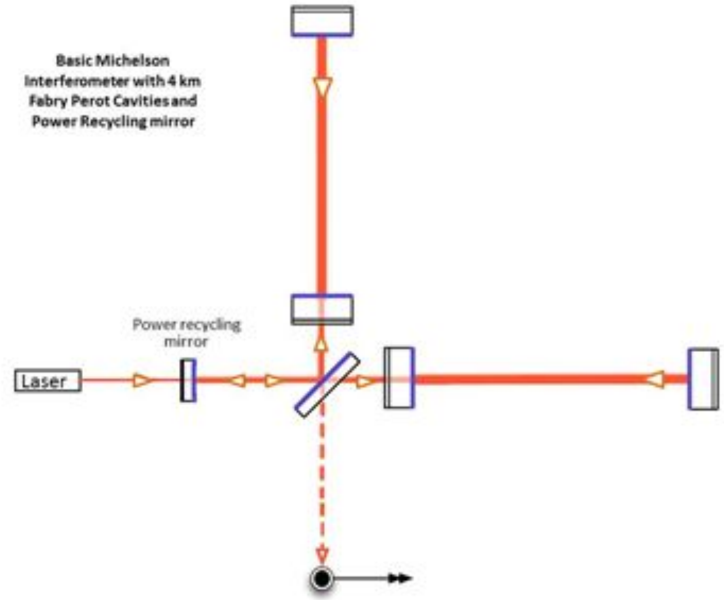
Gravitational Wave Detection

CMPE 257 Group 3 Project

Tom Casaletto
Anthony Fisher
Phil Shirts
Joel Wiser

Outline

- Problem Statement
- Pre-processing overview
- Continuous Wavelet Transform
- Q-Transforms
- Filtering/Short Time FFT
- Conclusions

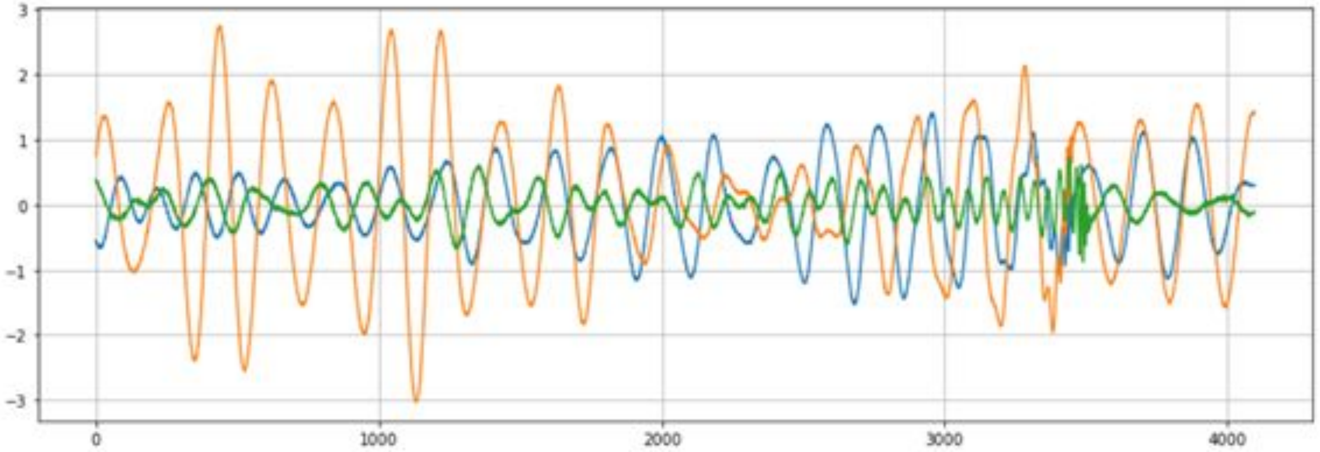


Detecting LIGO Gravitational Signals: (a kaggle.com competition)

Team: Tom Casaletto, Joel Wiser, Anthony Fisher, Phil Shirts

Markus • (430th in this Competition) • 2 months ago • Options • Report • Reply ^ 5

Just in case anybody wants to check their preprocessing on an extreme example: id 098a464da9 is an extremely strong signal, easily visible by eye and therefore not at all typical.



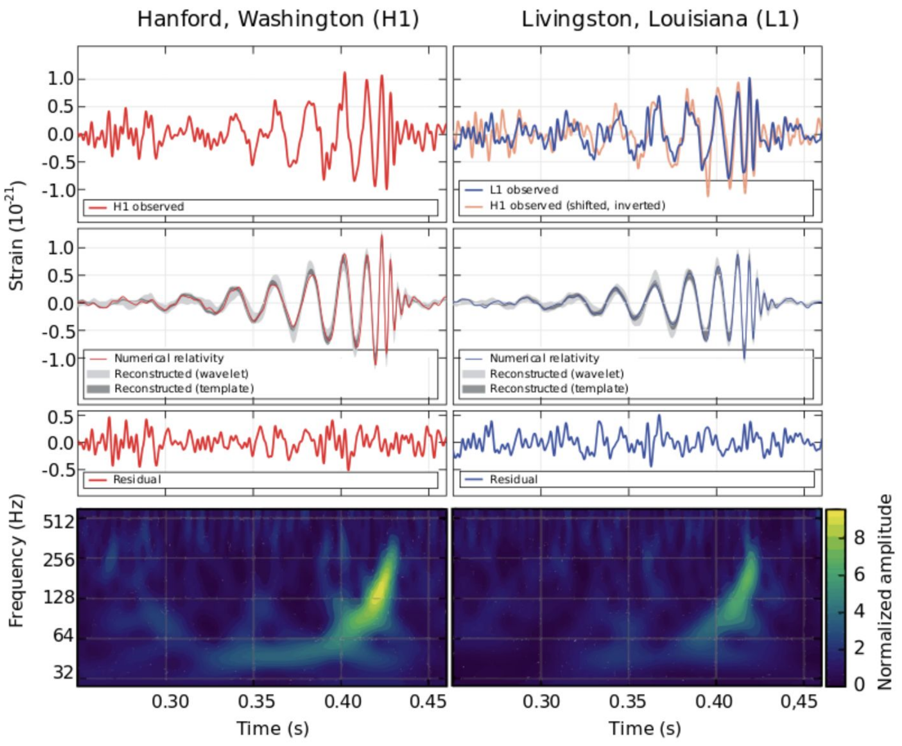
Abstract

- Signals of gravity waves due to black hole mergers have been detected at the Laser Interferometry Gravitational Observatory (LIGO). The LIGO group started a Kaggle.com challenge to use machine learning to detect (synthetic) gravitational waves from LIGO data. We chose to participate in this challenge. We employed multiple analysis techniques.

Illustration of gravitational waves produced by two orbiting black holes. [Credit: Henze/NASA]

Introduction

- Colliding black holes create gravity waves
- 3 detectors located around the world
- Signal not visible to the eye in this dataset
- Kaggle competition to use machine learning to find signals buried in the noise
- Dataset:
  - 560,000 Record Training Set
  - 226,000 Record Test Set
  - 50/50 split for signal/no-signal
  - Each record contains outputs from each of the 3 detectors
  - Each output is time series of length 4096 data points

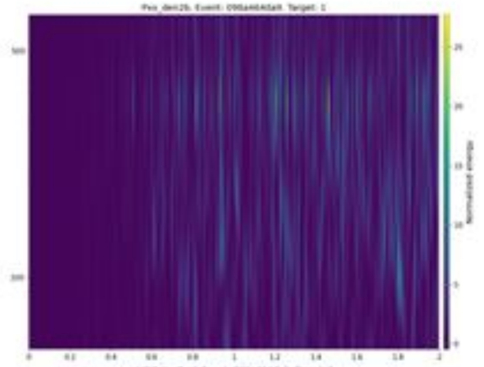


Some pre-processing

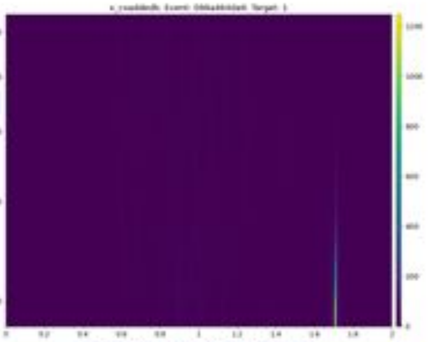
- Normalize Data
- Standardize Data
- Identify a sample with signal and test
  pre-processing approaches against it
- Periodograms:
- Co-Addition of channels
  - Naïve,
  - Interleaved
- Stochastic Resonance
  - Uniform Noise
- Q Transforms
  - https://dcc.ligo.org/public/0035/G040521/000/G040521-00.pdf

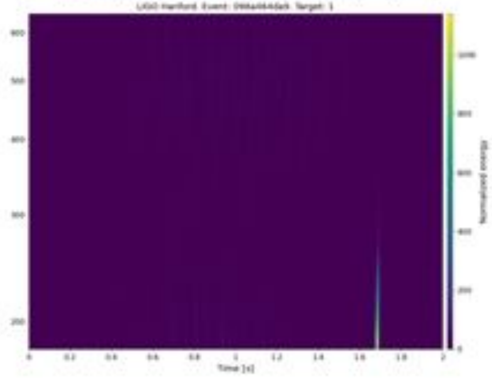
Schematic of one (of three) Laser Interferometer Ground Observatory (LIGO)

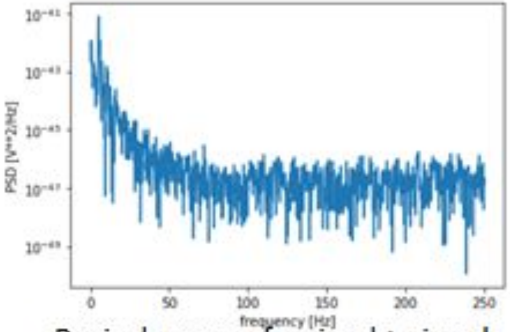
Comparing some different pre-processing outputs of noise + strong signal



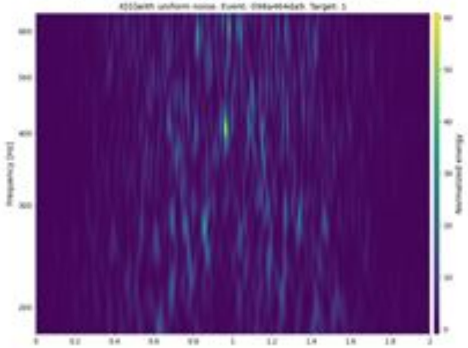
Spectrograms of different data preprocessing Steps some showing an identifiable signal and some not: 1.) Interleaved normalized data From 3 sources, 2.) co-added signals, 3.) naïve single channel, 4.) uniform noise added (stochastic resonance attempt).





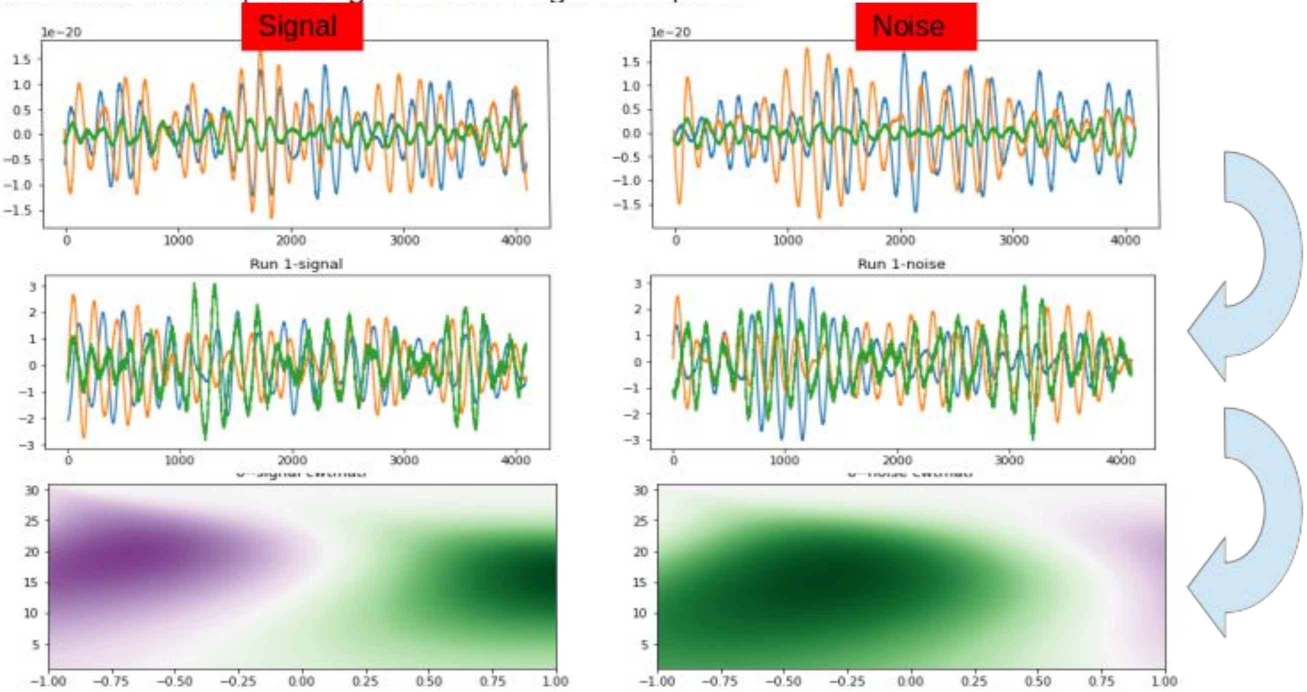


Periodogram of noised + signal



Continuous Wavelet Transform (CWT) Idea

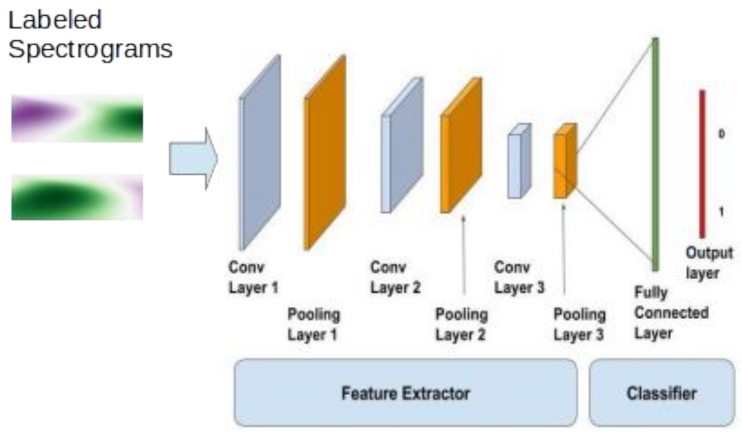
- Transform detector time series into spectrograms (images)
- Train CNN with spectrogram training examples



Normalize signals
(z transform)

Perform CWT transform
Apply bandpass filter
(20-500 Hz)

Convolutional Neural Net (CNN) for CWT




```
model = models.Sequential()
model.add(layers.Conv2D(32, (10, 10), activation='relu', input_shape=(300,97,1)))
model.add(layers.MaxPooling2D((6, 6)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(2))
model.summary()

model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])
model.fit(X_train,y_train,epochs=10)
```

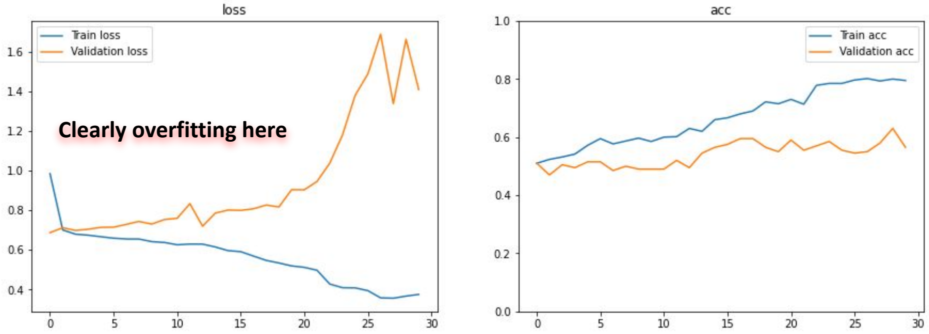
https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/s

CWT/CNN Results (1 of 2)

1000 examples, 60/20/20 split

```
#evaluting the model
model1.evaluate(X_test,y_test)
show_final_history(history1)
```

```
7/7 [==============================] - 0s 17ms/step - loss: 1.8847 - accuracy: 0.4350
```



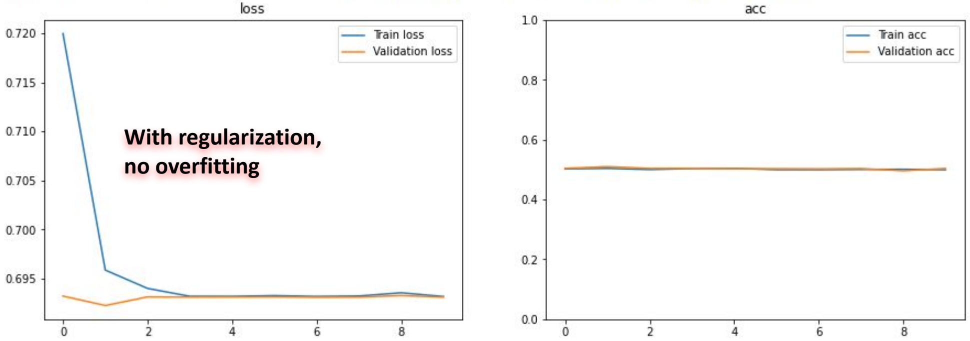
Epoch 1/30
19/19 [==============================] - 2s 52ms/step - loss: 0.9
Epoch 2/30
19/19 [==============================] - 1s 36ms/step - loss: 0.7
Epoch 3/30
19/19 [==============================] - 1s 37ms/step - loss: 0.6
Epoch 4/30
19/19 [==============================] - 1s 36ms/step - loss: 0.6745 - accuracy: 0.5417 - val_loss: 0.7046 - val_accuracy: 0.4950
...
Epoch 14/30
19/19 [==============================] - 1s 36ms/step - loss: 0.6151 - accuracy: 0.6200 - val_loss: 0.7861 - val_accuracy: 0.5450
Epoch 15/30
19/19 [==============================] - 1s 36ms/step - loss: 0.5965 - accuracy: 0.6600 - val_loss: 0.8014 - val_accuracy: 0.5650
Epoch 16/30
19/19 [==============================] - 1s 37ms/step - loss: 0.5913 - accuracy: 0.6667 - val_loss: 0.8000 - val_accuracy: 0.5750
Epoch 17/30
19/19 [==============================] - 1s 36ms/step - loss: 0.5691 - accuracy: 0.6800 - val_loss: 0.8078 - val_accuracy: 0.5950

CWT/CNN Results (2 of 2)

10000 examples, 60/20/20 split

```
model1.evaluate(X_test,y_test)
show_final_history(history1)
```

```
63/63 [==============================] - 17s 266ms/step - loss: 0.6931 - accuracy: 0.5050
```

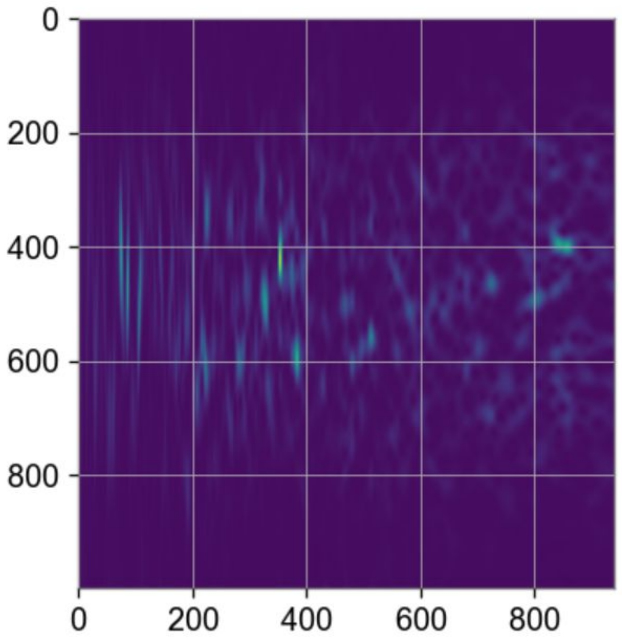


Epoch 1/10
188/188 [==============================] - 210s 1s/step - loss: 0.
Epoch 2/10
188/188 [==============================] - 199s 1s/step - loss: 0.
Epoch 3/10
188/188 [==============================] - 199s 1s/step - loss: 0.
Epoch 4/10
188/188 [==============================] - 204s 1s/step - loss: 0.
Epoch 5/10
188/188 [==============================] - 200s 1s/step - loss: 0.
Epoch 6/10
188/188 [==============================] - 200s 1s/step - loss: 0.6933 - accuracy: 0.4997 - val_loss: 0.6932 - val_accuracy: 0.5030
Epoch 7/10
188/188 [==============================] - 200s 1s/step - loss: 0.6932 - accuracy: 0.4997 - val_loss: 0.6931 - val_accuracy: 0.5025
Epoch 8/10
188/188 [==============================] - 200s 1s/step - loss: 0.6932 - accuracy: 0.5005 - val_loss: 0.6931 - val_accuracy: 0.5035
Epoch 9/10
188/188 [==============================] - 200s 1s/step - loss: 0.6936 - accuracy: 0.5010 - val_loss: 0.6933 - val_accuracy: 0.4960
Epoch 10/10
188/188 [==============================] - 199s 1s/step - loss: 0.6932 - accuracy: 0.4995 - val_loss: 0.6931 - val_accuracy: 0.5040

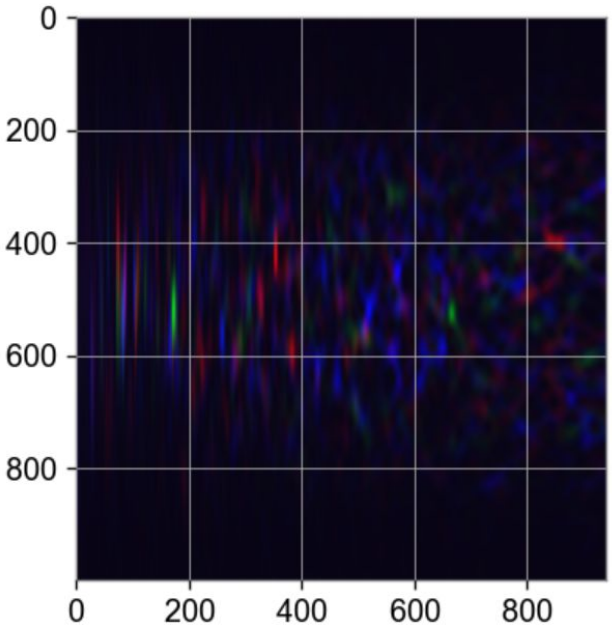
Anthony's Approach

- Used the gwpy library, which has useful tools for gravitational waves.
- Applied a Q-transform to the original time-series data. This produced an image (right) for each time-series.
- Stacked the 3 sources together as different channels, like in an RGB image.
- Used min-max normalization to keep each pixel value between 0 and 1.
- The transform greatly increased the size of the data, from 4096x3 to 1000x940x3.
- Resized the spectrogram image to 128x120x3 to make it more manageable, *hopefully* without losing too much information.
- Created a custom Generator class to feed the data into the model.
- Used a CNN to attempt to classify the spectrogram images.

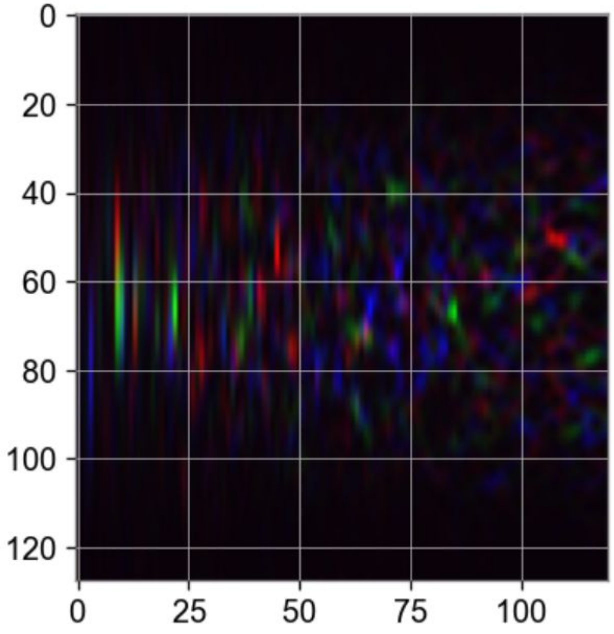
Q-transform



Q-Transforms, 3 channels



Full size, 1000 x 940



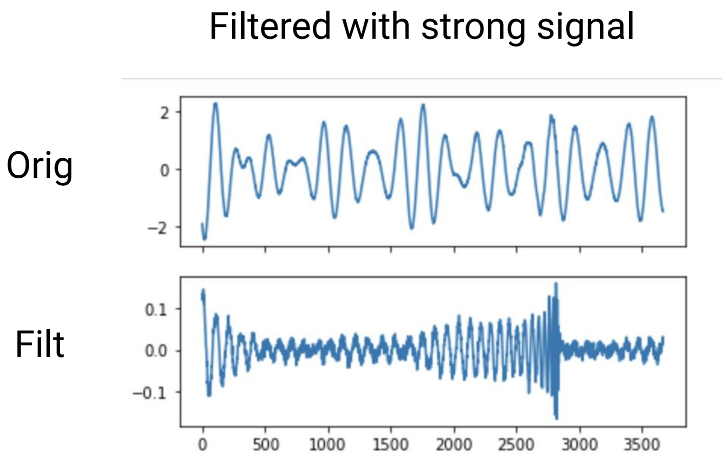
Reduced size, 128 x 120

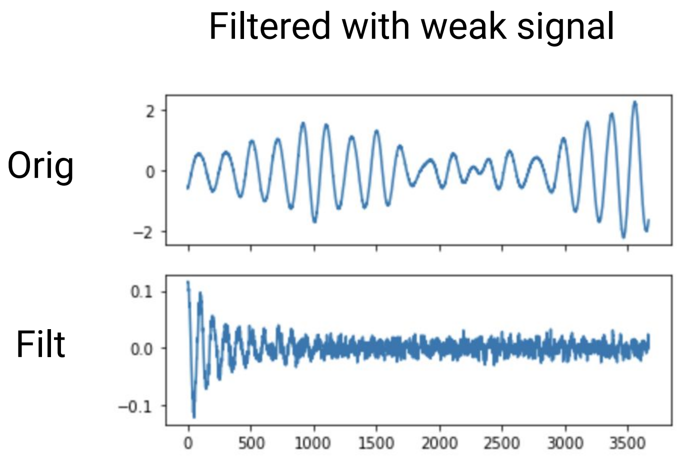
Model

```
model = models.Sequential()
model.add(layers.Conv2D(32, (6, 6), activation='relu',
     input_shape=(data_shape[0], data_shape[1], data_shape[2])))
model.add(layers.MaxPooling2D((4, 4)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Preprocessing: Filtering Approach

- Strong noise between 10 - 15Hz and also at 300Hz
- Tried several filters
    - Low pass, band pass
    - Butterworth, Chebyshev
- Final Filter Chosen
    - Bandpass Butterworth filter
    - 20th order, 20 - 500Hz

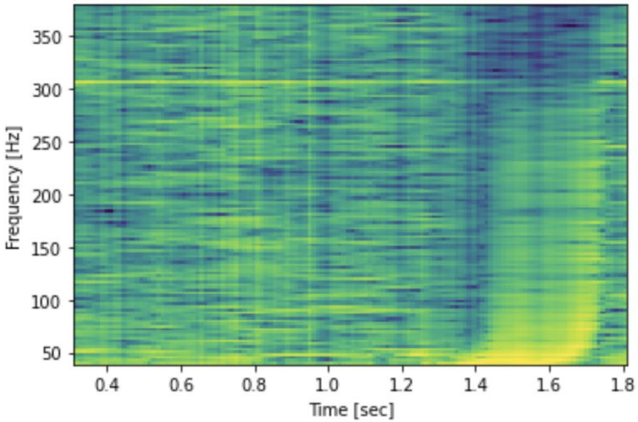




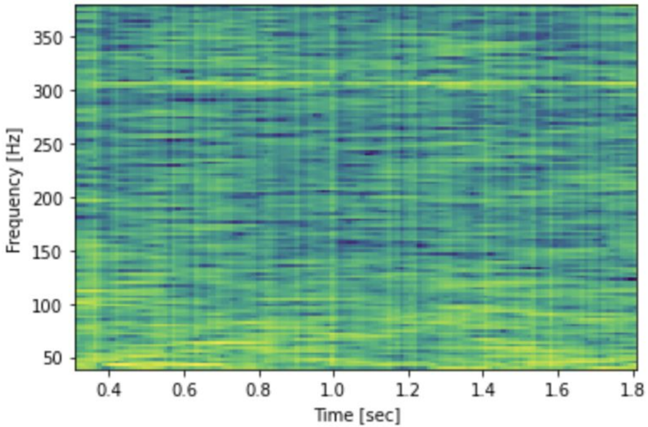
Preprocessing: Filtering - Spectrograms

- Turn the 1D data into 2D for CNN
  - Short Time FFT (STFT)
    - Used different window lengths & overlaps
    - Used different windows: Hann, Hamming, Tukey
  - Spectrogram
    - Frequency vs Time

Spectrogram showing strong signal

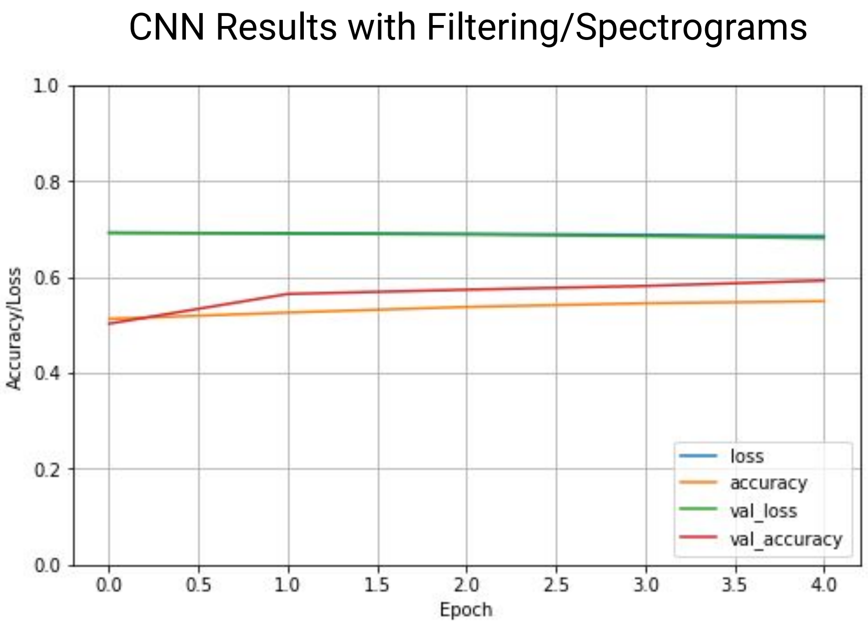


Spectrogram with weak signal



Preprocessing: Filtering - CNN Results

- Run CNN on 2D data
  - Similar model setup to other CNNs shown
  - 2D input data is 128 x 128 x 3
  - Tried SGD and ADAM
- Final run:
  - 100,000 Training Samples
  - 5 Epochs
  - Batch size: 250
  - Loss = 0.685
  - Val Loss = 0.681
  - Accuracy = 0.550
  - Val Accuracy = 0.592



Conclusions

- Focused on different methods for pre-processing
- Different preprocessing did not significantly change the CNN model accuracy
- Future efforts could concentrate on Neural Network improvements
  - Hyperparameter tuning
  - Use Deep Neural Network
  - Use Recurrent Neural Network - directly on the time series

Reference Materials

- https://www.kaggle.com/xuzongniubi/g2net-efficientnet-b7-baseline-training
- https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
- https://www.tensorflow.org/tutorials/images/cnn
- https://colab.research.google.com/github/gw-odw/odw-2021/blob/master/Tutorials/Day_1/Tuto%201.3%20Q-transforms%20with%20GWpy.ipynb
- https://www.analyticsvidhya.com/blog/2021/08/beginners-guide-to-convolutional-neural-network-with-implementation-in-python/
- https://medium.com/intelligentmachines/convolutional-neural-network-and-regularization-techniques-with-tensorflow-and-keras-5a09e6e65dc7
- https://arxiv.org/pdf/1809.04356.pdf
- https://journals.aps.org/prd/abstract/10.1103/PhysRevD.73.122003