university of groningen / faculty of science and engineering

LEARNING FROM DEMONSTRATION FOR ISOLATING FOREST FIRES USING CONVOLUTIONAL NEURAL NETWORKS

Bachelor's Project Thesis

Yoes Ywema, y.ywema@student.rug.nl
Supervisor: dr. M.A. Wiering

Abstract: Nowadays there are frequently occurring forest fires. A forest fire can be remedied by spraying water on it or digging firebreaks around it, so the fire cannot spread. The latter approach is studied in this thesis. With learning from demonstration a system is trained to dig firebreaks in a simulation with different forest fire environments. A deep convolutional neural network is used in this project to learn how to map environmental input to useful moves. The network is trained with different amounts of data and compared with a human interactive approach. In this human interactive approach the system asks a human to solve fires that it cannot isolate itself. The system can be used very effectively in simple environments but has to be expanded for more complex ones. In more complex environments a river and a number of houses result in a more complex sequence of steps to isolate the fire. Furthermore extra training data provides a way to improve most of the models that were previously trained with less data. Training models with extra training data, generated by the human interactive approach, resulted in the largest number of improved models.

1 Introduction

1.1 Forest Fires

Roughly since the industrial revolution, people began to emit exponentially growing amounts of greenhouse gases (GHGs). Today we know that these GHGs contribute to the rise of the average temperature on earth. This has many direct and indirect effects on our daily life. Storms are getting more excessive, the sea level is rising and the number of forest fires that occur is increasing. The focus of this project will be on the latter. Forest fires are a threat to the inhabited world. Not only is the fire able to hurt people, it also destroys our forests. The fires on their turn contribute even more to the amount of GHGs in the air.
Once a forest is dried out by the weather it gets sensitive to the heat and the threshold to ignite drops. The spreading of a fire depends on different parameters like the wind, the substrate and the humidity of the area. A fire can be extinguished by spraying water on it or digging firebreaks around the fire. Both methods can be effective. For large forest fires spraying water will not help much, therefore this project focuses on the last one: digging firebreaks. The breaks around the fire ensure that the fire does not spread any further, but creating these breaks requires coordination. Without having a bird's-eye view on the fire, it would be a dangerous job. A person could be isolated and caught by the fire. In this project simulated satellite views are used for coordinating the forest fire control.
Previous approaches have been researched to isolate forest fires in simulations. One of these is the Reinforcement Learning approach (Wiering and Doringo, 1998). In this approach one or more agents receive rewards or punishments for actions they perform. The agents are getting rewards for digging firebreaks in correct directions, but will be punished for the size of the area that is burned (including a big punishment when the agent itself will burn).
Another approach is one in which evolutionary neural networks are used to generate sub goals and to assign agents to the different sub goals (Wiering et al., 2005). In that project multiple agents are used to create and solve different sub goals and therefore cooperate in order to extinguish a forest fire in simulation. The simulation is a quite simplistic one containing only forest.

1.2 Learning From Demonstration

This project will continue the research in this field by using learning from demonstration (Schaal, 1997). The strategy used in this approach is an agent that learns (offline) to make the right choices based on examples that a demonstrator provides.
Learning from demonstration (LfD) has been successful in different areas. For example in robotics, for the task to mimic human movements (Schaal, 1997). In that research a 30 seconds long demonstration in pole balancing resulted in a reliable performance.
It will be interesting to see whether the algorithm is able to generalise the actions of the agents (learned from demonstrations) to isolate forest fires.
Also in the area of games, LfD proved its power. The game of Go was long seen as one of the most challenging games in the world of Artificial Intelligence. An approach of solely convolutional neural networks was applied (Sutskever and Nair, 2008) on the game in which 36.9 % of the human player moves were correctly predicted. In 2016 (Silver et al., 2016) Alpha Go was able to beat the European Go champion 5 out of 5 times. It has achieved this great result by learning from human experts before applying Reinforcement learning (RL).
In a study about the game 'Donkey Kong' (Ozkohen et al., 2017) a Multi Layer Perceptron (MLP) is first trained on how to play the Donkey Kong game (offline) with demonstration data. Afterwards Temporal difference learning, a form of RL, was used to optimize actions of the player. It appears that LfD performed well (successfully completes around 50% of the games), but the RL algorithm wasn't able to increase the performance of the game player, it even led to a worse performance. This is in contrast with the Deep Q-learning from Demonstration approach on a variety of games (Hester et al., 2018). This algorithm (DQfD) was able to increase performances after learning from demonstration, via (online) Q-learning.
In this paper Deep Convolutional Neural Networks (DCNN) for LfD are used. The DCNN already proved to be very useful in playing Atari games (Mnih et al., 2013). It was able to learn successful control policies from raw pixel data (84×84 inputs).

1.3 This Research

The benefit of using a deep convolutional neural network (LeCun et al., 1989) in this project is that the network might be able to extract spatial information from the map, which can be used to make a correct prediction about whether to dig a firebreak or move in a certain direction. In this project the research question I'm trying to answer is: "Can learning from demonstration with a deep convolutional neural network (DCNN) effectively be used to isolate forest fires in simulation?".
Furthermore in this research I will have a look at a Human interactive DCNN (HI-DCNN) approach. I will investigate whether this approach will improve the performance of the DCNN. Additionally this approach will be compared with an approach in which the DCNN is improved with just more data. The effects of these different methods will be investigated on different map sizes with different environments.

2 Method

2.1 Glossary

Episode: From the initial state (a randomly placed agent and a starting fire on a map) towards the end of the fire, the dead of the agent or until the fire has spread all around the border.
Epoch: One round of using all training data once to train a model with the DCNN. Often multiple epochs are necessary to train a model well.
Model: Set of defined layers and model parameters in a neural network that processes environmental input (via computations) into an output (move).
Move: A move is one of the possible actions an agent can take in a certain state. 5 different moves are possible (heading N,S,W,E or Dig).

2.2 Environment

The environment which is used in this forest fire problem is a simulated satellite view on a forest. This will give a $n \times n$ map with n^2 cells. There are 9 different types of cells. A cell can contain:

- An agent (bulldozer/woodcutter)
- Forest
- A house
- Water
- Dirt (a place where the bulldozer has been digging)
- Fire (burning forest)
- Fire (burning house)
- Burned forest
- Burned house

The type of the cell can be observed while looking at the map. In figure 2.1 such a map is shown.

The cells also have some unobservable attributes. Forest cells and houses have a current temperature, fuel (determines how long something can be on fire), ignition threshold (determines how easy something ignites) and a heat potential (how much heat is applied to surrounding cells). For houses the fuel is bigger than for the forest cells, so an assumption that I make is that the fire burns longer in the inhabited areas.
The agent has the opportunity to choose from 5 different actions. Moving north, east, south, west (cardinal directions) or to dig at the place where the agent is at that

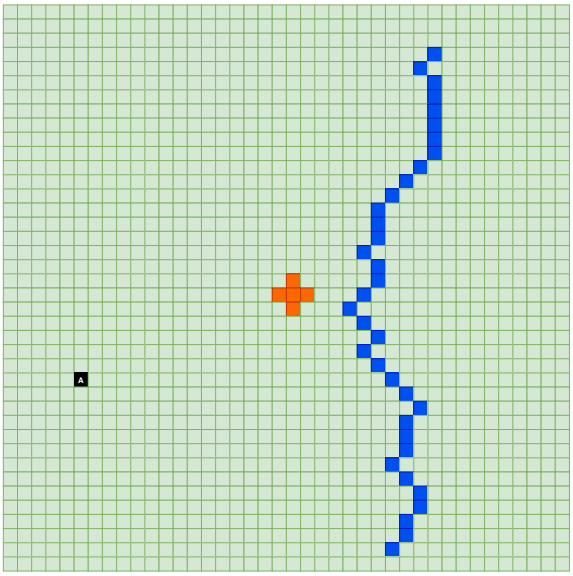

Figure 2.1: 40×40 map containing an agent (black), fire (orange), forest (green) and a river (blue)

specific moment. All these actions last one time step. There are also some restrictions on the movement of the agent. The agent cannot traverse water and houses. Once it will try to do this, the position of the agent will not change while the fire is spreading. When the agent tries to traverse fire, it will die.
The fire always starts in the middle of the map. The agent will be positioned anywhere on a maximum radius of:

$$max(r) = \frac{s}{2} - 1 \tag{2.1}$$

In which r is the radius and s is the size of the map (in 1 dimension). The latter prevents the agent from being placed outside of the map, but the agent can still be placed on the border. This way the initial positioning of the agent is randomized.

In this thesis a distinction will be made between simple and complex environments. There are two settings which will have an influence on the environment: the **size** of the map and the **complexity**. The size can be either 10×10, 20×20 or 40×40, which will result in a generated map with a width and height of this size. There are four types of complexity:

- Only containing forest
- Containing forest and houses
- Containing forest and a river
- Containing forest, houses and a river

So the most simple environment is the one with a 10×10 sized map only containing forest, fire and an agent. The most complex one is a 40×40 sized map containing forest, fire, a river and multiple houses. In total there are 12 different environments. The number of houses that is placed on the map is determined by the following formula:

$$h = 3 * \frac{s^2}{100} \tag{2.2}$$

This number of houses (h) is chosen to let the density be equal over different map sizes (s).
The x and y coordinates of the houses are determined randomly.
The river is generated by randomly specifying distances (between 1 and 3) from the upper and lower border. These values determine the range of the river in y-coordinates. Then the x-coordinate of the upper starting point of the river is randomly chosen (inside the range of possible x-coordinates). Afterwards the river randomly moves down, down-left or down-right through the forest until it has reached the determined distance from the lower border. This also allows the river to be in between the agent and the fire.
This way different random maps are generated for each episode.

2.3 Representation Environment

For a DCNN the data preparation is an important step. Also for our N×N maps, thinking about a suitable representation is required. On the map there are N^2 cells which have different properties. Not all properties are necessary for the DCNN to know. For example, a DCNN doesn't benefit much from knowing about the temperature of a cell. It requires less memory to only show whether a cell is already on fire or not (binary representation). As a result the system can learn to save the cells that aren't on fire yet. A continuous value for the temperature would add some information about which cell will ignite earlier but will make computations in the system way more time consuming.
So the goal in the preparation step is to include relevant information in a usable way. A way to do this is by using multiple binary maps (vision grids) as input data (Knegt et al., 2018). Showing a few layers of data with relevant information in the form of a binary grid speeds up the learning process.
A map with continuous values wouldn't be appropriate in this case. You might for example think about using RGB values for the map to classify which action should be taken. This is computationally much heavier than the vision grid representation. In this forest fire problem the input data isn't complex and therefore this vision grid representation is useful.
So for this problem the environment is represented by the following 6 binary matrices:

- Agent position (1)
- Fire positions (2)
- Water positions (3)

- House positions (4)
- Dirt positions (5)
- Tree positions (6)

In figure 2.2 the translation from an environmental map into 6 binary maps is shown.

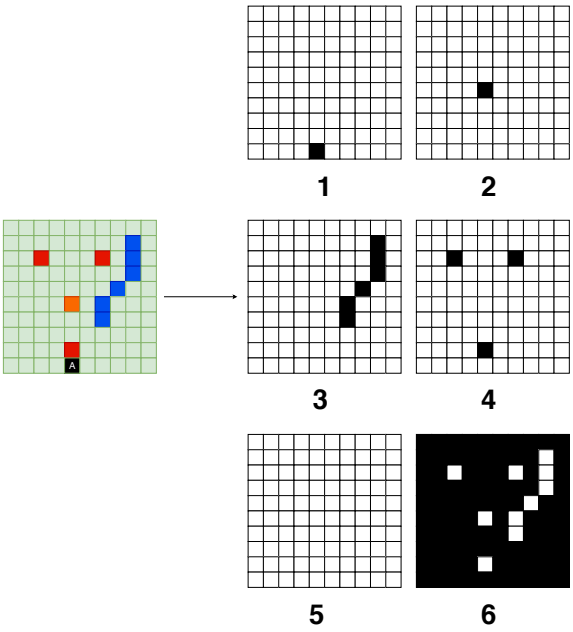

Figure 2.2: 10×10 map containing an agent (black), fire (orange), forest (green), houses (red), a river (blue) and no dirt (brown) translated into 6 binary maps where black is 1 and white is 0.

2.4 Input Data

This project uses learning from demonstration. Therefore the data is created by a human. This means I control the agent to solve the forest fires. All moves I perform are collected in a folder with moves for a specific environment. The data is separated by size and complexity. For all 12 different complexity settings, episodes are played to create training data and to create validation data.
Although with all settings the same amount of episodes are played, there will be differences in the amount of moves that are done. Especially the size of the map will have a big influence on the amount of data gathered since a larger map will result in agents that are further away from the fire, and therefore require more moves to reach and finally to isolate the fire. Also the complexity will play a role in the number of moves that must be done in order to isolate a fire, since walking around obstacles (houses and water) will require more moves. Whenever I accidentally did send the agent into the fire, the last fatal move is removed. At that moment the episode is finished. This was done in order to make sure the agent is not going to imitate this detrimental behavior.

3 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a neural network that is used for example in image recognition, classification and processing pixel data. Its foundation lies in 1989 (LeCun et al., 1989). How it works and why it is useful in these areas is explained below.

3.1 Neural Network

A Neural Network consists of three or more (then it is called a 'deep neural network') layers with nodes. It contains one layer with input nodes (x_i) one or more layers with hidden nodes (h_i) and a layer with output nodes (y_i). i denotes the node of relevance in a specific layer.
An input node (x_i) is attached with a weight (w_{ij}) to a hidden layer node (h_j). The value for a hidden node is computed by :

$$h_j = f(\Sigma_i x_i * w_{ij}) \tag{3.1}$$

Where $f()$ is a nonlinear activation function. The values for the output nodes are then computed by:

$$y_k = f(\Sigma_i h_j * w_{jk}) \tag{3.2}$$

These layers are connected to each other with multiple weights, which make parallel calculations. This way the input is processed to get a desired output. In figure 3.1 a visual representation of a simple neural network is shown.

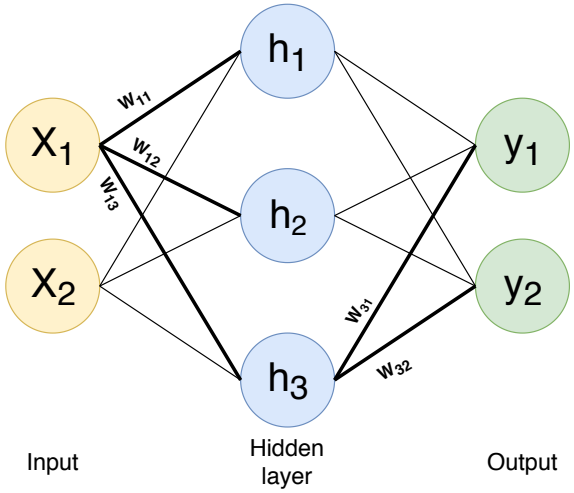

Figure 3.1: Simple neural network with 1 hidden layer

Training the network can be done by using an optimization algorithm. In an optimization algorithm an input example is presented to the network. The network calculates an output. The difference between the output value (y_k) and the target (t_k) value, called **error**, determines how the weights should be adjusted in order to come up with a more correct output the next time. The target value is the value that you want your system to

give as output. For example for the y_2 neuron the following update rule can be applied:

$$w_{32} = w_{32} + \alpha * (t_2 - y_2) * h_3 \tag{3.3}$$

α is the learning rate which often is a small positive number.
Updating all weights in the network back until the input layer can be done with the **backpropagation** algorithm (Rumelhart et al., 1986).

3.2 Convolution Layer

In a CNN a sequence of calculations is performed on its input. These inputs can be represented by large matrices, filled with values. An important step in a CNN is the convolution operation. In this operation, the goal is to extract relevant features from an input. One example of this is shown in figure 3.2. Performing a convolution with a 3×3 input matrix can tell you whether the pixel in the middle is different from its surrounding pixels. In figure 3.2 a filter is applied to two different inputs. The output value 8 tells there is a big contrast between the pixels, while the output 0 tells there is no contrast.

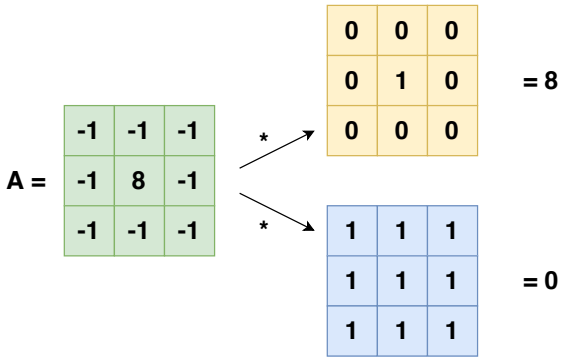

Figure 3.2: A contrast detecting kernel 'A' applied on two different inputs

Note that performing a convolution operation is not the same as applying traditional matrix multiplication. With convolution the operator shifts over the entire input matrix. It computes a new value for each pixel in the middle of the window.
The result will be a new (almost) same sized matrix containing the information about the presence of a specific feature, called a **feature map**. This is harder for the pixels at the side of a picture since there are not enough pixels to multiply with. To overcome this problem **zero-padding** can be used, which is a method to surround the picture with pixels with a value of zero. This makes it possible to apply filters on the sides of the picture and therefore control the size of the feature map. The amount of different convolution kernels you apply on some input is called the **depth**. Each kernel results in a different feature map. The size of the steps one uses to shift over a picture in x and y direction is called the **stride**.

3.3 Pooling Layer

In a pooling layer data is summarized. This can be done in order to downsample a representation. There are multiple types of pooling, but the one that is used in this project is max-pooling. Max-pooling works as follows. Assume we have a 4×4 matrix, and we are going to perform max-pooling on it with a pool size of 2×2 and a stride of 2. The maximum values for each 2×2 block will be taken and will be put in one new cell afterwards. In this way you can down sample a 4×4 input towards a 2×2 input. A visual representation of this is shown in figure 3.3.

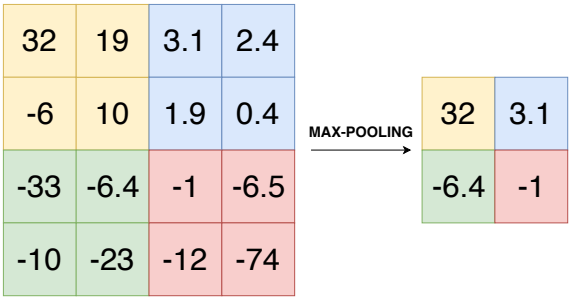

Figure 3.3: Max-pooling with a 2×2 filter

3.4 Architecture

Now the architecture of the neural network that is used in this project is described. The deep convolutional neural network that is used consists of an input layer (**I**), 4 hidden layers of which respectively a convolution layer (**II**), a maximum pooling layer (**III**), two convolution layers (**IV** and **V**) and finally an output layer (**VI**). The shape of the input data in layer **I** is of N×N×6. Which means six 2D matrices with the size of the map (N×N). The number 6 is because the environment is represented by 6 layers of binary vision grids as we saw in section 2.3.
Layer **II** is a convolution layer as described in section 3.2. This layer consists of 100 filters/kernels. The kernel size of this layer is 2×2 and the activation function that is used is the ReLU function. The ReLU function is used because of its efficiency and effectiveness. There is no padding applied and the stride is 1.
The third layer (**III**) is a maximum pooling layer. The pool size is 3×3 and it shifts over the output from the first layer with a stride equal to 1.
Layer **IV** is a convolution layer with 50 kernels, a kernel size of 2×2, no padding and a stride of 1.
The last hidden layer (**V**) is again a convolution layer but now with 25 kernels and again a kernel size of 2×2, no padding and a stride of 1.
The output of this last layer is flattened. Which means that it is translated into a 1D vector. This vector is fully connected with weights to an output layer with 5 output nodes. The values from the output layer are normalized by the Softmax function. This function

converts the outputs to probability like values between 0 and 1. These values can easily be compared to the target values of the outputs (0 or 1).

3.5 Activation Functions

ReLU: The Rectified Linear Unit (ReLU) is a function that passes all positive inputs just as they are as output, but outputs negative inputs as 0:

$$output = \begin{cases} 0, & \text{if } input \leq 0 \\ input, & \text{otherwise} \end{cases} \tag{3.4}$$

Softmax: The softmax activation function is a function that transforms all outputs of the neural network into probability values between 0 and 1. It does this by the following formula:

$$P(y_i) = \frac{e^{y_i}}{\Sigma_j^C e^{y_j}} \tag{3.5}$$

In this equation y_i is the output of the neural network for action i. C is the total amount of possible actions and $P(y_i)$ is the translated output into a probability value.

3.6 Loss Function

Now training the model means adjusting all weights between nodes in different layers in the whole neural network in such a way that it reduces the loss (difference between output and target value). In this project the categorical cross-entropy loss is used. After the output has been normalized by the softmax function the cross-entropy loss is calculated in the following way:

$$L = -\Sigma_i^C t_i \log(P(y_i)) \tag{3.6}$$

In which L is the loss and t_i is the target output for action i (1 for the target desired action, 0 otherwise) .
The loss can be minimized. In figure 3.4 a visual representation of an example loss function is shown with a loss depending exclusively on weight w. In a neural network the loss depends on multiple weight values. Gradient g is the slope of the loss function to a specific weight and can be used to minimize the loss L. This is done by using an optimizer.

3.6.1 Adam Optimizer

In this project the loss is minimized by the Adam optimizer (Kingma and Ba, 2014), which is an extension of stochastic gradient descent. The Adam optimizer minimizes the loss by adaptive learning rate optimization. The Adam optimizer is currently recommended as the default algorithm to use (Ruder, 2016). As the inventors of the Adam optimizer stated the advantages are that it "only requires first-order gradients with little memory requirement" and "the method computes individual

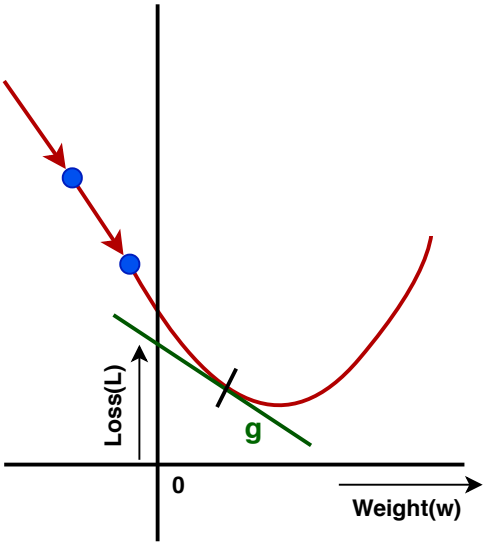

Figure 3.4: Example of a loss function with gradient (g) and loss (L) depending exclusively on weight (w)

adaptive learning rates for different parameters from estimates of first and second moments of the gradients" (Kingma and Ba, 2014). This is in contrast with most familiar methods at the time of publication in which the learning rate is a constant value. For this method the first (m_t) and second (v_t) moment are calculated at each iteration:

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t \tag{3.7}$$

m_t is the first moment at time step t, β_1 is the exponential decay rate for the first moment (by default 0.9). m_{t-1} is the moment at the previous time step and g_t is the gradient of the loss function to a specific weight. The second moment can be calculated the following:

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * g_t^2 \tag{3.8}$$

Where v_t is the second moment at time step t, β_2 is the exponential decay rate for the second moment (by default 0.999), v_{t-1} is the second moment at the previous time step and g_t^2 is the squared gradient of the loss function.
The next step is the bias correction. In other words a correction for the fact that $m_0 = 0$ and $v_0 = 0$ since these values are an initial guess (bias). The for bias corrected first moment can be computed by:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.9}$$

$\hat{m}_t$ is the for bias corrected first moment, β_1^t is the exponential decay rate for the first moment to the power t (time step). In the same way the for bias corrected second moment is calculated:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{3.10}$$

$\hat{v}_t$ is the for bias corrected second moment, β_2^t is the exponential decay rate for the second moment also to the power t (time step). Finally these values are used in the following formula to update a particular weight:

$$w = w - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{3.11}$$

α is the learning rate which is a hyper parameter that can be set.

3.7 Validation

When training the system to reproduce your behaviour (imitating moves from similar situations) the goal is to stop training at the moment the system performs best. But when is this? It is actually hard to have a solid answer to this question. The accuracy of the system can be computed each training epoch. The accuracy of the model is determined by:

$$Accuracy = \frac{cP}{tP} \tag{3.12}$$

A correct prediction (cP) means that the output of the model (a predicted move) matches the true output (performed move) of the validation data for some input. tP represents the total number of all predicted moves in the validation data.
Computing the accuracy based on training data is a bad practice since this doesn't prevent you from creating an over-fitted model. You might perfectly predict each step the model has to take after presenting an example of data from the training set. But when presenting completely new/different input to the over-fitted model it will not be able to predict useful moves.
Instead the validation data is collected to measure how well the model is performing. This data isn't used to train the model, but only to give an impression about how well target moves are predicted.

3.7.1 Early Stopping and Model Checkpoints

The accuracy on the validation data is optimized in order to get the best model for your data. Early stopping and model checkpoints are two methods to find the optimal model. With early stopping you stop the training of a monitored value (accuracy or loss). You don't stop immediately when the system hasn't improved, but instead you use model checkpoints to save the best model and go on with training. When after a number (patience) of epochs the model hasn't improved, the saved model from the model checkpoint is finally selected.

4 Experimental Setup

4.1 Methods

4.1.1 DCNN

For all 12 different environments, I played 50 training episodes and 20 validation episodes for which feedback (corresponding action to environmental input) is provided. With this data 30 models are trained for each environment (360 models in total). After the models have been trained the system is tested. Each model is tested by letting it play 100 random new episodes.

4.1.2 DCNN with Extra Data

To find out what the influence is of using more data, another 50 training and 20 validation episodes are gathered. Now for each environment 100 training episodes, and 40 validation episodes are played. Again 30 models are trained on this data, and to test 100 new random episodes are played by each one of the models.

4.1.3 Human Interactive DCNN

Inspired by DAGGER (Data set Aggregation), which is nowadays one of the most common algorithms for imitation learning (Attia and Dayan, 2018), a new Human Interactive LfD approach is introduced. First a model is trained based on the 50 training and 20 validation episodes from the DCNN. This trained model starts playing random episodes. For each episode in which the model fails to isolate the fire, the same episode starts again and a human gets asked to control the agent to isolate the fire. This way another 50 training episodes and 20 validation episodes are gathered. Similarly 30 models are trained and each one is tested on 100 new random episodes. The advantage of this method is that no data is gathered for episodes that the system can already solve, instead extra data for 'harder' episodes is created. In algorithm 4.1 the pseudo-code for this method is shown.

Algorithm 4.1 Human Interactive DCNN

```
initialize D ← getData()
π ← DCNN(D)                    // Model is trained
i=1
 while i ≤ N do
   E = getRandomEpisode()
    fireContained ← play(π,E)
    if fireContained == FALSE then
      D ← D ∪ humanReplay(E)
       i = i + 1
   end
end
π* ← DCNN(D)
```

4.2 Hyper Parameters

For this thesis in each DCNN the total loss on the validation data is monitored by early stopping with a pa-

tience of 10. This means that the model has to keep reducing the loss value each 10 epochs otherwise it will stop training. This value appeared to be a good value in preliminary experiments. The model that is saved as model checkpoint is the one with the maximum accuracy on the validation data.
The loss function is optimized by the Adam optimizer with the hyper parameters that are shown in table 4.1:

Table 4.1: Hyper parameters for the Adam optimizer

α	0.005
β_1	0.9
β_2	0.999
ϵ	10e-8

The last three ($\beta_1, \beta_2, \epsilon$) are chosen by default while with the first one (α) tests were done on 10×10 maps with respect to the final performance rates. These experiments couldn't be done for each single environment and algorithm since this would take too long. The value of 0.005 for α resulted in stable performances.

4.3 Performance Measure

The performances of the three methods are measured based on the number of fires that are isolated in the 100 episodes. Because 30 models are created for each sort of map, the average number of isolated fires (μ) is computed over the 30 samples. Additionally the standard deviations (σ) and standard errors (SE) are computed.

5 Results

The figures below show the results of averaged numbers of contained fires over multiple models. The yellow bars indicate the scores for DCNN with 50 example episodes of training data. The orange bars show the amount of isolated fires for 50+50 extra random examples of training episodes (DCNN-EXTRA). The red bars represent the performance of the Human Interactive DCNN with 50+50 episodes derived from replaying failed episodes (HI-DCNN). The characters below the figures show whether the environment consists of forest (F), forest and houses (F+H), forest, houses and a river (F+H+R) or forest and a river (F+R). The error bars indicate one standard error (σ) of uncertainty above and below the mean. In appendix A a table is provided with mean values, standard deviations and standard errors for each different environment and algorithm. In appendix B statistics are set out to examine whether the results of different methods are significantly different.

5.1 Results on 10×10 Maps

Figure 5.1 shows the results for the 10×10 map. The first thing you see is that there is no bar for HI-DCNN in

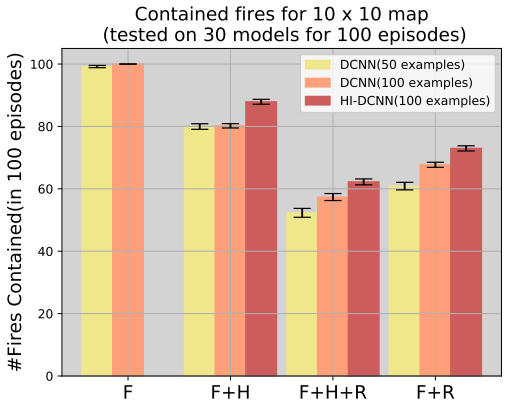

Figure 5.1: Average number of fires contained (out of 100) for 10×10 maps, F = forest, H = houses and R = river.

the forest (F) mode. This is due to the fact that DCNN with 50 examples already performs very well and often (22/30 models) manages to isolate all (100/100) forest fires. Therefore it would be time consuming to find 50 episodes in which a DCNN model isn't able to isolate the fire. Also DCNN-EXTRA with 100 example episodes is able to isolate all fires for 30 models, so therefore in this case HI-DCNN seems unnecessary.
In each environment except for the forest+houses DCNN-EXTRA improves the performance of the DCNN models. Furthermore from these results it becomes clear that for each environment HI-DCNN performs better than both DCNN and DCNN-EXTRA. This also holds statistically, as you can see in Appendix B.1.

5.2 Results on 20×20 Maps

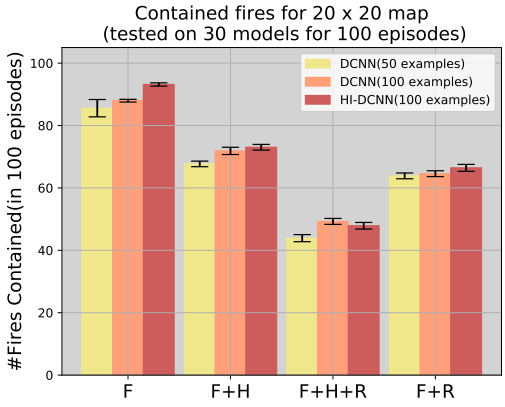

Figure 5.2: Average number of fires contained (out of 100) for 20×20 maps, F = forest, H = houses and R = river.

In figure 5.2 the 20×20 maps show a slight decrease in overall performance compared to the 10×10 maps. In the forest mode extra data (DCNN-EXTRA) examples do not result in a statistically higher number of contained fires. Nevertheless HI-DCNN actually

does increase the performance of DCNN and also performs significantly better than using 50 extra training episodes (DCNN-EXTRA). In the forest+houses and forest+houses+river environment both approaches (DCNN-EXTRA and HI-DCNN) improve the performance of DCNN. However whether the extra data is derived from random training examples or failed examples doesn't matter significantly. For the forest+river environment no significant differences in performance are found for the different algorithms.

5.3 Results on 40×40 Maps

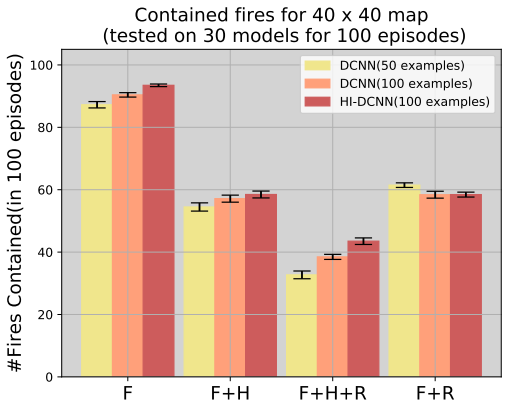

Figure 5.3: Average number of fires contained (out of 100) for 40×40 maps, F = forest, H = houses and R = river.

As can be seen in figure 5.3, performances of the agents in the 40×40 maps are approximately the same as for the 20×20 maps in the forest environment. For the other environments the performance has dropped compared with the 20×20 maps. In the forest environment HI-DCNN beats both other approaches. Nevertheless DCNN-EXTRA is also able to significantly increase the performance of DCNN by using more training episodes. In the forest+houses environment HI-DCNN significantly outperforms DCNN, however no significance can be found between HI-DCNN and DCNN-EXTRA. Also the extra training examples from DCNN-EXTRA do not significantly upgrade the models as compared to DCNN. In the forest+houses+river environment DCNN-EXTRA significantly improves the performance of DCNN, HI-DCNN actually improves it even more and reaches a significantly higher amount of contained fires than DCNN-EXTRA. At last in the forest+river environment DCNN surprisingly performs significantly better than both DCNN-EXTRA and HI-DCNN. So in this case more data actually made the performance of the model worse. Between DCNN-EXTRA and HI-DCNN no statistically significant difference can be found.

5.3.1 Observations

Something that happens in the forest+houses and forest+houses+river environments, is that the agent gets stuck behind a house. Then it executes the same move over and over, which results in being stuck while the fire continues to spread. This also happens when an agent is placed on one side of a river and the fire is on the opposite one. Then it is hard to move around the river and isolate the fire. Instead it will sometimes get stuck at one side of the river while the fire spreads around it.
Nevertheless, the agent is most of the times able to successfully isolate fires that started next to a river. The agent then uses a part of the river to isolate the fire like in figure 5.4.

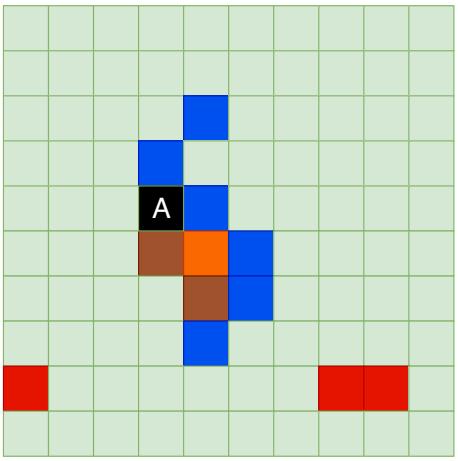

Figure 5.4: 10×10 map in which the agent (black) has used the river (blue) to isolate the fire (orange).

6 Conclusions

In the forest environment 93% of the fires are contained after training a HI-DCNN model with 100 examples for both 20×20 and 40×40 maps. This shows that the learning from demonstration approach can be effectively used in the forest fire problem for these strongly simplified environments. With more data this percentage could probably be higher (by looking at the results of using more data in this project).
For this kind of environment the results can be explained by the fact that models already isolate most of the fires after initial training (DCNN). After creating a model from the initial data, only the fires that are not solved by this model are replayed by a human to generate new data. Providing solutions for exactly these episodes in HI-DCNN did improve the performances significantly more than the same amount of random episodes in DCNN-EXTRA.
Even though specific input layers indicate at what positions houses are located in the forest+houses environment, the performance is lower than for the forest environment. This means that the agent isn't optimally able

to learn how to handle these obstacles (houses). This also makes sense since the houses are randomly placed and therefore confront the agent in different ways. Because the agent handles these obstacles in different ways in the input data, this behaviour is hard to generalise.
In the forest+river environment some interesting things happen. In the 10×10 maps more data results in (as expected) higher performances. For the 20×20 maps all performances are about the same, so more data doesn't improve the models. Now the most surprising result is that in the 40×40 maps extra data makes the performance of the agent significantly worse. Apparently solutions to forest fires, in which a river is involved, do not benefit from the used amount of extra examples to isolate the fire.
DCNN-EXTRA reaches significantly higher performances in 7 out of 12 environments than DCNN. This means that overall more data results in better outcomes in the forest fire control problem. Nevertheless in this research it also becomes clear that by improving the models with a human interactive approach a larger amount of models are improved. With this latter approach in 9 out of 11 environments the performances of the models are significantly increased. Comparing the performances of HI-DCNN and DCNN-EXTRA it becomes clear that 6 out of 11 times HI-DCNN performs better than DCNN-EXTRA. In the other 5 out of 11 models no statistical significance between the models can be shown. Therefore HI-DCNN is concluded to be the most effective method in solving the forest fire problem. Looking at the simpler environments, the answer to the main research question (Can learning from demonstration with a deep convolutional neural network (DCNN) effectively be used to isolate forest fires in simulation?) is yes. In the forest environment with the HI-DCNN approach, more than 93 percent of the fires were contained. For the other environments learning from demonstration is less accurate and is not always able to control the forest fires effectively. However in these more complex environments the use of extra data often shows higher numbers of contained fires. Therefore with these methods, more data probably can lead to effective forest fire control strategies.

7 Discussion

It must be said that for all different environments and algorithms merely 50 or 100 examples of episodes are used. This is, in the world of machine learning (in which learning from demonstration can be placed), a small number. Quite effortlessly more data could be generated for a specific environment. But in this research project there were many different environments and training took quite some time (about 6 hours for the most intensive environments). Therefore no more data is generated in this project. Although this research provides indicative positive results for the performances of the different algorithms in the forest fire problem, because of the small amounts of data it is not yet possible to draw firm conclusions.
Also the used environments in this project resembles a simplified representation of a real forest fire. So the results in this project do not account for real world outcomes.
An improvement to the current methodology might be data augmentation. This is a method in which inputs can be flipped, cropped, rotated or shifted in order to increase the diversity of the input data. This project would probably benefit from this because this might provide extra solutions for moving around obstacles from different sides. Also rotated data might give the system more information about isolating the fire from different sides.
Another way to overcome the problem with avoiding obstacles might be solved by a function that prohibits the agent to make invalid moves. Also a boolean could be added to the input data for the DCNN, in order to state whether a move is an invalid one. I did develop a function for avoiding invalid moves, but later on disabled it again. This function made the problem easier for the DCNN, but distorts a real and proper understanding about what the neural network has learned from demonstration data.
Finally in this thesis only single agents are used to isolate the forest fires. Of course this is not as effective as using multiple agents that cooperate efficiently in order to isolate a fire. This could be researched in future work.

References

Attia, A. and Dayan, S. (2018). Global overview of imitation learning. *arXiv preprint arXiv:1801.06503*.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. (2018). Deep Q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

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

Knegt, S. J., Drugan, M. M., and Wiering, M. A. (2018). Opponent modelling in the Game of Tron using reinforcement learning. In *ICAART (2)*, pages 29–40.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, A., Wierstra, D., and Riedmiller, M.

(2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Ozkohen, P., Visser, J., van Otterlo, M., and Wiering, M. (2017). Learning to play Donkey Kong using neural networks and reinforcement learning. In *Benelux Conference on Artificial Intelligence*, pages 145–160. Springer.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and Mcclelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA.

Schaal, S. (1997). Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–503.

Sutskever, I. and Nair, V. (2008). Mimicking Go experts with convolutional neural networks. In *International Conference on Artificial Neural Networks*, pages 101–110. Springer.

Wiering, M. and Doringo, M. (1998). Learning to control forest fires. *Proceedings of the 12th international Symposium on 'Computer Science for Environmental Protection'*, pages 378–388.

Wiering, M., Mignogna, F., and Maassen, B. (2005). Evolving neural networks for forest fire control. *Benelearn '05: Proceedings of the 14th Belgian-Dutch Conference on Machine Learning*, pages 113–120.

A Appendix

Table A.1: Statistical results for 10×10 maps: mean (μ), standard deviation (σ) and standard error (SE)

Environment	Algorithm	μ	σ	SE
Forest	DCNN	99.20	1.96	0.36
	DCNN EXTRA	100	0	0
Forest+Houses	DCNN	79.97	4.79	0.89
	DCNN EXTRA	80.20	3.75	0.70
	HI-DCNN	87.90	4.22	0.78
Forest+Houses+River	DCNN	52.3	7.65	1.42
	DCNN EXTRA	57.37	6.04	1.12
	HI-DCNN	62.23	5.10	0.95
Forest+River	DCNN	60.87	6.54	1.21
	DCNN EXTRA	67.70	4.38	0.81
	HI-DCNN	72.97	4.45	0.83

Table A.2: Statistical results for 20×20 maps: mean (μ), standard deviation (σ) and standard error (SE)

Environment	Algorithm	μ	σ	SE
Forest	DCNN	85.57	14.97	2.78
	DCNN EXTRA	87.97	2.36	0.44
	HI-DCNN	93.17	2.82	0.52
Forest+Houses	DCNN	67.70	4.80	0.89
	DCNN EXTRA	71.87	6.27	1.16
	HI-DCNN	73.03	4.93	0.92
Forest+Houses+River	DCNN	43.9	6.12	1.14
	DCNN EXTRA	49.27	5.14	0.95
	HI-DCNN	47.87	5.74	1.07
Forest+River	DCNN	63.87	5.03	0.93
	DCNN EXTRA	64.57	5.04	0.94
	HI-DCNN	66.43	5.97	1.11

Table A.3: Statistical results for 40×40 maps: mean (μ), standard deviation (σ) and standard error (SE)

Environment	Algorithm	μ	σ	SE
Forest	DCNN	87.24	5.85	1.02
	DCNN EXTRA	90.40	3.81	0.71
	HI-DCNN	93.47	2.25	0.42
Forest+Houses	DCNN	54.47	7.11	1.32
	DCNN EXTRA	57.13	6.08	1.13
	HI-DCNN	58.47	5.94	1.10
Forest+Houses+River	DCNN	32.7	6.69	1.24
	DCNN EXTRA	38.47	4.37	0.81
	HI-DCNN	43.50	5.67	1.05
Forest+River	DCNN	61.47	3.96	0.74
	DCNN EXTRA	58.40	5.94	1.10
	HI-DCNN	58.43	4.24	0.79

B Appendix

Table B.1: Comparison to show significant differences between gathered results for 10×10 maps. Normality tested by the Shapiro Wilk test. P-values for parametric tests are computed by a two tailed independent t-test, for non-parametric tests the p-value is computed by a Wilcoxon rank-sum test.

Environment	Comparison	Parametric?	t-value	P-value	Significant?
Forest	DCNN - DCNN EXTRA	NO	-2.00	0.0459	**YES**
Forest+Houses	DCNN - DCNN EXTRA	YES	-0.207	0.837	NO
	DCNN - HI DCNN	YES	-6.69	1.04e-8	**YES**
	DCNN EXTRA - HI DCNN	YES	-7.35	8.24e-10	**YES**
Forest+Houses+River	DCNN - DCNN EXTRA	NO	-2.52	0.0117	**YES**
	DCNN - HI DCNN	NO	-4.95	7.60e-7	**YES**
	DCNN EXTRA - HI DCNN	YES	-3.32	0.00160	**YES**
Forest+River	DCNN - DCNN EXTRA	YES	-4.68	2.21e-5	**YES**
	DCNN - HI DCNN	YES	-8.23	6.14e-11	**YES**
	DCNN EXTRA - HI DCNN	YES	-4.54	2.86e-5	**YES**

Table B.2: Comparison to show significant differences between gathered results for 20×20 maps. Normality tested by the Shapiro Wilk test. P-values for parametric tests are computed by a two tailed independent t-test, for non-parametric tests the p-value is computed by a Wilcoxon rank-sum test.

Environment	Comparison	Parametric?	t-value	P-value	Significant?
Forest	DCNN - DCNN EXTRA	NO	0.0221	0.982	NO
	DCNN - HI DCNN	NO	-4.82	1.44e-6	**YES**
	DCNN EXTRA - HI DCNN	NO	-5.38	7.39e-8	**YES**
Forest+Houses	DCNN - DCNN EXTRA	YES	-2.84	0.00632	**YES**
	DCNN - HI DCNN	NO	-4.17	0.000102	**YES**
	DCNN EXTRA - HI DCNN	NO	-0.787	0.434	NO
Forest+Houses+River	DCNN - DCNN EXTRA	YES	-3.62	0.000637	**YES**
	DCNN - HI DCNN	YES	-2.55	0.0136	**YES**
	DCNN EXTRA - HI DCNN	YES	0.978	0.332	NO
Forest+River	DCNN - DCNN EXTRA	YES	-0.529	0.599	NO
	DCNN - HI DCNN	YES	-1.77	0.0821	NO
	DCNN EXTRA - HI DCNN	YES	-1.29	0.204	NO

Table B.3: Comparison to show significant differences between gathered results for 40×40 maps. Normality tested by the Shapiro Wilk test. P-values for parametric tests are computed by a two tailed independent t-test, for non-parametric tests the p-value is computed by a Wilcoxon rank-sum test. The * indicates that the first sample has a significantly higher mean than the second sample (which is the other way around with all t-values below≤0)

Environment	Comparison	Parametric?	t-value	P-value	Significant?
Forest	DCNN - DCNN EXTRA	NO	-2.41	0.0157	**YES**
	DCNN - HI DCNN	YES	-4.96	7.14e-7	**YES**
	DCNN EXTRA - HI DCNN	NO	-3.25	0.00117	**YES**
Forest+Houses	DCNN - DCNN EXTRA	NO	-1.54	0.130	NO
	DCNN - HI DCNN	YES	-2.33	0.0237	**YES**
	DCNN EXTRA - HI DCNN	NO	-0.845	0.401	NO
Forest+Houses+River	DCNN - DCNN EXTRA	NO	-3.89	0.000301	**YES**
	DCNN - HI DCNN	NO	-6.63	1.34e-8	**YES**
	DCNN EXTRA - HI DCNN	YES	-3.79	0.000382	**YES**
Forest+River	DCNN - DCNN EXTRA	YES	2.28	0.0224	**YES***
	DCNN - HI DCNN	NO	2.34	0.0195	**YES***
	DCNN EXTRA - HI DCNN	NO	0.0370	0.971	NO