Core Project
(PRJ3001)

Final Report

Detecting Gravitational Waves

By

Suhas Gumma

1700278C203

Supervisor

Dr. Soharab Hossain Shaik

Associate Professor SOET-CSE

Department of Computer Science and Engineering
School of Engineering and Technology

December 2021

Acknowledgement

I would like to express my sincere gratitude to my supervisor Dr Soharab Hossain for constant guidance and encouragement throughout the course of the project. He has given various options and the freedom to choose the project. He has given access to the GPU server for us to make use of. I owe my thanks to BML Munjal University for creating a curriculum where students get involved in research and can implement what they have learnt in graduation.

Thank You.

Suhas Gumma

1700278C203

2018

04/12/2021

BML MUNJAL UNIVERSITY™

BML Munjal University, Gurgaon, Haryana

Index

Abstract

Understanding Gravitational waves are the key to further probing into the workings of our Universe. GW signals are unimaginably tiny ripples in the fabric of space-time and even though the global network of GW detectors are some of the most sensitive instruments on the planet, the signals are buried in detector noise. So, it is a challenge of utmost importance to detect the presence of gravitational waves. In recent times there is more and more usage of deep learning models to solve problems of various kinds. This is not an exception either. In the following sections, we describe how to transform the time series signals and develop a Convolutional neural network model to detect the gravitational wave present in the signal.

1. Problem Definition

Understanding Gravitational waves are the key to further probing into the workings of our Universe. GW signals are unimaginably tiny ripples in the fabric of space-time and even though the global network of GW detectors are some of the most sensitive instruments on the planet, **the signals are buried in detector noise.** Developing proper ML models to detect the Gravitational waves is the problem we are going to tackle.

2. Project Objectives

The main objective of the project is to detect GW signals from the mergers of binary black holes. The data we are dealing with is not gathered from the mergers but rather a carefully simulated one. Specifically, to build a model to analyze simulated GW time-series data from a network of Earth-based detectors.

To further simplify it, we are provided with a training set of time series data containing simulated gravitational wave measurements from a network of 3 gravitational wave interferometers (LIGO Hanford, LIGO Livingston, and Virgo). **Each time series contains either detector noise or detector noise plus a simulated gravitational wave signal. The task is to identify when a signal is present in the data (target=1).** Though the task seems simple, it is quite challenging and significant nonetheless.

3. Challenges

- Processing the Signal.
- The time series should be transformed into other forms to be used with different ML models.
- Dealing with large data.
- High computational power is required especially when Deep Learning models come into the picture.

4. Deliverables

- As mentioned above, the objective of our project/model is to predict the presence of GW signals in the data. Our model will output a probability score(between 0 and 1) of the presence of GW Signal.
- The model developed will detect the signal with an AUC score of at least 0.75.

5. Literature Review

Time series classification is becoming more prominent and common these days. But, there are relatively few specific algorithms for classifying the time series. One way to deal with this is to consider each data point in the time series as an independent variable and apply normal classification algorithms like logistic regression. But, it completely takes one of the fundamental qualities of a time series out of the equation, the order. To approach this problem, there are many time-series specific algorithms coming into the picture. A few of them are:

- Distance-based (KNN with dynamic time warping)
- Interval-based (TimeSeriesForest)
- Dictionary-based (BOSS, cBOSS)

- Frequency-based (RISE — like TimeSeriesForest but with other features)
- Shapelet-based (Shapelet Transform Classifier).

Some of these were tried to detect the gravitational waves mixed in noise but the desired accuracy is not achieved.

6. Description of the Dataset

- The training set of time series data contains simulated gravitational wave measurements from a network of 3 gravitational wave interferometers (LIGO Hanford, LIGO Livingston, and Virgo). Each time series contains either detector noise or detector noise plus a simulated gravitational wave signal.
- 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.
- In summary, there are 560K npy files for training and 226K npy files for testing in which each npy file contains 3-time series data.
- And 2 CSV files. One for training and one for testing. Each contains 2 columns. The first column represents the npy file id(unique of course) and the second column contains the target (0 if GW wave is not present else 1).

7. Proposed Methodology

7.1) Connect to the college GPU server for computational resources.

- Connect to the college VPN.
- Establish an ssh tunnel between the Personal Computer and the GPU server.
- Create a Jupyter notebook in the GPU server and access it through the Personal Computer.

- All the work will be done in this Jupyter notebook.

7.2) Download the dataset.

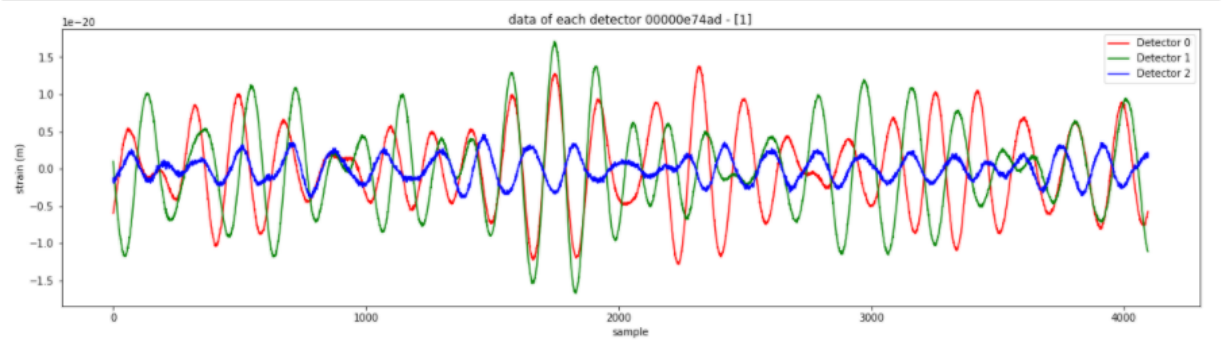
- The dataset is quite large (75 GB). So, download it directly into the GPU server.
- Download it by running simple commands and Kaggle API.

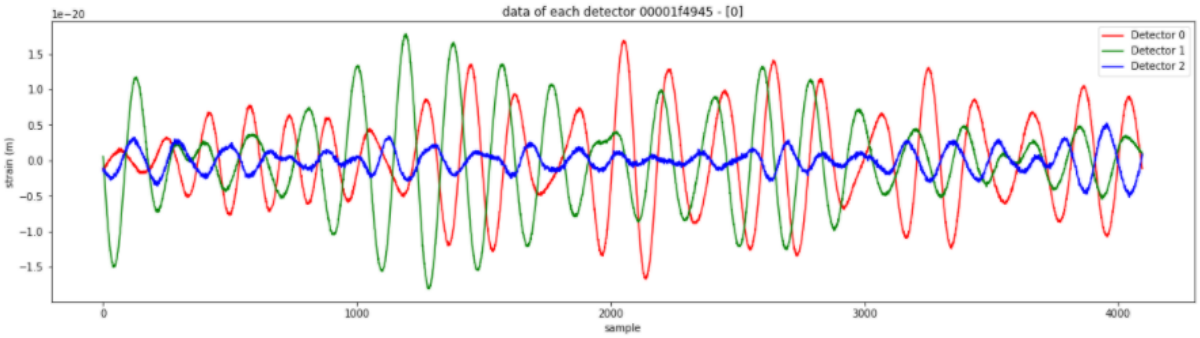
7.3) Pre-processing the data.

7.3.1 Access unit data by file name.

- The structure of the data is somewhat complicated. The first three characters in the file name give the structure of the dataset. Using that load the data into a NumPy array.

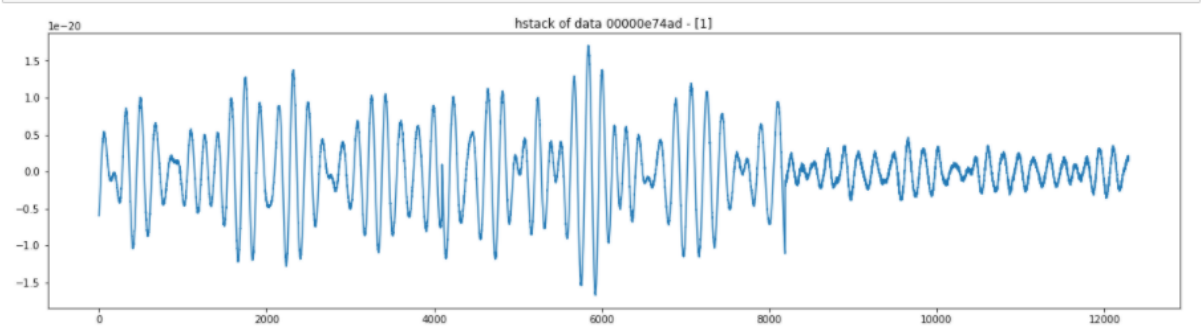
```
#Function to get 3 signals in an np array from the given id.
#If Id 00001f4945 id is given, gets the np array of three signals from that id.
def getThreeSignalArray(identifier, isTrain):
    firstChar = identifier[0]
    secondChar = identifier[1]
    thirdChar = identifier[2]

    if isTrain:
        path = f'train/{firstChar}/{secondChar}/{thirdChar}/{identifier}.npy'

    else:
        path = f'test/{firstChar}/{secondChar}/{thirdChar}/{identifier}.npy'

    return np.load(path)
```

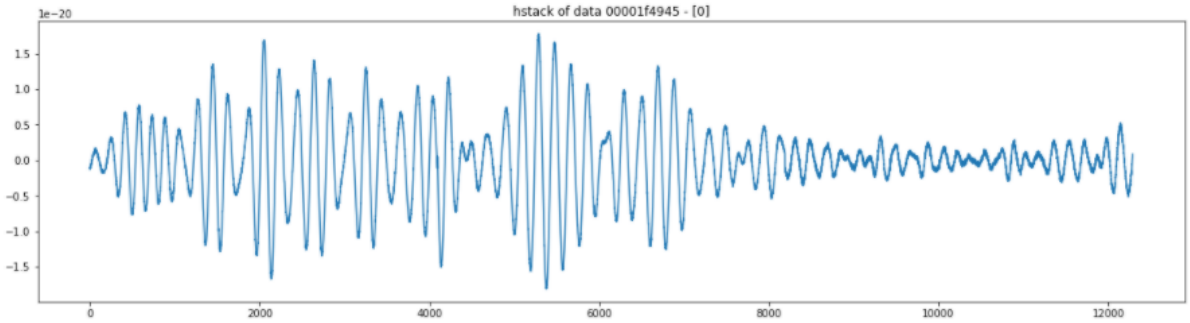




7.3.2) Stack the three time-series data into a single NumPy array.

- For the convolutional neural networks, we tried many ways of passing three-time series data.
- Out of all the ways, the one which gave higher accuracy was when all the three merged together.
- We can merge them and still manage to split them again by using the “hstack” method of NumPy.





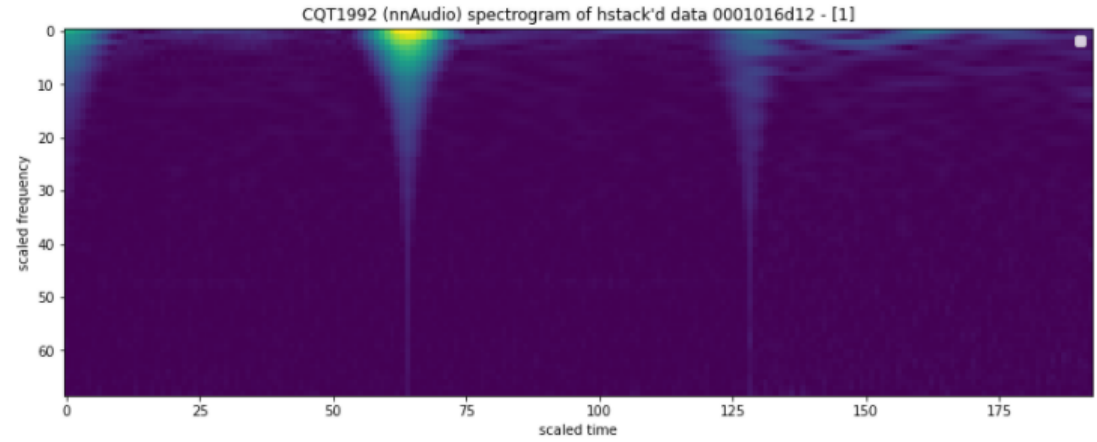
7.3.3. Normalize the data.

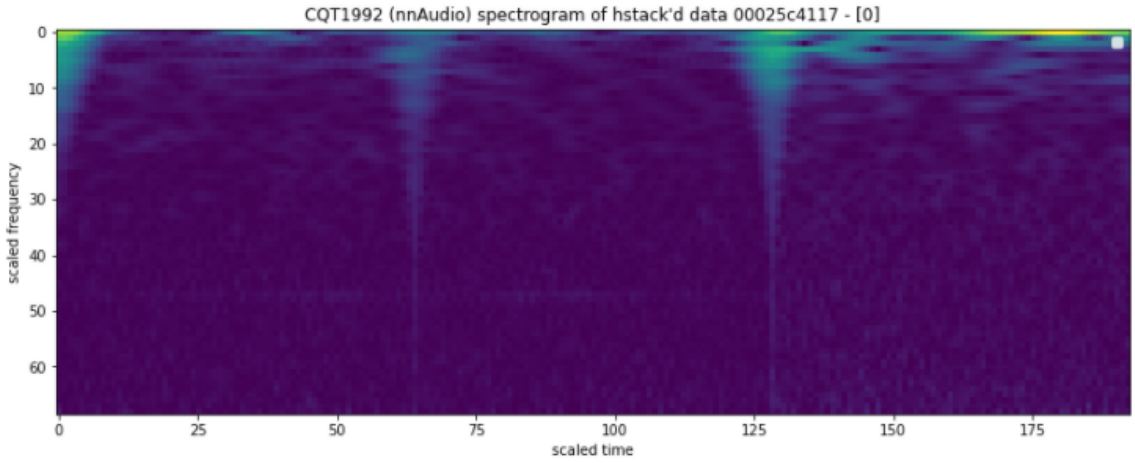
- Data is normalised to maintain a common scale and to speed up the learning process.
- Take the merged time series of all three detectors and divide each value by the highest value present in the data.
- Now every value is in the range of 0-1.

7.4. Get the CQT spectrogram of the merged time series

- The data we have is in the time domain. We convert it into the time-frequency domain so that we have a spectrogram of the time series data. This spectrogram will be passed to the CNN model to get the classification done.

- Typically a Fourier transformation is used to transform time-domain data to frequency domain data. Here we used Q-transformation which is similar to the Fourier transformation. Q-transformation has been giving more accurate results, especially with the data dealing with gravitational waves.
- Q- transformation was introduced by J.Brown in 1992. We got the Q-transformation spectrogram by using the “CQT1992v2” class in the “nnAudio” library which is implemented based on the paper “An efficient algorithm for the calculation of a constant Q transform” written by J.Brown in 1992.





7.5. Dealing with large data.

The dataset has around 520K NumPy files and has a size of 75 GB. Usually, we load the whole dataset once. It won't be a problem if the dataset is small. But, we are dealing with large data here. Even the most state of the art configuration won't have enough memory space to process the data the way we used to do it. There is a solution to tackle this problem. We generate the dataset on multiple cores in real-time and feed it right away to the deep learning model. This can be implemented very easily by using the "DataGenerator" class provided by Python's high-level package, Keras.

```
class DataSetGenerator(Sequence):

    def get_sample(self, id, is_train):
        pass

    def __init__(self, identifiers, y=None, batch_size=256,
        shuffle=True, no_channels=10, no_classes=10, name="Unknown DataSet"):
        self.shuffle = shuffle
        self.batch_size = batch_size
        self.identifiers = identifiers
        self.y = y
        self.name = name
        if y is not None:
            self.is_training = True
        else:
            self.is_training = False
        self.shape = self.get_sample(self.identifiers[0], self.is_training).shape
        print(f"{self.name} - Shape of each sample: {self.shape}\n")


    def __len__(self):
        return math.ceil(len(self.identifiers)/self.batch_size)

    def __getitem__(self, index):
        batch_ids = self.identifiers[index * self.batch_size:(index + 1) * self.batch_size]
        if self.y is not None:
            batch_y = self.y[index * self.batch_size: (index + 1) * self.batch_size]


        list_x = np.array([self.get_sample(x, self.is_training) for x in batch_ids])

        batch_X = np.stack(list_x)


        if self.is_training:
            return batch_X, batch_y
        else:
            return batch_X

    def on_epoch_end(self):
        if self.shuffle and self.is_training:
            ids_y = list(zip(self.identifiers, self.y))
            shuffle(ids_y)
            self.identifiers, self.y = list(zip(*ids_y))
```

7.6. Defining the CNN architecture.

The CQT spectrograms generated from the merged time series are passed as inputs for the CNN and an output ranging from 0 to 1 is given with ‘0’ being “No Signal” and ‘1’ indicating the presence of the gravitational wave signal.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 69, 193, 3)	30
efficientnet-b1 (Functional)	(None, None, None, 1280)	6575232
global_average_pooling2d_1 (	(None, 1280)	0
dense_2 (Dense)	(None, 32)	40992
dense_3 (Dense)	(None, 1)	33

Total params: 6,616,287
Trainable params: 6,554,239
Non-trainable params: 62,048

7.6.1) The Input Layer

- The size of CQT spectrogram generated by using nnAudio library is (69 X 193). So, the input shape will also be (69 X 193).

7.6.2) The Convolutional Layer.

- The second layer is the “conv2D” layer with 3 kernels and each kernel having the shape (3X3).

7.6.3) The EfficientNet-b1 layer.

- The EfficientNet family are CNN models designed by GoogleAI. As the name suggests, they are designed to get efficient results with fewer parameters. EfficientNet architecture is the most state-of-the-art architecture available currently. We added “efficientnet-b1” to our model because it gave better results compared to “efficientnet-b0”. The weights are pre-trained from ‘imagenet’ dataset.

7.6.4) Dense Layer with RELU activation function.

- ‘efficientnet-b1’ does the flattening and the final flat layer is connected to another dense layer with ‘RELU’ activation function and size 32.

7.6.5) Final layer with sigmoid activation function.

- As we need a final value between 0 and 1, we added a sigmoid layer in the end.

```
def get_cqt1992_model(shape):
    model = Sequential([
        layers.InputLayer(input_shape=(shape[0],shape[1],1)),
        layers.Conv2D(3,3,activation='relu',padding='same'),

        EfficientNetB1(include_top=False,input_shape=(),weights='imagenet'),
        layers.GlobalAveragePooling2D(),
        layers.Dense(32,activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])

    model.summary()


    model.compile(optimizer=optimizers.Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=[metrics.AUC()])
    return model
```

7.7) Loss Function

- As we are dealing with binary classification problems, the Binary-cross-entropy function suits best to measure the loss.

7.8) Metrics to validate the results.

- For Binary classification problems, “AUC” score is the best metric to measure the validity of the model.
- A graph is plotted between precision and recall with varying thresholds. The area under the resulting curve is the “AUC”.
- “AUC” gives the measures the ability of a classifier to distinguish between different classes.

7.9) Training the model.

- Load the datasets using the Keras DatasetGenerator class.
- As we don’t have labelled test datasets, we split the training dataset into train and validation datasets to evaluate the performance of the model.

- Pass the training dataset for the CNN model built.

```
model.fit(
    train_dataset,
    validation_data=valid_dataset,
    epochs=5,
    verbose=1,callbacks = [callbacks]
)
```

8. Results

- The AUC score of the validation dataset after the first epoch is 0.82, 0.85 after the second epoch.

- The model probably did not overfit because the training loss and validation loss were steadily decreasing.

9. Conclusions and Future Scope

Deep learning models are the game changer for detecting the Gravitational-wave signals buried in the noise. The results after applying Q-transformation to the time series and passing to a simple CNN architecture were truly astonishing. But, remember that these are simulated signals. No matter how much has been taken to make them as real as possible, the real-time data consists of more complications than the simulated ones. There is a lot of work to do on this topic. Ensemble models can be used to increase the accuracy of the predictions. Noise can be identified and further classified into various groups which would give more info about the noise which can be used to develop better detectors.

10. References

[1]*Academics.wellesley.edu*, 2021. [Online]. Available: http://academics.wellesley.edu/Physics/brown/pubs/effalgV92P2698-P2701.pdf.

[2]"CNN Architectures : VGG, ResNet, Inception + TL", *Kaggle.com*, 2021. [Online]. Available: https://www.kaggle.com/shivamb/cnn-architectures-vgg-resnet-inception-tl.

[3]"G2Net Gravitational Wave Detection | Kaggle", *Kaggle.com*, 2021. [Online]. Available: https://www.kaggle.com/c/g2net-gravitational-wave-detection/overview.

[4]"ShieldSquare Captcha", *Enhancing gravitational-wave science with machine learning*, 2021. [Online]. Available: https://iopscience.iop.org/article/10.1088/2632-2153/abb93a/pdf.

[5]"Everything you can do with a time series", *Kaggle.com*, 2021. [Online]. Available: https://www.kaggle.com/thebrownviking20/everything-you-can-do-with-a-time-series/notebook#Some-important-things.

[6]"A detailed example of data generators with Keras", *Stanford.edu*, 2021. [Online]. Available: https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.