Table of contents

[1 Introduction 5](#_Toc70252561)

[1.1 Limitations 5](#_Toc70252562)

[2 Datasets 5](#_Toc70252563)

[3 Model 7](#_Toc70252564)

[4 RDD2020 Training 8](#_Toc70252565)

[4.1 Optimizer 8](#_Toc70252566)

[4.2 Image size 8](#_Toc70252567)

[4.3 Priors settings 9](#_Toc70252568)

[4.4 Data augmentation 10](#_Toc70252569)

[4.5 Weighted loss 10](#_Toc70252570)

[4.6 Results 11](#_Toc70252571)

[5 Transfer learning TDT4265 13](#_Toc70252572)

[5.1 Configuration 13](#_Toc70252573)

[5.2 Results 14](#_Toc70252574)

[6 Further work 15](#_Toc70252575)

[6.1 Cropping 15](#_Toc70252576)

[6.2 Tiling 15](#_Toc70252577)

[6.3 Augmentation 15](#_Toc70252578)

[7 Referances 16](#_Toc70252579)

# Introduction

This project is part of course Computer Vision and Deep Learning at NTNU. The goal of this project is to develop an object detection network that is able to detect and classify different kinds of road damage. Training of the network has been done on two datasets: RDD2020 and TDT4265 dataset that has been created for this project and contains images of Norwegian roads. The network used in this project has been built upon SSD model that has been used in Assignment4 in this course.

## Limitations

The main limitation in development of the model has been access to GPU for training. Therefore, there are still few techniques that should be tested. Some of these has been described in chapter 6. In this project the main focus has been on training with RDD2020 dataset, with the assumption that similar techniques will work on TDT4265 dataset. More work should be done to improve the final results on TDT4265 dataset, but because of time limitations it has not been in focus.

# Datasets

|  |  |
| --- | --- |
| Damage type | Label |
| Longitudinal crack | D00 |
| Lateral crack | D10 |
| Complex crack | D20 |
| Pothole | D40 |

Understanding dataset is an important part of developing a successful deep learning network. Therefore, in this chapter we will look at some characteristics of the RDD2020 and tdt4265 dataset, that are used for training road damage detection network. As mentioned in the introduction the is to find and recognize different road damage types. We are using four different labels to classify the damage type, that can be seen in the table below. As a class we had a task to annotate road damage for Norwegian roads. This gave me quite good understanding of challenges when it comes to finding and classifying cracks and holes. Long shadows and wet spots on the road and snow laying on the side of the road can make it often difficult to see the cracks. Other factors that might make the process challenging are reflections in the windshield and video quality. The reason for that is that the cracks in the road are often very thin and small disturbance make the mall object harder to recognize. I would say that the property of the road crack, it’s thickness and length is a trait that will make the object detection challenging. In addition, classifying the road damage itself can be sometime challenging. Longitudinal crack has some common features with a complex crack and a complex crack might resemble a pothole. Road damage detection is generally much easier with high resolution images, where all details are easily visible. Since all images are taken from a car the environment of each image is bit similar. Some interesting observations are that upper of the image contains almost always buildings and sky. It means that the objects we want to detect will almost never appear in this part of the image. In the graph below we can see distribution of coordinates of bounding boxes in rdd2020 and tdt4265 datasets. We can clearly see that most objects lay in the lower part of the images in the rdd2020 dataset. In the tdt4265 dataset there are no boxes in the lowest part since this part contains the dashboard, and the road is only in the middle stripe of the image. Other interesting aspect of the datasets is number of different types of road damage. Below we can see how many of different classes we can find in each training dataset. In the RDD2020 dataset there are about half as few D10 labels compared to the others. In the TDT4265 dataset we have much more of D00 classes compared to the others. This will clearly be reflected in final mAP where D10 will most likely get lowest accuracy in RDD2020 dataset and D00 will get highest accuracy in TDT4265 dataset. One way of reducing the effects of the imbalance has been discussed in chapter 3.5.

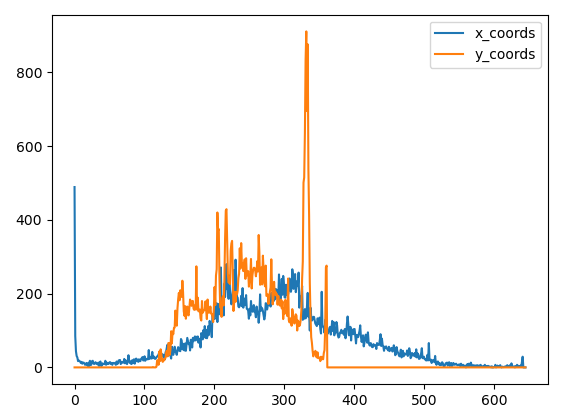


Figure 4: TDT4265 position of the bounding boxes in 640x360 images (x-axis: position, y-axis: #boxes)

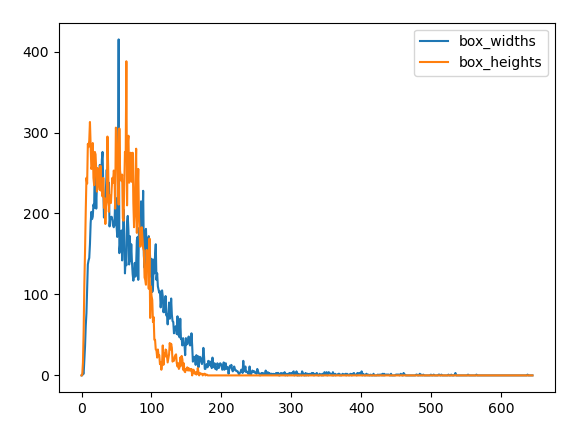


Figure 3:TDT4265 size of bounding boxes in 640x360 images (x-axis: length, y-axis: #boxes)

Figure 2: RDD2020 position of the bounding boxes in 500x500 images (x-axis: position, y-axis: #boxes)

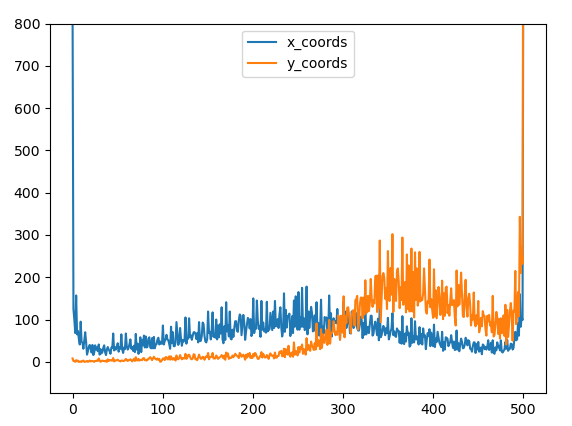


Figure 1: RDD2020 size of bounding boxes in 500x500 images (x-axis: length, y-axis: #boxes)

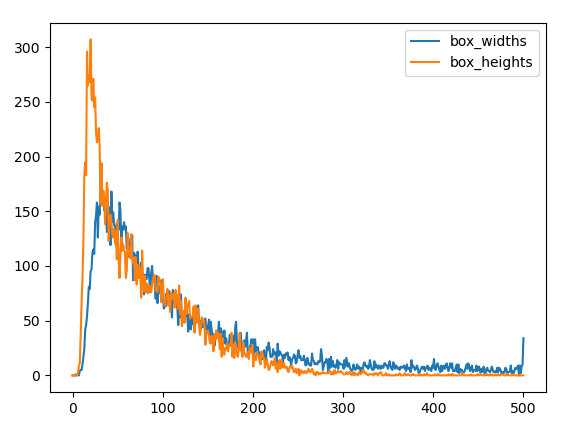




Table 1: Classes in train datasets

# Model

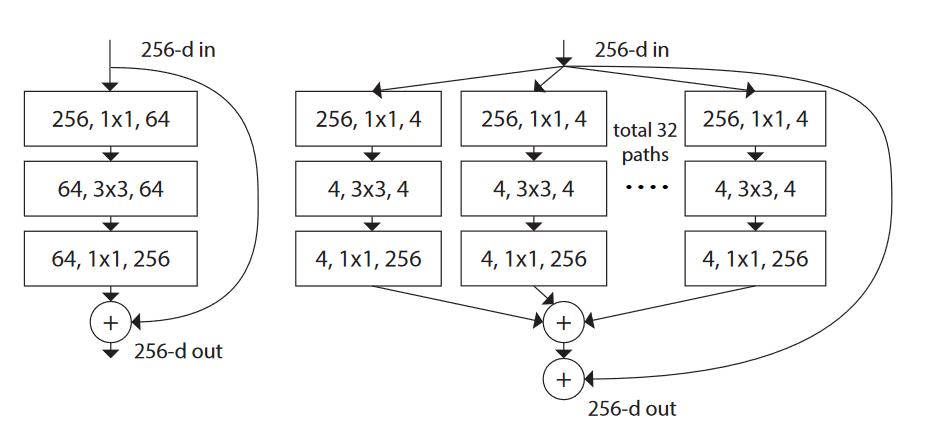


Figure 5: **Left:** A block of ResNet. **Right:** A block of ResNeXt with cardinality=32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels). [1]

To implement object detection network, I am using ssd model from assignment 4 as a starting point. The main change is choice of backbone model. To achieve good result using a pretrained SoA model was preferable. After some research and some quick tests, the choice fell on ResNeXt network. A ResNeXt model, pretrained on ImageNet dataset can be imported from torchvision model library. ResNeXt model in some ways combines the architectures of ResNet [2] and Inception [3] models. ResNeXt model “adopts VGG/ResNets’ strategy of repeating layers, while exploiting the split-transform-merge strategy in an easy, ex-tensible way”.[1] However in contrast to Inception model “the transformations to be aggregated are all of the same topology. This design allows us to extend to any large number of transformations without specialized designs”[1]. ResNeXt network used as backbone in this task has structure very similar to a ResNet model, but the ResNet blocks has been replaced by ResNeXt block. Compression of these two can be seen in Figure 3.

To implement the ResNeXt network as a backbone we need to remove the fully connected network from the imported ResNext model. The backbone model has been summarized in the Table 1. When training the network on we are not freezing any layers. Reason for that is that objects in RDD2020 dateset are quite different than objects in ImageNet dataset. We are however still initializing model with weights from pretrained model since it’s much better starting point than random initialization, these filters are probably closer to desired solution than random ones.

Feature blocks that has been used for detection objects are outputs from block 5, 6 and 7. Sizes of these blocks are show in Table 1. Reason for not using more feature blocks are memory issues that I ran into when trying to include output from block 4.

|  |  |  |
| --- | --- | --- |
| Output size | # block |  |
| 250x250x64 | 0 | Conv2d |
|  | 1 | BatchNorm2d |
|  | 2 | ReLU |
| 125x125x64 | 3 | MaxPool2d |
| 125x125x256 | 4 | Bottleneck0  Bottleneck1  Bottleneck2 |
| 63x63x512 | 5 | Bottleneck0  Bottleneck1  Bottleneck2  Bottleneck3 |
| 32x32x1024 | 6 | Bottleneck0  Bottleneck1  Bottleneck2  Bottleneck3  Bottleneck4  Bottleneck5 |
| 16x16x2048 | 7 | Bottleneck0  Bottleneck1  Bottleneck2 |

Table 2: Backbone structure (sizes specified for input=500x500)

# RDD2020 Training

The model has been first trained on the RDD2020 dataset, the result that gave best performance has then been transferred and used to train the tdt4265 dataset. Most time has definitively been used on training rdd2020 dataset, with hope that similar approaches will give good results on tdt4265 dataset. These two datasets contain very similar objects so this assumption should not be unreasonable.

## Optimizer

Optimizer used to train the networks is Adam. Adam has been chosen in this task since it often gives great results without need for lot of parameter tuning. Since time has been main restriction for me in this project Adam was an OK choice. However it has been shown that for instance SGD algorithm is better at generalization then adaptive optimizers [4]. Therefore, I would prefer to implement a AGD optimizer with learning rate scheduler, which should give better results. Optimizer used in all the training: Adam: lr= 10^-4, amsgrad=True

## Image size

In the beginning the images has been resized to resolution 300x300. However, after closer inspection of the data I have concluded that higher resolution should have great positive impact on the end result. Reason for that is that crack in the row can often be very thin and can be treated similar to small objects. The easiest way to improve performance of ab object detection network on small images is to increase the resolution, such that image contains more details and the objects are easier to recognise. When the image is scale down lot of the details disappear, and recognition of detailed and small objects becomes difficult. The images have been therefore resized to 500x500 from the original 600x600. This improved the end result greatly and it can be seen in chapter 3.6. Disadvantage with this approach is that computation time also increases, and it might not be acceptable if the images need to be analysed live with a given frame rate.

## Priors settings

When the input image has been resized, one should also customize the ssd configuration of priors and bounding boxes to match the new input. This is a detail that I have overlooked at the beginning since I was satisfied with the results. After realising the mistake, I came back at changed the settings such that smaller objects should be detected in the first layers and bigger objects in the later ones. Sizes of anchor boxes was now spanning almost whole image from 60 to 510 pixels and has been calculated using standard formula [5]. This however led to much worse results, than settings from the 300x300 images. What seems to be reason for this behaviour are the actual sizes of the bounding boxes. From the Figure 5 we can see that there are almost no bounding boxes that are bigger than 250 pixels. Therefore, using settings for bounding boxes from 300x300 images gave good results because it matched the actual boxes. Therefore, the configuration has been customized such that bounding boxes will span from 10 to 250 pixels as shown in Figure 5.

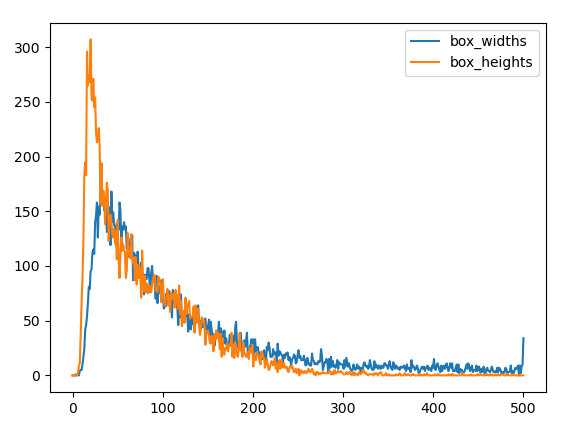


Figure 6: Bounding box sizes and priors max-min configuration

1. MIN\_SIZES: [[10, 10], [80, 80], [150, 150]]
2. MAX\_SIZES: [[80, 80], [150, 150], [250, 250]]

Aspect ratios hasn’t been changed significantly as the configuration matched sizes of objects in the dataset.

1. ASPECT\_RATIOS: [[2,3], [2,3], [2,3,4]]

Result of these changes can be found in chapter 3.6.

## Data augmentation

Data augmentation is a proven method to improve performance of a network [6]. It was a natural way to try to improve performance of the network. Data augmentation technique that I have applied first was random horizontal mirroring of the images. This seemed like a very easy technique that should not worsen the accuracy since it is a natural transformation for images of road. Using horizontal mirror augmentation with Adam optimizer, I got a baseline to improve the model and try other augmentation techniques.

Next augmentation techniques that were tried were different image effects as adjusting sharpness, brightness, contrast etc. What effect should be used was chosen at random, where each effect had same probability to be chosen. The effects were applied such that the image should still look somehow natural after the transformation. Other augmentation technique that has been used was erasing which augmentation technique that has shown improvement in object classification tasks [7]. Here I had three augmentations that were chosen randomly where there was erased random pixels in the image, random areas of the image and random horizontal and vertical stripes. These erasing and effect augmentations were tried in different combinations and on different scales, but none of the results managed to beat the baseline. Reason for such result is how detailed the objects we want to detect are. As mentioned earlier road crack are often very thin. Applying an effect to such object may disturb it so much that it becomes hard to recognize.

After failed data augmentation with random effects and erasing, I tried other augmentation techniques that doesn’t disturb the objects. The augmentations I ended up with was sample cropping and random rotation. These augmentation techniques had positive effect on the networks performance and was kept for all future training.

## Weighted loss

As we have seen in the chapter Dataset there is some imbalance in the number of classes of different road damage types (Tabel 1). Most noticeable is how there is about half as many D10 labels compared to other labels in RDD2020 dataset and much more of D00 class labels in TDT4265 dataset. One solution to the imbalance is use of weighted cross entropy losses.[8] Method that was used for the RDD2020 dataset was inspired by *inverse class frequency*. Each weight is calculated as: , where is a hyperparameter and is precent of samples of the j-th class which has been calculated in Tabel 1. The has been chosen such that most frequent class gets . Use of weighted loss improved slightly performance of the less frequent classes. The result of this change can be seen in chapter 3.6, where we can see improvement in validation of D10 class.

## Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Run name | Augmentation | Weighted loss | Input size | Priors config | Aspect ratios | mAP |
| Org (pink) | Mirror | No | 300x300 | default | default | 0.3219 |
| Aug (black) | Mirror, Effect, Erasing | No | 300x300 | default | default | 0.3018 |
| Loss (purple) | Mirror | Yes | 300x300 | default | default | 0.3242 |
| Bigger (orange) | Mirror | Yes | 500x500 | default | default | 0.3711 |
| bigger\_aug (green) | Mirror, Rotate, Crop | Yes | 500x500 | default | default | 0.3775 |
| bigger\_priors (blue) | Mirror, Rotate, Crop | Yes | 500x500 | custom | custom | 0.3748 |

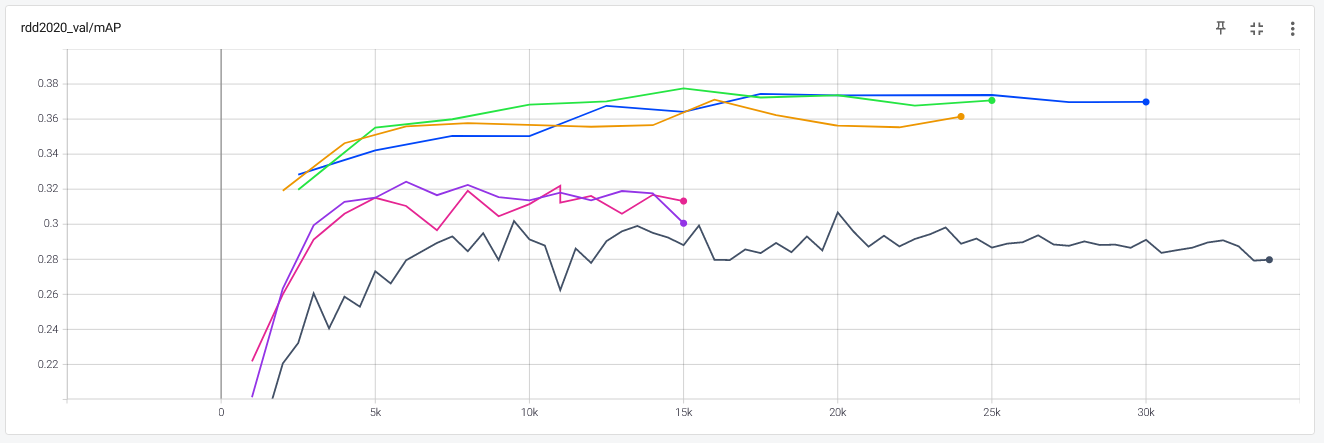
Tabel 3: Results RDD2020 dataset

Figure 7: RDD2020 mAP

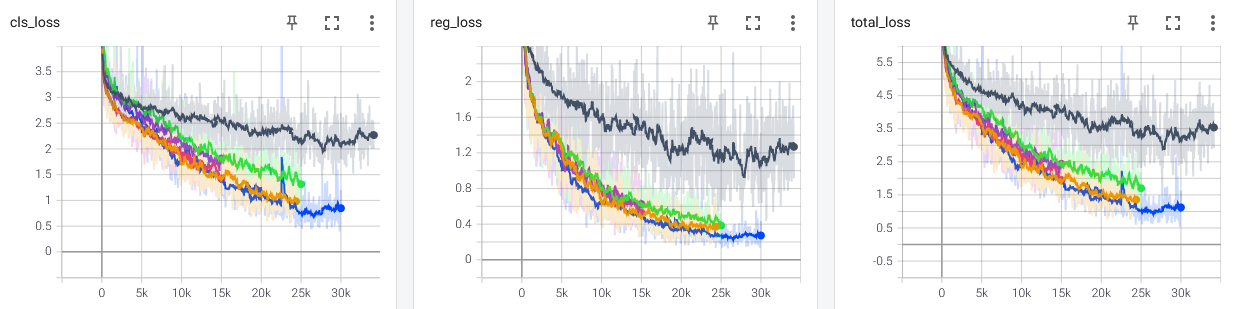


Figure 8: RDD2020 Losses

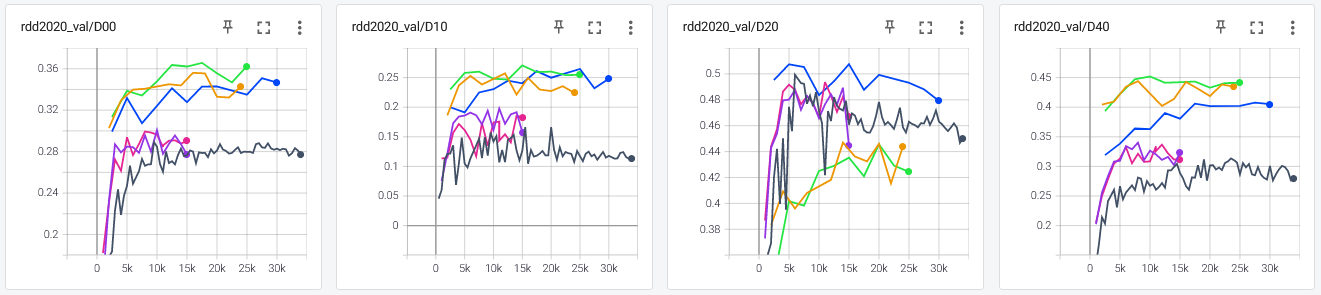


Figure 9: RDD2020 classes accuracy

From the result above we can clearly see that increasing input image size had greates improvement, which has been discribed in more detail in chapter 3.2. Also difference between weighted loss and not weighted can be seen specially D10 class accuracy (pink and purple). Adding of data augmentation has some positive effect on the accuracy. Interesting part is that accuracy of detection of D20 objects gets much lower, when the prior configs are not customized. Reason for that is most probable that D20 objects are bigger and with configuration for small prior sizes, these were not included. This difference has been elimineted after customization of prior configuration. All the different parameter used for the training are specified in the table below.

|  |  |  |
| --- | --- | --- |
| Parameter |  | Comment |
| OUT\_CHANNELS | [512, 1024, 2048] |  |
| FEATURE\_MAPS | [[63, 63], [32, 32], [16, 16]] | Input: 500x500 |
| MIN\_SIZES | [[10, 10], [80, 80], [150, 150]] | Input: 500x500 |
| MAX\_SIZES | [[80, 80], [150, 150], [250, 250]] | Input: 500x500 |
| FEATURE\_MAPS | [[38, 38], [19, 19], [10, 10]] | Input: 300x300 |
| ASPECT\_RATIOS | [[2,3], [2,3], [2,3,4]] |  |
| BOXES\_PER\_LOCATION | [6, 6, 8] |  |
| IMAGE\_SIZE | [500, 500] or [300, 300] |  |
| PIXEL\_STD | [0.229, 0.224, 0.225] |  |
| PIXEL\_MEAN | [0.485, 0.456, 0.406] |  |
| BATCH\_SIZE | 10 |  |
| LR | 1e-4 |  |
| Weighted loss | [1,k/0.27,k/0.15,k/0.32,k/0.27]; k=0.32 |  |

Note: Default parameters are not included in the tabel.

# Transfer learning TDT4265

Results from training on RDD2020 dataset that has been described in previous parts of this document has then been used to train the model on TDT4265 dataset. Result that I decided to use as starting point are labelled as “bigger\_priors” in the Results chapter. This model achieved second highest mAP. Reason for not choosing model with highest mAP is that “bigger\_aug “ model had greater inbalnce in accuracy of the classes. Specially the D20 class had 10 precent lower accuracy than on that has been achieved with “bigger\_priors” model. Therefore weigts that has been used as initializtion for training on TDT4265 were loaded from checkpoint of “bigger\_priors” on iteration 17500.

## Configuration

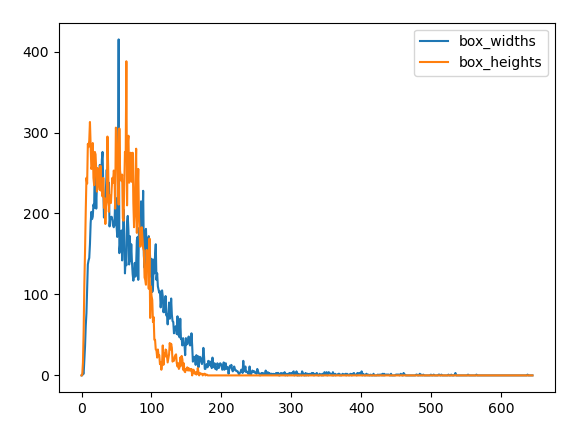


Figure 10: TDT4265 Bounding box sizes and priors max-min configuration

Techniques and setting that has been used to train on TDT4265 dataset are almost the same as ones used for training of “bigger\_priors” model. Images in the TDT4265 dataset have size of 1920x1080. For the training the images has been resized to 640x360, which kept the origanl ratio of 16:9. Sizes of feature maps has been resized accordingly. Configuration sizes of bounding boxes has changed such that the mach the sizes of actual bounding boxes and span from 10 to 220 pixels, using same strategy as described in chapter 3.3. This configuration has been visualized in Figure 8. Since the claas label inbalances are greater in this dataset comapred to RDD2020 dataset, the loss weighting has also been changed so it match current dataset better.

|  |  |  |
| --- | --- | --- |
| Parameter |  | Comment |
| OUT\_CHANNELS | [512, 1024, 2048] |  |
| FEATURE\_MAPS | [[80, 45], [40, 23], [20, 12]] |  |
| MIN\_SIZES | [[10, 10], [60, 60], [120, 120]] |  |
| MAX\_SIZES | [[60, 60], [120, 120], [220, 220]] |  |
| ASPECT\_RATIOS | [[2,3], [2,3], [2,3,4]] |  |
| BOXES\_PER\_LOCATION | [6, 6, 8] |  |
| IMAGE\_SIZE | [640, 360] |  |
| PIXEL\_STD | [0.229, 0.224, 0.225] |  |
| PIXEL\_MEAN | [0.485, 0.456, 0.406] |  |
| BATCH\_SIZE | 10 |  |
| LR | 1e-4 |  |
| Weighted loss | [1, 1, 8, 8, 9] |  |

## Results

1. mAP: 0.1091
2. D00 - Linear Longitudinal Crack: 0.3140
3. D10 - Linear Lateral Crack: 0.0095
4. D20 - Alligator and Other Complex Cracks: 0.0896
5. D40 - Pothole: 0.0233

Main focus in this project has been on training on RDD2020 dataset with the assumption that same techniques will improve performance of TDT4265 dataset.

# Further work

## Cropping

In chapter 2, it was discussed how road damage occurs only in some parts of the image. Therefore, cropping the image such that the part that that will anyway not contain any objects, might be a valid strategy. This way will not waste computation on searching for road damage in the sky and could use higher resolution to represent road damage without sacrificing too much computational power

## Tiling

One technique that has been proposed to improve detection of small objects is use of tiling. Image is split into smaller overlapping tiles. Object detection is performed on these, and the results are merged to achieve full result. [9] It is unsure if this strategy would work on road damage detection, as the size of bounding boxes is not always small even that the road crack itself can be very thin/small. However customizing this technique for road damage detection might show some positive results.

## Augmentation

We have seen that adding augmentation that doesn’t disturb image itself, had positive effect on the model’s performance. In this project only horizontal mirroring, sample cropping and random rotation has been used. Augmentation methods as translation and padding could be tried. Random erasing might also show some positive effects if implemented correctly.

## Backbone model

ResNeXt model used in this project reduces size of the feature maps very quickly. Input to first ResNeXt block has size of 125x125, when input image was 500x500. We have seen that use of higher resolution had big influence on model’s performance. Therefore editing the network such that layers are reduced slower or less might be a valid strategy. Also use of dilation in convolution layers had shown improved results on object detection and is used in network like DetNet. [10] Reason for not editing the backbone network in this project, is that pretrained models tend to perform better. If the backbone was edited, we would preferably pretrain it on ImageNet, which was not allowed in this project. Therefore only pretrained models from torchvision were tested and used.

* Other loss weight starategies
* Testing on TDT4265

# Referances

1. Xie, S., et al., *Aggregated Residual Transformations for Deep Neural Networks.* arXiv pre-print server, 2017.

2. He, K., et al., *Deep Residual Learning for Image Recognition.* arXiv pre-print server, 2015.

3. Szegedy, C., et al., *Rethinking the Inception Architecture for Computer Vision.* arXiv pre-print server, 2015.

4. Ashia, et al., *The Marginal Value of Adaptive Gradient Methods in Machine Learning.* arXiv pre-print server, 2018.

5. Muthuraja, M., et al., *Black-Box Optimization of Object Detector Scales.* arXiv pre-print server, 2020.

6. Zoph, B., et al., *Learning Data Augmentation Strategies for Object Detection*. 2020, Springer International Publishing. p. 566-583.

7. Zhong, Z., et al., *Random Erasing Data Augmentation.* Proceedings of the AAAI Conference on Artificial Intelligence, 2020. **34**(07): p. 13001-13008.

8. Trong and K. Yamamoto, *Resolving Class Imbalance in Object Detection with Weighted Cross Entropy Losses.* arXiv pre-print server, 2020.

9. Ozge Unel, F.a.O., Burak O. and Cigla, Cevahir, *The Power of Tiling for Small Object Detection.* 2019.

10. Li, Z., et al., *DetNet: A Backbone network for Object Detection.* arXiv pre-print server, 2018.