Using a convolutional neural network to identify the shape of WIMPs in the galaxy center

Mattias Hollen Aasen

December 14, 2022

Abstract

In this project we develop a convolutional neural network that will attempt to solve the binary classification problem of differentiating between images containing dark matter signals and images without. We compare one model that works on large images and one that works on small ones, in addition to having Gaussian noise with a standard deviation of 0.8 added to its training images. We found both models reaching an accuracy close to 100% and an AUC of 1, but the second model outperforms the larger one in terms of speed, using 4 seconds per epoch as opposed to 7 minutes per epoch.

1 Introduction

The nature of dark matter is one of the most exciting mysteries of physics in our time. It is said to make up 25% of the content of our galaxy, outweighing the 5% that is regular matter by far, yet so far it has eluded any attempt at observing it. Theories range from conventional matter that we know from the standard model of physics to modifications of the gravitational theory or unconventional particles. Since the 1980s the most popular candidate for dark matter has been the Weakly Interacting Massive Particle (WIMP), which is theorized to only interact through the weak force. Observing a particle like this poses a serious challenge, but if it exists one should be able to observe photons that emerge from these weak interactions. WIMPs annihilating against each other or decaying through weak interactions could produce particles from the standard model of physics that can interact through the photoelectric effect, producing very high-energy gamma rays which we can observe.

The Cherenkov Telescope Array (CTA) is a future observatory that will be located on both the northern hemisphere on the island of La Palma, Spain, and on the sourthern hemisphere in the Atacama Desert in Chile [1]. With a combination of three different telescopes the telescope array is expected to observe photons with an energy spanning a range of 20 GeV up to 300 TeV, exceeding any previous ground-based observatories and the known electromagnetic spectrum . The wide field of view, improved sensitivity and angular resolution of the CTA opens up for a more detailed imaging of gamma sources than ever before. This is especially interesting for observing the galaxy center, one of the key science projects for the CTA, as it is crowded with gamma ray sources that so far have been difficult to differentiate between.

In this project we will look at the possibility of training a Convolutional Neural Network to identify the shape of a WIMP , hereby referred to as dark matter, in the galaxy center based

on its geometrical properties. This project will be a proof of concept where we will aim to make our network recognize the dark matter signal with an accuracy above 70% and an AUC above 0.5. All of the code used for this project is made using python and can be found at github.

2 Theory

2.1 Supervised Neural Networks

A neural network, as seen in figure 1, is a type of machine learning model that builds up the foundation of deep learning [2]. The first part of the network is the input layer which consists of input nodes, also referred to as features, which we define as the vector $\mathbf{x}$. Following the input layer is the first hidden layer, containing some m number of hidden nodes. Each node consists of two functions; a summation operator and an activation function. The summation operator connects each node to all the features in the input layer, and can be written as the equation

$$\text{input for node j} = b + \sum_{i=1}^{n} x_i w_{i,j} \quad [3] \quad ,$$

where $w_{i,j}$, which we refer to as the weight, corresponds to a numerical variable unique for each node j and feature i, and b is what we refer to as the bias.
The bias works as a node alongside but independent of the layer going into the next hidden layer in the network. Its numerical value is typically set to 1 and it is connected to each node in the next layer with its own set of weights. For a hidden layer consisting of m nodes the bias can be considered as a vector $\mathbf{b}$ of size m where each element is connected to its own node in the layer. After the total input has been calculated and passed into the node it is put into an activation function. There are different types of activation function a network can utilize, but the common feature of them is that if some condition is met the value of the node will be set to 0. Otherwise the value will be passed on to the next layer. The process will be repeated for every hidden layer in the network until we reach the output layer.
In the case where the supervised neural network acts as a classifier it will assign every input to one of the categories defined for the problem at hand. A binary classifier uses only two categories, as the name suggests. The most standard metrics used to measure how well a network performs are accuracy and loss. Accuracy tells us how many percent of the total predictions were correct while the loss function is a measure of how far off a prediction is. The loss for a single example tells us the difference between the actual output and the target output. As we want the model to calculate outputs as close to the targets as possible, we seek to minimize the loss for the entire model.

There are different methods for measuring loss in a neural network. The one we consider in this project is the cross-entropic loss function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) \quad , \tag{1}$$

where $L(x, y, \theta) = -\log p(y|x;\theta)$ is the loss for each input [4]. The gradient for the loss is calculated through the equation

$$\mathbf{g} = \boldsymbol{\nabla}_\theta J(\theta) = \frac{1}{m} \sum \boldsymbol{\nabla}_\theta L(x^{(i)}, y^{(i)}, \theta)$$

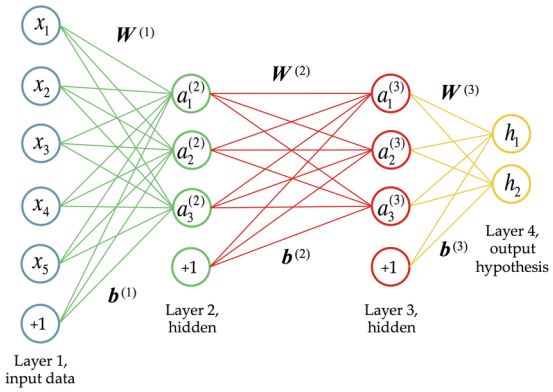


Figure 1: An illustration of how a neural network is built up . Here W denotes the different weights, b the bias and a the nodes, forming the parameters θ that are updated for every epoch in order to optimize the model[5].

which is used to steer the loss function to a minimum, which is where $\nabla_\theta J(\theta) = 0$, ideally the global minimum.
Stochastic gradient descent will then change the parameters in the model by the relation

$$\theta \leftarrow \theta - \epsilon g$$

where ϵ is the learning rate. We see that the choice of learning rate decides how much the parameters change after each epoch. Choosing a too large learning rate might hinder the model from reaching the minimum, while a learning rate that is too small could lead to a very slow network. The learning rate is an example of what is referred to as hyperparameters, which are parameters we define before training the model. The hyperparameters are important for training the network but are not part of the final model. Other examples of hyperparameters are the number of hidden layers, number of nodes in each layer, number of epochs and batch size.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network that uses a linear operation called convolution in at least one of its layers. For a 2-dimensional input like a grey-scale image, which only has one value for each pixel, convolution can be defined as a discrete operation of the form

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \quad ,$$

with I being the input and K being the kernel [6]. In the case where both are two-dimensional tensors we can consider convolution as a series of dot product operations, as shown in figure 2.

We can imagine a 2 x 2 tensor cut-out of the input tensor interacting with the 2 x 2 kernel, forming a dot product as the first element in an output tensor. After this operation the kernel moves one step to the right and interacts with a new cut-out, repeating the process until there are no new columns to move onto, after which it moves down a step to repeat the process for the new row. For a $(m \times n)$ input tensor and a $(u \times v)$ kernel tensor there will be an output map

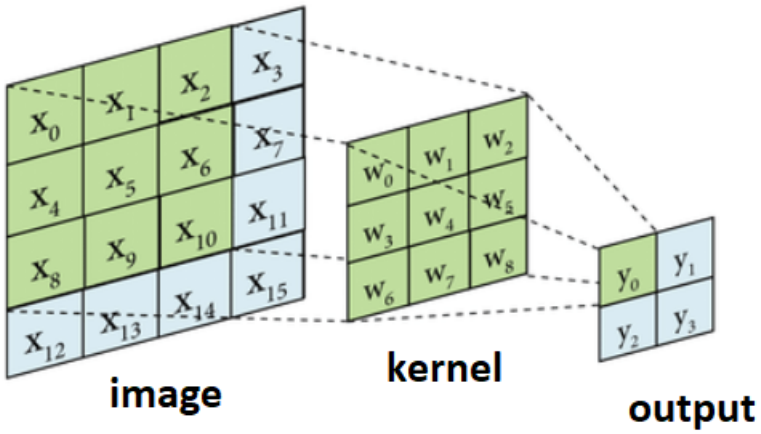


Figure 2: A visualization of the convolutional operation [7].

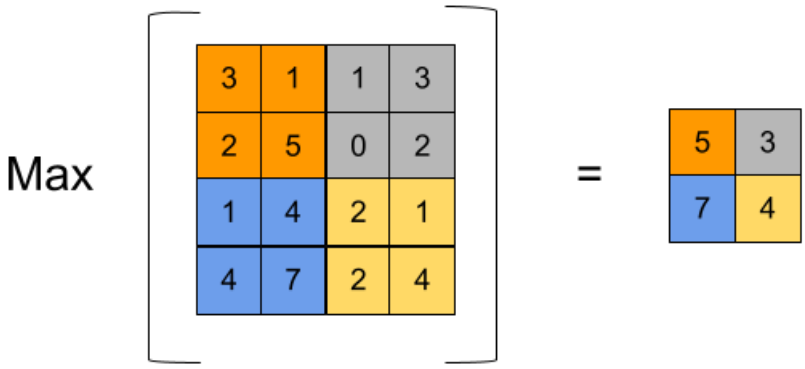


Figure 3: An illustration of how the max pooling function modifies the output units [8].

with dimensions $((m - u + 1) \times (n - v + 1))$.
The result of this is that the convolutional layers is built up on sparse interactions, where each output unit only interacts with a few output units. This is in contrast to a traditional neural network layer where each output is connected to every input. This in turn leads to a lot less calculations when computing the output, which cuts down on the runtime. The kernels in the convolutional layers require a lot less parameters than the nodes in a traditional layer. If the input from an image consist of millions of pixels the kernel in the convolutional layer can produce meaningful outputs with kernels consisting of tens or hundreds of pixels. This relatively small number of parameters is useful for reducing overfitting and reducing the memory usage of the model.

A typical convolutional layer also contain a pooling function which is used to modify the output of the layer. An example is the maximum pooling function as seen in figure 3 , which uses the maximum value of a given region of adjacent outputs to replace the value in a specific output unit.

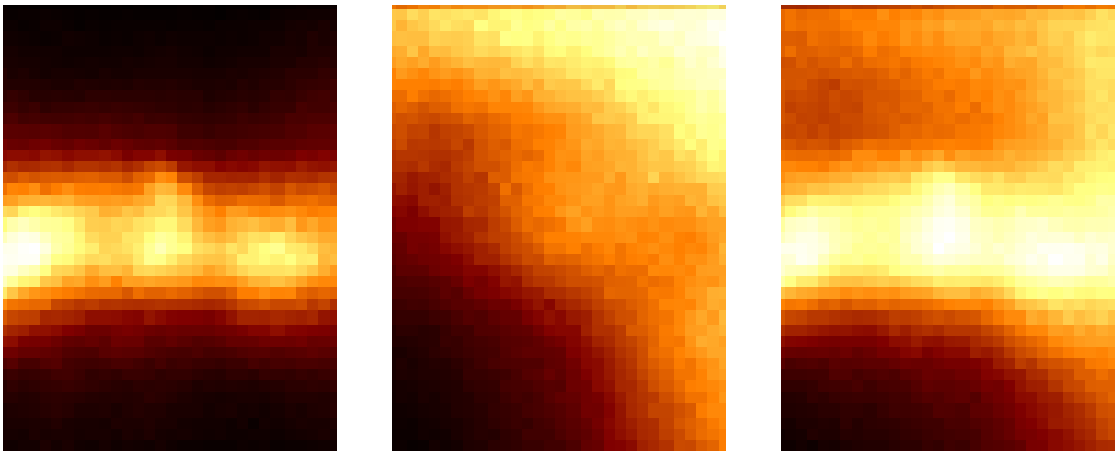
3 Methods

3.1 Simulating events

In order to train the network a set of 12 000 images were created. 6000 of these images were created with a dark matter source included, while the other 6000 were created without. The images were created by simulating observations of the galactic center using the gammapy package for python. Each simulation is defined with a unique seed for the random generator of photon counts. We define the observation to be pointed towards the galaxy center at coordinates $(0°, 0°)$ and the exposure time to be 180 hours. The instrument response function (IRF) we use is $south\_z60\_50h$. We define the energy range to go from 10 GeV to 100 TeV. For this reason we use the southern telescope, which can observe gamma rays with larger energy than the northern ones.
The background is created by using models from the first data challenge CTA launched, and examples of the simulated observations can be seen in figure 4 and 5. 1/3 of the images are created using the IEM as the background model, another 1/3 by using Fermi bubbles as background and the remaining 1/3 by using a combination of these two models. Due to the Fermi bubbles model having a larger energy flux than the IEM model we increase the amplitude of the IEM model to the same magnitude in order for both models to be visible. Also, for the 6000 images containing dark matter we turn the amplitude of the dark matter source up. Initially there was an attempt to use the dark matter source model at its original flux strength, where the theory was that a neural network would spot the shape of an excess given by this source. However, each simulation is a Poisson distribution with certain conditions given by the source models. The fluctuation of photon counts would likely be larger than the excess of photon counts given by the dark matter signal, and it would be impossible for a network to spot the shape of the signal. This was made evident as we first trained the network on this batch of images and ended up with 50% accuracy on the training set, which tells us that the network cannot spot any pattern and is just guessing for each example.
Instead, we start at an amplitude that is barely visible to the naked eye compared to the background and gradually increase it until it becomes clearly visible. This is done in a for-loop with 10 iterations. In this project we ignore the physical implications of the flux strength, but this is something we will have to address when we want to create a test set based on more realistic simulations.

3.2 Image size and cropping

Creating the simulations was the most time-demanding part of this project, and was heavily linked to the image size. At first we created images that were larger than the default size set by gammapy. The idea was that the more pixels we have the larger variation between each image we would have, as each of the simulations are generated with a unique seed. If the images were made too small there is a risk of the images being very similar. 200 images are created for each choice of model and dark matter strength (in the case of the 6000 images with dark matter). If each of these images are more or less the same then the network will use 60 different images 200 times each, which most likely will create a huge problem with overfitting.

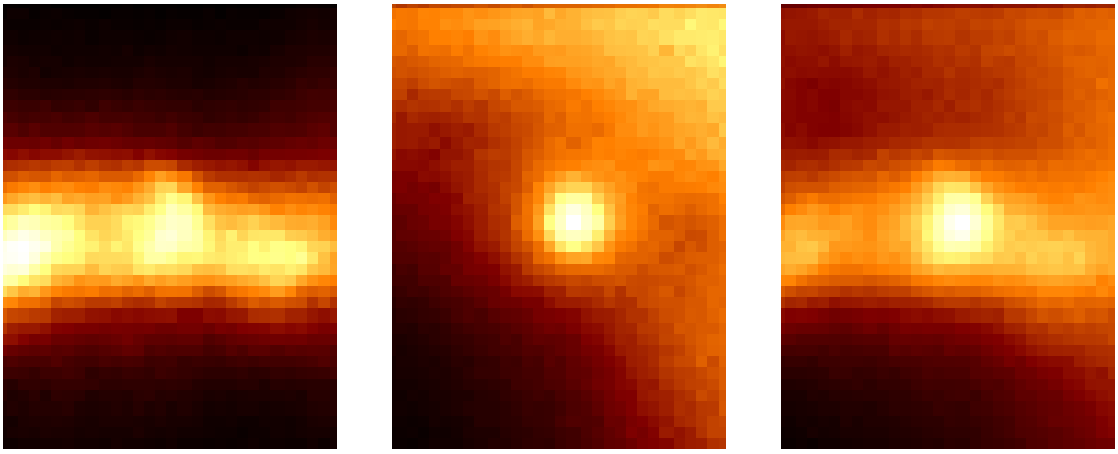


(a) Galactic center with only IEM.

(b) Galactic center with only Fermi Bubbles.

(c) Galactic center with both IEM and Fermi Bubbles.

Figure 4: Examples of the simulations made without a dark matter source.



(a) Galactic center with only IEM.

(b) Galactic center with only Fermi Bubbles.

(c) Galactic center with both IEM and Fermi Bubbles.

Figure 5: Examples of the simulations made containing a dark matter source.

It took around 20 hours to create the images of pixel size 1500×1364. However, these images are just plots generated through gammapy by using matplotlib. In order to avoid having the network learn parts of the image that are not relevant we crop out the title, axis and scale bar. We are left with an image of size 920×920 pixels, and each pixel contain 3 RGB values. When the image is loaded into the model during training it will then be represented by a 3-dimensional array with 2,539,200 elements. We want these values to be normalized between 0 and 1, meaning each element will be a float32 or float64 value. If we keep it to float32, where each array element takes up 4 bytes of memory, this ends up being around 10 MB. The laptop training this model has a total memory of 16 GB RAM, so it should be able to handle a lot of images loaded as arrays at the same time. However, a model taking an input of this size will require a lot of parameters. All of these will be loaded at once, which require a lot of additional memory.
Waiting multiple hours to see what effect tuning some hyperparameter has did not seem like a good way to spend time this early in the project with what is basically a proof of concept. Another training set was therefore created, with a smaller image size of 90×120 pixels. An alternative to this approach could be to use more powerful hardware to increase the available memory and therefore the batch size to a more reasonable level. Another approach could be to increase the number of convolutional layers in the model, which will decrease the total number of parameters in the model and thereby free up enough memory to increase the batch size. Some attempt was put into exploring this approach but was only met by memory issues.

3.3 Neural network setup

In table 1 we see the structure of first model trained on the set of large images. Our code uses a tensorflow function to split the images into a training set which consists of 90% of the images and a validation set which consists of 10% of the images. The images are loaded from two folders, one for dark matter and one without, which is how the code creates the labels. Initially there was an idea of saving the images in array form instead of a raw image format, but given the size of the images this quickly took up most of the storage available on the machine.
Initially we used a total of 40 epochs for the model, but due to the time it took to train the network we reduced it down to 20. We use the Adam optimizer with a learning rate of $\epsilon = 0.0005$. There was an attempt at using a dynamic learning rate ranging from $\epsilon = 0.1$ to $\epsilon = 0.0001$, but the model performed worse with this than with the constant learning rate. The loss function used for the model is the cross-entropic loss function 1. The dense layers use the RelU activation function, which sets a node to 0 if the sum of weights are lower than 0. In the final layer we use a sigmoid activation function, which ensures that the output is a value between 0 and 1. The batch size was set to 4, as any larger value gave us memory issues.

The structure of the neural network for the smaller image set can be seen in table 2. We introduce a layer adding Gaussian noise to the input, using tensorflows Gaussian layer with a standard deviation of 0.8 and seed = 5. As the images contain far less pixels than originally planned there is a risk of there being a low variation between the images, so adding noise to the training set should help prevent the model from overfitting. Initially we used a learning rate of $\epsilon = 0.005$ and a batch size of 100, but we got better results by turning the learning rate to $\epsilon = 0.0005$ and batch size to 500. We use a total of 100 epochs and the rest of the hyperparameters remain the same as for the first model. The number of epochs can be reduced by having the model stop training after certain conditions are met, for example if there is no improvement to the loss function after 10 epochs. However, as the training time is rather short for this model it was decided to control the number of epochs manually.

Layer	Output Shape	Parameters
Conv2D	(None, 918, 918, 32)	896
Batch Normalization	(None, 918, 918, 32)	128
MaxPooling2D	(None, 459, 459, 32)	0
Conv2D	(None, 457, 457, 64)	18496
Dense	(None, 457, 457, 64)	4160
MaxPooling2D	(None, 228, 228, 64)	0
Conv2D	(None, 226, 226, 64)	36928
Dense	(None, 226, 226, 64)	4160
Flatten	(None, 3268864)	0
Dense	(None, 64)	209207360
Dense	(None, 10)	650
Dense	(None, 1)	11

Table 1: The model structure for the first neural network, trained on the large images. The model has a total number of 209,272,789 parameters.

Layer	Output Shape	Parameters
Rescaling	(None, 90, 120, 3)	0
GaussianNoise	(None, 90, 120, 3)	0
Conv2D	(None, 88, 118, 32)	896
MaxPooling2D	(None, 44, 59, 32)	0
Dense	(None, 44, 59, 32)	1056
Conv2D	(None, 42, 57, 32)	9248
Dense	(None, 42, 57, 32)	1056
MaxPooling2D	(None, 21, 28, 32)	0
Conv2D	(None, 19, 26, 32)	9248
Flatten	(None, 15808)	0
Dense	(None, 16)	252944
Dense	(None, 1)	17

Table 2: The model structure for the neural network trained on the smaller images. The model has a total number of 274,465 parameters.

For both models we measure the accuracy, loss and the AUC of the network.
The AUC is the area under the ROC curve, which is the curve for a plot of the true positive rate (TPR) against the false positive rate (FPR) [9]. The TPR, also known as sensitivity, is the ratio of predictions the network gets correct out of all the simulations with dark matter in them and is defined as

$$TPR = \frac{TP}{TP + FN} \quad ,$$

while the FPR is the ratio of predictions the network gets wrong out of all the simulations without dark matter in them and is defined as

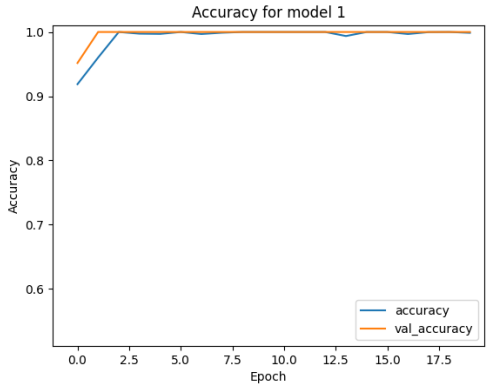
$$FPR = \frac{FP}{FP + TN} \quad .$$

Measuring the AUC allows us to see if the network does a good job at seperating the images with dark matter from those without.

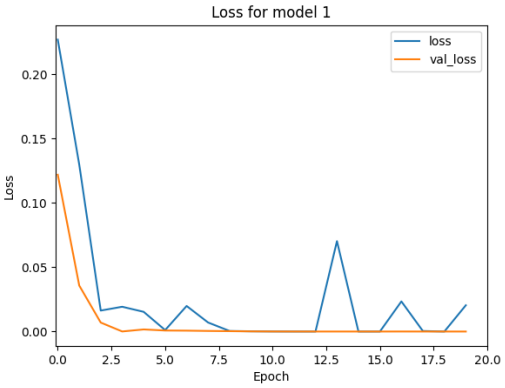
4 Results and discussion

The results for the first model trained on the set of large images can be seen in figure 6. We see that the model quickly learns how to differentiate between images containing dark matter and those without. We did not have to spend much time tuning hyperparameters to get this result. We only reduced the epochs from 40 to 20 and tried to implement a dynamic learning rate, but apart from that our initial choice of hyperparameters proved the best. The network uses 7 minutes per epoch, which makes the entire training time 140 minutes for 20 epochs.
The accuracy in figure 6a is more or less 100% after a couple of epochs, the loss in figure 6b is close to 0 and the AUC in figure 6c is 1 at almost every step during training. There is no underfitting by the model with such a high accuracy. Based on what the validation data tells us the model does not overfit either. However, the validation and training data are taken from the same set of images. If the variation between the simulation is insignificant the images are too similar, which would explain how we would get such great results after only a couple of epochs. We relied on the gammapy simulation to create variation between the images, but it is likely that the variation is not significant enough for the simplicity of the task presented to the network. After all, we are learning the network to tell images with a sphere in the middle from images without.

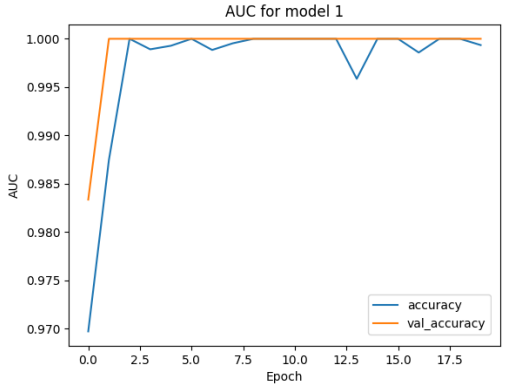
The results for the second model can be seen in figure 7. As we see for the accuracy and loss in figures 7a and 7b the model ends up overfitting in the later epochs of the training, but around epoch 40 to epoch 60 the model seems to perform very well. What happens after is the model keeps training on the training set and is memorizing rather than generalizing , and it becomes worse at classifying the validation set. This shows why it is a good idea to implement code that stops the network from training after the performance stops improving for some number of epochs. If we were going to use our network on an independent test set we would rather stop it at 50 epochs than 100. Our model spends 4 seconds per epoch, which means the total training time is a little less than 7 minutes for 100 epochs.
In figure 7c we see the AUC for the second model. It is interesting to note that for the validation set it remains 1 during most of the training even if the accuracy and loss of the validation set fluctuates. The AUC should tell us how well the network predicts which class a feature belongs to. Somehow it is excellent at predicting this even when the accuracy for the validation set drops. This could be due to a bug somewhere in the code.
Another interesting part of the plots is that the scores for the validation set is better than for the training set at certain times. This could be due to the fact that we are adding noise to the training set, but the validation set remains untouched. At the point where the network is generalizing properly instead of memorizing the training set it will classify the simpler images in the validation set more accurately than it classifies the images with added noise in the training set.



(a) Training and validation accuracy for model 1.

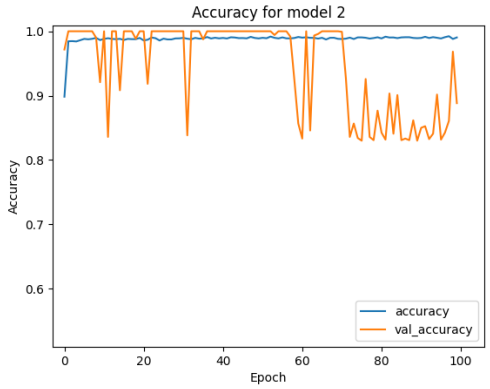


(b) Training and validation loss for model 1.

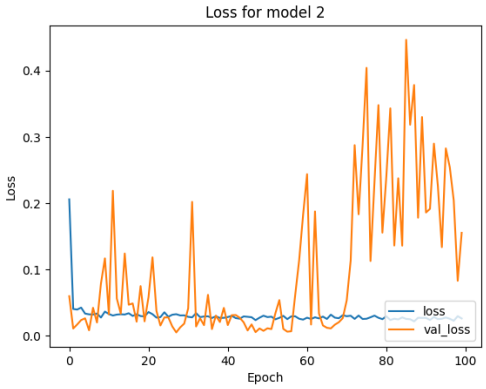


(c) Training and validation AUC for model 1.

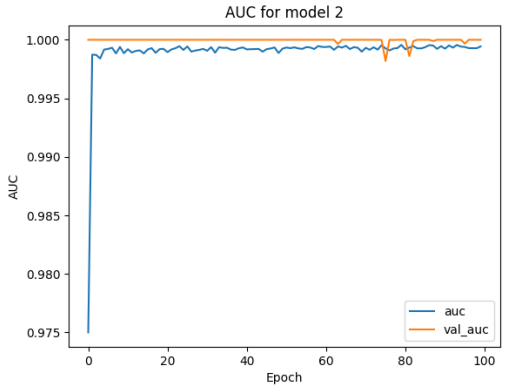
Figure 6: Performance of the first model trained on the large set of images. We use a learning rate $\epsilon = 0.005$, batch size $= 4$ and 20 epochs.



(a) Training and validation accuracy for model 2.



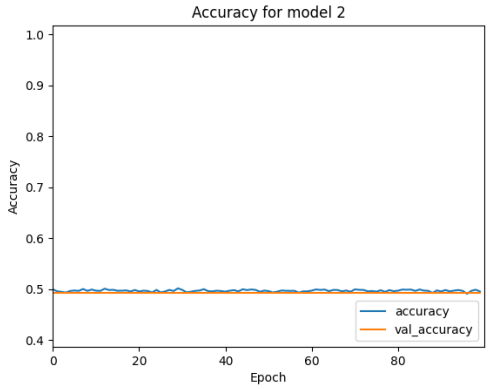
(b) Training and validation loss for model 2.



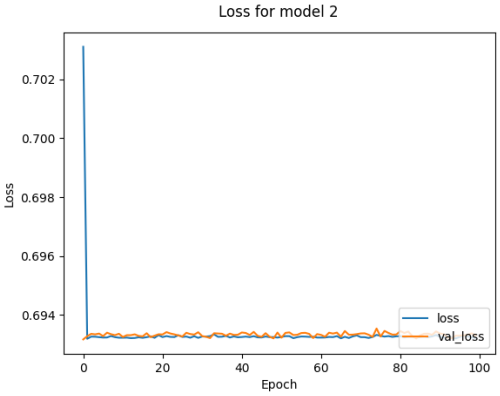
(c) Training and validation AUC for model 2.

Figure 7: Performance of the first model trained on the large set of images. We use a learning rate $\epsilon = 0.0005$, batch size = 100 and 100 epochs.

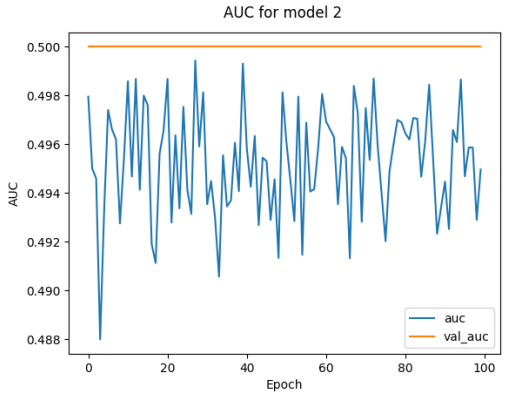
In figure 8 we see the performance of model 2 after we turn the learning rate from $\epsilon = 0.0005$ up to $\epsilon = 0.005$. When the learning rate is too large the network fails at approaching a minimum for the loss function, and no matter how many epochs it trains for it does not manage to move further down the loss curve. After having tried out different choices of hyperparameters for the model it is clear that the one with the single largest impact is the learning rate. A suboptimal choice of batch size, number of layers or nodes in the layers can make the difference between an accuracy of 95% and 65%, but choosing a learning rate that is too large makes the network unable to learn anything.



(a) Training and validation accuracy for model 2.



(b) Training and validation loss for model 2.

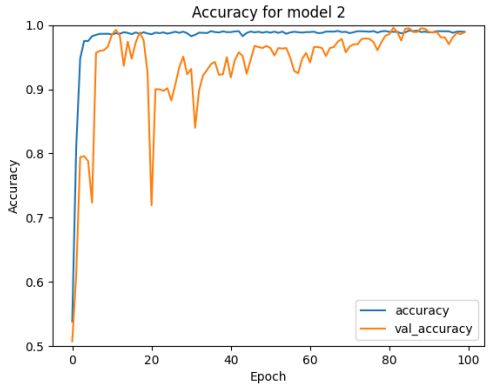


(c) Training and validation AUC for model 2.

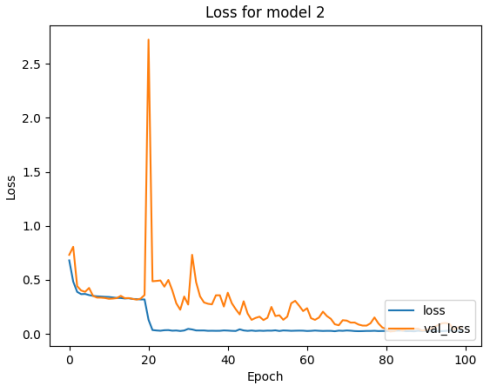
Figure 8: Performance of the first model trained on the large set of images. We use a learning rate $\epsilon = 0.005$, batch size $= 100$ and 100 epochs.

In figure 9 we see the performance of the second model after we change the batch size from 100 to 500, with the learning rate set at $\epsilon = 0.0005$. We see that the model spends longer to achieve a good, stable accuracy for the validation set when compared to the performance in figure 7. However, the performance is a lot more stable at later epochs and there is no sign of overfitting. All in all it seems to be a better choice to increase the batch size to 500 and to keep it at 100, and 100 epochs seems to be sufficient for the model. Also here we see the AUC score in figure 9c behaving unexpectedly from the accuracy in figure 9a.
A stable accuracy of almost 100% and an AUC of 1 for both the test set and validation set is as good as it can get for our model, so this felt like a natural place to conclude the project. Our goal has been achieved beyond the target we initially set, but then again this task should not be complicated for a CNN.

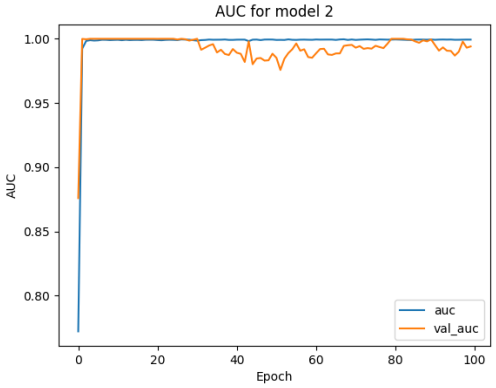
We need a test set more sophisticated than what we have currently produced in order to conclude whether the network can spot a sphere given a large variation in the background. It is very likely that we have not added enough variation in our image sets, especially for the validation set. Another problem with our approach is that it is not based on realistic observations of dark



(a) Training and validation accuracy for model 2.



(b) Training and validation loss for model 2.



(c) Training and validation AUC for model 2.

Figure 9: Performance of the first model trained on the large set of images. We use a learning rate $\epsilon = 0.0005$, batch size $= 500$ and 100 epochs.

matter signals. The next step would therefore be to use analytical methods used by the CTA and other observatory collaborations to create realistic simulations of observations of the galactic center.
Another problem with our approach is that what we are teaching the network is simply recognizing a geometric shape, which for now has been localized in the same point in every image. There are many gamma ray sources in the galaxy which share the spherical shape of our dark matter signal, so further information is needed for a network to be able to differentiate between dark matter and everything else.
It is unlikely that the network produced in this project will have the potential to find a dark matter signal in its current shape. Still, it is a beginning, and there are multiple possibilities for where to go from here. The simplest way ahead would be to look beyond the galactic center, which is densely populated with gamma ray sources of all energies. We could use the model to scan the rest of the sky for faint gamma ray sources with a spherical geometry. There is no guarantee that a signal spotted by our approach will necessarily be a dark matter signal, but it would indicate regions of interest in the vast area of the night sky.
We can also expand on the network's ability to recognize spheres and teach it to recognize the shapes of other interstellar objects, creating a classification tool for different gamma ray sources in the galaxy center. In the case where we want to find new structures this could be an interesting approach. It would also be interesting to see if an unsupervised network could do this better when we include new geometric structures not previously documented in the center.
Another possibility is to teach a network how to recognize dark matter signals based on other parameters than merely the geometrical shape of the signal, and use our current network's ability to identify the spherical shape as one part of this. These other parameters are more significant when we try to determine the nature of an observed gamma ray source. We could for example create a network that can tell the difference between different dark matter profiles, or one that can recognize different decay channels.

5 Conclusion

In this project we have shown that we can train a neural network to spot the geometric shape of a dark matter signal. We exceeded our goal of a 70% accuracy and an AUC of 0.5 by far, reaching 100% accuracy for our validation set in our first model and 99% for our second model. We also achieve an AUC that is close to 1 in both cases. We also saw how reducing the image size from 920×920 to 90×120 reduced the time per epoch from 7 minutes to 4 seconds, making it easier to optimize hyperparameters manually.
The behaviour of our first model hints at the backgrounds our simulation produce are very similar, as the network immediately manages to learn how to predict images with dark matter and without. Adding a layer of Gaussian noise to our second model proved a simple way to increase the variation of images, which we see from how it became harder for our network to immediately separate images with dark matter from those without. However, a simple validation set drawn from the same pool of images as the training set and without any noise added to it does not indicate how well our network will perform on images with new backgrounds.
Creating a test set with a lot more variation in the background will show if our network truly manages to separate images with dark matter in them from those without, regardless of the background. On its own this is not sufficient to spot a dark matter signal from an observation, especially in the galaxy center. However, it might serve as an addition to other methods or a way to indicate regions of interest in the less populated areas of the night sky, away from the galaxy center.

References

[1] *Science with the Cherenkov Telescope Array*. WORLD SCIENTIFIC, feb 2018. https://doi.org/10.1142%2F10986.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 6. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Michael Taylor. *Neural Networks - A Visual Introduction For Beginners*. Blue Windmill Media, 2017.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 5. MIT Press, 2016. http://www.deeplearningbook.org.

[5] https://www.robot-magazine.fr/machine-learning-deep-learning/.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 9. MIT Press, 2016. http://www.deeplearningbook.org.

[7] https://medium.com/@thepyprogrammer/2d-image-convolution-with-numpy-with-a-handmade-sliding-w

[8] https://analyticsindiamag.com/comprehensive-guide-to-different-pooling-layers-in-deep-learnin

[9] Khashayar Namdar, Masoom A. Haider, and Farzad Khalvati. A modified auc for training convolutional neural networks: Taking confidence into account. *Frontiers in Artificial Intelligence*, 4, 2021.