Convolutional Neural Networks
for
Fast Radio Burst Classification

University of Cape Town

Bram Schönfeldt
SCHABR004

Supervised by
Dr Jonathan Shock
&
Prof Amanda Weltman

1st November 2020

ABSTRACT

Convolutional Neural Networks (CNNs) are a type of artificial neural network particularly suited to processing image data that have recently shown success in classifying Fast Radio Bursts (FRBs), millisecond-duration bursts of radio photons largely observed to have extra galactic origins. One of the major challenges of detecting fast radio bursts is sorting through the large amounts of terrestrial radio frequency interference that falsely trigger fast radio burst detection pipelines.

In this project we use CNNs to distinguish between pulsars and radio frequency interference based on their de-dispersed frequency-time graphs. We experiment with architectures of varying depths, input shapes and investigate the impacts of batch normalisation and dropout as regularisation techniques on model performance.

Using a significantly simpler architecture, our best models are able to achieve accuracy, precision and recall values $> 98\%$ which is comparable to performance seen in the literature.

DECLARATION

1. I know that plagiarism is a serious form of academic dishonesty.
2. I have read the UCT document *Avoiding Plagiarism: A guide for students*, am familiar with its contents and have avoided all forms of plagiarism mentioned there.
3. Where I have used the words of others, I have indicated this by the use of quotation marks.
4. I have referenced all quotations and properly acknowledged other ideas borrowed from others.
5. I have not and shall not allow others to plagiarise my work.
6. I declare that this is my own work.

Signature:

CONTENTS

ACKNOWLEDGEMENTS

To *Dr Jonathan Shock*, *Prof Amanda Weltman* and *Prof Ben Stappers* for giving their time, knowledge and encouragement, I am truly grateful. It is rare that an honours project affords one time to think about one's place in the universe.

I also have the *National Research Foundation* to thank for helping to fund my honours year. Studies are substantially easier to focus on when one does not have to worry about how to fund them. Beyond the National Research Foundation, I am immensely appreciative to my family, particularly my mother *Prof Hannah Le Roux*, for supporting my studies to date.

Lastly to *Joséphine* for bringing hope to otherwise particularly bleak times, I likely owe my sanity.

INTRODUCTION

1.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of artificial neural network particularly suited to processing visual inputs because of their ability to exploit spatial correlations and learn hierarchical internal representations of data (LeCun, Kavukcuoglu and Farabet 2010).

More generally, CNNs are suited to data with a *known grid-like topology*

As opposed to having neurons connected to every feature of the input data, which we typically think of as an image with height, width and a number of channels, sparsely connected *filters*, or kernels, connect to local portions of the image. These filters are relatively small volumes of weights which we take the dot product of with a local portion of the image. The result of this dot product is large if there is a 'similarity' between the filter and the local portion of the image it connects to. This *sparse connectivity* helps reduces the memory requirements and improves the statistical efficiency of the network (Goodfellow, Bengio and Courville 2016). There is also the notion that a filter might have relevance at many parts of the image. Thus the same filter is typically repeated periodically along the whole image. These repeated filters all share the same weights, which is known as *parameter sharing* or having *tied weights*, and helps further reduce the memory requirements of the network (Goodfellow, Bengio and Courville 2016).

A typical filter shape is $3 \times 3 \times C$; C is the number of channels (e.g. 3 for an RGB image) of the input volume

We explain the components of CNNs that leverage these concepts of sparse connectivity, parameter sharing in a later section. In the meanwhile, we turn out attention to tracing the history of modern convolutional networks.

1.1.1. A Brief History

This section is not pertinent to the application of CNNs to FRBs. The hurried reader may wish to skip or skim over it.

The conception of CNNs was influenced by work by Hubel and Wiesel in the 1960s that tried to understand the visual cortex by studying the response of neurons in the visual cortex of cats to various visual stimuli (Hubel and Wiesel 1962). Their findings included identifying simple and complex organisations of the receptive fields of different cells, identifying the importance of shape, position and orientation of visual stimuli in obtaining responses from cells, and identifying discrete columns of cells in the cortex with the same receptive-field axis orientation. They go on to suggest that these columns of cells, or *functional units*, might be hierarchical, with cells with complex receptive fields receiving projections from a number of cells with simple fields in the same column (Hubel and Wiesel 1962).

Self-organising meaning the model learns in an unsupervised manner.

In 1980, the *Neocognitron*, a "self-organising neural network model for a mechanism of pattern recognition unaffected by shift in position", was introduced based on the hierarchical model of the visual cortex proposed by Hubel and Wiesel (Fukushima and Miyake 1982). This model comprised an input layer followed by a number of modular structures, each of which is composed of an initial layer of "S-cells", similar to the simple cells introduced by Hubel and Wiesel, and a secondary layer of "C-cells", similar to their complex cells. Through repeated stimulation, the C-cells in this neural network learnt to respond to only specific stimulus patterns, and, importantly, this response was independent of the position of the pattern in the input field. The Neocognitron represents one of the first neural networks with local connectivity and hierarchical structure among its neurons (Wang et al. 2019).

Later, with the popularisation of the *backpropagation* algorithm by Rumelhart, Hinton and Williams 1986, neural networks trained using backpropagation were shown as promising candidates for image classification problems. Notably, the "LeNet-5" CNN, trained using backpropagation, was applied to handwritten digit recognition with performance surpassing the other methods at the time. The stated "main message" of the paper that introduced LeNet-5 was that better pattern recognition systems would come out of automatic learning, as opposed to hand-picked heuristics (LeCun et al. 1998).

From 2010 to 2017, the ImageNet Large Scale Visual Recognition Challenge

(ILSVRC), provided a platform - a now over 14 million, labelled, image data-set - to showcase the performance of many of the state-of-the-art visual object recognition algorithms (Russakovsky et al. 2015). In 2012, a CNN called "AlexNet" attracted wide-spread attention when it won the 2012 ILSVRC, significantly outperforming the competing models (Krizhevsky, Sutskever and Hinton 2012). Since then, variations on CNNs have consistently achieved state of the art performances on the ImageNet data-set, which remains a benchmark for image classification tasks. These CNNs include VGGNet (Simonyan and Zisserman 2014), GoogLeNet/Inception-v1 (Szegedy et al. 2015), and ResNet (He et al. 2016).

1.1.2. Components

Convolutional layers

The view of convolutional layers that we take is that of a set of learnable filters connected to local portions of an image. In practice, the data inputted into CNNs are tensors with a height H, width W and a number of depth channels C. Additionally, these data are typically batched into B batches.

For an introduction that relates to the convolution operation, see the section *The Convolution Operation* in Goodfellow, Bengio and Courville 2016

Thus the input tensor X has shape $B \times H \times W \times C$.

Let us consider how a convolutional layer K consisting of a set of I filters, each of width W', height H' and depth matching the input tensor C interacts with our input tensor X. This interaction takes the form of a weighted sum along the depth channels, and within the portion of the input tensor's height and width that each filter is connected to. If we assume that a filter is 'useful' at all positions in the input tensor, then it makes sense to repeat it so that a copy of the filter is connected at every position in the input tensor. The elements of the output of a convolutional layer are thus:

Having multiple filters with the same weights is called *weight sharing*

Here, we assume that we start indexing at 0, as is the case in practice, e.g. in Python

$$\mathsf{Z}_{b,h,w,i} = \sum_{h',w',c} \mathsf{X}_{b,h+h',w+w',c}\mathsf{K}_{i,w',h',c}$$

Note that without any *padding*, the output volume will shrink slightly and in order to maintain the same width and height, we need to add $H' - 1$ rows and $W' - 1$ columns of padding around the input tensor.

Another aspect of convolutional layers is the *stride* of the filters. We have previously assumed that a filter is connected to every spatial position in the input tensor. However, we may want to only connect the filter every s positions. For instance, when $s = 2$, the filter is connected at every second positions in the input tensor. We modify the previous equation to take the stride into account:

$$\mathsf{Z}_{b,h,w,i} = \sum_{h',w',c} \mathsf{X}_{b,h\cdot s+h',w\cdot s+w',c}\mathsf{K}_{i,w',h',c}$$

Notice that in increasing the stride, we decrease the width and height of the output tensor.

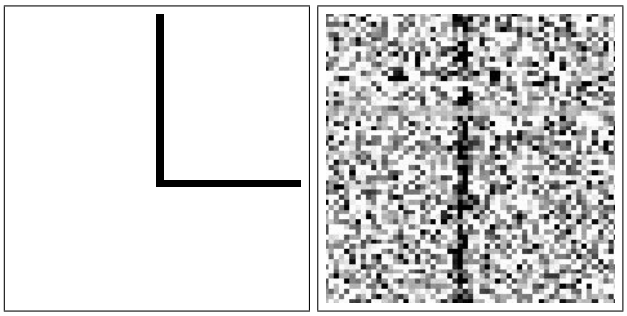
As a quick illustration of the impacts of applying a filter to an image, we define two $3 \times 3 \times C$ filters, where C is the number of channels in the image, which pick up on horizontal and vertical lines. One channel 'slice' from each of these filters is given by:

Source code for all the illustrations that follow can be found in this GitHub *repository*

$$\mathsf{K}_{\text{horizontal}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathsf{K}_{\text{vertical}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

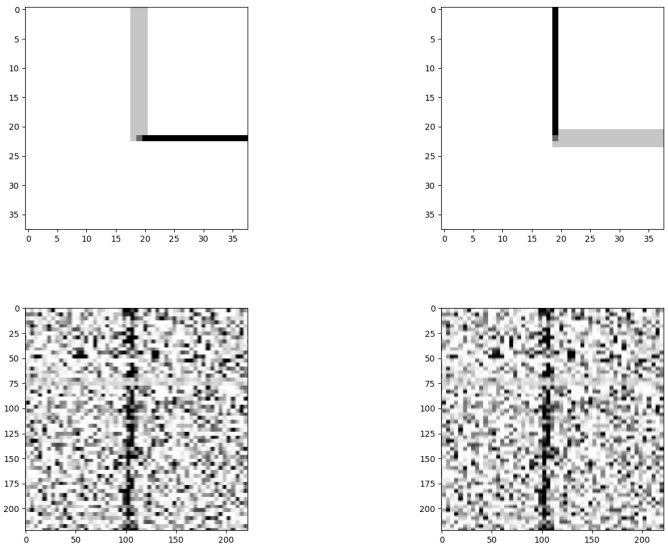
These same slices repeat along the depth of the filter

We consider the following 2 example images. An *L* of pixels, and an example of a pulsar from the dataset of this project.



Left: *L* of pixels ($40 \times 40 \times 4$);
Right: de-dispersed *pulsar* ($224 \times 224 \times 3$)

Horizontal filter	Vertical filter



The effects of applying this $3 \times 3 \times c$ filter is a lot more pronounced in the small L

As can be seen, the horizontal filter has the effect of sharpening horizontal lines and blurring vertical lines (the inverse is true for the vertical filter).

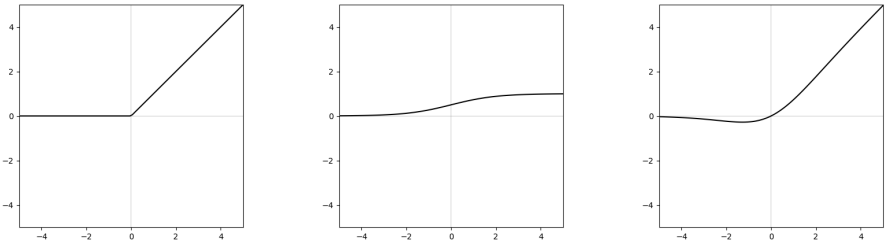
Another thing to note in this illustration is the impact of the size (height and width) of the filter. The pulsar image is large ($224 \times 224 \times 3$) in comparison to the L image ($40 \times 40 \times 4$) and the effects of the filters on the pulsar are not as apparent. It will likely take successive applications of filters to pick up on more large scale patterns in the image.

In the data processing section (2.1) we will further discuss the current data format.

Activations

A linear transformation followed by an activation function lies at the heart of every deep neural network (Ramachandran, Zoph and Le 2017). Activation functions are typically non-linear transformations applied to the output of layers and help neural networks to model non-linear relationships. We briefly illustrate a selection of activation functions:

ReLU	Sigmoid	Swish
$\max(z, 0)$	$\sigma(z) = (1 + e^{-z})^{-1}$	$z \cdot \sigma(\beta z)$



Rectified Linear Units ReLUs, which have the appeal of simplicity, are the most widely used activation function and are a good default choice (Jarrett et al. 2009; Nair and Hinton 2010; Glorot, Bordes and Bengio 2011). However, a ReLU is not without its faults. Since it's derivative is discontinuous at 0, there is the risk of losing gradient information (Misra 2020).

Research into finding better activation functions includes Google Brain's exhaustive and reinforcement learning-based search that led to the discovery of *Swish* (Ramachandran, Zoph and Le 2017). More recently, *Mish* has been proposed as an improvement to *Swish* (Misra 2020).

Pooling layers

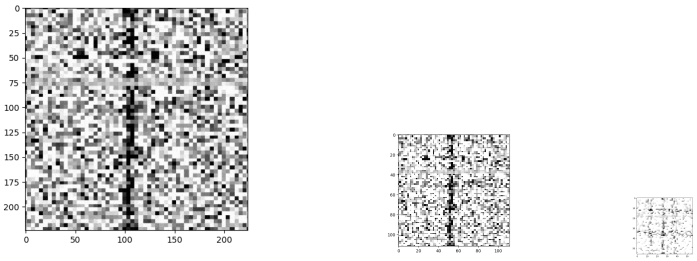
Pooling layers 'down-sample' tensors by summarising local portions of the tensor with a statistic. In addition to reducing the size of the input of the proceeding layer, pooling helps make the learnt representations invariant to small translations of the input (Goodfellow, Bengio and Courville 2016). This down-sampling typically preserves the number of channels of the input tensor, and reduces the tensor along the height and width dimensions.

One popular pooling layer is the *max pooling layer*, which summarises a local portion of a tensor with the maximum value in that portion (Zhou and Chellappa 1988). Similar to the filters a convolutional layer, units in a max pooling layer have a width and height and are arranged along the input volume based on a stride value.

We briefly illustrate the impact of applying max pooling using 2×2 units with

stride 2 to our pulsar image:

Original image	1^{st} max pool	2^{nd} max pool



From this we can observe the effect that the stride of 2 has on reducing the size of the image and the effect that the max operation has on lightening the image.

Another type of pooling layer is the *global average pooling layer*. For an input volume with height H, width W and a number of depth channels C, global average pooling produces an output of size C by averaging over the spatial information $H \times W$.

Global average pooling layers have been proposed as an alternative to fully-connected layers on the end of CNNs (Lin, Chen and Yan 2013). A global average pooling layer requires no parameters, so is less computationally intensive compared to a fully-connected layer. Also, having the outputs* from each filter from the previous layer summarised by a single value which is then fed into a sigmoidal function for classification encourages correspondence between the classes and the feature maps. Since spatial information is averaged over, global average pooling layers are 'robust' to spatial translations of the input data (Lin, Chen and Yan 2013).

*Sometimes called *feature maps* (LeCun, Kavukcuoglu and Farabet 2010)

1.1.3. Training CNNs

Normalisation

Normalisation in machine learning involves processing data so that it lies on some common scale, typically with the goal of improving the stability of the model. Normalisation applies to not only the data fed into a model, but to the

outputs at different stages of a model, particularly DNNs. The introduction of *batch normalisation*, which normalises ouputs at different stages of DNNs, substantially improved the stability and reduced the training time of state-of-the-art DNNs (Ioffe and Szegedy 2015). We briefly mention common practices in processing and normalising image data based on the top entries into ILSVRC (Russakovsky et al. 2015) before moving on to an explanation of batch normalisation.

See *Machine Learning Mastery* for more on data processing practices

In AlexNet (Krizhevsky, Sutskever and Hinton 2012), Inception-v1 (Szegedy et al. 2015), VGG (Simonyan and Zisserman 2014) and ResNet (He et al. 2016), all either winners or runners up in ILSVRC, the image data were standardised to a set spatial shape, and the pixel values were *centered*. Although not always explicitly stated, this centering typically involved estimating the mean pixel value per channel of the images in the training set and then subtracting that from the pixels in the corresponding channels of each image. A later Inception implementation, Inception-v3, (Szegedy et al. 2016) made available in the *Keras Applications library* scales the pixel values to be in range $[-1, 1]$.

This centering of data around zero by subtracting an estimated mean is similarly used in batch normalisation (Ioffe and Szegedy 2015). In batch normalisation, this mean is calculated over a so-called mini-batch, a subset of the full data-set which we typically use to estimate the gradient in stochastic gradient descent. Beyond centering its inputs, batch normalisation scales them by dividing by the standard deviation of the mini-batch. Finally, in order to maintain representational power of the network, the normalised inputs are scaled and shifted by some learnable parameters. Formally, for a batch of m inputs each with d dimensions, $\mathcal{B} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_m\}$, $\boldsymbol{x}_i = (x_1, x_2, ..., x_d)$, batch normalisation performs the following transformation:

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i \qquad \text{Batch means}$$

$$\boldsymbol{\sigma}^{\circ 2}_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}_i - \boldsymbol{\mu}_{\mathcal{B}})^{\circ 2} \qquad \text{Batch variances}$$

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{\mu}_{\mathcal{B}}}{(\boldsymbol{\sigma}_{\mathcal{B}} + \boldsymbol{\epsilon})^{\circ 1/2}} \qquad \text{Normalised input}$$

$$\boldsymbol{y} = \boldsymbol{\lambda} \odot \hat{\boldsymbol{x}} + \boldsymbol{\beta} \doteq BN(\boldsymbol{x}; \boldsymbol{\lambda}, \boldsymbol{\beta}) \qquad \text{Scale and shift}$$

where $\boldsymbol{\epsilon}$ is a constant added for numerical stability, $\boldsymbol{\lambda}$ and $\boldsymbol{\beta}$ are learn-able parameters that scale and shift the inputs and $\circ 2$, $\circ 1/2$ and $\odot$ represent element-wise

or *Hadamard* operations. During inference, instead of estimating the mean and variance from mini-batches, the overall mean and variance observed over training are used. For the variance, the unbiased estimate is used.

Inference variance:

$$\frac{m}{m-1}\mathbb{E}[\boldsymbol{\sigma}_{\mathcal{B}}^{\circ 2}]$$

Despite batch normalisation's relative simplicity, reasons for why it improves performance are said to be poorly understood (Santurkar et al. 2018). The original explanation of it reducing the *internal covariate shift* has been deemed speculative (Lipton and Steinhardt 2018). Santurkar et al. 2018 go further and say that distributional stability of layer inputs have little to do with the success of batch normalisation. Instead, batch normalisation makes the optimisation landscape significantly smoother, inducing more stable behavior of the gradients (Santurkar et al. 2018).

Batch normalisation nevertheless appears in most recent CNNs (Szegedy et al. 2016; Szegedy et al. 2017; Howard et al. 2017; Zhang et al. 2018a)

Hyper-parameter Optimisation

Before CNNs begin to learn* a set of weights and biases, choices have to be made concerning the configuration of the model and the algorithm used to minimise the loss function. These choices made before training begins involve defining a number of *hyper-parameters*. Since choosing the optimal set of hyper-parameters can be a time-consuming process, a number of automatic hyper-parameter optimization techniques have been researched and applied to deep learning. We briefly explain a selection of these techniques.

*Use a form of stochastic gradient descent and backpropagation to update the weights such that a loss is minimised

For a more thorough review, see Yu and Zhu 2020 or Yang and Shami 2020

One of the most commonly used approaches is that of *grid search* where a grid of configurations is defined based on the Cartesian product of finite sets of values of different hyper-parameters of interest. Each configuration is evaluated by fitting a model with the configuration and recording some performance metric (e.g. lowest validation loss achieved). Further fine-grained grid searches may be conducted around promising configurations. Although easily parallelisable, grid search is computationally intensive. For k hyper-parameters with n distinct values, the complexity of grid search is $O(n^k)$

Example grid for 2 different filter dimensions and 2 different activations:

$(1, \text{ReLU})$ $(1, \text{Swish})$
$(3, \text{ReLU})$ $(3, \text{Swish})$

Random search, where each configuration consists of a set of randomly chosen (within defined ranges) hyper-parameters, has been shown to on par with if

not better than grid search using a fraction of the computation time (Bergstra and Bengio 2012). Random search shares the benefits of conceptual simplicity and ease of parallelization with grid search, but improves upon grid search by allowing a pre-defined budget of how many configurations to explore to be set, and by effectively searching a larger configuration space (Bergstra and Bengio 2012).

Unlike grid search and random search where each evaluation is independent, *Bayesian optimisation* makes use of information from previous evaluations to inform the next evaluation, allowing it to find promising configurations in fewer iterations (DeCastro-García et al. 2019). Bayesian optimisation consists of:

1. a probabilistic *surrogate model* of the objective function (e.g. the validation loss)
2. an *acquisition functions* which determines the next configuration to evaluate

It iteratively explores and exploits* promising configurations of hyper-parameters in the following way (Yu and Zhu 2020):

*Ideally finding a balance between the two

1. A prior distribution is built of the surrogate model
2. The set of hyper-parameters that perform best on the surrogate model are determined
3. Using the acquisition function on the current surrogate model, a set of hyper-parameters are sampled
4. The performance of this set of hyper-parameters is determined by fitting the actual model with them and recording some indicator of performance (the objective function)
5. The surrogate model is updated based on these results

Steps 2 to 5 are repeated for a defined number of iterations, or until some notion of convergence is achieved and the best performing hyper-parameters are reported

The surrogate model can be any analytic function or non-parametric model. The prior distribution describes the hypothesis on the objective function and the posterior distribution is fitted based on the evaluations (Yu and Zhu 2020). Choices for the surrogate model include *random forests* (Hutter, Hoos and Leyton-Brown 2011), *Tree Parzen Estimators* (TPE) (Bergstra et al. 2011) and *Gaussian*

processes (Rasmussen and Williams 2006), and choices for the acquisition function include *probability of improvement* (Kushner 1964), *Gaussian Process Upper Confidence Bound* (GP-UCB) (Srinivas et al. 2009), and the *expected improvement algorithm* (Mockus, Tiesis and Zilinskas 1978). Of these, Gaussian processes and the expected improvement algorithm are the most popular choices of surrogate model and acquisition function (Yu and Zhu 2020).

However, care must be taken when using Gaussian processes for hyper-parameter optimisation since they assume continuous input variables (Garrido-Merchán and Hernández-Lobato 2020). Hyper-parameters often take on integer or categorical values and can be conditional on other hyper-parameters. Surrogate models with tree-based structures* are one solution to this problem, particularly for modelling conditional hyper-parameters (Yu and Zhu 2020).

*such as random forests and TPE

Although Bayesian optimisation is able to find better configurations in fewer iterations by learning from past evaluations, this reliance on previous evaluations makes it a sequential process and unable to benefit from parallelisation without modification. One such modification that has been made to Bayesian optimisation involves borrowing from *HyperBand* - a parallizeable *bandit-based* approach to hyper-parameter optimisation.

HyperBand extends the idea of *successive halving* (Jamieson and Talwalkar 2016), which involves allocating a budget B* amongst n_i configurations and then dropping the $n_i/2$ worse performing configurations at the end of each iteration, effectively doubling the resources allocated to the best $n_{i+1} = n_i/2$ performing configurations in the next iteration. HyperBand improves upon successive halving by addressing the trade-off between the budget per configuration B/n and the number of configurations n. If the number of configurations is too large, the budget B/n for each configuration becomes too small and promising configurations might be dropped too soon. On the other hand, if the number of configurations is too small, promising configurations might be left unexplored. HyperBand explores a range of budgets and number of configurations by essentially* performing a grid search over n, and using successive halving as a subroutine for each each value of n (Li et al. 2017).

*for instance, the size of the training set

*It is a little bit more sophisticated and worth reading the paper

BOHB combines the budget control and computational efficiency in the early stages of training of HyperBand with the robustness and strong final stage performance of Bayesian optimisation, and shows state-of-the-art performance in optimising the hyper-parameters of a range of different machine learning mod-

els including CNNs (Falkner, Klein and Hutter 2018).

Regularisation

In practice, we are interested in how well a model classifies data which has not already been classified. In this context, our hope is that our model will gain a general understanding of what characterises a FRB and be able to recognise those FRBs that have not been discovered yet. Strategies that are designed to improve how well models generalise to unseen data are collectively known as *regularisation* (Goodfellow, Bengio and Courville 2016). Regularisation places constraints on the quantity and type of information that a model can store, forcing the model to focus on learning the most pertinent patterns.

We start by discussing *parameter norm penalties* which limit the capacity of the model by penalising the objective function with the addition of a term related to a norm of the model's parameters. Denoting the objective function as J, and its penalised counterpart as $\tilde{J}$, we introduce 2 popular types of regularisation:

We typically only penalise the weights, and not the biases

$$\tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \quad \text{and} \quad \tilde{J}(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) = J(\boldsymbol{w};\boldsymbol{X},\boldsymbol{y}) + \lambda\|\boldsymbol{w}\|_1$$

L_2 regularisation* L_1 regularisation

*Technically the L_2 norm squared. More generally, the penalty term could be based any L_p-norm

where λ is a parameter weighting how much the objective is penalised, $\|\boldsymbol{w}\|_2^2 = \boldsymbol{w}^\top\boldsymbol{w}$ is the sum of all the squared weights and $\|\boldsymbol{w}\|_1 = (\boldsymbol{w}^\top\boldsymbol{w})^{1/2}$ is the sum of the absolute values of the weights.

L_2 regularisation, commonly known as *weight decay*, does as its name suggests - decays the weights of a model by a constant factor at each step of learning. In comparison, L_1 regularisation has the effect of pushing certain weights towards 0, resulting in a more sparse weight matrix (Goodfellow, Bengio and Courville 2016).

Alongside batch normalisation, one of the most popular regularisation techniques is *dropout*, which has been shown[1] to be more effective than other regularization techniques such as weight decay, filter norm constraints and sparse activity regularisation in deep neural networks (Srivastava et al. 2014). The key idea behind dropout is to randomly drop units along with their connections from a neural network during training. Thus, dropout can be thought of as a practical means of training the ensemble[2] consisting of all "thinned" sub-

[1]The same network architecture was trained on the MNIST data set with different regularisation techniques

[2]Averaging predictions from an ensemble of networks falls under the broader idea of *bagging*

networks that can be formed by removing non-output units from a network (Goodfellow, Bengio and Courville 2016).

During training, a mask $\boldsymbol{r}$ consisting of 0s and 1s is applied to the outputs of the non-output layer $\boldsymbol{y}$. For a neural network with L hidden layers where each layer is indexed with the superscript $l \in \{1, 2, ..., L\}$, the feed-forward operation with dropout can be described as:

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p) \\ \tilde{\boldsymbol{y}}^{(l)} &= \boldsymbol{r}^{(l)} \odot \boldsymbol{y}^{(l)} \\ \boldsymbol{y}^{(l+1)} &= \sigma(\boldsymbol{W}^{(l+1)} \tilde{\boldsymbol{y}}^{(l)} + \boldsymbol{b}^{(l)}) \end{aligned}$$

The outputs are masked, $\odot$ is the element-wise product

where $\boldsymbol{W}^{(l+1)}$ and $\boldsymbol{b}^{(l)}$ are the weights and biases of layer $l+1$, and σ is its activation function. $\boldsymbol{r}^{(l)}$ is a vector of independent random variables each of which follows a Bernoulli distribution and takes on the value 1 with probability p and is 0 otherwise.

During testing, dropout is not used, and the weights are scaled by $\boldsymbol{W}_{\text{test}}^{l} = p\boldsymbol{W}^{l}$ so that the expectation of the outputs remains the same. Similarly, one can scale the weights during training by $1/p$ (Srivastava et al. 2014).

However, despite dropout's success in improving generalisation in deep neural networks, one might need to take care when integrating it into CNNs, especially alongside other regularisation techniques, particularly Batch Normalisation (BN) (Cai et al. 2019). Li et al. 2019 investigate why dropout and batch normalisation often lead to worse performance when combined in certain modern neural networks*, but can also lead to improved performance as in wide residual networks (Zagoruyko and Komodakis 2016). Worsened performance is attributed to conflicting shifts in variance arising from BN and dropout when moving from the training to the testing phase (Li et al. 2019). Dropout scales the outputs by a factor of $1/p$ during training, which is then not present during testing, whereas dropout keeps the variance constant during testing based on the variance observed during training. One of their proposed solutions to deal with this shift in variance introduced by dropout is a reformulation of dropout called *Uout* which multiplies the outputs of the previous layer by a vector of independent random variables, each following a uniform distribution (Li et al. 2019).

*For instance, Li et al cite the paper that introduces Batch Normalisation (BN) (Ioffe and Szegedy 2015), which mentions that dropout can be removed when BN is used. However, Ioffe and Szegedy do not definitively say that there is a 'disharmonious' relationship between the two, as in Li et al. 2019

$$\tilde{\boldsymbol{y}}^{(l)} = \boldsymbol{r}^{(l)} \odot \boldsymbol{y}^{(l)} \quad r_j^{(l)} \sim \text{Uniform}(-\beta, \beta)$$

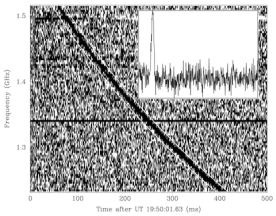
Cai et al. 2019 speculate that the placement of dropout layers before BN layers is detrimental since the BN layers, which seek to stabilise the mean and variance of their inputs, receive inputs from the dropout layer with fluctuating means and variances. Instead, they propose to place dropout after nonlinear layers and before convolutional layers, with batch normalisation applied to the outputs of the convolutional layers.

They also investigate alternatives to traditional dropout, such as *drop-channel*, which is inspired by the observation that neurons are to the canonical neural network as channels are to convolutional neural networks. They propose that dropping channels would be structurally more effective in regularisation:

$$\begin{aligned} r_c^{(l)} &\sim \text{Bernoulli}(p) \\ \tilde{\mathsf{Y}}_{h,w,c}^{(l)} &= r_c^{(l)} \mathsf{Y}_{h,w,c}^{(l)} \\ \mathsf{Z}_{h,w,i}^{(l+1)} &= \sum_{h',w',c} \tilde{\mathsf{Y}}_{h+h',w+w',c}^{(l)} \mathsf{K}_{i,w',h',c}^{(l+1)} \end{aligned}$$

Here, $\mathsf{Y}^{(l)}$ is the output volume from layer l, K_i^{l+1} is the i^{th} filter/kernel of of layer $l+1$. What is important to note is that the channels, each a 2D slice with width W and height H, are masked, as opposed to the individual entries of $\mathsf{Y}^{(l)}$.

1.2. Fast Radio Bursts



The Lorimer Burst (Lorimer et al. 2007)

Transients are astronomical events taking place over human-observable timescales.

Fast Radio Bursts (FRB) are astronomical phenomena, or more precisely *transients*, characterized by one or more very bright, millisecond-duration bursts of radio photons. In this introduction, we also use the term FRB to refer to the characteristic bursts of radio photons caused by these phenomena. Their allure goes beyond their enigmatic origins - they provide a means of better understanding the nature of our universe (Walters et al. 2019). Several FRBs have been localised to host galaxies outside of our own (Chatterjee et al. 2017; Bannister et al. 2019; Ravi et al. 2019; Macquart et al. 2020; Prochaska et al. 2019; Marcote et al. 2020). However, a galactic *magnetar* has also been the recent source of a FRB (Andersen et al. 2020a).

There are many theories about what FRBs are and whether there are even multiple classes of FRBs. There are some FRBs which repeat (Chatterjee et al. 2017; Marcote et al. 2020), while others appear to be once-off (Bannister et al. 2019;

Prochaska et al. 2019). One of the leading theories for the origin of at least some FRBs is that they are caused by *magnetars* (Andersen et al. 2020b) - young, highly magnetized neutron stars which occasionally emit huge bursts of X-rays and gamma-rays (Kaspi and Beloborodov 2017). - which is supported by the recent detection of a bright millisecond-duration radio burst from a galactic magnetar (Andersen et al. 2020a).

See frbtheorycat.org for a catalogue of FRB theories.

The first FRB detection happened in 2007 (Lorimer et al. 2007) and since then over 100 verified detections have been reported (Petroff et al. 2016). With a new generation of radio telescopes coming online over the next few years, increasingly more FRB detections are expected (Walters et al. 2019). With these new sources of FRB data on the horizon, along with the fast pace of research in the field, there is hope that we will soon understand these radio transients better, as well as our universe better by making use of FRBs to probe deep into space and the invisible, intergalactic medium that these pulses travel through (Petroff, Hessels and Lorimer 2019; Walters et al. 2019).

See frbcat.org for a catalogue of published FRBs.

Searching for Fast Radio Bursts

As FRBs travel through the ionized interstellar medium, they experience a frequency-dependent delay, quantified by a *dispersion measure* (DM). Searches for FRBs are typically done by looking at high time and frequency resolution radio astronomy data and then compensating for the frequency-dependent delay, or "dispersion" of the bursts. The de-dispersed data is then averaged over the frequency channels to generate a time-series which is then convolved with box-car kernels. If the result of this convolution exceeds a certain threshold, the observation is flagged for further visual inspection by a human (Agarwal et al. 2020).

Although advances in de-dispersion algorithms and and GPU-accelerated pipelines have made real-time FRB searches possible (Barsdell et al. 2012; Zackay and Ofek 2017), Gaussian noise and Radio Frequency Interference (RFI), especially man-made terrestrial RFI, create increasingly larger numbers of false positive detections (Gary, Liu and Nita 2010), which can be unfeasible for humans to sift through. According to Connor and Leeuwen 2018, the ratio of "apparently worthwhile" candidates to true FRBs can easily be over $10^5 : 1$, so being able to precisely distinguish between RFI and FRB can save multiple hours of human inspection. There are general RFI mitigation techniques (Eatough, Keane

and Lyne 2009; Gary, Liu and Nita 2010; Dumez-Viou, Weber and Ravier 2016), however convolutional neural network based models have recently shown promise, particularly in being robust to noise (Zhang et al. 2018b).

1.3. Machine Learning & Fast Radio Bursts

Applying Deep Learning to Fast Radio Burst Classification, Connor and Leeuwen 2018

This paper provides the first instance of deep learning applied to classifying single FRBs. They create a hierarchical DNN with takes in multiple types of data associated with each FRB candidate, specifically, its frequency-time graph, pulse profile, DM-time array, and multibeam detection data. Each of these data "products" are processed by separate ANNs, with slightly different architectures depending on the nature of the data product. The results from these separate ANNs are then fed into a single "hybrid" network which classifies the candidates as FRB or RFI.

They note that the hierarchical network can be adapted to take in different (or fewer) data products

At the time that this paper was written, only a couple of FRBs had been confirmed (Petroff et al. 2016). Thus, in order to create a large enough data-set to make training a DNN feasible, they opted to combine data from Galactic pulsars with simulated FRBs, preferring to use simulated data in the training set.

Data

Two data-sets were constructed, one based on CHIME Pathfinder data, and the other based on Apertif data. The training set based on the CHIME data comprised:

Links to *CHIME* and *Apertif*

4 650 human-inspected false positives that triggered the de-dispersion pipeline with a Signal to Noise ratio (S/N) > 10

4 550 simulated FRB

100 giant pulses from the Crab or single pulses from B0329+54

These numbers are often preceded by "*roughly*" in the paper

The Apertif training set comprised:

10 623 false positives

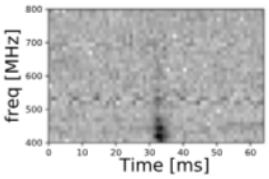
9 800 simulated FRB added to real data

800 single pulses from Galactic pulsars

Architecture

The models were developed using Keras (Chollet et al. 2015) and TensorFlow (Abadi et al. 2015). We briefly describe the data products and their associated ANN below:

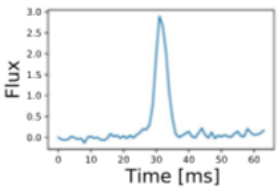
1. *Frequency-time data* - de-dispersed frequency-time graphs were used as inputs to a 2 covolutional layer CNN. An example taken from the paper is show to the right. This CNN's architecture is described below based on the code available on GitHub, which can be found *here*. The model did not appear to use padding, hence the outputs from each $[5 \times 5]$ convolutional layers shrank the width and height by 4.



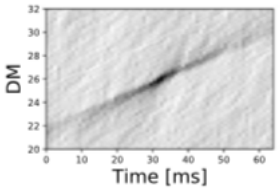
Layer	*Description*	*Output shape*
Input		$32 \times 64 \times 1$
Conv.	$[5 \times 5]$, stride 1, ReLU	$28 \times 60 \times 32$
Max-pool	$[2 \times 2]$, stride 2	$14 \times 30 \times 32$
Dropout	$p = 0.6$*	$14 \times 30 \times 32$
Conv.	$[5 \times 5]$, stride 1, ReLU	$10 \times 26 \times 64$
Max-pool	$[2 \times 2]$, stride 2	$5 \times 13 \times 64$
Dropout	$p = 0.6$	$5 \times 13 \times 64$
Flatten		4160
FC	ReLU	1024
Dropout	$p = 0.5$	1024
FC	Softmax	2

*We use the convention of p in dropout defining the probability that a neuron is kept

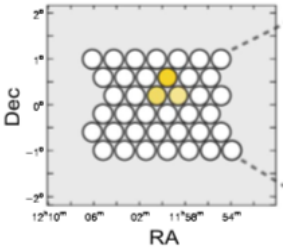
2. *Pulse Profile* - a time series obtained by collapsing the de-dispersed frequency time data along the frequency axis was used as the input to a 1 dimensional CNN, but with an otherwise similar architecture to the 2D CNN used above.



3. *DM–time array* - a DM-time array consists of rows of frequency-time data, collapsed along the frequency axis, for different DMs. FRBs exhibit a 'bow-tie' pattern, as illustrated to the right. A CNN similar to the frequency-time CNN, takes in 100×64 DM-time arrays.



4. *Multibeam Detections* - multiple beams are often used to detect FRBs, so a vector of the S/N for each beam involved in a detection was fed into a feed-forward ANN. Since events of interest are only expected to be detected in at most a couple of adjacent beams, the ANN had to learn to reject detections that came from multiple non-adjacent beams.



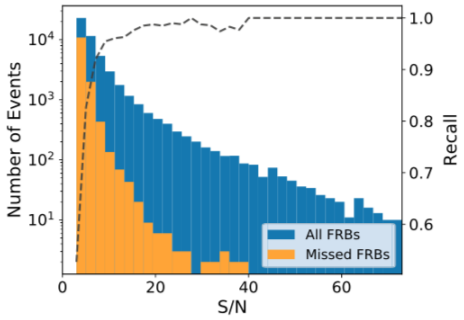
The activations from the final fully connected layers of these 4 ANNs are then concatenated and fed into a feed-forward ANN which classified each observation as FRB or RFI.

Stochastic gradient descent was used to train the hybrid model with a binary cross-entropy loss. For further details on the implmentation, see the source code on *Github*.

Results

In order to measure the performance of their model, Connor and Leeuwen make use of *accuracy*, *recall*, *precision*, and the *F1* score as metrics. See the appendix A.1 if you are unfamiliar with how these metrics are defined.

Precision is an important metric in practice because it relates to the rate at which a model triggers false alarms (RFI) (Zhang et al. 2018b). Recall is also important because it relates to the rate at which FRBs are missed. One of Connor and Leeuwen's findings was a decrease in recall as the S/N decreased. The false-negative rate drops to 0.5 for low S/N, indicating that their model effectively classifies observations with sufficiently low S/N at random.



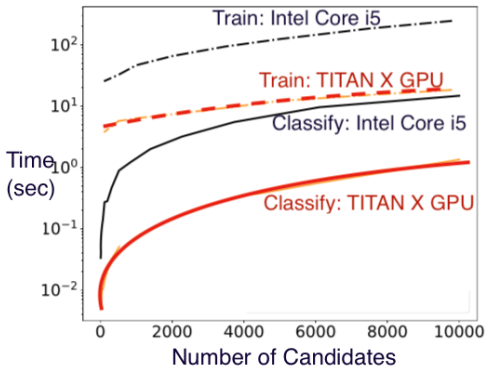
Their hybrid DNN achieved accuracy, precision and recall values surpassing 99%. Based on their described validation procedure, we imagine these values are based on performance on a withheld testing set containing Galactic pulsars, although this is not explicitly stated in the paper.

In order to assess the impacts of training their model on simulated data, they conducted tests where they trained 2 DNNs (1 for CHIME, 1 for Apertif) independently on data-sets of simulated FRBs and hand-labelled false negatives (RFI), and then tested those 2 models on data-sets containing real single pulses from Galactic pulsars. The CHIME model was able to achieve a recall of $\sim 99\%$, and the Apertif model was able to recover 99.7% of the several hundred Galactic pulsar events.

One of the interesting results that comes out of the paper is on the *relative input performance* of the frequency-time, DM-time, pulse profile, and multi-beam data. They also compared this to the performance of the combined inputs. They found that the de-dispersed frequency time data provided the best predictive power, followed by the DM-time array. On the other hand, the 1D pulse profile, which had already been used for classification via the de-dispersion detection algorithm, had a relatively low predictive power. The combined input performance did not significantly outperform the de-dispersed frequency time data, however, it did have an advantage with regard to special cases of false positives such as multi-beam peryton detections (Petroff et al. 2015) and low-DM events that the frequency-time model could easily have classified as FRBs. Additionally, it had an advantage in classifying events that had weak frequency-time signals but had a signal distribution in DM-time space resembling real FRBs.

In order to assess the feasibility of implementing their model in a real FRB detection pipeline, they recorded the training and classification time against the

number of candidates of their model.



They considered performance on both an Nvidia GTX Titan X GPU and a 2.9 GHz Intel Core i5-5287U CPU, and found that classification took $\sim 100\mu s$ per frequency-time array on the GTX GPU, and a few ms per candidate on on the Intel CPU.

$\mu s = 10^{-6}$ of a second.

For Apertif, in situations where, after RFI-mitigation, 10 candidates are considered "interesting" per second and per compound beam - amounting to > 10 million candidates per day - they speculate that their proposed hybrid model running on a single GPU could easily further classify candidates.

They also pursue an interesting discussion on whether DNNs could replace de-dispersion algorithms entirely. In terms of computational complexity, they conclude that, although not as efficient as optimized de-dispersion algorithms*, their classification model has lower computational complexity than most brute-force searches. They further note that accelerators, such as GPUs, might give DNNs further real life advantages. Ultimately, they argue that replacing de-dispersion with deep learning will only be of value where there is significantly high S/N, and that existing de-dispersion algorithms are either optimal or near-optimal in signal recovery.

*such as subband or tree de-dispersion

Fast Radio Burst 121102 Pulse Detection and Periodicity: A Machine Learning Approach, Zhang et al. 2018b

Zhang et al. 2018b present the first successful application of deep learning to *direct* detection of FRBs from raw spectrogram data, reporting the detection of

72 new pulses from the repeating FRB 121102. These detections were made by reanalysing the C-band observation by Breakthrough Listen using a CNN, specifically a variation of the wide residual network proposed by Zagoruyko and Komodakis 2016. Using their new sample of detections, they explored trends in the pulse fluence, arrival times, and frequency structure.

Data

The data for the training and test sets came from 5 hours of Breakthrough Listen observations with no detected pulses. The filterbank data produced by the Breakthrough Listen digital backend are 2D spectrograms that span 4–8 GHz with a frequency resolution of 366kHz and a sampling time of 0.350 ms (MacMahon et al. 2018). From the filterbank data, individual frames were created by making cuts along the time axis every 256 samples, which roughly captured the dispersed duration of the previously observed pulses of FRB 121102 (Gajjar et al. 2018). The frames were then normalised by subtracting the mean and dividing by the standard deviation per channel per frame. The choice of *instance normalisation* as opposed to batch normalisation was made to reduce the influence of occasional very bright RFI.

$4 \times 10^6/366 \approx 10929$ frequency channels

From these 5 hours of raw spectrogram data, around 400 000 frames were obtained. Then, simulated pulses with a range of dispersion measures, bandwidths, pulse widths and amplitudes were injected onto half of the frames. Of the 5 hours, 4 and a half hours were used for the training set and half an hour was reserved for the testing set.

Architecture

Source code can be found at this Github *repository*

Modifying the standard wide residual network (Zagoruyko and Komodakis 2016), they used an initial convolutional layer with a $[32 \times 1]$ filter/kernel with stride 32, reducing the input complexity and increasing the S/N by collapsing every 32 frequency channels. This layer was followed by a convolutional *block* that used $[7 \times 7]$ filters. Throughout the network, they made use of strided convolutions, as opposed to max-pooling layers, to reduce the size of the output volumes. These large convolutional kernels and wide strides in the initial layers

were chosen to reduce the data rate since each pixel in the input spectrogram did not contain independent useful information.

The convolutional block used throughout the architecture consists of 2 sub-blocks - [BN $\rightarrow$ ReLU $\rightarrow$ Conv. layer] - separated by a dropout layer. Below, we describe the filters in each convolutional block and the number of repetitions of each block. The number of filters in each block can be deduced from the output shape. Overall, there are 17 convolutional layers.

See He et al. 2016 and Zagoruyko and Komodakis 2016 for more information on the blocks

Layer/Block	*Description*	*Output shape*
Input		$10929 \times 256 \times 1$
Conv. layer	$[32 \times 1]$	$342 \times 256 \times 1$
Conv.	$[7 \times 7]$	$171 \times 128 \times 32$
2$\times$Conv.	$[3 \times 3]$	$42 \times 32 \times 32$
3$\times$Conv.	$[3 \times 3]$	$10 \times 8 \times 64$
2$\times$Conv.	$[3 \times 3]$	$5 \times 4 \times 128$
Gl.Avg-pool		1×128
FC		2

Here, Conv. denotes a convolutional block. Only the first convolutional layer is a single layer

One of the motivations behind using a residual network is its use of "skip connections" where inputs into every few stacked layers are given shortcuts over those layers. Formally, the building blocks of residual networks are:

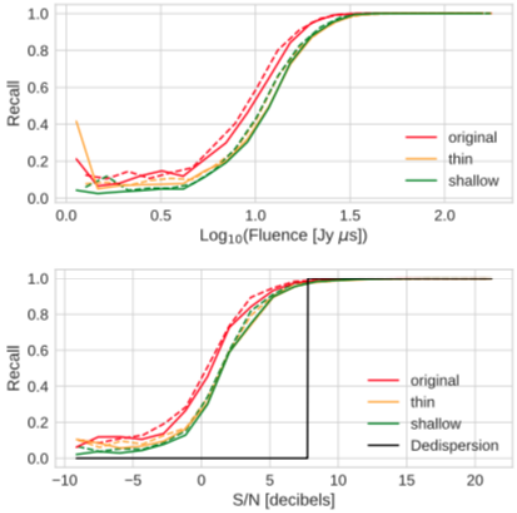
$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{x}, \{\boldsymbol{W}\}_i) + \boldsymbol{x}$$

where $\boldsymbol{x}$ are the inputs to the stack of layers, $\{\boldsymbol{W}\}_i$ are the weights associated with the stack of layers and $\mathcal{F}$ represents the overall transformation of the stack of layers, called a "residual mapping" in the original paper (He et al. 2016). In this context, the choice to use residual networks was motivated by the fact that FRBs have a relatively simple, low-level quadratic form, but are also surrounded by high pixel noise. Zhang et al. claim that the skip connections encourage propagation of low-level features such as a quadratic burst much deeper into the network.

Results

Their model was trained for 20 hours on a Nvidia Titan Xp GPU and achieved an accuracy of 93%, a recall of 88% and a precision of 98% on the withheld

test set. Following on from Connor and Leeuwen 2018 where the recall was observed to suffer for FRB with low S/N, Zhang et al. investigated the relationship between the simulated FRB fluence and S/N and the model recall scores.



Dashed lines - training recall
Solid lines - test recall

The model recall scores were recorded for both their original model, as well as thinner (half the units per layer) and shallower (half the layers) variants. From these plots, one can also observe that the test and training recall scores are very close, indicating relatively little over-fitting.

As pointed out by Connor and Leeuwen 2018, inference speed is an important consideration in practical applications. This model was capable of processing around 800 images* per second on a Nvidia GTX 1080 GPU.

equivalent to 70s of observation time

FETCH: A deep-learning based classifier for fast transient classification Agarwal et al. 2020

This paper makes use of *transfer learning* to classify single pulses based on their frequency-time graphs and their DM-time arrays. *Transfer learning* involves re-purposing a pre-trained model to a new task. In this context, various, pre-trained, state-of-the-art CNNs were re-purposed to classify single pulses by connecting a feed-forward neural network to the end of the pre-trained models that maps their learnt representations of the images to the binary classification FRB or RFI.

Data

Two types of data were used as input to their model: the de-dispersed frequency-time graphs, and the DM-time arrays. These data were either obtained from the Green Bank Telescope (GBT) and the 20m telescope, both located at the Green Bank Observatory, or were simulated and injected into data taken with GREENBURST (Surnis et al. 2019) or FLAG (Rajwade et al. 2019).

The GBT data were recorded by 2 different back-ends - the GREENBURST back-end, which used the *L-band* receiver of the GBT, and the FLAG back-end which used the Focal plane L-band Array for the Green Bank Telescope (FLAG) receiver.

The 20m telescope data were also recorded by 2 different back-ends - the Skynet (Langston et al. 2013; Smith, Caton and Hawkins 2016) and the GBTrans (Golpayegani and Lorimer 2019) back-end. The composition of the training, validation and test sets are shown in table 1.1.

Instrument (backends)	Source	T+V DM-T	T+V F-T	Test
FLAG	RFI	32 720	6 000	2 790
(FLAG)	Sim. FRB	20 000	8 500	-
	Pulsar	-	-	2 288
GBT L-Band	RFI	-	6 000	2 170
(GREENBURST)	Sim. FRB	20 000	8 500	-
	Pulsar	-	-	1 376
Green Bank 20m (Skynet)	RFI	9 854	8 000	2 359
(GBTrans)	Pulsar	-	3 000	3 000
Total	FRB*	40 000	20 000	6 664
	RFI	42 574	20 000	7 319

T+V: Train + Validation; DM-T: Dispersion Measure - Time; F-T: Frequency Time. Train-Validation split: 85%(Train), 15%(Validation). FRB*: technically simulated FRB and pulsars

Table 1.1: Training, validation and testing set data sources and composition, taken from Agarwal et al. 2020.

In addition to the simulated FRBs, the RFI candidates were augmented in order to double their numbers (the numbers in table 1.1 include the augmented RFI). The augmentation techniques they used were a *flip along the time-axis* of the de-dispersed frequency time data, and a *flip along both the time and DM axes* of the DM-time arrays.

The input data were standardised so that they had the same size and shape. In the case of the de-dispersed frequency-time data, this involved binning the time axis so that the pulse profile lay between 1-4 bins of the origin, and then re-sizing the images to 256 pixels by averaging the frequency axis and trimming out the extra pixels.

In the case of the DM-time data, the time axis was again binned and cropped and the DM axis consisted 256 values ranging from 0 to 2 times the DM of the candidate.

Additionally, the frequency-time and DM-time data were scaled to have zero median, and unit standard deviation. It is not mentioned whether this was done though batch or instance normalisation.

Architecture

The models were developed using Keras (Chollet et al. 2015), and TensorFlow (Abadi et al. 2015). The pre-trained models that were considered were:

See *here* for documentation of the pre-trained models.

- Xception, Chollet 2017
- VGG16, VGG19, Simonyan and Zisserman 2014
- ResNet50, He et al. 2016
- DenseNet121, DenseNet169, DenseNet201, Huang et al. 2017
- InceptionV3, Szegedy et al. 2016
- InceptionResNetV2, Szegedy et al. 2017
- MobileNet, Howard et al. 2017
- MobileNetV2, Sandler et al. 2018

The DM-time and frequency time data were fed into two separate pre-trained models. Attached to the front of each of these pre-trained models was a convolutional layer consisting of 3, $[2 \times 2]$ convolutional filters. The output of this convolutional layer was a volume with 3 depth channels, courtesy of the 3 filters

in the convolutional layer, making the initial 1-channel, grey-scale data compatible with the expected 3 channel, RGB inputs of the pre-trained models.

Appended to the end of each pre-trained model, in place of the usual classification layer was a 2 unit fully-connected layer followed by the softmax function in order to output a probability for each observation.

These two models were initially trained separately, and then combined into a hybrid model using a multiplicative fusion approach (Park et al. 2016). This involved replacing the top classification layer of the model corresponding to the frequency-time data and the model corresponding to the DM-time data with classification layers of k-units each. The element-wise product of these 2 classification layers was used to combine them into a single k-unit layer. The output from this layer was then fed into a 2 unit, fully-connected layer, which was again followed by the soft-max function to obtain probabilities.

Adam (Kingma and Ba 2014) was used as the optimizer, and *binary cross-entropy* was used as the loss-function. As additional regularization, Gaussian noise with zero mean and unit standard deviation was injected into the input data at each epoch, which has been show to help prevent over-fitting (Jiang et al. 2009).

Results

All the models listed under the *architecture* section were trained and compared using their validation accuracy on the DM-time and frequency-time data-sets. For each of the 2 data-sets, the top 5 models were identified from which 25 pairs of models were formed and assessed in terms of their accuracy, recall, and F1 score on test data. Different values of the hyper-parameter k were also considered. Finally, a criteria of *accuracy, recall and F1 score* > 99.5% was used to filter the models. For models with different k-values, the model with the highest F1 score was kept. This resulted in 11 models, of which the top model achieved an accuracy of 99.8%, recall of 99.92% and F1 score of 99.87%.

k - size of the fully connected layer
Metrics explained in A.1

Top model: DenseNet121 + Xception

The time it took their models to classify candidates was measured on a NVIDIA GTX–1070 and a NVIDIA Titan–Xp and had mean values of 12 ± 1 ms and 6.7 ± 0.9 ms respectively. Based on a conservative estimate of 20 ms per candidate, they estimated that their models could work in real time for candidate rates $< 10^8$ per hour.

RESULTS

2.1. Data

2.1.1. Context

The data-set is composed of real, labelled data obtained as part of the MeerTRAP project (Sanidas et al. 2017) to commensally search for new fast radio transients including FRBs and radio emitting neutron stars such as pulsars and Rotating Radio Transients (RRAT). The data-set consists of 10944 observations of known pulsars, and 10371 observations of RFIs. The pulsar data were obtained from occasions when the MeerTRAP project "piggybacked" on the pulsar timing project, MeerTIME (Bailes et al. 2016), which observes known pulsars to study their emission and rotational behaviour. Some of these pulsars are bright enough that we can detect single pulses from them and thus can be used as surrogates for FRB pulses which have broadly similar characteristics. The RFIs correspond to data taken at the same time but is what is seen in those beams that were not looking* at the known pulsars.

*MeerTRAP can look simultaneously in up to 800 different directions

2.1.2. Data processing

The models were trained on de-dispersed frequency-time graphs, which are cropped out of larger images that consist of 4 graphs. We provide an example of one of the images from the data-set in figure 2.1.

There were 3 subtly different positions that the de-dispersed graphs occupied in the images which changed along with the overall dimensions of the image.

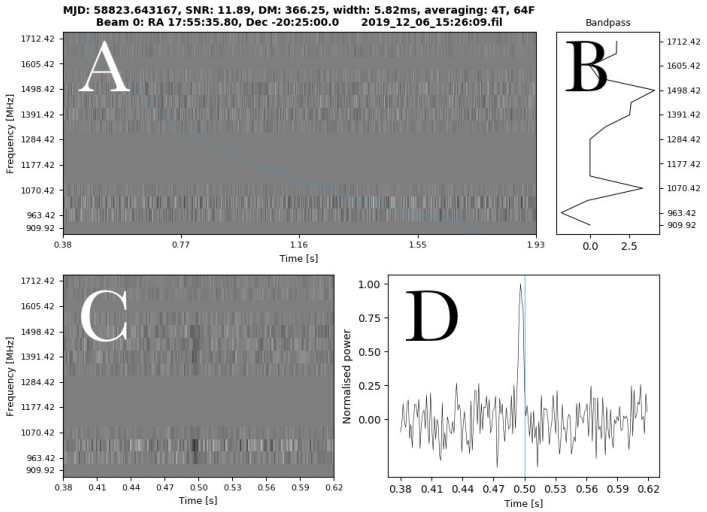


Figure 2.1: A: Frequency-time graph B: Frequency bands collapsed along time axis, C: De-dispersed frequency-time graph, D: Time series - de-dispersed frequency-time graph collapsed along frequency axis and normalised

It was not a completely trivial process to arrive at these findings.

First, the dimensions of each of the images in the data-set had to be determined. Once they were determined, a bounding box needed to be defined that would neatly crop out the bottom left de-dispersed frequency time graph. We made the assumption that within images of the same dimensions, the graphs would appear in the same place, even if they did vary between images with different dimensions. This assumption was manually checked for over 100 different observations, randomly selected from the data-set. Of course, this still does not mean the assumption is correct, but one might as well re-classify the data-set if one intends on going through the whole data-set. The dimensions of the images and how we chose to crop them are documented below.

Code for determining the dimensions of the images and visualising different crops can be found at this GitHub *address*

Image shape	*Crop specifications*
height, width	offset height, offset width, target height, target width
689, 944	369, 86, 268, 359
689, 943	369, 86, 268, 359
631, 922	337, 83, 245, 351

This way of specifying the crop is in line with the TensorFlow *crop_to_bounding_box* function

Overall, given the file name of an image, the following process was applied:

1. The image was read from jpeg format into a TensorFlow *Tensor*
2. The pixel values were scaled from 8-bit unsigned integers in the range $[0, 255]$ to 32-bit single-precision floating-points in the range $[0, 1]$. For certain applications we subtracted the mean pixel value, calculated over the training data-set from each image's pixels and scaled the pixels by the standard deviation of all the training image pixels.

   Training data
   mean: 0.54
   standard deviation: 0.15
3. The de-dispersed frequency-time graph was *cropped out* of the Tensor depending on the dimensions of the Tensor
4. The cropped graphs were converted to *greyscale*
5. By default, the cropped, greyscale graphs were resized to have height and width equal to 224. The final dimensions of each de-dispersed frequency-time graph were $(224, 224, 1)$. We will indicate the cases where other dimensions were considered.

One of the things to be wary of is that due to the way in which the graphs were saved as images, one piece of information is stored over multiple pixels. Thus, there are a lot of redundant pixels capturing the same information.

2.1.3. Training, Validation and Test Sets

Once all the file-paths to the pulsar and RFI observations have been saved to a data-set; training, validation and test data-sets are obtained by taking disjoint sets of the file-paths, and then processing those file-paths to obtain data-sets of de-dispersed frequency-time Tensors. The composition of these data-sets are shown in the following table:

The code used to create the data-sets can be found at the following GitHub *address*

Data-set	*% of total (approx.)*	*Count*
Training set	70%	14920
Validation set	15%	3197
Test set	15%	3198

It is clearly advantageous to randomise the order in which the data is presented to the model, but also necessary to do this randomisation in a way that can

be reproduced. We introduce randomisation when creating a list of all the file paths, using the a pseudo-random process with a defined seed. For this, we use the Python random package (Van Rossum 2020).

2.2. Implementations

2.2.1. Baseline model

We start by establishing a *baseline model* based on the architecture used by Connor and Leeuwen 2018 to process the de-dispersed frequency time data. Our baseline model was used as a component of in their hybrid model, though it was the component found to have the best predictive power. This, along with the different data-set, make direct comparisons unfair. This model comprises:

All the code for this baseline model is made available in the *Colab notebook*

Layer	*Description*	*Output shape*
Input		$d \times d \times 1$
Conv.	$[5 \times 5]$, stride 1, ReLU	$d \times d \times 32$
Max-pool	$[2 \times 2]$, stride 2	$d/2 \times d/2 \times 32$
Dropout	$p = 0.6$	$d/2 \times d/2 \times 32$
Conv.	$[5 \times 5]$, stride 1, ReLU	$d/2 \times d/2 \times 64$
Max-pool	$[2 \times 2]$, stride 2	$d/4 \times d/4 \times 64$
Dropout	$p = 0.6$	$d/4 \times d/4 \times 64$
Flatten		4160
FC	ReLU	1024
Dropout	$p = 0.5$	1024
FC	Sigmoid	1

d is the width and height of the image input into the model, assuming square images

We have modified the final layer to have a single unit with a sigmoidal activation function as opposed to their 2 unit softmax layer, and we explicitly use padding on the convolutional layers so that the image dimensions are only reduced on the pooling layers.

Connor and Leeuwen 2018 found that spatial dimensions of 32×64 allowed their images sufficient signal per pixel. Given that we use a different data-set, we have decided to fit the model on images resized to 56×56, 112×112

and 224×224 pixels and record various performance metrics of models trained on each image size. Depending on the size of the input, the model had the following number of parameters:

Input spatial dimensions	56×56	112×112	224×224
Number of model parameters	12 899 201	51 434 369	205 575 041

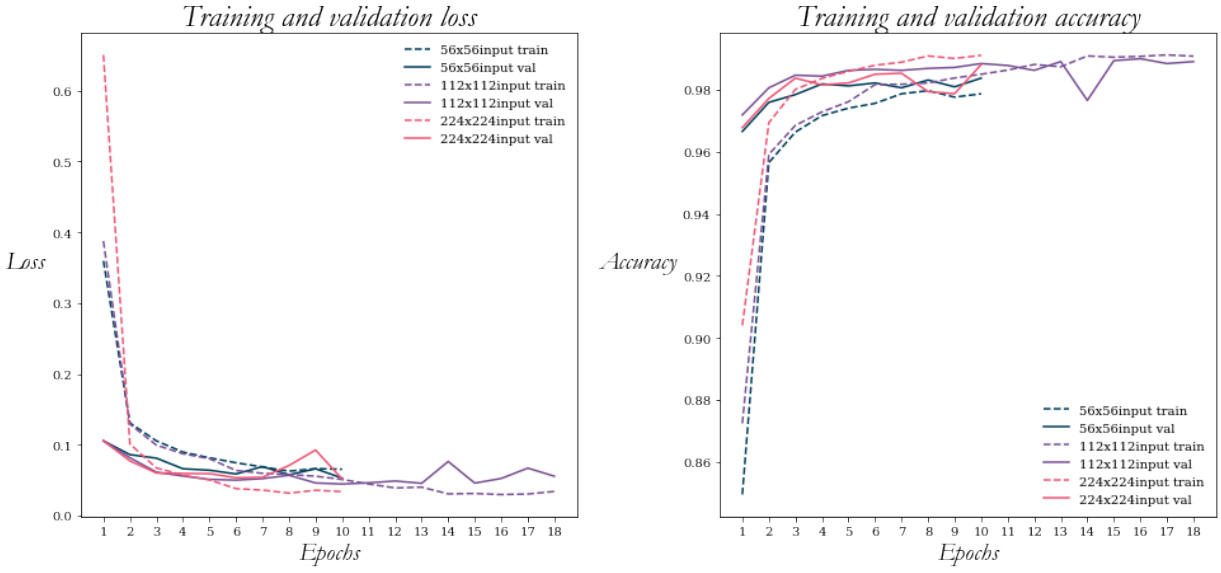
The majority of these parameters are introduced in the 1024 unit fully connected layer.

A model was trained for 20 *epochs** on each of the 3 data-sets and its training and validation *accuracy* and *loss* was recorded at the end of each epoch. Binary cross-entropy was used as the loss function, and the training data-set comprised batches of 32 observations. Adam (Kingma and Ba 2014) was used as the optimizer.

*full passes through the training data set

Adam parameters:
Learning rate: 0.001
1^{st} and 2^{nd} moment decay rates:
$\beta_1 = 0.9$, $\beta_2 = 0.999$
ϵ: 10^{-7}

We present the training and validation trajectories below. Solid lines are used to indicate performance on the validation data-set, while dashed lines are used to indicate performance on the training data-set.



As can be seen from all 6 lines, which stop short of the full 20 epochs, *early-stopping* with a "patience" of 2 epochs was used. Early-stopping is a technique where training terminates if the model fails to improve by a certain margin within a period of time. In this case, if the loss failed to decrease within 2 epochs, the training procedure terminated. Due to an unintentional bug in the

code, the training loss triggered termination. Ideally, early stopping is based on some validation metric. Fortunately, we also saved the weights of the model with the lowest validation loss for each of the tests we conducted; the only real loss was time lost on training models beyond the point where their validation loss stopped decreasing.

All of the trajectories jump up fairly rapidly to accuracy $> 97\%$. Interestingly, the model trained on the smallest data-set appears to perform better on the validation set than on the training set. Typically, we see models over-fitting to the training data, indicated by better accuracy on the training data than the validation data, as can be seen with the 224×224 model.

We also evaluate the model on a withheld test data-set:

We underline the best score for each metric

Model	56×56	112×112	224×224
Loss	0.0548	<u>0.0456</u>	0.0533
Accuracy	0.9797	<u>0.9894</u>	0.9828
Precision	0.9688	<u>0.9854</u>	0.9769
Recall	0.9920	<u>0.9938</u>	0.9895

Based on these results, there is clearly flexibility with regard to how the inputs are resized, but the model trained on data resized to 112×112 seems to show marginally better performance.

Overall, this baseline model would appear to be a good benchmark for further models. It is hard to distinguish which aspects of the model make it successful. It uses wider filter dimensions, $[5 \times 5]$, than the usual $[3 \times 3]$ filters in its convolutional layers. Using wider kernels in the earlier layer is something that Zhang et al. 2018b also recommended. This baseline model also has a relatively large fully-connected layer as its second last layer. For each of the input sizes, the number of parameters in the fully-connected layer is $1024(d/4 \times d/4 \times 64) + 1024$*. Regularisation in the form of dropout was also used - another aspect that may contribute to this model's strong performance.

*Even in the case $d = 56$, the number of parameters will be 12 846 080

2.2.2. *Varying Model Depth*

All the code for this experiment is available in *this Colab notebook*

We experiment with models of different depths. We take inspiration from VGG Net (Simonyan and Zisserman 2014) which is composed of blocks consisting of stacks of convolutional layers with $[3 \times 3]$ filters and ReLU activations, followed by a max pooling layer. By stacking 2 $[3 \times 3]$ convolutional layers, we gain an effectively larger receptive field, i.e. $[5 \times 5]$, but are able to introduce non-linearities between the layers (Simonyan and Zisserman 2014). We also make use of global average pooling on the second last layer, as opposed to flattening the output and then using fully connected layers. Zhang et al. 2018b also made use of global average pooling in their FRB model, though this might not be so much an intentional design decision as something inherited from the wide residual network architecture (Zagoruyko and Komodakis 2016). Global average pooling as a means to reduce model complexity and encourage better convolutional filters is discussed in Lin, Chen and Yan 2013.

We describe the model template below, noting that the blocks - [Conv. $\rightarrow$ Conv. $\rightarrow$ Max-pool] - will be repeated up to D times:

Layer	*Description*	*Output shape*
Input		$224 \times 224 \times 1$
For $d \in \{1, ...D\}$ repeat:		
Conv.	$[3 \times 3]$, stride 1, ReLU	$224/2^{d-1} \times 224/2^{d-1} \times 8 \cdot 2^d$
Conv.	$[3 \times 3]$, stride 1, ReLU	$224/2^{d-1} \times 224/2^{d-1} \times 8 \cdot 2^d$
Max-pool*	$[2 \times 2]$, stride 2	$224/2^d \times 224/2^d \times 8 \cdot 2^d$
End of repeated blocks		
Gl.Avg-pool		$8 \cdot 2^d$
FC	Sigmoid	1

*Max-pool is dropped on last iteration $d = D$

The number of units in each layer can be deduced from the output shape but, simply, the number of units used in each convolutional block followed the pattern $16, 32, 64, ...$

We create models of different depths by using different numbers of repetitions of the blocks, $D \in \{2, 3, 4, 5\}$. Each of these models we label $2Dconv$ since for each given D, the model has $2D$ convolutional layers. We document the number of learnable parameters for each of these models and, for comparison, we

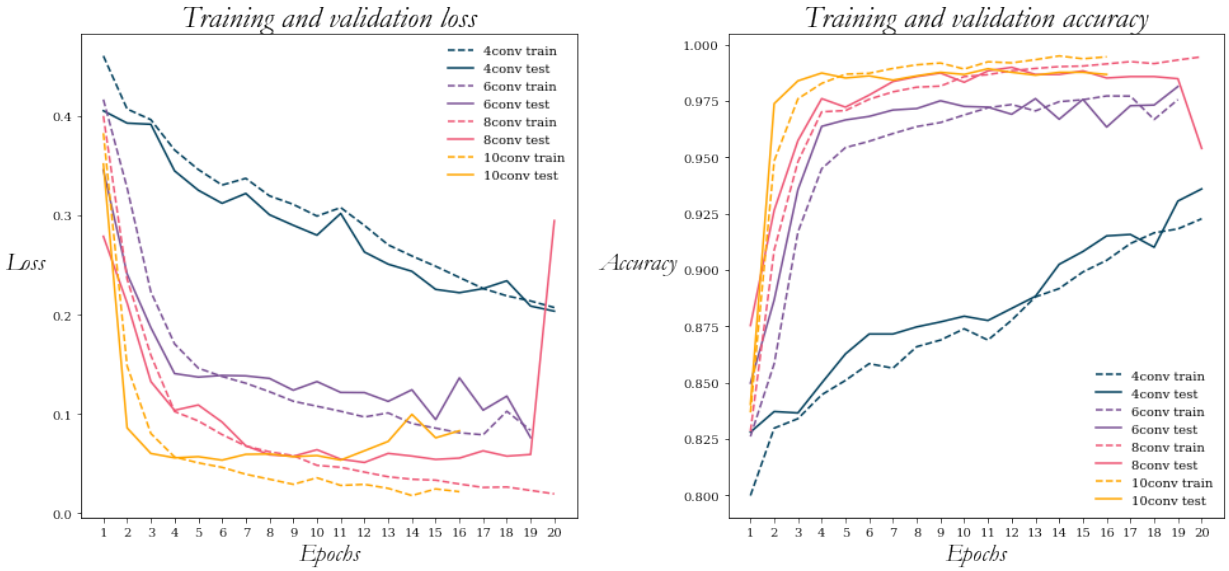
also list the number of parameters for ResNet50 (He et al. 2016), DenseNet121 (Huang et al. 2017) and Xception (Chollet 2017) based on their implementations in *Keras Applications* (Chollet et al. 2015). ResNet50, DenseNet121 and Xception were all models considered by Agarwal et al. 2020*, and their top performing model made use of DenseNet121 and Xception to process the frequency-time and DM-time data (in that order).

*Furthermore, they made use of the pre-trained models from Keras Applications

Model	4conv	6conv	8conv	10conv	ResNet50	DenseNet	Xception
No. params.	16 401	71 857	293 361	1 178 737	25 636 712	8 062 504	22 910 480

As before, each of the $2D$conv models were trained for a maximum of 20 epochs, with early-stopping in place. Binary cross-entropy was used for the loss function, and again Adam (Kingma and Ba 2014) was used as the optimizer.

We present the training and validation trajectories below. As before, solid lines represent the validation trajectory and dashed lines represent the training trajectory. The validation lines are misleadingly labelled "test" in the legend.



The smallest model, shown in dark blue, appears to not converge within the 20 epochs and merits further training.

With the exception of the smallest model, 4conv, as training progressed the validation loss ended up higher than the training loss. Similarly the validation accuracy ended up lower than the training accuracy. This is evidence that the larger models over-fit to the training data-set. As model size increases, it is cap-

able of modelling more intricate patterns in the data, but also of memorising the training data to a larger extent (Arpit et al. 2017). We observe that the overall loss is lower as the model depth increases, but that the difference between the training and validation performance also widens as the model depth increases. Limiting the complexity of a model is one way of reducing over-fitting, but one has to simultaneously ensure it is complex enough to model the data.

We tested each of these models on the withheld test set* and recorded the accuracy, loss, precision and recall. These scores are reported below:

*different to the validation set, despite erroneous plot legends

Model	4conv	6conv	8conv	10conv
Loss	0.2054	0.0728	0.0475	0.0501
Accuracy	0.9362	0.9794	0.9866	0.9859
Precision	0.9351	0.9762	0.9859	0.9811
Recall	0.9397	0.9834	0.9877	0.9914

Best baseline metrics
Loss: 0.0456
Accuracy: 0.9894
Precision: 0.9854
Recall: 0.9938

These scores, particularly for the 8 and 10 conv architecture are not as high as the $> 99\%$ scores presented by (Agarwal et al. 2020) and (Connor and Leeuwen 2018). However, this still shows relatively strong performance by models with significantly fewer parameters, and trained on purely de-dispersed frequency time data. Both Agarwal et al. 2020 and Connor and Leeuwen 2018 used additional forms of data. The performance of the 8 and 10 convolutional layer models is close to that of our baseline model. Since the baseline model made use of dropout as a form of regularisation during training, our next step will be to investigate the impacts of regularisation.

In the *Colaboratory notebook*, the misclassified observations from the test set are presented along with raw model predictions $\in [0, 1]$. The activations of each layer are also plotted for different instances. These results will remain in the notebook due to their considerable size. Looking through the plots helps give one a sense of what the effects of each layer of transformations are.

2.2.3. *On the Effects of Regularisation*

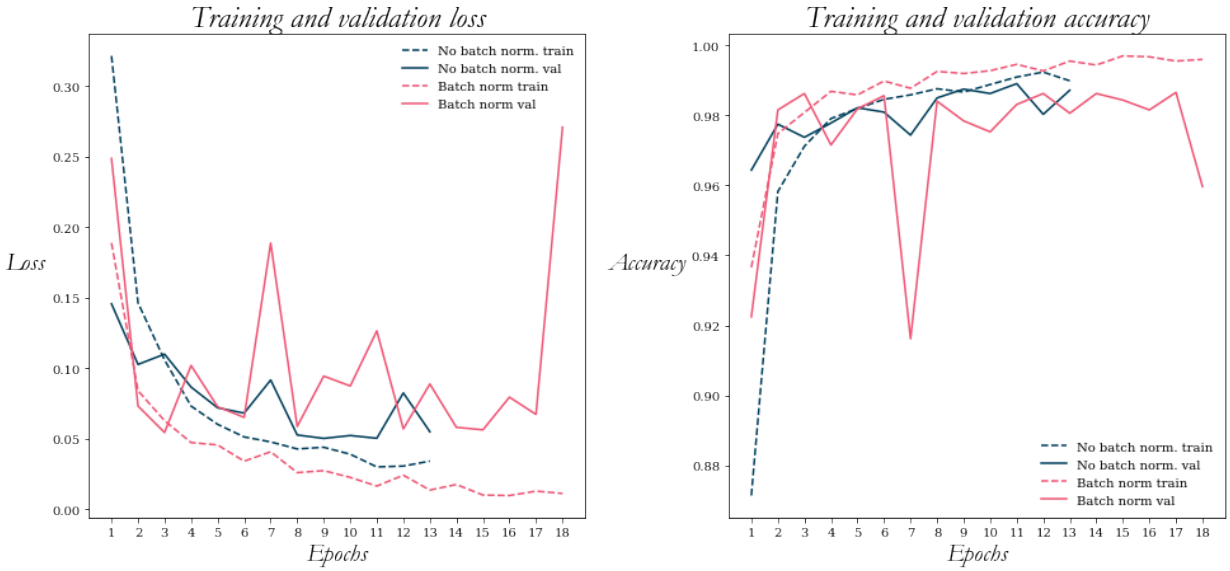
The code to these experiments is available in *this Colab notebook*

We now introduce regularisation techniques to our models in the hopes of improving their performance. At this stage, we only consider the 8 convolutional layer model, which showed promising performance above. We modify the blocks of the architecture to optionally include batch normalisation, dropout or both. The modified blocks now have the form:

$$[\text{Conv.} \rightarrow \text{BN (?)} \rightarrow \text{ReLU} \rightarrow \text{Dropout(?)}] \times 2 \rightarrow \text{Max-pool}$$

This placement of the batch normalisation and dropout layers are based on the recommendations of Cai et al. 2019. For dropout, we sought to test the differences between *channel* and *neuron* level dropout. We also sought to test different values of p - the probability a channel or neuron is kept during dropout.

We start by training the model with and without batch normalisation. At this stage, we do not include dropout. As before, the model was trained for a maximum of 20 epochs using binary cross-entropy as the loss, Adam (Kingma and Ba 2014) as the optimizer and with early-stopping in place.



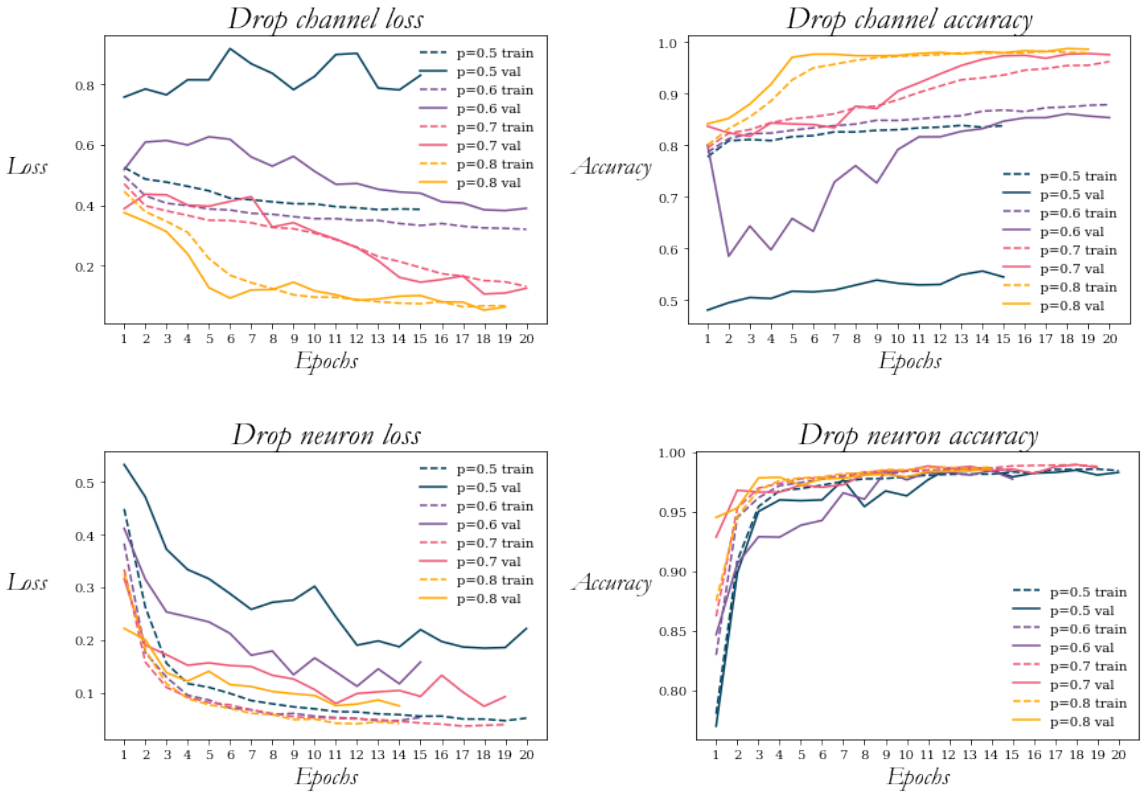
Batch normalisation appears to improve the loss and accuracy on the training sets, however at the cost of stability and performance on the validation sets.

The performance of the model with and without batch normalisation is very similar on the withheld test data:

The model weights associated with the lowest validation loss were used here.

	loss	*accuracy*	*precision*	*recall*
With batch normalisation	0.0508	0.9850	0.9799	<u>0.9908</u>
Without batch normalisation	<u>0.0441</u>	<u>0.9853</u>	<u>0.9858</u>	0.9852

We move on to considering the impacts of dropout on the model's performance. We consider both drop-channel and tradition dropout which we call *drop-neuron*. For each type of dropout, we considered $p \in \{0.5, 0.6, 0.7, 0.8\}$. The training and validation accuracy and loss trajectories are presented below.



The performance of the models that use drop-channel with high probabilities of dropping channels ($p < 0.7$) is poor, particularly on the validation data-set. The drop-channel loss and accuracy trajectories look particularly wild. If there is hope for drop-channel, it is with $p = 0.8$. For $p = 0.8$ the training and validation trajectories are very close indicating good general performance.

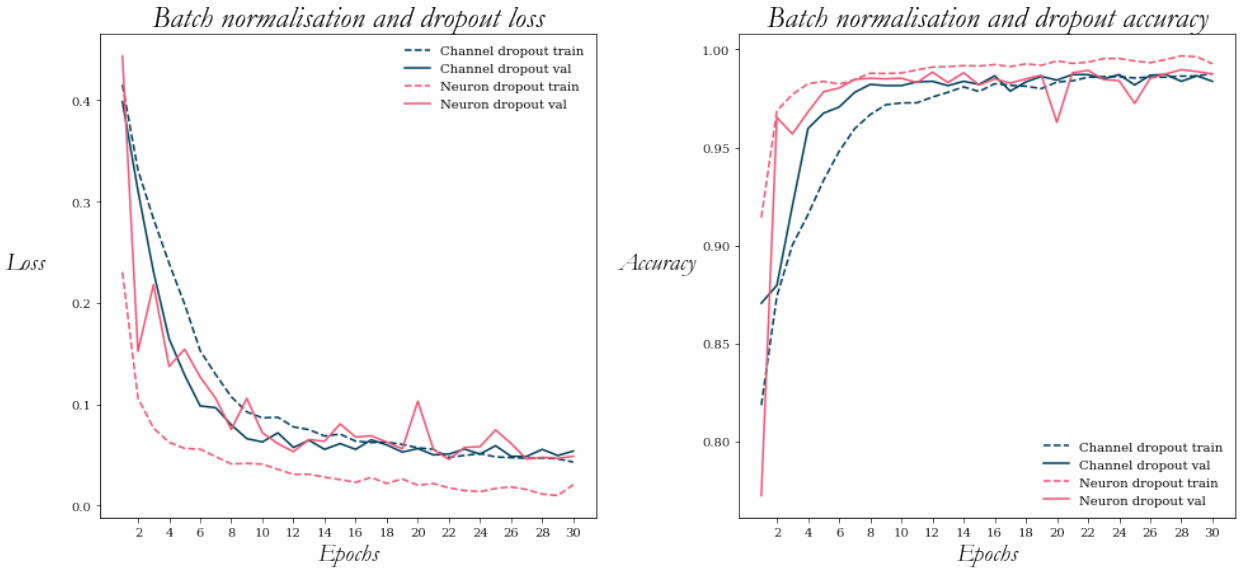
Drop-neuron also suffers from bad performance for small values of p, though

not as bad as drop-channel. The training performance is consistently better than the validation performance indicating it does not entirely get rid of over-fitting.

Again, we present the performance on the withheld test data:

	loss	*accuracy*	*precision*	*recall*
Drop channel: $p = 0.5$	0.7489	0.4956	<u>1.0000</u>	0.0080
$p = 0.6$	0.3826	0.8508	0.8843	0.8130
$p = 0.7$	0.1016	0.9784	0.9791	0.9785
$p = 0.8$	<u>0.0466</u>	0.9866	0.9853	<u>0.9883</u>
Drop neuron: $p = 0.5$	0.1777	0.9853	0.9882	0.9828
$p = 0.6$	0.1099	0.9825	0.9900	0.9754
$p = 0.7$	0.0670	<u>0.9894</u>	0.9908	<u>0.9883</u>
$p = 0.8$	0.0649	0.9878	0.9895	0.9865

We now consider whether employing both drop-out and batch normalisation result in any further improvements. We consider high values of p, $p = 0.8$, and consider both channel and neuron based dropout. This time we consider 30 epochs as opposed to 20.



The interesting result here is that there is a noticeable difference between the training and validation trajectories for drop-neuron, and a relative lack of difference in trajectories for drop-channel. There is evidence that drop channel helps the model generalise better, and that drop-neuron does not entirely eliminate over-fitting.

The performance on the withheld test set is similar, although drop channel appears to do marginally better. Out of interest, we also regularised our 10 convolutional layer model with batch normalisation and drop-channel with $p = 0.8$. The test performances of these three models are presented below:

	loss	*accuracy*	*precision*	*recall*
Drop channel & BN	0.0398	0.9881	0.9853	0.9914
Drop neuron & BN	0.0430	0.9869	0.9895	0.9846
10conv with drop channel & BN	0.0391	0.9872	0.9823	0.9926

The training and validation trajectories of the 10conv model are included at the end of this *notebook*.

We further explored using Bayesian optimisation as a means of fine-tuning the dropout rates of each dropout layer in the model. We were able to explore 39 different configurations before exceeding the usage limit for GPU on the Colaboratory platform. From the 39 configurations that were explored, configurations with relatively low dropout rates, corresponding to high values of p, $p = 0.95$, performed relatively well. However, configurations that included relatively high levels of dropout on just the beginning or end of the model with relatively low dropout rates otherwise also performed relatively well. We refitted models with dropout rates inspired by these configurations but none of them were able to surpass performance already documented.

Due to the untimely termination of the hyper-parameter search we have not done a thorough write-up on these results. For the interested reader, these results are available *here*.

2.2.4. *Final Rankings*

We summarise the performance of the top 5 models based on test accuracy. All of these metrics were observed on the withheld test data:

Name	*Loss*	*Accuracy*	*Precision*	*Recall*	*No. params*	*Training epochs*
112×112 baseline	0.0456	0.9894	0.9854	0.9938	51 434 369	20
Drop neuron $p = 0.7$	0.0670	0.9894	0.9908	0.9883	293 361	20
Drop channel $p = 0.8$ & BN	0.0398	0.9881	0.9853	0.9914	294 321	30
Drop channel $p = 0.8$	0.0466	0.9866	0.9853	0.9883	293 361	20
8conv	0.0475	0.9866	0.9859	0.9877	293 361	20

These final rankings are by no means the final say on which model to choose, particularly because one might be interested in ranking the models based on the other recorded metrics. However, one might favour the simplicity of the 8 convolutional model in comparison to the baseline model. In all the test performances we have presented we have endeavoured to underline the top performance for each metric. One model seldom scores the highest performance in all of the metrics considered.

One of the shortcomings of these tests are their lack of repetitions. Exploring average performance would be more insightful than the instances we have presented. However, training these models on this ≈ 20000 observation data-set required long periods of time and serious computational resources. All of these tests were made possible by GPU made available on the Google Colaboratory platform, and even with GPU, some of these tests succeeded the platform's usage limits. Future work will be focused on learning better *high performance computing* practices.

Conclusion

In this project, we provided brief glimpses into the worlds of convolutional neural networks and fast radio bursts, highlighting the intrigue of fasts radio bursts and their potential to better understand the universe, along with the potential for convolutional neural networks to help discover these elusive bursts.

We briefly traced the history of modern CNN architectures and elaborated on their components, illustrating these components in code*. We reviewed essential techniques for training CNNs: *normalisation* - how to process the inputs into and through deep neural networks, *hyper-parameter optimisation* - how to configure networks and *regularisation* - how to improve the ability of a network to generalise to unseen data. Within this review, we picked up on recent research into the potentially conflicting and misunderstood roles of *batch normalisation* and *dropout* in CNNs. We then summarised recent work on applying CNNs to FRB classification, identifying the types of data that were used, the architectures of the models used and the results obtained.

*Made available via *GitHub*

Based on the architecture used by Connor and Leeuwen 2018, we implemented a baseline model for classifying FRBs based on their de-dispersed frequency-time graphs and considered the performance of the baseline model on different 'resizings' of the graphs. As our own practical contribution to classifying fast radio bursts, we created convolutional neural networks of varying depths, and applied them to a real labelled data-set of pulsars and radio frequency interference obtained from the MeerTRAP project (Sanidas et al. 2017). We applied different regularisation techniques to the models and observed the disruptive behaviour of excessive dropout when applied per channel, but its benefits in helping model generalisation when applied in moderation. Ultimately, we showed that even with a simple architecture, strong performance can be achieved in classifying FRBs using significantly fewer parameters than recent models.

This project touched on discovering network architectures and hyper-parameters guided by optimisation routines. However, it devoted the majority of its time to understanding the impacts of different architectures and regularisation techniques using grid-based searches, which unfortunately are time consuming and inefficient when it comes to searching for optimal hyper-parameters (Bergstra and Bengio 2012). The use of a FRB classifier, perhaps, lies more in its application than in learning about the effects of changes in each of its hyper-parameters at length. Beyond our promises to invest in better high performance computing practices, we strongly advocate further work on efficiently automating the exploration of optimal architectures and hyper-parameters.

It should be mentioned that this falls under the broader field of *AutoML* which aims to automate the construction of machine learning pipelines and put them in the hands of domain experts who do not necessarily have machine learning expertise (He, Zhao and Chu 2019). Work with astronomical data is very much a context where AutoML could be explored further. Alongside making machine learning techniques more accessible, making them more transparent is of great importance. Recent work by Zhang et al. 2020 uses *saliency maps* (Simonyan, Vedaldi and Zisserman 2013) to improve the interpretability of FRB classifications by DNNs through creating maps of the most influential portions of the input data to their classification. A similar analysis for the models presented in this work is greatly needed and will be a focus of future research.

Finally, we hope that the care taken in presenting this project will not only mean it has enduring relevance for its writer, but perhaps also to a wider audience - lofty aspirations for a fourth year project.

APPENDIX

A.1. Metrics

One way of capturing the differences between the true label and the predicted label of an observation in a classification problem is by using the *confusion matrix*. We present an aptly themed confusion matrix in this context where classifying an observation as *positive* means classifying it as a FRB, and *negative* means classifying it as RFI:

		True label	
		Positive/FRB	Negative/RFI
Predicted label	Positive/FRB	TP	FP
	Negative/RFI	FN	TN

True Positive (TP)
False Positive (FP)
True Negative (TN)
False Negative (FN)

The entries on the diagonal, True Positive (TP) and True Negative (TN), correspond to correctly classifying a FRB as a FRB, and correctly classifying RFI as RFI. The off-diagonal entries correspond to the 2 types of misclassifications that can be made: *falsely* classifying RFI as a *positive* observation, a FRB (FP) and *falsely* classifying a FRB as a *negative* observation, RFI (FN).

Based on the confusion matrix, we can define the following metrics:

$$\begin{aligned}
\text{accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \\
\text{precision} &= \frac{TP}{TP + FP} \\
\text{recall} &= \frac{TP}{TP + FN} \\
\text{F1} &= \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}}
\end{aligned}$$

Accuracy is the proportion of observations correctly classified, *precision* is the proportion of observations classified as positive (labelled FRB) that were correctly classified, *recall* is the proportion of true observations (actual FRBs) that were correctly classified and *F1* is the harmonic mean of recall and precision.

POTENTIALLY UNFAMILIAR TERMS

dispersion measure The dispersion measure, which is proportional to the number of free electrons along the line of sight, is given by the equation:

$$\mathrm{DM} = \int_0^d \eta_e(l)\mathrm{d}l$$

where η_e is the electron number density, l is a path length and d is the distance to the FRB (Petroff, Hessels and Lorimer 2019) . 19

L-band the L-band is a range of frequencies in the radio spectrum from 1 to 2 gigahertz . 28

magnetar magnetars are young, highly magnetized neutron stars which display a wide array of X-ray activity including short bursts, large out- bursts, giant flares and quasi-periodic oscillations, often coupled with interesting timing behavior (Kaspi and Beloborodov 2017). 18, 19

transfer learning transfer learning involves re-purposing a pre-trained model to a new task. In the context of CNNs, a pre-trained model that has already learnt to identify useful low-level features is often re-purposed by connecting a feed-forward neural network to the end of the pre-trained model and then training the appended feed-forward neural network to the specific task. 27

transient transients are astronomical objects or phenomena that take place over relatively small timescales - somewhere on the scale of milliseconds or a few years, as opposed to millions or billions of years. 18

Acronyms

ANN Artificial Neural Network. 20–22

BN Batch Normalisation. 17, 18, 26, 40, 43, 44

CNN Convolutional Neural Network. 1, 5–7, 11, 13, 16, 17, 21, 22, 25, 27, 45, 49

Conv. Convolution. 21, 26, 34, 37, 40

DM Dispersion Measure. 19, 20, 22, 23, 27–30, 38

DNN Deep Neural Network. 12, 20, 23, 24, 46

FC Fully-Connected Layer. 21, 26, 34, 37

FLAG Focal plane L-band Array for the Green Bank Telescope. 28

FN False Negative. 47

FP False Positive. 47

FRB Fast Radio Burst. 1, 6, 16, 18–28, 31, 37, 45–48

GBT Green Bank Telescope. 28

Gl.Avg-pool Average Pooling. 26, 37

GP-UCB Gaussian Process Upper Confidence Bound. 15

GPU Graphic Processing Unit. 19, 44

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 6, 7, 12

Max-pool Max Pooling. 21, 34, 37, 40

Acronyms

ReLU Rectified Linear Unit. 10, 21, 26, 34, 37, 40

RFI Radio Frequency Interference. 19, 20, 22–25, 27, 28, 31, 33, 47

RGB Red Green Blue. 30

RRAT Rotating Radio Transients. 31

S/N Signal to Noise ratio. 20, 22, 24, 25, 27

TN True Negative. 47

TP True Positive. 47

TPE Tree Parzen Estimators. 14, 15

UCT University of Cape Town. 2

BIBLIOGRAPHY

Abadi, Martin, et al. 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. `http://tensorflow.org/`.

Agarwal, Devansh, et al. 2020. 'FETCH: A deep-learning based classifier for fast transient classification'. *Monthly Notices of the Royal Astronomical Society* 497 (2): 1661–1674.

Andersen, BC, et al. 2020a. 'A bright millisecond-duration radio burst from a Galactic magnetar'. *arXiv preprint arXiv:2005.10324*.

– . 2020b. 'A bright millisecond-duration radio burst from a Galactic magnetar'. *arXiv preprint arXiv:2005.10324*.

Arpit, Devansh, et al. 2017. 'A Closer Look at Memorization in Deep Networks'. In *ICML*.

Bailes, M, et al. 2016. 'MeerTime-the MeerKAT Key Science Program on Pulsar Timing'. In *MeerKAT Science*, 011.

Bannister, Keith W, et al. 2019. 'A single fast radio burst localized to a massive galaxy at cosmological distance'. *Science* 365 (6453): 565–570.

Barsdell, Benjamin R, et al. 2012. 'Accelerating incoherent dedispersion'. *Monthly Notices of the Royal Astronomical Society* 422 (1): 379–392.

Bergstra, James S., et al. 2011. 'Algorithms for Hyper-Parameter Optimization'. In *Advances in Neural Information Processing Systems 24*, ed. by J. Shawe-Taylor et al., 2546–2554. Curran Associates, Inc. `http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

Bergstra, James, and Yoshua Bengio. 2012. 'Random search for hyper-parameter optimization'. *The Journal of Machine Learning Research* 13 (1): 281–305.

Cai, Shaofeng, et al. 2019. 'Effective and efficient dropout for deep convolutional neural networks'. *arXiv preprint arXiv:1904.03392*.

Chatterjee, S, et al. 2017. 'A direct localization of a fast radio burst and its host'. *Nature* 541 (7635): 58–61.

Chollet, François. 2017. 'Xception: Deep learning with depthwise separable convolutions': 1251–1258.

Chollet, François, et al. 2015. *Keras*. `https://github.com/fchollet/keras`.

Connor, Liam, and Joeri van Leeuwen. 2018. 'Applying deep learning to fast radio burst classification'. *The Astronomical Journal* 156 (6): 256.

DeCastro-García, Noemí, et al. 2019. 'Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm'. *Complexity* 2019.

Dumez-Viou, Cédric, Rodolphe Weber and Philippe Ravier. 2016. 'Multi-Level Pre-Correlation RFI Flagging for Real-Time Implementation on UniBoard'. *Journal of Astronomical Instrumentation* 5 (04): 1641019.

Eatough, R P, E F Keane and AG Lyne. 2009. 'An interference removal technique for radio pulsar searches'. *Monthly Notices of the Royal Astronomical Society* 395 (1): 410–415.

Falkner, Stefan, Aaron Klein and Frank Hutter. 2018. 'BOHB: Robust and Efficient Hyperparameter Optimization at Scale'. In *International Conference on Machine Learning*, 1437–1446.

Fukushima, Kunihiko, and Sei Miyake. 1982. 'Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition'. In *Competition and cooperation in neural nets*, 267–285. Springer.

Gajjar, V, et al. 2018. 'Highest frequency detection of FRB 121102 at 4–8 GHz using the breakthrough listen digital backend at the Green Bank Telescope'. *The Astrophysical Journal* 863 (1): 2.

Garrido-Merchán, Eduardo C, and Daniel Hernández-Lobato. 2020. 'Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes'. *Neurocomputing* 380:20–35.

Gary, Dale E, Zhiwei Liu and Gelu M Nita. 2010. 'A wideband spectrometer with RFI detection'. *Publications of the Astronomical Society of the Pacific* 122 (891): 560.

Glorot, Xavier, Antoine Bordes and Yoshua Bengio. 2011. 'Deep sparse rectifier neural networks'. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 315–323.

Golpayegani, Golnoosh, and Duncan Lorimer. 2019. 'GBTrans: A synergistic search for Fast Radio Bursts with Green Bank 20-m Telescope'. *AAS* 233:110–01.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville. 2016. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

He, Kaiming, et al. 2016. 'Deep residual learning for image recognition'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, Xin, Kaiyong Zhao and Xiaowen Chu. 2019. 'AutoML: A Survey of the State-of-the-Art'. *arXiv preprint arXiv:1908.00709*.

Howard, Andrew G, et al. 2017. 'Mobilenets: Efficient convolutional neural networks for mobile vision applications'. *arXiv preprint arXiv:1704.04861*.

Huang, Gao, et al. 2017. 'Densely connected convolutional networks'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4700–4708.

Hubel, David H, and Torsten N Wiesel. 1962. 'Receptive fields, binocular interaction and functional architecture in the cat's visual cortex'. *The Journal of physiology* 160 (1): 106.

Hutter, Frank, Holger H Hoos and Kevin Leyton-Brown. 2011. 'Sequential model-based optimization for general algorithm configuration'. In *International conference on learning and intelligent optimization*, 507–523. Springer.

Ioffe, Sergey, and Christian Szegedy. 2015. 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. In *International Conference on Machine Learning*, 448–456.

Jamieson, Kevin, and Ameet Talwalkar. 2016. 'Non-stochastic best arm identification and hyperparameter optimization'. In *Artificial Intelligence and Statistics*, 240–248.

Jarrett, Kevin, et al. 2009. 'What is the best multi-stage architecture for object recognition?' In *2009 IEEE 12th international conference on computer vision*, 2146–2153. IEEE.

Jiang, Y., et al. 2009. 'A study of the effect of noise injection on the training of artificial neural networks': 1428–1432.

Kaspi, Victoria M, and Andrei M Beloborodov. 2017. 'Magnetars'. *Annual Review of Astronomy and Astrophysics* 55:261–301.

Kingma, Diederik P, and Jimmy Ba. 2014. 'Adam: A method for stochastic optimization'. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, Alex, Ilya Sutskever and Geoffrey E Hinton. 2012. 'Imagenet classification with deep convolutional neural networks': 1097–1105.

Kushner, Harold J. 1964. 'A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise'.

Langston, Glen, et al. 2013. 'UNC SKYNET adds NRAO 20m Radio Telescope: Dynamic Research and Funding'. *AAS* 221:328–07.

LeCun, Yann, Koray Kavukcuoglu and Clément Farabet. 2010. 'Convolutional networks and applications in vision'. In *Proceedings of 2010 IEEE international symposium on circuits and systems*, 253–256. IEEE.

LeCun, Yann, et al. 1998. 'Gradient-based learning applied to document recognition'. *Proceedings of the IEEE* 86 (11): 2278–2324.

Li, Lisha, et al. 2017. 'Hyperband: A novel bandit-based approach to hyperparameter optimization'. *The Journal of Machine Learning Research* 18 (1): 6765–6816.

Li, Xiang, et al. 2019. 'Understanding the disharmony between dropout and batch normalization by variance shift'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2682–2690.

Lin, Min, Qiang Chen and Shuicheng Yan. 2013. 'Network in network'. *arXiv preprint arXiv:1312.4400*.

Lipton, Zachary C, and Jacob Steinhardt. 2018. 'Troubling trends in machine learning scholarship'. *arXiv preprint arXiv:1807.03341*.

Lorimer, DR, et al. 2007. 'A bright millisecond radio burst of extragalactic origin'. *Science* 318 (5851): 777–780.

MacMahon, David HE, et al. 2018. 'The breakthrough listen search for intelligent life: A wideband data recorder system for the Robert C. Byrd Green Bank Telescope'. *Publications of the Astronomical Society of the Pacific* 130 (986): 044502.

Macquart, J-P, et al. 2020. 'A census of baryons in the Universe from localized fast radio bursts'. *Nature* 581 (7809): 391–395.

Marcote, Benito, et al. 2020. 'A repeating fast radio burst source localized to a nearby spiral galaxy'. *Nature* 577 (7789): 190–194.

Misra, Diganta. 2020. 'Mish: A Self Regularized Non-Monotonic Activation Function'. *arXiv preprint arXiv:1908.08681*: 1–14.

Mockus, Jonas, Vytautas Tiesis and Antanas Zilinskas. 1978. 'The application of Bayesian methods for seeking the extremum'. *Towards global optimization* 2 (117-129): 2.

Nair, Vinod, and Geoffrey E Hinton. 2010. 'Rectified linear units improve restricted boltzmann machines'. In *ICML*.

Park, E., et al. 2016. 'Combining multiple sources of knowledge in deep CNNs for action recognition': 1–8.

Petroff, E, et al. 2016. 'FRBCAT: The Fast Radio Burst Catalogue'. *Publications of the Astron. Soc. of Australia* 33:e045.

Petroff, E, et al. 2015. 'Identifying the source of perytons at the Parkes radio telescope'. *Monthly Notices of the Royal Astronomical Society* 451 (4): 3933–3940.

Petroff, Emily, JWT Hessels and DR Lorimer. 2019. 'Fast radio bursts'. *The Astronomy and Astrophysics Review* 27 (1): 4.

Prochaska, J Xavier, et al. 2019. 'The low density and magnetization of a massive galaxy halo exposed by a fast radio burst'. *Science* 366 (6462): 231–234.

Rajwade, KM, et al. 2019. 'A 21 cm pilot survey for pulsars and transients using the Focal L-Band Array for the Green Bank Telescope'. *Monthly Notices of the Royal Astronomical Society* 489 (2): 1709–1718.

Ramachandran, Prajit, Barret Zoph and Quoc V Le. 2017. 'Searching for activation functions'. *arXiv preprint arXiv:1710.05941*.

Rasmussen, Carl Edward, and Christopher KI Williams. 2006. *Gaussian processes for machine learning. Number ISBN 0-262-18253-X*.

Ravi, V, et al. 2019. 'A fast radio burst localized to a massive galaxy'. *Nature* 572 (7769): 352–354.

Rumelhart, David E, Geoffrey E Hinton and Ronald J Williams. 1986. 'Learning representations by back-propagating errors'. *nature* 323 (6088): 533–536.

Russakovsky, Olga, et al. 2015. 'ImageNet Large Scale Visual Recognition Challenge'. *International Journal of Computer Vision (IJCV)* 115 (3): 211–252. doi:10.1007/s11263-015-0816-y.

Sandler, Mark, et al. 2018. 'Mobilenetv2: Inverted residuals and linear bottlenecks'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 4510–4520.

Sanidas, S, et al. 2017. 'MeerTRAP: A pulsar and fast transients survey with MeerKAT'. *Proceedings of the International Astronomical Union* 13 (S337): 406–407.

Santurkar, Shibani, et al. 2018. 'How does batch normalization help optimization?' In *Advances in Neural Information Processing Systems*, 2483–2493.

Simonyan, Karen, Andrea Vedaldi and Andrew Zisserman. 2013. 'Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps'.

Simonyan, Karen, and Andrew Zisserman. 2014. 'Very deep convolutional networks for large-scale image recognition'. *arXiv preprint arXiv:1409.1556*.

Smith, Adam B, Daniel B Caton and R Lee Hawkins. 2016. 'Implementation and operation of a robotic telescope on Skynet'. *Publications of the Astronomical Society of the Pacific* 128 (963): 055002.

Srinivas, Niranjan, et al. 2009. 'Gaussian process optimization in the bandit setting: No regret and experimental design'. *arXiv preprint arXiv:0912.3995*.

Srivastava, Nitish, et al. 2014. 'Dropout: a simple way to prevent neural networks from overfitting'. *The journal of machine learning research* 15 (1): 1929–1958.

Surnis, Mayuresh P, et al. 2019. 'GREENBURST: A commensal Fast Radio Burst search back-end for the Green Bank Telescope'. *Publications of the Astronomical Society of Australia* 36.

Szegedy, Christian, et al. 2015. 'Going deeper with convolutions': 1–9.

Szegedy, Christian, et al. 2017. 'Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning'. `https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806/14311`.

Szegedy, Christian, et al. 2016. 'Rethinking the inception architecture for computer vision'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2818–2826.

Van Rossum, Guido. 2020. *The Python Library Reference, release 3.8.2*. Python Software Foundation.

Walters, Anthony, et al. 2019. 'Probing diffuse gas with fast radio bursts'. *Physical Review D* 100 (10): 103519.

Wang, Wei, et al. 2019. 'Development of convolutional neural network and its application in image classification: a survey'. *Optical Engineering* 58 (4): 040901.

Yang, Li, and Abdallah Shami. 2020. 'On hyperparameter optimization of machine learning algorithms: Theory and practice'. *Neurocomputing* 415:295–316.

Yu, Tong, and Hong Zhu. 2020. 'Hyper-Parameter Optimization: A Review of Algorithms and Applications'. *arXiv preprint arXiv:2003.05689*.

Zackay, Barak, and Eran O Ofek. 2017. 'An accurate and efficient algorithm for detection of radio bursts with an unknown dispersion measure, for single-dish telescopes and interferometers'. *The Astrophysical Journal* 835 (1): 11.

Zagoruyko, Sergey, and Nikos Komodakis. 2016. 'Wide residual networks'. *arXiv preprint arXiv:1605.07146*.

Zhang, C, et al. 2020. 'Applying saliency-map analysis in searches for pulsars and fast radio bursts'. *arXiv preprint arXiv:2005.11066*.

Zhang, Xiangyu, et al. 2018a. 'Shufflenet: An extremely efficient convolutional neural network for mobile devices'. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856.

Zhang, Yunfan Gerry, et al. 2018b. 'Fast radio burst 121102 pulse detection and periodicity: a machine learning approach'. *The Astrophysical Journal* 866 (2): 149.

Zhou, Yi-Tong, and Rama Chellappa. 1988. 'Computation of optical flow using a neural network.' In *ICNN*, 71–78.