1. Introduction

In an increasingly technological world where self-driving cars are becoming viable, an important task is accurately detecting the traffic signs. This paper aims at classifying Danish traffic signs, trained on a dataset made up of various traffic signs collected in and around Aarhus, Denmark. In doing this we employ a convolution neural network, CNN, and showcase its capabilities and limitation in accurately classifying Danish traffic signs.

2. Dataset

We had the issue that no proper dataset for Danish traffic signs existed, so we decided to make our own. We did that by collecting different traffic signs and labeling them. This created 22 different classes but with limited diversity because of relatively few images in many of the classes. To try to remedy this problem we augmented the images by changing and adding noise, rotating and brightness. With this we gained ten new images from each one and served in making the dataset more diverse and applicable to real world applications.

This augmentation is done on the training dataset but not on the test one. Our training dataset has 1150 elements while our training dataset has 118.

3. Architecture

The goal of this project was to classify different traffic signs. To do this we decided on a Convolutional Neural Networks, CNN, as they perform well in image classification. We chose to create our own CNN. On figure xxx the different layers of the network can be seen.

Convolutional Layers (Conv2d): Utilizing convolutional filters with different sizes to extract hierarchical features from input traffic sign images.

Max Pooling Layers (MaxPool2d): employed for feature extraction and reducing dimension of data.

Dropout Layers (Dropout): Used to prevent overfitting.

Fully Connected Layers (Linear): Responsible for learning high-level representations from the extracted features.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 56, 56]	1,792
MaxPool2d-2	[-1, 64, 28, 28]	0
Conv2d-3	[-1, 128, 28, 28]	73,856
MaxPool2d-4	[-1, 128, 14, 14]	0
Conv2d-5	[-1, 256, 14, 14]	295,168
Conv2d-6	[-1, 256, 14, 14]	590,080
Conv2d-7	[-1, 128, 14, 14]	295,040
MaxPool2d-8	[-1, 128, 7, 7]	0
Dropout-9	[-1, 6272]	0
Linear-10	[-1, 512]	3,211,776
Dropout-11	[-1, 512]	0
Linear-12	[-1, 256]	131,328
Linear-13	[-1, 22]	5,654

Total params: 4,604,694
Trainable params: 4,604,694
Non-trainable params: 0

Input size (MB): 0.14
Forward/backward pass size (MB): 3.93
Params size (MB): 17.57
Estimated Total Size (MB): 21.64

Figure 1: Architecture of the custom the CNN

The designed CNN architecture presents a combination of convolutional, pooling, and fully connected layers. To link these layers together an activation function is necessary. We have chosen to use the RELU due to its relative ease in computation but also its good performance. The inclusion of the RELU activation function is to ensure that the model can capture non-linearity in the data. To be able to use our data we must resize our image to have the same data size. We resize it to 112 by 112 and is also why the first layer has the shape it has. The last layer has a shape of 22, because we have 22 different classes. this leads to a network of 4,604,694 trainable parameters. The actual layout of layers in our CNN can be seen in figure 1.

4. Training and validation

Before training and validation, we split the existing dataset into training and validation sets. We chose an 80-20 split because it strikes a nice balance between having sufficient data for training and prevents overfitting. This split of data for each class can be seen in figure 2.

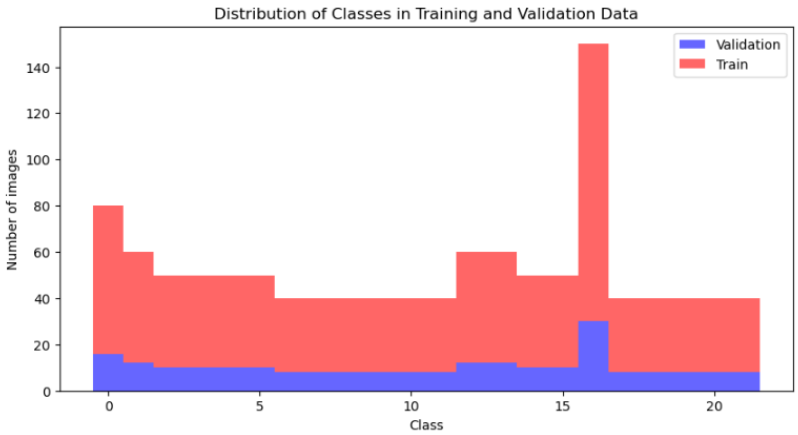


Figure 2: Distribution between validation and training data size

We also had to decide on the Adam optimizer and found that a learning rate of 0.001 and 40 epochs lead to good performance. We settled on these values by manual tuning and finding what performed well. For the loss function we used the cross-entropy loss function. Training the network with these parameters allows us to plot the loss and accuracy of each epoch iteration. A case could be made that less than 40 epochs is needed for accurately learning the features associated with each label. However, we must be aware of potential risks with overfitting but by evaluating that the training and validation loss is decreasing it is deemed to be acceptable.

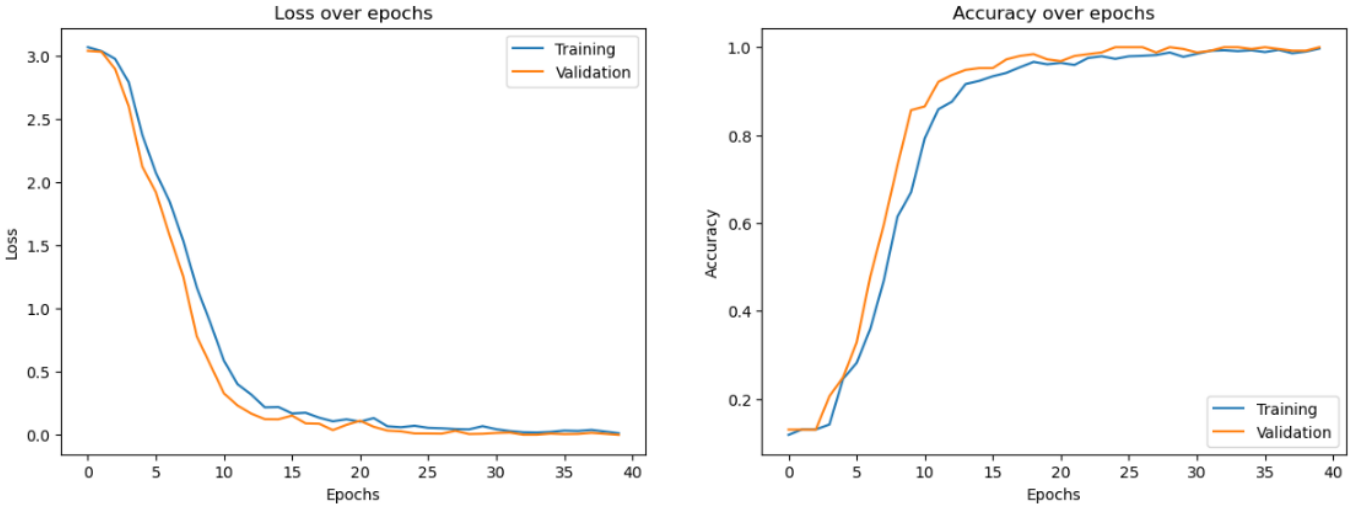
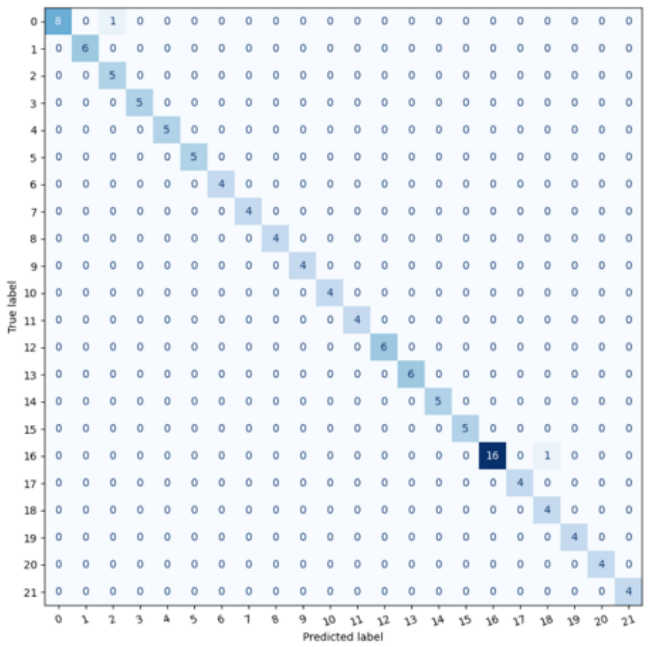


Figure 3: Loss and accuracy for plot for training of the CNN.

5. Results

After having trained the network, we can test whether the model can correctly determine the label of certain unseen images. If we have trained it correctly it should be able to label an image that it hasn't seen before, based on the features, it has learned during training. To check the relationship between the true label and the predicted label we create a 22 by 22 confusion matrix shown in figure 4. The confusion matrix shows that the networks labels most of the images correctly even though there are mislabeling. This happens in class 0 and 16 where the predicted label is class 2 and 18 respectively. This corresponds to an accuracy of 98%.



	precision	recall	f1-score	support
0	1.00	0.89	0.94	9
1	1.00	1.00	1.00	6
2	0.83	1.00	0.91	5
3	1.00	1.00	1.00	5
4	1.00	1.00	1.00	5
5	1.00	1.00	1.00	5
6	1.00	1.00	1.00	4
7	1.00	1.00	1.00	4
8	1.00	1.00	1.00	4
9	1.00	1.00	1.00	4
10	1.00	1.00	1.00	4
11	1.00	1.00	1.00	4
12	1.00	1.00	1.00	6
13	1.00	1.00	1.00	6
14	1.00	1.00	1.00	5
15	1.00	1.00	1.00	5
16	1.00	0.94	0.97	17
17	1.00	1.00	1.00	4
18	0.80	1.00	0.89	4
19	1.00	1.00	1.00	4
20	1.00	1.00	1.00	4
21	1.00	1.00	1.00	4
accuracy			0.98	118
macro avg	0.98	0.99	0.99	118
weighted avg	0.99	0.98	0.98	118

Figure 4: confusion matrix and accuracy table for the test set

This accuracy corresponds well to the fact that our accuracy in training reached around the same level. This indicates a network that has accurately learned the features corresponding to each label and hasn't been overfitting.

6. Discussion

While we can classify correctly in 98% of the time on our test-set, we must be aware of potential limitations. We were only able to reach this accuracy by cropping the images in order only to show the actual traffic sign. This is however not the most applicable to the case of a self-driving car, where this sort of cropping won't be done on each picture before classification. If we were to include the surroundings of the images, we don't have a dataset large enough to accurately estimate the features responsible for each class.

The actual performance of the CNN can be seen in figure 5 where ten random images has been picked form the test dataset and showed with their actual class and the predicted class by the CNN. From this it is evident that it can distinguish the classes.

Figure 5: ten randomly selected images from the test set with their actual and predicted class.

Despite the traditional high accuracy of 98%, we must discuss whether this is sufficient in an application as sensitive as that of self-driving car where a speed limit of let's say 60 km/h should never be classified as say 80 km/h.

7. Conclusion

Through training and validation using the 80-20 split and optimizing with the Adam optimizer and cross-entropy loss function, our network achieved 98% accuracy in both training and testing phases. While this high accuracy signifies successful learning of features associated with each label without overfitting, it's essential to recognize potential limitations.

The model's reliance on cropped images may limit its practicality usability, especially in the context of self-driving cars that require a broader context beyond the traffic sign itself. Additionally, the dataset's size might does not adequately capture diverse environmental features affecting sign recognition accuracy and therefore limits the usability of the network to only work on cropped images.

Moreover, despite the accuracy concerns do arrive depending on the application. For instance, in the use case of autonomous cars, the misclassification possesses a safety risk.

In conclusion, while achieving a 98% classification accuracy demonstrates the CNN's ability to identify Danish traffic signs, its practical application in the context such as autonomous cars demand on further considerations.

View the code at: https://github.com/AskeBredmose/trafic-sign-classifier. This is code and data gathered and made by the entire group.