NTNU | Norwegian University of Science and Technology

Department of Engineering Cybernetics

TDT4265 - Computer Vision and Deep Learning

Final Project

Detecting traffic objects from LIDAR images

Andreas Øie

May 6th, 2022

Contents

1 Introduction

In this project, our goal is to detect objects in traffic from LIDAR images collected around the campus (TDT4265 dataset). The project is separated into 4 separate parts, following a typical computer vision development cycle: dataset exploration and annotation (part 1), model development (part 2), model evaluation (part 3), and ”going beyond” (part 4).

1.1 Disclaimer

- The reported inference speeds (images/sec) may vary depending on background process and should be taken with a grain of salt.
- The reported load time (images/sec) may vary depending on what kind of background processes ran on my PC and should be taken with a grain of salt.
- Training time/speed is reported in Tensorboard’s relative time-scale.
- Accuracies in Tables are reported by Tensorboard’s smoothing value set to 0.6 (not biggest peak nor final reported value).
- All configuration files are placed in ../core/configs/
- Most figures can be found in the notebook: data_analysis.ipynb
- All config files are using the default configuration from configs/template/ssd300.py (output channels, learning rate, optimizer, batch size, epochs etc), except for when its specifically mentioned that I’ll change it.

2 Part 2 - Model Creation

This section provides details about the baseline model. The following sections concentrate on building the model and evaluating its performance. Furthermore, we examine how the *RetinaNet* model exceeds baseline performance with new architectual structure. In addtion, we show how previous established data explorations and findings can help building a more efficient neural network.

2.1 Creating a baseline

In this project, we're using a customized version of the original SSD300 as our basic model. The backbone we're utilizing is the key difference between this model and the one described in the paper [Liu et al. 2016]. The VGG backbone has been replaced by a new customized convolutional network architecture show in Table 1. In order to support the new image dimensions we change the anchor boxes to support the new dataset resolution going from 300×300 to 128×1024.

Output	Layer	Filters	Kernel	Stride	Padding
	Conv2D	32	3×3	1	1
	ReLU	–	–	–	–
	MaxPool2D	–	3×3	2	0
	Conv2D	64	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	64	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	128	3×3	2	1
32×256	ReLU	–	–	–	–
	Conv2D	128	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	256	3×3	2	1
16×128	ReLU	–	–	–	–
	Conv2D	256	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	128	3×3	2	1
8×64	ReLU	–	–	–	–
	Conv2D	128	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	128	3×3	2	1
4×32	ReLU	–	–	–	–
	Conv2D	128	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	64	3×3	2	1
2×16	ReLU	–	–	–	–
	Conv2D	128	3×3	1	1
	ReLU	–	–	–	–
	Conv2D	64	3×3	2	0
1×8	ReLU	–	–	–	–
shape	*type*	*quantity*	*shape*	*quantity*	*quantity*

Table 1: Showing the baseline convolutional neural network architecture

The model architecture consist of an initial layer sequence of [INPUT − CONV − RELU − POOL], while remaining subsequent layers consist of stacking [CONV − RELU − CONV − RELU] sequences. In total, we have six different primary layers which were we at each layer, extract features as outputs

volumes at various dimensions as seen in at the output-column in Table 1.

Using the default configuration provided, we establish a baseline evaluation. The idea behind evaluating a baseline model is to set a performance lower-bound for the metric evaluation. That is, we create a basic standard model with minimal modifications we can compare and validate new models against.

Training with the default configuration, core/configs/task21/tdt4265.py, with default parameters yields the following results in Table 2.

mAP	*mAP*	*AP*	*AP*	*images/s*	*relative*	*images/s*	-	-
@0.5	**@0.5:0.95**	**@person**	**@ car**	**Load time**	**Train time**	**Inference**	**Parameters**	**Filename**
0.1251	0.04672	6.97e-4	0.1854	99.0	16m 45s	11.8	2.8M	tdt4265.py

Table 2: Low bound performance metrics by the baseline model.

The model is running inferences at a decent speed compared to its number of parameters with close 12 images pr. second. Given the simplicity of this model, the total number of parameters is only at 2.8 million. Running the model for 50 epochs at a batch size of 32 only takes close to 17 minutes. On the basis of the achieved evaluation metrics, it seems fair to suggest that the model achieves high inference speed, but at the lack of accuracy. The general mAP stabilizes around 0.046 while for threshold @0.5, we see better performance showing a accuracy of close to 0.13. The combinations of the shallow network size and the lack of image, might be the cause of the poor accuracy of AP@person as seen in Figure 1. Looking at Figure 2, we can see the baseline model is slightly better at predicting the location of the bounding boxes then it is at deciding what category it is, for regression and classification respectively.

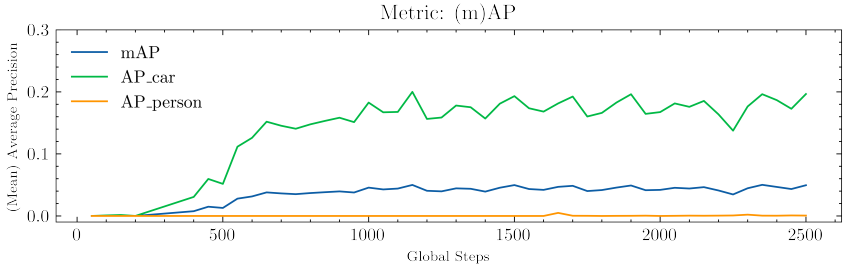


Figure 1: Comparing different AP metrics

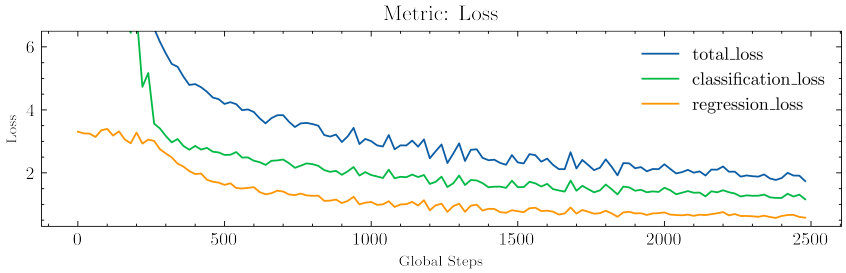


Figure 2: Comparing different loss metrics

2.2 Augmenting the Data

Following details involves the experimentation's of data augmentations. The base configurations is based upon the configuration used when training the baseline model in the previous section.

The baseline configuration uses the template data augmentations, which are composed on the CPU transforms for training: ToTensor, Resize and GroundTruthBoxesToAnchors. Similarily for validation we use ToTensor and Resize. As for GPU transforms we use the standard Normalize transforms provided in the starting template code. These augmentations act as the baseline configuration for further development. New data augmentations configurations uses these by default. To ensure logical and helpful augmentations, I've continously visualized ech addition to ensure it didn't ruin the general sense of the image.
We attempt to validate the performance of the baseline model by using the following data augmentations on the training data for CPU and GPU- operations.

Selected data augmentation techniques:

- **I1**, denotes RandomHorizontalFlip (see core/transforms/transform.py)
- **I2**, denotes RandomSampleCrop (see core/transforms/transform.py)
- **I3**, denotes ColorJitter (see core/transforms/gpu_transforms.py)
- **I4**, denotes RandomAdjustSharpness (see core/transforms/gpu_transforms.py)

As noted in the project details, I've included the CPU specific RandomHorizontalFlip and RandomSampleCrop transformations. In addition, we include two GPU parallelizable augmentations, ColorJitter and RandomAdjustSharpness. To ensure logical and helpful augmentations, we now have 5 CPU transforms and 2 GPU transforms. We often rely on using few to none data augmentations early on in modelling development, so this leads to the more safe choices of augmentation. RandomHorizontalFlip and RandomSampleCrop helps generating extra images by logically sound ideas: we flip it left-to-right or we choice small portions of the images. These are selected because they

make sense through visualization and they don't really induce additional trouble. To continue to keep it simple, we add two pixel changing augmentation by using the parallelizable ColorJitter and RandomAdjustSharpness. Through augmentation inspections and visualization, we find just appropriate values for these two to not really confuscate the model training too much. We add the ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1) with hue=0 as this can cause unexpected issues. In addition, we add a slight sharpness of 1.5x for the RandomAdjustSharpness operation.

Legend	Load time (images/s)	Train time (relative)	Inference (images/s)	Config
baseline	100.608	16m 4s	12.46	baseline.py
baseline+I1	101.036	15m 33s	12.36	baseline2.py
baseline+I2	120.905	13m 53s	12.35	baseline3.py
baseline+I3	94.485	16m 5s	12.36	baseline4.py
baseline+I4	101.587	15m 28s	12.36	baseline5.py
baseline+I1+I2	118.179	33m 34s	12.38	baseline23.py
baseline+I1+I3	100.168	48m 42s	12.33	baseline24.py
baseline+I1+I4	96.28	15m 59s	12.34	baseline25.py
baseline+I2+I3	118.635	1h 2m 26s	12.35	baseline34.py
baseline+I2+I4	122.57	13m 58s	12.35	baseline35.py
baseline+I3+I4	102.305	15m 36s	12.35	baseline45.py
baseline+I1+I2+I3+4	120.532	14m 24s	12.32	baseline50X.py

Table 3: Evaluation of various data augmentations.

Figure 3 shows the general mAP-metrics for the baseline data augmentation with single transform additions. As itemized above in 2.2, we use the baseline configuration and test how each $I1$, $I2$, $I3$ and $I4$ affect the performance metrics. We refer to the experimentation's listed in Table 3 above, where we can see further detail specifics. What we can see from testing single augmentation addition is that using certain combinations could greatly increase the training time. For instance: baseline+I2+I3 uses the longest training time of 1h 2m 26s.

Following figures shows the general metric: $metrics/mAP$, which indicates the overall effect the transformations have. During inspection, I've also monitored the other mAP-metrics at scales of mAP@0.5, mAP@0.75, $mAP@small$ and so on. Instead of displaying figures of all kinds of metrics, we select only to display the general mAP as the other metric results show similar trends.

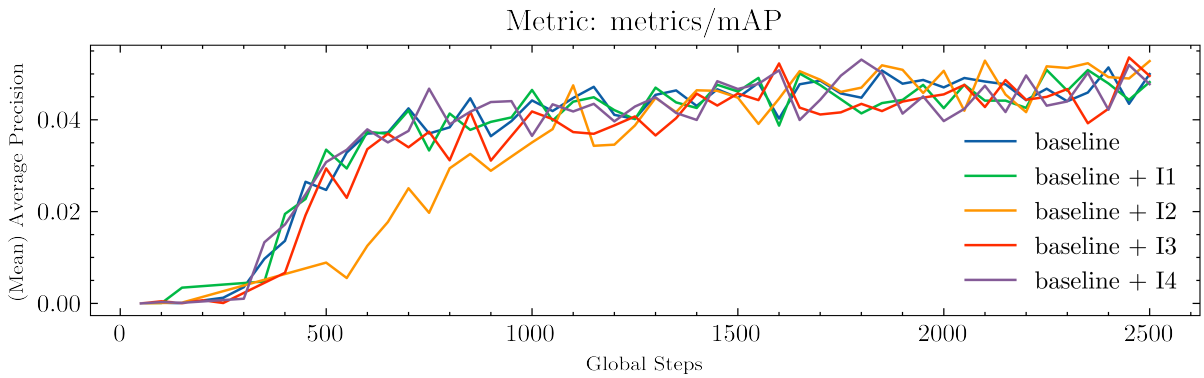


Figure 3: Introducing data augmentation 0

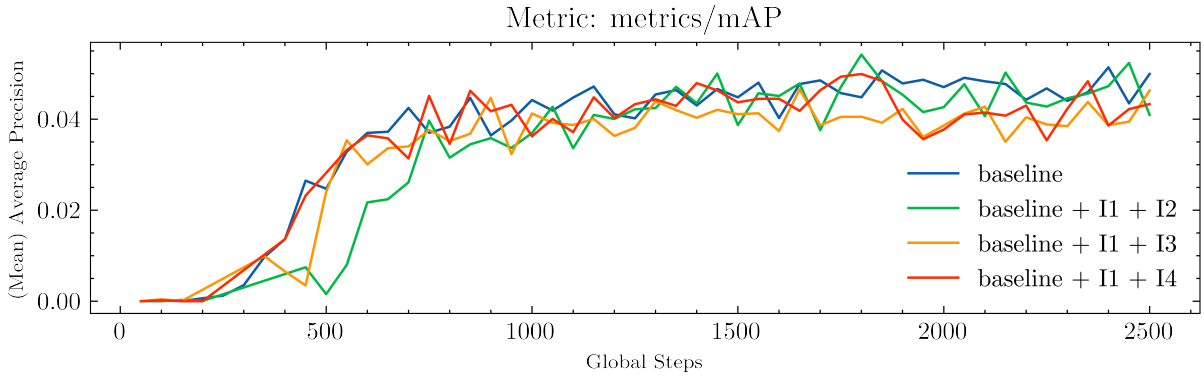


Figure 4: Introducing data augmentation 1

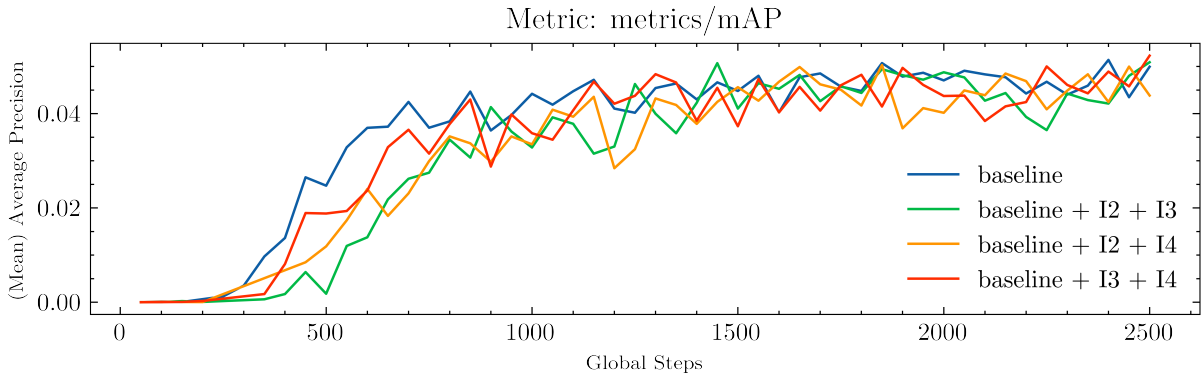


Figure 5: Introducing data augmentation 2

Finally, we begin to show that training with all transformations activate, actually reduces accuracy and results in more unstable training with progression spikes. However, certain attempts at training longer gives slight hope as they begin to stabilize. In addition, its worth mentioning that subsequent tasks involves creating bigger models, where data augmentations really come into play. By continuing onto the next task, then coming back to this task - I was able to confirm this. The selected data augmentations $I1$, $I2$, $I3$ and $I4$ helped when the total number of parameters increased.

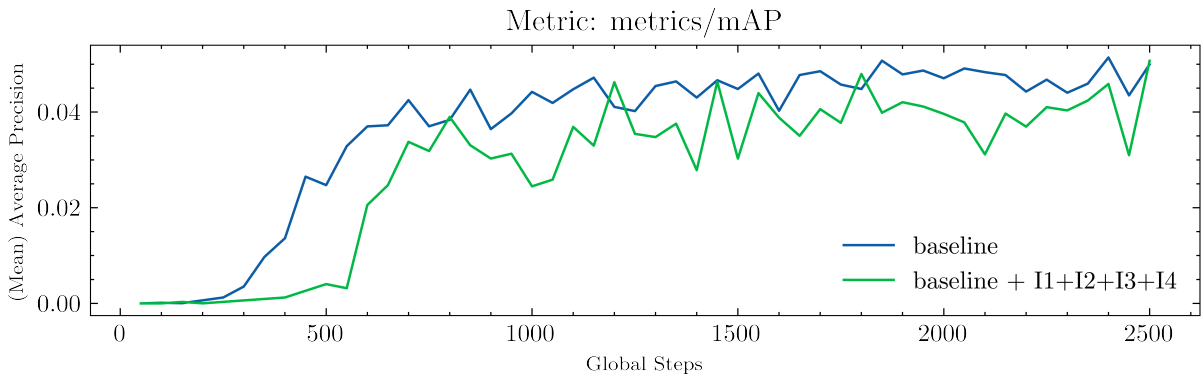


Figure 6: Introducing data augmentation 3

With this in mind, it seems fair to suggest that we're fine on-wards by using as the selected data augmentations, and not continue to add more and more. For reference, RetinaNet Lin et al. 2017 only uses horizontal image flipping.

2.3 Implementing RetinaNet

The following sections shows the progression of reproducing the RetinaNet Lin et al. 2017. We iterate over the baseline model , and convert it into a RetinaNet by implementing the following changes:

1. Replacing the backbone with a feature pyramid network (implemented in core/backbones/resnetfpn.py)

2. Replace hard-negative mining in the SSD loss with Focal loss (implemented in core/ssd/focal_loss.py)

3. Replace the single-layer regression and classification output heads with deeper convolutional nets (implemented in core/ssd/retinanet.py)

4. Initialize the bias of the last convolutional layer of the classification head to account for background class-imbalance (implemented in core/ssd/retinanet.py)

We iteratively create a new configuration per additions, where we maintain all parameters the same while training for a total of 100 epochs. This means we continue to use the added data augmentations listed in 2.2. The only thing changing from the previous baseline is the change of aspect ratios for the anchors - where the two last aspect ratios in the anchors parameters have been changed from [..., [2], [2]] to [..., [2, 3], [2, 3]], prior to the implementation stages of the RetinaNet. This is done prior to solving the next tasks since creating deeper heads later on, we're required to use the same amount of aspect ratios per feature map. By using the same aspect ratios for all implementation stages, we

ensure more fair comparisons. Following additions can be found in configs: task23/retina_P1.py, task23/retina_P2.py, task23/retina_P3.py and task23/retina_P4.py.

We structure the below sub-subsections with a summarizing sub-subsection at the end. This means in the following sub-subsections onward, we automatically refer to Table 4 and Figures (8, 7) in the end summary 2.3.6, as this keeps information tidy and natural for the description of the coming sections.

2.3.1 Exp 1: replacing the backbone with a feature pyramid network

We change the backbone of the SSD model, previously defined as the baseline convolutional neural network architecture in Table 1, with an more advanced architecture. We build the FPN Lin et al. 2017 by combining it with the initial layers of a pre-trained ResNet 34 model He et al. 2015 (see core/backbones/resnet.py for details).

With the new added backbone of the SSD model, we effectively increase teh model size by 2.3x. As seen in Table 4, we go from 2.8M to 6.5M parameters. The increase of model size gives it more memory capacity, but causes slower inference speed and increased relative training time. This is however at the gain of increased accuracy in all metrics across the board. We especially focus on the main mAP@0.5:0.95, which shows an increase of 40%. In terms of AP@person, we see an immense improvement of ~ 1800% ("unfair comparison").

2.3.2 Exp 2: replacing hard-negative mining in the SSD loss with Focal loss

Next up, we change the way the classification loss is calculated. By implementing Focal Loss, we seek to solve the issue with class imbalance in the dataset.

The results from this implementation shows itself significant in terms of how the loss progresses. Looking at Figure 8, the classification loss is quite a lot smaller then the previous method was. This in total helps the model to converge quicker. Comparing loss from Exp1 and Exp 2, we see that not only is it lower but also quite a lot more stable in terms of spikes in the loss progression. As regards to the gained performance in metrics, we see an improvement slight improvement across the board for Exp 2 in Table 4. The main mAP@0.5:0.95 shows an added increase of ~ 26%. As this change doesn't effect the model size specifically or forward architecture, we don't except to see any improvements in terms of inference speed either. We also note an big improvement in terms of model capabilities to detect persons, with an AP@person increase of ~ 320%.

2.3.3 Exp 3:: replacing the single-layer regression and classification output heads with deeper convolutional nets

The addition of deeper convolutional nets significantly increase the number of parameters from 6.5M to 11.M. The deeper heads consist of stacking [CONV-RELU] operations as per required in the paper. We add the un-mentioned detail regarding output channels, were we output the recommended 256 channels from the ResNet into the FPN as required. On top of that we create the deep conv. heads - matching the FPN channels, which in total increments the total number of parameters significantly which allows more information to be learned and retained between passes. Table 4 shows the difference by increasing the model capacity. Initially, we don't see big improvements as the mAP@0.5 and AP@car barely increased, AP@person decreased and mAP@0.5:0.95 showing little improvements. As Figure 7 shows, the deviations between Exp 2 and Exp 3 are very small. However, as Exp 3 is consistently above we do effectively consider this as an improvement. By creating a bigger model at the hope of increasing accuracy, we get the opposing trade-off with speed and training time. Table 4 shows the decrease from inference speed from 5.36 to 1.97 frames per second with the almost doubling of relative training time.

2.3.4 Exp 4: initializing the bias of the last convolutional layer of the classification head to account for background class imbalance

Finally, we initialize the output (final layer) heads of the previous added deeper conv. heads. This aims to help initial learning progressions as we set the biases to zero for all classes except the background class which we set as Equation 1 with constants as noted in the project assignment.

$$b = \log\left(p \cdot \frac{K-1}{1-p}\right) \tag{1}$$

Comparing the bias addition from Exp 3 to Exp 4, we can see that the overall metrics seems to improve across the board. Comparing the loss progressions in Figure 8 show little deviations, however looking at Figure 7, we do note a slight improvement early on which essentially increases the mean average precision but at the cost of slightly more spikes in the training stability. Specifically looking at mAP@0.5:0.95, we note an increase of ~ 12%.

2.3.5 *Exp 4: Alternative alpha-strategy for Focal Loss (extra)

Out of general interest, I've decided to test a weighted alpha strategy using the Focal loss. For comparison, I reuse the same configuration from Exp 4, but change the default alphas from [0.01, 1, 1, 1, 1, 1, 1, 1, 1] to the new strategy [0.01, 10, 1, 1, 1, 2, 1, 10, 2]. This new alpha strategy adds an addi-

tional emphasis on car and person followed by smaller additions to rider and bicycle. As previously, we reuse the reduction weight for the background class. This new alpha strategy, show in Figure 7, results in general improvements across the metric board. As seen in Table 4, we see the added boost for AP@person resulting in a 73% improvement in accuracy. As mentioned in section 1.1 regards to disclaimers, incorrect measurements and the use of the reported smoothed values shows the AP@car and mAP@0.5:0.95 have decreased. However, looking at Figure 7, we can clearly see the added benefit of using the new alpha strategy where we increase the importance for certain categories.

2.3.6 Metrics and Results

-	*mAP*	*mAP*	*AP*	*AP*	*images/s*	*relative*	*images/s*	-	-
Legend	**@0.5**	**@0.5:0.95**	**@person**	**@ car**	**Load time**	**Train time**	**Inference**	**Parameters**	**Filename**
*Baseline	0.1102	0.04357	5.95e-4	0.1729	119.3	28m 44s	12.18	2.8M	baseline_aug1234.py
Exp 1	0.1426	0.06100	0.01127	0.2222	113.5	32m 30s	5.38	6.5M	retina_P1.py
Exp 2	0.1956	0.07659	0.04713	0.2395	116.4	44m 53s	5.36	6.5M	retina_P2.py
Exp 3	0.1922	0.07819	0.03636	0.2481	114.7	1h 28m 38s	1.97	11.2M	retina_P3.py
Exp 4	0.2058	0.08738	0.04387	0.2633	110.5	1h 29m 40s	1.94	11.2M	retina_P4.py
*Exp 4	0.2245	0.08551	0.07588	0.2423	114.3	1h 38m 48s	1.89	11.2M	retina_P5.py

Table 4: Implementing RetinaNet: step by step progression metrics

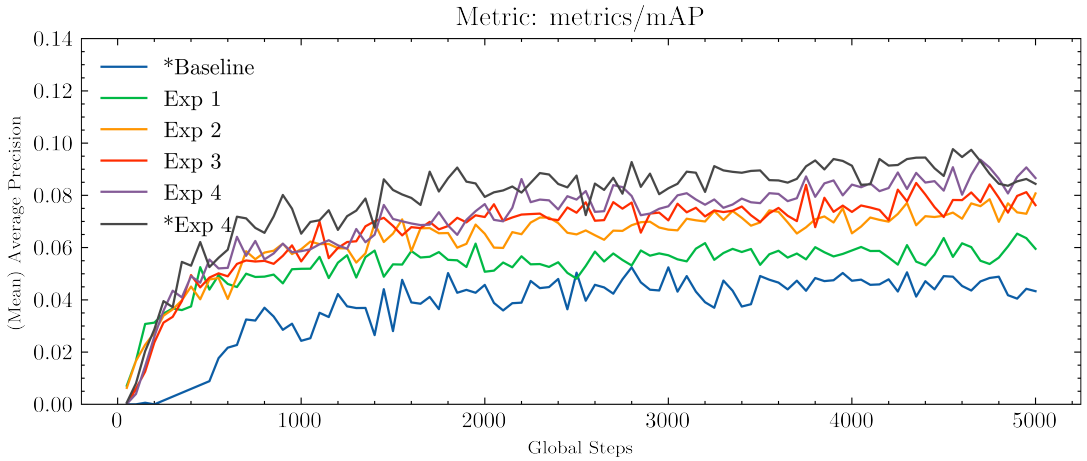


Figure 7: RetinaNet AP improvements. *Baseline includes data augmentations I1, I2, I3, and I4 to ensure fair comparisons *Exp 4 is the same as Exp 4, but with improved alpha

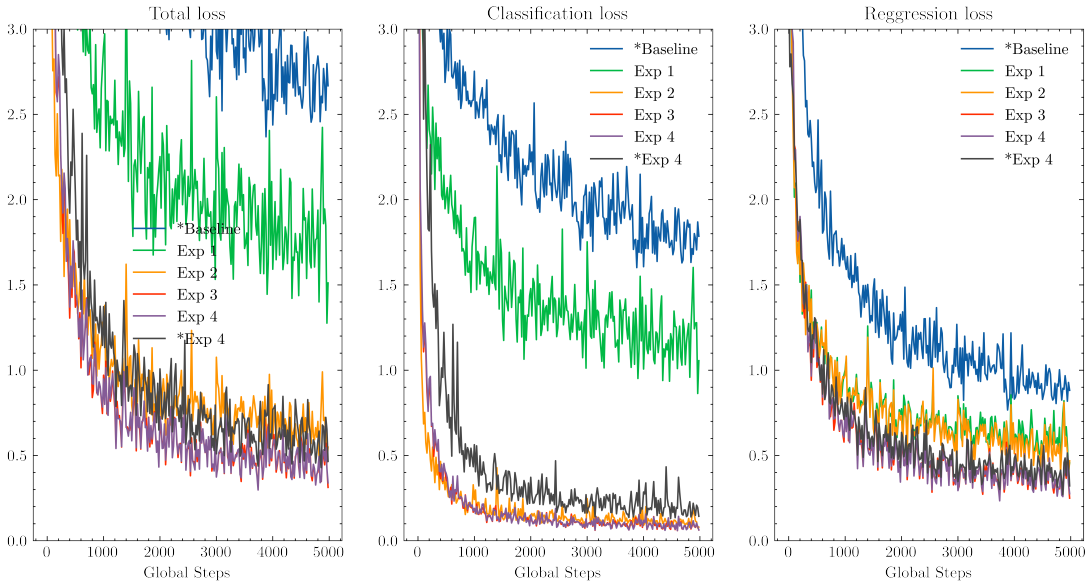


Figure 8: RetinaNet loss improvements. *Baseline includes data augmentations I1, I2, I3, and I4 to ensure fair comparisons.

2.4 Using the Exploration Knowledge

2.4.1 Disclaimer

First of, its with great sadness to view the total amount of incorrect bounding boxes in the provided dataset. By thoroughly inspecting all the images in the training set and validation set, I discovered numerous sources of error which will greatly affect the models upper performance limit and is generally a source of errors. Not only are the mistakes in the training data, but also in the actual validation data. As this is something I can't really do anything about, I will not emphasis it too much - but I strongly suspect my model is affected by them. Please see Appendix C for incorrect training annotations and Appendix D for incorrect validation annotations with line-by-line details and a few random example images to showcase the severity.

2.4.2 Visual observations

We note that we have a collection of various categories, where some are more represented then others. We note cars and person are most seen in images followed by bicycles, scooter and so on. The general size of the bounding boxes for the categories are pretty much consistent. Persons have very vertically shaped rectangles, cars have sort of quadratic (some wide) shape boxes. By skimming through the images, it is clear that these represent the most of the data - so decisions should be followed to enhance and support the detection of these objects to increase the general mAP.

2.4.3 Finding statistical observations

Following exploration analysis and results can be found can be found in notebooks data_analysis.ipynb and data_exploration.ipynb. As I've seen visually, the training data is mainly consisting of cars and persons as can be confirmed from the table below. I've also summarized the general average statistical properties for the bounding boxes. Looking at the width and height properties, we can see that most of the training data (75%) are approximatly around (W, H) = (36, 22).

Category	Total	in %
car	9563	52.3
person	4910	28.9
rider	1588	8.7
bicycle	1043	5.7
scooter	615	3.4
bus	445	2.4
truck	123	0.7

–	Area	X_min	Y_min	X_max	Y_max	Width	Height
count	18287	18287	18287	18287	18287	18287	18287
mean	682.5	411.2	72.3	431.4	99.0	19.6	26.5
std	1164.6	295.3	10.4	296.3	19.0	22.3	15.4
min	19.4	0.0	42.5	9.2	65.1	2.4	6.2
25%	121.1	118.7	66.1	143.2	84.1	6.9	13.9
50%	261.2	457.6	72.1	475.3	94.0	12.7	22.3
75%	675.3	556.5	78.0	577.3	116.4	22.0	36.3
max	7334.7	1006.8	102.7	1024.0	128.0	136.5	78.1

Table 6: Showing anchor distributions with general descriptive statistics of the training set

However, by looking at individual categories in Table 7 below, we can see that by looking at class specifics boxes, we get a slightly different representation. As seen visually above by inspection, we can from the table confirm that we have vertical skewed aspect ratios for persons, scooters etc. and more square and rectangular shapes for vehicles.

Type	Car	Person	Scooter	Bicycle	Rider	Bus	Truck
area	731.94	191.12	278.52	282.77	242.35	1067.2	1842.69
aspect ratio	14:13	7:26	9:29	13:20	2:7	33:31	41:44
width	28.13	7.15	9.4	13.7	8.45	33.89	41.53
height	26.02	26.73	29.63	20.64	28.68	31.49	44.37

Table 7: Statistics of average bounding boxes. Most are reported by the scaled pixel value (multiplied with the image size 128x1024.

Using the results from the visual inspection and the confirmed statistical properties we can designed the anchor boxes accordingly and improve the models performance. Moreover, we present the following suggestion for the anchors: shallow feature maps should focus on detecting the smaller objects (persons, scooters, bicycles and riders) whilst the deep feature maps should focus on detecting the bigger objects (cars, busses and trucks). Below Figure 9 shows how the average distribution of the various aspect ratios are represented training data. The ratios are quite positively skewed with a

mean ratio of approximately around 0.75, however looking at category-individual anchor-ratio distributions (see data_exploration.ipynb) it shows that certain categories are quite weird, possibly polluting the true distribution. For instance: trucks (partially buses also) have a very convex distribution, representing a lot of tall or wide anchor boxes, but very few in-between.

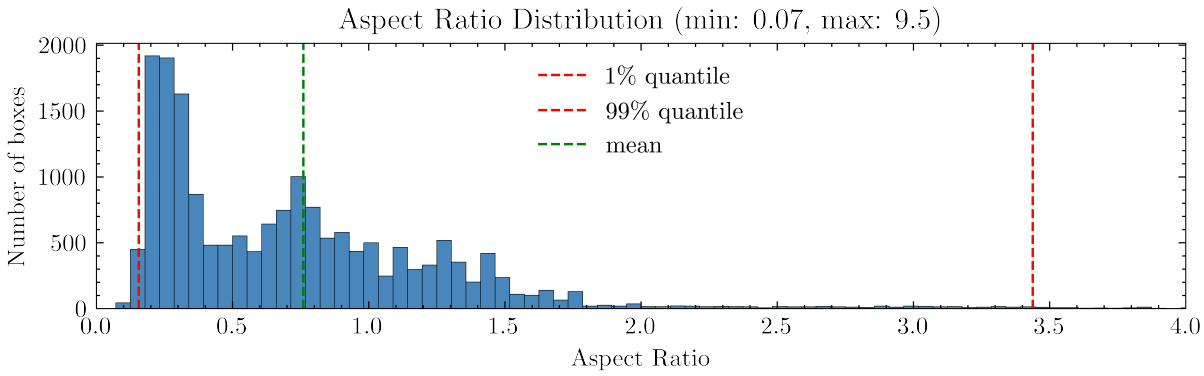


Figure 9: Aspect-Ratio distribution with outliers.

Continuing, I've also used teh K-Means clustering algorithm (see bottom of data_exploration.ipynb) for calculating aspect-ratio cluster per category, to discover appropriate aspect-ratios of the bounding boxes. Figure 8 below show the distribution of ratios calculated for each category.

Category	AR 1	AR 2	AR 3	AR 4	AR 5	AR 6
person	0.16	0.20	0.25	0.29	0.34	0.41
scooter	0.15	0.20	0.26	0.32	0.37	0.46
rider	0.15	0.20	0.25	0.30	0.37	0.53
bicycle	0.30	0.41	0.55	0.73	0.92	1.33
car	0.41	0.69	0.96	1.39	2.58	4.93
truck	0.46	0.57	0.69	1.08	1.30	1.75
bus	0.54	0.63	0.71	0.87	1.14	1.73

Table 8: Finding Aspect-Ratio clusters with K-Means for each category. We see naturally taller aspect-ratios for the smaller object such as persons, and more wider aspect-ratios for trucks and buses.

2.4.4 Implementing changes

Initially, I started to use Exp 1 (see core/configs/task23/retina_P1.py) as the baseline for testing our anchor configurations. However, after numerous attempts with poor improvements across the board using various variations of the calculated K-means ratios as well as testing out various hand-tailored aspect-ratio and box min_sizes, I finally concluded that it was not possible (or at least too tiresome) to improve with the current model.

It was not until I upgraded the model sizes to my biggest model, Exp 4 (retina_P4.py), that the im-

provements started too become evident. This could either be the effect of (a) increasing the general number of parameters from 6.5M to 11.2M or (b) the effect of using the focal loss which prevents the vast number of easy negatives from overwhelming the detector during training.

With this in mind, we instead of showcasing all previous failed training runs (see folder core/configs/task24/...), emphasize the final successful attempts providing the best results. Below in Figure 10, we see how the the default anchors were for aligned for Exp 4, whilst in Figure 11, we see how the clustered K-means boxes look like.

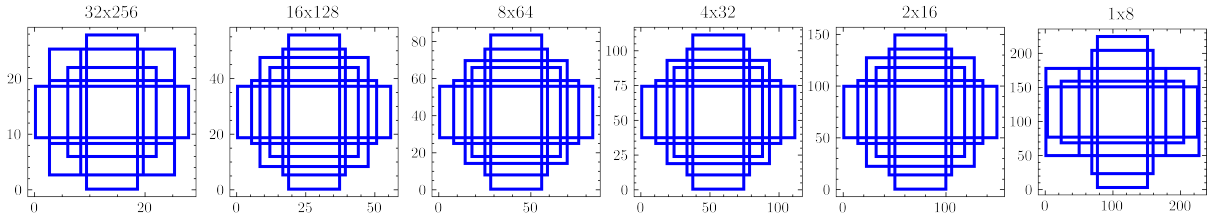


Figure 10: Default anchor aspect-ratios: [[2, 3], [2, 3], [2, 3], [2, 3], [2, 3], [2, 3]]

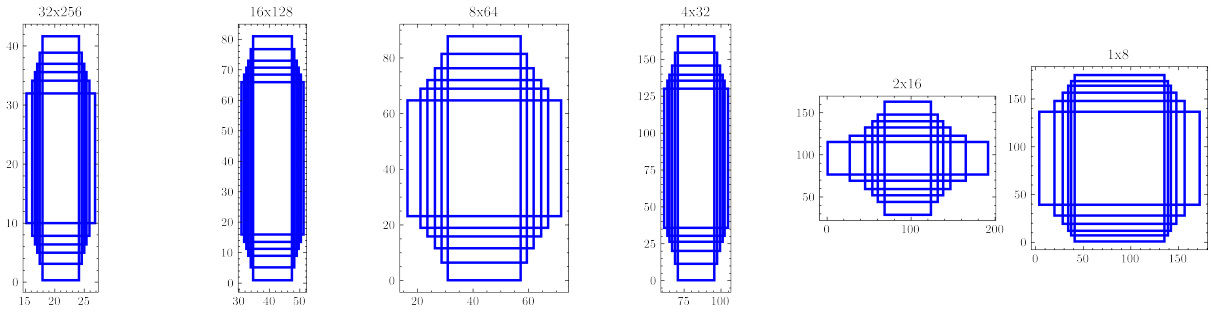


Figure 11: Clustered aspect-ratio sized anchor boxes using K-means, using above-mentioned values from Table 8.

In initial experiments, I tried to use the more vertically inclined aspect-ratio, from the cluster, in the first feature maps, and the more broader aspect ratios for the later feature maps (detecting smaller to bigger objects respectfully). That is: feature map [32x256] gets the aspect-ratio for the person, [16x128] gets aspect ratio for the scooter, ..., and [1x8] gets aspect-ratios for the bus (leaving out truck). This didn't work as good as it sounded theoretically, and might be because of the skewed distribution of the categories (in addition to the fact that we only have images of persons, riders, cars and buses represented in the validation set)

Using the information from the visual observations 2.4.2 and the skewed class distribution in the training data seen in Table 5, we instead of using all of the clustered anchor-ratios, only select the ones for persons and cars, as these combined represents about 80% of the training data as confirmed from Table 5. Looking at Figure 12, we allocate the persons ratios for the first two feature maps, while using car ratios for the rest.

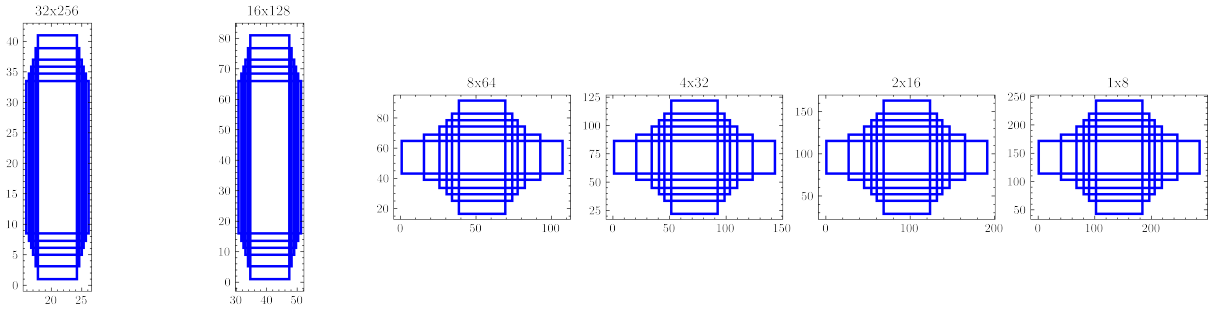


Figure 12: Using K-Means aspect-ratios for person for the first two feature maps, and car the last three.

This led to serious improvements, particularly for for detecting persons. We continue compare our Exp 4 "baseline" (using the default ratios) vs. the Exp 4 PKM (partial K-means) as seen in Figure 12. By the looks of the accuracy improvements in Figure 13, we can see a more stable improvement throughout leading to an accuracy increase of 4.3% (measure at end). Comparing classification and regression loss, showed small to negligible differences and between the two, thefore won't be displayed.

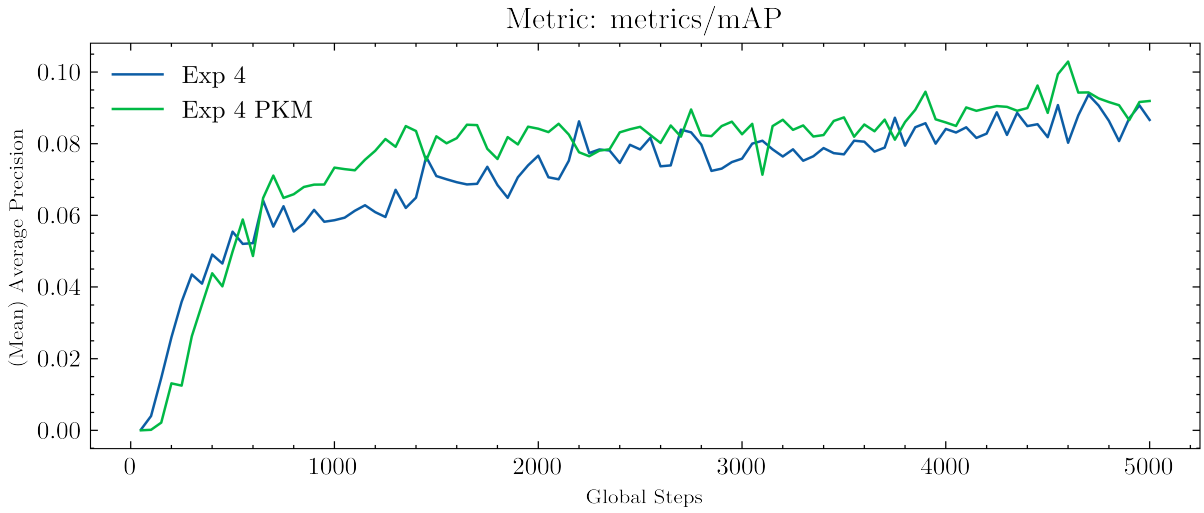


Figure 13: Showing general mAP comparing Exp 4 with and without partial K-means anchor-ratios.

As initially intended, I increased the average precision for persons considerably by almost 260%, as seen in Figure 14. Much of which would be the root cause of customizing the anchor ratios, though more specifically towards cars and persons which in the end, seemed to help the most.

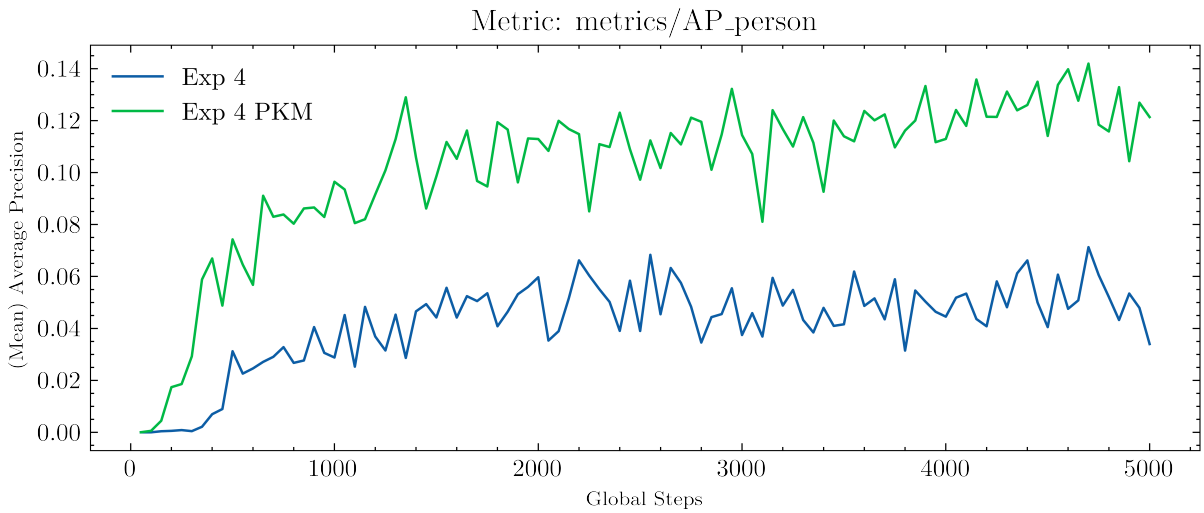


Figure 14: Showing general average-precision for person, comparing Exp 4 with and without partial K-means anchor-ratios.

We summarize the results below in Table 9, where we see the improvements across the board except for slightly worse on car. However, I suspect further experimentation would solve this e.g adjusting box_sizes, "fine-tuning" the clustered aspect-ratios or something in those lines.

-	*mAP*	*mAP*	*AP*	*AP*	*relative*	-	-
Legend	**@0.5**	**@0.5:0.95**	**@person**	**@ car**	**Train time**	**Parameters**	**Filename**
Exp 4	0.2058	0.08738	0.04387	0.2633	1h 29m 40s	11.2M	task23/retina_P4.py
Exp 4 PKM	0.2402	0.09115	0.1215	0.2411	1h 57m 13s	11.2M	task24/retina_P4_4.py

Table 9: Improving model Exp 4 by using customized anchor configuration calculated from K-means.

2.5 Extending the Dataset

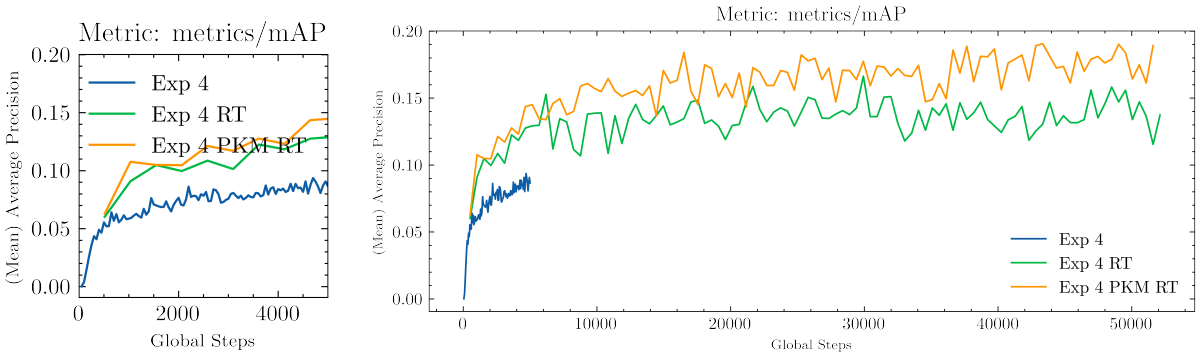
The effect of increasing the dataset size should almost always help. By allowing the model to view more images, it has the opportunity generalize a lot more then when comparing to a smaller dataset. Smaller dataset could result in poor approximation, bigger datasets could result in good approximations. This is to be consider when actually training from scratch, where the exception here is that pre-trained models could approximate well on smaller dataset.

We use the previously developed model *Exp 4* from config core/configs/task23/retina_P4.py, and train it from scratch with the added data while still using the same amount of epochs (100). In addition, we retrained another model using the same configuration from Exp 4 but this time using the new partial K-means anchor-ratios developed in the previous task.

-	*global*	*mAP*	*mAP*	*AP*	*AP*	*relative*	-
Legend	**Steps**	**@0.5**	**@0.5:0.95**	**@person**	**@ car**	**Train time**	**Filename**
Exp 4	5K	0.2058	0.08738	0.04387	0.2633	1h 29m 13s	task23/retina_P4.py
$\overline{Exp\ 4\ RT}$	5.16K	0.3018	0.1233	0.06146	0.3070	1h 20m 40s	task25/retina_P4_retrain.py
$\overline{Exp\ 4\ PKM\ RT}$	5.16K	0.3364	0.1361	0.1673	0.2764	1h 15m 12s	task25/retina_P4_retrain2.py
Exp 4 RT	52.12K	0.3254	0.1337	0.0793	0.2910	12h 27m 29s	task25/retina_P4_retrain.py
Exp 4 PKM RT	51.6K	0.4065	0.1776	0.2044	0.3192	12h 17m 2s	task25/retina_P4_retrain2.py

Table 10: Comparing model performance metrics with and without extended dataset, where $\overline{Exp\ 4\ RT}$ and $\overline{Exp\ 4\ PKM\ RT}$ represents only the first 5K global steps of the training sessions. Exp 4 RT and Exp PKM RT are trained for the full 100 epochs. RT symbols *retrained* and PKM denotes *partial K-means* (as seen in the task above).

Table 10 above clearly shows the difference in the achieved metrics with and without the extended dataset. Comparing Exp 4 and $\overline{Exp\ 4\ RT}$, we see an great increase in the mAP going from 0.2 to 0.3 in accuarcy, resulting in a 47% increase. Comparing with $\overline{Exp\ 4\ PKM\ RT}$, the difference is even higher, resulting in a 64% increase. The improvements holds for all other metrics as well, especially looking at the average precision for person, we get a initial increase of ~ 281%, or ~ 366% using the full 100 epochs (for $\overline{Exp\ 4\ PKM\ RT}$ and Exp 4 PKM RT respectively). Needless to say, extending the dataset has an enormous benefit.



(a) 5K global steps

(b) 52K global steps

Figure 15: Comparing model loss progression with and without extended dataset where both models have been trained on 100 epochs to showcase the difference clearly. (a) shows the first 5K iterations and how they compare on relatively alike scale, whilst (b) shows how the model learn when training for a longer time.

We can see how the training accuracy develops from the figure above, where we can clearly see the added benefit of using more data. The model trains with much greater mAP pretty much right of the start, and could be considered to stagnate around 15K-20K global steps.

As for the loss progression, the effect of introducing more unique images results in greater variations in the loss, as one would except. As far as it goes for loss, we actually do see an decline or at least a slower improvements compared to the Exp 4 -model. In addition, the training is more unsta-

ble leading for frequency spikes. We note classification loss is proving steadily, meaning the models are consistently classifying the correct categories for the boxes. However, for regression, we do see the models have slightly more issues positioning the bounding boxes correctly, leading to higher loss values.

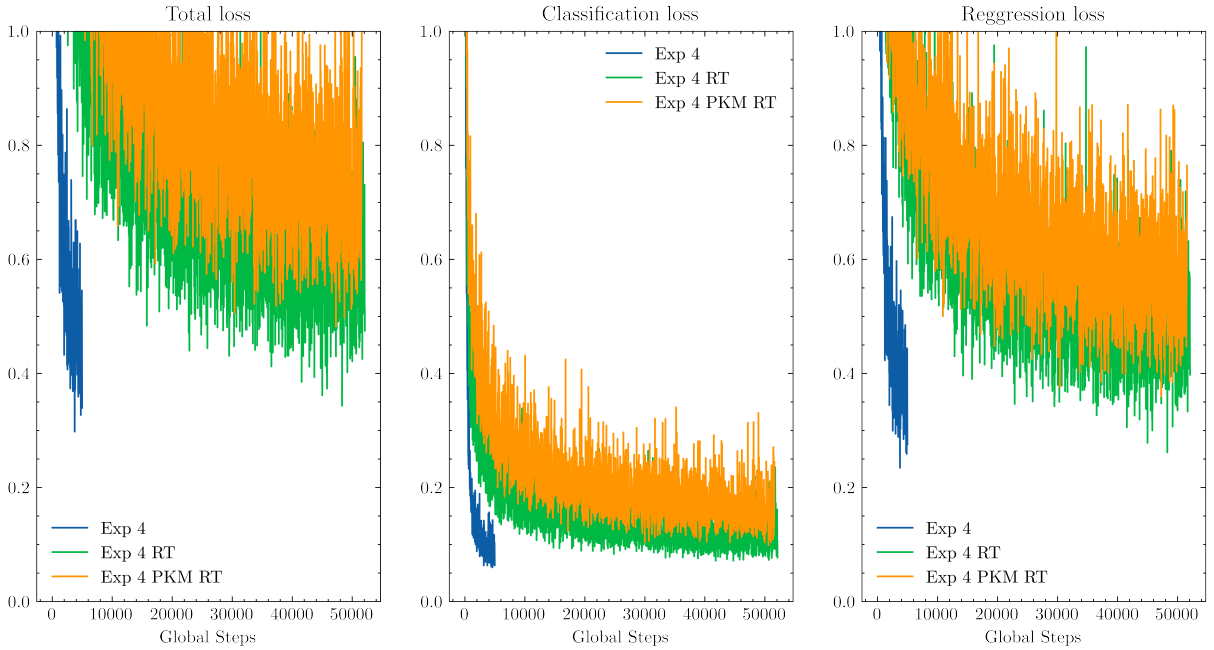


Figure 16: Comparing model loss progression with and without extended dataset.

Increasing the batch size would be one technique for make training more stable. Gradients in larger batches are more precise since they are calculated using more data. A bigger batch size, on the other hand, means fewer batches each epoch, which fewer chances for the model to update weights. In this case batch size isn't focused on for this task, however, it would be interesting to increase the batch sizes and adjust the learning rate slightly to reduce the training times as they models are taking a lot of time, as stated in Table 10.

3 Part 3 - Discussion and Evaluation

This section will focus on three influential modeling decisions which contributed to the progression of the project in its own way. We emphasis influential to signify something that would deem important to the projects as whole. Following, we itemized the selected modeling decisions as:

- Focal Loss with alternative alpha-weighting
- Deeper convolutional output heads
- Customized anchor boxes

We will for each subsection below, justify the models strengths, weakness and explain why it's contribution matters.

3.1 Focal Loss with alternative alpha-weighting

A particularly interesting find was to use a alternative alpha-weighting mentioned in Section 2.3.5. The addition of adding focal loss to the model truly helped the model performance across the board. The idea of using alpha-weighted for controlling category importance for tackling imbalance was important. The immediate effect of adding the alpha-weighted Focal loss improved the general performance metrics across the board. As previously mentioned, we had the effect of increasing the mAP of ~ 26%, most likely due to the increased average precision for persons. As seen in Table 4, going from Exp 1 to Exp 2 with new addition, we increased AP@person by going from 0.01127 to 0.04713, resulting in a accuracy increase of ~ 320%. One drawback of this addition is the added training time, as the classification loss is computed in a different way.

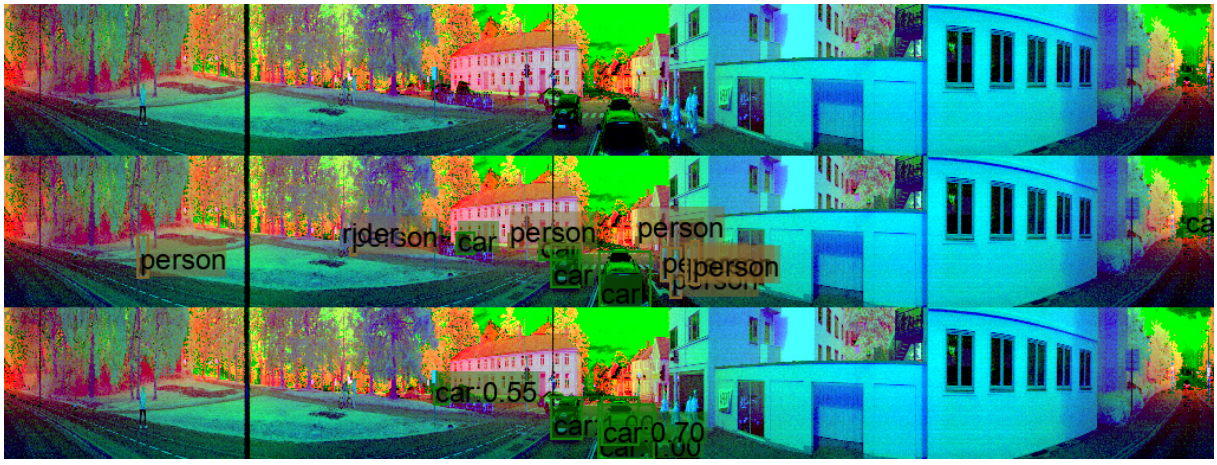


Figure 17: Inference on validation image with model Exp 1 with confidence = 0.5

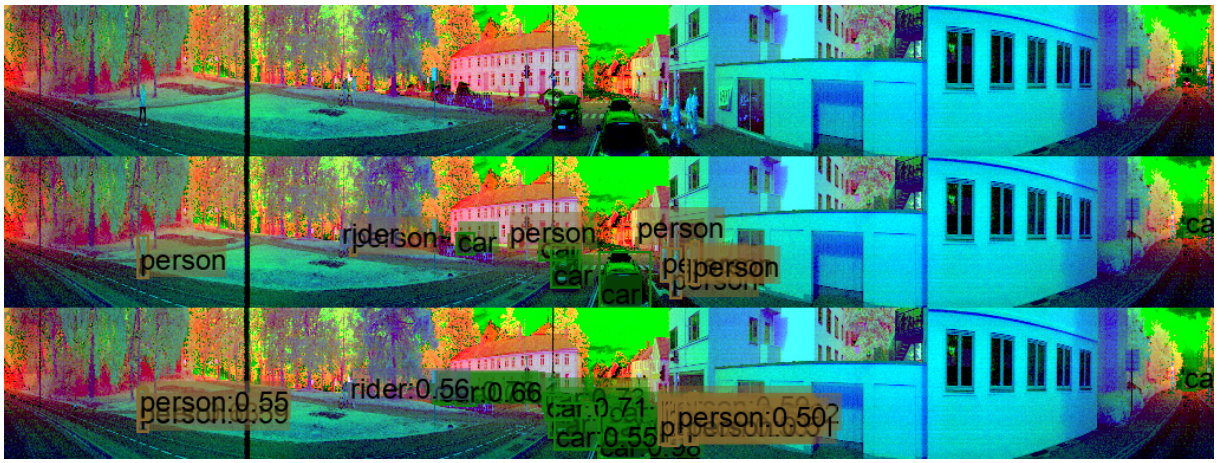


Figure 18: Inference on validation image with model Exp 2 with confidence = 0.5

By looking at Figure 17 and 18, we can see the improvements by adding focal loss, where the first model isn't able to detect the person on the left side and the persons on the sidewalk, which the Exp 2 shows. The added classifications is not only visible in this example image, but in the considerably lower classification loss (partially because we're summing instead of averaging the loss). However, comparing the loss with and without the focal, shows big deviations and the focal loss does infact contribute to better classifications.

3.2 Deeper convolutional output heads

Ideally, we want to create a model which is capable of retaining/compressing information. I mention this addition as influential due to its importance of increasing the number of parameters - that is: the model now has more potential weights to update, giving it the capability to retain more complex information between passes. By looking at the reported metrics from Table 4, we can't really see any added benefit of accuracy. However, it does lay the foundation for improvements. As we saw earlier when we used the baseline model when creating augmentations, we didn't really see the benefit as the model capacity was simply too small. As I later discovered this effect by increasing the model size, I found the importance of increasing the number of parameters to be quite influential. The disadvantages of this measure is, for this case, the not-so-great improvement in accuracy, but worse the added slow to inference speed. Simply by deepening the conv heads, we effectively decrease the inference speed by -63%. This makes sense as increase the model parameters from 6.5M to 11.2M requires more computations.

As we later introduced more data into the pipeline, the added conv heads are believed to play a huge part of the improvements made. As we were making fair comparison by training the same amount of global steps, we kind of put a lid on the potential of the increase model size. In this case, using a constant number of epochs, batch size and learning rate, the bigger model would've needed

more time to train. This is something that became evident, as we extended the dataset in Section 2.5. By increasing the dataset and extending the training time considerably, we could reap the benefits of the number of parameters the model has to compress information.

3.3 Customized anchor boxes

One of the most important factors for high-quality object detection is the anchor boxes. As we're using a custom dataset, it would only be natural to use custom anchor boxes as well to really specialize our detection model. For this question, we could highlight various effects of specializing anchor boxes - however for this instance, I find it interesting to emphasizes the improvements of detecting persons and cars, that is AP@person and AP@car metric.

As we previously saw in Section 2.4, we achieved an improvement of detection persons by ~ 281%, or ~ 366% using the full 100 epochs (for $\overline{Exp\ 4\ PKM\ RT}$ and Exp 4 PKM RT respectively). The main cause of this improvement was the specialization of the anchor boxes. As the default boxes are quite general in terms of their shapes (two square boxes, two wide boxes, two tall boxes) as seen in Figure 10, the K-Means generated anchor-ratios are much more shaped to their purpose. E.g persons are shaped to be slim and tall, very much alike the actual physique of a person (seen in the first feature map [32x256] of Figure 11), and cars are slightly more compact-shaped much alike how we see cars in the daily life.

By assigning the first two feature maps to use person-specific anchor-ratios and the last four on car-specific anchor-ratios, we essentially cover the main two categories specifically to improve the overall mAP. This makes sense, as previously mentioned, they both represent 79.14% of the training set alone. However, as we used extended data for the latter part, the effect became much more prominent for its improved performance as the extended data consist of 84.1% cars and persons combined.

The immediate drawbacks of using such specialized anchor boxes, is the effect it has on detecting bigger objects e.g trucks and buses. As these categories are quite under represented in the data, 2.43%+0.67% = 3.1% and 4.04%+2.45% = 6.49% for the training data and extended data respectfully. This makes them naturally quite difficult to detect and making fair comparison to other accuracy's for other categories to a relative scale is difficult.

3.4 Modeling weaknesses

In hindsight, it is always easy to critic previous decisions. For this section I'll emphasise a few approaches I would've changed if I were to deliver this to someone else. Even though there is a lot to talk about, I'll only highlight a few specifics in detail: data annotating, initial developement and model constraints.

3.4.1 Data Annotation

If this project would refer to as an approach for developing object detectors, I would've done things a little bit different. First of, as stated earlier, I would've ensured that the actual datasets was 100% perfectly annotated (recall disclaimer Section 2.4.1) as it it crucially import when developing e.g object detection models used for self-driving as it relies on highly safety-critical systems.

Next up, through previous data discoveries in Task 1.1, I would ensure that the validation data would've had better representations for all kinds of categories. It wouldn't make sense to deliver a sub-tested system to a customer with a model only trained at detecting *some* objects in traffic.

3.4.2 Initial developement

Something I think the development progress would benefit from, is actually making sure that it is theoretically possible to learn your data 100%. This goes hand-in-hand with the mentioned spesifics above, and should be taken seriously. In the beginning of the project, after making sure your data was solid, it is important to validate it in some way. A key idea here is to actually run your model on only partial sets of your data to ensure it can fit the data.

That is: if you can't overfit on a single batch of images - then how are you going to learn the whole dataset? The idea behind it is to uncover irregularities that jeopardises the training progression. If overfitting doesn't work, then we know something is wrong and have the possibility to debug. By following these measures and ensuring data quality, we can save immense time later on when we start to introduce complexity into the model.

3.4.3 Model selection and constraints

Something I really missed, and which is quite vital for object detection in traffic is the speed. If your model runs at 2 FPS then its utterly useless. Therefore, setting up base constraints to ensure safety is important prior to developing models. E.g. making speed a priority forces the developer to constantly verify every implementation detail and prevents him for blindly picking a random ResNet-1202 and just look at wether the mAP increases or not. For reference, the company comma.ai, which creates self-driving car kits, stated back in 2016 that they run their cameras at 20 Hz Santana and Hotz 2016.

It would be naturally to assume their detection systems also runs at these speed. By making this a lower-bound constraint, we effectively make sure that all model iterations on-wards satisfies this.

4 Part 4 - Model Explainability

Tasks up until this point have been primarily focused on model developement and performance analysis. Therefore it was found reasonable to select a task which emphasises neural network explainability.

To ensure good feature map activation's, I've created a tiny separate dataset which I've on purpose trained an model on to ensure it should overfit. By over-fitting on these images, we're sure that the neural network will have good activation's for reasonable correct placements of bounding boxes. This is done to ensure the class activation maps makes sense and minimize time spent debugging sub-optimal model performance yielding poor activation's.

Following section shows results conducted from the Notebook data_cam.ipynb. Please refer to it for more information regards to implementation details. The notebook is inspired and built on-top of code found in `https://github.com/jacobgil/pytorch-grad-cam`. In addition, all I've created a few helper functions inside tools/cam.py to ease the implementation.

4.1 Explaining the Model with CAM

To show how class activation maps, we've randomly used a image from the TDT4265 dataset. Figure 19 below shows the baseline image, which we will conduct the CAM on.

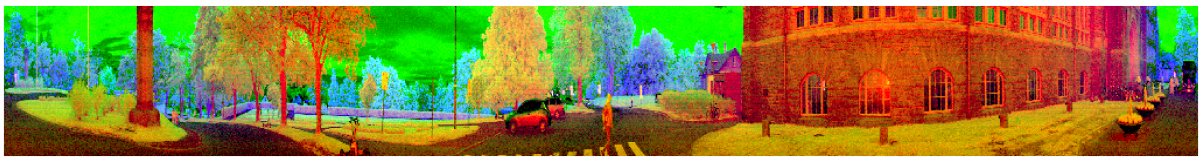


Figure 19: Unprocessed baseline image

Next up, we show how the actual score, label and bounding box predictions look from the model. By looking at Figure 20, we can see solid predictions with high probabilities per bounding box as our model which we trained up to an mAP of 80%.

Figure 20: Baseline image with predictions

Finally, we apply the CAM processing using existing predictions above. In addition, we add renor-

malization to the activation maps to force them to appear inside the actual bounding boxes. To simplify, we remove the display text and add a zoom to the center of the image to enhance the visualization. Figure 21 below show the final result of the CAM for the single image.

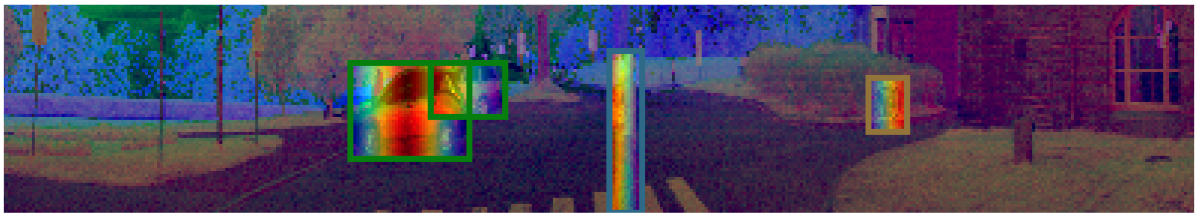


Figure 21: Baseline image with class activation maps

Bibliography

He, Kaiming et al. (2015). Deep Residual Learning for Image Recognition. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

Lin, Tsung-Yi et al. (2017). Focal Loss for Dense Object Detection. DOI: 10.48550/ARXIV.1708.02002. URL: https://arxiv.org/abs/1708.02002.

Liu, Wei et al. (2016). "SSD: Single Shot MultiBox Detector". In: Computer Vision – ECCV 2016. Springer International Publishing, pp. 21–37. DOI: 10.1007/978-3-319-46448-0_2. URL: https://doi.org/10.1007%2F978-3-319-46448-0_2.

Santana, Eder and George Hotz (2016). Learning a Driving Simulator. DOI: 10.48550/ARXIV.1608.01230. URL: https://arxiv.org/abs/1608.01230.

Appendix

A Presentation video

The presentation video can be viewed in full on YouTube.

`https://www.youtube.com/watch?v=9rOMoMoi5zs`

B Source Code

The source code can be viewed in full and is available on Github.

`https://github.com/andreasoie/neuralvision`

C Incorrect annotatations: TRAINING SET

We itemize the training set images with Image ID and source of error.

- 10, person is behind three
- 50-97, car is annotated through the car
- 121, empty car annotation
- 138-151, a lot of scooter annotations missing?
- 303-403, missing a lot of bicycles?
- 428-440, missing a lot of bicycles?
- 636-684, missing a lot of bicycles?
- 763-768, rider+bicycle is behind passing big black car?
- 801-804, rider+bicycle+person are behind threes?
- 824-830, car to the right is occluded by object
- 905-910, car in right lane is occluded by another car
- 994-999, person/rider (missing bicycle) is behind car in right lane (Figure 22)
- 1008-1018, car is occluded by passing truck (Figure 23)
- 1122-1124, person on sidewalk is occluded by a passing car
- 1205-1242, on right side, walking person, rider and car is kind of occluded with each-other

- 1223-1228, rider on left side is occluded by passing car
- 1243-1272, walking person on right side is partially occluded by car
- 1275-1293, walking person on right side is partially occluded by car
- 1308-1311, cars in line on left side occluded big big car behind
- 1343-1353, walking persons on sidewalk occluded by passing cars
- 1384-1399, walking persons on sidewalk occluded by passing cars

Figure 22: Single example from the training set. All images with Image ID from 994 to 999 shows person/rider (and missing bicycle) is behind car in right-hand lane.

Figure 23: Single example from the training set. All images with Image ID from 1008 to 1018 shows car is occluded by passing truck.

D Incorrect annotatations: VALIDATION SET

We itemize the validation set images with Image ID and source of error.

- 13-18, right side, person on sidewalk occluded by passing car
- 48-71, isn't bicycle also to be annotated where annotating a rider?
- 89-91, left side, person on sidewalk occluded by passing car
- 72-101, are persons in mirrors also persons?
- 102-107, left side, car/persons occluded by passing bus (Figure 24)
- 114, left side, car/persons occluded by passing bus
- 118-126 persons occluded by passing car
- 118-126 persons occluded by passing bus

- 118-126 parked cars occluded by passing car
- 118-126 bus occluded by another bus passing bus
- 129-131, left side, persons on sidewalk occluded by passing car
- 151-153, left side, persons on sidewalk occluded by passing car
- 159-162, left center, persons on sidewalk occluded by passing car
- 171-175, left center, persons on sidewalk occluded by passing car
- 180-182, center, persons on sidewalk occluded by passing car
- 187-189, left side, persons on sidewalk occluded by passing car
- 181-189, parked bus occluded by passing bus
- 202-244, left side: parked scooters missing annotations
- 226-257, right side: parked bicycles / scooters missing annotations
- 213-263, both sides: standing people with occasional occlusions
- 290-301, center: parked scooters missing annotations

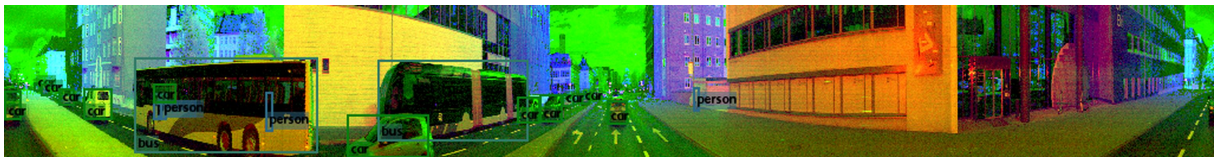


Figure 24: Single example from the validation set. Images with Image ID from 102 to 107 shows left side, car/persons occluded by passing bus/car