Classification and Prediction of Steering Direction in Self-Driving Cars

Dan Hemgren, Victor Hansjons Vegeborn, Jakob Svenningson

KTH Royal Institute of Technology

Abstract. Deep learning is one of the major frontiers in the field of autonomous cars. This report aims to implement and evaluate different architectures of perhaps the most commonly used neural network in the field - convolutional neural networks. Attempts are made to classify driving directions and predict steering angle with regression by training 5 respectively 4 different network architectures over a common dataset. The final networks had a accuracy of $\approx 91.7\%$ in the classification task and an MSE of ≈ 0.001 in the regression task. However, these results are discussed further as the distribution in the dataset may have influenced the results in a positive manner.

Keywords: deep learning, self-driving cars, convolutional networks, machine learning, computer vision

1 Introduction

Computer vision plays an integral part in the research of self-driving cars. In particular, using computer vision with convolutional neural networks (CNN, ConvNet) to decide steering direction has garnered much success in recent years [1]. The use of CNNs are not limited to self-driving cars, however. Following a landslide win in the ImageNet 2012 competition, with a CNN constructed by Krizhevsky et al halving the error rate of the competition [2], CNNs have been a leading approach for a majority of recognition and detection based tasks in computer vision [3].

The goal of this report is to implement and evaluate techniques for classification and regression of steering direction as described by Bouton and Hersey [4]. In our report, a number of CNNs of varying depth and feature sets are detailed and compared using a common dataset for training extracted from the open source dataset of driving data released by Udacity [5].

2 Background

2.1 Deep Learning

Since the ImageNet 2012 competition breakthrough of Krizhevsky et al [2], Deep Learning has garnered state of the art results in several fields including image and speech recognition [6]. For example, networks have been trained via reinforcement learning to play Atari games at an adept level [7], and to visually recognize the laws of physics by being trained with images of simulated falling towers [8].

Marcus describes Deep Learning as a statistical technique for classifying patterns based on sample data using neural networks. Marcus also states that, *'[i]n principle, given infinite data, deep learning systems are powerful enough to represent any finite deterministic mapping between any given set of inputs and a set of corresponding outputs'* and argues, conversely, that an inherent weakness to the technique of Deep Learning is its dependence on large datasets and lack of generality in the resulting trained networks [6].

2.2 Convolutional neural network

Convolutional neural network (CNN) is a type of neural network which has had great success when applied to image analysis tasks. A CNN consists of an input layer, an output layer and multiple hidden layers. There are several different types of hidden layers that can be used in a CNN. The types of hidden layers used in the networks in this report is presented in the list below.

- **Convolutional layers** consist of multiple small learnable filters. During the forward pass, each filter is moved along the width and height of the input volume and computes the dot product between the filter and the input at each sliding position [9].
- **Max Pooling layers** is used to reduce the width and height of the input from the previous layer. The size of the input is reduced by sliding a square (typically of size 2x2) over the input layer and transforming each 2x2 square of the input to a single number in the output volume. In max pooling, this is done by choosing the maximum value in each 2x2 square [9]. For instance, if we use a square of size 2x2 the width and height of the output volume will be half of the size of the input volume.
- **Dropout layers** drop nodes of a layer with a probability p. This is a technique used to prevent overfitting [10].
- **Fully connected layers** are connected to each neuron in the previous layer. This kind of layers are similar to the hidden layers in traditional neural networks.
- **Leaky ReLu layers** applies the activation $f(x) = \begin{cases} x & x > 0 \\ x * 0.01 & otherwise \end{cases}$

2.3 Bouton and Hersey

The project of this report is based on a 2016 paper written by Bouton and Hersey in which methods for predicting driving controls based on image input are examined [4]. In their report, Bouton and Hersey split the problem statement of predicting driving controls based on images into two subproblems:

– predicting the driving direction via classification (left, right, straight)
– predicting the steering angle via regression

The final CNN constructed in the paper for the classification problem is depicted in Fig. 1. Using this architecture, the authors manage to reach a $\approx 97\%$ success rate in predicting the discrete driving direction.

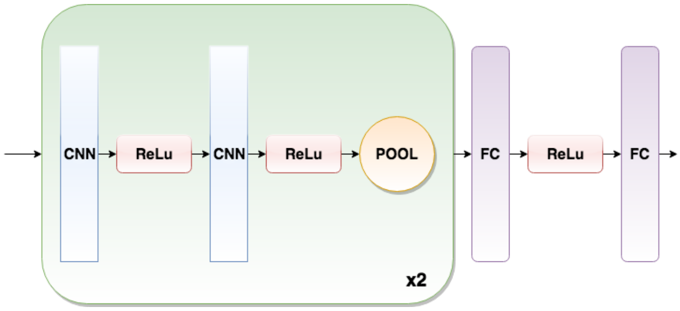


Fig. 1: Depiction of final classification network architecture of Bouton and Hersey

For the regression task, Bouton and Hersey present a number of architectures, all with varying degrees of added complexity over the CNN used for classification. The most successful architecture employs PReLu and an additional convolutional filter, resulting in a mean squared error of 0.0601 after 90 epochs.

3 Approach

The CNNs are constructed and trained using TensorFlow. Our base network configuration mirrors the final architecture decided on by Bouton and Hersey [4] except that our implmentation uses LReLu instead of ReLU as activation function; four convolutional layers, two pooling layers and two fully connected layers with LReLu regularization after each convolutional layer with a further LReLu following the initial fully connected layer, as depicted in Fig. 1. The anatomy of this base architecture is then altered to try and improve the test results over the dataset, for instance by adding or removing layers and features.

3.1 TensorFlow

TensorFlow is a machine learning system meant for large scale calculations with distributed computing developed by Google Brain. The name TensorFlow is derived from the fact that all data are modeled as *tensors* (n-dimensional arrays) in the system [11]. Within TensorFlow, a number of layers and features come predefined with parameters exposed for further configuring (such as convolutional and pooling layers, number of nodes per layer, learning rate et.c.). For this project, the Python client for TensorFlow was used.

3.2 Network architectures

Classification The final network architectures in the classification experiments is presented in Figure 2. Every convolution layer in all of the networks applied zero-padding in order to maintain the same width and height as the input volume. All training was done using a batch size of 32. The final parameter setup for each individual layer is presented in Figure 3 including the input and output volumes.

Network	**Architecture**
1	`INPUT->[[CONV->LReLU]*2->POOL]*2->[FC->LReLU]->OUTPUT`
2	`INPUT->[[CONV->LReLU]*1->POOL]*2->[FC->LReLU]->OUTPUT`
3	`INPUT->[[CONV->LReLU]*1->POOL]*1->[FC->LReLU]->OUTPUT`
4	`INPUT->[[CONV->LReLU]*1->POOL]*3->[FC->LReLU]->OUTPUT`
5	`INPUT->[[CONV->LReLU]*1->POOL]*2->[FC->LReLU]->DROPOUT->OUTPUT`

Fig. 2: The five different classification network architectures

Network 1	**Network 2**	**Network 3**	**Network 4**	**Network 5**
INPUT: 80x60x3	INPUT: 80x60x3	INPUT: 80x60x3	INPUT: 80x60x3	INPUT: 80x60x3
CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32
CONV2: F=3, S=1, K=32	POOL1: F=2, S=1	POOL1: F=2, S=1	POOL1: F=2, S=1	POOL1: F=2, S=1
POOL1: F=2, S=1	CONV2: F=3, S=1, K=64	FC: n=512	CONV2: F=3, S=1, K=64	CONV2: F=3, S=1, K=64
CONV3: F=3, S=1, K=64	POOL2: F=2, S=1	OUTPUT: n=3	POOL2: F=2, S=1	POOL2: F=2, S=1
CONV4: F=3, S=1, K=64	FC: n=512		CONV3: F=3, S=1, K=64	FC: n=512
POOL2: F=2, S=1	OUTPUT: n=3		POOL3: F=2, S=1	DROPOUT: p=0.8
FC: n=512			FC: n=512	OUTPUT: n=3
OUTPUT: n=3			OUTPUT: n=3	

Fig. 3: Classification network layer parameters

Regression The final network architectures in the regression experiments is presented in Figure 4. Every convolution layer in all of the networks applied zero-padding in order to maintain the same width and height as the input volume. All training was done using a batch size of 32. The final parameter setup for each individual layer is presented in Figure 5 including the input and output volumes.

Network	Architecture
A	INPUT->[[CONV->LReLU]*1->POOL]*1->[FC->LReLU]*1->OUTPUT
B	INPUT->[[CONV->LReLU]*1->POOL]*1->[FC->LReLU]*2->OUTPUT
C	INPUT->[[CONV->LReLU]*1->POOL]*2->[FC->LReLU]*1->OUTPUT
D	INPUT->[[CONV->LReLU]*1->POOL]*2->[FC->LReLU]*2->OUTPUT

Fig. 4: The different regression network architectures

Network A	Network B	Network C	Network D
INPUT: 80x60x3	INPUT: 80x60x3	INPUT: 80x60x3	INPUT: 80x60x3
CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32	CONV1: F=3, S=1, K=32
POOL1: F=2, S=1	POOL1: F=2, S=1	POOL1: F:2 S: 1	POOL1: F:2 S: 1
FC: n=512	FC: n=512	CONV2: F=3, S=1, K=64	CONV2: F=3, S=1, K=64
OUTPUT: n=1	FC: n=512	POOL2: F=2, S=1	POOL2: F=2, S=1
	OUTPUT: n=1	FC: n=512	FC: n=512
		OUTPUT: n=1	FC: n=512
			OUTPUT: n=1

Fig. 5: Regression network layer parameters

Optimizer and loss function All networks use the Adam optimizer algorithm. The Adam optimizer is currently the default optimization algorithm for CNNs [9]. The classification networks all use categorical cross entropy as loss function. The regression networks all use mean squared error (MSE) as loss function.

3.3 Dataset

The driving data used in this report has been provided by Udacity who has open-sourced hundreds of gigabytes of driving data [5]. However, due to limited time and resources the networks in this report has only been trained on a small portion of the available driving data, namely HMB1, HMB4 and HMB6 from the training data of the second Udacity Challange [12]. Udacity provides image data from three different cameras mounted on the car at every timestep (left, right and center). We have chosen to only use the image data from the center camera. Every image was labeled with a steering angle in the range between $-0,363$ and 0.539. The final dataset consisted of 11.5 minutes of country road driving data. In order to speed up training, the images were scaled down from 640x480 to 80x60. This mirrors the approach of Bouton and Hersey [4]. Fig. 6 shows how the dataset was split up in a train, validation and test set.

Train(70%)	Validation (10%)	Test (20%)
9647	1378	2755

Fig. 6: Dataset distribution over training, validation and testing.

For the classification part of the project, images were pre-processed and labeled with left, straight, and right, by a threshold of ± 0.02 set on the steering angle. The final distribution of the classification data is found in Fig. 7.

Left(43.20%)	Straight(37.12%)	Right (19.68%)
5951	5114	2712

Fig. 7: Dataset class distribution

The dataset is narrowed down by extracting three continuous drive segments. Out of these three segments, every 10th picture is allocated to the the validation set and every 5th picture to the test set, achieving the distribution depicted in Fig. 7. Examples of images sampled can be seen in Fig. 8.

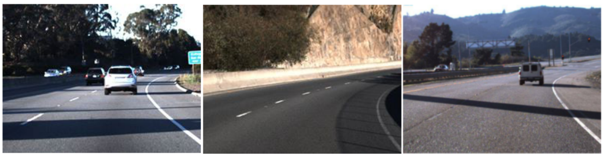


Fig. 8: Three images from the dataset labelled 'straight', 'right' and 'right'

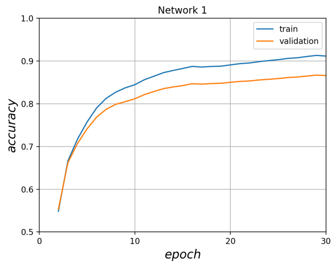
4 Results

4.1 Classification

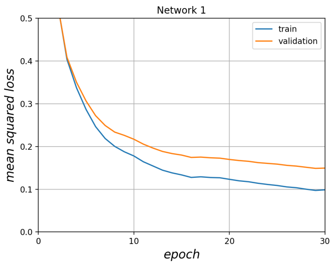
All five network architectures performed at $\geq 85\%$ accuracy on the classification task. The results of all five networks are presented in figure 9. Network 5 performed best and got an accuracy of $\approx 91.7\%$ on the test set, followed by network 2 at $\approx 90.1\%$ test accuracy. The loss and accuracy of the different networks is presented in figure 10, 11, 12, 13 and 14.

Network	Learning Rate, η	Train Accuracy	Validation Accuracy	Test Accuracy
1	0.001	0.913	0.867	0.877
2	0.001	0.935	0.891	0.901
3	0.001	0.910	0.863	0.877
4	0.001	0.937	0.888	0.898
5	0.001	0.945	0.907	0.917

Fig. 9: Results after 30 epochs of training

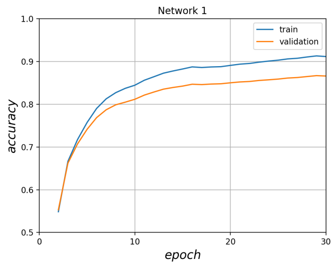


(a) Accuracy

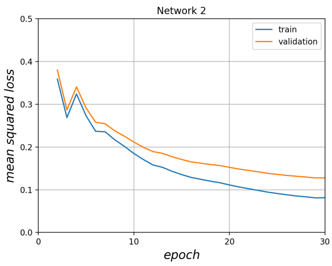


(b) Loss

Fig. 10: Network 1, $\eta = 0.001$

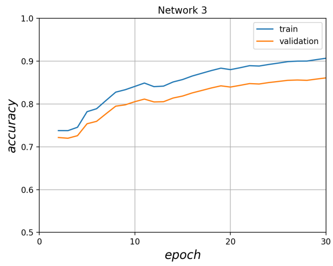


(a) Accuracy

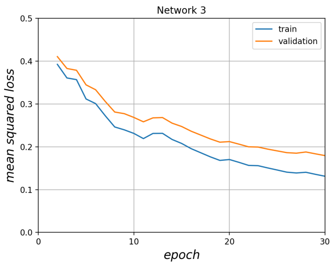


(b) Loss

Fig. 11: Network 2, $\eta = 0.001$

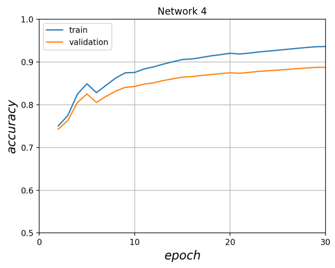


(a) Accuracy

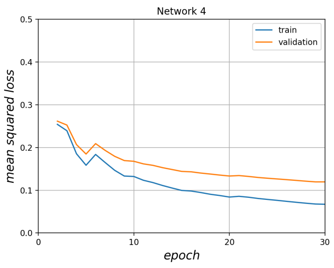


(b) Loss

Fig. 12: Network 3, $\eta = 0.001$

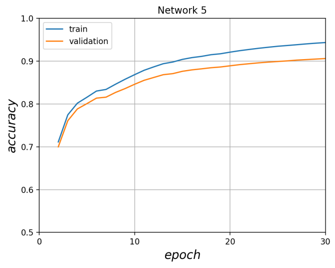


(a) Accuracy

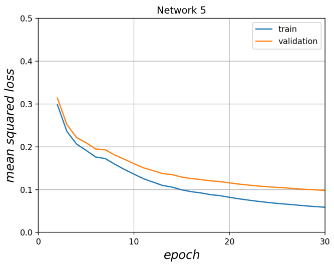


(b) Loss

Fig. 13: Network 4, $\eta = 0.001$



(a) Accuracy



(b) Loss

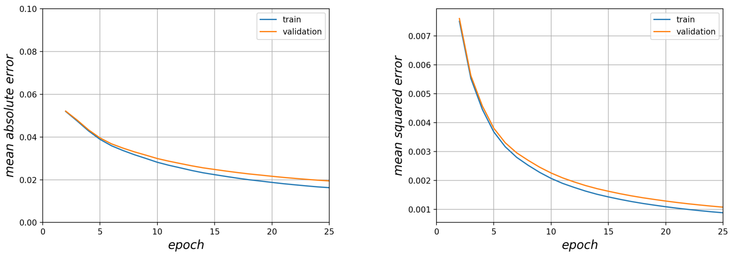
Fig. 14: Network 5, $\eta = 0.001$

4.2 Regression

The mean squared of the four regression network architectures after 10 epochs of training are presented in figure 15. The best performing network is network A which we trained for an increased amount of epochs. The mean squared error (MSE) and the mean absolute error (MAE) of the longer training session of network A can be seen in fig 16. Network A achieved roughly 0.001 MSE on the test set after 25 epochs. The prediction distribution by network A on the entire dataset is presented in figure 17.

Network	Learning Rate, η	Train MSE	Validation MSE	Test MSE
A	0.0001	0.00152	0.00167	0.00165
B	0.0001	0.00179	0.00194	0.00192
C	0.0001	0.00255	0.00263	0.00262
D	0.0001	0.00335	0.00343	0.00342

Fig. 15: Results after 10 epochs of training



(a) Mean absolute error (MAE) (b) Mean squared error (MSE)

Fig. 16: Network A, $\eta = 0.0001$, epoch=25

5 Conclusion

5.1 Classification

All classification networks performed at above 85% accuracy, but did not manage to perform as well as the networks in the report by Bouton and Hersey. A possible reason for this may be differences in the datasets. The base network architecture defined in Bouton and Herseys report, network 1 in Fig. 9, achieved an $\approx 88\%$ accuracy on our dataset. By comparison, the same network achieved

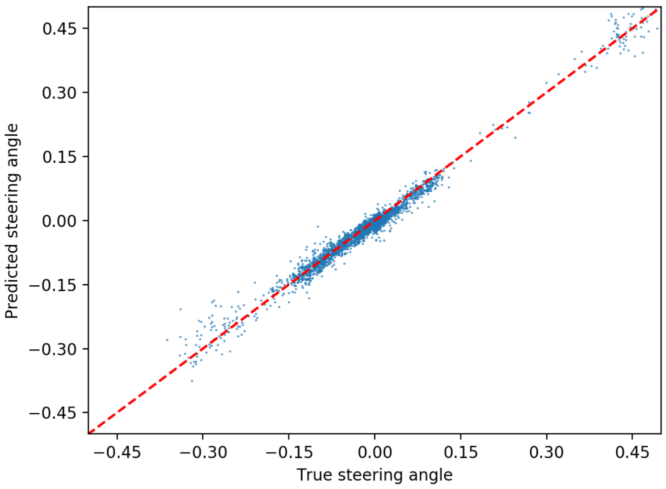


Fig. 17: Network A, prediction distribution

a 97% accuracy on the dataset sampled by Bouton and Hersey. This ≈ 9% difference can perhaps illuminate differences between the two datasets more so than differences in implementation. The distribution of discrete steering direction across our dataset is roughly 0.43/0.37/0.20 for left/straight/right (as seen in Fig. 7). The Bouton and Hersey dataset instead has a distribution of, again roughly, 0.31/0.46/0.23. Thus, while our dataset is slightly skewed towards the left class, their dataset is instead (almost equally) skewed towards the straight class. Another difference between the two datasets is the sheer size, with our dataset having a size of 13780 compared to the dataset used by Bouton and Hersey consisting of 12200 images. This yields a 13% increase in dataset size (but a ≈ 19% smaller test size due to difference in train/validation/test distributions). These differing attributes notwithstanding, little insight is given regarding chosen method of sampling in the Bouton and Hersey report besides the trimming of unwanted elements (such as long segments of footage where the car is stopped at a red light). A wider distribution of steering angle is also shown in their prediction distribution graph, hinting at a driving environment different from ours. Given our dataset, introducing a drop layer with 20% node dropping and removing two convolutional layers (network 5, as seen in Fig. 2) yielded a ≈ 4.0% increase in accuracy over the base architecture.

5.2 Regression

The starting point for the regression part of this project was based on the simplest network from the classification part (Network 3 in Fig. 2). The best performing network (A) had an MSE of ≈ 0.00165 while the worst performing network (D) had an MSE of ≈ 0.00342 after 10 epochs of training. This behaviour contradicted the assumption of convolutional networks performing better simply by adding additional layers. The general notion that larger networks require training on larger datasets [10] could be a contributing factor to our simplest network performing so well. Network A was trained further over 25 epochs and finally yielded an MSE of ≈ 0.001. The prediction distribution depicted in Fig. 17 show a fairly decent prediction rate of the steering angle in the range of ± 0.15. However, the network does not perform as well on the harder turns i.e outside of the previously mentioned range. This is probably a consequence of the distribution of steering angles in the dataset being very narrow as driving on the country side have limited amounts of hard turns.

5.3 Dataset criticism

As mentioned previously in the report, it was not possible to use the entire dataset in the project due to limited time and resources. Therefore, the authors choose to only use parts of the original dataset. The sampling was conducted by extracting three continuous drive segments which mostly included country road driving from the complete dataset. This gave a very homogeneous dataset consisting of only country road driving in sunny weather and therefore our results should be treated with caution. In order to properly judge the performance of our networks, a more diverse dataset would be needed with several different driving environments and weather conditions. If we would have sampled images randomly from the entire dataset, our dataset would have been more diverse and it is likely that the results for both the classification and regression task would have been different given the same sample size.

References

1. Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., Mujica, F., Coates, A., Ng, A.Y.: An empirical evaluation of deep learning on highway driving. CoRR **abs/1504.01716** (2015)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105
3. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521** (May 2015) 436 EP –
4. Heyse, J.F., Bouton, M.: End-to-end driving controls predictions from images. (December 2016)
5. Cameron, O.: Open sourcing 223gb of driving data udacity inc medium (Oct 2016)
6. Marcus, G.: Deep learning: A critical appraisal. CoRR **abs/1801.00631** (2018)
7. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518** (Feb 2015) 529 EP –
8. Lerer, A., Gross, S., Fergus, R.: Learning physical intuition of block towers by example. CoRR **abs/1603.01312** (2016)
9. Li, F.F.: Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/convolutional-networks/conv Accessed: 2018-05-20.
10. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1) (2014) 1929–1958
11. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. Volume 16. (2016) 265–283
12. Udacity: Challange 2. https://github.com/udacity/self-driving-car/tree/master/datasets/CH2 Accessed: 2018-05-22.