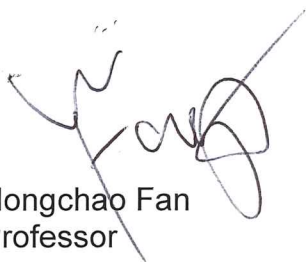


Extraordinary circumstances due to the Corona pandemic

This Master thesis is based on work that was accomplished in the spring semester 2020. In this period the Corona pandemic was active and influenced the work of several master students. The grading of the thesis must take the pandemic situation into consideration.

If this Master thesis is affected by the Corona pandemic, the student will point out the influenced elements in the beginning of the report. More details about this may also be explained later in the thesis.



Hongchao Fan
Professor

Address	Org. no. 974 767 880	Location	Phone	Executive officer
7491 Trondheim Norway	postmottak@iv.ntnu.no www.ntnu.no/ibm	Høgskoleringen 7 A	+47 73559665	Hongchao Fan hongchao.fan@ntnu.no Phone: +4773559665

Please address all correspondence to the organizational unit and include your reference.

Abstract

Geographical data is important to understand spatial relations. Many of today's open-source databases for GPS and spatial queries contains little to no information of the spatial context in an intersection. By increasing the information in an intersection, one can replace the need to process the surrounding environment one-the-fly through object detection by using preprocessed data stored in each intersection. This can reduce the limiting factor of computational resources, as the field of object detection and enormous databases constantly are made more and more complex.

This paper proposes a framework that can be used to estimate the position of detected objects in images from one of the worlds largest spatial street-view image database Mapillary. The framework first proposes an overview of the current state-of-the-art technologies for object detection, and then chooses the best suited network architecture to train a network to recognize traffic signs in images. From these detected objects, a monocular depth estimation is performed on the image using a pretrained network, which is used to calculate the depth disparity in the pixel space. In addition, several assumptions about the sizes of known objects, in order to propose a pixel-per-meter algorithm for calculating the position of the detected objects. Once an image is processed and given a position, the image is either placed in an existing intersection, or a new intersection is made by exploiting the information available in open-source spatial database APIs. The information retrieved through this framework is returned as a map layer in the form of a GeoJSON object.

This page is intentionally left blank

Sammendrag

Geografisk data er viktig for å forstå romlige relasjoner. Mange av dagens *open-source* databaser for GPS og romlige spørringer inneholder lite, til ingen data om den romlige konteksten i vegkryss. Ved å øke informasjonen i et vegkryss kan man i stede for å prosessere omgivelsene *on-the-fly* gjennom objektgjenkjenning, bruke ferdig prosessert data knyttet til hvert vegkryss og potensielt redusere behovet for enorme beregningsressurser, ettersom feltet innenfor objektgjenkjenning og enorme databaser konstant gjøres mer og mer kompleks.

Denne oppgaven fremlegger et forslag på et rammeverk som kan brukes til å beregne posisjonen til objekter funnet i bilder fra en av verdens største romlige *street-view* bildedatabaser Mapillary. Rammeverket fremlegger først en analyse av dagens *state-of-the-art* teknologier for bildegjenkjenning, og velger den beste av disse for å trene opp et nettverk for å kjenne igjen trafikskilter i bilder. I tillegg brukes et ferdig trent nettverk for å lokalisere trafiklysene i bilder. Utifra disse objektene, utføres en monokulær dybdeestimasjon gjennom et trent nettverk, som brukes til å beregne en dybdeforskjell i pixelrommet. Videre fremstilles det antagelser for størrelser for kjente objekter for å beregne en pixel-til-meter algoritme for å kalkulere posisjonen til det gjenkjente objektet. Når bildet er ferdig prosessert og objektene er gitt en posisjon, plasseres det i et enten eksisterende vegkryss, eller det opprettes et nytt vegkryss ved å benytte seg av informasjon fra *open-source* vegdatabase APIer. Informasjonen innhentet igjennom rammeverket returneres som et kartlag i form av et GeoJSON objekt.

This page is intentionally left blank

Preface

This master thesis is written for the Department of Civil and Environmental Engineering at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. The master thesis is a part of the study program Engineering & ICT, with a specialization in Geomatics. This paper is written in the spring 2020.

I would also like to thank my supervisor, Hongchao Fan, for his assistance during this duration, especially considering the difficult work environment of the Corona situation this spring. In addition, I would like to thank my fellow students for the shared insight and knowledge in the field.

Trondheim, 2020-06-28

Kristoffer Saastad

This page is intentionally left blank

Contents

1	Introduction	1
1.1	Background and motivation	2
1.2	Research goals	2
1.3	Limitations	3
1.4	Outline	3
I	Theory and related work	4
2	Theoretical background	5
2.1	Artificial neural networks	6
2.1.1	Artificial neurons	6
2.1.2	Activation functions	7
2.1.3	Feed-forward neural networks	9
2.1.4	Training a neural network	9
2.1.5	Loss functions	10
2.1.6	Backpropagation	13
2.1.7	Optimizers	13
2.1.8	Hyperparameters	16
2.2	Convolutional neural networks	17
2.3	Object detection	21
2.3.1	Evaluation metrics	21
2.3.2	Feature extraction	23
2.4	Depth estimation	24
2.4.1	Stereo rectification	24
2.4.2	Supervised depth estimation	25
2.4.3	Self-supervised depth estimation	26
2.4.4	Self-supervised monocular depth estimation	27

3	Datasets in neural networks	29
3.1	Object detection	30
3.2	Depth estimation	31
3.3	Test dataset	31
II	Methods and implementation	32
4	Methodology	33
4.1	Network architecture	34
4.1.1	Brief history of object detection	34
4.1.2	Review of network architectures	35
4.1.3	Feature pyramid networks	37
4.1.4	EfficientDet	38
4.2	Training the model	40
4.3	Estimating depth	42
4.3.1	Estimating position of the objects	45
4.4	Defining the intersection	49
4.5	Proposed map layer	50
5	Results	52
5.1	EfficientDet	53
5.2	Localization of the objects	59
III	Discussion and conclusion	63
6	Discussion	64
6.1	The model architecture	65
6.2	Prediction results	66
6.3	Depth estimation	67
7	Conclusion	69
7.1	Conclusion	70
7.2	Future work	71
A	Appendix	76
A.1	The full architecture of the EfficientDet network	76

Chapter 1

Introduction

In this chapter, the motivation and background of the paper is presented, as well as the research goals. An outline of the rest of the paper is also presented.

1.1 Background and motivation

With increased research in the machine learning field, many tasks can be solved autonomously, and the expectations of machine learning algorithms to automate day-to-day tasks is only increasing. This has put a huge pressure on the field in terms of both making machine learning algorithms more efficient, as well as the need for more data in order to continuously make model architectures more robust.

With the introduction of AlexNet by Krizhevsky et al. in 2012[2], object detection and feature extraction through supervised learning became commonly used to solve recognition problems, and has helped push the field of extraction important features a long way. Object detection classifier are used in for example autonomous vehicles, but with many of these computations being done on-the-fly, the computational power is a bottleneck because fast decision making requires fast processing. Pre-processing these feature extractions can reduces computational resources needed.

By peeking into open-source spatial databases, there is a lot of data yet to be generated. In terms of intersections, many of these databases, such as Open Street Map[29], an intersection is only represented as a single point with no information about the surrounding spatial environment. With more and more data becoming available, images with geospatial information can easily be accessed. The Mapillary open-source database[19] contains millions of geo-tagged, street-level images. By combining such available images with the research of object detectors, several research goals can be derived.

1.2 Research goals

- (1) Analyze the current state-of-the-art network architectures in order to train a network and predict traffic signs through object detection in images.
- (2) Apply a monocular depth prediction model to the images and establish an algorithm to predict an object's position without knowing the interior orientation parameters.
- (3) Propose a map layer using the framework proposed from the previous research goals for traffic intersections containing spatial and temporal properties for each intersection.

1.3 Limitations

Due to the advanced nature of the recent state-of-the-art network architectures and the pre-trained models being trained on many computers, the computational resources from a single computer limits the work and experimentation of the paper. In addition, assumptions made during the research phases causes errors in the estimation and limits the performance.

1.4 Outline

The structure of the paper is separated into three parts; Part 1: the theory and related work, Part 2: the implementation, methodology and results and Part 3: the discussion and conclusion. Each part is built up by chapters.

For part 1 consists of chapter 2 and 3. Chapter 2 consists of the theory and history behind the techniques used for the methodology and implementation. The chapter consist of a detailed explanation of how the object detection algorithms work and how the depth of an image is derived. Chapter 3 introduces and explains the datasets used during the implementation.

Part 2 consists of chapters 4 and 5. Chapter 4 explains the methodology and research behind the selected technologies, as well as a deep dive into how each method is implemented. Chapter 5 presents the results of the methods used as well as the results of the proposed framework.

Part 3 consists of chapters 6 and 7. Chapter 6 discussed the results presented in chapter 5, and the effects of the assumptions and choices made during the process. In chapter 7, the conclusion is presented as well as some thoughts about future work.

Part I

Theory and related work

Chapter 2

Theoretical background

This chapter dives into the theory behind the presented methods in chapter 4.

2.1 Artificial neural networks

Artificial neural networks(ANN) has throughout the year become more and more popular when it comes to many prediction tasks. The basic concept is to replicate the way the human brain works. An ANN's fundamental principle is to build a network of many simple units, called neurons[10][14]. A ANN consists of input, output and hidden layers, and is defined by how many neurons, layers and connections between the layers. The connection between each layer is called weights, which is used to store the network. The goal of an ANN is to transform the input so that the output layer can perform a prediction.

2.1.1 Artificial neurons

An artificial neuron, or node, is a mathematical representation of how the neurons in the human brain works, see figure 2.1. A networks input serves as the dendrites and are, together with the weights and a bias, processed in what is called a transfer function as the cell body. The bias is a constant scalar value that is added to ensure as least some of the neurons are activated. The transfer function is the bias added to then added to the weighted sum of the inputs. The output of the transfer function is then processed in the activation function (see the next section).

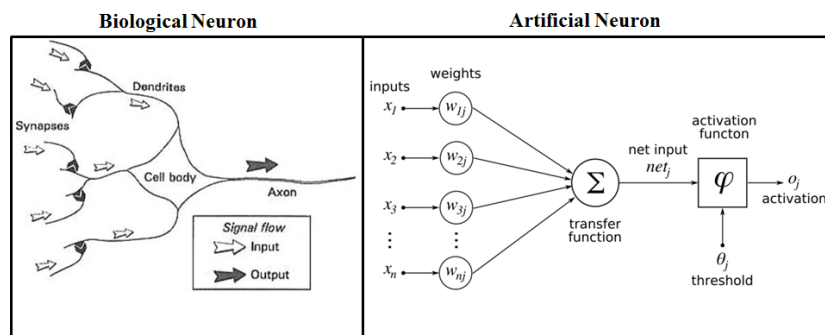


Figure 2.1: An artificial neuron[4]

2.1.2 Activation functions

Activation functions are mathematical equations that defines the output of neural network[14]. Compared to the biological neurons, the activation function is representing the axioms, which determines the rate of a cell is firing. An activation function normalizes a neuron's output often between 0 and 1 or -1 and 1[23]. The simplest of activation functions returns a binary value given some threshold and decides if the neuron is firing or not. Most modern neural networks are learning from more data and therefore uses non-linear activation functions. An important aspect of activation function is that it must be computationally efficient because they are calculated across all neurons, which there can be millions of. Non-linear functions makes backpropagation possible to create deep neural networks due to the functions having derivatives. The most common non-linear activation functions are:

Sigmoid Sigmoid uses smooth grading, which normalizes the output values to probabilities which ranges from [0,1]. A disadvantage of this function is that is can cause vanishing gradients, which is that for very high or low values of X, there is almost no change in prediction. This can cause the network to stop learning further. The mathematical expression as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

TanH The hyperbolic tangent activation function also uses smooth grading, which represent the ratio between the hyperbolics of sine and cosine and returns values withing the range of [-1, 1]. An advantage for this function is that is is zero centered, which makes it easier to map out extreme positives and negatives. This too is considered computational expensive and can cause vanishing gradients. Below is the mathematical expression:

$$\sigma(z) = \tanh(z)$$

Softmax Softmax is a normalized and generalization of the logistic exponential function. The function squeezes a K-dimensional vector z of K real numbers and normalizes it into a probability distribution of K probabilities between [0,1]. The component probabilities will sum up to 1. Therefore, the Softmax activation function has the advantage to handle multiple classes and

is often used in the final layer of a neural network to get the probability of each of the predicted classes. Below is the mathematical expression:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

for $j=1,\dots,K$ where j is the index in the list of the input vector.

ReLU ReLU, or Rectified Linear Unit, rectifies negative values to 0, meaning the network will only learn from positive outputs. A disadvantage of this activation function is the "dying ReLU problem", which means that if the output of a neuron is negative or 0, the network cannot perform backpropagation on that neuron. This activation function is considered computationally efficient and allows for the network to converge quickly. Although it looks linear, the derivative of the function allows for backpropagation. Below is the mathematical expression:

$$\sigma(z) = \max(0, z)$$

Swish The Swish activation function is a newly researched activation function discovered by Google. Swish is simply a self-gated version of the sigmoid activation function. Swish has an advantage of being considered computationally efficient. Looking at figure 2.2, the graph shows that the graph follows similar traits as ReLU, meaning it can converge quickly, but normalizes negative values instead of setting them to 0, which lets the network learn from those as well. Below is the mathematical expression:

$$\sigma(z) = z * \frac{1}{1 + e^{-z}}$$

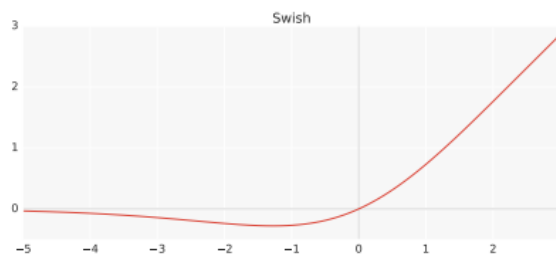


Figure 1: The Swish activation function.

Figure 2.2: The Swish activation function.

2.1.3 Feed-forward neural networks

Feed-forward neural networks is the most common ANN and is often added on top of more advanced networks for prediction purposes. A feed-forward neural network[35] consist of three base components in order to produce an output; the input layer, one or more hidden layers, and an output layer. An example of a multi-layer feed-forward network is shown in figure 2.3. In order to use hidden layers, non-linear activation functions are necessary. A feed-forward network without hidden layers are called a single layer perceptron, which uses linear activation function, while a feed-forward network with many hidden layers is considered a shallow or deep neural net. A network where each neruon from one layer outputs a connection weight to all of the neurons in the next layer is called a fully connected feed-forward neural network.

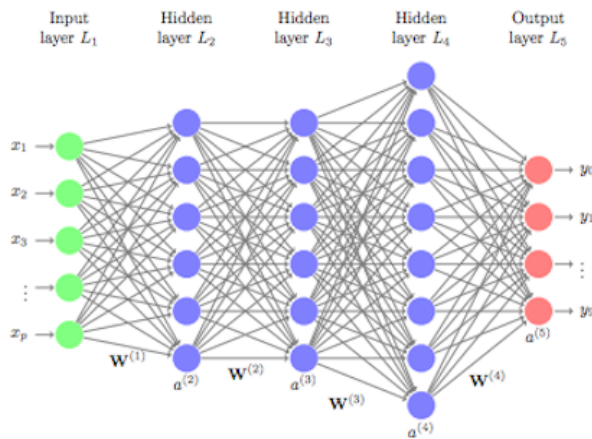


Figure 2.3: A fully connected feed-forward network.

2.1.4 Training a neural network

There exists many type of machine learning methods and researchers are constantly figuring out ways to improve machine learning. Some of the most popular methods used are supervised learning, which is where one shows the network the correct answer, letting it adjust based on if it was right or wrong. This requires labelled data, which is often considered a bottleneck as a network often requires a lot of data in order to produce good results. Another is unsupervised learning, which allows for the model to train itself. Another popular method is called reinforcement learning, where the model don't get

to know the answer during training, but instead is based on a reward system. For this paper, the focus will stay on the supervised learning method using ANNs.

The way a supervised ANN actually learns is by adjusting the connections in form of weights and biases between each neuron and determine the importance of that said neuron, both positive and negative. In supervised learning, the weights are adjusted after the output layer has been processed. Given the supervised nature of having the correct answer, an error, or loss, can be calculated through something called a loss function, which is explained in the next section 2.1.5.

2.1.5 Loss functions

In order for a neural network to learn, each node in each layer must know if they contributed to a better or worse result. This is the job of the loss function and is where the actual learning come into picture. The processed of having and input, run in through the network, and return an output is called forward propagation. After a forward propagation is done, the loss function calculates the difference between the predicted value and the truth label (the correct prediction). This calculated value is then sent back through the system to correct the weights. This process is called backpropagation (see section 2.1.6). Intuitively, the goal of training an ANN is to minimize the loss. There exists many more loss functions, but the ones listed are those relevant for this paper. Mainly, there are two different types of loss functions; classification and regression loss[41]. The main difference between the two is that the classification loss functions predicts a continuous value in the form of a probability for each predicted class and evaluated by accuracy, while the regressions loss functions predicts discrete values in form of integers.

Classification loss functions:

The truth labels for classification losses are on-hot encoded, meaning the every class except the correct one has a value of 0. That is, if the model is trained on recognizing [cat, dog] and it processed an image of a cat, the one-hot encoded vector would look like [1,0], meaning that the probability of it being a cat is 1, and 0 for a dog. For all classification loss functions, layer before the loss is calculated must use the Softmax activation function in order to output the predicted probabilities.

Cross-entropy The cross-entropy loss function measures the performance of a classification model between each predicted probability and the true probability. The loss is measured by the negative sum of all entropies between the predicted probability and the actual value. The entropy for each prediction is calculated by multiplying the truth label by the logarithm of the predicted probability. Below is the mathematical expression:

$$L(p, y) = - \sum_i y_i \log(p_i)$$

for $i=0,..M$, where M is the number of classes, y is the truth labels and p is the predicted probabilities.

Focal The Focal loss is a version of the cross-entropy loss function and is meant to help ease predictions where there are unevenly balanced training set, as well as for sparse data, which means that is is great for big dataset with many classes. The focal loss was first introduced for loss in dense object detection. Using normal cross-entropy, the model will get a high certainty of common cases, but will be very unsure of uncommon cases. To solve this, a weighted class α -balanced focal loss is introduced. α is a scaling factor which is decided based on the balancing of the dataset. The focal loss has the purpose of down-weighting easy examples and let the focus of the training stay on the hard negatives. The down-weighting is decided by a modulating factor $1-p_i^\gamma$, where p_i for a certain class and gamma is a focusing parameter that smoothly adjust the rate of down-weighting. This focusing parameter is a usually a constant value. Below is the mathematical expression:

$$FL(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

for $i=0,..M$, where M is the number of classes and p is the predicted probabilities.

Regression loss functions:

Unlike the classification loss functions, the labels for regression does not need to be one-hot encoded, meaning that the goal of the network given an input and truth label is to figure out that single truth.

Mean square error Mean square error (MSE) is the most commonly used regression loss function. MSE is simply the sum of squared distances between the target variable, and the predicted values. As the error is squared, a problem with the mean square error loss function is that wrongly predicted

values will result in extremes, meaning the loss is usually either very high or very low, making it vulnerable to outliers. Below is the mathematical expression:

$$MSE = \frac{\sum_{i=1}^n (y_i - y_y^p)^2}{n}$$

for $i=0,..,n$, where n is the number of inputs from the previous layer and p is the predicted probabilities at index i .

Mean Absolute Error Mean absolute error (MAE) is considered more robust as the MSE, as it doesn't square the difference between the predicted value and truth label, making it less susceptible for extremes. The absolute difference measures the average magnitude of errors in the set of predictions. Below is the mathematical expression:

$$MAE = \frac{\sum_{i=1}^n |y_i - y_y^p|}{n}$$

for $i=0,..,n$, where n is the number of inputs from the previous layer and p is the predicted probabilities at index i .

Smooth L1 The smooth L1 loss function is also called the Huber loss and is a combination between the MSE and MSE. As the goal of training a network is to minimize the loss, the MSE is good for values less than 1 or some selected threshold δ , as this minimizes the loss even more for good predictions. Therefore, the Smooth L1 loss function uses a similar version of MSE below some value δ , and similar version of MSE above it. Below is the mathematical expression for a single prediction:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta & \text{otherwise.} \end{cases}$$

where $f(x)$ is the predicted probability and y is the truth label.

The total loss of an output is then:

$$SmoothL1 = \sum_{i=0}^n L_\delta^i,$$

where $i=0,..,n$ and n is the number of predicted values

2.1.6 Backpropagation

So far, the metric for determining how the model so far performs is explained. This section will explain the networks learning process given some loss. The way the network is trained is by passing the estimated loss backwards, letting the weights adjust based on some optimization method. The most commonly used optimization method is called Stochastic Gradient Decent (SGD). More about optimizers in the next section. The backpropagation is based on a rule called weight update rule, which simply states that the new weight equals the old weight added the learning rate and some estimated adjustment calculated using the loss function. Figure 2.4 shows an example of a backpropagation on a small network. The next section will discuss the formulas used for different optimizers to adjust the weights and biases.

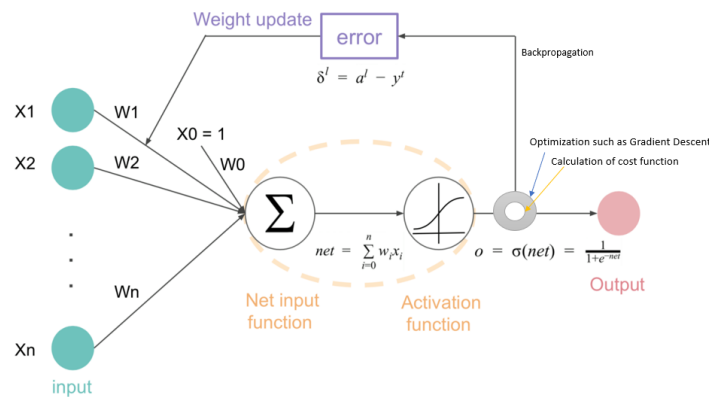


Figure 2.4: A simple illustration of a SGD backpropagation

The general update rule of each weight and bias is shown below, but some optimizers also adaptively scales the learning rate.

$$w_k \rightarrow w'_k = w_k - \eta \nabla L$$

$$b_k \rightarrow b'_k = b_k - \eta \nabla L$$

2.1.7 Optimizers

The task of an optimizer is to determine a how the weight Some are used to determine how to process the backpropagation using the error in order

to adjust the weights and biases, while some are used to adaptively update hyper-parameters. The explained optimizers to come are the ones relevant for this paper. All hyperparameters mentioned in this section is further explained in section 2.1.8.

Stochastic Gradient Decent As mentioned before, the Stochastic Gradient Decent (SGD) is the most commonly used optimizer used for back-propagation and stands as the classical foundation of using gradient descents[27]. The objective of the using gradient descents are in basic words to descend the slope of the derivative, i.e. to find and push the weights towards an approximate minimum. SGD uses the gradients of the loss function ∇L with respect to the weight for a small sample of randomly chosen training inputs and uses the average gradient to quickly get a good estimate of the overall gradient. ∇L and the learning rate η are used as follows:

1. SGD randomly picks out a small number of m randomly chosen training inputs as a mini-batch $m = [X_1, X_2, \dots, X_m]$.
2. Apply ∇L_{x_j} on the sample size and return the average value. As long as the sample size m is large enough, ∇L_{x_j} is esimated to be roughly equal to the average overall ∇L .

$$\nabla L = \frac{\sum_{j=1}^m \nabla L_{x_j}}{m} \approx \frac{\sum_x \nabla L_x}{n}$$

3. This approximate gradient is then used in the update rule explained in the previous section for all nodes and biases.

AdaGrad The learning rate being a hyperparameter is a constant set before training and is often set to a low value. This could cause a problem resulting in some neurons not learning quickly enough. As a result of this, AdaGrad was introduced by Duchi et al.[3]. The basic principle is to adaptively scale the learning rate to the gradient. The equation of AdaGrad's weight adjustment Θ_{t+1} a certain time-step t is as follows:

$$\Theta_{t+1} = \Theta - \frac{\eta}{\sqrt{\epsilon I + \text{diag}(G_t)}} g_t$$

where η is the initial learning rate, ϵ is some small value to a void division by zero, I is the identity matrix, g_t is the gradient estimate in time-step t and G_t is the sum of the squared outer products of the gradients until time-step

t.

Adagrad is especially effective in sparse datasets due to the scaled learning rate making frequent examples, but could for some cases cause the network to stop learning as the learning rate becomes really small.

RMSProp RMSProp is short for Root Mean Square Propagation and is similar to the SGD, but uses the momentum hyperparameter as well as adaptively updating the learning rate. The goal of an optimizer is to explain to adjust the weights and reduce the loss. Using standard gradient descent the adjustments using the gradients will oscillate back and forth moving closer and closer to the approximate minimum. The goal of RMS drop is to minimize this oscillation. Below is the mathematical formula:

$$v_t = \nu v_{t-1} + (1 - \nu)g_t^2$$
$$\Theta_{t+1} = \Theta_t - \frac{\eta}{\sqrt{v_t + \epsilon}}g_t$$

where η is the initial learning rate, ν is the momentum, g_t is the gradient at time t , v_t is the exponential average of the squares of the gradients along the weights.

RMSProp remove the problem of AdaGrad with deminishing learning rates as it uses the momentum to slowly adjust the learning rate. RMSProp also supports mini-batches, which speeds up the gradient descent.

Adam Adam, or Adaptive Moment Optimization, combines the properties of AdaGrad and RMSProp and uses multiple hyperparameters to control the exponential reduction of the moving averages. The moving averages is a set of gradients at time t . Adam compute the exponential average of the gradients v_t as well as the squares of the gradient s_t for each neuron. The learning rate is then multiplied with the exponential average of the gradients, and then divided by the root mean square of the exponential average of square gradients.

$$v_t = \beta_1 v_{t-1} + (1 - \beta_1)g_t$$
$$s_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$
$$\Theta_{t+1} = \Theta_t - \eta \frac{v_t}{\sqrt{s_t + \epsilon}}g_t$$

where β_1 and β_2 are hyperparameters.

2.1.8 Hyperparameters

The general nature of hyperparameters statically set the foundation of how the network should behave and are decided before the training starts.

Epoch The epoch has the function of telling the network the number of iterations it should train on the dataset. Due to the large nature of many dataset, it is normal to let each epoch choose a randomly shuffled sample set with a specific batch size to serve as a mini-batch, to try and generalize the gradients and speed up the learning process. The model is the train on the the dataset with for a certain number or steps each epoch.

Learning rate The learning rate is what defines how quickly a model is learning and is considered the step size of each updated weight. A large learning rate will make the gradient descent fluctuate back and forth around the approximate minimum, while a too small learning rate will take too long to reach it. Smaller learning rates are still used due to the nature of it moving towards the minimum. Mentioned in the above section, there are several ways to adaptively adjust the learning rate for each node, but the initial learning rate is still often adjusted during the training to push the network into even further reduce the loss. A typical method is by after a number of epochs, reduce the learning rate by some factor to anneal it over time. This method is called *stepwise annealing* . Another annealing method is the *cosine annealing*, which reduces the learning rate with the number of epochs based on the cosine function. Another much used method is the *cycling learning rate*, which given some boundaries during the epochs reduces the learning rate down to a minimum, and then jumps it to its initial value. This method solves the problem of a method getting stuck in local minimums, which is good for the generalization of the model.

Momentum As mentioned above, the momentum is used during the optimization process of the model. The momentum is often referred to as the learning rate of the learning rate. During gradient descent, the learning rate will cause the loss to oscillate towards a minimum. Looking at the loss space (see figure 2.5), the learning rate will make the gradient oscillate in one axis, but the momentum will push the loss closer to the minimum in the other axis, making the learning process faster.

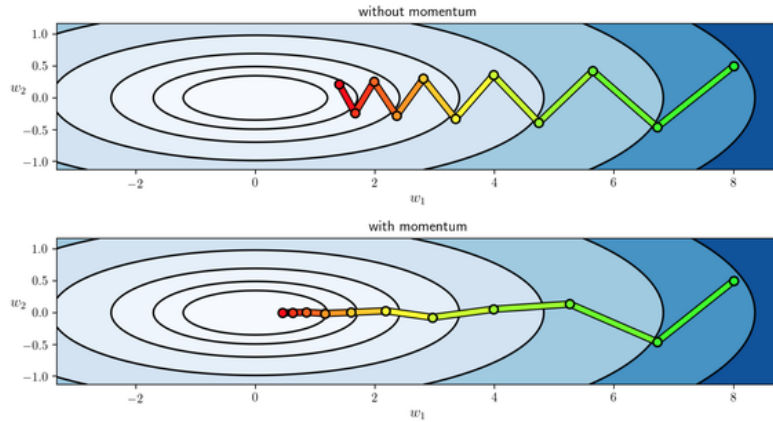


Figure 2.5: Learning rate and momentum in loss space

As both the momentum and learning rate could cause the model to overshoot the minimum, Nesterov Accelerates gradient (NAG) is introduced to solve this problem by making the momentum smarter. It uses the knowledge of the previous gradient step and moves it in the direction of the previous gradient instead (see figure 2.6).

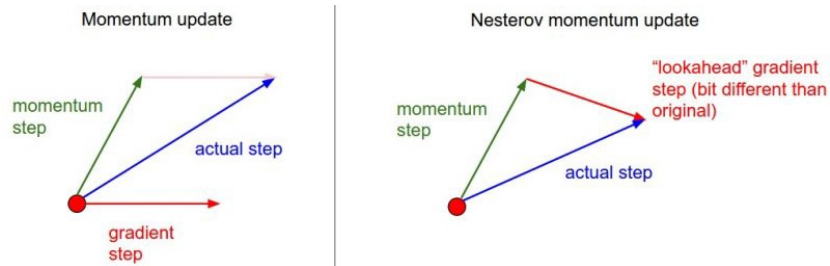


Figure 2.6: Shows how NAG uses the previous gradient step to push it in the right direction.

2.2 Convolutional neural networks

So far the general perception of training an artificial neural network has been explained, that is, for each iteration, some weights are adjusted to minimize the loss from the truth label. This process is called the classification process. A convolutional neural network (CNN) analyzes the input through feature

learning and can successfully capture the spatial and temporal dependencies in a dataset through relevant filters and kernels[45].

In the recent years, convolutional neural networks has show to be best suited to train models of a big input sizes. In a traditional fully-connected neural networks, a weight is passed from each neuron in one layer to each neuron in the next. Now, if the model processes inputs of thousands of nodes, each layer would have to process the weights of all of these inputs, which would require huge resources as well as time-consuming. Therefore, instead, a CNN processes multi-dimensional inputs. For an image this means that instead of processing a flattened image, the CNN process each image in 3D, the width, height and channels. The number of channel decides how many values each pixel contains - 3 for a RGB image.

By maintaining the spatial structure of the input, the CNN can be used to recognize specific features in for this case which makes them very well suited for processing images. A CNN uses the output of the convolutions to predict the classification through a fully-connected neural network applied at the end.

Architecture A CNN contains of three main parts, the input layer, the features extraction and classification, and the output layer. Ever since the first CNN's were designed, the goal of the architecture is to optimize the feature extraction both in form of evaluation metrics, but also speed. Figure 2.7 shows an example of a convolutional neural network. Famous CNN architectures such as LeNet, AlexNet and VGGNet has been the backbone for newer state-of-the-art model architectures and is still used today.

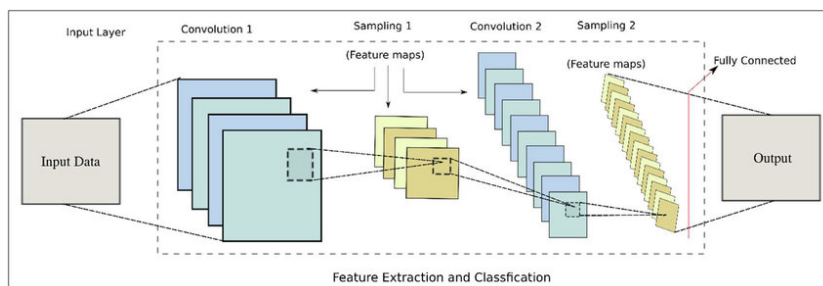


Figure 2.7: An example of a CNN's architecture.

Filters and kernels A convolution means to apply some function (filter) on some values in a kernel and produce some new value. The kernel is simply a window of size $k \times k$ which is slid across the input data applying

the chosen filter for each kernel. The filter is what actually decides how the input data should be processed and is considered the activation function of the convolution. The filter aggregates the values in the kernel, pass it through the function and returns the value to the next layer. A common filter is ReLU, and same as for ANN, simply return the maximum value of the kernel. By combining it with a scale $\alpha \leq 0$, the ReLU filter becomes scale-invariant. The function is as follows:

$$f(a, x) = \max(0, ax)$$

ReLU, when used in CNN, keeps all the same advantages and disadvantages as for traditional ANNs.

Figure 2.8 shows an example of a random filter applied to a kernel. The stride of a kernel decides the interval between neighboring kernels, e.g. a kernel with stride 1 convolutes each neighboring kernel.

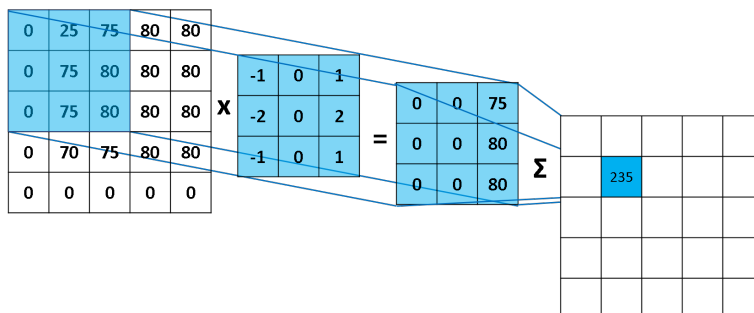


Figure 2.8: A kernel applied to an image.

As one can see in the figure above, without applying further rules, the output of a kernel with a stride reduces the image size by leaving one row and column empty. A padding can be applied by adding zeroes in said positions. The objective of the convolutional operation is to extract the high-level features from the input image. With multiple convolutional layers, the architecture adapts to the convoluted features as well.

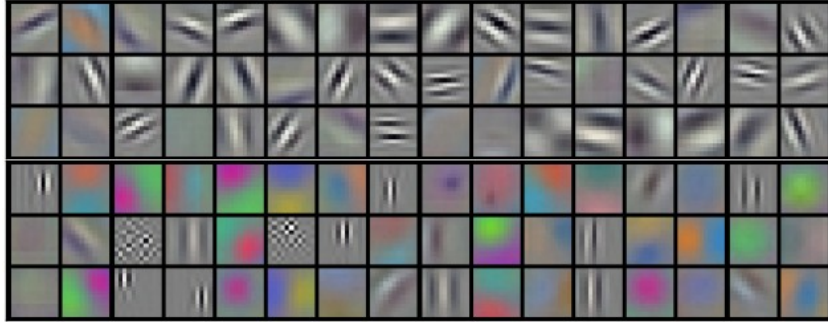


Figure 2.9: Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size $[11 \times 11 \times 3]$, and each one is shared by the 55×55 neurons in one depth slice[2]

Pooling layers Similar to the convolutional layer, the pooling layer is responsible for reducing the spatial size of the convolved feature. This is to decrease the computational power required to process the data through dimensionality reduction. Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model. The most used pooling is the max pooling. This returns the maximum value from the portion of the row. Max pooling also reduces the noise of the data, as only the dominant features are extracted.

Upsampling layers As the pooling layers reduces the spatial size of the image, the upsampling layers has the function of increasing the size of the image to the desired size. The most common way to do this is by using interpolation and resampling such as nearest neighbor.

The result of the convolutions and pooling and upsampling layers will serve as the input layer for a fully connected layer in a feed-forward neural network. In order to get the correct dimensionality before feeding it into a fully-connected layer, the convolutional output layer is flattened into a $n_{feat} \times 1$ array. The output layer must be reduced to the same number of nodes as the number of potential classes, e.g. if the model predicts three classes, the output layer must be a 3-neuron layer. Based on the optimization method chosen, the loss of the function is computed, and the weights are updated.

2.3 Object detection

Compared to the image detection classifiers, where the goal is to predict the class of one object in an image, object detection also involves identifying the position of one or more objects predicted in the image. Object detection classifiers produces a list of objects presented in the image with corresponding scores, as well as an aligned bounding box indicating the position and scale of every object. Using CNNs in object detection trains two networks, one for classifying the objects in an image, another for fitting the box around each object and given that this is supervised learning, the training data needs both truth labels for each object, as well as the location in the image in form of the bounding box. The truth label for the bounding box is call *ground truth box*.

2.3.1 Evaluation metrics

In order to know if a model is well trained or not, several evaluation metrics are defined based on the predictions. A simple classification task is simple to evaluate, but accounting for the object detection, a confidence score is introduced for each bounding box of the object detected[1].

IoU Intersection over Union is an evaluation metric that quantifies the similarity between the predicted bounding box and the ground truth box (gt) in form of a probability measure. The higher the IoU score, the closer the two boxes are to each other. The IoU measures the overlap/intersection of the bounding boxes divided by the union.

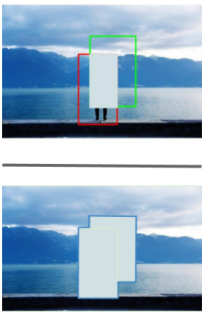
$$IoU_{pred}^{truth} = \frac{truth \cap pred}{truth \cup pred} = \frac{\text{img1}}{\text{img2}}$$


Figure 2.10: An example of the IoU between two bounding boxes[1].

Predictions To decide if the bounding box prediction is good enough or not, the IoU is measured, and based on a set threshold, values above this threshold is considered positive predictions, and those below are vice versa negative predictions. In the next sections, some evaluation metrics are calculated using *true positives*(TP), *true negatives*(TN), *false positives*(FP) and *false negatives*(FN). A true positive denotes that the object is there, and the IoU is above the threshold. True negatives denotes that the object isn't there, and the model does not detect it. False positives denotes that the object is there, but the IoU is below the threshold. False negatives denotes occurrences where the object is there, but the model doesn't detect it, meaning the predicted bounding box has no prediction.

Accuracy The accuracy is the percentage of true positives plus true negatives divided by every prediction. This is often misleading when dealing with imbalanced datasets.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision The precision is the probability of the predicted bounding boxes with respect to the actual ground truth boxes. This metric is in other words the probability of when an object is detected, the model is correct.

$$Precision = \frac{TP}{TP + FP}$$

Recall The recall is the rate of true positives, often referred to as the sensitivity of the predictions. It measures the probability of ground truth objects being correctly detected, i.e. how many of the actual objects did the model detect.

$$Recall = \frac{TP}{TP + FN}$$

Average precision AP The average precision is an evaluation metric that measures the performance of the model as it returns a single value that accounts for both the precision and recall. The average precision is also known as the area under curve (AUC) and measures sum of the maximum precision p for any recall \tilde{r} multiplied by the change in recall $\geq \tilde{r}$.

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

$$p_{interp}(r_{n+1}) = \max(p(\tilde{r}))_{\tilde{r} \geq r_{n+1}}$$

Mean Average precision mAP The mean average precision is simply over N classes, the mAP averaged the AP over all the N classes, i.e. the total performance for all classes.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i$$

2.3.2 Feature extraction

The research in object detection is still an ongoing process. In order to avoid having to train a network from scratch every time on either new datasets, finetuning a model towards the same dataset, or introducing new model architectures, one can use already general and robust pretrained networks to help speed up the process of learning. Using a pretrained network as a foundation for what the new task is, is called using a *backbone*. When it comes to object detection in images, as explained in the previous sections, the goal is to fit a bounding box around a classified object. The CNN has as a task to extract features of an image, and learn the model on those instead of the input images directly. Using an already robust and generalized model as a backbone and extract the most important features of it will make the new keep the important features of said backbone, and make the process of both detection and localization of objects much faster. ImageNet[15] is one of the most used backbones today as the network is trained on over 14 million images containing almost 22 000 classes. Another commonly used backbone are models CoCo dataset. CoCo stands for Common Objects in Context and contains of both training and validation data of over 120 000 images with multiple bounding boxes for each image for around 100 common objects.

The difference between fine-tuning a model and feature extraction in generally speaking either to, train the model further using similar data with the corresponding classes, typically a sample set of the classes already used in the model, or extract the important features from the network and use that as a foundation for training a new network. The first is called *fine-tuning*. A typical example using the MS CoCo dataset, is to fine-tune the model on new data to make it better at predicting less classes, e.g. busses, instead of all the 100 classes. *Feature extraction*[9] reduces the dimensionality of the original input data so it is more manageable for further training which is more commonly used when training on new datasets with new classes so

that the model already has a common opinion of what to look for both in terms of classification and localization of the bounding boxes.

2.4 Depth estimation

In order to convert the position of an object in an image to real distances, depth estimation is needed. There are several ways to approximate the depth of an image. In recent years, many have tried to estimate depth using deep learning networks as a lot of data can be collected using cameras and LiDAR[36] combined. The camera take a picture of the environment, and the LiDAR obtain the distance of each point.

2.4.1 Stereo rectification

Stereo rectification[7] the more traditional of obtaining distance to an object and is the task of using two images, detect the same feature in both objects and calculate the distance. In computer vision, the stereo vision uses triangulation based on epipolar geometry to determine the distance to an object. One of the tasks while using multiple cameras is to find the corresponding feature in both cameras. This problem is known as the *correspondance problem*. If the images has no geometric distortion, i.e. is in the same epipolar plane, the calculation is made through linear transformation. In general, affine transformations are made done by rotating X and Y axis to put the images on the same plane, scaling the image to the same size and rotate the Z axis to skew the image making the images align directly. If each camera is calibrated, i.e. the intrinsic orientation parameters (IOP) are known, the essential matrix provides the relations between the cameras. If the cases lacks this essential matrix, a fundamental matrix is derived by using at least some point correspondence. Below are two figures [2.11][2.12] to show the rectification and the transformation.

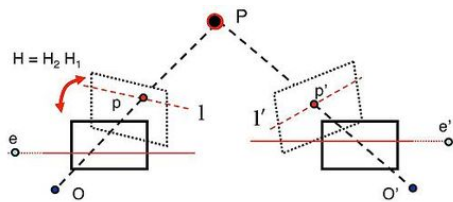


Figure 2.11: Image rectification using epipolar lines

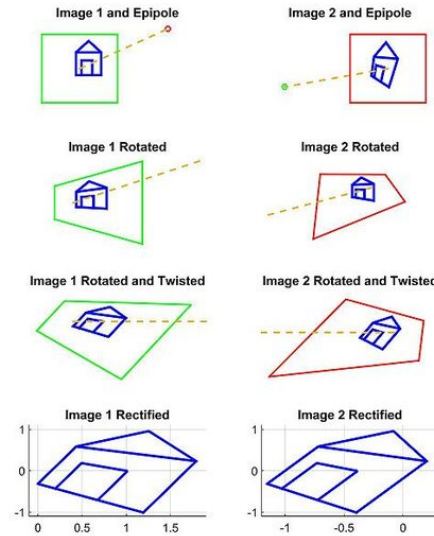


Figure 2.12: Transformation of the images.

2.4.2 Supervised depth estimation

Using labeled data, supervised depth estimation models have shown promising results in learning the relationships between color images and their corresponding depth. Different approaches are used in order to obtain good results, such as combining local predictions, non-parametric scene sampling which is a method of using scene parsing[30], i.e. spatial segmentation through CNNs, to try and classify the spatial correlation of pixels in an image, see figure 2.13.

Although the LiDAR data can serve as the ground truth for the distance, one can see from just the scene sampling that the distance alone is not enough to learn the spatial correlation in an image, which results in a lot of data need to be manually labeled, and serves as a bottleneck for fully-supervised methods.

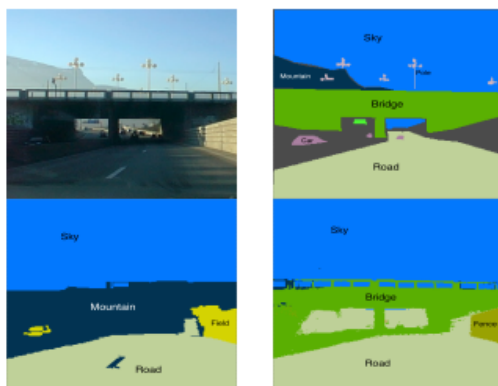


Figure 2.13: Top left: Query image; Top right: ground truth;bottom left: Superparsing method; bottom right:Sampling parsing method[30]

As a result of this, there has increasingly been proposed methods that exploits weakly supervised training data with other spatial correlations, such as object size, sparse ordinal depths, appearing matching and synthetic data generation in addition to the depths alone.

Recent work has shown that conventional structure-from-motion (SfM) pipelines can generate sparse training data for both the camera pose and depth[20]. SfM is typically run as a pre-processing step before the learning process is begun. SfM is a photometric range imaging technique for estimating three-dimensional structure from two-dimensional images. The principle of the technique is to find the correspondence between images in form of features such as corners. One of the most used feature detectors is the SIFT algorithm[32], which is a scale-invariant feature algorithm.

2.4.3 Self-supervised depth estimation

To help solve the problem of not having enough ground truth data, an alternative is to let the model use image reconstruction for the supervision. For this the model is given a set of images as input, either in the form of monocular sequences, or as stereo image pairs. By hallucinating the depth for each image and projecting it into nearby views, the model is trained by minimizing the image reconstruction error. In other words, the model tries to reconstruct the image by trying to figure out the spatiality between the pixels.

Self-supervised monocular training For a monocular self-supervised training model, temporal frames in the form of videos are used as the training data. In addition to predicting the depth, it also proposes a model to predict the camera pose, which is used during training to help constrain the depth estimation. This can be challenging in the occurrences of moving objects.

The basic principle of a monocular training technique is to apply some motion explanation mask allowing the model to ignore specific regions that violates the assumption of rigid scene motions. Later models have proposed more sophisticated motion models using multiple motion masks as well as learn from the occurrences of both rigid and non-rigid components to derive a *flow estimation*. In addition to this, self-supervised training typically relies on making assumptions about the appearance and material properties between frames. By also considering these properties with by optimizing the local structure based appearance loss, the model can predict the appearance in one image, with the view point of another image. This process is called an image synthesis.

Self-supervised stereo training Self-supervised stereo training has the input of two images in a stereo pair and is used during training to predict depth disparities, that is a pixel representation of the depth in an image. These disparities together with a left-right depth consistency term, can be used to train a monocular model. These stereo-based approaches has been extended with semi-supervised data, such as mentioned above for additional consistency and temporal information. The result of self-supervised stereo training models can be used for real-time depth predictions.

2.4.4 Self-supervised monocular depth estimation

This section describes the steps of using the introduced training methods to predict a depth estimation. By combining the process of image synthesis with the predicted depth disparity, the model extracts an interpretable depth for each pixel from the network. The depths are not certain, meaning that each depth interpretation could contain a large amount of possible incorrect depths per pixel which could, in principle, reconstruct the image correctly given the relative pose between two images. This is where the combination of the mono and stereo self-supervised training methods are combined, as the stereo methods typically addresses this ambiguity by enforcing a smoothness in the depth maps, as well as computing photo-consistency when solving per-pixel depth through global optimization[46].

One of the monitored loss of the model is called the photometric reprojection loss and is measured by expressing the relative pose for each image $I_{t'}$ with respect to the target pose of image I_t , where the time/iteration interval is $T_{t' \rightarrow t}$. With the prediction of the depth map D_t , the model selects the minimized error for each pixel, giving the minimized photometric reprojection loss:

$$L_p = \min(pe(I_t, I_{t' \rightarrow t}))$$

where

$$I_{t' \rightarrow t} = I_{t'}(\text{projection}(D_t, T_{t' \rightarrow t}, K))$$

Here, pe is the photometric reconstruction loss e.g. the Manhattan[11] distance in pixel space. $\text{proj}()$ are the resulting 2D coordinates of the projected depths D_t in $I_{t'}$ and $\langle \cdot \rangle$ is the sampling operator. K is in this case the notation used for the pre-calibrated IOPs.

The photometric reconstruction loss between the input and the output is calculated as follows:

$$pe(I_t, I_{t' \rightarrow t}) = \frac{\alpha}{2}(1 - SSMI(I_t, I_{t' \rightarrow t})) + (1 - \alpha)\|I_t - I_{t' \rightarrow t}\|$$

SSMI is short for Special Sensor Microwave Imager and is the data product of a Remote Sensing System e.g. LiDAR and used unified, physically based algorithms[40].

The last of the losses measured is the loss of an edge-aware smoothing:

$$L_s = |\delta_x d_t^*| e^{-\delta_x I_t} + |\delta_y d_t^*| e^{-\delta_y I_t}$$

where $d_t^* = \frac{d_t}{I_t}$ and is the mean-normalized inverse depth.

Figure 2.14 shows the proposed pipeline in[6] for a self-supervised monocular training depth estimation network.

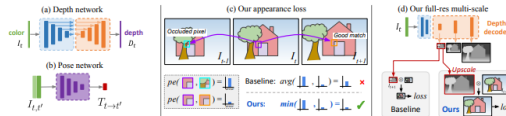


Figure 2.14: [6](a)Depth network: Uses a standard fully-convolutional network to predict depth. (b)Pose network: Predicts the pose between a pair of frames. (c)Per-pixel minimum reprojection: Matches each pixel to the view in which it is visible, leading to a sharper result. (d)Full-resolution multi-scale

Chapter 3

Datasets in neural networks

The following chapter presents the datasets used for object detection, depth estimation and testing.

3.1 Object detection

As we now know, the goal of an object detection classifier is not only to classify image, but also predict the object with a bounding box. This requires labeled data with corresponding ground truth boxes. For many years, different companies and associations have tried to develop huge, generalized datasets. Among them are the PASCAL VOC dataset, which the latest version from 2012 contains more than 20 classes for more than 9000 labelled images containing one or more objects. The most common way to evaluate each iteration of a dataset is through challenges, where developers and researchers can try and optimize models in order to reach the highest possible score, often through the mAP metric over all predictions.

COCO Briefly mentioned in section 2.3.2, the COCO dataset stands for Common Objects in Context and was introduced by Microsoft in 2015[42] with a goal of advancing the state-of-the-art object detection models by gathering images of complex everyday scenes in a natural environment. The dataset contains photos of 91 object types with a huge amount of 2.5 million labeled instances in 328k images. The dataset was collected through a crowd worked environment using an interface to the user to categorize the image. For this paper, the COCO dataset pretrained model is used to recognize the traffic lights, and used as training data for the selected backbone.

MTSD MTSD[5] was newly introduced by Mapillary in 2019 and is the current largest traffic sign database with over 300 000 labeled instances in over 100 000 images with more than 300 traffic sign classes. This dataset is the most diverse traffic sign dataset containing images across the whole world and is evaluated to be a strong baseline for detection and classification. In addition to the large scale of the dataset and unlike many other traffic sign databases, it also contains attributes, which can be included in the input layer during a training process. The dataset is freely available for academic research and can be requested through their website. A note towards the dataset is that it contains large, natural images with traffic signs in it, requiring more resources to process than some other datasets. Due to the traffic signs varying across different countries, there is a traffic sign class taxonomy. To account for this, the labels with the initial same purpose is labeled into the same class with a taxonomy notation behind it. An example of this is for is *regulatory-stop-g1*, where g1 is the notation of taxonomy.

GTSRB The GTSRB is short for the German Traffic Sign Recognition

Benchmark and was first introduced in 2010 and has more than 50 000 images for more than 40 classes. The image database consists of really small images of only a size 30x30, with corresponding bounding boxes. The dataset is freely available for download through their website[25].

3.2 Depth estimation

KITTI The KITTI takes advantage of their autonomous driving platform Annieway to retrieve their data[12]. The car used for data collection uses two high-resolution color and greyscale video cameras and a Velodyne laser scanner for ground truth distances as well as a GPS localization system. The car drives in the mid-city of Karlsruhe, Germany, on highways and rural areas. For each image, up to 30 pedestrians and 15 cars are available per image. An optimized version towards depth estimation will serve as the dataset for the monocular network to predict the depths for this paper’s proposed map layer.

3.3 Test dataset

Mapillary In addition to the traffic sign dataset, Mapillary also has a huge crowd-sourced street-level image database from all over the world[19]. This is available for developers through an API and makes requests based on area, a certain sequence or a single image possible. The return of the request is in the form of GeoJSON[13] . These images contains geospatial properties as well as the camera angle of the image. As the image database consists of crowd-sources images, the quality of the images varies a lot, often resulting in a lot of distortion. The proposed model architectures proposed in the next chapter will predict on these images and used for geospatial location of the image. A note to this dataset is that it does not contain the IOPs nor any labels, it is purely an image taken with a timestamp and location.

State Highways Authority As the goal of this paper is to produce a map layer for intersections, the Norwegian State Highways Authority API is used to find the closest intersection for the processed. The default map projection of this API is UTM zone 33[31] with the geodetic datum WGS84[37]

Part II

Methods and implementation

Chapter 4

Methodology

This chapter dives into detail on how the methods work and what parameters are used in the implementation, as well as some comparisons of how the techniques perform up against one another.

4.1 Network architecture

4.1.1 Brief history of object detection

Object detection is a widely researched topic even before deep learning got introduced by Krizhevsky et al. in 2012[47]. Early object detectors were based on manually crafted features. The sliding window classifiers were one of the first object detector, such as Haar-features, Non-maximum suppression, Histograms of gradients and more. After several years, Uijlings et al. in 2013, proposed a better algorithm based on regional proposals, the selective search. Instead of a sliding window, the proposed regions with high "objectiveness" were chosen. In the same era, the first deep learning neural networks were proposed by Krizhevsky et al., introduction AlexNet. The first deep learning neural network object detectors were based on a two-stage method, with a pipeline of given an input image, propose the regions and classify the proposed regions in form of an output.

An architecture called R-CNN[33] was proposed by Girshick et al. in 2013 and stood as a big step towards a new direction as up until then, other object detection architectures had plateaued trying to train on the PASCAL VOC dataset. R-CNN was the first region based CNN architecture. R-CNN combines two key techniques; apply a high-capacity CNN to the bottom-up selection search region proposals in order to localize the object and then fine-tune a supervised pre-trained neural net towards their domain.

In 2015, Fast R-CNN was proposed by the same team which used a CNN to both the proposed selective search regions as well as the classification. The same year, Faster R-CNN was proposed by Ren et al.[34] which introduced the first architecture to use a CNN to fully propose the regions, as well as the classification, meaning no more selective search.

In the later years state-of-the-art architectures such as Single-shot detectors (SSD) and YOLO - You Only Look Once have been designed. The unique approach of the Single-shot detectors is that it through several feature extraction CNN layers, proposes both the classification as well as the location of the object in the same network. SSD is considered a generally fast and moderately accurate model. YOLO uses the basic principle of only looking at just a single scale of features and a fully connected layer, and is considered one of the fastest architectures.

In the recent years, the use on advanced CNN architectures has blown up due to the large access of training data, which further requires good and

robust network architecture both in terms of computational resources, as well as the predictions. The next section will look into the performance of different architectures.

4.1.2 Review of network architectures

The list of potential networks to use for a traffic sign detection purpose could be endless, but looking at recent research papers comparing different architectures, some of the current famous and best will be compared. As a baseline for the architectures, the mAP will be used as the performance metric on the COCO dataset. For most part, the architectures proposed are designed by companies like Google Research Brain, Microsoft Research and Facebook AI Research.

Object Detection on COCO test-dev

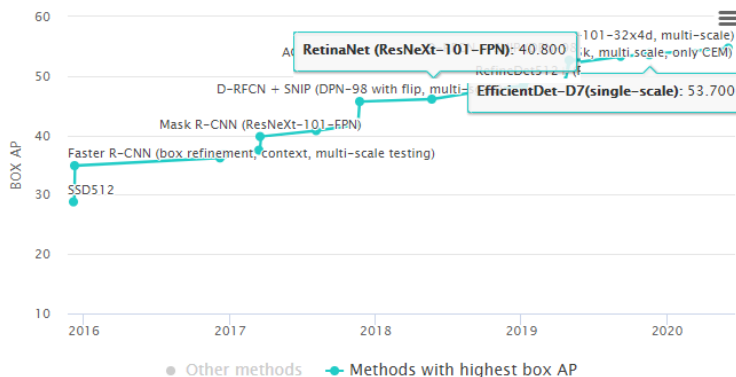


Figure 4.1: The current best performing network architectures on the COCO dataset (accessed 25.06.2020)

Figure 4.1 shows the current best performing network architectures at the moment. Now, as the model shows, there are a lot of variations using the same essential principles. For the comparison below, only some pros and cons with the architectures will be , but the paper will only go further into detail for the selected architecture.

Mask R-CNN Mask R-CNN was first proposed in 2017 by the Facebook

AI research team[17]. The basic principle of a mask R-CNN is to efficiently detect objects in an image while simultaneously generating a high-quality segmentation mask for each instance. In addition to detection classes and bounding boxes, the mask R-CNN network can easily generalize other tasks, such as estimating poses. The year of release, the mask R-CNN outperformed all proposed networks in the COCO suite challenges in 2016.

RetinaNet Following the year of the mask R-CNN, the Facebook AI Research team proposed yet another network architectures, the RetinaNet[44]. RetinaNet moved away from the current best performing two-step region-based approach. Instead, the model proposes a one-stage detector that is applied over a regular, dense sample of possible locations. This proves to be much faster than the two-stage architectures. In addition, the architecture was the first to propose a the focal loss, instead of the standard cross-entropy loss function. The results of Retinanet shows that it is faster, but still surpassing the other state-of-the-art detectors in terms of accuracy.

ResNet ResNet was proposed in 2015 by the Microsoft research team[18]. The model proposes a residual learning framework that are much deeper, e.g. more layers, than previously proposed methods. The network reformulates the network’s function of a layer by using an output of a layer as learning residuals referenced to other layers, i.e. the output of a layer is not only passed to the next, but several layers down. This improves the relative improvement of each layer. A ResNet network is often used combined with a number, e.g. ResNet18. This number represent the number of layers in the network. Although the deep nature of the network, the relative improvements speeds up the learning process. Building upon the deep residual networks, many architectures has been design, performing well in terms of accuracy and speed.

EfficientDet The EfficientDet model design is one of the most recent additions of state-of-the-art model architectures. It was proposed late 2019 by the Google Brain Research team[22]. The architecture proposes a weighted bi-directional feature pyramid network (BiFPN)4.1.3, which allows for easy and fast multi-scale features fusion. In addition the network uses compound scaling to uniformly scare the resolution, depth and widths for all backbone, feature network and class predictions at the same time. Tested on the COCO dataset, figure 4.2 shows that EfficientDet perform better than the above mentioned networks both in terms of latency, accuracy and the number of input parameters.

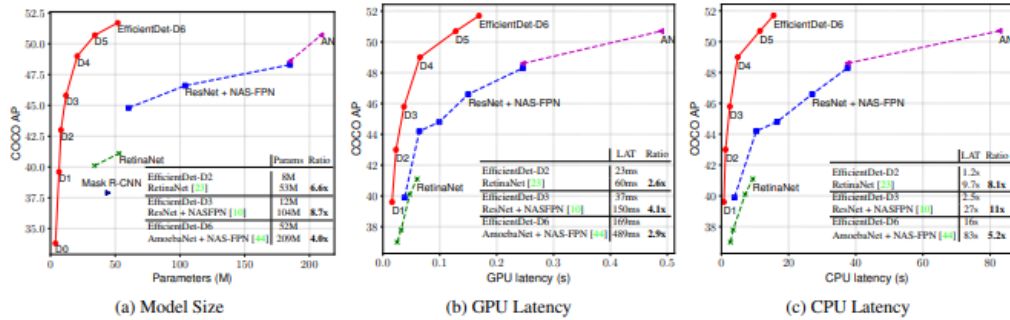


Figure 4.2: Different performance metrics for different network architectures. The latency is measure with a batch size one on the same machine.

Although the ResNet50 model architecture has been used to train traffic-sign detectors[5], as research goal 1.2 states, the goal is to try and implement the current best state-of-the-art network to solve the problem. Therefore, the EfficientDet network architecture is chosen for this paper.

4.1.3 Feature pyramid networks

Before dwelling further into the EfficientDet network architecture, the feature pyramid network needs to be explained[43].

A feature pyramid network (FPN) is a feature extractor designed as a pyramid to produce multiple feature map layers. The FPN uses a bottom-up and top-down approach. The bottom-up pathway is the usual convolutional network for feature extraction applied for different resolutions. The convoluted outputs are not only propagated through each layer, but is also passed to the top-down pathway for the same resolution. The feature map level in a stage of the pyramid uses the notation of P_n , where n is the level in the pathway.

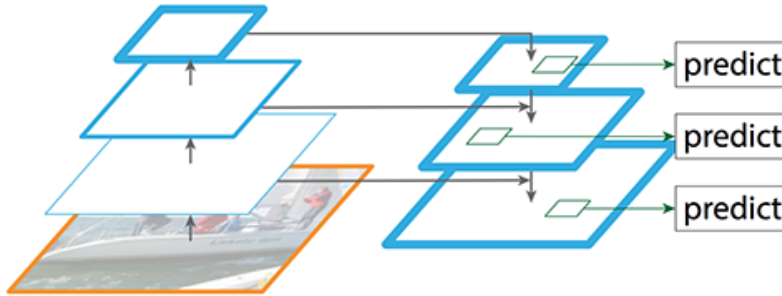


Figure 4.3: The general bottom-up, top-down FPN

4.1.4 EfficientDet

The basic principles of the EfficientDet network was briefly explained above. The EfficientDet network proposes the new type of FPN is the weighted bidirectional feature pyramid network (BiFPN). BiFPN use a more efficient way to aggregate features in one level of the pyramid and then output the list of new features. The BiFPN uses cross-scale connections between the nodes having more than one input edge. In addition to the cross-scale connection, an edge from the original input resolution is added to the output node of the same resolution. Unlike the traditional FPN's that only uses one top-down, bottom-up pathway, the BiFPN treat each top-down, bottom-up path as one feature layer and repeat the same layer multiple times. This helps enable more high-level feature fusions. Figure 4.4 visualizes the BiFPN layer.

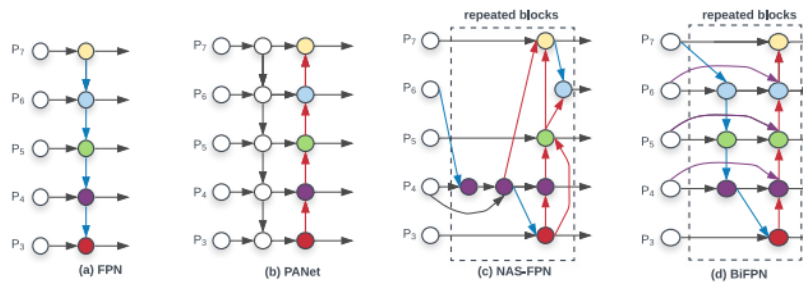


Figure 4.4: The evolution of the FPN from *a* towards the BiFPN in *d*

Unlike the other FPN, the BiFPN treats each input at different res-

olutions with different weights, as the observations shows that the input of different resolutions contribute to the output feature unequally. To address these weights, a *fast normalized fusion* is proposed:

$$O = \sum_i \frac{w_i}{\epsilon \sum_j w_j} I_i$$

where w_i is the learnable weight for image i and w_j are the all the j weights. ReLu is applied to ensure $w_i \leq 0$.

The full network architecture is shown in figure 4.5. The backbone of a model means using the feature extraction network of a selected model as an encoder of the input image, i.e. the backbone extracts features to a certain feature representation. The EfficientDet network is then basically the last steps of the entire model, efficiently extraction more high-level features. The BiFPN network serve as the feature network of the model, which takes 3-7 features P_3, P_4, \dots, P_7 and repeatedly apply top-down, bottom-up bidirectional feature fusion. By the way, the weights are shared across all levels of features. The feature fusions are fed as an input to the box and class network for prediction.

Furthermore, the EfficientDet network addresses the issue of resource constraints by proposing compound scaling. Traditionally, when wanting to scale up the network to try and increase performance, only the size of the backbone is scaled. EfficientDet proposes a family of scaling factors that jointly scale up the backbone in the form of width, depth and the image resolution. The compound scaling coefficient is ϕ . Note, that due to the computational resources being a limitation for this paper, only $\phi = 0$ will be used, but know that the number of BiFPN, box and class layers used is scales with ϕ . The BiFPN scales exponentially.

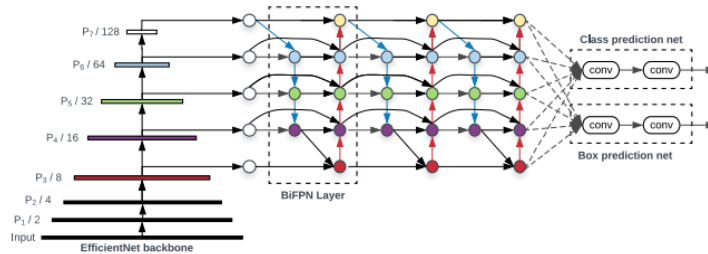


Figure 4.5: The EfficientDet architecture. It uses EfficientNet[21] as a backbone. The entire model is shown in A.1

4.2 Training the model

As mentioned $\phi = 0$ is used for the training due to the limitation of computational resources. This means that the network will be trained on the smallest possible version of EfficientDet. Using $\phi = 0$ set the following training parameters; Image resolution = 512, BiFPN input channels = 64 with a depth = 3 BiFPN layers and the depth of the box/class networks with a depth = 3 for each network. The pretrained weight used for the network is trained using the COCO dataset, as explained in section 3.1, contains over 2.5 million training instances. Although the prediction output are completely different classes, the network extracts the important features from the backbone and further train the network towards the new dataset with corresponding classes. A quick insight of the hyperparameters used for the pretrained weights; SGD optimizer with the normal momentum of 0.9, the learning rate linearly scales from 0 to 0.16 for the first epoch and is annealed using the cosine cycling decay rule, the activation function used is swish, the focal loss was used as a loss function for the classification, the model was trained for 300 epochs with a batch size of 128 on 32 TPU and evaluated with using the Normalized Mean Square (NMS) loss function.

Before explaining the experiment further, the impact of the computational resources needed will be addressed.

Computational resources As mentioned computational setup for the pretrained weights, the proposed model is trained on 32 TPUs. During this experiment, only 1 GPU is available, with only 16GB of potential CPU RAM. Although using $\phi = 0$ might seem like it makes the model architecture small, it is not. Processing the input data when training a network for object detection is usually done using data generators. The purpose of the generator is to distribute the computing power among both the CPU and the GPU. The learning process and the actual training of the network is done using GPU, but the data management and network memory is handled in the CPU. As the proposed networks used for training in this experiment, that is, the GTSRB and MTSD datasets, are way to big and contains way to many classes in order to run a full network. In general, the computational overhead stops at a batch size of 4 using the MTSD dataset, meaning only 4 images will be processed for each step. The step is a hyperparameter that decides how many input groups of the size of the batch size should be processed each epoch. If you want to train on the full dataset each epoch, the

step size is the number of training images divided by the batch size, but as reducing only the step size while still using the full dataset kind of defeats the purpose of learning, as each epoch would consist of a randomly selected dataset each epoch. Therefore, the dataset is reduced drastically.

Reducing the dimensionality As the the image sizes of the GTSRB dataset is only 32x32 images and only contains 43 classes, a batch size of 16 was possible, which made it possible to train using the whole dataset. More about the results of the training of GTSRB in chapter 5. For the MTSD dataset only a fraction of the dataset was used for training. See the next section.

Proposed dataset There are two dimensionality reductions to prepare the data for training the network. In section 3.1, the MTSD contains over 300 classes, where some of these classes were taxonomy versions of each other due to the difference of signs across the world, which makes the dataset less relevant for this experiment. The first dimensionality reduction is to reduce the number of predicted classes. Reading the documentation from the Norwegian State Highways Authority, especially three types of signs are associated with intersections, see figure 4.6. Based on this knowledge, the number of classes is reduced from over 300 to 5, using only the g1 taxonomy. These five classes are the following; yield, turn-left, turn-right, no-left-turn, no-right-turn. Unfortunately, the training network proposed in the results will therefore not recognize any other signs than these five. The *no-u-turn* class is not included due to the low number of instances compared to the other.



Figure 4.6: Norwegian State Highways Authority associated with intersections.

The second dimensionality reduction is the number of training samples used. Knowing that the pretrained weights are trained for 300 epochs, using all occurrences of the selected classes is still too many. A sample size of around 800 images containing these classes are used. More about this is the

result section. As the model learns to recognize these signs on such a small, an occurrence of overfitting might be the case. The validation dataset used was reduced in the same way, resulting in about 50 images.

Training parameters Now that the dataset has been reduced to a smaller sample, the actual training of the network can begin. The general procedure of training the network goes as follows:

(1) The network is trained for a maximum of 50 epochs or until the network converges on a batch size of 4 for the whole dataset freezing all the weights of the backbone up until the feature extraction layers used in EfficientDet, that is the BiFPN layer P_3, P_4, \dots, P_7 . Freezing the weights for a layer means that these weights won't be adjusted. The optimizer used is the Adam optimizer, as it proved to be much faster and yielding better results than the SGD optimizer. Same as the backbone, the smooth focal loss is used for classification loss, and the NMS is used for regression. The learning rate for this step is 0.001, but reduces when the mAP evaluation metric stops increasing.

(2) The network uses the trained weights after step (1). Now, the backbone is unfrozen, meaning the whole network will be trained on the sample dataset. The hyperparameters used in this section are the same except the batch size being changed from 4 to 1, and the learning rate reduced to 0.0001. The model is run for 50 epochs, or until the model converges.

(3) The last step is to train the model yet again on the same dataset, using an even lower learning rate, until the network converges.

The result of training the network is shown in the result chapter.

Traffic sign detection For the case of predicting the traffic signs, the EfficientDet trained on COCO is already trained to detect traffic signs, an inference using their pretrained weights are used.

4.3 Estimating depth

Using the theory explained in section 2.4.4, This section will cover the implementation and architecture used to solve this task.

The actual implementation of the theory derived by Godard et al.[6] is retrieved from an existing github repository implemented by nianticlabs[26], and for this paper, the result of the trained models are used to predict depths. Due to the lack of additional training data for this specific task, no further training is done.

The realization of the theory is the Monodepth2[6] model. The network is based on the U-net model architecture[28] for both the depth and pose networks. The U-net architecture uses the basic principle of processing multi-channel feature maps for many convolutional layers of different dimensionality and propagate each feature map of a certain level through the entire model (see figure 4.7). The U-shape is created by an almost balanced down and upsampling, but often loses some spatial size during the upsampling. The downsampling can be referred to as the encoding of the data, while the upsampling can be referred to as the decoding of the downsampled features maps.

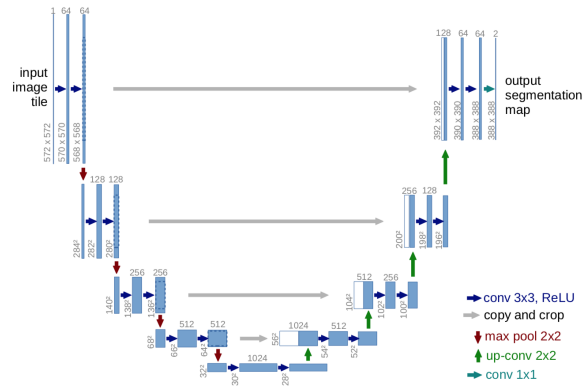


Figure 4.7: The U-net model architecture. Each box corresponds to a multi-channel feature map. Each multi-channel feature map is propagated through the entire network.

The Monodepth2 model use a ResNet18[18][4.1.2] model as an encoder, i.e. the first half of the U-net. The number behind ResNet stands for the amount of layers in the network. Using only 18 layers speeds up the process compared to using a bigger network. There is a slight difference between the depth estimation network and the pose network. This difference is the number of channels used for the input layer. As the pose is estimated though a pair of images, the number on channels used in the input layer is two times that of the depth network. Before training the ResNet18, the pretrained weights of ImageNet is loaded.

For the right-hand side of the network, the decoding of the feature map retrieved through the ResNet18 encoding. Figure 4.9 shows the architecture of the upsampling process. k is the kernel size, s is the stride, $chns$ is the

number of output channels for each layer, res is the downsampling factor for each layer relative to the input image. The input corresponds to the input of each layer where \uparrow is a 2x upsampling nearest-neighbor convolutions. The output of the activation function is sigmoid, while elsewhere, the activation function for the convolutions is ELU, is a version of the ReLU activation function, but maps negative values along -1, instead of flat 0. in The final sigmoid probabilities to depth by $D = \frac{1}{\alpha\sigma+b}$, where a and b are chosen to constrain the depth between 0.1 and 100 units.

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64, \text{stride } 2$
conv2_x	$56 \times 56 \times 64$	$3 \times 3 \text{ max pool, stride } 2$ $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	$7 \times 7 \text{ average pool}$
fully connected	1000	$512 \times 1000 \text{ fully connections}$
softmax	1000	

Figure 4.8: The standard ResNet18 network architecture.

The model is trained on the Kitti Eigen split, which is an already processed depth dataset[8]. The format of the input data is fed into the network in the form of monocular sequences (i.e. monocular and monocular+stereo) After static frames and removed, the result is a training dataset of 39 810 monocular triplets with 4 424 validation examples. The IOPs during training is the same for all the images, that is the focal length and baseline between the images are constant. The focal length is set to be the average all focal lengths in the Kitti Dataset. The average focal length used in the Kitti Dataset is set to 716.44 pixels.

The model is implemented in pytorch and trained for 20 epochs using the Adam optimizer, with a batch size of 12 and an output resolution of 640x192. The learning rate is set to 10^{-4} for the first 15 epochs and is reduced to 10^{-5}

Depth Decoder						
layer	k	s	chns	res	input	activation
upconv5	3	1	256	32	econv5	ELU [7]
iconv5	3	1	256	16	\uparrow upconv5, econv4	ELU
upconv4	3	1	128	16	iconv5	ELU
iconv4	3	1	128	8	\uparrow upconv4, econv3	ELU
disp4	3	1	1	1	iconv4	Sigmoid
upconv3	3	1	64	8	iconv4	ELU
iconv3	3	1	64	4	\uparrow upconv3, econv2	ELU
disp3	3	1	1	1	iconv3	Sigmoid
upconv2	3	1	32	4	iconv3	ELU
iconv2	3	1	32	2	\uparrow upconv2, econv1	ELU
disp2	3	1	1	1	iconv2	Sigmoid
upconv1	3	1	16	2	iconv2	ELU
iconv1	3	1	16	1	\uparrow upconv1	ELU
disp1	3	1	1	1	iconv1	Sigmoid

Pose Decoder						
layer	k	s	chns	res	input	activation
pconv0	1	1	256	32	econv5	ReLU
pconv1	3	1	256	32	pconv0	ReLU
pconv2	3	1	256	32	pconv1	ReLU
pconv3	1	1	6	32	pconv3	-

Figure 4.9: The upsampling neighbor convolutions between each layer.

for the last 5 epochs. One training process was approximately 15 hours using the monocular+stereo models. The result of the trained network has shown to prove better than other models with the goal of a prediction depth in monocular images[6].

4.3.1 Estimating position of the objects

As the goal of this paper is to predict object, estimate the depth, and place the object in a map layer, the variation of IOPs in different cameras make is so that the trained model of Monodepth2 can't directly be applied to estimate the depth of the images retrieved from the Mapillary database. The output of the monocular depth estimation returns scaled depths in meters, but since these depths are scaled by the image width, the pixel-wise depth estimation can be obtained by multiplying the image width with the inverse of the depth:

$$d_p = \frac{imagewidth}{D}$$

As the depth estimation can be converted to pixel space, the bounding boxes of the predicted objects can be resized to the same size of the depth estimation output size, that is to fit a 640x192 image. Now, in order to actually estimate the position of the detected objects, some assumptions about size has to be made.

The Norwegian State Highways Authority has documented the size of common traffic objects. The size of a traffic light (TL) is determined by how many LED-lamps there are[38]. Each lamp should have the cubic size of 200x200. Unfortunately, the number of lamps per traffic light is not detected. Although, most intersections in Norway uses three lamps, i.e. a red, yellow and green lamp. Therefore, the expected size of a traffic light is 600x200mm + some padding. The padding is set to 30mm around each lamp.

$$Size_{TL} = 690x230mm$$

The Norwegian State Highways Authority defines three sizes of traffic signs (TS). LS, MS and SS, which in English is small size, medium size and big size[39]. The size of the traffic sign always denotes the width of the virtual square bounding box around the sign. The sizes of LS, MS and SS is 600mm, 800mm and 1000mm respectively. For intersections specifically, LS is the imposed traffic sign size.

$$Size_{TS} = 600 \times 600 \text{mm}$$

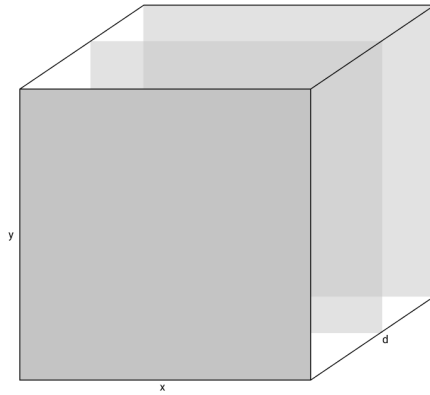


Figure 4.10: A simple illustration visualizing the depth cube in pixel space

The proposed method to derive the geospatial location of each object uses the assumptions about known object sizes to derive meters per pixel ratio of every predicted object, but strongly generalizes each case towards a fixed size, meaning there will be some error through the distortion of the image. Each depth can be visualized like a cube of depth layers shown in figure 4.10, where each layer represents a depth value along axis d . Each object will be represented in a single depth layer.

Figure 4.11 is figuratively speaking an eye-to-thumb illustration of how the position in pixel space is transformed to real distances and uses the basic principles of photography to derive the real-world distance. The dimensions of the conversion between the image pixel space as the real world is swapped in order to make the visualization easier, i.e. of course, the real world image should be much larger than the pixel image. In general, capital letters notates the real world values, while the lower case letter notates the pixel values.

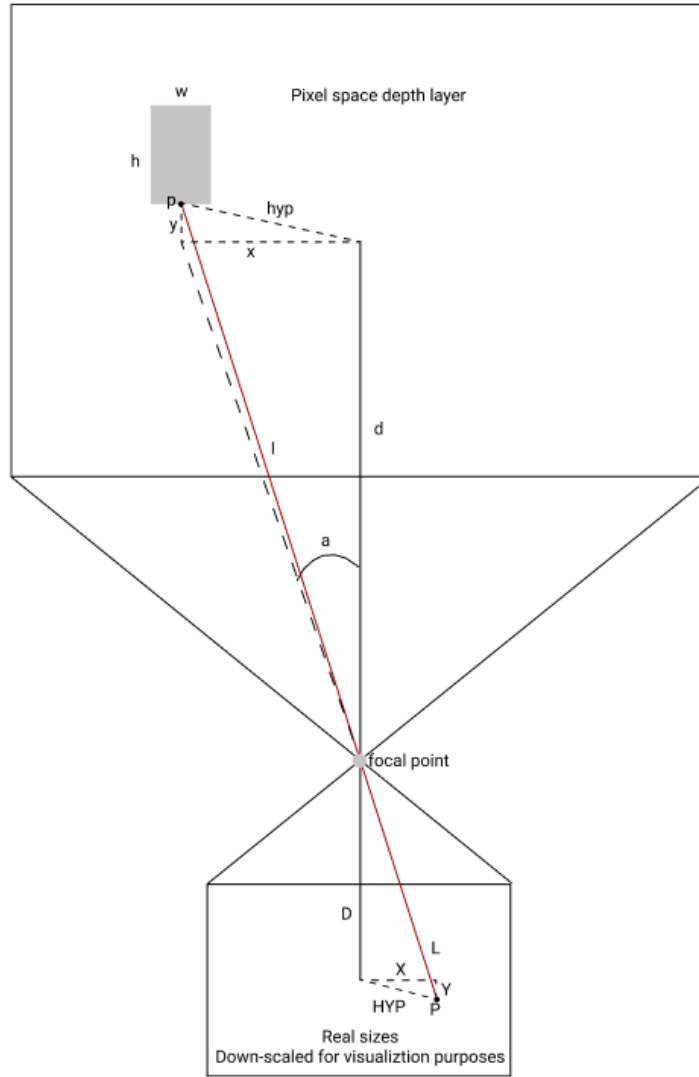


Figure 4.11: The proposed model for converting the pixel values to real world distances.

From the prediction, the pixel position of the bounding box is known. By dividing the known size for each object(o) side with the number of pixels it is represented by, a meter per pixel ratio (mpp) is calculated in both dimensions, x and y. The pixel width and height of the bounding box is

notated with w and h

$$mpp_x = \frac{o_x}{w}$$

,

$$mpp_y = \frac{o_y}{h}$$

Usually the focal length is known, making this process of converting values from pixels to meter more robust, but in the case of only having the length from the focal point to the object, the distance d from the focal point to the center on the depth layer is measured mathematically. From the depth estimation model, the value l is known. x and y are the value from the center of the image to an anchor estimated to be the midpoint along o_x and the bottom of the bounding box. The following mathematical formulas are calculated in order, and $\sqrt{x^2 + y^2}$,

$$d = \sqrt{l^2 - hyp^2} = \sqrt{l^2 - (x^2 + y^2)}$$

, Using the rule of shape similarity D is measured.

$$\frac{D}{d} = \frac{X}{x} \Rightarrow D = \frac{X}{x}d$$

, where $X = mpp_x x$ and $Y = mpp_y y$. The real world distance is measured.

$$L = \sqrt{D^2 + HYP^2} \Rightarrow L = \sqrt{D^2 + X^2 + Y^2}$$

To calculate the relative angle a of the new position, the angle between the focal distance d and x is calculated.

$$a = \arctan\left(\frac{x}{d}\right)$$

The relative angle a can be added the the camera angle ca to get the correct direction of the point each object. The camera angle is the angle from true north, called the azimuth. The relative angle is converted to an angle from the azimuth. The new angle is called α . Both the coordinates and the location of each image is requested using the Mapillary API3.3. The coordinates are represented in latitude and longitude in the geodetic datum WGS84[37]. As the default map projection used in State Highways Authoritys API is UTM33 with WGS84, the coordinates in the image is transformed to the same projection using Pyproj's python library[16].

The final coordinates are measured is simply measured by calculating the displacement from the image position for each axis in the projection, using the new calculated angle.

$$Coord_{object} = [pos_{i_x} + Lcos(\alpha), pos_{i_x} + Lsin(\alpha)]$$

,
 In the cases of traffic lights, the Norwegian State Highways Authority defines a fixed height of the pole where the traffic light is attached. This height is set to 1.5 meter, which can be used to project the estimated position down to the ellipsoid. Unfortunately, there is no fixed height of traffic signs, as it is measured using the surrounding topography, and the estimated position will have a additional error through height.

4.4 Defining the intersection

As explained earlier, the Mapillary images comes with an initial position. Depending on whether an already existing list of intersections, from now on called list of features, is present, the image iterates through the list of features and checks if it's position is inside the feature polygon. In the case of the image not being bounded by an existing intersection, or no existing list of features exists, the script will request all nearby intersections and pick the one that is closest and creates an entirely new intersection, and processes the image through the prediction networks at the same time. The nearby intersections is an iterative process where the algorithm iteratively increases the search range. As one of the image properties is the camera angle, the algorithm decides the closest in the direction of a 45 degree field of view. This is done by measuring the relative angle from the position of the image with each intersection inside the search area, and in the same matter at the relative angle was explained in the above section. If the relative angle to the camera angle is equal or less than 45 degrees, the intersection is kept as a potential closest intersection. Then, the closest intersection is chosen based on the euclidean distance. The field of view in this process is just a fixed value in order to remove non-potential intersection.

The nearby intersections is retrieved using the Norwegian State Highways Authority API querying all intersection inside a bounding box, as well as the speed limits inside the area. If there are multiple speed limit in an area, the maximum value is returned. The size of the bounding box is chosen by the speed limit times the preperation time. The preperation time means the amount of time needed to plan ahead in an intersection. For this paper, 7

seconds is chosen. The are in it self is then generated to as a square bounding box around the center point being the position of the closest intersection.

4.5 Proposed map layer

As a last step, after all chosen input images are processed, the list of features is proposed as a map layer in the form of a GeoJSON[13] file. The GeoJSON is a json standard for storing geometric data. The GeoJSON Working Group is responsible to sustain or update the standard. The latest form of GeoJSON is the verison RFC7946 and was published in 2016, updating the old standard established in 2008.

An intersection contains multiple geometric properties, that is, the geometry for the bounding box of the intersection, but also the geometry for each detected object. The following GeoJSON format is proposed:

```
{
  type: FeatureCollection ,
  features : [
    {
      type: Feature ,
      geometry : {
        type: GeometryCollection ,
        geometries : [
          # The first geometry of the list must always
          # be the bounding box of the intersection
          {
            type: Polygon
            coordinates : [ul , lr ]
          },
          # the rest of the object geometries
          # is the geometry of the predicted objects
          {
            type: Point ,
            coordinates : [x1 , y1 ]
          },
          {
            type: Point ,
            coordinates : [x2 , y2 ]
          },
        ]
      }
    }
  ]
}
```

```

    ...]
  },
  properties:{
    id: id,
    objects: [
      #It is important that the indexing of these
      object corresponds to the indexing+1 in the
      geometries list
      {
        key:image_key1 ,
        class:predicted class ,
        sizes:object_assumptions ,
        object:object_type
      },
      {
        key:image_key2 ,
        class:predicted class ,
        sizes:object_assumptions ,
        object:object_type
      },...
    ]
  },...
]
}

```

It is important to note that this proposed structure upholds all the standards proposed in the current GeoJSON standard (RFC7946).

Chapter 5

Results

This chapter presents the results of the methods used in terms of performance metrics with different variations of the methods as well as presenting the final results of the framework. The discussion of the results will be discussed in the next chapter.

5.1 EfficientDet

GTSRB dataset As mentioned section 4.2, the model was trained on the full dataset for 43 classes. The training used the pretrained weights of the EfficientDetD0 network over 50 epochs with a step size of 10 000 took 22 hours and 1 minute. The following charts shows the different losses with respect to the epoch. The loss for the training data is marked in blue, and the validation dataset is marked in red. A smoothing factor is applied to the loss history to make outliers less visible and helps interpret the graphs.

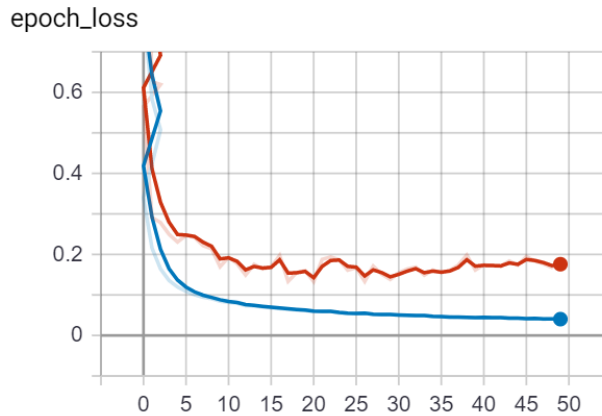


Figure 5.1: GTSRB total loss over 50 epochs

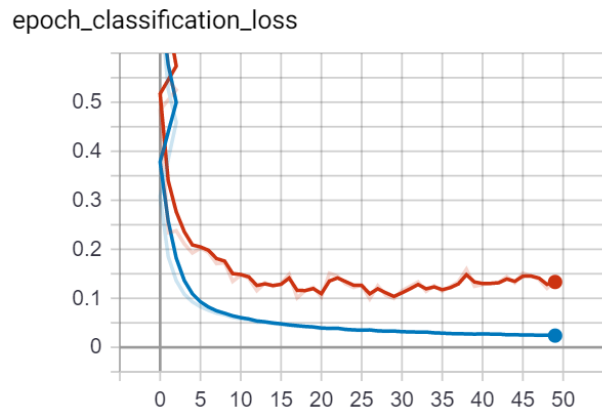


Figure 5.2: GTSRB classification loss over 50 epochs

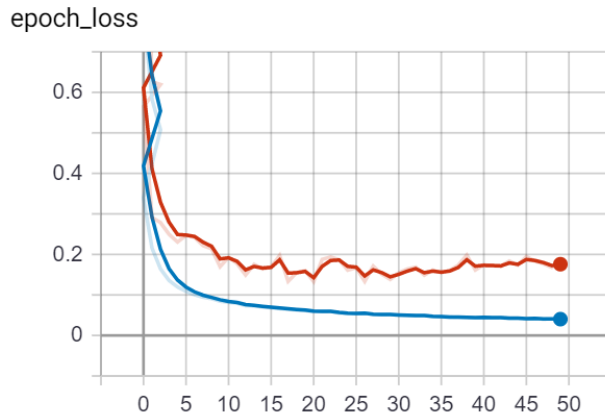


Figure 5.3: GTSRB regression loss over 50 epochs

The above figures shows the training losses during the time of training. The first of the figures shows the total loss of the training session. Already after 10 epochs, the both the training and the validation loss starts converging towards the minimum loss. In general, this a good time to stop the training process, change the hyperparameters and training the model again. Below are some test sample result after an inference.

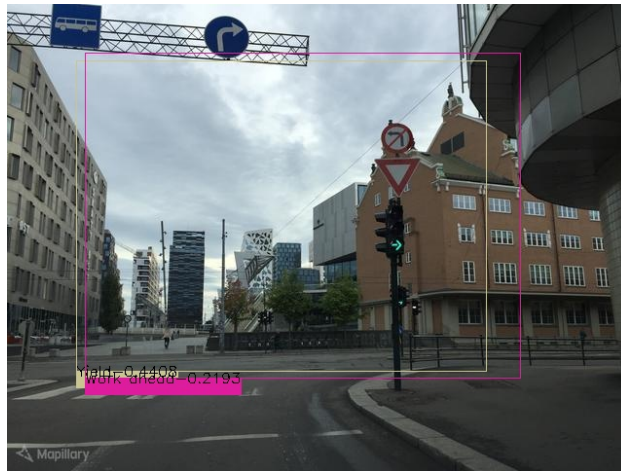


Figure 5.4: GTSRB inference to a Mapillary test image.

The training data consists of many, small images with one single bounding box per image, almost always covering the entire image. Running one

inference on one of the Mapillary test images, the result shows that the model can neither predict the traffic signs, as well as the bounding box. Running an inference on one of it's own test examples, the model performs next to perfect in both terms of fitting the bounding box, as well as predicting the class.



Figure 5.5: GTSRB inference on one of it's own test data

The GTSRB was only trained one time, not following the three-step training process proposed in the last chapter. This is due to the inference on the Mapillary test image clearly showing that the dataset is not suited for object detection in images for more than just the traffic sign. As this image contains many desired object, it will be used in each following section for context. The overall evaluation metric of the class and box prediction resulted in a mAP after 50 epochs of 0.962.

MTSD dataset The three step process explained in the methodology is used during the training of the network. Each session is trained until the training loss converges. Normally, it is the validation loss that decides how long the model should train, but due to the small size of the validation, and the dataset being a random sample of its original size, the validation is in this current situation not fit for that purpose.

Session	mAP	Time
1	0.0363	1h 5min
2	0.1497	2h 18min
3	0.2031	1h 2min

Table 5.1: The mAP and time spent training for each session

The losses for the initial session uses the frozen backbone, and uses the BiFPN features layer network to predict the outputs. As the figures for the first session shows, the training loss converges after 20 epochs. The validation loss converges slower than the training loss, which is expected behavior. The initial hyperparameters for the first session is explained in section 4.2, but the key of the first session is the learn the high level features of the dataset, using the existing features extracted through the backbone using a larger batch size. Table 5.1 shows the mean average precision for each of the sessions run. The result of the first session, run for 1 hour and 5 minutes, is very low of only 0.0363, meaning the network almost never achieves to predict the correct class/box. Given the size of the dataset, these are expected results.

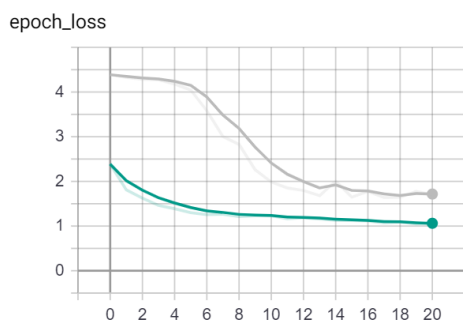


Figure 5.6: The total loss of the training over 20 epoch during stage 1

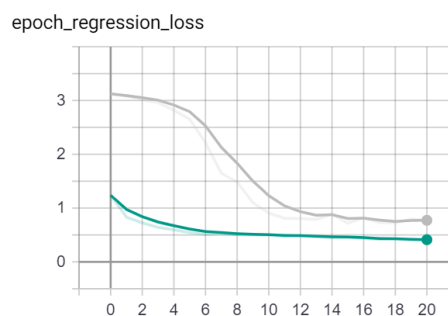


Figure 5.7: The regression loss of the training over 20 epoch during stage 1

The next session is then run with updated hyper parameters, that is, reducing the batch size and unfreezes the backbone. As the model now

tries to learn the whole network the features of the small image size, the predictions on the validation dataset as one can see from the graph below, gets really unpredictable. Although the learning rate is reduced 10, learning rate=0.0001 times from the previous session, the loss still fluctuates a lot more than the that of the previous session. After 62 epochs, the total loss converges and the training stops. Session 2 war run for 2 hours and 18 minutes, and resulted in a mAP of 0.1497, which is a significant increase compared to the previous session.

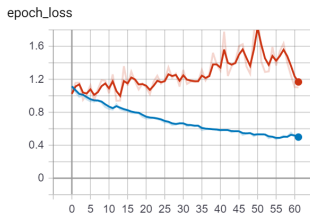


Figure 5.8: The total loss of the training over 62 epoch during stage 2

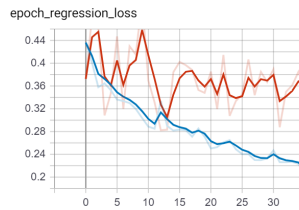


Figure 5.9: The total regression loss of the training over 62 epoch during stage 2

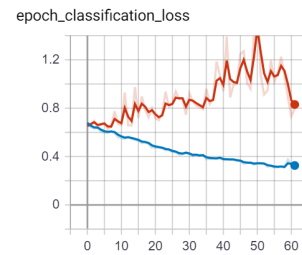


Figure 5.10: The total classification loss of the training over 62 epoch during stage 2

The only adjustments made for the last session is the learning rate being decreased even further, learning rate=0.00001. Instead of statically reducing the learning rate for this step, the cosine cycling annealing steps were tested using the same learning rate as the previous session, but is only caused the network to fluctuate even more, which is why the learning rate is set statically. The EfficientDet architecture still uses some learning rate decay throughout the epochs, but this is not cycling, meaning the expected behavior of this session is to converge quickly, as it is only used as a fine-tuning of the previous session. Session three resulted in a further increase in the mAP to 0.2031. The session lasted for 1 hour and 2 minutes.

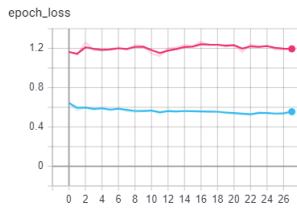


Figure 5.11: The total loss of the training over 27 epoch during stage 3

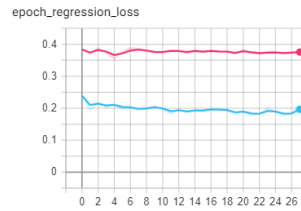


Figure 5.12: The total regression loss of the training over 27 epoch during stage 3

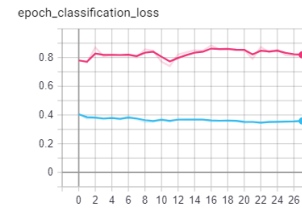


Figure 5.13: The total classification loss of the training over 27 epoch during stage 3

Below is the output of an inference on the Mapillary images. As one can see, there are for some cases redundant and wrong predictions, meaning that there are definitely room for further experiments with the architecture.



Figure 5.14: Good class and box predictions.



Figure 5.15: Redundant class and box predictions.

Traffic light For prediction the traffic lights, the pretrained weights of $\phi = 0$ is used, producing the following results:

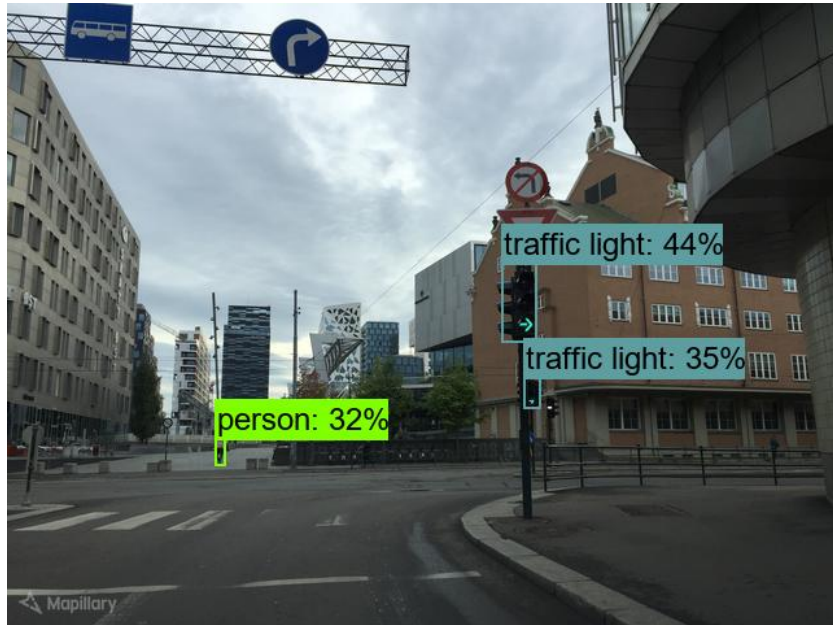


Figure 5.16: The result of an inference using EfficientDet-D0 pretrained weights on the COCO dataset

In addition to detecting the traffic lights, other classes of the COCO dataset can be detected, e.g. a person in the test image above. The predicted bounding boxes of the other detections are not used when estimating the depth. The score threshold for returning a detected object is set 0.2, as the expected mAP for the pretrained weights is 0.332.

5.2 Localization of the objects

This section used the proposed method of size assumption-based pixel-to-meter transformations to derive the position of the predicted objects. The monocular depth estimation is visualized by the inverse depth map matrix is form om a disparity heat map, where bright areas are considered close, and dark areas are considered far away. In context of comparison, the depth map and the predicted objects are pair-wise represented.

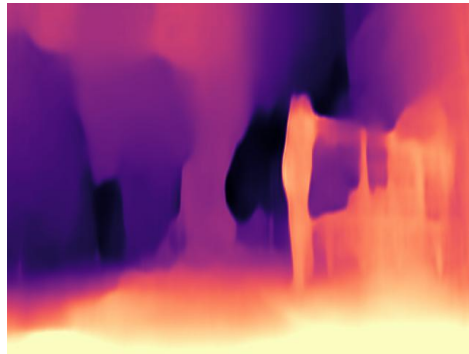


Figure 5.17: The depth map result from the Monocular depth estimation.



Figure 5.18: The detected traffic lights for the image.



Figure 5.19: The detected traffic signs

The above example is the output of all three prediction networks used in this proposed method. The method proposed in section 4.3 of converting the depth of in the pixel space to real-world distances. The following results are some extremes of a good and bad depth estimation.

In order to estimate if the projected distance was good or not, a ground truth distance quickly by retrieving the coordinates of the actual object through the State Highways Authority API and measure the euclidean distance between the points and compare it to the measured distance by the proposed techniques.

The first example a case of a good estimated depth and projected position. The euclidean distance between the position of the Mapillary image and left-hand traffic sign in the image is measured to 6.107 meters. The calculated

distance to the traffic light was measured to be 6.147 meters, yielding an error of only 4 cm, which is comparable to RTK measurements estimating an error of $2\text{cm} \pm 1\text{ppm}$ [24]. The State Highways Authority is not guaranteed to have geospatial information of the detected object, and for this image, this traffic light was the only available position. Looking at the disparity map, the traffic light at the left-hand size is very bright, meaning that it is considered closer to the camera.



Figure 5.20: The depth disparity map where the results were good.

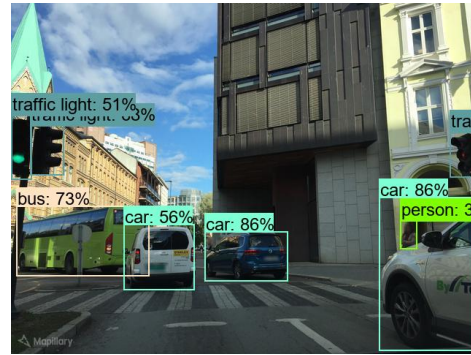


Figure 5.21: The predicted traffic lights of a good example.

The next example is an extremely bad depth estimation. The distance between the Mapillary image and the predicted traffic sign is estimated to be 47.638 meters. The depth estimation model calculates the real world distance to be 369.371 meters, i.e. the gross error of 321.723 meters. Looking at the depth disparity of the image. The area of the predicted is totally dark, i.e. the monocular depth estimation model consider it as the sky, or too far away using the maximum depth thresholds explained in section 4.3, that is, a maximum distance of 100 real-world meters.

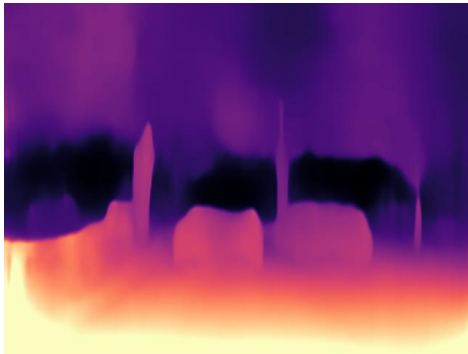


Figure 5.22: The depth disparity map of an extremely bad example.



Figure 5.23: The traffic signs detected in the image.

Due to the lack of ground truth distances, an iterative approach to comparing the predicted depth estimations towards the real distance is impossible, which is why no statistical analysis of the calculations are presented, but the general consensus is that closer i.e. brighter areas results in a more accurate depth calculations.

Part III

Discussion and conclusion

Chapter 6

Discussion

This chapter discusses what went wrong, the magnitude of errors and changes that could have been made.

6.1 The model architecture

The choice of the model architecture in it self showed through training the network using the GTSRB dataset that it can efficiently train a network based on a large dataset and produce very good results for a specific task. Due to the computational resources limitations, this paper could not delve into experimenting with other network sizes. As the benchmark on the COCO dataset shows an increase in performance as the network gets bigger, it would most likely be the case for this paper as well.

The fact that the model converged in about 10 epochs while trained on the whole dataset shows the potential of the model architecture being efficient, as well as yielding a high result on the GTSRB dataset. In order to compare the result of the model trained on MTSD, figure 6.1 shows some resulted mAP performances using the ResNet model architecture. Comparing the results shown in the last chapter, the performance of the ResNet architecture outperforms the papers resulting network, but, when considering the number of training instances used during their network, the results does not encourage the power of the EfficientDet model. The performance metrics used for the ResNet model was 30 epochs, using a batch size of 16 along 4 GPUS, meaning a batch size of 4 is run at each GPU. This makes the processing power of the training much faster, as the batch size can make the network learn quicker each during each iteration. In order to really compare these two model architectures trained of the MTSD dataset, a lot more computational power is needed.

A proposed method for increasing the computational power is by implementing so that it runs on cloud-based services. Such cloud-based services can distribute the work-force over many nodes, meaning no computation has to be run locally.

	mAP	mAP _s	mAP _m	mAP _l
MTSD				
FPN50 + classifier	81.1	69.4	85.0	87.2
FPN101 + classifier	83.4	76.4	85.8	87.3
TT100K				
[36] multi-scale	81.6	68.3	86.5	85.7
FPN50 + classifier	89.9 (+8.3)	83.9	93.0	84.3
+ <i>det pre-trained</i>	93.4 (+11.8)	88.2	94.8	93.6
+ <i>cls pre-trained</i>	95.7 (+14.1)	91.3	96.9	96.7

Figure 6.1: The result of using ResNet50 on the MTSD dataset. These metrics was proposed by the Mapillary’s own reasearch team[5]

6.2 Prediction results

The fact that the GTSRB dataset lacking the spatial context of the image resulted in it not being fit for detecting objects in the Mapillary test images, for both fitting the bounding box as well as classifying object. The reason for the high mAP or 0.962 is due to the overfitting of the network. The model was trained for 50 epochs on the full dataset for 22 hours, but reached the convergence loss in only 10 epochs at 5 hours, meaning that every epoch after than made the model better and better at detecting only its own dataset. This might have caused the model being so bad at detecting object in any other image. The theory of this could have been tested by applying some early stopping monitoring in order to stop the model automatically once it reaches convergence. In addition of trying to make the network more robust could be to use a cross-validation during training, which for each iteration spilts the dataset into k folds training and validation, which lets the model train on every single instance, as well as validating the model on a different dataset each iteration. The low amount of distortion in the images could also be a potential factor of why the robustness of the network was so low. Although the fitting of the anchor was bad, the detection of the object could in theory still be possible, as the BiFPN structure of the architecture learns the network to recognize features at different resolutions.

The low prediction scores after the network was trained for 3 sessions is defiantly due to the reduction of the dataset. Due to time constraints at the end, there was no time to test this theory, but is an available task for future work. The reason for choosing such a small sample set of the original dataset was firstly to, as explained earlier, reduce the computational resources needed for train the network, the other was to reduce the number of classes towards the relevant classes for this task only. Especially for the mAP performance metric, such a small amount of training data could basically cause the model to guess the class and box prediction. From the results in the previous chapter, the model fortunately learned enough from the feature extraction to make some predictions in the Mapillary images, making the pipeline of the proposed framework complete, although as shown, the redundant boxes causes the depth for each object being calculated several times. As of now, there is no evaluation considered when storing the objects in the intersection.

As for the recognition of the traffic lights, no further training was applied to the EfficientDet pretrained weights. The fact the the overall mAP on the 2.5 million COCO instances is 0.338 speaks for the robustness of the network, but doesn't mean that the prediction of the traffic lights are especially good. To fix this issue, some fine-tuning of the network could be done by either extracting only the traffic lights from the COCO dataset, and use that to further train the network to only detect traffic lights, or a totally new dataset of traffic lights could be used.

6.3 Depth estimation

The task of estimating depths is in general a very difficult task. The approach for this paper was to use a monocular depth estimation convolutional neural networks to solve this problem mostly due to the lack of IOP's for each image, that is, for each camera used for each image in the Mapillary dataset. The training data of the model was based on the KITTI dataset, which uses a highly calibrated cameras with a high resolution. This in it self is very good to train a good model, but when low-resolution images are fed through the network, the result becomes less reliable. Therefore the error in the dataset could cause the depth estimations to be bias towards the dataset used for the pretrained weights.

In addition to the input images already being at a disadvantage, the generalization of the assumptions made about size does not directly account

to the distortion of the images. Each object detected are expected to be of the same size in every occurrence, but this is of course not true in a real life scenario. The proposed real-depth calculations from pixel space scales the fixed size of the object, making it an OK assumption, but it does not take into consideration the rotation angle of the object nor the vertical or horizontal displacement of the object. The biggest error is probably in the generalization of the assumptions made, but an addition to the increase in wrong predictions the trained network can fit the box around the object. Towards the proposed intersection, a mis-prediction will make the ratio of the actual size and the bounding box wrong, further increasing the error of the estimated distance. When explaining the implementation and method of the pixel-to-meter framework, the final distance L to the object is the square root of polynomial of the assumption-based values, meaning that the error in the pixel space will propagate with a power of 2.

If intrinsic orientation parameters had been the same for each camera in the Mapillary database, a scale factor from the disparity map to the real world distances could have been calculated by

$$D = \frac{focallength * baseline}{disparity}$$

which is the formula used when the images given their calibration matrix is measured. Another approach the could be tackled to measure real world depth without the IOP for each camera is to obtain this scale by empirically generalize the depth factor for all the cameras used in the Mapillary dataset. It would be interesting to see how this would perform compared to the approach proposed in this paper.

As mentioned many times, the task of estimating depth is hard due to the inconsistency of images. The proposed network in [26] uses a pretty shallow network to encode the features from the input image. In a similar way as the fine-tuning of the traffic light prediction using EfficientDet could increase the performance of recognizing traffic lights, the tuning of the encoder network could cause the network to estimate even better depth disparities. As an example, it would be interesting to apply the EfficientDet model architecture instead of the ResNet18 model.

Chapter 7

Conclusion

This chapter presents the final conclusion of the paper and suggests some future work.

7.1 Conclusion

The motivation of the paper was to answer three proposed research goals. (1) Analyze the current state-of-the-art network architectures in order to train a network and predict traffic signs through object detection in images. (2) Apply a monocular depth prediction model to the images and establish an algorithm to predict an object's position without knowing the interior orientation parameters. (3) Propose a map layer framework for traffic intersections containing spatial and temporal properties for each intersection.

Through excessive research of the current state-of-the-art object detection networks, EfficientDet was selected. Due to the limitations of computational power, this model architecture was implemented and trained on both the GTSRB and MTSD traffic sign dataset benchmarks and resulted in two completely different results. Both networks were trained using the EfficientDet-D0 pretrained network with tuned hyperparameters. Due to the scale difference and spatial context in the training data, the GTSRB dataset could not be used to predict traffic signs in for the Mapillary images, even though a high mean average precision was achieved. In order to train a network using the MTSD dataset, a dimensionality reduction of the dataset had to be made by reducing the amount of training data, as well as predicted classes. This was used as the traffic sign detector on the Mapillary images and helped get further towards solving the first research goal.

By proposing a method to convert the depth disparity of the image space to real world, assumptions were introduced as a baseline for the algorithm. The depth disparity is derived by using a monocular depth estimation model that encodes the features of the images and decodes it by reconstructing the image through several CNNs approaches. The two methods combined resulted in many edge-case scenarios of good as well as bad results, but the best results can be compared to the accuracy of the RTK gps measurements, and is significantly faster and automated. This proposal was meant to issue research question number two, but due to the inconsistencies of the result, did not prove to solve it.

The last of the research goals combined the other two and represents the framework proposed in the paper. The proposed map layer consists of a multi-geometry feature collection and stored in form of a GeoJSON object where each intersection holds the properties of predicted object as well as the calculated position.

7.2 Future work

Derived from the discussion chapter, there are several things to be considered for future works, as this paper only had the time to take a step towards creating a road intersection map layer. The biggest source of error derived from this paper is the error and uncertainty in the estimated depth framework. More research has to be done in this field. An important feature of the approach proposed in this paper is that the IOPs are not available for the images processed. Due to the fixed size assumptions for the objects a step in the right direction would be to apply some affine transformations to the predicted bounding box and translate the fixed size to the shape of distorted object. An example of a method that could be used for this is edge detection algorithms to measure the relative size of the distorted image.

Another important step for the future is to further train the EfficientDet model on a larger dataset. In order to solve the problem of being locally limited by computational resources, as suggested in the discussion, support towards cloud-based services can be implemented in order to remove the local restraints.

Due to the time restrictions of this paper, a post-processing step of the predicted object position should be added in order to remove bad, or out of bound points.

Bibliography

- [1] Manal El Aidouni. *Evaluating Object Detection Models: Guide to Performance Metrics*. URL: <https://manalelaidouni.github.io/manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html>.
- [2] Alex Krizhevsky et al. *ImageNet Classification with Deep Convolutional Neural Networks*. URL: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.
- [3] John Duchi et al. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. URL: <http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf>.
- [4] *An introduction to artificial neural networks*. URL: <http://lovefordatascience.blogspot.com/2017/10/an-introduction-to-artificial-neural.html>.
- [5] Christian Ertler, Jerneja Mislej, Tobias Ollmann Lorenzo Porzi, Gerhard Neuhold, Yubin Kuang. *The Mapillary Traffic Sign Dataset for Detection and Classification on a Global Scale*. URL: <https://arxiv.org/pdf/1909.04422.pdf>.
- [6] Clement Godard, Oisin Mac Aodha, Michael Firman, Gabriel Brostow. *Digging Into Self-Supervised Monocular Depth Estimation*. URL: <https://arxiv.org/abs/1806.01260>.
- [7] Close-range.com. *Image rectification*. URL: http://www.close-range.com/docs/Image_rectification.pdf.
- [8] David Eigen, Rob Fergus. *Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture*. URL: <https://arxiv.org/pdf/1411.4734.pdf>.

- [9] DeepAI. *Feature Extraction*. URL: <https://deepai.org/machine-learning-glossary-and-terms/feature-extraction>.
- [10] Luke Dormehl. *What is an artificial neural network? Here's everything you need to know*. URL: <https://www.digitaltrends.com/cool-tech/what-is-an-artificial-neural-network/>.
- [11] Fred E. Szabo PhD. *Manhattan Distance*. URL: <https://www.sciencedirect.com/topics/mathematics/manhattan-distance>.
- [12] Andreas Geiger et al. *The KITTI Vision Benchmark Suite*. URL: <http://www.cvlibs.net/datasets/kitti/>.
- [13] GeoJSON. *GeoJSON*. URL: <https://geojson.org/>.
- [14] Knut Hinkelmann. *Neural networks*. URL: http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf.
- [15] ImageNet. *ImageNet*. URL: <http://www.image-net.org/>.
- [16] Jeffrey Whitaker. *PyProj documentation*. URL: <http://pyproj4.github.io/pyproj/stable/>.
- [17] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick. *Mask R-CNN*. URL: <https://arxiv.org/pdf/1703.06870.pdf>.
- [18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition*. URL: <https://arxiv.org/pdf/1512.03385.pdf>.
- [19] *Mapillary.com*. URL: <https://www.mapillary.com/>.
- [20] Andrea Vedaldi Maria Klodt. *Supervising the new with the old: learning SFM from SFM*. URL: http://openaccess.thecvf.com/content_ECCV_2018/papers/Maria_Klodt_Supervising_the_new_ECCV_2018_paper.pdf.
- [21] Mingxing Tan, Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. URL: <https://arxiv.org/pdf/1905.11946.pdf>.
- [22] Mingxing Tan, Ruoming Pang, Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*. URL: <https://arxiv.org/pdf/1911.09070.pdf>.

- [23] missinglink.ai. *7 Types of Neural Network Activation Functions: How to Choose?* URL: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>.
- [24] "Introduction to Network RTK". *Real Time Kinematic System For Precision Farming*. URL: <https://archive.is/20120203041216/http://www.agnav.com/RealTimeKinematicSystem>.
- [25] Institut für Neuroinformatik. *The German Traffic Sign Recognition Benchmark*. URL: <http://benchmark.ini.rub.de/?section=gtsrb&subsection=news>.
- [26] nianticlabs. URL: <https://github.com/nianticlabs/monodepth2>.
- [27] Michael Nielsen. *Neural Networks and Deep Learning*. URL: <http://neuralnetworksanddeeplearning.com/>.
- [28] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. URL: <https://arxiv.org/pdf/1505.04597.pdf>.
- [29] *OpenStreetMap*. URL: <https://www.openstreetmap.org/about>.
- [30] Mohammad Najafi Sarah Taghavi Namin Mathieu Salzmann Lars Petersson. *Sample and Filter: Nonparametric Scene Parsing via Efficient Filtering*. URL: <https://arxiv.org/pdf/1511.04960.pdf>.
- [31] Proj.org. *Universal Transverse Mercator (UTM)*. URL: <https://proj.org/operations/projections/utm.html?highlight=utm>.
- [32] Alexander Mordvintsev Abid K. Revision. *Introduction to SIFT (Scale-Invariant Feature Transform)*. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html.
- [33] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. *Object Detection in 20 Years: A Survey*. URL: <https://arxiv.org/pdf/1311.2524.pdf>.
- [34] RShaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Rich feature hierarchies for accurate object detection and semantic segmentation*. URL: <https://arxiv.org/pdf/1311.2524.pdf>.
- [35] Jurgen Schmidhuber. *Deep Learning in Neural Networks: An Overview*. URL: <https://arxiv.org/pdf/1404.7828.pdf>.

- [36] SNL. *lidar*. URL: <https://snl.no/lidar>.
- [37] Spatial reference. *EPSG:4326*. URL: <https://spatialreference.org/ref/epsg/wgs-84/>.
- [38] Statens vegvesen. *Trafikksignalanlegg*. URL: https://www.vegvesen.no/_attachment/61421/binary/964088?fast_title=H%C3%A5ndbok+N303+Trafikksignalanlegg.pdf.
- [39] Statens vegvesen. *Trafikksignalanlegg*. URL: https://www.vegvesen.no/_attachment/69739/binary/964083?fast_title=H%C3%5C%A5ndbok+N300+Trafikkskilt%5C%2C+del+3+Forbudsskilt%5C%2C+p%5C%5C%A5budsskilt%5C%2C+opplysningsskilt+og+skilt+med+trafikksikkerhetsinformasjon+%5C%2811+MB%5C%29.pdf.
- [40] Remote sensing systems. *SSMI/SSMIS*. URL: <http://www.remss.com/missions/ssmi/>.
- [41] Phuc Truong. *Loss functions: Why, what, where or when?* URL: <https://medium.com/@phuctrt/loss-functions-why-what-where-or-when-189815343d3f>.
- [42] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan C. Lawrence Zitnick, Piotr Dollar. *Microsoft COCO: Common Objects in Context*. URL: <https://arxiv.org/pdf/1405.0312.pdf>.
- [43] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie. *Feature Pyramid Networks for Object Detection*. URL: <https://arxiv.org/abs/1612.03144>.
- [44] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, Piotr Dollar. *Focal Loss for Dense Object Detection*. URL: <https://arxiv.org/pdf/1708.02002.pdf>.
- [45] Stanford univerty. *Convolutional Neural Networks (CNNs / ConvNets)*. URL: <https://cs231n.github.io/convolutional-networks/>.
- [46] Yasutaka Furukawan, Carlos Hernández. *Multi-View Stereo: A Tutorial*. URL: http://carlos-hernandez.org/papers/fnt_mvs_2015.pdf.
- [47] Zhengxia Zou, Zhenwei Shi, Member, IEEE, Yuhong Guo, and Jieping Ye, Senior Member, IEEE. *Object Detection in 20 Years: A Survey*. URL: <http://www.cvlibs.net/datasets/kitti/>.

Appendix A

Appendix

A.1 The full architecture of the EfficientDet network

Model: "efficientdet"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 512, 512, 3)]	0	
stem_conv (Conv2D)	(None, 256, 256, 32)	864	input_1[0][0]
stem_bn (BatchNormalization)	(None, 256, 256, 32)	128	stem_conv[0][0]
stem_activation (Activation)	(None, 256, 256, 32)	0	stem_bn[0][0]
block1a_dwconv (DepthwiseConv2D)	(None, 256, 256, 32)	288	stem_activation[0][0]
block1a_bn (BatchNormalization)	(None, 256, 256, 32)	128	block1a_dwconv[0][0]
block1a_activation (Activation)	(None, 256, 256, 32)	0	block1a_bn[0][0]
block1a_se_squeeze (GlobalAveragePooling2D)	(None, 32)	0	block1a_activation[0][0]
block1a_se_reshape (Reshape)	(None, 1, 1, 32)	0	block1a_se_squeeze[0][0]
block1a_se_reduce (Conv2D)	(None, 1, 1, 8)	264	block1a_se_reshape[0][0]
block1a_se_expand (Conv2D)	(None, 1, 1, 32)	288	block1a_se_reduce[0][0]
block1a_se_excite (Multiply)	(None, 256, 256, 32)	0	block1a_activation[0][0] block1a_se_expand[0][0]
block1a_project_conv (Conv2D)	(None, 256, 256, 16)	512	block1a_se_excite[0][0]
block1a_project_bn (BatchNormalization)	(None, 256, 256, 16)	64	block1a_project_conv[0][0]
block2a_expand_conv (Conv2D)	(None, 256, 256, 96)	1536	block1a_project_bn[0][0]

block2a_expand_bn (BatchNormali	(None, 256, 256, 96)	384	block2a_expand_conv[0][0]
block2a_expand_activation (Acti	(None, 256, 256, 96)	0	block2a_expand_bn[0][0]
block2a_dwconv (DepthwiseConv2D	(None, 128, 128, 96)	864	block2a_expand_activation[0][0]
block2a_bn (BatchNormalization)	(None, 128, 128, 96)	384	block2a_dwconv[0][0]
block2a_activation (Activation)	(None, 128, 128, 96)	0	block2a_bn[0][0]
block2a_se_squeeze (GlobalAvera	(None, 96)	0	block2a_activation[0][0]
block2a_se_reshape (Reshape)	(None, 1, 1, 96)	0	block2a_se_squeeze[0][0]
block2a_se_reduce (Conv2D)	(None, 1, 1, 4)	388	block2a_se_reshape[0][0]
block2a_se_expand (Conv2D)	(None, 1, 1, 96)	480	block2a_se_reduce[0][0]
block2a_se_excite (Multiply)	(None, 128, 128, 96)	0	block2a_activation[0][0] block2a_se_expand[0][0]
block2a_project_conv (Conv2D)	(None, 128, 128, 24)	2304	block2a_se_excite[0][0]
block2a_project_bn (BatchNormal	(None, 128, 128, 24)	96	block2a_project_conv[0][0]
block2b_expand_conv (Conv2D)	(None, 128, 128, 144)	3456	block2a_project_bn[0][0]
block2b_expand_bn (BatchNormali	(None, 128, 128, 144)	576	block2b_expand_conv[0][0]
block2b_expand_activation (Acti	(None, 128, 128, 144)	0	block2b_expand_bn[0][0]
block2b_dwconv (DepthwiseConv2D	(None, 128, 128, 144)	1296	block2b_expand_activation[0][0]
block2b_bn (BatchNormalization)	(None, 128, 128, 144)	576	block2b_dwconv[0][0]
block2b_activation (Activation)	(None, 128, 128, 144)	0	block2b_bn[0][0]
block2b_se_squeeze (GlobalAvera	(None, 144)	0	block2b_activation[0][0]
block2b_se_reshape (Reshape)	(None, 1, 1, 144)	0	block2b_se_squeeze[0][0]
block2b_se_reduce (Conv2D)	(None, 1, 1, 6)	870	block2b_se_reshape[0][0]
block2b_se_expand (Conv2D)	(None, 1, 1, 144)	1008	block2b_se_reduce[0][0]
block2b_se_excite (Multiply)	(None, 128, 128, 144)	0	block2b_activation[0][0] block2b_se_expand[0][0]
block2b_project_conv (Conv2D)	(None, 128, 128, 24)	3456	block2b_se_excite[0][0]
block2b_project_bn (BatchNormal	(None, 128, 128, 24)	96	block2b_project_conv[0][0]
block2b_drop (FixedDropout)	(None, 128, 128, 24)	0	block2b_project_bn[0][0]
block2b_add (Add)	(None, 128, 128, 24)	0	block2b_drop[0][0] block2a_project_bn[0][0]
block3a_expand_conv (Conv2D)	(None, 128, 128, 144)	3456	block2b_add[0][0]
block3a_expand_bn (BatchNormali	(None, 128, 128, 144)	576	block3a_expand_conv[0][0]
block3a_expand_activation (Acti	(None, 128, 128, 144)	0	block3a_expand_bn[0][0]
block3a_dwconv (DepthwiseConv2D	(None, 64, 64, 144)	3600	block3a_expand_activation[0][0]
block3a_bn (BatchNormalization)	(None, 64, 64, 144)	576	block3a_dwconv[0][0]
block3a_activation (Activation)	(None, 64, 64, 144)	0	block3a_bn[0][0]
block3a_se_squeeze (GlobalAvera	(None, 144)	0	block3a_activation[0][0]
block3a_se_reshape (Reshape)	(None, 1, 1, 144)	0	block3a_se_squeeze[0][0]
block3a_se_reduce (Conv2D)	(None, 1, 1, 6)	870	block3a_se_reshape[0][0]
block3a_se_expand (Conv2D)	(None, 1, 1, 144)	1008	block3a_se_reduce[0][0]
block3a_se_excite (Multiply)	(None, 64, 64, 144)	0	block3a_activation[0][0] block3a_se_expand[0][0]
block3a_project_conv (Conv2D)	(None, 64, 64, 40)	5760	block3a_se_excite[0][0]
block3a_project_bn (BatchNormal	(None, 64, 64, 40)	160	block3a_project_conv[0][0]
block3b_expand_conv (Conv2D)	(None, 64, 64, 240)	9600	block3a_project_bn[0][0]
block3b_expand_bn (BatchNormali	(None, 64, 64, 240)	960	block3b_expand_conv[0][0]
block3b_expand_activation (Acti	(None, 64, 64, 240)	0	block3b_expand_bn[0][0]
block3b_dwconv (DepthwiseConv2D	(None, 64, 64, 240)	6000	block3b_expand_activation[0][0]
block3b_bn (BatchNormalization)	(None, 64, 64, 240)	960	block3b_dwconv[0][0]
block3b_activation (Activation)	(None, 64, 64, 240)	0	block3b_bn[0][0]

block3b_se_squeeze (GlobalAvera (None, 240)	0	block3b_activation[0][0]
block3b_se_reshape (Reshape) (None, 1, 1, 240)	0	block3b_se_squeeze[0][0]
block3b_se_reduce (Conv2D) (None, 1, 1, 10)	2410	block3b_se_reshape[0][0]
block3b_se_expand (Conv2D) (None, 1, 1, 240)	2640	block3b_se_reduce[0][0]
block3b_se_excite (Multiply) (None, 64, 64, 240)	0	block3b_activation[0][0] block3b_se_expand[0][0]
block3b_project_conv (Conv2D) (None, 64, 64, 40)	9600	block3b_se_excite[0][0]
block3b_project_bn (BatchNormal (None, 64, 64, 40)	160	block3b_project_conv[0][0]
block3b_drop (FixedDropout) (None, 64, 64, 40)	0	block3b_project_bn[0][0]
block3b_add (Add) (None, 64, 64, 40)	0	block3b_drop[0][0] block3a_project_bn[0][0]
block4a_expand_conv (Conv2D) (None, 64, 64, 240)	9600	block3b_add[0][0]
block4a_expand_bn (BatchNormal (None, 64, 64, 240)	960	block4a_expand_conv[0][0]
block4a_expand_activation (Acti (None, 64, 64, 240)	0	block4a_expand_bn[0][0]
block4a_dwconv (DepthwiseConv2D (None, 32, 32, 240)	2160	block4a_expand_activation[0][0]
block4a_bn (BatchNormalization) (None, 32, 32, 240)	960	block4a_dwconv[0][0]
block4a_activation (Activation) (None, 32, 32, 240)	0	block4a_bn[0][0]
block4a_se_squeeze (GlobalAvera (None, 240)	0	block4a_activation[0][0]
block4a_se_reshape (Reshape) (None, 1, 1, 240)	0	block4a_se_squeeze[0][0]
block4a_se_reduce (Conv2D) (None, 1, 1, 10)	2410	block4a_se_reshape[0][0]
block4a_se_expand (Conv2D) (None, 1, 1, 240)	2640	block4a_se_reduce[0][0]
block4a_se_excite (Multiply) (None, 32, 32, 240)	0	block4a_activation[0][0] block4a_se_expand[0][0]
block4a_project_conv (Conv2D) (None, 32, 32, 80)	19200	block4a_se_excite[0][0]
block4a_project_bn (BatchNormal (None, 32, 32, 80)	320	block4a_project_conv[0][0]
block4b_expand_conv (Conv2D) (None, 32, 32, 480)	38400	block4a_project_bn[0][0]
block4b_expand_bn (BatchNormal (None, 32, 32, 480)	1920	block4b_expand_conv[0][0]
block4b_expand_activation (Acti (None, 32, 32, 480)	0	block4b_expand_bn[0][0]
block4b_dwconv (DepthwiseConv2D (None, 32, 32, 480)	4320	block4b_expand_activation[0][0]
block4b_bn (BatchNormalization) (None, 32, 32, 480)	1920	block4b_dwconv[0][0]
block4b_activation (Activation) (None, 32, 32, 480)	0	block4b_bn[0][0]
block4b_se_squeeze (GlobalAvera (None, 480)	0	block4b_activation[0][0]
block4b_se_reshape (Reshape) (None, 1, 1, 480)	0	block4b_se_squeeze[0][0]
block4b_se_reduce (Conv2D) (None, 1, 1, 20)	9620	block4b_se_reshape[0][0]
block4b_se_expand (Conv2D) (None, 1, 1, 480)	10080	block4b_se_reduce[0][0]
block4b_se_excite (Multiply) (None, 32, 32, 480)	0	block4b_activation[0][0] block4b_se_expand[0][0]
block4b_project_conv (Conv2D) (None, 32, 32, 80)	38400	block4b_se_excite[0][0]
block4b_project_bn (BatchNormal (None, 32, 32, 80)	320	block4b_project_conv[0][0]
block4b_drop (FixedDropout) (None, 32, 32, 80)	0	block4b_project_bn[0][0]
block4b_add (Add) (None, 32, 32, 80)	0	block4b_drop[0][0] block4a_project_bn[0][0]
block4c_expand_conv (Conv2D) (None, 32, 32, 480)	38400	block4b_add[0][0]
block4c_expand_bn (BatchNormal (None, 32, 32, 480)	1920	block4c_expand_conv[0][0]
block4c_expand_activation (Acti (None, 32, 32, 480)	0	block4c_expand_bn[0][0]
block4c_dwconv (DepthwiseConv2D (None, 32, 32, 480)	4320	block4c_expand_activation[0][0]
block4c_bn (BatchNormalization) (None, 32, 32, 480)	1920	block4c_dwconv[0][0]
block4c_activation (Activation) (None, 32, 32, 480)	0	block4c_bn[0][0]
block4c_se_squeeze (GlobalAvera (None, 480)	0	block4c_activation[0][0]
block4c_se_reshape (Reshape) (None, 1, 1, 480)	0	block4c_se_squeeze[0][0]

block4b_se_reduce (Conv2D)	(None, 1, 1, 20)	9620	block4b_se_reshape[0][0]
block4b_se_expand (Conv2D)	(None, 1, 1, 480)	10080	block4b_se_reduce[0][0]
block4b_se_excite (Multiply)	(None, 32, 32, 480)	0	block4b_activation[0][0] block4b_se_expand[0][0]
block4b_project_conv (Conv2D)	(None, 32, 32, 80)	38400	block4b_se_excite[0][0]
block4b_project_bn (BatchNormal)	(None, 32, 32, 80)	320	block4b_project_conv[0][0]
block4b_drop (FixedDropout)	(None, 32, 32, 80)	0	block4b_project_bn[0][0]
block4b_add (Add)	(None, 32, 32, 80)	0	block4b_drop[0][0] block4a_project_bn[0][0]
block4c_expand_conv (Conv2D)	(None, 32, 32, 480)	38400	block4b_add[0][0]
block4c_expand_bn (BatchNormal)	(None, 32, 32, 480)	1920	block4c_expand_conv[0][0]
block4c_expand_activation (Acti	(None, 32, 32, 480)	0	block4c_expand_bn[0][0]
block4c_dwconv (DepthwiseConv2D)	(None, 32, 32, 480)	4320	block4c_expand_activation[0][0]
block4c_bn (BatchNormalization)	(None, 32, 32, 480)	1920	block4c_dwconv[0][0]
block4c_activation (Activation)	(None, 32, 32, 480)	0	block4c_bn[0][0]
block4c_se_squeeze (GlobalAvera	(None, 480)	0	block4c_activation[0][0]
block4c_se_reshape (Reshape)	(None, 1, 1, 480)	0	block4c_se_squeeze[0][0]
block4c_se_reduce (Conv2D)	(None, 1, 1, 20)	9620	block4c_se_reshape[0][0]
block4c_se_expand (Conv2D)	(None, 1, 1, 480)	10080	block4c_se_reduce[0][0]
block4c_se_excite (Multiply)	(None, 32, 32, 480)	0	block4c_activation[0][0] block4c_se_expand[0][0]
block4c_project_conv (Conv2D)	(None, 32, 32, 80)	38400	block4c_se_excite[0][0]
block4c_project_bn (BatchNormal)	(None, 32, 32, 80)	320	block4c_project_conv[0][0]
block4c_drop (FixedDropout)	(None, 32, 32, 80)	0	block4c_project_bn[0][0]
block4c_add (Add)	(None, 32, 32, 80)	0	block4c_drop[0][0] block4b_add[0][0]
block5a_expand_conv (Conv2D)	(None, 32, 32, 480)	38400	block4c_add[0][0]
block5a_expand_bn (BatchNormal)	(None, 32, 32, 480)	1920	block5a_expand_conv[0][0]
block5a_expand_activation (Acti	(None, 32, 32, 480)	0	block5a_expand_bn[0][0]
block5a_dwconv (DepthwiseConv2D)	(None, 32, 32, 480)	12000	block5a_expand_activation[0][0]
block5a_bn (BatchNormalization)	(None, 32, 32, 480)	1920	block5a_dwconv[0][0]
block5a_activation (Activation)	(None, 32, 32, 480)	0	block5a_bn[0][0]
block5a_se_squeeze (GlobalAvera	(None, 480)	0	block5a_activation[0][0]
block5a_se_reshape (Reshape)	(None, 1, 1, 480)	0	block5a_se_squeeze[0][0]
block5a_se_reduce (Conv2D)	(None, 1, 1, 20)	9620	block5a_se_reshape[0][0]
block5a_se_expand (Conv2D)	(None, 1, 1, 480)	10080	block5a_se_reduce[0][0]
block5a_se_excite (Multiply)	(None, 32, 32, 480)	0	block5a_activation[0][0] block5a_se_expand[0][0]
block5a_project_conv (Conv2D)	(None, 32, 32, 112)	53760	block5a_se_excite[0][0]
block5a_project_bn (BatchNormal)	(None, 32, 32, 112)	448	block5a_project_conv[0][0]
block5b_expand_conv (Conv2D)	(None, 32, 32, 672)	75264	block5a_project_bn[0][0]
block5b_expand_bn (BatchNormal)	(None, 32, 32, 672)	2688	block5b_expand_conv[0][0]
block5b_expand_activation (Acti	(None, 32, 32, 672)	0	block5b_expand_bn[0][0]
block5b_dwconv (DepthwiseConv2D)	(None, 32, 32, 672)	16800	block5b_expand_activation[0][0]
block5b_bn (BatchNormalization)	(None, 32, 32, 672)	2688	block5b_dwconv[0][0]
block5b_activation (Activation)	(None, 32, 32, 672)	0	block5b_bn[0][0]
block5b_se_squeeze (GlobalAvera	(None, 672)	0	block5b_activation[0][0]
block5b_se_reshape (Reshape)	(None, 1, 1, 672)	0	block5b_se_squeeze[0][0]
block5b_se_reduce (Conv2D)	(None, 1, 1, 20)	18844	block5b_se_reshape[0][0]
block5b_se_expand (Conv2D)	(None, 1, 1, 672)	19488	block5b_se_reduce[0][0]

block5b_se_excite (Multiply)	(None, 32, 32, 672)	0	block5b_activation[0][0] block5b_se_expand[0][0]
block5b_project_conv (Conv2D)	(None, 32, 32, 112)	75264	block5b_se_excite[0][0]
block5b_project_bn (BatchNormal)	(None, 32, 32, 112)	448	block5b_project_conv[0][0]
block5b_drop (FixedDropout)	(None, 32, 32, 112)	0	block5b_project_bn[0][0]
block5b_add (Add)	(None, 32, 32, 112)	0	block5b_drop[0][0] block5a_project_bn[0][0]
block5c_expand_conv (Conv2D)	(None, 32, 32, 672)	75264	block5b_add[0][0]
block5c_expand_bn (BatchNormal)	(None, 32, 32, 672)	2688	block5c_expand_conv[0][0]
block5c_expand_activation (Acti)	(None, 32, 32, 672)	0	block5c_expand_bn[0][0]
block5c_dwconv (DepthwiseConv2D)	(None, 32, 32, 672)	16800	block5c_expand_activation[0][0]
block5c_bn (BatchNormalization)	(None, 32, 32, 672)	2688	block5c_dwconv[0][0]
block5c_activation (Activation)	(None, 32, 32, 672)	0	block5c_bn[0][0]
block5c_se_squeeze (GlobalAvera)	(None, 672)	0	block5c_activation[0][0]
block5c_se_reshape (Reshape)	(None, 1, 1, 672)	0	block5c_se_squeeze[0][0]
block5c_se_reduce (Conv2D)	(None, 1, 1, 28)	18844	block5c_se_reshape[0][0]
block5c_se_expand (Conv2D)	(None, 1, 1, 672)	19488	block5c_se_reduce[0][0]
block5c_se_excite (Multiply)	(None, 32, 32, 672)	0	block5c_activation[0][0] block5c_se_expand[0][0]
block5c_project_conv (Conv2D)	(None, 32, 32, 112)	75264	block5c_se_excite[0][0]
block5c_project_bn (BatchNormal)	(None, 32, 32, 112)	448	block5c_project_conv[0][0]
block5c_drop (FixedDropout)	(None, 32, 32, 112)	0	block5c_project_bn[0][0]
block5c_add (Add)	(None, 32, 32, 112)	0	block5c_drop[0][0] block5b_add[0][0]
block6a_expand_conv (Conv2D)	(None, 32, 32, 672)	75264	block5c_add[0][0]
block6a_expand_bn (BatchNormal)	(None, 32, 32, 672)	2688	block6a_expand_conv[0][0]
block6a_expand_activation (Acti)	(None, 32, 32, 672)	0	block6a_expand_bn[0][0]
block6a_dwconv (DepthwiseConv2D)	(None, 16, 16, 672)	16800	block6a_expand_activation[0][0]
block6a_bn (BatchNormalization)	(None, 16, 16, 672)	2688	block6a_dwconv[0][0]
block6a_activation (Activation)	(None, 16, 16, 672)	0	block6a_bn[0][0]
block6a_se_squeeze (GlobalAvera)	(None, 672)	0	block6a_activation[0][0]
block6a_se_reshape (Reshape)	(None, 1, 1, 672)	0	block6a_se_squeeze[0][0]
block6a_se_reduce (Conv2D)	(None, 1, 1, 28)	18844	block6a_se_reshape[0][0]
block6a_se_expand (Conv2D)	(None, 1, 1, 672)	19488	block6a_se_reduce[0][0]
block6a_se_excite (Multiply)	(None, 16, 16, 672)	0	block6a_activation[0][0] block6a_se_expand[0][0]
block6a_project_conv (Conv2D)	(None, 16, 16, 192)	129024	block6a_se_excite[0][0]
block6a_project_bn (BatchNormal)	(None, 16, 16, 192)	768	block6a_project_conv[0][0]
block6b_expand_conv (Conv2D)	(None, 16, 16, 1152)	221184	block6a_project_bn[0][0]
block6b_expand_bn (BatchNormal)	(None, 16, 16, 1152)	4608	block6b_expand_conv[0][0]
block6b_expand_activation (Acti)	(None, 16, 16, 1152)	0	block6b_expand_bn[0][0]
block6b_dwconv (DepthwiseConv2D)	(None, 16, 16, 1152)	28800	block6b_expand_activation[0][0]
block6b_bn (BatchNormalization)	(None, 16, 16, 1152)	4608	block6b_dwconv[0][0]
block6b_activation (Activation)	(None, 16, 16, 1152)	0	block6b_bn[0][0]
block6b_se_squeeze (GlobalAvera)	(None, 1152)	0	block6b_activation[0][0]
block6b_se_reshape (Reshape)	(None, 1, 1, 1152)	0	block6b_se_squeeze[0][0]
block6b_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	block6b_se_reshape[0][0]
block6b_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	block6b_se_reduce[0][0]
block6b_se_excite (Multiply)	(None, 16, 16, 1152)	0	block6b_activation[0][0] block6b_se_expand[0][0]
block6b_project_conv (Conv2D)	(None, 16, 16, 192)	221184	block6b_se_excite[0][0]

block6b_project_bn (BatchNormal	(None, 16, 16, 192)	768	block6b_project_conv[0][0]
block6b_drop (FixedDropout)	(None, 16, 16, 192)	0	block6b_project_bn[0][0]
block6b_add (Add)	(None, 16, 16, 192)	0	block6b_drop[0][0] block6a_project_bn[0][0]
block6c_expand_conv (Conv2D)	(None, 16, 16, 1152)	221184	block6b_add[0][0]
block6c_expand_bn (BatchNormali	(None, 16, 16, 1152)	4608	block6c_expand_conv[0][0]
block6c_expand_activation (Acti	(None, 16, 16, 1152)	0	block6c_expand_bn[0][0]
block6c_dwconv (DepthwiseConv2D	(None, 16, 16, 1152)	28800	block6c_expand_activation[0][0]
block6c_bn (BatchNormalization)	(None, 16, 16, 1152)	4608	block6c_dwconv[0][0]
block6c_activation (Activation)	(None, 16, 16, 1152)	0	block6c_bn[0][0]
block6c_se_squeeze (GlobalAvera	(None, 1152)	0	block6c_activation[0][0]
block6c_se_reshape (Reshape)	(None, 1, 1, 1152)	0	block6c_se_squeeze[0][0]
block6c_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	block6c_se_reshape[0][0]
block6c_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	block6c_se_reduce[0][0]
block6c_se_excite (Multiply)	(None, 16, 16, 1152)	0	block6c_activation[0][0] block6c_se_expand[0][0]
block6c_project_conv (Conv2D)	(None, 16, 16, 192)	221184	block6c_se_excite[0][0]
block6c_project_bn (BatchNormal	(None, 16, 16, 192)	768	block6c_project_conv[0][0]
block6c_drop (FixedDropout)	(None, 16, 16, 192)	0	block6c_project_bn[0][0]
block6c_add (Add)	(None, 16, 16, 192)	0	block6c_drop[0][0] block6b_add[0][0]
block6d_expand_conv (Conv2D)	(None, 16, 16, 1152)	221184	block6c_add[0][0]
block6d_expand_bn (BatchNormali	(None, 16, 16, 1152)	4608	block6d_expand_conv[0][0]
block6d_expand_activation (Acti	(None, 16, 16, 1152)	0	block6d_expand_bn[0][0]
block6d_dwconv (DepthwiseConv2D	(None, 16, 16, 1152)	28800	block6d_expand_activation[0][0]
block6d_bn (BatchNormalization)	(None, 16, 16, 1152)	4608	block6d_dwconv[0][0]
block6d_activation (Activation)	(None, 16, 16, 1152)	0	block6d_bn[0][0]
block6d_se_squeeze (GlobalAvera	(None, 1152)	0	block6d_activation[0][0]
block6d_se_reshape (Reshape)	(None, 1, 1, 1152)	0	block6d_se_squeeze[0][0]
block6d_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	block6d_se_reshape[0][0]
block6d_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	block6d_se_reduce[0][0]
block6d_se_excite (Multiply)	(None, 16, 16, 1152)	0	block6d_activation[0][0] block6d_se_expand[0][0]
block6d_project_conv (Conv2D)	(None, 16, 16, 192)	221184	block6d_se_excite[0][0]
block6d_project_bn (BatchNormal	(None, 16, 16, 192)	768	block6d_project_conv[0][0]
block6d_drop (FixedDropout)	(None, 16, 16, 192)	0	block6d_project_bn[0][0]
block6d_add (Add)	(None, 16, 16, 192)	0	block6d_drop[0][0] block6c_add[0][0]
block7a_expand_conv (Conv2D)	(None, 16, 16, 1152)	221184	block6d_add[0][0]
block7a_expand_bn (BatchNormali	(None, 16, 16, 1152)	4608	block7a_expand_conv[0][0]
block7a_expand_activation (Acti	(None, 16, 16, 1152)	0	block7a_expand_bn[0][0]
block7a_dwconv (DepthwiseConv2D	(None, 16, 16, 1152)	10368	block7a_expand_activation[0][0]
block7a_bn (BatchNormalization)	(None, 16, 16, 1152)	4608	block7a_dwconv[0][0]
block7a_activation (Activation)	(None, 16, 16, 1152)	0	block7a_bn[0][0]
block7a_se_squeeze (GlobalAvera	(None, 1152)	0	block7a_activation[0][0]
block7a_se_reshape (Reshape)	(None, 1, 1, 1152)	0	block7a_se_squeeze[0][0]
block7a_se_reduce (Conv2D)	(None, 1, 1, 48)	55344	block7a_se_reshape[0][0]
block7a_se_expand (Conv2D)	(None, 1, 1, 1152)	56448	block7a_se_reduce[0][0]
block7a_se_excite (Multiply)	(None, 16, 16, 1152)	0	block7a_activation[0][0] block7a_se_expand[0][0]
block7a_project_conv (Conv2D)	(None, 16, 16, 320)	368640	block7a_se_excite[0][0]

block7a_project_bn (BatchNormal (None, 16, 16, 320))	1280	block7a_project_conv[0][0]
resample_p6/conv2d (Conv2D) (None, 16, 16, 64)	20544	block7a_project_bn[0][0]
resample_p6/bn (BatchNormalizat (None, 16, 16, 64)	256	resample_p6/conv2d[0][0]
resample_p6/maxpool (MaxPooling (None, 8, 8, 64)	0	resample_p6/bn[0][0]
resample_p7/maxpool (MaxPooling (None, 4, 4, 64)	0	resample_p6/maxpool[0][0]
up_sampling2d (UpSampling2D) (None, 8, 8, 64)	0	resample_p7/maxpool[0][0]
fpn_cells/cell_0/fnode0/add (Ad (None, 8, 8, 64)	0	resample_p6/maxpool[0][0] up_sampling2d[0][0]
activation (Activation) (None, 8, 8, 64)	0	fpn_cells/cell_0/fnode0/add[0][0]
fpn_cells/cell_0/fnode0/op_aftc (None, 8, 8, 64)	4736	activation[0][0]
fpn_cells/cell_0/fnode1/resampl (None, 16, 16, 64)	20544	block7a_project_bn[0][0]
fpn_cells/cell_0/fnode0/op_aftc (None, 8, 8, 64)	256	fpn_cells/cell_0/fnode0/op_after_
fpn_cells/cell_0/fnode1/resampl (None, 16, 16, 64)	256	fpn_cells/cell_0/fnode1/resample_
up_sampling2d_1 (UpSampling2D) (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode0/op_after_
fpn_cells/cell_0/fnode1/add (Ad (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode1/resample_ up_sampling2d_1[0][0]
activation_1 (Activation) (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode1/add[0][0]
fpn_cells/cell_0/fnode1/op_aftc (None, 16, 16, 64)	4736	activation_1[0][0]
fpn_cells/cell_0/fnode2/resampl (None, 32, 32, 64)	7232	block5c_add[0][0]
fpn_cells/cell_0/fnode1/op_aftc (None, 16, 16, 64)	256	fpn_cells/cell_0/fnode1/op_after_
fpn_cells/cell_0/fnode2/resampl (None, 32, 32, 64)	256	fpn_cells/cell_0/fnode2/resample_
up_sampling2d_2 (UpSampling2D) (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode1/op_after_
fpn_cells/cell_0/fnode2/add (Ad (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode2/resample_ up_sampling2d_2[0][0]
activation_2 (Activation) (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode2/add[0][0]
fpn_cells/cell_0/fnode2/op_aftc (None, 32, 32, 64)	4736	activation_2[0][0]
fpn_cells/cell_0/fnode3/resampl (None, 64, 64, 64)	2624	block3b_add[0][0]
fpn_cells/cell_0/fnode2/op_aftc (None, 32, 32, 64)	256	fpn_cells/cell_0/fnode2/op_after_
fpn_cells/cell_0/fnode3/resampl (None, 64, 64, 64)	256	fpn_cells/cell_0/fnode3/resample_
up_sampling2d_3 (UpSampling2D) (None, 64, 64, 64)	0	fpn_cells/cell_0/fnode2/op_after_
fpn_cells/cell_0/fnode3/add (Ad (None, 64, 64, 64)	0	fpn_cells/cell_0/fnode3/resample_ up_sampling2d_3[0][0]
activation_3 (Activation) (None, 64, 64, 64)	0	fpn_cells/cell_0/fnode3/add[0][0]
fpn_cells/cell_0/fnode3/op_aftc (None, 64, 64, 64)	4736	activation_3[0][0]
fpn_cells/cell_0/fnode3/op_aftc (None, 64, 64, 64)	256	fpn_cells/cell_0/fnode3/op_after_
fpn_cells/cell_0/fnode4/resampl (None, 32, 32, 64)	7232	block5c_add[0][0]
fpn_cells/cell_0/fnode4/resampl (None, 32, 32, 64)	256	fpn_cells/cell_0/fnode4/resample_
max_pooling2d (MaxPooling2D) (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode3/op_after_
fpn_cells/cell_0/fnode4/add (Ad (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode4/resample_ fpn_cells/cell_0/fnode2/op_after_ max_pooling2d[0][0]
activation_4 (Activation) (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode4/add[0][0]
fpn_cells/cell_0/fnode4/op_aftc (None, 32, 32, 64)	4736	activation_4[0][0]
fpn_cells/cell_0/fnode5/resampl (None, 16, 16, 64)	20544	block7a_project_bn[0][0]
fpn_cells/cell_0/fnode4/op_aftc (None, 32, 32, 64)	256	fpn_cells/cell_0/fnode4/op_after_
fpn_cells/cell_0/fnode5/resampl (None, 16, 16, 64)	256	fpn_cells/cell_0/fnode5/resample_
max_pooling2d_1 (MaxPooling2D) (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode4/op_after_
fpn_cells/cell_0/fnode5/add (Ad (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode5/resample_ fpn_cells/cell_0/fnode1/op_after_ max_pooling2d_1[0][0]
activation_5 (Activation) (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode5/add[0][0]
fpn_cells/cell_0/fnode5/op_aftc (None, 16, 16, 64)	4736	activation_5[0][0]

fpn_cells/cell_0/fnode5/op_aftc	(None, 16, 16, 64)	256	fpn_cells/cell_0/fnode5/op_after_
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0	fpn_cells/cell_0/fnode5/op_after_
fpn_cells/cell_0/fnode6/add	(Ad (None, 8, 8, 64)	0	resample_p6/maxpool[0][0] fpn_cells/cell_0/fnode6/op_after_ max_pooling2d_2[0][0]
activation_6 (Activation)	(None, 8, 8, 64)	0	fpn_cells/cell_0/fnode6/add[0][0]
fpn_cells/cell_0/fnode6/op_aftc	(None, 8, 8, 64)	4736	activation_6[0][0]
fpn_cells/cell_0/fnode6/op_aftc	(None, 8, 8, 64)	256	fpn_cells/cell_0/fnode6/op_after_
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 64)	0	fpn_cells/cell_0/fnode6/op_after_
fpn_cells/cell_0/fnode7/add	(Ad (None, 4, 4, 64)	0	resample_p7/maxpool[0][0] max_pooling2d_3[0][0]
activation_7 (Activation)	(None, 4, 4, 64)	0	fpn_cells/cell_0/fnode7/add[0][0]
fpn_cells/cell_0/fnode7/op_aftc	(None, 4, 4, 64)	4736	activation_7[0][0]
fpn_cells/cell_0/fnode7/op_aftc	(None, 4, 4, 64)	256	fpn_cells/cell_0/fnode7/op_after_
up_sampling2d_4 (UpSampling2D)	(None, 8, 8, 64)	0	fpn_cells/cell_0/fnode7/op_after_
fpn_cells/cell_1/fnode8/add	(Ad (None, 8, 8, 64)	0	fpn_cells/cell_0/fnode8/op_after_ up_sampling2d_4[0][0]
activation_8 (Activation)	(None, 8, 8, 64)	0	fpn_cells/cell_1/fnode8/add[0][0]
fpn_cells/cell_1/fnode8/op_aftc	(None, 8, 8, 64)	4736	activation_8[0][0]
fpn_cells/cell_1/fnode8/op_aftc	(None, 8, 8, 64)	256	fpn_cells/cell_1/fnode8/op_after_
up_sampling2d_5 (UpSampling2D)	(None, 16, 16, 64)	0	fpn_cells/cell_1/fnode8/op_after_
fpn_cells/cell_1/fnode1/add	(Ad (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode1/op_after_ up_sampling2d_5[0][0]
activation_9 (Activation)	(None, 16, 16, 64)	0	fpn_cells/cell_1/fnode1/add[0][0]
fpn_cells/cell_1/fnode1/op_aftc	(None, 16, 16, 64)	4736	activation_9[0][0]
fpn_cells/cell_1/fnode1/op_aftc	(None, 16, 16, 64)	256	fpn_cells/cell_1/fnode1/op_after_
up_sampling2d_6 (UpSampling2D)	(None, 32, 32, 64)	0	fpn_cells/cell_1/fnode1/op_after_
fpn_cells/cell_1/fnode2/add	(Ad (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode2/op_after_ up_sampling2d_6[0][0]
activation_10 (Activation)	(None, 32, 32, 64)	0	fpn_cells/cell_1/fnode2/add[0][0]
fpn_cells/cell_1/fnode2/op_aftc	(None, 32, 32, 64)	4736	activation_10[0][0]
fpn_cells/cell_1/fnode2/op_aftc	(None, 32, 32, 64)	256	fpn_cells/cell_1/fnode2/op_after_
up_sampling2d_7 (UpSampling2D)	(None, 64, 64, 64)	0	fpn_cells/cell_1/fnode2/op_after_
fpn_cells/cell_1/fnode3/add	(Ad (None, 64, 64, 64)	0	fpn_cells/cell_0/fnode3/op_after_ up_sampling2d_7[0][0]
activation_11 (Activation)	(None, 64, 64, 64)	0	fpn_cells/cell_1/fnode3/add[0][0]
fpn_cells/cell_1/fnode3/op_aftc	(None, 64, 64, 64)	4736	activation_11[0][0]
fpn_cells/cell_1/fnode3/op_aftc	(None, 64, 64, 64)	256	fpn_cells/cell_1/fnode3/op_after_
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 64)	0	fpn_cells/cell_1/fnode3/op_after_
fpn_cells/cell_1/fnode4/add	(Ad (None, 32, 32, 64)	0	fpn_cells/cell_0/fnode2/op_after_ fpn_cells/cell_1/fnode2/op_after_ max_pooling2d_4[0][0]
activation_12 (Activation)	(None, 32, 32, 64)	0	fpn_cells/cell_1/fnode4/add[0][0]
fpn_cells/cell_1/fnode4/op_aftc	(None, 32, 32, 64)	4736	activation_12[0][0]
fpn_cells/cell_1/fnode4/op_aftc	(None, 32, 32, 64)	256	fpn_cells/cell_1/fnode4/op_after_
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 64)	0	fpn_cells/cell_1/fnode4/op_after_
fpn_cells/cell_1/fnode5/add	(Ad (None, 16, 16, 64)	0	fpn_cells/cell_0/fnode1/op_after_ fpn_cells/cell_1/fnode1/op_after_ max_pooling2d_5[0][0]
activation_13 (Activation)	(None, 16, 16, 64)	0	fpn_cells/cell_1/fnode5/add[0][0]
fpn_cells/cell_1/fnode5/op_aftc	(None, 16, 16, 64)	4736	activation_13[0][0]
fpn_cells/cell_1/fnode5/op_aftc	(None, 16, 16, 64)	256	fpn_cells/cell_1/fnode5/op_after_
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 64)	0	fpn_cells/cell_1/fnode5/op_after_

fpn_cells/cell_1/fnode6/add (Ad (None, 8, 8, 64)	0	fpn_cells/cell_0/fnode0/op_after_fpn_cells/cell_1/fnode0/op_after_max_pooling2d_6[0][0]
activation_14 (Activation) (None, 8, 8, 64)	0	fpn_cells/cell_1/fnode6/add[0][0]
fpn_cells/cell_1/fnode6/op_afte (None, 8, 8, 64)	4736	activation_14[0][0]
fpn_cells/cell_1/fnode6/op_afte (None, 8, 8, 64)	256	fpn_cells/cell_1/fnode6/op_after_
max_pooling2d_7 (MaxPooling2D) (None, 4, 4, 64)	0	fpn_cells/cell_1/fnode6/op_after_
fpn_cells/cell_1/fnode7/add (Ad (None, 4, 4, 64)	0	fpn_cells/cell_0/fnode7/op_after_max_pooling2d_7[0][0]
activation_15 (Activation) (None, 4, 4, 64)	0	fpn_cells/cell_1/fnode7/add[0][0]
fpn_cells/cell_1/fnode7/op_afte (None, 4, 4, 64)	4736	activation_15[0][0]
fpn_cells/cell_1/fnode7/op_afte (None, 4, 4, 64)	256	fpn_cells/cell_1/fnode7/op_after_
up_sampling2d_8 (UpSampling2D) (None, 8, 8, 64)	0	fpn_cells/cell_1/fnode7/op_after_
fpn_cells/cell_2/fnode0/add (Ad (None, 8, 8, 64)	0	fpn_cells/cell_1/fnode0/op_after_up_sampling2d_8[0][0]
activation_16 (Activation) (None, 8, 8, 64)	0	fpn_cells/cell_2/fnode0/add[0][0]
fpn_cells/cell_2/fnode0/op_afte (None, 8, 8, 64)	4736	activation_16[0][0]
fpn_cells/cell_2/fnode0/op_afte (None, 8, 8, 64)	256	fpn_cells/cell_2/fnode0/op_after_
up_sampling2d_9 (UpSampling2D) (None, 16, 16, 64)	0	fpn_cells/cell_2/fnode0/op_after_
fpn_cells/cell_2/fnode1/add (Ad (None, 16, 16, 64)	0	fpn_cells/cell_1/fnode1/op_after_up_sampling2d_9[0][0]
activation_17 (Activation) (None, 16, 16, 64)	0	fpn_cells/cell_2/fnode1/add[0][0]
fpn_cells/cell_2/fnode1/op_afte (None, 16, 16, 64)	4736	activation_17[0][0]
fpn_cells/cell_2/fnode1/op_afte (None, 16, 16, 64)	256	fpn_cells/cell_2/fnode1/op_after_
up_sampling2d_10 (UpSampling2D) (None, 32, 32, 64)	0	fpn_cells/cell_2/fnode1/op_after_
fpn_cells/cell_2/fnode2/add (Ad (None, 32, 32, 64)	0	fpn_cells/cell_1/fnode2/op_after_up_sampling2d_10[0][0]
activation_18 (Activation) (None, 32, 32, 64)	0	fpn_cells/cell_2/fnode2/add[0][0]
fpn_cells/cell_2/fnode2/op_afte (None, 32, 32, 64)	4736	activation_18[0][0]
fpn_cells/cell_2/fnode2/op_afte (None, 32, 32, 64)	256	fpn_cells/cell_2/fnode2/op_after_
up_sampling2d_11 (UpSampling2D) (None, 64, 64, 64)	0	fpn_cells/cell_2/fnode2/op_after_
fpn_cells/cell_2/fnode3/add (Ad (None, 64, 64, 64)	0	fpn_cells/cell_1/fnode3/op_after_up_sampling2d_11[0][0]
activation_19 (Activation) (None, 64, 64, 64)	0	fpn_cells/cell_2/fnode3/add[0][0]
fpn_cells/cell_2/fnode3/op_afte (None, 64, 64, 64)	4736	activation_19[0][0]
fpn_cells/cell_2/fnode3/op_afte (None, 64, 64, 64)	256	fpn_cells/cell_2/fnode3/op_after_
class_net/class-0 (SeparableCon multiple	4736	fpn_cells/cell_2/fnode3/op_after_fpn_cells/cell_2/fnode2/op_after_fpn_cells/cell_2/fnode1/op_after_fpn_cells/cell_2/fnode0/op_after_fpn_cells/cell_2/fnode7/op_after_
class_net/class-0-bn-3 (BatchNo (None, 64, 64, 64)	256	class_net/class-0[0][0]
box_net/box-0 (SeparableConv2D) multiple	4736	fpn_cells/cell_2/fnode3/op_after_fpn_cells/cell_2/fnode2/op_after_fpn_cells/cell_2/fnode1/op_after_fpn_cells/cell_2/fnode0/op_after_fpn_cells/cell_2/fnode7/op_after_
lambda_1 (Lambda) multiple	0	class_net/class-0-bn-3[0][0] class_net/class-1-bn-3[0][0] class_net/class-2-bn-3[0][0] class_net/class-0-bn-4[0][0] class_net/class-1-bn-4[0][0] class_net/class-2-bn-4[0][0] class_net/class-0-bn-5[0][0] class_net/class-1-bn-5[0][0] class_net/class-2-bn-5[0][0] class_net/class-0-bn-6[0][0] class_net/class-1-bn-6[0][0] class_net/class-2-bn-6[0][0] class_net/class-0-bn-7[0][0] class_net/class-1-bn-7[0][0] class_net/class-2-bn-7[0][0]

box_net/box-0-bn-3 (BatchNormal (None, 64, 64, 64)	256	box_net/box-0[0][0]
class_net/class-1 (SeparableCon multiple	4736	lambda_1[0][0] lambda_1[3][0] lambda_1[6][0] lambda_1[9][0] lambda_1[12][0]
lambda (Lambda) multiple	0	box_net/box-0-bn-3[0][0] box_net/box-1-bn-3[0][0] box_net/box-2-bn-3[0][0] box_net/box-0-bn-4[0][0] box_net/box-1-bn-4[0][0] box_net/box-2-bn-4[0][0] box_net/box-0-bn-5[0][0] box_net/box-1-bn-5[0][0] box_net/box-2-bn-5[0][0] box_net/box-0-bn-6[0][0] box_net/box-1-bn-6[0][0] box_net/box-2-bn-6[0][0] box_net/box-0-bn-7[0][0] box_net/box-1-bn-7[0][0] box_net/box-2-bn-7[0][0]
class_net/class-1-bn-3 (BatchNo (None, 64, 64, 64)	256	class_net/class-1[0][0]
max_pooling2d_8 (MaxPooling2D) (None, 32, 32, 64)	0	fpn_cells/cell_2/fnode3/op_after_
box_net/box-1 (SeparableConv2D) multiple	4736	lambda[0][0] lambda[3][0] lambda[6][0] lambda[9][0] lambda[12][0]
fpn_cells/cell_2/fnode4/add (Ad (None, 32, 32, 64)	0	fpn_cells/cell_1/fnode2/op_after_ fpn_cells/cell_2/fnode2/op_after_ max_pooling2d_8[0][0]
box_net/box-1-bn-3 (BatchNormal (None, 64, 64, 64)	256	box_net/box-1[0][0]
class_net/class-2 (SeparableCon multiple	4736	lambda_1[1][0] lambda_1[4][0] lambda_1[7][0] lambda_1[10][0] lambda_1[13][0]
activation_20 (Activation) (None, 32, 32, 64)	0	fpn_cells/cell_2/fnode4/add[0][0]
class_net/class-2-bn-3 (BatchNo (None, 64, 64, 64)	256	class_net/class-2[0][0]
class_net/class-0-bn-4 (BatchNo (None, 32, 32, 64)	256	class_net/class-0[1][0]
fpn_cells/cell_2/fnode4/op_afte (None, 32, 32, 64)	4736	activation_20[0][0]
box_net/box-2 (SeparableConv2D) multiple	4736	lambda[1][0] lambda[4][0] lambda[7][0] lambda[10][0] lambda[13][0]
fpn_cells/cell_2/fnode4/op_afte (None, 32, 32, 64)	256	fpn_cells/cell_2/fnode4/op_after_
box_net/box-2-bn-3 (BatchNormal (None, 64, 64, 64)	256	box_net/box-2[0][0]
box_net/box-0-bn-4 (BatchNormal (None, 32, 32, 64)	256	box_net/box-0[1][0]
max_pooling2d_9 (MaxPooling2D) (None, 16, 16, 64)	0	fpn_cells/cell_2/fnode4/op_after_
class_net/class-1-bn-4 (BatchNo (None, 32, 32, 64)	256	class_net/class-1[1][0]
fpn_cells/cell_2/fnode5/add (Ad (None, 16, 16, 64)	0	fpn_cells/cell_1/fnode1/op_after_ fpn_cells/cell_2/fnode1/op_after_ max_pooling2d_9[0][0]
activation_21 (Activation) (None, 16, 16, 64)	0	fpn_cells/cell_2/fnode5/add[0][0]
box_net/box-1-bn-4 (BatchNormal (None, 32, 32, 64)	256	box_net/box-1[1][0]
fpn_cells/cell_2/fnode5/op_afte (None, 16, 16, 64)	4736	activation_21[0][0]
class_net/class-2-bn-4 (BatchNo (None, 32, 32, 64)	256	class_net/class-2[1][0]
class_net/class-0-bn-5 (BatchNo (None, 16, 16, 64)	256	class_net/class-0[2][0]
fpn_cells/cell_2/fnode5/op_afte (None, 16, 16, 64)	256	fpn_cells/cell_2/fnode5/op_after_
max_pooling2d_10 (MaxPooling2D) (None, 8, 8, 64)	0	fpn_cells/cell_2/fnode5/op_after_
box_net/box-2-bn-4 (BatchNormal (None, 32, 32, 64)	256	box_net/box-2[1][0]
box_net/box-0-bn-5 (BatchNormal (None, 16, 16, 64)	256	box_net/box-0[2][0]
fpn_cells/cell_2/fnode6/add (Ad (None, 8, 8, 64)	0	fpn_cells/cell_1/fnode0/op_after_ fpn_cells/cell_2/fnode0/op_after_ max_pooling2d_10[0][0]

class_net/class-1-bn-5 (BatchNo (None, 16, 16, 64)	256	class_net/class-1[2][0]
activation_22 (Activation) (None, 8, 8, 64)	0	fpn_cells/cell_2/fnode6/add[0][0]
fpn_cells/cell_2/fnode6/op_afte (None, 8, 8, 64)	4736	activation_22[0][0]
box_net/box-1-bn-5 (BatchNormal (None, 16, 16, 64)	256	box_net/box-1[2][0]
fpn_cells/cell_2/fnode6/op_afte (None, 8, 8, 64)	256	fpn_cells/cell_2/fnode6/op_after_
class_net/class-2-bn-5 (BatchNo (None, 16, 16, 64)	256	class_net/class-2[2][0]
class_net/class-0-bn-6 (BatchNo (None, 8, 8, 64)	256	class_net/class-0[3][0]
max_pooling2d_11 (MaxPooling2D) (None, 4, 4, 64)	0	fpn_cells/cell_2/fnode6/op_after_
fpn_cells/cell_2/fnode7/add (Ad (None, 4, 4, 64)	0	fpn_cells/cell_1/fnode7/op_after_max_pooling2d_11[0][0]
box_net/box-2-bn-5 (BatchNormal (None, 16, 16, 64)	256	box_net/box-2[2][0]
box_net/box-0-bn-6 (BatchNormal (None, 8, 8, 64)	256	box_net/box-0[3][0]
activation_23 (Activation) (None, 4, 4, 64)	0	fpn_cells/cell_2/fnode7/add[0][0]
class_net/class-1-bn-6 (BatchNo (None, 8, 8, 64)	256	class_net/class-1[3][0]
fpn_cells/cell_2/fnode7/op_afte (None, 4, 4, 64)	4736	activation_23[0][0]
fpn_cells/cell_2/fnode7/op_afte (None, 4, 4, 64)	256	fpn_cells/cell_2/fnode7/op_after_
box_net/box-1-bn-6 (BatchNormal (None, 8, 8, 64)	256	box_net/box-1[3][0]
class_net/class-2-bn-6 (BatchNo (None, 8, 8, 64)	256	class_net/class-2[3][0]
class_net/class-0-bn-7 (BatchNo (None, 4, 4, 64)	256	class_net/class-0[4][0]
box_net/box-2-bn-6 (BatchNormal (None, 8, 8, 64)	256	box_net/box-2[3][0]
box_net/box-0-bn-7 (BatchNormal (None, 4, 4, 64)	256	box_net/box-0[4][0]
class_net/class-1-bn-7 (BatchNo (None, 4, 4, 64)	256	class_net/class-1[4][0]
box_net/box-1-bn-7 (BatchNormal (None, 4, 4, 64)	256	box_net/box-1[4][0]
class_net/class-2-bn-7 (BatchNo (None, 4, 4, 64)	256	class_net/class-2[4][0]
box_net/box-2-bn-7 (BatchNormal (None, 4, 4, 64)	256	box_net/box-2[4][0]
class_net/class-predict (Separa multiple	3501	lambda_1[2][0] lambda_1[5][0] lambda_1[8][0] lambda_1[11][0] lambda_1[14][0]
reshape_1 (Reshape) (None, None, 5)	0	class_net/class-predict[0][0] class_net/class-predict[1][0] class_net/class-predict[2][0] class_net/class-predict[3][0] class_net/class-predict[4][0]
box_net/box-predict (SeparableC multiple	2916	lambda[2][0] lambda[5][0] lambda[8][0] lambda[11][0] lambda[14][0]
activation_24 (Activation) (None, None, 5)	0	reshape_1[0][0] reshape_1[1][0] reshape_1[2][0] reshape_1[3][0] reshape_1[4][0]
reshape (Reshape) (None, None, 4)	0	box_net/box-predict[0][0] box_net/box-predict[1][0] box_net/box-predict[2][0] box_net/box-predict[3][0] box_net/box-predict[4][0]
classification (Concatenate) (None, None, 5)	0	activation_24[0][0] activation_24[1][0] activation_24[2][0] activation_24[3][0] activation_24[4][0]
regression (Concatenate) (None, None, 4)	0	reshape[0][0] reshape[1][0] reshape[2][0] reshape[3][0] reshape[4][0]

Total params: 3,877,421
Trainable params: 234,897
Non-trainable params: 3,642,524