Duckiebot Lane Follower with Convolutional Neural Networks
- Progress Report 1 -

Burak Aksoy
aksoyb2@rpi.edu

May 15, 2019

Contents

1 Introduction

1.1 Lane Following Algorithm Overview

Duckiebot project already has a Lane Following algorithm which takes the camera images, detects lines, calculates its position and orientation in the lane, apply some control dynamics, decides required linear and angular velocity, and sets the wheel speeds accordingly. Lane Following algorithm overview flow can be seen in Figure 1.

This algorithm is already implemented in Robotic Opereating System(ROS) using Python. The source files can be found in RPI Robotics Duckietown GitHub repository.

In our previous work, we have implemented the same algorithm also in Robot Raconteur(RR) framework using Python. The source files can be found in in this link.

We compared the performance results of ROS and RR experimenting on a Duckiebot powered with Raspberry Pi 2 Model B and observed average rate of completing the algorithm as in Table 1

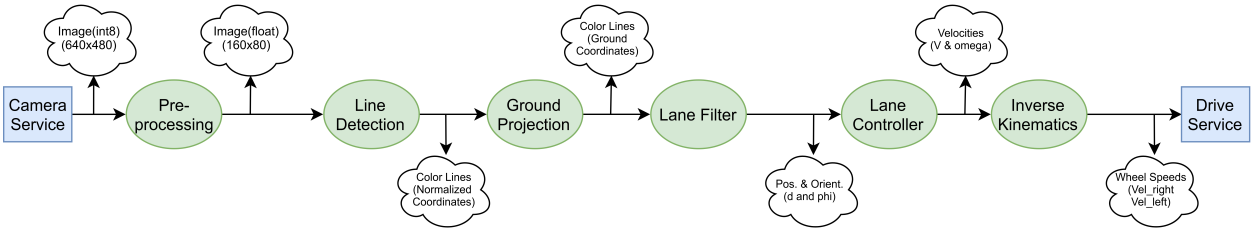


Figure 1: Lane Following algorithm overview flow.

Framework	Rate(per second)
ROS	9.5
RR	11.0

Table 1: Performance comparison between RR and ROS

The obtained result shows that RR gives approximately 1.5 completion rate per second faster response than ROS. This gain is mainly result of fast Camera and Drive Service that RR provides (Rectangular, blue backgrounded steps in Figure 1). Other steps of the algorithm (Elliptic, green backgrounded steps in Figure 1) are basically gives the same performance due to using the same programming language and code on the same machine.

1.2 Deep Learning Idea

Our next aim is improving the performance of these steps eliminating them and instead, using a Deep Learning approach with a trained model. We use a Convolutional Neural Network(CNN) since we are dealing with images and our model will solve a regression problem since we are generating continuous messages rather than classification. Since Raspberry Pi 2 Model B is not powerful enough for a deep learning image processing, we use Intel Movidius Neural Compute Stick for trained model inferences from camera images. After creating a Duckietown map as in Figure 2 and let the Duckiebot travel in this map using the already implemented RR lane following algorithm, data is collected with saving the camera images and corresponding control messages into external memory.

2 Training Approach 1: Predict d and ϕ from Image

Please see Figure 3 to visualize this approach.

This kind of approach replaces some relatively heavy steps such as Line detection, Ground Projection and Lane Filter steps from the algorithm flow with a trained CNN model and predicts position d in range (-0.15,0.30) meters and orientation ϕ in range (-1.5,1.5) radian errors with respect to lane as a regression problem. The reason behind choosing this idea first, rather than an end-to-end approach, is that extracting only d and ϕ information can give more flexibility in terms of controller and setting different speed values for the robot which might be useful for later applications.

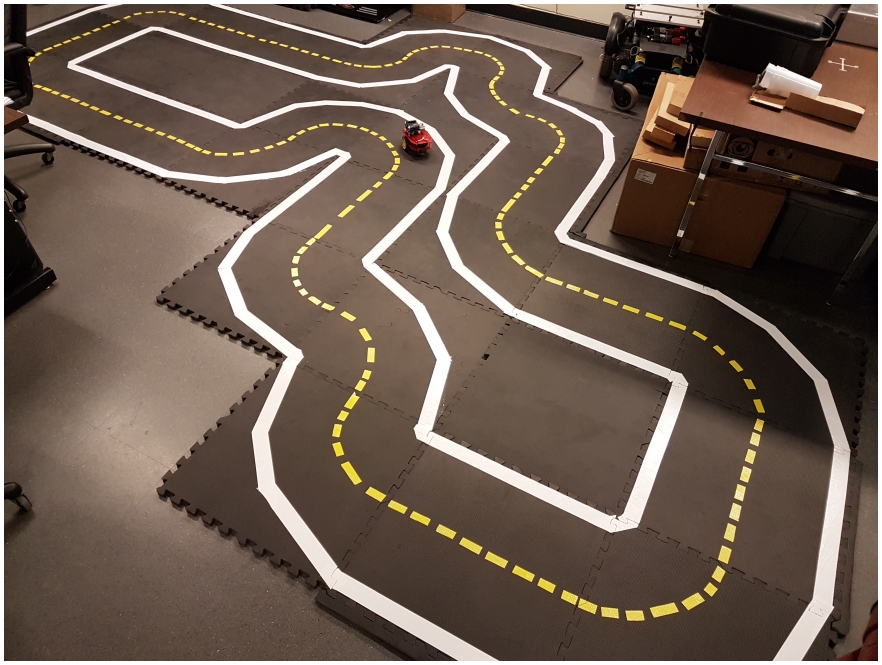


Figure 2: Training Map.

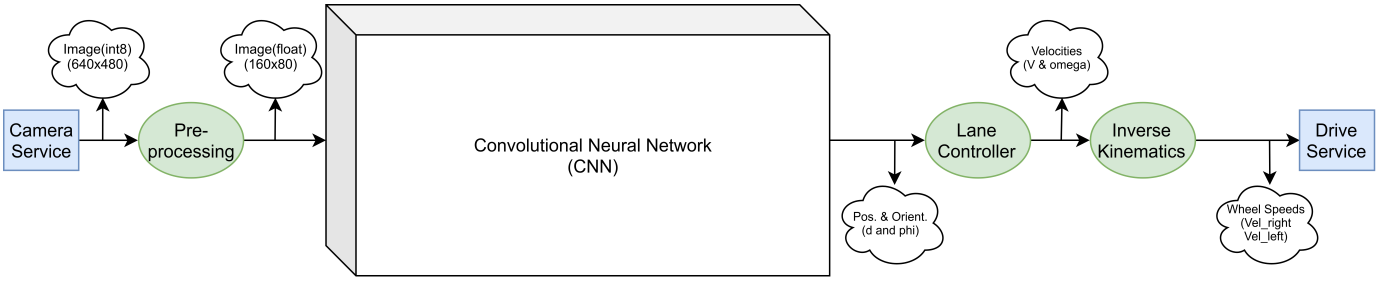


Figure 3: Lane Following algorithm overview flow with 1st Approach CNN.

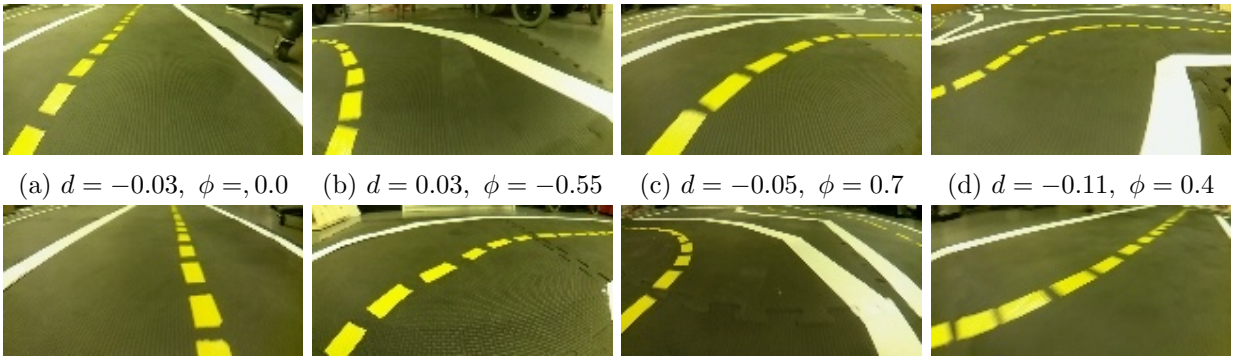
2.1 Training Data

Data for this approach has 9010 colored(3 channel) images in total with size 160x80. Data is divided as 8000 training and 1010 validation images, and they are labeled with their corresponding d and ϕ values. Some sample images from data can be seen on Figure 4.

This data obtained saving the images when Duckiebot travel in the map three loops in one direction and three loops in the other direction. 3483 images are saved during a relatively faster travel to capture the larger d and ϕ values and 5527 images are saved during a normal speed travel.

2.2 CNN Architecture 1

Overall, The first implemented architecture (CNN Architecture 1) takes 160x80x3 images as input and has 4 convolutional layers with filter sizes (7x7x3x16), (5x5x16x32), (5x5x32x64), (5x5x64x64) respectively. Each convolution layer is followed by a Max-Pooling layer with a window size (2x2) with stride 2. After convolution and max-pooling layers, tensor is flattened to a vector with length 3200. A fully connected layer is added with size 1024 and then d and ϕ is calculated as output. A visualization of this architecture can be seen on Figure 5



(a) $d = -0.03,\ \phi =, 0.0$ (b) $d = 0.03,\ \phi = -0.55$ (c) $d = -0.05,\ \phi = 0.7$ (d) $d = -0.11,\ \phi = 0.4$

(e) $d = 0.13,\ \phi = 0.1$ (f) $d = -0.05,\ \phi = 0.65$ (g) $d = -0.03,\ \phi = 0.05$ (h) $d = -0.07,\ \phi = 0.75$

Figure 4: Some sample training images and their labels.

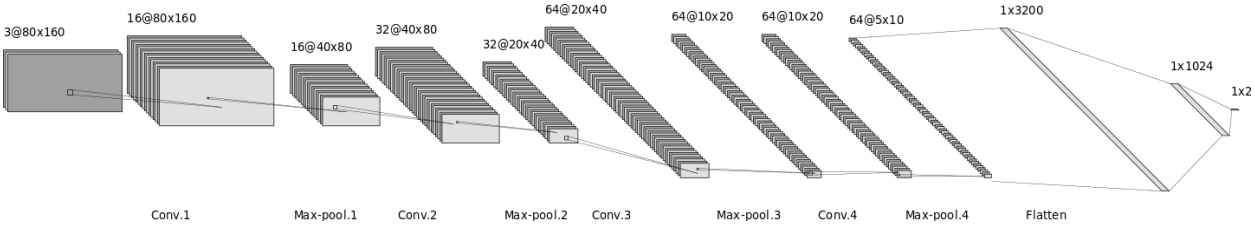
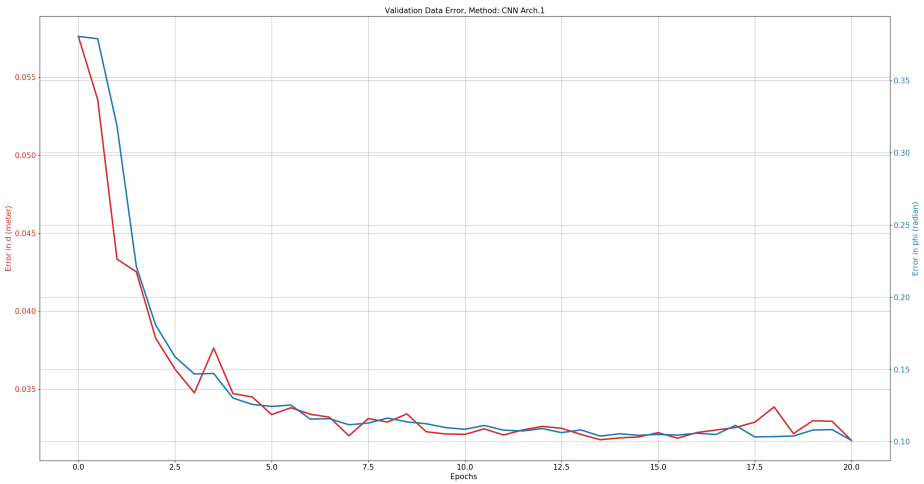


Figure 5: CNN Architecture 1 Visualization.

2.2.1 CNN Architecture 1: Error Results

Obtained training results of CNN Architecture 1 in terms of validation data average error values in d and ϕ with respect to epochs passed can be seen at left of figure 6. the right side of the Figure 6 shows resulted error values. Based on these results, we obtained in average 3.25 cm error in d and 6.45 degree error in ϕ for prediction from the camera images.



(a) Errors wrt. Epochs

	d(cm)	phi(deg)	phi(rad)
Errors	**3.25**	**6.45**	0.1125

(b) Resulted Errors

Figure 6: Validation Data Errors with CNN Architecture 1

2.2.2 CNN Architecture 1: Timing Results

Since we are trying to improve the performance of algorithm, it is important to see the timing results of Lane following algorithm with CNN and compare it with the existing RR performances. What we observe from Figure 7 is that we decreased the loop time by approximately 4.5 ms which is not a sufficient improvement considering the error results. In the next section, we will examine another CNN architecture to decrease the inference time of the CNN model without increasing the error rates.

	Image Acquisition + Preprocessing	Prediction + Control	Set Wheel Speed	TOTAL	FPS
RR	48.12 (ms)	26.59 (ms)	15.21 (ms)	89.92 (ms)	11.12
RR + CNN Arch 1	48.12 (ms)	21.94 (ms)	15.21 (ms)	85.27 (ms)	11.73

Figure 7: CNN Architecture 1 Timing Performance.

2.3 CNN Architecture 2

With our new CNN architecture we built a simpler model. Overall, our new architecture (CNN Architecture 2) again takes 160x80x3 images as input and has 4 convolutional layers with filter sizes (7x7x3x16), (5x5x16x32), (5x5x32x64), (5x5x64x64) respectively. This time, however, convolution layers are not followed by Max-Pooling layers. Instead, we will apply convolutions with stride 2 and the obtained tensor sizes will be the same. After strided convolution layers, the tensor again flattened to a vector with length 3200. A fully connected layer is added with size 1024 and then d and ϕ is calculated as output. A visualization of new simpler architecture can be seen on Figure 8

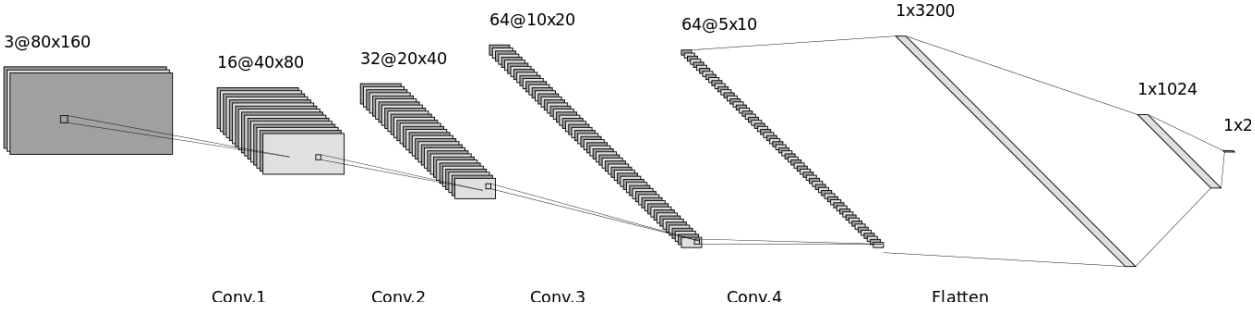
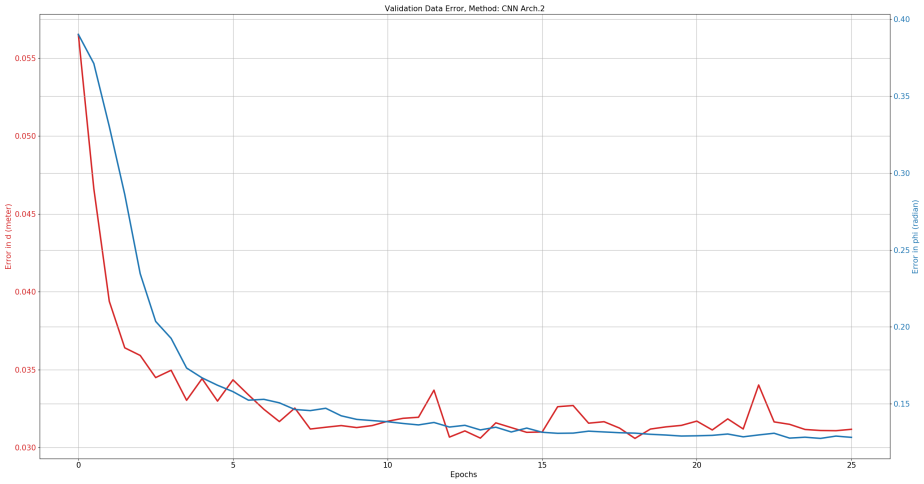


Figure 8: CNN Architecture 2 Visualization.

2.3.1 CNN Architecture 2: Error Results

Obtained training results of CNN Architecture 2 in terms of validation data average error values in d and ϕ with respect to epochs passed can be seen at left of figure 9. the right side of the Figure 9 shows resulted error values. Based on these results, we obtained in average 3.11 cm error in d and 7.34 degree error in ϕ for prediction from the camera images. Compared with the previous architecture, this one performed slighlty better in d and slightly worse in ϕ. Therefore, the timing performance will be more decisive for which one to be used.



(a) Errors wrt. Epochs

	d(cm)	phi(deg)	phi(rad)
Errors	**3.11**	**7.34**	0.1281

(b) Resulted Errors

Figure 9: Validation Data Errors with CNN Architecture 2

2.3.2 CNN Architecture 2: Timing Results

Since we are trying to improve the performance of algorithm, it is important to see the timing results of Lane following algorithm with CNN and compare it with the existing RR performances. What we observe from Figure 10 is that we decreased the loop time by approximately 10.5 ms with respect to RR and approximately 6 ms with respect to CNN Arch.1. Since this result with 12.6 FPS can be considered as a performance improvement for our system, we decided to use CNN Architecture 2.

	Image Acquisition + Preprocessing	Prediction + Control	Set Wheel Speed	TOTAL	FPS
RR	48.12 (ms)	26.59 (ms)	15.21 (ms)	89.92 (ms)	11.12
RR + CNN Arch 1	48.12 (ms)	21.94 (ms)	15.21 (ms)	85.27 (ms)	11.73
RR + CNN Arch 2	48.12 (ms)	15.87 (ms)	15.21 (ms)	79.2 (ms)	12.62

Figure 10: CNN Architecture 2 Timing Performance.

2.4 Real World Experimentation with Training Approach 1

Until now, we mainly focused on finding the best possible fast and accurate CNN model without actual experimenting the lane following algorithm with CNN. Using the discussed models above, we let the Duckibot travel, however none of the models with Training Approach 1 did not perform a sufficiently successful lane following.

In the next section, we will modify our training approach to a new one.

3 Training Approach 2 (End-to-end): Predict V_{left} and V_{right} from Image

Please see Figure 11 to visualize this approach.

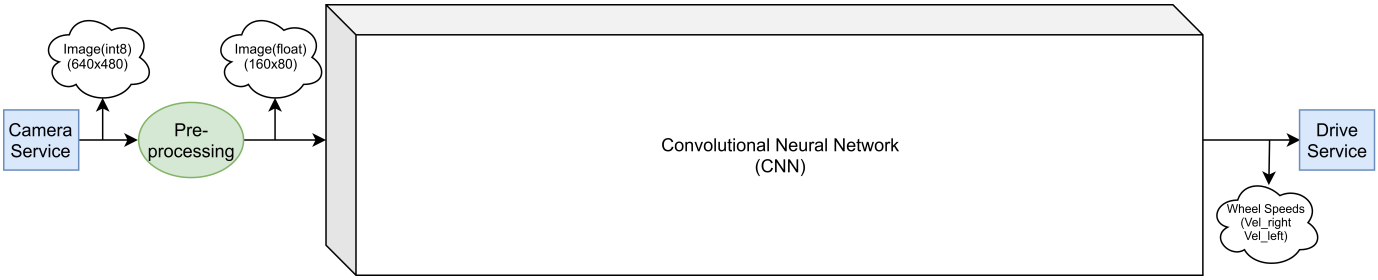


Figure 11: Lane Following algorithm overview flow with 2nd Approach CNN.

This approach replaces Lane Controller and Inverse Kinematics steps in addition to the relatively heavy in the previous approach. Algorithm flow with a trained CNN model and predicts V_{left} and V_{right} values both in range $(-14, 58)$ for our training data as a regression problem. Speed value 100 means full speed of the wheels in forward direction and -100 means full speed in the reverse direction. Neutral going forward speed is approximately 22.5. Mean and standard deviation of whole speed data for V_{left} and V_{right} are observed as 22.5 and 10.2 respectively.

3.1 Training Data

Data for this approach has 5378 colored(3 channel) images in total with size 160x80. Data is divided as 4950 training and 428 validation images, and they are labeled with their corresponding V_{left} and V_{right} values.

This data obtained saving the images when Duckiebot travel in the map three loops in one direction and three loops in the other direction. In this approach since we are directly learning the required wheel speeds, all data saved during normal speed travel.

3.2 Approach 2: CNN Architecture

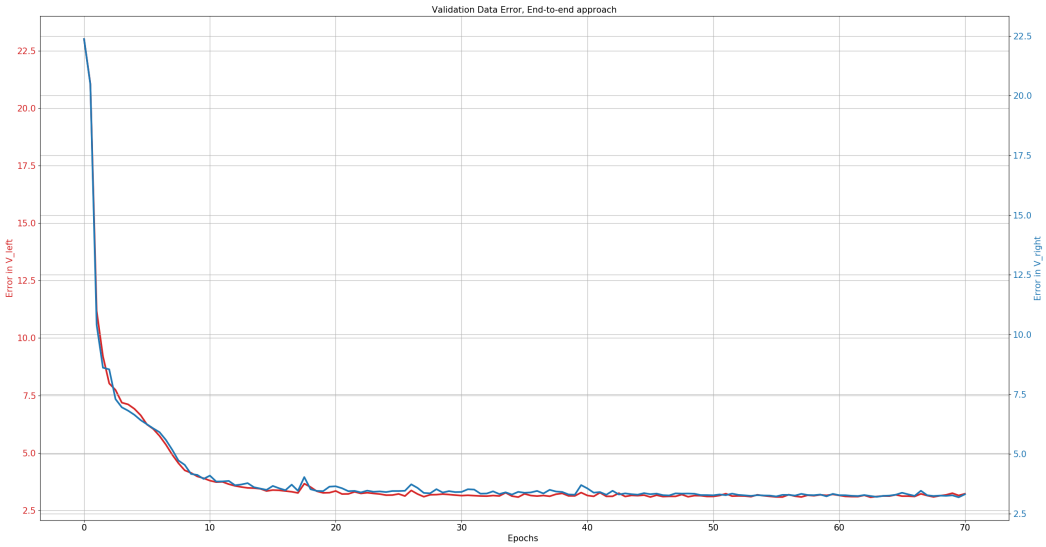
We will use CNN Architecture 2 examined in the previous section due to its observed time performance improvement.

3.2.1 Approach 2: Error Results

Obtained training results of Approach 2 in terms of validation data average error values in V_{left} and V_{right} with respect to epochs passed can be seen at left of figure 12. the right side of the Figure 12 shows resulted error values.

3.2.2 Approach 2: Timing Results

Overall timing results can be seen at Figure 13. Since our new approach eliminates simple steps as well, we observed a small improvement in time relative to best timing performance of previous approach.



(a) Errors wrt. Epochs

	V_left	V_right
Errors	3.22	3.31

(b) Resulted Errors

Figure 12: Validation Data Errors with Training Approach 2

		Image Acquisition + Preprocessing	Prediction + Control	Set Wheel Speed	TOTAL	FPS
	RR	48.12 (ms)	26.59 (ms)	15.21 (ms)	89.92 (ms)	11.12
Approach 1	RR + CNN Arch 1	48.12 (ms)	21.94 (ms)	15.21 (ms)	85.27 (ms)	11.73
	RR + CNN Arch 2	48.12 (ms)	15.87 (ms)	15.21 (ms)	79.2 (ms)	12.62
Approach 2	RR + CNN Arch 2	48.12 (ms)	15.16(ms)	15.21 (ms)	78.49	12.74

Figure 13: Timing Performances.

3.3 Real World Experimentation with Training Approach 2 and Conclusions

End-to-end approach showed a observable lane following real world performance which can be seen in here. When we compare the Deep Learning accuracy of lane following with RR and ROS, we observe that Deep Learning approach did not perform as accurate as explicitly implemented algorithms. The further improvements can be achieved with rummaging the data and remove noisy or wrongly labeled images or finding a better CNN architecture.