Isolating Wildfires using a waypoint generating CNN-based Multi-agent system

Bachelor's Project Thesis

Jonathan Hewlett, s3665615, j.w.j.hewlett@student.rug.nl,
Markus Bäuerlein, s3664899, m.bauerlein@student.rug.nl,
Thijs Visee, s2982129, t.p.visee@student.rug.nl
Supervisors: Dr M.A. Wiering, m.a.wiering@rug.nl
Jelle Visser MSc, jelle.visser@rug.nl
Henry Maathuis MSc, h.maathuis@rug.nl

Abstract: As a result of climate change, the forest fire season is becoming longer and more ferocious. One technique for controlling and mitigating the damage of forest fires is to use heavy machinery to clear paths of undergrowth, removing the fuel for the fire to continue spreading. In this paper we investigate the feasibility of using a convolutional neural network (CNN) to determine waypoints for multiple agents in a simulated environment, so that these agents can dig the firebreaks between these points to contain the fire. The CNNs are trained by learning from demonstration with the varying environmental conditions such as the wind direction and propagation speed of the fire. Different CNN architecture designs were used to predict the waypoint position per agent and the overall results of each of these variants were compared. The results show that this approach to controlling forest fires in the simulated environment is not only possible but very effective for CNN-architectures that represent relative positions as continuous values. While for discrete output representations there remains some improvement, this paper provides ample foundations for future research to investigate the limitations of our approaches and to model more realistic simulations.

1 Introduction

1.1 Wildfires and Motivation

Wildfires are a purely destructive phenomenon when viewed through human eyes. Every year, thousands of people suffer from the consequences of the fires, through forced displacement, inhaling toxic smoke or worse. One can also imagine the consequences for animal wildlife, living in affected areas without the benefits of technology. Nevertheless, fires are a normal environmental phenomenon playing a role in natural selection processes and leaving fertile soil and less competition for highly adapted species. Fire-resistant trees such as certain species of pine have developed strategies such as non-inflammable barks, fire-induced sprouts, fire-activated seeds and dropping lower branches to mitigate the effects of burning and even take advantage of it NationalForestFoundation (2021).
Humans have been able to curb the dangers of the inevitable outbreaks of wildfires through different measures of containment. Next to the more publicly known fire-fighting efforts on the ground and from the air, so-called "fire-line" or "firebreaks" are dug either using machinery or with handheld tools.
However, the status quo is in the process of being altered to our disadvantage through a different kind of human interference. Climate scientists around the world are increasingly concerned with the destabilisation of global climate patterns and the potential side effects which are set to include extreme weather conditions. This is likely leading to longer forest fire seasons in regions affected and will result in larger as well as more frequent forest fires, including in regions that are hitherto unaffected. Already in the early 1990s an estimated 1.4 billion tonnes of carbon dioxide was released into the atmosphere by forest fires each year Andreae and Goldammer (1992). These effects are already measurable EPA (2021), at least in the United States where there are reliably

documented sources of data around this topic. As it appears that the frequency and amount of destruction of wildfires will increase in the future, there is a considerable need to manage the risks and improve isolation techniques. With Artificial Intelligence (AI) already having played a major role in optimising all kinds of processes in human society, it is believed that it might be beneficial to utilise it in automatising the coordination of wildfire isolation.

1.2 Previous Work

Previous research has investigated the viability of automating the coordination of agents constructing the aforementioned firebreaks. Reinforcement Learning algorithms Sutton, Barto, et al. (1998) have been proposed as a possible solution Wiering and Doringo (1998) and more recently, promising results have been achieved using such techniques Hammond, Schaap, Sabatelli, and Wiering (2020). In Reinforcement Learning, for each action taken a reward is generated the more desirable the action the greater the reward, this encourages advantageous behaviour. In our example, digging firebreaks in appropriate positions may be rewarded, as opposed to losing more land, or even an agent, to the fire.

Another approach is the use of Convolutional Neural Networks (CNN), first introduced by Yann LeCun to classify handwritten digits Y. LeCun and Jackel (1989). Since then, they have been confirmed to be useful for a wide variety of applications associated with image recognition, for example in the game of Go Silver et. al. (2016). Previous work on the topic of wildfires already employed CNNs to predict actions on a timestep-per-timestep basis Rocholl (2020). Multiple vision grids were used to represent the state of the environment, including the fire, firebreaks, current and previous positions of active and inactive agents. Vision grids are the pixel values of the environment through which a CNN perceives relevant features, vision grid channels are separated and filtered to only display specific features, such as firebreaks. CNNs are able to take multi-channel images as input and extract spatial information, lending themselves well to such a task. Using the state of the environment encoded in a multi-channel image, again the immediate movements of individual agents could be determined and the multi-agent system (MAS) could successfully contain fires in simplified environments of a maximum grid side length of 61 cells.

It is important to use MAS in research striving to contain wildfires because in the real world the rates of spread of the fire and the limitations of digging speed do not allow for single-agent solutions. Simulations and algorithms thus need to take into account a group of agents that work together to contain fires effectively.

Both Hammond et al. (2020) and Rocholl (2020) combined their techniques with Learning from Demonstration (LfD) where their agents could learn offline from pre-generated data, before engaging with the simulation. LfD is useful in these cases in order to reach a human-level performance. In other situations, like Silver et. al. (2016), it can be used to kick-start learning and then reach superhuman performance by incorporating other techniques, such as reinforcement learning to optimise for desirable behaviour through a reward function.

1.3 This Research

This research is predominately inspired by Rocholl (2020), but instead of predicting the optimal action to take for every timestep, different CNN architectures have been implemented to prescribe appropriate waypoints for each agent over a longer time frame. This may come closer to reality, where AI may coordinate the different agents, but the agents themselves are not fully autonomous or even simply made up by a team of humans with shovels. It would not be feasible then to update their behaviour on a second-to-second basis, but rather achieve coordination through the assignment of sub-goals. Hence, the research question we aim to answer is: *"Can a waypoint generating convolutional neural network coordinating a multi-agent system be effective at containing wildfires in a simulated environment?"*.

In order to find answers to that, a new versatile wildfire simulation was built to collect LfD-data and test CNN predictions. Four different CNN variants are compared, providing new waypoints relative to the active agent in different output encodings.

2 Methods

2.1 Environment

In order to simulate a forest fire we created a 256 by 256 grid of interconnected cells, each with the properties needed to represent somewhat realistic forest fires. The properties are as follows (for exact parameter values see Table 4.2):

- **Fuel:** This is the amount of fuel that could be burnt in a fire, once ignited the fire consumes the fuel until it is exhausted, meaning it cannot catch fire again.
- **Temperature:** Each cell starts with a temperature of zero, apart for the centre cell that is initially set on fire. Burning cells transfer heat to adjacent cells at each time step.
- **Ignition threshold:** When a cell heats up enough, it reaches the ignition threshold and combusts.

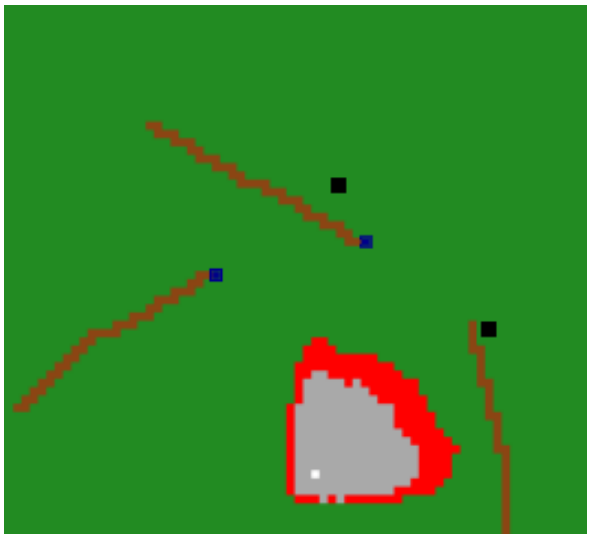

Figure 2.1: A section of the 256 by 256 environment during the data gathering process.

Green: Forest	**Red: Fire**
Gray: Burnt out cells	**Brown: Firebreak**
Blue: Agents	**Black: Waypoint**
White: Centre point	

As can be seen in Figure 2.1, the graphical user interface (GUI) gives information on how the simulation progresses, either by the directions of a human or a trained CNN. We differentiate between cell types and cell states, which is a design choice benefiting adaptability and reusability. All of our cells are initialised as forest cells with specific properties. The possible cell states are:

- **Normal:** The cell type is displayed by the GUI in its original colour and treated by the simulation according to its initial properties.
- **Firebreak:** These cells cannot be ignited anymore, therefore they prevent the fire spreading from neighbouring cells.
- **On fire:** Cell types with a finite ignition threshold will start burning once the threshold is reached.
- **Burnt out:** Cells which have exhausted their fuel and cannot be reignited.
- **Agent:** Showing the user where the agents are located.

While the simulation progresses, the state of the cells may change but the underlying type remains. This has the advantage that the simulation can be expanded with additional cell types representing water area or infrastructure, as done by Ywema (2020). We followed the model-view-controller (MVC) paradigm, allowing us to easily switch between gathering data and testing trained CNNs. There are two parts of the controller, one for human input and one for CNN-generated waypoints.
In the following paragraphs the dynamics of the environment are described. At the beginning of the simulation, a fire is started in the centre cell of the environment. From here on the burning cells heat up their neighbours in the four compass directions. As the cells heat up and reach a temperature threshold, they will ignite, unless they are a firebreak, situated beyond the borders of the environment or if the fuel has been exhausted. There is a multiplication parameter guiding this process. It is in the range $[0, 1]$ and represents the chance that heat is spread in a particular time step, thereby linearly influencing the fire propagation speed. Cells

may receive heat from multiple neighbours and once they exceed the ignition threshold specific to their cell type, they will ignite and spread the heat in the following time steps.
Letting the fire expand in the grid only in the four compass direction prevents the fire from spreading evenly in a circular fashion, which is why we introduced a random heat spread parameter sampled from a uniform distribution. It provides a baseline factor in range $[0.5, 1.5]$ for the spread from cell-to-cell which is then multiplied corresponding to wind direction and speed.
The simulation allows for setting one of eight possible wind directions that are assigned upon initialisation of the environment. These represent the cardinal and inter-cardinal compass directions. There are 5 discrete integer wind speed values from [0, 4], the first of which does not affect the fire propagation at all, leading to a circular spread. The other values linearly increase the effect of the wind, by directing the fire spread faster in the direction of the wind and slower against it. Wind speed and direction influence the heat spread from cell to cell so that the propagation speeds up in the direction of the wind and vice versa.
Perpendicular to the direction of the wind the heat that spreads between cells is a value from the uniform distribution $H = U(0.5, 1.5)$. In the direction of the wind the heat value H is again dependent on an arbitrary value from U but the value is guided by the formula: $H = U(0.5, 1.5) \cdot (1 + \frac{windspeed}{3})$. Against the wind the dissipation of heat between neighbouring cells is changed according to: $H = \frac{U(0.5,1.5)}{1+\frac{windspeed}{3}}$.
In all our simulations the initial settings remain constant until the end of the fire containment episode. All our code, data and results are publically available. [1]

2.2 Representation of Environment

Finding an appropriate representation of the environment to use as CNN-input is a crucial step in constructing the waypoint generation pipeline. One wants to encode all the necessary information without cluttering the input, possibly leading to slower network performance and less predictive capabilities. The CNNs of the kind we are using take inputs

[1] https://github.com/markus-brln/BachelorThesisForestFireControl

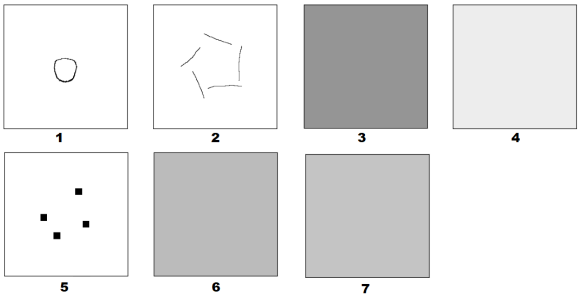

Figure 2.2: Representation of the state of the environment from Figure 2.1 for the CNN. 1. Active fire, 2. Firebreaks, 3. Wind direction, 4. Wind speed, 5. Other agents, 6. Active agent x-position, 7. Active agent y-position. The values for the cells are in the range [0, 1], where 0 corresponds to white, and 1 to black.

of the form $[batchsize, columns, rows, channels]$ where the batch size is simply the amount of images that pass per training cycle (see section 3.1). Therefore it is practical to provide multi-channel images, similar to how the `bitmap` format contains three different values for each pixel. Figure 2.2 shows the representation of the included input information.
Since our activation functions, which we will discuss later, are not suitable for non-normalised input, all values of the multi-channel input image are normalised between $[0, 1]$. Our approach is inspired by earlier work where so-called "binary vision grids" were used to encode the environment of the Atari game *Tron* Knegt, Drugan, and Wiering (2018).
Channel one is the only channel providing information about the fire spread. We did not include channels for the cell temperature or the amount of available fuel since these values can effectively be inferred from the first channel. Arguably, they would not contribute much to the overall picture of the environment. There was no need for channels containing the forest and burnt out cells either, as this information can be derived from the actively burning and agent position channels. The second channel contains the firebreaks, which are set to have a width of at least three pixels to increase "visibility" for the network. The next two channels encode the eight different wind directions and five wind speeds. The south direction is associated with a value of 0 and going clockwise increasing values are assigned to the direction until reaching south-

east with a value of 1.
In the example, the wind direction was set to be north, which is assigned the index 4, resulting in a pixel value of 4/7. Likewise, the wind speed was at level three out of five, yielding 0.5. The inactive agents are placed on the fifth channel according to their x and y positions, but are represented as 10 by 10 boxes with value 1. The intention is to increase the prominence of the agent position in the environment image when it is processed by the convolutional and pooling filters of the CNN.
While binary encoding might have decreased the cost of computation and memory requirements, we decided to represent the active agent's x and y positions in separate continuous channels. The values are normalised to the range $[0, 1]$, meaning that all pixels of channel 6 in Figure 2.2 have the value $0.38 = \frac{97}{256}$ which corresponds to x = 97, with our environment size of 256 pixels. This makes it easier for the CNN to pick up the exact location of the most important agent.

2.3 Input Data

Following the principle of learning from demonstration, all training and validation data is gathered by humans. While the simulation runs, the active agent is indicated visually and a new waypoint is assigned to that agent via a mouse click. The distance of the waypoints to the agents is capped at the distance the agents can travel in the given time frame, leading to more homogeneous data. The success of the algorithm is determined by the containment strategy humans use when collecting training data. For the different levels of difficulty the focus of the human controllers is more on successfully containing the fire than minimising the area burnt, thereby avoiding risky behaviour of agents operating close to the fire.
In order to make results comparable, there is a common pool of data for each experiment. Furthermore, all four architectures receive training and test input of the form described earlier.

3 Convolutional Neural Network

Convolutional Neural Networks are seen as the gold standard of computer vision, inspired by animal vision and based on existing neural networks. They were first developed for handwriting recognition and rely on sequences of layers to extract key features from an input image Y. LeCun and Jackel (1989). This section covers the background from neural networks and the purposes of each of the layers.

3.1 Neural Network

Neural networks store information with mathematical models generated by an algorithmic learning process which propagates activation signals through a series of connected layers of nodes, that generates a function to map an input and to an output a simplified example can be seen in Figure 3.1.

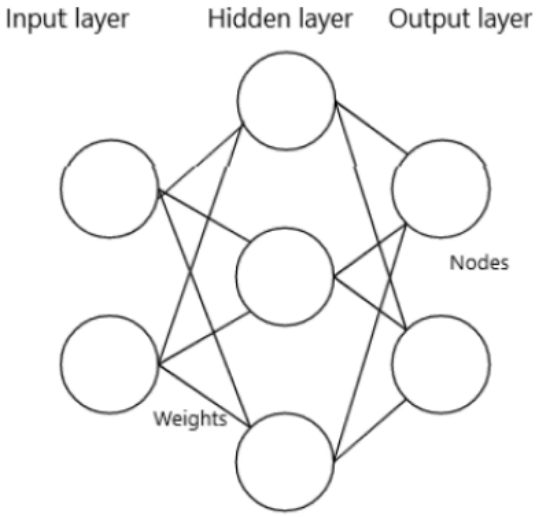

Figure 3.1: Diagram of a 3 layer artificial neural network, displaying how the nodes(circles) and weights(lines) are interconnected.

The layer(s) between the input nodes X and the output nodes Y is (are) known as the hidden layer(s) H and the connections W between them can be adjusted during training, after the network has taken the input and made an output prediction. This allows for information to be represented and propagated through each layer of the network. The value of a node (H_j) in the hidden layer, connected to a node (X_i) from the input layer by a weight (W_{ij}) is calculated by the following equation, the subscript denotes a specific node from a layer:

$$H_j = f(\sum_{i=1}^{n} W_{ij} \cdot X_i) \tag{3.1}$$

where $f()$ represents a non linear activation function described further in Section 3.2. The propagation through the network occurs as the sum of all the previous layer's activation functions. In addition to the weighted sum of all inputs and weights to the activation function, a bias term is added as an intercept (similar to a linear equation) to allow for a better fit of the prediction to the input data by shifting the origin of the activation function horizontally.

Since the training relies on constant feedback, a series of inputs and corresponding outputs are collected. This set is then split at a point into training data and testing data. The network is given an example from the training data and makes a prediction. The difference between the predicted output and the desired output is measured using a mathematical function to compute the *loss*. The weights between nodes are adjusted subsequently to account for the loss using an optimization algorithm, such as back-propagation. Over time, the network will react to certain features in the input and be able to predict the output more accurately Plaut, Nowlan, and Hinton (1986).

3.1.1 Convolution layer

A convolutional layer is the defining feature of a CNN. Our network accepts input of the form $N \times N \times D$, where N is the side length to the image and D the number of channels. We therefore need convolutional layers that use two-dimensional filter matrices. This filter scans across the image on its different channels, performing the convolution by calculating the dot product between the input image and the filter. As this filter moves in strides across the whole input image, it can determine how the pixel values in different sections of the image differ from one another. Simple features, such as contrast between an edge and the background, can be recognized by the initial layers yet later layers can recognise more complex features. This produces a *feature or activation map* with the results of the dot product with the filter and spatial positions of the input image.
The dimensions of the output of a CNN are governed by the input dimensions, stride lengths and the amount of filters. The latter directly determines the depth of the output image, so convoluting a $N \times N \times D$ image with F filters would result in an $N \times N \times F$ matrix. Higher amounts of strides generally mean smaller feature maps with the first two output dimensions being inversely proportional to the stride lengths applied to them.

3.1.2 Pooling Layer

The pooling layer effectively downscales an input. It takes a pool of adjacent entries and extracts a single value from this group of entries, most commonly the maximum (max pooling) or the mean (average pooling) value. It will then move over to the next pool and repeat the previous process. In the output, the dimensions are reduced. Each pooling layer requires certain characteristics, the pool size and the stride. The pool size is the "dimensions" in each pool, whereas the strides indicate the distance between the different pools.
Pooling can take place over multiple dimensions as well, e.g. in an image or a 3-dimensional array. In this project, only 2-dimensional max pooling layers are used, with a pool size of 2×2 and non-overlapping strides of 2.

3.1.3 Fully Connected Layer (Dense Layer)

A fully connected layer is a layer within a neural network that has all nodes of the former layer connected to all the nodes of the layer. It is rather expensive computationally because of its high connectivity, namely $M \times N$ connections where M is the amount of outputs in the previous layer and N the number of units in the fully connected layer. The activation function in a fully connected layer then needs to incorporate all activations of the previous layer for each of its units, resulting in a high amount of computations.
Nevertheless, it is useful in the final layers of a CNN for tackling a problem as described in this paper. The convolutional layers are tasked with extracting the features of the provided information within specific sub-regions of the image. The fully connected layers combine the found features, to allow the network to output a 'conclusion' from the whole input image.

3.2 Activation Functions

An activation function takes one or more inputs to a neuron and calculates a value for the output of that neuron, thereby determining the range of possible output values. During back-propagation

their derivative (i.e. function slope) is used to adjust the biases of a network-layer and the weights between layers in order to minimize the loss (see Section 3.3) calculated for a particular layer. For our research two different activation functions are used, the Rectified Linear Unit (ReLU) and the Sigmoid function.

3.2.1 ReLU

ReLU is the most commonly used activation function in deep learning, and is mathematically described in Equation 3.2. It will output the passed variable x if and only if $x \geq 0$, else it will output zero. The advantages of the ReLU activation function mainly consist of its computational efficiency, allowing the network to be trained more quickly and with fewer activated nodes Ramachandran, Zoph, and Le (2017).

$$f(x) = max(0, x) \tag{3.2}$$

whereby *f(x)* represents the output of the function and x the input of the function.

3.2.2 Sigmoid

The Sigmoid function is used in many other fields too, and its mathematical description can be found in Equation 3.3. The Sigmoid has an S-shape and is used to dampen large inputs, as well as having a smoothened 'threshold' for the neuron to become active.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

here *f(x)* is the output of the sigmoid activation function while x is the input.

3.3 Loss Functions

3.3.1 Mean Squared Error

The Mean Squared Error (MSE), as the name suggests, is concerned with the average of the squares of the errors for each output category. The MSE loss function is defined mathematically as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y_i})^2 \tag{3.4}$$

where y_i is the labelled output for the training data, $\tilde{y_i}$ is the given output by the neural network, $_i$ is a specifier of an output and n the size of the output vector.

3.3.2 Binary Cross-Entropy

In general, cross-entropy functions take two probability distributions as input which allows for the difference between the two to be quantified. Typically, they are applied as loss functions for classification problems.
Binary cross-entropy is a loss function for binary classification, where there is a single neuron in the output layer. The prediction is a value between 0 and 1, such as "does this image contain a cat?", hence the probability distribution is $[P, 1 - P]$. To evaluate the loss the Equation 3.5 is used.

$$Loss = -\frac{1}{N} \sum_{i=1}^{N} [y_i \cdot \log(\hat{y_i}) + (1 - y_i) \cdot \log(1 - \hat{y_i})] \tag{3.5}$$

where y_i is the labelled output for the training data, $\hat{y_i}$ is the given output by the neural network, $_i$ is a specifier of an output and N the size of the output vector.

3.3.3 Categorical Cross-Entropy

Categorical cross-entropy loss, or softmax loss, is used when multiple classes exist and the output layer has n neurons for the number of classes, hence for the probability distribution all of these n outputs sum to 1. The categorical cross-entropy loss function is defined mathematically in Equation 3.6.

$$Loss = -\sum_{i=1}^{N} (y_i \cdot \log(P(\hat{y}) \cdot w_i) \tag{3.6}$$

where y_i is the labelled output for the training data, $\hat{y_i}$ is the given output by the neural network, $_i$ is a specifier of an output while N the size of the output vector and w_i is the weighting of the given output class.

3.4 Different Architectures

The representation of the CNN output is of high importance to the success of an architecture because it influences how the loss must be minimized and ultimately the difficulty of the stochastic function-approximation the CNN has to perform. We opted for waypoint representations relative to the agents because early experiments with waypoints with respect to the whole environment resulted in inaccurate predictions. In the following sections four dif-

ferent variants are introduced, whose performances will later be compared.

3.4.1 XY-Variant

The first variant's output is of the form $[\Delta x, \Delta y, drive/dig]$. The first two elements encode the position of the waypoint relative to the current agent and are in the range $[-1, 1]$. After the CNN's prediction these values are post-processed such that the following holds true:

$$|\Delta x| + |\Delta y| = timesteps\_between\_assignments.$$

A digging agent can travel one cell per time step. Thus, the next waypoint is set to be as many steps away from the current position as there are time steps in between waypoint assignments, given by the hyperparameter *timeframe*. In our datasets the agents always moved their maximum distance between waypoint assignments, but in case agents should wait somewhere or only move parts of the maximum distance this scaling may pose problems.
The third element of the output tensor determines whether the agent is going to drive or dig, depending on a *digging-threshold*. In case the agent is supposed to drive, Δx and Δy are simply multiplied by 2. This is not problematic because in the training data the Δx and Δy of driving-waypoints are also normalized such that the dig/drive element of the output tensors implies this scaling.

layer	outputs	kernel	stride
Input	7	-	-
Conv2D+ReLU	16	2	1
Conv2D+ReLU	16	2	2
MaxPooling2D	16	2	2
Conv2D+ReLU	32	2	2
Conv2D+ReLU	32	2	2
MaxPooling2D	32	2	2
Flatten	2048	-	-
Dense+Sigmoid	48	-	-
Dense+Sigmoid	32	-	-
Dense+Linear	3	-	-

Table 3.1: Hyperparameters of the XY-network. Kernel and stride scalars are interpreted as tuples with both elements having the same value.

Table 3.1 shows the configuration of this architecture-variant. It is important to notice that the final fully connected layer has a linear, or pass-through activation function, enabling the network to output negative numbers as required for Δx and Δy. During training and validation (Section 3.6) the MSE was applied to quantify the difference between desired and predicted output.

3.4.2 Angle Variant

The second variant's output is of the form $[cos(\theta), sin(\theta), r, drive/dig]$, where θ and r are derived from Δx and Δy as in the first architecture, using Equations 3.7 and 3.8:

$$\theta = \frac{\Delta y}{\Delta x}, \tag{3.7}$$

$$r = \sqrt{\Delta x^2 + \Delta y^2}. \tag{3.8}$$

layer	outputs	kernel	stride
Input	7	-	-
Conv2D+ReLU	16	2	1
Conv2D+ReLU	16	2	1
MaxPooling2D	16	2	2
Conv2D+ReLU	32	2	2
Conv2D+ReLU	64	3	2
MaxPooling2D	64	2	2
Flatten	16384	-	-
Dense+ReLU	48	-	-
Dense+ReLU	32	-	-
Dense+Linear	4	-	-

Table 3.2: Hyperparameters of the angle-network. Kernel and stride scalars are interpreted as tuples with both elements having the same value.

The configurations of this architecture are shown in Table 3.2. Compared to the XY-variant, it is inherently less computationally efficient, due to the smaller strides in the second and the increased filter size of the final convolutional layer. For the initial fully connected layers the *ReLU* activation function was used, as the usage of the sigmoid activation function caused a trained network to ouput equal values consistently, regardless of input.
For this architecture, MSE was applied as the loss function for training and validation (Section 3.6)

3.4.3 Box Variant

The box variation is so called since an agent has a limited range of positions that it can reach from the initial position within a certain timeframe, this subsection of the environment can contain many different positions and if highlighted on the gui would appear as a diamond-like box with the agent at its centre. Since the agent can only move in the 4 cardinal directions, its total displacement between waypoint generation is a combined change in x and y coordinates of 20 positions as shown in Equation 3.4.1, which translates into 841 different positions in the environment that the agent can reach. We scale each of these positions down to an array of length 61, as if the agent could only move 5 times between waypoint generation. Each one of these positions represents roughly 14 positions in the simulation environment. Since not every value in the positional array occurs equally in the training data, we take the inverse of the frequency of an output position and use this as the weighting for the loss function as in Equation 3.3.3.

The CNN outputs a prediction of an agent's waypoint position in the condensed version of the positional array by using categorical cross-entropy loss as described in Subsection 3.3.3. This array position is then scaled back to one within the larger environment by reversing the previous operation. In addition the architecture makes a separate prediction for whether the agent should dig or drive to the waypoint location using binary cross-entropy in Subsection 3.3.2.

Table 3.3 shows the layers of the CNN architecture, here the split output tensors for digging movement and the position can be seen. The network includes a small chance of neuron dropout to reduce the risk of overfitting. While the digging output uses two fully connected layers with sigmoidal activation from Equation 3.3 since it predicts a value between 0 and 1. The positional output uses a ReLU as previous layers but in the output layer there is softmax since we use categorical cross entropy and select a single value from the position array.

layer	outputs	kernel	stride
Input	7	-	-
Conv2D+ReLU	16	2	1
Conv2D+ReLU	16	2	1
MaxPooling2D	16	2	2
Conv2D+ReLU	32	2	2
Conv2D+ReLU	32	2	2
MaxPooling2D	32	2	2
Flatten	8192	-	-
Dropout	0.1	-	-
Digging Output			
Dense+Sigmoid	16	-	-
Dense+Sigmoid	1	-	-
Position Output			
Dense+ReLu	64	-	-
Dropout	0.1	-	-
Dense+Softmax	61	-	-

Table 3.3: Hyperparameters of the Box-network. Kernel and stride scalars are interpreted as tuples with both elements having the same value. The branching of the network into output predictions for digging or driving and position are indicated by indented lines of the table.

3.4.4 Segments Variant

This last variant was a late addition to the other three architectures and implements another idea on how to represent desired positions relative to the agents. It discretises the direction in which the agent needs to move and a binary value to indicate the digging or driving movement. The angle in which the agent can move is equally divided into 16 segments of 22.5° each. Each segment is given an entry in an output vector of 16 and assigned a value of 1.0 in the training data while non-desirable directions are simply set to 0.0. As for the XY-variant, the agents will always move as far as the time frame between assignments allows. The models are trained using the categorical cross-entropy loss function (3.3.3 for the general direction, and binary cross-entropy (3.3.2) as a loss function for the digging or driving indication.

3.5 (Adam) Optimizer

A neural network improves by adjusting the weights between nodes to reduce the empirical loss and an optimizer aims to find a set of weights that reduces this loss. In this particular project, the

layer	outputs	kernel	stride
Input	7	-	-
Conv2D+ReLU	16	2	1
Conv2D+ReLU	16	2	2
MaxPooling2D	16	2	2
Conv2D+ReLU	32	2	2
Conv2D+ReLU	32	2	2
Flatten	8192	-	-
Dropout	8192	-	-
Dense	64	-	-
Direction Output			
Dense+ReLU	32	-	-
Dropout	32	-	-
Dense+Softmax	16	-	-
Digging Output			
Dense+ReLU	16	-	-
Dense+Sigmoid	1	-	-

Table 3.4: Hyperparameters of the Segment-network. Kernel and stride scalars are interpreted as tuples with both elements having the same value. The dropout rate across Dropout layers is 0.2. The branching of the network into output predictions for digging or driving and position are indicated by indented lines of the table.

Adam, Kingma and Ba (2014) optimizer is used, as it is one of the leading adaptive learning rate algorithms and has been empirically demonstrated to perform comparatively or even better than similar algorithms for CNNs Berrah and Laboissière (1999). The Adam optimizer is a variation of the class of stochastic gradient descent algorithms. It combines aspects of RMSProp, Hinton, Srivastava, and Swersky (2012) and AdaGrad, Duchi, Hazan, and Singer (2011) with a term for momentum. For each parameter Adam has an adaptive learning rate, calculated with both the exponentially decaying mean of previous gradients m_t the first moment (mean) to represent momentum. As well as the exponentially decaying average of the gradients squared v_t which applies the second moment (variance). The first and second moment are calculated using Equations 3.9 and 3.10 respectively.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{3.9}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{3.10}$$

where β_1 is the hyperparameter for the first moment decay rate, β_2 the hyperparameter for the second moment decay rate and g_t is the loss function gradient.

The initialisation of the first and second moment results in an early bias since m_t and v_t will be a vector of zero values. To correct the bias in the first moment Equation 3.11 is used and for the second moment Equation 3.12.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3.11}$$

$$\hat{v}_t = \frac{m_t}{1 - \beta_2^t} \tag{3.12}$$

where $\hat{m}_t$ is the corrected first moment, $\hat{v}_t$ is the corrected second moment and t is the epoch number.

Using the previous values the return value of the Adam weight update function can be computed, the weights are updated using Equation 3.13.

$$w_t = w_{t-1} - \alpha \cdot \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon} \tag{3.13}$$

where w_t is the weights vector at time t, α is the learning rate hyperparameter and ϵ a constant.

The aforementioned hyperparameter ϵ is required to prevent divisions by zero, and is therefore kept extremely small. The exact values for each hyperparameter can be seen in Table 4.2.

3.6 Validation

Separating random samples from the training set to use for validation is a major tool for avoiding overfitting. We used 20% of our data to get an indication of how well our models might perform on unseen data. Early stopping was used with a patience of 10. This means that if the model does not improve on the validation set for 10 consecutive epochs, training stops and the model with the lowest validation loss is saved for testing.
We also created randomised test sets of 100 input-output pairs and checked for the mean and the distribution of the prediction error when tuning the networks. This would give hints about how far off the predictions would be in terms of cells, as the raw losses of the different architectures do not allow for such intuitions.

4 Experimental Setup

Four different environment configurations were chosen in order to test the three architecture variants against each other. For each of these 12 experiments, 30 CNN models were trained and tested when integrated in the simulation.

4.1 Levels of Difficulty

The first and simplest environment configuration, denoted as *Baseline*, is characterized by the agents being placed in a star-like formation around the centre. Their initial position uncertainty is set to 10, meaning that their positions are slightly randomly initialized in the 256 by 256 environment. There is no wind and the fire expands in a circle from the centre with a speed of around 1/10th that of the digging agents. The strategy with which the fire is contained in the training data has a very strong influence on the behaviour and success of the trained models. To keep it simple, in this first experiment the agents only dig firebreaks around the slowly expanding fire in a clock-wise circular fashion.

The second environment adds *Wind* from different directions and at different speeds. The overall wind speed was set a bit higher, resulting in fire propagation speeds of 0.15-0.3 times slower than the agent digging speed, depending on the wind speed level. The strategy was to drive to the centre and encircling the fire closer to its origin. For faster fire speeds care is taken to make sure the agents do not move too close to the fire in the direction of the wind.

The third experiment does not involve wind, but instead increases the uncertainty in the agent initialisation positions to 30. This requires the human operator to mix digging and driving to mitigate the randomisation.

The last experiment was dubbed *Uncertain+Wind* and simply combines the complexity of the previous two environments together. The containment strategy is similar to the previous one, trying to even out gaps with driving at twice the digging speed. The firebreaks are dug a lot closer to the fire in order to minimise the amount of burnt forest. Next to the parameters of the environment this also contributes to a harder problem for the algorithms.

Table 4.1 shows how much training data was provided for each experiment. The third column gives an indication about how many complete fire containment runs the number of samples correspond to. We arrive at this approximation by dividing by five agents and then again by about eight waypoint assignments per run.

Environment variant	samples	est. runs
Baseline	1435	36
Wind	4355	110
Uncertain	4265	105
Uncertain+Wind	5200	130

Table 4.1: Amounts of training data (input-output pairs) per environment and a corresponding estimate of runs recorded during LfD.

4.2 Hyperparameters

The first six entries in Table 4.2 represent the hyperparameters governing the characteristics of the environment. They decide how fast the fire spreads and how long an individual cell remains burning. While their absolute values are of lesser importance, their ratios were adapted to create the fire propagation behaviour. We tried to make the resulting containment problems easily solvable by a human, yet non-trivial for simple algorithms.

The hyperparameters for the Adam Optimizer were kept at default. We experimented with higher learning rates leading to fewer training epochs, but also made training more unstable which resulted in static output for all input images.

For CNN training the batch size was set to 64 which gives a time advantage over using smaller batch sizes. Larger batch sizes, up to the constraints of the hardware, also have the effect that gradients become more stable because they are averaged over a larger subset of the training data. We made use of early stopping where after 10 epochs of failing to improve on the validation loss, the weights of the model with the best performance on the validation set were restored and saved for testing. Said validation set contained 20% of the total amount of randomly shuffled training data such that it represented the whole data set well, decreasing the chance for overfitting.

Another hyperparameter is the *digging-threshold*, determining how the CNN output for digging or driving would be interpreted. As not all architectures output binary values for this decision,

the threshold was set to 0.5 for all architectures and experiments, leading to a digging agent if the model's output exceeds that value. The distances to the new waypoints scale linearly with the time frame between waypoint assignments. In order to avoid inaccuracies in the CNN outputs greatly affecting the agent's behaviour, this value is set to 20 for generating training data and all experiments.

index	parameter	value
1	fire speed factor (no wind)	0.3
2	fire speed factor (wind)	0.5
3	heat spread factor	0.5-1.5
4	ignition threshold	3
5	fuel consumption per time step	3
6	cell fuel	20
7	ϵ	1e-7
8	β_1	0.9
9	β_2	0.999
10	α	0.001
11	batch size	64
12	patience (early stopping)	10
13	validation split	0.2
14	digging threshold	0.5
15	time frame	20

Table 4.2: Hyperparameters of the environment (rows 1-6), Adam Optimizer (rows 7-10), CNN training (rows 11-13) and during test runs (rows 14-15).

4.3 Performance Measure

The performance of our algorithms is measured by counting the successful fire containments out of all attempts. Per experiment, 100 tests are run to capture the average amount of containments, together with the standard deviation σ and standard error (SE). For the successful runs, when the fire is entirely surrounded by firebreaks, the amount of potentially burnt cells is measured as well. This is done using a breadth-first-search algorithm that starts in the middle cell like the fire and searches for non-firebreak cells until it has reached the bounds of the contained area. It includes a heuristic that regularly checks newly explored cells for an unobstructed path to the boundaries of the environment. If no firebreak cells are in the way it means that the fire is not yet contained and the algorithm can be interrupted, greatly improving the speed of the automatic testing.

5 Results

In this section we present the results of the experiments with the four different architectures tested on four environment configurations. The exact values for all means and standard deviations can be found in Appendix C. Figure 5.1 shows the mean number of successful fire containments out of 100 simulation runs averaged over the 30 models per environment per architecture. One can see that the XY and angle-architectures successfully contain the fire most frequently. While all architectures perform best on the *Baseline* environment where the angle-variant is capable of near guaranteed containment at a rate of 96.9/100 and a standard deviation of 4.9. The box-variant has the worst containment rate overall, even compared to the segments approach that is also based on binary classification.

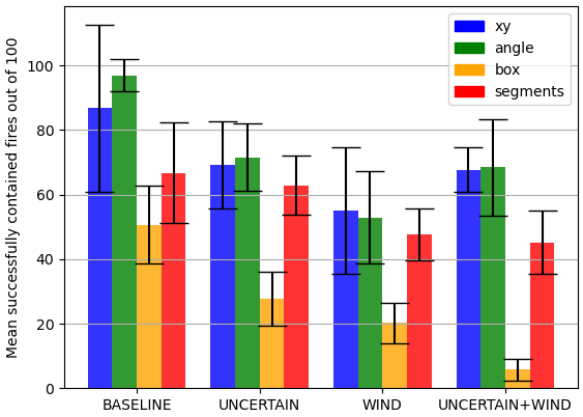

Figure 5.1: Mean total successfully contained fires out of 100 for 30 trained models per environment per architecture.

Figure 5.2 shows the mean number of burnt cells in the environment after a successful containment. To be clear, the rate at which the fire was contained by an architecture does not influence these values, only the area encircled by the agents for successful runs. This metric could allow us to identify differences between the environment configurations in terms of containment strategies used in the LfD. Furthermore, we could find possible deviations of certain model variants from the strategies

which would express themselves in different values within an environment configuration. The bar plot shows that there are differences between the environment configurations with the *Baseline* environment having the highest mean values ranging between 5738 and 6480 burned cells out of about 65.000 (256^2) cells in the environment. The differences between the model variants seem small and the highly overlapping error bars indicate that we will not be able to make claims of statistical significance about them.

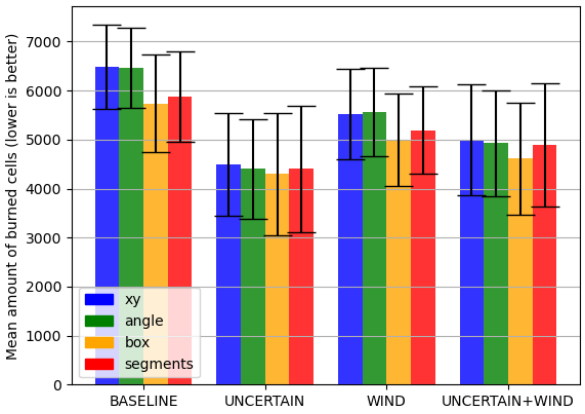

Figure 5.2: Mean number of cells burnt in the environment per architecture after successful fire containment.

6 Conclusions

This research aimed at comparing CNN-architectures with different output representations prescribing waypoints relative to the agents' current positions. With the XY- and angle-variants we had two approaches that operated in a continuous space around the agent. The box- and segment-variants attempt to solve the problem through categorical classification, leading to a finite number of discrete possible positions relative to the agent's current position.

Figure 5.1 indicates that the former architectures perform better in terms of rate of containment. This is backed by the statistical tests shown in appendix A where significant differences in means were found for nearly all comparisons to the other two variants. No such differences could be found between XY and angle output representations themselves.

While we can group the box and segments variants together, the performance of the former is noticeably worse, especially in the last three environments. One explanation may be that there are 61 possible positions to choose from as opposed to 16. While it has the added benefit of providing us with information about the distance to the next waypoint and not only the angle, in our training and test data sets this feature was never used and thus comes as a disadvantage. We theorise that the greater number of dimensions in the output tensor increases the difficulty of the classification problem and therefore causes the box-variant to fall behind the segments-approach when both receive the same amount of training data. This could be solved by dividing the space around the agent into fewer possible waypoint-positions. However, it would also result in less accurate control of the agent because the boxes would incorporate even more single cells in the environment.

For some architecture-environment combinations the error bars and corresponding standard deviations (see Appendix C) are unusually large. This can be explained by the fact that some models failed to recognize patterns in the input and thus produced static, i.e. unchanging output. This is especially the case for the XY-variant and may hint towards a weakness in learning stability of this particular architecture.

Concerning the amounts of burned cells displayed in Figure 5.2, there appear to be no differences between the network architectures when they achieve successful runs. This was to be expected since all algorithms learn from the same demonstration data. However, there were statistically significant differences between the overall means of burned cells of the four environments, regardless of CNN architecture, as determined by one-way ANOVA ($F(3, 26884) = 4243, p < 0.01$) (see Appendix B). Tukey post-hoc tests revealed that for all group comparisons the differences in means were significant ($p < 0.01$). That does not come as a surprise because the containment strategies differ in how close to the fire the agents operate. For example, due to the fact that the *Baseline* environment was solved without letting the agents drive towards the middle first, the area contained by them was largest. Nevertheless, we need to be cautious with declaring these results meaningful because we have about 5000 to 9000 data points

per group which amplifies statistical significance. The means of these large groups lie not very far apart and have relatively large standard deviations (see Table C.3).

Returning to the research question "*Can a waypoint-generating convolutional neural network coordinating a multi-agent system be effective at containing wildfires in a simulated environment?*", the results show that for the XY and angle variants this is the case even for the most complex of environments where a greater than 50% success rate was achieved. The categorical classification methods, in particular the box-variant did not manage to reliably contain the fires in environments with increased complexity.

7 Discussion

The results of this project have shown that convolutional neural nets are able to successfully contain a forest fire in basic simulations. However, the simulations as in this project are small-scale and not reflective of real-world situations. Current wildfires cover much more complex areas than considered in this project and require hundreds, if not thousands, of people to be contained. Moreover, the environment can be far more chaotic and unpredictable with sparks from wildfires known to travel in the wind and create new pockets of fire up to 30 metres away. It would be interesting to see future research considering more complex and realistic environments. With different terrain types or more realistic wind simulation, many other directions are possible.

As the data provided when training the networks plays a large role in the performance of the networks, it is important to understand the consequences of different containment strategies for the generation of the training data. In real-world situations, encircling the fire by moving clockwise will not always be the most optimal strategy. It would be interesting to see if there are architectures that can be taught to apply different strategies and choose the correct one given more complex situations. One could also investigate the influence of specific containment strategies on our architectures.

The amount of data itself is also highly influential. Not enough data or a lower prevalence of specific situations in the data can lead to inaccuracies by the network. By augmenting the data or automatising the data generation one could obtain more significant results compared to those discussed in this paper.

Another direction to explore is transfer learning. One could train architectures on a basic form of the problem and continue training afterwards on more complex problems to see if this improves performance or reduces the required amount of training data. Successful results could have implications for the handling of possible extensions to the problem environment.

The initial goal of this research was to use an auto-encoder architecture to classify individual pixels in the environment as either waypoints or non-waypoints. These waypoint locations would have been fed to other algorithms to assign them to the different agents, making the approach more flexible by allowing for different amounts of agents. After initial problems with generating a 256×256 output, trying to simplify the classification problem by using 64×64 or even 16×16 outputs still yielded poor results. While we switched to other architecture variants due to time constraints, we still believe it to be possible to achieve reasonable results by further tweaking the architecture and providing more demonstration data. Alternatively, one could investigate the effectiveness of first applying LfD with reinforcement learning techniques and then improving the initial performance with reinforcement learning alone to fine-tune the behaviour.

The representation of the environment state as the input for the neural networks could also be reconsidered. One example we can provide is the encoding of the wind direction. In this project, an image channel with discrete uniform values is provided as part of the input, categorising the wind direction. This results in the network having to learn that the wind going north or north-east has closely related consequences for the fire propagation. However, the same information can be provided by using two scalar values, each representing either the longitudinal or the latitudinal direction. This way, the relation between the north and northeast directions can be implied beforehand.

This project has not focused on minimising the total area burnt. One could attempt reinforcement learning techniques to achieve less land loss,

instead of focusing only on fire isolation, as this would be an objective in real-world situations.

References

Meinrat O Andreae and Johann Georg Goldammer. Tropical Wildland Fires and other Biomass Burning: Environmental Impacts and Implications for Land Use and Fire Management. *Conservation of West and Central African Rainforests (K. Cleaver et al., eds.)*, pages 79–109, 1992.

A.-R. Berrah and R. Laboissière. *SPECIES: An Evolutionary Model for the Emergence of Phonetic Structures in an Artificial Society of Speech Agents*, volume 1674. Springer Berlin, 1999.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of machine learning research*, 12(7), 2011.

Environmental Protection Agency EPA. Climate Change Indicators: Wildfires, 2021. URL https://www.epa.gov/climate-indicators/climate-change-indicators-wildfires.

Travis Hammond, Dirk Jelle Schaap, Matthia Sabatelli, and Marco A Wiering. Forest Fire Control with Learning from Demonstration and Reinforcement Learning. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural Networks for Machine Learning Lecture 6a Overview of Mini-batch Gradient Descent. *Cited on*, 14(8):2, 2012.

Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Stefan JL Knegt, Madalina M Drugan, and Marco A Wiering. Opponent Modelling in the Game of Tron Using Reinforcement Learning. In *ICAART (2)*, pages 29–40, 2018.

NationalForestFoundation. How Trees Survive and Thrive After A Fire, 2021. URL https://www.nationalforests.org/our-forests/your-national-forests-magazine/how-trees-survive-and-thrive-after-a-fire.

David C. Plaut, Steven J. Nowlan, and Geoffrey E. Hinton. Experiments on Learning by Back Propagation. Technical report, 1986.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. *CoRR*, arXiv preprint arXiv:1710.05941, 2017. URL http://arxiv.org/abs/1710.05941.

N. Rocholl. Isolating Wildfires Using a Convolutional Neural Network based Multi-Agent System. Bachelor's thesis, University of Groningen, The Netherlands, 2020.

D. Silver et. al. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, page 484–489, 2016.

Richard S Sutton, Andrew G Barto, et al. *Introduction to Reinforcement Learning*, volume 135. MIT press Cambridge, 1998.

M. Wiering and M. Doringo. Learning to Control Forest Fires. *Proceedings of the 12th international Symposium on 'Computer Science for Environmental Protection'*, page 378–388, 1998.

J. Denker D. Henderson R. Howard W. Hubbard Y. LeCun, B. Boser and L. Jackel. Handwritten Digit Recognition with a Back-propagation Network. *NIPS*, pages 79–109, 1989.

Yoes Ywema. Learning from Demonstration for Isolating Forest Fires Using Convolutional Neural Networks. Bachelor's thesis, University of Groningen, The Netherlands, 2020.

Contributions

Jonathan Hewlett

- Initial work on the simulation environment including nodes, GUI properties and wind and fire spreading
- Helping to collect training data
- Data augmentation of training data including rotation and shifting
- Design of the box CNN architecture, including the data augmentation and compatibility with the NN controlled simulation
- Updating and adjusting model
- Tweaks for translation from saved data to box CNN output
- Main writing contributions: Introduction, Methods, CNN, Results

Markus Bäuerlein

- Help with simulation
- Data collection
- Setting-up of pipeline for data saving, translation to input/output pairs, CNN training
- Work on pixel-classification architecture until new variants were introduced
- XY-variant, assisting other variants
- Automation of CNN performance testing
- Results gathering, statistical tests
- Main writing contributions: Methods, Experimental Setup, Results, Conclusions

Thijs Visee

- Setup and programming of the first model, view and controller for running the simulation and data generation.
- Angle variant
- Data translation for angle variant
- Data translation for segments variant
- Main writing contributions: CNN section, Discussion
- Assisting with results gathering

A Appendix

Treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Significant?
XY vs. Angle	3.32	0.0939	NO
XY vs. Box	11.85	0.0010	YES
XY vs. Segments	6.59	0.0010	YES
Angle vs. Box	15.17	0.0010	YES
Angle vs. Segments	9.90	0.0010	YES
Box vs. Segments	5.26	0.0017	YES

Table A.1: *Baseline* environment comparison between architecture variants. Significant differences were found using a one-way ANOVA (F(3, 116) = 45.89, p = 1.1102e-16). The Tukey HSD post-hoc test is used to determine significance between treatment pairs using a confidence level $\alpha = 0.01$.

Treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Significant?
XY vs. Angle	1.14	0.8336	NO
XY vs. Box	21.17	0.0010	YES
XY vs. Segments	3.26	0.1023	NO
Angle vs. Box	22.31	0.0010	YES
Angle vs. Segments	4.40	0.0124	YES
Box vs. Segments	17.91	0.0010	YES

Table A.2: *Uncertain* environment comparison between architecture variants. Significant differences were found using a one-way ANOVA (F(3, 116) = 108.18, p = 1.1102e-16). The Tukey HSD post-hoc test is used to determine significance between treatment pairs using a confidence level $\alpha = 0.01$.

Treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Significant?
XY vs. Angle	0.90	0.9000	NO
XY vs. Box	14.22	0.0010	YES
XY vs. Segments	3.05	0.1414	NO
Angle vs. Box	13.32	0.0010	YES
Angle vs. Segments	2.15	0.4292	NO
Box vs. Segments	11.17	0.0010	YES

Table A.3: *Wind* environment comparison between architecture variants. Significant differences were found using a one-way ANOVA (F(3, 116) = 43.27, p = 1.1102e-16). The Tukey HSD post-hoc test is used to determine significance between treatment pairs using a confidence level $\alpha = 0.01$.

Treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Significant?
XY vs. Angle	0.39	0.9000	NO
XY vs. Box	34.47	0.0010	YES
XY vs. Segments	12.54	0.0010	YES
Angle vs. Box	34.86	0.0010	YES
Angle vs. Segments	12.93	0.0010	YES
Box vs. Segments	21.93	0.0010	YES

Table A.4: *Uncertain+Wind* environment comparison between architecture variants. Significant differences were found using a one-way ANOVA (F(3, 116) = 267.40, p = 1.1102e-16). The Tukey HSD post-hoc test is used to determine significance between treatment pairs using a confidence level $\alpha = 0.01$.

B Appendix

Treatments pair	Tukey HSD Q statistic	Tukey HSD p-value	Significant?
Baseline vs. *Uncertain*	153.42	0.0010	YES
Baseline vs. *Wind*	65.64	0.0010	YES
Baseline vs. *Uncertain+Wind*	102.22	0.0010	YES
Uncertain vs. *Wind*	71.81	0.0010	YES
Uncertain vs. *Uncertain+Wind*	39.63	0.0010	YES
Wind vs. *Uncertain+Wind*	31.30	0.0010	YES

Table B.1: Comparison of amounts of burned cells per environment configuration across all architectures. Significant differences were found using a one-way ANOVA (F(3, 26884) = 4,243.71, p = 1.1102e-16). The Tukey HSD post-hoc test is used to determine significance between treatment pairs using a confidence level $\alpha = 0.01$.

C Appendix

Environment	Architecture	Mean contained fires out of 100	Standard deviation
Baseline	XY	86.8	25.8
	Angle	96.9	5.0
	Box	50.7	12.0
	Segments	66.8	15.5
Uncertain	XY	69.3	13.5
	Angle	71.6	10.4
	Box	27.8	8.4
	Segments	62.9	9.2
Wind	XY	55.1	19.6
	Angle	52.9	14.4
	Box	20.3	6.3
	Segments	47.7	8.1
Uncertain+Wind	XY	67.8	7.0
	Angle	68.5	14.9
	Box	5.8	3.3
	Segments	45.2	9.7

Table C.1: Mean amount of contained fires out of 100 per architecture per environment, averaged over 30 models with standard deviation. The bar plot in figure 5.1 is based on these values.

Environment	Architecture	Mean amount of burned cells	Standard deviation
Baseline	XY	6480	865
	Angle	6466	825
	Box	5738	996
	Segments	5873	916
Uncertain	XY	4495	1047
	Angle	4406	1014
	Box	4301	1250
	Segments	4401	1282
Wind	XY	5513	919
	Angle	5566	907
	Box	4997	937
	Segments	5187	888
Uncertain+Wind	XY	5006	1131
	Angle	4932	1080
	Box	4611	1134
	Segments	4899	1258

Table C.2: Mean amount of burned cells per architecture per environment, with standard deviation. The bar plot in figure 5.2 is based on these values.

Environment	Mean amount of burned cells	Standard deviation
Baseline	6216	944
Uncertain	4418	1133
Wind	5381	933
Uncertain+Wind	4941	1147

Table C.3: Mean and standard deviation of amounts of all burned cells per environment regardless of architecture variant.