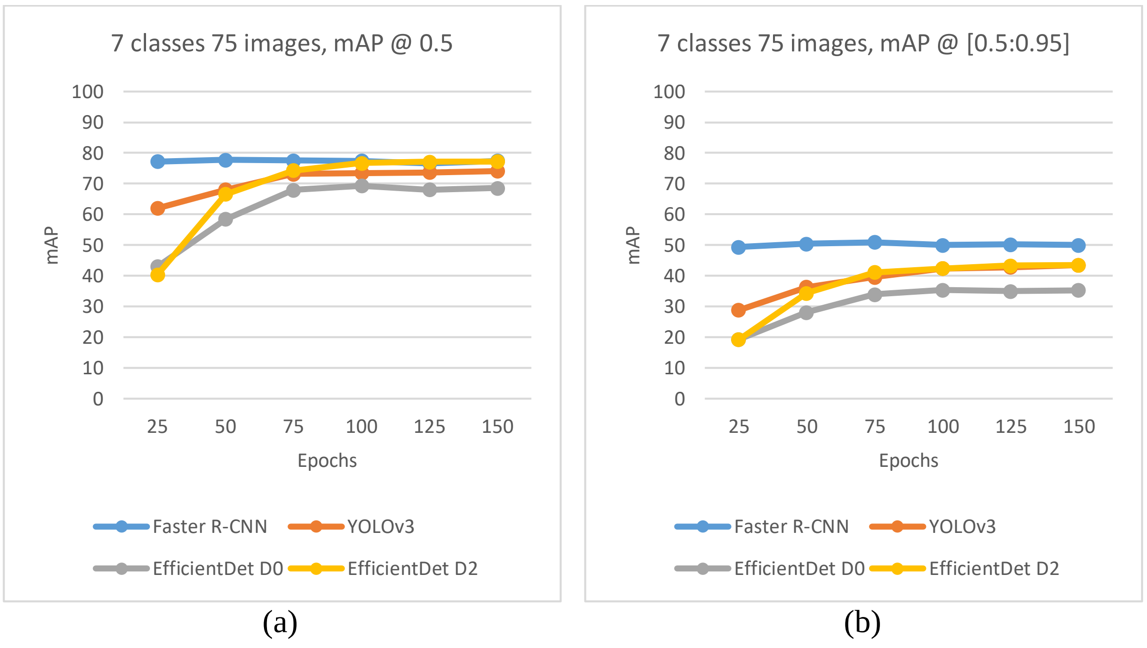


Figure 43. 7 classes 75 images results

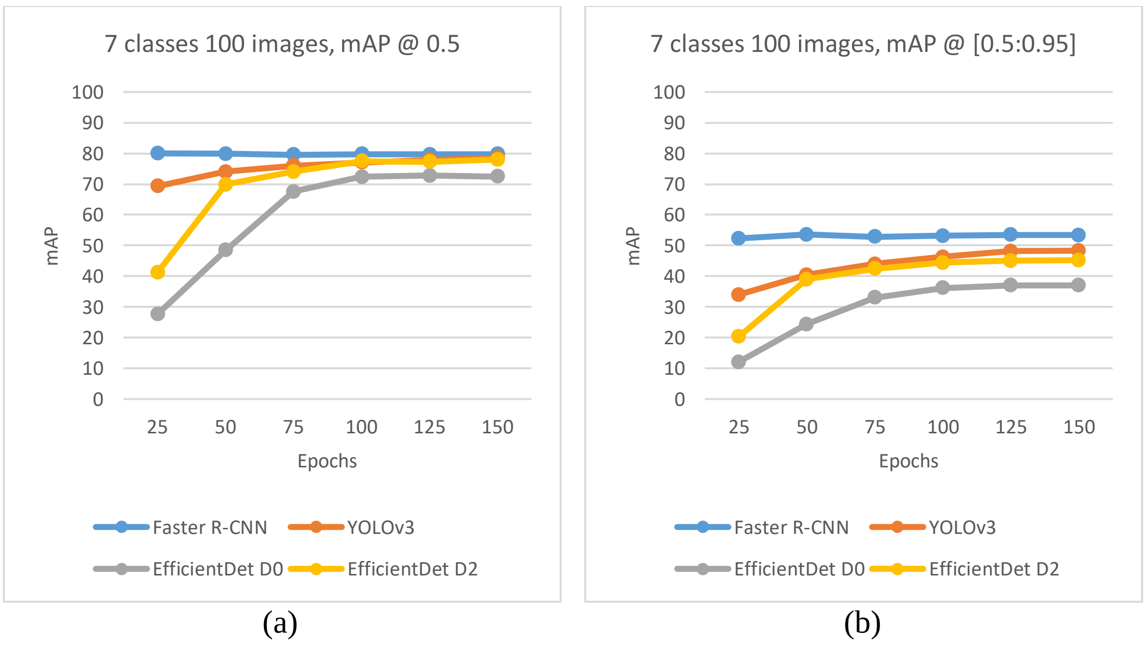


Figure 44. 7 classes 100 images results

6.8 Eight classes

The classes for these evaluations are church, cannon, tractor, lighthouse, monument, tank, bridge, and windmill.

Overall accuracy saw a slight increase with the addition of the windmill class. This can be due to it having clearly defined characteristic features and clear borders where it ends and starts. Some examples of the class can be seen in figure 45.

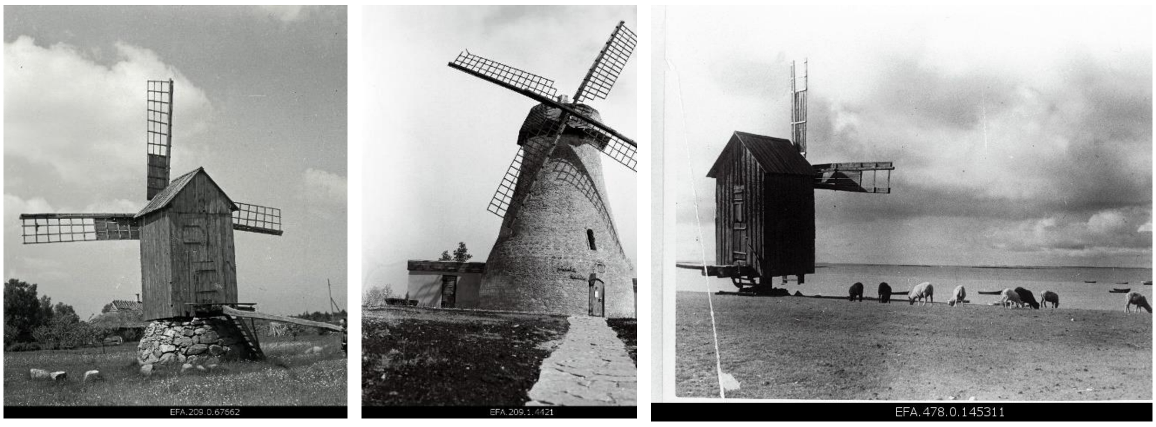
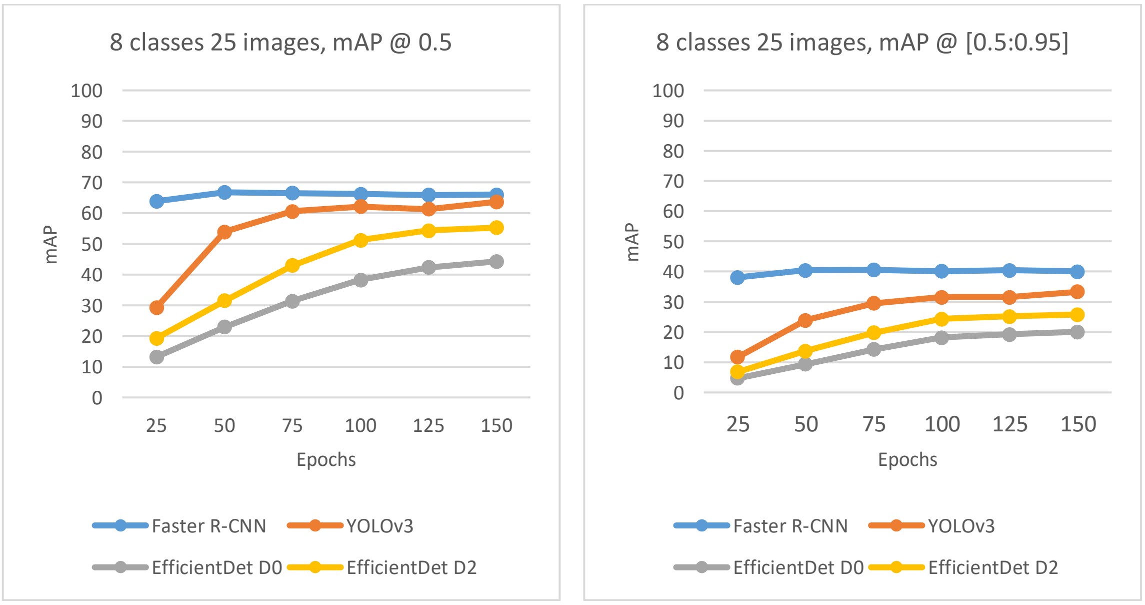


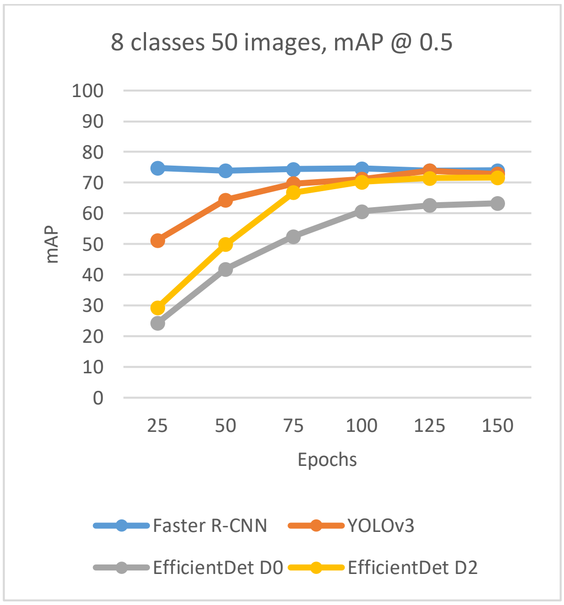
Figure 45. Windmill examples

Faster R-CNN remained the best at per-epoch training performance and the most accurate at localizing the objects as seen in figures 46(b) – 49(b). Additionally, it maintained the highest detection accuracy as seen in figures 46(a) – 49(a). YOLOv3 and EfficientDet D2 managed to keep pace with each other.

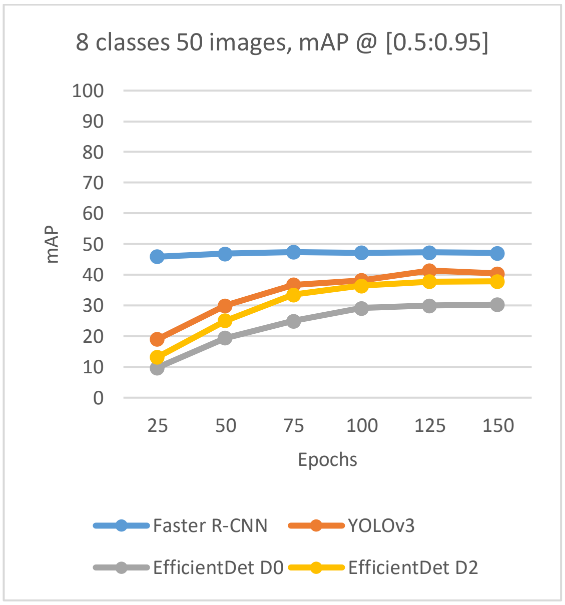


(a) (b)

Figure 46. 8 classes 25 images results

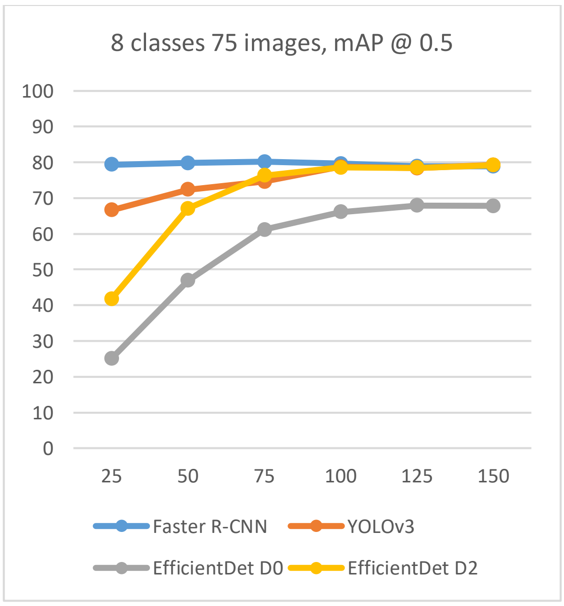


(a)

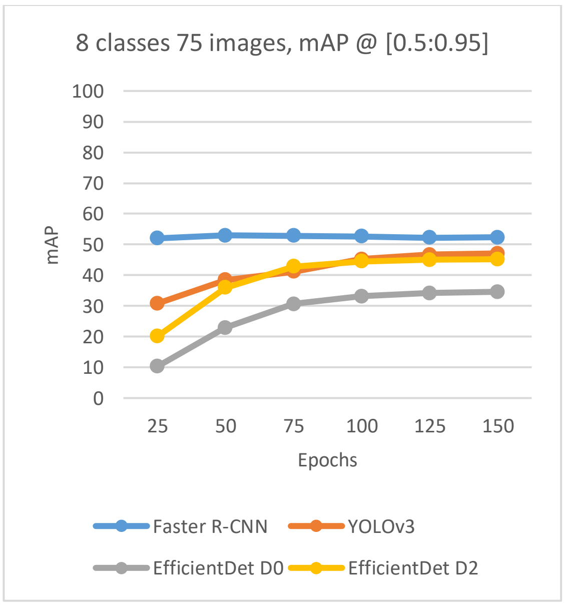


(b)

Figure 47. 8 classes 50 images results

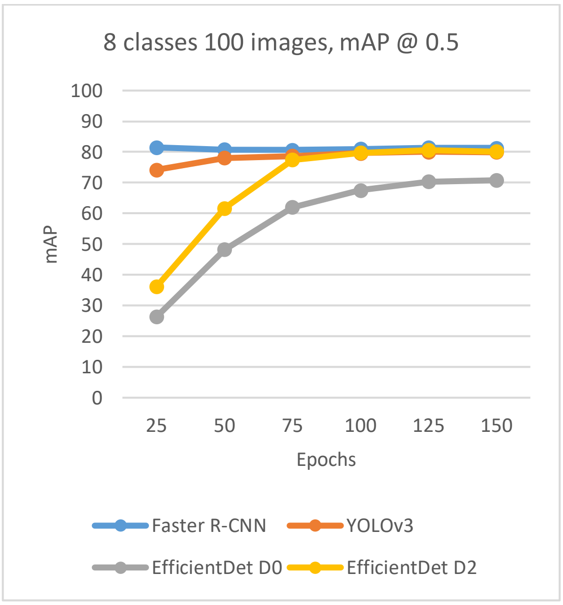


(a)

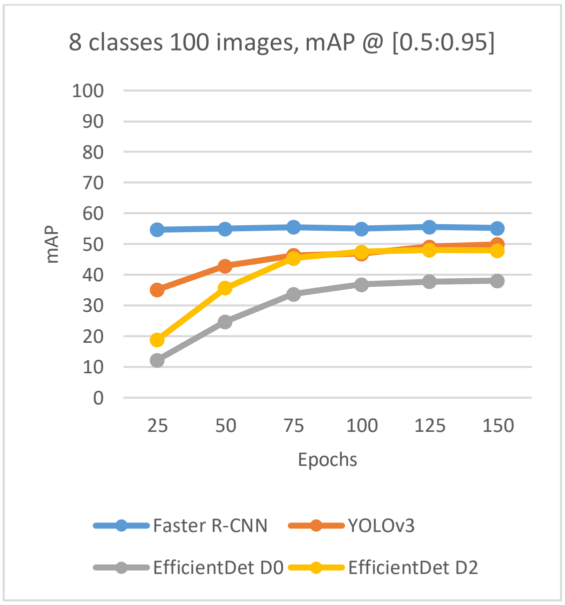


(b)

Figure 48. 8 classes 75 images results



(a)



(b)

Figure 49. 8 classes 100 images results

6.9 Nine classes

The classes for these evaluations are church, cannon, tractor, lighthouse, monument, tank, bridge, windmill, and altar.

The overall localization accuracy took a slight dip, when comparing to the accuracies of eight classes. The altar class is one that's embedded into the rest of its environment, thus it doesn't have such clearly defined borders of where it starts and ends. Some examples of the class can be seen in figure 50.

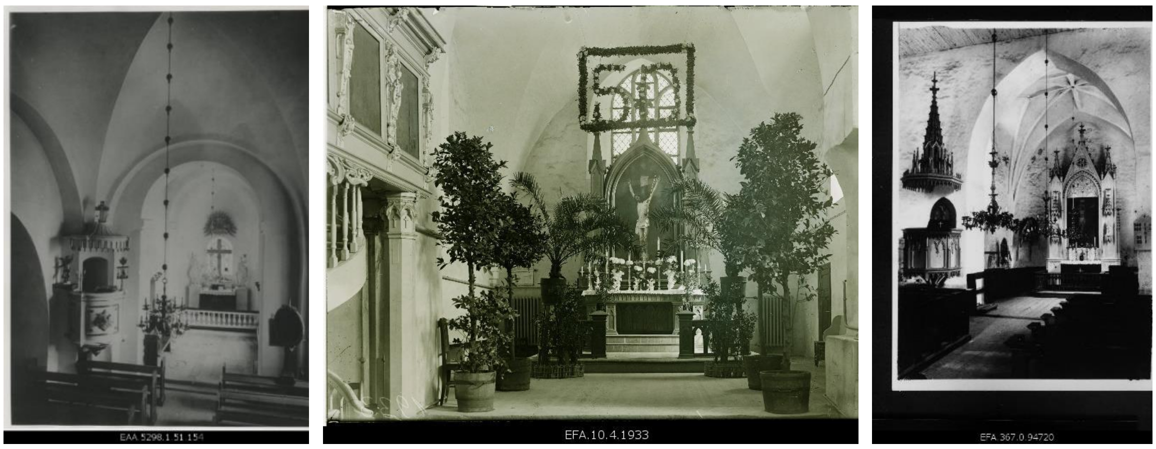
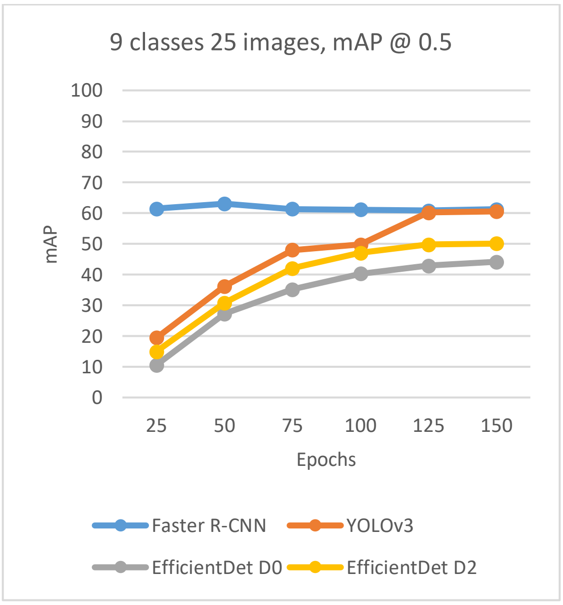


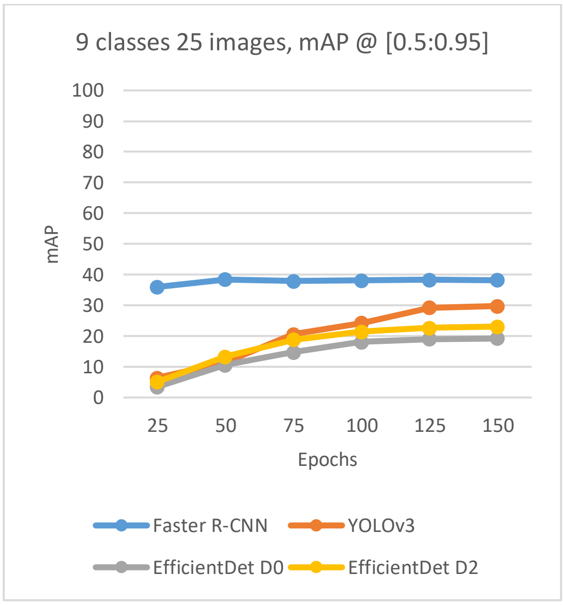
Figure 50. Altar examples

Faster R-CNN remained the most accurate at object detection and localization as seen in figures 51 – 54. Also, it had the best per-epoch training performance. YOLOv3 and

EfficientDet D2 came close to it with detection accuracy as seen in figures 51(a) – 54(a). However, what's interesting to note is that YOLOv3 came close to Faster R-CNN's performance with 25 images, while for most of the rest of the classes Faster R-CNN has been clearly ahead at 25 image datasets.

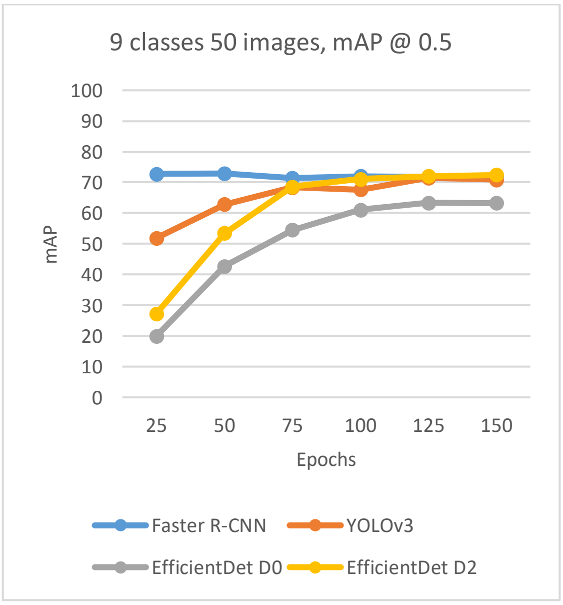


(a)

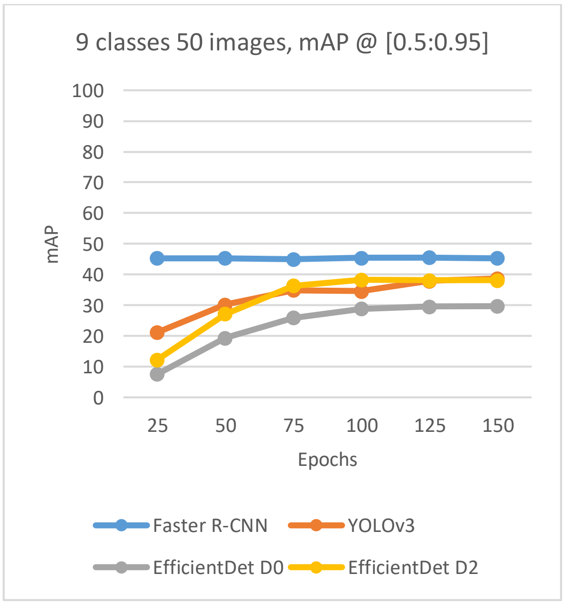


(b)

Figure 51. 9 classes 25 images results

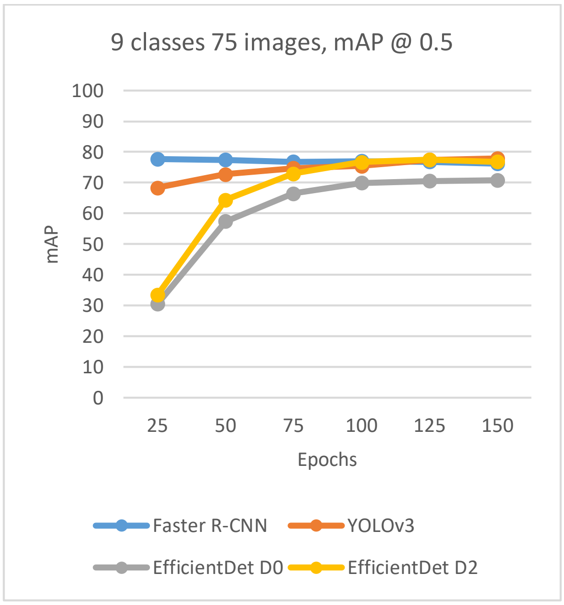


(a)

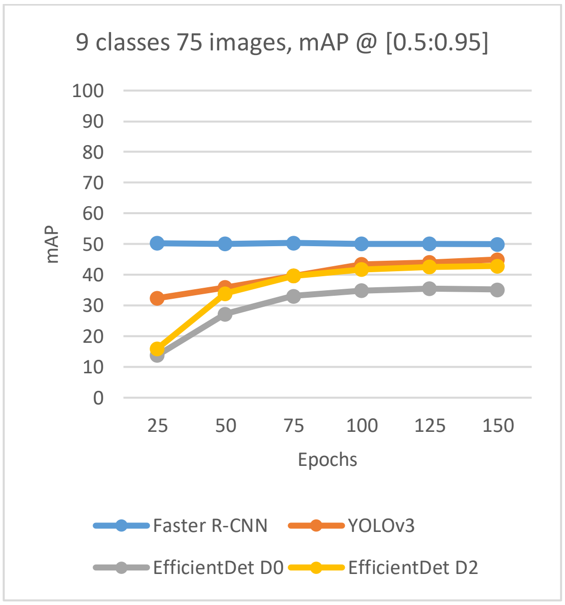


(b)

Figure 52. 9 classes 50 images results

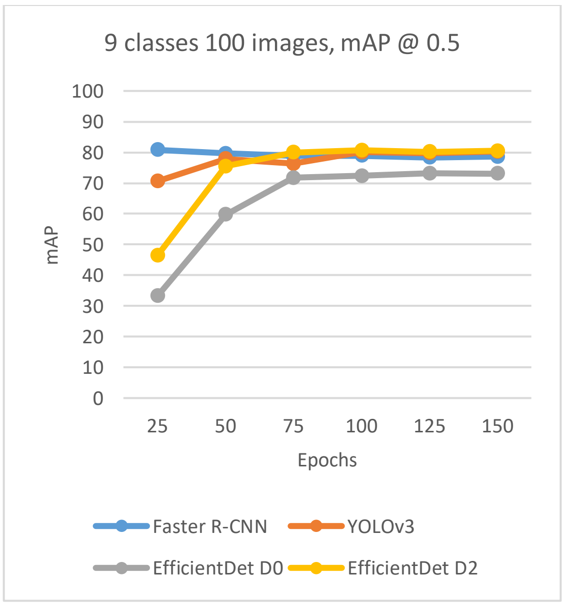


(a)

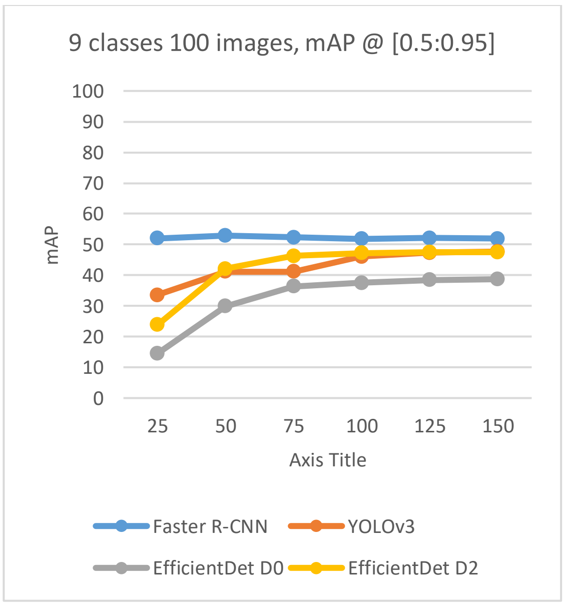


(b)

Figure 53. 9 classes 75 images



(a)



(b)

Figure 54. 9 classes 100 images results

6.10 Ten classes

The classes for these evaluations are church, cannon, tractor, lighthouse, monument, tank, bridge, windmill, altar, and ruins.

The final class added was ruins. It is the most abstract class, as ruins can be a crumbled castle, or a house damaged in war time. The uncertainty of the class can also be noted in the drop in accuracy. Some examples of ruins are shown in figure 55.

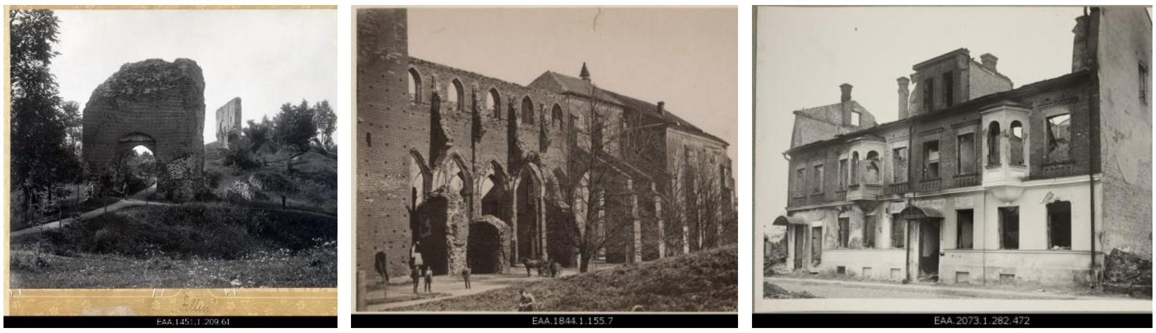
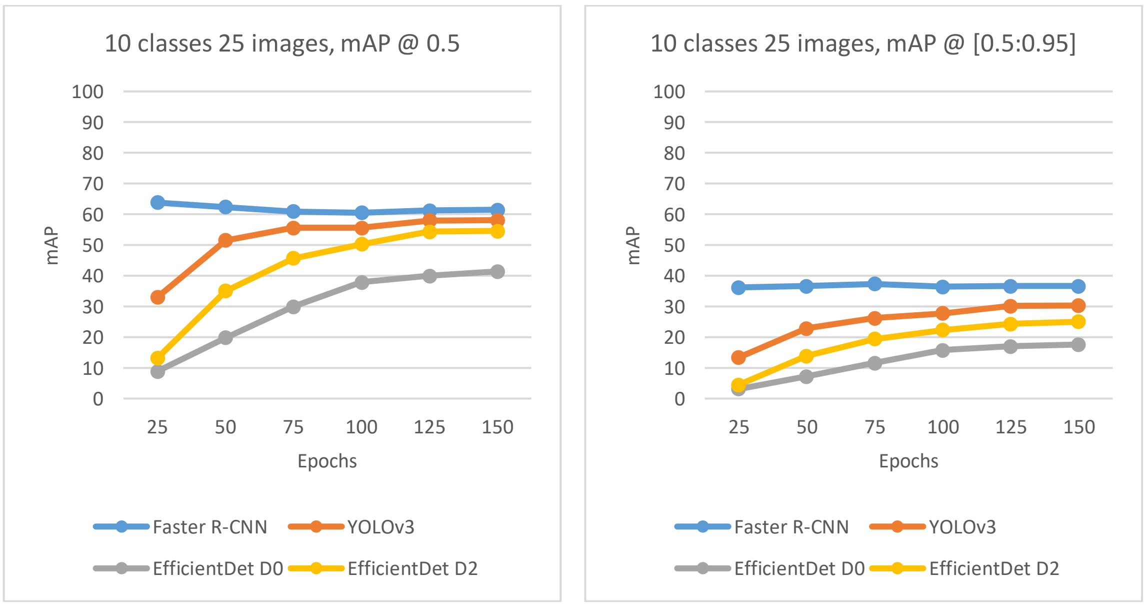


Figure 55. Ruins examples

A selection of detections over all the classes, using the 10 classes 100 images model, can be seen in figures A – N in the appendix.

Faster R-CNN's comparative localization accuracy doesn't change with the final class. It remains the most accurate at localizing as seen in figures 56(b) – 59(b). There is also no difference in the per-epoch training performance, in which Faster R-CNN is the best. However, YOLOv3 and EfficientDet D2 surpassed its detection accuracy at 100 images, as seen in figure 59(a).



(a) (b)

Figure 56. 10 classes 25 images results

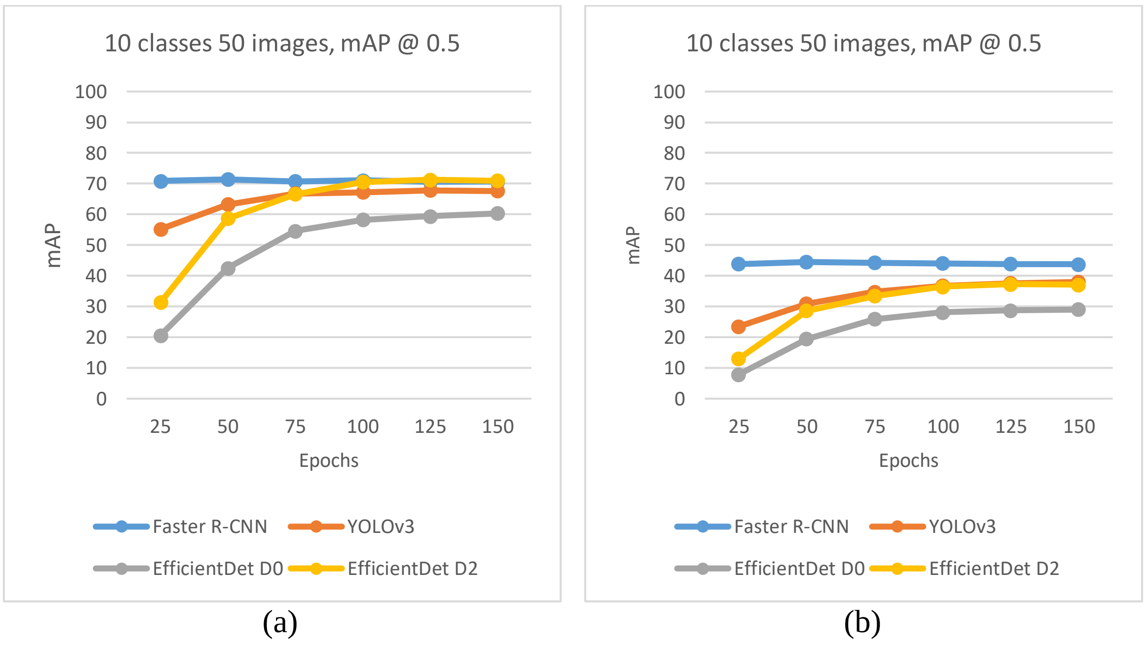


Figure 57. 10 classes 50 images results

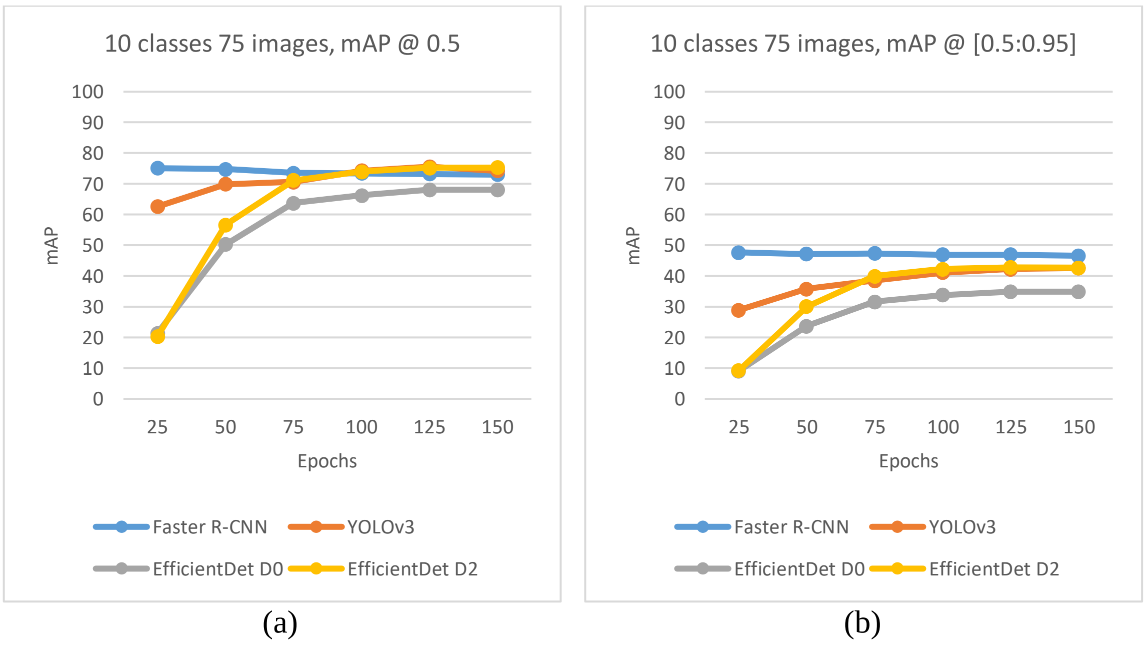
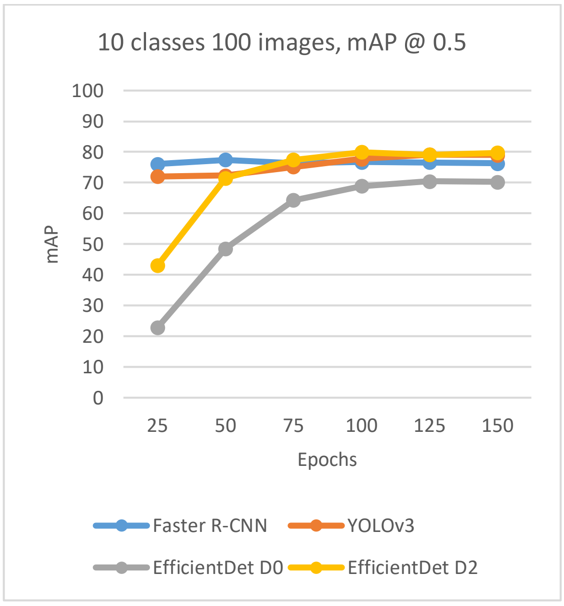
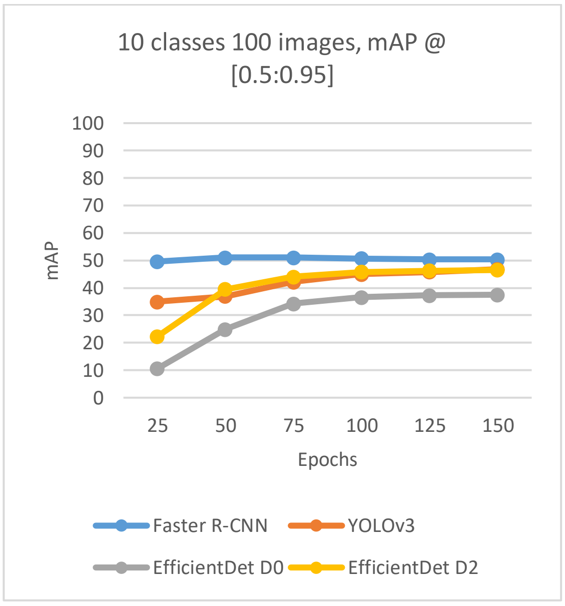


Figure 58. 10 classes 75 images results



(a)



(b)

Figure 59. 10 classes 100 images results

6.11 Detection speeds

The average time spent processing an image in the validation set is used to describe the detection speeds. The detection tests are run using a 16 GB Tesla P100. The models used are those trained with 10 classes and 100 images.

Table 5. Detection speeds

Algorithm	**Time spent per image (s)**
Faster R-CNN	0.10
YOLOv3	0.01
EfficientDet D2	0.14
EfficientDet D0	0.07

Table 5 shows results that are supported by existing literature. YOLOv3 is the fastest algorithm, followed by EfficientDet D0. Faster R-CNN and EfficientDet D2 are considerably slower.

As the environment where object detection is used is not a real time environment, then the detection speeds don't play into it that greatly. They can be used as a tiebreaker in choosing the appropriate algorithm.

6.12 Per class maximum mAPs

Looking at the per class overview of the 25 image datasets, then it can be noted that Faster R-CNN with Detectron2 is the most capable at generalizing from a smaller dataset as seen in figure 60. Additionally, it has the most accurate results in localizing the objects as can be seen in figure 61.

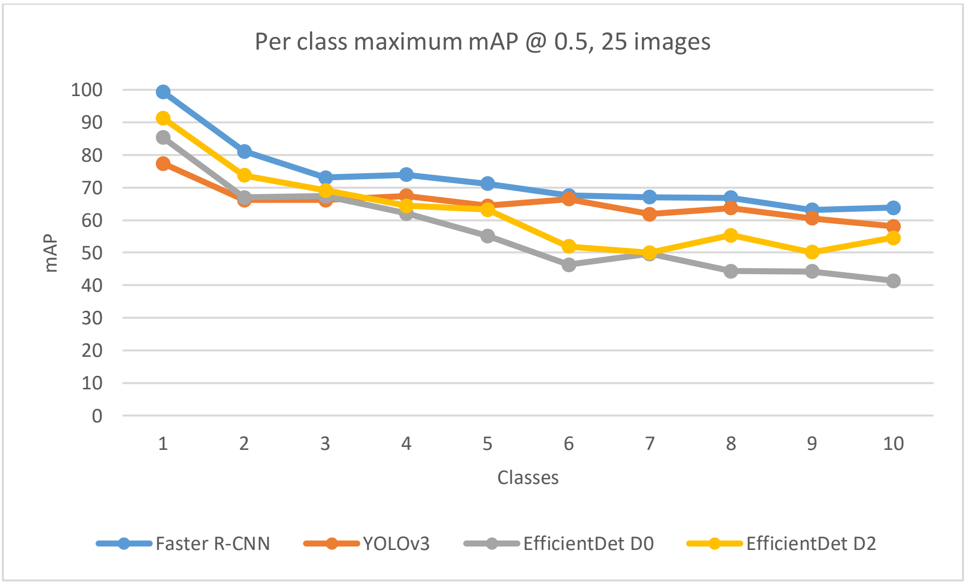


Figure 60. Per class maximum mAP @ 0.5, 25 images

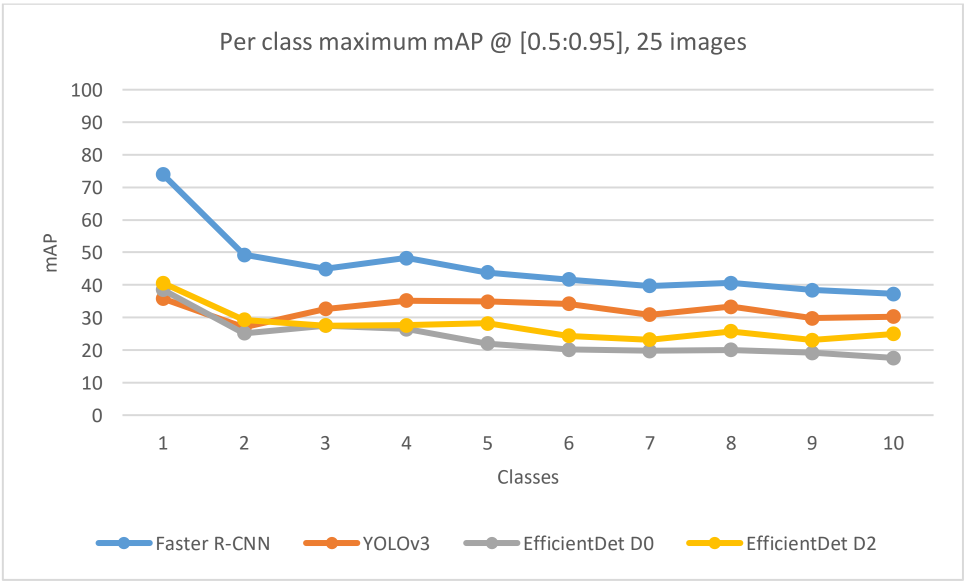


Figure 61. Per class maximum mAP @ [0.5:0.95], 25 images

The basic truth of machine learning holds true with the 100 images datasets. As more data gets added, then the algorithms become more accurate. Both YOLOv3 and EfficientDet D2 manage to catch up to Faster R-CNN and even surpass it at certain points. While EfficientDet D0 can match the performance of other algorithms at a smaller number of classes, then halfway through it starts falling behind as can be seen in figure 62.

However, while EfficientDet D2 and YOLOv3 catch up and even surpass Faster R-CNN in detection at certain points, then Faster R-CNN is still the most accurate at localizing with none of the other algorithms coming close to its results as can be seen in figure 63.

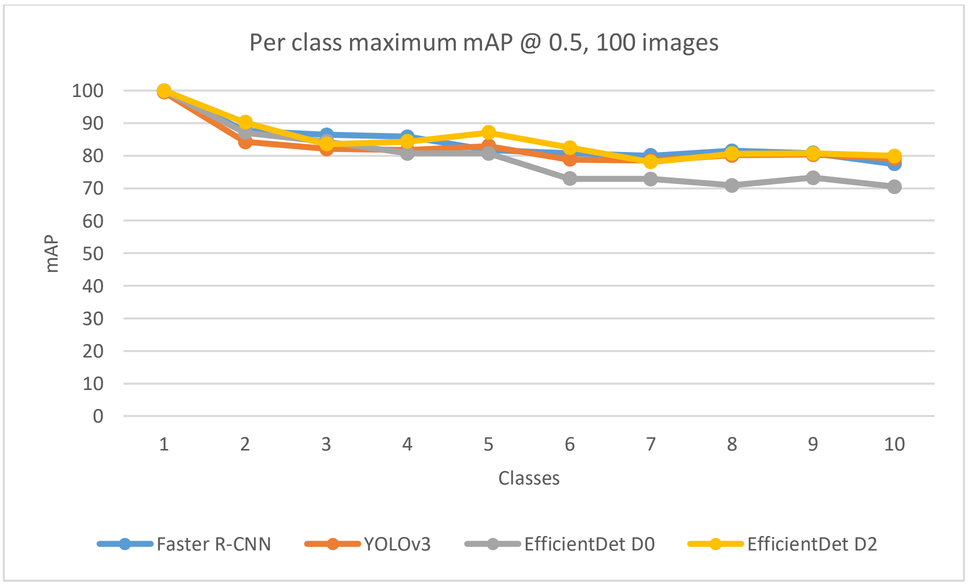


Figure 62. Per class maximum mAP @ 0.5, 100 images

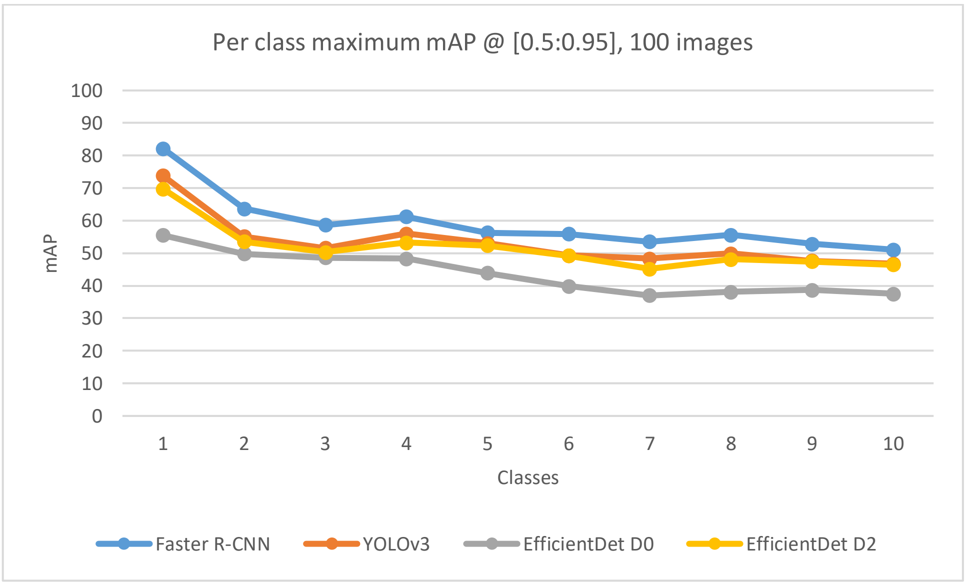


Figure 63. Per class maximum mAP @ [0.5:0.95], 100 images

When considering the average accuracy of the maximum mAPs @ 0.5 over all the classes, displayed in figure 64, then EfficientDet D2 gets a slightly higher accuracy than Faster R-CNN on dataset sizes of 100, with 0.47, and 50, with 0.38. Faster R-CNN gets a slight lead in the dataset size of 75 images with a mAP difference of 0.32. However, when looking at 25 images, then Faster R-CNN has the clear lead with a mAP difference between them of 10.32.

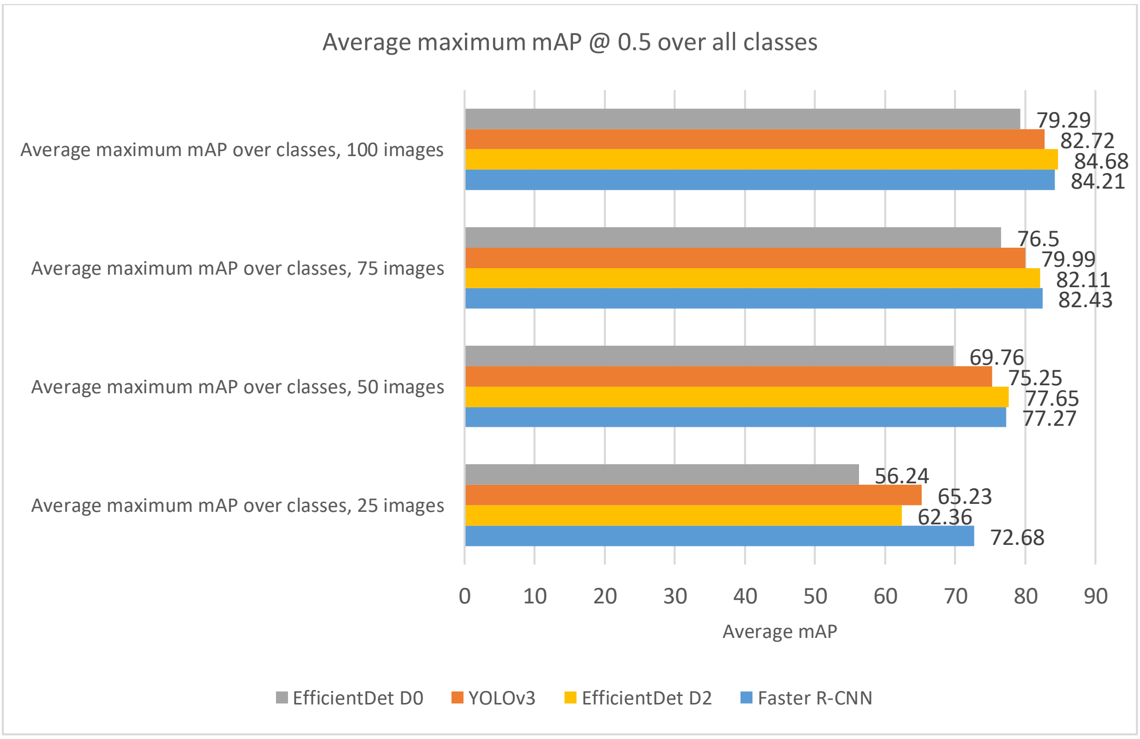


Figure 64. Average mAP over all classes

The difference in accuracy in the 50, 75, and 100 datasets are negligible. However, considering the limited nature of historical images, then 25 image datasets can feasibly happen. Thus, the detection accuracy differences with 25 images carries considerable weight. Additionally, getting the bounding box as accurate as possible can matter, as historical photos can have low quality and an offset bounding box can cause confusion.

Considering the accuracy, detection time, and being considerably faster at learning in terms of epochs, from the algorithms tested, Detectron2's implementation of Faster R-CNN is the best fit for automatically labelling historical images.

7 Summary

Over the course of this thesis an annotation solution was created, and an algorithm was chosen to integrate automatic labelling using object detection into the Ajapaik web environment. Ajapaik is an online historical image archive.

Existing annotation solutions were considered for the environment but there were drawbacks in how input was handled. They drew boxes in one of two ways. One of the ways was by drawing a box of pre-defined height and width at the centre of the screen when the annotation process was started. This requires the user to perform extra actions by having to resize and move the drawn box. The other was by having the user holding down the mouse button and dragging. This eliminates compatibility with touch screen devices. Thus, an annotation solution was developed for the website.

Three aspects were considered when choosing an algorithm for automatic labelling.

1) Which algorithm learns the fastest in terms of epochs?

2) How do the algorithms' accuracies increase with additional data?

3) How fast is the detection speed of the different algorithms?

Four algorithms were used for the comparisons. Detectron2's implementation of Faster R-CNN, YOLOv3, EfficientDet D2, and EfficientDet D0.

Detectron2's implementation of Faster R-CNN was found to provide the fastest increase in accuracy in terms of epochs with it achieving its maximum, or near maximum, value by epoch 25. EfficientDet D2 was able to pass Faster R-CNN in average object detection accuracy for datasets with 50, and 100 images by a negligible margin. However, with 25 images, Faster R-CNN had a strong lead when compared to the other algorithms. YOLOv3's object detection accuracy fluctuated between matching and slightly falling behind EfficientDet D2 and Faster R-CNN. EfficientDet D0 had the worst performance on most cases.

Detectron2's implementation of Faster R-CNN was also the most accurate in terms of object localization accuracy, with none of the other algorithms coming close. It was followed by EfficientDet D2, YOLOv3, with EfficientDet D0 trailing considerably behind.

In terms of detection speed, YOLOv3 was the fastest, as expected, based on existing literature. It was followed by EfficientDet D0, Faster R-CNN, and EfficientDet D2. As the environment isn’t a real time system, then the speed differences didn’t carry as much weight as accuracies.

Considering the above findings, Detectron2’s implementation of Faster R-CNN was concluded to be the best at the task of automatic labelling in the context of historical images.

Further work must be done to integrate Faster R-CNN into Ajapaik's environment. The data flow between Ajapaik and Faster R-CNN must be planned. However, a bigger challenge is creating a training environment. Considering the hardware and funding limitations, then a possible solution could be integration with Google Colab.

Further research can be done to investigate the inner workings of Detectron2 to see how it is able to learn so fast. Additionally, different configuration options could be tested to see if EfficientDet D0 could match YOLOv3’s performance, as claimed by the EfficientDet paper [10].

References

[1] Fotis. [WWW] http://www.ra.ee/fotis (20.04.2020)

[2] Rahvusarhiiv. [WWW] https://www.ra.ee/ (20.04.2020)

[3] Albumid. [WWW] https://sakulugu.ee/pildid/ (21.04.2020)

[4] Vanadpildid. [WWW] https://www.vanadpildid.net/ (21.04.2020)

[5] Ajapaik. [WWW] https://ajapaik.ee/ (21.04.2020)

[6] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2015) You Only Look Once: Unified, Real-Time Object Detection. arXiv:1506.02640.

[7] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C. (2015). SSD: Single Shot MultiBox Detector. arXiv:1512.02325.

[8] Girshick, R., Donahue, J., Darrell, T., Malik, J. (2013) Rich feature hierarchies for accurate object detection and semantic segmentation. arXiv:1311.2524.

[9] Ren, S., He, K., Girshick, R., Sun, J. (2015) Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. arXiv:1506.01497.

[10] Tan, M., Pang, R., Le, Q.V. (2019) EfficientDet: Scalable and Efficient Object Detection. arXiv:1911.09070.

[11] Rosebrock, A. Intersection over Union (IoU) for object detection. [WWW] https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (22.04.2020)

[12] COCO – Common Objects in Context. [WWW] http://cocodataset.org (22.04.2020)

[13] Tan, M., Le, Q.V. (2019) EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv:1905.11946.

[14] The PASCAL Visual Object Classes Homepage. [WWW] http://host.robots.ox.ac.uk/pascal/VOC/ (22.04.2020)

[15] ImageNet. [WWW] http://image-net.org/about-stats (22.04.2020)

[16] Open Images V6 – Description. [WWW] https://storage.googleapis.com/openimages/web/factsfigures.html (22.04.2020)

[17] Search Home | Museoiden Finna. [WWW] https://museot.finna.fi/?lng=en-gb (22.04.2020)

[18] Annotorious – Image Annotation for the Web. [WWW] https://annotorious.github.io/index.html (23.04.2020)

[19] marker.js - Javascript image annotation library. [WWW] https://markerjs.com/ (23.04.2020)

[20] Easy Annotation – a Javascript Image Annotation Library. [WWW] http://easyannotation.com/index.html (23.04.2020)

[21] Wikidata:Data access – Wikidata. [WWW] https://www.wikidata.org/wiki/Wikidata:Data_access (23.04.2020)

[22] Powerful and efficient Computer Vision Annotation Tool. [WWW] https://github.com/opencv/cvat (23.04.2020)

[23] Labelbox: The leading training data platform. [WWW] https://labelbox.com/ (26.04.2020)

[24] Welcome to Colaboratory – Colaboratory. [WWW] https://colab.research.google.com/ (29.04.2020)

[25] YOLOv3 in Pytorch > ONNX > CoreML > iOS. [WWW] https://github.com/ultralytics/yolov3 (29.04.2020)

[26] Detectron2 is FAIR's next-generation platform for object detection and segmentation. [WWW] https://github.com/facebookresearch/detectron2 (29.04.2020)

[27] Google Brain AutoML. [WWW] https://github.com/google/automl (29.04.2020)

[28] Detectron2 Model Zoo and Baselines. [WWW] https://github.com/facebookresearch/detectron2/blob/master/MODEL_ZOO.md (25.04.2020)

[29] YOLO: Real-Time Object Detection. [WWW] https://pjreddie.com/darknet/yolo/ (25.04.2020)

[30] Uijlings, J. R., van de Sande, K. E., Gevers, T., Smeulders, A. W. (2013). Selective search for object recognition - *International Journal of Computer Vision (IJCV).*

[31] Kathuria, A. What's new in YOLO v3? [WWW] https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b (28.04.2020)

[32] Liu, S., Qi, L., Qin, H., Shi, J., Jia, J. (2018) Path Aggregation Network for Instance Segmentation. arXiv:1803.01534.

[33] Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. – *NIPS*

[34] Zhou, X., Wang, D., Krähenbühl, P. (2019) Objects as Points. arXiv:1904.07850

[35] Redmon, J., Farhadi, A. (2018) YOLOv3: An Incremental Improvement. arXiv:1804.02767

[36] Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P. (2017) Focal Loss for Dense Object Detection. arXiv:1708.02002

[37] He, K., Zhang, X., Ren, S., Sun, J. (2015) Deep Residual Learning for Image Recognition. arXiv:1512.03385

[38] Ghiasi, G., Lin, T.Y., Pang, R., Le, Q.V. (2019) NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. arXiv:1904.07392

[39] Fu, C.Y, Liu, W., Ranga, A., Tyagi, A., Berg, A.C. (2017) DSSD: Deconvolutional Single Shot Detector. arXiv:1701.06659

[40] Redmon, J., Farhadi, A. (2016) YOLO9000: Better, Faster, Stronger. arXiv:1612.08242

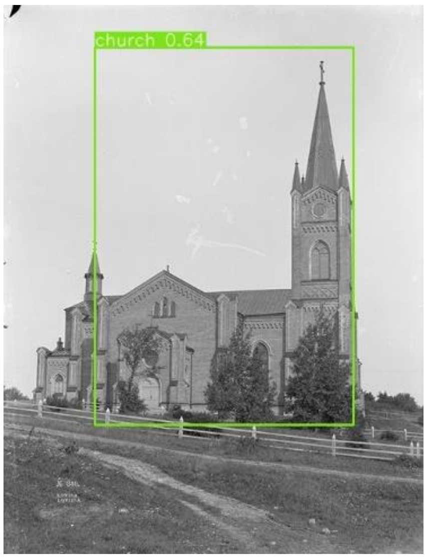
[41] Dai, J., Li, Y., He, K., Sun, J. (2016) R-FCN: Object Detection via Region-based Fully Convolutional Networks. arXiv:1605.06409

[42] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S. (2016) Feature Pyramid Networks for Object Detection. arXiv:1612.03144

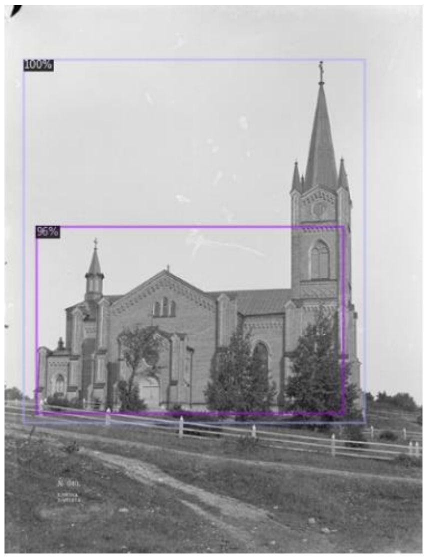
[43] Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K. (2016) Aggregated Residual Transformations for Deep Neural Networks. arXiv:1611.05431

[44] ImageNet Large Scale Visual Recognition Competition (ILSVRC)
[WWW] http://www.image-net.org/challenges/LSVRC/ (12.05.2020)

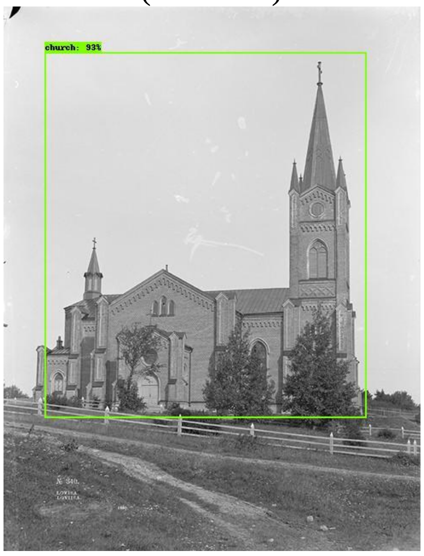
Appendix



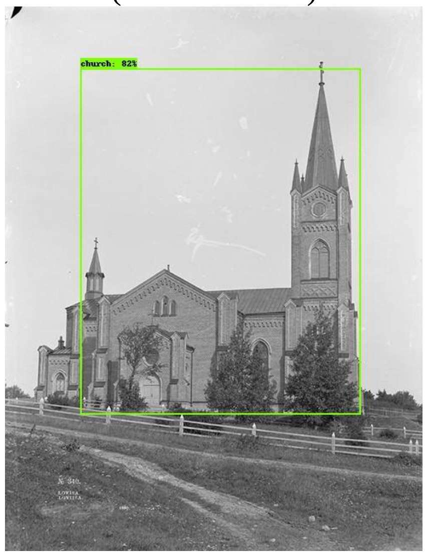
(YOLOv3)



(Faster R-CNN)



(EfficientDet D2)



(EfficientDet D0)

Figure A. Partially unsuccessful church detection examples, 10 classes 100 images model

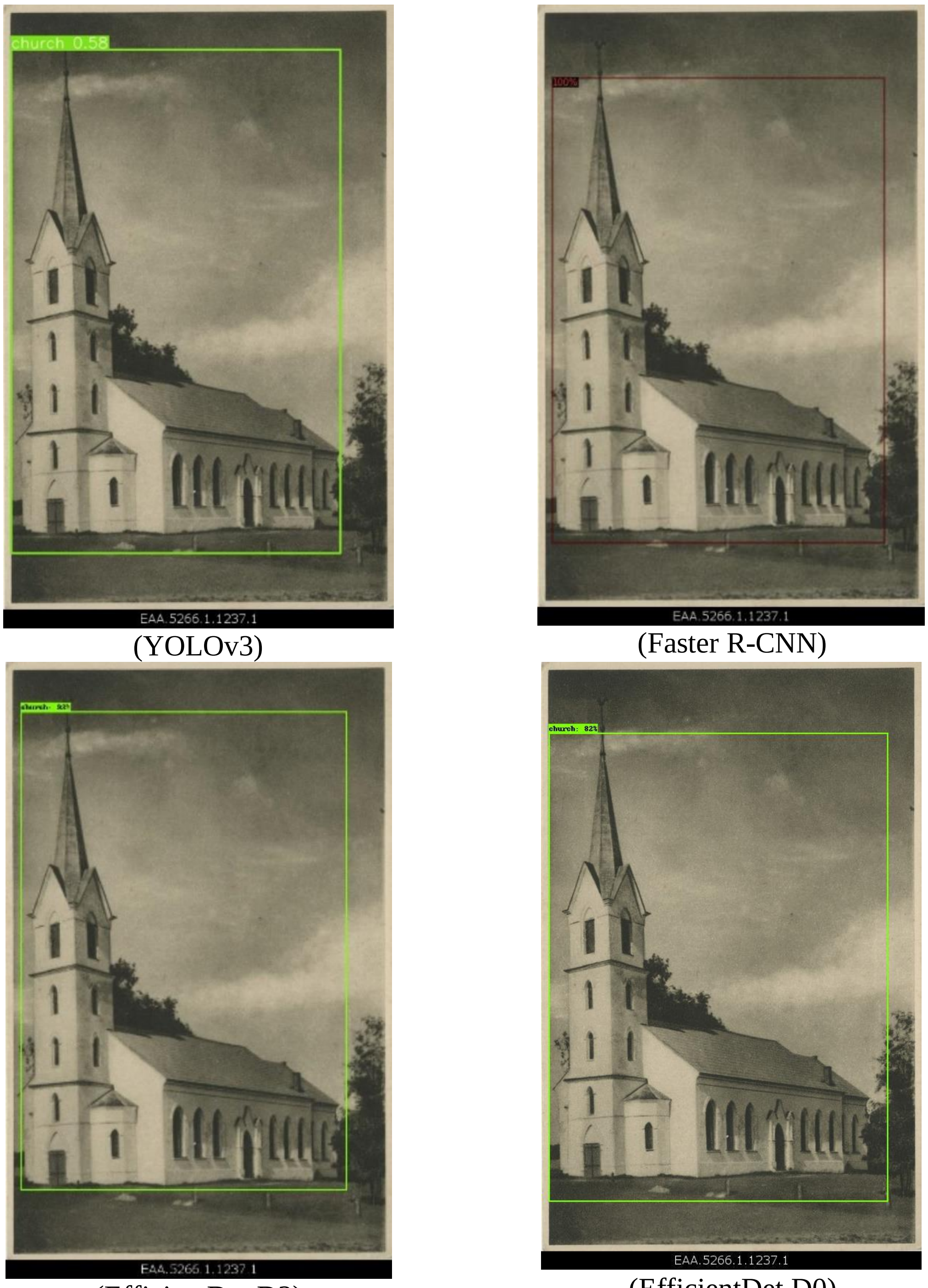
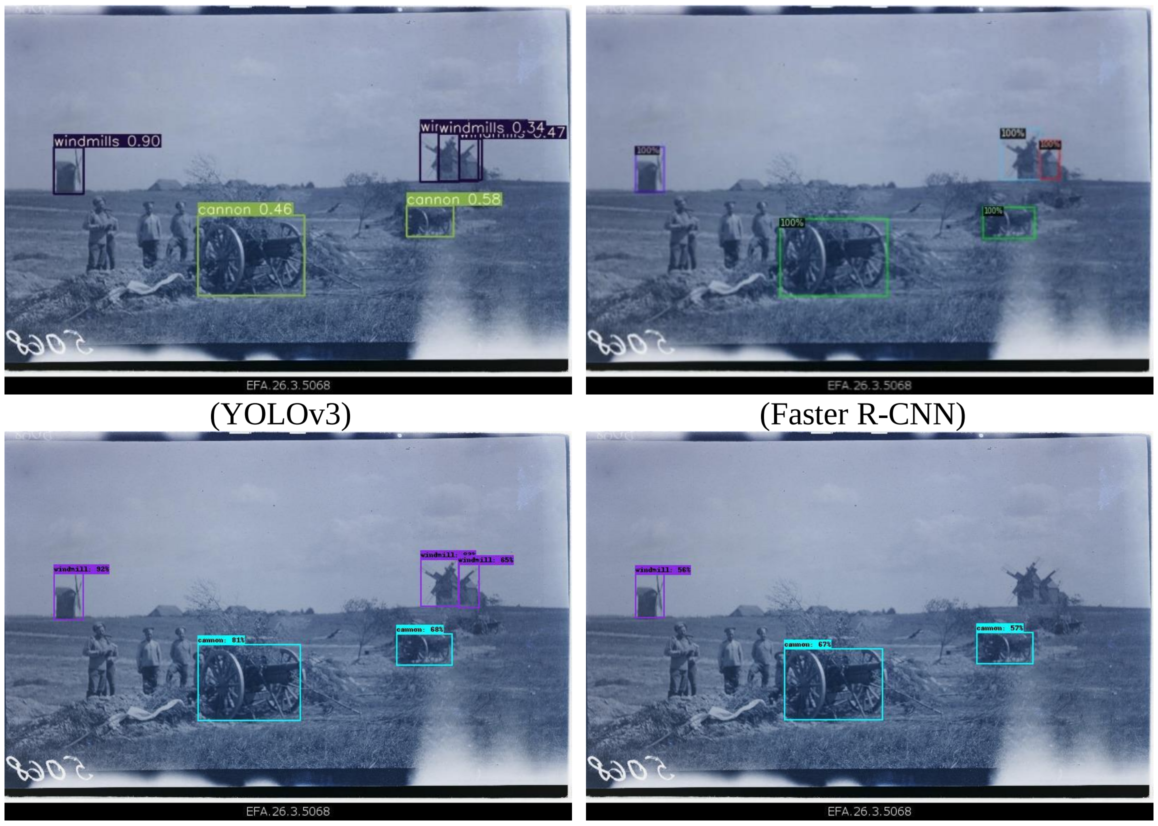


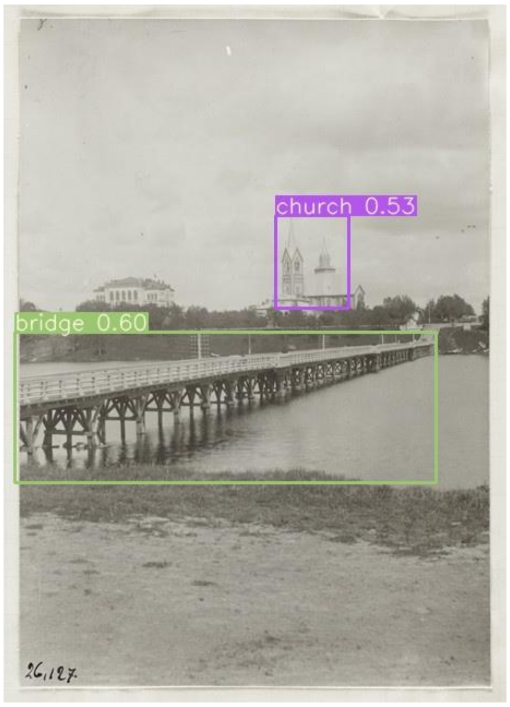
Figure B. Successful church detection examples, 10 classes 100 images model



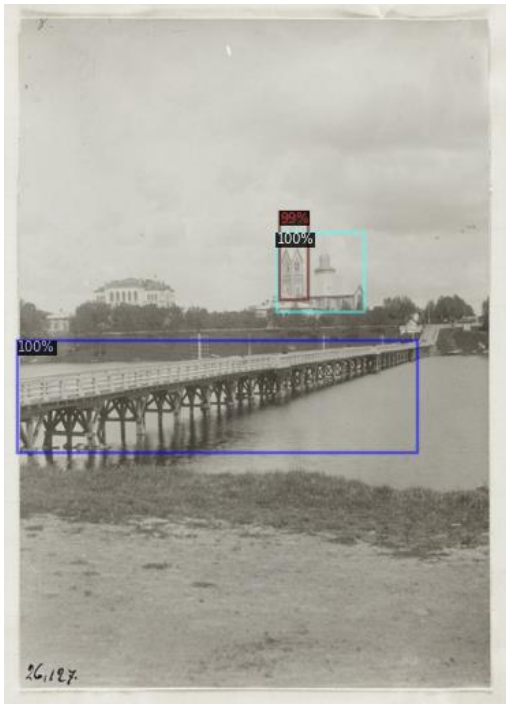
(YOLOv3) (Faster R-CNN)

(EfficientDet D2) (EfficientDet D0)

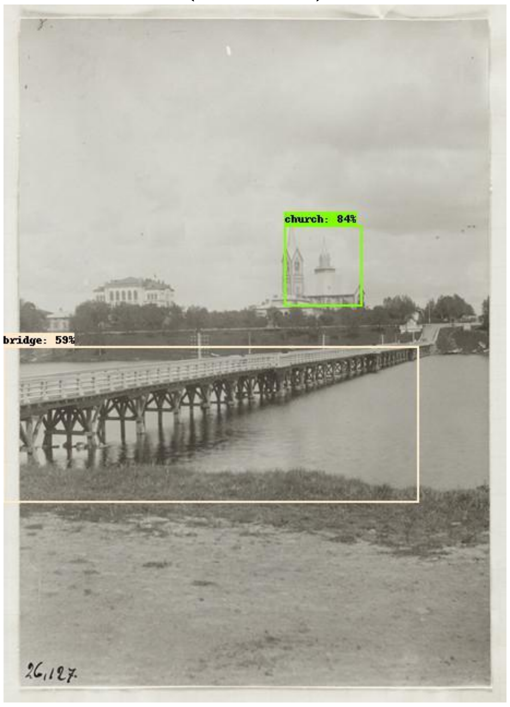
Figure C. Cannon and windmill detection examples, 10 classes 100 images



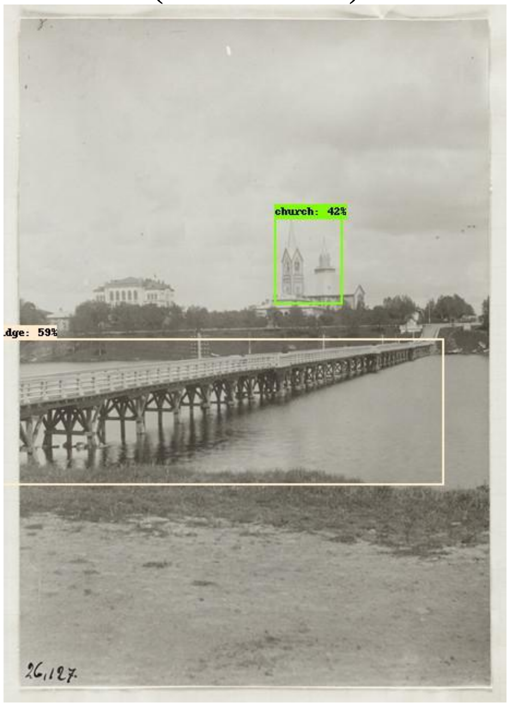
(YOLOv3)



(Faster R-CNN)

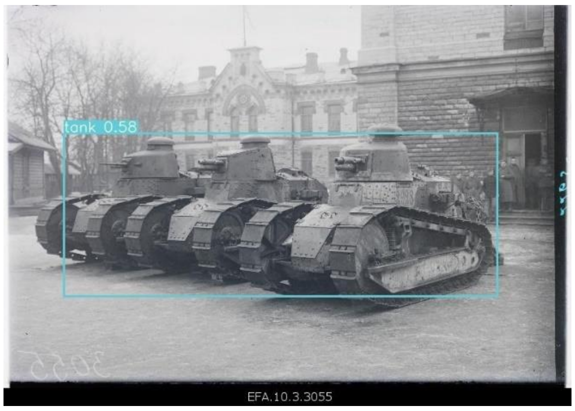


(EfficientDet D2)

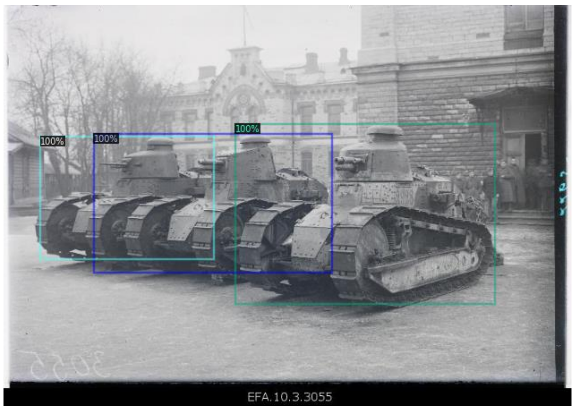


(EfficientDet D0)

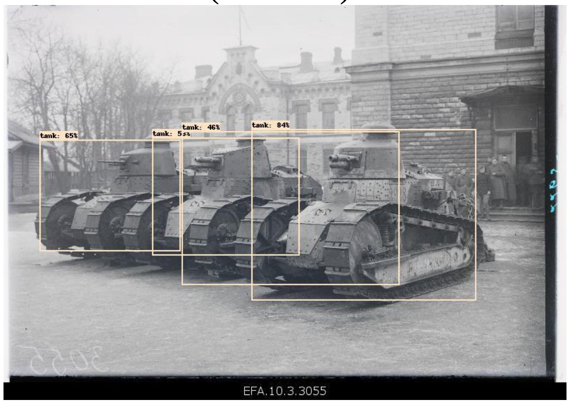
Figure D. Church and bridge detection examples, 10 classes 100 images model



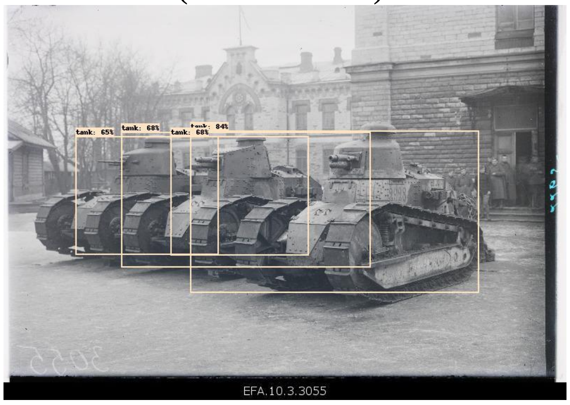
(YOLOv3)



(Faster R-CNN)

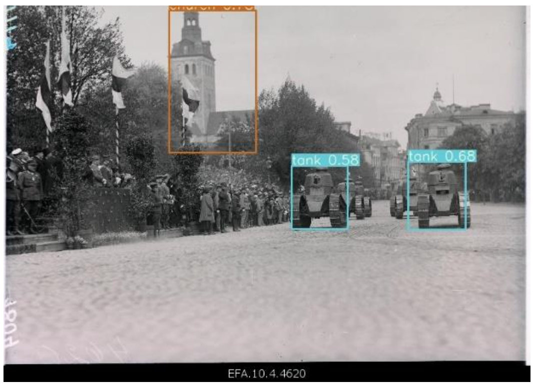


(EfficientDet D2)

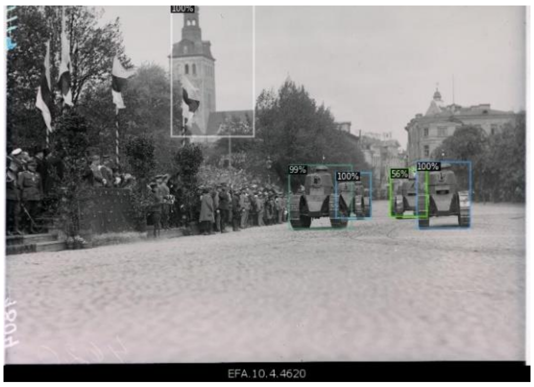


(EfficientDet D0)

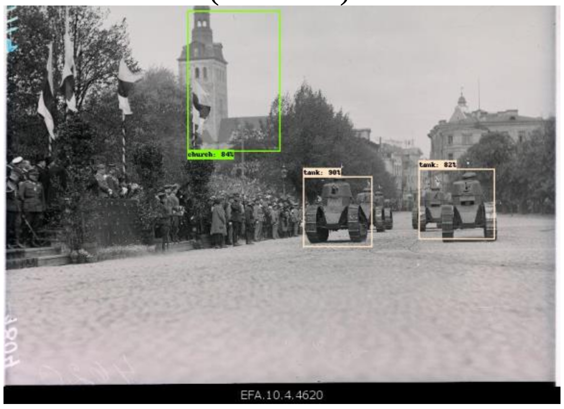
Figure E. Partially successful tank detection examples, 10 classes 100 images model



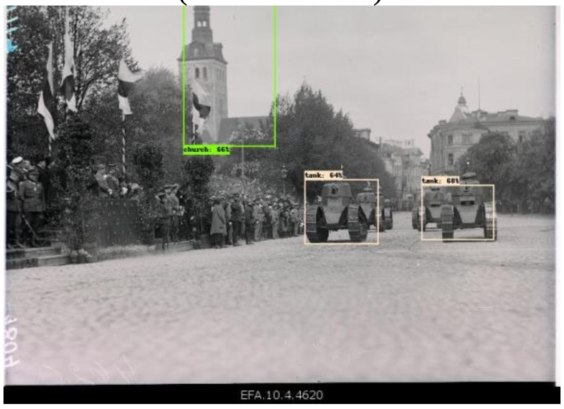
(YOLOv3)



(Faster R-CNN)

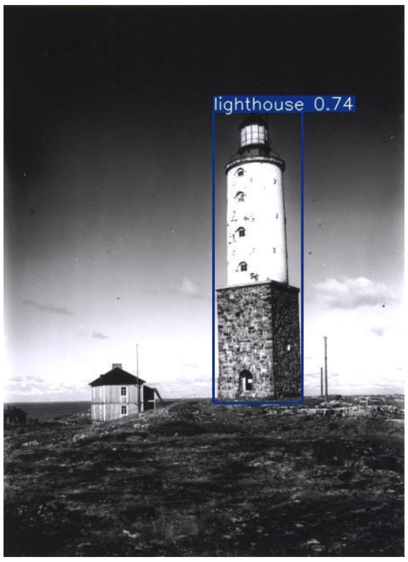


(EfficientDet D2)

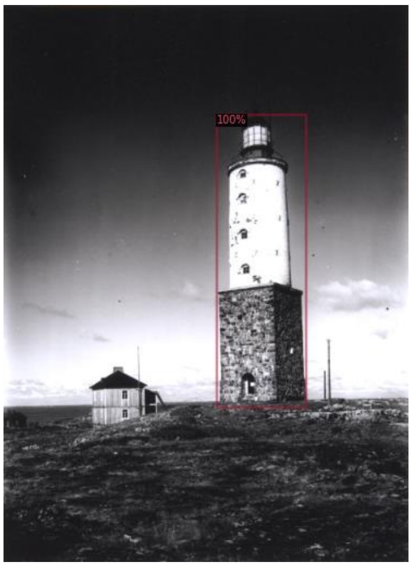


(EfficientDet D0)

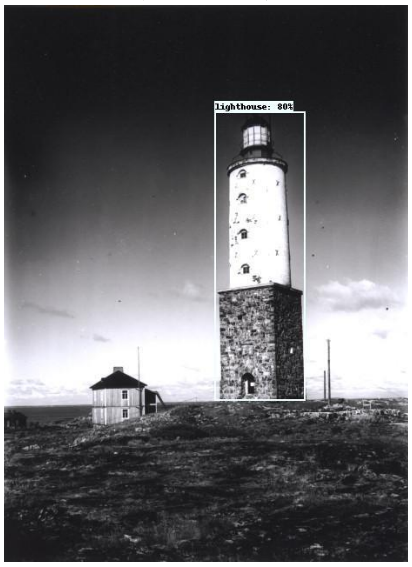
Figure F. Tank and church detection examples, 10 classes 100 images model



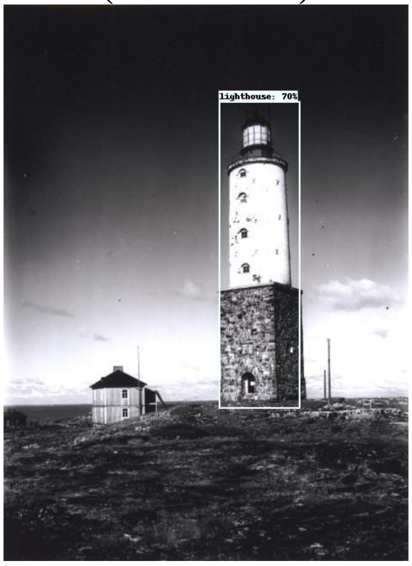
(YOLOv3)



(Faster R-CNN)

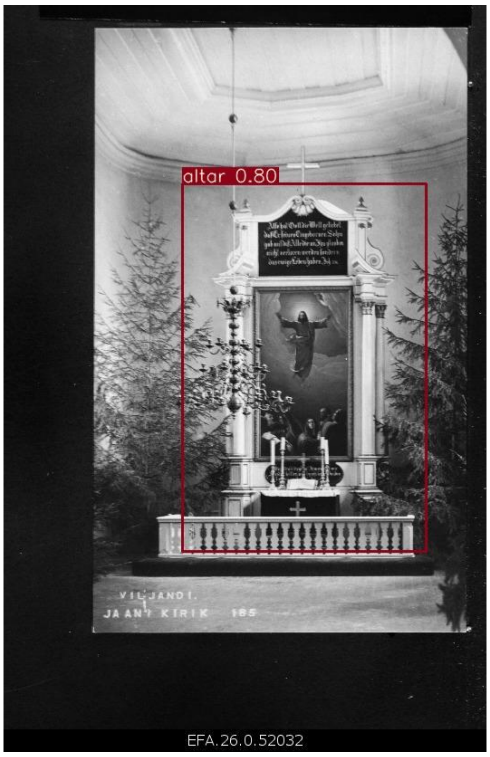


(EfficientDet D2)

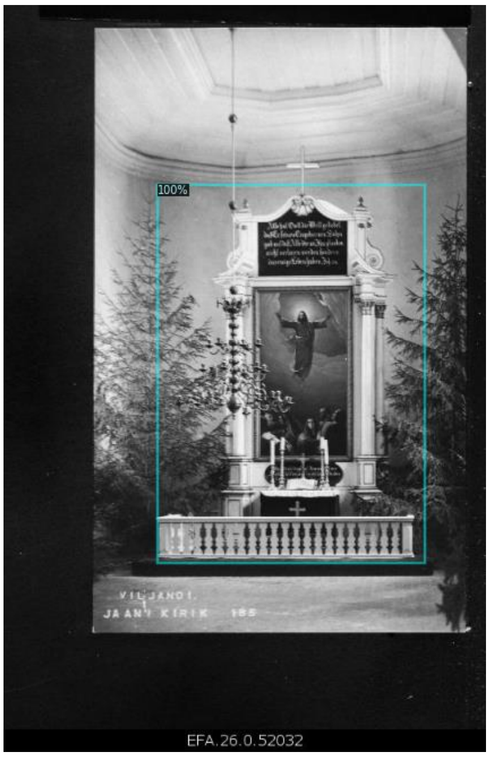


(EfficientDet D0)

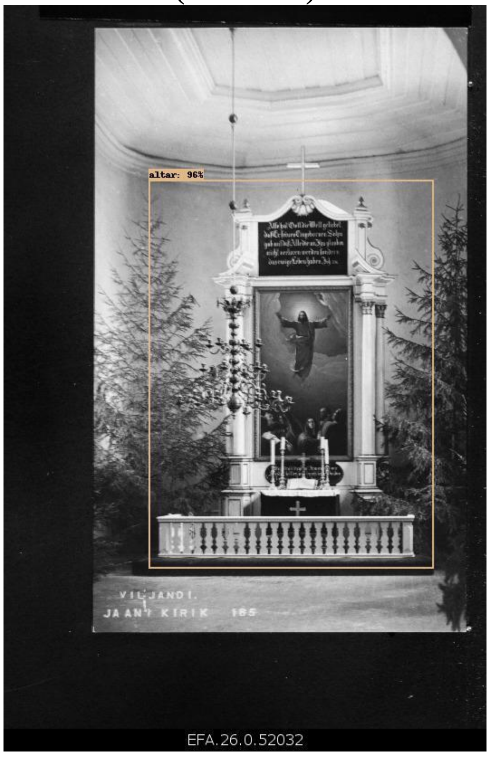
Figure G. Lighthouse detection examples, 10 classes 100 images model



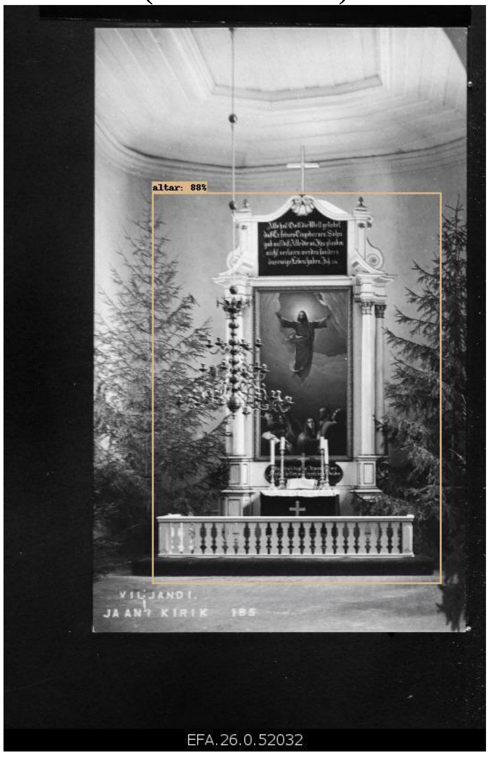
(YOLOv3)



(Faster R-CNN)

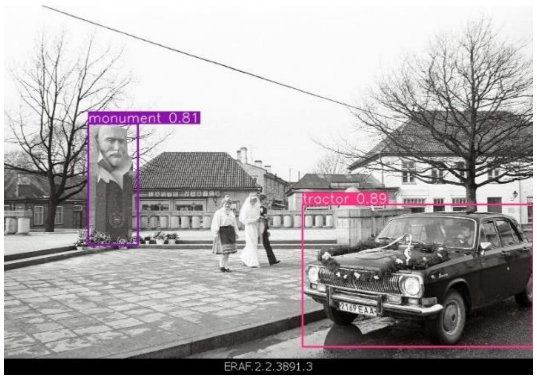


(EfficientDet D2)

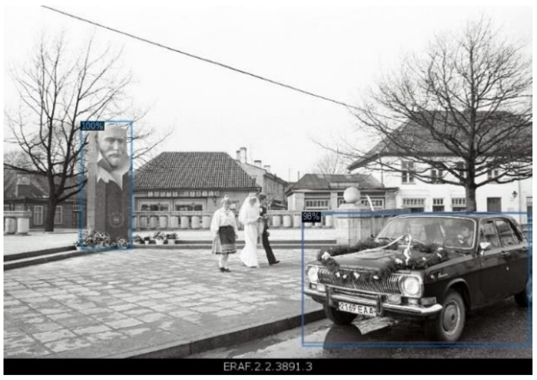


(EfficientDet D0)

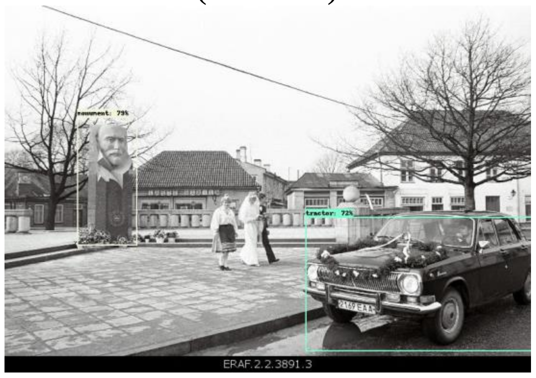
Figure H. Altar detection examples, 10 classes 100 images model



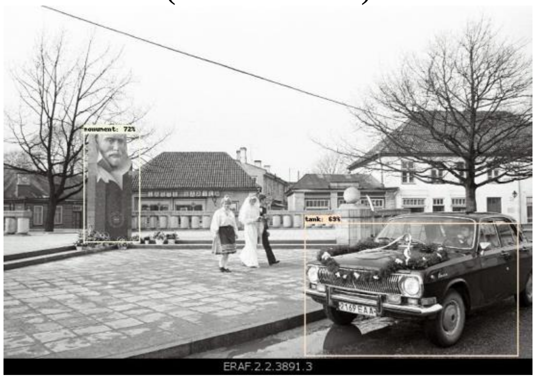
(YOLOv3)



(Faster R-CNN)



(EfficientDet D2)



(EfficientDet D0)

Figure I. Successful monument, faulty tractor/tank detection example, 10 classes 100 images model

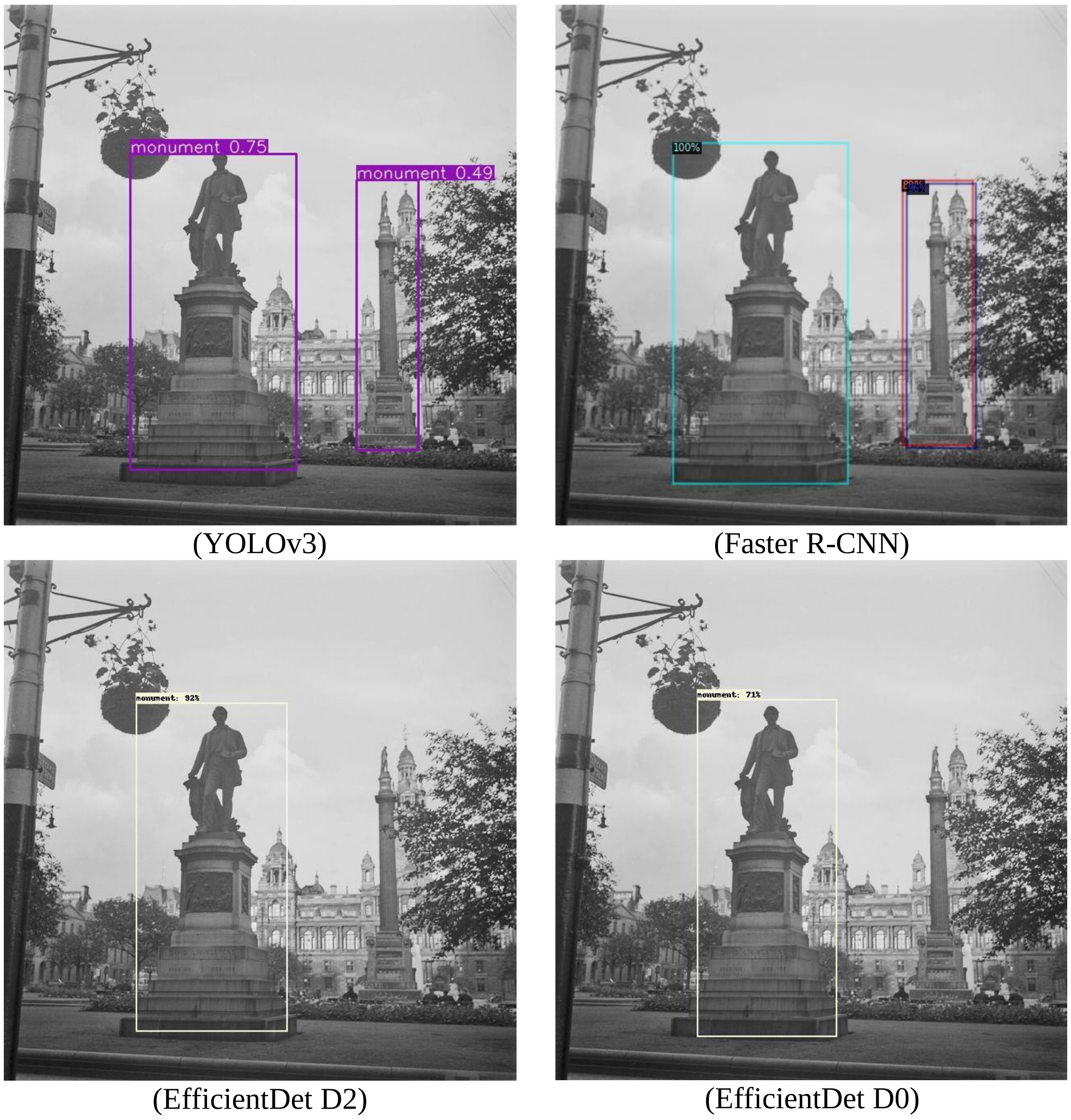
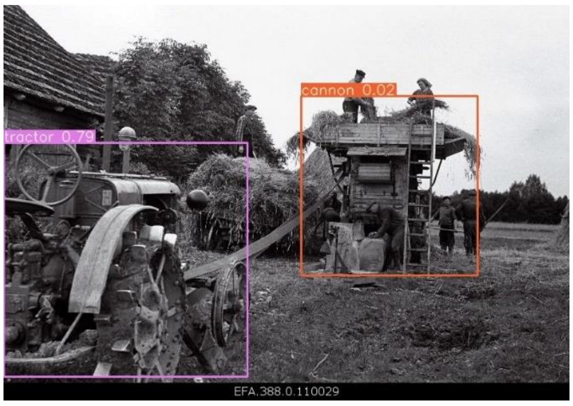
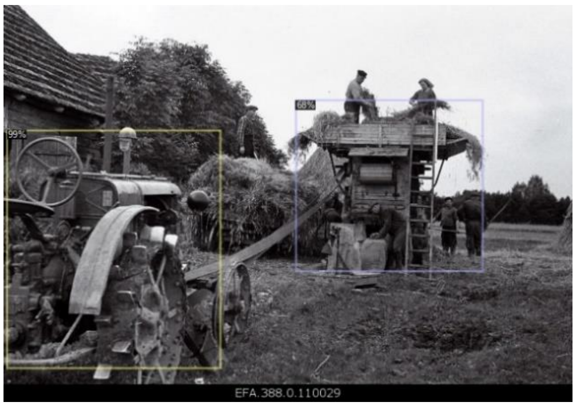


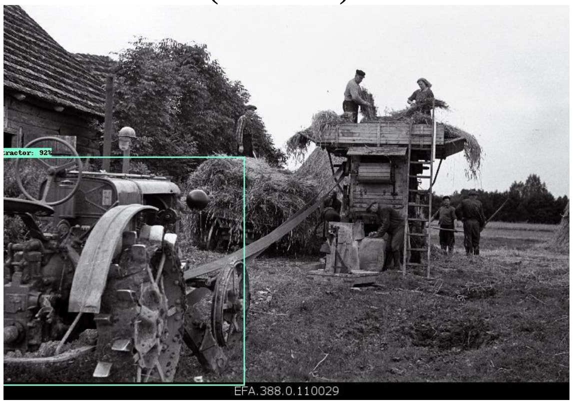
Figure J. Monument detection example, 10 classes 100 images



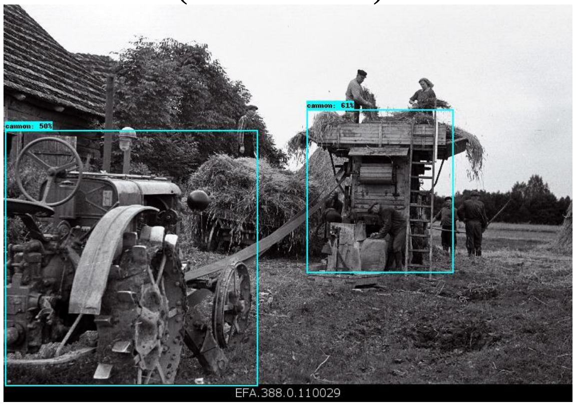
(YOLOv3)



(Faster R-CNN)

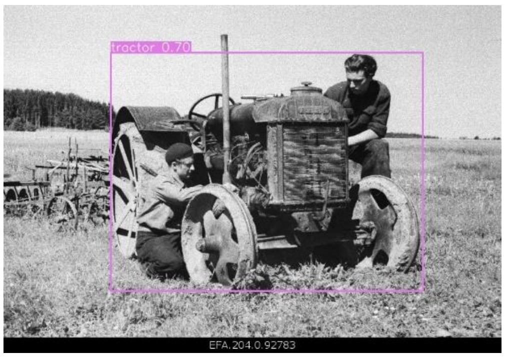


(EfficientDet D2)

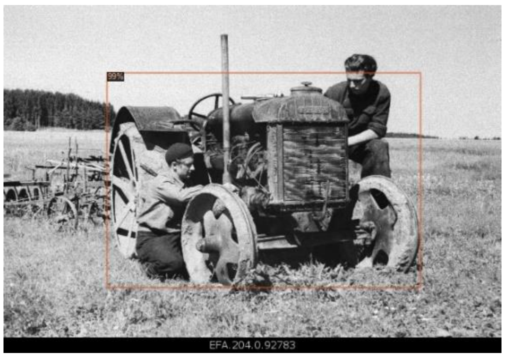


(EfficientDet D0)

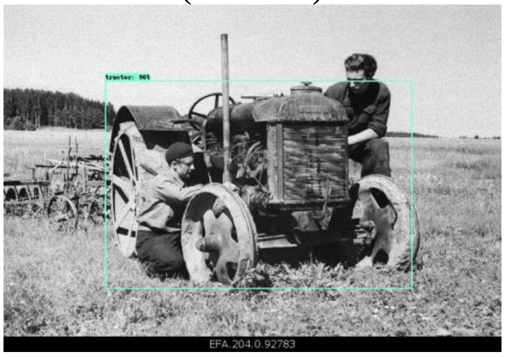
Figure K. Partially correctly labelled tractor, falsely labelled cannon detection example, 10 classes 100 images model



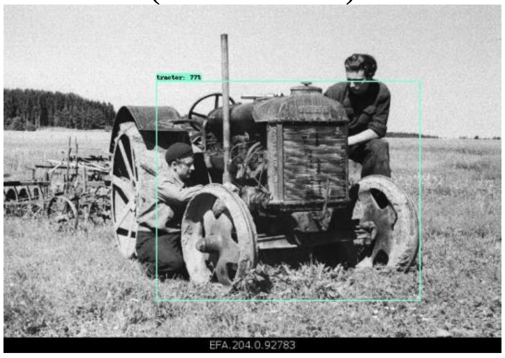
(YOLOv3)



(Faster R-CNN)

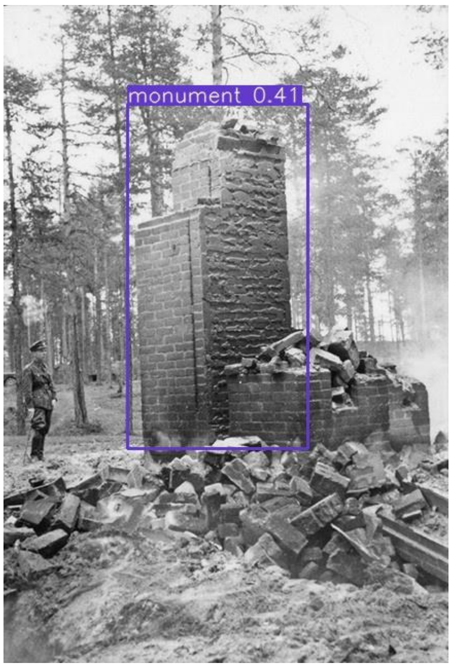


(EfficientDet D2)

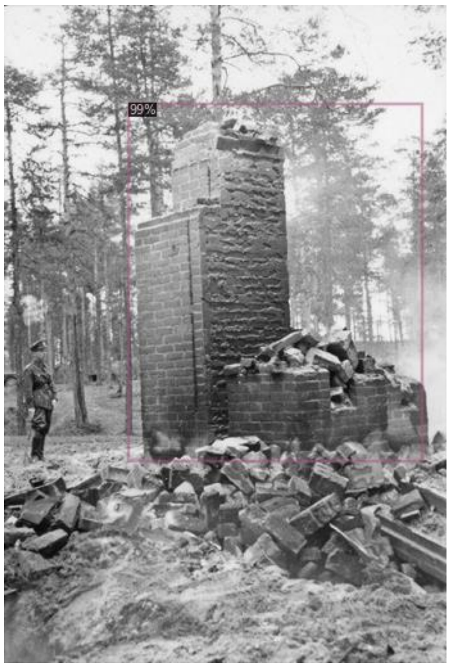


(EfficientDet D0)

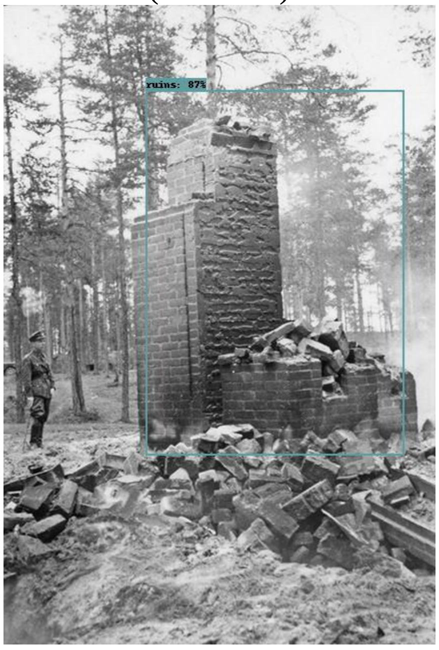
Figure L. Tractor detection example, 10 classes 100 images model



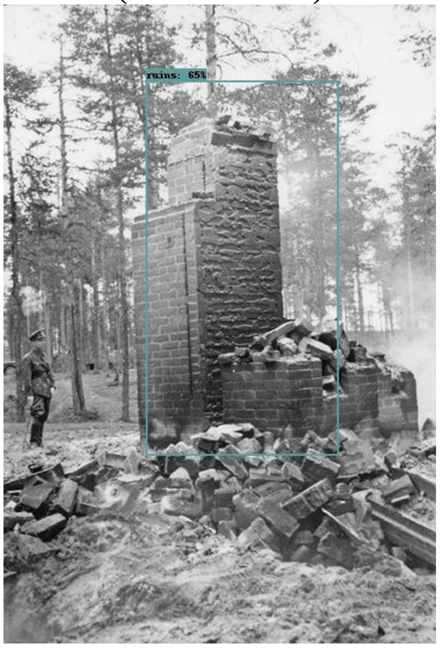
(YOLOv3)



(Faster R-CNN)

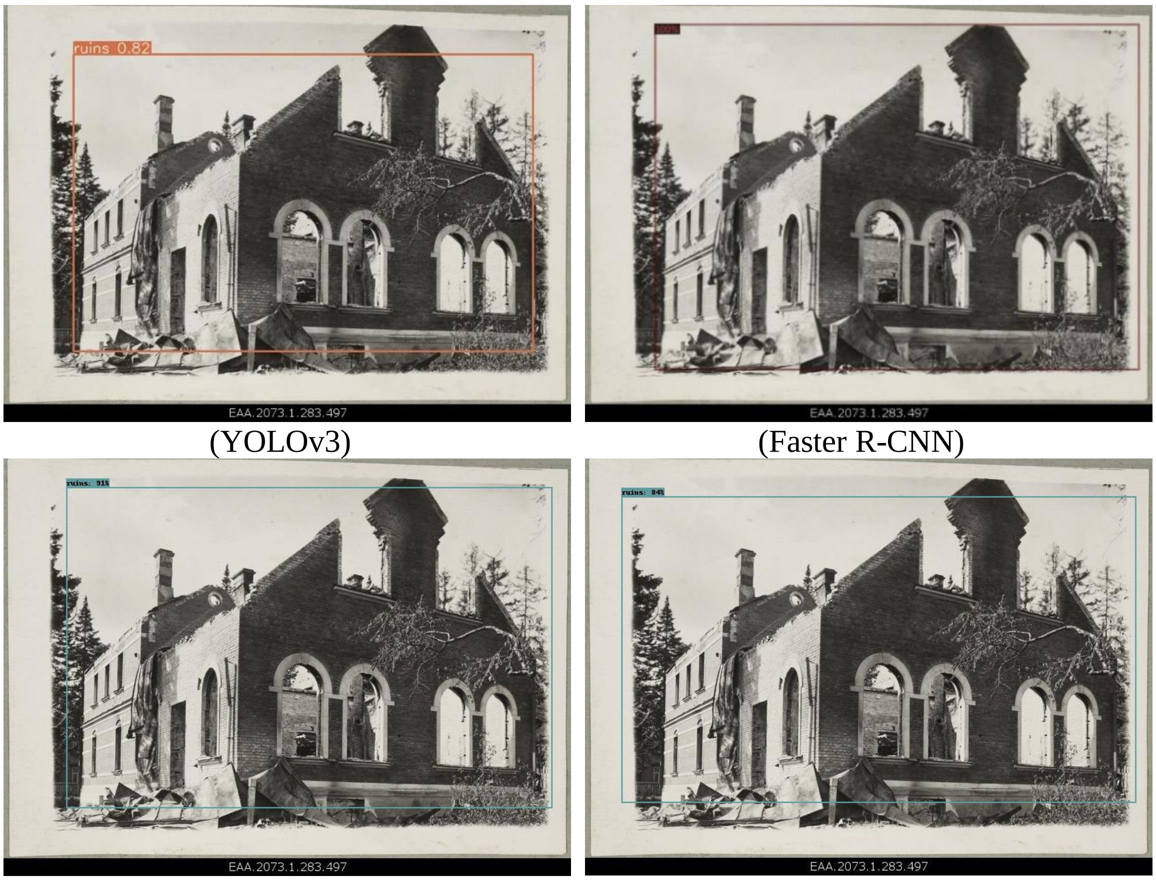


(EfficientDet D2)



(EfficientDet D0)

Figure M. Partially correctly ruins detection example, 10 classes 100 images model



(YOLOv3) (Faster R-CNN)

(EfficientDet D2) (EfficientDet D0)

Figure N. Ruins detection example, 10 classes 100 images model