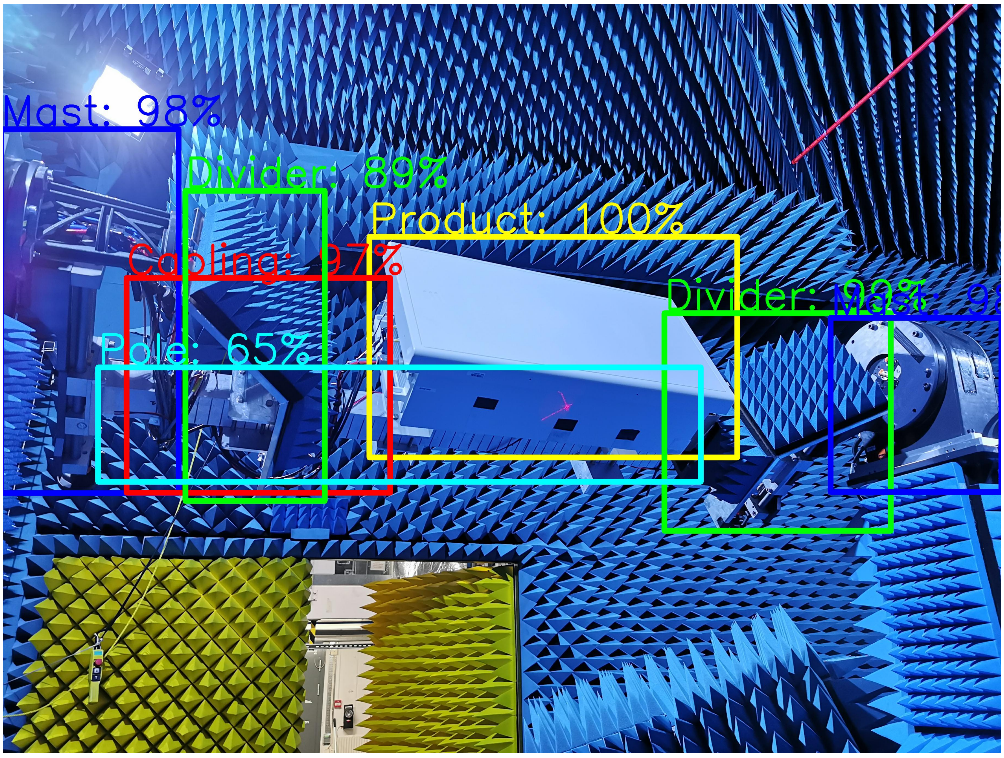
CHALMERS
UNIVERSITY OF TECHNOLOGY



Interference Object Detection using TensorFlow Lite and Transfer Learning for Android Devices

An Application for Detecting Points of Interference in an Anechoic Testing Chamber Leveraging the SSD Network

Master's thesis in Complex Adaptive Systems

KASPER HALL, NOEL HALL

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

MASTER'S THESIS 2022

Interference Object Detection using TensorFlow Lite and Transfer Learning for Android Devices

An Application for Detecting Points of Interference in an Anechoic Testing Chamber Leveraging the SSD Network

KASPER HALL, NOEL HALL

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022

Interference Object Detection using TensorFlow Lite and Transfer Learning for Android Devices
An Application for Detecting Points of Interference in an Anechoic Testing Chamber Leveraging the SSD Network
KASPER HALL, NOEL HALL

Supervisors: Christian Heina, Ericsson and Giovanni Volpe, Department of Physics
Examiner: Giovanni Volpe, Department of Physics

Master's Thesis 2022
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Detection result obtained by running inference on a model with image from test set.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Interference Object Detection using TensorFlow Lite and Transfer Learning for Android Devices
An Application for Detecting Points of Interference in an Anechoic Testing Chamber Leveraging the SSD Network
KASPER HALL, NOEL HALL
Department of Physics
Chalmers University of Technology

Abstract

With the rapid evolution of machine learning and artificial intelligence faster and more robust network architectures are developed. This is possible due to the increase in computational power, improved algorithms and the creation of large scale annotated datasets. Re-purposing these state of the art networks using transfer learning allows for customized models to be created and applied to niche problems. In this paper, we create an object detection application able to detect interference points in anechoic testing chambers. The application runs detection on a mobile device using networks created with TensorFlow Lite. Utilizing the detection result the application can give advice on how to improve the installation in the testing chamber and can thus enforce a baseline for how installations are conducted increasing the repeatability of tests. The end product is an android application running on a mobile device able to detect interference points in 13 FPS for two different testing chambers. The two object detection networks used achieved a mean average precision score of 0.8765 and 0.8650 and a average recall score of 0.7212 and 0.6997 respectively.

Keywords: Android Studio, Anechoic Chamber, Machine Learning, Object Detection, Single Shot Detector, TensorFlow, TensorFlow Lite, Transfer Learning

Sammandrag

Med stora framsteg inom maskininlärning och artificiell intelligens kan snabbare och robustare nätverk skapas. Detta möjliggörs av bättre datorer, förbättrade algoritmer och uppkomsten av stora annoterade dataset. Med hjälp av transfer learning kan dessa moderna nätverk återanvändas till anpassningsbara modeller för nischade problem. I rapporten skapas och används en object detection applikation för att upptäcka störningsmoment i installationer i en ekofri testkammare. Mobilapplikationen använder sig av nätverk skapade med hjälp av TensorFlow Lite för att hitta störningsmoment. Genom att använda resultaten från nätverken kan applikationen rekommendera förbättringar för installationen. Med detta kan en minimal standard för installationer införas vilket ökar repeterbarheten för testen. Den slutgiltiga applikationen fungerar för android enheter och kan upptäcka störningspunkter i 13 FPS för två olika testkammare. De två nätverken som användes uppnår ett mean average precision värde på 0.8765 and 0.8650 och ett average recall värde på 0.7212 och 0.6997.

Acknowledgements

We would like to express our gratitude to our supervisors Giovanni Volpe at Chalmers University of Technology and Christian Heina at Ericsson for their excellent guidance. Further, thanks to Ericsson for giving us access to their testing chambers and allowing us to collect data for our datasets. Without this support the project would not be possible.

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
AP	Average Precision
AR	Average Recall
AUC	Area Under the Curve
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
EUT	Equipment Under Test
FN	False Negative
FP	False Positive
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
mAP	Mean Average Precision
ML	Machine Learning
R-CNN	Region Based Convolutional Neural Network
ReLU	Rectified Linear Unit
R-FCN	Region Based Fully Convolutional Networks
SSD	Single Shot Detector
TP	True Positive
TPU	Tensorflow Processing Units
VGG	Visual Geometry Group
YOLO	You Only Look Once

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

The field of machine learning (ML) and artificial intelligence (AI) keeps on evolving and new use cases are discovered every year. One sub-field which has advanced a lot in the last decade is object detection, this is the practise of detecting semantic objects in images and video feeds. This evolution comes primarily from the increase in computational power, improved algorithms and creation of large scale annotated datasets [6]. Object detection is versatile and can be used in many areas such as self-driving cars, video surveillance and medical diagnoses.

Training a network to perform object detection is a time consuming task which require a lot of data. To bypass these drawbacks state of the art networks pre-trained on huge datasets can be re-purposed using a technique called transfer learning. The idea behind transfer learning is to avoid training an entire network. Instead, by keeping the feature selection of the pre-trained network and only teaching it the new mappings between features and objects the time and data needed for training a high quality object detector is greatly reduced. Evaluating the performance of the object detector is commonly done using the metrics precision, recall, mean average precision (mAP) and the speed at which it can perform predictions.

Many state of the art object detection networks maximize performance on the expense of speed. This means that they are designed for producing high quality prediction in a reasonable time on a graphics card. These types of networks are not fit to run on mobile devices which lack the computational power needed, and especially not as a real time application. Fortunately, there are networks designed exactly with this use case in mind. These networks in combination with libraries designed for deploying machine learning solutions allow for implementing robust models trained on self made datasets without having to create and train a model from scratch. This, as mentioned before, greatly reduces the implementation time and is what allows this project to be feasible in a limited time frame.

1.2 Problem Statement

When developing radio equipment, testing is a crucial step to determine the performance of a product. Faulty tests may lead to working products being labeled as defective or the potential of a product being severely underestimated. During testing, the strength of a directed beam from a product is measured and to get a truthful measurement all interference has to be minimized. The main source of interference comes from signals reflecting off surfaces such as walls and equipment. To combat this problem the evaluation of a product is performed in an anechoic chamber [7]. This helps reduce the amount of interference coming from the walls, however, signals reflected off the equipment can still be a major source of interference.

It is not evident by looking at a product rig how much the installation affects the measured result. The effect of problem areas such as hanging wires, the mast, the device, etc. on the result is unpredictable. With a lot of variables affecting the results it is hard to measure the performance of the product and get a reliable statistic of what it is capable of. Additionally, changing the installation of these components changes the entire mounting. With countless number of potential ways to install a component and with no common standard for installations repeatability of a test becomes near impossible.

The task of detecting problem areas is challenging. For the detection results to be useful it is crucial that no object is unnoticed or misclassified. A misclassified object may prove detrimental as the dynamic of the image changes. For example, a wiring job may be considered well executed if it align with the pole. However, if the pole is not detected then the drawn conclusion of the image may be that the wiring is bad. To avoid this a consistent model is needed. Additionally, since the application is to run on a mobile device the network size needs to be small and the inference time low.

1.2.1 Scope

A number of goals have been set to reach the desired application. The final application is expected to be capable of detecting points of interference as well as relay relevant information depending on the detection.

- Achieve high quality custom datasets containing images from the ancehoic chambers at Ericsson.
- Obtain models which perform well on test data to run inference in real time.
- Apply the models in an application to locate and suggest actions to minimize interference sources using the camera feed of a mobile device.
- Evaluate performance of the end product in the measurement facility using several interference scenarios.

In the process of reaching these goals a set of challenges has been recognized.

- Research and evaluate different object detection models to find suitable networks for the application.
- Gather and annotate images to create datasets which can be used in conjunction with transfer learning to train the networks.
- Design a reliable method for using object detection anchor boxes to discover

possible points of interference.
- Create a method for using points of interference to propose possible actions that can be taken to improve the installation.
- Build a mobile application combining the developed functionality into an easy to use format.

By meeting these requirements the application is able to detect and recommend fixes to resolve mounting mistakes found in the anechoic testing chambers. This application can enforce a minimum baseline for how installations of antenna systems are carried out in the anechoic testing chambers. This will enforce a standard for how installations are done and will reduce variation in results between tests increasing the repeatability of the tests.

1.3 Thesis Outline

The report is divided into six chapters, introduction, theory, method, result, discussion and conclusion. The first chapter, introduction, introduces the subject of object detection, machine learning and the problem statement which the paper aims to solve. The problem statement is followed by a scope where the goals of the project is discussed and challenges are recognized. Lastly, an array of related works from different fields are presented.

The second chapter, theory, aims to provide relevant background information about different techniques, tools and programs used in the paper. This knowledge enables the reader to follow the steps taken in the method section and also to follow the discussions and conclusions of the paper.

The third chapter, method, describes the path taken and the decisions made when achieving the goals from section 1.2.1. The chapter also explains how the different challenges were faced, using techniques introduced in the theory chapter.

The fourth chapter, results, presents the results gathered from the training and evaluation of different networks, the final application and the impact of poor installations.

The fifth chapter, discussion, discusses the results presented in the previous chapter and relates them to the goals of the project described in section 1.2.1.

The last chapter, conclusion, summarizes the outcome of the project and evaluates its success. Additionally potential future work is presented and discussed.

1.4 Related Works

Object detection is a powerful tool that has been applied successfully to solve problems within a multitude of fields. Below a set of papers displaying the possibilities of object detection is presented alongside a paper discussing how to create a high quality dataset.

1.4.1 Object Detection in Medicine

Research has been done previously in implementing object detection to aid in medical diagnoses. In the paper *Detecting and classifying lesions in mammograms with Deep Learning* Ribli et al. [8] proposed a network modeled using the faster R-CNN architecture VGG16 to detect lesions in mammograms and determine whether they are malignant or benign. Their solution achieves an Area Under the Curve (AUC) score of 0.85 on their validation set and a 90% accuracy on the INbreast dataset with only 0.3 false positive marks per image.

In *Applying faster r-cnn for object detection on malaria images* Hung and Carpenter [9] propounded an method for detecting malaria in microscopy images of malaria-infected blood. The network is a combination of two steps, a faster R-CNN network followed by an AlexNet both pretrained on the ImageNet dataset [10]. Using only the first step the obtained accuracy is 59% and using the whole model this increased to 98%. Comparing their method to an human annotator they found that the network far surpassed human annotations which achieved an accuracy of 72%.

1.4.2 Object Detection in Self-Driving Cars

In the paper *Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions* Simhambhatla et al. [11] applies three networks, Single Shot Detector (SSD), Region Based Convolutional Neural Networks (R-CNN), Region Based Fully Convolutional Networks (R-FCN) for the task of detecting traffic objects in varying weather conditions, *Day, Night, Rainy* and *Snowy*. The study is conducted in two steps, the first step assesses the pre-trained version of the networks to find the best network for the task. The second step studies the effects of applying transfer learning to the best performing network. The results conclude that the R-CNN is the best pre-trained network. Further, the custom network re-trained using transfer learning achieves better mAP score in two of the weather conditions, comparable in one and worse in one. Also, using the custom network reduces inference times by 3 seconds per image which is a 85% speed increase.

1.4.3 Object Detection in Mobile Devices

As previously mentioned there are differences which has to be considered when running object detection on a mobile device instead of on a computer. One such difference is the amount of computational power available in the two devices. In the paper *Mobile Object Detection using TensorFlow Lite and Transfer Learning* Alsing [12] employ and compare TensorFlow models pre-trained for usage in mobile devices for the detection of post-it notes. The models are downloaded from the TensorFlow model zoo and fine tuned for the desired task using transfer learning. Solving the problem using transfer learning saves a lot of time and lowers the need of data. The resulting networks are able to find post-it notes with over 90% accuracy for models that run inference with a frame rate of around 2 frames per second (FPS).

1.4.4 Dataset Creation

There is no universal standard present when creating a dataset. However, Gebru et al. in the paper *Datasheets for datasets* [13] proposes a datasheet used for describing the characteristics, usage and general information of a dataset. Adding this information allows its users to make informed decisions when working with the dataset thus avoiding discriminatory biases. This idea has since then been adopted both by other academic researchers and companies.

2

Theory

2.1 Object Detection

Object detection is a field within ML where computer vision is used to identify and locate objects in an image or video. The identification allows for labeling the different objects in the scene while the localization allows for determining the locations of the objects. In object detection each object of interest is located and identified, this is what separates object detection from image classification. In standard classification the scene is labeled and not every object in the scene [14]. This difference is displayed in Figure 2.1.

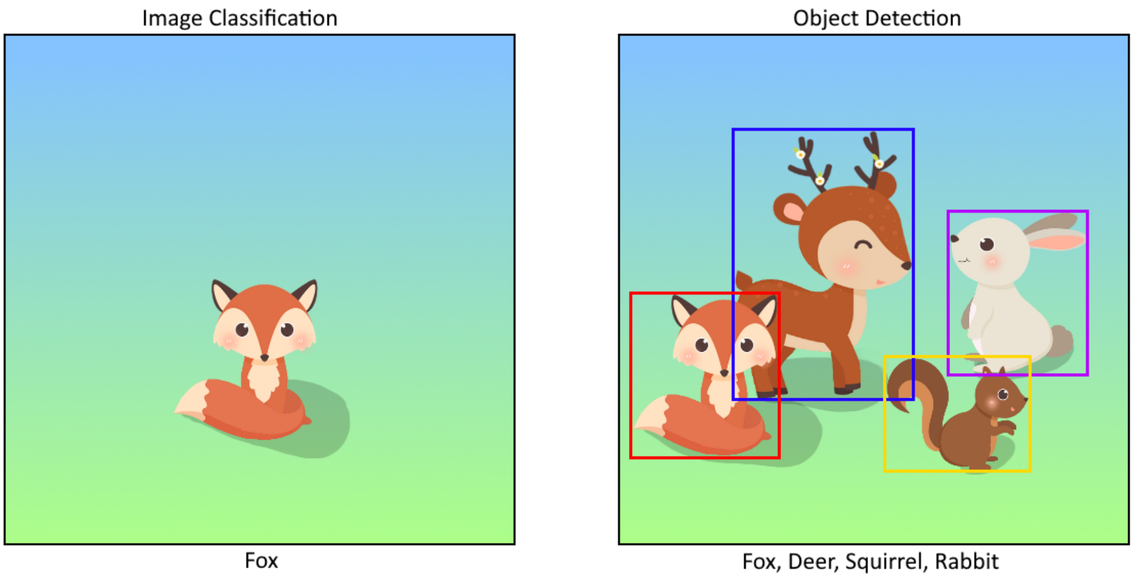


Figure 2.1: Figure displaying the difference between object detection and classification [1]. Classification determines a label for the entire scene while object detection determines a label and location for each object in the scene.

Object detection is commonly done according to one of two methods. Either using ML as done in conventional approaches or using deep learning (DL) as done in state of the art object detection networks.

2.1.1 Machine Learning

The ML approach to object detection is traditionally done in three steps, informative region selection, feature extraction and classification. Informative region selection is done by scanning the image using a multiscale sliding window. In feature extraction computer vision techniques such as Histogram of Oriented Gradients features [15–17] and Scale-Invariant Feature Transform [18, 19] are employed to identify pixels that belong to the same object. Lastly, a classifier is used to distinguish between the detected objects. The classification step is usually done using a support vector machine [20], AdaBoost [21] or deformable part-based model [22] [6].

2.1.2 Deep Learning

As more research in DL is done new tools capable of learning semantic, high-level, deep features are discovered. These tools help combat the problems present in conventional approaches [6]. The deep architecture of this approach allows for networks to learn more complex features than what a shallow one could. Further, the powerful training algorithms makes it possible to learn informative object representations without the need to design features manually [23].

State of the art networks such as SSD [4], You only look once (YOLO) [24] and Visual Geometry Group (VGG) [25] all use deep learning architectures. In these, a row of convolutional neural networks (CNN) are used to perform feature selection on the input image. The feature selection result is then handled by a classifier which predicts the bounding boxes and labels of each object [14].

2.1.2.1 Convolutional Neural Network

A CNN is made up of layers of feature maps. Any feature map inside the CNN is an induced multichannel image in which each pixel is a specific feature obtained from adjacent features in the previous layer [23]. Between the layers various transformations can be applied to the feature maps, such as filtering and pooling [26].

The filtering transformations convolutes a filter matrix of learnt weights with the values of adjacent neurons and applies a nonlinear function, for example sigmoid [27] or Rectified Linear Unit (ReLU) [28], to receive final responses. There are also multiple variations of the pooling transformation, such as max pooling, min pooling, average pooling, etc. which compiles the information of adjacent neurons into a single value to produce more robust feature descriptions [28] [23].

Finally, the last layer of the CNN is a fully connected layer where the neurons no longer resides within a matrix and instead are flatten out to a single row. Here, depending on the task of the network different activation functions are applied to the fully connected layer to produce the desired output.

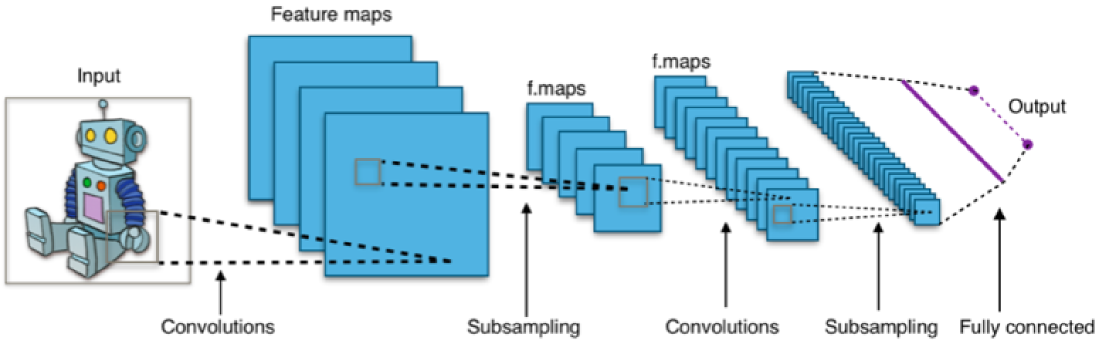


Figure 2.2: A convolutional network with multiple layers [2] [3].

2.1.2.2 Nonlinear Activation Functions

Using nonlinear activation functions allows the network to learn complex structures in the data which a network with a linear activation would not capture. One frequently used activation function is ReLU which is explained in Equation 2.1 and displayed in Figure 2.3.

$$f(x) = \begin{cases} 0 & \text{for x} < 0 \\ x & \text{for x} \geq 0 \end{cases} \tag{2.1}$$

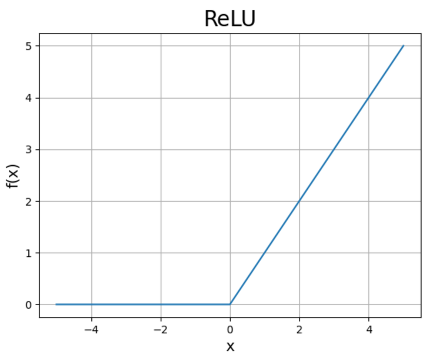


Figure 2.3: ReLU activation function.

Another commonly used activation function is sigmoid which is explained in Equation 2.2 and displayed in Figure 2.4. Similar to ReLU this is a nonlinear activation function capable of allowing a network to learn complex behaviours. Further, the sigmoid activation has some advantages over ReLU such as that the activation has a maximum value of 1 which stops the activation from blowing up. However, unlike ReLU sigmoid is more susceptible to vanishing gradients which drastically reduces the effectiveness of training on deep networks to the point that nothing is being learnt [29].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

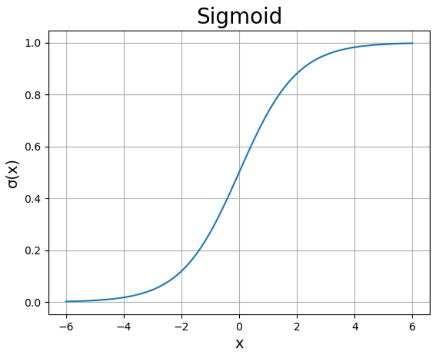


Figure 2.4: Sigmoid activation function.

Further, activation functions such as the softmax function is used to transform the output of a model into the correct output type. Softmax is commonly used in object detection networks such as the SSD. It is a nonlinear activation function used for predicting output classes. This is done by normalizing the output of the network to a probability distribution over the classes transforming an input vector of K values into a weighted vector of K values that sum to 1. This is done according to Equation 2.3.

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

Here K is the number of classes and $z_i \in \mathbb{R}$ are the elements of the input vector.

2.1.2.3 Pooling

Pooling layers are used to reduce the size of feature maps. This is done by performing a operation on a specific area of the feature maps and shifting the location of the area until the entire feature map has been covered. The type of operation used depends on which pooling layer is applied, a comparison on the results of applying different pooling layers can be seen in Figure 2.8. Max pooling selects the largest value within the area resulting in only the most prominent light features of the previous feature map being kept. This is done according to Equation 2.4 and the result of which can be seen in Figure 2.5 [30].

$$y_{kij} = \max_{(p,q)\in R_{ij}} x_{kpq}, \tag{2.4}$$

Where y_{kij} is the resulting output for feature map k around a local neighborhood R_{ij} centered at i, j for which x_{kpq} is the element at position p, q.

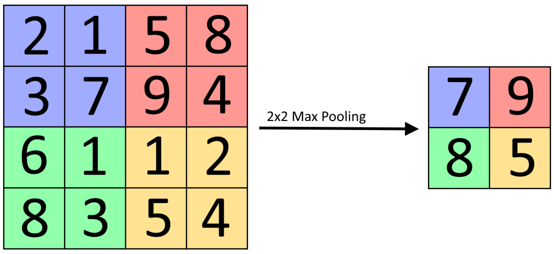


Figure 2.5: Max pooling layer with filter size of 2x2 and stride 2.

A second type of pooling is min pooling which similarly to max pooling only selects one value in a neighbourhood of pixels. The min pooling however, selects the smallest value which leads to only the most prominent dark feature of the previous feature map being kept. This is done according to Equation 2.5 and the result of which can be seen in Figure 2.6 [31].

$$y_{kij} = \min_{(p,q)\in R_{ij}} x_{kpq}, \tag{2.5}$$

Here the variables are consistent with Equation 2.4.

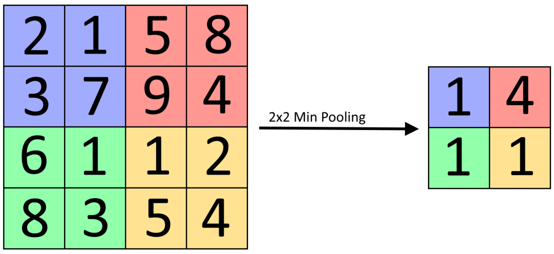


Figure 2.6: Min pooling layer with filter size of 2x2 and stride 2.

A third type of pooling is average pooling which works by taking the average of the values in an area. This results in a downsampled feature map with low translation invariance, meaning that the resulting smaller feature maps are good representations of the previous ones. This is done according to Equation 2.6 and the result of which can be seen in Figure 2.7.

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} x_{kpq}, \tag{2.6}$$

Here $|R_{ij}|$ is the area of the pooling window and remaining variables are consistent with Equation 2.4 [30].

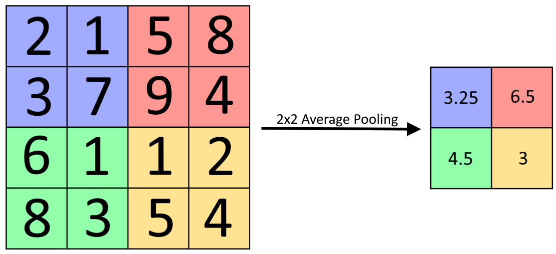


Figure 2.7: Average pooling layer with filter size of 2x2 and stride 2.

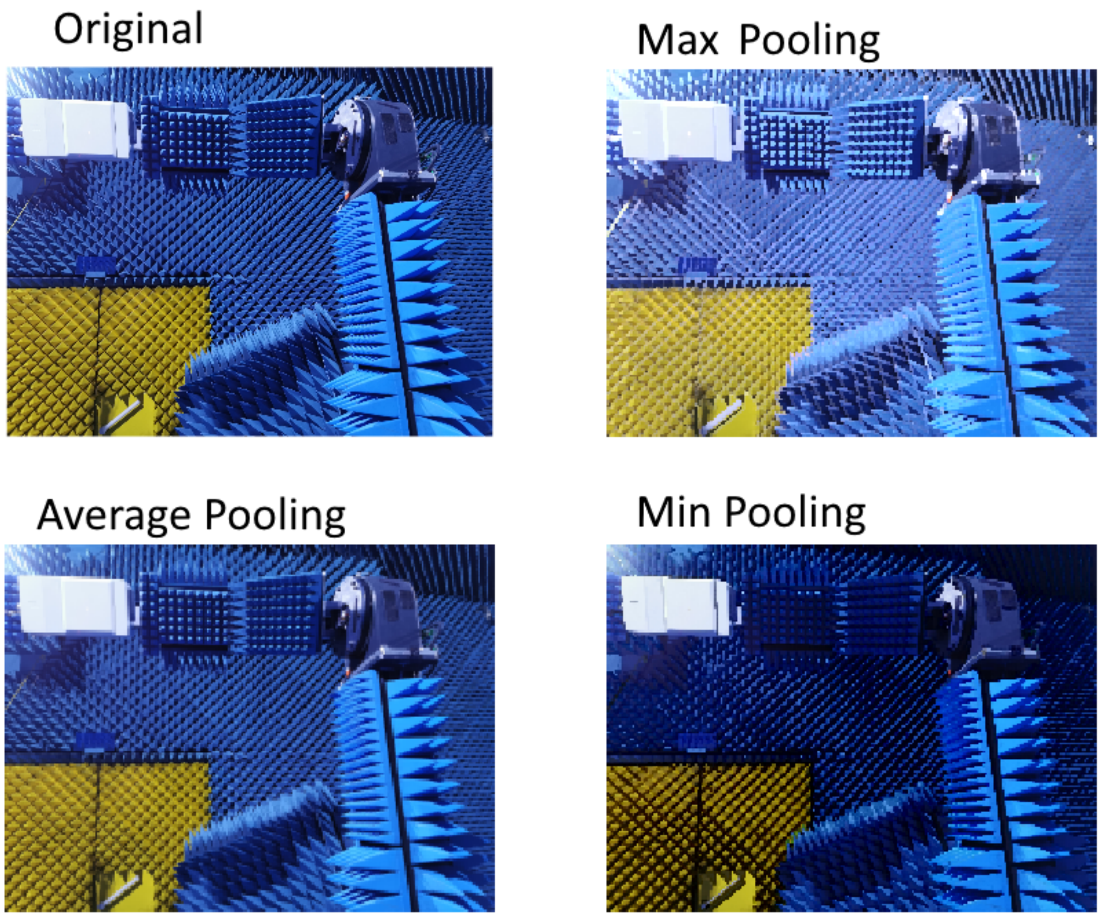


Figure 2.8: A comparison between different pooling layers on an image.

2.1.3 Training/Detection

Detecting objects in an image is done using anchor boxes. An anchor box is a prediction made by a network containing the location of the bounding box (X, Y, Height, Width) and the predicted class with its confidence score [32]. During training and inference the network creates thousands of anchor boxes for each image. To evaluate the anchor boxes a metric called Intersection over Union (IoU) is used. This metric is calculated as the Area of intersection divided by the area of union between the predicted bounding box and the ground truth, this is visually explained in Figure 2.9.

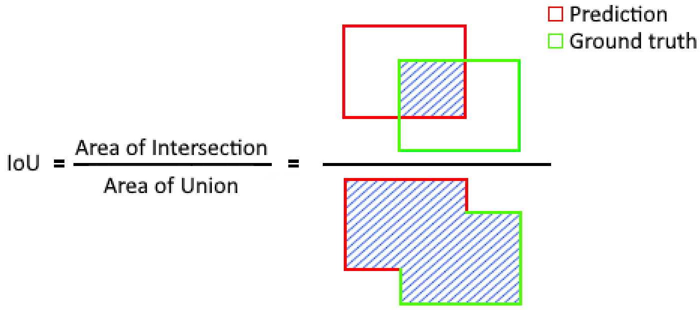


Figure 2.9: Visualization of the IoU of a predicted bounding box and the ground truth bounding box.

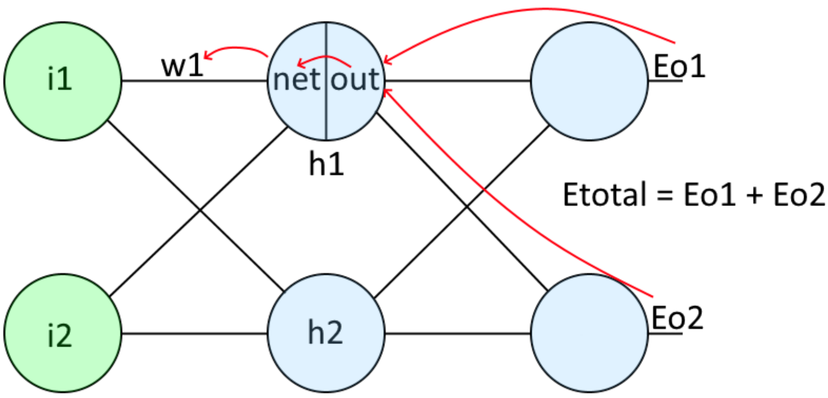
For each of the anchor boxes the IoU score determines whether or not it detects an object, if the score is larger than some set threshold, often 0.5, the object is detected. Next, to find the best anchor box for each object the IoU scores are compared and the worst boxes removed until only the box with the highest score remains. This procedure can be seen in Figure 2.10. For the network to learn how to predict accurate anchor boxes the weights of the network are changed according to the last prediction and a learning rate using a method called backpropagation. Also, it is common to use a decaying learning rate as it supports faster learning of the general properties of the data while allowing for fine tuning in the later stages of the training [33]. The goal of the training is to modify the weights of the network so that the predictions are more similar to the ground truth anchor boxes.

Figure 2.10: Anchor box predictions of two objects in an image being reduced to the final predictions. The red anchor boxes represent all boxes which has an IoU score of higher than the threshold, the blue boxes have the highest IoU score for the respective object and the black boxes are the final predictions of the different objects.

2.1.3.1 Backpropagation

Training a network using supervised learning aims to modify the weights of the network so that the outputs ($\hat{Y}$) are closer to some target outputs (Y) which exist in a training set. By updating the weights of the network the goal is to have it perform well on test data which it has never seen before. Changing the weights of the network is done as an iterative process using backpropagation. This is a technique of relating the effect a weight has on the overall output error and changing the weights accordingly. The first step to performing backpropagation is to calculate the error between $\hat{Y}$ and Y using a loss function. For example, the SSD network architecture uses a weighted sum of a confidence and localization loss to determine the error as seen in Equation 2.10. Using this error, the second step is to compute the partial derivative of the error with respect to every weight, this is done using the chain rule for partial derivatives and can be seen in Figure 2.11. The last step is to update each weight to decrease the error. The new weight value is calculated as seen in Equation 2.7 where η is the learning rate and $\frac{\delta E}{\delta W}$ is the error at weight W. Once the weights are updated a new input can be fed to the network and the backpropagation process can repeat [34].

$$W = W - \eta \frac{\delta E}{\delta W} \tag{2.7}$$



$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} * \frac{\partial E_{o2}}{\partial out_{h1}}$$

Figure 2.11: The figure displays how the error travels back through the network using backpropagation. The error travels back in steps, therefore, in order to find the derivative of the error with respect to w1 the error at h1 has to be computed first. The equations of E, out_{h1} and net_{h1} depend on which activation functions and loss functions are used.

2.1.4 Evaluation

When evaluating an machine learning models there exist a myriad of different evaluation metrics. Three of the main metrics used for evaluating the accuracy of an object detection model are precision, recall and mAP [35].

2.1.4.1 Precision

Precision is a metric which measures how accurately the model makes predictions, that is, it gives the percentage of true positive predictions amongst all positive predictions. In object detection this would refer to the number of correctly classified objects over the total number of classified objects. The precision score is calculated as seen in Equation 2.8.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.8}$$

In the equation above TP stands for true positives and FP for false positives. Deciding whether a predicted box is a TP or a FP is done using the IoU score of the predictions. If the prediction is of the correct class and has an IoU score larger than the set threshold it is a TP and if the prediction is of either the wrong class with a high enough IoU score or has too low IoU score the predictions is considered a FP [36].

2.1.4.2 Recall

Recall is a metric which measures how good the model is at finding all objects in an image. The recall score is calculated as seen in Equation 2.9.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.9}$$

In the equation above TP once again stands for true positives and FN stands for false negatives. The TP are calculated in the same way as for the precision in section 2.1.4.1 and the FN are the objects present in the image which the model failed to detect [36].

2.1.4.3 Mean Average Precision

The name mean average precision is deceptive as the metric is not the mean of the average precision scores. Instead, the mAP score is the area under the precision-recall curve averaged over all classes. Additionally, there exist a metric called average precision (AP) which is mAP but for a single class. AP is visualized for different IoU thresholds in Figure 2.12. The way that mAP and AP are used in the industry differs slightly. According to COCO there is no difference between AP and mAP while in PASCAL VOC AP is calculated for one object class while mAP is averaged over all object classes. The way they are used might also differ between object detection challenges [36].

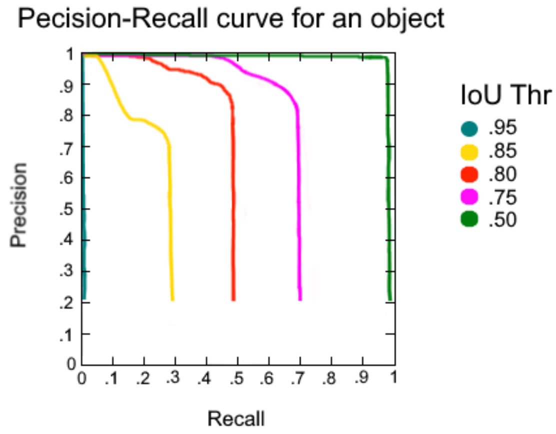


Figure 2.12: Precision-Recall curve for different IoU thresholds.

2.1.5 Single Shot Detector

The SSD is a state of the art object detection network architecture. It is a lightweight model that can detect objects with high frame rate and thus can be applied to embedded systems and run object detection in real time. The architecture is based on a feed-forward convolutional network followed by a non-maximum suppression set to produce the final detection. The early layers are based on a standard architecture used for high quality image classification called the base network. On top of the base network feature layers with different aspect ratios can be added to allow for detection on multiple scales, Figure 2.13. Multiscale detection helps with achieving a high accuracy even though a low resolution input is used, using a low resolution input further increases the speed of the network. In many networks accuracy is something which has to be sacrificed to achieve fast predictions. However, in the SSD the bounding box proposals and the subsequent pixel or feature re-sampling stage has been removed which is a common part in other state of the art networks. Removing this results in a large speedup for the SSD. In a comparison between the SSD and other state of the art networks the SSD outperforms the others both in accuracy and speed as displayed in Table 2.1 [4].

Model Name	Speed (FPS)	COCO mAP
SSD	**59**	**74.3**
Faster R-CNN VGG16	7	73.2
YOLO	45	63.4

Table 2.1: Comparison between SSD, Faster R-CNN and YOLONET [4].

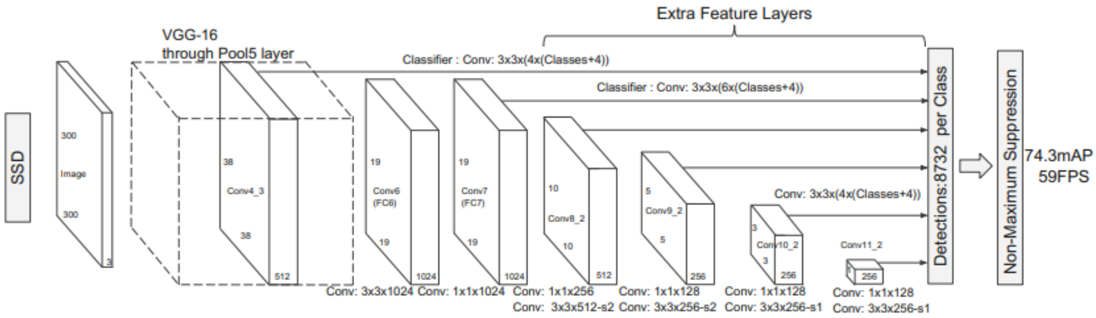


Figure 2.13: A visualization of the SSD model structure [4].

The SSD uses an image and the respective ground truth boxes for training. Multiple default boxes with different aspect ratios are evaluated at each location in several feature maps. For each of these boxes the shape offset and confidences for all classes are predicted. These default boxes are modified at prediction time to better encapsulate the different objects in the image [4].

The loss function used by the SSD is a weighted sum of a confidence loss (L_{conf}) and a localization loss (L_{loc}), this can be seen in Equation 2.10. It is important to notice that the SSD only penalizes predictions from positive matches [37]. A positive match is where the default bounding box has an overlap with the ground truth box which is higher than a set threshold (often 0.5). Lastly, the value of α is a scalar which determines how much the loss will depend on the localization part and the confidence part respectively, this parameter is set to 1 when running cross validation.

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \tag{2.10}$$

In the loss function above, L_{conf} is a softmax loss which is calculated using multiple classes confidence scores (c) and an indicator for how well the default boxes match the ground truth box (x, Equation 2.13). The confidence, softmax, loss can be seen in Equation 2.11. In the last element of the equation the confidence score of class *0* is subtracted, this is a class reserved to indicate that there are no objects in the image. The localization half of the loss function (L_{loc}) is a smooth L1 loss which is the mismatch between the ground truth box (g) and the predicted box (l) [37]. The localization, smooth L1, loss can be seen in Equation 2.12. In the equation cx and cy are used as offset to the default bounding box d which has width w and height h. Lastly, N is the number of matched default boxes.

$$\begin{aligned} L_{conf}(x,c) &= -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg}^{N} log(\hat{c}_i^0) \\ \text{where } \hat{c}_i^p &= \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \end{aligned} \tag{2.11}$$

$$
\begin{aligned}
&L_{loc}(x,l,g)) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k}\text{smooth}_{L1}(l_i^m - \hat{g}_j^m) \\
&\hat{g}_j^{cx} = \frac{g_j^{cx} - d_i^{cx}}{d_i^w} \qquad \hat{g}_j^{cy} = \frac{g_j^{cy} - d_i^{cy}}{d_i^h} \\
&\hat{g}_j^{w} = log\Big(\frac{g_j^w}{d_i^w}\Big) \qquad \hat{g}_j^{h} = log\Big(\frac{g_j^h}{d_i^h}\Big)
\end{aligned} \tag{2.12}
$$

$$
x_{ij}^p = \begin{cases} 1 & \text{if } IoU > 0.5 \text{ between default box } i \text{ and ground truth box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases} \tag{2.13}
$$

2.2 Transfer Learning

Training a CNN require a lot of data to achieve good results. However, more often than not large-scale datasets does not exist for niche ML problems. Additionally, training a network from scratch on a large dataset requires a lot of time. Commonly used datasets in object detection and image classification such as the COCO dataset and ImageNet contain $\sim 328k$ and $\sim 14.2m$ images respectively. Training a network on a dataset of this size will take weeks to finish [38] [12]. An object detection model can be split into two main parts, feature extraction which consists of a series of convolutional layers and classification which consists of fully connected layers as seen in Figure 2.14. A fully trained network is able to both extract important features from an image and use these features to identify objects in an image. To reduce the time spent training a network the feature selection of a pre-trained network can be reused while the mappings between features and objects (the classification) are retrained. This requires only a fraction of the data as the majority of the models does not have to be changed. This works since state of the art networks are proficient at finding all features in an image even without knowing what objects the different features correspond to. By freezing the feature selection layers and re-training the classification the models can be trained in a fraction of the time, with a fraction of the data and still obtain great results.

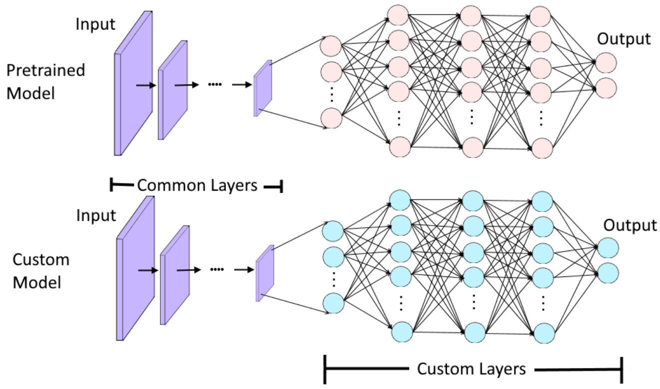


Figure 2.14: Transfer learning on a pre-trained model using a new fully connected final layer.

2.3 Dataset

A dataset is a collection of labeled data consisting of inputs and expected outputs. Both the input and expected output can have varying types depending on what task the dataset is created for. For example, a dataset created for object detection has images as inputs and anchor boxes as expected outputs while one created for natural language processing uses text strings for both input and expected output. The characteristics of a dataset is highly influential towards the behaviour of a model as a model can not be expected to perform well in an environment not covered by the dataset. Further, the performance of a model will reflect any biases present in the dataset [13]. One such bias occurs when a non-balanced dataset is used as the network can achieve good results even when disregarding rare classes.

2.4 Data Augmentation

Data augmentation is the practice of applying filters and transformation to the data to increase the size and quality of the dataset. This is an effective way to combat overfitting, especially when working with limited data. Many augmentation methods are used to teach the model to generalize. It works under the assumption that an augmented image is able to provide information not accessible from the original image [39]. There are a multitude of different data augmentation techniques, some of which are displayed in Figure 2.15.

2.4.1 Geometric Augmentation

2.4.1.1 Translation

Adding translation to an image results in shifting it either left, right, up or down. This augmentation type is especially useful to remove a positional bias which if not removed would result in the trained network predictions to only be accurate on centered data [39]. Such bias occurs when the majority of training data is centered which is often seen in for example face recognition datasets.

2.4.1.2 Rotation

Applying rotation to the image can be helpful to remove a rotational bias, say all images in the training set captures the object horizontally but in reality it might be viewed from a skewed angle. Then adding small changes in the angle of the training input allows the network to also perform well when the angle of the object differs [40]. As is seen in Figure 2.15 rotation causes fields not covered by the image to appear, these are often set to either a constant value between $[0, 255]$ or filled with random or Gaussian noise [39].

2.4.1.3 Flipping

Flipping the input image is a common and simple data augmentation. It is often used as a method for increasing the size of the dataset and the flip is generally done in the horizontal direction [41].

2.4.2 Color Augmentation

Altering the brightness, contrast and saturation of the images in the training set can be useful when the conditions in which the input images are taken varies [39]. Changing the brightness can be used to simulate varying levels of lighting and since different cameras and lenses provides unique levels of contrast and saturation augmenting these will allow the network to predict for images captured with different devices.

In addition to this, color augmentation can be done by removing the color from the input data, setting it to grayscale. Transforming the images to grayscale removes excess information while keeping important features such as shape, edges and texture making the training more efficient [42]. Additionally, research has shown that the use of grayscaled images can result in similar accuracy at a lower computational cost than colored images [42].

2.4.3 Random Erasing

In addition to the augmentation methods mentioned so far which all have been elementary there exist more complex techniques such as random erasing. This works by selecting a rectangle at a random location with random size and setting its content to either a constant value between $[0, 255]$ or replace it with noise. This augmentation type is used to artificially create occlusion which will make the model robust when handling objects that are partially covered as it is forced to recognize objects from a smaller set of visible characteristics [43].

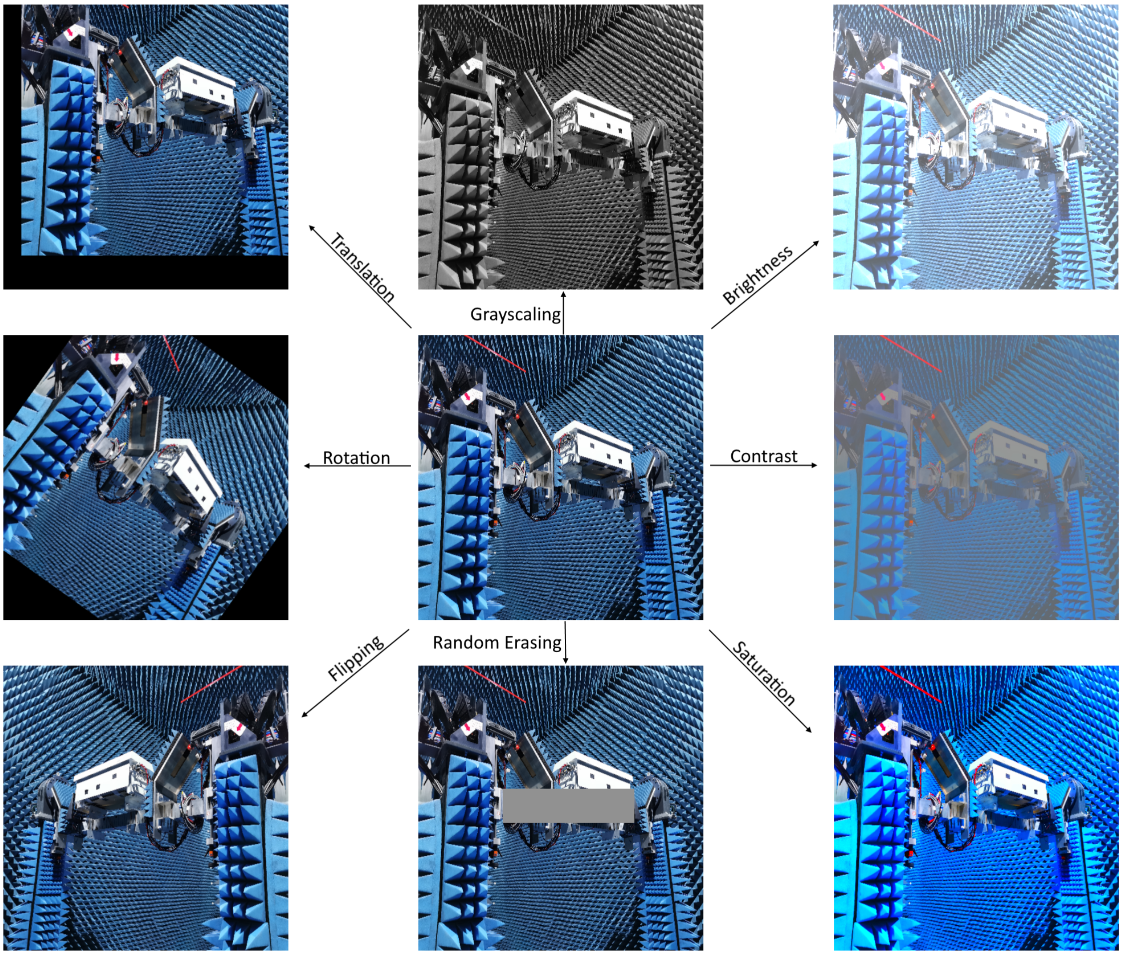


Figure 2.15: Resulting images after performing different augmentations.

2.5 Confidence Threshold

The confidence threshold is the minimum confidence score of a prediction needed for a detected object to be present in the final results. When making predictions each detection is given a confidence score for each of the classes. These scores represent how certain the model is that the different classes are present within the predicted bounding box. If the confidence score of an anchor box is lower than a set threshold then the object is discarded from the prediction. To determine the certainty of the network a softmax activation function is often applied. Generally a confidence score threshold of around 0.5 is used, however in the paper *Confidence score: the forgotten dimension of object detection performance evaluation* Wenkel et al. [44] discusses a method for finding the optimal confidence score for a model. Using this method the recommended confidence score for the *SSD-MobileNetV2-320-FPN* is 0.4 which is therefore the selected confidence threshold used in this paper.

2.6 TensorFlow

TensorFlow [45] is an open source library for machine learning developed by Google. It provides a large assortment of machine learning models and algorithms that can be applied to a number of domains including image recognition, natural language processing and voice recognition. A TensorFlow application can be depicted as a dataflow graph where each node corresponds to a mathematical operation, eg. ReLU or Convolutional Layer while the edges are multidimensional data arrays called tensors. For TensorFlow to be practical to use, applications can be written in a multitude of languages with Python being the typical choice [46]. However, to provide faster execution the algorithms provided by TensorFlow are written in high performance C++ [47]. Additionally, TensorFlow allows for the implementation of models on a vast assortment of heterogeneous systems with little to no changes being required [48]. Examples of systems include local machine, cloud clusters, mobile devices, Central Processing Units (CPU), Graphics Processing Units (GPU) and TensorFlow Processing Units (TPU).

2.6.1 TensorFlow Lite

TensorFlow Lite [49] is an extension of the TensorFlow library that uses operations that have been optimized for mobile devices and enables inference directly on the device with lower latency. Creating a TensorFlow Lite model is straightforward and is either done using the TensorFlow Lite model maker [50, 51] or by converting a trained TensorFlow model using the TensorFlow Lite converter [52]. Currently the support for the TensorFlow Lite converter only covers SSD models [53] resulting in the collection of possible pretrained models being as seen in Table 2.2. A reason to use the limited selection of models available for conversion is that the training of the model can then be performed on a larger dataset. To finalize a converted network metadata is added. The metadata contains information about the input and output of the model and consist of both human readable parts and machine readable parts [54].

Model Name	Speed (ms)	COCO mAP
SSD MobileNet V2 320x320	**19**	20.2
SSD MobileNet V1 FPN 640x640	48	29.1
SSD MobileNet V2 FPNLite 320x320	22	22.2
SSD MobileNet V2 FPNLite 640x640	39	28.2
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	**39.6**

Table 2.2: Statistics of pre-trained models from the TensorFlow 2 Detection Model Zoo [5] pretrained on the COCO dataset, that can be converted to TensorFlow Lite.

2.7 Android Studio

When developing applications for Android devices the official IDE to use is Android Studio [55]. There are three languages that are fully supported for Android app development, Kotlin, Java and C++. However, with the use of extensions additional languages such as Go can also be used [56]. Designing an app can be done either using XML or through a drag and drop menu, allowing for easy customization. Further, testing an application can be done either on a physical device or on an emulator available for download in Android Studio.

2.7.1 CameraX

CameraX [57] is a support library for Android Studio which helps with the development of camera features in applications. It uses lifecycle-aware components to perform camera operations such as preview, image analysis, image capture and video capture. Furthermore, CameraX also resolves device compatibility issues removing the need for device specific code. Using the image analysis of the CameraX the application can access each frame of a cameras video feed and use it to perform image processing or machine learning inference. This image analysis method is highly customizable with options to determine how the frames are handled, the size of the frames, etc. [58].

2.8 Real Time Applications

A real time application is a program which achieves enough frames per second for a users actions to appear instant. An example of a real time application is the camera application on a mobile phone which displays the camera view in real time to the user. A frame rate in the vicinity of 24 FPS is needed to accomplish this [59]. If the frame rate drops below that point actions will look sluggish and delayed. If the application is slower than the required 24 FPS it can instead be considered interactive. In an interactive application the FPS is slightly lower and the actions are somewhat delayed. For an application to have an interactive frame rate it can be expected to run at a speeds in the vicinity of 10 FPS [60].

2.9 Installation Components

When measuring the performance of radio equipment in a chamber the installation of the equipment under test (EUT) is a critical element for the overall success and usefulness of the collected data. A proper and uniform installation provides better measurement accuracy, better repeatability and a reduced risk of damage to equipment and personnel. The components are installed in differently sized chambers with names reflecting the length in meters between the product and a reflector. Two chambers are studied in this paper and will henceforth be referred to as the A9 and A3 chambers.

2.9.1 Base

The base is a platform on which the mast is attached. The base can be used for yaw rotation which allows the product to be evaluated in varying orientations during testing. An example of the base is shown in Figure 2.16 and a graph displaying yaw rotation can be seen in Figure 2.17.

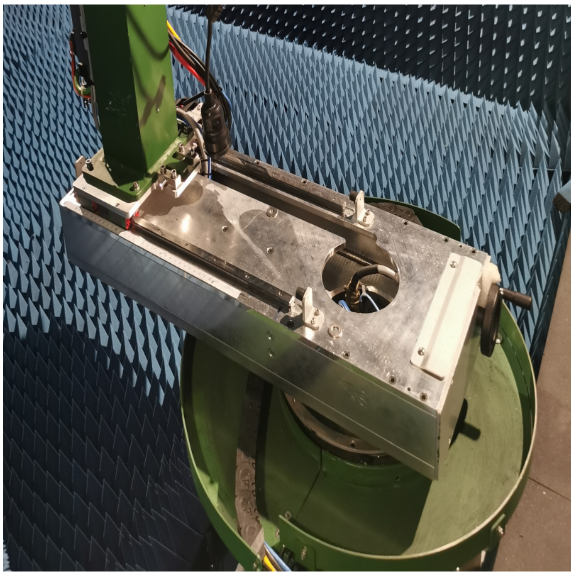


Figure 2.16: Example image of the base in the A3 chamber.

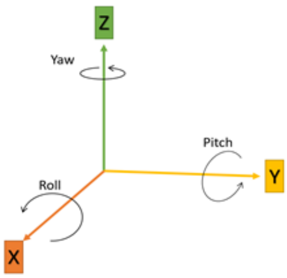


Figure 2.17: Image displaying the axis of different rotations.

2.9.2 Cabling

Cable management is a crucial part of the chamber mounting. By tightening and hiding cabling against the pole the operations of the chamber will be smoother and yield more accurate results. Further, a proper cable managed mounting will reduce the risk of damaging the chamber and cabling during rotation of the mount. This will in turn increase the longevity of the chamber thus reducing costs. An example of the cabling is shown in Figure 2.18.

Figure 2.18: Example of the cabling in an installation.

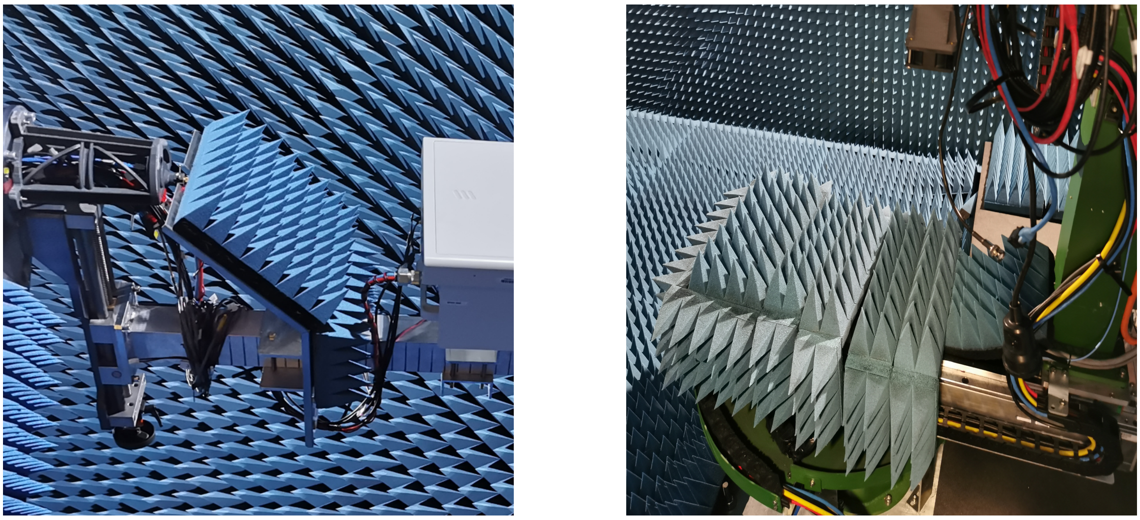
2.9.3 Cable Cover

The cable cover shields cables which are moved along with the mast during rotation. As these cables are a fixed part of the installation and cannot be removed this shielding is used to prevent damages to the cables. Using the cable cover will increases the longevity of the cables thus reducing costs. An example of the cable cover is shown in Figure 2.19.

Figure 2.19: A figure displaying a cable cover which is visible in the A3 chamber but not in the A9 chamber.

2.9.4 Divider

A divider is in its simplest form a removable piece of absorber which can shield parts of the chamber such as the mast or pole from the radios signals. Using dividers it is possible to prevent reflections and re-radiation from surfaces which would normally not be covered. Proper use of dividers help increase measurement accuracy and reduce the risk of requiring re-measuring. Also, there exist different dividers for different radio signal wavelengths and thus the correct ones has to be used to achieve the best result. Examples of the divider in both the A3 and the A9 chamber can be seen in Figure 2.20.



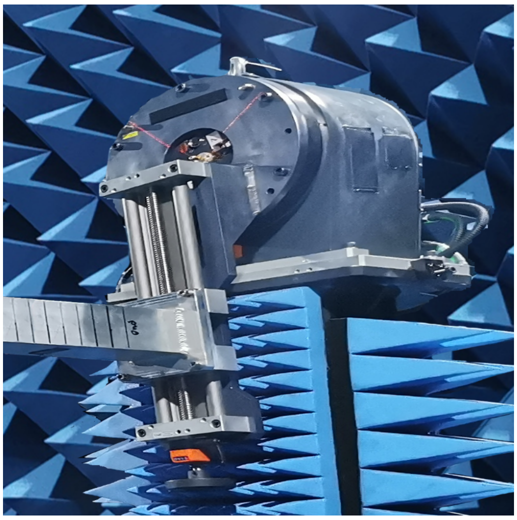
(a) Example of a divider in the A9 chamber.

(b) Example of a divider in the A3 chamber.

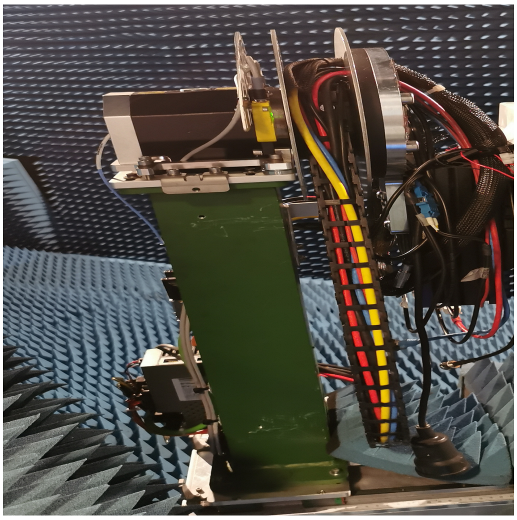
Figure 2.20: Two dividers, one from each of the chambers.

2.9.5 Mast

Either one or two masts are present when mounting and measuring in a chamber. The mast provide a surface to mount on while also providing rotation of the EUT. The mast is often located inside or near the quiet zone and can be moved depending on the desired mounting and size of the EUT. The quiet zone of an anechoic chamber is a volume where the reflected electromagnetic waves are lower than a set minimum [61]. There is also a weight limit of the mast to ensure smooth and safe operation. Examples of the mast for the A3 and A9 chamber can be seen in Figure 2.21.



(a) Example of a mast in the A9 chamber.

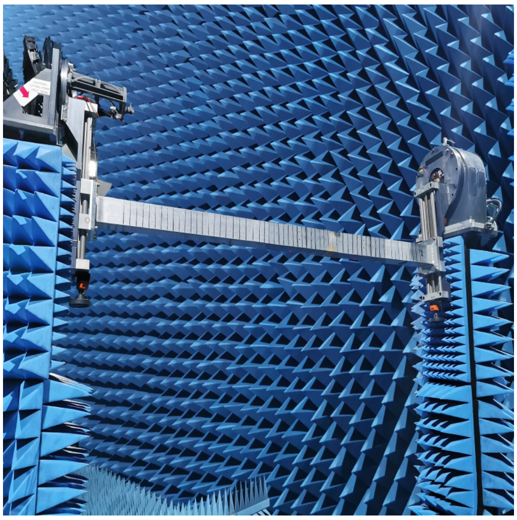


(b) Example of a mast in the A3 chamber.

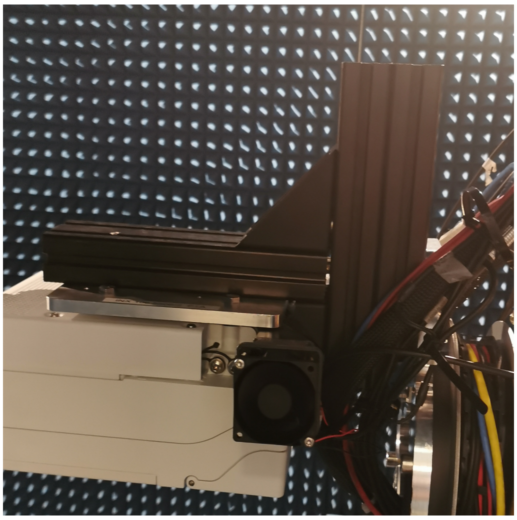
Figure 2.21: Two masts, one from each of the chambers.

2.9.6 Pole

A pole is used in the measuring chamber to hold the EUT. It is mounted using one or two masts depending on the weight and size of the EUT. Using the pole allows for yaw and pitch rotation instead of yaw and roll, Figure 2.17, simplifying the movement of non-complex radiation measurements. The pole is often placed as to allow the EUT to rotate around the turntable center of rotation. There exist many variations of poles which consists of different shapes and materials. These materials often have weight limits to ensure smooth and safe operations. Examples of the pole for the A3 and A9 chamber can be seen in Figure 2.22.



(a) Example of a pole in the A9 chamber.

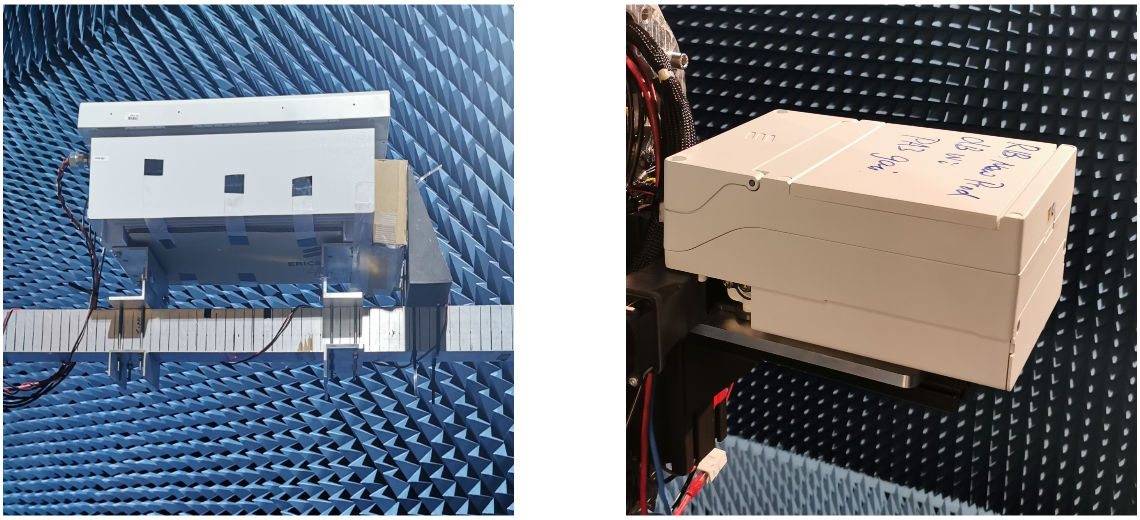


(b) Example of a pole in the A3 chamber.

Figure 2.22: Two poles, one from each of the chambers.

2.9.7 Product

The EUT or product is the radio unit mounted and tested in the chamber. There exist a lot of products with different sizes and weights that broadcast at different frequencies. As mentioned above the size and weight determine the type of pole it is mounted on and the number of supporting masts needed. Examples of the product for the A3 and A9 chamber can be seen in Figure 2.23.



(a) Example of a product in the larger A9 chamber.

(b) Example of a product in the smaller A3 chamber.

Figure 2.23: Two antenna products, one from each of the chambers.

3
Method

3.1 Dataset

Running detections for two types of chambers where the components and installation differs requires two different networks. To train these networks one dataset is required for each of the chambers, the A9 and A3 datasets.

3.1.1 Collecting Data

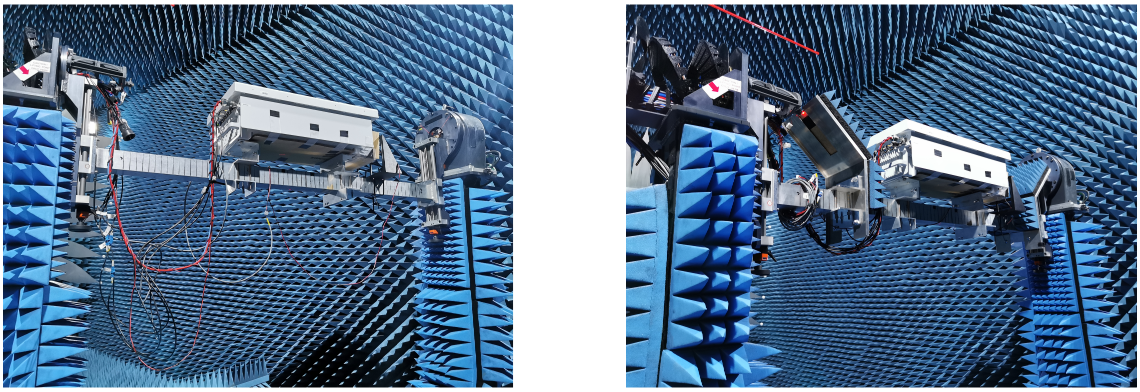
The majority of the images were captured by hand in the anechoic chambers at the Ericsson Lindholmen facility. Further, additional images collected from chambers at other Ericsson locations were available. The mounting in these chambers had minor differences compared to the ones at the Lindholmen site. These images were included in the datasets to improve the generalization skill of the networks. The images are of varying sizes since different cameras and camera modes affect the number of pixels. Each image is annotated using labelImg which is a free, opensource tool for graphically labeling images [62]. The bounding boxes and annotations are saved in pascal VOC format which can be seen in Figure 3.1.

```
<annotation>
     <folder>images</folder>
     <filename>1.jpg</filename>
     <path>D:\ImgAnnotation\images\1.jpg</path>
     <source>
          <database>Unknown</database>
     </source>
     <size>
          <width>3648</width>
          <height>2736</height>
          <depth>3</depth>
     </size>
     <segmented>0</segmented>
     <object>
          <name>Mast</name>
          <pose>Unspecified</pose>
          <truncated>0</truncated>
          <difficult>0</difficult>
          <bndbox>
               <xmin>1157</xmin>
               <ymin>1013</ymin>
               <xmax>2131</xmax>
               <ymax>1673</ymax>
          </bndbox>
     </object>
```

Figure 3.1: Bounding box and annotations in the pascal VOC format.

3.1.2 A9 Dataset

The A9 dataset consists of roughly 1700 images and 10 000 annotations containing the objects *Cabling, Divider, Mast, Pole* and *Product*. The images contain the objects in different rotations, compositions and with varying precision in the installation, further, the images are taken from different angles. Example images from the A9 dataset can be seen in Figure 3.2 and the number of annotations for each object can be seen in Figure 3.3.



(a) Example image with bad mounting.

(b) Example image with good mounting.

Figure 3.2: Images taken from the A9 dataset.

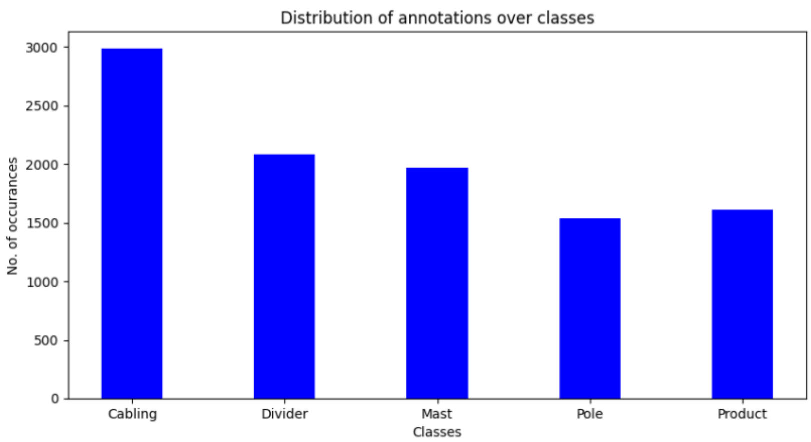
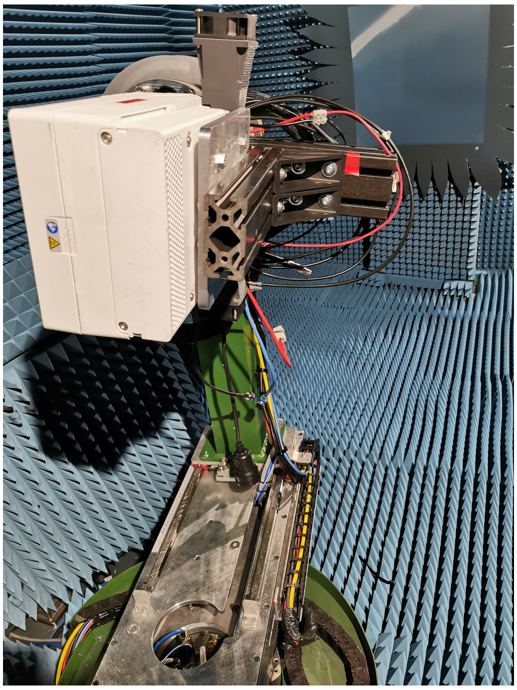


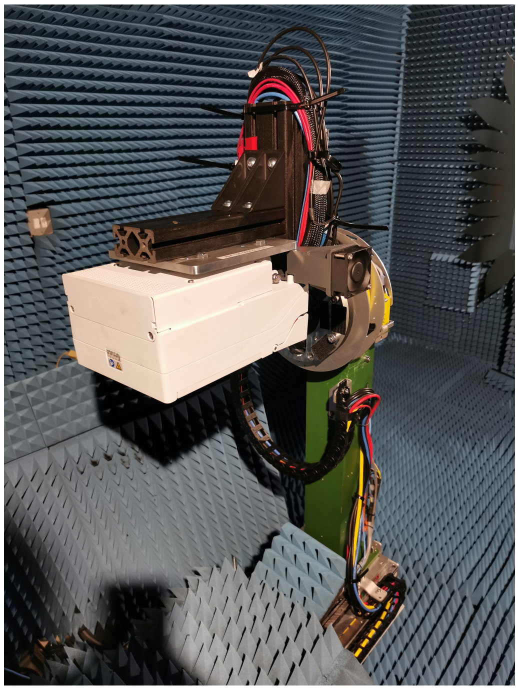
Figure 3.3: The number of annotations for each class in the A9 dataset.

3.1.3 A3 Dataset

The A3 dataset is smaller consisting of around 730 images and 5300 annotations. The objects present in these images are *Base, Cabling, Cable Cover, Divider, Mast, Pole* and *Product*. Similarly to the A9 dataset the images capture the mounting in different rotations, compositions, installations and angles. Example images from the A3 dataset can be seen in Figure 3.4 and the number of annotations for each object can be seen in Figure 3.5.



(a) Example image with bad mounting.



(b) Example image with good mounting.

Figure 3.4: Images taken from the A3 dataset.

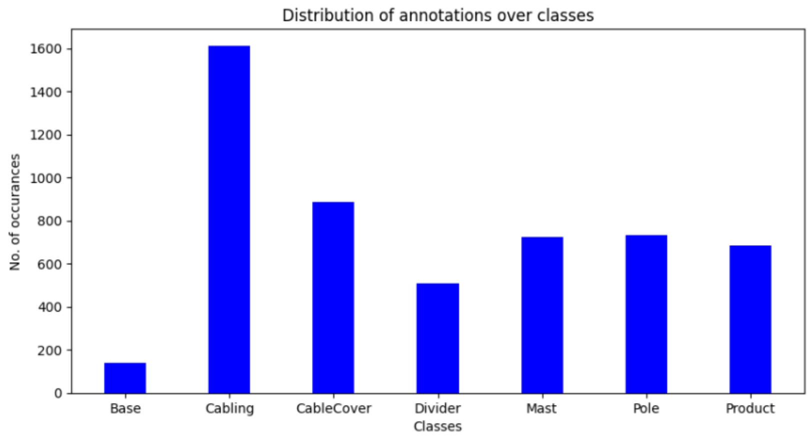


Figure 3.5: The number of annotations for each class in the A3 dataset.

3.2 Creation of Models

When choosing which network models to use in the application there were a set of requirements that had to be fulfilled. Firstly, since the application was to run on a mobile device and perform real time predictions the computational cost of performing inference had to be low. Secondly, the model along with the application had to be small enough to fit in the device's memory, requiring the model to be small in size. Finally, the model had to be able to accurately detect and predict the classes of the objects in the scene.

As described in section 2.6.1 there are a handful of networks which support transformation from a TensorFlow model to a TensorFlow Lite model (Table 2.2). The major difference between the available models is the size of the network and the size of the input. Depending on which model is used there is a trade off between speed and accuracy. Two designs were chosen, SSD MobileNet V2 FPNLite 320x320, SSD MobileNet V2 FPNLite 640x640 where the prior is smaller in size with faster inference but with lower accuracy. These were chosen since according to Table 2.2 the models could satisfy the requirements stated above where both had the potential to achieve a high frame rate running inference on a mobile device.

3.2.1 Training the Models

For the models to be able to detect the objects from the dataset it required retraining which was done in the form of transfer learning. While running transfer learning the entire feature selection portion of the models were frozen meaning that their weights were not changed. Instead, only the fully connected output layers responsible for classification was retrained using the datasets described in section 3.1. The datasets were split 90%/10% into a training and a test set. The training set was used for training the models while the test set was used to evaluate their performance. The training was carried out for 70 000 - 100 000 steps with a varying batch size and a constantly changing learning rate as can be seen in Figure 3.6. During the training an assortment of metrics were tracked such as, regularization loss, classification loss, localization loss and total loss which was used to get an overview of the performance of the networks at different stages of the training.

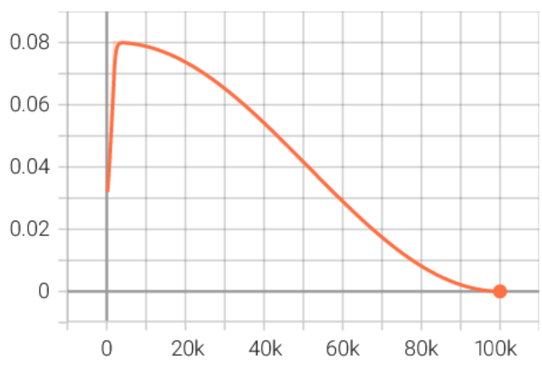


Figure 3.6: Example learning rate used during model training.

3.2.2 Data Augmentation

As mentioned in section 2.4 data augmentation can be applied to a dataset in order increase its quality and size. Every image fed to the network first had its size scaled to match the expected input size of the model. Additionally, when training the models three different data augmentation techniques were applied to every image with a probability of 50%. The augmentations used were vertical flip, random rotation and random crop. Further, experiments with other augmentations such as grayscaling was performed however no substantial difference in the performance was identified.

3.2.3 TensorFlow Lite Models

To prepare the models for usage on a mobile device they were converted to TensorFlow Lite models using the TensorFlow Lite converter. Furthermore, in order for the models to be usable in conjunction with the application metadata was added to them. This metadata specifies the shape and type of the input and output as well as gives a description of the model. The metadata was modeled according to the standard required by TensorFlow Lite and expected by CameraX.

3.2.4 Evaluating Models

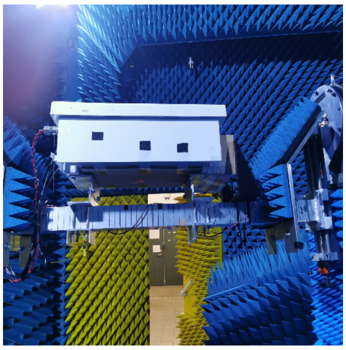
Evaluating the performance of the models was done using the four metrics, precision, recall, mAP and frame rate. The two main metrics studied were the mAP and the frame rate, this is, as explained in section 2.1.4, because the mAP score is based on the precision-recall curve. As the application aimed to be running in real time using a continuous video feed the models had to achieve a high enough frame rate, if a model could not reach a high enough frame rate it was neglected even if the mAP score was promising.

3.3 Creation of the App

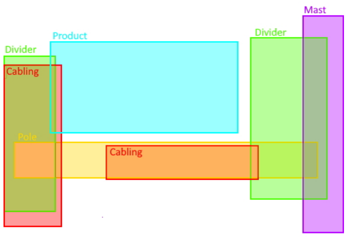
An application was created to provide the user with an intuitive interface for detecting and displaying interference points for each of the chambers. Further, the application uses these results to relay relevant information about the product installation. The app was created for android devices using the official IDE for android app development Android Studio. For the application to work as intended a number of core functionalities were implemented.

Firstly, live input in the form of a video feed is captured and displayed to the user. This is done by accessing the camera of the device and mirroring the view onto the display using the preview functionality of CameraX. This recreation of the camera view can be seen in the background of Figure 4.5a.

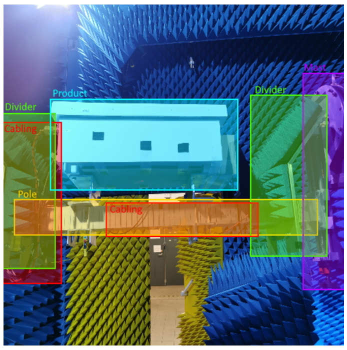
Secondly, utilizing the video feed input and an object detection network the application detects all relevant objects in each input frame and displays it to the user, Figure 4.5a. To run detection on each frame an ImageAnalysis object from the CameraX library is created. This analyzer employs a listener which triggers on every new frame and runs a method with the frame as an input. The analyzer runs on a separate thread and may therefore receive frames faster than it can process them. If the current frame has not yet been processed all incoming frames are discarded until the analyzer is free. In the Analyze method the frame is scaled down to match the input size of the network. The re-scaled frame is processed by the network which returns the predicted labels and locations of the objects. These predictions are then scaled up to the size of the display and drawn on a transparent canvas overlaying the video feed, Figure 3.7. Displaying the bounding boxes on a separate plane allows the frame rate of the image feed to remain constant which makes the app look and feel smoother.



(a) Example of an input frame.



(b) Example output for the input frame.



(c) Final result when overlaying boxes over the input frame.

Figure 3.7: Steps taken by the application to overlay detection results onto the video feed.

Thirdly, as the application uses separate networks for the different chambers an options menu was implemented which contains options to switch between models and more. As the video feed is running three options are displayed as seen in Figure 4.5b. These options are *Chamber A9* which when selected uses the model trained on the A9 dataset, *Chamber A3* which when selected uses the model trained on the A3 dataset and *Display Interference* which allows the user to toggle between displaying the bounding boxes of all objects and only the objects which are interfering with the tests. Displaying interference is further explained in section 3.4 and the effect of the option can be seen in Figure 4.5e.

Fourthly, to give directions for possible improvements the user can pause the video feed, Figure 4.5c. When paused the application displays the previous prediction and an additional option appear in the menu, Figure 4.5d. This new option called *Propose Fixes* opens a popup window with instructions on actions to take to improve the installation, Figure 4.5f. The proposed actions are based on the detection information of the last frame.

Lastly, the application has a number of settings for which the functionality is not always self explanatory and the installations has multiple components which the installer may be unfamiliar with. To make the functionality of the settings and the purpose of the components clear a *Help* page was implemented. The *Help* page is composed by 8 different windows containing information either about the settings or the components present in the chamber which the user can swipe between, Figure 4.6. Each window is created as its own fragment class in Android Studio making them easy to modify and style.

3.4 Toggle Area of Interference

After detection has been performed the results are evaluated to determine whether there are any possible points of interference in the image. The object classes are divided into two groups, fixed and non-fixed objects. The fixed objects are the base, cable cover, mast, pole and product that regardless of the care given to the mounting will be roughly the same, while cabling and dividers are non-fixed as they may vary greatly between mounts. The non-fixed objects can be further divided into components that increase and decrease the interference in the chamber. For example hanging cables increase the number of reflective surfaces thus increase interference while dividers cover other objects reducing the number of reflective areas and decreasing the interference. The application gives advice on how to improve the installation of the non-fixed objects using all the detection information of the image. The advice given and the method of producing them varies between the chambers.

3.4.1 Suggest Fixes A9

The two non-fixed objects, cables and dividers, are evaluated in two different ways. The cables are evaluated on the overlap between their bounding box and the bounding boxes of other objects in the image. If the overlap between the cables and other objects is less than a set threshold the installation is poor and is therefore considered a possible cause for interference. In a good installation the overlap would be high as the cables are wrapped tight around the pole. The method used for calculating the overlap is further explained in section 3.4.3. The dividers however are not evaluated on their overlap with other components, instead, they are evaluated on whether or not they are detected in the image.

The threshold parameter used to decide if a cable installation is good was determined through analyzing the cable overlap in well executed installations and in poorly executed installations. The results gathered are presented in section 4.4 and a threshold of 0.75 was used.

If the cabling or dividers are deemed interference points then a list of suggested fixes is given when the setting *Propose Fixes* is pressed. Some of the proposed fixes can be seen in figure 4.5f.

3.4.2 Suggest Fixes A3

For the A3 dataset there are three different conditions that are checked when determining whether an installation is good or not. Firstly, similarly to the A9 dataset the cables are evaluated on the overlap between their bounding box and the bounding boxes of other objects in the image, if this overlap is less than a set threshold the installation is deemed poor. Secondly, the installation is evaluated based on whether the dividers are detected or not. Lastly, the installation is also evaluated on whether or not the base is showing. If the base is detected the installation is poor as it is supposed to always be covered by dividers.

Similar to how the threshold parameter was selected for the A9 chamber a threshold was set for the A3 chamber. This result is also presented in section 4.4 and a threshold of 0.8 was used.

Identically to the A9 chamber if any interference points are detected then a list of proposed fixes is displayed when *Propose Fixes* is pressed. The proposed actions are the same for both chambers with A3 having additional actions to cover the base.

3.4.3 Calculate Overlap

When describing how the overlap is calculated A will represent a cable bounding box, B and C represents boxes of any other class. I_{AB} and I_{AC} are the intersections between A and B and A and C respectively. Additionally L_1 and L_2 are lists used for storing the intersection points. A visual representation can be seen in Figure 3.8. The overlap is calculated by taking the intersection between A and B, if I_{AB} exists it is saved to L_1. Additionally I_{AB} is checked against every previous entry in L_1. Next the intersection between A and C is checked. Similar to I_{AB} it is added to L_1 and checked against every previous entry. In this check the area I_{ABC} is found to overlap between the different overlaps and is therefore saved in L_2. When all the boxes have been checked the total overlap is calculated as

$$\frac{\sum Area(L_1) - \sum Area(L_2)}{Area(A)} \tag{3.1}$$

Pseudo code for obtaining the overlaps of the cabling can be found in Appendix A.

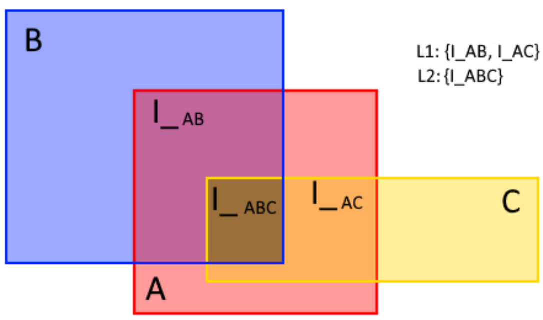


Figure 3.8: Visual representation of how the total overlap is calculated.

4

Results

4.1 Models

The models trained and evaluated are the *SSD MobileNet V2 FPNLite 320x320* and *SSD MobileNet V2 FPNLite 640x640* both trained on the A9 and A3 dataset.These models were trained with a varying amount of steps and different batch sizes to find which resulted in the best network. After all the training was completed only the best version of each network was kept and is displayed in this section.

4.1.1 Training

When training the models a set of metrics were recorded. These metrics were *classification loss*, *localization loss*, *regularization loss* and *total loss*. The results of training the different networks can be seen in Figure 4.1 and the final values are displayed in Table 4.1.

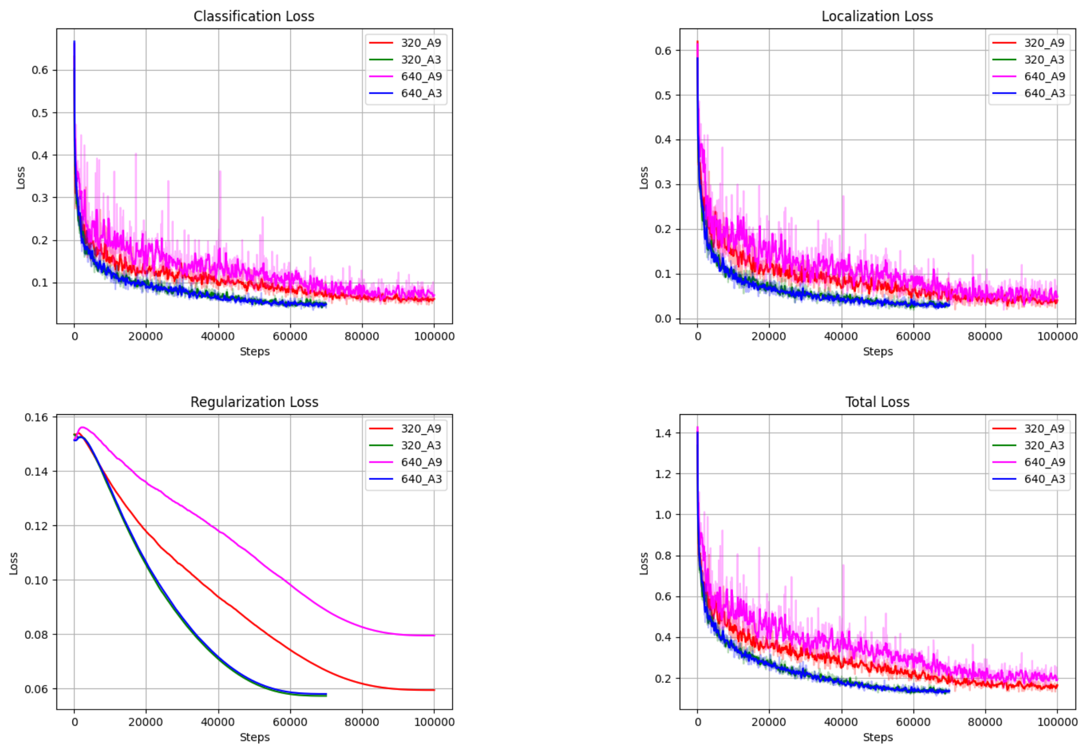


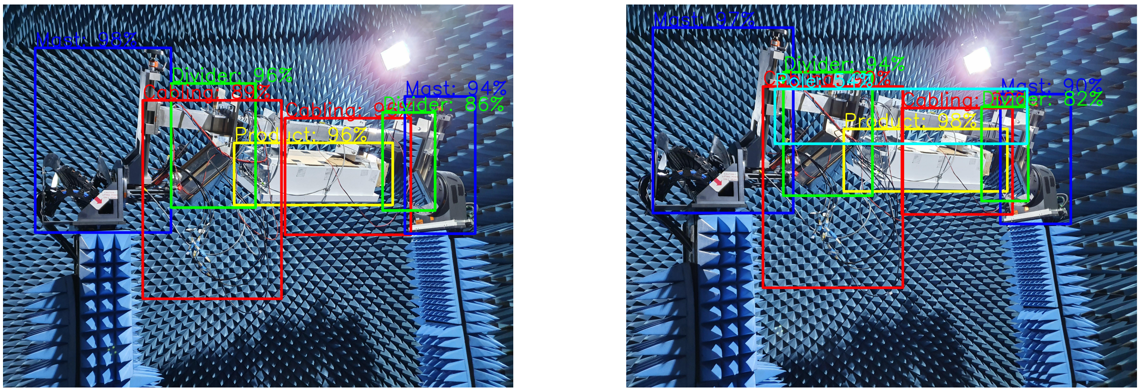
Figure 4.1: Metrics recorded during training for the different networks.

Model	Classification Loss	Localization Loss	Regularization Loss	Total Loss
SSD 320x320 A9	0.0473	0.0192	0.0594	0.1328
SSD 320x320 A3	0.0399	**0.0181**	**0.0573**	0.1202
SSD 640x640 A9	0.0490	0.0186	0.0795	0.1484
SSD 640x640 A3	**0.0350**	0.0198	0.0580	**0.1129**

Table 4.1: Table of metrics obtained during training.

4.1.2 Detection Result

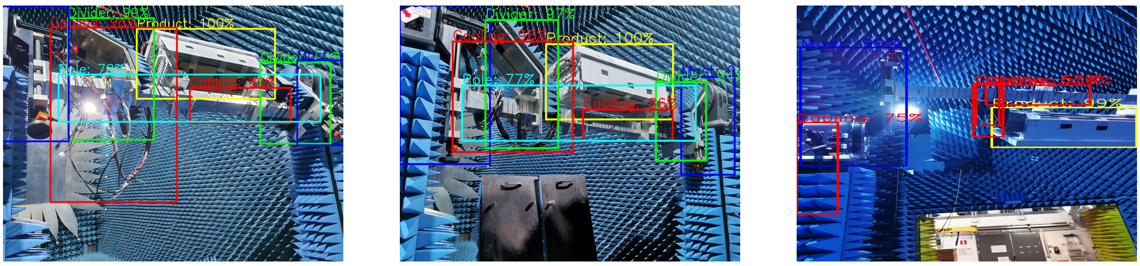
Running inference using the *SSD MobileNet V2 FPNLite 320x320* trained on the A9 dataset the results seen in Figures 4.2 and 4.3 were obtained. These figures contain detection results of the mounting in different rotations and with different installations. The results will be further discussed in chapter 5.



(a) Detection result finding all components but the pole.

(b) Detection result where all components are found.

Figure 4.2: Comparison between two frames where the angle of the images differs slightly.



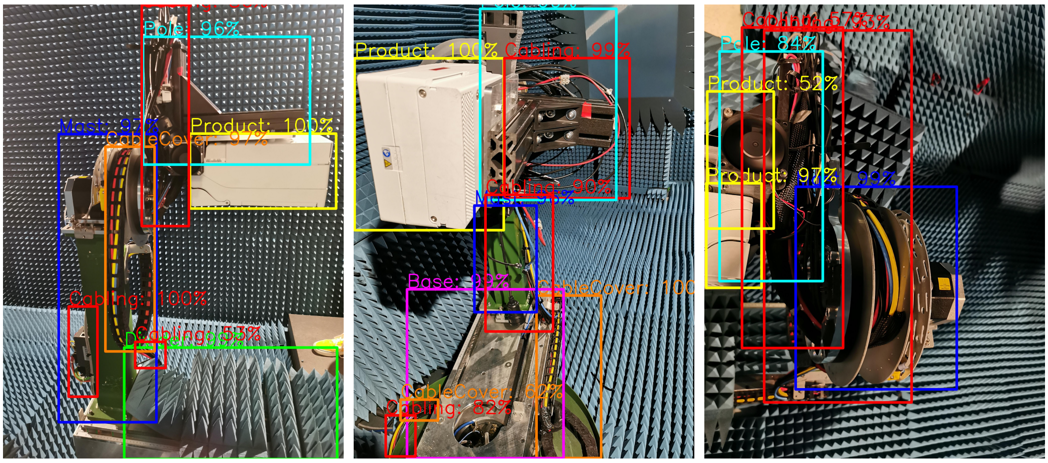
(a) Detection on a poorly executed installation.

(b) Detection on a well executed installation.

(c) Detection with multiple overlapping predictions and a missing pole.

Figure 4.3: Detection results on images from the A9 test set with varying installations and accuracy.

Running inference on the *SSD MobileNet V2 FPNLite 320x320* trained on the A3 dataset the images in Figure 4.3 was obtained. The figure displays three different installations and predictions. The first shows a good installation, the second a bad installation and the third poor predictions. The results will be further discussed in chapter 5.



(a) Detection on a well executed installation.

(b) Detection on an installation with base showing and poor cable management.

(c) Detection with multiple overlapping predictions.

Figure 4.4: Detection results on images from the A3 test set.

4.1.3 Detection in Chamber

One of the goals stated in Section 1.2.1 was to test the final application in the anechoic chambers for different interference scenarios. A video displaying detections in the A3 chamber in varying rotations and installations can be seen here.

4.1.4 Evaluation

The accuracy of the trained models are evaluated on images from the respective test sets. The results from these evaluations are used to determine the performance of the networks. Using previously unseen data to run evaluations expect to give similar results as running the application in the chambers.

4.1.4.1 Evaluation Metrics

The results obtained from evaluating the models on the test sets can be seen in Table 4.2 and Table 4.3. In Table 4.2 the mAP for different IoU scores and the average recall (AR) for images with at most 1, 10 and 100 predictions are displayed while Table 4.3 displays the different average losses over all test images.

Model	mAP	mAP@.5IOU	mAP@.75IOU	AR@1	AR@10	AR@100
SSD 320x320 A9	0.6184	**0.8765**	0.7024	0.5216	0.7098	0.7212
SSD 320x320 A3	0.6246	0.865	0.7027	**0.6203**	0.6917	0.6997
SSD 640x640 A9	**0.6258**	0.8701	**0.7063**	0.5286	**0.7223**	**0.7349**
SSD 640x640 A3	0.6126	0.8515	0.6820	0.6111	0.6806	0.6896

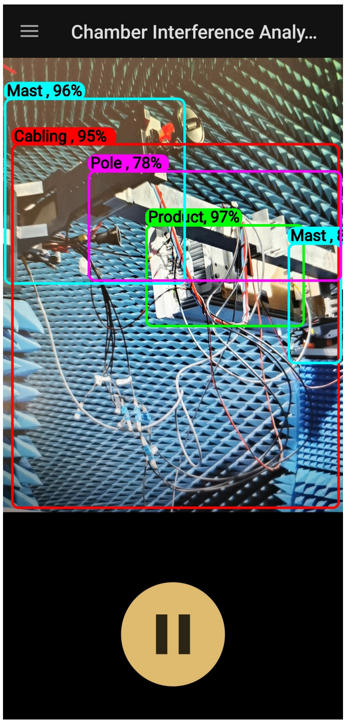
Table 4.2: Table of evaluation metrics obtained from test set.

Model	Classification Loss	Localization Loss	Regularization Loss	Total Loss
SSD 320x320 A9	**0.1992**	**0.1354**	0.0594	**0.3940**
SSD 320x320 A3	0.3399	0.1977	**0.0573**	0.5949
SSD 640x640 A9	0.1999	0.1368	0.0794	0.4162
SSD 640x640 A3	0.4314	0.2229	0.0580	0.7123

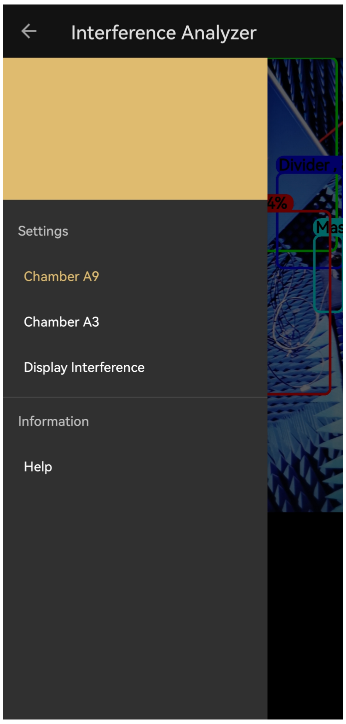
Table 4.3: Table of evaluation metrics obtained from test set.

4.2 Application

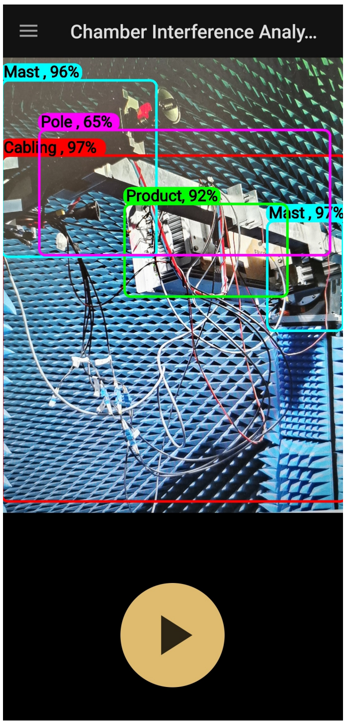
The base functionality of the application is displayed in Figure 4.5 and Figure 4.6. The application runs object detection on a live image feed, Figure 4.5a. When the app is running a number of settings exist, Figure 4.5b, which are described in detail in section 3.3. Pausing the detection, Figure 4.5c, allow for studying a prediction, display interference and also enables the app to provide feedback of the installation, Figures 4.5d, 4.5e and 4.5f. If the *Help* button in the menu is pressed the user is presented with information about the available settings and the different installation components, Figure 4.6.



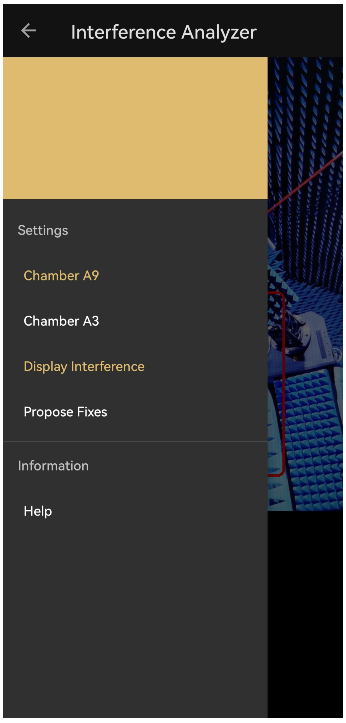
(a) Application on startup with running video feed.



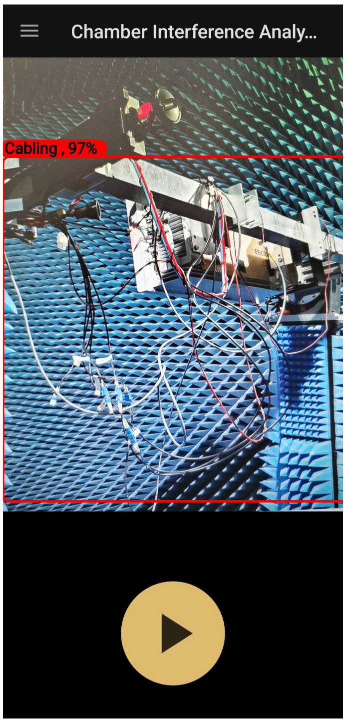
(b) Options available when running detection.



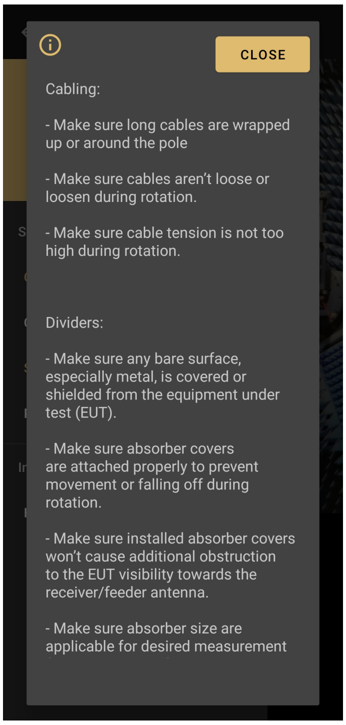
(c) Application when paused.



(d) Options available when detection is paused.

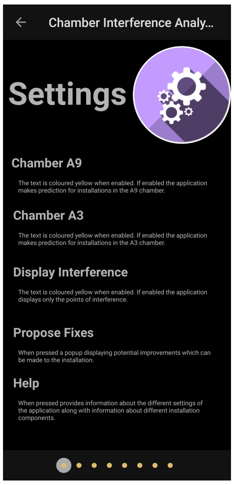


(e) Resulting view of only displaying problem areas.

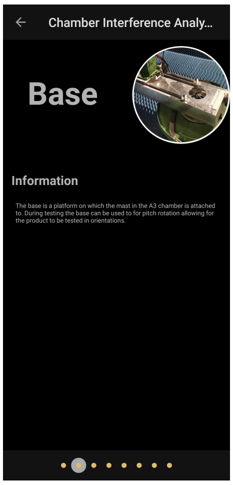


(f) Popup created when propose fixes are pressed.

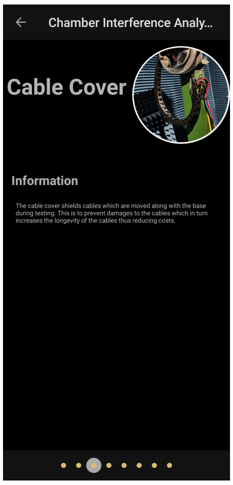
Figure 4.5: Different views of the application.



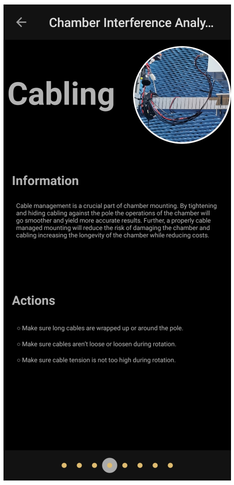
(a) Help page with information about the settings.



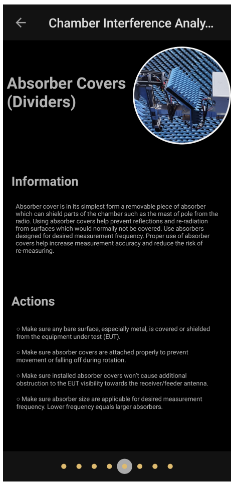
(b) Help page with information about the base.



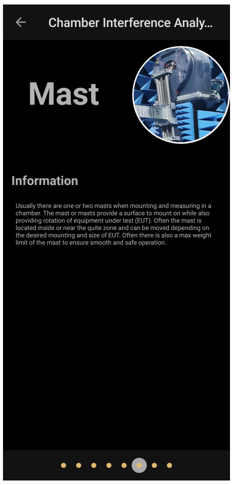
(c) Help page with information about the cable covering.



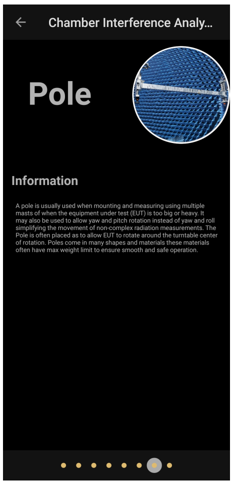
(d) Help page with information about and actions to improve cabling.



(e) Help page with information about and actions to improve divider.



(f) Help page with information about the mast.



(g) Help page with information about the pole.

(h) Help page with information about the product.

Figure 4.6: Information about the installation of different components which is shown if a user press the Help button in the menu.

4.2.1 Frame Rate

The speed of the final application varies depending on the size of the network used. Figure 4.7 displays the achieved frame rate of each network when performing inference and drawing the results. Each box is created using 100 samples collected using the application with the respective network. The average speeds of the networks can be seen in Table 4.4. The camera feed displayed in the background of the application runs at 60 FPS regardless of which network is used.

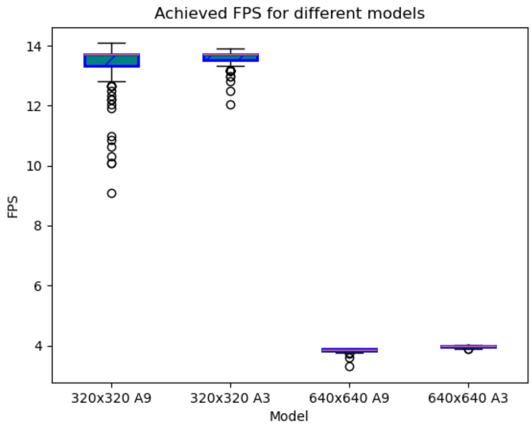


Figure 4.7: Box plots displaying the prediction speeds achieved by the networks with varying size.

Model	Average FPS	Standard Deviation FPS
SSD 320x320 A9	13.45	0.61
SSD 320x320 A3	**13.57**	0.23
SSD 640x640 A9	3.85	0.06
SSD 640x640 A3	3.97	**0.03**

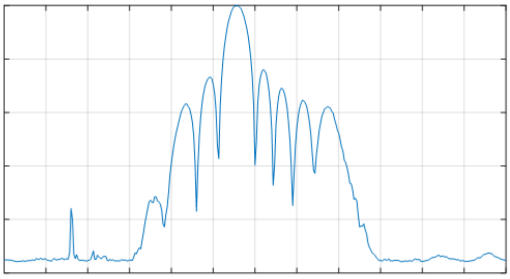
Table 4.4: Table of FPS for different networks.

4.2.2 Size

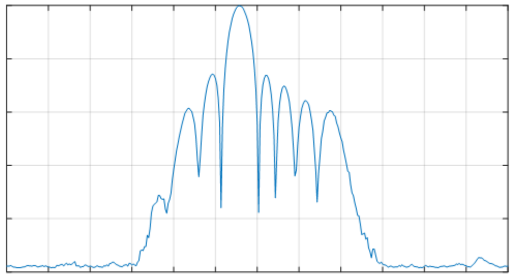
The size of the final application is 74 MB where 3327 kB and 3269 kB are the sizes of the A9 and A3 network respectively. The remaining memory usage comes from the supporting code and visual elements. This size is measurable to other minuscule applications.

4.3 Impact of Interference in Tests

Missing or improper use of dividers can cause interference and impact measurement results. In Figure 4.8 the impact of reflections and re-radiation due to bare surfaces during measurements can be seen and compared to the impact of not having reflection and re-radiation. This interference causes a spike in an unwanted direction disrupting the measurements as can be seen in the figure.



(a) Measurement without properly installed dividers.

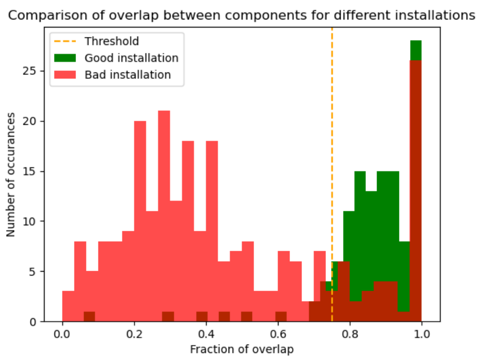


(b) Measurement with properly installed dividers.

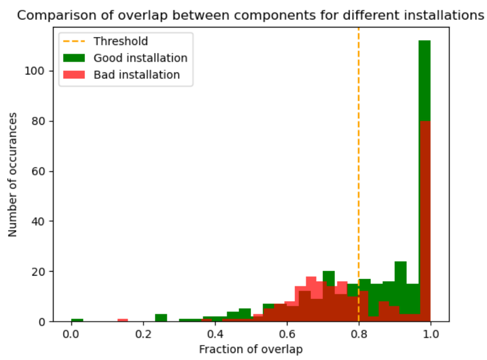
Figure 4.8: Difference in measurements between installation with and without dividers. The labels and values of the x and y axis has been removed as it is considered confidential information by Ericsson.

4.4 Deciding Overlap Threshold

The thresholds used to detect interference points in cable mountings were decided through analyzing the overlap of cables with other installation components. The Figures 4.9a and 4.9b displays two histograms of the overlaps found in the images. From these plots an overlap threshold of 0.75 and 0.8 is chosen for the A9 and A3 chamber respectively as it provides a good separation between the types of installations. In the Figure 4.10 two images of the same installation taken from different angels are displayed. For these images the overlap varies greatly despite originating from the same installation.

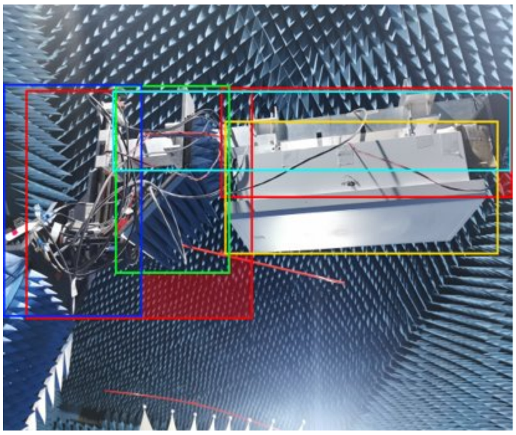


(a) The overlap of cables with other components for images with good and bad mounting in chamber A9.

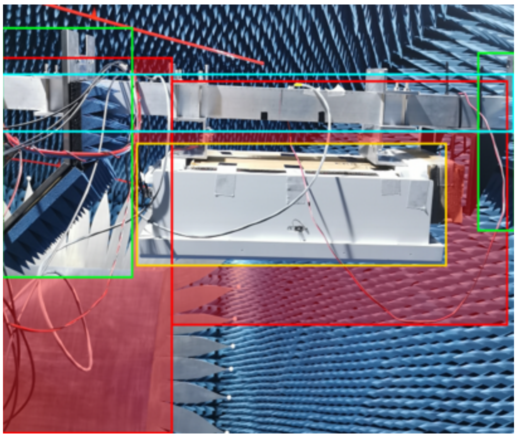


(b) The overlap of cables with other components for images with good and bad mounting in chamber A3.

Figure 4.9: Overlaps between cables and other components for the different chambers.



(a) Image of poor installation where a majority of the cables overlap with other components. The area which does not overlap is colored in red.



(b) Image of the same installation captured at an angle which reduces the amount of cables which overlap with other components. The area which does not overlap is colored in red.

Figure 4.10: Two images of the same installation taken from different angles.

5
Discussion

The aim of the project was to develop a method for reducing interference within radio testing chambers. This was achieved using the methods propounded in section 3 which are based on the theory presented in section 2. The achieved results were presented in section 4 and will be further discussed here.

5.1 Model Evaluation

5.1.1 Accuracy

From the evaluation statistics displayed in Table 4.3 it is evident that the smaller (SSD MobileNet V2 FPNLite 320x320) networks achieves similar or better results than the larger (SSD MobileNet V2 FPNLite 640x640) networks. This is unexpected as a model which receive more information as an input should be able to produce more accurate predictions. Some possible reasons for this outcome could be that the larger networks had to be either trained for longer, trained using a larger batch size or be trained on a larger dataset. These conclusion comes from the fact that a larger model has more parameters and will therefore require more training. However, as the inference times for the larger networks greatly exceeded the set requirement additional investigation into the problem was not conducted.

5.1.2 Detection Results

In Figures 4.2 and 4.3 detection results of the A9 network when fed images from the test set were presented. The first set of results, Figure 4.2a and 4.2b, displays seemingly identical images where the the only difference is that the rotation of the camera differs slightly. In image 4.2b all objects are detected while the pole is missing from 4.2a. This phenomenon of the camera needing to be slightly angled to reliably detect the pole is caused by a rotational bias in the model. An attempt to remove this bias was done by introducing data augmentation in the form of random rotation, while this reduced the bias it did not remove it completely. Further attempts to remove the bias were made by introducing additional images to the dataset. The focus of these images was to display a clear view of a centered pole with no rotation and few other visible components. This was done to provide the network with images of the pole to learn the basic structure of it.

In the second set of images seen in Figure 4.3 we can see how the network handles predictions on varying installations. From the first image 4.3a it is clear that the network is proficient at localizing and labeling the objects. However, as previously mentioned the network is not equally good at detecting all components. From the figure it is evident that it struggles somewhat with the pole. This can both be seen from the predicted bounding box of the pole being slightly out of position and the confidence score of the prediction being 20% lower than for the other objects.

Comparing Figure 4.3a and 4.3b it is clear that the network is skilled at detecting objects both when the installation is good and bad. This is reassuring as the detection of all objects allows for more accurate potential improvements to be presented to the user. However, by looking at Figure 4.3c we find that the model will sometimes overlap predictions as seen for the cables next to the product. Additionally, observing the leftmost cables we find that the bounding box extend further than required which could skew the result of the overlap calculation.

The images in figure 4.4 displays detection results of the A3 network when fed images from the test set. In the first figure, Figure 4.4a, the components are properly installed and the prediction is good. All components in the image are detected and the bounding boxes fit nicely around each object. This is also the case for the second figure, Figure 4.4b, for which the installation is poor with hanging wires and a base that is showing. In this figure we can see that the base has been detected with a confidence of 99%. This means that even though it is severely underrepresented in the dataset the network is still able to learn to detect it.

However, the third figure, Figure 4.4c, displays a mistake where the network predicts two overlapping and fairly similar bounding boxes for the same object. One of the bounding boxes encapsulates the cables wrapped around the pole while the other also includes cables visible inside the mast. Comparing this prediction to Figure 4.3c it is clear that both models will occasionally predict overlapping boxes.

When taking a closer look at the extra bounding boxes we can see that they generally have a substantially lower confidence score meaning that the model was unsure whether or not to include the box in the prediction. This results in the extra boxes being present in some but not all frames resulting in a flickering of uncertain bounding boxes when predicting in real time.

Removing the duplicate boxes is no simple task as increasing the confidence threshold could introduce new implications. One such implication is that a higher confidence threshold increases the risk of the model failing to detect important objects in the image.

5.1.3 Detection in Chamber

From the video in Section 4.1.3 displaying the results from the different interference scenarios it evident that the final application is able to achieve accurate detections for multiple types of installations. From the results displayed in the video it is clear that the final applications satisfies most of the goals set. The final product is both able to detect and relay possible points of interference to the user. The only goal not met by the application is to achieve real time inference as the methods and networks used achieved a highly interactive inference at best. Furthermore, the application was not tested for the A9 chamber due to ongoing construction in the chamber.

5.2 Overlap Threshold

In the images comparing the overlap for cables displayed in Figure 4.9 a large spike is present when the fraction of overlap equals 1.0 for both chambers and installation types. This spike indicates that a high number of cables overlap completely with other components of the installation. This is to be expected as the overlap of different components are highly affected by the angle in which the image was captured. For example, if an image with poorly mounted cables is taken from the side then they will with a high probability overlap a lot with other components, as seen in Figure 4.10. Thus, when selecting the overlap threshold for the different chambers these points were overlooked as they do not provide information about how to separate a good and bad installation. Additionally, while the histograms of good and bad installations are well separated in the A9 dataset resulting in an clear choice of the threshold the values of the A3 chamber are somewhat merged resulting in a more difficult choice. The reason behind this is that the installation in the A3 chamber is compact. Thus, regardless of how poorly the installation is done there will be a high overlap by default making the window separating the good and bad installations small. The minor difference in overlap between a good and bad installation in the A3 chamber can be seen in Figures 4.4a and 4.4b.

5.3 Application Speed

From the FPS results displayed in section 4.2.1 and the speed performance thresholds presented in section 2.8 it is evident than neither of the models allows the application to run inference in real time. However, when using the smaller (SSD MobileNet V2 FPNLite 320x320) networks the FPS is high enough allowing the application to run inference while also being highly interactive. Further, as previously mentioned the predictions of the network and the video feed runs on separate threads which assists the application in appearing smoother even when time between prediction updates are longer. Additionally, this also benefits the larger (SSD MobileNet V2 FPNLite 640x640) networks since when the device is held somewhat stationary the predictions will overlap well with the displayed image resulting in them being usable in static predictions. Although, for static predictions using a larger network running on a GPU would most likely provide far better results.

5.4 Application Size

A limitation stated in the introduction of the paper was the memory constraint of a mobile device. This constraint required the models used to be small enough to fit into the devices memory. This issue was mainly solved by converting networks into the TensorFlow Lite format which drastically reduced the size of the models. The total size of the application is comparable to any minuscule application. From this it is evident that larger networks can be used without any concern that it will not fit in the memory. However, as previously mentioned and shown a larger network has increased inference times. This increased inference time lowers the achieved FPS reducing the responsivity of the application and worsens the experience for the end user.

5.5 Impact of Interference in Tests

As seen in Figure 4.8 failing to use dividers correctly negatively affects the tests as signals are reflected off uncovered surfaces disrupting the measurements. The spike seen in Figure 4.8a reveal that there is a large signal radiating in the wrong direction. Such an error alters the data gathered which in turn can produce parameters not within the acceptable interval. Without the context of the installation this result would mean that there is something wrong with the product. Since this is not an acceptable result the product would not be ready for deployment and additional costs and time has to be spent examining this non-problem.

Further, the plots are not always generated and examined. This means that when a defect measurement is obtained it is not immediately traced back to a faulty installation and re-measurements are preformed. These unnecessary re-measurements could be avoided by inspecting the chamber for potential interference points before the testing began. Thus the developed application has the potential of saving both money and time.

6
Conclusion

In this paper we describe a method of applying object detection to locate interference points in Ericsson's anechoic test chambers and use this information to relay advice about how to improve the installation. Additionally, the paper presents results from implementing the methods into an application. The project was divided into two main parts. First, the creation of the datasets and training of the object detection models. Second, the creation of the application combining the functionality of the models with methods of using the detection results to relay relevant information to the user.

Two different datasets were created containing images and annotations from two differently sized anechoic chambers. They contain a total of 1700 and 730 images each which when applying transfer learning is more than enough training data. These dataset can be used as is or expanded upon in the future to allow for training models for further projects within Ericsson.

The resulting product is an android application containing two trained object detection networks. The detections achieve a highly interactive frame rate of 13 FPS which are overlaid over a video feed running at 60 FPS creating the illusion of greater speeds being achieved. The networks are able to detect and discern between the different objects with a high accuracy. However, a slight rotational bias is present for one of the networks. Further, the application is able to use the detection results to provide the user with potential improvements for the installation.

In the paper, models of different sizes are introduced, trained and evaluated. However, the final product only employ the smaller (SSD MobileNet V2 FPNLite 320x320) networks. This is due to the fact that the smaller network achieved both faster inference times and similar or better accuracy. For the A9 chamber the model achieved a maximum mAP score of 0.8765 and a maximum recall score of 0.7212. While for the A3 network the model achieved a maximum mAP score of 0.865 and a maximum recall score of 0.6997.

6.1 Future Work

To reduce training times as well as to achieve a higher accuracy score it is important to have a balanced dataset. An unbalanced dataset can lead to problems such as varying accuracy between the classes where the minority class is hard to to identify [63]. As seen in Figures 3.3 and 3.5 the datasets used were not balanced with the majority of the annotations belonging to Cabling for both chamber A9 and A3. Future work could therefore include balancing the datasets, training and evaluation new models on these datasets. The balancing can be done either by introducing more images containing the minority classes to the datasets or by introducing cost-sensitive training where making mistakes classifying minority classes results in a larger loss for the network. This forces the network to pay more attention to rare classes. Further, conducting a comparison between the models before and after the balancing could prove interesting.

The models can achieve higher speeds if we choose to quantize them reducing the numerical accuracy of the predictions and thus decreasing the accuracy. Quantization of the models was not applied in this paper therefore, studying the effects of it in terms of speed and accuracy could prove interesting for future works.

Another topic for future studies could be to compare a larger assortment of models or possibly to train the models used in this paper using additional data augmentation methods, extended datasets, longer training sessions, etc. and investigate whether an increase in performance at a reasonable frame rate is obtainable.

Bibliography

[1] M. Elgendy, *Deep learning for vision systems.* Simon and Schuster, 2020.

[2] Aphex34, "File:typical cnn.png." `https://commons.wikimedia.org/wiki/File:Typical_cnn.png`.

[3] "Attribution-sharealike 4.0 international." `https://creativecommons.org/licenses/by-sa/4.0/legalcode`.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[5] "Tensorflow 2 detection model zoo." `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`.

[6] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[7] B. Chung and H. Chuah, "Design and construction of a multipurpose wideband anechoic chamber," *IEEE Antennas and Propagation Magazine*, vol. 45, no. 6, pp. 41–47, 2003.

[8] D. Ribli, A. Horváth, Z. Unger, P. Pollner, and I. Csabai, "Detecting and classifying lesions in mammograms with deep learning," *Scientific reports*, vol. 8, no. 1, pp. 1–7, 2018.

[9] J. Hung and A. Carpenter, "Applying faster r-cnn for object detection on malaria images," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 56–61, 2017.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[11] R. Simhambhatla, K. Okiah, S. Kuchkula, and R. Slater, "Self-driving cars: Evaluation of deep learning techniques for object detection in different driving conditions," *SMU Data Science Review*, vol. 2, no. 1, p. 23, 2019.

[12] O. Alsing, "Mobile object detection using tensorflow lite and transfer learning," 2018.

[13] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. D. Iii, and K. Crawford, "Datasheets for datasets," *Communications of the ACM*, vol. 64, no. 12, pp. 86–92, 2021.

[14] F. L. INCORPORATED, "Object detection guide." `https://www.fritz.ai/object-detection/`.

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, pp. 886–893, Ieee, 2005.

[16] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *European conference on computer vision*, pp. 428–441, Springer, 2006.

[17] N. Dalal, *Finding people in images and videos.* PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2006.

[18] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[19] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.

[20] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[21] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[22] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2009.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[27] D. J. Finney, *Probit analysis: a statistical treatment of the sigmoid response curve.* Cambridge university press, Cambridge, 1952.

[28] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, pp. 1–6, Ieee, 2017.

[29] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[30] D. Yu, H. Wang, P. Chen, and Z. Wei, "Mixed pooling for convolutional neural networks," in *International conference on rough sets and knowledge technology*, pp. 364–375, Springer, 2014.

[31] M. Basavarajaiah, "Maxpooling vs minpooling vs average pooling." `https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95f`

[32] Y. Zhong, J. Wang, J. Peng, and L. Zhang, "Anchor box optimization for object detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1286–1294, 2020.

[33] Y. Li, C. Wei, and T. Ma, "Towards explaining the regularization effect of initial large learning rate in training neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[34] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[35] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 international conference on systems, signals and image processing (IWSSIP)*, pp. 237–242, IEEE, 2020.

[36] S. Yohanandan, "map (mean average precision) might confuse you!." `https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2`.

[37] J. Hui, "Ssd object detection: Single shot multibox detector for real-time processing." `https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9b`

[38] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*, pp. 1–10, 2018.

[39] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of big data*, vol. 6, no. 1, pp. 1–48, 2019.

[40] J. Solawetz, "Why and how to implement random rotate data augmentation." `https://blog.roboflow.com/why-and-how-to-implement-random-rotate-data-augmentation/`.

[41] J. Shijie, W. Ping, J. Peiyi, and H. Siping, "Research on data augmentation for image classification based on convolution neural networks," in *2017 Chinese automation congress (CAC)*, pp. 4165–4170, IEEE, 2017.

[42] H. M. Bui, M. Lech, E. Cheng, K. Neville, and I. S. Burnett, "Using grayscale images for object recognition with convolutional-recursive neural network," in *2016 IEEE Sixth International Conference on Communications and Electronics (ICCE)*, pp. 321–325, IEEE, 2016.

[43] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 13001–13008, 2020.

[44] S. Wenkel, K. Alhazmi, T. Liiv, S. Alrshoud, and M. Simon, "Confidence score: the forgotten dimension of object detection performance evaluation," *Sensors*, vol. 21, no. 13, p. 4350, 2021.

[45] G. Inc., "Tensorflow." `https://www.tensorflow.org/`.

[46] Google, "Api documentation." `https://www.tensorflow.org/api_docs/`.

[47] D. Demirović, E. Skejić, and A. Šerifović-Trbalić, "Performance of some image processing algorithms in tensorflow," in *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 1–4, IEEE, 2018.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[49] G. Inc., "Tensorflow lite." https://www.tensorflow.org/lite.
[50] Google, "Tensorflow lite model maker." https://www.tensorflow.org/lite/guide/model_maker.
[51] Google, "Object detection." https://www.tensorflow.org/lite/examples/object_detection/overview.
[52] Google, "Tensorflow lite converter." https://www.tensorflow.org/lite/convert.
[53] "Running on mobile with tensorflow lite." https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/running_on_mobile_tensorflowlite.md.
[54] "Adding metadata to tensorflow lite models." https://www.tensorflow.org/lite/convert/metadata.
[55] A. Studio, "Meet android studio." https://developer.android.com/studio/intro, accessed = 2022-03-16.
[56] F. Lardinois, "Google makes kotlin a first-class language for writing android apps." https://techcrunch.com/2017/05/17/google-makes-kotlin-a-first-class-language-for-writing-android-apps/, accessed = 2022-03-16.
[57] A. Studio, "Camerax." https://developer.android.com/training/camerax.
[58] A. Studio, "Image analysis." https://developer.android.com/training/camerax/analyze.
[59] Y.-F. Ou, T. Liu, Z. Zhao, Z. Ma, and Y. Wang, "Modeling the impact of frame rate on perceptual quality of video," in *2008 15th IEEE International Conference on Image Processing*, pp. 689–692, IEEE, 2008.
[60] T. A. Funkhouser and C. H. Séquin, "Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pp. 247–254, 1993.
[61] V. Rodriguez, "Basic rules for anechoic chamber design, part one: Rf absorber approximations," *Microwave Journal*, vol. 59, no. 1, pp. 72–86, 2016.
[62] T. D. Lin, "labelimg." https://github.com/tzutalin/labelImg, 2022.
[63] M. M. Rahman and D. N. Davis, "Addressing the class imbalance problem in medical datasets," *International Journal of Machine Learning and Computing*, vol. 3, no. 2, p. 224, 2013.

A
Appendix

```
for (x in cables) {
    for (y in otherComponents) {
        z = x.intersect(y)
        if (z != null){
            for (p in L1){
                q = z.intersect(p)
                if (q != null){
                    L2.add(q)
                }
            }
            L1.add(z)
        }
    }
    var overlap = 0f
    for (a1 in L1) {
        overlap += getArea(a1)
    }
    for (a2 in L2) {
        overlap -= getArea(a2)
    }
    val area = getArea(x)
    return overlap/area
    L1.clear()
    L2.clear()
}
return overlap
```

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

AVANCEZ
1829

CHALMERS
UNIVERSITY OF TECHNOLOGY