Final Project Report

5th September 2022

Classification of SETI Signatures

Introduction

The Drake equation is reported as the second most famous scientific equation [https://www.seti.org/drake-equation-index]. It was invented in 1961 by Frank Drake, as a way of estimating how many intelligent alien technological civilisations exist in the Milky Way galaxy.

$$N = R_* \cdot f_p \cdot n_e \cdot f_l \cdot f_i \cdot f_c \cdot L$$

The Drake equation invented by Frank Drake for estimating probability of intelligent extra terrestrial life in the Milky Way. [1]

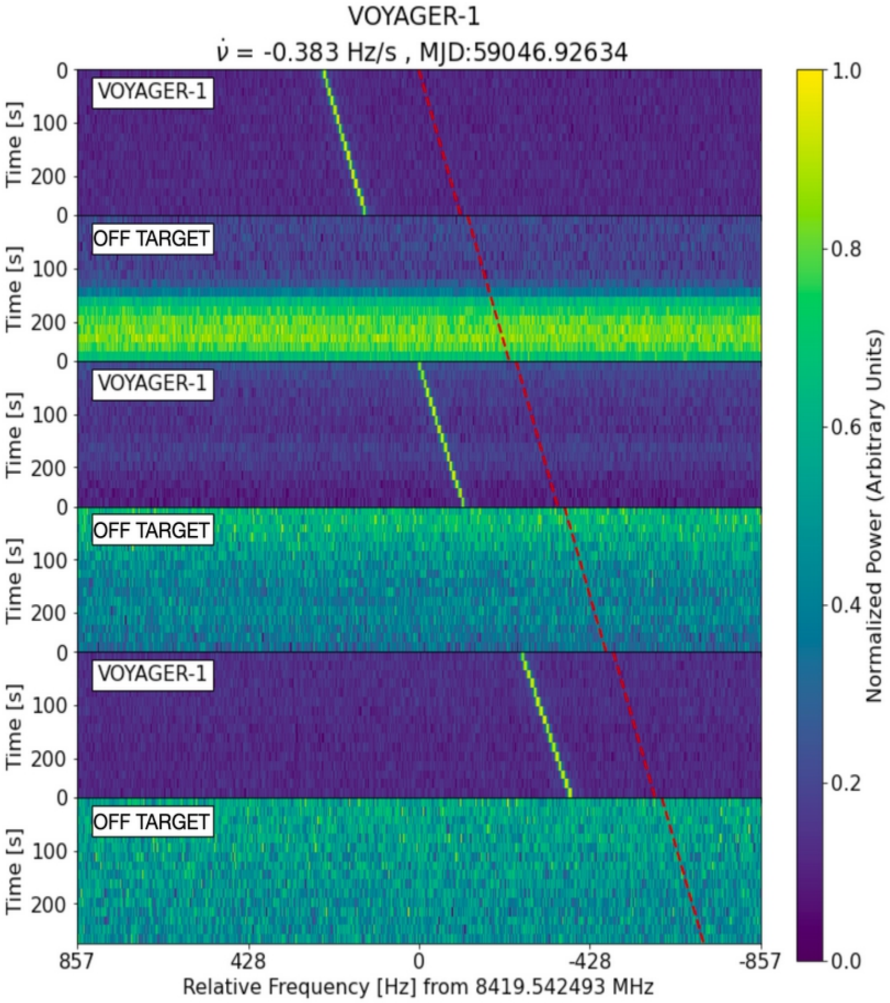
This equation and the related search for extraterrestrial intelligence (ETI) at the time lead to the SETI (Search for Extraterrestrial Intelligence) institute being incorporated in 1984 [1]. In July 2015 the Breakthrough Listen program was launched by Yuri Milner and Stephen Hawking, which committed 100 million USD to the search for ETI [2]. Its aim is to use radio telescopes to scan one million nearby stars for technosignatures, which are electromagnetic signals that evidence intelligent life.

The traditional approach for detecting technosignatures in the radio telescope readings is to use dedicated hardware and DSP (Digital Signal Processing) techniques. These are essentially handcrafted equations that are implemented on hardware, which allows them to process large amounts of data quickly. This is important because the Breakthrough Listen project can produce petabytes of data each day [3].

It is also important for technosignature detection methods to have a low false negative rate, this is because the signals are rare so it is critical an genuine candidates are not overlooked. However the false positive rate should also be low, since any possible detection needs to be manually reviewed, so if there were a lot of them it would not be feasible for people to review them all.

In recent years deep learning methods have been successfully applied to detecting items in images with a high degree of accuracy. To explore this in the context of the search for extraterrestrial intelligence, SETI launched a Kaggle competition in 2021 [4]. This competition provided a set of 2D spectogram readings from the Green Bank Telescope, some of which were injected with artificial technosignatures. The aim of the competition was to detect as many of these signals as accurately as possible using machine learning.

An example of what a data sample looks like is shown below:

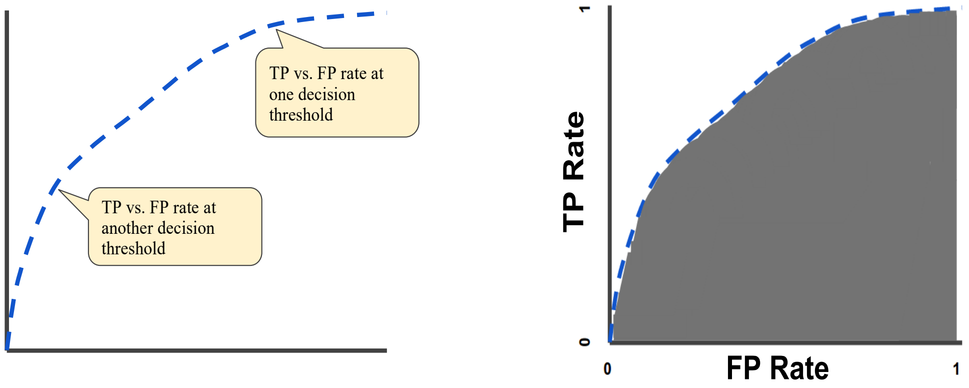


Technosignature in a spectogram [5]

This shows an image composed of 6 spectograms, three of which contain a technosignature from the Voyager 1 spacecraft. The sequence in the images is “ABACAD”. Where “A” is an on target reading, where the telescope is pointing at the region of interest, and the others are for different off target regions.

The reason for using this sequence is to help mitigate one of the biggest challenges in technosignature detection, background noise. Since the telescopes are based on Earth, there is a large amount of electromagnetic noise from human sources. These can mask the the faint candidate technosignatures. So by taking off target readings, the signals present in them can be compared with on target ones to determine what may be unwanted noise. However another challenge is that the technosignatures will not appear in the same position in the spectrogram, due to the relative motion of the Earth based telescope with the target stars. So it is not a simple case of subtracting off target signals from on target ones.

The Kaggle competition made use of the Receiver Operatoring Characteristic Area Under Curve (ROC AUC) metric to measure the performance of the submissions. This tracks the two important metrics that were mentioned earlier, true positive (or recall) and false positive rates. These are plotted on a graph and the area under the curve is calculated to obtain an ROC AUC score:



ROC curve (left) and ROC AUC (right) [6]

The line on the plot represents the true positive versus false positive rate for different classification thresholds, this is called the ROC curve. The area under the curve, or ROC AUC, gives an indication of the quality of a set of predictions. With 0 being all wrong predictions and 1 being perfect predictions.

This metric is suited to this competition because a deep learning model outputs a probability of how confident that a sample is a positive one, i.e. that it contains a technosignature. A threshold is then chosen to map this to a binary classification. However the ROC AUC represents how good the predictions are for all possible thresholds.

The winning entry for the competition had an ROC AUC score of 0.96782 [7], which will be challenging to improve on. However there have been many advances in image classification with machine learning that should make this possible.

Aims and Objectives

The aim of the project is to use the dataset from the Kaggle competition to train a deep learning classification model. It should have an accuracy close to the winning entry, possibly exceeding it.

The techniques that will be used will include a simple technique such as building a CNN model from scratch and more advanced techniques such as using an ensemble of backbone models trained using transfer learning.

Literature Review

The following literature will be reviewed in this section:

- **First place entry for the SETI Breakthrough Listen Kaggle competition** [8]

  An overview of the first place Kaggle competition entry.

- **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan et al. 2019)** [9]

  The paper that introduced the EfficientNet model, which was used by the second place competition entry [https://www.kaggle.com/competitions/seti-breakthrough-listen/discussion/266397].

- **Meta Pseudo Labels (Pham et al. 2021)** [10]

  Paper for the model that achieved state of the art (SOTA) performance on ImageNet Top-1 in 2021.

- **Machine Vision and Deep Learning for Classification of Radio SETI Signals (Harp et al. 2019)** [12]

  An overview of different machine learning approaches for SETI signal classification.

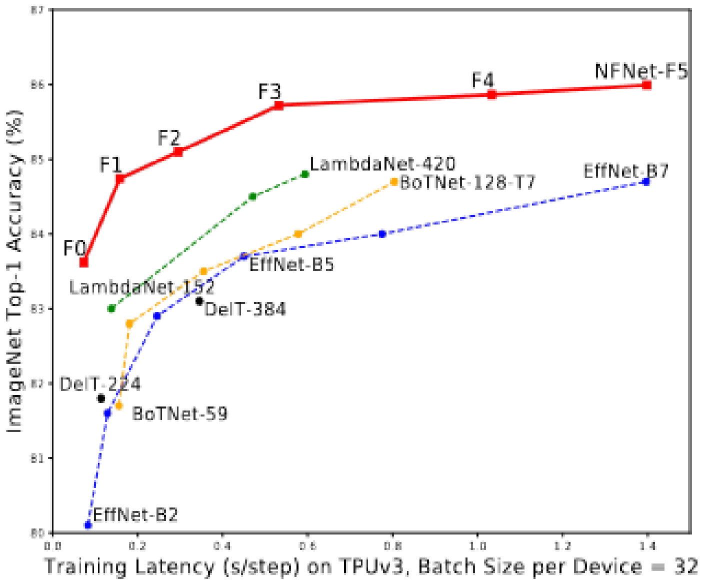
First Place Entry for SETI Breakthrough Listen Kaggle Competition

Summary

This solution used a transfer learning approach. A pre-trained Normalisation Free (NFNet) [13] Convolutional Neural Network (CNN) model was used as a backbone model. This was fine tuned for the SETI task by further training it on the competition dataset.

NFNet models scored highly on the ImageNet Top-1 benchmarking dataset during 2021 [11], which is when the Kaggle competition was launched.

The graph below show high performing ImageNet Top-1 models from 2021:



Plot of best performing ImageNet Top-1 models in 2021 [8]

The learning pipeline for the first place entry shown below:

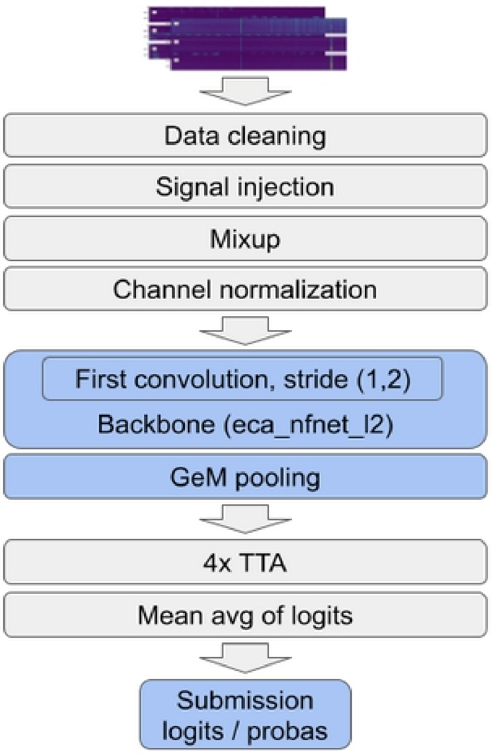
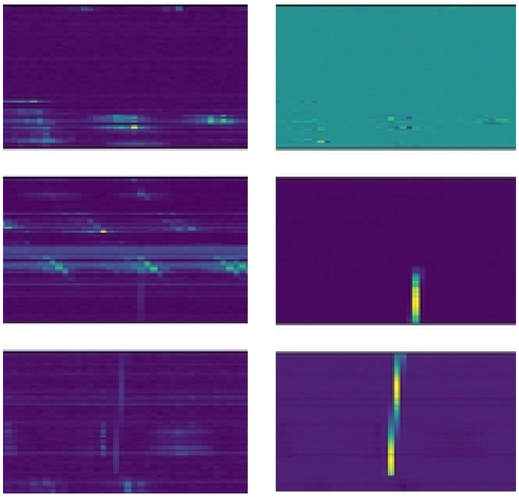


Diagram of first place pipeline architecture [8]

The pipeline consists of the following stages:

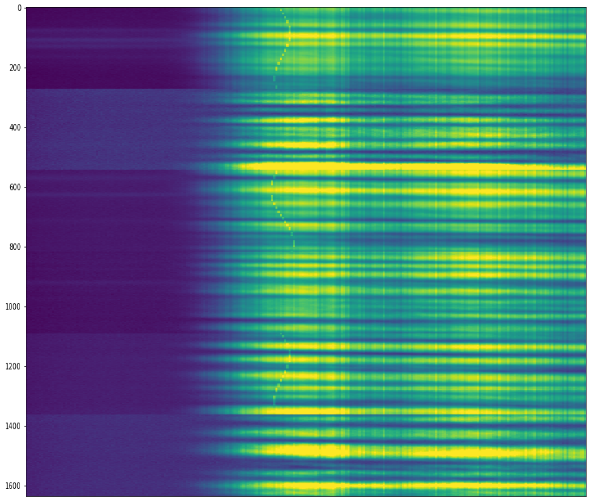
Pre-processing

In the stage the data was cleaned to remove background noise. This not only helped the signals to be identified, but also prevented the model from overfitting on background noise, which sometimes repeated across samples. This was done by comparing the difference of the first column in each image to all other columns and images. Original columns were then replaced with the difference of the norm of multiple matching samples. This helped boost the signal to noise ratio.



Original sample on the left and a cleaned sample on the right [8]

It was also noticed that an “s-shaped signal” was prominent in the test data but not the training data. To address this they generated samples with s-shaped signals and background noise in their training data. The SetiGen tool which generates synthetic SETI signals was used for this [29].



A sample with an injected s-shaped signal.
[https://www.kaggle.com/c/seti-breakthrough-listen/discussion/266385]

Mixup [https://arxiv.org/abs/1710.09412] was also used, which overlays two images with a transparency effect to generate a new one. This helps reduce overfitting on training data by effectively creating new training samples.

Channel normalisation was also finally applied after the mixup step. This puts all of the image data values in a fixed range, which allows the model to process them more efficiently, hence allowing it to converge faster.

Training

The training process used transfer learning based on a pre-trained backbone model, eca_nfnet_l2 [14]. Which had SOTA on ImageNet Top-1 in 2021 with an accuracy of 90.2% [12].

This model uses Efficient Channel Attention (ECA) [15], and is normalisation free (NF) [16], i.e. it does not contain normalisation layers. ECA is a technique for using cross-channel information efficiently. Normalisation free omits batch normalisation layers in CNN models, which normalise values for faster training. However batch normalisation introduces a dependency on batch size which this model avoids.

Trainable GeM pooling was also used. This is a form of mean pooling [17] where the mean value for regions in the image was used. This helps improve the contrast of an image by emphasising salient features.

To further improve the signal to noise ratio, only on target images in a sequence were used, which were those that could possibly contain a technosignature. Also the resolution of the images was increased by using a CNN kernel with stride (1, 2) in the first convolution layer. This is a common trick in Kaggle competitions which helps the model extract relevant features more easily.

Test Time Augmentation (TTA) was used to increase the number of training samples using techniques such as horizontal and vertical flipping. This helps prevent overfitting on the original training data.

A 5 fold cross validation strategy was used in the training phase. This splits the data into 5 parts, where 4 parts are used for training and 1 for validation in different permutations. This increases the amount of training that can be done with the available test data.

Classification

A vector containing the probability of each class was used to perform a binary classification, i.e. if a signal in the image if it exists. This is referred to as “mean avg. of logits” in the architecture diagram.

Conclusion

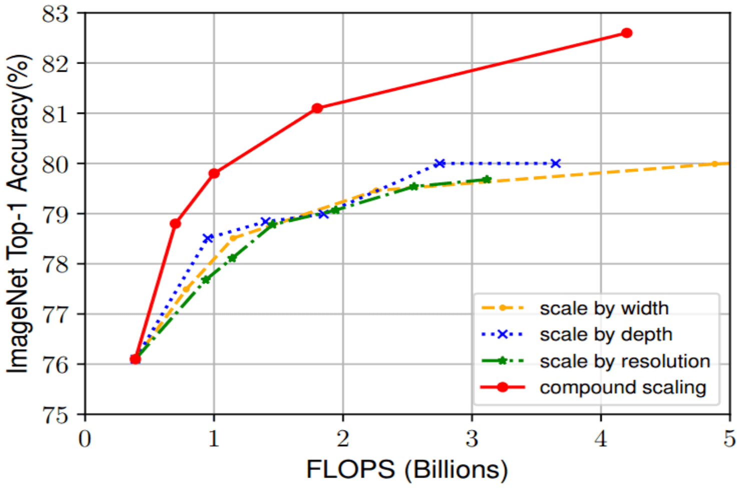
Although this solution achieved a very high score. I think its main weakness is that it uses CNN models from 2021. In 2022 there are better performing models available which could further improve the ROC AUC score. An example is Meta Pseudo Labelling, which is the current SOTA on the ImageNet benchmark, which I will discuss later on.

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (Tan et al. 2019) [18]

Summary

EfficientNet was the model used in the second place solution [19], this is based on the paper “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” by Tan et al 2019. It shows that three main traditional hyperparameters are used for CNN tuning. The layer width, depth and the input image resolution. There is also the option to use a compound scaling effect which combines all three.

The following plot shows that compound scaling using all three hyperparameters allows for better accuracy than scaling parameters individually:



Plot of effect of CNN hyperparameter tuning [18]

This makes sense since when the input image resolution increases, you need more layers and channels to detect all of the features in the image (Tan et al. 2019). By using a combination of all three scaling factors, Tan et al. have been able to scale up a baseline model to create a family of models called EfficientNets. Which were able to achieve better accuracy while using fewer model parameters and a faster inference speed compared with the previous SOTA in 2021.

The technique for selecting the baseline involves using a neural architecture search which optimises for both accuracy and FLOPs (Floating Point Operations Per Second). This resulted in the EfficentNet-B0 network. In addition to the Convolutional, Pooling and FC layers typically found in a CNN, it also used MBConv layers. MBConv is based on the Mobile Net models [16] which were specifically designed to be lightweight so they can run on mobile devices. These made use of deliberate bottlenecks, where the image size was reduced during a convolution operation to force the model to retain the most salient features. These bottlenecks were originally introduced in the ResNet paper [20], where the convolutions reduced and then increased the number of dimensions:

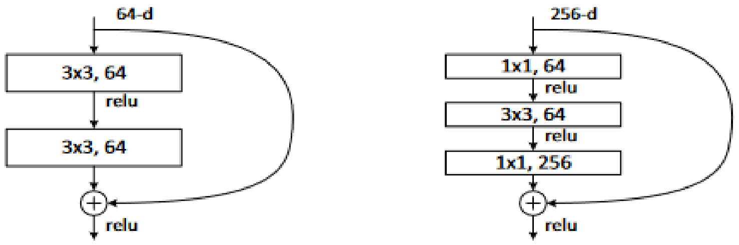


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Diagram of CNN bottleneck module [17]

A similar concept is referred to in “Squeeze-and-Excitation Networks” Hu et al. 2019 [21]. This resulted in the SENet model. The basic idea is that a parameter is added to each channel of a CNN model so the weight of the channel can be adjusted. This parameter is calculated by “squeezing” the feature maps into a vector of fixed size using pooling and then feeding the vector through a NN which outputs another vector of similar length. This vector is then used as weights for each feature map or channel based on how important it was for the classification (Hu et al. 2019)

Conclusion

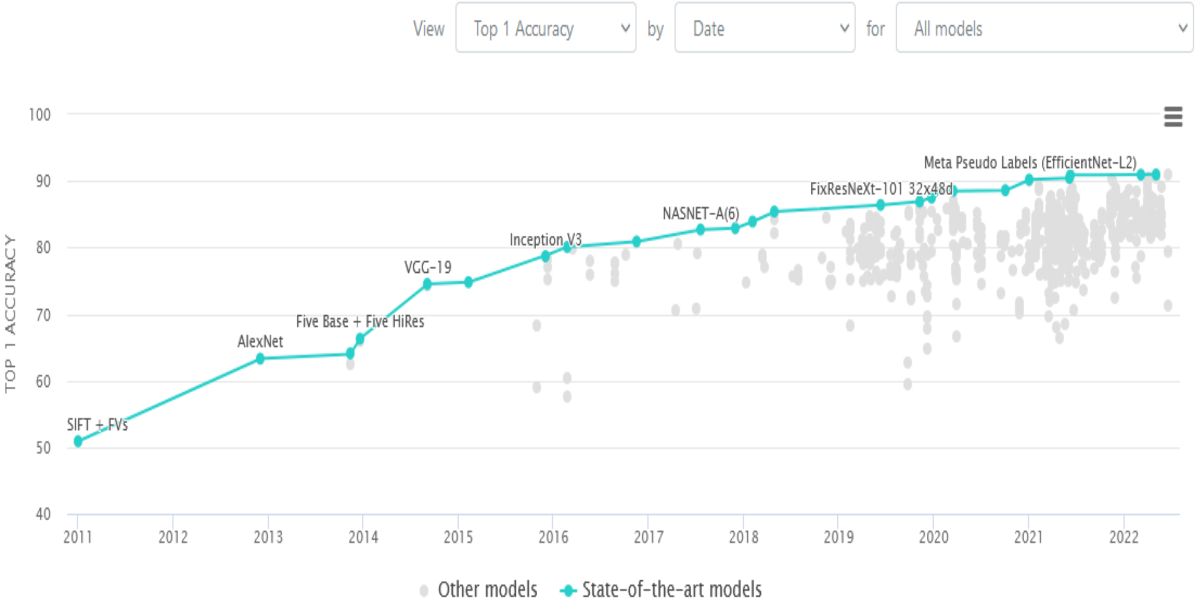
Although EffNets were second to NFNets in the ImageNet benchmark in 2021, they demonstrates that the use of bottlenecks can improve accuracy. It is worth exploring this in conjunction with better performing models to boost their performance.

A possible disadvantage is that since the bottleneck compresses data, it may cause subtle the technosignature signals to be lost when the signal to noise ratio is low. Therefore the effect of this approach will need to be monitored carefully and alongside image preprocessing techniques that can help boost the signal to noise ratio.

Meta pseudo-labels (2021) [10]

Summary

Meta Pseudo Labels (Pham et al. 2021) is a semi-supervised method that had the best SOTA accuracy in 2021 on ImageNet Top-1 of 90.2%. This model is also called EfficientNet-L2, which the backbone model for the Kaggle winning entry, ECA NFNet L2, is based on.



View Top 1 Accuracy by Date for All models

Plot of ImageNet Top-1 best models in 2022 [12]

Meta Pseudo Labels builds on a pre-existing Noisy Student method [22].

The Noisy Student method trains a teacher model with labelled data. The teacher model is then used to infer labels for unlabelled data, which is then used to train a student model. The student is then made into a teacher and the process is repeated.

Meta Pseudo Labels further builds on the Pseudo Labels approach by feeding back the students performance on the labelled dataset to the teacher. Thus allowing the teachers parameters to be updated depending on the students performance. So the teacher is learning from the student, hence the use of the term “meta”.

This new approach gave an accuracy score of 90.2% on ImageNet Top-1, which improves on the previous SOTA by 1.6% [12].

Conclusion

Although Meta Pseudo Labels is not the current ImageNet SOTA, it has better performance than NFNet, which was the best model available during the Kaggle competition. So utilising this model and other high performing models, such as CoCa [23] which is the current SOTA will hopefully improve the solution accuracy.

It is worth trying different models to see how much they improve the results, or possibly a combination of multiple models in an ensemble architecture.

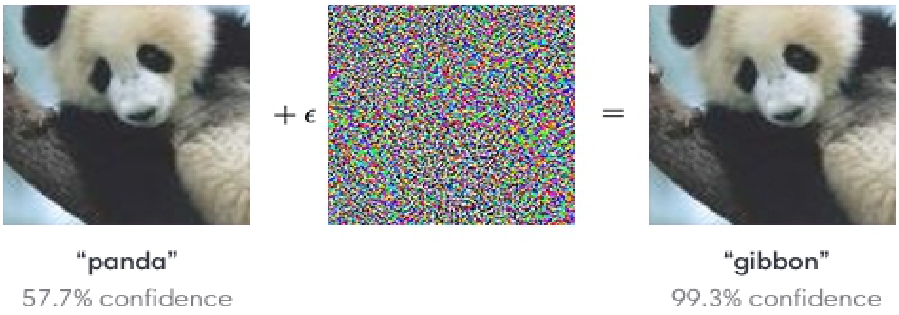
Signals (Harp et al. 2019) [11]

Summary

Harp et al. 2019 discuss traditional classification methods used by the SETI project in “Machine Vision and Deep Learning for Classification of Radio SETI Signals” [11]. These include using a SonATA (SETI on Allan Telescope Array) hardware categoriser which uses handcrafted equations to extract signals from a voltage/time diagram generated during a radio telescope survey. Its main advantage over machine learning classification is that it is able to detect signals at a lower signal to noise ratio. An obvious drawback of SonATA is that since it uses handcrafted equations, it is only able to categorise a limited number of signals.

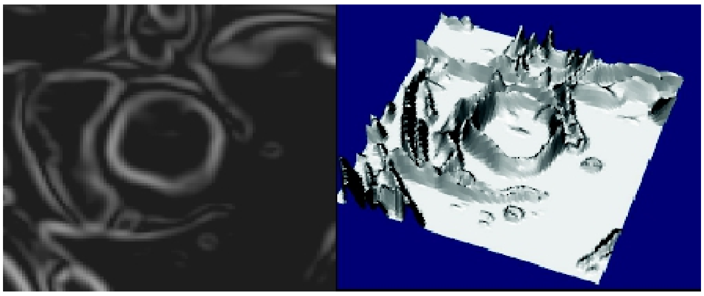
Harp et al. explain that machine learning methods such as CNN's are able to overcome the limitations of the SonATA categoriser. This is because the modelling of the functions to detect signals is done automatically by training the CNN model. However, a disadvantage of CNNs is that they are more susceptible to noise and require a higher signal to noise ratio.

Since noise is essentially random, it is difficult for a neural network (NN) to categorise it. This is because a feature representing noise would likely be high-dimensional and would require a large amount of training to categorise it (Harp et al. 2019). This weakness of NN's has been shown with adversarial examples for CNN's, where adding barely perceptible noise to an image can cause a miss-classification [23]. This is because CNN's do not learn intuitive features, but instead uses what essentially looks like random noise to a human (Harp et al. 2019).



An adversarial attack using added noise to cause a misclassification. [24]

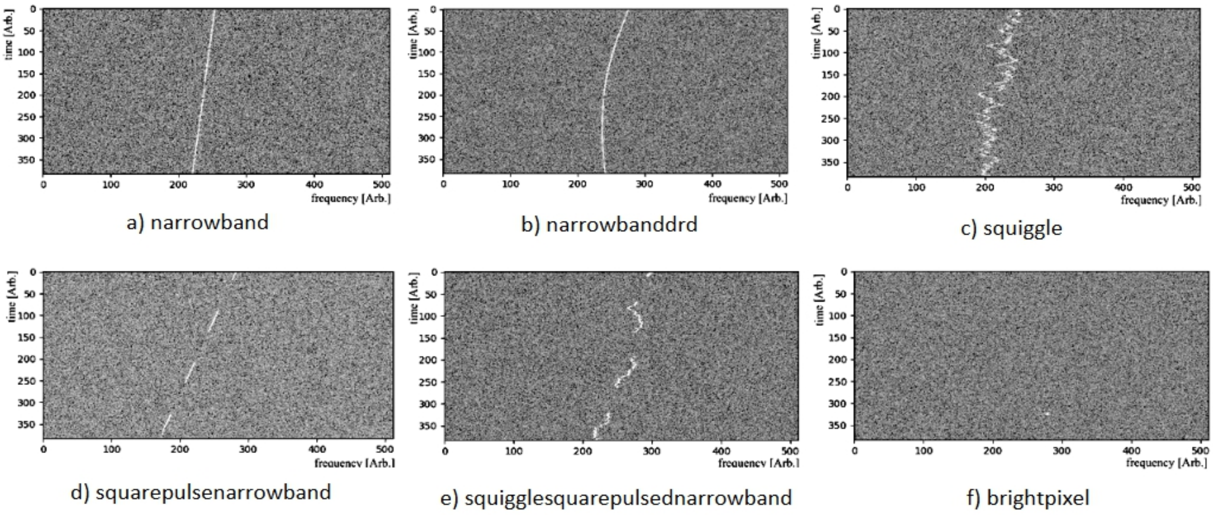
This sensitivity to noise can be mitigated in my project by using pre-processing techniques to increase the signal to noise ratio. A pre-processing technique mentioned in Harp et al. to reduce noise is the Watershed algorithm, which views a spectogram as a height map. An imaginary water drop is released on a pixel, where it moves towards a local basin. The boundaries between basins are then defined as points where the water droplet has an equal chance of moving in different directions. This allows boundaries to be identified, which are more likely to be part of the signal than noise.



A grayscale image (left) has boundaries detected (right) using the Watershed algorithm. [25]

As explained in the introduction, because of the large amount of data generated by Breakthrough Listen, a low false positive rate is important. Fortunately Harp et al. have found that CNN's are able to achieve a low false positive rate.

Harp et al. mention six types of signals that the SETI project looks for, categorised by their shape:



a) narrowband

b) narrowbanddrd

c) squiggle

d) squarepulsenarrowband

e) squigglesquarepulsednarrowband

f) brightpixel

Signal types searched for by SETI project. 26]

Conclusion

The signal types mentioned previously are relevant to my project since I can make sure that there is a fair representation of these signals types in the training data as compared with their presence in test data. I can also make use of techniques such as Setigen [29] to generate labelled test data for each signal type. This may help improve the accuracy of the classifier since it will be possible to generate samples without any noise, allowing the model to learn signal features more easily.

The particular types of NN's that Harp et al. mention are RNN's and CNN's. The CNN's that are mentioned are AlexNet, VGG and Inception, which are quite dated and have now been superseded by better performing models as per the ImageNet benchmark [12]. Additionally RNN's have gone out of favour, with CNNs gaining better scores on ImageNet. For this reason I will mostly focus on contemporary image classification methods models that rank highly on the ImageNet benchmark.

Project Design

Project Template

The following project template is used as a basis for this report:

CM3015 Machine Learning and Neural Networks
Deep Learning on a public dataset

This requires picking a dataset from kaggle.com and developing a deep learning classification model for it. The techniques in Deep Learning with Python, F. Chollet [27] should be applied, starting with simple techniques and then applying more advanced ones from Chapter 7.

Overview and Domain

The project domain is to solve the problem presented in the Kaggle competition, SETI Breakthrough Listen – E.T. Signal Search. An overview of this is to use machine learning to perform binary classification on 2D spectogram images obtained from a radio telescope. Where a positive classification indicates the presence of an injected technosignature.

The main challenge is that the spectograms contain background noise and the injected signals are faint. Another challenge is that there are a variety of signals which do not remain stationary in sequential samples, since celestial bodies are not stationary relative to Earth.

To facilitate the competition a dataset of 6000 samples is provided. The classification work will be carried out against this dataset.

Users

The primary users of this project can be considered to be the Breakthrough Listen team from the Berkley SETI Research Center in the University of California. Since they are involved in the search for extraterrestrial intelligence, and also launched the Kaggle competition. Although the competition is over they may be interested in novel techniques that produce good results.

Secondary users are Kaggle members who have an interest in the Breakthrough Listen competition, or related competitions where similar techniques can be applied.

The deep learning research community may also be interested in any novel methods.

A wider user group can be considered anyone who wishes to use deep learning for image classification tasks. This can include applications such as detecting anomalous signals in medical EEG (electroencephalogram) recordings, or classifying baby cries [28].

Work Plan

Data Exploration

The first step will be to do an exploratory analysis of the provided data. This will involve verifying that it is free from inconsistencies such as missing or invalid values. The dataset will also be visualised to get a sense of the shape of the data and what the samples look like. These steps will make sure that the data can be loaded in a format that is suitable for training models on.

Another important aspect is making sure that there is an equal distribution of each class in the provided data. If there is not an equal distribution, then the data should be balanced by using a subset. Alternatively there are techniques that can be used to train on an unbalanced dataset, such as a weight sampler or applying more weight to rare classes.

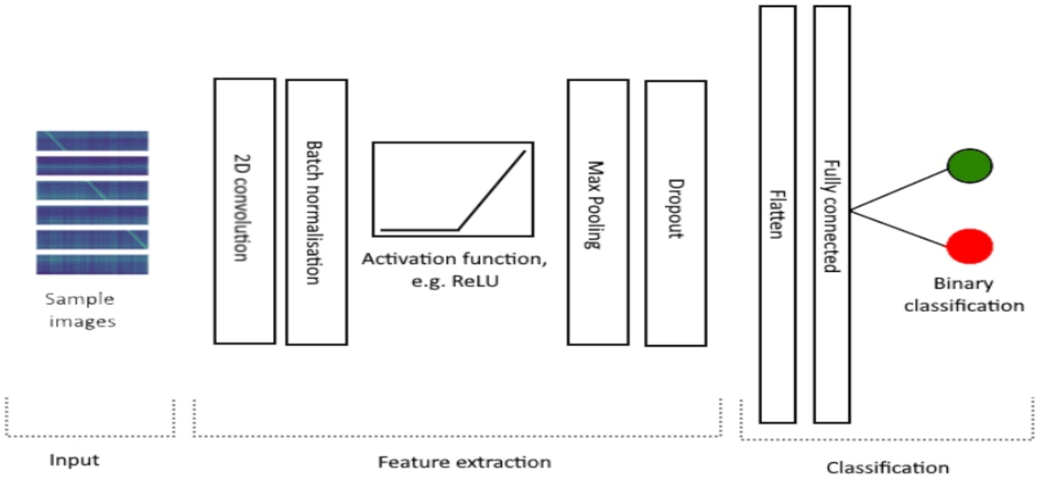
The dataset will be split into training, validation and test sets. The test set will be created before any work on the model starts, this will ensure that the model has not seen this data which ensures a fair evaluation on unseen data.

If it is found that there is not enough data for each class, then cross-validation can be used to increase the training data permutations. Additionally data augmentation techniques such as mixup or Test Time Augmentation (TTA) can be used where existing images are manipulated to create new training samples. Setigen [29] is also an option that can be used to create new data samples by injecting generated technosignatures into images.

Finally a large concern highlighted from the research is the amount of background noise that is present. To alleviate this techniques for reducing the noise will be explored. This can include Gaussian smoothing [20], pixel by pixel comparison across samples or other techniques. However care should be taken that the faint signals are not lost when removing noise.

Baseline model

For the baseline model I will build a CNN model from scratch. This will be built up from shallow to deep and a variation of layers will be tested to see which work best. The general model architecture will look like the following:



General CNN model architecture

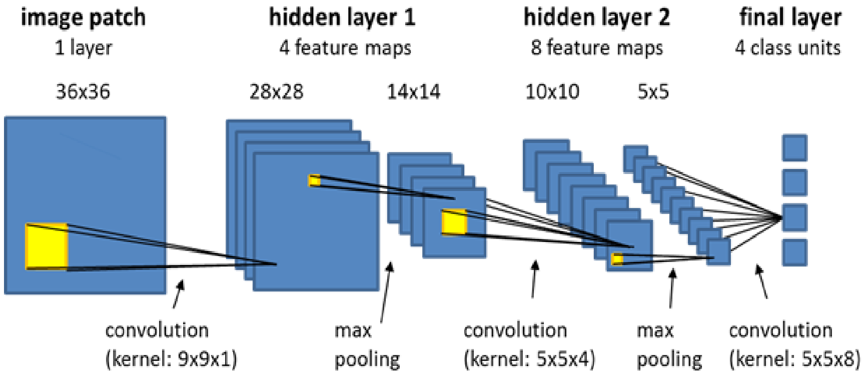
The input will consist of six images from a data sample. However these images will undergo transformations such as resizing, denoising and normalisation. These will make it easier for the model to process the data.

The feature extraction step is responsible for learning filters that can be used to recognise signals. This section will be repeated as the model is made deeper. The layers that will be used are:

- 2D convolution for learning feature maps.
- Batch normalisation for reducing the range of values, allowing for more efficient processing.
- An activation function such as Rectified Linear Unit (ReLU), which introduces non-linearity to the model. Allowing it to be approximate a function for recognising signals.
- A max pooling layer for reducing the amount of data outputted from a layer and helping prevent overfitting.
- Dropout which will randomly deactivate some of neurons, which aids in reducing overfitting on training data.

The output of the feature extraction phase will then be passed to a flattening layer, which converts the data into a column layer. This is then passed to a fully connected layer, where the neurons are connected to all of the values in the flattened layer. The fully connected layer will then output a binary classification.

A CNN model uses patches (also called kernels) that slide across an image to learn filters that can pick up on certain patterns. As the data moves through the model layers, more abstract features are learnt. A combination of the activation of each filter is then used to make a classification.



Example of CNN kernels [31]

Manual fine tuning

The results of the baseline model will then be evaluated using accuracy, ROC AUC and F1 scores. A confusion matrix will also be used to see where any bias in the classifications lies.

Using the metrics different variations of the baseline model will tried. This will include using different combinations of layers, different CNN kernel sizes and activation functions.

A suitable signal to noise ratio is also important for achieving a high accuracy. So time will be spent tweaking the image preprocessing. This can involve tweaking the image size, transformation, training on only the on target images and trying different denoising and signal boosting techniques.

Transfer learning model

Once the baseline model has been established, I will look at using a transfer learning based approach that was common for the top competition entries. Transfer learning is a technique were a large model that has been trained on a large amount of data is used for feature extraction on a related task. The model is then fine tuned to the new task by training on the new dataset.

To do this I will attempt to reproduce the first place competition entry [6]. However I may make some simplifications that do not have a large effect on the accuracy due to time constraints.

Hyperparameter optimisation

Although I will be using pre-trained models and using transfer learning, there is still some scope for hyperparameter tuning. This includes parameters such as the number of folds used during cross-validation, the depth of fully connected layers, how the training data is augmented, learning rate and weight decay where applicable.

Since the nature of this project is exploratory in nature, I am not able to comprehensively list all possible hyper-parameters, but will likely discover new ones during the project implementation.

Ensemble model

Once the advanced model is optimised I will look into using different backbone models. This involves using models that have SOTA performance on ImageNet. For example Meta Pseudo Labels [19], which was not available during the time of the Kaggle competition. However I will be using Python packages that come with these models prebuilt and pre-trained, so I will be limited by what is available in the public domain.

Finally I will use an ensemble of the predictions for each of the backbone models. The hope is that if the models have different strengths and weaknesses, then using an overall consensus for a predicted label will increase the overall accuracy.

Testing and Evaluation Plan

Since SETI technosignatures are sparse, the most important metric will be recall (sensitivity), or the number of positive cases that are accurately predicted. This will make sure that the chance of missing a technosignature is minimised.

Another important metric is the false positives rate, since the amount of data being processed is large and these need to be manually reviewed. So precision (number of correct positive predictions) is also worth tracking.

The F1 score, which is the harmonic mean between precision and recall will also provide a useful method for comparing the various approaches.

The area under the Receiver Operating Characteristic curve, ROC AUC, will also be calculated. Since this is the metric used to score the Kaggle competition entries. So it will be useful in comparing against other competition submissions.

Confusion matrices can also be used to visually identify where the model weaknesses lie. This along with data visualisation techniques such as plotting falsely categorised samples, will also be useful when optimising the models.

The dataset will also have a portion split out into a test set that the model will not be trained on. This is important because it will allow the model to be fairly evaluated on how it may perform on unseen data.

Important Technologies

The main technologies that will be used for this project are Jupyter Notebook [32] and the PyTorch deep learning framework [33].

The reason I have chosen PyTorch over TensorFlow [34] is that while it has a similar level of adoption and documentation, it has an object oriented API (Application Programming Interface) which is more intuitive to me because of my experience with other OOP (Object Oriented Programming) languages.

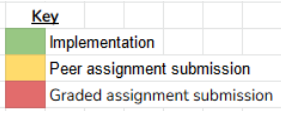
Another important aspect is the ability to visualise results and processes. To achieve this I will make use of the standard Python plotting libraries such as matplotlib [35]. The Weights and Biases API [26] will also be leveraged for plotting performance metrics for the different approaches.

Pre-trained models be downloaded using the PyTorch Image Models repository [14], which is a popular source for backbone models in Kaggle notebooks.

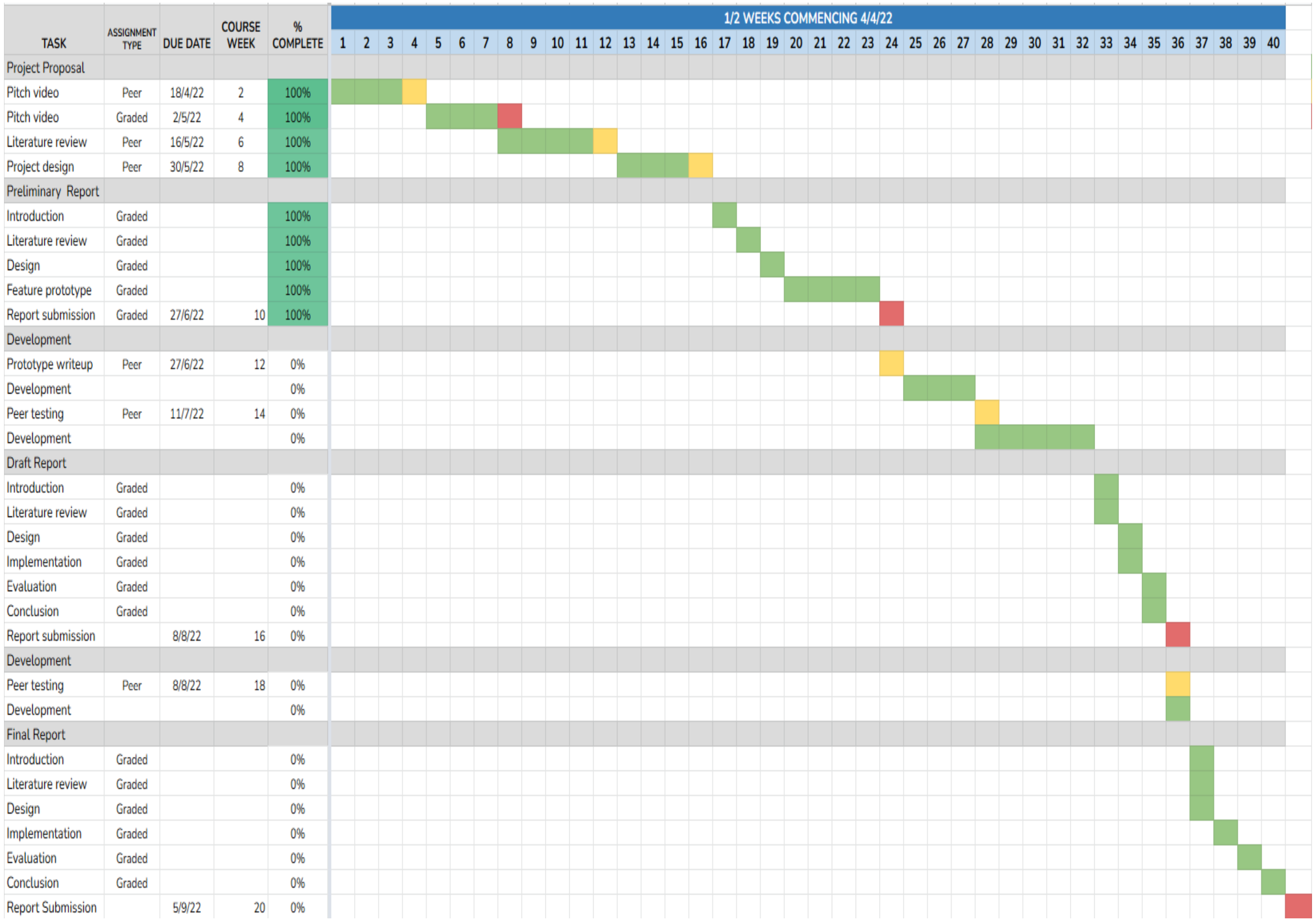
If I find that my personal computer does not have enough performance to train models quickly enough, then I will consider using a cloud based server.

Gantt chart

The Gantt chart for my project is shown below. It uses the following key for cell colours.



Key	
🟩	Implementation
🟨	Peer assignment submission
🟥	Graded assignment submission



TASK	ASSIGNMENT TYPE	DUE DATE	COURSE WEEK	% COMPLETE	1/2 WEEKS COMMENCING 4/4/22																																								
					1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	
Project Proposal																																													
Pitch video	Peer	18/4/22	2	100%	🟩	🟩	🟩	🟨																																					
Pitch video	Graded	2/5/22	4	100%					🟩	🟩	🟩	🟥																																	
Literature review	Peer	16/5/22	6	100%								🟩	🟩	🟩	🟩	🟨																													
Project design	Peer	30/5/22	8	100%													🟩	🟩	🟩	🟨																									
Preliminary Report																																													
Introduction	Graded			100%																	🟩																								
Literature review	Graded			100%																		🟩																							
Design	Graded			100%																			🟩																						
Feature prototype	Graded			100%																				🟩	🟩	🟩	🟩																		
Report submission	Graded	27/6/22	10	100%																								🟥																	
Development																																													
Prototype writeup	Peer	27/6/22	12	0%																								🟨																	
Development				0%																									🟩	🟩	🟩														
Peer testing	Peer	11/7/22	14	0%																												🟨													
Development				0%																												🟩	🟩	🟩	🟩	🟩									
Draft Report																																													
Introduction	Graded			0%																																	🟩								
Literature review	Graded			0%																																	🟩								
Design	Graded			0%																																		🟩							
Implementation	Graded			0%																																		🟩							
Evaluation	Graded			0%																																			🟩						
Conclusion	Graded			0%																																			🟩						
Report submission		8/8/22	16	0%																																				🟥					
Development																																													
Peer testing	Peer	8/8/22	18	0%																																				🟨					
Development				0%																																				🟩					
Final Report																																													
Introduction	Graded			0%																																					🟩				
Literature review	Graded			0%																																					🟩				
Design	Graded			0%																																					🟩				
Implementation	Graded			0%																																						🟩			
Evaluation	Graded			0%																																							🟩		
Conclusion	Graded			0%																																								🟩	
Report Submission		5/9/22	20	0%																																									🟥

Implementation

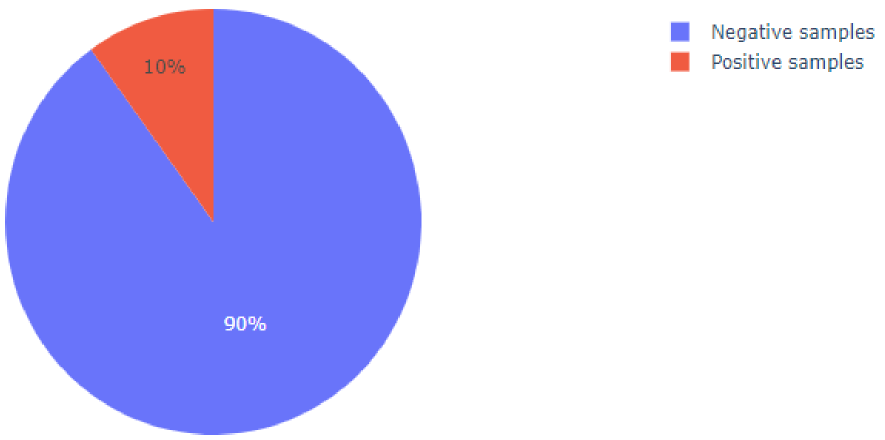
Code repository

The Jupyter notebooks for various models are available to view at:

https://github.com/uolcw/final-project

Exploratory data analysis

I found during exploratory data analysis that the data distribution was heavily biased towards negative samples:



Initial dataset distribution

This bias would lead the model to be more likely to classify samples as negative.

I also found that despite upgrading my graphics card to a GTX 1070ti with 8 GB RAM, it was taking 5 minutes for a single batch to be processed during training. I found that I was limited by memory to a batch size of about 50. So it was not feasible to train on the entire dataset of 60,000 samples. Although using more powerful hardware on the cloud was an option, I did not wish to pay for computing resource during the exploration phase.

To alleviate both of these issues I took a random subset of 1000 negative and 1000 positive samples, as well as a smaller subset of 250 negative and 250 positive samples. This allowed me to have a much smaller balanced dataset that can be used to validate my models, before training on the more time consuming full dataset.

Random seed initialisation

Allows reproducibility.

CNN model from scratch

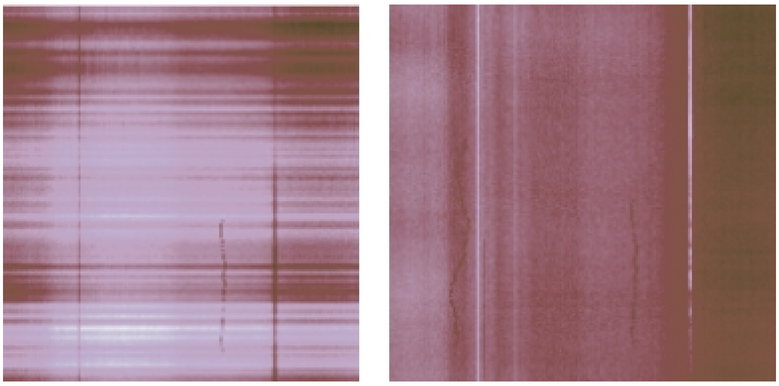
The CNN model built from scratch had almost the same architecture as initially planned.

However during evaluation of the different model variations, I found that the dropout layer was preventing the model from learning, even when set to a low dropout value of 0.01%. This is likely because the signals were faint and dropout was causing import features to be lost. So the final custom CNN model look what was originally designed but with 8 feature learning layers.

Although adding more layers improved the classification accuracy, I decided not to make the model to deep due to time and resource constraints. Instead I decided to focus on the pre-trained models which consist of hundreds of layers that have already been optimised.

Image processing

For reducing the noise in the images I used a technique I found from a Kaggle notebook. This involves decomposing images into two factor matrices. The multiplying a matrix from an on target image with one from an off target image. This resulted in an estimation of the background noise which can be subtracted from the on target image. The following diagram shows the effectiveness o reducing the background noise:



Removing background noise from an original image (left) to produce an augmented image (right).

Another technique that was used was to add and normalise all of the on target images. This reduces the amount of data that the model has to process and also increases the density of signals when present. Both of these techniques allow the model to converge faster.

Advanced model training

The advanced model used a transfer learning. So one of the first steps is to download a pre-trained model:

This is done via an instance of a PyTorch *Module* class representing the model with the *timm* package in the *init__()* method as follows:

```
timm.create_model(model_name, pretrained=pretrained, in_chans=input_channels)
```

Once the model has been created it is loaded on the GPU:

```
model.to('CUDA')
```

This is important since training on the GPU is much faster then using the CPU and RAM.

Next a loss criterion function is created. Loss is a measure of how well a model is performing by comparing predictions with ground truth labels for samples. This is also loaded on the GPU to allow faster calculations:

```
criterion = nn.BCEWithLogitsLoss().to('CUDA')
```

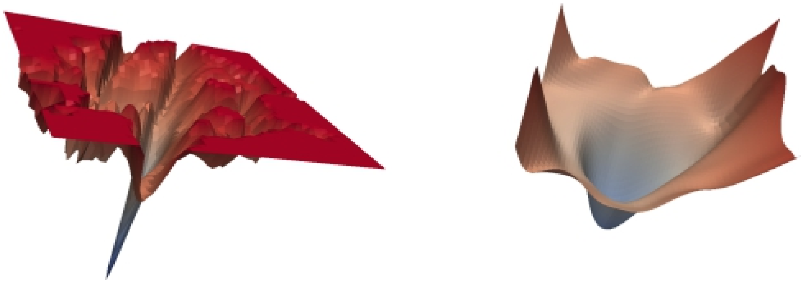
Next an optimiser is initialised. This function is used to reduce the amount of loss that a neural network experiences by updating weights during the back propogation step:

```
SAM(model.parameters(), base_optimizer, lr=config['lr'],
weight_decay=config['weight_decay'])
```

Here I use the SAM (Sharpness Aware Minimization) optimiser [https://arxiv.org/abs/2010.01412]. This takes the following arguments:

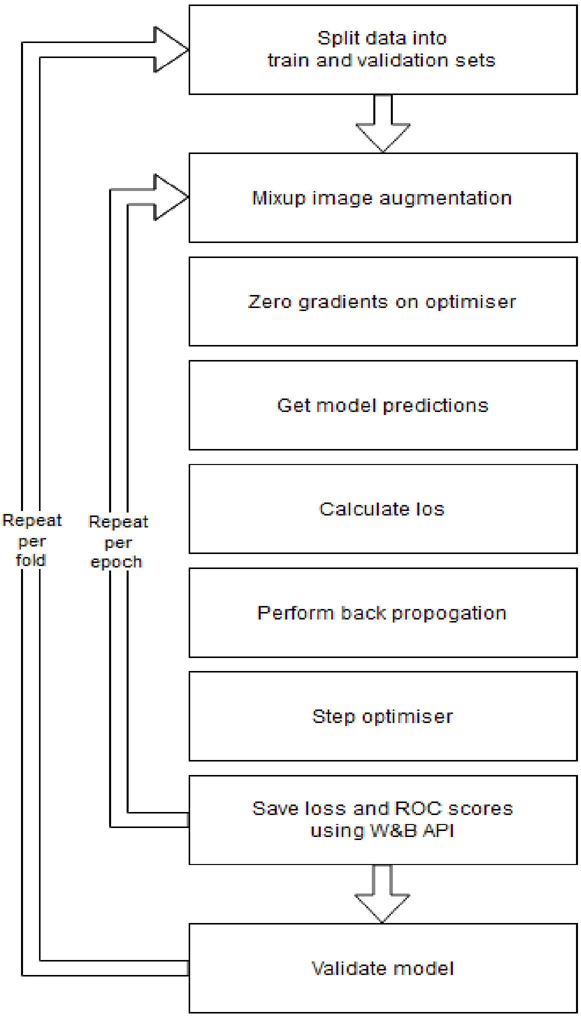
- Model parameters.
- Base optimiser: AdamP in this case.
- Learning rate: How large steps to take in the loss landscape.
- Weight decay: How fast to change the weights.

The SAM optimiser works by minimiser sharp loss changes by using the AdamP gradient descent algorithm. This causes it to seek out regions with uniformly low loss, making changes in loss overtime more stable:



The loss landscape for an optimiser without SAM (left)
and an optimiser with SAM (right)
[https://arxiv.org/pdf/2010.01412.pdf]

Next the model training and validation loop was executed which has the following outline:



Model training pipeline

K-folds cross validation

The training process utilises K-Fold cross validation:

```
kfold = StratifiedKFold(n_splits=config['num_folds'], shuffle=True,
random_state=config['seed'])

for fold, (trn_idx, val_idx) in enumerate(kfold.split(train_df, train_df['target'])):
      perform training...
```

This is a technique whereby the training data is split into K subsets (or folds). The model is then trained for multiple iterations, where a different fold is used for model validation and the rest of the data is used for training each iteration. This allows for more training to be done using the same data whilst reducing overfitting on train data.

The model is also repeatedly trained on each fold, where each iteration is referred to as an epoch.

```
for epoch in range(1, config['epochs'] + 1):
        train(train_loader, model, criterion, optimizer, epoch, scheduler)
        valid_predictions, valid_targets = validate(valid_loader, model, criterion,
epoch)
```

More precisely, each epoch consists of training and validating the model using the K-Folds strategy. The ROC AUC (Receiver Operator Characteristic Area Under Curve) score is then calculated, which will be discussed in the evaluation section of this report. Then the model state is saved, allowing it to be used in the future.

The training step for each epoch follows the standard method used for deep learning models. Where the model is trained using the test and validation data, the model loss (quality of predictions) is calculated and then back-propagation is performed. The purpose of back-propagation is to adjust the weights of the neurons in each model layer, to reduce the loss. This is done with a gradient descent algorithm, which in this case is the Sharpness Aware Minimisation (SAM) optimiser.

Mixup

In addition to the previous training steps, a technique called mixup is utilised. This involves augmenting the sample images, by overlaying pairs using transparency. This has the effect of generating more training data and allowing the model to generalise by preventing overfitting.

Mixup is applied during the model training in each epoch:

```
images, targets_a, targets_b, lam = mixup(images, target.view(-1, 1))
```

As well as applying back-propagation to it after the loss is calculated to allow it to generate more of the samples which the model is misclassifying:

```
mixup_criterion(criterion, model(images), targets_a, targets_b, lam).backward()
```

Backbone models

The models used for training were pretrained backbone models from the *timm* Python module [https://github.com/rwightman/pytorch-image-models/blob/master/results/benchmark-infer-amp-nchw-pt112-cu113-rtx3090.csv]. In particular the following two models were used, which have been trained on the ImageNet dataset:

- NFNet (eca_nfnet_l0) [35]
  This is an implementation of the Brock et al. normalisation free NFNet model [36] from 2021 which has an ImageNet top 1 accuracy of 86% and 24.14 million model parameters.

- NF RegNet (nf_regnet_b1) [https://arxiv.org/abs/2101.00590].
  This is an impelementation of the model described in RegNet: Self-Regulated Network for Image Classification, Xu et al. This had an ImageNe Top-1 accuracy of 76.57%. and contains 10.22 million parameters.

- EfficientNet B3 (tf_efficientnet_b3) ([37] which is based on Tan et al's EfficientNet model [38].
  This has an ImageNet top 1 accuracy of 88.24% and 12.23 million model parameters.

The final classification layer of the models were modified to match the number of classes for the SETI dataset, i.e. 2:

```
        self.model.classifier = nn.Linear(n_features, output_features, bias=True)
```

However apart from that no other modifications were made. This means that the pre-existing model weights were used, which allows the feature extraction that they learned from large datasets to be leveraged for the SETI signal classification task.

Optimisation

In addition to the implementations described above, many different manual variations were tried with different layers for the scratch CNN model. However both the scratch CNN model and the backbone based transfer learning approach had a number of hyperparameters available for tuning. These were automatically tuned using the *Ray Tune* automatic hyperparameter tuner [https://docs.ray.io/en/latest/index.html]. This tries combinations of the various hyperparameters and reports the best performing ones:

The best hyperparameters for the scratch CNN model were found to be [Appendix A]:

- Learning rate: 1.0790248868880749e-10
- Batch size: 4

The best transfer learning for hyperparameters for the ECA NFNet model were:

- Learning rate: 1e-04
- Number of folds: 3
- Weight decay: 1e-5

This was then used as the basis to perform the full training runs for each model. This was done on the cloud with the PaperSpace [paperspace.com] and Google Colaboratory [https://colab.research.google.com/] services to allow concurrent training and access to more powerful GPU's. The resulting notebooks for these training runs are available at:

- Scratch CNN: https://github.com/uolcw/final-project/blob/main/22_09_3_7_custom_CNN.ipynb

- ECA NFNet backbone: https://github.com/uolcw/final-project/blob/main/22-09-3%20-%204%20-%20eca_nfnet_l0.ipynb

- EfficientNet backbone: https://github.com/uolcw/final-project/blob/main/22-09-3%20-%205%20-%20tf_efficientnet_b3.ipynb

- Regnet backbone: https://github.com/uolcw/final-project/blob/main/22-09-3%20-%206%20-%20nf_regnet_b1.ipynb

Ensemble approach

Finally an ensemble method was used that took the majority prediction for each trained backbone as the final prediction. The notebook for this is available at:

https://github.com/uolcw/final-project/blob/main/22-09-4%20-%201%20-%20ensemble.ipynb

Evaluation

Each of the models were evaluated using the following metrics:

- True positive rate (TPR), also called sensitivity or recall
- False positive rate (FPR)
- Accuracy (precision)
- ROC AUC

In addition to this confusion matrices were printed for an overview of the predictions. The validation loss and accuracy were plotted for each training epoch as well.

The following tables give an overview for the scores for the final run with a 30% subset taken from a dataset of 1000 positive and 1000 negative samples.

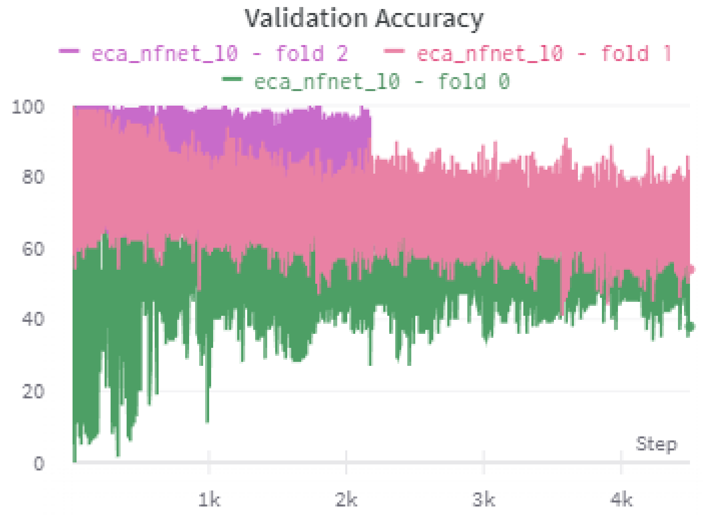
ECA NFNet backbone

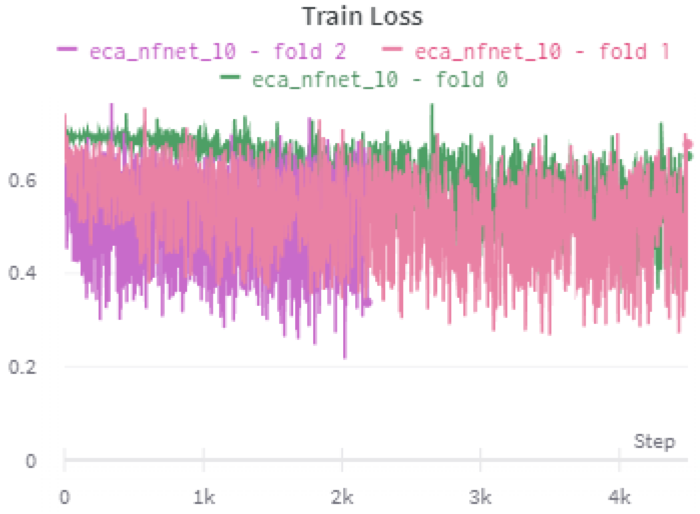
Trained for 300 epochs with 3 folds.

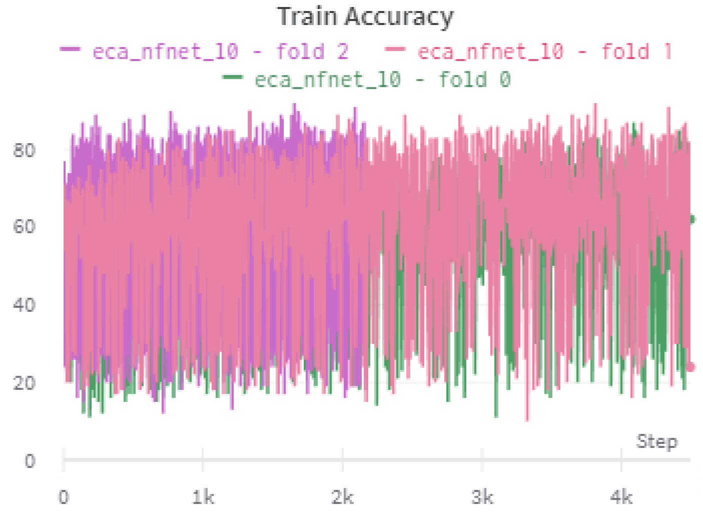
Confusion Matrix		
Label **Prediction**	0	1
0	189	176
1	99	131

Metric scores:

- **TPR**: 0.4267
- **FPR:** 0.3438
- **Precision**: 0.5696
- **ROC AUC**: 0.506


Validation Accuracy
eca_nfnet_l0 - fold 2
eca_nfnet_l0 - fold 1
eca_nfnet_l0 - fold 0
100
80
60
40
20
0
Step
1k
2k
3k
4k


Train Loss
eca_nfnet_l0 - fold 2
eca_nfnet_l0 - fold 1
eca_nfnet_l0 - fold 0
0.6
0.4
0.2
0
Step
0
1k
2k
3k
4k


Train Accuracy
eca_nfnet_l0 - fold 2
eca_nfnet_l0 - fold 1
eca_nfnet_l0 - fold 0
80
60
40
20
0
Step
0
1k
2k
3k
4k

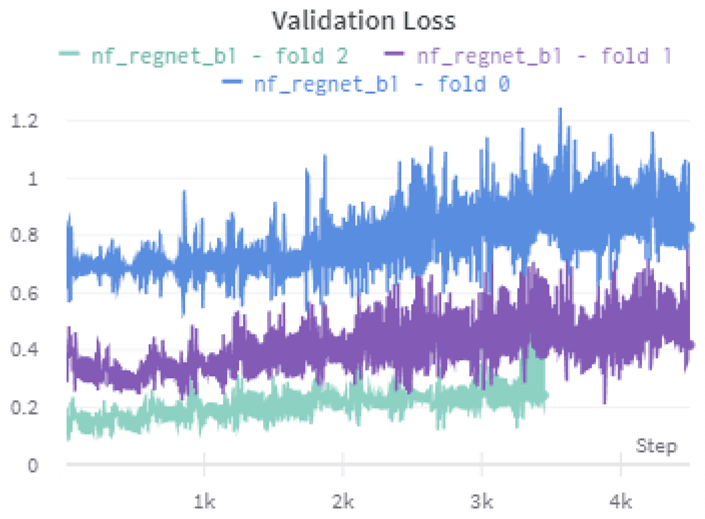
RegNet backbone

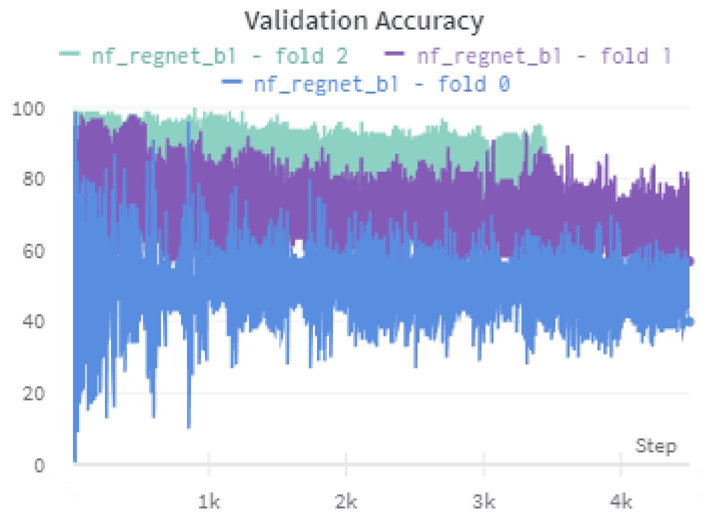
Trained for 300 epochs with 3 folds.

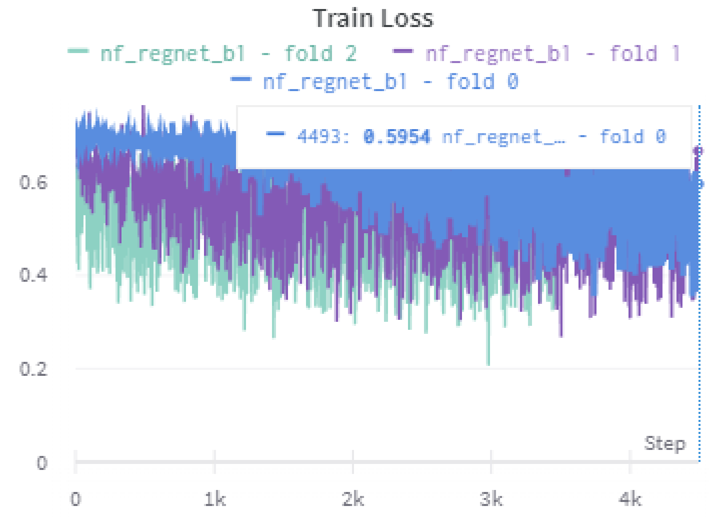
Confusion Matrix		
Label **Prediction**	0	1
0	191	186
1	97	121

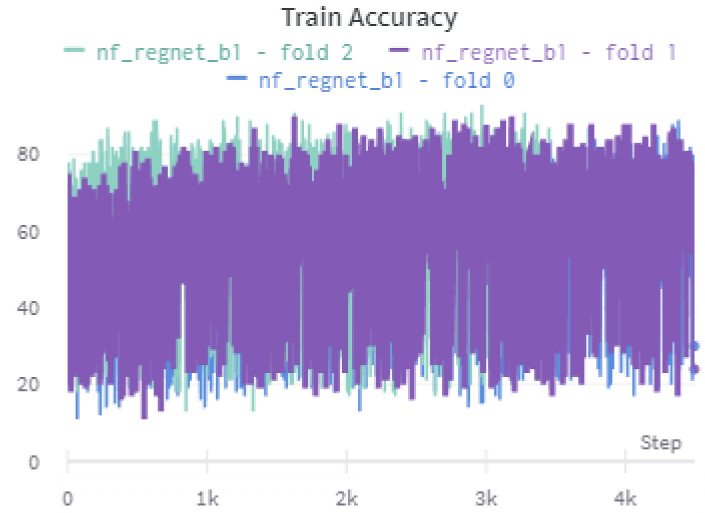
Metric scores:

- **TPR**: 0.3941
- **FPR:** 0.3368
- **Precision**: 0.555
- **ROC AUC**: 0.493









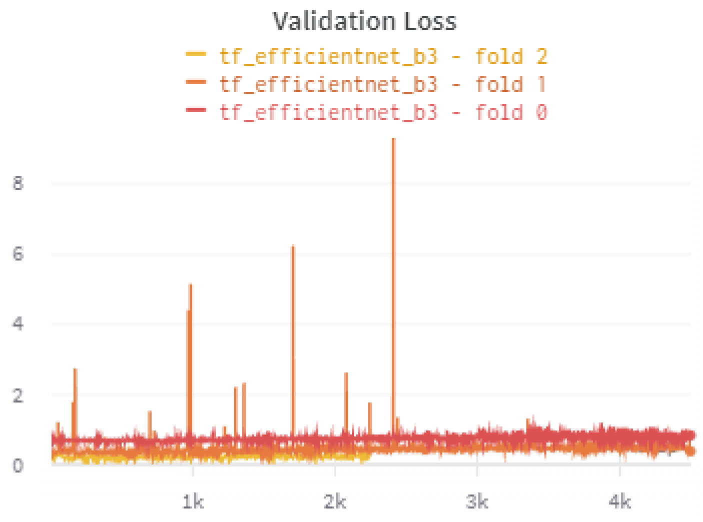
EfficentNet backbone

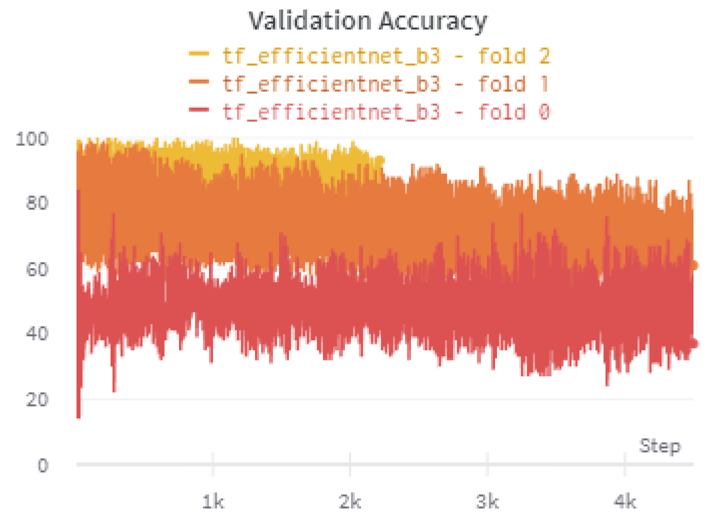
Trained for 300 epochs with 3 folds.

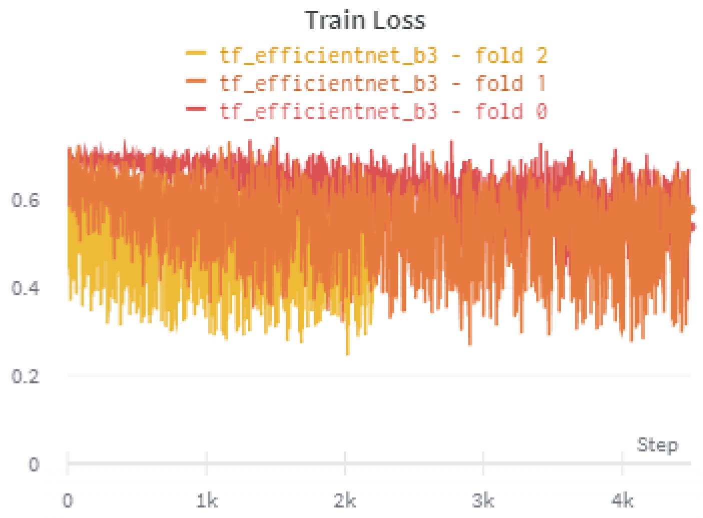
Confusion Matrix		
Label Prediction	0	1
0	191	175
1	97	132

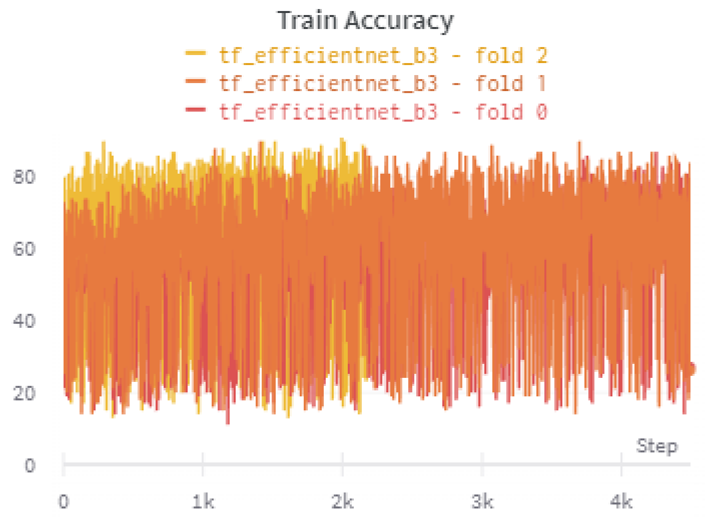
Metric scores:

- **TPR**: 0.43
- **FPR:** 0.3368
- **Precision**: 0.5764
- **ROC AUC**: 0.508


Validation Loss
tf_efficientnet_b3 - fold 2
tf_efficientnet_b3 - fold 1
tf_efficientnet_b3 - fold 0
8
6
4
2
0
1k
2k
3k
4k


Validation Accuracy
tf_efficientnet_b3 - fold 2
tf_efficientnet_b3 - fold 1
tf_efficientnet_b3 - fold 0
100
80
60
40
20
0
Step
1k
2k
3k
4k


Train Loss
tf_efficientnet_b3 - fold 2
tf_efficientnet_b3 - fold 1
tf_efficientnet_b3 - fold 0
0.6
0.4
0.2
0
Step
0
1k
2k
3k
4k



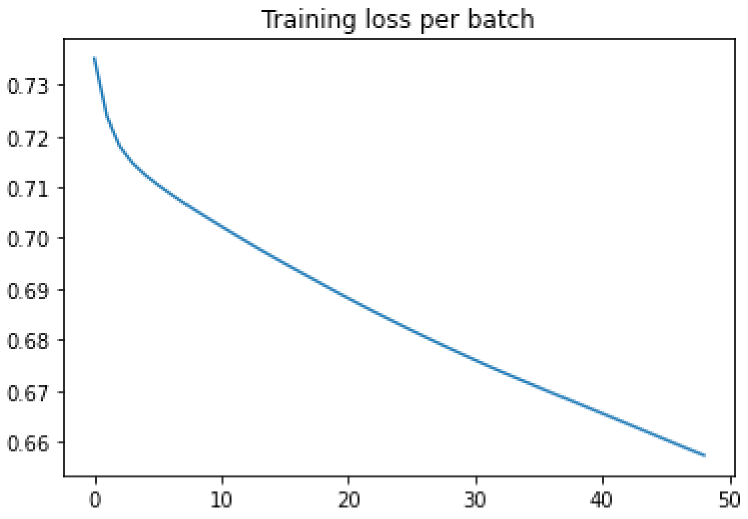
CNN from scratch

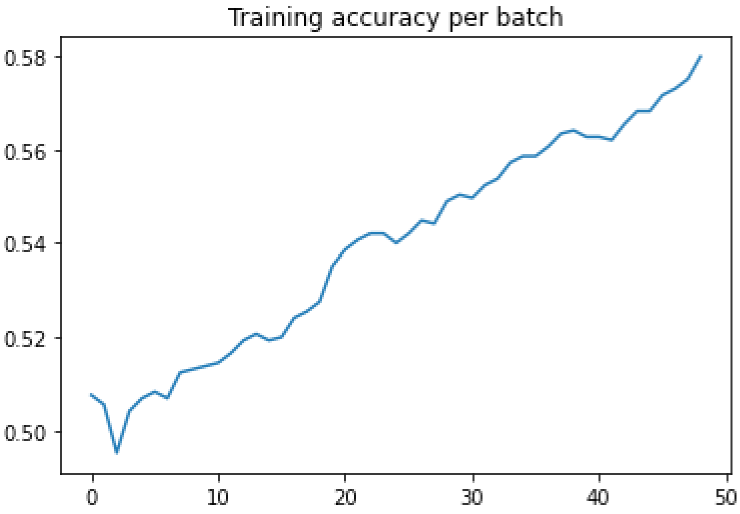
Trained for 50 epochs.

Confusion Matrix		
Label Prediction	0	1
0	269	274
1	22	25

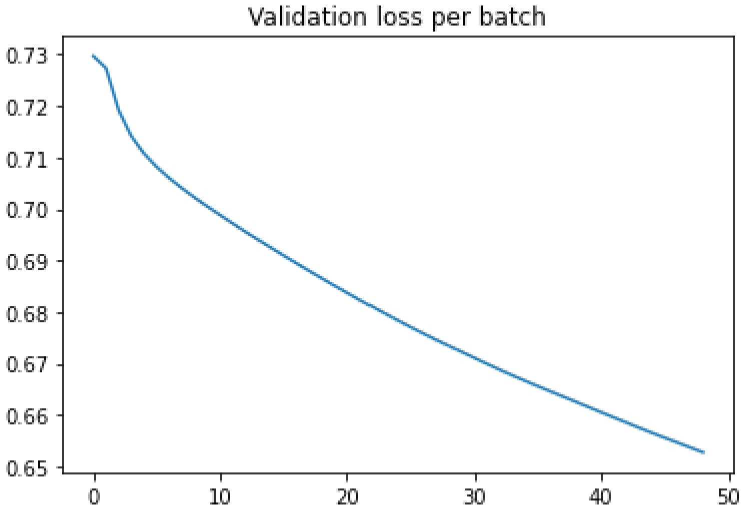
Metric scores:

- **TPR**: 0.0836
- **FPR:** 0.756
- **Precision**: 0.5319
- **ROC AUC**: 0.5

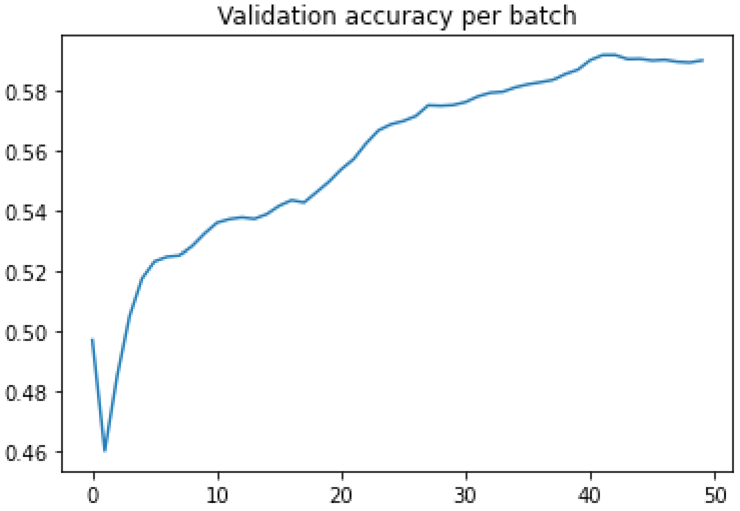




Training accuracy per batch



Validation loss per batch



Validation accuracy per batch

Ensemble

Used the modal prediction from the EfficentNet, RegNet and ECA NFNet backbones.

Results from predictions against the test set.

Confusion Matrix		
Label Prediction	0	1
0	269	274
1	22	25

Metric scores:

- **TPR**: 0.4267
- **FPR:** 0.2438
- **Precision**: 0.5696
- **ROC AUC**: 0.508

Summary

As you can see the backbone models both performed significantly better on the training data than the test data. This indicates that the test data was overfitted. This was mitigated in part by by using test time augmentation such as image transformations and mixup, more can possibly be done to combat this.

For the CNN model built from scratch a similar trend can be seen with the training accuracy climbing faster than the validation accuracy. This also suggests model overfitting.

Another aspect of the results that shows an area of improvement is the overall TPR, FPR, precision and ROC AUC metric scores. Whilst these performed better than random, they were still low compared to the Kaggle competition winners. For example the best ROC AUC score I achieved was 0.508 using the ensemble approach, where as the Kaggle competition winner had an ROC AUC score of 0.967.

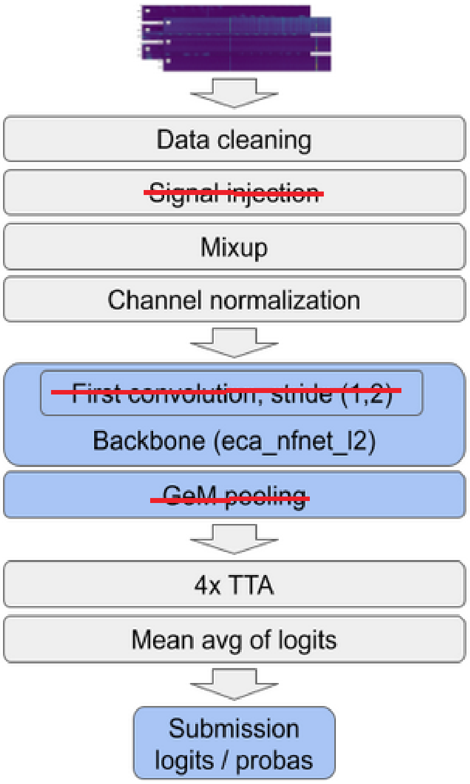
The first place solution also trained over the entire dataset of 60,000 samples. However due to the limited processing power of my machine and a limited budget, I used a smaller subset of 2,000 samples. Also I chose a higher number of epochs and folds since I found that the models were consistently getting stuck in a local minima of loss 0.693 if a smaller number of iterations were used.

Other factors that made my models perform worse than the competition winner are:

- Training for a shorter amount of time
- Using less data augmentation
- Inferior background noise removal technique

Kaggle first place solution architecture comparison

The first place solution on Kaggle made use of the following architecture. Which I tried to emulate, and have crossed out in red the parts that I have not yet been able to implement successfully:



I did attempt to modify the first convolutional layer of the pretrained model to use a stride of (1, 2). This is a trick mentioned in the first place Kaggle solution that increases the image resolution:

```
input_layer_weights = model.model.stem.conv1.weight.cuda()

model.model.stem.conv1 = nn.Conv2d(config['input_channels'], 16, kernel_size=(3, 3), stride=(1,
2), padding=(1, 1))

# restore original layer weights
with torch.no_grad():
    model.model.stem.conv1.weight.copy_(input_layer_weights)

model.cuda()
```

However this resulted in the model getting stuck at a loss of 0.693 during training. So I abandoned this approach, however I would like to revisit this if time permits to see why I was getting these results and if increasing the number of epochs or further denoising will help.

I did not perform GeM pooling and signal injection as originally planned due to time constraints.

Conclusion

Overall performance

In the prototyping phase, there were three main limitations that I came across:

- Memory limitations when using a batch size larger than 5.
- The training and validation loss being stuck in a local minima at the value of 0.693.
- Over 50 minutes taken for each training epoch.

The first two issues were mitigated by upgrading my graphics card to an Nvidia GT 1080 Ti with 11 GB dedicated memory. This allowed me to use the PyTorch CUDA build, which runs the training and validation tasks on the GPU. As a result I was able to increase the batch size to 50, which also allowed the model to learn sufficiently to escape the local minima of 0.693 loss. For the final model training runs I used more powerful hardware available on the cloud, which allowed me to train for a larger number of epochs in a reasonable amount of time. However I was limited by budget for these as the free tiers have an automated shutdown duration and limit the GPU types available.

Although moving the model training and validation tasks to the GPU did greatly reduce the time taken for each epoch, they were still taking ~20 minutes for each training step. The total training data set size was 46.9 GB. This would require an order of days to complete the training. So I decided to use a subset of 2, 000 out of 60, 000 samples. This will also have affected the final model performance as there were fewer examples to to learn from.

For the CNN model from scratch, it can be seen that it was classifying very few positive samples. This is because it only had a depth of 8 hidden layers, which limited its ability to learn features, especially when they are subtle signals with a large amount of background noise. User more layers would have helped with this, as would have training on more samples. However this is what the pretrained backbone models have achieved and are available for free, so there was little point in pursuing this approach further.

Now that I have completed my initial objective of training with a pretrained model which has a higher ImageNet top-1 score. I have a better idea of the avenues that I will investigate in the final projects phase.

I was able to complete my main project objectives, which were:

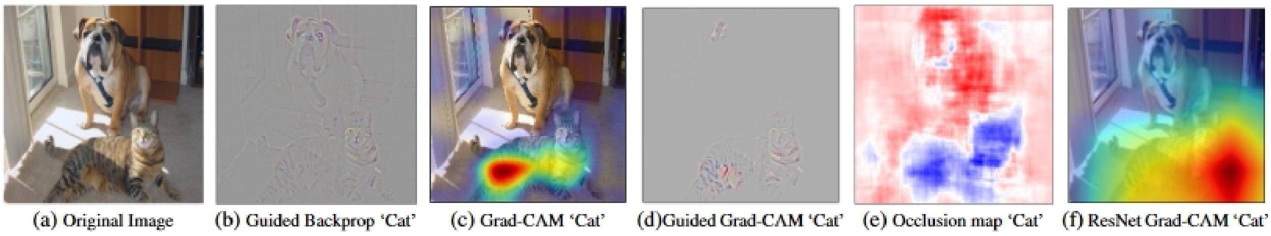
- Build and train a CNN model from scratch
- Train three backbone models and use an ensemble approach
- Use optimisation techniques such as hyperparameter optimisation and image denoising.

Further improvements

To help improve the signal to noise ratio I will into image cleansing techniques, that will remove the background of the images. For example, the first place competition entry did a column by column comparison for each positive sample image with every other sample. Although this is computationally expensive it would lead to much cleaner signals to detect. Furthermore, I will look into signal injection using a technique like Setigen [29]. This will allow me to create more positive samples without background noise that will allow the model to learn more effectively.

Another issue that was seen is that the models performed better on the training data than the test data, a symptom of overfitting. This can be mitigated by training on a larger dataset and using fewer training cycles. Additional techniques such as normalisation and generating samples with signal injection will also help with this.

Another avenue to investigate is use CNN visualisation techniques, which will give me an idea of which features that the model is extracting and may provide insights for improving its performance. One solution for this is Grad-CAM [https://arxiv.org/pdf/1610.02391.pdf] which can highlight the areas that a CNN model uses the most to make a classification:



Grad-CAM visualisations of CNN regions of interset.
[https://arxiv.org/pdf/1610.02391.pdf]

One of the areas that I have not fully explored is plotting more of the data samples. Both for training data and for results that are being misclassified. Doing this may help me to achieve insights that can help improve the model.

Closing remarks

I think with the amount of time and hardware resources available I was able to achieve decent results that are better than random. With more time and hardware recourse, I believe that I would have been able to tune my ensemble model to have a score close to the Kaggle competition winner.

Another limitation was the amount of knowledge I had of both the PyTorch ecosystem and the various machine learning techniques used for image classification. Now that I have a decent grasp of how the techniques in the winning entry are implemented, this will be less of a hurdle in the future.

References

[1] June 17, 2022. SETI. https://www.seti.org/drake-equation-index

[2] June 17, 2022. Yuri Milner. https://yurimilner.com/

[3]
Matthew Lebofsky, Steve Croft, Andrew P. V. Siemion, Danny C. Price, J. Emilio Enriquez, Howard Isaacson, David H. E. MacMahon, David Anderson, Bryan Brzycki, Jeff Cobb, Daniel Czech, David DeBoer, Julia DeMarines, Jamie Drew, Griffin Foster, Vishal Gajjar, Nectaria Gizani, Greg Hellbourg, Eric J. Korpela, Brian Lacki, Sofia Sheikh, Dan

[4] June 17, 2022. Kaggle. https://www.kaggle.com/competitions/seti-breakthrough-listen

[5] June 17, 2022. SETI Breakthrough Listen – E.T. Signal Search | Kaggle. https://www.kaggle.com/competitions/seti-breakthrough-listen

[6] https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc

[7] https://www.kaggle.com/competitions/seti-breakthrough-listen/leaderboard

[8] https://www.kaggle.com/c/seti-breakthrough-listen/discussion/266385

[9]
Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Retrieved June 27, 2022 from http://arxiv.org/abs/1905.11946

[10]
Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V. Le. 2021. Meta Pseudo Labels. Retrieved June 27, 2022 from http://arxiv.org/abs/2003.10580

[11]
G. R. Harp, Jon Richards, Seth Shostak Jill C. Tarter, Graham Mackintosh, Jeffrey D. Scargle, Chris Henze, Bron Nelson, G. A. Cox, S. Egly, S. Vinodababu, and J. Voien. 2019. Machine Vision and Deep Learning for Classification of Radio SETI Signals. Retrieved June 23, 2022 from http://arxiv.org/abs/1902.02426

[12] June 17, 2022. ImageNet. https://www.image-net.org/

[13] June 17, 2022. Hugging Face. https://huggingface.co/timm/eca_nfnet_l0

[14]June 17, 2022. timm models. https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/nfnet.py

[15]
Qilong Wang, Banggu Wu, Pengfei Zhu, Peihua Li, Wangmeng Zuo, and Qinghua Hu. 2020. ECA-Net: Efficient Channel Attention for Deep Convolutional Neural Networks. Retrieved September 5, 2022 from http://arxiv.org/abs/1910.03151

[16]

Andrew Brock, Soham De, Samuel L. Smith, and Karen Simonyan. 2021. High-Performance Large-Scale Image Recognition Without Normalization. Retrieved September 5, 2022 from http://arxiv.org/abs/2102.06171

[17]
Filip Radenović, Giorgos Tolias, and Ondřej Chum. 2018. Fine-tuning CNN Image Retrieval with No Human Annotation. Retrieved June 27, 2022 from http://arxiv.org/abs/1711.02512

[18]
Mingxing Tan and Quoc V. Le. 2020. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Retrieved June 27, 2022 from http://arxiv.org/abs/1905.11946

[19] https://www.kaggle.com/competitions/seti-breakthrough-listen/discussion/266397

[20]
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. Retrieved June 27, 2022 from http://arxiv.org/abs/1512.

[21]
Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. 2019. Squeeze-and-Excitation Networks. Retrieved June 27, 2022 from http://arxiv.org/abs/1709.01507

[22]
Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V. Le. 2020. Self-training with Noisy Student improves ImageNet classification. Retrieved June 27, 2022 from http://arxiv.org/abs/1911.04252

[23]
Bo-Ching Lin, Hwai-Jung Hsu, and Shih-Kun Huang. 2020. Testing Convolutional Neural Network using Adversarial Attacks on Potential Critical Pixels. In 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, Madrid, Spain, 1743–1748. DOI:https://doi.org/10.1109/COMPSAC48688.2020.000-3

[24] June 17, 2022. Open AI Adversarial Example. https://openai.com/blog/adversarial-example-research/

[25] June 17, 2022. Wikipedia Watershed (image processing). https://en.wikipedia.org/wiki/Watershed_(image_processing)

[26]
G. R. Harp, Jon Richards, Seth Shostak Jill C. Tarter, Graham Mackintosh, Jeffrey D. Scargle, Chris Henze, Bron Nelson, G. A. Cox, S. Egly, S. Vinodababu, and J. Voien. 2019. Machine Vision and Deep Learning for Classification of Radio SETI Signals. Retrieved September 5, 2022 from http://arxiv.org/abs/1902.02426

[27] Deep Learning with Python, Second Edition. F. Chollet

[28]
Lillian Le, Abu Nadim M.H. Kabir, Chunyan Ji, Sunitha Basodi, and Yi Pan. 2019. Using Transfer Learning, SVM, and Ensemble Classification to Classify Baby Cries Based on

Their Spectrogram Images. In 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), IEEE, Monterey, CA, USA, 106–110. DOI:https://doi.org/10.1109/MASSW.2019.00028

[29]
Bryan Brzycki, Andrew P. V. Siemion, Imke de Pater, Steve Croft, John Hoang, Cherry Ng, Danny C. Price, Sofia Sheikh, and Zihe Zheng. 2022. Setigen: Simulating Radio Technosignatures for the Search for Extraterrestrial Intelligence. AJ 163, 5 (May 2022), 222. DOI:https://doi.org/10.3847/1538-3881/ac5e3d

[30] https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[31] https://docs.ecognition.com/eCognition_documentation/User%20Guide%20Developer/8%20Classification%20-%20Deep%20Learning.htm

[32] June 17, 2022. Project Jupyter | Home. https://jupyter.org/

[33] June 17, 2022. PyTorch. https://pytorch.org/

[34] June 17, 2022. TensorFlow. https://www.tensorflow.org/

[35] June 17, 2022. Matplotlib. - Visualization with Python. https://matplotlib.org/

[36] June 17, 2022. Weights and Biases. https://wandb.ai/site

Appendix

[A] Automatic hyperparameter tuning

```
+------------------------------+------------+-----------------+--------------+-------------+-------------
+------------+---------------------+
| Trial name                   | status     | loc             |   batch_size |          lr |        loss |
accuracy |   training_iteration |
|------------------------------+------------+-----------------+--------------+-------------+-------------
+------------+---------------------|
| train_and_validate_e8fb5_00000 | TERMINATED | 127.0.0.1:13152 |           64 |
0.0877714   |  718.889    |   0.53006  |                    5 |
| train_and_validate_e8fb5_00001 | TERMINATED | 127.0.0.1:16648 |           16 |
9.42906e-07 |    0.695561 |   0.482895 |                    1 |
| train_and_validate_e8fb5_00002 | TERMINATED | 127.0.0.1:14640 |            8 | 2.26772e-
06 |    0.69231  |   0.509868 |                    5 |
| train_and_validate_e8fb5_00003 | TERMINATED | 127.0.0.1:12020 |           16 | 0.262037
| 2831.06     |   0.499342 |                    2 |
| train_and_validate_e8fb5_00004 | TERMINATED | 127.0.0.1:7024  |            4 | 6.38297e-
07 |    0.693878 |   0.536667 |                    1 |
| train_and_validate_e8fb5_00005 | TERMINATED | 127.0.0.1:3732  |           64 |
0.0196495   |    4.19667  |   0.549851 |                    5 |
| train_and_validate_e8fb5_00006 | TERMINATED | 127.0.0.1:13672 |           16 |
0.00092736  |    0.694761 |   0.479605 |                    1 |
| train_and_validate_e8fb5_00007 | TERMINATED | 127.0.0.1:16728 |            4 | 1.21288e-
07 |    0.695625 |   0.48     |                    1 |
| train_and_validate_e8fb5_00008 | TERMINATED | 127.0.0.1:12932 |            4 | 1.07902e-
10 |    0.691361 |   0.5      |                    5 |
| train_and_validate_e8fb5_00009 | TERMINATED | 127.0.0.1:2112  |           64 |
0.00619404  |    0.75502  |   0.470387 |                    5 |
| train_and_validate_e8fb5_00010 | TERMINATED | 127.0.0.1:14476 |            8 | 5.36519e-
06 |    0.692285 |   0.552632 |                    5 |
| train_and_validate_e8fb5_00011 | TERMINATED | 127.0.0.1:12736 |           64 |
1.36131e-09 |    0.698244 |   0.537202 |                    1 |
| train_and_validate_e8fb5_00012 | TERMINATED | 127.0.0.1:9792  |           64 | 5.55845e-
06 |    0.69558  |   0.523065 |                    1 |
| train_and_validate_e8fb5_00013 | TERMINATED | 127.0.0.1:16788 |            8 | 2.30026e-
10 |    0.694329 |   0.536184 |                    1 |
| train_and_validate_e8fb5_00014 | TERMINATED | 127.0.0.1:15124 |           64 |
6.51919e-05 |    0.693203 |   0.487649 |                    5 |
| train_and_validate_e8fb5_00015 | TERMINATED | 127.0.0.1:16964 |           64 |
0.0425503   |  132.298    |   0.48244  |                    2 |
| train_and_validate_e8fb5_00016 | TERMINATED | 127.0.0.1:2112  |            8 | 0.756471
|    0.701297 |   0.473684 |                    1 |
| train_and_validate_e8fb5_00017 | TERMINATED | 127.0.0.1:16724 |           64 |
0.0102669   |    3.78987  |   0.49494  |                    2 |
| train_and_validate_e8fb5_00018 | TERMINATED | 127.0.0.1:16708 |           16 |
1.32269e-05 |    0.699064 |   0.509211 |                    1 |
| train_and_validate_e8fb5_00019 | TERMINATED | 127.0.0.1:3912  |           64 | 3.46018e-
07 |    0.693625 |   0.513542 |                    4 |
+------------------------------+------------+-----------------+--------------+-------------+-------------
```

```
+------------+----------------------+


Best trial config: {'lr': 1.0790248868880749e-10, 'batch_size': 4}
Best trial final validation loss: 0.6913613645235698
Best trial final validation acc: 0.5
```