Comparing the Differences in Performance between Different Approaches for Convolutional Neural Network Parameter Tuning for Road Sign Classification

To what extent Convolutional Neural Network performance in terms of prediction accuracy is influenced by pooling filter size, dropout rate and the learning rate of the network for road sign classification?

Computer Science Extended Essay

Word count: 3987

Table of contents

1. Introduction

Machine learning plays an important role in extracting knowledge from huge quantities of data by constructing a mathematical model from the used dataset by ‘learning“. This approach in data mining allows a specific algorithm to learn from the given data by itself without the help of a programmer. The functionality of such an algorithm relies on building a model based on the inputs in the training phase to be used in predicting new rules for the next decisions (Smola). Computer vision is the field of study surrounding how computers see and understand digital images and videos. Computer vision spans all tasks performed by biological vision systems, including "seeing" or sensing a visual stimulus, understanding what is being seen, and extracting complex information into a form that can be used in other processes (DeepAI). Currently, the field of image classification is widely used in various fields such as the automotive industry, healthcare, retail, security, robotics and more (Choudhury). There also is constant research for computer vision algorithms. Large companies such as Google, Tesla and up and coming startups (Zoox, Optimus Ride) are huge players in image recognition research (Patel).

In order for computers to apply computer image classification algorithms accurately in real-world applications, algorithms have to analyse large amounts of data beforehand which requires a lot of computational resources. There are several factors that affect the time it takes to analyse given datasets for specific algorithms. These parameters also influence the ability to recognize and classify similar images that were not seen by the network model before.

This paper focuses on exploring and comparing the ways in how Convolutional Neural Networks could be tuned for computer vision applications for classifying different road signs. The aim of this paper is to identify the best hyperparameter values (dropout rate, learning rate, max-pooling filter size) for tuning CNNs for road sign classification in regards to the accuracy of the model.

The results of this research might help to better understand the general hyperparameter tuning trends which could be used for the fine-tuning of the CNNs, not only for road sign classification but also for other classification tasks. In addition, the results of this paper might give better insights into one of the components for creating a vision-based autonomous navigation system.

To investigate the relationship between the accuracy of the CNNs and hyperparameters (learning rate, pooling filter size, dropout rate), Convolutional Neural Networks training algorithms will be programmed and trained on German Traffic Sign Recognition Benchmark (Stallkamp, Schilpsing, et al.) with different combinations of all three learning parameters. Later these networks will analyse prelabeled road sign images which haven't been seen by the networks in the training phase (validation data). The results of the CNNs prediction performances with different parameter combinations will be analysed.

2. Background Information

2.1 Computer vision and Deep learning

Computer Vision is a field of study that enables computers and systems to derive significant information from digital images, videos or other visual inputs (IBM). It can also be described as a way to find features of the objects to classify them (Thilakarathne). Computer Vision uses a variety of techniques including Deep Learning and traditional Computer Vision algorithms. Such traditional algorithms as SIFT (Scale-Invariant), SURF (Speeded-Up Robust Features), BRIEF (Binary Robust Independent Elementary Features) are some of the most popular approaches for feature extraction. However, this traditional approach of feature extraction in image classification comes with a difficulty - it requires a manual selection of features for each given image. Meaning that when the number of classes is high or the clarity of the image is low, it is unreasonably difficult to adopt traditional computer vision algorithms (Thilakarathne).

Deep learning methods used for computer vision are achieving not only better performance benchmarks but also eliminating the need to manually engineer specialised methods as a single deep learning model can learn the meaning from the images and perform required vision tasks (Brownlee). Deep learning has been used to solve various computer vision problems (Image Classification, Object Detection / Segmentation, Image Style Transfer / Colorization) and much more.

2.2 Convolutional Neural Networks (CNN)

A Convolutional Neural Network, also known as CNN or ConvNet, is one of the most popular algorithms for deep learning. Like other networks, CNN is composed of an input layer and many hidden layers (Matlab). Modern CNNs owe their design inspiration from biology, group theory and a dose of experimental tinkering (Zhang, Lipton, et al.). In addition to their efficiency in achieving accurate models, CNNs tend to be computationally efficient due to the fact that they require fewer parameters than fully-connected architectures, and their convolutions can be easily parallelized across GPU cores (Zhang, Lipton, et al.). CNNs are a type of feedforward neural network that is created to process data that is in the form of an array with a degree of spatial structure (Credi). An example of that could be an RGB image that consists of three 2D arrays (each 2D array for the colour channel, *Figure 1*), which could also be described as a 3D tensor.

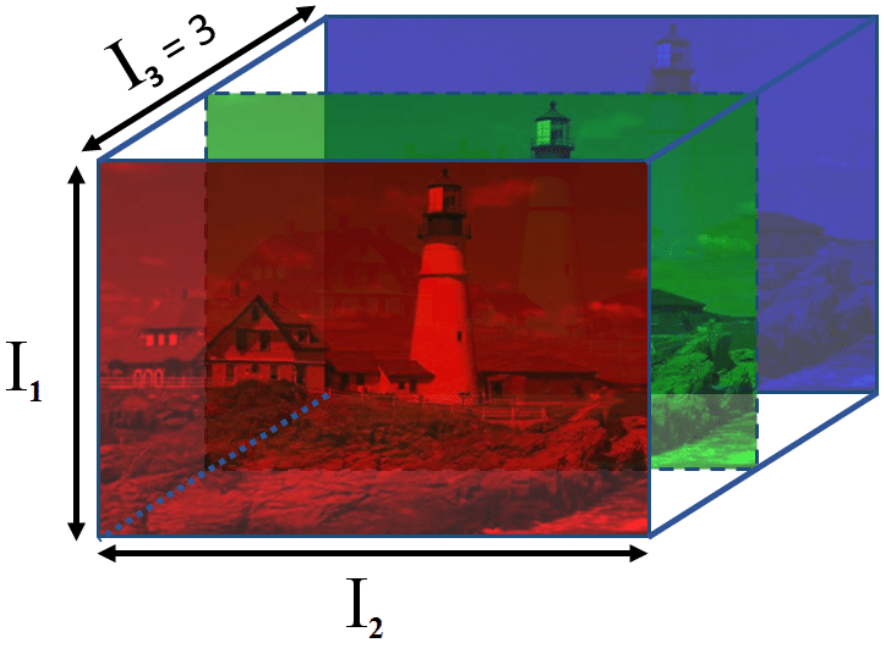


Figure 1: 3rd-order Tensor representation of a colour image (Qureshi)

Generally, tensors give the means to define n-dimensional arrays that have any number of dimensions. An example of a tensor is given below.

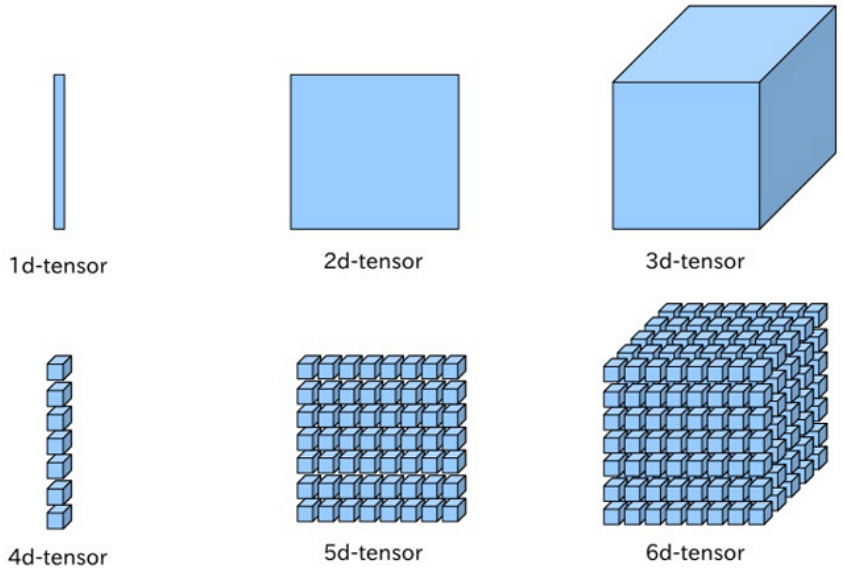


Figure 2: N-Dimensional Tensor representation (Ansari)

Usually, CNN architecture consists of several essential layer types including Convolutions, Pooling, Padding, Flatteting and others (Medmain). These layers will be discussed below.

2.2.1 Convolutions

This is the main component of a CNN. Convolution is a mathematical operation that multiplies a set of weights (kernel) with the input tensor (Lendave). An image kernel is a small weight matrix that "extracts" features by applying certain weights to the input image (Powell). This convolution operation is shown in the figure below.

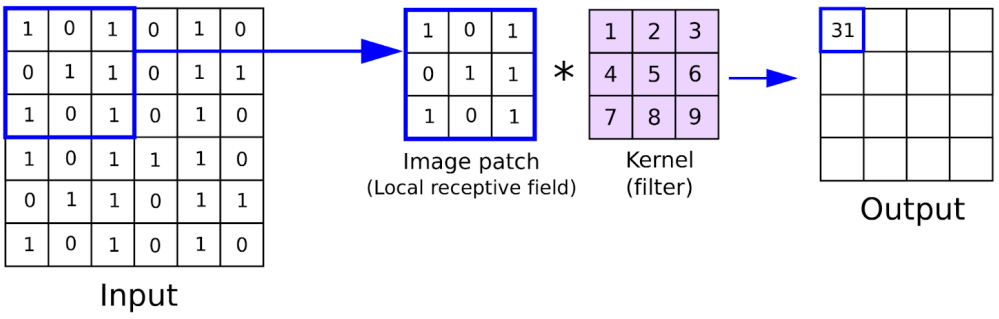


Figure 3: Graphical representation of Convolution (Reynolds)

This way the kernel “slides” through a whole input tensor (an image), multiplies every patch and outputs a new tensor. Stride is a parameter that “slides” the image path by the number of the steps that it is set to (Lendave).

2.2.2 Pooling

Pooling is a technique used to generalise the presence of certain features in the input image. This method is used to simplify the network as much of the input data is superfluous. The figure below shows what the pooling method does.

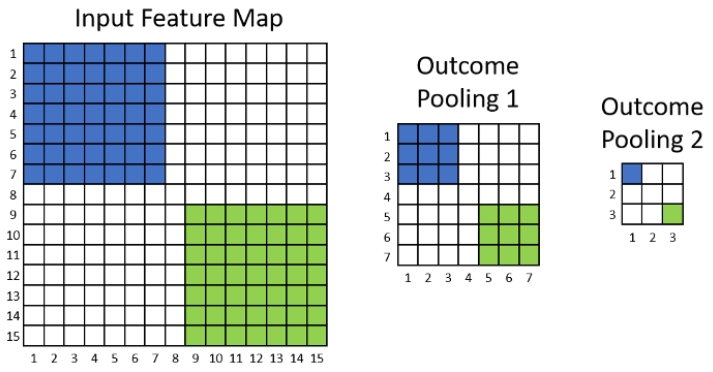


Figure 4: A visual representation of Pooling operation (Sousa)

This operation outputs a feature map that is smaller in size compared to the original input image. There are several methods to perform this subsampling operation (Basavarajaiah).

The *Max Polling* method selects the pixel with the maximum value from the filter that goes through the input image, whereas the *Average Pooling* technique calculates the average value and outputs it to the feature map. The difference between the methods is shown in the figure below:

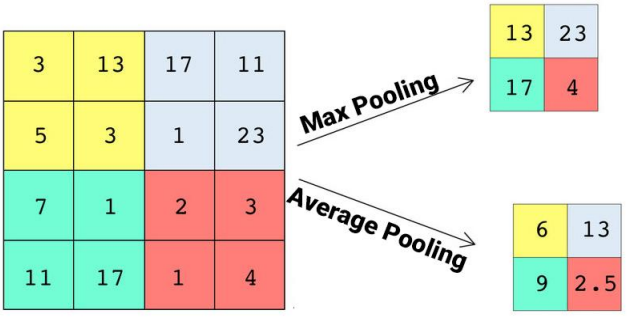


Figure 5: An example of *Max Pooling* and *Average Pooling* operations. Adapted from (Aljaafari)

2.2.3 Padding and Stride

However in some cases, it is important to use techniques that do not affect the size of the tensors too much. This is necessary because in most cases, kernels have width and height greater than 1, meaning that after applying many convolutions on the primary input, there is a tendency to get outputs that are considerably smaller than the input. For example, if we start with an image that is 240 pixels by 240 pixels, 10 layers of 5x5 convolutions reduce the image by 30 %, meaning that some relevant information from the original image might be lost, this is one of the most popular techniques to handle this issue (Zhang, Lipton, et al.).

2.2.4 Padding

When applying many convolutional layers, generally pixels are lost at the perimeter of the image. Even though usually small kernels are used, after many convolutions this loss of pixels can add up. This problem can be fixed by adding extra pixels of filler around the boundary of the input image, this way increasing the size of the image.

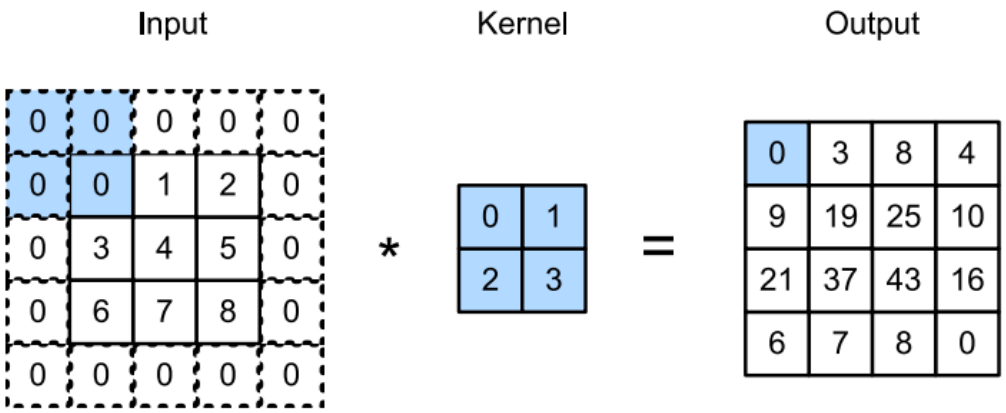


Figure 6: Two-dimensional cross-correlation with padding (Zhang, Lipton et al.)

2.2.5 Flattening

After the *Pooling* operation the relevant features were extracted from the original image. The Flattening process is needed to convert a matrix into a one-dimensional vector so that it could be processed by an Artificial Neural Network (ANN). The figure below shows a visual representation of the Flattening procedure.

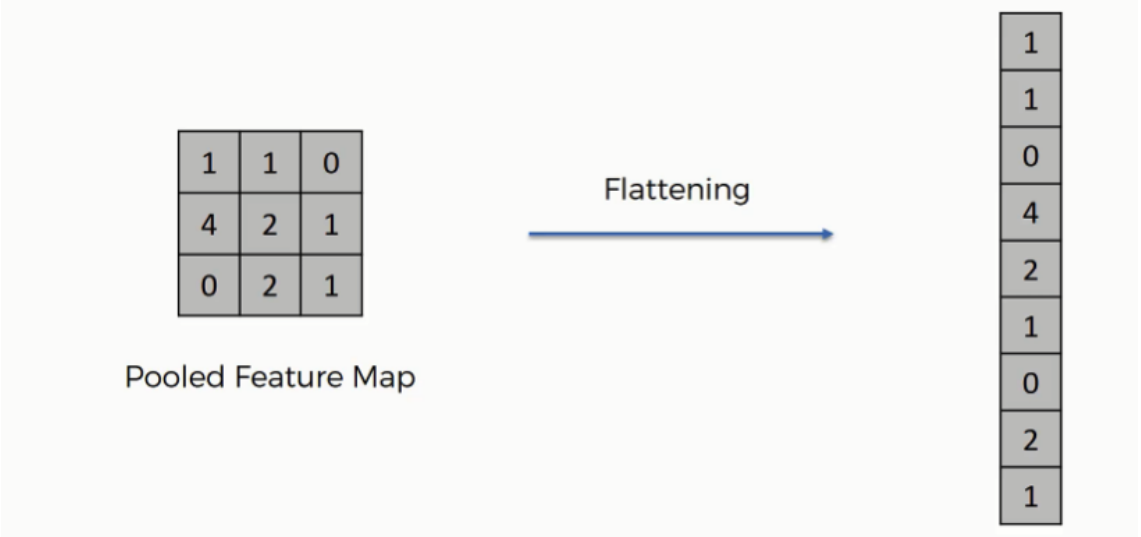


Figure 7: Graphical representation of Flattening operation (Medmain)

2.3 Artificial Neural Network (ANN)

Previously flattened feature maps will be forwarded to the last layer of the CNN, which is a simple ANN classifier. An example of neural network architecture is shown in the figure below.

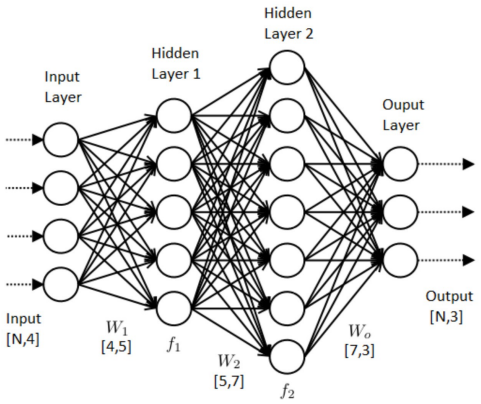


Figure 8: Graphical representation of an Artificial Neural Network (Pankajray)

A neural network consists of input, hidden and output layers of neurons (Figure above), each artificial neuron represents a mathematical function (Simplilearn). Firstly a neuron adds the value of every neuron from the previous layer and multiplies it by *weigh* variable. Each connection between neurons has its own separate *weight* variable. These values will be modified during the training phase (Arnx). In addition, a *bias* variable can be added to the total value that was calculated in the previous step, the *bias* variable can also be modified during the training phase (Gebel). After the summation, an activation function is applied to get the value between 0 and 1, usually the sigmoid function is used (Arnx). The last layer of neurons represents the output layer where each neuron represents a probability that it has of being a correct prediction.

2.3.1 Overfitting

Overfitting is one problem that can occur when training the neural network. When the accuracy of the training dataset for CNN model training is greater than the accuracy of the testing data - the data that was not seen by the network before (Admin). An example of neural network training history that suffers from overfitting is shown in the figure below.

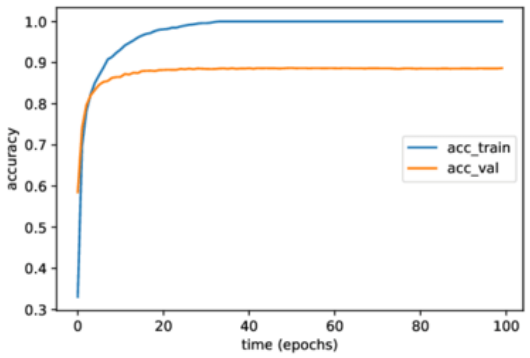


Figure 9: CNN training accuracy graph representing Overfitting (Salman)

The figure demonstrates how during the network training *Training Accuracy (yellow line)* is consistently higher than the *Validation Accuracy (blue line)*. This does indicate that the model that is being trained is too complex for the problem that it is solving (Admin). Meaning that the model has too many features. By having too many features the model learns the training data so well that the performance significantly drops when the model tries to classify new data. To better visualise overfitting, a simple classification problem is shown in the figure below.

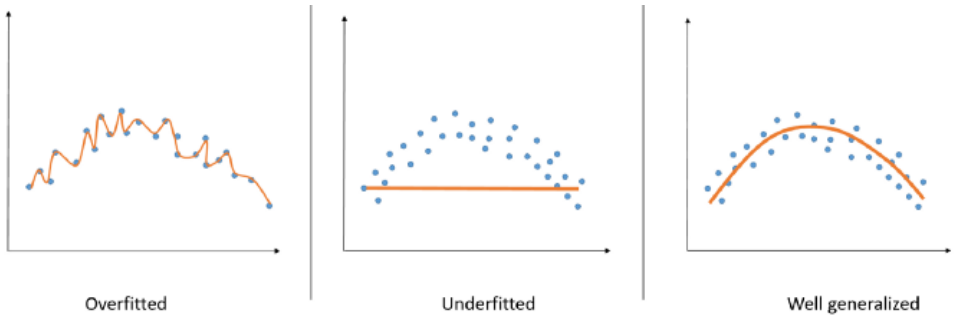


Figure 10: A graphical representation of Overfitted, Underfitted and adequate model (Alto)

There are several ways overfitting could be handled. A simple, but computationally expensive approach would be to use a bigger dataset for model training, however, this approach is not always feasible.

2.3.2 Data Augmentation

Another technique used to mitigate overfitting is Data Augmentation. With this technique, dataset size can be artificially increased by modifying original data in various ways without the need to create a bigger dataset with manual training data gathering. Some of the most popular data augmentation techniques are shown in the figure below (Chernytska).

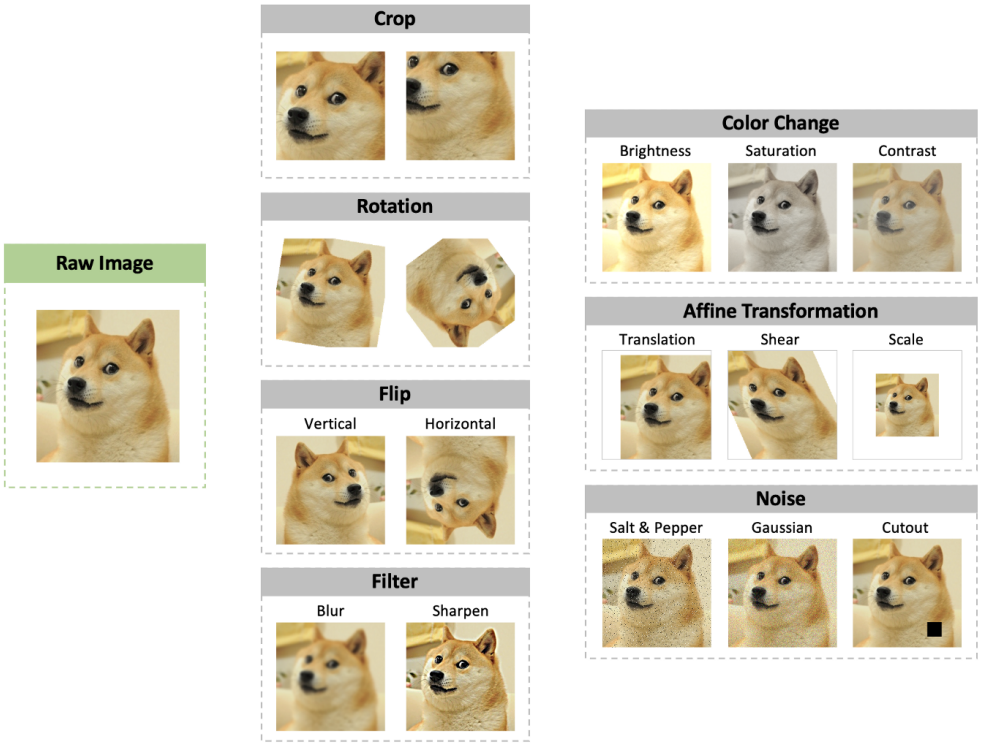


Figure 11: Types of most popular Image Augmentation techniques (Chernytska)

2.4 Hyperparameters

Overfitting can also be mitigated by hyperparameter tuning. There are lots of parameters that could be tuned to mitigate overfitting e.g. *Regularisation*, *Weight Initialization*, *Dropout Regularisation* and others (Admin). However, it will not be possible to explore all of them due to the scope of this paper. However, there will be several hyperparameters that will be discussed below.

2.4.1 Dropout Regularisation

Dropout is a technique that randomly drops units from the neural network during the training stage (Srivastava et al.). The figure below shows the node dropout during the training phase.

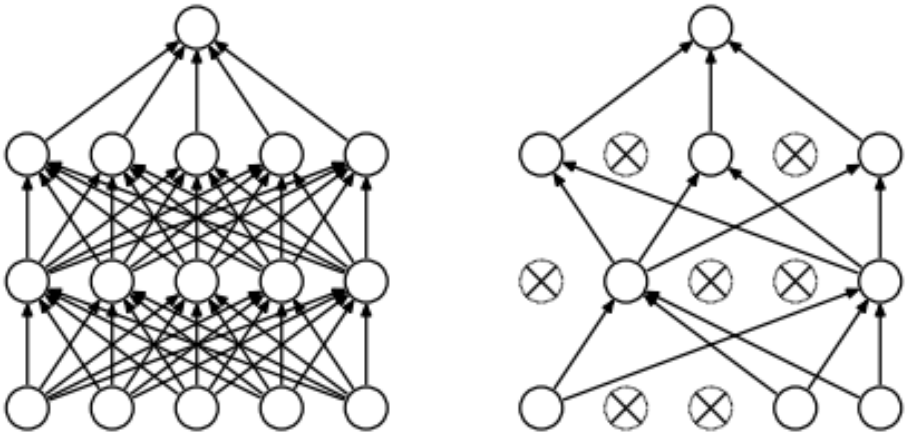


Figure 12: Left: A standard Neural Network with 2 hidden layers. Right: A thinned Neural Network produced by applying Dropout on the Neural Network on the left (Srivastava et al.)

With this technique, unique neurons learn not to rely too much on other neurons and learn something *meaningful* themselves (Minhas). Dropout can be applied after pooling, convolutions or fully connected layers.

2.4.2 Learning rate

The learning rate characterises how fast the network updates its parameters. The CNN architect has to find the balance between fast and slow learning rates, as a low learning rate slows down the learning process drastically but usually converges smoothly whereas a larger learning rate significantly increases the learning speed but is at risk of not converging (Radhakrishnan).

This is shown in the figure below:

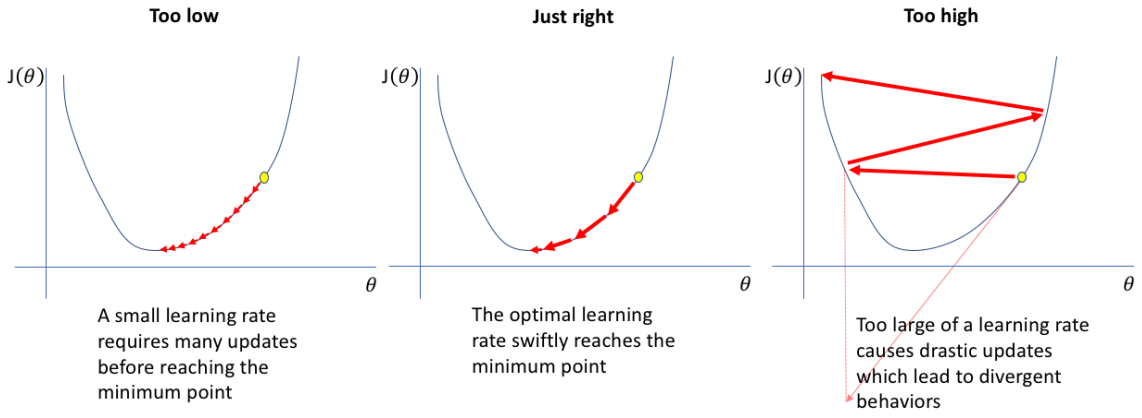


Figure 13: Visual representation of different learning rates used for the model (Jordan)

2.5 Activation functions

2.5.1 Rectified linear unit (ReLU)

ReLU is an activation function that provides a simple nonlinear transformation. It can be defined as

$$ReLU(x) = max(x, 0).$$

Given the input x, it is defined as the maximum of the x and 0. The graph below shows the ReLU function graphically.

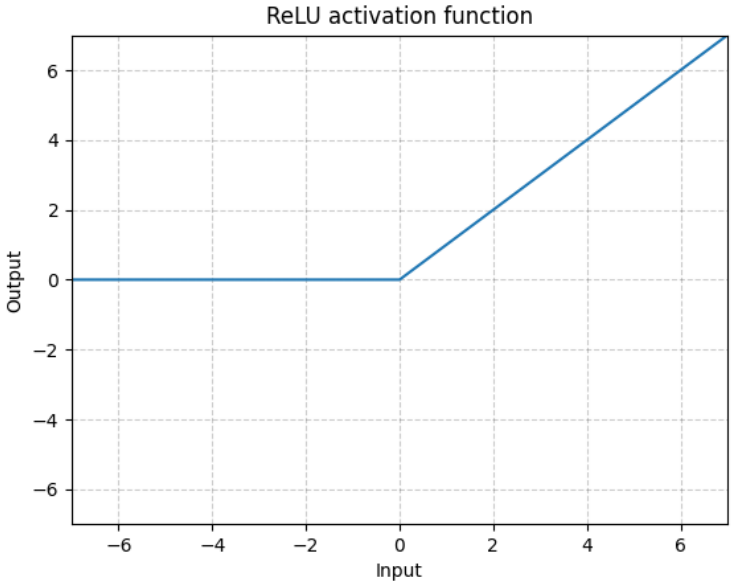


Figure 14: A graphical representation of ReLU function

The ReLU function is used for Deep Neural network training (CNNs) as it is not possible to use sigmoid-like activation functions as they are too computationally expensive without any additional benefit when dealing with huge CNNs (Thakur).

3. Methodology of experiment

3.1 Dataset

German Traffic Sign Recognition Benchmark (GTSRB) is a public dataset (Stallkamp, Schilpsing, et al.) that consists of more than 50000 thousand images of 43 roadsign classes used in Germany (*Figure below*). The dataset consists of 32x32 size 3 channel images. The dataset is split into two batches, the first part of the dataset images are dedicated to training the CNN and the remaining images are dedicated to validating the accuracy of the trained model.

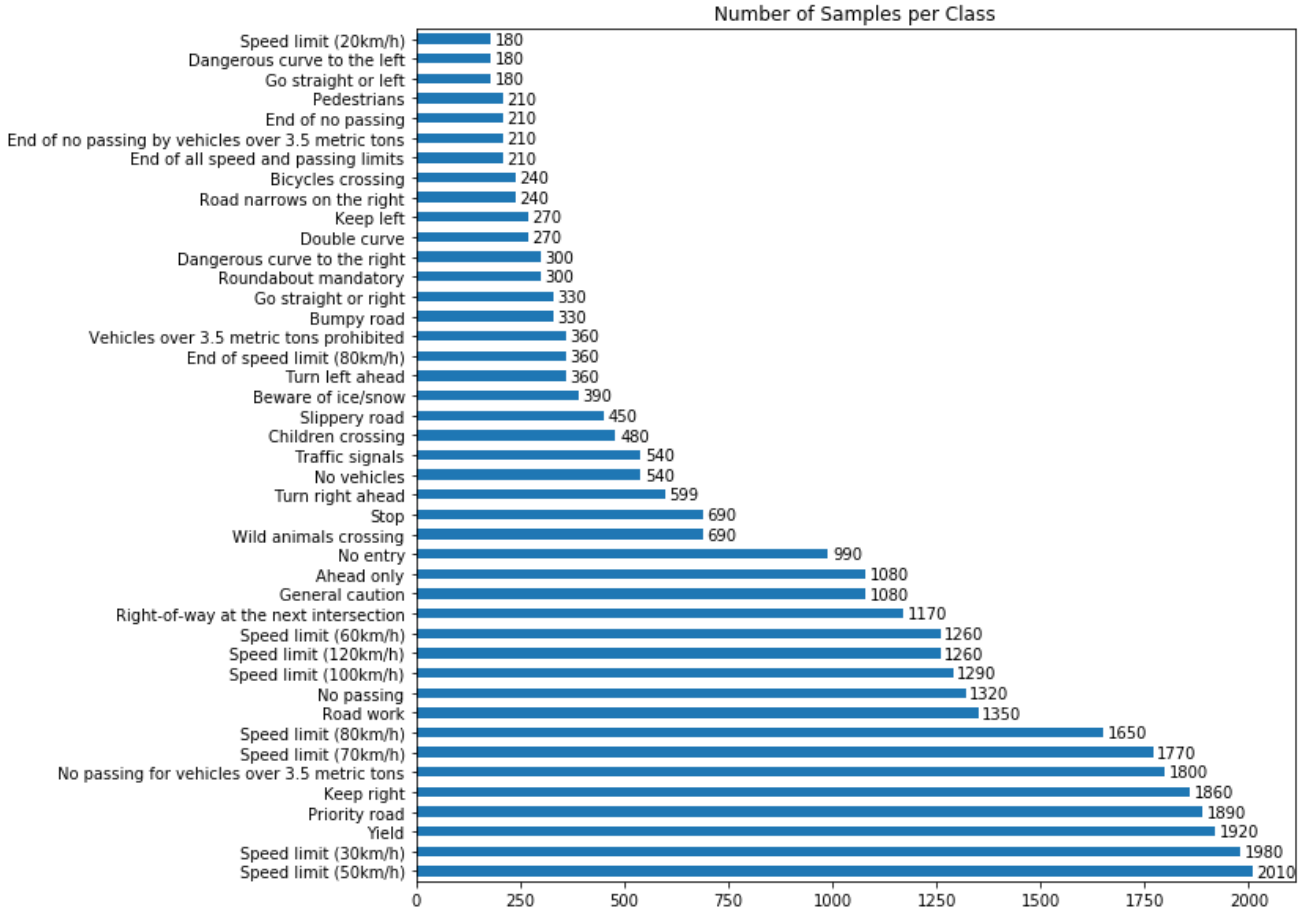


Figure 15: A graphical representation of GTSRB dataset with Number of Samples per Class (Tracey)

To have more evenly distributed different classes of the dataset, and increase dataset size in general to overcome possible overfitting some Data Augmentation techniques described in 2.2.5 (increasing motion blur, noise, augmenting the angle of the image, changing colour and etc.) will be used to modify each class and add those augmented images to the training dataset.

3.2 Variables

3.2.1 Independent variables

The variables in the table below will be altered for training different CNNs. The specifics of these parameters were discussed in sections 2.2.2 and 2.4.1-2.

Variable (Parameter)	Parameters used in experiment
Dropout rate	0.0, 0.2, 0.5, 0.8
Learning rate	0.01, 0.005, 0.001, 0.0001
Max Pool filter size	2x2, 3x3

Table 1: The parameters used in the model training.

The most commonly used dropout rates are between 0.5 and 0.8 (Brownlee). However, this research will also include a very small dropout rate of 0.2 and a network without any probability of dropout (0.0) to find any significant performance changes.

One of the default *learning rates* used to train CNNs is 0.01 (Brownlee), however, a network with such a relatively high *learning rate* might be at risk of not converging at all. Thus, smaller *learning rates* were also selected.

In addition to the default *Pooling filter size* used in the model that was adapted from Lascelle, the Poolin filter size was increased to 3x3 to reduce the required computation.

3.2.2 Dependent variables

Accuracy of the model

The main parameter to be optimised is the accuracy of the CNNs predictions. The accuracy value of each neural network is obtained by comparing the predictions of the neural network with test data, which wasn't seen by the network before. The accuracy is calculated by dividing the number of all correct predictions by the total predictions of the network. In addition, the networks will be evaluated by comparing the accuracy of the training history data and its improvements by each epoch.

Time

Since it is difficult to measure how other processes are influencing the computational capabilities of the machine, the training time of the neural networks will be discussed with caution.

3.3 Experiment

3.3.1 CNN Setup

Firstly, a CNN architecture that was heavily adapted from Staravoitau and Lascelle was implemented using *Python* with *TensorFlow* and *Keras* libraries. The *diagram* below shows an abstract architecture model that is similar to the one used in the experiment.

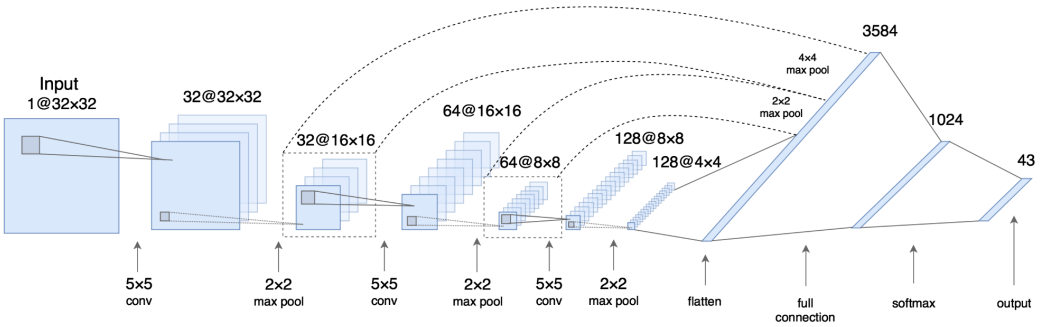


Figure 16: A visual representation of CNN architecture similar to the one used in the experiment. Adapted from (Staravoitau)

This initial network uses *Max Pooling* filters that are of size 2x2, in addition to this CNN, another CNN will be generated with the size of Max Pooling filters to the size of 3x3. By varying parameters shown in section 3.2.1 (Dependent variables), 32 CNNs will be trained. To ensure that results were not accidental, the same network architecture with the same parameters was retrained. In total 64 CNN models were trained.

3.3.2 Hardware used for training

Usually, CNNs are trained on powerful GPUs as GPUs can perform multiple simultaneous computations this way, drastically reducing the time needed to train the models. However, the author of this paper did not have access to a GPU with enabled CUDA capabilities, thus the networks were trained on i7-8550U.

3.3.3 Experiment procedure

The experiment was carried out in the following manner:

1. A CNN with Pool filter size 2x2 was selected as a network
2. The networks were trained with all of the following Dropout (0.0, 0.2, 0.5, 0.8) and Learning rate (0.01, 0.005, 0.001, 0,0001) combinations. In total 16 trained CNN models.
3. The experiment was repeated and all of the models were retrained.
4. A CNN with a Pool Filter size of 3x3 was selected as a network.
5. Steps 2 and 3 were repeated with a CNN with a 3x3 Pool filter size

4. Experiment results

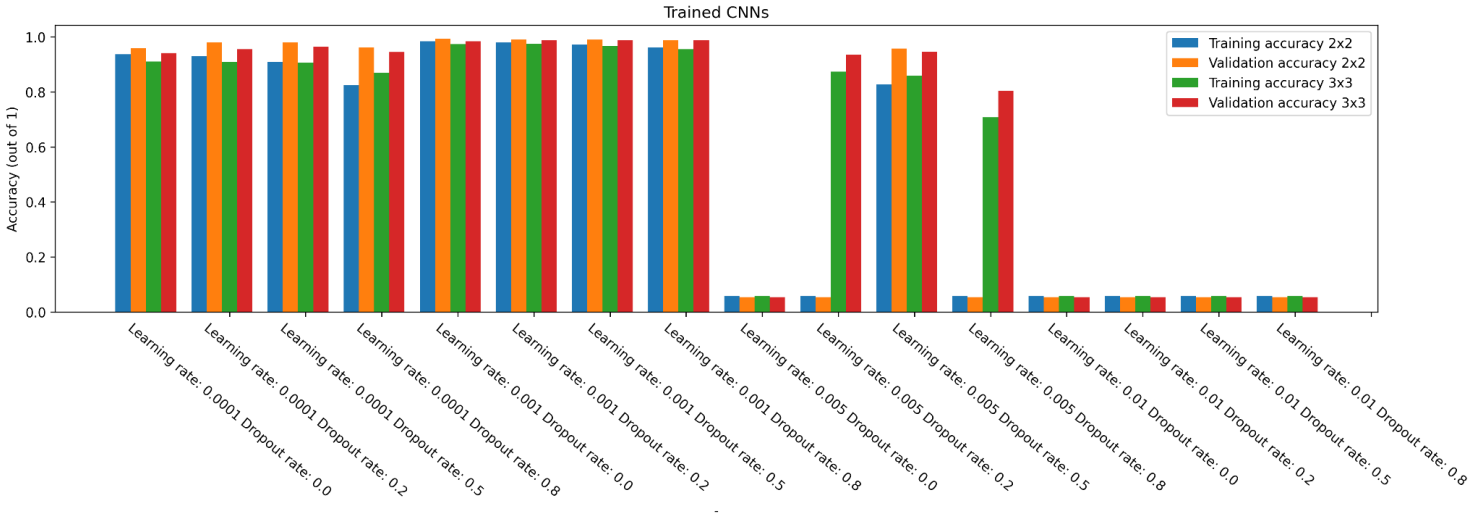
4.1 Data analysis

Every neural network was trained two times with the same parameters and random seed to ensure that the results were not accidental or significantly impacted by external factors.

The general trends of accuracy results of the models with the same parameters remained consistent.

4.1.1 Learning rate analysis

In all of the configurations (with different Dropout rates and Pool filter sizes) where the *learning rate* variable was set to 0.01 the neural networks did not converge in the 16 training epochs. This general trend can be seen in the graph below and *table 2*.

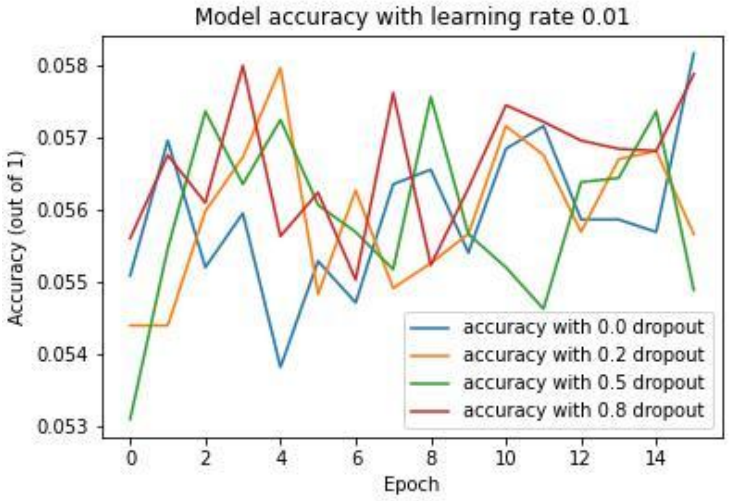


Graph 1: Trained CNNs' results

		Pooling filter size 2x2		Pooling filter size 3x3	
Learning rate	Dropout rate	Training Accuracy (Out of 1)	Validation Accuracy (Out of 1)	Training Accuracy (Out of 1)	Validation Accuracy (Out of 1)
0.0001	0.0	0.937	0.959	0.91	0.941
	0.2	0.93	0.98	0.909	0.955
	0.5	0.909	0.98	0.907	0.964
	0.8	0.824	0.961	0.87	0.945
0.001	0.0	0.984	0.993	0.974	0.984
	0.2	0.98	0.991	0.975	0.988
	0.5	0.973	0.99	0.967	0.987
	0.8	0.962	0.989	0.955	0.987
0.005	0.0	0.057	0.054	0.058	0.054
	0.2	0.057	0.054	0.874	0.936
	0.5	0.827	0.957	0.859	0.946
	0.8	0.058	0.054	0.708	0.805
0.01	0.0	0.058	0.054	0.058	0.054
	0.2	0.058	0.054	0.057	0.054
	0.5	0.058	0.054	0.057	0.054
	0.8	0.058	0.054	0.057	0.054

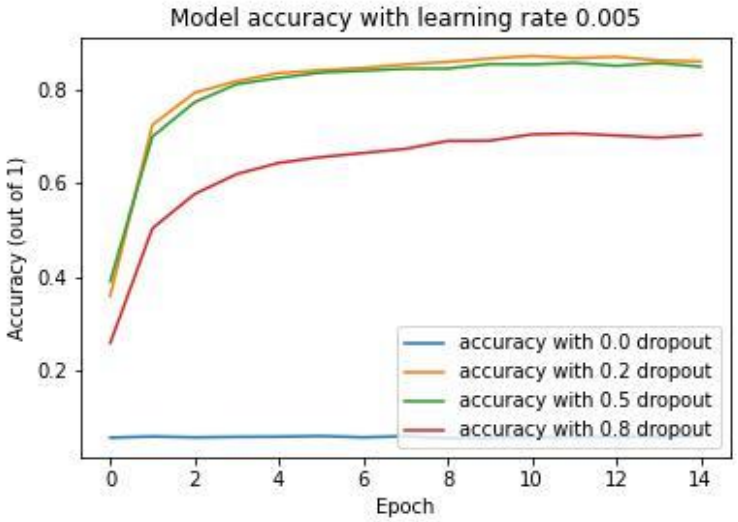
Table 2: Trained CNNs' classification accuracy results (Out of 1)

By analysing further the CNNs' training history with a learning rate of 0.01 the graph below shows that the highest validation accuracy achieved by the trained network "bounced" around 5.5 %. The figure seen below is super "chaotic" as it shows a divergent behaviour of the network. It is also important to note that the graph is "zoomed in" as the changes in accuracy are low in comparison to the graphs of models that converged.



Graph 2: A CNN accuracy graphical representation with different Dropout rates and Learning rate of 0.01

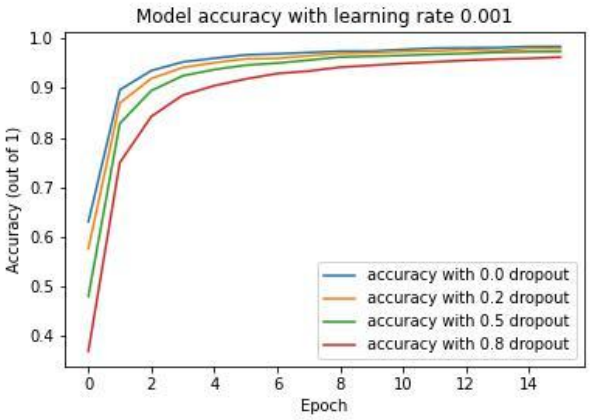
The similar observation of not converging in most cases could be made about networks with a learning rate of 0.005, however, some models with a Polling size of 3x3 achieved a relatively good accuracy that is around 80 % and is shown in the graph below.



Graph 3: A CNN accuracy graphical representation with different Dropout rates and Learning rate of 0.005

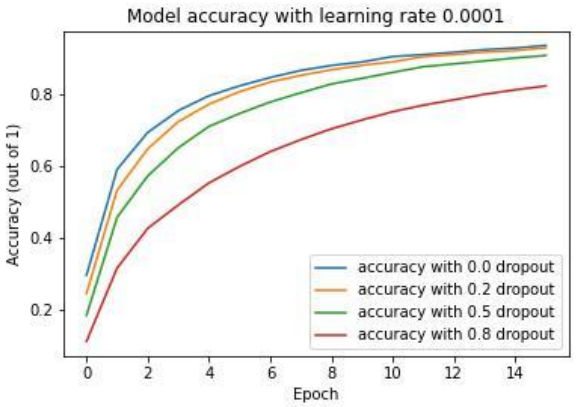
However, these networks most likely converged due to a combination of other factors, different dropout rates, and pooling filter sizes. This can be deducted from *graph 1*. In contrast, all of the models with learning rates of 0.0001 and 0.001 converged successfully.

This drastic difference can be seen when the learning rate was changed from 0.005 to 0.001. The networks trained with the learning rate of 0.001 (graph below) converged and in all of the cases achieved an accuracy of more than 90 %. In addition, most models approached 90 % validation accuracy only after 4 epochs; this was achieved due to the fact that the learning rate was still relatively high, but low enough for the network to converge.



Graph 4: A CNN accuracy graphical representation with different Dropout rates and Learning rate of 0.001

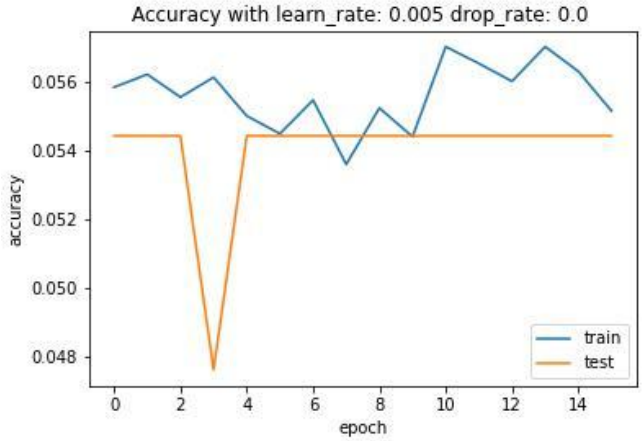
In addition, CNN's with a learning rate of 0.0001 also reached a reasonable performance, around 90 %. However, as the Learning rate of 0.0001 is much smaller in comparison to the CNNs with a learning rate of 0.001, most likely these networks need more epochs to fully converge and reach their global maxima.



Graph 5: CNNs' validation accuracy results with different Dropout rates and Learning rate of 0.0001

4.1.2. Dropout rate analysis

In comparison to *learning rate* changes, there are no such dramatic changes in CNNs' performance between different *dropout rates,* as most of the models converged. However, models with a Dropout rate of 0.5 with a Pooling filter size of 2x2 and models with Dropout rates of 0.2, 0.5 and 0.8 with a Pooling filter size of 3x3 have more interesting results as these models converged in comparison to other models with a Learning rate of 0.005. The only model that did not converge with the Pooling filter size of 3x3 was with a Dropout rate of 0 (graph below).

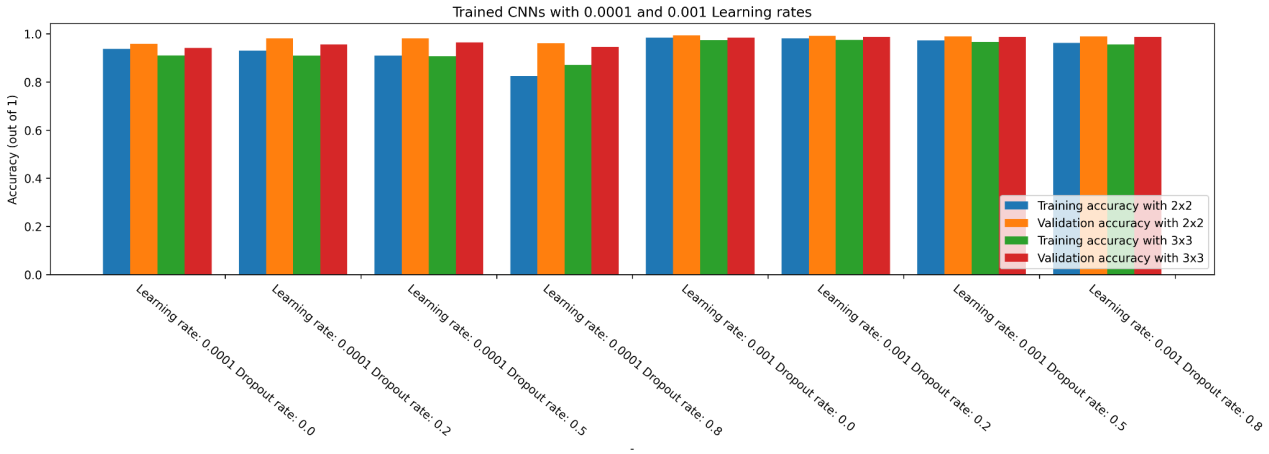


Graph 6: A CNN model training results with the Dropout rate of 0 and Learning rate of 0.005

This model's accuracy bounced around 5 % similar to the models from *graph 2*. This model most likely did not converge due to overfitting as it learned the training data "too well" as any of the nodes were not "dropped" (deleted) as the dropout probability was set to 0 %. This might look counterintuitive, as the network with more parameters should have better classification performance, however, it is important to note that the model is just a large function with trained parameters, so a more complex model with more parameters does not guarantee that this model will be superior to the one with fewer parameters, it only increases its potential to be better.

4.1.3 MaxPool filter size analysis

There was a general trend for networks with filter sizes of 2x2 to have a slightly better classification validation accuracy in comparison to the networks that used Pooling filters of size 3x3, however, this tendency is shown in the graph below with models with Learning rates of 0.0001 and 0.001 as in most cases networks with higher Learning rates did not converge.



Graph 7: Trained CNNs' results with learning rates of 0.0001 and 0.001

These results might look quite foreseeable as networks that used CNN architectures with Pooling filter sizes of 2x2 scaled-down feature maps much less than those with 3x3.

In addition, the models that were trained with filter sizes of 3x3 had on average lower training times of 5 % (Table below).

Size of Pooling filter	2x2	3x3
Average network training time trial 1	28.6 minutes	27.2 minutes
Average network training time trial 2	28.8 minutes	27.4 minutes

Table 3: The model training time results.

The relative difference in training time results looks predictable as the computer has to do fewer operations with filter sizes of 3x3. However, it is important to note that the experiment was carried out only for 16 epochs with a CPU instead of a GPU optimized for training. Meaning that the training time results might greatly differ when training with a dedicated GPU for more epochs.

5. Further research

5.1 Datasets

Most of the countries in Europe have really similar features with few exceptions as the regulation mainly refers to the *1968 Vienna Convention on Road Signs and Signals*. It would be interesting to try the neural networks that were trained for this paper to classify the signs from the other countries and compare the accuracy of these different datasets. In addition, it would be logical to train these CNNs with more sign images from other countries.

5.2 Different hyper-parameters and CNN architectures

An extension of this paper could explore the ways of how different hyper-parameters that were not discussed in section 2.2 influence the accuracy of CNN predictions and methods they could be tuned for higher accuracy.

5.3 Object localization

Another further exploration could include real-time road sign localization in the image space. This could be a further step to implementing these road sign classification CNNs that were trained in this paper for computer vision-based autonomous vehicle navigation.

5.4 Raw data processing

Usually, the data that is used for training CNNs is “friendly” for human eyes as they are presented in such a format that humans are used to seeing, meaning that there is a lot of irrelevant information for CNN processing and training. Thus further research could explore the CNN performance by using raw camera sensor data without any prior preprocessing. This approach was discussed in a Lex Friedman podcast (#252) with Elon Musk, CEO of the *Tesla*

6. Conclusion

This paper explored and analysed how Dropout rate, Learning rate and Pooling filter sizes affect the performance of CNN models for classifying road signs.

The results show that in general higher Learning rates and lower Dropout rates negatively influence the accuracy of the CNN models. However, selecting an appropriate Learning rate is much more important, as by selecting a Learning rate that is too big, the network will never converge. In comparison, different Dropout rates did not have such tremendous differences.

However, some networks with Dropout rates of 0.0 and 0.2 had an overfitting problem. This most likely was caused due to overtraining. Meaning that the network learned the training data too well, thus having slightly worse accuracy with validation data.

The effects of different Pooling filter sizes in terms of network training speed was also predictable, as a model with a smaller filter (2x2) had to do more operations than a network with a bigger (3x3) filter size, thus the average training times were lower with 2x2 filters. However, it wasn't really possible to predict the effects of different filter sizes on a network's accuracy. However, the results (*graph 1 and table 2*) showed that the networks with filter sizes of 2x2 had a better classification performance in general. From the experiment data, the final conclusion could be made that trained networks for 16 epochs can achieve an accuracy of 99 % with a learning rate of 0.001 and different dropout rates.

It is also important to note the results of the CNN architectures and parameters explored in this paper might differ with different data, for example, different vehicle model classification, however, general trends should remain the same.

7. Bibliography

Aljaafari Nura. “Example of max pooling and average pooling operations” Research Gate, (n.d.) https://www.researchgate.net/figure/Example-of-max-pooling-and-average-pooling-operations-In-this-example-a-4x4-image-is_fig4_332092821. Accessed 6 January 2022

“Admin Jobs”, “ How to Treat Overfitting in Convolutional Neural Networks” Analytics Vidhya, 7 September 2020, https://www.analyticsvidhya.com/blog/2020/09/overfitting-in-cnn-show-to-treat-overfitting-in-convolutional-neural-networks/#:~:text=Overfitting%20indicates%20that%20your%20model,of%20overall%20Deep%20Learning%20Models. Accessed 6 January 2022

Alto Valentina. “Data Augmentation in Deep Learning” Medium, 19 July 2020, https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28. Accessed 6 January 2022

Arnx Arthur. “First neural network for beginners explained (with code)” Towards Data Science 13 January 2019, https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf Accessed 15 August 2021

Brownlee Jason. “Understand the Impact of Learning Rate on Neural Network Performance ” Machine Learning Mastery 25 January 2019, https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/#:~:text=The%20learning%20rate%20is%20a,the%20model%20weights%20are%20updated.&text=The%20learning%20rate%20may%20be,when%20configuring%20your%20neural%20network. Accessed 6 January 2022,

"9 Applications of Deep Learning for Computer Vision" Machine Learning Mastery, 13 March 2019, https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/ Accessed 6 January 2022,

"How to Configure the Learning Rate When Training Deep Learning Neural Networks" Machine Learning Mastery 23 January 2019, https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/#:~:text=Specifically%2C%20the%20learning%20rate%20is,the%20size%20of%20the%20step. Accessed 13 October 2021

Basavarajaiah Madhushree. "Maxpooling vs minpooling vs average pooling" Medium, 8 February 2019, https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9 Accessed 6 January 2022

Chernytska Olga. "Comple Giude to Data Augmentation for Computer Vision" Medium, 1 June 2021, https://towardsdatascience.com/complete-guide-to-data-augmentation-for-computer-vision-1abe4063ad07 Accessed 3 February 2022

Credi Jacopo. "Traffic sign classification with deep convolutional neural networks" Chalmers University of Technology, 2016, https://publications.lib.chalmers.se/records/fulltext/238914/238914.pdf Accessed 6 January 2022

Choudhury, Ambika. "8 Uses Cases Of Image Recognition That We See In Our Daily Lives." Analytics India Magazine, 5 May 2019, analyticsindiamag.com/8-uses-cases-of-image-recognition-that-we-see-in-our-daily-lives/. Accessed 6 January 2022

"DeepAI" "Multilayer Perceptron" deepai.org, (n.d.), deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

Deepanshi. "Beginners Guide to Artificial Neural Network" Analytics Vidhya, 25 May 2021, https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/ Accessed 6 January 2022

Fridman Lex, et al. "Self-Driving Cars Lectures" Youtube, 2019, https://www.youtube.com/playlist?list=PLrAXtmErZgOeY0lkVCIVafdGFOTi45amq Accessed 18 June 2020

Fridman Lex, Musk Elon. "Elon Musk: SpaceX, Mars, Tesla Autopilot, Self-Driving, Robotics, and AI | Lex Fridman Podcast #252" Youtube, 28 December 2021 https://www.youtube.com/watch?v=DxREm3s1scA, Accessed 28 December 2021

Ganesh Prakhar. "Types of Convolution Kernels: Simplified" Towards data science, 18 October 2019, https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37 Accessed 6 January 2022

Gebel Lukasz. "Why We Need Bias in Neural Networks" Towards Data Science 21 August 2020, https://towardsdatascience.com/why-we-need-bias-in-neural-networks-db8f7e07cb98 Accessed 28 October 2021

"IBM" "What is computer vision?" ibm.com, (n.d.), https://www.ibm.com/topics/computer-vision Accessed 9 January 2022

Jordan Jeremy. "Setting the learning rate of your neural network." Jeremyjordan 1 March 2018, https://www.jeremyjordan.me/nn-learning-rate/ Accessed 6 January 2022

Lascelle Kael. “Building a Road Sign Classifier” Towards Data Science 9 September 2020, https://towardsdatascience.com/road-sign-classification-learning-to-build-a-cnn-7771373179d3 Accessed 18 June 2021

Mahony Niall, et al. “Deep Learning vs. Traditional Computer Vision” (n.d.) https://arxiv.org/ftp/arxiv/papers/1910/1910.13796.pdf Accessed 7 January 2022

“Matlab” MathWorks “Introducing Deep Learning with MATLAB” (n.d.), https://se.mathworks.com/campaigns/offers/next/deep-learning-ebook.html Accessed 6 January 2022

“Medmain” “My first steps into the world of A.I.” Medium, 29 October 2018, https://medium.com/@Medmain/revolutional-convolutional-neural-nets-c5264940144 7 Accessed 6 January 2022

Minhas Singh Manpreet. “Techniques for handling underfitting and overfitting in Machine Learning” Medium. 6 June 2021, https://towardsdatascience.com/techniques-for-handling-underfitting-and-overfitting-in-machine-learning-348daa2380b9 Accessed 6 January 2022

Ng Andrew. “Machine Learning” Coursera.org, (n.d.) https://www.coursera.org/learn/machine-learning Accessed 13 August 2021

Patel Ankur. “Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data.”, O'Reilly Media, 2019. Accessed 12 November 2021

Powell Victor. “Image Kernels” setosa.io (n.d.) https://setosa.io/ev/image-kernels/

“PyTorch Documentation” ReLU function documentation, https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html Accessed 6 January 2022

Pankajray. “Artificial Neural Network (ANN)” Medium, 4 June 2020, https://medium.com/ai-knowledge/artificial-neural-network-ann-ed6fa5b9c1b0 Accessed 12 January 2022

Qureshi Muhammad. “3rd-order Tensor representation of a color image” ResearchGate, (n.d.), https://www.researchgate.net/figure/3rd-order-Tensor-representation-of-a-color-image_fig2_307091456 Accessed 15 February 2022

Rosebrock Adrian “Traffic Sign Classification with Keras and Deep Learning” pyimagesearch, 4 November 2019 https://pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/ Accessed 27 January 2022

Radhakrishnan Pranoy. “What are Hyperparameters? And How to tune the Hyperparameters in a Deep Neural Network?” Medium, 9 August 2017 https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyper parameters-in-a-deep-neural-network-d0604917584a Accessed 27 January 2022

Sermanet Pierre and LeCun Yann. “Traffic Sign Recognition with Multi-Scale Convolutional Network” (n.d.) Courant Institute of Mathematical Sciences, New York University http://yann.lecun.com/exdb/publis/pdf/sermanet-ijcnn-11.pdf Accessed 27 January 2022

Staravoitau, Alex. “Traffic signs classification with a convolutional network” (n.d.) https://navoshta.com/traffic-signs-classification/ Accessed 6 January 2022

Smola and Vishwanathan. “Introduction to machine learning,” Methods Mol. Biol, 2014. Accessed 13 November 2021

“SuperDataScience Team”. “Convolutional Neural Networks (CNN): Step 3 - Flattening” Super Data Science, 18 August 2018,

https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening Accessed 1 January 2022

Srivastava, Hinton, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” Journal of Machine Learning Research, 14 June 2014 https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf Accessed 1 January 2022

Stallkamp, Schlipsing, et al. “Benchmarking machine learning algorithms for traffic sign recognition” 20 February 2012, http://www.sciencedirect.com/science/article/pii/S0893608012000457 Accessed August 2021

Sousa Mark. “Visualizing the Fundamentals of Convolutional Neural Networks” Towards Data Science, 8 December 2019, https://towardsdatascience.com/visualizing-the-fundamentals-of-convolutional-neural-networks-6021e5b07f69 Accessed 6 January 2022

“Simplilearn” “What is Perceptron: A Beginners Guide for Perceptron” Simplilearn, 16 December 2021, https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron Accessed 6 January 2022

Thilakarathne Haritha. “Deep Learning Vs. Traditional Computer Vision” NaadiSpeaks, 12 August 2018, https://naadispeaks.wordpress.com/2018/08/12/deep-learning-vs-traditional-computer-vision/ Accessed 12 August 2021

Thakur Ayush. “ReLU vs. Sigmoid Function in Deep Neural Networks: Why ReLU is so Prevalent” wandb.ai (n.d) https://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in

-Deep-Neural-Networks-Why-ReLU-is-so-Prevalent--VmlldzoyMDk0MzI Accessed 6 January 2022

Tracey Thomas. “Recognizing Traffic Signs with CNNs” Medium, 24 February 2019 https://medium.com/@thomastracey/recognizing-traffic-signs-with-cnns-23a4ac66f7a7. Accessed 28 September 2021

Wolfewicz Arne. “Deep learning vs. machine learning - What’s the difference” Levity, 9 November 9 2021, https://levity.ai/blog/difference-machine-learning-deep-learning#:~:text=Machine%20learning% Accessed 26 January 2022

Zhang, Lipton, et al. “Dive into Deep Learning” 15 July 2019, https://d2l.ai/d2l-en.pdf Accessed 23 June 2021

8. Appendix

The following code snippets were used to create CNN models and manipulate training and validation data. The code was run using *Jupyter Notebook* inside *Python's* virtual environment.

```
import numpy as np
import matplotlib.pyplot as plt
import keras
import pickle
import pandas as pd
import random
import cv2
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import adam_v2
from keras.utils.np_utils import to_categorical
from keras.layers import Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
np.random.seed(0)

def grayscale(image):
  image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  return image

image = grayscale(X_train[1000])
plt.imshow(image, cmap=plt.get_cmap('gray'))
plt.axis('off')
print(image.shape)

def equalize(image):
  image = cv2.equalizeHist(image)
  return image

image = equalize(image)
plt.imshow(image, cmap=plt.get_cmap('gray'))
plt.axis('off')
print(image.shape)

def preprocessing(image):
  image = grayscale(image)
  image = equalize(image)
  image = image/255
  return image

X_train = np.array(list(map(preprocessing, X_train)))
X_valid = np.array(list(map(preprocessing, X_valid)))
X_test = np.array(list(map(preprocessing, X_test)))
```

```
from keras.preprocessing.image import ImageDataGenerator

data_gen = ImageDataGenerator(width_shift_range=0.1,
                              height_shift_range=0.1,
                              zoom_range=0.2,
                              shear_range=0.1,
                              rotation_range=10.)

data_gen.fit(X_train)

batches = data_gen.flow(X_train, y_train, batch_size=20)
X_batch, y_batch = next(batches)

fig, axs = plt.subplots(1, 15, figsize=(20, 5))
fig.tight_layout()

for i in range(15):
  axs[i].imshow(X_batch[i].reshape(32, 32))
  axs[1].axis('off')

print(X_batch.shape)

def create_model(l_rate, dropout):
    model = Sequential()
    model.add(Conv2D(60, (5, 5), input_shape=(32, 32, 1), activation='relu'))
    model.add(Conv2D(60, (5, 5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3)))

    model.add(Conv2D(30, (3, 3), activation='relu'))
    model.add(Conv2D(30, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3)))

    model.add(Flatten())
    model.add(Dense(500, activation='relu'))
    model.add(Dropout(dropout))
    model.add(Dense(43, activation='softmax'))

    #Compile Model
    optimizer = adam_v2.Adam(learning_rate=l_rate)
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

learning_rates = [0.01, 0.005, 0.001, 0.0001]
dropout_rates = [0.0, 0.2, 0.5, 0.8]

parameter_combinations = []
for learning_rate in learning_rates:
    for dropout_rate in dropout_rates:
        parameter_combinations.append([learning_rate, dropout_rate])

print(parameter_combinations)

def generate_metadata(learn_rate, drop_rate):
    return f'learn_rate-{learn_rate}_drop_rate-{drop_rate}'
```

```
p2_training_histories = []
p3_training_histories = []

for file in os.listdir():
    if file.split('_')[0] == 'p2':
        p2_training_histories.append(file)
    elif file.split('_')[0] == 'p3':
        p3_training_histories.append(file)
for i in p3_training_histories:
    print(i)

for combination in parameter_combinations:
    learn_rate = combination[0]
    drop_rate = combination[1]
    model = create_model(learn_rate, drop_rate)

    metadata = generate_metadata(learn_rate, drop_rate)
    print(f'training model with: {metadata} pool= 3x3 ')
    history = model.fit_generator(data_gen.flow(X_train, y_train, batch_size=50),
                                  steps_per_epoch=X_train.shape[0]/50,
                                  epochs=16,
                                  validation_data=(X_valid, y_valid),
                                  shuffle = 1)

    save_training_data(metadata)
    model.save(f'{metadata}_trained_model')

def generate_graphs(histories, category='accuracy'):
    for i, rate in enumerate(histories):

        plt.clf()
        if category == 'accuracy':
            plt.ylabel('Accuracy (out of 1)')
        else:
            plt.ylabel('Loss')
        plt.xlabel('Epoch')

        legend_train = []

        for history_file in rate:

            history = np.load(f'{history_file}', allow_pickle='TRUE').item()
            meta =
"".join(history_file.split("p3_training_history_learn_rate-")).split("_drop_rate-")
            meta = "".join("-".join(meta).split("_.npy")).split("-")

            print(meta[0])
            plt.title(f'Model {category} with learning rate {meta[0].replace(".npy", "")}')

            print(f'{meta[0]} {meta[1]}')

            plt.plot(history[f'{category}'])
            legend_train.append(f'{category} with {meta[1].replace(".npy", "")} dropout')

        if category == 'accuracy' or category == 'val_accuracy':
            plt.legend(legend_train, loc='lower right')
        else:
            plt.legend(legend_train, loc='upper right')

        plt.plot()
        plt.savefig(f'./graphs_cluster_p3/{history_file}_{category}.jpeg')
        print(i)
```

```
def generate_bar(bar_data, b2):
    width = 0.2
    x = np.arange(8)
    y1 = [] #accuracy
    y2 = [] #validation
    y3 = []
    y4 = []
    x_label = []
    for i in bar_data:
        if i[0][0] == "0.0001" or i[0][0] == '0.001':
            y1.append(i[2][0])
            y2.append(i[2][1])
    for i in b2:
        if i[0][0] == "0.0001" or i[0][0] == '0.001':
            y3.append(i[2][0])
            y4.append(i[2][1])
#         x_label.append("-".join(i[0]).replace(".npy", ""))
#         label = f"Learning rate: {i[0][0]} Dropout rate: {i[0][1]}".replace(".npy", "")
#         x_label.append(label)

    print(f'{y1} \n\n\n{y2}\n\n\n{x}', end="\n\n")
    plt.figure(figsize=(18, 4))
    plt.bar(x-0.2, y1, width)
    plt.bar(x, y2, width)
    plt.bar(x+0.2, y3, width)
    plt.bar(x+0.4, y4, width)
    plt.xticks(x+0.6, x_lab[0:8], rotation=-40)
    plt.xlabel("-")
    plt.ylabel("Accuracy (out of 1)")
#     plt.legend(["Training accuracy","Validation accuracy"])
    plt.legend(["Training accuracy with 2x2","Validation accuracy with 2x2","Training accuracy with
3x3","Validation accuracy with 3x3"], loc='lower right')
    plt.title("Trained CNNs with 0.0001 and 0.001 Learning rates")
    plt.gcf().set_dpi(300)
    plt.show

#     for i in x_label:
#         x_lab.append(i)
#     plt.savefig("./bars/bar_3x3.jpeg", dpi=300)

generate_bar(p2_bar_data, p3_bar_data)
```