Machine Learning Classification of Gravitational Waves

Tom Casaletto
Advanced Technology Center
Lockheed Martin
Chicago, USA

Anthony Fisher
OPCoE
Lockheed Martin
Sunnyvale, USA
anthony.fisher@sjsu.edu

Phillip Shirts
Solar and Astrophysics Lab.
Lockheed Martin
Palo Alto, USA
phillip.g.shirts@lmco.com

Joel Wiser
OPCoE
Lockheed Martin
Littleton, CO, USA
joel.wiser@sjsu.edu

***Abstract*— Several data preprocessing, analysis and machine learning techniques were employed in our group's effort to identify synthetic gravitational waves from data provided by the Kaggle.com G2NetGravitation Wave Detection Research Prediction Competition, "Find gravitational wave signals from binary black hole collisions."**

Keywords—gravitational waves, machine learning, Convolutional Neural Net (CNN), Q-Transforms, data pre-processing, continuous wavelet transform, spectrograms, Kaggle contest

I. Introduction

Gravitational waves are "ripples" in space-time caused by the movement of massive objects. The faster the movement and the more massive the object, the stronger the gravitational wave created. Albert Einstein's 1915 general theory of relativity predicted the existence of gravitational waves, and they were detected for the first time 100 years later, in 2015, by twin Laser Interferometer Gravitational-wave Observatory (LIGO) observatories. The gravitational waves that were detected were generated by colliding black holes. The gravitational wave signals are very weak, of order 10^{-21}. Convolutional Neural Network (CNN) approaches are a promising avenue for identifying the weak gravitational wave signals. In order to encourage development of CNN approaches to Gravitational wave detection, Kaggle.com is hosting a competition, "G2Net Gravitational Wave Detection, "Find gravitational wave signals from binary black hole collisions" with $15,000 in prizes for top signal detectors using CNN. (https://www.kaggle.com/c/g2net-gravitational-wave-detection). The final submission deadline is September 20, 2021.

II. Data Set

The data set consists of 560,000 training records and 226,000 test records: 50% of the records contain a signal and 50% do not. The embedded signals were synthetically generated. "The parameters that determine the exact form of a binary black hole waveform are the masses, sky location, distance, black hole spins, binary orientation angle, gravitational wave polarisation, time of arrival, and phase at coalescence (merger). These parameters (15 in total) have been randomized according to astrophysically motivated prior distributions and used to generate the simulated signals present in the data, but are not provided as part of the competition data. Each data sample (`npy` file) contains 3 time series (1 for each detector) and each spans 2 sec and is sampled at 2,048 Hz." There are approximately 77 GB of data in the test and train data sets.

III. Continuous Wavelet Transform

In this section we describe the approach to take advantage of the success of Convolutional Neural Networks (CNN) in processing images for classification. This has been applied across multiple domains. Here we propose to transform the 1-dimensional time series from each detector site into a 2-dimensional image using the Continuous Wavelet transform (CWT). This is accomplished by taking the spectrogram of the CWT.

The CWT is given by transforming a time signal into its frequency (psi) and amplitude (a) components as shown in Fig. 1.

A diagram of the pre-processing is shown in Fig. 2. A training example where there is a gravitational wave is shown

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t)\bar{\psi}\left(\frac{t-b}{a}\right) dt$$

Fig. 1. Continuous wavelet transform equation.

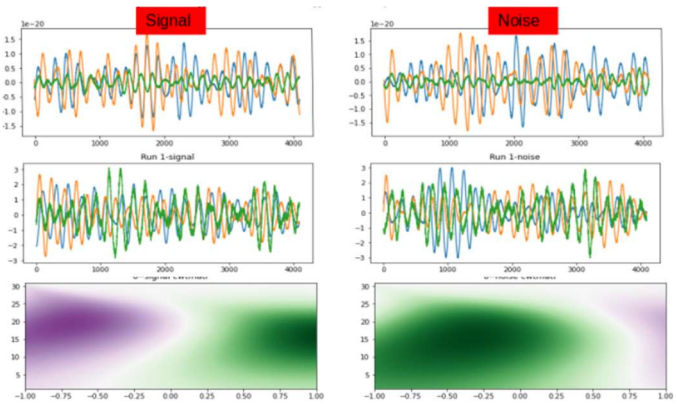

Fig. 2. Pre-processing steps for continuous wavelet transform method.

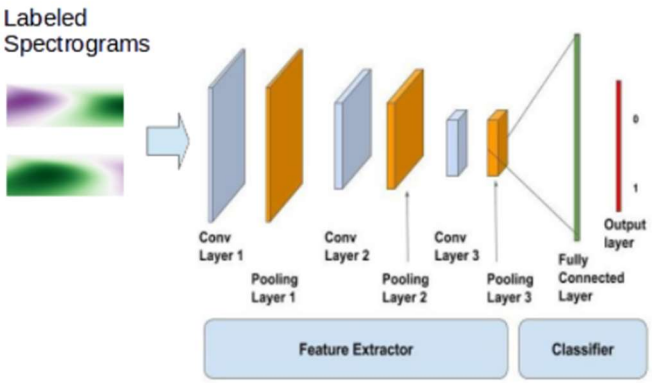

Fig. 3. Diagram of CNN structure used with continuous wavelet transform approach.

on the upper left (Signal). Similarly, a training example where there is no gravitational wave (just detector noise) is shown in the upper right (Noise). The first row of diagrams shows detector output for each of the 3 sites for 2 seconds (4096 samples). From the naked eye it is hard to tell if these represent time series from a gravitational wave or just noise. Note the scale of the Y-axis is 1e-20. The second row of diagrams shows the result of the first step of pre-processing: normalizing the signals by the z-transform. Again it is difficult to tell the difference between detector output with a gravitational wave and with just noise. The third row of diagrams shows the CWT spectrogram of the normalized signal after applying a bandpass filter to remove frequencies below 15 Hz and above 500 Hz. It was given in the problem statement that typical gravitational waves are around 350 Hz.

Fig. 3 shows a diagram of the CNN used to process the spectrogram images. Each image is 300x97 pixels. As shown the CNN consists of multiple convolutional and pooling layers for the feature extractor. The output layer classifies the image as 1 for gravitational wave detected or 0 for no detection.

Fig. 4 shows the summary description of the Tensorflow model.

Fig. 5 shows the initial results of running 1000 training examples through the model using a 60/20/20 training/test/validation split. During the training, the accuracy increased with each epoch, eventually reaching 0.795 after 30 epochs. The divergence of the accuracy and validation shows that overfitting is happening. Though it appears our model is getting more accurate, running the test data through achieves a

```
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)
```

Fig. 4. CNN description.

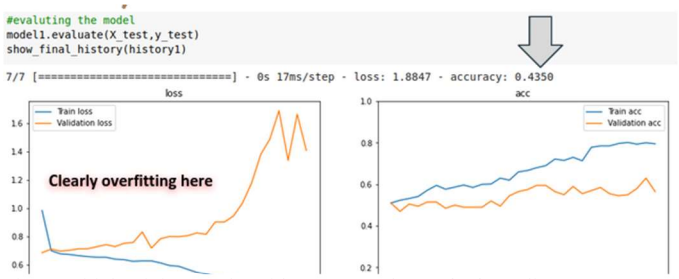

Fig. 5. Initial training results with 1000 samples (60/20/20 split).

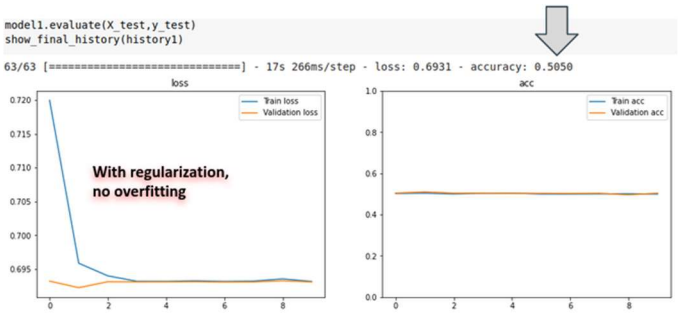

Fig. 6. Results with 10,000 training samples and regularization.

score of 0.43. If this were the entirety of the training set we should actually take the opposite of whatever our model predicts! However, the low score is probably due to the small amount of training/test samples used for this analysis.

Fig. 6 shows the results of running 10000 training examples through the model using a 60/20/20 training/test/validation split. Here we use regularization in training the model and see that overfitting is not occurring. However, our test accuracy is 0.505 so our model is not much better than flipping a coin. This indicates our pre-processing is not doing much to differentiate detector output with gravitational waves from those with just noise. So more work needs to be done on the pre-processing step. Some ideas include:

1) Changing the bandpass filter parameters to allow more or less data through.

2) Making larger/smaller images to pass into the CNN.

In summary, an approach was developed to perform pre-processing on detector output using the Continuous Wavelet Transform. The data was processed through a Convolutional Neural Network using Tensorflow. Results were inconclusive regarding the ability of the model to detect gravitational waves and ideas for further study were presented.

IV. Q-Transforms

A. Technical Approach

For pre-processing in this approach, we transformed each time-series data sample into a spectrogram using a Q-transform. The resulting spectrogram is an image showing the power of different frequencies within the signal as it moves through time. The resulting images were also resized to a smaller resolution to be more manageable.

Each sample included readings from three different gravitational wave detectors. Each reading was transformed into

```
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'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Fig. 7. CNN configuration for Q-transform approach.

a spectrogram and then the three were stacked together as different channels, much like the red, green, and blue channels that make up an RGB image. This stack of three spectrograms was used as the input to a convolutional neural network.

Because each sample was fairly large, and there were a huge number of samples in the dataset, it was impossible to hold all samples necessary for training in memory at once. So, to train the network (or make test predictions), we wrote a custom generator class derived from keras' Sequence class. This allowed only a small batch of data to be loaded into memory at once. The generator would read in a batch of data (either the raw time-series or pre-processed data, if available), perform the necessary transformations (if desired), and send the result to the neural network model. It would repeat this for as many batches as were necessary.

The model was configured as shown in Fig. 7. The spectrogram was resized so that the values data_shape[0], data_shape[1], and data_shape[2] were 128, 120, and 3 respectively.

B. *Experimental Methodology*

We initially began training the convolutional neural network model on smaller amounts of data, 1,000 or fewer samples. With this small amount of data, the model appeared to overfit the data, similar to as was seen in the wavelet approach. Over 10-20 epochs, the training accuracy would increase nearly to 1 while training loss decreased. But, the validation accuracy stayed nearly the same (around 0.55) or decreased in later epochs. We attempted to counter this overfitting effect by using a larger number of data samples.

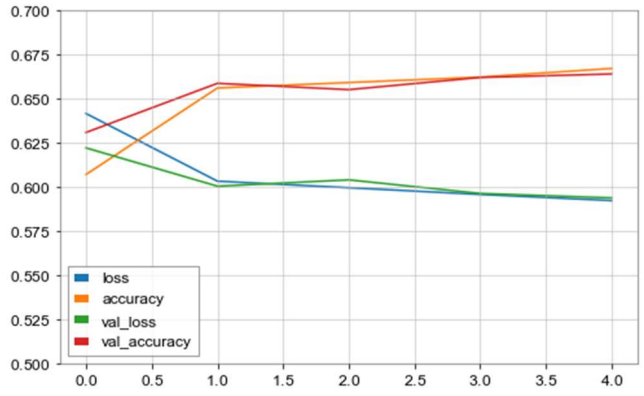

Fig. 8. Q-transform training results using 200,000 samples after 5 epochs. Epoch on x-axis.

The best results for the Q-transform method were obtained by using the first 200,000 data samples for training and validation with a random 80% train, 20% validation split. After five training epochs, the model was able to achieve 66% accuracy on the validation set. These results are shown in Fig. 8.

V. Filtered Spectrograms

A. *Technical Approach*

Another approach taken given that the data was very noisy was to first try filtering the time series data. Several filters were tried using the standard set of filters found in the "scipy" library. The final filter chosen was a 20th order Butterworth bandpass filter. The Butterworth filter has a flat response which was desired in the region of interest of between about 20 to 500 Hz. The price is that the dropoff at the band edges is not as steep as other filters. To try and make up for that a higher order filter was used. To gauge the response of the filter on the data, a time series that had a stronger signal than most was used. Unfortunately, a strong enough filter could not be found to make the weaker signals in the time series emerge enough to be seen by eye. The best that could be done was to try to reduce the noise levels without reducing the signal.

Next the filtered signals were passed through a Short Time Fourier Transform (STFT). The purpose of this was to do several Fourier transformations while sliding along the signal in time. In the strong signal, this showed the increase in frequency content when the signal was present. In the weak signals,

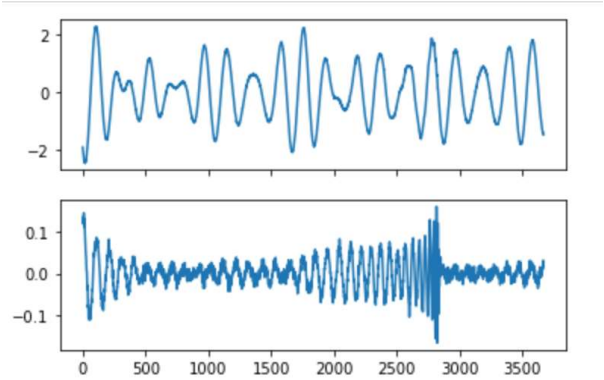

Fig. 9. Unfiltered/Filtered with strong signal.

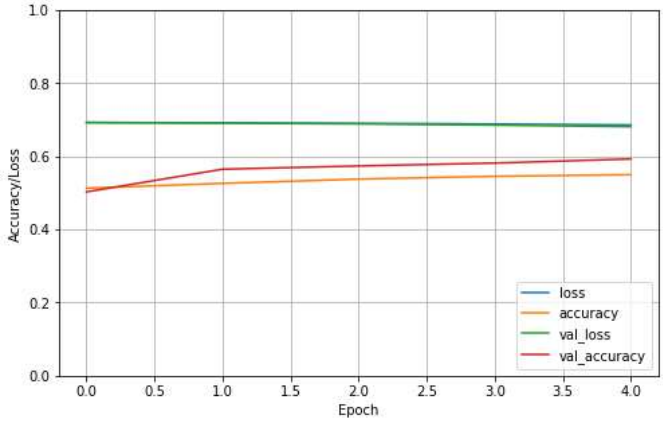

Fig. 10. CNN Results with filtering and spectrograms.

nothing was obvious by eye, but the hope was that by running the transformed data through a CNN that it would be able to detect what could be faint patterns.

B. Experimental Methodology

The data generated using filtering of the time series data that was then run through an STFT to create spectrograms was run through the same CNN that was used for the Q-transforms. The spectrograms were first reduced to 128 x 128 and stacked by 3s corresponding to the 3 sensors also similar to the Q-transform data. Several parameters were adjusted to try to achieve better results. The best results were found by running 5 epochs of 100,000 training samples with batch sizes of 250. With this setup, the accuracy and validation accuracy of the model steadily increased 0.55 and 0.59 respectively. When run on all test data and submitted to Kaggle, the model scored 0.6681.

VI. Other Pre-Processing Exploration

We identified data pre-processing as key to signal identification in data. We explored several other pre-processing approaches:

1) Co-addition of the three synthetic signals.
Any gravitational wave signal should appear at all LIGO observatories. Even though any signal present in the data might be offset by as much as 7 milliseconds due to speed-of-light (and gravitational waves) between the different LIGO observatories, depending on the orientation of the signal in space relative to the observatories, we thought it reasonable to see if co-addition of co-temporal data instances (where were ingested into our computing environment as equal length numpy arrays) would increase the relative strength of the signal, and hence its detectability. Prior to co-addition each signal channel was standardized.

2) Interleaving the three co-temporal synthetic signals.
Similarly, for three LIGO instance data elements, the array elements were combined into a single numpy arrav with their elements interleaved, with the thought that that might make it easier for a periodogram, or other technique to identify a common signal:

```
#interleave and then standardize three co-temporal gravitational wave signal candidates
def interleave_samples( _id, target):
    path = convert_id_to_path(_id)
    x = np.load(path)
    print(_id)
    x0 = x[0].reshape(-1,1)
    x1 = x[1].reshape(-1,1)
    x2 = x[2].reshape(-1,1)
    x_123 = np.dstack((x0,x1,x2)).ravel()
    x_123 = x_123.reshape(-1, 1)
    scaler_xcomposite=StandardScaler() #original
    scaler_xcomposite.fit(x_123)          #original
    x_out = scaler_xcomposite.transform(x_123)
    print("_id == ", _id)
  return x_out
```

```
print(max(x[0]))
uniform_x0=np.random.uniform(low=0.0, high=.1*max(x[0]), size=len(x[0]))
print(uniform_x0.shape)
plt.plot(uniform_x0+x[0])
```

```
1.411032460706869e-20
(4096,)
[<matplotlib.lines.Line2D at 0x7f436fcb2790>]
```

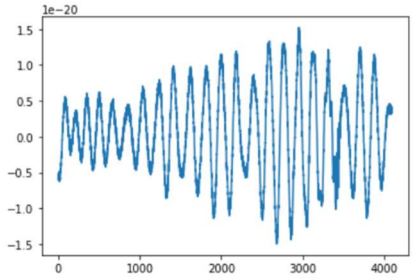

Fig. 11. Uniform noise injection.

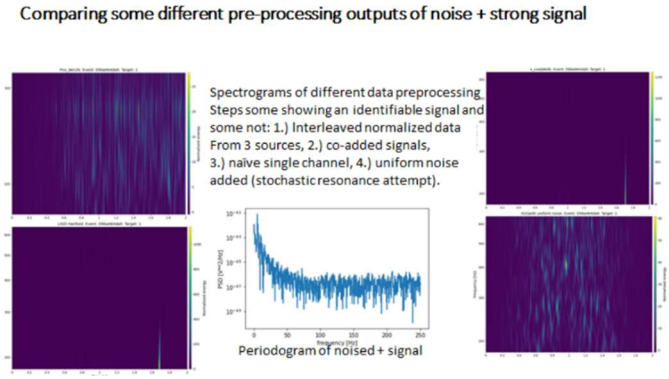

Fig. 12. Comparison of different pre-processing approaches.

3) Stochastic resonance
In the cited paper, a non-linear cavity was used to enhance in the LIGO to enhance the detectability of the gravitational wave signal. We experimented with injecting uniformly distributed noise of low relative strengths into the provided data set elements. So far, no enhanced detectability of gravitational signals was noticed.

4) Periodograms & Q Transforms
A strong known signal from the data set of signals was identified and the above preprocessing steps were there analyzed with periodograms and Q-transform to see if the strong signal still retained visible identifiability. In the case of simple co-addition this was still true.

VII. Conclusion

We used multiple different approaches to pre-processing the gravitational wave data and had mixed success. The filtering approach as well as the Q-transforms showed the most promise, but both still had much room for improvement.

VIII. Group Contributions

All group members had good contributions to this project. Phil focused mostly on identifying different possibilities for pre-processing approaches. Tom chose the continuous wavelet transform approach and built a model based on that. Joel explored the filtering methods and the associated model. Anthony chose the Q-transform method and built a model around that.

Our code can be found in our Github repo at https://github.com/TomCasaletto/CMPE257.

References

[1] "GW open Data Workshop #4 (2021)", [Online]. Available: https://www.gw-openscience.org/static/workshop4/. [Accessed: 14-Sep-2021].

[2] J. C. Brown, "Calculation of a constant Q spectral transform," J. Acoust. Soc. Am. 89 (1), January 1991, pp.425-434.

[3] A. Amidi and S. Amidi, "A detailed example of how to use data generators with Keras", [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly. [Accessed: 15-Sep-2021].

[4] Ismail Fawaz, H., Forestier, G., Weber, J. *et al.* Deep learning for time series classification: a review. *Data Min Knowl Disc* **33,** 917–963 (2019). https://doi.org/10.1007/s10618-019-00619-1.

[5] G. G. Karapetayan, "Application of stochastic resonance in gravitational-wave interferometer," in Physical Review D, vol. 73, iss. 12, https://link.aps.org/doi/10.1103/PhysRevD.73.122003