Programming an Autonomous Driving Car

Kamyab Mirhosseini
[1] *University of Nottingham, School of Physics and Astronomy, Nottingham, United Kingdom*

This report is part of a Machine Learning project explaining the programming of an autonomous driving car. We have employed several techniques such as end-to-end deep learning, object classification, and transfer learning in order to predict for steering angles and speed, based on images of a basic track. The system learns to drive along the track following the lane markings and stop at obstacles on the road. It learns to stop and go at traffic lights and turn left or right according to the signals on the road.

Keywords: Deep learning, Convolutional neural networks, Supervised learning

INTRODUCTION

An autonomous car is a vehicle capable of sensing its environment and operating accordingly without the involvement of human input. Some of the main benefits of using autonomous cars is to reduce traffic congestion and transportations costs (with each autonomous car providing an estimate of 2000 – 4750 USD per year in societal benefits), reduce traffic accidents, free up parking and most importantly reduce urban CO2 emissions worldwide (Fagnant & Kockelman, 2015).

In the past, before the presence of Deep learning and Convolutional neural networks (CNNs), handcrafted computer vision systems such as HAAR (Viola & Jones, 2001) and Local binary patterns (Ojala et al., 1996) were used to extract features of images and then classify them into categories. CNNs have revolutionized the detection of patterns where features are learned automatically from training examples. CNNs have been one of the first revolutionary approaches to solve commercial problems for almost 20 years (Jackel et al., 1995).
In this report we discuss both the basic and more recently published approaches to predict speed and steering angles based on images.

The approaches used to tackle this task are rule-based learning where the system is given certain rules to predict for angle and speed in given situations, and the end-to-end approach where the system learns automatically to predict for angle and speed by learning from the extracted features of images. The choice of algorithm for each of these approaches include the Machine Learning (ML) algorithms; Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Aim and overview of the project

The aim of this project was to predict steering angles and speed, based on images of the track. An end-to-end supervised learning approach was used in a two-phased process of object classification and the prediction of steering angle and speed (using Transfer Learning (TL) to further train the models).

MATERIALS AND METHODS

The dataset used for training included 13793 images collected by driving around a basic track with traffic lights, arrows and various objects, where corresponding steering angles and speed were obtained. A total of 1020 raw images were later used for testing the models and making predictions. In order to normalise the angle and the speed predictions the following steps were taken

$$angle_{norm} = \frac{angle - 50}{80} \tag{1}$$

$$speed_{norm} = \frac{speed - 0}{35} \tag{2}$$

that would result in both predictions to be a value between 0 and 1. Furthermore, the values of speed were further simplified to a binary value of either 0 or 1 with the threshold of 0.5.

A variety of different resources were used in order to solve this project including Tensorflow Keras with functional API used in most of the models, except the Nvidia model where sequential API was used (Abadi et al., 2016). The different layers used were CNNs, Dropout (Srivastava et al., 2014), Dense and Batch Normalisation layers, with activation functions sigmoid and Exponential Linear Unit (ELU) (which was used instead of ReLU because of having a curve that eliminates vanishing gradient). In our models, we made use of the ADAM optimizer (Kingma & Ba, 2014), which is an algorithm for stochastic gradient based optimization based on lower order moments.

Early stopping was implemented to stop the model training where improvement is not observed in the validation loss after a set number of epochs. As well as early stopping, checkpoint callbacks were also used in order to save each checkpoint separately to allow us to manually go back to a saved checkpoint in the event of occurrence of overfitting.

Augmentations

As we observed that working on the original dataset can be improved by data augmentation, we used different levels of augmentation in order to find out which ones produce the best result. The augmentations used in our

models consisted of Gaussian noise, blur, zoom, brightness, exposure, pan, flip and rotation. We flipped the entire dataset to have double the amount of training images in order to eliminate bias towards the right turns, training angles were also flipped by deducting the value of the angle from 1 (E.g. $1 - 0.3 = 0.7$).
Rotation was carefully implemented in order to prevent mismatching of the angles. Furthermore, we created an additional subset of augmented images of under-represented scenarios such as the arrow signals, green traffic lights and female pedestrians on the road in order to have a better representation of them.

Object Classification
We labelled the objects using LabelImg Graphical Image Annotation and Roboflow. We labelled a sufficient number images of all the objects seen in the training dataset. (1. Person 2. Green Light 3. Red Light 4. Car 5. Box 6. Tree 7. Left Signal 8. Right Signal), and split the labels into two subsets of speed and angle related labels.

Transfer Learning (TL)
The purpose of using transfer learning is to connect the features learned in lower layers of a model and transfer them to another dataset (Taylor & Stone, 2009). We used inductive transfer learning to connect our models together. This was done by removing the last layers of each model and concatenating the last layers and adding extra dense layers as needed. We froze the entire model except the new added layers and we trained the new layers for a few epochs to warm up the new layers. We then unfroze the entire model and trained with a low learning rate. This was done in different approaches and a combination of learning rates and dense layers.

The Nvidia Model (Bojarski et al., 2016)
The Nvidia model was the first implemented model as a learning ground for the project which helped us get familiar with the dataset and the process and move on to a more complex model. We changed the output dense layer of the Nvidia model to give 2 predictions instead of 1, i.e. speed and steering angle. A number of different input preprocessings were tested to obtain better results. We observed that for this model, inputting the full image resized to half of its original size resulted in a loss of less than 0.05. An interesting observation at this point was that only the bottom half of the image is good enough to make predictions for steering angle. This idea was then implemented in our later models.

Moving forward from the Nvidia model, we developed a model inspired by the structure of the VGG model (Simonyan & Zisserman, 2014) in which the layers are structured in blocks of two convolutional layers followed by a maxpool layer.

The Branched Model
This model uses two different branches to predict angle and speed separately where the last layer is concatenated. As seen in Figure 1, the model receives two different preprocessed input images in each branch, with the left branch receiving the bottom half grayscale image and the right branch receiving the coloured full image, and both images are resized to 50%.

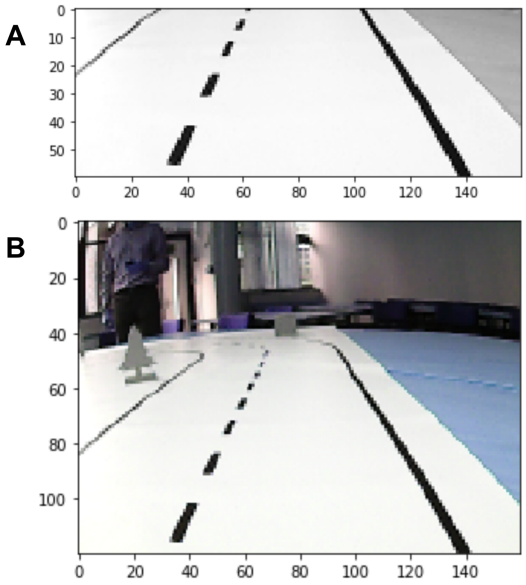


Figure 1. **A:** The input image of the left channel used to predict angle **B:** The input image of the right channel used to predict speed

We developed this model with the idea of having two separate branches predict either speed or steering angle, since the scenario includes arrows, we had to create a branch that can make use of both inputs (half image predicting angle and the full image reading the arrows). Appendix A shows the structure of the branched model where the angle side receives a branch from the speed side in order to be able to detect the arrows.

Twin Engine Model
In this model, we had two independent models combined to predict speed and angle. Each unit was trained in two phases and concatenated to make the final model. The structure of the model is shown as in Appendix B. We preprocessed the images for each model separately. Based on what we observed in the Nvidia model where the bottom half of the image was sufficient to predict angle, we decided to invert the bottom third of the image with the hope that the road in front of the vehicle would be observed and the model would learn the association of the angle and the shape of the road in front of the car. Therefore, the angle model received a grayscale image with inverted lower third, and the speed model received the coloured full image. Both images were also resized to half. (See Fig. 2)

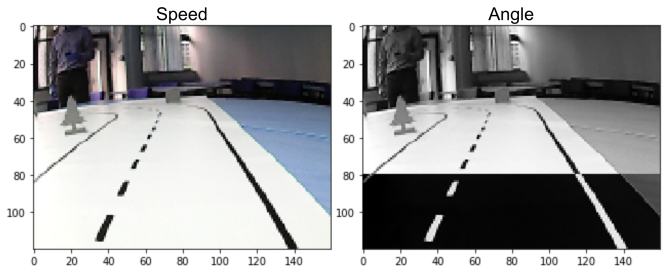


Figure 2. The input given to each channel of the Twin Engine model

Training was done in 2 phases. In phase 1, we trained each model separately for object classification. The angle unit was trained on 5 objects including left and right arrows. The remaining three labels included the road itself to further improve the association of the road shape and the angles (As seen in Figure 3). In this model we changed the last dense layer to a sigmoid with 5 units.

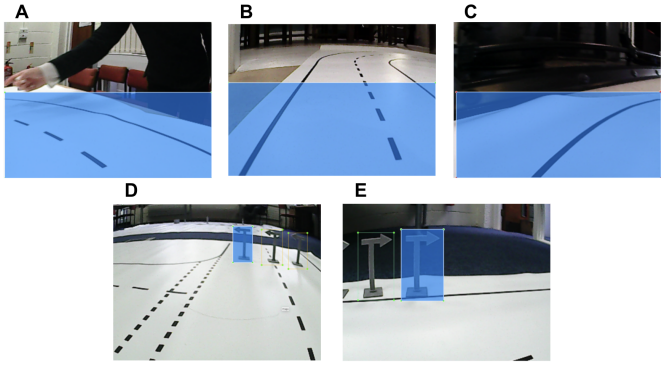


Figure 3. Angle labels with **A-C** corresponding to extreme left, straight and extreme right, and **D-E** corresponding to the arrows

Similarly we trained another model with the same structure except the last dense layer which was changed to a sigmoid with 6 units and trained to detect objects including obstacles and traffic lights. Furthermore, in phase 2 we used TL to make use of the learning obtained in phase 1 and further training to predict angle and speed separately. We replaced the last layers of each model with a dense layer having a single unit to predict angle and speed independently. We froze the entire model except the new added layer, and we trained the model for 1 epoch at a learning rate of 0.001 to warm up the new layer. We then unfroze the entire model and trained with a low learning rate of 0.00001 for 50 epochs.

The Simple Model

At this point we wanted to see if a simple model could train to learn to predict for both steering angle and speed. The input given to this model was only a resized and normalised coloured full image. The model structure also follows the same structure of the VGG with blocks of two convolutional layers and a maxpool layer.

RESULTS

After running the models for an estimate of 50 epochs each, we observed some interesting results for each model with three all models having a validation loss of less than 0.01.
The branched model successfully obtained a validation loss of 0.006 as seen in Figure 4.

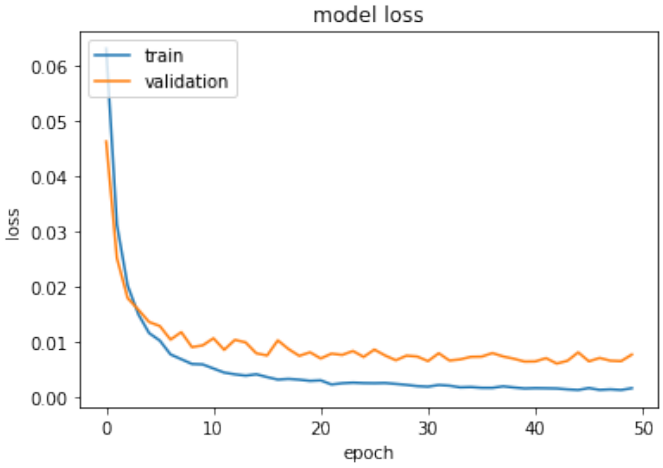


Figure 4. Training and validation Loss of the branched model after 50 epochs; Training Loss: 0.001 Validation Loss: 0.006

Figure 5 illustrates the results of the twin engine model. Figure 5A and 5B are the results of the phase 1 of training for speed and steering angle respectively. Figure 5C and 5D are further training of both models after implementing TL where the speed model achieved a validation loss of 0.02913 and the angle model achieved a validation loss of 0.00539.

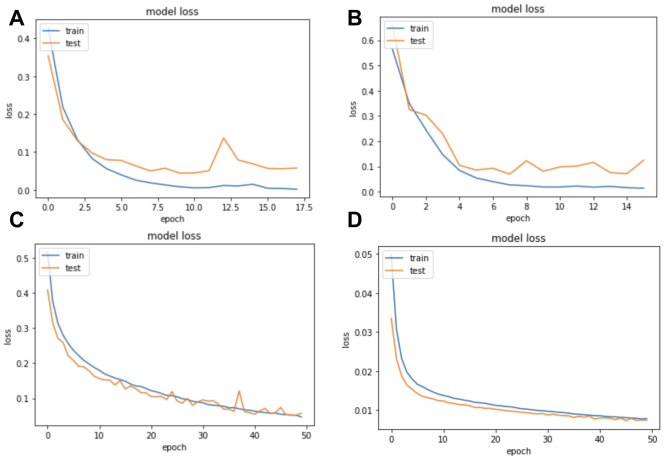


Figure 5. (A) Speed phase 1 (B) Angle phase 1 (C) Speed with TL (D) Angle with TL Validation loss for angle: 0.00539 Validation loss for speed: 0.02913

The simple model was also able to successfully achieve a validation loss of 0.00529 as seen in Figure 6A. From this point we used this model as a base line to carry out further experiments such as removing the dropout layer. Figure

6B is a plot of the loss after removing the dropout layer and this achieved a validation loss of 0.006.

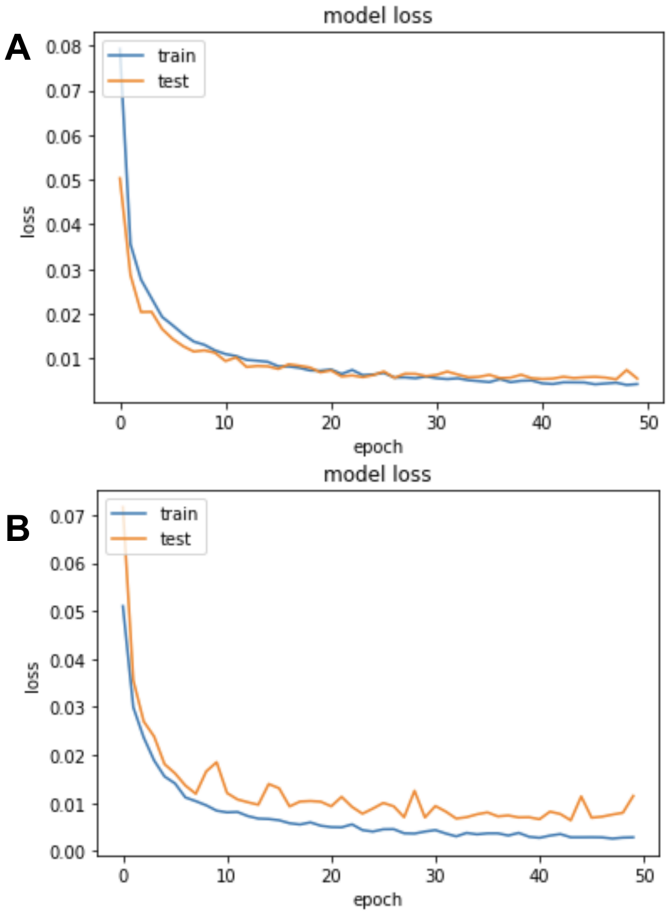


Figure 6. Training and validation loss of the simple model after 50 epochs; **A:** With dropout layer Validation Loss = 0.00529 **B:** Without dropout layer Validation Loss = 0.006

Figure 7 shows real predictions on the test data using the simple model (with the dropout layer).

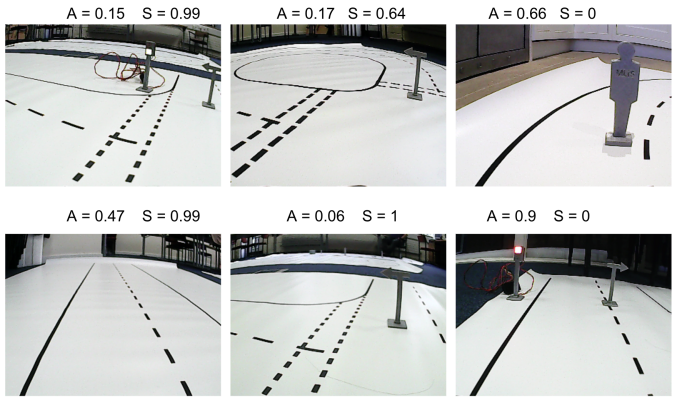


Figure 7. Real predictions of angle (A) and speed (S) on the test data using the simple model with dropout layer

In order to prepare for the live test, we ran our model through Autopilot and found out the inference time needs to be reduced. As the number of parameters in our models were very large we decided to reduce them by adding a filter size of of (5, 5) and a stride of (2, 2). By doing this the number of model parameters were reduced and the inference time improved from an estimate of 140ms to 40ms.

Table I. Validation loss comparison of the Twin Engine model when implementing object classification by transfer learning and without object classification

With object classification		Without object classification	
Speed	Angle	Speed	Angle
0.02913	0.00539	0.0354	0.00363

DISCUSSION

During the process of this project we observed that it is important to consider having a properly represented dataset where all scenarios are equally represented. Under-representations in the dataset caused one of our test models to have less accuracy when predicting for under-represented scenarios.

Another important point we observed was that by developing our own custom data generator we gain an advantage of having more control over the input of the model.

When retraining the model on small subsets of training images such as small scenarios, we need to consider including all the basic scenarios as well. This is because the model may forget the previous learning obtained.

After training the model with and without object classification we observed the following results: As we can see in Table 1. in contrast to our expectations, the implementation of object classification did not have a compelling effect on the losses. This might be due to the fact that we did not implement object detection fully. The use of bounding boxes and training for object detection and minimizing the intersection over union (IoU) value can be used in future projects to improve the performance of object detection.

Furthermore, we experimented more with the simple model and trained a version with the addition of dropout on dense layers. This resulted in a total loss of 0.01404 where in our case was not an improvement to the current model.

During the project we further observed that over-augmentation can happen and distort the training images. In this case scenarios such as red and green light or left and right signals could not be distinguished. Therefore learning these scenarios would not be effective, therefore over-augmented images required to be removed.

We also further implemented L2 regularizer on each

model to observe any improvement, however as the purpose of regularizers is to prevent the weights from becoming too large, our models did not show improvement as the large weights may not have been seen in the model.

In conclusion we found that the simple end-to-end approach was good enough to obtain good results and was light enough to be implemented in the live test with good inference time. In future studies rule based and reinforcement learning can be used to tackle this problem, and possibly object detection with full use of bounding boxes.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). Tensorflow: A system for large-scale machine learning. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zieba, K. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

Fagnant, D. J., & Kockelman, K. (2015). Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transportation Research Part A: Policy and Practice, 77, 167-181.

Jackel, L. D., Sharman, D., Stenard, C. E., Strom, B. I., & Zuckert, D. (1995). Optical character recognition for self-service banking. AT&T technical journal, 74(4), 16-24.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

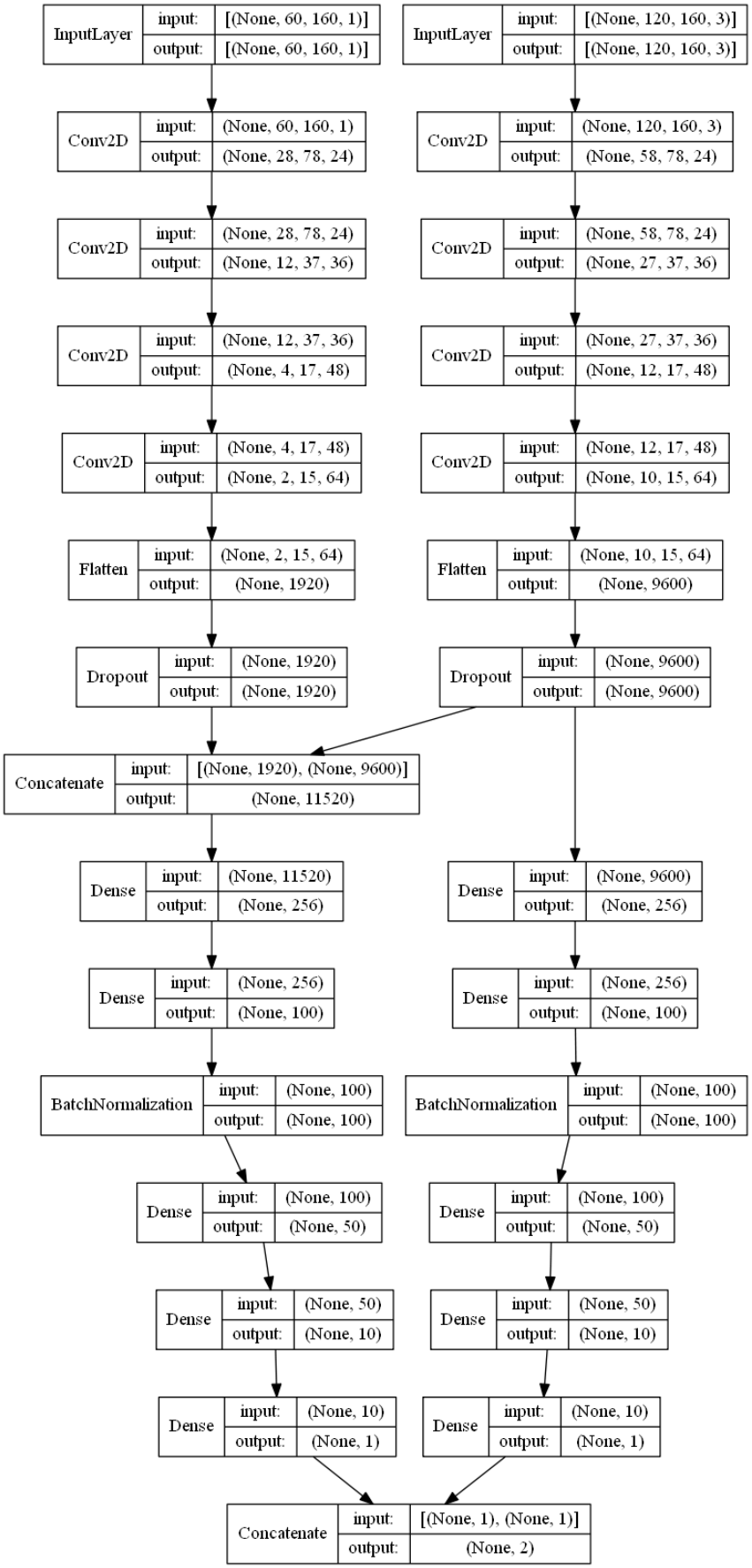
Ojala, T., Pietikäinen, M., Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. Pattern recognition, 29(1), 51-59.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

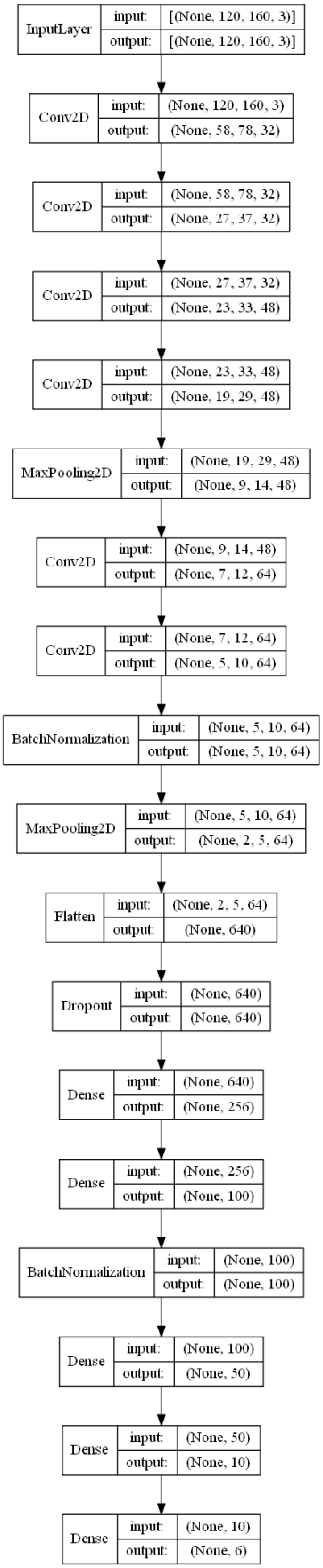
Taylor, M. E., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research, 10(7).

Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). IEEE.

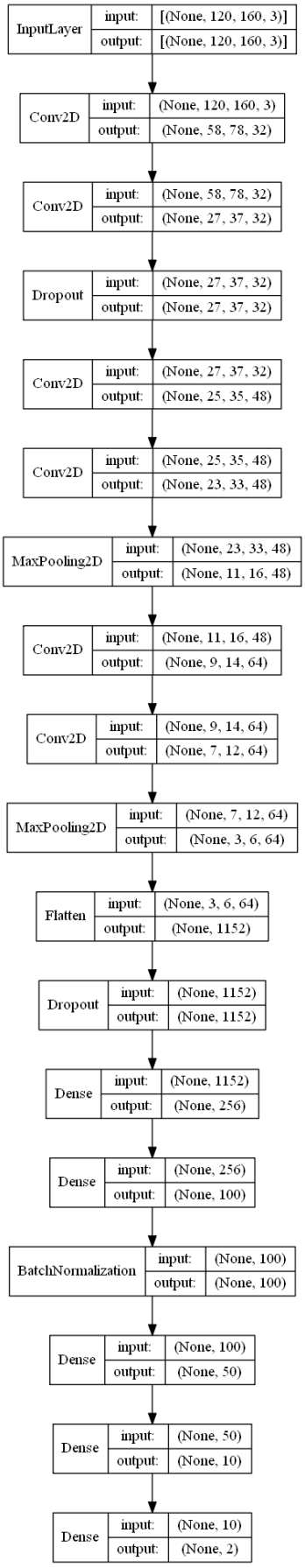
Appendix A


InputLayer | input: [(None, 60, 160, 1)] | output: [(None, 60, 160, 1)]
InputLayer | input: [(None, 120, 160, 3)] | output: [(None, 120, 160, 3)]
Conv2D | input: (None, 60, 160, 1) | output: (None, 28, 78, 24)
Conv2D | input: (None, 120, 160, 3) | output: (None, 58, 78, 24)
Conv2D | input: (None, 28, 78, 24) | output: (None, 12, 37, 36)
Conv2D | input: (None, 58, 78, 24) | output: (None, 27, 37, 36)
Conv2D | input: (None, 12, 37, 36) | output: (None, 4, 17, 48)
Conv2D | input: (None, 27, 37, 36) | output: (None, 12, 17, 48)
Conv2D | input: (None, 4, 17, 48) | output: (None, 2, 15, 64)
Conv2D | input: (None, 12, 17, 48) | output: (None, 10, 15, 64)
Flatten | input: (None, 2, 15, 64) | output: (None, 1920)
Flatten | input: (None, 10, 15, 64) | output: (None, 9600)
Dropout | input: (None, 1920) | output: (None, 1920)
Dropout | input: (None, 9600) | output: (None, 9600)
Concatenate | input: [(None, 1920), (None, 9600)] | output: (None, 11520)
Dense | input: (None, 11520) | output: (None, 256)
Dense | input: (None, 9600) | output: (None, 256)
Dense | input: (None, 256) | output: (None, 100)
Dense | input: (None, 256) | output: (None, 100)
BatchNormalization | input: (None, 100) | output: (None, 100)
BatchNormalization | input: (None, 100) | output: (None, 100)
Dense | input: (None, 100) | output: (None, 50)
Dense | input: (None, 100) | output: (None, 50)
Dense | input: (None, 50) | output: (None, 10)
Dense | input: (None, 50) | output: (None, 10)
Dense | input: (None, 10) | output: (None, 1)
Dense | input: (None, 10) | output: (None, 1)
Concatenate | input: [(None, 1), (None, 1)] | output: (None, 2)

Appendix B


InputLayer
input: [(None, 120, 160, 3)]
output: [(None, 120, 160, 3)]
Conv2D
input: (None, 120, 160, 3)
output: (None, 58, 78, 32)
Conv2D
input: (None, 58, 78, 32)
output: (None, 27, 37, 32)
Conv2D
input: (None, 27, 37, 32)
output: (None, 23, 33, 48)
Conv2D
input: (None, 23, 33, 48)
output: (None, 19, 29, 48)
MaxPooling2D
input: (None, 19, 29, 48)
output: (None, 9, 14, 48)
Conv2D
input: (None, 9, 14, 48)
output: (None, 7, 12, 64)
Conv2D
input: (None, 7, 12, 64)
output: (None, 5, 10, 64)
BatchNormalization
input: (None, 5, 10, 64)
output: (None, 5, 10, 64)
MaxPooling2D
input: (None, 5, 10, 64)
output: (None, 2, 5, 64)
Flatten
input: (None, 2, 5, 64)
output: (None, 640)
Dropout
input: (None, 640)
output: (None, 640)
Dense
input: (None, 640)
output: (None, 256)
Dense
input: (None, 256)
output: (None, 100)
BatchNormalization
input: (None, 100)
output: (None, 100)
Dense
input: (None, 100)
output: (None, 50)
Dense
input: (None, 50)
output: (None, 10)
Dense
input: (None, 10)
output: (None, 6)

Appendix C


InputLayer
input: [(None, 120, 160, 3)]
output: [(None, 120, 160, 3)]
Conv2D
input: (None, 120, 160, 3)
output: (None, 58, 78, 32)
Conv2D
input: (None, 58, 78, 32)
output: (None, 27, 37, 32)
Dropout
input: (None, 27, 37, 32)
output: (None, 27, 37, 32)
Conv2D
input: (None, 27, 37, 32)
output: (None, 25, 35, 48)
Conv2D
input: (None, 25, 35, 48)
output: (None, 23, 33, 48)
MaxPooling2D
input: (None, 23, 33, 48)
output: (None, 11, 16, 48)
Conv2D
input: (None, 11, 16, 48)
output: (None, 9, 14, 64)
Conv2D
input: (None, 9, 14, 64)
output: (None, 7, 12, 64)
MaxPooling2D
input: (None, 7, 12, 64)
output: (None, 3, 6, 64)
Flatten
input: (None, 3, 6, 64)
output: (None, 1152)
Dropout
input: (None, 1152)
output: (None, 1152)
Dense
input: (None, 1152)
output: (None, 256)
Dense
input: (None, 256)
output: (None, 100)
BatchNormalization
input: (None, 100)
output: (None, 100)
Dense
input: (None, 100)
output: (None, 50)
Dense
input: (None, 50)
output: (None, 10)
Dense
input: (None, 10)
output: (None, 2)