BASELINE IS ALL YOU NEED

COMP9417 PROJECT OF 2021 T2

Yiyang Huang
z5313425@ad.unsw.edu.au

Chelsea Chaffey
z5311727@ad.unsw.edu.au

1 Introduction

Caused by the violent collision of black holes or neutron stars, gravitational waves are tiny ripples in spacetime that race across the universe at the speed of light. In September 2015, the Laser Interferometer Gravitational-Wave Observatory (LIGO) made the first detection of a gravitational wave, with an amplitude a thousand times smaller than the nucleus of an atom[1]. While this discovery was a scientific triumph, the detection of more subtle gravitational waves remains a difficult challenge.

Gravitational signals are buried in detector noise, and require substantial filtering to become observable. In addition, detectors produce vast quantities of data that take extensive resources to analyse. The G2Net Kaggle competition aims to explore and develop efficient machine learning processes that reliably detect gravitational wave signals.

Motivation Developing ML algorithms and models for gravitational wave (GW) detection could boost the sensitivity of detectors, aid in noise removal and signal characterization, and lead the development of next-generation experiments. As gravitational wave detection enables physicists to directly observe the presence of invisible, supermassive stellar remnants, these signals provide the opportunity to test how general relativity behaves under extreme conditions and to measure the rate of the universe's expansion. Applying Machine Learning methods in this field has the real potential to further the frontiers of physics - and the insights gleaned could profoundly shape our knowledge of the universe's origins, as well as its ultimate fate.

Summary In this report, we explore methods of processing and reshaping the raw data to best detect gravitational waves amongst the detector noise. We then apply and analyse the performance of three baseline models individual samples - Logistic Regression, Decision Trees, and Gaussian Naive Bayes. Finally, we investigate the image-tailored application of Convolutional Neural Networks and Efficient Net and its performance in gravitational wave detection. Link to results and code: `https://drive.google.com/drive/folders/14DaB4tXGXbt55a4U5OJZEZ1WXDIydzmQ?usp=sharing`.

2 Related Literature

Logistic Regression Logistic Regression is a popular baseline that performs probabilistic linear classification. We choose logistic regression as a benchmark in this report to further analyze the characteristics of the dataset and behavior of other baseline models.

Decision Tree Tree induction [7], given by its high variance and low bias, it is somewhat complementary to the low variance and high bias traits of logistic regression. However its restricted search in the model space allows it to obtain more nonlinear patterns of the dataset.

Gaussian Naive Bayes Since we are dealing with continuous data, we choose to use Gaussian Naive Bayes as another baseline for further analysis. It's efficient training and inference compare to tree inductions and logistic regression invokes us to discuss the trade-off between accuracy and efficiency.

Convolutional Neural Networks Convolutional Neural Network (ConvNets) [10] is considered as the major break through in Computer Vision and dominates the field for decades. We also investigate performance of fundamental ConvNets on our dataset, while viewing a sample as an image.

ResNet The evolutionary model ResNet [3] resolves the degradation of convolutional neural network when it is scaled substantially in depth. Consequently, it enhances the ability of traditional deep learning method to explore more high level features as number of layers increased.

EfficientNet EfficientNets [9], which is a family of ConvNets that are scaled not only in depth, but scaled in both resolution and width of the network in contrast to ResNet that only deepen the model. This adaptive nature of EfficientNet ensure its popularity in the deep learning community and widely used in the Kaggle competition.

3 Data Exploration and Processing

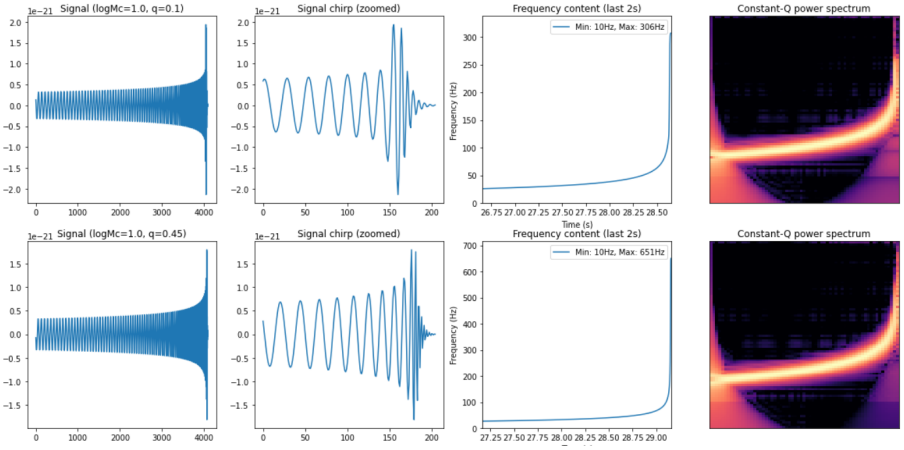
The G2Net Competition [2] provides competitors with a large (70 GB) set of time series data. We are provided 560000 labelled training samples, which classify the sample by whether a gravitational signal is present (target = 1) or absent (target = 0). Each data sample contains three time series measurements, which are simulated from three independent gravitational wave interferometers: LIGO Hanford, LIGO Livingston, and Virgo. Each time series contains either detector noise, or detector noise with a simulated gravitational wave signal applied on top. Our model must identify when a signal is present in the data.

Due to the dataset's immense size, we decided to randomly sample 5000 stratified features (0.5GB) to form the basis for our training and test data. First, we researched the form of gravitational waves and performed some exploratory data analysis. In this section, we also investigated a range of preprocessing techniques for waveform data to investigate its shape and improve learning outcomes.

3.1 Gravitational Waves

Gravitational waves are produced by colliding massive astrophysical objects. As the objects rotate rapidly around each other, the signal oscillates smoothly, before rapidly ascending to a peak and disappearing. As visualized in figure 1, the peak, or "chirp", is the most identifiable part of the signal.

Figure 1: Gravitational Wave Simulation [**gravity_wave_simulation**]



The frequency at which the gravitational waves oscillate is dependent on the mass of the system involved, with the chirp varying rapidly from 50Hz to over 5000Hz. However, as the first gravitational wave detected (LIGO, 2015) had a chirp of frequency 250Hz [1], we estimate that most of the training examples will fall around this boundary.

3.2 Standardization & Normalization

Standardization and Normalization are both regular preprocessing techniques in the field of machine learning. Normalization maps values into a specific range, which we chose to be [-1, 1] since gravitational waves oscillate around zero. Standardization maps values to the Gaussian distribution, where we transform the data to have a unit standard deviation.

We employed four variations of standardization and normalization in our initial data exploratory process.

1. **Set Normalization** - rescaling all data samples together. Preserves differences in magnitude between detectors.
2. **Channel Normalization** - rescaling data from each detector separately. Eliminates differences in magnitude between detectors.
3. **Set Standardization** - Standardizing all data samples together. Preserves differences in magnitude between detectors.
4. **Channel Standardization** - Standardizing data from each detector separately. Eliminates differences in magnitude between detectors.

A visualization of various combinations of these approaches on two data samples can be found in the appendix (Figure 12). After this, we experimented with the effectiveness of regular preprocessing on improving the performance of the dataset on 5-fold CV L2 Logistic Regression.

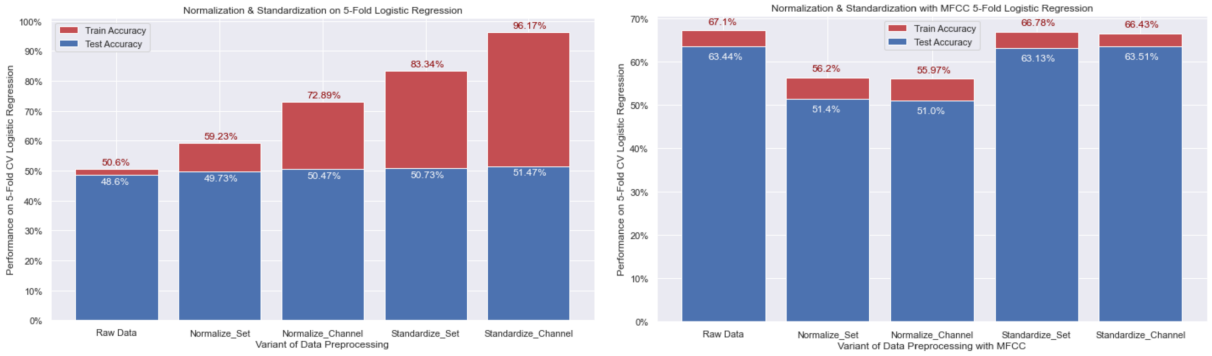


Figure 2: Performance of Standardized and Normalized Data on Logistic Regression. Left image: Processing methods on raw data. Right image: mfcc is applied post-processing.

As seen in Figure 3.2, when these processing techniques are applied on raw data, the test accuracy of logistic regression remains stagnant at 50%. Moreover, the train accuracy of Logistic regression increases substantially, which indicates these processing techniques in fact lead to greater overfitting. We also noticed that the time taken for the model to converge increased almost a hundredfold. A cause of this might be that techniques like standardization may increase collinearity between variables that are already highly collinear, as in the case of oscillating waveform data, thus making the task of logistic regression more difficult.

However, when the processed data is then MFCC transformed, the results become more interesting. Normalization clearly decreases the performance of the model, whereas Standardization creates little difference to the outcome. The poor performance associated with Normalization indicates that mapping features to pre-defined ranges eliminated information that could otherwise reveal the signal's presence. This is unsurprising since detecting gravitational signals relies on certain frequencies having a higher amplitude, which can be alterned in normalization.

It is clear that simple preprocessing methods are ineffective for this dataset. As such, we conducted extensive research into how gravitational waves are detected, and preprocessing techniques tailored to the task of signal analysis.

3.3 Noise in gravitational instruments

The Amplitude Spectral Density (ASD) describes the distribution of a signal's strength into its corresponding frequency components. Because gravitational instruments are highly sensitive, they record a substantial amount of noise across a range of frequencies. Understanding the distribution of noise across the ASD is crucial for developing a preprocessing method for gravitational wave data.

Figure 3: Noise recorded by LIGO Detectors across ASD

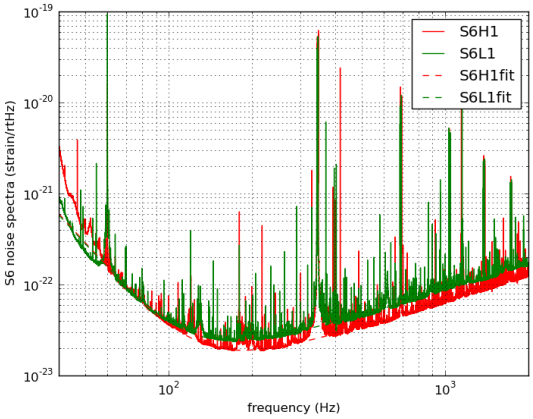


Figure 4 indicates the Amplitude Spectral Density for both LIGO detectors. The underlying smooth curve forms the 'noise floor', below which a significant amount of noise is regularly detected. The visible peaks in the graph are believed to be caused by instrumental effects, including seismic noise, imperfect electronic shielding, and magnetic coupling.

We will investigate three preprocessing methods:

1. **Amplitude Spectral Density (ASD) Whitening.** This technique aims to reduce the low frequency noise that dominates the visible signal.
2. **Bandpass Filter**. This technique will remove low and high frequency signals, which are less likely to indicate the collision of black holes.
3. **Mel-Frequency Ceptral Coefficient (MFCC)**. This technique is a method of feature extraction that converts a signal to its short-term power spectrum.

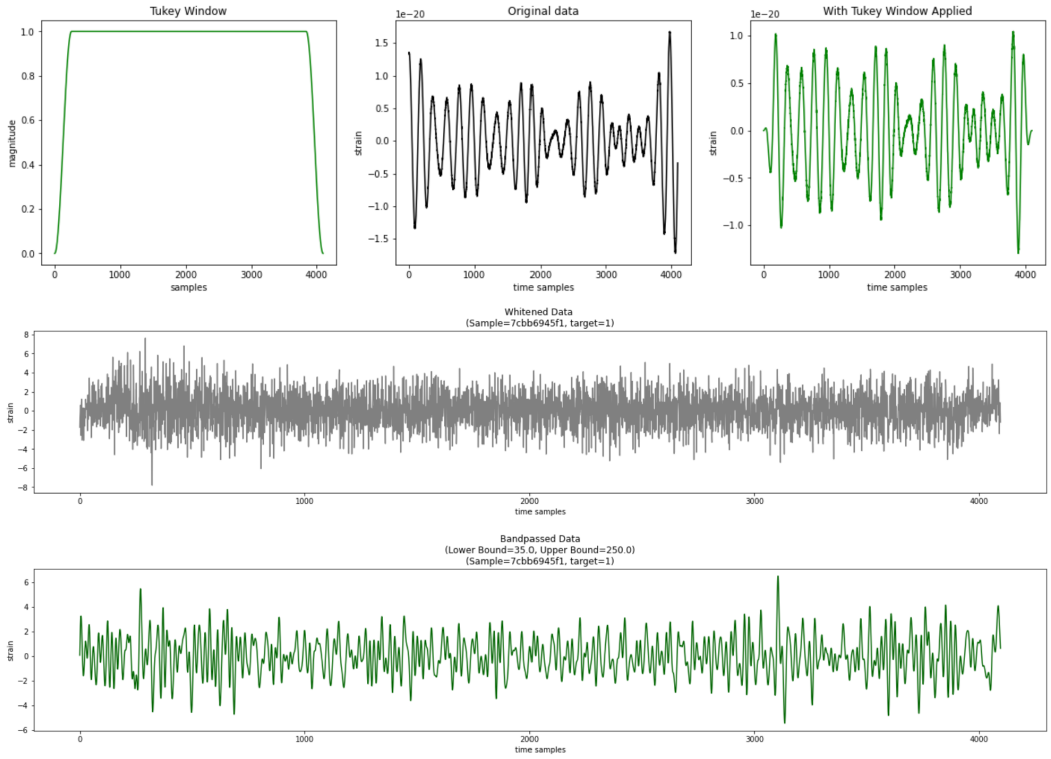
3.4 Amplitude Spectral Density Whitening

The data is dominated by low frequency noise, which is seen in the large oscillations in the visualized training data.We implemented our own variation of Amplitude Spectral Density (ASD) Whitening, a technique that uses **Fast Fourier Transform (FFT)** to efficiently reduce noise.

The first step is to pass the raw data through a Window Function, reducing the phenonemon of Spectral Leakage. This unwanted sideffect of applying FFT to a finitely windowed data sample causes additional lobes to be inserted into the DFT spectrum. We experimented with the effectiveness of both the **Blackman Window** and the **Tukey Window** in our data preprocessing stage. We believe the Tukey Window is more effecitve overall because its unique shape and steep gradient allows most of the original amplitudes to be preserved for further signal identification. Data transformed with this window achieved a slightly higher test accuracy across the baseline models.

Secondly, we transformed the data with FFT, a ubiquitous algorithm that expresses the data as an array of frequency components. After applying a phase shift and time shift, we calculated the Power Spectral Density (PSD) on the raw data sample. We then divided the Fourier Transform of the normalized Power Spectral Density, which will down-weight frequencies with high noise. Finally, we computed the Inverse Fourier Transform to return to the original time domain.

Figure 4: Visualization: ASD Whitening and Bandpass (sample=7cbb6945f1)



3.5 Bandpass Filter

A **bandpass filter** is a signal filter that allows only the desired frequencies to pass through. While the lower frequencies are removed by ASD Whitening, higher frequencies dominate the whitened sample, making data analysis difficult. We used the **Butterworth Filter**, which is designed to be 'maximally flat', and to avoid creating synthetic ripples in the filtered data. This filter is also commonly used digital filters in audio circuits and motion analysis, implying its viability.

However, the bandpass filter introduces two new hyperparameters: the lower bound and the upper bound. Because of the distinctive chirp shape of the gravitational signal, we decided the upper bound is more crucial to performance on the processed dataset. If the upper bound is too low, the telltale chirp may be lost. However if it is too high, excess higher frequency noise will make the signal harder to distinguish.

In the very first gravitational wave detection (LIGO, Sept 2015), the signal oscillated at 35Hz, before peaking at 250Hz in the distinctive gravitational "chirp" [1]. Using these values as guides, we set up an experiment that measured how the upper bond affects the train and test performance of a low variance L2 Linear Regression model. This experiment will be discussed later in the report.

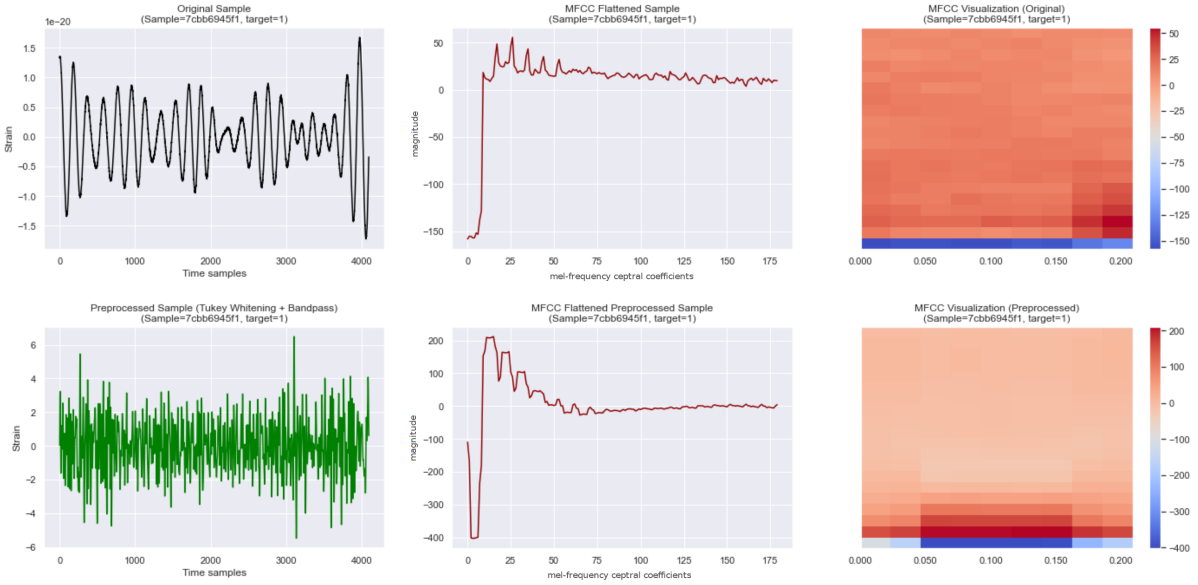
3.6 Mel-Frequency Cepstral Coefficient (MFCC) Processing

Widely used in machine learning for speech recognition and audio filtering, MFCC is the coefficient representation of a sound's short term power spectrum. In summary, the data sample is mapped to its Fourier Transform, and triangular overlapping windows are employed to map the spectrum's powers to the logarithmic Mel Sale. Then, the log of the power at each frequency is taken. Finally, Discrete Cosine Transform (DCT) is computed on these logs, and the amplitudes are returned as the resulting MFCC.

We used the Librosa package's implementation of MFCC to transform each sample. On our dataset, MFCC reduces the feature space from 4096 to 150, improving performance and convergence across all baselines, particularly logistic

regression. This is likely because it avoids the high collinearity between features present in an oscillating strain, which in literature has been shown to lead to arbitrary coefficient selection and lower performance.

Figure 5: Visualization: MFCC (sample=7cbb6945f1)



As observed in Figure 5, the shape of the MFCC on the original data signal is fairly different to the shape on the whitened and bandpassed data signal. The lower plateu and less jagged plot on the whitened MFCC is likely caused by the filtering of low and high frequencies in the processing stage. We found that MFCC is an effective feature extraction tool for the baseline models, and used this algorithm extensively to compliment our other pre-processing techniques.

4 Hyperparameter Optimization

4.1 Bandpass Optimization

We ran an experiment to consider the optimization of the upper bound of Bandpass Filtering. We used Logistic regression as it is very stable with a high bias and low variance, which makes it an effective mode of comparison.

While there was no clear peak for the upper bound, we observed that as the bound approached 250, the difference between test and training accuracy became smaller. This implies the data passing through this filter may be more reflective of the underlying data distribution. It also supports the research that many gravitational 'chirps', the most distinctive part of the signal, occur at around 250Hz, making a cutoff at this range quite effective.

4.2 Optimization of Regularization Constant

While the feature-rich nature of the data suggests that the L1 penalty may be appropriate, practical experimentation demonstrated that an L1 logistic regression was far less time efficient, while producing only marginally better results.

We also investigated whether there is an optimal parameter 'C' for L2 logistic regression.

As seen in the Figure 7, there appears to be little correlation between the constant of regularization and the performance of the model. Smaller constants provide similar, if not slightly better, performance than larger constants, and converge at a far greater speed.

The performance of small C values shows that even when the model is highly regularized, the model's performance remains steady. This implies that the loss function may be making arbitrary

Figure 6: Searching for Optimality: How Bandpass Upper Bound affects Logistic Regression

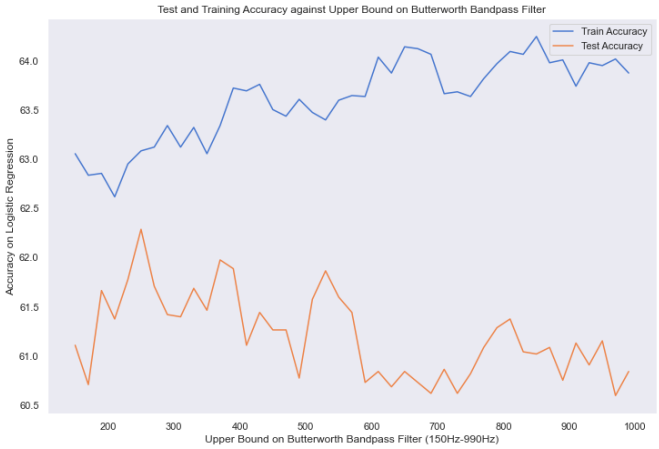


Figure 7: Searching for Optimality: How Constant of Regularization affects Logistic Regression

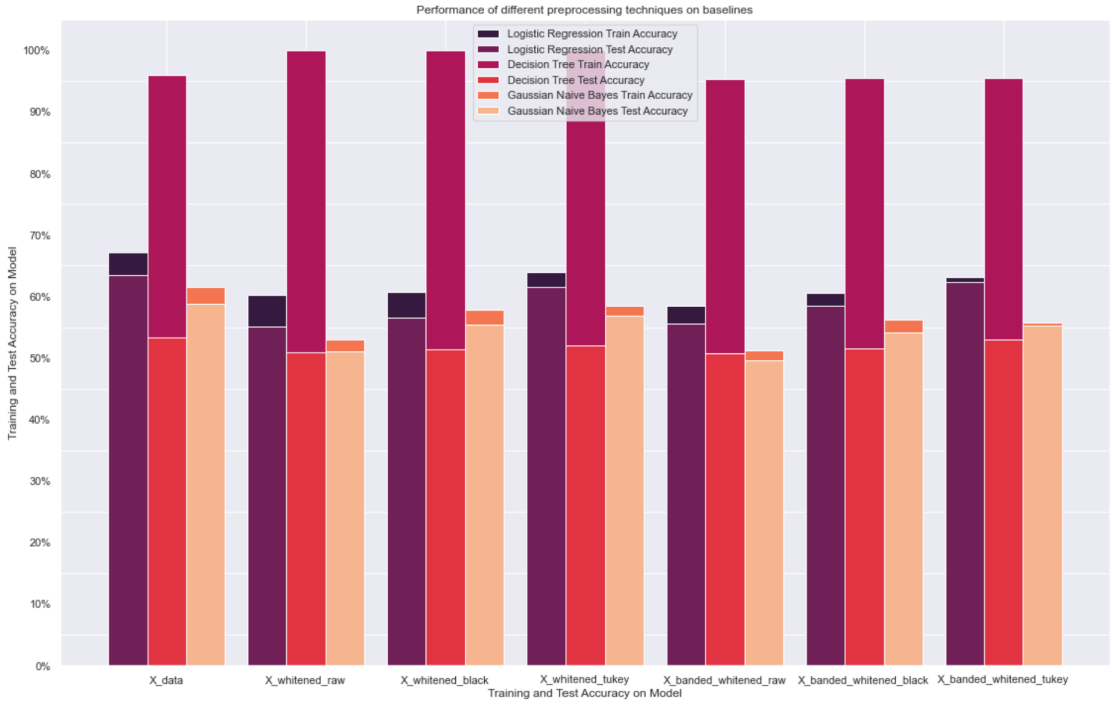
For these reasons, we decided to choose C = 0.1 as a regularization constant for further L2 Logistic Regression GridSearch applications.

5 Baselines

We compared the performance of variants of preprocessed data against three baseline algorithms: L2 Logistic Regression, Decision Tree Classifier, and Gaussian Naive Bayes. All datasets were transformed with MFCC before training, and run with 5-fold Cross Validation. The purpose was to measure the effectiveness of different preprocessing techniques.

From the results in Figure 8, the original dataset, X_data, gave the highest test accuracy performance overall. However, the dataset X_banded_whitened_tukey gave comparable results, with a smaller difference between test and training accuracies indicating less overfitting. It appears that our employment of popular wave signal processing techniques such as whitening and bandpassing was not significantly effective on performance of the baselines. This could be due to information being lost during the process, or due to being constrained to a relatively small test/training set of 5000 samples. It could also be a result of baseline approaches being less ineffective at dealing with a very large feature space in general, and thus being less able to take advantage of whitening and bandpassing benefits. We do observe that that while windows reduce performance before whitening, they raise performance afterwards. It is also clear that the Tukey Window is more effective than the Blackman window.

Figure 8: Preprocessing Techniques on Baseline Performance



5.1 Logistic Regression

Logistic Regression is a low variance and high bias algorithm that provides stable results, confirming its status as an effective baseline. We used the L2 penalty, as exploratory experimentation with L1 showed an unworkably long runtime with very little benefit in test accuracy. We also used the optimized C parameter of 0.1 from our previous experimentation, which indicates high regularization.

In regards to Figure 8, Logistic Regression is by far the best performing baseline, reaching a test accuracy of over 63%. Although we were able to achieve a reliable results, we noticed the model did not converge at times. One reason might be high collinearity between the features. This means feature selection will make an arbitrary choice on which feature to select, leading to a high degree of overfitting. While we increased regularization to account for this, it may still have an impact.

5.2 Decision Tree

We applied Decision Tree to investigate the performance of datasets on a high variance low bias model. We did not apply pruning on decision trees due to the failed attempt of feature selection for logistic regression, thus we want to ensure all baseline models face the same amount of features and noise.

From Figure 8, the test accuracy of all dataset performed by decision tree are lower than the accuracy of logistic regression, which is coherent with the observation that logistic regression generally performs better on small dataset (given that we are only training on 5000 samples) than tree inductions [8].

Overall, decision trees acquires the worst performance on most of the test sets. We conclude the following possible reasons for this poor performance:

1. Our hyperparameter min_sample_leaf equals to 1, returns by the GridSearchCV. Thus causing the generalization ability of the model to further reduced.
2. The tree is un-pruned to ensure the same amount of feature and noise are considered by decision tree and logistic regression.

5.3 Gaussian Naive Bayes

We applied Gaussian Naive Bayes due to the continuous nature of time-series data we were given. Gaussian Naive Bayes is a widely used Naive Bayes Classifier where we assume that each feature of the data comes from a class conditional Gaussian distribution.

The class-conditional densities are given by:

$$\hat{h}(x) = \arg\max_c P(y = c \mid x, \mu_c, \Sigma_c),$$

where $\hat{h}(x)$ is the predicted class, and $c \in C$ is all the possible class, μ_c is the class-specific mean vector and Σ_c is the class-specific covariance matrix [5].

In terms of accuracy, Gaussian Naive Bayes have moderate performance comparing to other baselines we had experimented (Figure 8). Yet, the difference between training and testing accuracy is also the lowest, suggesting that the model is simply under-fitted, unlike the Decision Trees that displays an enormous variance.

We also observed that the speed of convergence is the fastest comparing to other baselines.

6 Convolutional Neural Networks

Given the shape of a sample in G2Net is (3, 4096), which remains the same after being processed with ASD and bandpass filters. The similar behavior of the baselines (Figure 8) provoke us to perceive the given sample as an single-channel image of size (1, 3, 4096) instead of splitting them into three samples of size (1, 4096) as performed in section 4.

Hence, in this section, we empirically observe the incompetence of vanilla convolutional neural network to extract features from inputs with extreme scaling. We further conduct experiments to explore the performance of deeper neural networks such as ResNet in capturing higher level features for time-series data. We then compare the results of ResNet to EfficientNet, which is popular within the Kaggle community and study how the compound scaling would affect the training results.

Results from this section are compared between the original dataset and the transformed dataset (whitened and banded tukey) that gives the best performance on logistic regression.

Even though the results of EfficientNet, ResNet and our own ConvNet are all not satisfying, but it indicates the effect of bandpass filtering and whitening the data, which demonstrates a significant difference in behavior to the raw data and reflects it impact on performance more evidently than the baseline experimenting.

6.1 Binary Cross Entropy Loss

All 'loss' mentioned in section 5 are Binary Cross Entropy Loss (BCELoss), implemented by pytorch as BCEWithLogitsLoss which incoperates the Sigmoid function and the BCELoss.

The BCE loss function is given by:

$$L_N = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log\left(\sigma(x_n)\right) + (1 - y_n) \cdot \log\left(1 - \sigma(x_n)\right)\right],$$

where N is the size of the batch. We choose this loss function because it suits the binary classification problem we are facing. The range of BCE loss is $[0, \infty)$

6.2 Our ConvNet

We consider our own ConvNet as a benchmark for our further investigation in the family of convolutional neural networks.

6.2.1 Raw Data

Our basic convolutional neural network only contains 1 convolutional layer with 3 filters where the kernel is in size of 3 by 3, a stride 1 and padding that maintains the dimension of the input, following by a maxpooling layer with window size of 2. The ConvNet is then followed by 4 linear layers and sigmoid function within the loss.

This singleton structure is due to the nature of ConvNet, where the spatial dimensions gradually shrinks but the channel dimension expands overtime. Given that our sample only has a height of 3, any extra non-padded operation will cause the sample to become a vector instead of matrix, hence to avoid this, we decide to experiment on another deeper ConvNet (See Appendix 8.3).

The more ConvNet 2 we designed in this report is still a simple ConvNet, but allows the reshaped sample of size (96, 128) to be trained to a more depth, and attempt to learn mid-level features of the input with its limited depth. We consider max pooling instead of average pooling due to the fact that most values inside the tensor are similar. We also attempt to implement a batch normalization to the ConvNet, which did not give any promising improvements in the losses, hence we did not include it in this implementation of ConvNet.

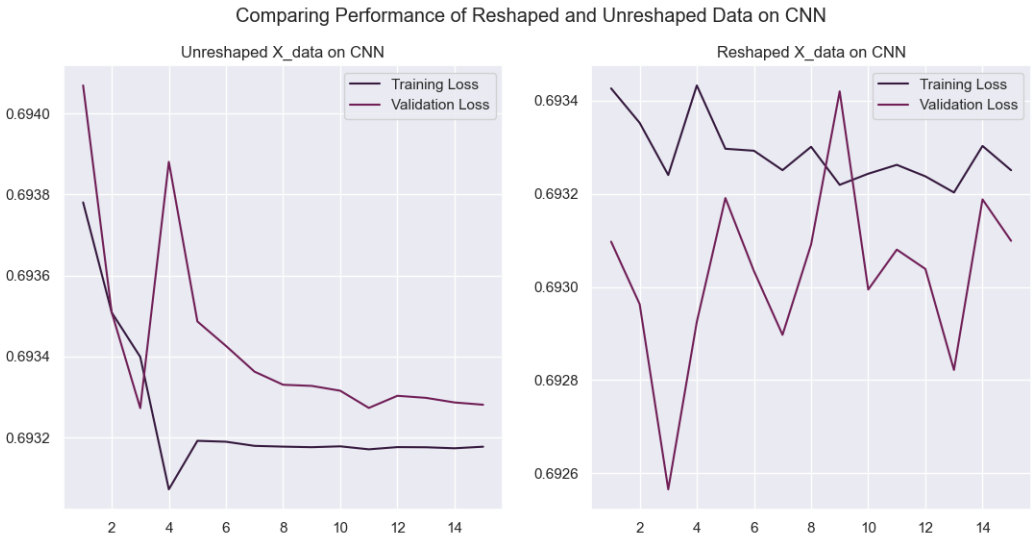


Figure 9: Right: Reshaped raw data X of size (96, 128) and trained on the complex ConvNet with learning rate 0.0001 and optimizer RMSprop. Left: un-reshaped raw data X, trained with the same configurations as the left, but on the first ConvNet which converges quickly to the loss of ≈ 0.6932.

Hence, we conclude our observations from Figure 9 as follows:

Observations - 1

1. Our ConvNet is extreme. It is extremely wide, given that one of the input's dimension is enormous, which we identify this model as a high resolution and large width network [9].
2. Neither the change in the ConvNet structure or the shape of the input affects the early convergence of the model. We deduce this is likely to because of the high-variance raw data.
3. The higher training loss (plot on the left in Figure 9) is a possible consequence of implementing dropout layers since dropouts are not performed during validation. We implements dropout in order to reduce the variance of the model.

Hence, the result in Figure 9 is expected, given that the shallowness of our network results in its inability to capture higher level features and fined-grained details [9] that enables the network to distinguish between noise and actual signals within the raw gravitation waves.

6.2.2 Optimization

In this part, we choose to train our model on those reshaped samples of size (96, 128) as we continue to use the ConvNet mentioned in the above section, given that there is no evidence to prove the superiority of persevering the original data shape (Figure 9).

Stochastic (Batch) Gradient Descent Throughout the report, we conduct batch gradient descent with batch size of 8 (on local machine) and 16 (on Kaggle), i.e. gradient is update once per batch. We perform this by splitting dataset into batches through the pytorch DatasetLoader interface. We choose SGD because it is a all-time choice of optimizer in Machine Learning community and would provide insights to further exploration on other optimizer.

RMSprop We compare the performance of SGD with RMSprop implemented in pytorch. RMSprop, root mean square propagation, is another optimization method and a mini-batch generalization of Rprop [4]. RMSprop calculates moving average of the squared gradients for each weight, and then divide the gradient by square root of the mean square [6]. Empirically, RMSprop optimizer restricts the vertical oscillation which enables the algorithm to take larger steps to attain global minimum in the horizontal direction. Also, as it lies in the realm of adaptive learning rate, it has significant impact on avoiding vanishing gradient problem.

The weight updating rule given by RMSprop is:

$$E\left[g^2\right]_t = \beta E\left[g^2\right]_{t-1} + (1-\beta)\left(\frac{\delta L}{\delta w}\right)^2$$

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{E\left[g^2\right]_t}}\frac{\delta L}{\delta w}$$

where η is the learning rate and β is a moving-average hyperparameter, defaulted to 0.9 in our ConvNet.

In our case, the method is suitable as we observe early convergence of the losses from Figure 9. Our previous approach to counteract this convergence by reducing the learning rate (from 0.0025 to 0.0001) does not reveal any significant improvements, hence we like to further improve the magnitude by applying such adaptive learning rate method of RMSprop.

As we can see from Figure 6.2.2, the loss of ConvNet with stochastic gradient descent is still restricted in the range of $[0.69, 0.7]$, similar to the results from Figure 9. The plot on the right, which performs the RMSprop, also coincide with our observations from Section 5.2.1. The training loss of X_banded_whitened_tukey decrease as we expected, whereas it's validation loss increase drastically as a result of high variance intrinsic to the data and an evident sign of over-fitting.

Observations - 2

1. Stochastic Gradient Descent is unable to escape the local stationary point it encounters in our data, no matter if the data is preprocessed or not.
2. RMSprop allows the learning rate we set at the beginning ($\eta = 0.0001$) to adapt overtime by the average of the gradients squared, which benefits training as it speeds up the convergence to minimum.
3. The irreducible high-variance within the data and the result of over-fitting suggests we should continue to approach the deeper ConvNet but also increase the magnitude of regularization for the over-fitting issue.

6.3 ResNet and Efficient Net

Since there is no significant difference between the performance of reshaped or un-reshaped data, we mostly utilize the reshaped version in order to conduct experiments on deeper ConvNets without padding the sample with extra zeros, which potentially requires the network to learn the irrelevance of the padding zeros in the samples.

6.3.1 Why ResNet?

ResNet is an evolutionary architecture in the realm of computer vision for the last few years due to its simplicity within skip-connection and the elegance of identity mapping. We are convinced that ResNet could resolve the degradation

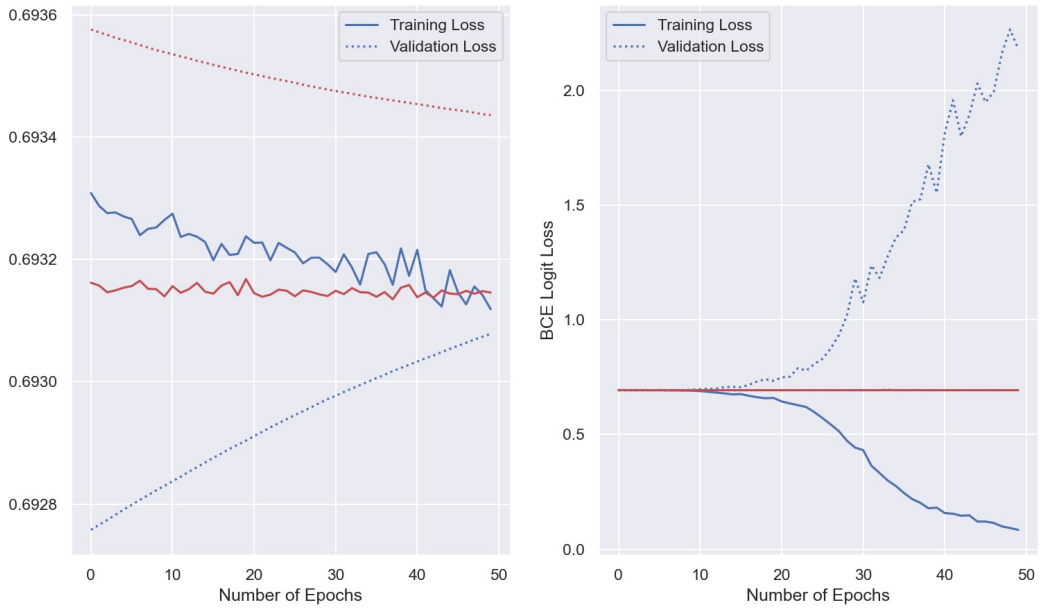


Figure 10: Both results are from the same model trained using the same configuration with the same reshaped samples. Left: SGD, Right: RMSprop. Red: X_data, Blue: X_banded_whitened_tukey. Dotted lines represent validation loss and solid lines represent training loss.

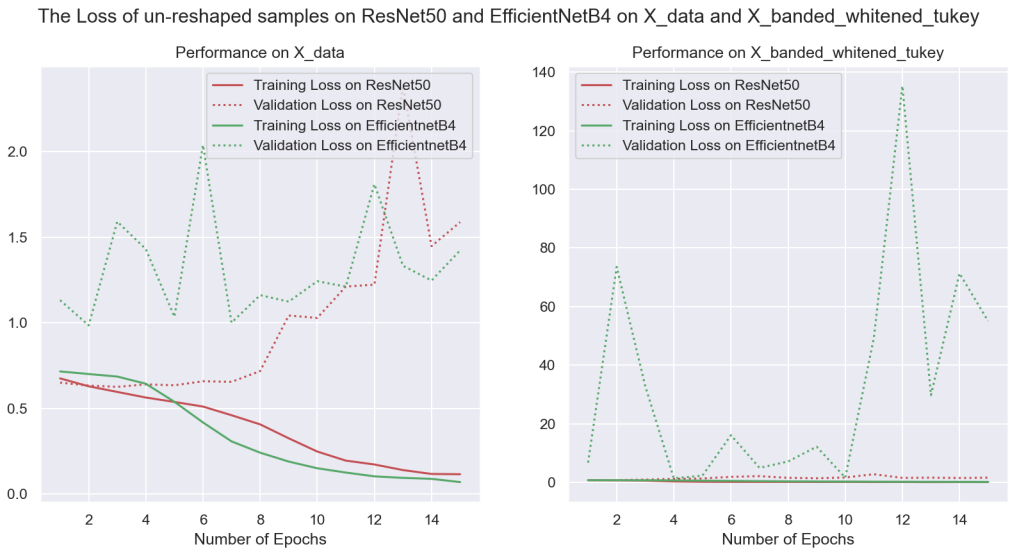


Figure 11: The comparison between performance of ResNet50 and EfficientNetB4 trained on unreshaped datasets. ResNet50: red; EfficientNetB4: green. The dotted lines represent the validation losses whereas the solid lines represent the training losses. Left: X_data, Right: X_banded_whitened_tukey.

problem existing in our model - model with more layers perform worse on the dataset then the singleton ConvNet, i.e.

the drastic increase of validation loss, represent as over-fitting of the model. We believe the learning gravitational waves relies on capturing the high level features, which can only be enriched by increasing the number of stacked layers [3].

We chose ResNet50 as the model because it is the most popular within the ResNet family among the machine learning community and our previous test on ResNet18 does not perform any better than our ConvNet 2, resulting in a loss around [0.69, 0.7] for both training and validation. The implementation we use for ResNet50 is from timm and it is not pretrained on any dataset.

6.3.2 Why EfficientNet?

While ResNet focuses on scaling the ConvNet according to depth, EfficientNet propose a compound scaling method that scale width, depth and resolution of the network which achieves a better accuracy than scaling only one factor of the network [9].

We decide to conduct experiment on EfficientNet for the following reasons:

1. We are interested in why EfficientNet is such a popular family of ConvNets in Kaggle community.
2. Considering the extreme dimensions of our input (without reshaping), we believe the compound method of scaling would enhance the results of ResNet or even outperform ResNet as it is more adapted to the original input.

We choose to use EfficientNetB4 because this is most broadly used implementation among other EfficientNets. The coefficients of B4 are 1.4 and 1.8 for width and depth respectively. The default resolution is 380. The dropout rate is 0.4.

6.3.3 Observations - 3

On the **transformed dataset** (X_banded_whitened_tukey), ResNet50 performs better overall than any model discussed before, since it training loss converges to 0.09 and validation loss remains at 1.15, which is considerably much lower than EfficientNetB4 (Figure 11). Which EfficientNetB4 demonstrates the explosion of validation loss to approximately 140.

We conclude the following possible reasons of such situation, mostly due to the scaling factors and the default resolution implemented in the version of B4:

1. Our sample size is (3, 4096) when unshaped and (96, 128) which totally contains 12288 pixels. Whereas the default resolution of EfficientNetB4 is size (380, 380), which is substantially larger.
2. The inconsistency within image size results in large amount of zero-padding, which cause the ConvNet to learn the irrelevance of 0s.
3. The transformed data does not contains any high level features that requires a neural network deep as the depth after being scaled by the EfficientNetB4.
4. ResNet50's skip-connection possibly reduce the variance slightly since it is designed to avoid the degrading performance problem in deeper layers, which enables ResNet to preserve more low-level features as well as those high-levels.
5. We simply don't have enough amount of data to train a model with less variance.

We came to the following observations, which remained unsolved:

1. Given the dropout rate of EfficientNetB4 is 0.4, which is considerably a high magnitude of regularization, the validation loss should be less than what presented.
2. The irreducible and extensive gap between validation and training loss phenomenon might not be caused by overfitting.
3. Transformed data could be potentially over-regularized by the models.

On the **raw dataset**, we observe that both ResNet50 and EfficientNetB4 present similar behavior over the 15 epochs. Both training loss converges in the same speed but different step size, and both have validation loss increasing in a steadier pace than on the transformed dataset. The behavior is similar to what we've observed in Figure 6.2.2, where our ConvNet's performance using RMSprop produce a similar results on the blue lines.

Hence, from experimenting with the EfficientNet and ResNet, the popular deep learning models, we conclude that there is an intrinsic variance within the data after transformation, which leads all the 3 models, our ConvNet, ResNet50, and the EfficientNetB4 to behave in a similar manner with an exponentially increasing validation loss.

7 Conclusion

Data processing takes a significant role for this dataset. In order to achieve a trainable and worth-learning dataset, the method of data processing should be further studied and tested. MFCC is the most effective processing technique due to its ability to extract features and reduce the feature space. Logistic Regression's stability and applicability makes it the best performing baseline.

From our observations and experiments, the raw data easily converges to the loss of 0.69 no matter the setup of other hyperparameters and the model itself. When the data is processed, it is more likely to experience over-fitting and lead to huge performance gap between training and validation loss (in ConvNet), and training and test accuracy (in baselines).

Our work empirically proves the significance of preprocessing for time-series gravitational waves data, and the flaws of convolutional neural networks on data with large amount of noise.

8 Future Work

Data Processing From what we have observed, the use of whitening and bandpass filtering must be further refined to better isolate gravitational signals without important information loss. We can also further research MFCC, to improve its effectiveness further. Some selective standardization within channels may help tailor the performance a little more.

Convolutional Neural Networks From what we have observed, there are still more potential improvements in the structure of ConvNet to achieve a better performance than what we had acquired. Given the over-fitting phenomenon on ResNet50 and EfficientNetB4, we could consider larger regularization magnitude on the model, such as adding more dropout layers or applying batch normalization in a deeper neural network (deeper than what we present in the report), since ResNet50 have a overall better performance. It would also be more ideal if we could train on a larger dataset to ensure there is no any inherited bias and variance from small-sample learning.

References

[1] Adrian Cho. *Gravitational waves, Einstein's ripples in spacetime, spotted for first time*. URL: https://www.sciencemag.org/news/2016/02/gravitational-waves-einstein-s-ripples-spacetime-spotted-first-time.

[2] European Gravitational Observatory - EGO. *G2Net Gravitational Wave Detection*. 2021. URL: https://www.kaggle.com/c/g2net-gravitational-wave-detection.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: (June 2016), pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/ (visited on 11/24/2020).

[4] Geoff Hinton. *RMSprop*. 2018. URL: https://web.archive.org/web/20210414234536/http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[5] Stefan Hrouda-Rasmussen. *(Gaussian) Naive Bayes - An introduction to the (Gaussian) Naive Bayes model with theory and Python implementation*. URL: https://towardsdatascience.com/gaussian-naive-bayes-4d2895d139a.

Table 1: Baseline **Test** Accuracy on Data Processed Using Different Techniques

Preprocessed Data Variant	Linear Regression	Decision Tree	Gaussian Naive Bayes
X_data	63.4444	53.2889	58.7556
X_whitened_raw	55.1333	50.9556	51.0000
X_whitened_black	56.5778	51.4000	55.4667
X_whitened_tukey	61.4667	52.0667	56.9333
X_banded_whitened_raw	55.6444	50.7111	49.6000
X_banded_whitened_black	58.4444	51.5111	54.1111
X_banded_whitened_tukey	62.2889	53.0222	55.2444

Table 2: Baseline **Train** Accuracy on Data Processed Using Different Techniques

Preprocessed Data Variant	Linear Regression	Decision Tree	Gaussian Naive Bayes
X_data	67.0952	96.0000	61.4667
X_whitened_raw	60.2381	100.0000	53.0095
X_whitened_black	60.7238	100.0000	57.8286
X_whitened_tukey	63.9905	100.0000	58.4476
X_banded_whitened_raw	58.4762	100.0000	51.2381
X_banded_whitened_black	60.5143	95.4857	56.1905
X_banded_whitened_tukey	63.0857	95.4857	55.6571

[6] Jason Huang. *RMSprop - Optimization*. 2020. URL: https://optimization.cbe.cornell.edu/index.php?title=RMSProp.

[7] Niels Landwehr, Mark Hall, and Eibe Frank. "Logistic Model Trees". In: *Machine Learning* 59.1 (May 2005). ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-005-0466-3. URL: http://link.springer.com/10.1007/s10994-005-0466-3 (visited on 07/31/2021).

[8] Claudia Perlich et al. "Tree Induction vs. Logistic Regression: A Learning-Curve Analysis". In: ().

[9] Mingxing Tan and Quoc V Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: (), p. 11.

[10] Y. Le Cun and B. Boser. "Handwritten Digit Recognition with a Back-Propagation Network". In: (1988). URL: https://proceedings.neurips.cc/paper/1987/file/a684eceee76fc522773286a895bc8436-Paper.pdf (visited on 07/07/2021).

9 Appendix

9.1 Data Visualization

9.2 Additional Normalization & Standardization Results

9.3 Additional Bandpass Upper Bound Optimization

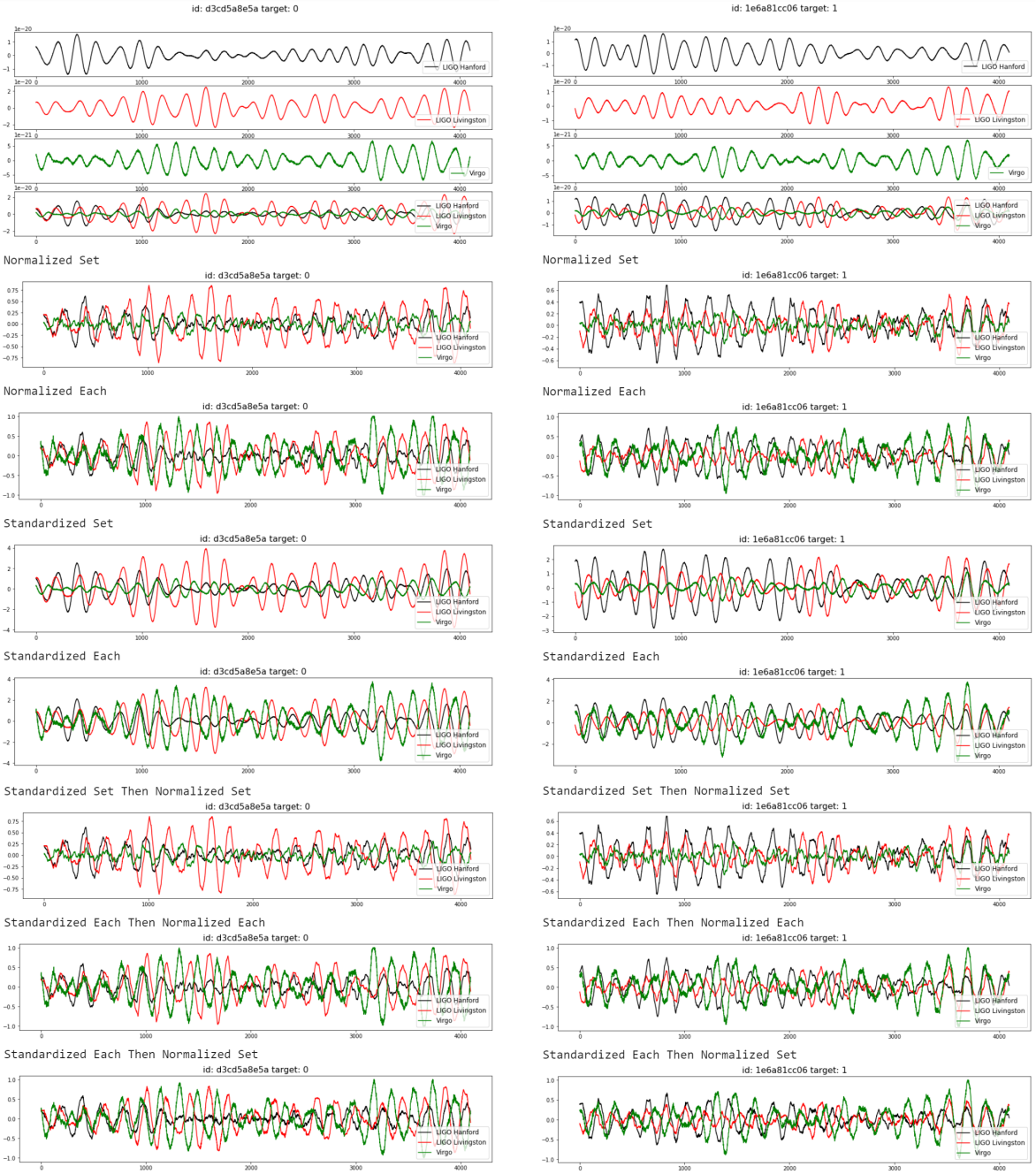
9.4 Baselines

9.5 Convolutional Neural Network

Structure of our ConvNet:

1. Convolutional layer, 3 filters with kernel size 3, stride 1 and no padding.
2. Maxpooling layer with kernel size 2
3. Convolutional layer, 6 filters with kernel size 3, stride 1 and no padding.
4. Maxpooling layer with kernel size 2

Figure 12: Visualization of Dataset



5. Dropout layer with rate 0.2
6. Convolutional layer, 12 filters with kernel size 3, stride 1 and same padding.
7. Maxpooling layer with kernel size 2
8. Dropot layer with rate 0.2
9. Convolutional layer, 24 filters with kernel size 3, stride 1 and same padding.
10. Maxpooling layer with kernel size 2

Figure 13: Pairplot of Relationship between Detectors

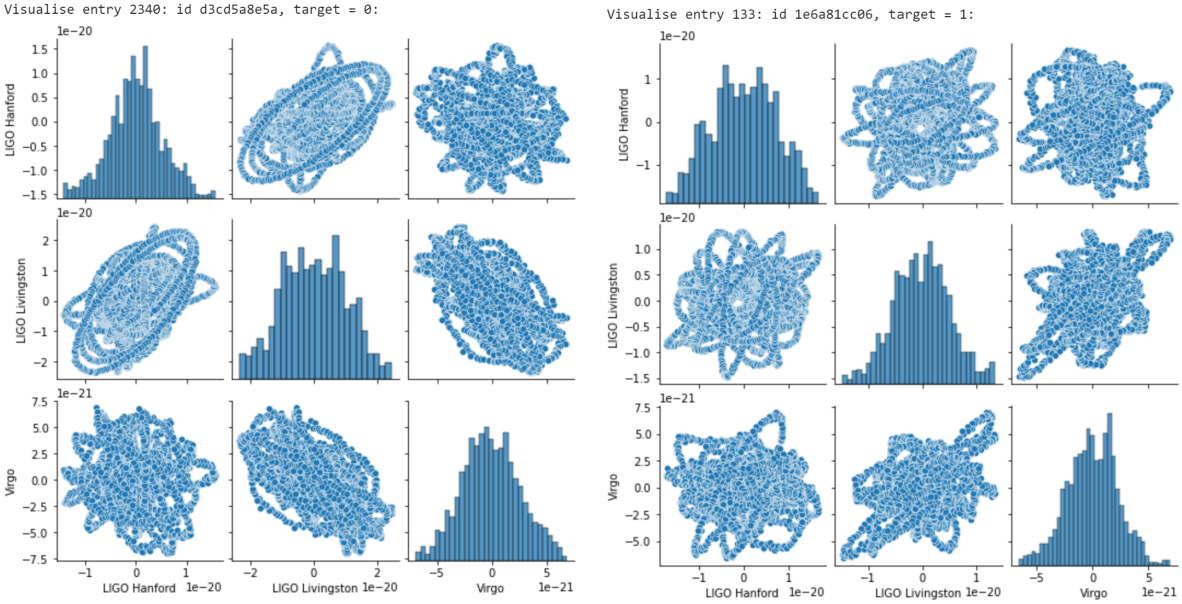


Figure 14: Left: MFCC applied before processing. Right: Tukey Whitened Bandpassed Dataset

Figure 15: Upper Bound Optimization Left: 150Hz to 300Hz Right: 230Hz to 288Hz