NetFilEx: A Neural Network Filament Extractor

Sharif Khan-Bennett

Student ID: 1799796
Supervisor: Dr. Ian Stevens

School of Physics and Astronomy
University of Birmingham
United Kingdom

May 14, 2021

Word Count: 7175 (LaTeX)

Abstract

We present a neural network-based object extraction algorithm. NetFilEx cuts FITS files into (128×128) pix sized frames and searches each one for Non-Thermal Filaments. The algorithm outputs mosaics of the NTFs, stored in an HDUList. The Convolutional Neural Network has been trained on the MeerKAT survey of the Galactic Centre and has achieved a validation accuracy of 99.61%. These mosaics can be used for subsequent analysis by packages such as FilFinder, as demonstrated in this paper. Extraction algorithms like these will become increasingly necessary with the next generation of radio telescopes. Traditional object classification methods require a significant degree of human interaction, which will not be feasible as the amount of data from radio surveys rapidly grows. The automation facilitated by convolutional neural networks will shorten the data pipeline between telescopes and researchers.

NetFilEx: A Neural Network Filament Extractor

Sharif Khan-Bennett[1]★
[1]*School of Physics and Astronomy, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK*
Student ID: 1799796
Supervisor: Dr. Ian Stevens
Word Count: 7175 (LaTeX)

14 May 2021

ABSTRACT

We present a neural network-based object extraction algorithm. NetFilEx cuts FITS files into (128 × 128) pix sized frames and searches each one for Non-Thermal Filaments (NTFs). The algorithm outputs mosaics of the NTFs, stored in an HDUList. The Convolutional Neural Network (CNN) has been trained on the MeerKAT survey of the Galactic Centre and has achieved a validation accuracy of 99.61%. These mosaics can be used for subsequent analysis by packages such as FilFinder, as demonstrated in this paper. Extraction algorithms like these will become increasingly necessary with the next generation of radio telescopes. Traditional object classification methods require a significant degree of human interaction, which will not be feasible as the amount of data from radio surveys rapidly grows. The automation facilitated by CNN will shorten the data pipeline between telescopes and researchers.

1 INTRODUCTION

The development of the Square Kilometre Array (SKA) indicates a new generation of radio astronomy. Upcoming surveys will provide us with more precise and deeper imaging of radio sources (Norris (2011); Prandoni & Seymour (2015)). For example, the SKA is expected to observe $\frac{3}{4}$ of the sky at a resolution of $0.1''$, compared to the FIRST survey, which observes $\frac{1}{4}$ of the sky at a resolution of $5''$ (Braun et al. (2015); Taylor (2012)). Traditionally, objects in these surveys would be classified by eye, as in the Galaxy Zoo citizen project (Lukic et al. (2018)). However, soon there will be so much data that this approach will become unfeasible. This project aims to develop and test an automated object detection and extraction algorithm using a CNN.

These networks have become increasingly popular in the past few decades, spurred by increased computing capabilities (specifically the development of the GPU) and greater storage and access to data. Although GPUs were initially developed for computer gaming, their paradigm is optimised for the type of operations used in CNN's, namely, matrix multiplication. As well as a general uptake in the use of machine learning algorithms, CNN's have found a niche in astronomy. They have been widely used to classify various astrophysical objects, including radio galaxies (Lukic & Brüggen (2017); Aniyan & Thorat (2017); Lukic et al. (2018); Alhassan et al. (2018)) and supernova remnants (Cabrera-Vives et al. (2016); Liu et al. (2019)), in addition to being used in the search for exoplanets (Pearson et al. (2017)), and even extraterrestrial life (Harp et al. (2019)).

Here, a CNN will be developed using TensorFlow (Abadi et al. (2016)) to detect NTFs in the galactic centre (GC). NTFs are radio sources that have been used to study the magnetic field of the GC in great detail, but there is currently no agreed-upon mechanism to explain how they are formed. With the development of next-generation radio telescopes, NTFs will continue to be discovered in the GC and potentially in external galaxies in the future. This algorithm will be able to quickly search through upcoming surveys and extract NTFs with minimal human input. The algorithm outputs an HDUList of mosaics containing NTFs, which can be used for subsequent analysis.

The structure of this paper is as follows. Section 2 outlines some previous research into NTFs. Section 3 explains the theory behind artificial neural networks, convolutional neural networks and generative adversarial networks. Section 4 outlines the method taken to build, train and implement the CNN into the extraction algorithm. Section 5 presents the performance of the CNN and includes an analysis of some of the extracted NTFs. Section 6 discusses improvements and future work.

2 NON-THERMAL FILAMENTS

NTFs are radio-sources which were first observed in the GC about 40 years ago (Yusef-Zadeh et al. (1984)). They have been found to occupy the central 200pc of the GC. This central 200pc region is known as the central molecular zone, a chaotic region characterised by high densities, large velocity dispersions, high temperatures, and an abundance of molecular gas (Morris & Serabyn (1996); Heywood et al. (2019)). NTFs found here tend to be aligned perpendicularly to the galactic plane, while shorter NTFs are more randomly aligned (Lang et al. (1999a); LaRosa et al. (2001); Morris (2007)). They emit highly polarised light caused by synchrotron radiation. This phenomenon is where relativistic electrons continually accelerate by spiralling around magnetic field lines, causing light emission. The rates at which these electrons spiral is called the rotation measure and has found to be in the range of 100rad m^2-5500rad m^2 (Yusef-Zadeh et al. (1984); Yusef-Zadeh & Morris (1987); Gray et al. (1995);

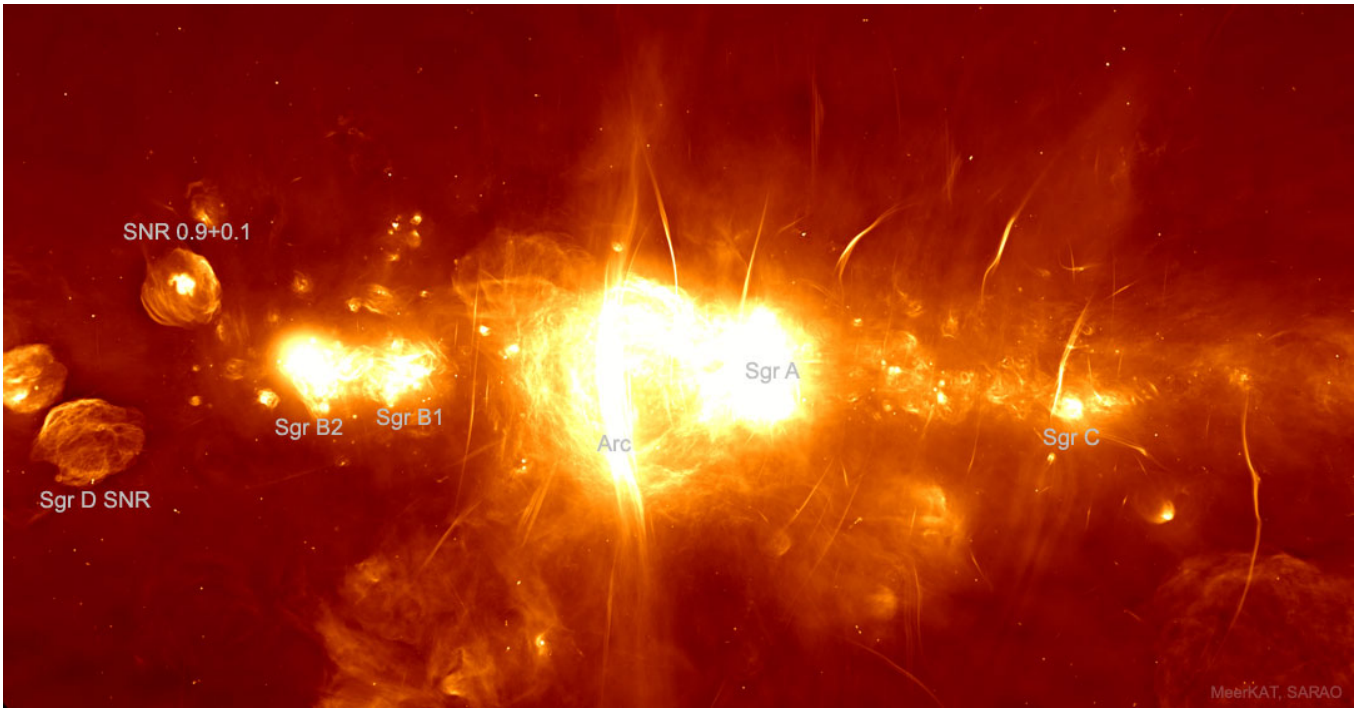

Figure 1. The inaugural image of the GC from MeerKAT. The supermassive black hole, Sgr A* is the bright object labelled at the centre of the image. Several NTFs can be seen in the image; they are the bright thread-like structures predominantly located above the galactic plane. "The Arc" has been labelled and is composed of many adjacent NTFs. Image courtesy of SARAO (SARAO)

Lang et al. (1999b); Paré et al. (2019)). NTFs are very elongated, they are up to ~ 50pc long and 0.5pc wide, (Morris & Yusef-Zadeh (1985); Gray et al. (1991, 1995); Morris (2007); Arzoumanian et al. (2011)). This results in quite a striking morphology, see Figure 1. The timescale over which the NTFs evolve is determined by the electron diffusion timescale given as $3 \times 10^5 yr$ (Bicknell & Li (2001c)).

The alignment of the NTFs and their composition have been used to support the hypothesis of a pervasive dipolar magnetic field with local disturbances in the central molecular zone. It has been shown that a magnetic field strength of $10\mu G - 1mG$ must exist in the GC to provide the NTFs with enough rigidity to withstand the ram pressure of the galactic wind and maintain their linear morphology (Yusef-Zadeh & Morris (1987); Gray et al. (1995); Bicknell & Li (2001c); Yusef-Zadeh et al. (2005); LaRosa et al. (2005); Morris (2007)).

While their morphology with respect to the magnetic field is understood, the origins of the spiralling electrons are less so. Some early formation mechanisms involved the existence of shock fronts (Yusef-Zadeh et al. (1984); Yusef-Zadeh (2003)), cosmic strings (Bicknell & Li (2001a)), and star wakes (Gray et al. (1991); Nicholls & Strange (1995)). However, these mechanisms were not sufficient at describing all of the observed phenomena exhibited by the NTFs. More recent mechanisms include the interaction between molecular clouds and the galactic wind (Shore & LaRosa (1999)) and the tidal destruction of molecular clouds by the supermassive black hole, Saggitarius A* (Coughlin et al. (2021)).

3 DEEP LEARNING THEORY

Deep learning refers to a type of machine that learns to make decisions without being explicitly programmed how to do so. An example of this is supervised learning, where the algorithm makes predictions, p_n from data samples and compares these predictions with provided numeric labels or targets, t_n. During training, the algorithm modifies itself to minimise the difference between its predictions and the labels. The aim is that after training, the model will be able to continue making correct predictions for samples on which it has not previously trained. For data classification, the algorithm achieves this by recognising which features in the data are characteristic for each data class. This paper will focus on deep learning in the context of binary classification, therefore, $t_n = \{1, 0\}$.

3.1 Artificial Neural Networks

3.1.1 Generic Architecture

Artificial neural networks (ANNs) are a type of deep learning algorithm. They consist of a series of connected nodes or neurons and transmit signals like the neurons in the brain (Fitch (1944); Fukushima (1980, 2007)). The samples which are inputted to an ANN are drawn from some high-dimensional data distribution $\mathbf{x_n} \sim \mathbf{D}$. Each component of this vector, x_i, becomes an input to a neuron in the neural network's input layer, see fig 2.

The network performs a series of transformations on $\mathbf{x_n}$ such that at the final layer, a single number is outputted, forming the model's prediction. These transformations occur in the hidden layers of the

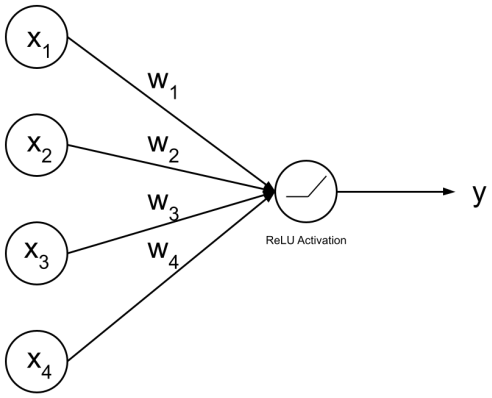

Figure 2. A schematic showing the connectivity of a single neuron in a densely connected layer.

neural network. The neurons in these layers perform transformations using parameters called the weights, w_i, and biases b. The output of a single neuron is given by:

$$y = \sum_{i=1}^{m} w_i x_i + b \tag{1}$$

where m is the total number of input connections.

These neurons are arranged into a series of interconnected hidden layers. The output of a single layer can then be given by the matrix transformation:

$$y = \mathbf{W} \cdot \mathbf{x} + b. \tag{2}$$

Generally, $\mathbf{D}$ in a non-linear distribution. For the model to accurately represent this, non-linearities need to be included in the linear transformation above. Stacking non-linear functions can be used to represent any continuous, differentiable function to a high (but not infinite) accuracy (Cybenko (1989); Chen & Chen (1996)). Activation functions transform y to provide this non-linearity. This project will use the ReLU activation function as it is efficient as well as numerically stable (Nair & Hinton (2010); Changpinyo et al. (2017)), see section 3.1.2. It is defined as $max(0, x)$ and has the effect of removing negative inputs.

All of the weights and biases are stored in a tensor $\mathbf{\Theta}$, which is initialised randomly. After the data has passed through the hidden layers, the sample needs to be classified. All of the previous transformations are combined using a densely connected layer. This is a layer that connects to all neurons in the previous layer to output a single number. This number then enters a sigmoid function which maps the output to the range (0,1), forming the class prediction, p_n. This process is called forward-pass and can be described using the function: $f(\mathbf{\Theta}) : \mathbf{x_n} \longrightarrow p_n$. The first prediction is likely to be wrong given that $\mathbf{\Theta}$ is currently random.

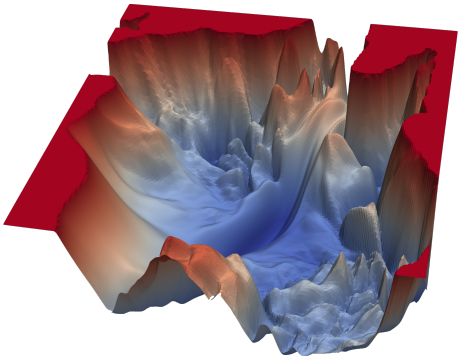

Figure 3. A 3D representation of the loss surface. This is found by marginalising the high-dimensional surface and representing it on only two, roughly orthogonal basis vectors. Image courtesy of (Li et al. (2018))

3.1.2 Loss and optimisation

During training, the data is divided into subsets known as batches and sent through the model one at a time. One epoch corresponds to one complete set of batches passing through the model. Batches are used to reduce the memory demands of the model. It also has the effect of introducing noise to the model, which acts as a regulariser and can limit overfitting (Cotter et al. (2011)), see section 3.1.3. The performance of the model is evaluated for each batch using a training loss function. Its role is to quantify the difference between p_n and t_n. For binary classification, the binary cross-entropy loss is most appropriate:

$$L = \frac{1}{N}\sum_{n=1}^{N} l_n = -\frac{1}{N}\sum_{n=1}^{N}\Big[t_n \log p_n + (1 - t_n)\log(1 - p_n)\Big] \tag{3}$$

where N is the batch size. To increase the accuracy of its predictions, the model aims to minimise the loss function. For example, minimising the loss for a single sample when $t_n = 1$ involves minimising $-\log p_n$, which is achieved by adjusting $\mathbf{\Theta}$ such that $p_n \longrightarrow 1$, and vice versa for $t_n = 0$. In a similar fashion to linear regression, this process involves finding the hyper-plane, parameterised by $\mathbf{\Theta}$ which best separates the distributions of each class, $\mathbf{D_1}$ and $\mathbf{D_0}$.

The training loss can be expressed as $L = f\big(p(\mathbf{x}, \mathbf{\Theta}) - t\big)$. After each batch has passed through the network, the loss is measured, forming a hyper-surface called the loss surface in $\mathbb{R}^T$ space, where T is the number of trainable model parameters. For perspective, this paper uses a model which employs $6,858,413$ trainable parameters! See Figure 3 for a 3D visualisation of this loss surface.

During training, the goal of the optimiser is to find the minimum of this loss surface corresponding to the model making the highest possible rate of accurate predictions. The coordinates of this minimum correspond to the optimum set of model parameters, $\mathbf{\Theta}^*$, where

$$\mathbf{\Theta}^* = \underset{\mathbf{\Theta}}{\operatorname{argmin}}\, L(p(\mathbf{x}, \mathbf{\Theta}) - t). \tag{4}$$

The parameters are updated after each batch via a process called

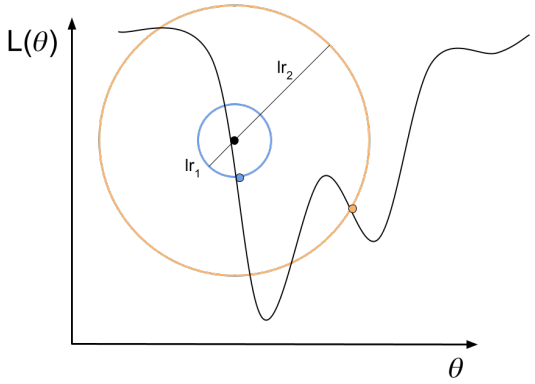

Figure 4. A schematic showing the loss surface in one dimension. The choice of learning rate heavily affects how the network optimises. The black circle represents the optimiser at time t, and the blue and orange circles represent the optimiser at time t+1 depending on the choice of learning rate. Due to the larger choice of learning rate, the yellow optimiser has overshot the global minimum in the loss surface. This can result in an unstable convergence as the loss surface is usually highly non-convex, and a large learning rate can cause the loss to diverge as the optimiser wildly jumps around. On the other hand, if the optimiser were approaching the minimum from the other side of the valley, too small of a learning rate would result in the optimiser getting stuck in local minima.

back-propagation. This updates $\mathbf{\Theta}$ according to the gradient descent equation:

$$\mathbf{\Theta}_{t+1} = \mathbf{\Theta}_t - \gamma_t \nabla L(\mathbf{\Theta}_t) \tag{5}$$

where t is the current batch number and γ_t is the learning rate which sets the step size towards the minimum (Hecht-Nielsen (1989); Nocedal & Wright (2006)), see Figure 4.

$\nabla L(\mathbf{\Theta}_t)$ is given by:

$$\nabla L(\mathbf{\Theta}_t) = \sum_{n=1}^{N} \nabla l_n(\mathbf{\Theta}_t) \tag{6}$$

where N is the number of data samples in a single batch. $\nabla L(\mathbf{\Theta}_t)$ is propagated back through the neural network layer by layer in a process called back-propagation. This relies on chain-rule differentiation with respect to each model parameter. The neurons in each layer sequentially adjust their weights and biases in the direction of smaller loss (Hecht-Nielsen (1989); Amari (1993)). The advantage of using ReLU activations is that, unlike sigmoid and tanh, it is non-saturating. Activations with saturating extremities result in the gradient vanishing and slow training when propagated through deep networks.

One of the critical parameters in deciding the stability of the model is the learning rate, γ. This needs to be selected with care. If too small of a value is chosen, the optimiser will update very slowly and likely get caught in local minima. On the other hand, if γ is set too large, the optimiser will wildly jump around the loss surface and not converge (Abadi et al. (2015)), see Figure 4. A solution to this is the use of the "ADAM" optimiser. This optimiser updates the learning rate depending on the local gradient of the loss surface. The gradient is expected to be greater when further away from the global minimum. Here, ADAM sets a relatively high learning rate to approach the minimum quickly. As the gradient falls, ADAM reduces the learning rate to ensure stable convergence (Kingma & Ba (2017)).

3.1.3 Regularization

As mentioned in section 3, the model will ideally continue making correct predictions for samples on which it has not previously trained. To assess the model's performance in this regard, the **D** set is split into two subsets. The training set is actively used to update the model's parameters via back-propagation. The validation set only passes forwards through the model and ensures that the model's predictive power can be generalised to unseen data.

A problem affecting neural networks is overfitting which is where the model starts to fit all of the noise unique to the training dataset. It is caused by the hyper-plane separating $\mathbf{D_1}$ and $\mathbf{D_0}$ becoming too jagged, resulting in unseen samples starting to fall on the wrong side of the hyper-plane. This is characterised by the training loss continuing to decrease while the validation loss beginning to diverge resulting in poor generalisability. Overfitting can be combated by applying a dropout layer which sets the output of a proportion of the neurons to zero. It forces the model to make more robust connections rather than just memorising the training set (Srivastava et al. (2014); Smirnov et al. (2014); Abadi et al. (2015)). Also, early-stopping can be applied. This is a command provided by Keras which automatically stops training when the validation loss curve begins to diverge (Yao et al. (2007)).

Additionally, batch normalisation can be applied after a layer, normalising the output for a single training batch. This reduces the effect of a phenomenon known as covariate drift, resulting in a smoother loss surface and more stable optimisation (Ioffe & Szegedy (2015)).

3.2 Convolutional Neural Networks

CNN's are set up to handle 2D data, like images. This data is arranged as a grid of numbers representing pixel intensity. Usually, images have three layered grids for colour channels, red, green and blue. The images in this project are in greyscale, and so there is only one channel. It would be possible to vectorise an image where the rows of all the pixel values are connected end-to-end and sent into the network as $\mathbf{x_n}$. However, this vector would look significantly different if an identical object appeared at a different position in the image. This is very inefficient as the vectors for identical classes would look completely different; it would take an extremely long time to learn the representations of different objects. So for a network dealing with images, there must be a way to implement locality and spatial invariance in the layers. These conditions can be satisfied with the use of the convolution operation.

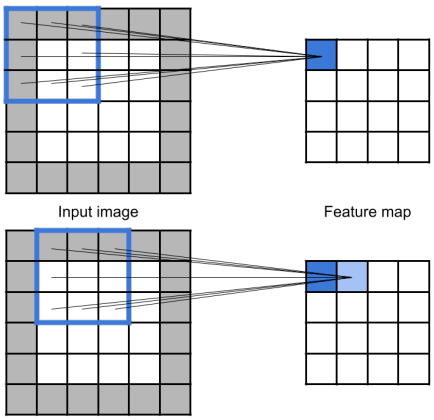

Figure 5. A schematic showing the "same" convolution of a single kernel with a (3×3) receptive field and with a stride of 2. The border of grey cells are zeros which have been placed around the input image to ensure that the feature map is not downsampled.

3.2.1 Convolutional Layers

CNN's use many "kernels" or "filters" instead of neurons to analyse an image, $\mathbf{x_n}$. These are small 2D grids of numbers that multiply with the pixel values in a defined receptive field of the input image to give a single pixel value output. If a kernel corresponding to a particular feature is multiplied by a receptive field containing that same feature, a bright output pixel is produced. This allows the filters to identify local features in the image. The receptive field is then shifted across the entire image one stride at a time, producing corresponding pixels on an output image known as a feature map, see Figure 5. The feature map becomes the input to the following layer.

The CNN is similar to the previously described ANN where the neurons are replaced by many kernels, the weights w are the values in each kernel, and the data within the receptive field can be thought of as x_i. Since there are many kernels to detect different features per layer, each producing its own feature map, the output from a layer is a volume of data as deep as the number of kernels (O'Shea & Nash (2015); Dumoulin & Visin (2016)). Layers of kernels will begin to detect higher-order features in the data. For example, the first layer would become sensitive to edges and corners, whereas later layers would become sensitive to entire shapes.

There are two ways in which the convolution operation can be applied. "Valid" convolution ensures that the receptive field remains in the borders of the input image. Since the receptive field is bigger than one pixel, this results in the feature map being downsampled. Alternatively, "Same" convolution can be applied where the input image is padded with a border of zeros to ensure the feature map is not downsampled (Dumoulin & Visin (2016)). This project used "Same" convolutions with a 3×3 kernel size.

3.2.2 Pooling Layers

Pooling is used to reduce the resolution of the feature maps by reducing the number of pixels in the feature map. This is useful to apply after the convolutional layer as it improves computation time by removing useless pixels. There are two commonly used pooling techniques. Max pooling outputs only the largest pixel value in a receptive field, whereas average pooling outputs the averaged pixel values in a receptive field (Dumoulin & Visin (2016); Alhassan et al. (2018)). This project used a combination of Max and Average pooling, both with a 2×2 receptive field.

The previous layers can be applied in a CNN in a modular way. Generally, the number of layers should scale with the complexity of the data set. Once the data has passed through all these layers, classification occurs, and performance is quantified, triggering training similarly as with the ANN.

3.3 Generative Adversarial Networks

For a neural network to generalise to unseen data, a vast number of samples are required for the network to accurately represent $\mathbf{D_1}$ and $\mathbf{D_0}$. Often, especially in the case of astronomy, a wealth of data may not be available. This requires data augmentation and supplementation to be used. A supplementation technique is the use of generative adversarial networks (GANs).

GANs are an example of unsupervised learning since the input data is not labelled, but all of the same class, say class 1. They work by learning the underlying data distribution, $\mathbf{D_1}$ from a set of samples. Then, an arbitrary number of new, hyper-realistic samples can be drawn.

GANs optimise through a two-player game. One player is the generator; the other is the discriminator. For GANs that deal with images, each component has an architecture similar to a CNN; consequently, this class of GANs are often referred to as a deep convolutional generative adversarial network, DCGAN (Radford et al. (2015); Fang et al. (2018)). The role of the generator is to upsample a Gaussian noise vector into a realistic data sample. The role of the discriminator is to identify whether the image it has received has been generated or is real. The generator learns only from the feedback from the discriminator. The discriminator learns from the accuracy of its predictions. This training process is a "zero-sum game" because the generator is trying to minimise the same function that the discriminator is trying to maximise. It is ideal for both components to train at a relatively similar rate. If one of the models were to optimise too quickly, it would overpower the other model, preventing it from learning how to improve. If they optimise at roughly equal rates, the discriminator gets better at identifying the images while the generator simultaneously gets better at generating realistic samples. The system optimises when the generator can fool the discriminator 50% of the time (Goodfellow et al. (2014); Abadi et al. (2015); Radford et al. (2015); Fang et al. (2018)). See Figure 6 for a schematic of this model. The architecture of the discriminator and generator used in this paper can be found in tables A1 and A2.

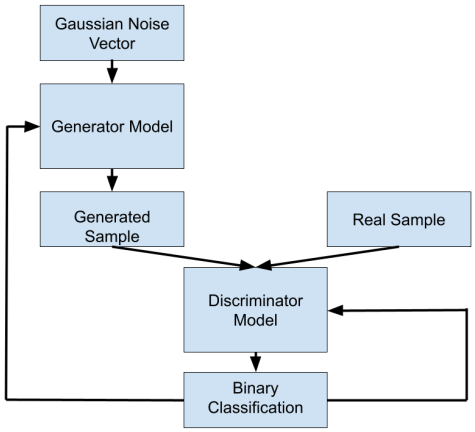

Figure 6. A schematic showing the generic architecture of a GAN.

4 METHOD

4.1 Data Preparation and Augmentation

4.1.1 Cutting

The original science frame of the GC has dimensions 11035×8731 pix. This must be cut into smaller samples to input to the CNN. Typically, the input data size should be recursively divisible by 2 to improve training stability (O'Shea & Nash (2015)). CNN's are very resource heavy and so smaller samples are desirable. However, the samples should not be so small that physical features are lost. It was found that (128×128) pix sized samples were most suitable; this corresponds to $\sim 74''$ squares. At sizes smaller than this, some NTFs began to occupy the entire width of the sample, which would interfere with the amount of information that the CNN could learn about the NTFs. The science frame was cut twice into samples of this size, where the second set of samples were offset from the first set in the horizontal and vertical directions. This was to increase the distribution of edge-interactions between the NTF and the edge of the frame, resulting in 9520 samples.

4.1.2 Pre-processing

The background noise in all the samples was removed to increase the efficiency of the model. This allows the CNN to focus on the essential features in the image. Usually, in samples that contained NTFs, the brightest object was the NTF. This allowed a threshold brightness to be set. It was found that most NTFs were at least 1σ brighter than their background. Pixels with intensities smaller than this were set to equal the threshold. Then, the pixel values of each sample were normalised to the range [-1, 1]. This allows the CNN to make identifications only based on object morphology as opposed to source brightness. It also ensures that the inputs have a similar mean which improves the model's ability to converge. See Figure 7.

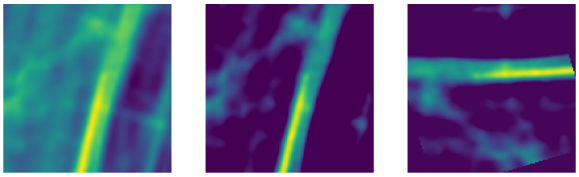

Figure 7. The steps involved in augmenting each sample. The first image is a raw sample. The middle image shows the same sample after being sigma clipped and normalised. The final image shows an example of one of the final output samples after having undergone a random rotation.

4.1.3 Sorting into classes

After all the images have undergone the pre-processing, they were sorted into their two classes; NTF: 1 & Not NTF: 0. This was initially done using the "FilFinder" package (Koch & Rosolowsky (2015)) FilFinder identifies filamentary structure in images using a host of pruning parameters as described in section 5.2. A subset of handpicked NTF samples were used to optimise these parameters by maximising the rate of NTF detections in this subset. These optimal parameters were used to separate the samples into classes 0 and 1. Then, all the samples were checked by eye to ensure that they had been sorted correctly. This resulted in 515 samples in $\mathbf{D_1}$, and 9005 samples in $\mathbf{D_0}$.

77 samples were removed from $\mathbf{D_1}$ as they were deemed too poor in quality: NTFs too dim compared to the background or only partially visible were removed. Partially visible NTFs were removed because they were similar in appearance to objects in the science frame which were not NTFs. They were removed in the interest of reducing the rate of false-positive detections. A potential problem of removing these samples is that the model cannot learn the sample space corresponding to partially visible NTFs. This can be tackled by increasing the distribution of edge interactions described above and introducing padding as described in section 4.3. Another 116 samples were separated from $\mathbf{D_1}$ as they needed to undergo a different augmentation process from all the other samples. While the majority of $\mathbf{D_1}$ samples underwent random rotations as a part of data augmentation, see section 4.1.4. These separated samples had NTFs that appeared very close to the corner of the frame. Objects too close to the corners may not appear in the final frame after being rotated; see Figure 8. This would have resulted in a large proportion of samples being labelled with a 1, but with no NTF, or only a very small portion of an NTF present. Finally, 1785 samples were removed from $\mathbf{D_0}$ to have a more balanced number of samples per class, see section 4.1.4. These samples were removed from the borders of the science frame. Samples appearing here looked very similar to one another since the density of objects drops further away from the galactic plane. Therefore, these samples contributed very little diversity to $\mathbf{D_0}$. It was found that reducing the sample space of class 0 had a minimal effect on validation accuracy. This resulted in a total of 438 samples in $\mathbf{D_1}$, and 7220 samples in $\mathbf{D_0}$.

4.1.4 Augmentation and Supplementation

This algorithm aims to identify NTFs in the central region of the milky way and other galaxies. External galaxies are randomly ori-

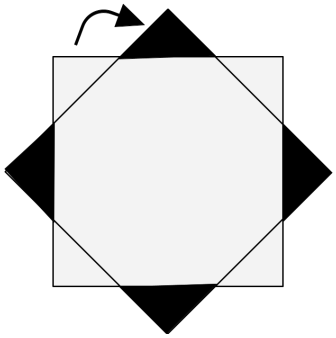

Figure 8. A schematic showing how the dark corners of the samples can be lost after being rotated. The empty corners are filled with equal pixel values, given by the median pixel value in the sample post normalisation.

Class	Training set	Validation set
NTFs	19209	8233
Not-NTFs	19289	8267

Table 1. Table showing the number of samples of each class found in the training and validation sets. There is a small imbalance here due to the random separation of the samples. However, the final data set was very large such that this variance is relatively small.

ented with respect to Earth. Therefore, for the network to generalise to these NTFs, it should learn that NTFs can exist at any observing angle. So, all the samples were duplicated with random planar rotations. Each rotated sample was also reflected about the vertical axis to increase the sample space further; see Figure 7.

The black corners in Figure 8 are portions of the unrotated image which would not have been present after rotation. The displayed case is the most extreme, but there will always be a portion of information lost for any rotation, not a multiple of 90°. This is why samples with NTFs appearing in the corners underwent a different augmentation process which involved only vertical and horizontal flips. Samples undergoing the standard augmentation process gained four regions of created space shown by unfilled corners in Figure 8. These corners were filled in with the median pixel value of the sample to try and reduce the appearance of a hard border in the final sample. It was ensured that samples both in $\mathbf{D_1}$ and $\mathbf{D_0}$ underwent rotation so that the CNN would not associate this border with only one of the classes.

To ensure that the accuracy metrics found during training represent the model's accuracy with respect to both data classes, the two classes must appear in the data set with a roughly equal frequency. As mentioned in section 4.1.3, class 0 appears $\sim$ 16 times more frequently than class 1. First, many of the samples from $\mathbf{D_0}$ were removed. Then, an imbalanced number of rotations was applied to samples in each dataset to re-balance the frequency of samples. These samples were labelled with a 0 and a 1 accordingly and split into training and validation datasets with a ratio of 70 : 30, see table 1.

To test the effectiveness of including generated NTFs in the training set, 1500 synthetic NTFs were generated. These samples also underwent rotations producing 12000 samples, 9644 of which were used in the training portion of $\mathbf{D_1}$ creating a 50 : 50 split between real samples and generated samples. These generated samples were only introduced to the training dataset to measure its effect on the model's ability to classify unseen NTFs in the validation set. Two models were trained, one with the real dataset and one with the dataset which had been supplemented with generated NTFs. Data were entered into the models in batch sizes of 32, resulting in 1205 batches and training iterations per epoch.

4.1.5 Post-training adjustments

After the first set of models had been trained, it was found that there was a high rate of false-positive detections. This was thought to be due to the abundance of objects such as supernova remnants and molecular gas clouds in the science frame. These objects looked very similar to NTFs when cut into (128 × 128) pix sized samples. The trained model was used to re-categorise the samples into $\mathbf{D_1}$ and $\mathbf{D_0}$. Then, the false positives were identified and used to overpopulate $\mathbf{D_0}$ by a factor of 10 to improve this. The new dataset was used to retrain the GAN and the CNNs to great effect.

4.2 CNN Architecture

The Keras tuner function was used to build the CNN model which produced the greatest validation accuracy. This tuner works by partially training models with various initialised hyper-parameters and selecting only the best candidates to train further. The worst performing candidates are continually dropped until one remains.

The hyper-parameters chosen to optimise, and the range of values to optimise over included: number of "convolution blocks", between 3 and 7 where, each convolutional block comprised of two convolution, batch normalisation, ReLU activation layers and a pooling layer. Also optimised were the number of filters in each convolution block between 32 and 512, the type of pooling per block, between max and average pooling, the width of the densely connected layer between 30 and 512, the dropout rate between 0 and 0.5 and finally, the maximum learning rate between 1×10^{-4} and 1×10^{-2} on a logarithmic scale. The optimal values are listed in table 2.

See Figure 9 for a schematic of this model.

4.3 Mosaic Creation

Once the CNN was trained, it was implemented into the NetFilEx algorithm. The algorithm takes a ".FITS" image, a pre-trained model, desired sample dimensions and a padding option as is arguments. First, it cuts the provided image into samples of predefined size. Each sample is then processed identically to the pre-processing used on the samples when training the model, as described in section 4.1.2. The CNN is then used to produce class predictions which were rounded to 1's and 0's and placed in a 2D array at positions corresponding to where relevant samples are found in the science frame. Then, neighbouring 1's were grouped, and their array indices stored. Here, "neighbouring" refers to the eight entries surrounding each array position. If the number of elements in one of these groups is less than two, this detection is ignored since NTFs almost always extend across multiple samples, so an isolated detection has a higher

Hyper-parameter	Optimal value
Convolutional Blocks	6
Kernels_1	128
Pooling_1	Max
Kernels_2	416
Pooling_2	Max
Kernels_3	64
Pooling_3	Average
Kernels_4	448
Pooling_4	Max
Kernels_5	320
Pooling_5	Max
Kernels_6	32
Pooling_6	Average
Dense	310
Dropout rate	0.4
Maximum learning rate	0.0039985

Table 2. Table showing the optimal values for the number of convolutional blocks, kernels, pooling type, dense connections, dropout rate and maximum learning rate with respect to **D** (real samples only).

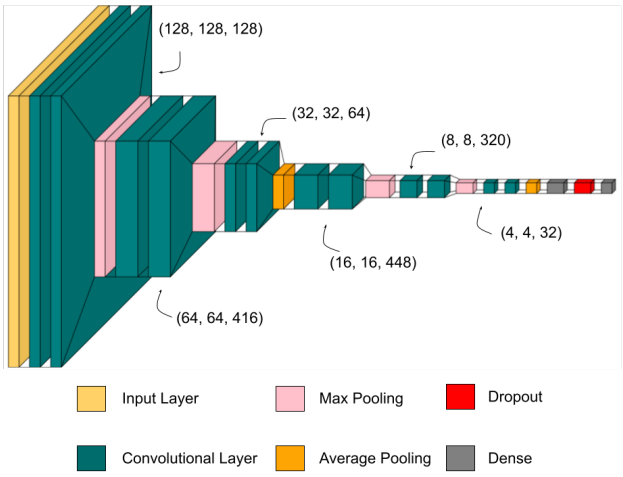

Figure 9. A schematic showing the final model architecture. Note that batch normalisation and ReLU activation layers have been omitted but appear after each convolutional layer. Also a global average pooling layer before the first dense layer has been omitted.

Figure 10. Model accuracy and loss when training on real samples.

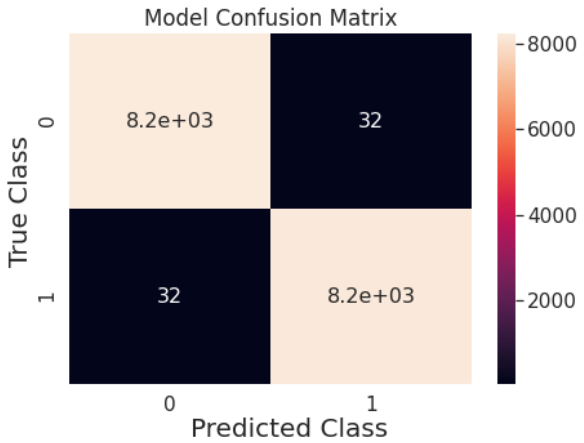

Figure 11. Confusion matrix for the model's best epoch when training on real samples.

probability of being a false positive. The remaining indices were used as coordinates to locate the frames corresponding to NTF detections. Each of these samples was appended to an HDUList.

Additionally, if padding is "True", the surrounding eight samples are also included in the HDUList regardless of whether an NTF had been detected there. This was the second measure employed to combat the lack of partially visible NTF representation in **D**. The "reproject" package is then used to combine all the samples and coordinate information in the HDUList into a single mosaic.

This method means that NTFs that meet in a single sample are included in the same mosaic. When the algorithm searches the image of the GC, it finds 40 NTF regions (many of which contain multiple NTFs). See Figures B1 and B2.

Also, note that although this algorithm was initially trained to extract NTFs, with some minor tweaks to the sample pre-processing, it can be used to extract any object in a ".FITS" image if supplied with a pre-trained binary classification model.

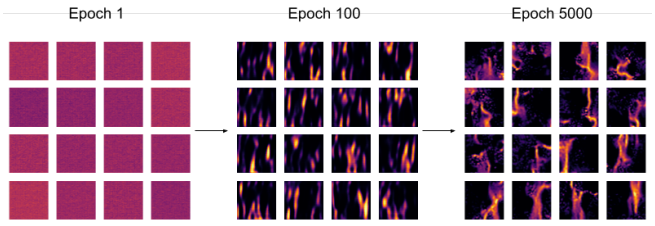

Figure 12. GAN-generated NTFs at epochs 1, 100, and 5000.

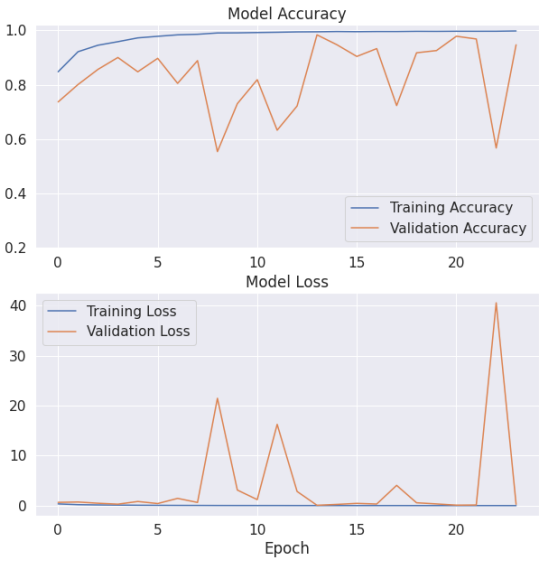

Figure 13. Model accuracy and loss when training on real and GAN-generate samples in a 50 : 50 split.

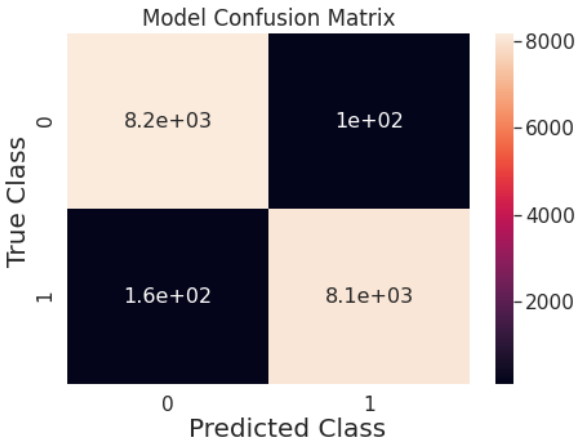

Figure 14. Confusion matrix for the model's best epoch when training on real and GAN-generate samples in a 50 : 50 split.

5 RESULTS AND DISCUSSION

5.1 CNN Performance

5.1.1 Real data training-set

Figure 10 shows the model accuracy and loss over 42 epochs. A callback setting was used to halt training once the validation accuracy had not improved for 10 epochs. By epoch 32, the model had achieved a validation accuracy of 99.61%. Once training had finished, the parameters were restored to their state at epoch 32.

A possible explanation of the fluctuations in the validation loss during training is the noise in **D** and possible outliers in the validation dataset. If the outliers exist nearer the separating hyper-surface, as the hyper-surface fluctuates due to batch noise, the loss for these validation points could significantly change or even result in samples being misclassified if they exist near enough to the boundary.

It can be seen that around epoch 20, the model starts to overfit, characterised by the divergence of the validation loss from the training loss. This can be visualised as the separating hyper-surface starting to closely wrap around all of the training samples near the boundary between $\mathbf{D_1}$ and $\mathbf{D_0}$. In doing this, the model assumes a degree of granularity in **D**, which does not truly exist. This jaggedness of the hyper-surface results in a greater proportion of validation points finding themselves further away from the hyper-surface than when it was smooth. This causes an increase in validation loss. However, it can be seen that even though the validation loss starts to increase after epoch 20, the validation accuracy continues to improve. This is because the hyper-surface is still effective at separating $\mathbf{D_1}$ and $\mathbf{D_0}$; it is only the average distances between the hyper-surface and the validation points which is increasing.

Also, it can be seen that the validation metrics perform better than the training metrics at two points. This is because when the validation set undergoes forward-pass, dropout is not applied. It could be inferred that dropout was applied to kernels particularly useful in categorising the samples at these earlier epochs.

Figure 11 shows a confusion matrix for the model's performance for the most accurate epoch. A confusion matrix can be used to see how the model performed with respect to each class. It can be seen that the model was able to identify each class with an equal degree of accuracy.

5.1.2 GAN training-set

Figure 12 shows the GAN output at epochs 1, 100, and 5000. Unfortunately, the results do not closely resemble NTFs. It appears that the GAN is giving too much attention to the irregularities which occasionally appear in NTFs despite the majority of NTFs being very uniform. This could be a result of the generator overfitting.

Figure 13 shows the training performance of the same model when training on a dataset constructed of 50% real NTFs and 50% generated NTFs. The model was set up with the same callback as before, and it achieved a validation accuracy of 98.38% on the 14^{th} epoch. It can be seen that the training is much less stable. It was first believed that this could result from batch noise caused by too small of a batch size. The batch size was increased to 64, and the model was retrained but to little effect. Instead, the fluctuating validation

Training-Set	Precision	Recall	F1
Real Data	0.996	0.996	0.996
GAN-supplemented Data	0.988	0.981	0.984

Table 3. Table showing the precision, recall and F_1 score for both models at their best performing epoch.

loss could simply result from the generated NTFs not accurately representing $\mathbf{D_1}$, resulting in a much noisier dataset. This would then cause the loss surface to become much more jagged and irregular, making the optimisation process less stable.

Figure 14 shows the confusion matrix of the model trained with supplemented data for the most accurate epoch. It can be seen that the addition of generated NTFs increased the rate of both false positive and false negative detections, with the false negative detections being more adversely affected. The increased rate of false positives could result from the generated NTFs blurring the boundary between the two classes. Many of the generated NTFs resembled the false positives used to overpopulate $\mathbf{D_0}$ in section 4.1.5. Consequently, the addition of the generated NTFs could have distorted $\mathbf{D_1}$ such that there was a greater overlap with $\mathbf{D_0}$. This could result in the model placing the hyper-surface closer to the region occupied by $\mathbf{D_0}$, resulting in more non-NTF sampled falling on the wrong side of the hyper-plane. This could explain the greater rate of false positives detections. Simultaneously, regions of the hyper-surface would have been pushed the other direction [1] to result in a higher rate of false negatives. It is clear that the GAN was not able to represent $\mathbf{D_1}$ accurately; it appears to have created a distribution of samples that overlaps both $\mathbf{D_1}$ and $\mathbf{D_0}$ to some degree.

A model's performance can be further evaluated using the precision, recall and F_1 metrics. The precision is a measure of how valid the model's predictions are or how well it can correctly identify NTFs from non-NTFs. The recall is a measure of how complete the results are or how well the model can correctly identify all NTFs. The F_1 measure is a harmonic mean of the two. They are defined as

$$Precision = \frac{TP}{TP + FP}, \tag{7}$$

$$Recall = \frac{TP}{TP + FN}, \tag{8}$$

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{9}$$

See table 3, for a summary of these metrics.

It is clear from these metrics and the confusion matrices that the dataset containing only real NTFs were most suitable for training a generalisable model. As a result, this was the final model used in the NetFilEx algorithm. Figures B1 and B2 shows the outputted NTF mosaics from NetFilEx.

[1] The phrase "other direction" is used here in an arbitrary sense, it does not necessarily mean "opposite direction" given that the parameter space is high dimensional and the hyper-plane is non-linear. This only means a direction towards the samples of $\mathbf{D_1}$.

5.1.3 Final considerations regarding the applicability of NetFilEx

In the context of an object extraction algorithm, the performance of the CNN is only as good as the quality of the data used in training. If the training data is not representative of the unseen data, then misclassifications will be made. It is believed that the removal of some of the NTFs and the smaller representation of partially visible NTFs would have affected the ability of the CNN to classify all of the cuts from the science frame. This manifested itself in some of the mosaics showing incomplete NTFs. For example, based on the coordinate information provided, two of the mosaics in the second row of Figure B1 appear to be the same NTF which has been falsely separated. The reason for their split would have been due to at least one of the cuts in the line of connected 1's predictions being misclassified as a "0".

Additionally, there still appears to be two mosaics containing no NTFs in the last two rows of Figure B2. Objects like these have proven to be a problem for the CNN in making classifications. It is believed that they are falsely classified as an NTF because, at the (128×128) pix scale, the gas closely resembles certain NTFs. Overpopulating $\mathbf{D_0}$ with the false negatives in section 4.1.3 was done to increasing degrees to try and prevent this. However, these particular frames have remained persistent. Possible solutions include further increasing the representation of the false negatives in $\mathbf{D_0}$ or building a model with a higher capacity.

Future work might be able to build a more sophisticated CNN. It has been shown that skip connections in the model's architecture effectively smooth the loss surface, resulting in a much more convex optimisation process. This will allow the optimiser to converge quicker and more reliably (Li et al. (2018)). Additionally, activation functions such as Mish (Misra (2019)), and Swish (Ramachandran et al. (2017)) have been shown to improve the accuracy of the CNN over ReLU.

As more telescopes in the SKA become available, the availability of data on NTFs in the galactic centre will increase. This will allow for a more resolved sample space on which subsequent CNNs can train. Future work will develop more accurate CNNs which will have lower false positive and false negative rates.

It was mentioned that algorithms like this could help detect NTFs in external galaxies. This will require further development in the SKA's capabilities. Assuming that external NTFs are $\sim$ 0.1pc wide and external galaxies exist on the scale of Mpc away, this gives a required angular resolution on the order of $\sim 0.01''$, an order of magnitude more sensitive than what is expected from the SKA Taylor (2012). Using Reyleigh's criterion with a wavelength of 21.4 cm, this gives a required diameter of $\sim$ 2600 km. Although the SKA does have a maximum baseline diameter greater than this [2]. Further engineering advancements are required to increase the detect external NTFs.

If NTFs exist in external galaxies, it is hoped that algorithms like this will be able to classify NTFs in future surveys instead of humans. For this to be realised, the data of the NTFs found in the GC must be representative of the NTFs found in other galaxies. This is dependant on how the magnetic field in external galaxies compares

[2] https://www.skatelescope.org/wp-content/uploads/2011/03/SKA-Brochure_web.pdf

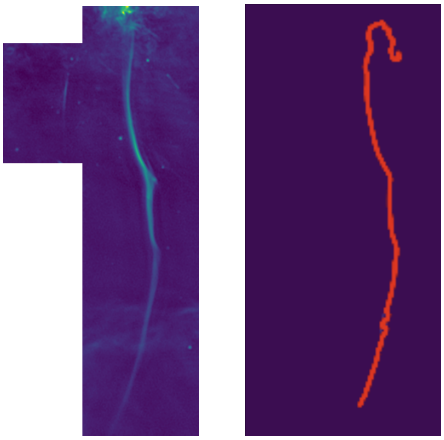

Figure 15. The 36^{th} mosaic showing "The Snake", and its skeleton provided by FilFinder.

to that of the GC. It could be the case that external magnetic fields are not dipolar. In this case, NTFs could have a completely different shape. Also, By considering charged circular motion, if the magnetic field strength in external galaxies were different, the widths of NTFs would be expected to change. This could then have a knock-on effect on the expected length of the NTFs as electrons are confined to orbit in a broader or narrower region. Finally, if the conditions at the centre of external galaxies are different such that the formation mechanism is affected, the energies of the electrons could be different. These considerations could alter the appearance of NTFs in external galaxies, which would have severe adverse effects on the performance of this algorithm, given that it classifies on morphology alone. If this were the case, a larger dataset would be required to be built, representing all of the NTFs expected to be found in these surveys. This process could be aided with the use of GANs more sophisticated than the one used in this project.

5.2 FilFinder results

The mosaics could then be used to investigate the properties of the NTFs. FilFinder was used to investigate the properties of the NTF centred at galactic coordinates ($l = 359.139°, b = -0.226°$), commonly referred to as "The Snake" (Lang (1998); Lazendic et al. (1999); Lang (1999); Bicknell & Li (2001b)) see Figure 15. To allow FilFinder to correctly interpret quantities in physical units such as distance or angular separation, the distance to the object needed to be provided. This was given as 8178 ± 24pc, the distance to Sgr A* (Gravity Collaboration et al. (2019)). Since all currently discovered NTFs occupy the central 200pc region surrounding Sgr A*, this introduces a total maximum error of 2.46% into all distance-dependant properties discovered by FilFinder.

FilFinder identifies filamentary structure in images first by creating a mask to focus on a smaller image region. It then skeletonizes the remaining objects using a host of pruning criteria. It was found that three parameters were most effective in identifying NTFs. These include "smooth_size", which removes small noise variations over a defined length scale and was set to 1pc, "adapt_thresh", which sets the

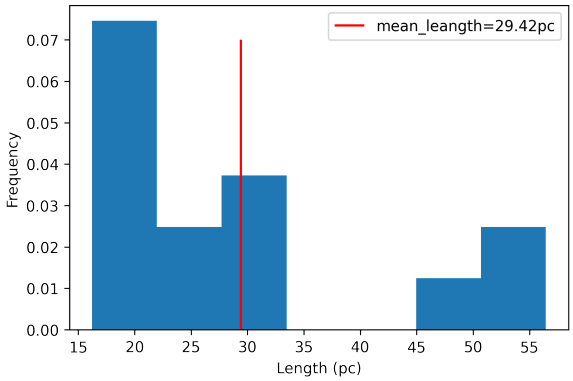

Figure 16. Histogram showing the frequency of lengths of 14 NTFs in the GC.

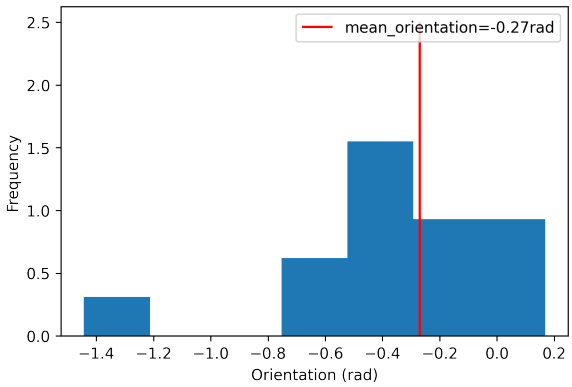

Figure 17. Histogram showing the frequency of orientations of 14 NTFs in the GC. Note that the outlier has been excluded in the calculation of the mean.

expected width of a filament and was set to 0.5pc and "glob_thresh", which sets the expected pixel intensity of a filament and was set to 0.0007. To further ensure that only the NTF is skeletonised, the minimum skeleton length can be set. Here, this was set to 20pc. These criteria produced a skeletonised filament shown in Figure 15.

FilFinder used the skeleton and the provided distance to the object to calculate the length, average orientation and average curvature. These were found to be 63.03 pc, −0.02 rad, and 0.49 rad respectively.

FilFinder also struggles to independently skeletonise NTFs which overlap with each other or that overlap with noisy regions. Of all of the mosaics, 14 were able to pass through FilFinder successfully. Their lengths and orientations were recorded and are presented in Figures 16 and 17.

The histograms show that the 14 NTFs have lengths between 15pc and 55pc in accordance with previous research (Morris & Yusef-Zadeh (1985); Gray et al. (1991, 1995); Morris (2007); Arzoumanian et al. (2011)). They are also all nearly perpendicular to the galactic plane. This excludes the outlier, which instead lies almost parallel to the galactic plane. This outlier is the final mosaic in Figure B2. These observations support the hypothesis of a dipolar magnetic field with local irregularities in the central molecular zone (Yusef-Zadeh & Morris (1987); Gray et al. (1995); Bicknell & Li (2001c); Yusef-Zadeh et al. (2005); LaRosa et al. (2005); Morris (2007)).

6 CONCLUSION

This project has looked at the development of a CNN to classify NTFs in the GC. The motivation for doing this is so that future higher resolution surveys of the GC and deeper surveys of external galaxies can quickly be searched for NTFs. This will shorten the pipeline in getting data to researchers who intend to investigate NTFs by displaying their appearances and coordinates. The CNN was trained on data from the GC provided by MeerKAT.

The built model achieved a validation accuracy of 99.61% at classifying NTFs and was implemented into NetFilEx. A GAN was built to attempt to supplement the training dataset. However, due to a flawed model or overfitting, it was not able to produce representative NTFs. While most of the training was performed on a local machine, certain more intensive elements, including hyperparameter tuning was performed using Google's GPU accelerator. Finally, some of the outputs from NetFilEx was used to investigate some properties of filaments, including "The Snake".

Once the SKA is complete, there will be a greater amount of data from the GC at higher resolutions than ever before. Although there will also be a considerable increase in the depth of surveys, due to the extremely narrow morphology of NTFs, it is believed that the SKA will not be able to resolve NTFs at its current angular resolution. The increase in the SKA's capabilities was considered, and it was found that an increase of angular resolution by one order of magnitude would be required before the SKA could detect NTFs in external galaxies. According to Rayleigh's criterion, the SKA has a baseline diameter greater than the threshold required to observe external NTFs. However, there must be other limitations to overcome before the SKA can detect external NTFs.

Once this is achieved, the model's applicability was discussed in regards to how well the CNN could generalise to unseen NTFs. In the case of external NTFs, this would be heavily dictated by the similarity of the magnetic field of external galaxies with the milky way. The magnetic field highly influences the appearance of the NTFs, and since the CNN makes predictions on morphology alone, a larger sample space would be needed to accommodate for varying magnetic field conditions. In the meantime, there will soon be a wealth of high-resolution data which could be used further to train the CNN in preparation for future surveys.

There is still much work that can be done to improve the performance of this CNN. This area of research is growing at an incredibly fast rate, and many improvements can be made to the CNN to increase accuracy and training stability. The use of more exotic architectures and activation functions was mentioned, and there are a considerable number of other alterations with which one could experiment.

Future work could look at implementing another CNN into the algorithm to directly output some physical properties of the NTFs, which relied on FilFinder to calculate in this investigation. This would require labelling each sample with the value of the relevant physical property. The labels could be found initially using external packages like FilFinder. A different loss function and output activation function would be required, such as Mean Squared Error and Softmax. Then the algorithm could learn to identify the properties of sections of the filaments. When combining the frames into the mosaics, all the relevant predictions from each NTF detection could be averaged to output physical properties such as radius and orientation, or be added together to obtain properties such as length. Additionally, a semantic segmentation algorithm could be developed as a replacement to the CNN in this algorithm. This could be used to highlight NTFs in the FITS images without extracting mosaics, which would be helpful for investigations concerning NTFs and their surrounding region.

REFERENCES

Abadi M., et al., 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/

Abadi M., et al., 2016, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). USENIX Association, Savannah, GA, pp 265–283, https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

Alhassan W., Taylor A. R., Vaccari M., 2018, MNRAS, 480, 2085

Amari S., 1993, Neurocomputing, 5, 185

Aniyan A. K., Thorat K., 2017, ApJS, 230, 20

Arzoumanian D., et al., 2011, A&A, 529, L6

Bicknell G. V., Li J., 2001a, Publ. Astron. Soc. Australia, 18, 431

Bicknell G. V., Li J., 2001b, Publ. Astron. Soc. Australia, 18, 431

Bicknell G. V., Li J., 2001c, ApJ, 548, L69

Braun R., Bourke T., Green J. A., Keane E., Wagg J., 2015, in Advancing Astrophysics with the Square Kilometre Array (AASKA14). p. 174

Cabrera-Vives G., Reyes I., Förster F., Estévez P. A., Maureira J.-C., 2016, in 2016 International Joint Conference on Neural Networks (IJCNN). pp 251–258, doi:10.1109/IJCNN.2016.7727206

Changpinyo S., Sandler M., Zhmoginov A., 2017, CoRR, abs/1702.06257

Chen T., Chen H., 1996, Circuits Systems and Signal Processing, 15, 671

Cotter A., Shamir O., Srebro N., Sridharan K., 2011, in Proceedings of the 24th International Conference on Neural Information Processing Systems. NIPS'11. Curran Associates Inc., Red Hook, NY, USA, p. 1647–1655

Coughlin E. R., Nixon C. J., Ginsburg A., 2021, MNRAS, 501, 1868

Cybenko G., 1989, Mathematics of Control, Signals and Systems, 2, 303

Dumoulin V., Visin F., 2016, arXiv e-prints, p. arXiv:1603.07285

Fang W., Zhang F., Sheng V., Ding Y., 2018, Computers, Materials & Continua, 57, 167

Fitch F. B., 1944, Journal of Symbolic Logic, 9, 49–50

Fukushima K., 1980, Biological Cybernetics, 36, 193

Fukushima K., 2007, Scholarpedia, 2, 1717

Goodfellow I. J., Pouget-Abadie J., Mirza M., Xu B., Warde-Farley D., Ozair S., Courville A., Bengio Y., 2014, Generative Adversarial Networks (arXiv:1406.2661)

Gravity Collaboration et al., 2019, A&A, 625, L10

Gray A. D., Cram L. E., Ekers R. D., Goss W. M., 1991, Nature, 353, 237

Gray A. D., Nicholls J., Ekers R. D., Cram L. E., 1995, ApJ, 448, 164

Harp G. R., et al., 2019, arXiv e-prints, p. arXiv:1902.02426

Hecht-Nielsen 1989, in International 1989 Joint Conference on Neural Networks. pp 593–605 vol.1, doi:10.1109/IJCNN.1989.118638

Heywood I., et al., 2019, Nature, 573, 235

Ioffe S., Szegedy C., 2015, in Bach F., Blei D., eds, Proceedings of Machine Learning Research Vol. 37, Proceedings of the 32nd International Conference on Machine Learning. PMLR, Lille, France, pp 448–456, http://proceedings.mlr.press/v37/ioffe15.html

Kingma D. P., Ba J., 2017, Adam: A Method for Stochastic Optimization (arXiv:1412.6980)

Koch E. W., Rosolowsky E. W., 2015, Filament Identification through Mathematical Morphology (arXiv:1507.02289)

LaRosa T. N., Lazio T. J. W., Kassim N. E., 2001, The Astrophysical Journal, 563, 163

LaRosa T. N., Brogan C. L., Shore S. N., Lazio T. J., Kassim N. E., Nord M. E., 2005, ApJ, 626, L23

Lang C. C., 1998, in American Astronomical Society Meeting Abstracts. p. 104.03
Lang C. C., 1999, in Falcke H., Cotera A., Duschl W. J., Melia F., Rieke M. J., eds, Astronomical Society of the Pacific Conference Series Vol. 186, The Central Parsecs of the Galaxy. p. 498
Lang C. C., Anantharamaiah K. R., Kassim N. E., Lazio T. J. W., 1999a, ApJ, 521, L41
Lang C. C., Morris M., Echevarria L., 1999b, ApJ, 526, 727
Lazendic J., Burton M., Yusef-Zadeh F., Wardle M., Green A., Whiteoak J., 1999, in Combes F., Pineau des Forêts G., eds, H2 in Space. p. E.39
Li H., Xu Z., Taylor G., Studer C., Goldstein T., 2018, Visualizing the Loss Landscape of Neural Nets (arXiv:1712.09913)
Liu W., et al., 2019, Research in Astronomy and Astrophysics, 19, 042
Lukic V., Brüggen M., 2017, in Brescia M., Djorgovski S. G., Feigelson E. D., Longo G., Cavuoti S., eds, Vol. 325, Astroinformatics. pp 217–220, doi:10.1017/S1743921316012771
Lukic V., Brüggen M., Banfield J. K., Wong O. I., Rudnick L., Norris R. P., Simmons B., 2018, MNRAS, 476, 246
Misra D., 2019, arXiv e-prints, p. arXiv:1908.08681
Morris M., 2007, arXiv e-prints, pp astro–ph/0701050
Morris M., Serabyn E., 1996, Annual Review of Astronomy and Astrophysics, 34, 645
Morris M., Yusef-Zadeh F., 1985, AJ, 90, 2511
Nair V., Hinton G. E., 2010, in Proceedings of the 27th International Conference on International Conference on Machine Learning. ICML'10. Omnipress, Madison, WI, USA, p. 807–814
Nicholls J., Strange E. L., 1995, The Astrophysical Journal, 443, 638
Nocedal J., Wright S. J., 2006, Numerical Optimization, second edn. Springer, New York, NY, USA
Norris R., 2011. pp 21 – 24, doi:10.1109/eScienceW.2010.13
O'Shea K., Nash R., 2015, An Introduction to Convolutional Neural Networks (arXiv:1511.08458)
Paré D. M., Lang C. C., Morris M. R., Moore H., Mao S. A., 2019, ApJ, 884, 170
Pearson K. A., Palafox L., Griffith C. A., 2017, Monthly Notices of the Royal Astronomical Society, 474, 478–491
Prandoni I., Seymour N., 2015, Revealing the Physics and Evolution of Galaxies and Galaxy Clusters with SKA Continuum Surveys (arXiv:1412.6512)
Radford A., Metz L., Chintala S., 2015, arXiv e-prints, p. arXiv:1511.06434
Ramachandran P., Zoph B., Le Q. V., 2017, arXiv e-prints, p. arXiv:1710.05941
SARAO, APOD: 2019 July 8 - The Galactic Center in Radio from MeerKAT, https://apod.nasa.gov/apod/ap190708.html
Shore S. N., LaRosa T. N., 1999, ApJ, 521, 587
Smirnov E., Timoshenko D., Andrianov S., 2014, AASRI Procedia, 6, 89–94
Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R., 2014, Journal of Machine Learning Research, 15, 1929
Taylor A. R., 2012, Proceedings of the International Astronomical Union, 8, 337–341
Yao Y., Rosasco L., Caponnetto A., 2007, Constructive Approximation, 26, 289
Yusef-Zadeh F., 2003, ApJ, 598, 325
Yusef-Zadeh F., Morris M., 1987, ApJ, 322, 721
Yusef-Zadeh F., Morris M., Chance D., 1984, Nature, 310, 557
Yusef-Zadeh F., Wardle M., Muno M., Law C., Pound M., 2005, Advances in Space Research, 35, 1074–1084

Layer	Output shape
Convolutional Blocks	6
Input layer	(128, 128, 128)
Convolution	(64, 64, 64)
Leaky ReLU	-
Dropout	-
Convolution	(32, 32, 128)
Leaky ReLU	-
Dropout	-
Flatten	131072
Dense	1

Table A1. Discriminator Architecture. This model has a total of 337,665 trainable parameters

Layer	Output shape
Dense	8192
Batch Normalisation	-
Leaky ReLU	-
Reshape	(32, 32, 8)
Convolutional Transpose	(32, 32, 4)
Batch Normalisation	-
Leaky ReLU	-
Convolutional Transpose	(64, 64, 2)
Batch Normalisation	-
Leaky ReLU	-
Convolutional Transpose	(128, 128, 1)

Table A2. Generator Architecture. This model has a total of 836,646 trainable parameters

APPENDIX A: GAN ARCHITECTURE

APPENDIX B: ALGORITHM RETURNS

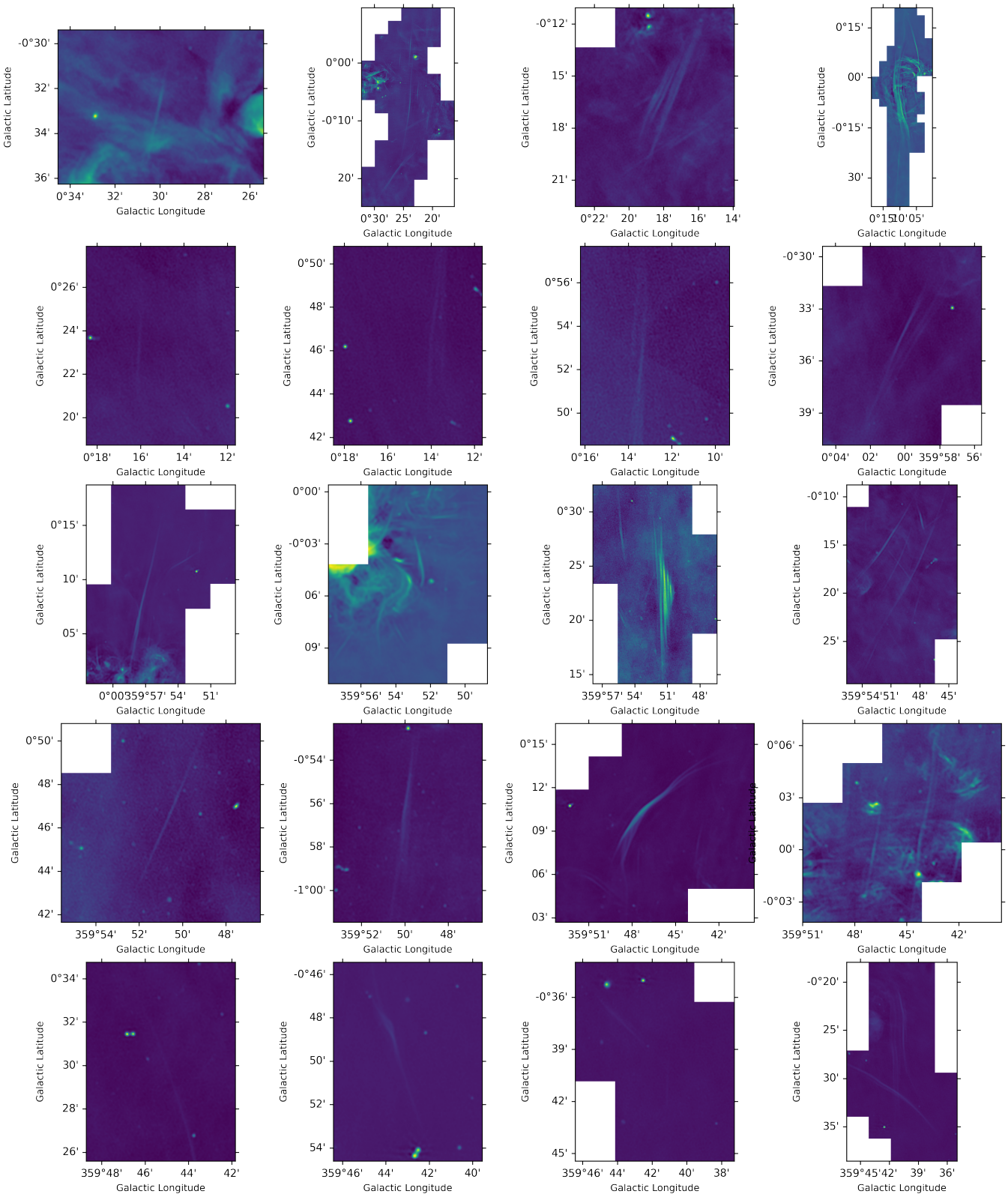

Figure B1. Output of NetFilEx Pt.1

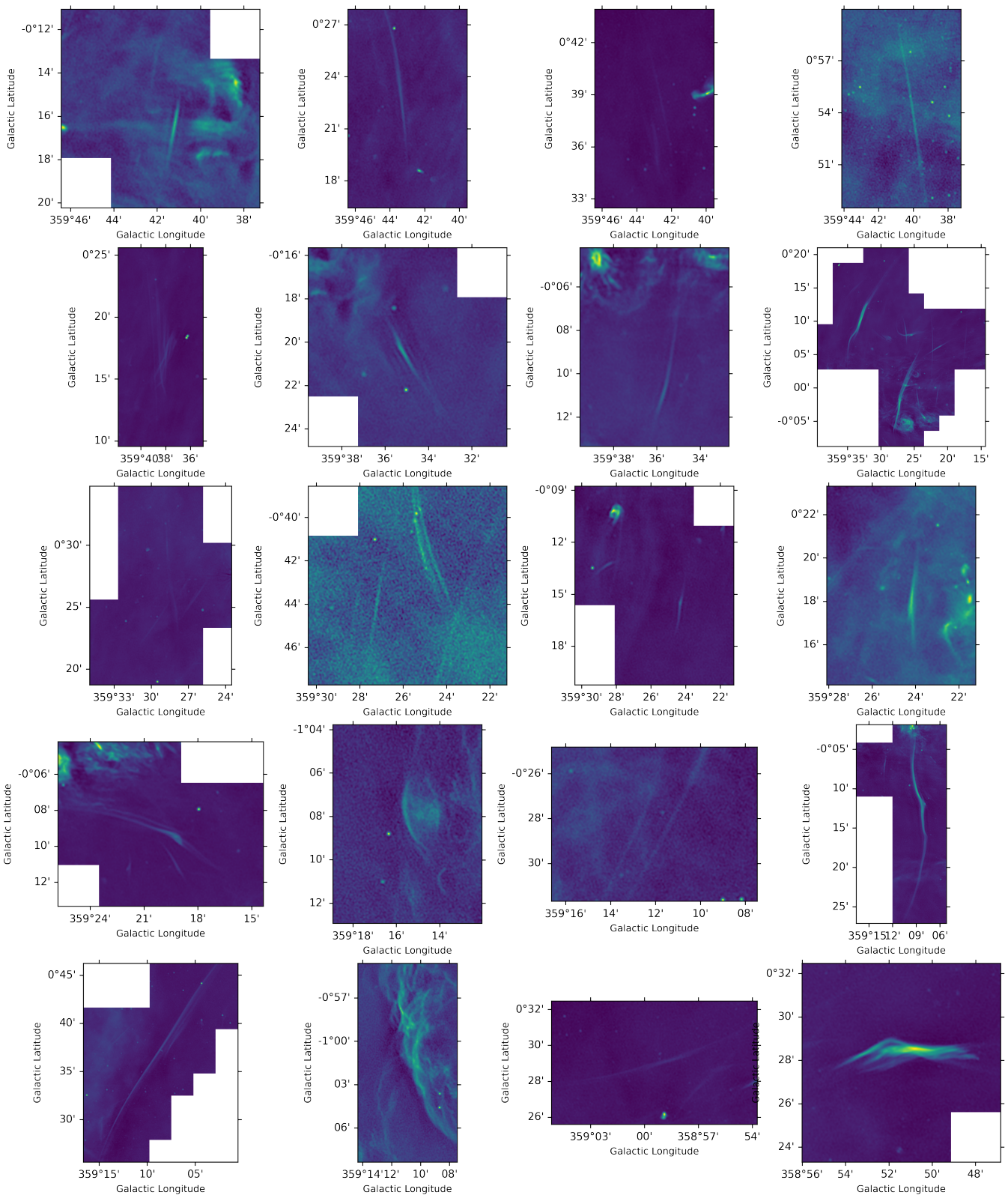

Figure B2. Output of NetFilEx Pt.2

`Netfilex`(*hdu*, *pretrained_model*, *x_dim*, *y_dim*, *Padding="True"*)[source]

Create mosaics of extracted objects.

Returns: HDUList.

Parameters:

hdu: ".FITS" file

".FITS" file containing NTFs to be extracted (or any other object given the relevant pretrained model) which can be loaded with `fits.open("filename")`

pretrained model: Keras model instance

A pretrained model which can be loaded with `tf.keras.models.load_model("filename")`

x_dim, y_dim: int

Dimensions to cut the fits file and input to the pre-trained model.

Padding: {"True", "False"}

When "True", the eight frames surrounding each filament detection are included in the mosaic. Used to ensure objects are complete in the output mosaic if objects straddle multiple frames.

- NetFilEx can be found here: https://github.com/sharifkb/NetFilEx-A-Neural-Network-Filament-Extractor
- To run the algorithm on NTFs, download the pre-trained model here: https://drive.google.com/drive/folders/1XUR9qbvCkBSELu52vAi0O6byP1TCrPxx?usp=sharing
- If using a pre-trained model different to the one above, ensure that the frame pre-prosessing in the "`Classification`" function is identical to the pre-processing used when training the alternate model.