University of Freiburg

Deep Learning Lab

Neurorobotics/Robotics

Assignment 3

Author:
Mostafa Hussein
Eduardo Alvarado

Supervisor:
Andreas Eitel

December 18, 2017

Abstract

This report is summarizing the task of Assignment 3 which was implementing a convolutional neural network using *tensorflow* and *keras* package for visual motion planning (imitating A* Algorithm). The report is divided into 4 sections as follows: Section 1 represents a simple explanation of the implementation, section 2 represents the designed CNN, section 3 represents the results and the evaluation of the performance, and section 4 represents some improvements.

1 Convolutional Neural Network

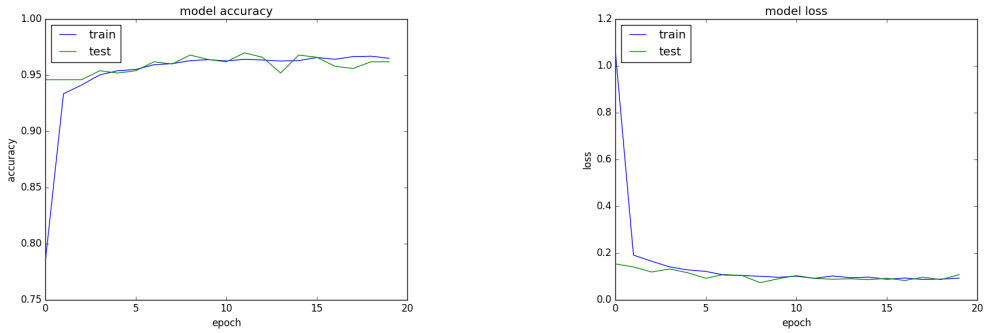
This exercise was to implement an agent using a CNN to perform visual planning and achieve maximum performace (as close to A* as possible). Firstly, the training data were loaded by loading *get_data*, and then agent was trained with this dataset. A *Simulator* and a *TransitionTable* were initialized at the beginning, and we begin by saving the training and validation dataset using the *get_train* and *get_valid* functions from the *TransitionTable*. The model is initialized using *keras* package, which is an API that acts over *TensorFlow*. *Dense*, *Activation*, *Conv2D*, *MaxPooling2D*, *Dropout* and *Flatten* layers were imported from *keras.layers*.

2 Convolutional Neural Network Design

The designed CNN consists of three same padded convolutional layers with number of features 128, 64, 32 and *kernel_size* 5x5, 3x3, 3x3 respictevly, each followed by a *ReLu* activation function and a *MaxPooling2D* layer. Afterwards, we added a *Dropout* layer of 0.25. Then we have a *Flatten* layer, followed by a *Dense* layer of 512, a *ReLu*, and a *Dropout* of 0.5. Finally, we have our last *Dense* layer of 5, and a *softmax* activation for classification method, which will give out an output of five actions in total: stay still, up, down, right and left. Then the best action will be chosen according to the highest probability. The layer parameters were randomly tried out from a range of values, and these parameters we have, were the ones who gave out best results and performance. In addition, we used *Adam* optimizer with its default *learning_rate* and a *categorical_crossentropy* loss. The model was trained with *epochs* = 20 and a *batch_size* = 64. Moreover, the data were shuffled for each training epoch in order to have better performance.

3 Results

After training, we had a training error of 3.49%, training loss of 0.0956, validation error of, and 3.8% validation loss of 0.113. The plots for our results are shown below in Figure 1.



(a) Model Accuracy (b) Model Loss

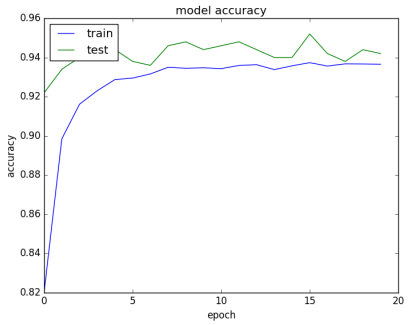
Figure 1: Model Accuracy and Loss Results

3.1 Local View

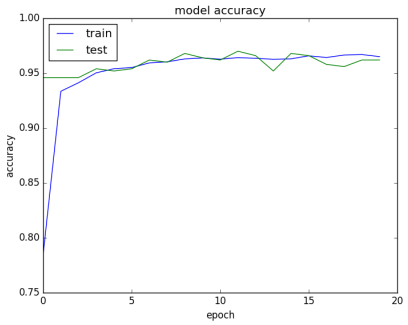
To test the implementation, we have to run the *test_agent* with our model loaded inside. Initially, *test_agent* was getting random actions to reach the goal, and obviously, these actions were not performing, and the agent never reached its goal. However, after training our agent, we loaded the model into the *test_agent* file, then we used the model to predict the actions based on the current state. From our point of view, the trained agent reaches the goal target in all states that it start with, which reflects that it has an accuracy of nearly 100%. That was concluded after running the *test_agent* several times and observing the results from starting at different random states, and seeing how the agent reaches its goal. However, in some few rare cases, the the results were not so optimal. To clarify, the agent was always reaching its goal, but sometimes, it takes few extra steps more than that was achieved by A$*$ algorithm.

3.2 Changing History Length or View Size

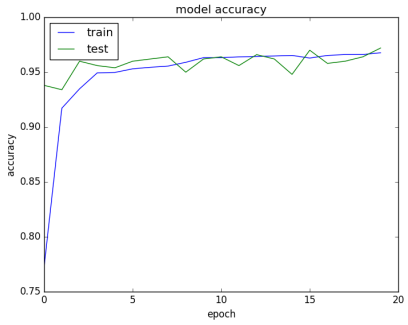
A second experiment is conducted on the network to see how will history length does affect the accuracy of the trained agent. Accordingly, we tried different values for history lengths, which are 1, 4, 7 and 10 respectively. It was expected that the history length is directly proportional with the accuracy, which means if it is decreased, the accuracy will decrease as well, however, if it is increased, the accuracy will also increase. As expected, an accuracy of 94.2% was achieved by having a history length of 1, which is lower than the accuracy of 96.2% achieved by having a history length of 4. Nevertheless, a history length of 7, gave a greater accuracy of 97.2%. In contrary to our expectations, increasing the history length to more than 7, did not give us a better accuracy at all. For example, history length of 10, gave us a lower accuracy of 95.8%. The results can be shown below in Figure 2.



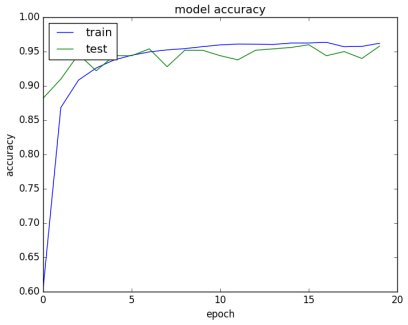
(a) History length = 1



(b) History length = 4



(c) History length = 7



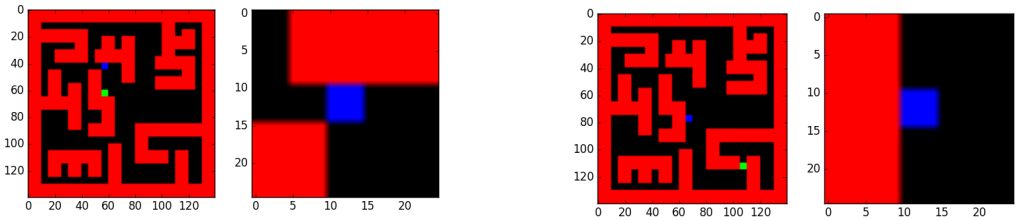
(d) History length = 10

Figure 2: Model Accuracy and Loss Results

In case of changing the view size, we are modifying also the scope/range which the agent can see. Accordingly, as expected, when we tried to run the model with a *prob_size* less than 5 (default), the agent did not perform that well, due to the fact, that its range of view was much lower. However, when we increased the view, the performance was a little bit better, because it had an insight of a bigger view along its way to reach the target.

3.3 Changing Target Location

As expected, changing the Target location would lead to a huge error, due to the fact that the agent is trained with a specific target location. Since we have trained our model according to one concrete target location as in Figure 3a, if we just change it after our training, the model will not work as expected. The trained model will work only for that exact target location, because it is trained to reach exactly that point; otherwise, if we change the target as in Figure 3b, the agent will not know what to do once it reaches the spot (just move around the point where the target is supposed to be).



(a) Default Target Location

(b) Modified Target Location

Figure 3: Changing Target Location

3.4 Changing the Map

Since we have trained our model according to one specific configuration of the environment, the agent e.g. will face unexpected obstacles in the way for the target location, or will not take the optimal path as A* does. Therefore, if we change the map after training, the model will not work.

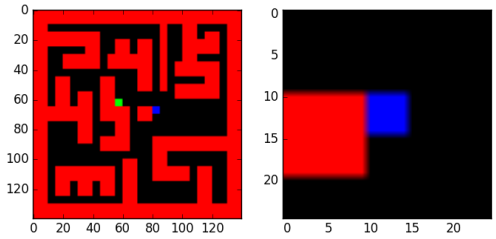


Figure 4: Changing the map

In the above Figure 4, we can appreciate that we modified the map, to have some additional obstacles, like the one in the right frame. The agent does not know how to act, when it reaches some of these points, since the map is different to the one we previously trained on.

4 Improvements

In order to improve our agent to be able to solve any Map and reach any Target location, two improvements are suggested. Firstly, instead of training on just one target location, training is done instead by considering all the locations in the map as target locations, and with different map shapes. After that, the training locations as well as the maps are augmented in the training dataset of the agent. For example, if we decide to use one map, we could train it for every possible target within that environment. Obviously, the major disadvantage of this method, is the computing power required to run such training, which would be huge and probably impossible to carry out, moreover, the size of the dataset will become so huge and will increase exponentially.

The second suggested solution and the more efficient one is by using reinforcement learning which is done by making the agent train itself using rewards, and using the Q-learning method and get the sum of the future rewards (Q-Values) to construct the Deep Q network, and this is different than the method used in this report as it doesn't use the A* algorithm to get the training dataset, instead it just keeps running to a specified number of episodes while improving itself in each run until reaching a satisfactory result.