UPPSALA
UNIVERSITET

UPTEC F 18028
Examensarbete 30 hp
27 Juni 2018

Object Detection in Infrared Images using Deep Convolutional Neural Networks

Markus Jangblad

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Object Detection in Infrared Images using Deep Convolutional Neural Networks

Markus Jangblad

In the master thesis about object detection(OD) using deep convolutional neural network(DCNN), the area of OD is being tested when being applied to infrared images(IR). In this thesis the, goal is to use both long wave infrared(LWIR) images and short wave infrared(SWIR) images taken from an airplane in order to train a DCNN to detect runways, Precision Approach Path Indicator(PAPI) lights, and approaching lights. The purpose for detecting these objects in IR images is because IR light transmits better than visible light under certain weather conditions, for example, fog. This system could then help the pilot detect the runway in bad weather. The RetinaNet model architecture was used and modified in different ways to find the best performing model. The models contain parameters that are found during the training process but some parameters, called hyperparameters, need to be determined in advance. A way to automatically find good values of these hyperparameters was also tested. In hyperparameter optimization, the Bayesian optimization method proved to create a model with equally good performance as the best performance acieved by the author using manual hyperparameter tuning. The OD system was implemented using Keras with Tensorflow backend and received a high perfomance (mAP=0.9245) on the test data. The system manages to detect the wanted objects in the images but is expected to perform worse in a general situation since the training data and test data are very similar. In order to further develop this system and to improve performance under general conditions more data is needed from other airfields and under different weather conditions.

Handledare: Joakim Lindén
Ämnesgranskare: Juozas Vaicenavicius
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F18028
Tryckt av: Uppsala

Populärvetenskaplig sammanfattning

Deep Convolutional Neural Network (DCNN) har de senaste 6 åren kommit att bli det stora verktyget för bildbehandling när det gäller objektdetektering i bilder. Detta examensarbete undersöker hur man kan använda DCNN för att detektera landingsbanor och landingslampor i infraröda bilder tagna från ett flygplan som går in för landning på en flygplats. Anledningen till att detta kan vara av intresse kommer från att infrarött ljus transmitteras bättre än synligt ljus i vissa väderförhållanden, till exempel lätt dimma. Ett objektdetekteringssystem skulle då kunna användas som ett hjälpmedel för piloter för att detektera landingsbanor och lampar vid dålig sikt då det är ett krav att ha visuell kontakt med dessa objekt för att få tillstånd att landa.

För att skapa en modell användes bilder tagna från en långvågsinfraröd (LWIR) kamera och en kortvågsinfraröd (SWIR) kamera som har filmat totalt 12 inflygningar till en flygplats för att skapa ett dataset. Det finns flera olika typer av DCNN-modeller för att bygga ett objektdetekteringsnätverk och för detta projekt valdes one-stage detektorn RetinaNet som sedan konfigurerades på olika sätt eftersom modellen från början är designad för att utföra objektdetektering i färgbilder. Färgbilder består av tre kanaler (rött, grönt och blått) medan de infraröda bilderna endast har en. Därför testades olika varianter på hur första lagret i modellen skulle modifieras. Ett försök i att kombinera LWIR och SWIR för att ge modellen maximalt med information testades. De modeller som användes innehåller miljoner av parameterar som behöver bestämmas och detta görs under en träningsprocess då bilddatan delas upp i två delar där modellen får träna på träningsdata och evalueras på testdata. Dock finns parameterar som inte kan bestämmas under träningsprocessen utan måste bestämmas manuellt, dessa kallas hyperparametrar. I detta examensarbete undersöks även några metoder för att automatiskt bestämma hyperparametrar, kallat hyperparameteroptimering. Mer ingående görs en praktiskt undersökning av bayesisk optimering med en Gaussisk process-modell skapad med information från tidigare tester och med hjälp av en acquisition-function tas ett beslut hur värdena på hyperparameter ska förändras för att sedan testa en ny modell. Från resultaten framgår att en sådan metod klarar av att skapa en modell som presterar lika bra som det bästa nätverket som författaren lyckades skapa genom att manuellt bestämma hyperparametrarna.

Från de olika modeller som testades fanns att den bästa modellen lyckades detektera objekt på ett önskat sätt. Dock kvarstår fortfarande frågan kring hur generell modell som har skapats då träningsdatan och testdatan är väldigt lika eftersom all data är hämtad från liknande väderförhållanden på inflygningar till samma flygplats. Därför är behovet av mer data stort för att kunna skapa en mer generell modell och dessutom kunna garantera en viss prestanda. Speciellt intressant vore data taget under sämre väderförhållande som dimma då detta är ett av de tänkta applikationsområderna. Noterbart var också att den bästa modellen inte var den som använde både LWIR och SWIR för objektdetektering utan istället den som använde en bild åt gången vilket talar emot att maximalt information ger bäst resultat.

Detta kan bero på att LWIR och SWIR bilderna inte är helt synkade vilket leder till att objekten inte hamnar på samma plats i de båda bilderna. En annan förklaring kan ligga i att de vikterna som användes var tränade för tre färgkanaler och inte för två olika infraröda kanaler.

Acknowledgements

There are some people I would like to mention for helping me with this master thesis. I would like to thank my supervisor Joakim Lindén at Saab for always being available when advise was needed and for valuable input. The support and a genuine interest in my work have helped me a lot.

I would also like to thank my subject reader Dr Juozas Vaicenavicius at the IT department, division of systems and control at Uppsala University for valuable feedback and good brain-storming sessions. Our meetings has always been precious time and a vital part of the progress of the thesis. Also thanks for opening my eyes for Bayesian optimization.

Moreover I would like to thank Erasmus Cedernaes at Saab for always offering support when it comes to different computer issues or just offering some python expertise. An sxtra acknowledgement goes out to Per Emilsson at Saab for being second reader of this thesis and providing valuable feedback getting this report to where it is now.

Contents

1 Introduction **6**
- 1.1 Background . . . 6
- 1.2 Objectives . . . 7

2 Theory **7**
- 2.1 Deep Convolutional Neural Networks . . . 7
  - 2.1.1 Layers . . . 8
  - 2.1.2 Loss Functions . . . 11
  - 2.1.3 Training procedure . . . 13
  - 2.1.4 Regularization . . . 15
- 2.2 Model architecture . . . 16
  - 2.2.1 One-Stage Detectors . . . 16
  - 2.2.2 Two-Stage Detectors . . . 17
  - 2.2.3 Mobilenet . . . 17
  - 2.2.4 ResNet . . . 17
  - 2.2.5 Fully Convolutional Network . . . 18
  - 2.2.6 RetinaNet . . . 18
  - 2.2.7 Transfer learning . . . 20
- 2.3 Hyperparameter optimization . . . 20
  - 2.3.1 Grid search . . . 21
  - 2.3.2 Random grid search . . . 21
  - 2.3.3 Bayesian Optimization . . . 21
- 2.4 Performance metrics . . . 23
  - 2.4.1 Confusion matrix . . . 23
  - 2.4.2 Precision . . . 24
  - 2.4.3 Recall . . . 24
  - 2.4.4 Mean Average Precision (mAP) . . . 24

3 Task **25**
- 3.1 Infrared Images . . . 25
- 3.2 COCO dataset . . . 27
- 3.3 Infrared dataset . . . 27

4 Method **28**
- 4.1 Training on the COCO dataset . . . 29
- 4.2 Training on infrared dataset . . . 29
  - 4.2.1 Converting a network from RGB to IR-images . . . 30
  - 4.2.2 Changing backbone network . . . 31
- 4.3 Bayesian Optimization . . . 32

5 Results **32**
5.1 Training on COCO dataset . . . 33
5.2 Training on IR images . . . 33
5.2.1 Different IR-Datasets . . . 34
5.2.2 Input image size . . . 34
5.2.3 Data augmentation . . . 35
5.2.4 CLAHE . . . 35
5.2.5 Different input layers . . . 35
5.2.6 Different backbone network . . . 36
5.3 Hyperparameter optimization . . . 36

6 Discussion **37**
6.1 The model trained on the COCO dataset . . . 37
6.2 The infrared dataset . . . 37
6.2.1 Prediction time vs. performance . . . 37
6.2.2 Lowering the prediction time . . . 37
6.2.3 Different input layers . . . 38
6.3 Hyperparameter optimization . . . 38

7 Conclusion **39**

8 Further work **39**

1 Introduction

The area of computer vision deals with the goal of creating systems that can process images just like humans, or even better. The computer vision tasks can be divided into three different areas. The ability to sort images into different classes depending on the content is called *image classification*, seen in figure 1. The ability to identify multiple objects in images and determine their location by putting a bounding box around the object is called object detection, seen in figure 2. A more difficult task is not only to put a bounding box around objects in images but to classify every pixel in the image, this is called semantic segmentation and is shown in figure 3.

Figure 1: Example of image classification

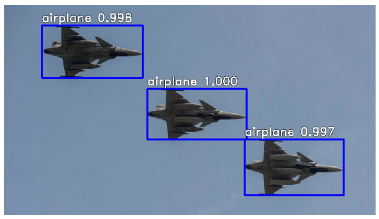

Figure 2: Example of object detection

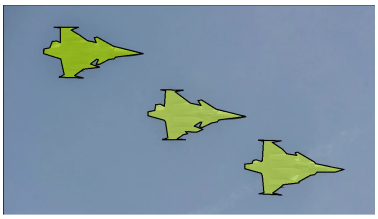

Figure 3: Example of segmentation

There are many different methods of implementing computer vision and lately it has been deep convolutional neural networks (DCNN) that have performed best in these tasks for the last 6 years[1]. Today convolutional nerual networks (CNN) is an active field of research and new research progress is constantly released. In Microsoft Common Object in Context object detection challenge (COCO) the best perfomance has increased with 20% in two years[2].

1.1 Background

Deep learning is an area of machine learning that has grown dramatically over the last years together with the capacity of Graphics processing units(GPU) and is today widely used in different areas, for example, email filtering, speech recognition and marketing purposes. A popular area where machine learning and especially deep learning is applied is computer vision. When it comes to computer vision, DCNNs are most often used ever since this type of model won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2012 by a model named AlexNet, [3]. The competition is to classify 150.000 images into 1000 different categories and has been won by a DCNN ever since.

This master thesis will focus on presenting and explaining DCNNs as well as applying them and investigating their performance in object detection tasks. In the beginning, an already trained standard DCNN will be used in order to examine the performance on a GPU in a situation where a lot of training data is available. Next we look at a more specific task in which the goal is to detect different visual references from an image taken from an airplane with an infrared camera. A visual references can be the runway or the approaching lights

and the pilot is required to have visual contact with one of the required visual reference at a certain altitude in order to be allowed to land on the runway. This alititude is called the decision height [4], [5]. The use of infrared images can then be useful for detecting these objects when the weather conditions are bad and an airplane is approaching an airport. This task is different from the previous because we do not have an already existing network, we have infrared instead of RGB images instead, and the amount of data is more limited.

In order to perform object detecting from airborne systems, images need to be processed at a high frame rate since the object detection needs to be performed in real time. Therefore the time for a model to make a prediction based on an image is of great interest.

1.2 Objectives

The master thesis was conducted at Saab Avionic Systems, Engineering Department, Video & Graphics and the objectives are

- Literature study on DCNN for image classification and object detection with a goal to present and explain pros and cons of different types of neural network architectures.
- Literature study on how the selection of the hyperparameters of a model can be optimized.
- Practical study on the object detection performance of a state-of-the-art DCNN trained on a GPU, Nvidia Titan Xp.
- To adjust and retrain the model on infrared images taken from an airplane with an infrared camera in order to detect essential landmarks for approaching an airfield: runway, approaching lights and PAPI lights.
- Study the performance of the new model.
- Use a method for optimizing the hyperparameters of the new model
- Compare the performance of the standard model and the one with optimized hyperparameters.
- Examine the prediction time of different network architectures.

2 Theory

2.1 Deep Convolutional Neural Networks

The basic Convolutional Neural Network (CNN) idea comes from looking at how our brain works with neurons connected to other neurons where one neuron can activate another neuron, [6]. In the same way, a CNN is built. Neurons are organized in layers connected

to other layers of neurons. How these neruons are connected depends on the structure of each layer and each layer has a specific function(i.e Convolutional layer, Pooling layer, Flattening layer, etc.). The way of combining these different types of layers to perform a specific task is called a neural network and for image processing it is common to use many convolutional layers, therefore the name: *Convolutional Neural Network*. The first layer of an image processing network, called the *input layer*, takes the pixel values from an image as an input and the last layer, the *output layer*, gives you a result for the task you want to perform. If the task is to classify an image into one of ten different classes the output layer would consist of a vector of length 10 with probability of the image being classified in each class. The layers between the input- and the output-layer are called *hidden layers* and a CNN with many hidden layers is called a Deep CNN. An example of a deep CNN can be seen in figure 4.

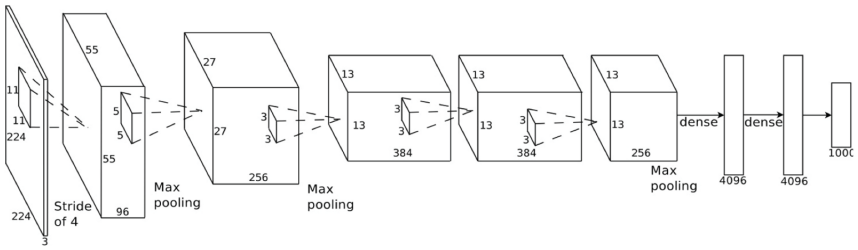

Figure 4: An example of a deep convolutional neural network where each block contains of many convolutional layers [7].

2.1.1 Layers

Convolutional Layer

The convolutional layer is based on the convolution operation

$$s(t) = \int x(a)w(t-a)da = x * w \tag{1}$$

where x is convolved with a function w. This can be done in a similar way for images in a 2D-convolution where an input image is convolved with a kernel, visualized in figure 5.

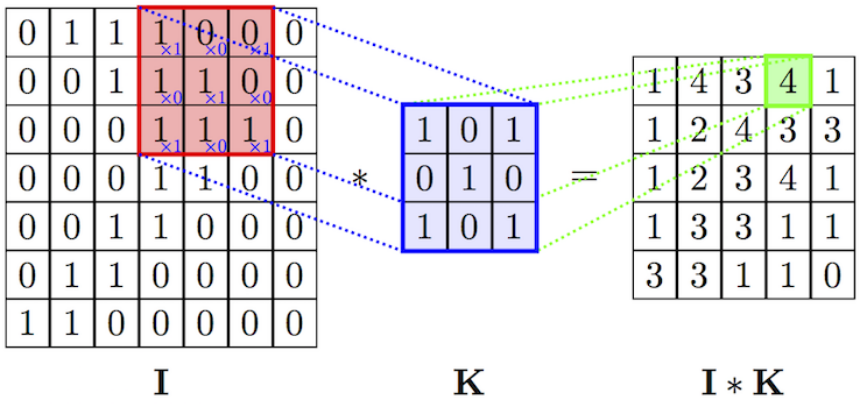

Figure 5: An example of a convolution layer with a kernel of size 3x3 that convolves over a 2D-input creating a 2D-output [8].

This kernel has a size that is specified in the model architecture by the creator of the model. This kernel will then "scan the input" spatially and at each location yield an output value. Each kernel, or filter as it is alternatively called, can be seen as a feature detector that scans the image for specific features, i.e different type of shapes, edges, circular patterns or specific color combinations. By combining many of these kernels the output will be a feature map telling something about what features were found in an image. Another hyperparameter needed for layers with a filter is the stride with which the filter moves spatially over the input, which affects the size of the output. The formula for the convoltional layer are

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \cdot I_{x+i-1,y+j-1} \tag{2}$$

where I is the input and K the kernel as in figure 5.

Pooling Layer

The pooling layer is used to decrease the size of the data to make computations easier in the network and also to prevent overfitting. The pooling layer uses filters with a certain size (2x2 is most common,[9]) to go through the input and use the max-function for each position of the filter, see figure 6. Average-pooling can also be used as a substitute to max-pooling.

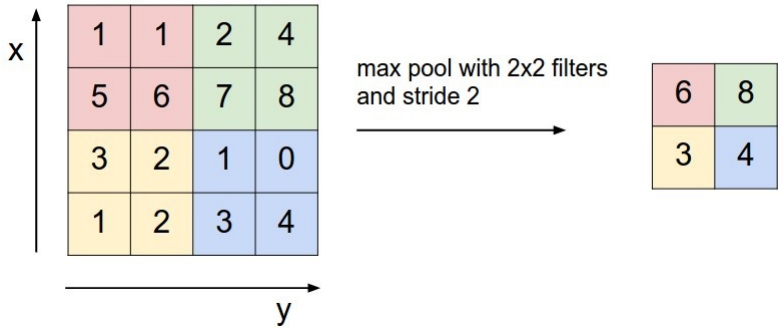

Figure 6: An example of a max pooling layer with 2x2 filters and stride 2, where the output is the max value from each filter[9].

Activation Functions

While a Deep CNN has many different types of layers connected to each other, it also contains non-linear activation functions. The non-linear activation functions are needed for the model to be able to capture variations and characteristics in the data that a simple linear model can't do. There are many different activation functions but the most commonly used are the sigmoid function and Rectified Linear Unit(ReLU). The sigmoid function can be seen in figure 7 and has an output that is always in the range of [0,1].

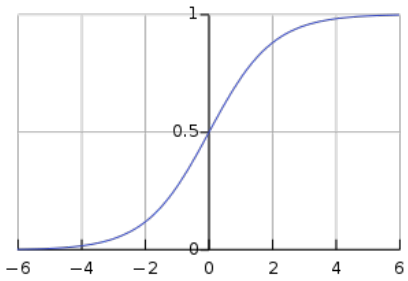

Figure 7: The sigmoid function[9].

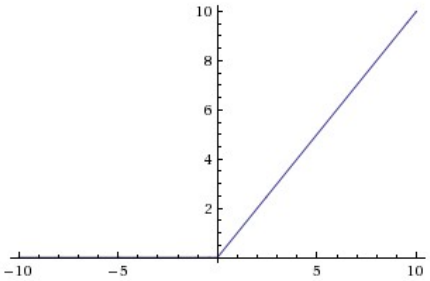

Figure 8: The ReLU function[9].

The downside with sigmoid functions is that it saturates and therefore the gradient will go to zero and this is not desireable when training a network. More common choice in image applications is the ReLU function in figure 8. Here the gradients will not saturate for large inputs but the ReLU comes with the risk of vanishing gradient problem during training which means that they will not be activated again by any data. This problem can be solved by changing the learning rate of the model, [9]. There is also a modified ReLU, called Leaky ReLU, with the purpose to avoid the vanishing gradient problem of the ReLU but the standard ReLU is still used more often, because of computational simplicity.

Fully Connected layer

The fully connected layer can be seen in figure 9 and is a layer in which every neuron is connected to all the neurons in the output.

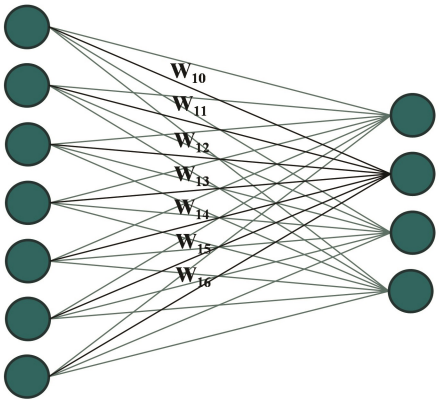

Figure 9: An example a fully connected layer.

Every neuron connection is defined by a weight that is a parameter that will be learned by the system. In figure 9, the second output neuron on the right side is connected to each and every input neuron on the left by the weights $W_{10} - W_{16}$ so the value of the neuron can be calculated as

$$\sum_i W_{1i} x_i + b \tag{3}$$

where x_i is the input value of the neuron and b is a bias. The values of the weights and biases are determined during the training process. The fully connected layers are most common at the end of an image-processing network in order to connect all the features in an image to generate a score for each possible object class, telling us the probability of the image containing that object.

Classifiers
At the end of the network, a classifier is needed to determine how the model should classify the input and create an output from the system that can be interpretable. Two common examples of classifiers are the Support Vector Machine and the Softmax classifier. The classifier are formulated in different ways depending on the task to be performed.

2.1.2 Loss Functions

When an image is passed through a network a prediction is created. The prediction is compared to the correct answer, defined by the dataset, called the groundtruth. The difference between the prediction and the groundtruth results in a loss and the goal with the training process is to minimize this loss. The goal is to minimize the expected value of the loss from a model and in the case of the cross-entropy loss, the expected loss is approximated as

$$\mathbb{E}[-\log p_i] \approx \frac{1}{n}\sum_{i=1}^{n} -\log p_i = \frac{1}{n}\sum_{i=1}^{n} L_i \tag{4}$$

where L_i is the loss for one training example and the total loss L is approximated as the mean over all examples. How the loss is calculated depends on how the loss function is defined. One of the most common loss functions is the cross-entropy loss in 4. This loss function is useful for image classification tasks but different tasks need different loss functions. For example, in the detection problem in which bounding boxes are estimated around the objects, a regression loss function can be used to get a measure on how well the bounding box is placed in the image.

Cross-entropy
The cross-entropy loss is used when a model contains the softmax classifier. The softmax classifier gives a probability score for each object class. The loss functions is calculated as

$$L_i = -\log\left(\frac{f_{y_i}}{\sum_j f_j}\right) \tag{5}$$

where f is the softmax function and y_i is the output for the correct class.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{6}$$

with z as a real valued score vector.

Regression loss
The regression loss computes the loss as the squared norm of the difference between the true value and the predicted value.

$$L_i = \|g - y_i\|_2^2 \tag{7}$$

where g are the predicted values and y_i the true ones. This loss function can be used when the goal is to find the coordinates of a bounding box when performing object detection.

Loss functions for object detection
When calculating the loss in image classification tasks, it is easy to define what is a correct classification and what is an incorrect classification. When it comes to object detection in images it is not as trivial. For object detection, the system needs to also put a bounding box around objects in the image and therefore one needs to define how close to the ground truth the predicted bounding box location should be. Therefore we need to use the metric Intersection over Union (IoU). The IoU is a measurement of how much the predicted box correlates with the ground truth box and is calculated as

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion} \tag{8}$$

,see figure 10.

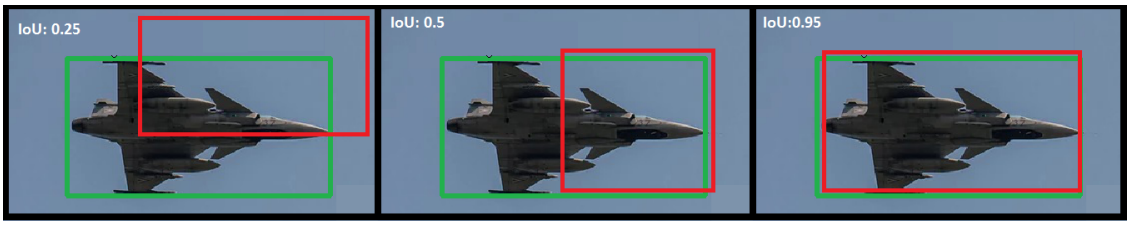

Figure 10: Three examples of when the predicted red box and the green groundtruth box has an IoU=0.25, IoU=0.5 and IoU=0.95.

An IoU threshold value is set to define what counts as a correct prediction in object detection. The loss function for object detection can be subdivided into two loss components. The classification loss and the regression loss, which handles the error in the bounding box coordinates.

Focal loss
The focal loss originates from the cross entropy loss and adresses one of the problems with one-stage detectors: the big imbalance between detecting objects and detecting background. This imbalance is due to the anchor approach that one-stage detectors use(see section 2.2.1). In one image around 100k anchors can be applied and the major of these anchors are often easy background classifications and only a few hard object classifications. The focal loss function wants to focus on the hard examples when calulating the loss and not as much on

the easy ones and is formulated as

$$\mathrm{FL}(p) = -\alpha(1-p)^{\gamma}\,\log(p) \tag{9}$$

where p is the probability of a box being classified to a specific class from the softmax function as in equation 6. α is a weighting factor in the range of $[0, 1]$ and γ is a focusing parameter which controls the contributions to the loss from easy and hard examples and counteracts class imbalance. A typical value for γ is $\gamma = 2$. $\gamma = 0$ and $\alpha = 1$ corresponds to a normal cross-entropy loss. When training, if an anchor have an IoU above 0.5 with a groundtruth box it is assigned to the object with a one-hot vector. If IoU is less than 0.4 it is assigned to background in order to know how to calculate the loss[17].

2.1.3 Training procedure

The different layers in a Deep CNN contain a set of parameters that needs to be tuned in order for the network to be able to learn a wanted task. This is done during the training process where the network is trained on a dataset. The dataset needs to contain many different images of the objects that the system will be used to identify. Often over 1000 images are avaibable per object class (i.e COCO2017 dataset contains 118.000 images with 80 different object classes, [10].) The data is then divided into three groups: training, validation and test. The network is trained on the training data and during the training process, continuously checked against the validation data to see if the network can do well on data it has never seen before. The validation set is used to avoid overfitting, when the network only manages to perform well on the training data. When satisfactory results are achieved on both the training and the validation data, the network is evaluated on the test data to get a final result of the performance of the network. The division of the dataset will typically be 80% training data 15% validation data and 5% test.

Backpropagation

When a batch of training images has been passed through the model and a loss has been calculated the parameters of the model are updated to improve the model by using backpropagation. Backpropagation calculates the gradient for each parameter in the network by starting from the calculated loss and then propagating back through the network. A simple example with one neuron can be seen in figure 11, where the incoming gradient from the connection z between two neuron is propagated back through the neuron to the connections x and y by applying the chain rule. This will then give gradients for each of the parameters in the network and can be updated using an optimization technique.

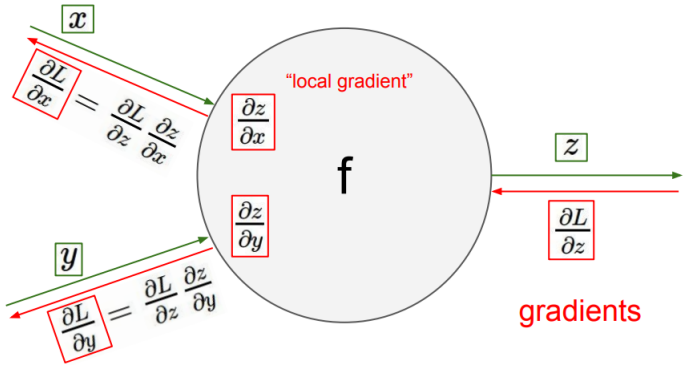

Figure 11: An illustrative example of backpropagation through a node, where L is the calculated loss[9].

Optimization

The goal of the training process is to find parameter values that optimize the performance of our network. This is done by the loss on the training set, by updating the values of the parameters by using the gradient calculated in the backpropagation. There are different approaches for this optimization that are all based on the gradient calculated using back-propagation. Stochastic gradient descent updates the parameters in the direction of the gradient with a fixed step size called learning-rate, that needs to be determined by the designer. To converge to a good local minimum faster, a momentum term can be introduced that can pass saddle point at shallow local minima in order to find a good local minimum. A popular optimizer is the Adam optimizer[11] that uses the momentum technique and an adaptive learning-rate for every parameter instead of adjusting the learning-rate overall. In all the different type of optimizers, a clipnorm can be introduced in order set a maximum norm of the gradients to a specific value.

Overfitting

A common problem when training a neural network is overfitting. Overfitting occurs when a model performs very well on the data it was trained on but performs bad on data it has never seen before. The model has specialized on the training data instead of created a generalized model. An example of overfitting can be seen in figure 12.

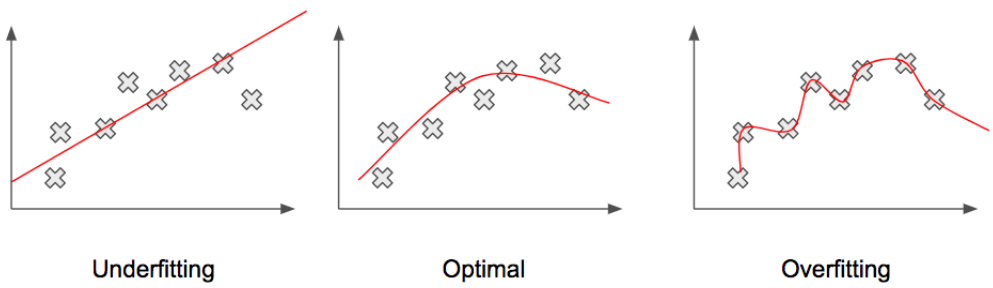

Figure 12: Three examples of estimating a function to data, one underfitted, one optimal and one overfitted function[12].

To avoid overfitting one can use regularization. Too much regularization can result in under-

fitting instead, meaning the model is not able to perform well on any data. Another method to avoid overfitting is data augmentation. This means that the images can be augmented in different ways, for example horizontal and vertical flips, rotation, scaling, and color inversion. This creates a more versatile dataset and can be very helpful when you have a limited amount of data in your dataset.

Preprocessing
Data augmentation is one part of the different kinds of preprocessing that can be applied to the data before starting to train a network in order to improve the results. The mean of the data is also removed in order to get a zero mean dataset. In images the mean pixel value over all images is subtracted from all pixels to get all input data symmetric around zero. A general image preprocessing tool when you have images with low contrast is constrast limited adaptive histogram equalization(CLAHE). CLAHE divides the image into different sections and calculates histrograms for every section and adjusting the brightness of the pixels in that section[13], [14].

2.1.4 Regularization

In order to prevent a model from overfittning to the training data during the training process different regularization techiques can be used. A dropout layer, shown in figure 13, can be applied to the model. A connection between two neurons will be randomly dropped with a fixed probability.

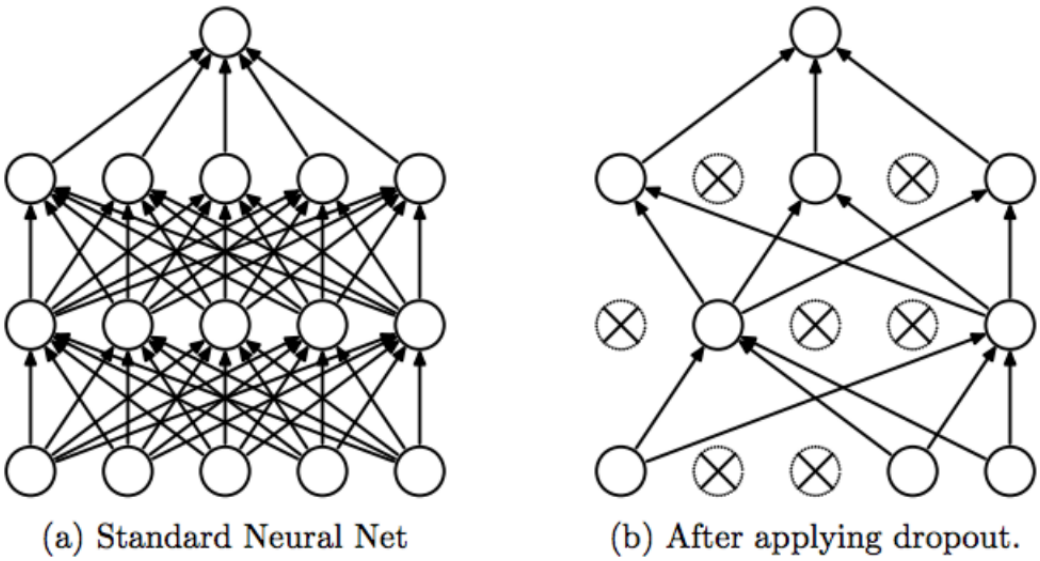

Figure 13: The effect of a dropout layer. To the left we see a standard neural networks and to the right the same network with dropout applied[9].

This forces the model not to rely on a single node and be more robust. After the model has been trained, the dropout layer is no longer applied, leading to a deterministic model.

The probability that a node will be dropped is controlled by a dropout probability that is determined when designing the network. Other popular regularization methods are $L2$- and $L1$- regularization which prevent the parameters to fit perfect to the training data, thus avoiding overfitting.

2.2 Model architecture

The way of combining layers into a model depends on the task to be performed and the typical model architecture has been changing and new ways of combining layers are constantly released in order to find the best model since AlexNet won ILSVRC in 2012. Just comparing the model architecture of AlexNet consisting of 8 layers with the winner of the same competition three years later, ResNet (ILSVRC winner 2015), with its 152 layers one notices the big difference in architecture. When performing object detection tasks, the model architectures can be divided into two different families: one-stage detectors and two-stage detectors.

2.2.1 One-Stage Detectors

The one-stage detectors use a sliding window approach across the image and applies *anchors* at several locations of feature maps created by the network. This is illustrated in figure 14.

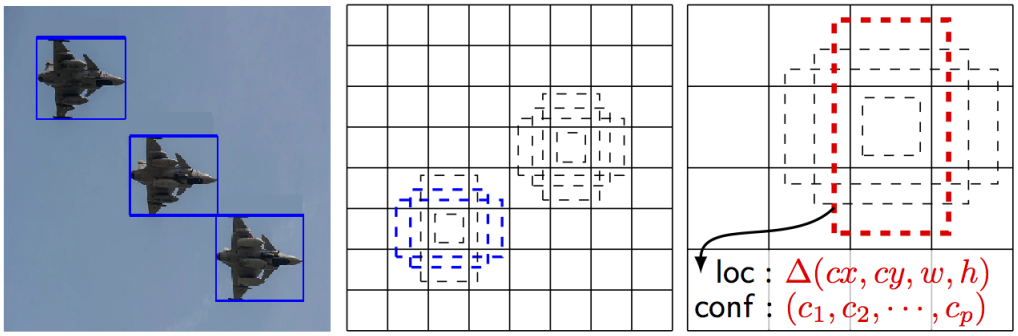

Figure 14: Shows how different feature maps are divided into different spatial location and at each and every location anchors are assigned[15].

These anchors are boxes with fixed scales and ratios to capture objects of different shapes and sizes. If three different scales and three ratios are being used a total of 9 anchors are assigned to each location. The anchor scales and ratios are hyperparameters. The anchors are then classified to create predictions and the coordinates of the box are regressed. The output from each anchor is a classification score and bounding box coordinates and on one image, between 40k-100k anchors can be applied[17]. To avoid detecting the same object mutliple times non-max supression is used which groups overlapping detections of the same objects into one detection, see figure 15.

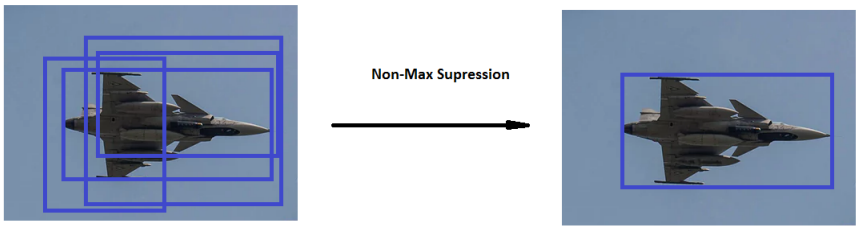

Figure 15: Example of before and after non-max supression is applied to an image, resulting in only one prediction.

Example of popular one-stage detectors are Yolo[16], RetinaNet[17] and SSD[18].

2.2.2 Two-Stage Detectors

The two-stage detector instead uses two networks. One network, called region proposal network, finds interesting regions in the image where objects are most likely to be found and then a second network to classify these regions and output the object class it belongs to and bounding box coordinates. Example of two stage detectors are the R-CNN[19], fast R-CNN[20] and faster R-CNN[21]. The two-stage detectors usually have better performance than one-stage detectors but have a higher computational time although recent papers have been suggesting that the gap between one-stage and two-stage are shrinking when it comes to prediction time[22].

2.2.3 Mobilenet

In 2017 Mobilenet was introduced which is an image classification network that aimed to create small image classification networks targeting mobile devices, have a short prediction time and still achieve high accuracy[23]. Mobilenet uses depthwise separable convolutional layers instead of normal convolutional layers in order to decrease the computational cost of the network. The depthwise separable layer divides the traditional convolutional layer into two layers. First it applies depthwise convolutional filter on each input channel and then uses a 1x1 convolution to combine the depthwise convolutional filter. This reduces the computational cost compared to a normal convolutional layer with

$$\frac{1}{N} + \frac{1}{D_k^2} \tag{10}$$

where N is number of filters and D_k is the size of the kernel[23].

2.2.4 ResNet

Another popular image classification network is the Residual Net (ResNet), developed in 2015. ResNet has different types of configurations depending on the wanted size of the

model, i.e. ResNet50, ResNet101 and ResNet152 have 50, 101 and 152 layers respectively. The uniqueness of ResNet is the residual building block shown in figure 16.

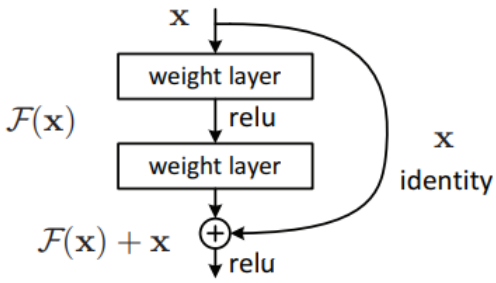

Figure 16: The layout of the residual block.[24].

The great success with the residual block was that it enabled deeper networks without any accuracy degradation problem that can occur with deep networks [24].

2.2.5 Fully Convolutional Network

In the standard neural network there is usually a fully connected layer at the end of the network. The idea behind a fully convolutional network (FCN) is to only apply convolutional layers with different amount of filters in order to achieve a certain output. The advantage with a FCN compared to using fully connected layer is that a FCN can handle different input sizes.

2.2.6 RetinaNet

The RetinaNet is one of the popular one-stage detectors to refer to as a state-of-the-art network [25] and the model architecture of RetinaNet consists of three parts: a backbone network, a feature pyramid network and classification and regression networks all combined into one network. The backbone of the network is usually a part of a pretrained image classification network, for example ResNet or Mobilenet pretrained on ImageNet images. The backbone network is used to extract features from the image and with the help of the feature pyramid network creating feature maps. Anchors are then applied to each feature maps, see figure 14, and the classification and regression network performs classification and bounding box regression on all the anchors. The output of the network will be a classification vector for every anchor from the classification network and 4 bounding box coordinates per anchor from the regression network. The structure of the RetinaNet can be seen in figure 17. Non-max supression is then use to create final detections.

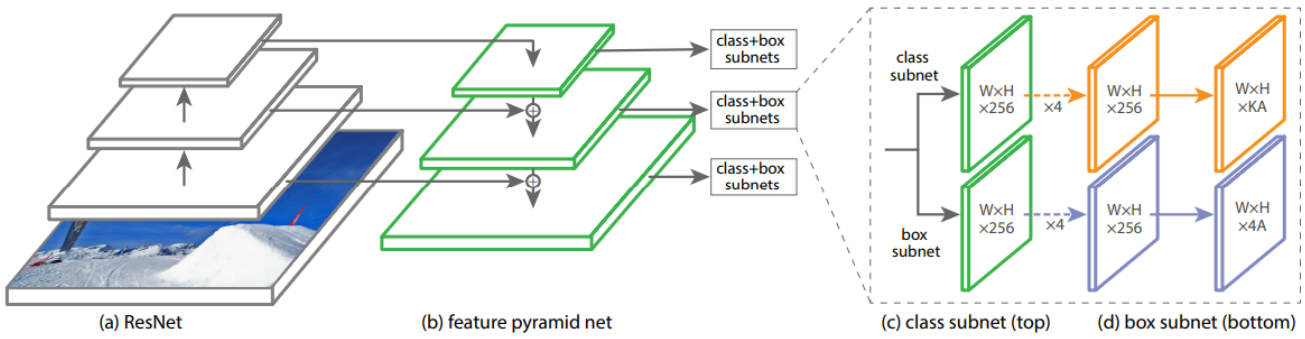

Figure 17: The model architecture of RetinaNet [17].

The regression network and classification network are both fully convolutional network where the classification network uses a sigmoid activation at the end as a classifier and the regression network has a bounding box regressor to predict the box coordinates. The number of parameters in the network depends on what backbone network is used. The RetinaNet with ResNet50 (Resnet with 50 layers) as backbone contains 36M parameters while the RetinaNet with Mobilenet as backbone has 13M parameters. The RetinaNet uses the focal loss as loss function with hyperparameters $\alpha = 0.25$ and $\gamma = 2$ for classification and regression loss for bounding box regression.

Feature Pyramid Network

From the backbone network in RetinaNet we extract feature maps from three different levels of the network in order to detect objects at different scales and different low and high level features in an image, shown in figure 17a). The extracted feature maps have different resolution and contains different level of features. The feature map with highest resolution contains low level features and the feature map with lowest resolution contains high level features. High resolution is needed in order to detect small objects and high level features are needed to find complex objects. The feature pyramid network uses different feature maps and upsample and combine them in a so called "top-down pathway with lateral connections" according to figure 18 with an upsampling layer and a 1x1 convolutional layer.

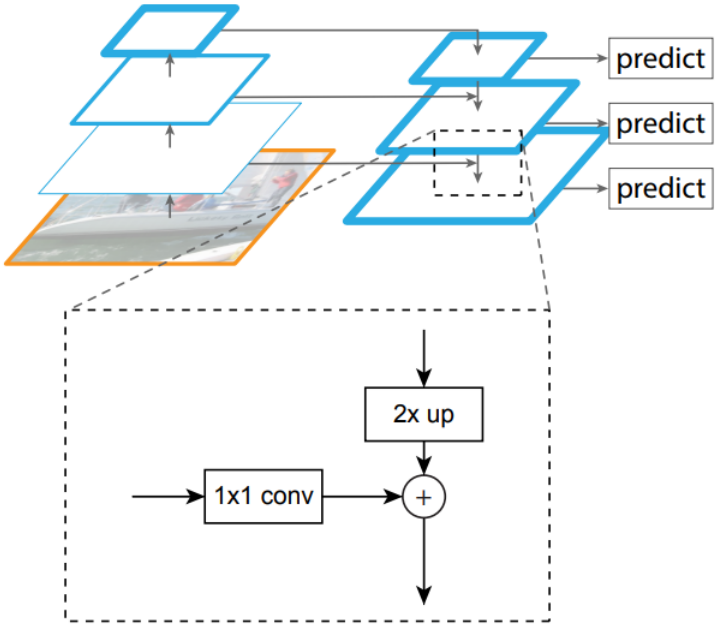

Figure 18: The architecture of the feature pyramid network where the backbone layer is added through a 1x1 convolutional layer with a higher feature map that is passed through an upsampling layer creating three new feature maps that are combinations of the old feature maps[26].

By combining the different different feature maps in a feature pyramid network, new feature maps are created. The new feature maps are combinations of high and low features and different resolutions. This results in stronger feature maps containing both information about high level features at a higher resolution, leading to better object detecting results[26].

2.2.7 Transfer learning

When image classification and object detection tasks are similar to previously performed tasks, one can use the trained weights used in that model and only train the last layers of the classifier to adjust the model to detect new objects. Models trained on the dataset Imagenet are common to extract the weights from and are used for transfer learning since Imagenet contains over 14 millions annotated images for image classifications in over 20 thousand classes[27]. Transfer learning is often used for computer vision since features in images are often similar, indipendent on what objects that are to be detected. Transfer learning can also reduce the training time of a network if the pretrained weights do not need to be modified. In RetinaNet transfer learning is applied when using for example ResNet as backbone network with Imagenet trained weights.

2.3 Hyperparameter optimization

In every neural network, there are many settings and choices that cannot be trained and learned by the system; these are called hyperparameters. Examples of hyperparameters are

the learning rate of the training optimizer, the model architecture and the regularization parameters. In CNN we also have stride parameters in the convolutional and pooling layers. The wish to find a way to automatically choose the hyperparameters is an active research area.

2.3.1 Grid search

An intuitive way of finding the optimal setting of hyperparameters would be to test every combination of hyperparameters and choose the model that performs best. This method is called *grid search*. This is the safest way to optimize the hyperparameters but has the major drawback of increasing computations as $\mathcal{O}(k^n)$ where n are the number of hyperparameters with k possible values. With many hyperparameters and a large network where each model can take days to train, this method is not feasible.

2.3.2 Random grid search

An alternative to grid search is doing a *random grid search*. In a random grid search you randomly choose different settings of hyperparameters within a domain for several interations and choose the setting that performs the best. This has been resulting in good models, [28]. The advantage of random grid search and grid search is that they can be run on parallel GPUs since each iteration is not depending on the result from the previous iteration.

2.3.3 Bayesian Optimization

A more sofisticated way of optimizing hyperparameters is called *Bayesian optimization* and has been becoming a popular way of automatically choose the best set of hyperparameters[29]. Bayesian Optimization is good to use when it is computationally unfeasible to use grid search when for example you have a large number of hyperparameters or each model take hours/-days to train. The Bayesian optimization uses prior experiences when choosing a new setting of hyperparameters to test and does this to iterate through different settings and finding the setting that maximizes a performance or minimizes a loss function. Bayesian optimization wants to optimize following equation

$$x^* = \arg\max_{x \in \chi} f(x) \tag{11}$$

where x^* is the optimal setting of the hyperparameters, $f(x)$ the performance metric you want to maximize and χ is the hyperparameter space. This optimization technique is not fully automatic. The designer still needs to define a hyperparameter space, choose a prior over functions, and choose an aqcuisition function. For hyperparameter optimization Gaussian Process models commonly are used as a prior for the function $f(x)$ with the mean function and the covariance functions incorporating the prior information. The Gaussian Process models are used due to their flexibility[30]. An example can be seen in figure 19.

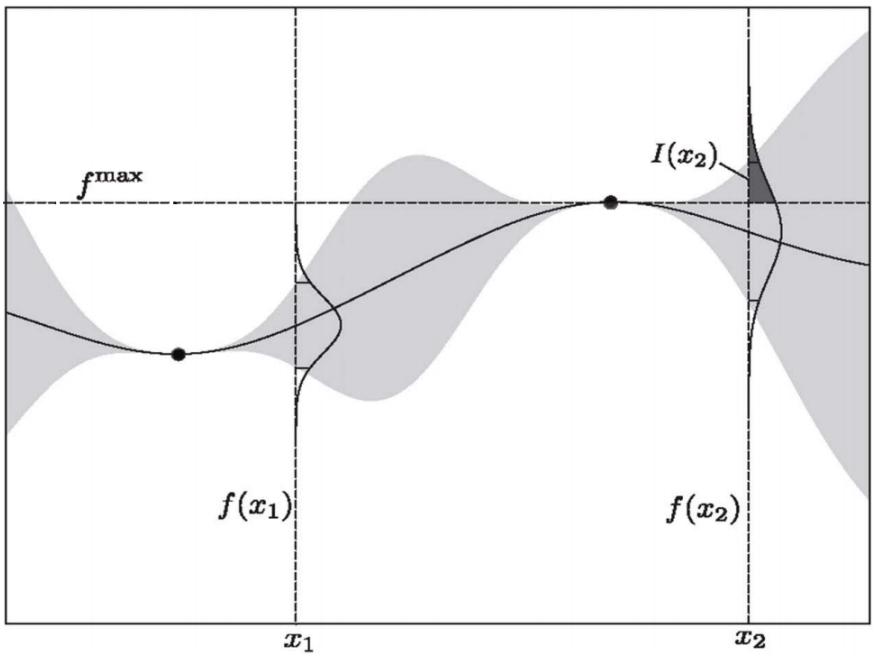

Figure 19: A Gaussian process model used to show the perfomance as a function of the choice of the hyperparameter x, where the dots are some prior tests and the acqusition function 'Probability of improvement' is visualized to decide whether to test x_1 or x_2 in the next trial[31]. The marked area $I(x_2)$ shows the probability of improvement at x_2.

The acquisition function uses the Gaussian process model to determine what spot in the hyperparameter space to explore next. One example of an acquisition function is the probability of improvement(PI) function:

$$PI(x) = P(f(x) \geq f^{max}) = \Phi\left(\frac{\mu(x) - f^{max}}{\sigma(x)}\right) \tag{12}$$

where $\mu(x)$ and $\sigma(x)$ is the mean and variance of the Gaussian process model for hyperparameter setting x. Using the acquisition function together with the guassian process model through several iterations the best setting of hyperparameters can be found. Another acqusition function is the expected improvement(EI) which has the equation:

$$EI(x) = \sigma(x)\left(\left(\frac{\mu(x) - f^{max}}{\sigma(x)}\right)\Phi\left(\frac{\mu(x) - f^{max}}{\sigma(x)}\right) + \mathcal{N}\left(\frac{\mu(x) - f^{max}}{\sigma(x)}; 0, 1\right)\right) \tag{13}$$

One of the major difference between the probability of improvement(PI) and the expected improvement(EI) is that EI search for where the improvement is expected to be the biggest whereas PI only focuses on to achieve an improvement, thus risking of falling down in local minimas[32]. A downside with Bayesian optimization is that you can't use parallell GPUs for computing several models at the same time since you need to evaluate a model before you can choose a new hyperparameter setting although there has been research about how this process can be parallized[30]. In order to perform Bayesian optimization when training time is long [29] suggests an algorithm to evaluate the function on subsets of the training

data and extrapolate the perfomance on the whole set, called FABOLAS in the paper.

The Bayesian optimization have been shown to converge faster to an optimal setting than random grid search, see figure 20, but in the long run the two methods are still comparable.

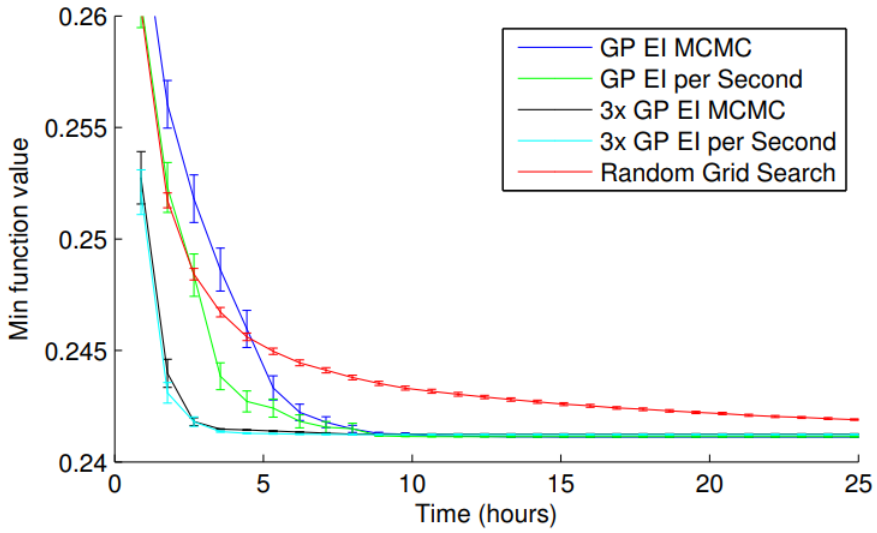

Figure 20: Comparison between a random grid search and four different Bayesian optimization methods with different acquisition functions, [30]. GP stands for Gaussian Process model and EI stands for the Expected Improvement with different modifications.

Therefore, Bayesian optimization is a better choice when evaluation time is long, like in many machine learning applications with large datasets and big models.

2.4 Performance metrics

To measure the performance of a model there are different types of metrics that can be used. They all are dervied from the confusion matrix to be discussed next.

2.4.1 Confusion matrix

The confusion matrix is a general measurement that other metrics are calculated from and it is a way of summarize the performance of a model by sorting all predictions made by the model into different categories depending on whether the prediction was correct or false. The performance of a binary classification task can be summarized in the confusion matrix in figure 21.

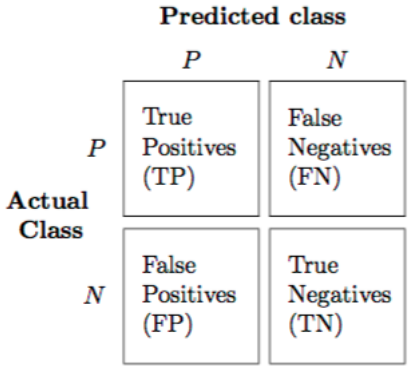

Figure 21: A confusion matrix for a case with two possible prediction classes[33].

From figure 21 there are two different scenarios for a prediction:

- When the predicted class and the actual class are the same, called true positive and true negative.
- When the predicted class is not the same as the actual class, called false negatives and false positives, [34].

2.4.2 Precision

The precision metric for a class shows how many percent of the predicted class are the actual class, calculated as

$$\frac{TruePositives}{PredictedPositives}$$

2.4.3 Recall

The recall metric for a class shows the percent of the actual class that is actually predicted as the correct class, calculated as

$$\frac{TruePositives}{ActualPositives}$$

In image classification tasks a threshold value on the prediction score is set to determine when an image is classified as that class.

2.4.4 Mean Average Precision (mAP)

The metric that is used to evaluate the COCO object detection challenge is the *mean Average Precision* (mAP), [10]. For every object class one can calculate the average precision from

the precision/recall curve. The precision/recall curve is the precision as a function of the recall. In order to get different recall levels, the threshold score for detection is altered. If the threshold score is set to zero, the recall would be one but the precision would be very low. For different recall levels the precision is measured and a precision/recall curve is plotted. The average precision(AP) is the average of this curve. This is then calculated for each object class and the mAP is the mean over the AP for every object class, [35]. The formula for the AP is

$$AP = \frac{1}{N} \sum_{r \in \{0,\dots,1\}} p(r) \tag{14}$$

where N are the number of recall levels between 0 and 1, p is the precision at recall level r.

There are different types of mAP metrics, for example between two large object detection challenges: the Pascal challenge and the COCO challenge. Pascal only measures the mAP for a IoU at 0.5, mAP@IoU=0.5, and COCO instead measures the mAP at ten IoU between 0.5:0.95 and averages the performance, mAP@IoU=0.5:0.95. This means that the Pascal metric focuses more on classification, instead of the COCO that focuses more on localization [36].

3 Task

The practical task of the thesis is to implement and train DCNN on a dataset to perform object detection. One dataset is a the COCO dataset, used in the COCO object detection challnege. The other dataset are more specific and consists of infrared images in order to perform object detection on three object classes: runways, approaching lights and PAPI lights.

3.1 Infrared Images

The infrared spectrum contains ligth with wavelengths bewteen $0.9 - 12\mu m$ and in this infrared spectrum the infrared light can be divided into three different ranges: short wave infrared radiation (SWIR), mid wave infrared radiation (MWIR) and long wave infrared radiation (LWIR).The ranges are visible in the spectrum in figure 22, [37].

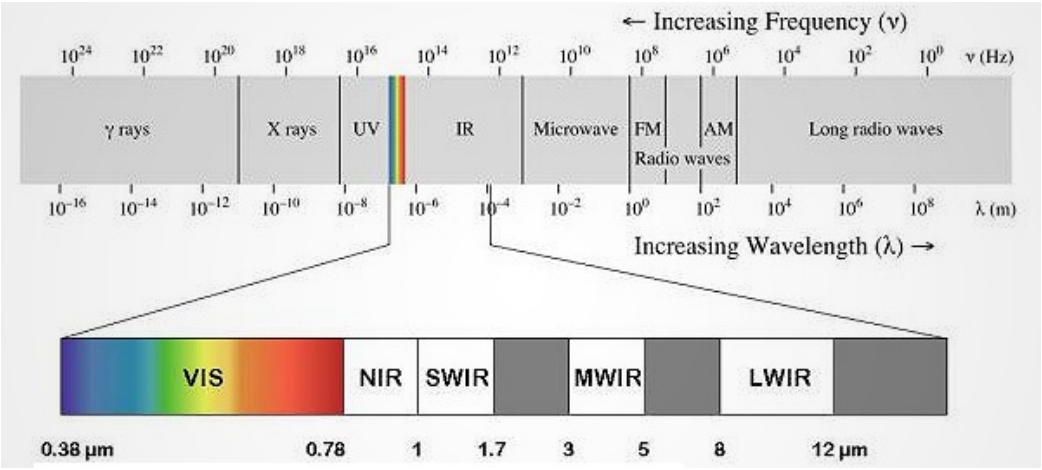

Figure 22: The light spectrum with both visible light and infrared light [37].

The SWIR detects light that is reflected from the sun and the MWIR and LWIR detects light emitted from an object, depending on the temperature of the object [38]. The difference between a LWIR and a SWIR image are visualized in figure 23 and 24.

Figure 23: An example of a LWIR image of a runway.

Figure 24: An example of a SWIR image of a runway

In the SWIR image, figure 24, the lights from the approaching lights can easily be seen while they are not seen in the LWIR image, figure 23. An advantage with infrared light is that fog becomes somewhat transparent and therefore infrared cameras can be used as a visual assistance when visibility is low due to fog or during night. This is discovered when looking at the spectral transmission in different weather conditions. During light fog both SWIR and LWIR has better transmission than the visible light. This is showed in figure 25 [39].

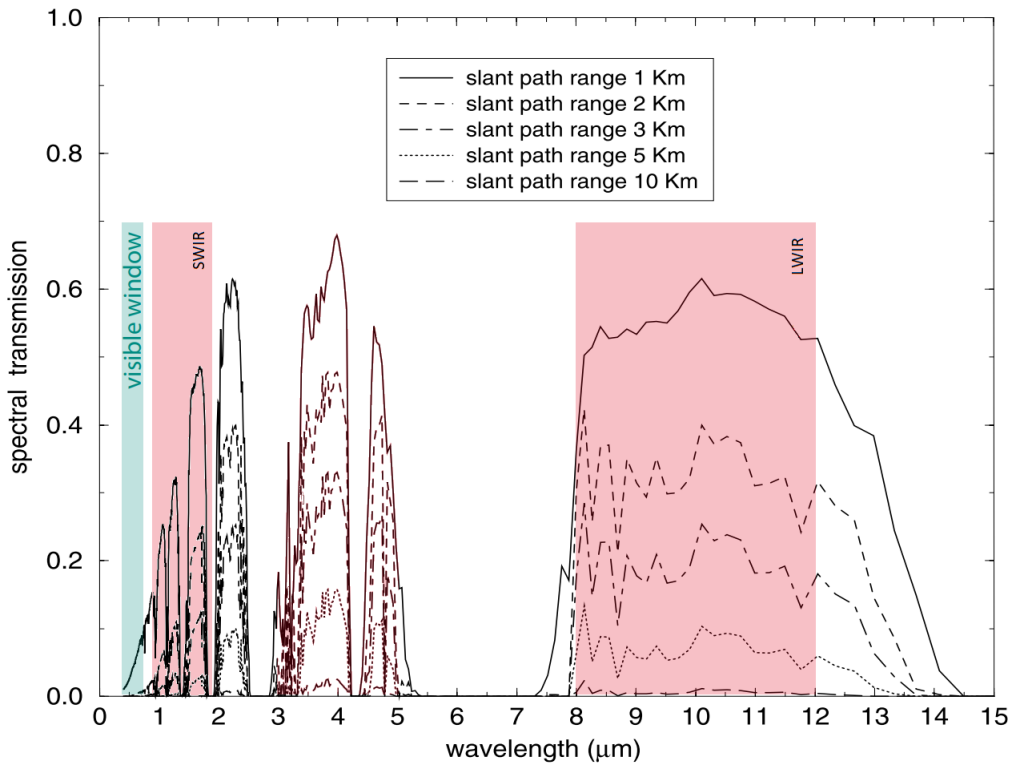

Figure 25: The spectral transmission as a function of different wavelenghts at different ranges for light fog[39].

3.2 COCO dataset

The COCO dataset contains of over 200.000 annotated RGB images with 80 object classes of common objects, for example, car, airplane, person, cup and TV.

3.3 Infrared dataset

The infrared image set provided by Saab consists of LWIR and SWIR videos from an airplane approaching an airport. In total, the dataset is based on videos recorded on two different flight dates containing 12 approaches to the same airport. For every approach there is one SWIR video and one LWIR video. These videos were transformed into annotated images giving around 25000 images in the dataset. In the images, three objects were annotated: the runway, the approaching lights, and the PAPI lights, shown in figure 26.

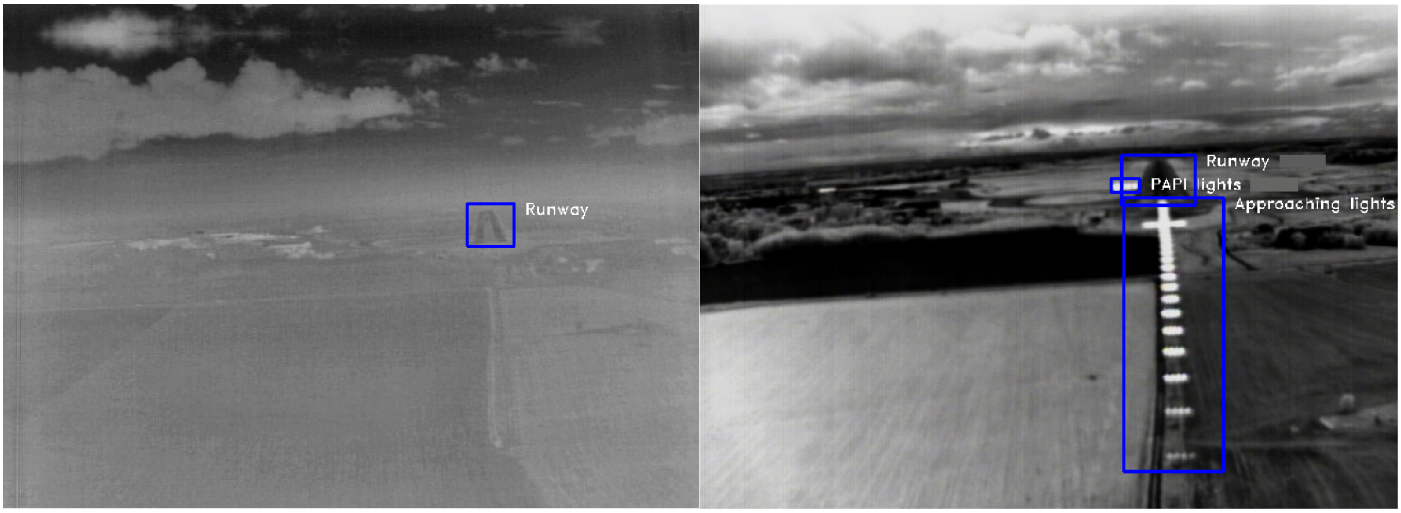

Figure 26: To the left: A LWIR image with the runway marked out. To the right: A SWIR image with the PAPI lights, the runway and the approaching lights marked out.

The LWIR images only have the runway annotated since the approaching lights and PAPI lights are not visible in this spectrum. Each image has a resolution of 656x474 pixels and has one color channel.

4 Method

The one-stage network RetinaNet was choosen as the model architecture to be implemented. RetinaNet was choosen since a state-of-the-art network was to be examined and since low prediction time is of interest. A one-stage detector was choosen since it generally has a lower prediction time than a two-stage detector. The model was implemented in Keras using Tensorflow as backend software. A github repository was used and modified in the implementation [40]. The training of models was executed in different stages. First the COCO dataset was used to train a model for object detection. Then the infrared image dataset was used to train a model to detect runways, approaching lights, and PAPI lights in infrared images. The model was implemented in different configurations. Finally Bayesian optimization was implemented and tested in order to optimize hyperparameters' of the model. All the stages are explained in section 4.1, 4.2 and 4.3.

All the networks were trained on a Nvidia Titan X GPU and for evaluation the metric mAP was used with the criteria IoU=0.5, same as the Pascal object detection challenge metric. During the training process the Adams optimizer was used with learning rate $=10^{-5}$ and clipnorm 10^{-3}. The training was run for 50 epochs with 10000 steps per epoch. All trained models used a backbone network with pretrained weights on the Imagenet dataset.

4.1 Training on the COCO dataset

The RetinaNet with the ResNet with 50 layers (ResNet50) as backbone network was implemented. The backbone network was used with pretrained weights on the Imagenet. The model was trained on the COCO dataset train set to perform object detection in RGB images. The model was then evaluated on the validation set with the mAP@IoU=0.5 metric.

4.2 Training on infrared dataset

The RetinaNet was also used as the architecture for training models on the infrared dataset. All models used a backbone network with weights pretrained on the Imagenet dataset. To visualize the model performance 7 image examples (5 SWIR images and 2 LWIR) at different ranges were used to evaluate predictions to complement the mAP evaluation metric. The example images can be seen in figure 27.

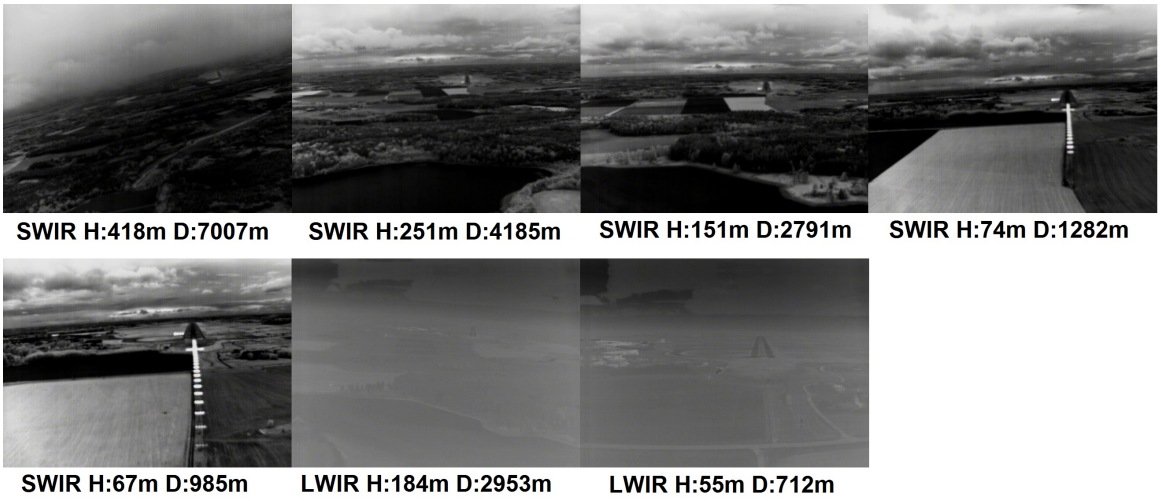

Figure 27: Seven example images used to visualize the performance of models. H means height above ground and D denotes Distance to the runway.

Following different modifications of the network were tested out

- Input image size (varied between 224x224 and 1384x1000). If the original image had a smaller size, it was upscaled in the preprocessing of the image.
- Testing different datasets. One model trained on a SWIR dataset and one only trained on a LWIR dataset. Models trained on a dataset consisting of both SWIR and LWIR images were also tested.
- Dividing the SWIR dataset into two datasets based on the distance from the runway. One long distance dataset and one short distance dataset in order to create two different model that can be applied at different ranges.

- Different amounts of data augmentation in the preprocessing of the images. The different types that were tested were scaling, horizontal flip, vertical flip and rotation. Examples of augmented images can be seen in figure 28.
- Different backbone of the system, (ResNet50 and Mobilenet), see section 4.2.2
- Different input layers to the network, see section 4.2.1
- Application of contrast limited adaptive histogram equalization to the LWIR images before training the model.
- No updating of the parameters of the backbone network during the training process, called frozen backbone.

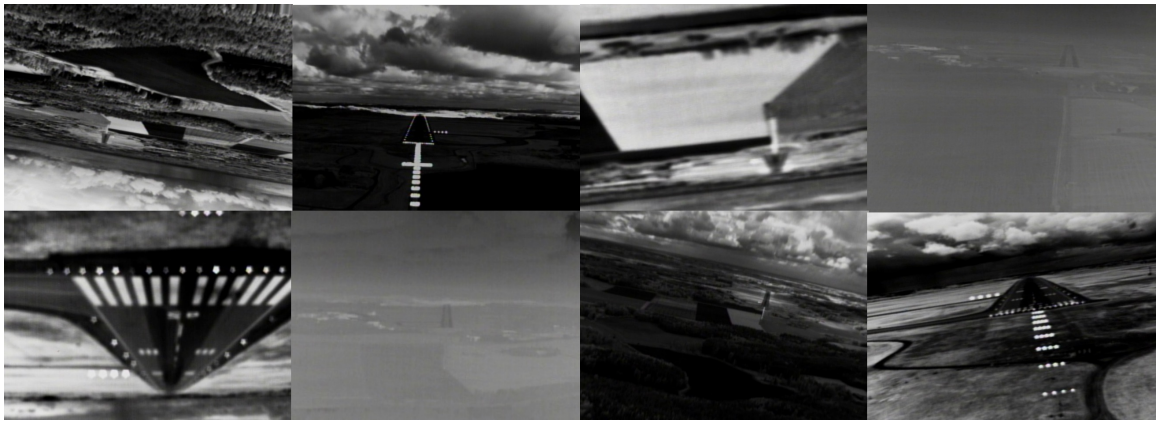

Figure 28: Some examples of different types of data augmentation.

4.2.1 Converting a network from RGB to IR-images

Since the IR-images only have one channel (luminance) and the RGB images has three channels(red, green and blue) the RetinaNet needed to be modified to fit the IR-images. The following approaches were tested and evaluated:

- The same model architecture as the one trained on the COCO dataset with RGB images was used and the IR-image was duplicated and used as input for all three channels. Visualized in figure 29.a)
- A new input layer was put before the RetinaNet that takes the LWIR image and the SWIR image and a convolutional layer with three filters was inserted. In that way, we convert a two channel input into three channels. By using both LWIR and SWIR as a single input image, maximal information was given to the system. Visualized in figure 29.b)
- Two channels were used (one LWIR and one SWIR) as input layer and apply the first backbone layer directly to the input layer. Visualized in figure 29.c)

- The weights of the first convolutional layer were modified and summed up across channels since this would correspond to the equivalent of duplicating the one channel on three input channels, this is illustrated in equation 15.

$$w_1 r + w_2 g + w_3 b \quad \rightarrow \quad (w_1 + w_2 + w_3) i \tag{15}$$

where w_1 is the weights that are applied to the red input channel, w_2 is the weights that applied to the green input channel and w_3 for the blue input channel and i is the infrafred input channel. This is true in the case the infrared channel is used as all the three input channels as in figure 29.a)

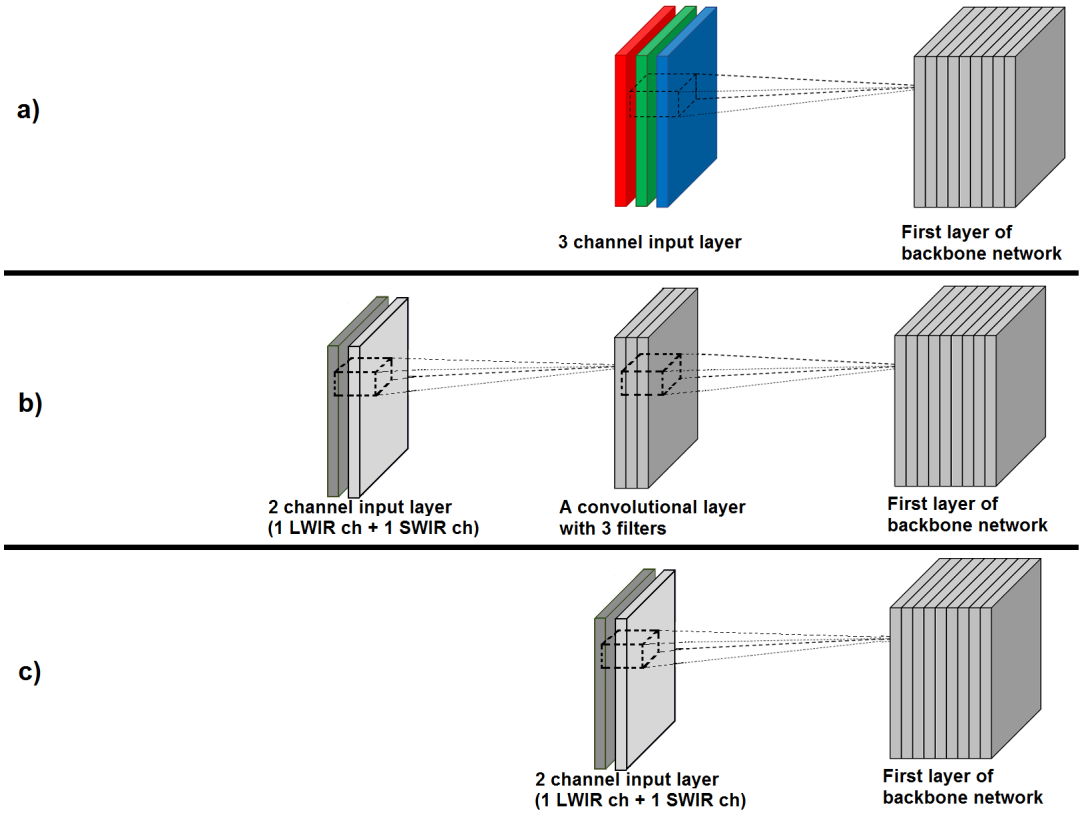

Figure 29: Visualisation of the three different approaches of modifing the model architecture.

4.2.2 Changing backbone network

The standard RetinaNet uses the ResNet as backbone network in its architecture and in order to decrease prediction time of the network the Mobilenet was used as the backbone instead. In order to use the pretrained weights on Imagenet the input image to Mobilenet needs to be of fixed square size (128x128, 160x160, 192x192 or 224x224). The Mobilenet with 224x224 as input was implemented as a backbone with the RetinaNet model and the performance was evaluated and the prediction time measured. A RetinaNet model with

ResNet50 backbone and input size 224x224 was also implemented and tested in order to get a comparison.

4.3 Bayesian Optimization

A Bayesian hyperparameter optimization was tested with a Gaussian process model used as the function prior with 'Expected Improvement' as the acqusition function. The hyperparameter setting that was used previously to train models in section 4.2 was implemented for a reference mAP and a Bayesian optimizer was run for 27 iterations in order to maximize the mAP. A random grid search was also performed for 27 iterations with the same hyperparameters for comparison. The hyperparameters and the ranges in the parameter space that were chosen to optimize were

- Learning rate (range: $10^{-6} - 10^{-2}$),
- Standard deviation of the weight initialization (range: $10^{-3} - 1$),
- Clipnorm (range: $10^{-1} - 10^{-4}$),
- Anchor scales (Three parameters, range: $0.5 - 2.2$)

The Bayesian Optimization was implemented using GPyOpt, a Python framework for Gaussian process modelling[41]. Since the chosen model takes 12 hours to train and evaluate, the number of training images per epoch was decreased from 10000 to 5000 to decrease the training time and increase the number of possible iterations. The execution time for 27 iterations was around 8 days. The used covariance function was the ARD Matérn 5/2 kernel[30].

5 Results

From the obtained results the following can be stated.

- The model trained on the COCO dataset in order to detect common objects got a mAP=0.5071.
- From the models trained on IR images in order to detect runway, approaching lights and PAPI lights
  - training one model on both LWIR and SWIR images manages to make the most correct predictions on the example images, see Table 2.
  - More data augmentation gives higher mAP, see Table 4.
  - Larger input image size gives a higher mAP but also a higher prediction time, see Table 3.
  - Out of the different input layers tested, the one where a single image was passed in on three channels gave the best performance, see Table 6.

  - In general, the approaching lights object has the highest AP out of the three object classes.
  - Using ResNet backbone results in a higher mAP compare to Mobilenet but also a longer prediction time, see Table 7.

- When using a Bayesian optimization for setting the hyperparameters resulted in a mAP=0.9060 compared to the reference network with a mAP=0.9062.

5.1 Training on COCO dataset

Training the RetinaNet on the COCO dataset resulted in a mAP of 0.5071 and an example object detection made by the model can be seen in figure 30.

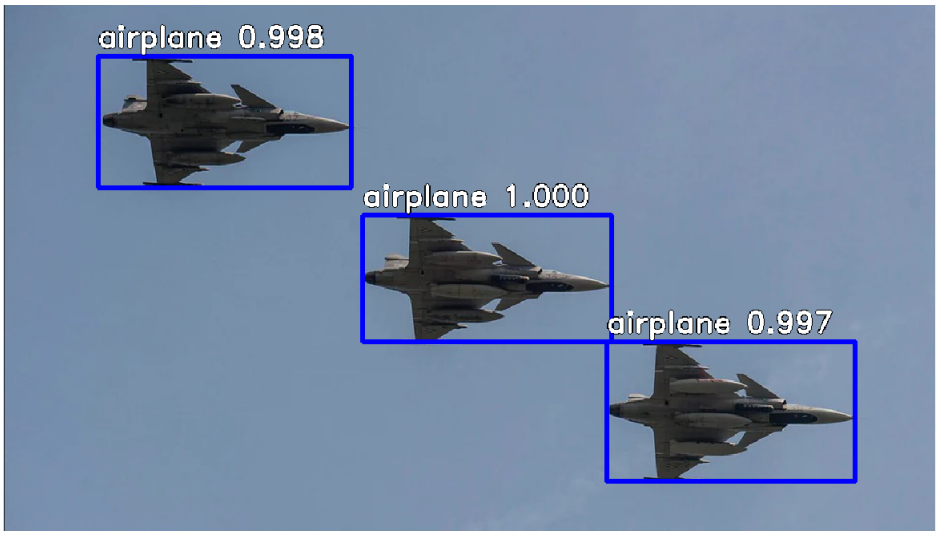

Figure 30: An example of object detection performed by a model trained on the COCO dataset. The image shows the predicted class and the prediction score.

5.2 Training on IR images

Unless otherwise noted, the following parameters were used: RetinaNet architecture with ResNet50 as backbone network. Input image size 830x600px and the horizontal flip used for data augmentation. The used dataset contains both the SWIR and the LWIR images. The required classification threshold was set to 0.5.

5.2.1 Different IR-Datasets

The mAP from models trained on different datasets can be viewed in Table 1.

Table 1: The mAP for models trained on different datasets

Dataset	mAP	AP Runway	AP Approaching Lights	AP PAPI Lights
LWIR	0.5866	0.5866	-	-
SWIR	0.7823	0.6774	0.9379	0.7317
SWIR Short range	0.8958	1.000	0.9411	0.7464
SWIR Long range	0.7828	0.6127	0.9530	0.7828
LWIR + SWIR	0.8463	0.7531	0.9439	0.8463

The results for letting the different models perform object detection on the example images are summarized in Table 2. No false positives were noted, only false negatives.

Table 2: The results from prediction on the chosen example images using models trained on different datasets. Yes indicates correct prediction, No indicates a false negative of specified class. If no class is specified, false negatives of all classes were made.

Dataset	SWIR H:67m D:985m	SWIR H:74m D:1282m	SWIR H:151m D:2791m	SWIR H:251m D:4185m	SWIR H:418m D:7007m	LWIR H:184m D:2953m	LWIR H:55m D:712m
LWIR	No	No	No	No	No	Yes	Yes
SWIR	Yes	Yes, no PAPI lights	Yes	Yes	No	No	No
SWIR Short range	Yes	Yes, no PAPI lights	No	No	No	No	No
SWIR Long range	Yes, no PAPI lights	Yes, no PAPI lights	Yes	Yes	No	No	No
LWIR+SWIR	Yes	Yes, no PAPI lights	Yes	Yes	No	Yes	Yes

5.2.2 Input image size

The results from alternating the input image resolution and shape are displayed in Table 3. The models were trained on the LWIR + SWIR dataset and the only data augmentation used was the horizontal flip.

Table 3: The mAP for the different models with different input image size.

Input image size	mAP	Prediction time (ms)	AP Runway	AP Approaching Lights	AP PAPI Lights
224x224	0.2446	18	0.1501	0.5419	0.0417
830x600	0.8463	42	0.7541	0.9439	0.8410
1107x800	0.8646	61	0.7894	0.9429	0.8614
1384x1000	0.8142	80	0.6804	0.8897	0.8725

5.2.3 Data augmentation

The results from varying the amount of data augmentation are displayed in Table 4.

Table 4: The mAP achieved from the same model trained on a dataset with different amount of data augmentation applied.

Data Augmentation	mAP	AP Runway	AP Approaching Lights	AP PAPI Lights
No augmentation	0.7823	0.6774	0.9379	0.7317
Vertical Flip Horizontal Flip	0.7788	0.6585	0.9576	0.7204
Vertical Flip Horizontal Flip Rotation	0.8463	0.7541	0.9439	0.8410
Vertical Flip Horizontal Flip Rotation Scaling	0.8554	0.7648	0.9613	0.8401

5.2.4 CLAHE

The results from using CLAHE as image preprocessing on the LWIR dataset can be viewed in Table 5. The results come only from runway detection since approaching lights and PAPI lights are not visible in the LWIR images.

Table 5: The mAP of two models with and without using CLAHE as additional image preprocessing

Preprocessing	mAP
With CLAHE	0.6650
Without CLAHE	0.6066

5.2.5 Different input layers

From the testing of different modification of the input layer to the RetinaNet, the performance is presented in table 6.

Table 6: The performance of models with different input layers. The different input layers are described in detail in section 4.2.1.

Input layer	mAP	Prediction time (ms)	AP Runway	AP Approaching Lights	AP PAPI Lights
Three channels, Fig 29 a)	0.9245	61	0.8817	0.9900	0.9018
Two channels, Fig 29 b)	0.8513	61	0.8281	0.9618	0.7640
Two channels, Fig 29 c)	0.8620	61	0.8843	0.9747	0.7269

5.2.6 Different backbone network

The results from using different backbone network and their prediction time can be seen in Table 7. The input image size was fixed at 224x224.

Table 7: The perfomance and the prediction time when using different backbone network.

Backbone	mAP	Prediction time (ms)	AP Runway	AP Approaching Lights	AP PAPI Lights
ResNet 50	0.2446	18	0.1501	0.5419	0.0417
Mobilenet	0.1888	14	0.1320	0.4253	0.0093

5.3 Hyperparameter optimization

The results from the two tested hyperparameter optimization methods can be seen in Table 8.It is noticable that both methods have a perfomance close to the reference performance.

Table 8: The best performance of the two optimization methods

Optimization method	Highest mAP
Reference network	0.9062
Bayesian optimization	0.9060
Random grid search	0.9027

The Bayesian optimization gave the best mAP of 0.9060 with settings stated in table 9 together with the values used to train the reference model:

Table 9: The resulting values of the Bayesian optimization function compared to the Reference values

Hyperparameter	Value Bay. Opt.	Value Reference
Learning rate	10^{-5}	10^{-5}
Standard deviation	10^{-2}	10^{-2}
Clip norm	10^{-2}	10^{-2}
Anchor scales	(1.012, 1.011, 1.623)	(1.000, 1.2600, 1.4142)

6 Discussion

6.1 The model trained on the COCO dataset

The results obtained when training the RetinaNet on the COCO dataset are similar to the results from the published paper (mAP = 0.5320 for validation on COCO with images size 830x600 and IoU@0.5,[17]). This indicates that the implementation of the model is correctly made.

6.2 The infrared dataset

The best results on the test data when training on the infrared dataset are with high mAP, mAP=0.9245. However, we do not know how general this performance is since the dataset only include data from an airplane approaching the same airfield in similar weather conditions, giving very similar images. It would be interesting to be able to train this model on more versatile data and be able to evaluate the model for different airfields and even more interesting in weather conditions when the visibility is worse to see if the model is able to detect the runway when the human eye cannot. The uniform data can also be a major reason why data augmentation improves the performance. More augmentation generates a more versatile dataset, comparing with the original.

The reason to why the "approaching lights" class generally achieves the highest AP is probably due to the fact that the approaching lights have a clear shape and contrast towards the background and is also visibly the easiest object to detect.

6.2.1 Prediction time vs. performance

The results point to that a larger image resolution gives better results although it also results in longer prediction times. The reason behind the improvment of performance with larger input images could be that some objects in some of the images are too small to be detected. For example, the PAPI lights class that generally is the smallest object in the images and also has the most improvement in AP when the input image is increasing. This could mean that the sizes of the anchor boxes of the model are too large and can therefore not detect small objects. Therefore the anchor scales was one of the parameters used in the hyperparameter optimization and from the results the Bayesian optimization did not find significantly different values for the anchor scales that was able to perform better. This could also be the consequence of the Bayesian optimization function not trying enough different values during 27 iterations.

6.2.2 Lowering the prediction time

In order to lower the prediction time both reducing input image size and changing to a Mobilenet as a backbone network works. Although from my result the performance is a

trade-off with the prediction time since a model with lower prediction time generally has a lower mAP.

6.2.3 Different input layers

From the different variations of the input layers the model with one image used as input on three channels performed the best and this is quite interesting when the intuitive feeling would suggest that the approach when using both the LWIR and SWIR image for a prediciton would present more information to the system. One possible explanation to why the 2 channel in system is not outperforming 3 channels could be that two layers corresponding to projections are sufficient to preserve the original info. For some unknown reason the optimisation procedure did not find those projections and information could be lost before the first convolutional layer of the backbone networks with 64 filters is applied. This is why the input layer with two channels as input layer and applying the first layer of the backbone network with 64 filters directly to the input layer. However that did not increase the performance significantly. The number of channels in the input layer have no impact on the prediction time.

6.3 Hyperparameter optimization

From the different tests of hyperparameter optimization techniques I can see that both random grid search and Bayesian optimization manages to perform approximate as well as the best model I managed to train with my own settings without using any optimization methods, called the reference model. It could be that the performance of the model is not sensitive to the chosen hyperparameters that was optimized although the learning rate is stated as one of the most important hyperparameters in a DCNN,[9]. It is interesting to see that random grid search manages to perform as well as Bayesian optimization on only 27 iterations, where theory would suggest that random grid search would need many interations to compete with Bayesian optimization. This would also point to that the performance of the model is not very sensitive to the hyperparameters as stated before. It is also interesting that the Bayesian optimization manages to find almost identical values to the reference values in 27 iterations. Since the examined RetinaNet version takes approx 6 hours to train, the number of iterations I was able to do was limited. Therefore an interesting approach could be to implement the FABOLAS method designed for large datasets and models that take long time to train.

7 Conclusion

In conlusion, I can say that an object detection network with high mAP was created in order to detect approaching lights, runway and the PAPI lights in infrared images. The question still remains how general the model is on images from other airfields and weather conditions and most, interestingly, in worse weather conditions when the visibility is poor. More data is the key to allow us to ensure a good perfomance of the model.

When it comes to the prediction time, it takes longer to predict on higher resolution images, and the performance is increased with larger images. The prediction time can therefore be seen in a trade-off with performance to a certain point when performance no longer further improves with a larger image due to the difficulty to detect small objects.

Before this system can be implemented in an airplane both the data scarcity issue and shorter prediction time need to be addressed since both high performance and real-time prediction need to be fullfilled in practice.

Regarding hyperparameter optimization function I managed to implement a Bayesian optimization that performed equally well as both random grid search and the reference model when optimizing over a set of hyperparameters. Even though it does not perform better than the reference network, it shows that it can be useful in cases when no knowledge about the model is available beforehand to get a good starting point of hyperparameter settings quickly to work further with.

Finally, I can say that when searching for the state-of-the-art object detecting network one realizes that this area is under great development with new ideas, papers and improvements being realeased almost every day. It is a challenge just to keep up with all progress. The future of object detection and image analysis using machine learning will be an interesting topic to follow.

8 Further work

Possible continuations and interesting topics to further dig into would be to implement and examine the FABOLAS paper [29] regarding hyperparameter optimization using Bayesian optimization since this method seems to be more adapted to this type of model when training time is long.

The are still other very interesting models that would be interesting to implement and compare with RetinaNet, for example, the light-head RCNN [22] model that claims to be a two-stage detector that is as fast as a one-stage detector. As for the application of this project to help the airplane pilot with object detection, the need to deal with the prediction time/performance trade off is a very important issue, since we want a network that performs

well and at a high framerate.

A step towards a final product would also be to implement this network on an embedded platform instead of a GPU and investigate how the performance and the prediction time change. To address the need for more data it would be interesting to see how a model trained on synthetic data can perform on real images.

References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, 2012

[2] http://cocodataset.org/#detection-leaderboard

[3] http://image-net.org/challenges/LSVRC/2012/supervision.pdf

[4] https://www.skybrary.aero/index.php/Decision_Altitude/Height_(DA/DH)

[5] https://www.skybrary.aero/index.php/Visual_References

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org

[7] https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f40359318721

[8] https://cambridgespark.com/content/tutorials/convolutional-neural-networks\-with-keras/index.html

[9] F.Li, J.Johnson and S.Young, *Course notes from the course Neural Networks for Visual Recognition*, Stanford University, 2017, http://cs231n.github.io/

[10] http://www.cocodataset.org

[11] D.P.Kingma and J.L.Ba, *Adam: A Method for Stochastic Optimization*, 2015,

[12] https://pythonmachinelearning.pro/a-guide-to-improving-deep-learnings-performance/

[13] S.M Pizer, *Adaptive Histogram Equalization and its Variations*, 1986

[14] S.M Pizer, *Contrast-Limited Adaptive Histogram Equalization: Speed and Effectiveness*, 1990

[15] https://towardsdatascience.com/deep-learning-for-object-detection-a\-comprehensive-review-73930816d8d9

[16] J.Redmon, S.Divvala, R. Girshick and A.Farhadi, *You only look once: Unified, real-time object detection.* , 2016

[17] T.-Y.Lin, P.Goyal, R. Girshick, K.He and P.Dollár, *Focal loss for dense object detection.* , 2017

[18] W.Liu, D.Angeuelov, D.Erhan, C.Szegedy, S.Reed, C-Y.Fu and A.C.Berg, *Ssd: Single shor multibox detector* , 2016

[19] R. Girshick, J. Donahue, T. Darrell and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation* , 2014

[20] R. Girshick, *Fast r-cnn* , 2015

[21] S.Ren, K.He, R. Girshick and J.Sun, *Faster r-cnn: Towards real-time object detection with region proposal network* , 2015

[22] Z.Li, C.Peng, G.Yu, X.Yang, Y.Deng, J.Sun, *Light-Head R-CNN: In Defense of Two-Stage Object Detector* , 23 November 2017

[23] A.Howard, M.Zhu, B.Chen, D.Kalenichenko, W.Wang, T.Weyland, M.Andreetto, H.Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications* , 2017

[24] K.He, X.Zhang, S.Ren and J.Sun, *Deep Residual Learning for Image Recognition* , 2015

[25] C.Peng, T.Xiao, Z.Li, Y.Jiang, X.Zhang, K.Jia, G.Yu and J.Sun, *MegDet: A Large Mini-Batch Object Detector*, 2017, https://arxiv.org/pdf/1711.07240.pdf

[26] T.-Y.Lin R. Girshick, K.He, P.Dollár, B.Hariharan and S.Belongie, *Feature Pyramid Networks for Object Detection.* , 2017

[27] www.image-net.org

[28] J.Bergstra and Y.Bengio, *Random Search for Hyper-Parameter Optimization*, 2012,

[29] A.Klein, S.Falkner, S.Bartels, P.Hennig and F.Hutter, *Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets* , 2017

[30] J.Snoek, H.Larochelle and R.P.Adams, *Practical Bayesian Optimization of Machine Learning Algorithms* , 2012

[31] Nando de Fretias, *Slides from Bayesian optimisation and deep learning*, University of Oxford , 2014, www.mlss2014.com/files/defreitas_slides1.pdf

[32] www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf

[33] https://rasbt.github.io/mlxtend/user_guide/evaluate/confusion_matrix/

[34] http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology

[35] M.Everingham, L.Van Gool, C.Williams, J.Winn and A.Zisserman, *The Pascal Visual Object Classes (VOC) Challenge*,2009, homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

[36] S.Bell, C.Zitnik, K.Bala and R.Girshick, *Inside-Outside Net: Detecting Objects in Context with Skip Pooling and Recurrent Neural Network*, 2016, https://www.cs-cornell.edu/~sbell/pdf/cvpr2016-ion-bell.pdf

[37] http://www.raptorphotonics.com/technology/visswir-camera

[38] http://www.edmundoptics.com/resources/application-notes/imaging/what-is-swir

[39] FLIR, *Seeing through fog and rain with a thermal imaging camera*, http://www.flirmedia.com/MMC/CVS/Tech_Notes/TN_0001_EN.pdf

[40] github.com/fizyr/keras-retinanet

[41] Univeristy of Sheffield, sheffieldml.github.io/GPyOpt