Data Science in Practice - DSE315

Final Project Report

By ~ Anushka Shreyam 20050

Analysis and Prediction of Gravitational Waves Signals from Binary Black Hole

1. Introduction

1.1 Problem Description:

The aim of this project is to detect GW signals from the mergers of binary black holes. Specifically, I built this model to analyze synthetic GW time-series data from a network of Earth-based detectors (LIGO Hanford, LIGO Livingston and Virgo). The Data was simulated with a sampling rate of 2048 Hz. Each of the time-series originated by the corresponding detector comprises a channel (three in total). In the context of this notebook, data has been already standardized (with training set mean and standard deviation), transposed (to ease channels last format) and saved to TensorFlow Records format. Such data can be found in the provided link, Training Dataset and Test Dataset. Since it is a classification task, the output is the black hole merger occurrence probability.

1.2 Model Implementation:

After many iterations, the model ended up being a 2D Convolutional Neural Network (CNN) preceded by a series of time-series processing techniques. Such a model contains a series of building blocks several of them presented as trainable Tensorflow Keras layers. These are described as follows:

Tukey Window (trainable/non-trainable): Introduces a tapering effect that forces the signal amplitude to decay until having zero values at the ends. It is applied to avoid artifacts stemming from discontinuities when taking Fourier transforms or similar.

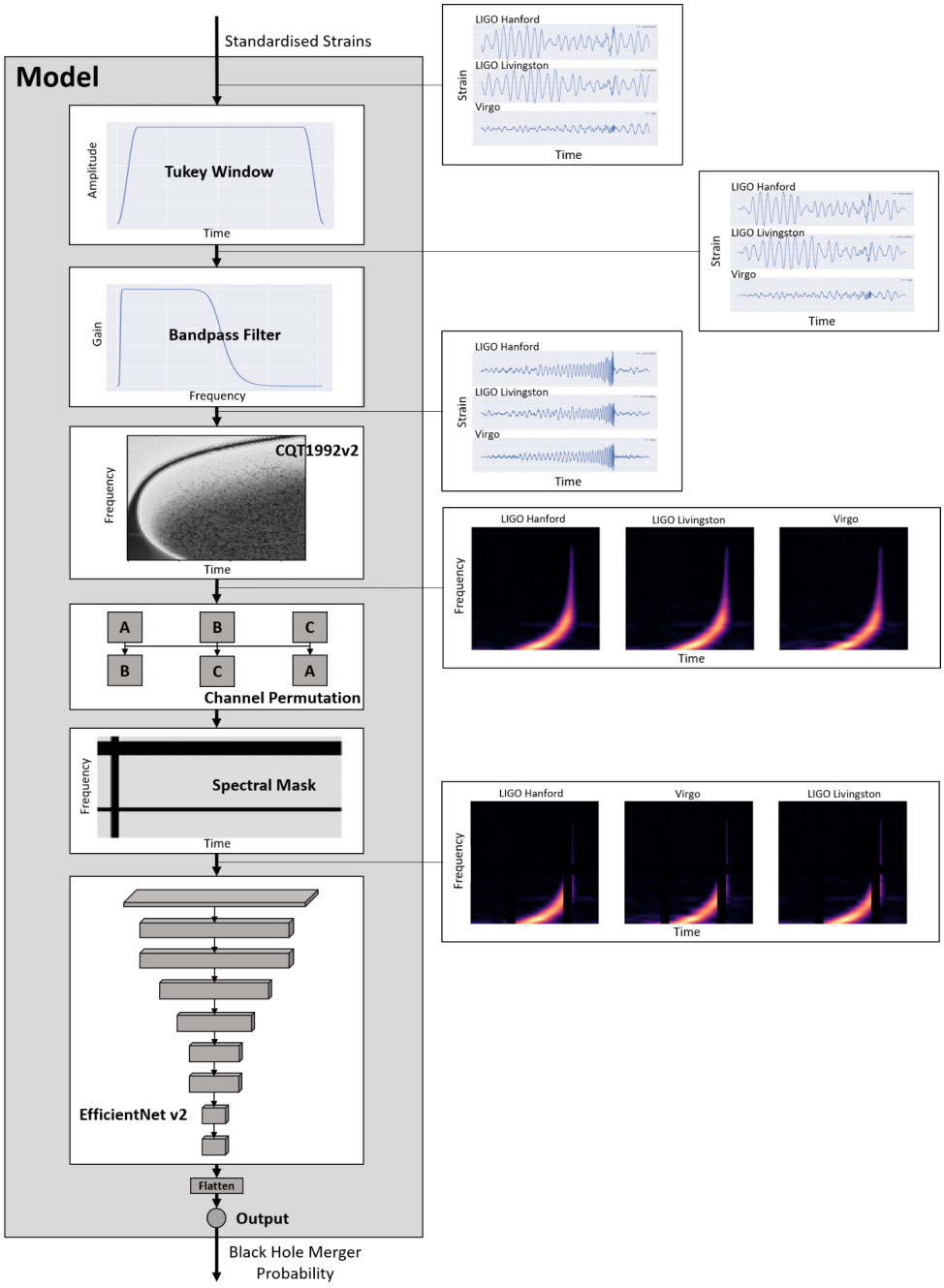
Bandpass Filter (trainable/non-trainable): Applies a filter with the frequency response of a Butterworth filter. The idea is to filter out or attenuate frequencies that have nothing to do with the merger.

Constant-Q Transform (trainable/non-trainable): Transforms the time-domain signal into the time-frequency domain. In other words, it converts a time-domain signal to a spectrogram. Particularly, the PyTorch CQT1992v2 implementation from nnAudio has been taken as reference. It has been re-implemented in TensorFlow for being one of the most cost-effective solutions offered by the library. As an additional functionality, the layer keeps track of maximum spectrogram magnitudes and normalizes its values to a preset range for the sake of stability. The output spectrogram is later resized with bilinear interpolation to adapt it to the downstream CNN recommended input sizes.

Channel Permutation (non-trainable): Randomly decides whether to apply a stochastic permutation of the channels. The aim is to make the prediction a bit less independent of the detector it comes from, presumably acting as a regularization layer.

Spectral Masking (non-trainable): Randomly decides whether to stochastically mask certain time or frequency bands. Similar to the permutation, the idea is for this layer to act as a regularizing operation.

Convolutional Neural Network (trainable/non-trainable): Used as backbone to extract the relevant features to be ingested by a single fully-connected neuron with sigmoid activation (after flattening). Given its performance-complexity trade-off in the ImageNet dataset, the EfficientNet family from AutoML was selected as a more than appropriate model for this purpose.



2. Configuration:

2.1 Environment Configuration:

Before starting with implementation-specific details, let's configure:

- Make sure the Colab environment type is set to TPU going to Runtime → Change runtime type → TPU
- Mount your Google Drive to save any output model after the execution.
- Install any library that might be missing from the defaults.

2.2 Code Configuration:

```
class Config:
    N_SAMPLES, N_DETECT = 4096, 3
    FROM_TFR = False
    MODEL_TRAIN = True
    MODEL_SAVE_NAME = "Model_Ref.h5"
    MODEL_PRELOAD = False
    MODEL_PRELOAD_NAME = "Model_Ref.h5"
    HISTORY_NAME = "history_train.csv"

    MODEL_PATH = Path("models")
    CKPT_PATH = Path("checkpoints")

    SPLIT = 0.98
    SEED_SPLIT = 21
    BATCH_SIZE = 128
    BATCH_SIZE_TEST = 32
    EPOCHS = 1
    LEARNING_RATE = 0.0001
    MODEL_PREDICT = True
    PREDICTIONS_NAME = "submission.csv"
    TUKEY_SHAPE = 0.25
    TRAINABLE_TUKEY = False

    DEGREE_FILT = 6
    F_BAND_FILT = (20., 500.)
    TRAINABLE_FILT = True

    SAMPLE_RATE = 2048
    F_BAND_SPEC = (20., 500.)
    HOP_LENGTH = 64
    BINS_PER_OCTAVE = 12
    WINDOW_CQT = "hann"
    TRAINABLE_CQT = False

    IMAGE_SIZE = 260

    P_PERM = 0.1

    P_MASK = 0.1
    N_MAX_MASK = 2
    W_MASK = (0, IMAGE_SIZE // 6)

    MODEL_ID = "efficientnetv2-b2"
    MODEL_ID_WEIGHTS = MODEL_ID

### Plotting ##################################################################
    PLOT_EXAMPLE = True
    PLOT_TEST = False
```

By the object oriented speciality of python, further necessary classes will be formed:

- **Acceleration(object)** : General Hardware Acceleration Class
- **GeneralUtilities(object)** : General Utilities Class
- **PlottingUtilities(object)** : Plotting Utilities Class
- **TFRDatasetCreator(object)** : Class to aid in the creation of tensorflow records datasets
- **DatasetGeneratorTF(object)** : Class to aid in the creation of dataset pipelines using tensorflow.
- **TukeyWinLayer(tf.keras.layers.Layer)** : Layer that applies a Tukey window function to an input time series, where the possibility of training the shape parameter is given. Not usable with TPU.
- **WindowingLayer(tf.keras.layers.Layer)** : Layer that applies a window function to an input time series.
- **BandpassLayer(tf.keras.layers.Layer)** : Layer that applies a bandpass Butterworth filter in the frequency domain, where the possibility of training frequency from the filter is given.
- **PermuteChannel(tf.keras.layers.Layer)** : Layer that randomly permutes the channels from data to avoid overfitting in the context of G2Net.
- **UtilitiesAug(object)** : Utilities class for augmentations/regularisations.
- **SpectralMask(tf.keras.layers.Layer)** : Layer that applies spectral masks to an input spectrogram. Not usable with TPU.
- **TimeMask(tf.keras.layers.Layer)** : Layer that applies a single time mask to an input spectrogram.
- **FreqMask(tf.keras.layers.Layer)** : Layer that applies a single frequency mask to an input spectrogram.
- **UtilitiesCQT(object)** : Class with local auxiliary function for spectrograms.
- **CQTLayer(tf.keras.layers.Layer)** : nn-audio implementation and extended to bind the output spectrogram to the input of an image based model.

3. Model Training/ Model Summary :

```
Instructions for updating:
Restoring a name-based tf.train.Saver checkpoint using the object-based restore API. This mode uses global names to match variables,
Model: "model"
```

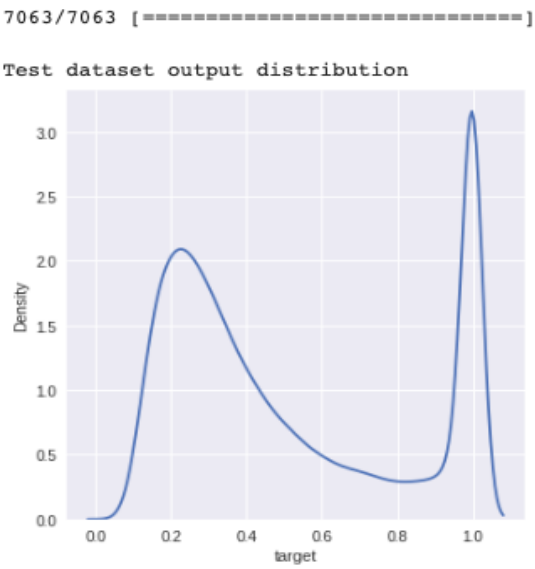
Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	[(None, 4096, 3)]	0	
window (WindowingLayer)	(None, 4096, 3)	4096	input[0][0]
bandpass (BandpassLayer)	(None, 4096, 3)	2049	window[0][0]
cqt (CQTLayer)	(None, 56, 64, 3)	229378	bandpass[0][0]
resize (Resizing)	(None, 260, 260, 3)	0	cqt[0][0]
permute (PermuteChannel)	(None, 260, 260, 3)	0	resize[0][0]
mask_t (TimeMask)	(None, 260, 260, 3)	0	permute[0][0] mask_t[0][0]
mask_f (FreqMask)	(None, 260, 260, 3)	0	mask_t[1][0] mask_f[0][0]
efficientnetv2-b2 (EffNetV2Mode	(None, 1408)	8769374	mask_f[1][0]
flatten (Flatten)	(None, 1408)	0	efficientnetv2-b2[1][0]
dense (Dense)	(None, 1)	1409	flatten[0][0]

```
Total params: 9,006,306
Trainable params: 8,690,544
Non-trainable params: 315,762
```

4. Predicting with the Model :

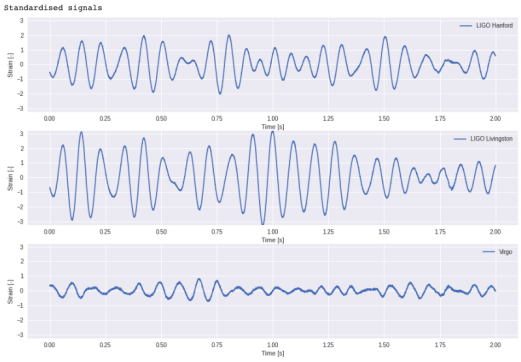
4.1 Test Dataset Output Distribution:

```
7063/7063 [==============================] - 167s 23ms/step
```

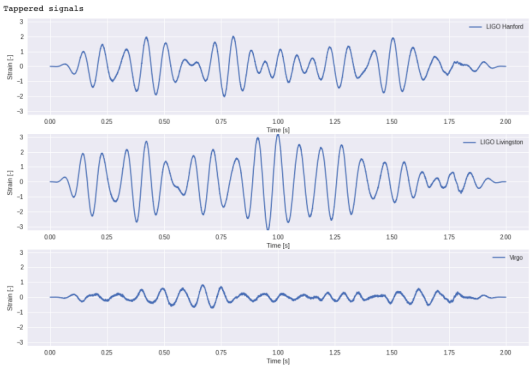


Now let's try to break down the model into its constituent parts and plot the intermediate data for a single example. You might see that, depending on the example, visually identifying a merger chirp becomes difficult or even impossible mainly due other low and high frequency sources of noise (in this case simulated).

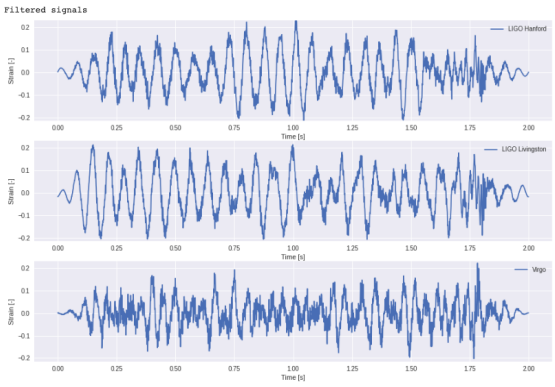
4.2 Standardized Signals :



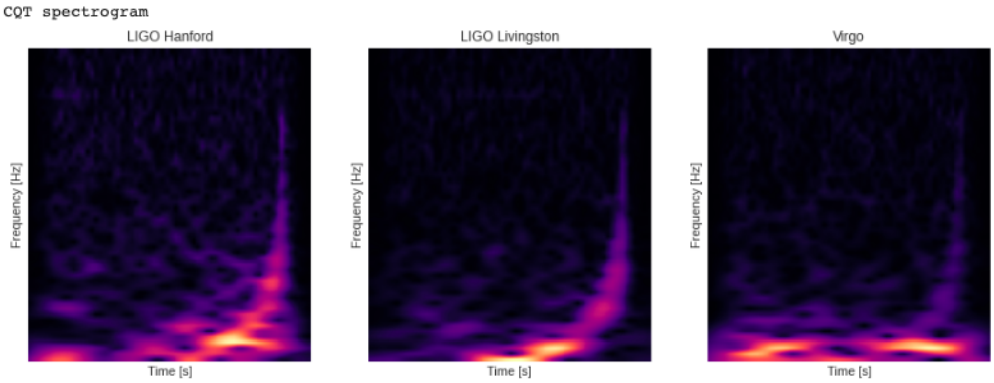
4.3 Tapered Signals:



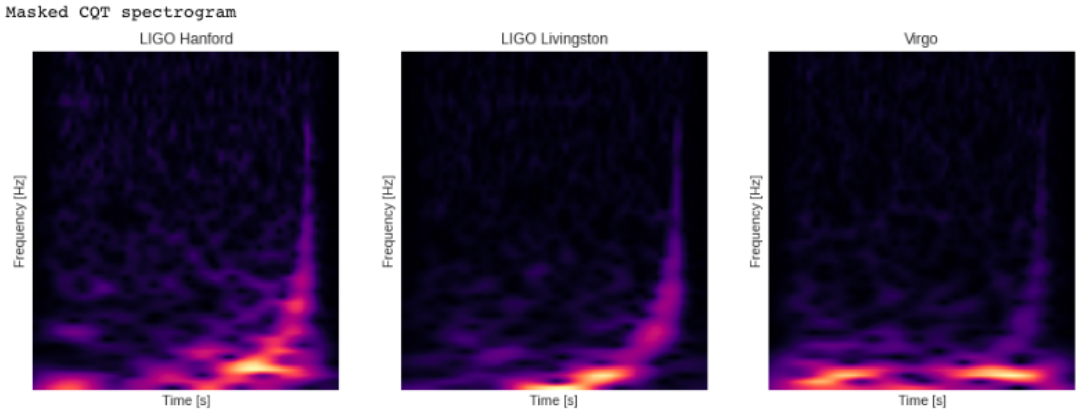
4.4 Filtered Signals:



4.5 C-QT Spectrogram:



4.6 Masked CQT spectrogram:



5. Conclusion :

This model basically analyze simulated GW time-series data from a network of Earth-based detectors.The best G2Net single model was obtained with trainable Tukey window and bandpass layers, a non-trainable C-QT, a resize size of 384, an EfficientNet v2 S backbone and training in a Tesla V100 (16GB) GPU with a batch size of 32 (learning rates in the range 0.0003 to 0.00001). The score was boosted with an averaging ensemble of several models with different settings.

References:

- https://www.kaggle.com/c/g2net-gravitational-wave-detection
- https://github.com/AnushkaShreyam/GW-Signals-from-Binary-Black-Hole
- https://github.com/tensorflow/tensorflow