Assignment 2: Applied Deep Learning

Jan Janiszewski, UCNumber:2023198681*
Yorick Scheffler, UCNumber: 2023176104
jasiekj1@gmail.com
yorick.scheffler@student.hpi.com
University of Coimbra
Coimbra, Coimbra, Portugal

ABSTRACT

This report accompanies the Colab-Notebooks that are created to solve the task given in this assignment. This is a practical introduction to work with Autoencoders and object detection.

KEYWORDS

CNN, Neural Networks, Autoencoder, Yolo, Object Detection

1 CREATING & TRAINING AUTOENCODERS (PART1)

General Implementation

In the beginning, we started with preparing the data. For the training and evaluation of the autoencoders, we used again an 80%-20% data split. The different data splits can be obtained via the random_split function pytorch provides. As architecture, we used a convolutional-based architecture in the encoder and decoder with linear layers inside the bottleneck. This part 1 section will be split into 3 parts the first will go through a normal autoencoder then a variational autoencoder will be presented and the third will be a denoising autoencoder. This part will conclude with a comparison of the results of all three approaches.

In table 1 the different hyperparameters that were used for all approaches are compactly shown.

Hyperparameter	LR	Batch size	Epochs	Optimizer
Values	0.001	256	10	Adam

Table 1: Hyperparameters used for all Autoencoders

1.1 Normal Autoencoder

In 1 is the architecture of the model depicted. We used a straightforward architecture with three convolutions on the encoder side. In the bottleneck, two linear layers are used to learn the extracted features in the latent space. Correspondingly to the encoder, there are three transposed convolutions used in the process of recreating the image. ReLU activation functions and batch normalization layers are in between the other layers. They support a smoother learning process. In the end, there is a Sigmoid function used to recreate the captured white-to-black values for the recreated image.

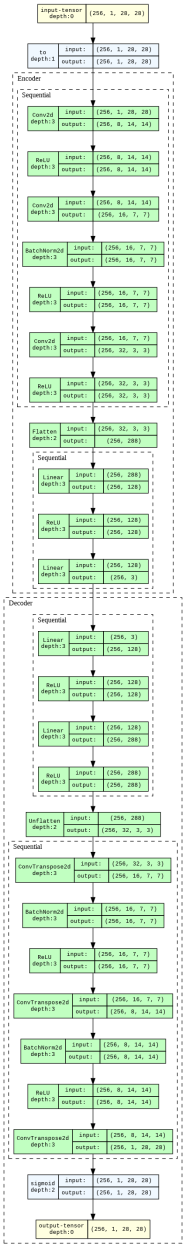

Figure 1: Autoencoder Architecture

*Both authors contributed equally to this research.

The training process is fairly straightforward. That is shown in this plot 2 depicting the loss going down during training.

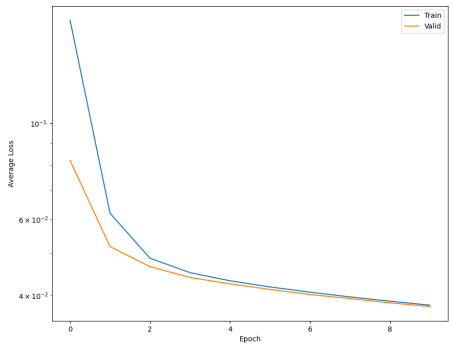

Figure 2: Development of Loss during training

The interesting plots for the evaluation are the visualization of the latent space and the qualitative comparison of the reconstruction pairs. The first one is shown in 3 and the latter in 4. The plot of the latent space shows with which numbers the model had problems differentiating from each other. The biggest problem for the AE is that it struggles with "9" & "4". A possible explanation is that it detects a "circle" in the upper part of the image that isn't present in any other number. This problem can also be seen in the qualitative comparison, where a misclassification and with that a reconstruction error occurred, where the model interpreted a "4" as a "9". In the tSNE visualization is also visible that "3", "5" and "8" are grouped rather close together. It's also visible that some errors occur where the actual "8" is reconstructed as "3", which is indicated by the red dots in the light green cloud of dots in the scatterplot.

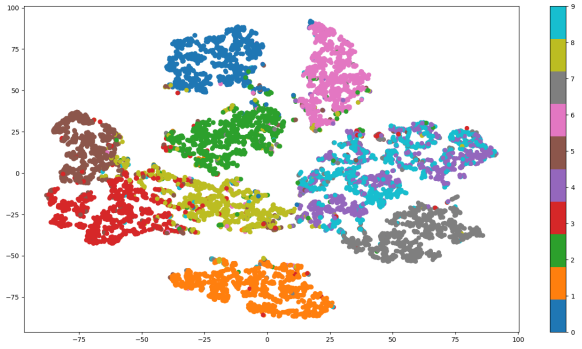

Figure 3: Latent Space AE with TSNE

The last evaluation metric for the AE is the accuracy of our feature extractor. Running evaluation code for it in Colab resulted in an accuracy of 72.05%. A look into the confusion matrix after the encoder layers would probably also show the problem visual in the tSNE plot that a lot of misclassifications are occurring between "9" and "4".

Figure 4: Qualitive comparison between the recreated Images from normal AE and the Input

1.2 Variational Autoencoder

In the beginning, the architecture of the variational AE is again depicted. This is in figure 5. The architecture looks a bit different since a variational AE can fulfill a bit different tasks. An AE of this type learns the representation of its input data in a way that it is also able to create similar new images to the ones it was trained on. If an AE is for a task like this it tries to store the characteristic features of its input data in the latent code. This variable is encoded as "z" in the Colab-code. We think that also explains the a bit strange-looking architecture compared to the normal AE. The encoder just passes the latent code representation to the decoder, which then recreates the image, based on the extracted features.

The loss curve is also not as good as the one during the training of the other AEs. Despite also dropping off significantly in the beginning it starts to remain a bit stagnant after just a few epochs, where just minor increases are achieved. This is shown in figure 6. Maybe there is also an error in the implementation of the variational AE, but after checking the implementation thoroughly, we couldn't identify one.

We couldn't pinpoint a possible error from the other evaluation metrics. First in that regard the visualization of the latent space with the tSNE plot. This is visualized in figure 7. In general, we can see that the clouds of dots of the separate classes aren't as clearly dividable compared to the other AEs. The big problem of not differentiating between "4" and "9" remains. Additionally, it seems not to be able to distinguish "5" and "8".

The qualitative Evaluation of images also shows a bit worse performance compared to the others. This is shown in plot 8. The issue of some misclassifications and because of that false recreations are showing up there. In addition, the output seems to be the noisiest, despite using convolutions with a stride of 2 in all architectures, changing that might also improve our results. However, the improvement should apply to all the architectures kind of equally, because the feature extractor might learn more accurately the features. Evaluating the accuracy like previously for the normal AE shows just an accuracy of 58.91%. This is not totally surprising considering that the latent space seems to have the worst representation for correctly differing classes.

1.3 Denoising Autoencoder

The denoising autoencoder uses the same architecture as the normal autoencoder. In the beginning, we explore again the loss curve that shows development during training. This is shown in figure 9. It shows no significance and looks similar to the normal AE.

It is more interesting to look at the tSNE visualization and the qualitative comparison. They are depicted in figure 10 and figure 11. The tSNE representation is really interesting because even with

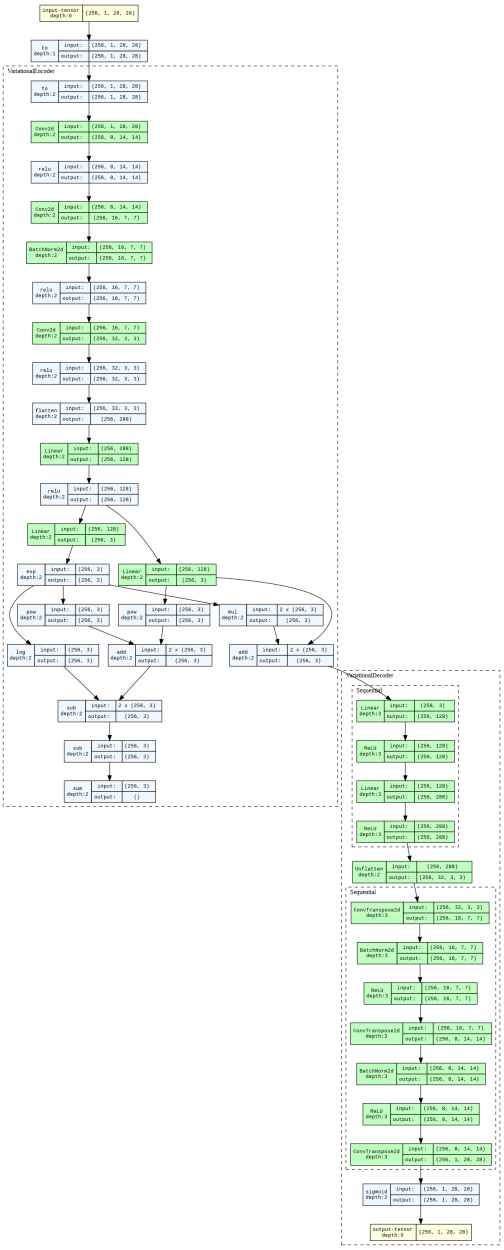

Figure 5: Architecture variational AE

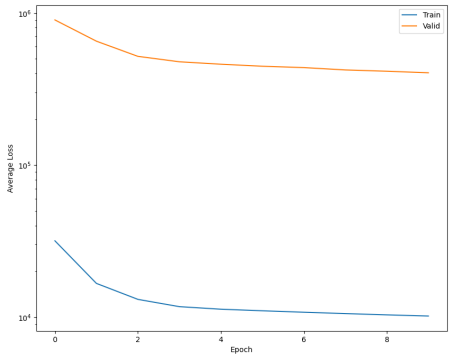

Figure 6: Loss of vAE training

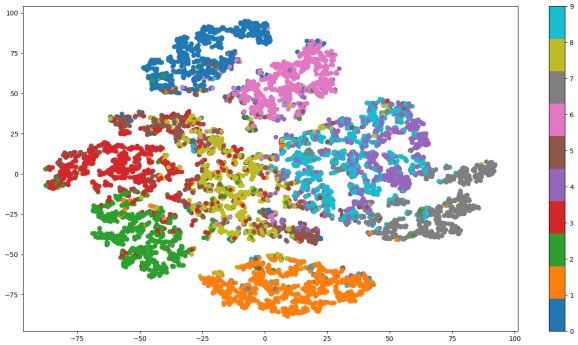

Figure 7: Latent Space vAE with TSNE

Figure 8: Qualitive comparison between the recreated Images from vAE and the Input

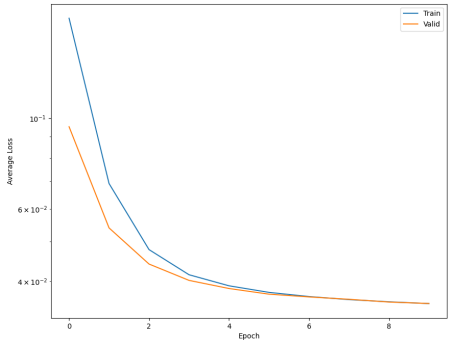

Figure 9: Development of Loss during training of denoising AE

the noise factor the problem remains that it can't differentiate between "4" and "9". The scatter clouds of both numbers are really overlapping each other. A difference is for example where exactly the representations are projected with the help of the tSNE algorithm. The representation of "7" is now above the aforementioned cloud. In the normal AE, it was below it. There are a couple of these occurrences comparing these two tSNE plots. The other problem also seems to remain that it couldn't find a clear cut between light green, red, and brown classes. This problem might be even a little bigger in the denoising AE, but this can not be told for sure from assessing the plot.

The qualitative comparison looks a bit better compared to the normal AE, there seems to be less misclassification between "4" and

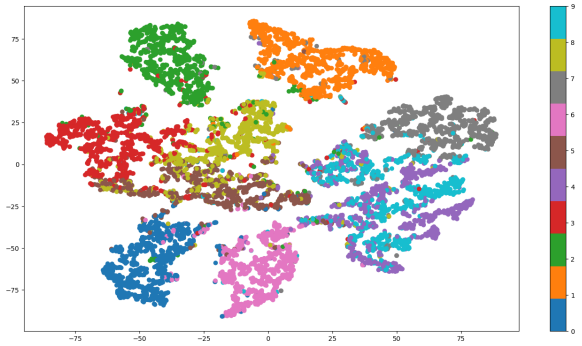

Figure 10: Latent Space denoising AE with TSNE

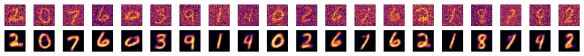

Figure 11: Qualitive comparison between the recreated Images from denoising AE and the Input

"9". This visual assessment can only be seen as an indicator, it could also be just true for this certain subset of the test data. A stronger evidence is the accuracy of the feature extractor. This encoder architecture achieves an accuracy of 75.40%. This is compared to the normal AE just a minimal improvement. So the better-looking qualitative representation seems to have just occurred by chance.

1.4 Summary of Results

To conclude we want to briefly summarize the results from testing with the three AEs. First of all the denoising AE has performed best, that is probably due to the added noise, which forces it to learn more compact representations compared to the normal AE. The normal AE might focus sometimes on details, which could explain the slightly worse performance. For example, the noise added in the distinguishing parts between "8" and "3" could have helped the denoising AE. The variational AE performed worse. This could be due to the idea behind this architecture to focus on generative capabilities in this architecture. These capabilities haven't been tested in the evaluation, because the creation of new samples wasn't a task. This quality an AE of this kind has compared to the other AEs might come with some drawbacks in performance in tasks like this.

2 PART II: OBJECT DETECTION

The next task was to train object detector YOLOv5 network. The training was conducted in two modes:

- from scratch
- transfer learning, using weights provided on YOLOv5's GitHub webpage.

The training was done using a subset of the KITTI dataset. These are traffic images collected with various types of sensors and files with labels and coordinates of the objects present in the images. For the purpose of our training, all objects except vehicles were removed from the label files.

2.1 Network training

The dataset was split into training and testing data in an 80/20 ratio, the split was done at random.

The hyperparameters used when training each network are listed in table 2.

Hyperparameter	value
Number of epochs	50
Learning rate	0.01
Optimizer	SGD

Table 2: Hyperparameters used for training networks

Training mode	mAP50	mAP50-95	Precision	Recall
From scratch	0.568	0.24	0.645	0.524
Transfer learning	0.805	0.499	0.849	0.671

Table 3: Obtained results from different training modes

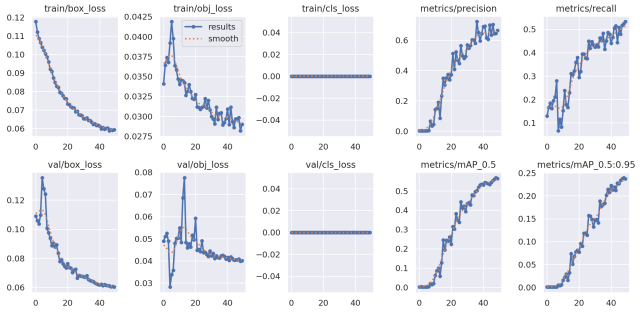

Figure 12: Curves from training from scratch

Figure 13: Example of object detection made my network trained from scratch

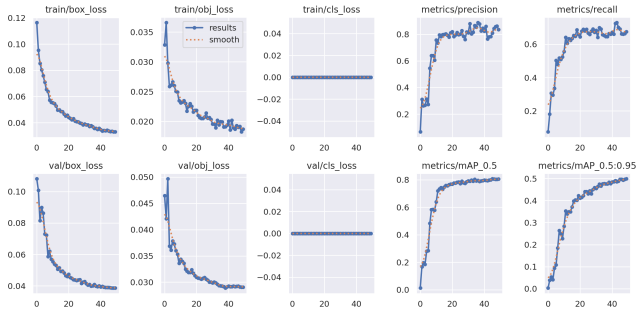

Figure 14: Curves from transfer learning

Figure 15: Example of object detection made my network trained with transfer learning

2.2 Analysis of the results

Comparing the curves in training depicted in 12 and 14 show a few differences. The differences are expected the training from scratch doesn't converge as smoothly as the transfer learning. This is probably because the from-scratch trained model still needs to learn low-level features like edge detection that are already given in the pre-trained one. It is also visible in the metrics depicted in the image like precision or recall that seem to hit an upper ceiling, where the from-scratch trained has a more or less continuous increase. The overall better results that the pre-trained model achieves are probably due to the lower amount of training data that was used in the from-scratch training. As previously pointed out we work here with a subset of the KITTI dataset, where only 100 images are used to train or vice versa fine-tune the corresponding model. The use of a bigger data set could lead to similar performance with both models. In the end, the gap in the metrics is fairly large. It is between 15-25% better in all 4 evaluated metrics.

3 GOOGLE COLAB WITH CODE

All code that was used can be found here: Assignemnt