A REPORT ON

Training SqueezeNAS models with RailSem Dataset

by

Amid Mozelli

Supervisor(s):

Alexander Wendt

-

Vienna, Austria

December 2020

Contents

Chapter 1

Introduction and Motivation

Our task is to take the state of the art models of the authors in [1], which is a hardware aware implementation, and retrain them with a new data set provided by the company Mission Embedded to see if they perform good enough for applying them in the field of autonomous driving on rails. In this chapter, we give an overview of the problem and the techniques for solving it.

1.1 Approach

Neural Architecture Search (NAS) has been effectively used to develop low-latency networks for image classification. In this paper, the authors claim to have the first proxy-less hardware-aware search targeted for dense semantic segmentation. What we are trying to do is to re-implement their networks in order to have an experiment with another dataset to check their performance.

1.1.1 Model Architecture

In [1] authors use a gradient-based NAS method to optimize a super-network for both high semantic segmentation accuracy as well as low latency on the target hardware. They explore three search spaces of encoders namely, *Small*, *Large* and *XLarge* for semantic segmentation networks consisting of sequential Inverted Residual Blocks [2]. Their search space was chosen to be similar to the MobileNetV2 [2] and MobileNetV3 [3] search spaces so they can directly compare their segmentation optimized networks to their classification optimized networks. Figure 1.1 shows the architecture of the search space they have used for building their networks. There are different choices of architecture in each block that would shape the corresponding network depending on the task. In each architecture search, they constrain the macro-architecture and find optimal parameters for each block.

In order to make computation and optimization of the stochastic supernetwork tractable, each superblock picks a candidate block independent of the choices of other superblocks (Figure 1.2). Thus,

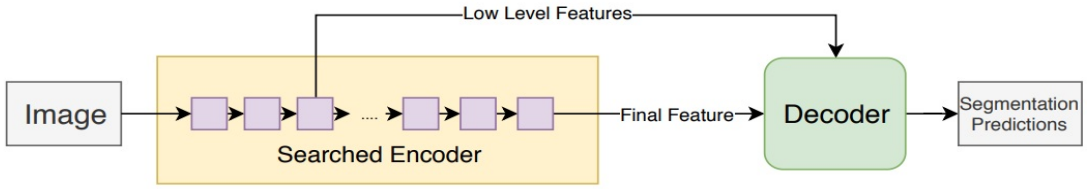


Figure 1.1: General Encoder-Decoder Structure of the Segmentation Networks [1].

they can model the choice of a candidate block as sampling from an independent categorical distribution where the probability of choosing candidate block j for superblock i in the network is $p(i, j)$. We define this probability using the softmax function on our architecture parameters (θ) for each superblock.

$$p(i, j|\theta) = \frac{e^{\theta_{i,j}}}{\sum_{j}^{13} e^{\theta_{i,j}}}. \tag{1.1}$$

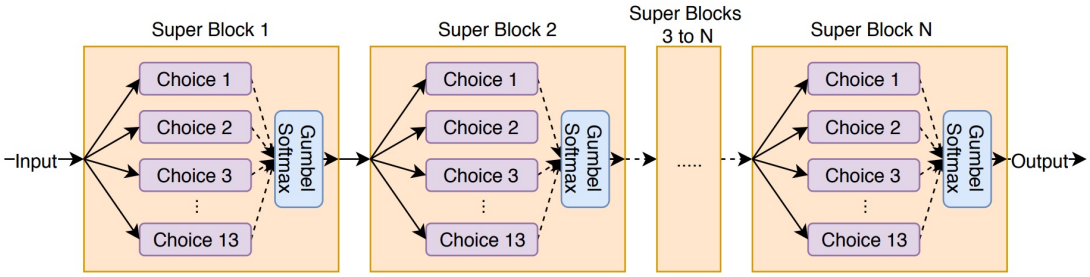


Figure 1.2: Diagram of a supernetwork with N superblocks.

Figure 1.3 shows the architecture of the Inverted Residual blocks they use in the search space. They are parameterized so that the number of groups (g = 1, 2) in the 1x1 convolutions, the dilation rate of the depthwise convolution (d = 1, 2), the kernel size (k = 3, 5), and the expansion ratio (e = 1, 3, 6) may vary for different candidate blocks.

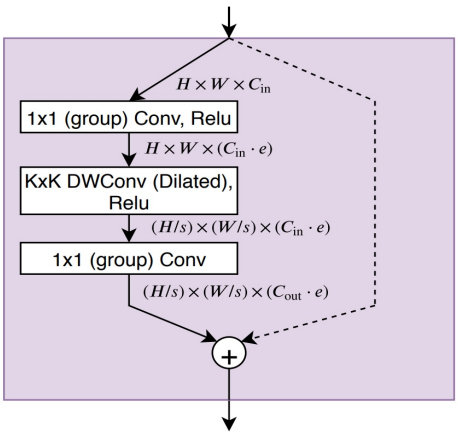


Figure 1.3: Architecture choice for blocks.

Hardware-Agnostic Search

For their hardware-agnostic architecture searches, they apply the NAS method with a Multiplier–Accumulators (MACs) minimization objective to create networks that are on the pareto-optimal tradeoff curve of MACs vs mIOU. To implement this, for each block i in the network, we compute the number of Multiply-Accumulates for each candidate block j and store the results in the lookup table C such that $C(i, j) = MACSi, j$. We then perform an independent search in each of the three search spaces and obtain three MAC-optimized *SqueezeNAS-MAC networks*.

Hardware-Aware Search

The hardware-aware searches use the same NAS algorithm and architectural search space as the hardware-agnostic approach, but now they use a latency minimization objective for the resource-aware loss; formulated as $C(i, j) = Latency_{i,j}$. To compute the latency of every candidate j in each block i, they measure the inference time of all candidates on the target platform. They conduct 3 new independent hardware-aware searches that target the latencies measured from the hardware-agnostic networks. The results of these searches yield the three *SqueezeNAS-LAT* networks. The hardware-aware searches find networks that have significantly higher accuracies at the same or lower latency compared to the hardware-agnostic networks. The latency-optimized networks have a higher number of MACs, but they still run faster on our target device.

1.1.2 Semantic Segmentation

Semantic Segmentation is the process of labeling each pixel in an image, so each object is classified with the pixels in the image having the corresponding label. Images used in the semantic segmentation problems are mostly high resolution with higher pixel values so the networks can identify each object with their space of pixels. Networks designed for segmentation tasks usually have a U-shaped architecture as they get fed with bigger pictures, so they downsize the images, extract their features, and then again upscale the image for the output.

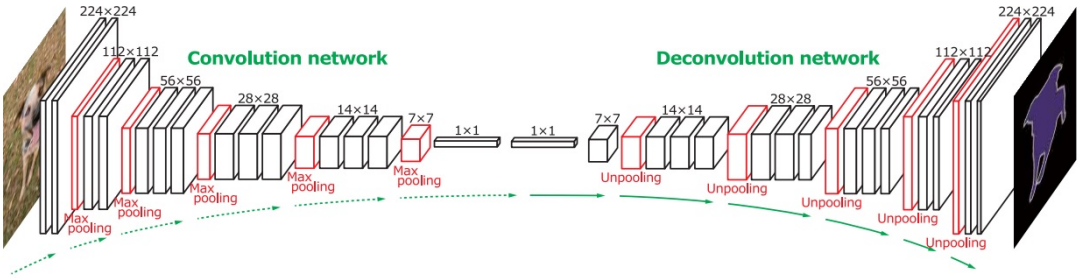


Figure 1.4: Architecture of a semantic segmentation network [4].

1.2 Problem Description

Our assignment is to conduct an experiment using the designed networks from SqueezeNAS [1]. These models are already trained on the CityScapes [5] dataset to detect urban-related objects. We are provided with the RailSem [6] dataset in order to prepare the networks for a task of segmentation for rail objects and all the other rail-based and rail-related items(we explain all the datasets in Section 2.2).

1.3 Expected Results

Our goal is to get the best possible inference results from the networks trained with the new dataset. Authors in [1] claim that their networks perform better in latency and accuracy measurements than the MobileNetV3 [3]. If this would also be the case with our trained network, we then have a perfect optimized network for the task of rail segmentation for use in the field of rail autonomous driving. We would then compare our results with other networks at the end of the experiment to provide the readers with an accurate result.

Chapter 2

Preparation

In this chapter we explain how we decided to tackle the problem and start with the paper reproduction. We also have a brief introduction of the datasets used in the experiments.

2.1 Paper

We are focusing on reproducing the results of the paper “SqueezeNAS: Fast neural architecture search for faster semantic segmentation" [1] so we can understand the logic behind their work in order to employ them for our task in hand to get the best results.

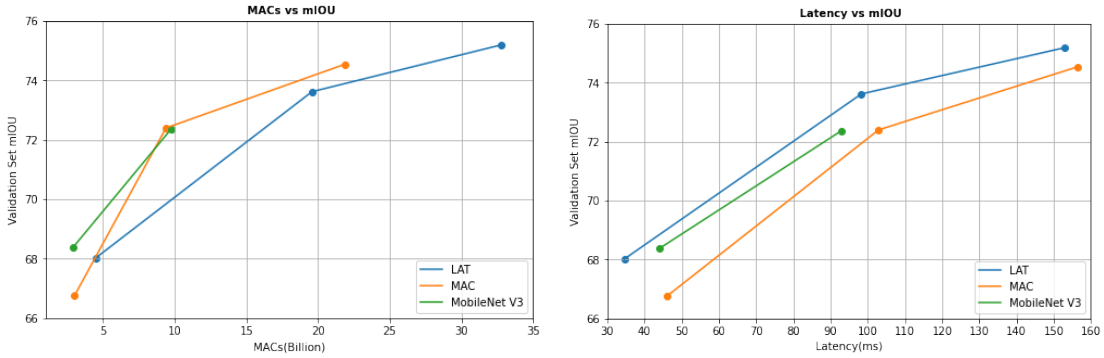
2.1.1 SqueezeNAS

Authors in [1] have introduced their approach to the neural architecture search for semantic segmentation task. They perform a proxy-less hardware aware search and build two types of networks, MAC-Optimized and LAT-Optimized.

- MAC-Optimized models are designed to use the least MAC possible in their networks.
- LAT-Optimized models are designed to have the least possible latency in their predictions.

Both models have three different sizes (Small, Large, Xlarge). The small and large models are comparable to the MobileNetV3 [3], which also has two different sizes. Figure 2.1 shows the comparison of MACs and latency in relation to the validation set mIOU between the SqueezeNAS models and MobileNetV3 [3] models. Authors are trying to prove wrong the mainstream ideas in the computer vision community in this research by suggesting their assumptions on:

- Target task dependant accuracy: SqueezeNet has a lower accuracy than VGG on ImageNet but is more accurate in identifying similar patches in a set of images.



(a) MACs vs mIOU on Cityscapes validation set.

(b) Latency vs mIOU on Cityscapes validation set.

Figure 2.1: Our measurement results for Latency vs mIOU

- Target task dependent Deep Neural Network (DNN) design: Choosing the right DNN for a target computing platform (e.g. CPU, GPU or TPU).
- Fewer MACs will not necessarily result in lower latency: Depending on the processor and the kernel implementations, different convolution dimensions run faster or slower even with the same amounts of MACs.

2.2 Datasets

We explain two datasets that are used in this experiment. One is CityScapes which our primary networks are trained on and the other one is the RailSem which we are using for our trainings.

2.2.1 CityScapes

CityScapes [5] is a large-scale dataset that consists of a wide range of urban images across 50 different countries, including different times of the year, with high-quality pixel-level annotations. For segmentation tasks, CityScapes provides dense pixel-level annotations for 5000 images at 1024x2048 resolution pre-split into training (2975), validation (500), and test (1525) sets. Label annotations for segmentation tasks have more than 30 classes commonly encountered during driving scene perception.The SqueezeNAS networks are trained on CityScapes coarse training set annotations(fig. 2.3) and the CityScapes fine training set annotations(fig. 2.2).

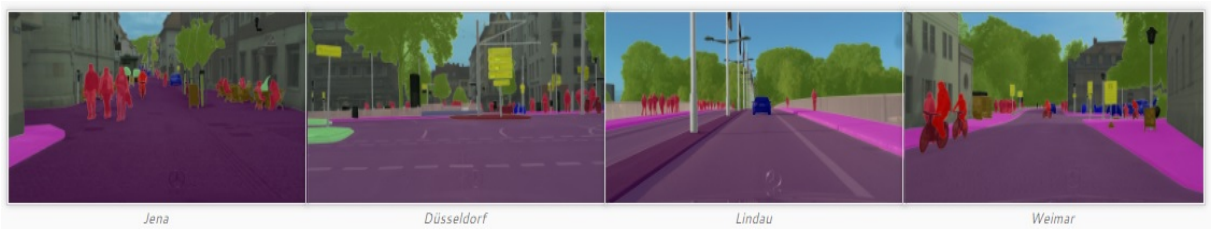


Figure 2.2: Fine annotations.

Figure 2.3: Coarse annotations.

2.2.2 RailSem

The data set that we are using in our experiment is from AIT (Austrian Institute of Technology) and can be found in this LINK. RailSem19 offers 8500 unique images taken from the ego-perspective of a rail vehicle (trains and trams). Extensive semantic annotations are provided, both geometry-based (rail-relevant polygons, all rails as polylines) and dense label maps with many Cityscapes-compatible road labels. Many frames show intersection areas between road and rail vehicles (railway crossings, trams driving on city streets). RailSem19 is useful for rail applications and road applications alike [6]. This dataset is the one provided by the Mission-Embedded company in order for us to train the SqueezeNAS models, so we can see how they would perform in this segmentation task.

Figure 2.4: RailSem Dataset

2.3 Methods

2.3.1 Environment Setup

NVIDIA Jetson Xavier

We have decided to use the same GPU-Device as in the paper, namely “NVIDIA Jetson Xavier". The device needed to be set up, which took some time for us. There are certain JetPack software packages with which the device should be flashed and run. When we started the project, JetPack 4.3 was the newest release, so our device was set up with that version.

Pytorch

Jetson Xavier needs a specific build of PyTorch that suits its hardware architecture, so we had to use the specific wheel for that. There are pre-build wheels for this issue on the NVIDIA webpage that can be accessed through this LINK.

Docker

Docker is a platform that uses OS-level virtualization and to deliver software packages called containers. After facing some problems with the environment setups using the JetPack for the Xavier (which we later solved), we decided to use Docker containers for our implementations. There are so-called “Docker Images" that contain the OS/Software package, and we run these images in an environment called “Docker Container." There is a specific ML-Container for NVIDIA devices used for our task. The version of NVIDIA Docker image that we have used can be found in the following LINK.

2.3.2 Reproduction of paper

After setting up the Xavier and finishing the environment setup, what we did first was to reproduce the paper’s results. For that matter, we re-implemented the evaluation code of the paper on NVIDIA Xavier (same GPU-Device that they used in the paper), and since there was not any specific code for the latency measurements, we added Cuda-events and wrapped our “model(input)" function with enable timings and also “cudnn.benchmark" activated at the beginning in order to have the most accurate measurement possible on the target hardware. There are multiple power modes on the NVIDIA-Xavier we have tried to measure the latency on the two of them, which are MAXN (figs. 2.8a and 2.8b) and 30W (figs. 2.8c and 2.8d) power modes. You can find the results of the latency measurements of the LAT-Optimized networks in Figure 2.5 and the MAC-Optimized networks in Figure 2.6.

Networks	Paper Values(ms)	Measurements in MAXN mode(ms)	Measurements in 30W mode(ms)
SqueezeNAS LAT Small	34.57	43.18	57.99
SqueezeNAS LAT Large	98.28	117.72	171.44
SqueezeNAS LAT XLarge	152.98	174.4	264.85
MobileNet V3 Small	44.01	----------	----------
MobileNet V3 Large	92.78	----------	----------

Figure 2.5: LAT-Optimized latency measurements

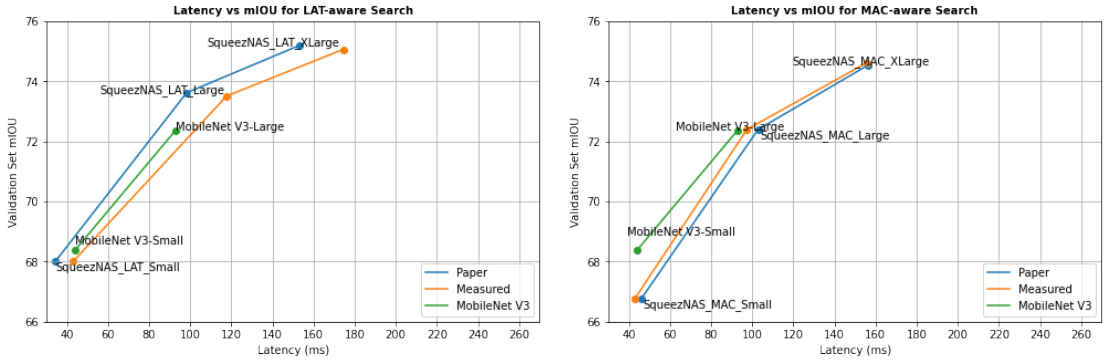
Networks	Paper Values(ms)	Measurements in MAXN mode(ms)	Measurements in 30W mode(ms)
SqueezeNAS MAC Small	46.01	42.94	49.45
SqueezeNAS MAC Large	102.9	97.48	111.17
SqueezeNAS MAC XLarge	156.41	155.74	262.9
MobileNet V3 Small	44.01	----------	----------
MobileNet V3 Large	92.78	----------	----------

Figure 2.6: MAC-Optimized latency measurements

Architecture	Class mIOU	Category mIOU	MACs(G)
SqueezeNAS MAC Small	66.77	83.93	3.01
SqueezeNAS MAC Large	72.4	86.18	9.39
SqueezeNAS MAC XLarge	74.62	86.99	21.84
SqueezeNAS LAT Small	68.01	84.13	4.47
SqueezeNAS LAT Large	73.51	86.58	19.57
SqueezeNAS LAT XLarge	75.06	87.28	32.73

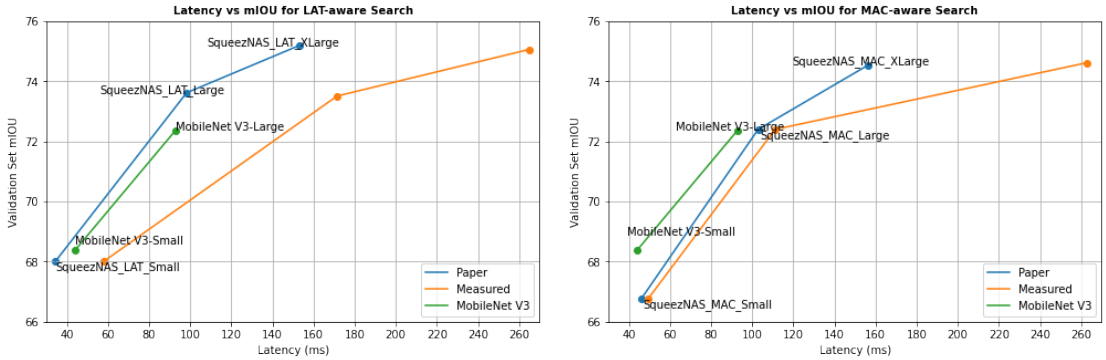
Figure 2.7: mIOU Values

The results for the mIOU calculation are also to find in Figure 2.7. We get the same results for the mIOU values, but there are two major differences in our results. First is the variation of latency values in general; the second one is the disagreement with the assumption of LAT-Optimized models, as they are supposed to be faster than the MAC-Optimized models; the opposite is the case here. The reason for that might be that we did not use TensorRT models for our Latency measurements as we faced a problem converting Onnx versions of our models to TensorRT. We have gathered our latency results in Figure 2.8 for a better comparison.



(a) LAT-Optimized in MAXN power mode

(b) MAC-Optimized in MAXN power mode



(c) LAT-Optimized in 30W power mode

(d) MAC-Optimized in 30W power mode

Figure 2.8: Our measurement results for Latency vs mIOU

Chapter 3

Execution

Our data from RailSem is a bit different from the CityScapes, and since the models are already trained with the CityScapes, we needed to prepare our new data in order to use them for a session of training on the SqueezeNAS models.

3.1 Pre-Processing

Image size in CityScape is 1024x2048 for Railsem; on the other hand, it is 1080x1920, so we needed to reshape our data before feeding them to the network. The RailSem dataset labeling is somehow different from CityScape, but they also use 19 trainable labels, which are mentioned in [6]. RailSem does not have a colormap function, so we have implemented a mapping function inspired by the label's script in the CityScape dataset [1] which is included in the helpers part of the package. Our input images are normalized with the standard deviation and mean values provided by the CityScape and used in the paper. Our masks, however, are remained as they are. RailSem provides 8500 images with masks, so we decided to use 4000 for our small models, 6000 for large, and 8000 for the xlarge models with a validation split of 0.3. We keep the last 500 images for the testing part.

3.2 Training

There are two types of models, namely LAT/MAC-Optimized, each having three different sizes. For each of our six models, we take the weights of already trained models, and we try to implement the transfer learning by retraining those models with the new data. As mentioned earlier, the main goal of this task is to be able to detect the rail objects in the provided images

As we have a multi-class segmentation task, we use the Cross-Entropy loss function for our training.

We ignore the label 255 in our loss function. Our optimizer is Stochastic Gradient Descent (SGD), and the learning rate value is 0.001, which would decrease to 1e-5 after 25 epochs. We use batch-size one as they have also done in the paper, and we train each of the models for 30 epochs with RailSem data. We save the models with the least validation-loss value, and we also save each of them after each epoch for a better comparison at the end.

3.2.1 Segmentation Models Pytorch

Segmentation Models Pytorch is the package specified for the segmentation problems implemented in Pytorch. There are different implemented networks ready for training and other examples. We made use of some of the training tools of this package for our task. The repository of this package can be found HERE.

Chapter 4

Results

After the training, we performed the testing phase, which we would demonstrate in this chapter.

4.1 Inference

The latency measurements are done with the Cuda-events here as we described in Section 2.3.2, so we can have a fair comparison. We have gathered our results in figs. 4.1 and 4.2 to compare them with the results from the models trained on the CityScape dataset.

4.1.1 IOU Scores

We are displaying some of the predictions our model has made for some example images with better IOU-scores. Figures 4.3 to 4.5 are showing the LAT-Optimized models' predictions, and Figures 4.6 to 4.8 are showing the MAC-Optimized models predictions. The mIOU values for the whole test set can be found in Figure 4.1. For this IOU-calculation task, we have tried THIS implementation, but we adjusted the tool so that it would work with our data.

Architecture	mIOU
SqueezeNAS MAC Small	36.62
SqueezeNAS MAC Large	41.48
SqueezeNAS MAC XLarge	44
SqueezeNAS LAT Small	40.49
SqueezeNAS LAT Large	42.12
SqueezeNAS LAT XLarge	46.76

Figure 4.1: mean-IOU measurements on validation data set

4.1.2 Latency Measurements

We performed the latency measurements on the NVIDIA Xavier to compare the results to our earlier measurements. We have provided the results in Figure 4.2. The measurements here are done in MAXN power mode on Xavier.

Architecture	Latency(ms)
SqueezeNAS MAC Small	34.36
SqueezeNAS MAC Large	76.23
SqueezeNAS MAC XLarge	178.78
SqueezeNAS LAT Small	39.79
SqueezeNAS LAT Large	116.68
SqueezeNAS LAT XLarge	180.18

Figure 4.2: mean-IOU measurements on validation data set

4.2 Summarize

In general, the LAT-Optimized models perform better in the mIOU calculations, and if we go back to Figure 2.7 in Chapter 2, we can see that the models trained on CityScape are also behaving similarly.

In the latency section, however, we have the same problem as before, and the MAC-Optimized models have faster latency times which is no in agreement with what the author suggest.In comparison to Figures 2.5 and 2.6 both MAC/LAT-Optimized models are performing better in average on RailSem data set in latency measurements.

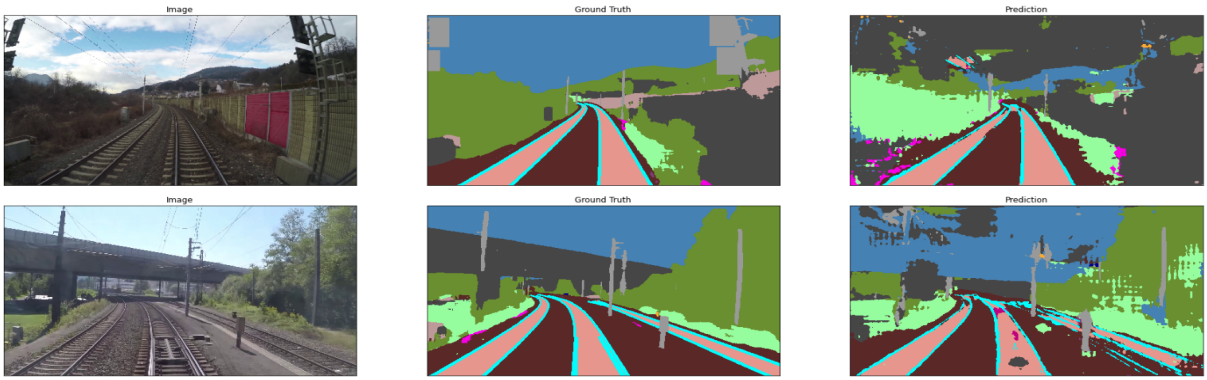


Figure 4.3: SqueezeNAS-LAT-Small (IOU:0.56-0.74)

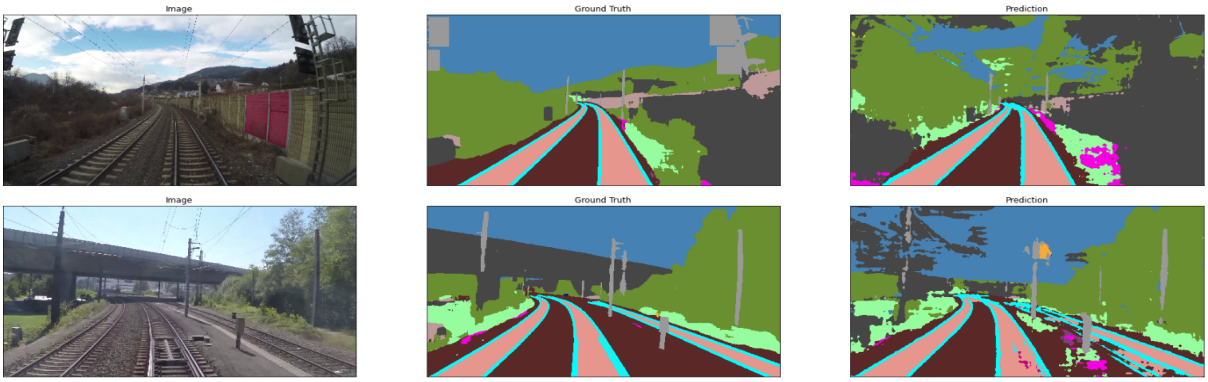


Figure 4.4: SqueezeNAS-LAT-Large (IOU:0.68-0.78)

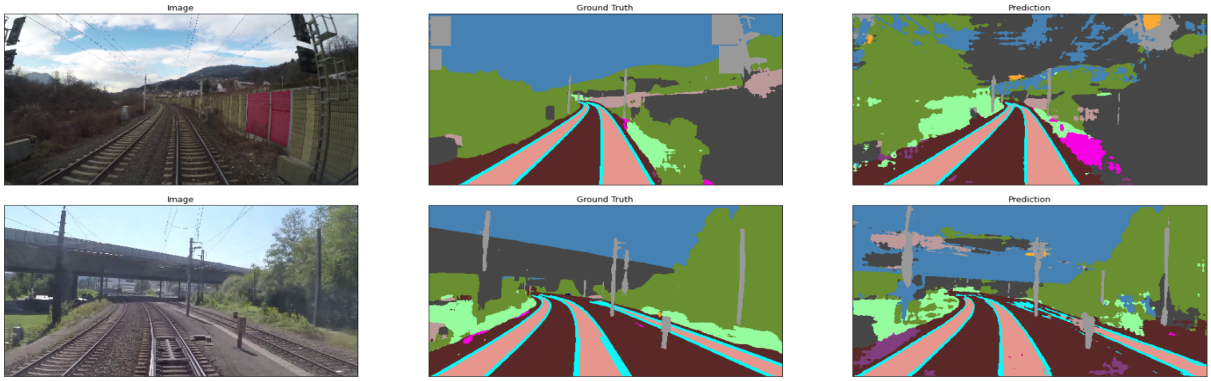


Figure 4.5: SqueezeNAS-LAT-XLarge (IOU:0.68-0.77)

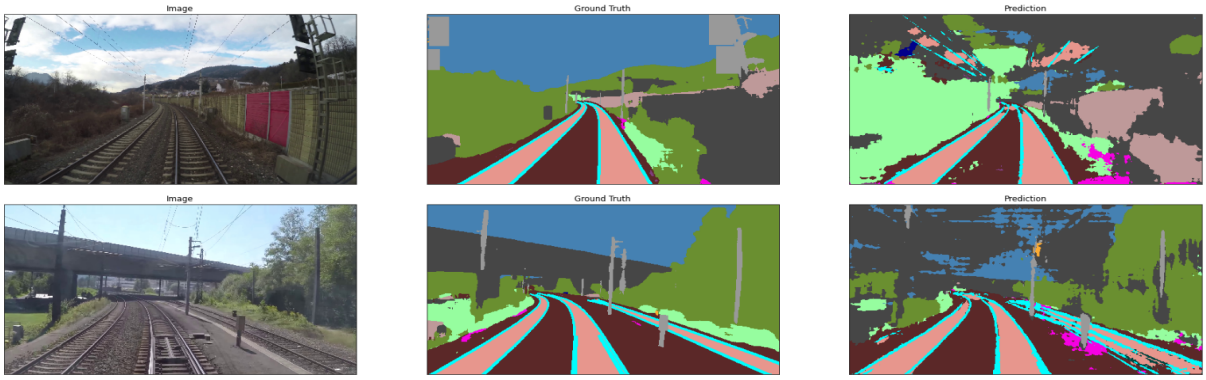


Figure 4.6: SqueezeNAS-MAC-Small (IOU:0.54-0.65)

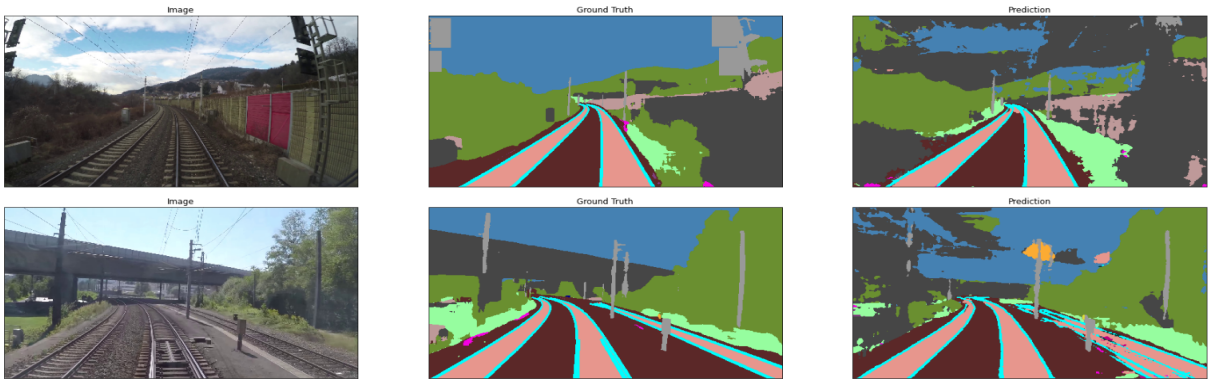


Figure 4.7: SqueezeNAS-MAC-Large (IOU:0.60-0.73)

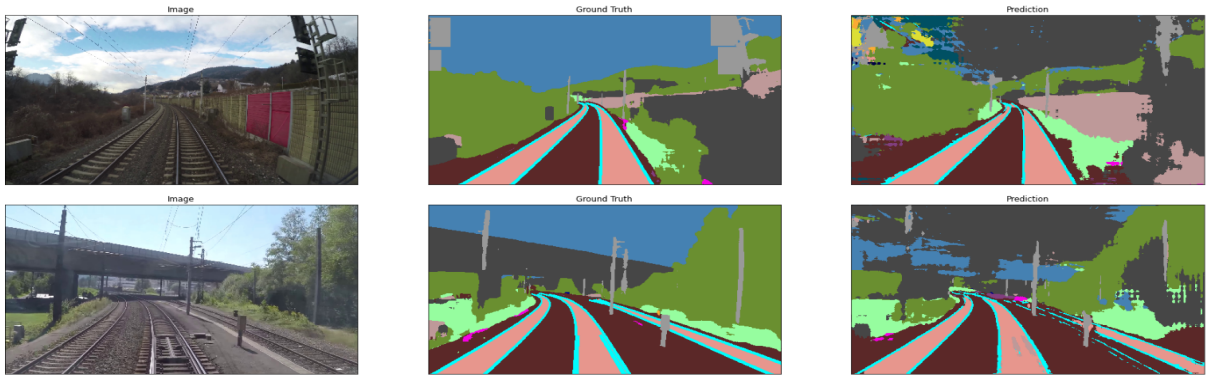


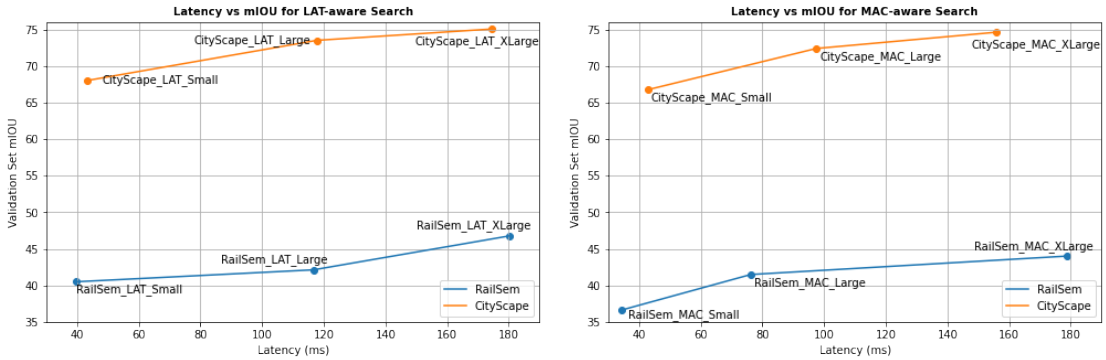
Figure 4.8: SqueezeNAS-MAC-XLarge (IOU:0.58-0.64)

Chapter 5

Conclusion

We have collected all the results from RailSem and CityScape models in Figure 5.1. It can be seen that the models trained with RailSem can be slightly faster than the ones trained with CityScape. Nevertheless, the IOU-scores in the RailSem trained models are significantly lower. The best performing model in the mIOU part would be the SqueezeNAS-Lat-XLarge (Figure 4.5) with 46.76% accuracy, the latency time of 180.18ms as for the same model in CityScapes trained models we get 75.06% accuracy in 174.4ms pro image. However, in the latency section, the best performing model is the SqueezeNAS-MAC-Small, which with 34.36ms is about 26% faster than the original (CityScapes trained) model and has the accuracy of 36.62% mIOU on the test set. The corresponding model has a latency time of 42.94ms with the mIOU score of 66.77% (All the measurements and comparisons can be found in the provided Excel file *Squeeze_NAS_Report.xlsx*).

We notice in our predictions that the rail objects in Figures 4.3 to 4.8 are detected fairly so that it can fulfill part of our goal for this experiment. Having low mIOU values in general, on the other hand, might make them less considerable for our task.



(a) LAT-Optimized models comparison

(b) MAC-Optimized models comparison

Figure 5.1: Our measurement results for Latency vs mIOU

5.1 Future Work

The first thing that might help expand the result would be converting the models in TensorRT so we can compare the inference in that mode as NVIDIA suggests it would enhance the models' performance. The problem that we have faced was with the Onnx engine. As there is no tool for converting a PyTorch model to TensorRT directly, one has to convert the Pytorch model into Onnx type and then from Onnx to TensorRT. Our conversion was successful up to the Onnx models, but from there to TensorRT, we get some errors about the engines responsible for the conversion. The version of the TensorRT used in our experiment was 7.1.0. Secondly, having the RailSem tested on other networks might give us some insight into how decent was our models' performance and if there are any issues with the dataset. Performing the same precedure on the DeepLabV3 models with MobileNetV3Small on RailSem dataset has resulted in 56% accuracy which is slightly better than our results.

Bibliography

[1] A. Shaw, D. Hunter, F. Landola, and S. Sidhu, "Squeezenas: Fast neural architecture search for faster semantic segmentation," in *Proceedings of the IEEE international conference on computer vision workshops*, 2019, pp. 0–0.

[2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[3] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324.

[4] H. Noh, S. Hong, and B. Han, "Learning deconvolution network for semantic segmentation," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[5] M. Cordts, M. Omran, S. Ramos, T. Scharwächter, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset," in *CVPR Workshop on the Future of Datasets in Vision*, vol. 2, 2015.

[6] O. Zendel, M. Murschitz, M. Zeilinger, D. Steininger, S. Abbasi, and C. Beleznai, "Railsem19: A dataset for semantic rail scene understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.