DeepClean Neural Network for Gravitational Wave Noise Reduction

By John Choi, Matthew Vigil, Laura Jian, Preethi Karpoor

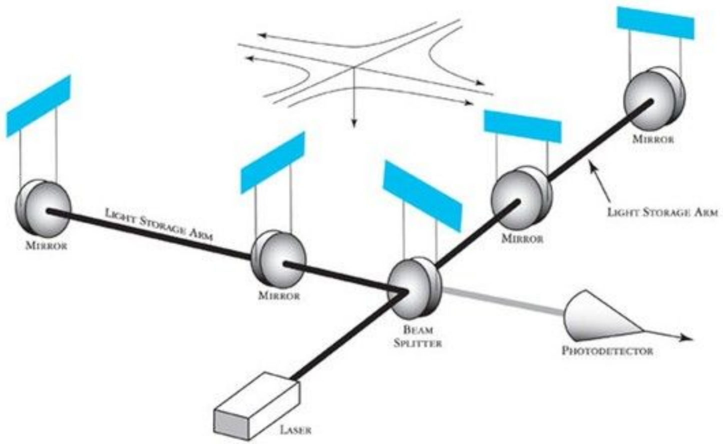
Group 5

Background and Motivation

- Recent Gravitational Wave (GW) observations have led to a spur in noise reduction pipelines.
- The working principles of light interferometers allows numerous channels of introduced noise.
- Using Machine Learning (ML) architectures such as autoencoders and Convolutional neural networks (CNN’s), detection systems can learn to filter out noise in the data.
- We are motivated to lower the sensitivity threshold for anomalous event detections in order to expand our knowledge on GWs.

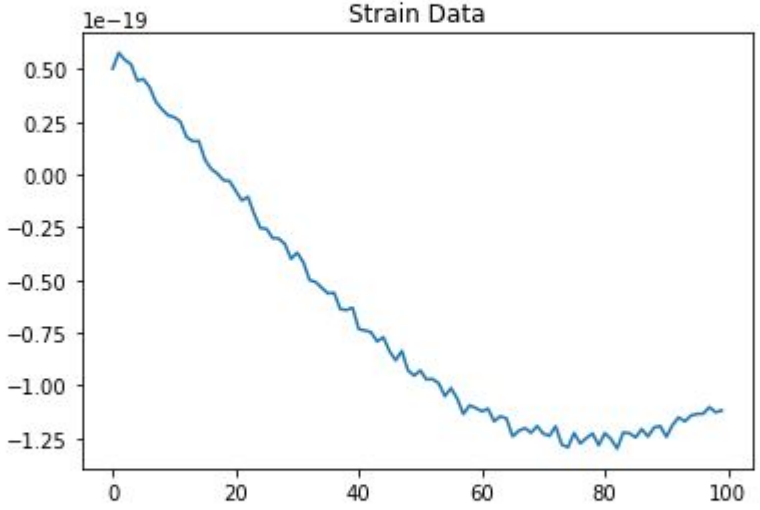
LIGO working Principle

- LIGO, a light interferometer system uses light interference to detect shifts or waves in spacetime.
- The arms of the interferometer are 2.5 miles long and have an accuracy of 1/10,000th the width of a proton.
- Due to the sensitivity, LIGO picks up numerous channels of noise.

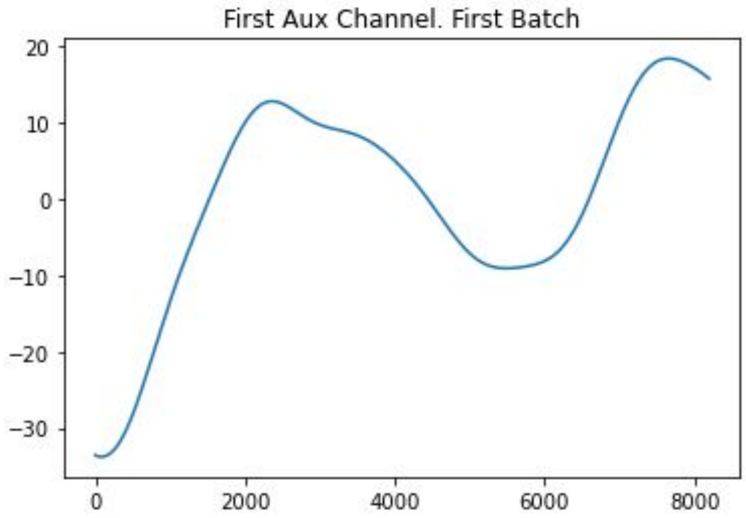


Types of Data

GW Strain Data:



Auxiliary Witness Data (Noise, 21 Channels):



Working Concept for Noise Reduction/ Paper Introduction

- Our work based around replicating Ormiston et. al.'s work: **Noise reduction in gravitational-wave data via deep learning**
- We are attempting to recreate the DeepClean Network for Noise reduction
- Our process is as follows:
  - Data Pre-Processing
  - Model Architecture
  - Model Training
  - Model Testing/Inference
  - Model Inference Post-Processing
  - Noise Reduction Pipeline
  - Result Analysis

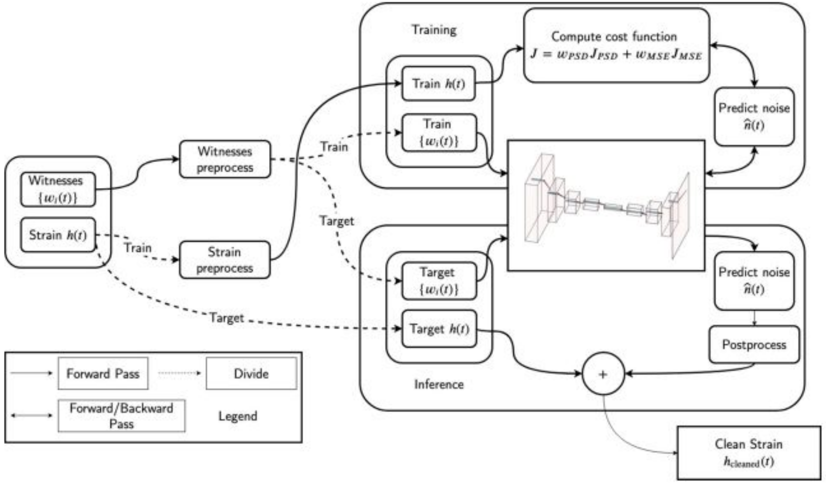


FIG. 2. Workflow diagram of the noise subtraction pipeline.

Data Pre-Processing:

Batching
8th Order Butterworth Filter
Z-score / Standardization
Windowing

Presented by: John Choi

Batching:

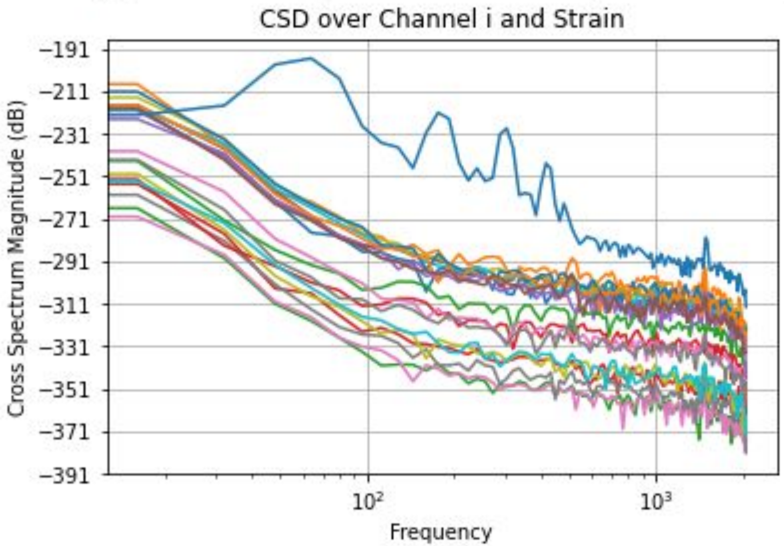
- We performed batching on both the GW strain and witness channel data.
- Breaking up the original datasets into 1000 smaller sets of length 8192 data points, or 2s of GW data at a 4096/s sampling rate.
- Attempted to make the batching as large as possible to maximize training/validation data size.

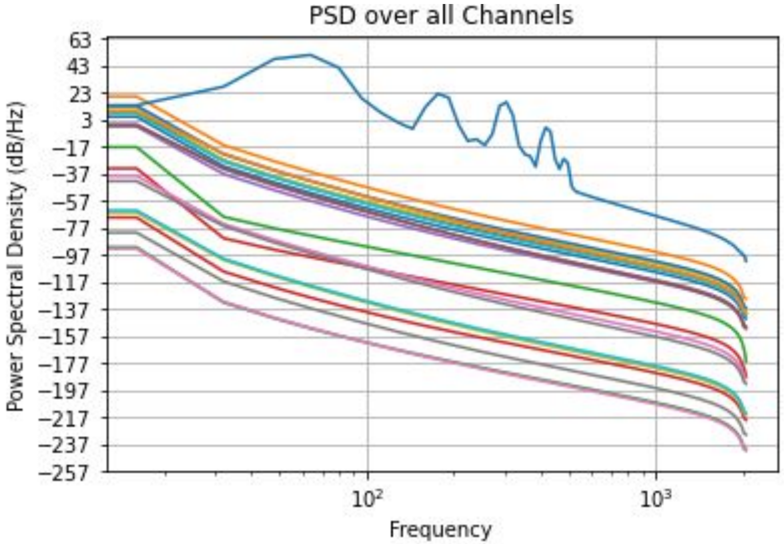
8th Order Butterworth Filter

- Designed to be a band pass filter
- Pass band was decided from the cross-spectral density (CSD) and power-spectral density (PSD) analysis of the Strain and Witness Channels
- The pass band used in this project was 0Hz to 0.3 Hz
- This removes any unwanted power contributions outside the CSD interactions
- The 8th Order characteristic gives this filter a roll off slope of -160 dB per Decade

CSD (Left) and PSD (Right). Pass Band identification

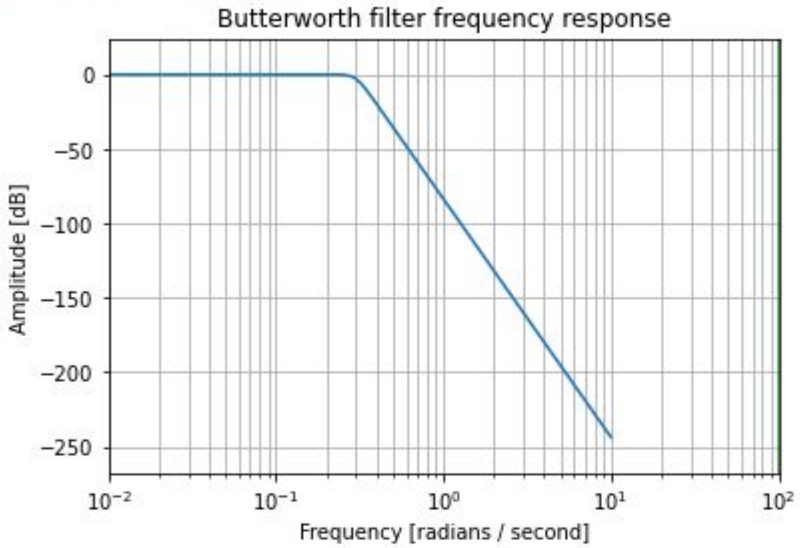
Finding pass-band frequencies via Cross Spectral Density Analysis



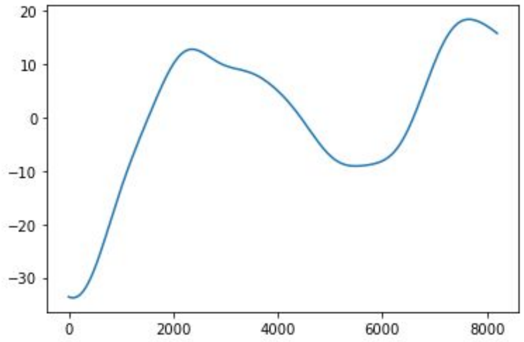


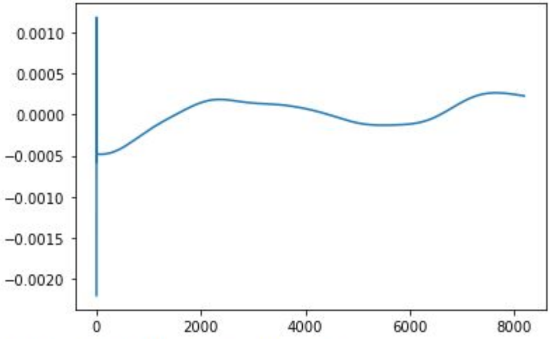
Butterworth Filter and Application to Data

Building Butterworth Filter



Applying Butterworth to data
(1000, 8192, 21)



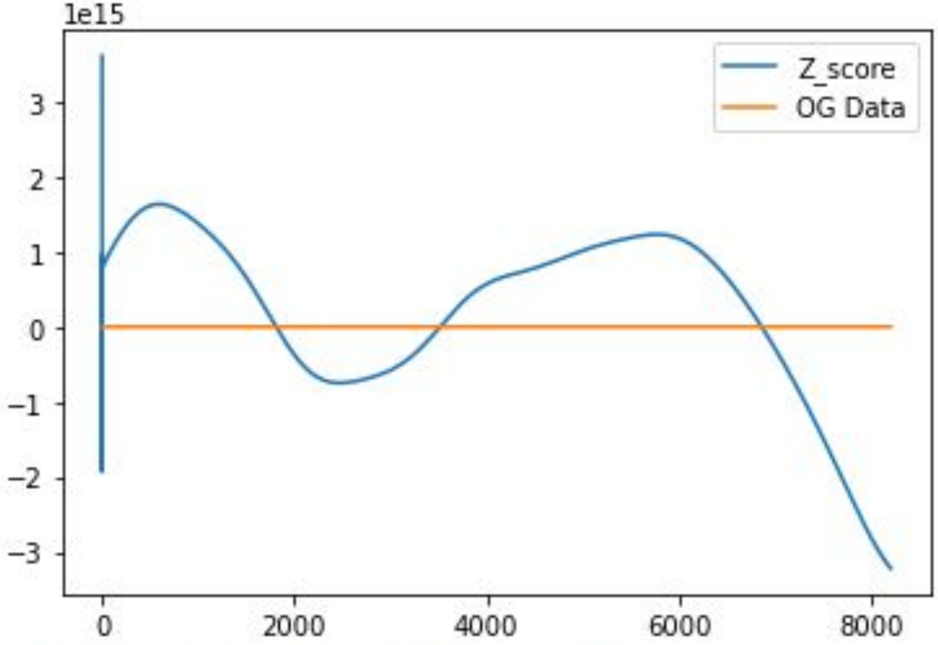


Butterworth filter applied to data

Z-score / Normalization

- Z-score was applied in order to mitigate any numerical instabilities in the custom loss function.
- Applied to the Strain and the Witness noise data.
- When applied to Witness Noise data, we can remove any bias from a single contributing channel

Z-Score of the Filtered Data



Windowing (WIP)

- To increase model efficiency, Ormiston et. al. windowed the data with an overlap of 7.75 seconds
  - 96.875% of the total window size
- Testing data windowed with an overlap of 4 seconds
  - 50% of the total window size
- This is still being implemented as we have been able to train and test the model without it
  - Windowing will be a feature we will test in order to see its changes on the overall performance
  - We expect this to slow training time.

Model Architecture:

DeepClean Structure

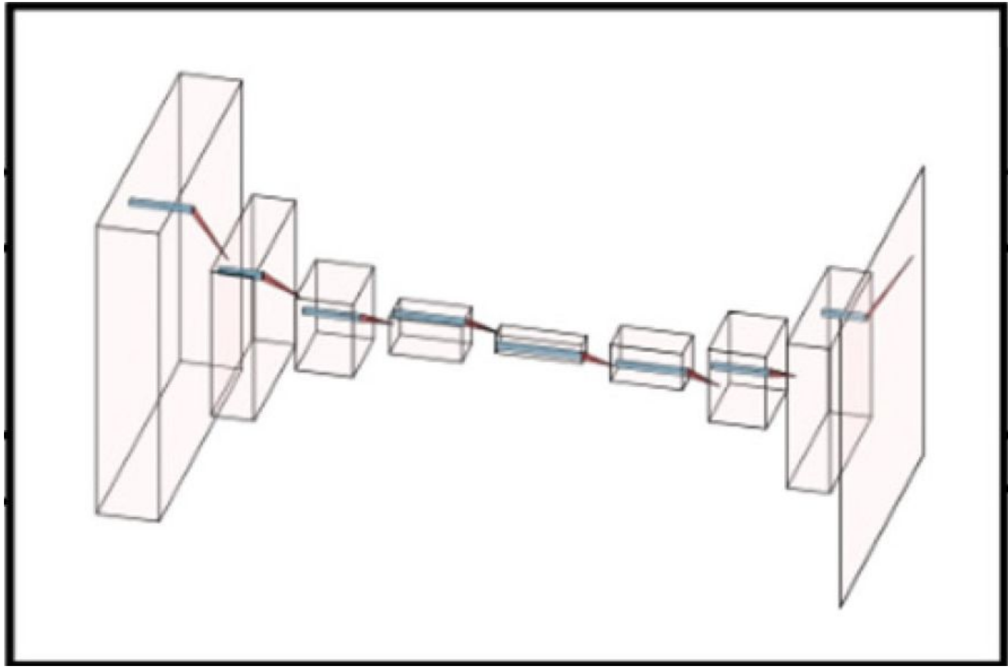
Layer Behavior/Parameters

Loss Function

Presented by: Matthew Vigil

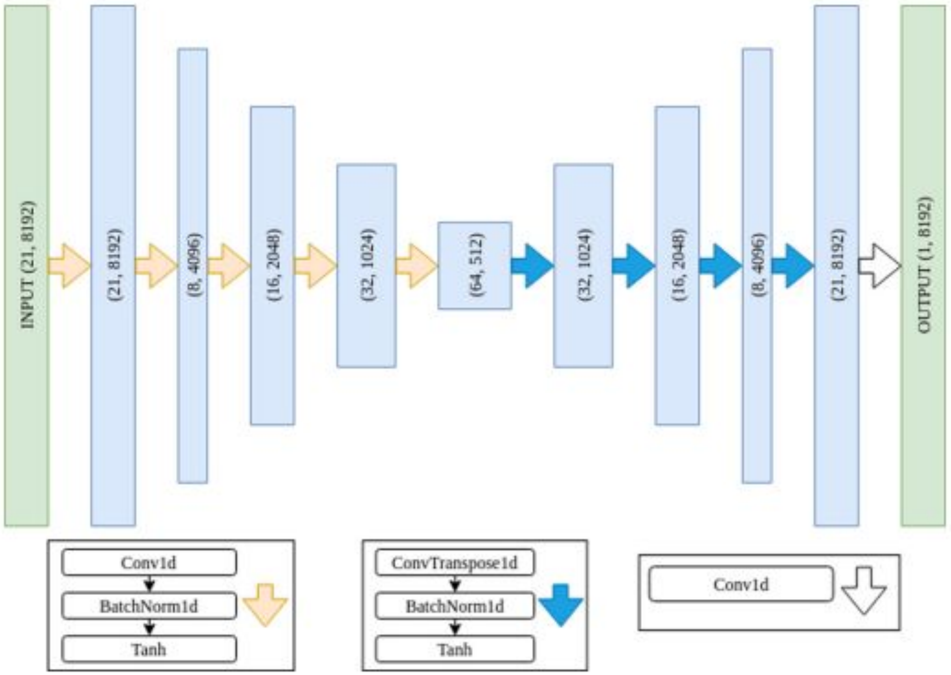
Model Architecture: DeepClean

- Once the data is preprocessed, it is input into DeepClean, a 1D Convolutional Neural Network (CNN).
- DeepClean accepts a 21-channel set of witness data and predicts the noise present within the strain.
- The input is passed through multiple 1DConv layers for downsampling, then an equal amount of 1DConvTranspose layers for upsampling, building a set of parameters.
- Validation takes in witness channel data and produces the predicted noise using the mapped parameters.
- This is postprocessed and subtracted from the original strain, 'cleaning' the original signal.



Layers, Inputs, Outputs

- The input layer accepts all 21 channels, with each subsequent layer “learning” features from layer to layer.
- CNN architecture allows for retention of long time-series and long-term features.
- Each layer uses a stride=2 to half time-series length, and double number of channels and vice versa for 1DConvTranspose, ensuring same size before output.
- Each layer except output is followed by batch normalization (Batch1d) to improve training efficiency.



Loss Function: Custom vs MSE

- DeepClean utilizes a custom loss function to calculate mapped parameters.
- The ASD and MSE components (Eqn. 8) are summed, and weighted with the term w to focus on their spectral line or broadband data, respectively.
- This custom loss would have to be written in TensorFlow, so preliminary training was done using only MSE.

$$\vec{\theta} = \mathrm{argmin}_{\vec{\theta}'} J[h(t), \mathcal{F}(w_i(t); \vec{\theta}')]. \tag{3}$$

$$J_{\mathrm{asd}} = \frac{1}{f_2 - f_1} \int_{f_1}^{f_2} W(f) \sqrt{S[r, r](f)} df \tag{4}$$

$$r(t) = h(t) - \mathcal{F}(w_i(t); \vec{\theta}), \tag{5}$$

$$J_{\mathrm{mse}} = \frac{1}{N} \sum_{i=0}^{N-1} r[i]^2, \tag{7}$$

$$J = w J_{\mathrm{asd}} + (1 - w) J_{\mathrm{mse}}, \tag{8}$$

Model Parameters

- Use a nonlinear tanh activation function.
- Uses ADAM gradient descent.
- Padding is set to 0 to preserve time-series length.
- Kernel_size = 7 for all layers.
- Learning rate set to 1x10^-3.
- From literature, training typically takes 5-10 epochs.

```
47/47 [==============================] - 2s 47ms/step - loss: 4.9569e-04 - accuracy: 0.0000e+00 - val_loss: 0.0012 - val_
accuracy: 0.0000e+00 - lr: 5.9324e-04
Epoch 6/10
47/47 [==============================] - 2s 47ms/step - loss: 4.2972e-04 - accuracy: 0.0000e+00 - val_loss: 8.0193e-04 -
val_accuracy: 0.0000e+00 - lr: 5.3417e-04
Epoch 7/10
47/47 [==============================] - 2s 43ms/step - loss: 3.8105e-04 - accuracy: 0.0000e+00 - val_loss: 5.4396e-04 -
val_accuracy: 0.0000e+00 - lr: 4.8099e-04
Epoch 8/10
47/47 [==============================] - 3s 54ms/step - loss: 3.4360e-04 - accuracy: 0.0000e+00 - val_loss: 4.0816e-04 -
val_accuracy: 0.0000e+00 - lr: 4.3310e-04
Epoch 9/10
47/47 [==============================] - 2s 43ms/step - loss: 3.1382e-04 - accuracy: 0.0000e+00 - val_loss: 3.3433e-04 -
val_accuracy: 0.0000e+00 - lr: 3.8998e-04
Epoch 10/10
47/47 [==============================] - 2s 45ms/step - loss: 2.8961e-04 - accuracy: 0.0000e+00 - val_loss: 2.9512e-04 -
val_accuracy: 0.0000e+00 - lr: 3.5115e-04
```

Model Training/Testing:

Data Splitting

Loss Performance

Presented by: Matthew Vigil

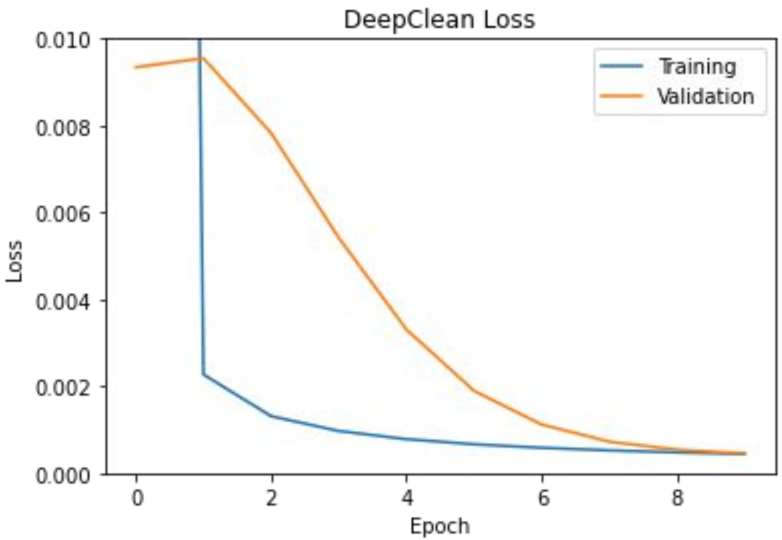
Training Method

- Time-series data splits should set validation as the “latest” data.
- Strain and witness channel data is split 4:1 training to test, with the latest rows used as testing data.
- Splitting and pre-processing will be integrated to validate results.

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

#Check input shape, given dataset is sampled at 4096Hz
model = keras.models.Sequential(name="deepclean")
#Convolution Layers
model.add(layers.Conv1D(filters=21, kernel_size=7, strides=1, padding="same", activation="tanh",\
                        input_shape=(X_train_batch.shape[1], X_train_batch.shape[2])))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(filters=8, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(filters=16, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(filters=32, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(filters=64, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())

#Deconvolution Layers
model.add(layers.Conv1DTranspose(filters=32, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1DTranspose(filters=16, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1DTranspose(filters=8, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1DTranspose(filters=21, kernel_size=7, strides=2, padding="same", activation="tanh"))
model.add(layers.BatchNormalization())
model.add(layers.Conv1D(filters=1, kernel_size=7, padding="same", name = "output"))

model.summary()
```

Training Results

- Training loss fell below 5x10^-4 in multiple runs, with convergence by ~8 epochs.
- Loss levels off by ~6 epochs, later than described in Ormiston et. al.
- The model took ~35s to build, with a training time of ~16s for 1st epoch and a mean 2s for each subsequent epoch, training on data of shape (1638, 8192, 21).



Model Inference Post-Processing:

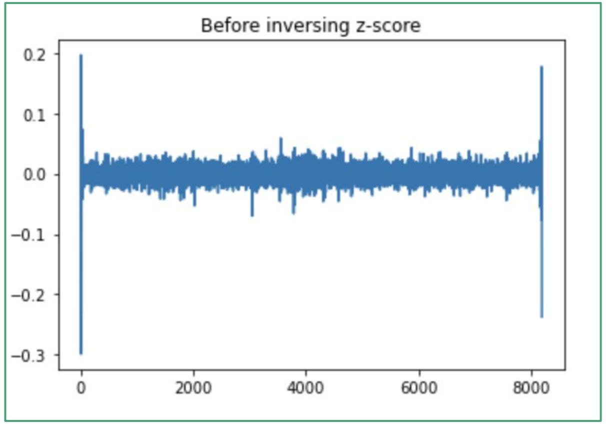
Inverse Z-score

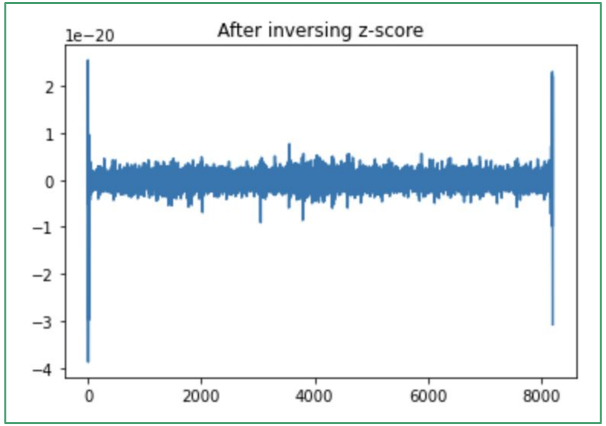
8th Order Butterworth Filter

Presented by: Laura Jian

Inverse Z-score

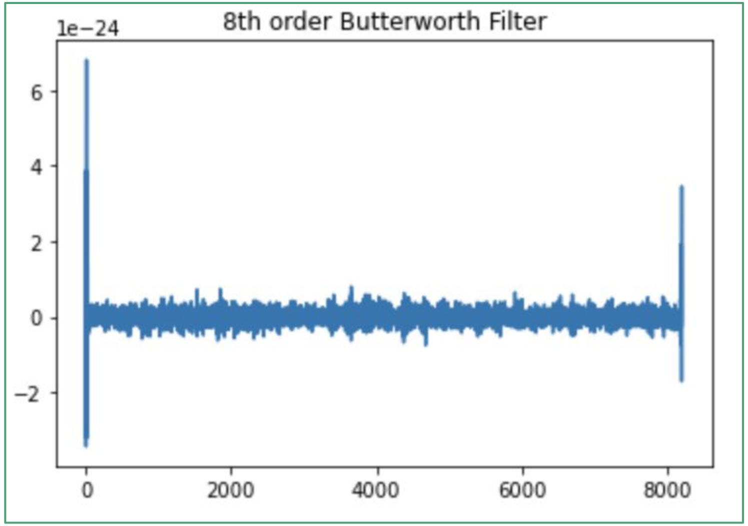
- After the model has made its predictions from the Z-scored data we applied an inverse Z-score to the predictions
- We multiply by the Std Dev and add the Mean back to all the data points in the predicted noise
- This returns the predicted noise datas original units and range which is needed for the subtraction pipeline





8th Order Butterworth Filter

- Since the original Witness Noise Data was filtered, we need to apply the same filter to the output predictions
- Without filtering, we introduce instabilities and power contributions outside our desired pass band.
- We apply the same filter pass band of 0Hz to 0.3Hz to the model predicted noise

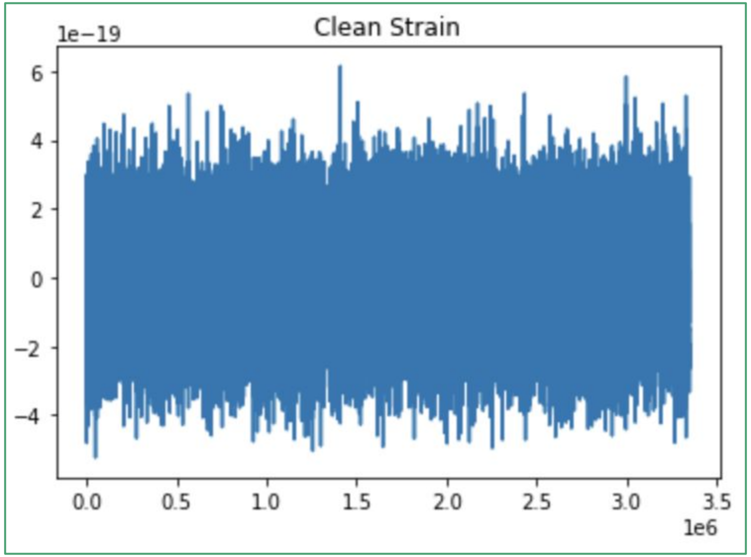


Noise Reduction:

Presented by: Laura Jian

Removing Noise & SNR

- To get our clean strain, we subtract our full bandwidth strain from our noise
- Signal-to-Noise Ratio (SNR), original-to-clean signal difference, and amplitude spectral density (ASD) rato were used as metrics.



$$SNR = \frac{P_{signal}}{P_{noise}}$$

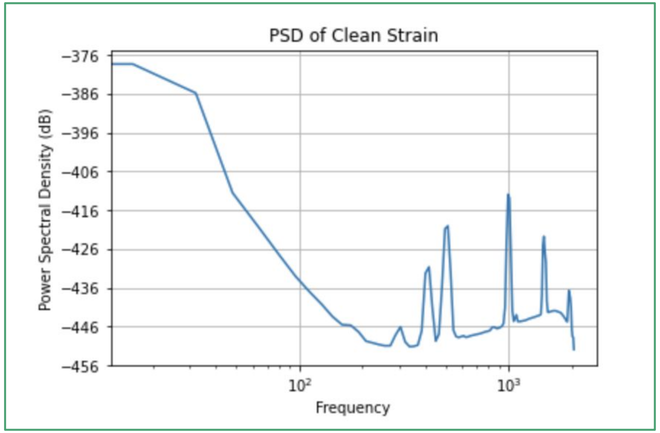
$$P(x) = 1/N \cdot \sum_{n=0}^{N-1} x(n)^2$$

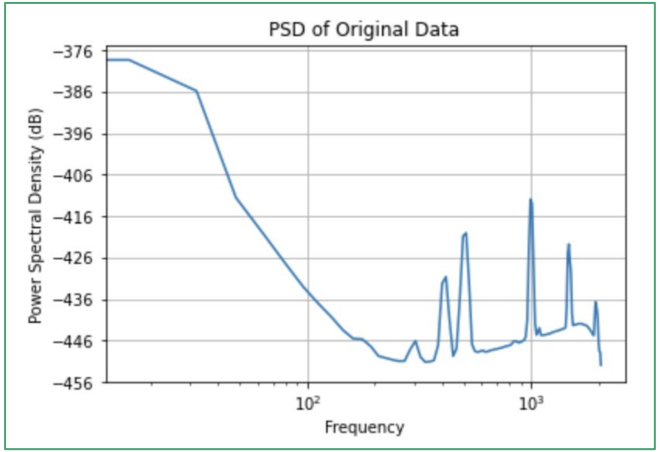
```
SNR_dB = 146.37
```

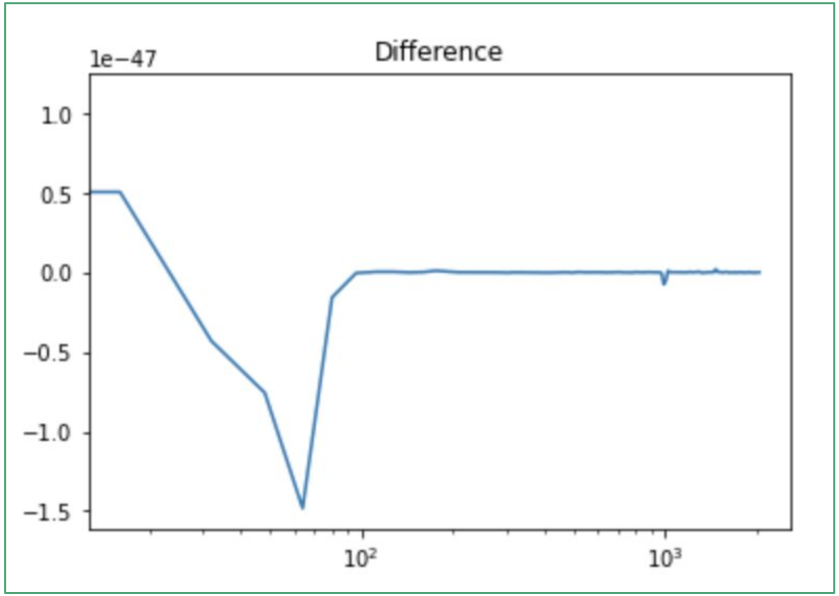
Result Analysis:

Presented by: Preethi Karpoor

Power Spectral Density

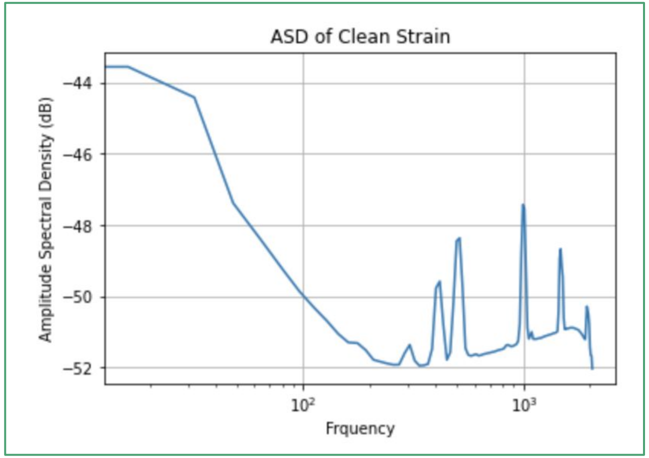


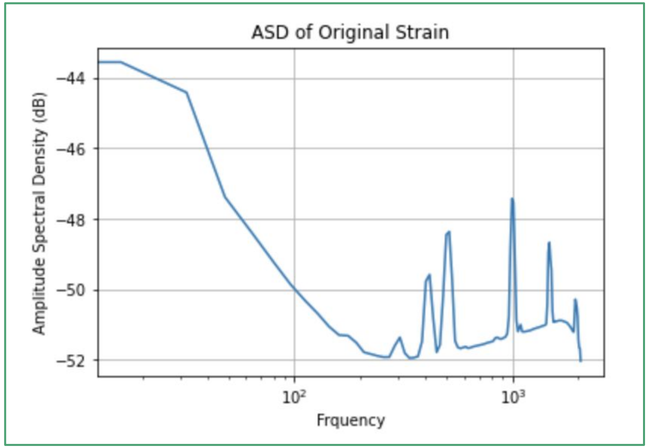


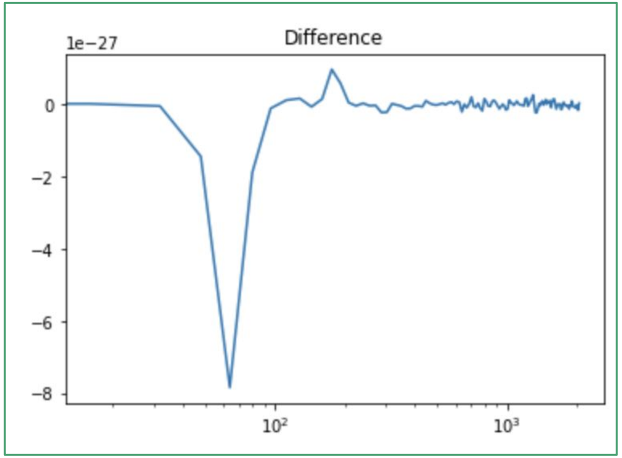


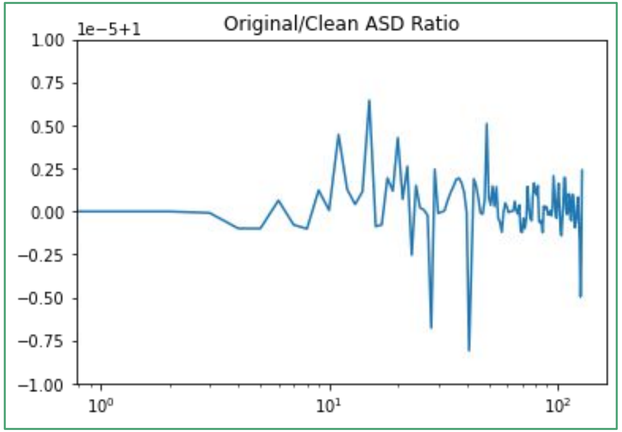
The final aim of Deep Clean Algorithm is to effectively subtract the external noise to the best extent possible and produce a clean signal. We see through the plots above that the difference between the Power spectral density(PSD) of output strain and input strain is indeed very small, thereby leading to a large signal-to-noise

Amplitude Spectral Density









We can see the cleaning performance again here through the Amplitude Spectral Density (ASD) ratios as well. ASD is essentially square root of PSD.

Summary and Further Work

Presented by: Preethi Karpoor

Results

- Calculated SNRs were ~65-147 dB.
- Model training and validation near expectations from Ormiston et. al.
- Difference in ASD between clean and original strain was in the magnitude of 10^-27, with a original to clean ASD ratio in the magnitude of 1+1E10^-5.
- Preliminary DeepClean performance subtracted a maximum difference of ~8x10^-27 strain units at approximately 65 Hz.

Conclusion

- Future work would include:
    - Integrating pipeline components.
    - Implementation of the custom loss to add the ASD component.
    - Modifying window length adding/removing DeepClean layers to study relation of training and prediction time vs. dataset size.
    - Modifying learning rate to test training performance.
    - Implementing minibatch feeding.

References

1. R. Ormiston, T. Nguyen, M. Coughlin, R. X. Adhikari, and E. Katsavounidis, “Noise reduction in gravitational-wave data via deep learning,” Phys. Rev. Res., vol. 2, p. 033066, Jul 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.2.03306613

Acknowledgements

Our sincere gratitude to Alec Gunny and Prof. Duarte for their valuable inputs and guidance!

GitHub Repository:

https://github.com/telmar3/PHYS139_FinalProject