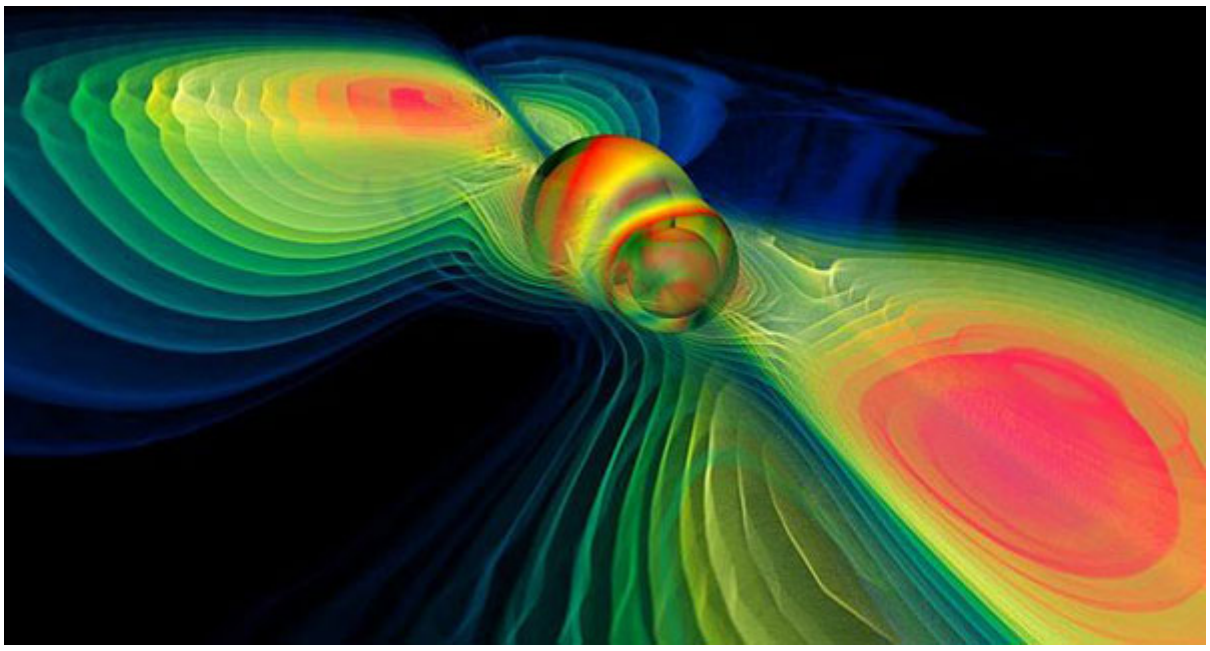
Gravitational Waves

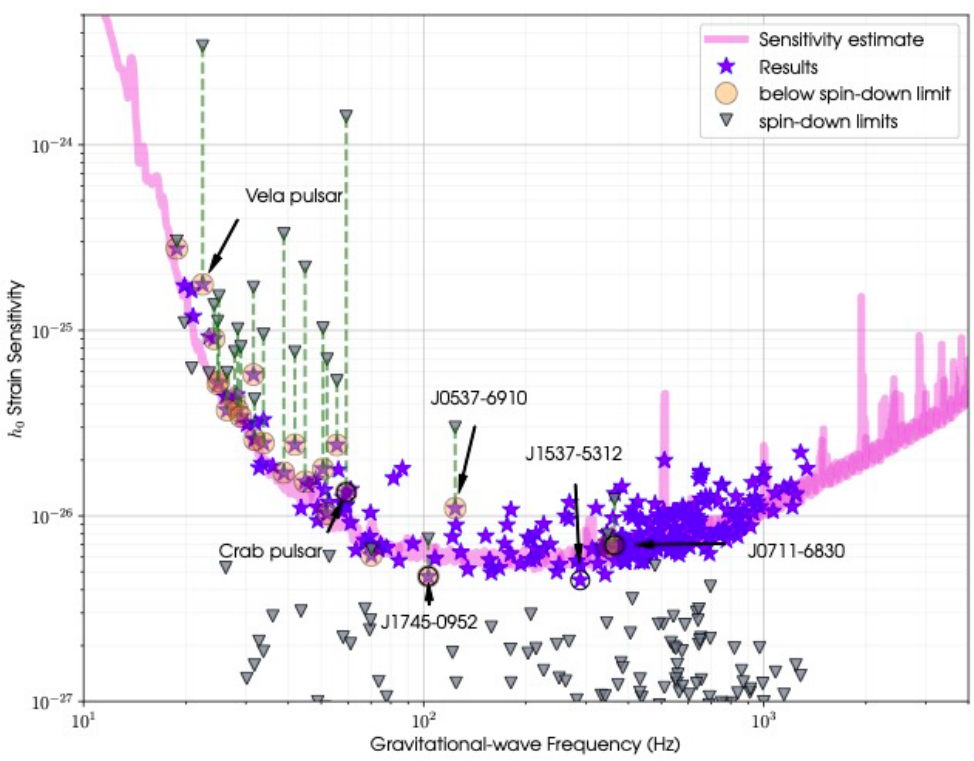
Gravitational waves are disturbances or ripples in the curvature of spacetime, generated by accelerated masses, that propagate as waves outward from their source at the speed of light. They were first proposed by Oliver Heaviside in 1893 and then later by Henri Poincaré in 1905 and subsequently predicted in 1916 by Albert Einstein on the basis of his general theory of relativity. Later he refused to accept gravitational waves. Gravitational waves transport energy as gravitational radiation, a form of radiant energy similar to electromagnetic radiation.Newton's law of universal gravitation, part of classical mechanics, does not provide for their existence, since that law is predicated on the assumption that physical interactions propagate instantaneously (at infinite speed) – showing one of the ways the methods of Newtonian physics are unable to explain phenomena associated with relativity.

The first indirect evidence for the existence of gravitational waves came in 1974 from the observed orbital decay of the Hulse–Taylor binary pulsar, which matched the decay predicted by general relativity as energy is lost to gravitational radiation. In 1993, Russell A. Hulse and Joseph Hooton Taylor Jr. received the Nobel Prize in Physics for this discovery. The first direct observation of gravitational waves was not made until 2015, when a signal generated by the merger of two black holes was received by the LIGO gravitational wave detectors in Livingston, Louisiana, and in Hanford, Washington. The 2017 Nobel Prize in Physics was subsequently awarded to Rainer Weiss, Kip Thorne and Barry Barish for their role in the direct detection of gravitational waves.



Challenge Context

When scientists detected the first class of gravitational waves in 2015, they expected the discoveries to continue. There are four classes, yet at present only signals from merging black holes and neutron stars have been detected. Among those remaining are continuous gravitational-wave signals. These are weak yet long-lasting signals emitted by rapidly-spinning neutron stars. Imagine the mass of our Sun but condensed into a ball the size of a city and spinning over 1,000 times a second. The extreme compactness of these stars, composed of the densest material in the universe, could allow continuous waves to be emitted and then detected on Earth. There are potentially many continuous signals from neutron stars in our own galaxy and the current challenge for scientists is to make the first detection, and hopefully data science can help with this mission.



This image, taken from a 2021 paper by the LIGO-Virgo-KAGRA collaboration, shows the maximum amplitude of a continuous wave any of these neutron stars could emit without being found by the search analyses. Circled stars show results constraining the physical properties of specific neutron stars. Traditional approaches to detecting these weak and hard-to-find continuous signals are based on matched-filtering variants. Scientists create a bank of possible signal waveform templates and ask how correlated each waveform is with the measured noisy data. High correlation is

consistent with the presence of a signal similar to that waveform. Due to the long duration of these signals, banks could easily contain hundreds of quintillions of templates; yet, with so many possible waveforms, scientists don't have the computational power to use the approach without making approximations that weaken the sensitivity to the signals.

G2Net is a network of Gravitational Wave, Geophysics and Machine Learning. Via an Action from COST (European Cooperation in Science and Technology), a funding agency for research and innovation networks, G2Net aims to create a broad network of scientists. From four different areas of expertise, namely GW physics, Geophysics, Computing Science and Robotics, these scientists have agreed on a common goal of tackling challenges in data analysis and noise characterization for GW detectors.

Installing and importing necessary dependancies

In [1]:
```
!pip install rirориro
!pip install visualkeras
!pip install git+https://github.com/PyFstat/PyFstat@python37
```

Show hidden output

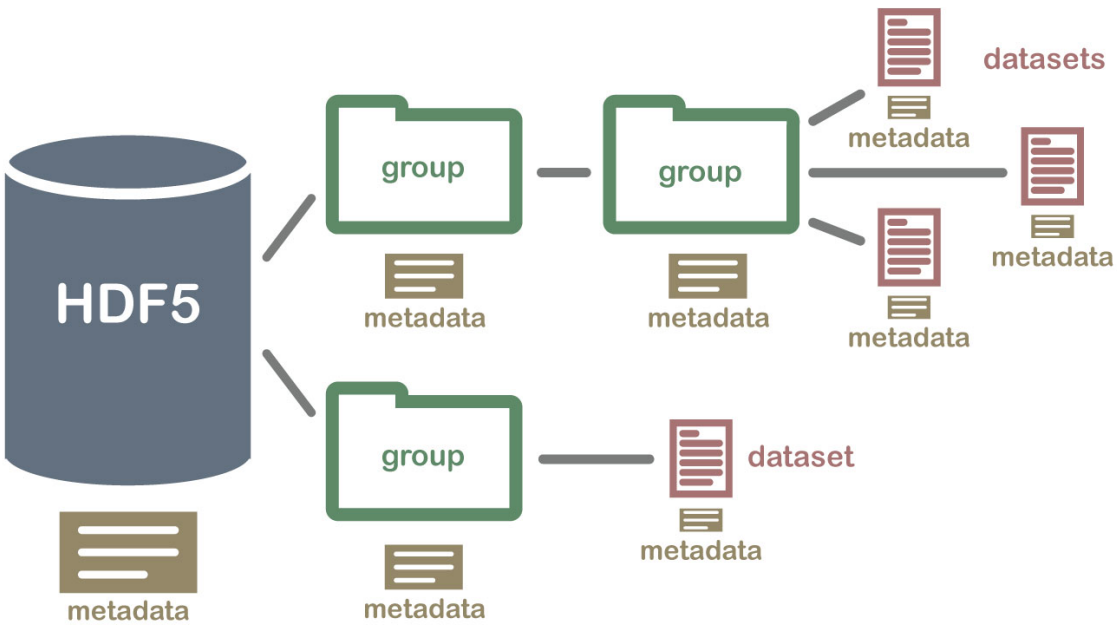
In [2]:
```
import os
import h5py
import gc
import glob
import math
import random
import warnings
import pyfstat
import librosa
import librosa.display
import numpy as np
import pandas as pd
import seaborn as sns
import visualkeras
import rirориro.inspiralfuns as ins
import rirориro.mergerfirstfuns as me1
import rirориro.matchingfuns as mat
import rirориro.mergersecondfuns as me2
import tensorflow as tf
from scipy.signal import istft
from tensorflow.keras import regularizers
from tensorflow.keras import layers
import tensorflow_addons as tfa

from pathlib import Path
from scipy import stats
from tqdm.notebook import tqdm
from scipy import signal
import matplotlib.pyplot as plt
from IPython.display import HTML, display
from tensorflow.keras.utils import plot_model
from sklearn.model_selection   import train_test_split

sns.set_theme()
%matplotlib inline
warnings.filterwarnings('ignore')
display(HTML('<style>.font-family:verdana; word-spacing:1.5px;</style>'))
```

In [3]:
```
DATA_PATH = Path('../input/g2net-detecting-continuous-gravitational-waves')
TRAIN_PATH = DATA_PATH/'train'
TEST_PATH = DATA_PATH/'test'
train_example_with_signal_path = TRAIN_PATH/'cc561e4fc.hdf5'
train_example_without_signal_path = TRAIN_PATH/'fb6db0d08.hdf5'
```

HDF files exploration

Hierarchical Data Format (HDF) is a set of file formats (HDF4, HDF5) designed to store and organize large amounts of data. Originally developed at the U.S. National Center for Supercomputing Applications, it is supported by The HDF Group, a non-profit corporation whose mission is to ensure continued development of HDF5 technologies and the continued accessibility of data stored in HDF. In keeping with this goal, the HDF libraries and associated tools are available under a liberal, BSD-like license for general use. HDF is supported by many commercial and non-commercial software platforms and programming languages. The freely available HDF distribution consists of the library, command-line utilities, test suite source, Java interface, and the Java-based HDF Viewer (HDFView). The current version, HDF5, differs significantly in design and API from the major legacy version HDF4.

HDF5



The HDF5 format is designed to address some of the limitations of the HDF4 library, and to address current and anticipated requirements of modern systems and applications. In 2002 it won an R&D 100 Award. HDF5 simplifies the file structure to include only two major types of object: HDF Structure Example

- Datasets, which are typed multidimensional arrays
- Groups, which are container structures that can hold datasets and other groups

Our dataset structures

- **ID** is the top group of the HDF5 file and links the datapoint to it's label in the train_labels csv (group)
- **frequency_Hz** contains the range frequencies measured by the dectors (dataset)
- **H1** contains the data for the LIGO Hanford dececter (group)

```
  - SFTs is the Short-time Fourier Transforms amplitudes for each timestamp at each frequency (dataset)
- timestamps contains the timestamps for the measurement (dataset)
```

- **L1** contains the data for the LIGO Livingston dececter (group)

```
  - SFTs is the Short-time Fourier Transforms amplitudes for each timestamp at each frequency (dataset)
- timestamps contains the timestamps for the measurement (dataset)
```

For good EDA check this great work of **Edward Crookenden** (https://www.kaggle.com/code/edwardcrookenden/g2net-getting-started-eda)

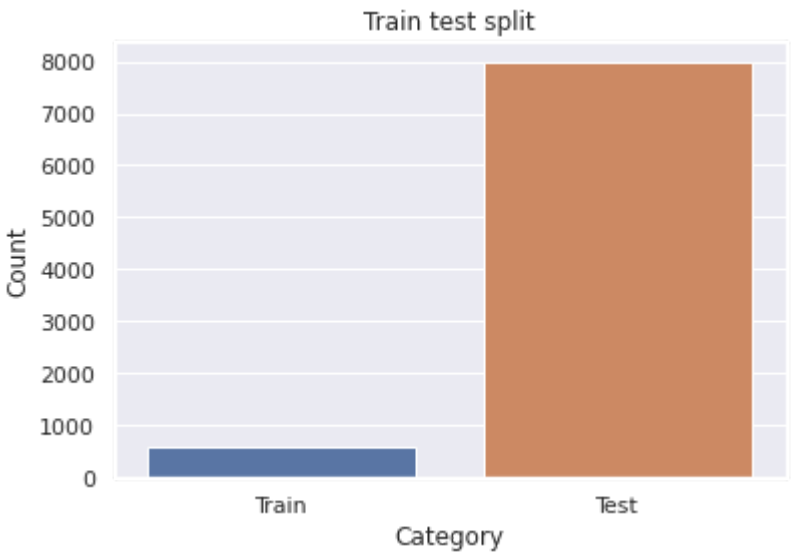
Loading meta data

In [4]:
```
labels_df = pd.read_csv(DATA_PATH/'train_labels.csv')
labels_df.head()
```

Out[4]:

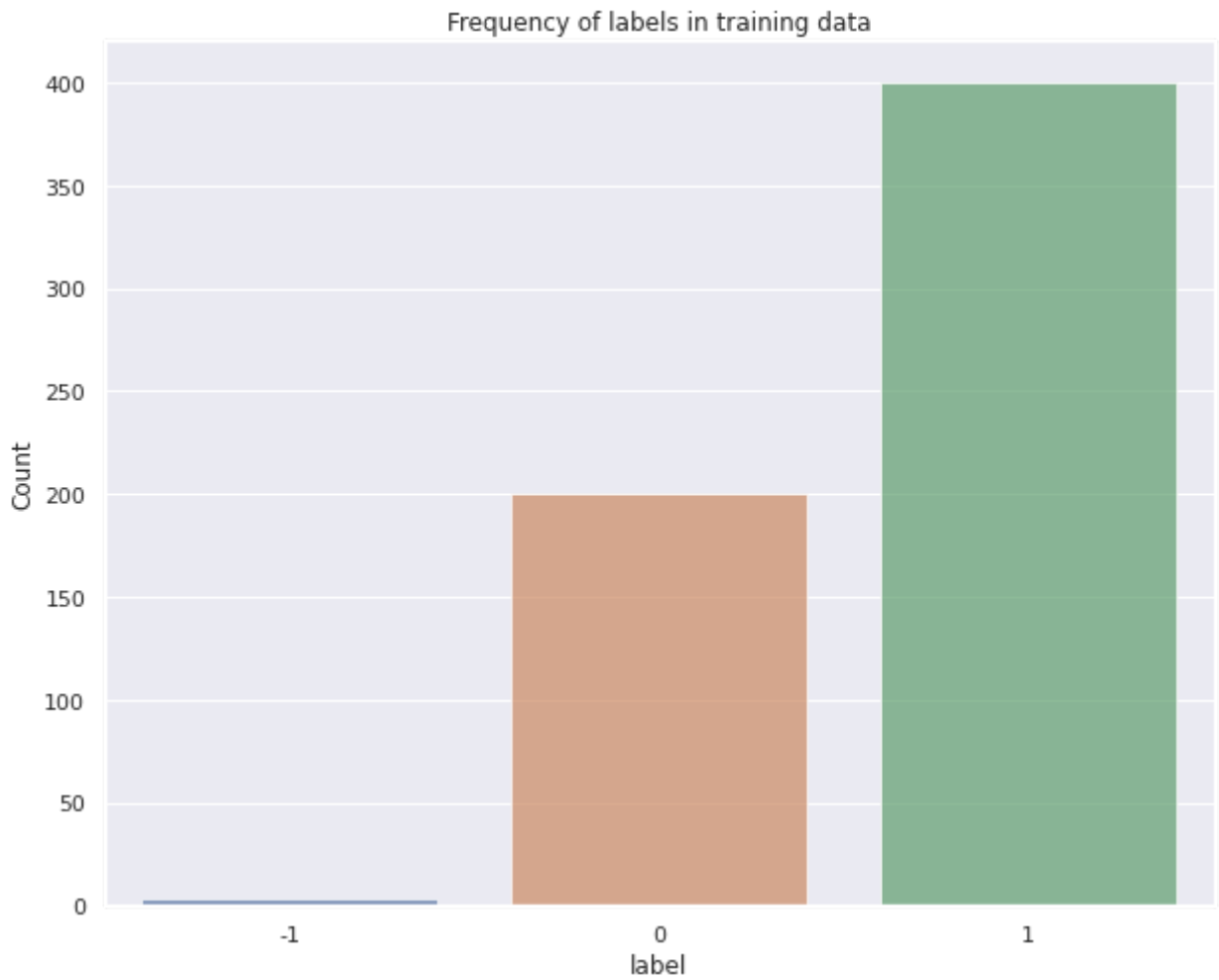
	id	target
0	001121a05	1
1	004f23b2d	1
2	00a6db666	1
3	00f36a6ac	1
4	010a387db	1

In [5]:
```
plt.figure(figsize=(6,4))
sns.barplot(['Train', 'Test'], [len(os.listdir(TRAIN_PATH)), len(os.listdir(TEST_PATH))]);
plt.title(f'Train test split', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('Category', fontsize=12)
plt.show()
```



This is unusual as we most of the time see data splits as 80:20 but in our case it is 1:16 which indicates that the competition creators are encouraging participants to generate their own data.

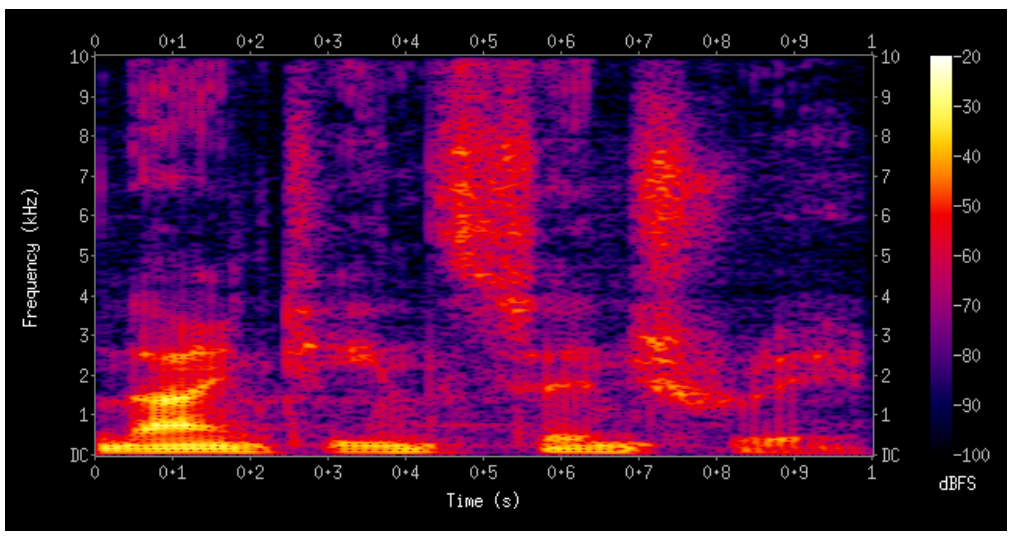
In [6]:
```
label_count  = labels_df['target'].value_counts()
plt.figure(figsize=(10,8))
sns.barplot(label_count.index, label_count.values, alpha=0.7)
plt.title(f'Frequency of labels in training data', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.xlabel('label', fontsize=12)
plt.show()
```



The target labels; 1 if the data contains the presence of a gravitational wave, 0 otherwise. (Please note the presence of a small number of files labeled -1. Physicists are currently unable to determine the status of these files.)

Spectogram analysis

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. When applied to an audio signal, spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data are represented in a 3D plot they may be called waterfall displays.



Spectrograms are used extensively in the fields of music, linguistics, sonar, radar, speech processing, seismology, and others. Spectrograms of audio can be used to identify spoken words phonetically, and to analyse the various calls of animals. A spectrogram can be generated by an optical spectrometer, a bank of band-pass filters, by Fourier transform or by a wavelet transform (in which case it is also known as a scaleogram or scalogram). Scaleograms from the DWT and CWT for an audio sample A spectrogram is usually depicted as a heat map, i.e., as an image with the intensity shown by varying the colour or brightness.

In [7]:

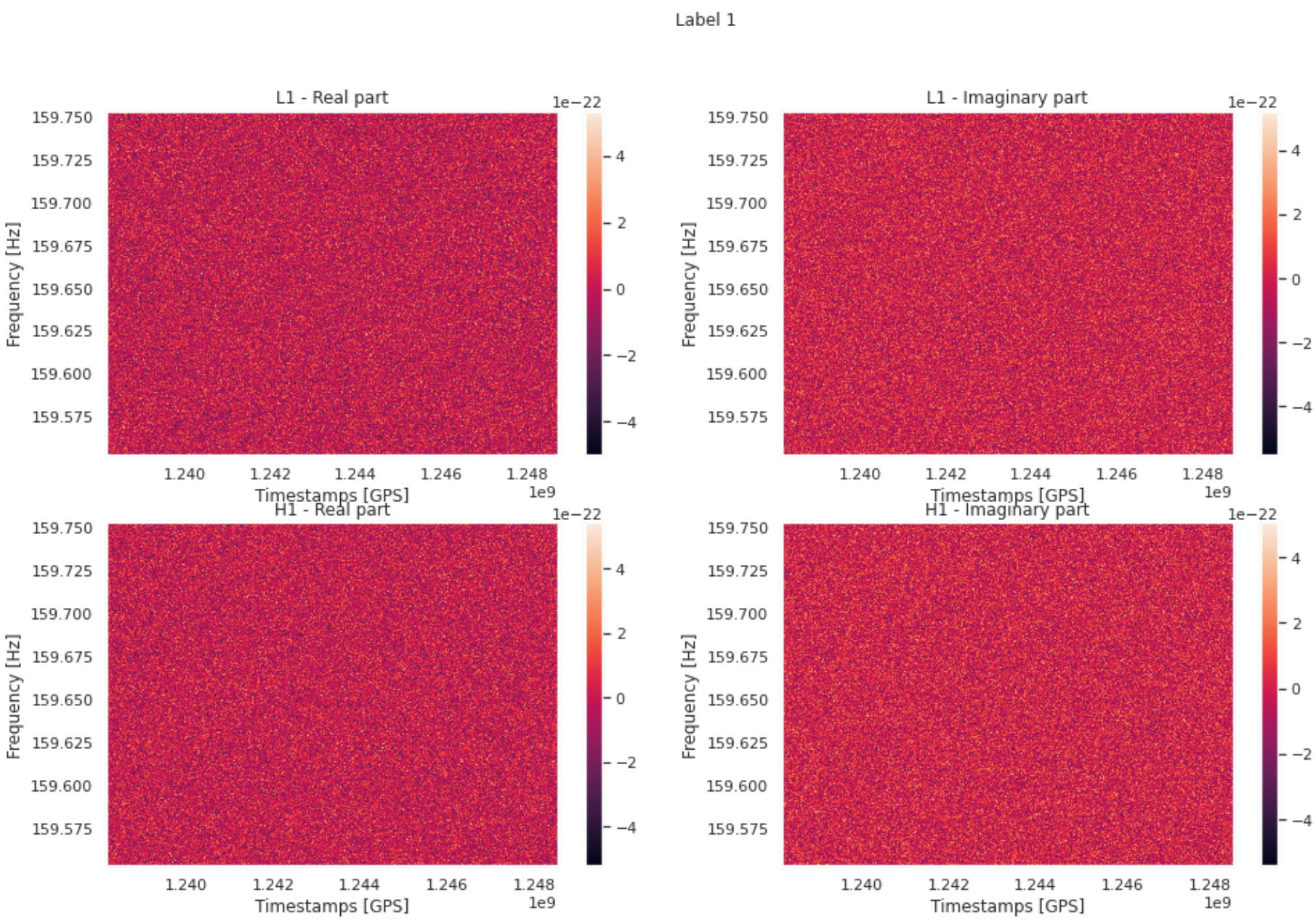
```
def extract_data_from_hdf5(path, labels):
    '''
    Extracts data from hdf5 file and puts it into a dict. It also adds the label
    '''

    data = {}

    with h5py.File(path, "r") as f:

        ID_key = list(f.keys())[0]

        # Retrieve the frequency data
        data['freq'] = np.array(f[ID_key]['frequency_Hz'])

        # Retrieve the Livingston decector data
        data['L1_SFTs_amplitudes'] = np.array(f[ID_key]['L1']['SFTs'])
        data['L1_ts'] = np.array(f[ID_key]['L1']['timestamps_GPS'])

        # Retrieve the Hanford decector data
        data['H1_SFTs_amplitudes'] = np.array(f[ID_key]['H1']['SFTs'])
        data['H1_ts'] = np.array(f[ID_key]['H1']['timestamps_GPS'])

        # Get label from training labels if in training set
        data['label'] = labels.loc[labels.id==ID_key].target.item()

    return data
```

In [8]:

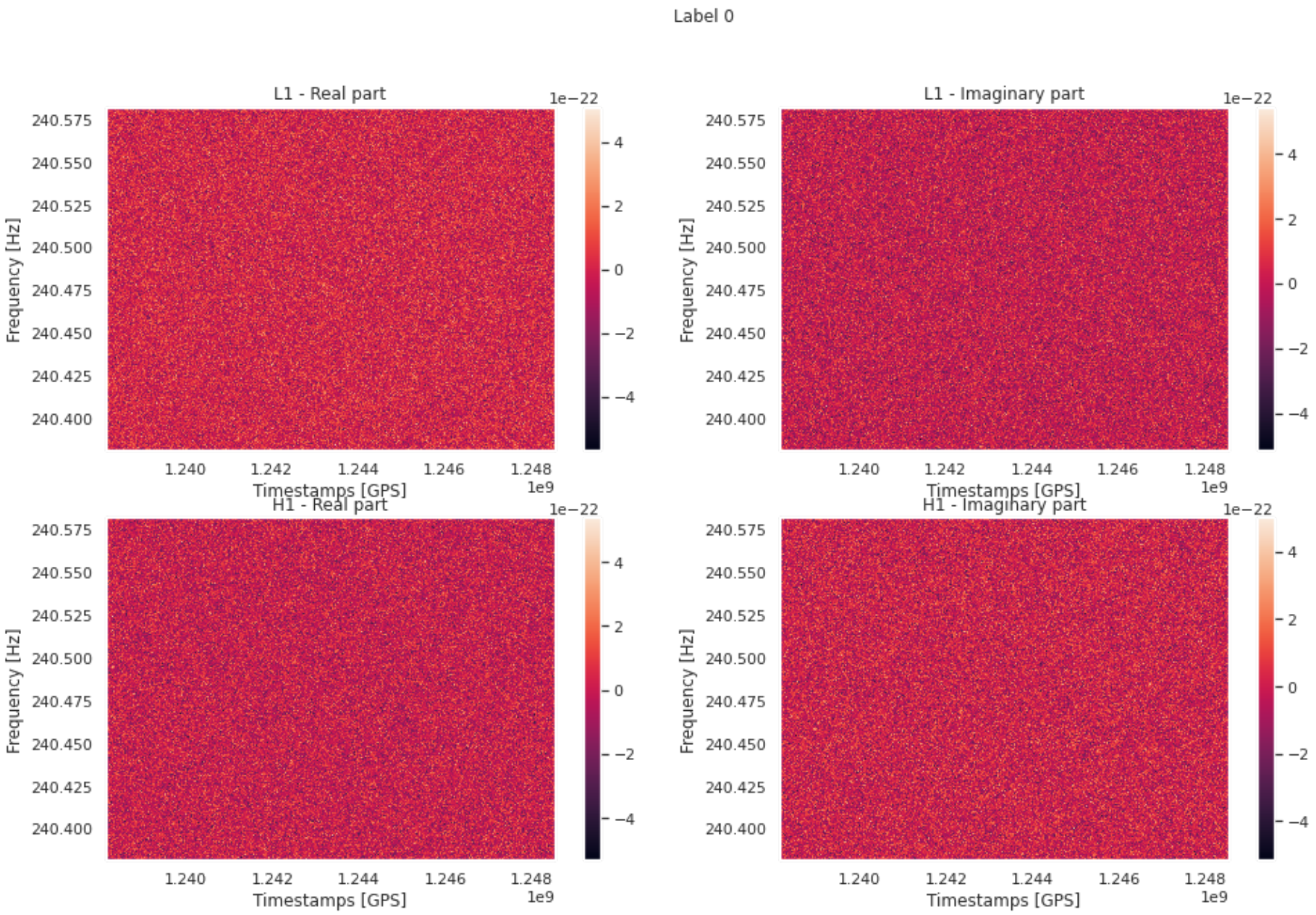
```
def extract_data_from_hdf5_reduced(path, labels):
    '''
    Extracts data from hdf5 file and puts it into a dict. It also adds the label
    '''

    data = {}

    with h5py.File(path, "r") as f:

        ID_key = list(f.keys())[0]

        # Retrieve the frequency data
        data['id'] = ID_key
        data['freq'] = np.array(f[ID_key]['frequency_Hz'])

        # Retrieve the Livingston decector data
        data['L1_ts'] = np.array(f[ID_key]['L1']['timestamps_GPS'])

        # Retrieve the Hanford decector data
        data['H1_ts'] = np.array(f[ID_key]['H1']['timestamps_GPS'])

        # Get label from training labels if in training set
        data['label'] = labels.loc[labels.id==ID_key].target.item()

    return data
```

In [9]:
```
def plot_spectograms(data):
    '''
    Shows the real and imaginary amplitudes of the SFTs as spectograms for both detectors
    '''

    fig, ax = plt.subplots(2, 2, figsize=(16, 10))
    fig.suptitle(f"Label {data['label']}")

    for ind, detector in enumerate(['L1', 'H1']):
        ax[ind][0].set(xlabel="Timestamps [GPS]",
                       ylabel="Frequency [Hz]",
                       title=f"{detector} - Real part")
        ax[ind][1].set(xlabel="Timestamps [GPS]",
                       ylabel="Frequency [Hz]",
                       title=f"{detector} - Imaginary part")


        c0 = ax[ind][0].pcolormesh(data[f"{detector}_ts"], data['freq'],
                                   data[f"{detector}_SFTs_amplitudes"].real)
        c1 = ax[ind][1].pcolormesh(data[f"{detector}_ts"], data['freq'],
                                   data[f"{detector}_SFTs_amplitudes"].imag)

        fig.colorbar(c0, ax=ax[ind][0])
        fig.colorbar(c1, ax=ax[ind][1])

    plt.show()
```

In [10]:
```
data = extract_data_from_hdf5(train_example_with_signal_path, labels_df)
plot_spectograms(data)
```



In [11]:
```
data = extract_data_from_hdf5(train_example_without_signal_path, labels_df)
plot_spectograms(data)
```



Time Domain Dataset Preparation

The STFT is invertible, that is, the original signal can be recovered from the transform by the inverse STFT. The most widely accepted way of inverting the STFT is by using the overlap-add (OLA) method, which also allows for modifications to the STFT complex spectrum. This makes for a versatile signal processing method, referred to as the overlap and add with modifications method.

The inverse Fourier transform of X(τ,ω) for τ fixed:

$$x(t) = \frac{1}{w(t-\tau)} \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\tau, \omega) e^{+i\omega t}\, d\omega.$$

In [12]:
```
data_s = extract_data_from_hdf5(train_example_with_signal_path, labels_df)
data_w_s = extract_data_from_hdf5(train_example_without_signal_path, labels_df)
amp1 = data_s['L1_SFTs_amplitudes']
f1 = data_s['freq']
amp2 = data_w_s['L1_SFTs_amplitudes']
f2 = data_w_s['freq']
f_m1 = sum(f1)/len(f1)
f_m2 = sum(f2)/len(f2)
print('Mean of label 1 signal frequency is {}'.format(f_m1))
print('Mean of label 0 signal frequency is {}'.format(f_m2))
```

```
Mean of label 1 signal frequency is 159.65305555555554
Mean of label 0 signal frequency is 240.48194444444454
```

In [13]:
```
_, xrec1 = signal.istft(amp1, f_m1)
_, xrec2 = signal.istft(amp2, f_m2)
xrec1_r = signal.resample(xrec1, 16707)
xrec2_r = signal.resample(xrec2, 16707)
```

In [14]:
```
print('Signal size in frequency domain is {}'.format(amp1[0].shape[0]))
print('Signal size in Time domain is {}'.format(xrec1.shape[0]))
print('Signal size in Time domain(resamples) is {}'.format(xrec1_r.shape[0]))
```

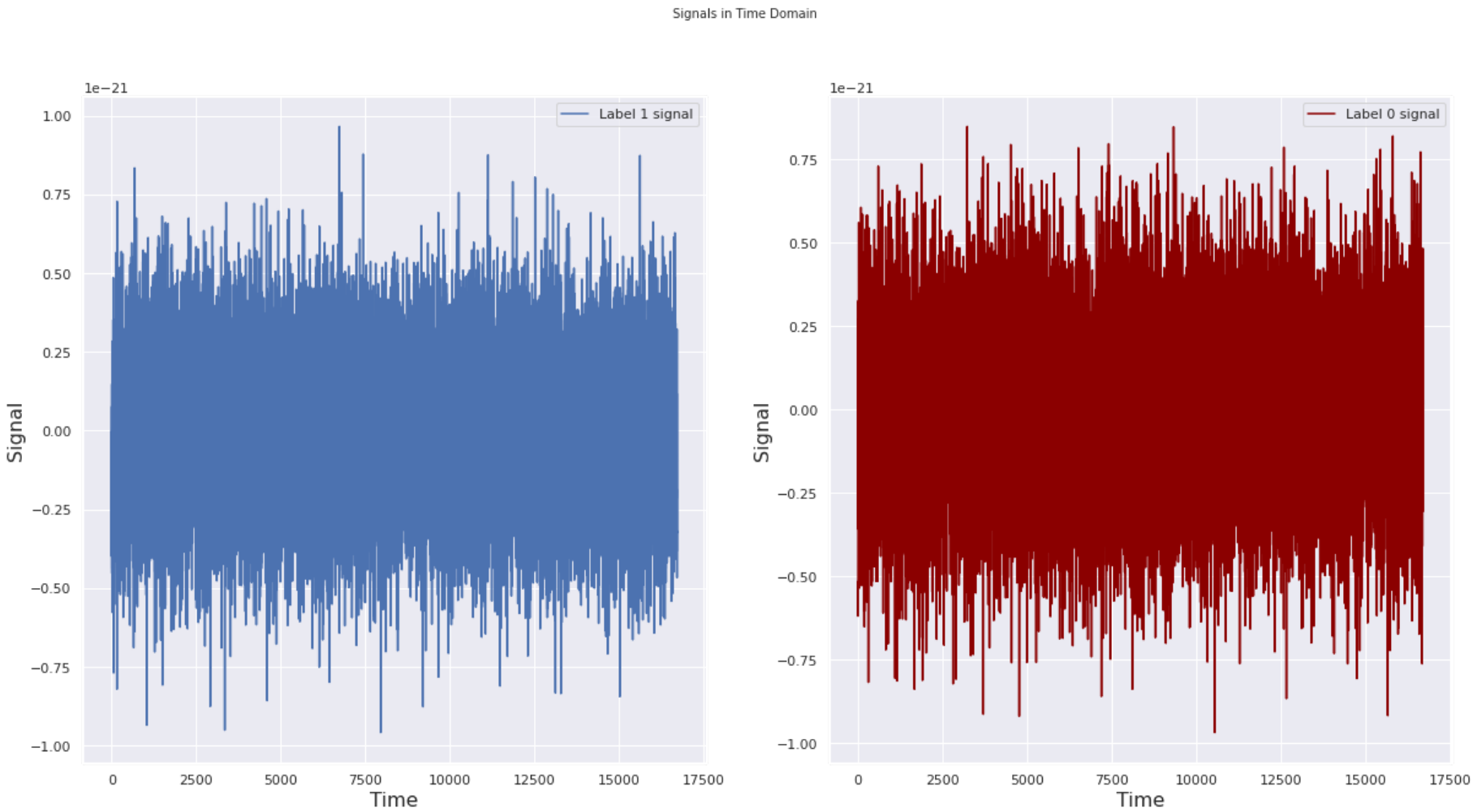
```
Signal size in frequency domain is 4655
Signal size in Time domain is 1670786
Signal size in Time domain(resamples) is 16707
```

As it seems the original time domain sequence size is way too large that's why we will be resampling the signal in order to be able to deal with it during modeling. This will reduce tremendous amount of data informativity but will help the correct architecture to converge and also will be friendly when it comes to computational resources.

In [15]:
```
plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)
plt.suptitle('Signals in Time Domain', fontsize=10)
plt.ylabel('Signal', fontsize=16)
plt.xlabel('Time', fontsize=16)

plt.plot(xrec1_r, label='Label 1 signal')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Signal', fontsize=16)
plt.xlabel('Time', fontsize=16)

plt.plot(xrec2_r, label='Label 0 signal', c = 'darkred')

plt.legend(loc='upper right')
```

Out[15]:

```
<matplotlib.legend.Legend at 0x7f3d602c1390>
```



In [16]:
```
%%time
time_df = pd.DataFrame()
for p in tqdm(os.listdir(TRAIN_PATH), total=len(os.listdir(TRAIN_PATH))):
    final_data = dict()
    data = extract_data_from_hdf5(DATA_PATH/'train'/p, labels_df)
    amp1 = data['L1_SFTs_amplitudes']
    amp2 = data['H1_SFTs_amplitudes']
    f = data['freq']
    label = data['label']
    f_m = sum(f)/len(f)
    _, xrec1 = signal.istft(amp1, f_m)
    _, xrec2 = signal.istft(amp2, f_m)
    xrec1_r = signal.resample(xrec1, 16707)
    xrec2_r = signal.resample(xrec2, 16707)
    final_data['L1_resampled_time'] = xrec1_r
    final_data['H1_resampled_time'] = xrec2_r
    final_data['label'] = label
    time_df = time_df.append(final_data, ignore_index=True)
```

```
CPU times: user 5min 56s, sys: 12.5 s, total: 6min 9s
Wall time: 8min 7s
```

In [17]:
```
time_df = time_df.sample(frac=1.0)
```

In [18]:
```
time_df.shape
```

Out[18]:
```
(603, 3)
```

In [19]:
```
time_df.head()
```

Out[19]:

	L1_resampled_time	H1_resampled_time	label
157	[-1.8960645e-22, 2.2951528e-22, -1.6093122e-22...	[2.1975002e-23, -1.4147607e-22, 5.31988e-22, 1...	0.0
78	[-1.7919311e-22, -1.4602406e-22, -2.5456822e-2...	[3.6339386e-22, -1.3032497e-22, 2.0978053e-22,...	1.0
7	[1.7573351e-22, 1.8281694e-22, -8.4587397e-22,...	[1.1651898e-22, 6.798967e-24, 2.6429788e-23, 9...	1.0
570	[-1.8871895e-22, -7.741742e-23, 1.3920881e-22,...	[-2.9927686e-22, 3.0806175e-22, -6.354769e-23,...	0.0
376	[4.7447446e-22, -6.9310013e-23, -2.788609e-22,...	[1.3894915e-22, -2.576136e-22, -1.9931798e-22,...	0.0

Dataset Exploration

In [20]:
```
%%time
train_df = pd.DataFrame()
for p in tqdm(os.listdir(TRAIN_PATH), total=len(os.listdir(TRAIN_PATH))):
    data = extract_data_from_hdf5_reduced(DATA_PATH/'train'/p, labels_df)
    train_df = train_df.append(data, ignore_index=True)
```

```
CPU times: user 4.25 s, sys: 2.12 s, total: 6.37 s
Wall time: 15.4 s
```

In [21]:
```
train_df.head()
```

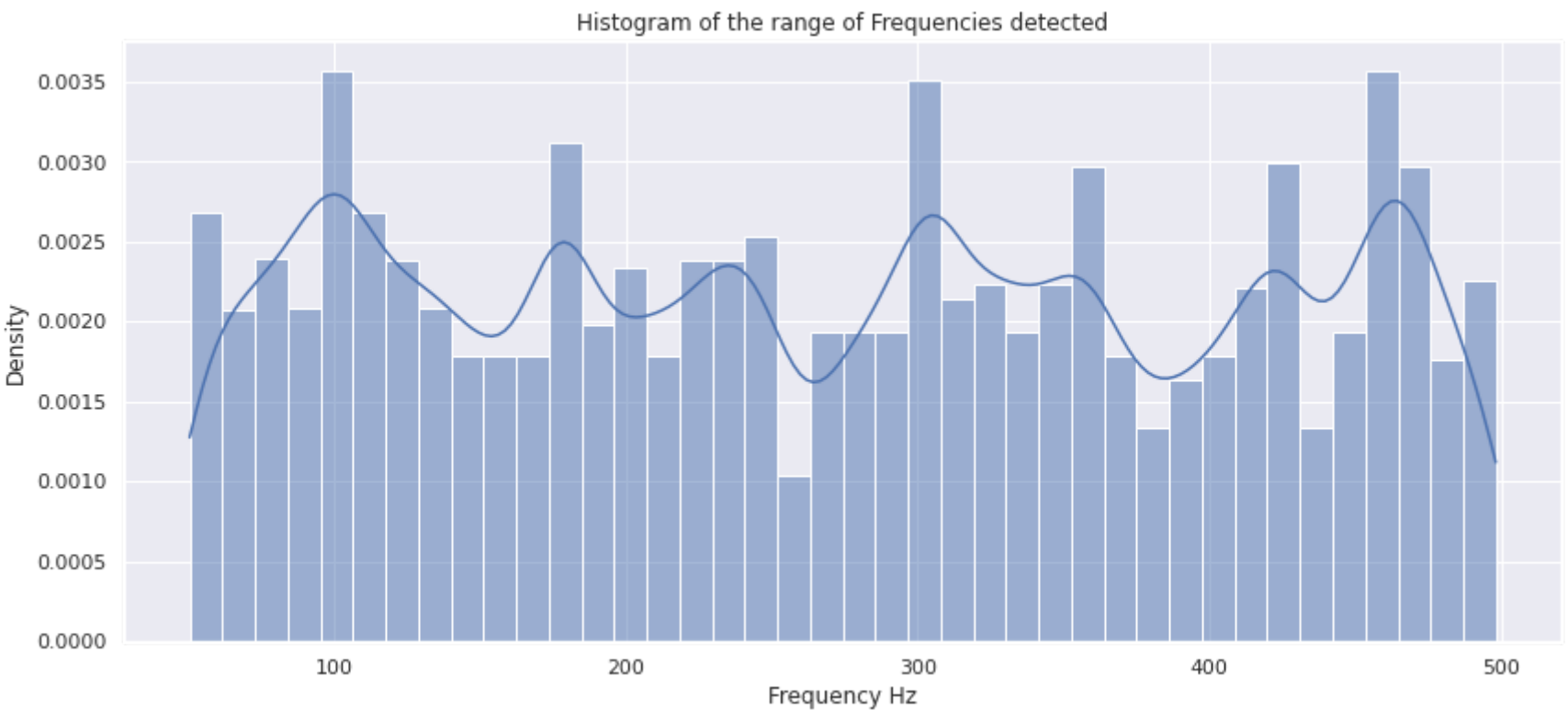
Out[21]:

	id	freq	L1_ts	H1_ts	label
0	0517ef7fe	[99.72777777777777, 99.72833333333332, 99.7288...	[1238178603, 1238180403, 1238182203, 123818400...	[1238170914, 1238175422, 1238177222, 123817902...	1.0
1	09531cde3	[383.4066666666667, 383.40722222222223, 383.40...	[1238171878, 1238173678, 1238175478, 123817727...	[1238166446, 1238168246, 1238170046, 123817184...	1.0
2	067b3fb4b	[415.4405555555556, 415.4411111111111, 415.441...	[1238174562, 1238188464, 1238190264, 123819206...	[1238177971, 1238179771, 1238184917, 123818671...	1.0
3	4f61f078b	[247.44722222222222, 247.4477777777778, 247.44...	[1238171750, 1238173550, 1238175350, 123817715...	[1238168490, 1238170290, 1238172090, 123817389...	1.0
4	7c2e38128	[424.7238888888889, 424.72444444444443, 424.72...	[1238171138, 1238172938, 1238174738, 123817653...	[1238188274, 1238190074, 1238191874, 123819367...	1.0

In [22]:
```
train_df = train_df.loc[train_df['label'] != -1]
```

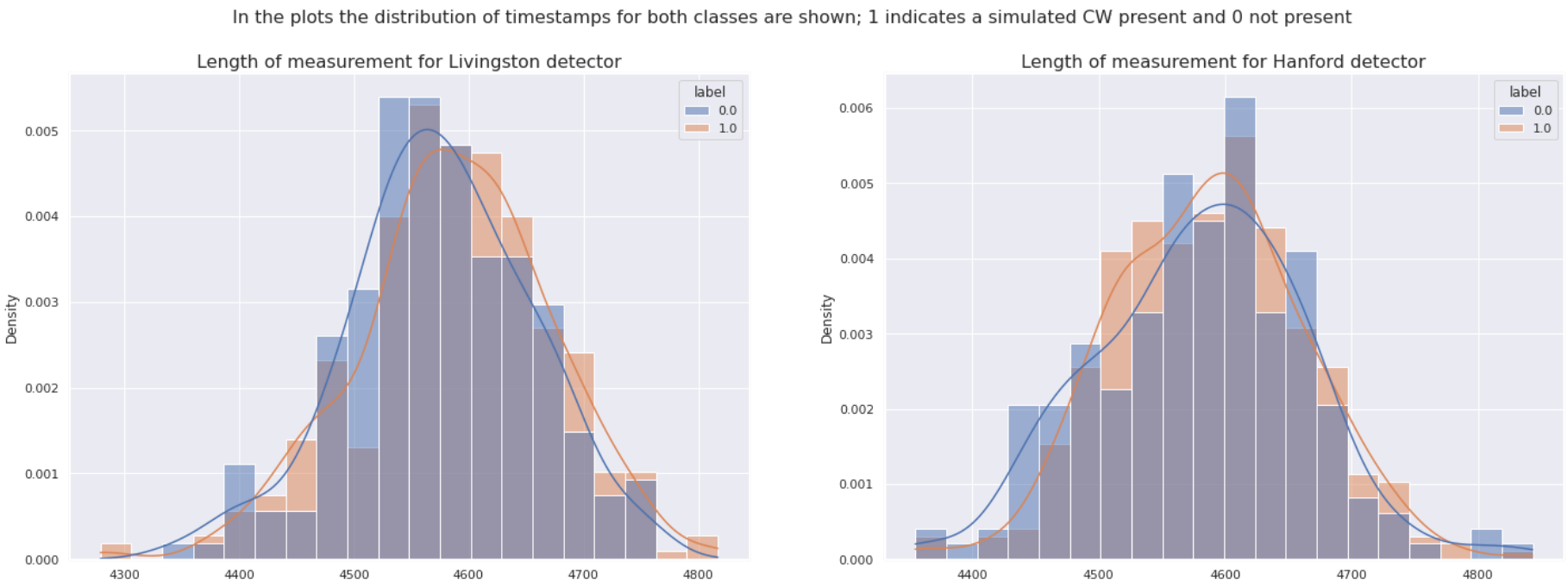
Frequencies Distribution

In [23]:
```
plt.figure(figsize=(14,6))
sns.histplot(x=list(np.hstack(train_df['freq'])), stat="density", common_norm=False, bins=40, kde=True)
plt.title('Histogram of the range of Frequencies detected');
plt.xlabel('Frequency Hz')
plt.show()
```



Time stamps Distribution

In [24]:
```
fig, ax = plt.subplots(1,2, figsize=(24,8))
fig.suptitle(f"In the plots the distribution of timestamps for both classes are shown; 1 indicates a simulated CW present and 0 not present", fontsize=16)
sns.histplot(
        x=list(len(i) for i in train_df['L1_ts']),hue=train_df['label'],
        stat="density", common_norm=False, bins=20, ax=ax[0], kde=True).set_title('Length of measurement for Livingston detector', fontsize=16);

sns.histplot(
        x=list(len(i) for i in train_df['H1_ts']),hue= train_df['label'],
        stat="density", common_norm=False, bins=20, ax=ax[1], kde=True).set_title('Length of measurement for Hanford detector', fontsize=16);
```



Simulating gravitational waves and evaluating their detectability in Python

Rirориro is a Python package to simulate the gravitational waveforms of binary mergers of black holes and/or neutron stars, and calculate several properties of these mergers and waveforms, specifically relating to their observability by gravitational wave detectors."

"The gravitational waveform simulation of Rirориro is based upon the methods of Buskirk and Babiuc-Hamilton (2019), a paper which describes a computational implementation of an earlier theoretical gravitational waveform model by Huerta et al. (2017), using post-Newtonian expansions and an approximation called the implicit rotating source to simplify the Einstein field equations and simulate gravitational waves. Rirориro's calculation of signal-to-noise ratios (SNR) of gravitational wave events is based on the methods of Barrett et al. (2018), with the simpler gravitational wave model Findchirp (Allen et al. (2012)) being used for comparison and calibration in these calculations."

Link for the paper (https://arxiv.org/abs/2103.06943)

cc. of GW simulation: this part was written by Geir Drange (https://www.kaggle.com/code/mistag/reverse-engineering-create-clean-gw-signals) and presented by Marília Prata (https://www.kaggle.com/code/mpwolke/g-waves-rirориro)

"Rirориro is a set of Python modules containing functions to simulate the gravitational waveforms of mergers of black holes and/or neutron stars, and calculate several properties of these mergers and waveforms, specifically relating to their observability by gravitational wave detectors. Rirориro combines areas covered by previous gravitational wave models (such as gravitational wave simulation, SNR calculation, horizon distance calculation) into a single package with broader scope and versatility in Python, a programming language that is ubiquitous in astronomy. Aside from being a research tool, Rirориro is also designed to be easy to use and modify, and it can also be used as an educational tool for students learning about gravitational waves."

"The modules “inspiralfuns”, “mergerfirstfuns”, “matchingfuns”, “mergersecondfuns” and “gwexporter”, in that order, can be used to simulate the strain amplitude and frequency of a merger gravitational waveform. The module “snrcalculatorfuns” can compare such a simulated waveform to a detector noise spectrum to calculate a signal-to-noise ratio (SNR) for that signal for that detector. The module “horizondistfuns” calculates the horizon distance of a merger given its waveform, and the module “detectabilityfuns” evaluates the detectability of a merger given its SNR."

More information on the pip installation can be found here: https://pypi.org/project/rirориro/ (https://pypi.org/project/rirориro/)

Tutorials for Rirориro can be found here: https://github.com/wvanzeist/rirориro_tutorials (https://github.com/wvanzeist/rirориro_tutorials)

Full documentation of each of the functions of Rirориro can be found here: https://wvanzeist.github.io/ (https://wvanzeist.github.io/)

https://github.com/wvanzeist/rirориro (https://github.com/wvanzeist/rirориro)

In [25]:

```
# Code by Geir Drange https://www.kaggle.com/code/mistag/reverse-engineering-create-clean-gw-signals

# Parameters:
# logMC: system mass (0.0-2.0)
# q: mass ratio (0.1-1.0)
# D: distance (Mpc)
# merger_type: 'BH'=binary black hole merger, 'NS'=binary neutron star merger
# flow: low frequency (Hz)
def gen_gw(logMc=1.4, q=0.8, D=100.0, flow=10.0, merger_type='BH'):
    M, eta = ins.get_M_and_eta(logMc=logMc,q=q)
    start_x = ins.startx(M,flow)
    end_x = ins.endx(eta,merger_type)
    x, xtimes, dt = ins.PN_parameter_integration(start_x,end_x,M,eta)
    realtimes = ins.inspiral_time_conversion(xtimes,M)
    i_phase, omega, freq = ins.inspiral_phase_freq_integration(x,dt,M)
    r, rdot = ins.radius_calculation(x,M,eta)
    A1, A2 = ins.a1_a2_calculation(r,rdot,omega,D,M,eta)
    i_Aorth, i_Adiag = ins.inspiral_strain_polarisations(A1,A2,i_phase)
    i_amp = ins.inspiral_strain_amplitude(i_Aorth,i_Adiag)
    i_time = realtimes
    i_omega = omega
    sfin, wqnm = me1.quasi_normal_modes(eta)
    alpha, b, C, kappa = me1.gIRS_coefficients(eta,sfin)
    fhat, m_omega = me1.merger_freq_calculation(wqnm,b,C,kappa)
    fhatdot = me1.fhat_differentiation(fhat)
    m_time = me1.merger_time_conversion(M)
    min_switch_ind = mat.min_switch_ind_finder(i_time,i_omega,m_time,m_omega)
    final_i_index = mat.final_i_index_finder(min_switch_ind,i_omega,m_omega)
    time_offset = mat.time_offset_finder(min_switch_ind,final_i_index,i_time,m_time)
    i_m_time, i_m_omega = mat.time_frequency_stitching(min_switch_ind,final_i_index,time_offset,i_time,i_ome
ga,m_time,m_omega)
    i_m_freq = mat.frequency_SI_units(i_m_omega,M)
    m_phase = me2.merger_phase_calculation(min_switch_ind,final_i_index,i_phase,m_omega)
    i_m_phase = me2.phase_stitching(final_i_index,i_phase,m_phase)
    m_amp = me2.merger_strain_amplitude(min_switch_ind,final_i_index,alpha,i_amp,m_omega,fhat,fhatdot)
    i_m_amp = me2.amplitude_stitching(final_i_index,i_amp,m_amp)
    m_Aorth, m_Adiag = me2.merger_polarisations(final_i_index,m_amp,m_phase,i_Aorth)
    i_m_Aorth, i_m_Adiag = me2.polarisation_stitching(final_i_index,i_Aorth,i_Adiag,m_Aorth,m_Adiag)
    return np.array(i_m_time), np.array(i_m_Aorth), np.array(i_m_Adiag), np.array(i_m_freq)
```

"The function returns two waves that represent orthogonal/diagonal waves. The output timescale that is returned is non-linear, so to convert these signals into uniform sampled signals as in the dataset, we need to resample. The function below will resample the gravitational wave signals to 2048Hz. It is crude though, based on nearest sample, but good enough for studying spectrums. Interpolation would be more proper."

In [26]:

```
SR = 2048 # target sample rate (Hz)
# Parameters:
# dt: time series
# amp: amplitude signal
# seg: output sequence length (seconds)
def resample(dt, amp, seg=2.0):
    end = dt[-1]
    start = end - seg
    d = np.zeros(int(SR*seg))
    for i in range((int(SR*seg))):
        t = start + i/SR
        d[i] = amp[np.where(dt == dt[np.abs(dt-t).argmin()])[0][0]]
    return d
```

In [27]:
```
def plot_sig(dt, sig1, sig2=None, seg=2.0):
    end = dt[-1]
    start = end - seg
    plt.figure(1)
    plt.plot(dt, sig1)
    peak = np.max(np.abs(sig1))
    plt.axis([start,end,np.min(sig1)-peak/10,np.max(sig1)+peak/10])
    if sig2 is not None:
        plt.plot(dt, sig2)
    plt.xlabel('Time (s)')
    plt.ylabel('Strain amplitude')
```

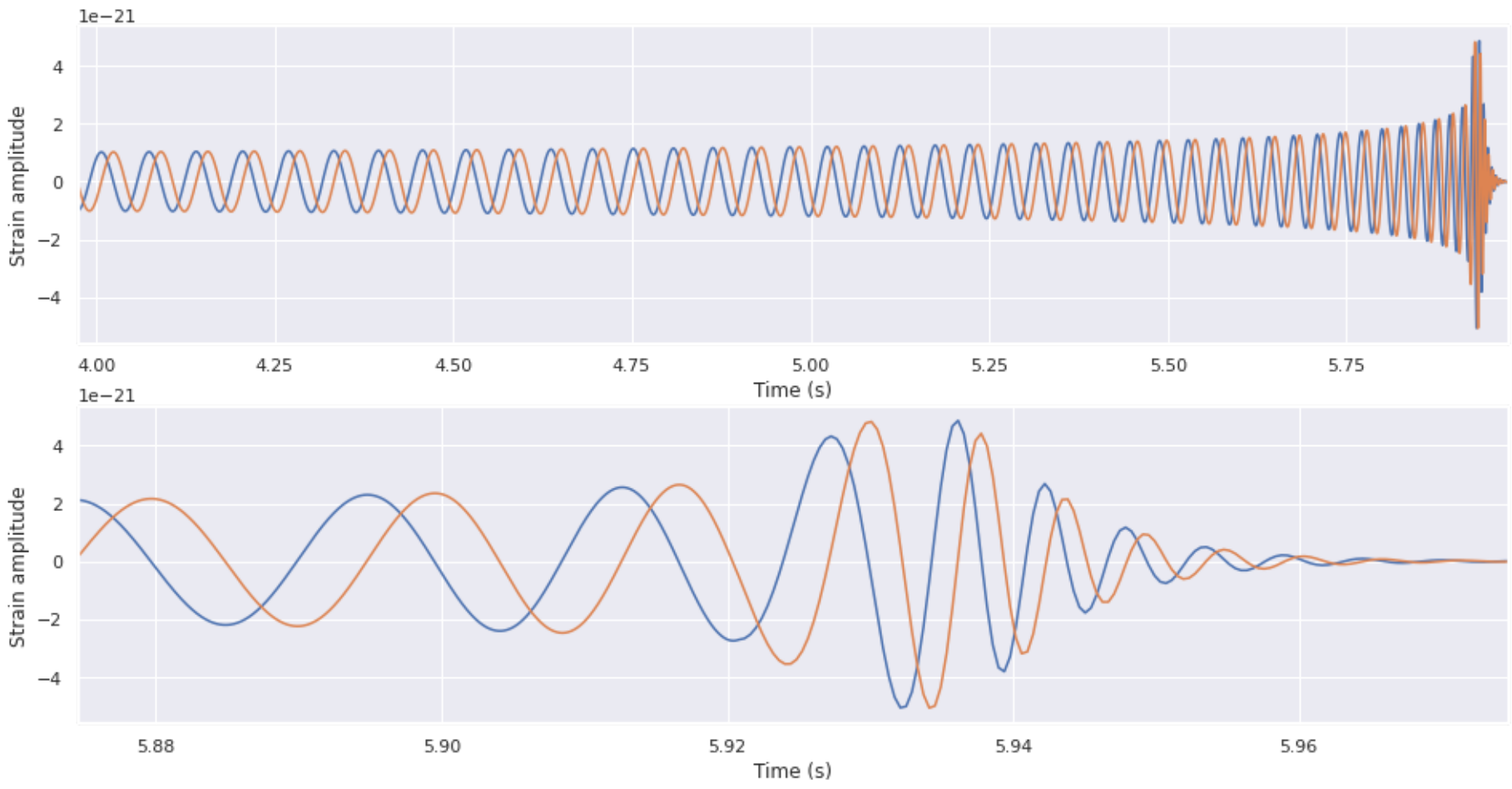
Test signal generation

In [28]:
```
m_time, m_Aorth, m_Adiag, m_freq = gen_gw(logMc=1.4, q=0.2)
```

How Gravitational waves get detected

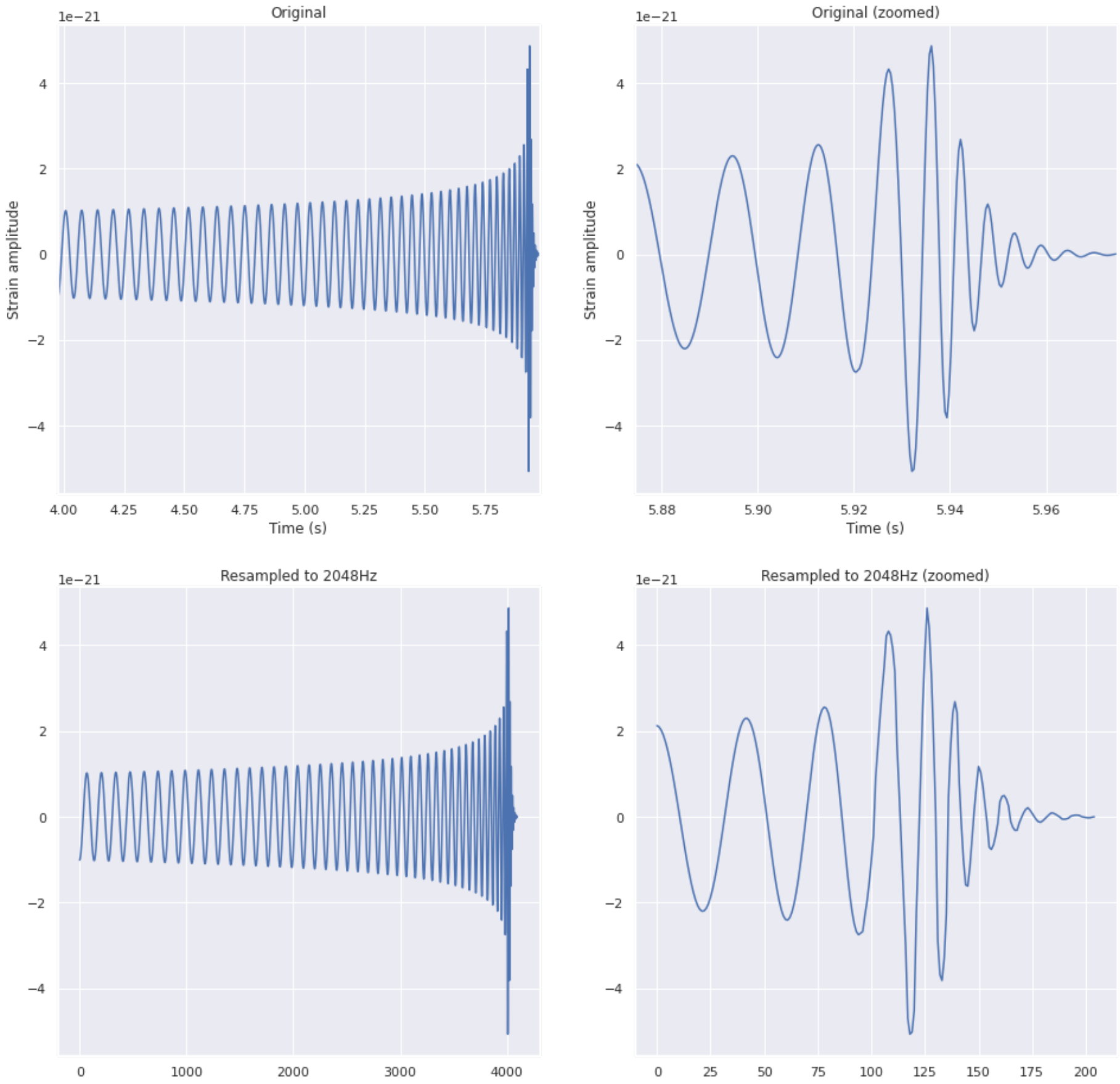
"When a gravitational wave passes by Earth, it squeezes and stretches space. LIGO can detect this squeezing and stretching. Each LIGO observatory has two “arms” that are each more than 2 miles (4 kilometers) long. A passing gravitational wave causes the length of the arms to change slightly. The observatory uses lasers, mirrors, and extremely sensitive instruments to detect these tiny changes."

In [29]:
```
fig = plt.figure(figsize=(16,8))
plt.subplot(2, 1, 1)
plot_sig(m_time, m_Aorth, m_Adiag, seg=2)
plt.subplot(2, 1, 2)
plot_sig(m_time, m_Aorth, m_Adiag, seg=.1)
```



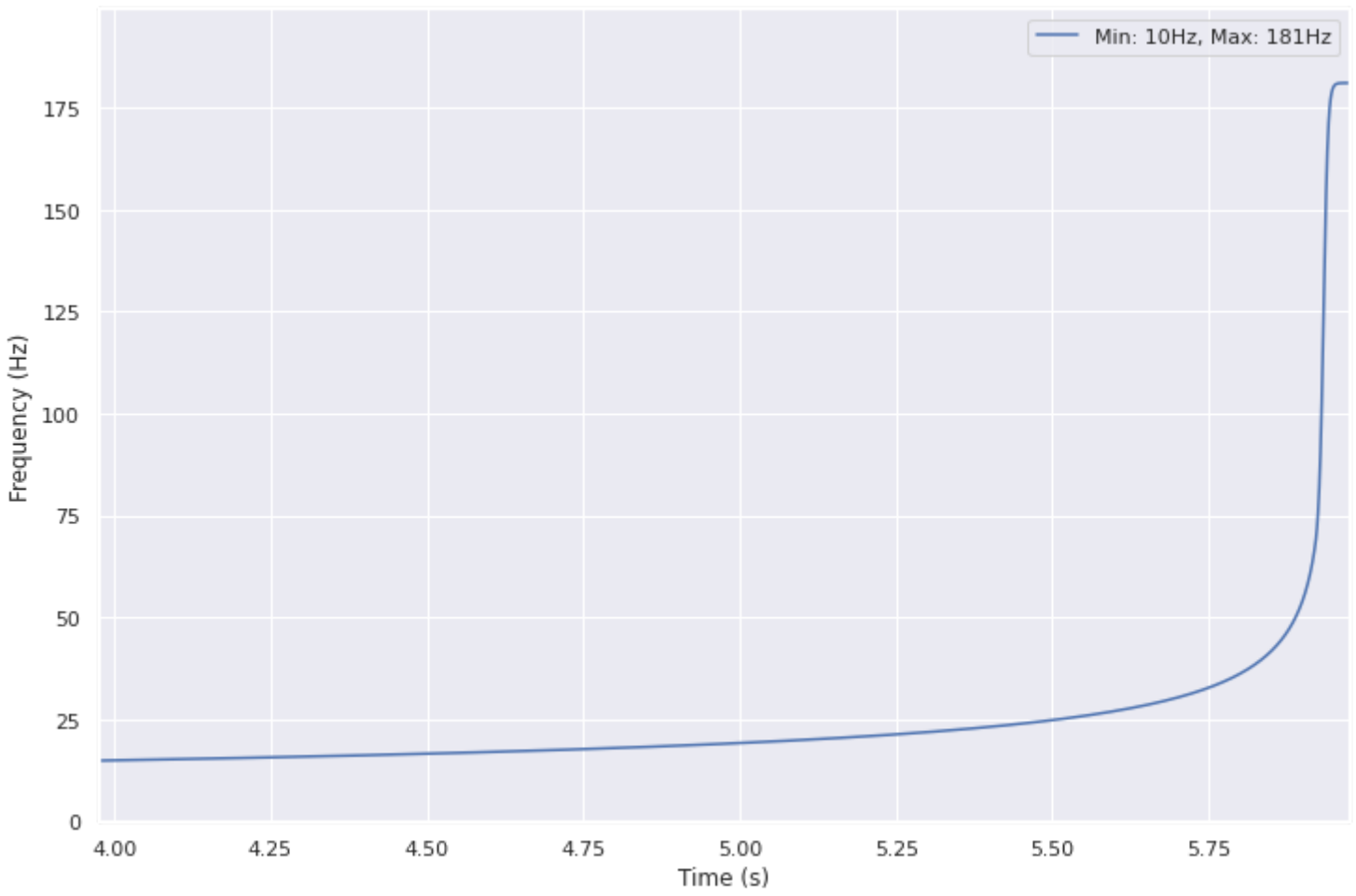
Resample the signal to 2048Hz (only the orthogonal part)

In [30]:
```
d1 = resample(m_time, m_Aorth, 2.0)

fig = plt.figure(figsize=(16,16))
plt.subplot(2, 2, 1)
plot_sig(m_time, m_Aorth, seg=2)
plt.title('Original')
plt.subplot(2, 2, 2)
plot_sig(m_time, m_Aorth, seg=.1)
plt.title('Original (zoomed)')
plt.subplot(2, 2, 3)
plt.plot(d1)
plt.title('Resampled to 2048Hz')
plt.subplot(2, 2, 4)
plt.plot(d1[-205:])
plt.title('Resampled to 2048Hz (zoomed)');
```



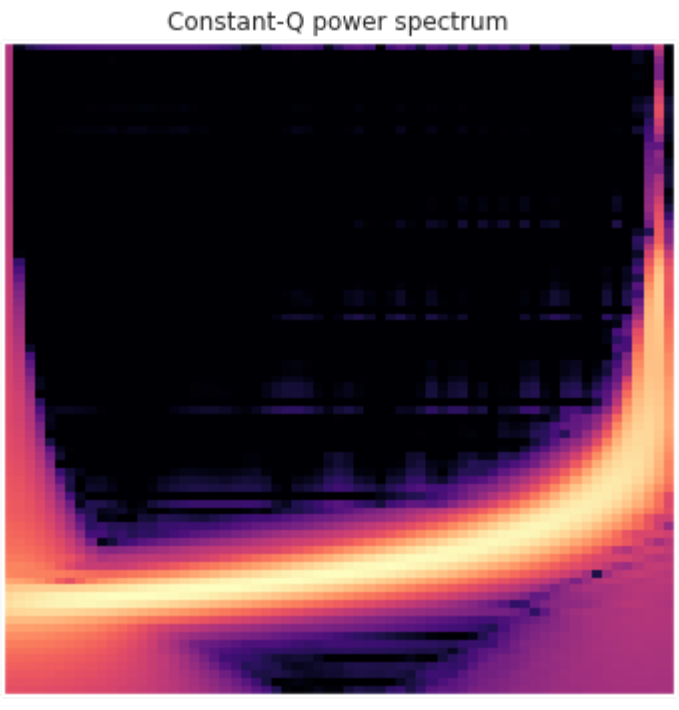
Visualizing Frequency Vector

In [31]:
```
fig, ax = plt.subplots(figsize=(12,8))
plt.plot(m_time, m_freq, label="Min: {}Hz, Max: {}Hz".format(int(np.min(m_freq)), int(np.max(m_freq))))
peak = np.max(np.abs(m_freq))
plt.axis([m_time[-1] - 2.0 if m_time[-1] >= 2.0 else m_time[0], m_time[-1], 0 , np.max(m_freq)+peak/10])
ax.legend()
plt.xlabel('Time (s)')
plt.ylabel('Frequency (Hz)');
```



Visualizing spectrum in frequency domain using Constant-Q transform

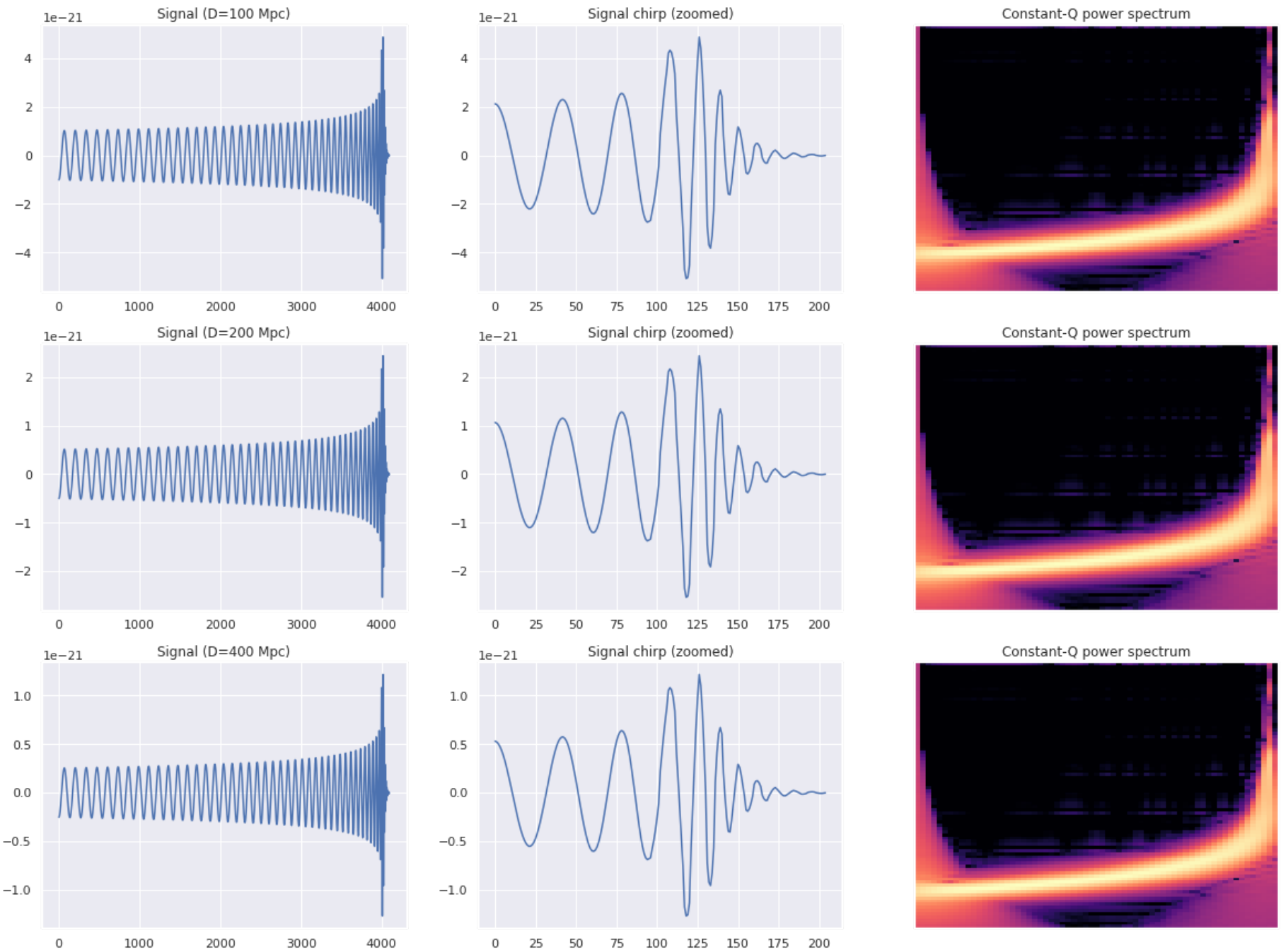
In [32]:
```
hop_length = 64
C = np.abs(librosa.cqt(d1/np.max(d1), sr=SR, hop_length=hop_length, fmin=8, filter_scale=0.8, bins_per_octav
e=12))
fig, ax = plt.subplots(figsize=(6,6))
img = librosa.display.specshow(librosa.amplitude_to_db(C, ref=np.max),
                               sr=SR*2, hop_length=hop_length, bins_per_octave=12, ax=ax)
ax.set_title('Constant-Q power spectrum');
```



Amplitude vs. Distance(Inverse square law verification)

In [33]:

```
hop_length = 64

fig = plt.figure(figsize=(20,15))
dist = [100., 200., 400.]
for m in range(len(dist)):
    m_time, m_Aorth, _, m_freq = gen_gw(logMc=1.4, q=0.2, D=dist[m])
    rd = resample(m_time, m_Aorth, 2.0)
    # time series
    ax = plt.subplot(len(dist), 3, 1+m*3)
    plt.plot(rd)
    plt.title('Signal (D={} Mpc)'.format(int(dist[m])))
    # zoomed times series (chirp)
    ax = plt.subplot(len(dist), 3, 2+m*3)
    plt.plot(rd[-205:])
    plt.title('Signal chirp (zoomed)')
    # Q-Transform
    ax = plt.subplot(len(dist), 3, 3+m*3)
    if m == 0:
        smax = np.max(rd)
    C = np.abs(librosa.cqt(rd/smax, sr=SR, hop_length=hop_length, fmin=8, filter_scale=0.8, bins_per_octave=
12))
    if m == 0:
        Cmax = np.max(C)
    img = librosa.display.specshow(librosa.amplitude_to_db(C, ref=Cmax), # was np.max
                                   sr=SR*2, hop_length=hop_length, bins_per_octave=12, ax=ax)
    ax.set_title('Constant-Q power spectrum');
```



Why the surprising results?

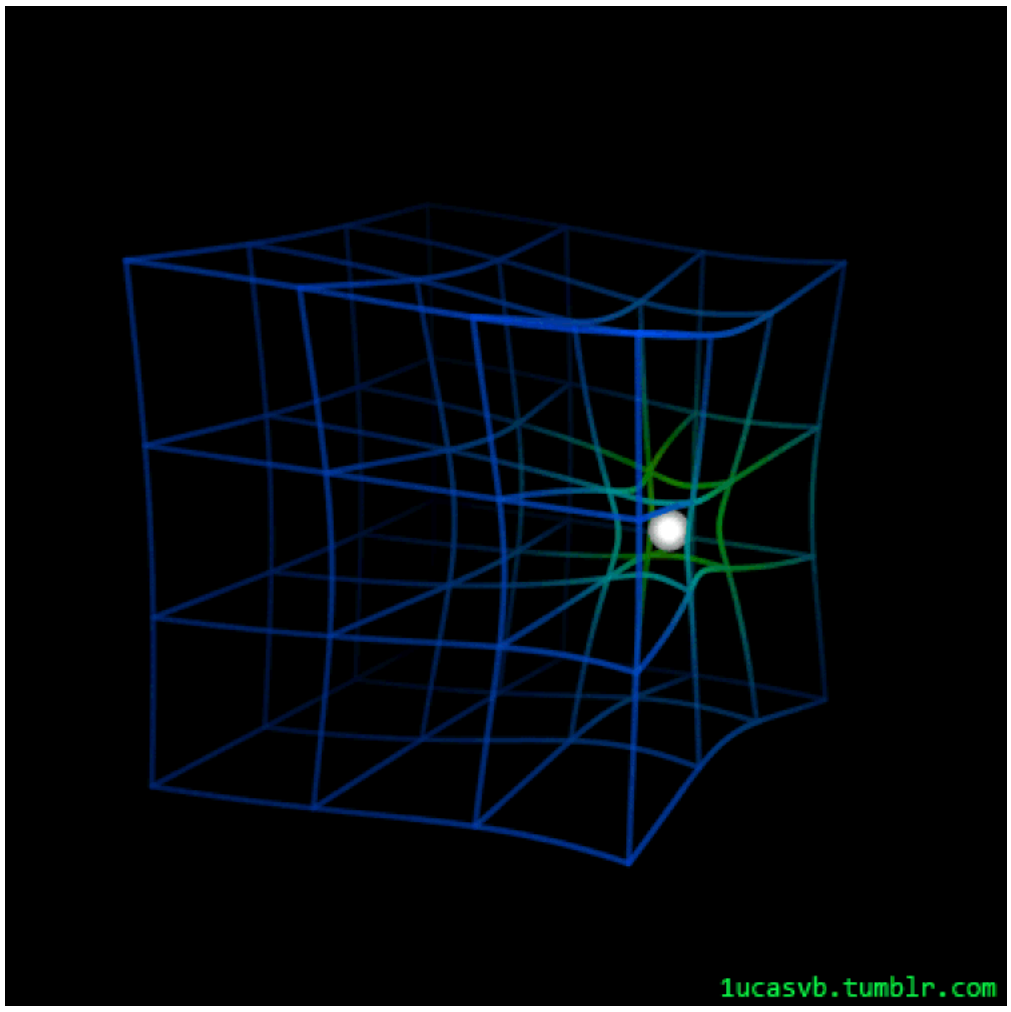
Surprisingly we can notice that gravitational waves amplitude doesn't follow the inverse square law but why?

In order to answer this question we have to explain the difference between monopolic, diapolic and quadrapolic signals. First off, there are fundamental ways that light and gravitational waves are the same. They both:

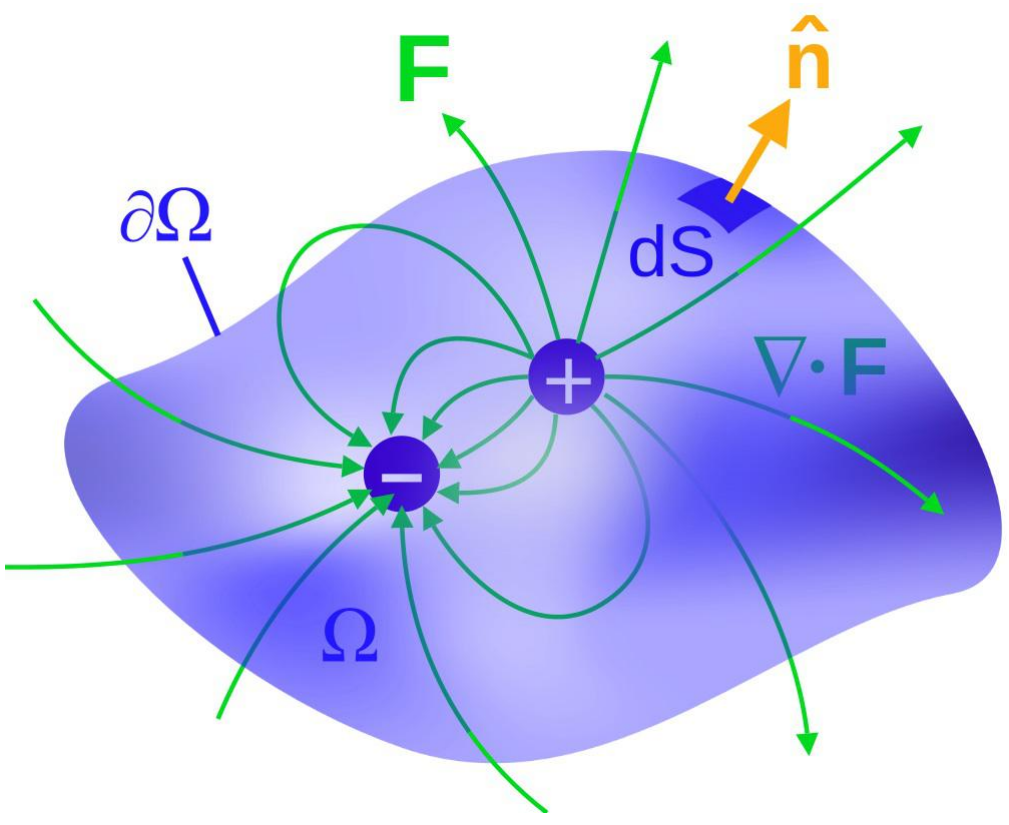
- do carry energy,
- do reach infinite distances,
- do spread out over space (in roughly a sphere) as you move farther away,
- and will be detectable, at a certain distance, in proportion to the magnitude of the signal.

Because the geometry of space is the same for both light and gravitation, the difference between these two behaviors must lie in the nature of the signal that we can detect.

To understand that, we need to understand how gravity is a fundamentally different kind of force than electromagnetism. This will lead us to better understand how gravitational radiation (our gravitational waves) behave differently than electromagnetic radiation (light) when we allow it to propagate across the vast distances of intergalactic space.



If you want to create electromagnetic or gravitational radiation, how could you do it? The simplest way you could imagine — which (spoiler) doesn't work — would be to spontaneously create or destroy charge in a region of space. Having a charge pop into (or out of) existence would create radiation of a very specific type: monopole radiation. Monopole radiation is what happens when you have a change in the amount of charge that's present.

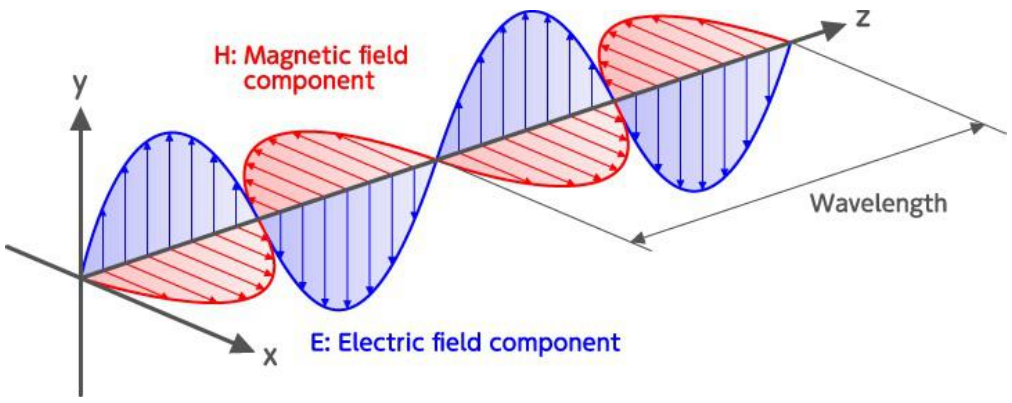


We cannot do this for either electromagnetism or gravitation, however. In electromagnetism, electric charge is conserved; in gravitation, mass/energy

is conserved. The fact that we don't get monopole radiation is important for the stability of our Universe. If charge or mass could spontaneously be created or destroyed, existence would be extremely different!

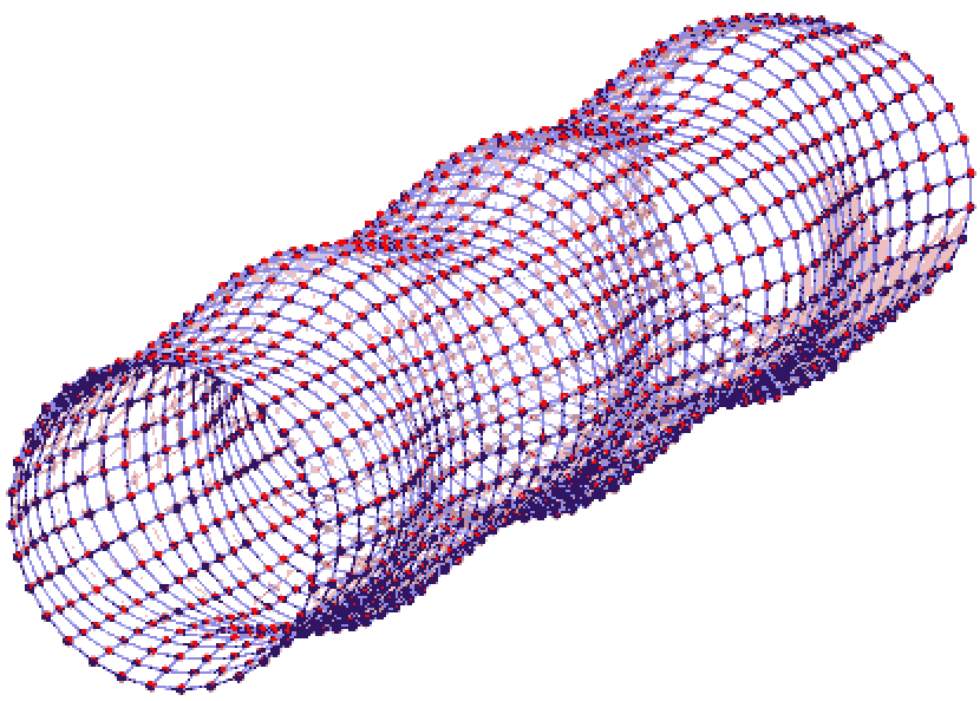
If charge and mass/energy are conserved, then the next step is to either move your charges (or masses) rapidly back-and-forth, or to take charges of opposite signs and change the distance between them. This would create what we call dipole radiation, which changes the distribution of charge without changing the total amount of charge.

In electromagnetism, this creates radiation, because moving an electric charge back-and-forth changes the electric and magnetic fields together. This matters, because changing electric and magnetic fields that are mutually perpendicular to each other and in-phase if wis what an electromagnetic wave actually is. This is the simplest way to make light, and it radiates just like you're familiar with. The light carries energy, and the energy is what we detect, which is why objects appear dimmer as 1/r2 the farther away they are.



In gravity, however, freely moving a mass doesn't make gravitational radiation, because there's a conservation rule about masses in motion: the conservation of momentum. Similarly, separating masses doesn't make gravitational radiation either, because the center of mass remains constant. There's also a conservation rule about masses moving at a certain distance from the center of mass: the conservation of angular momentum.

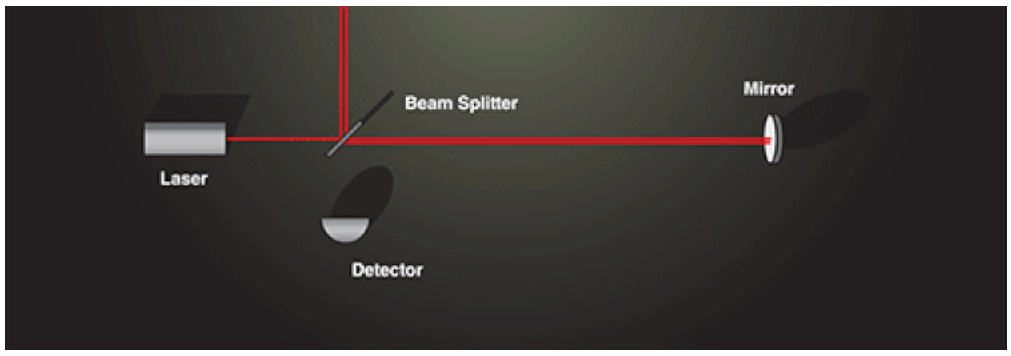
Because energy, momentum, and angular momentum are conserved, you have to go past both monopole and dipole moments; you need a specific change in how the masses are distributed around their mutual center of mass. The simplest way to imagine this is to take two masses and have them mutually rotate around their center of mass, which results in what we call quadrupole radiation.



The amplitude of gravitational quadrupolar radiation falls off as 1/r, meaning the total energy falls off as 1/r2, just as it did for electromagnetic radiation. But this is where the fundamental difference between gravitation and electromagnetism comes in. There's a big difference between what you can physically detect for quadrupole and dipole radiation.

For electromagnetic (dipole) radiation, when the photons hit your detectors, they get absorbed, causing a change in the energy levels, and that change in energy — which remember, falls off as 1/r2 — is the signal you observe. That's why objects appear to dim according to an inverse square law.

For gravitational (quadrupole) radiation, however, it doesn't get directly absorbed in a detector. Rather, it causes objects to move towards or apart from one another in proportion to the amplitude of the wave. Even though the energy falls off as 1/r2, the amplitude only falls off as 1/r. That's why gravitational waves fall off according to a different law than electromagnetic waves.



But the amplitude, as we received it, compressed and expanded the entire Earth by about the diameter of three protons. The energy is huge and falls off as 1/r2, but we cannot detect energy for gravitational waves. We can only detect amplitude, which (thankfully) only falls off as 1/r, which is a very good thing. The amplitudes may be tiny, but if we can detect any signal at all, it's only a small step forward to detecting that same magnitude signal at any distance.

For more check this! (https://www.forbes.com/sites/startswithabang/2019/03/02/ask-ethan-why-dont-gravitational-waves-get-weaker-like-the-gravitational-force-does/?sh=201fe40b2f58)

Gravitational wave signal generation

Standard CW signals can be parameterised in terms of two sets of parameters: the Doppler-modulation parameters λ and the amplitude parameters A. The former encode how the frequency of a signal modulates due to its intrinsic frequency evolution and the movement of the Earth in the Solar system, while the latter describes the overall amplitude of a CW depending on the parameters of the source.

For a CW emmitted by a rapidly-spinning and isolated neutron star (NS), Doppler-modulation parameters include the frequency F0 and the linear spindown parameter F1, both taken at a reference time tref, and the sky position in terms of the right ascension Alpha and declination Delta angles of equatorial cordinates. Amplitude parameters, on the other hand, include the average amplitude of a CW signal h0, the initial phase of the signal phi, the polarization angle psi and (the cosine of) the inclination angle of the source cosi, which gives us the relative orientation of the NS with respect to the detector.

As described in the signal tutorial, the amplitude of a CW signal is usually expressed in terms of the noises's amplitude using depth D or signal-to-noise ratio (SNR) ρ. For our purposes, the former is essentially a quotient

$$\mathcal{D} = \frac{\sqrt{S_\mathrm{n}}}{h_0}$$

while the latter is a more involved expression wich also depends on the duration of the dataset at hand and the detector's response function. It is important to note, however, that ρ and D scale reciprocally: "weak" signals have a low SNR and a high depth (since they are "buried deeper into the noise" than a strong signal).

Data generation

As mebtioned before from the split of train and test datasets the challenge creators are encouraging participants to generate their own data but in out case we will keep it at the point of generating samples for explainations and the inbalanced classes problem will be solved with the class weight parameter while training.

A specific sample requires of background noise and optionally a signal. In order to generate noise, one needs to specify a set of detectors (H1 or L1 in this case), the duration of the sample and the Amplitude Spectral Density of the noise sqrtSX. CW analyses are simple in this front, as sqrtSX is proportional to the (stationary) standard deviation of an underlying zero-mean Gaussian process.

Sample duration can be specified in two ways. If the sample contains contiguous data (i.e. the detector was taking science-quality data uninterrupted), one can simply specify the starting time and duration of the sample using tstart and duration. Data with gaps, on the other hand, can be generated by specifying a specific set of timestamps using the timestamps option.

Data is saved as a list of Short Fourier Transforms (SFTs). The duration and windowing of these SFTs can also be modified using Tsft, SFTWindowType and SFTWindowBeta. Most analyses tune Tsft around 1800 seconds order to ensure the power of a putative CW signal stays within a bin.

For more info check this **notebook** (https://www.kaggle.com/code/rodrigotenorio/generating-continuous-gravitational-wave-signals)

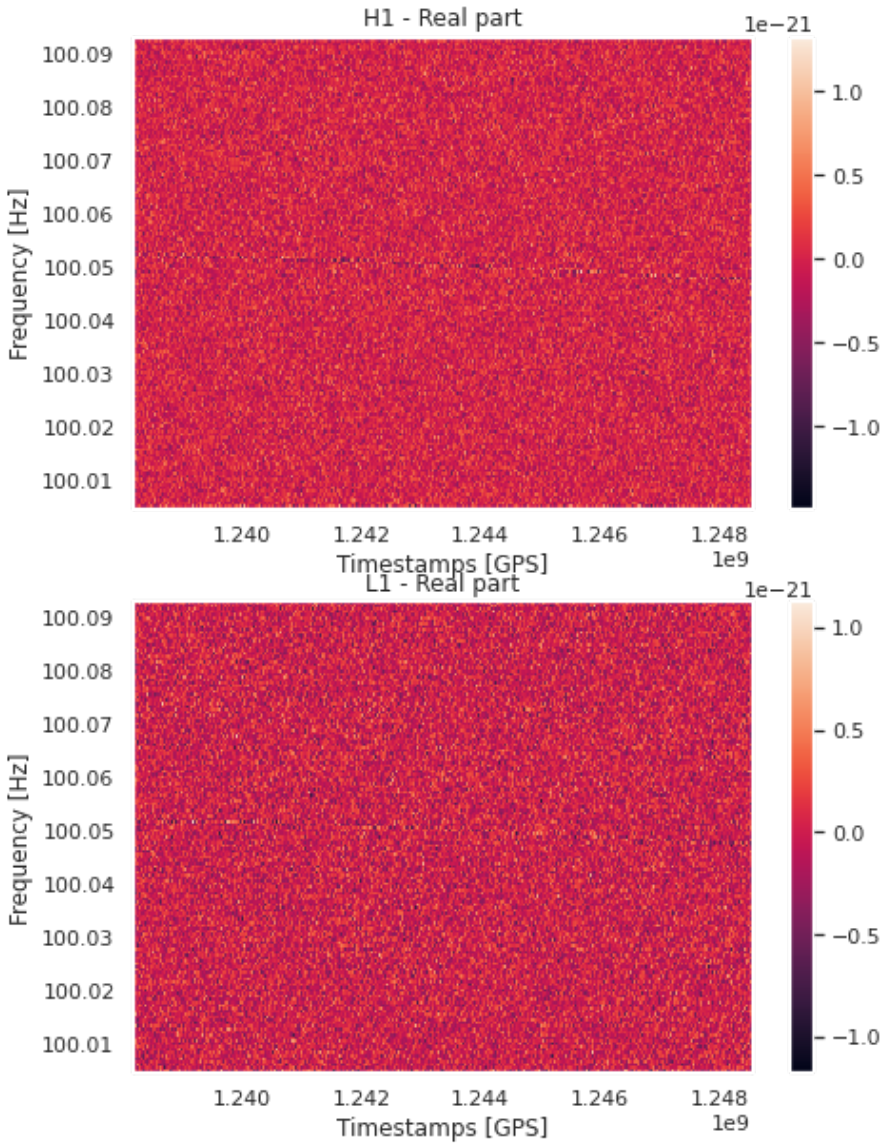
In [34]:
```
# Generate signals with parameters drawn from a specific population
num_signals = 2

# These parameters describe background noise and data format
writer_kwargs = {
                 "tstart": 1238166018,
                 "duration": 4 * 30 * 86400,
                 "detectors": "H1,L1",
                 "sqrtSX": 1e-23,
                 "Tsft": 1800,
                 "SFTWindowType": "tukey",
                 "SFTWindowBeta": 0.01,
                }
```

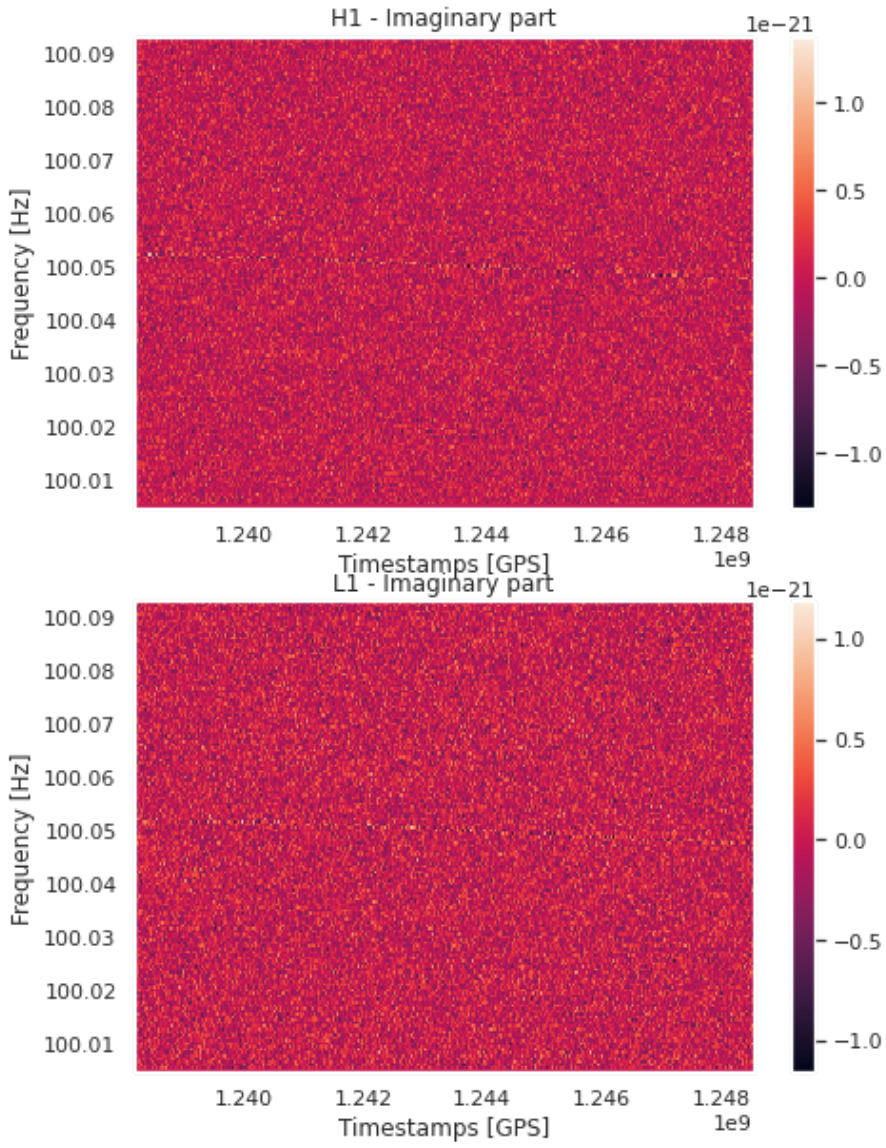
In [35]:
```
# This class allows us to sample signal parameters from a specific population.
# Implicitly, sky positions are drawn uniformly across the celestial sphere.
# PyFstat also implements a convenient set of priors to sample a population
# of isotropically oriented neutron stars.
signal_parameters_generator = pyfstat.AllSkyInjectionParametersGenerator(
    priors={
        "tref": writer_kwargs["tstart"],
        "F0": {"uniform": {"low": 100.0, "high": 100.1}},
        "F1": lambda: 10**stats.uniform(-12, 4).rvs(),
        "F2": 0,
        "h0": lambda: writer_kwargs["sqrtSX"] / stats.uniform(1, 10).rvs(),
        **pyfstat.injection_parameters.isotropic_amplitude_priors,
    },
)
```

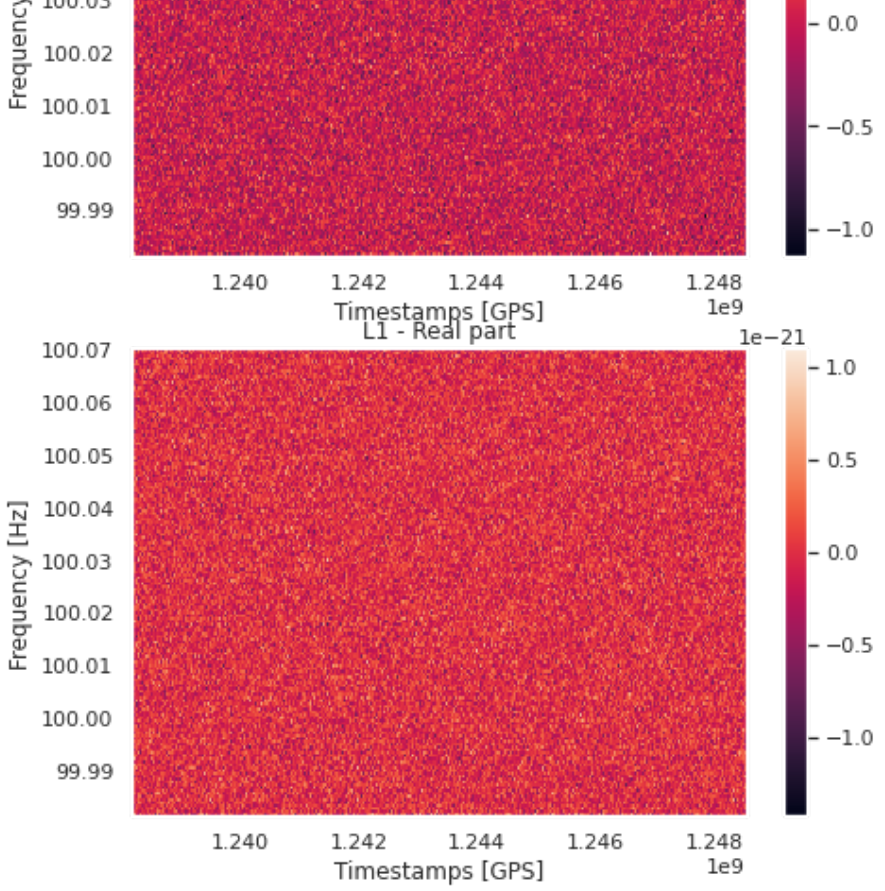
In [36]:

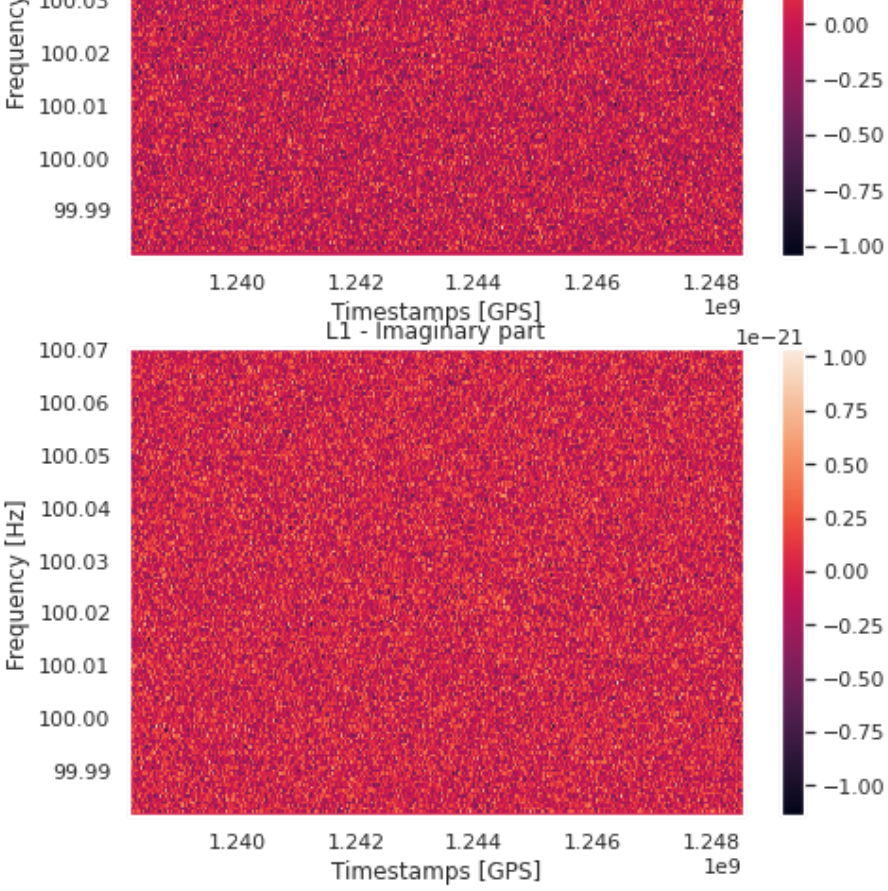
```
snrs = np.zeros(num_signals)

for ind in range(num_signals):

    # Draw signal parameters.
    # Noise can be drawn by setting `params["h0"] = 0
    params = signal_parameters_generator.draw()
    writer_kwargs["outdir"] = f"PyFstat_example_data_ensemble/Signal_{ind}"
    writer_kwargs["label"] = f"Signal_{ind}"

    writer = pyfstat.Writer(**writer_kwargs, **params)
    writer.make_data()

    # SNR can be compute from a set of SFTs for a specific set
    # of parameters as follows:
    snr = pyfstat.SignalToNoiseRatio.from_sfts(
        F0=writer.F0, sftfilepath=writer.sftfilepath
    )
    squared_snr = snr.compute_snr2(
        Alpha=writer.Alpha,
        Delta=writer.Delta,
        psi=writer.psi,
        phi=writer.phi,
        h0=writer.h0,
        cosi=writer.cosi
    )
    snrs[ind] = np.sqrt(squared_snr)

    # Data can be read as a numpy array using PyFstat
    frequency, timestamps, amplitudes = pyfstat.utils.get_sft_as_arrays(
        writer.sftfilepath
    )

    fig, ax = plt.subplots(2, 2, figsize=(16, 10))
    fig.suptitle(f"Signal {ind} - SNR: {snrs[ind]:.2f}")
    for d_ind, detector in enumerate(amplitudes.keys()):
        ax[d_ind][0].set(xlabel="Timestamps [GPS]",
                         ylabel="Frequency [Hz]",
                         title=f"{detector} - Real part")
        ax[d_ind][1].set(xlabel="Timestamps [GPS]",
                         ylabel="Frequency [Hz]",
                         title=f"{detector} - Imaginary part")

        c0 = ax[d_ind][0].pcolormesh(timestamps[detector], frequency,
                                     amplitudes[detector].real)
        c1 = ax[d_ind][1].pcolormesh(timestamps[detector], frequency,
                                     amplitudes[detector].imag)

        fig.colorbar(c0, ax=ax[d_ind][0])
        fig.colorbar(c1, ax=ax[d_ind][1])
    plt.show()
```

Signal 0 - SNR: 288.23





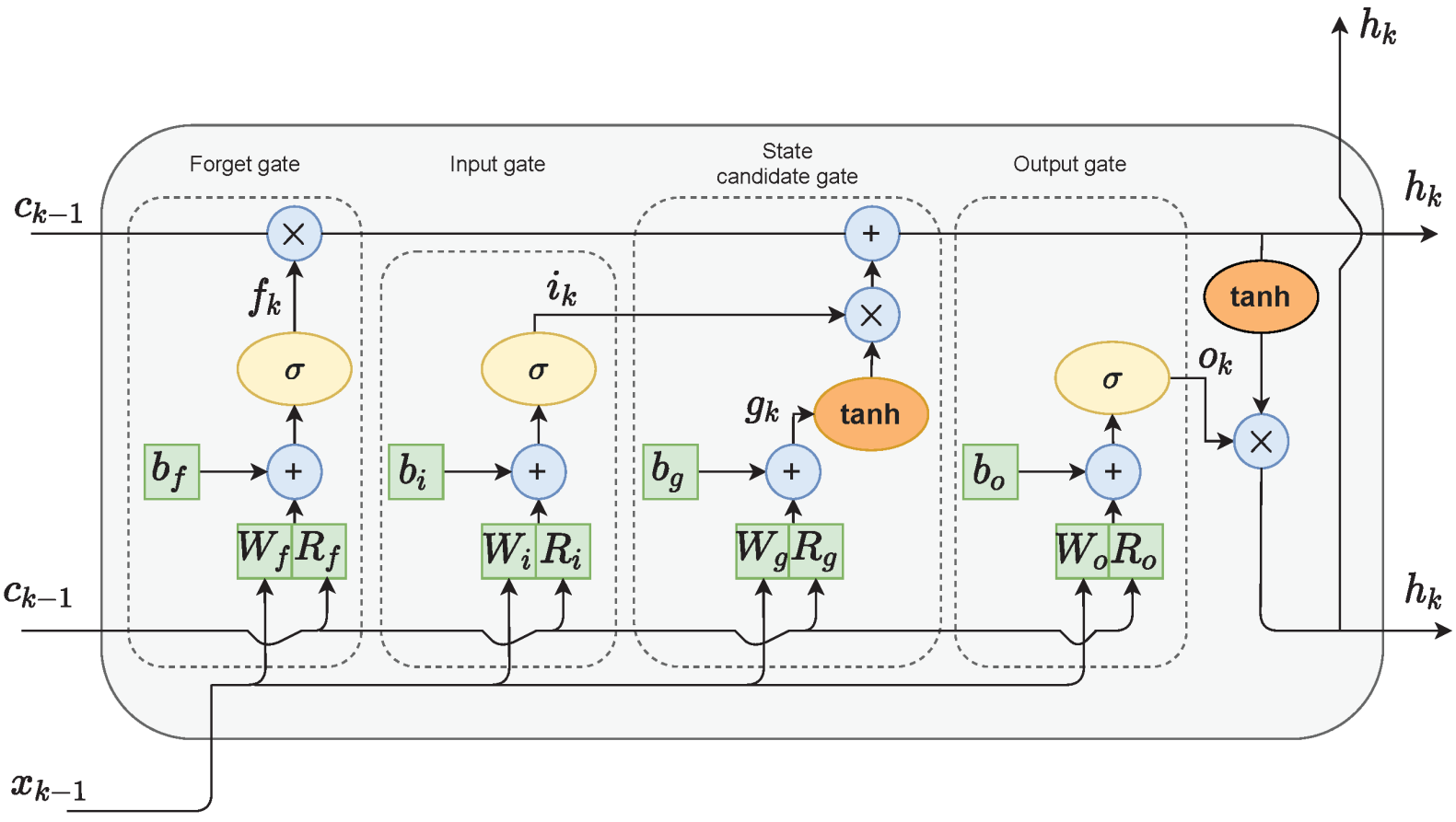




Signal 1 - SNR: 164.88

Modeling with LSTM in Time Domain

Long short-term memory (LSTM) is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network (RNN) can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition, machine translation, robot control, video games, and healthcare. LSTM has become the most cited neural network of the 20th century.



The name of LSTM refers to the analogy that a standard RNN has both "long-term memory" and "short-term memory". The connection weights and biases in the network change once per episode of training, analogous to how physiological changes in synaptic strengths store long-term memories; the activation patterns in the network change once per time-step, analogous to how the moment-to-moment change in electric firing patterns in the brain store short-term memories. The LSTM architecture aims to provide a short-term memory for RNN that can last thousands of timesteps, thus "long short-term memory".

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.

The compact forms of the equations for the forward pass of an LSTM cell with a forget gate are:

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma_h(c_t)
\end{aligned}
$$

In [37]:

```
X_l = np.asarray(time_df['L1_resampled_time'].to_list())
X_h = np.asarray(time_df['H1_resampled_time'].to_list())
y = np.asarray(time_df['label'].to_list())

X_l = np.expand_dims(X_l, axis=1)
X_h = np.expand_dims(X_h, axis=1)
```

In [38]:

```
def create_lstm(x_input):

    lstm = tf.keras.Sequential([

        tf.keras.layers.LSTM(128, kernel_initializer='normal',input_shape=(1, X_l.shape[2]), return_sequence
s=True),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.LSTM(128, kernel_initializer='normal', return_sequences=True),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.LSTM(128, kernel_initializer='normal', return_sequences=True),
        tf.keras.layers.Dropout(0.2),

        tf.keras.layers.Dense(units=128, kernel_initializer='normal', activation='relu',
                              kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3),
                              bias_regularizer=regularizers.L2(1e-2),
                              activity_regularizer=regularizers.L2(1e-3)),
        tf.keras.layers.Dropout(0.5)
    ])

    features = lstm(x_input)
    x = layers.Dense(256, activation='relu')(features)
    return x
```

In [39]:

```
def get_lstm_model():
    # 1) Hanford
    h_input = tf.keras.layers.Input(shape=(1,X_l.shape[2]), name='x_h')
    # 2) Livingston
    l_input = tf.keras.layers.Input(shape=(1, X_l.shape[2]), name='x_l')

    h_out = create_lstm(h_input)
    l_out = create_lstm(l_input)



    # Concatenate embeddings
    x = tf.keras.layers.Concatenate()([h_out, l_out])
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(128, kernel_initializer='normal')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    # Target prediction in range [0,1] with sigmoid activation
    output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

    # Model
    inputs = [h_input, l_input]
    model = tf.keras.models.Model(inputs=inputs, outputs=output)

    optimizer = 'adam'

    model.compile(
        optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics = [
            tf.keras.metrics.AUC() ,
        ]
    )
    return model
```

In [40]:
```
tf.keras.backend.clear_session()
gc.collect()
lstmmodel = get_lstm_model()
```

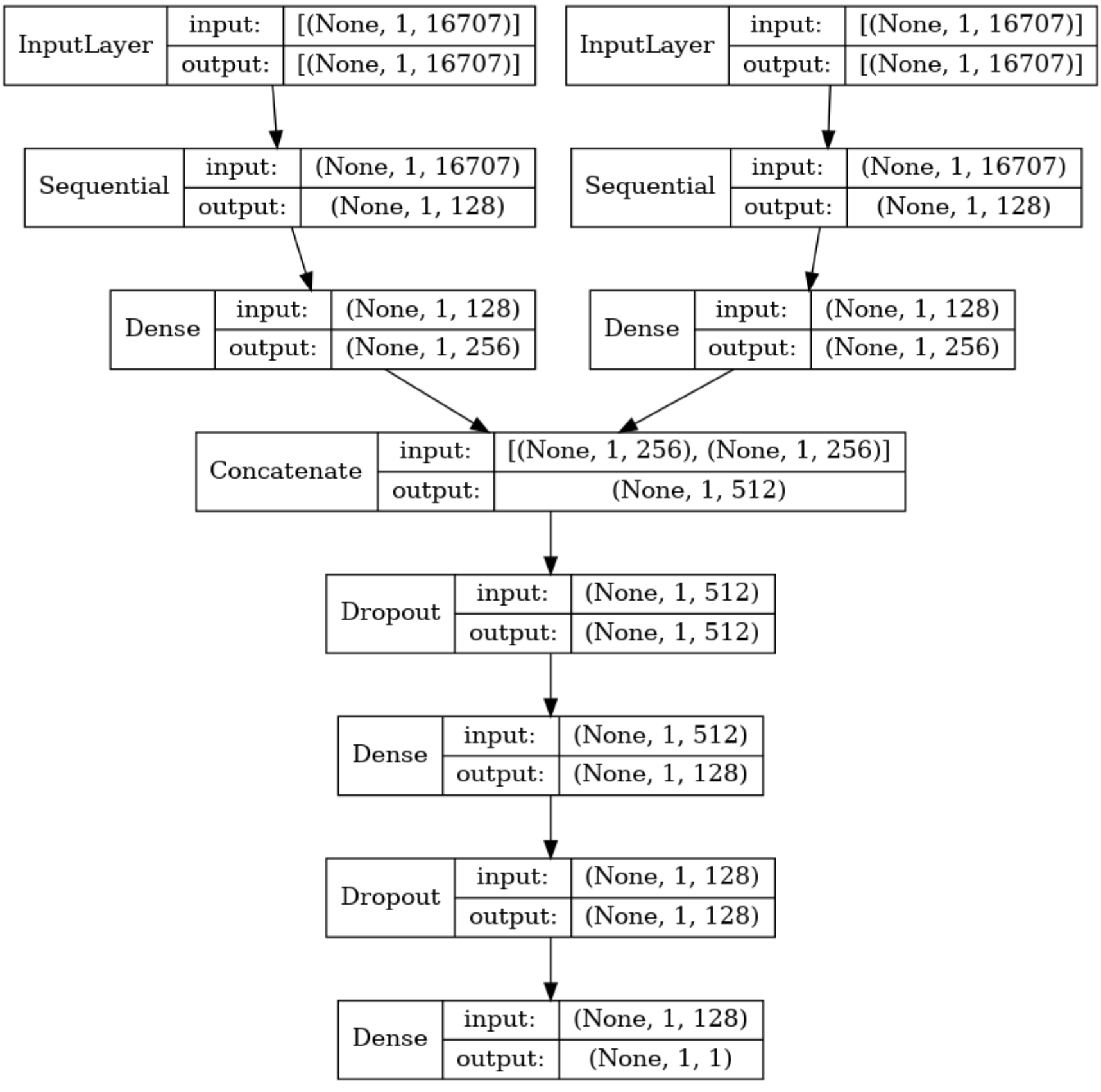
```
2022-10-14 17:24:13.640206: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread po
ol with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
```

In [41]:
```
print(lstmmodel.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
x_h (InputLayer)                [(None, 1, 16707)]   0
__________________________________________________________________________________________________
x_l (InputLayer)                [(None, 1, 16707)]   0
__________________________________________________________________________________________________
sequential (Sequential)         (None, 1, 128)       8899712     x_h[0][0]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 1, 128)       8899712     x_l[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1, 256)       33024       sequential[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1, 256)       33024       sequential_1[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1, 512)       0           dense_1[0][0]
                                                                 dense_3[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 1, 512)       0           concatenate[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 1, 128)       65664       dropout_8[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 1, 128)       0           dense_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 1, 1)         129         dropout_9[0][0]
==================================================================================================
Total params: 17,931,265
Trainable params: 17,931,265
Non-trainable params: 0
__________________________________________________________________________________________________
None
```

In [42]:
```
plot_model(lstmmodel, show_shapes=True, show_layer_names=False, to_file="lstmmodel.png")
```

Out[42]:



In [43]:

```
lstmhistory = lstmmodel.fit(
        x=[X_l, X_h],
        y=y,
        epochs = 50,
        validation_split=0.2,
        verbose = 1,
    )
```

```
2022-10-14 17:24:18.945158: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185] None of the
MLIR Optimization Passes are enabled (registered 2)
```

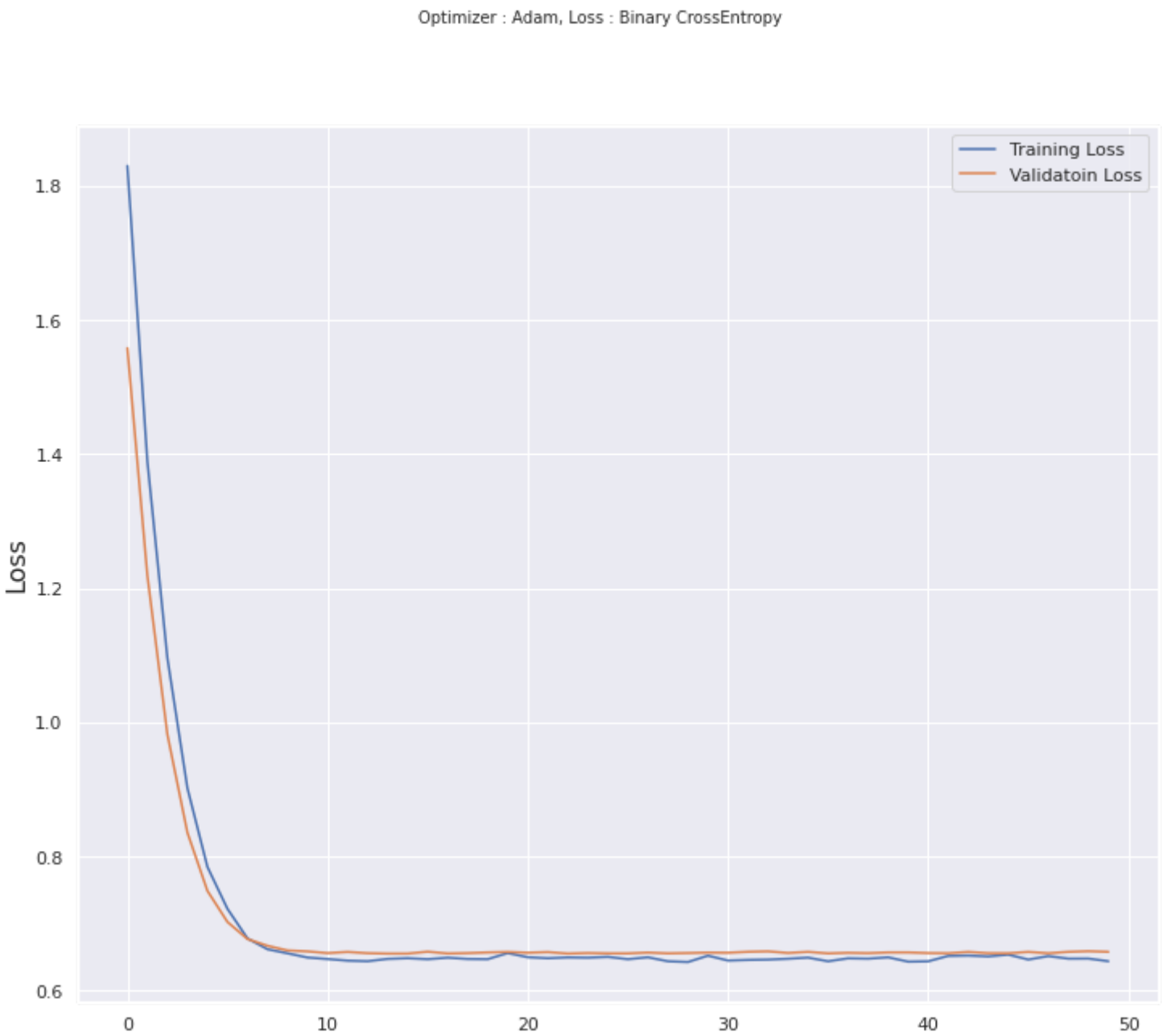
```
Epoch 1/50
16/16 [==============================] - 15s 247ms/step - loss: 1.8301 - auc: 0.5603 - val_loss: 1.5583
- val_auc: 0.5000
Epoch 2/50
16/16 [==============================] - 1s 71ms/step - loss: 1.3891 - auc: 0.5019 - val_loss: 1.2190 -
val_auc: 0.5000
Epoch 3/50
16/16 [==============================] - 1s 72ms/step - loss: 1.0960 - auc: 0.4774 - val_loss: 0.9820 -
val_auc: 0.5000
Epoch 4/50
16/16 [==============================] - 1s 71ms/step - loss: 0.9019 - auc: 0.5029 - val_loss: 0.8355 -
val_auc: 0.5000
Epoch 5/50
16/16 [==============================] - 1s 71ms/step - loss: 0.7844 - auc: 0.4895 - val_loss: 0.7487 -
val_auc: 0.5000
Epoch 6/50
16/16 [==============================] - 1s 72ms/step - loss: 0.7219 - auc: 0.4610 - val_loss: 0.7023 -
val_auc: 0.5000
Epoch 7/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6778 - auc: 0.5252 - val_loss: 0.6768 -
val_auc: 0.5000
Epoch 8/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6616 - auc: 0.5132 - val_loss: 0.6668 -
val_auc: 0.5000
Epoch 9/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6555 - auc: 0.4738 - val_loss: 0.6598 -
val_auc: 0.5000
Epoch 10/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6492 - auc: 0.4813 - val_loss: 0.6584 -
val_auc: 0.5000
Epoch 11/50
16/16 [==============================] - 1s 75ms/step - loss: 0.6470 - auc: 0.4743 - val_loss: 0.6558 -
val_auc: 0.5000
Epoch 12/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6446 - auc: 0.4989 - val_loss: 0.6576 -
val_auc: 0.5000
Epoch 13/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6437 - auc: 0.5082 - val_loss: 0.6557 -
val_auc: 0.5000
Epoch 14/50
16/16 [==============================] - 1s 74ms/step - loss: 0.6472 - auc: 0.4831 - val_loss: 0.6551 -
val_auc: 0.5000
Epoch 15/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6482 - auc: 0.4751 - val_loss: 0.6551 -
val_auc: 0.5000
Epoch 16/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6468 - auc: 0.4954 - val_loss: 0.6580 -
val_auc: 0.5000
Epoch 17/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6490 - auc: 0.4737 - val_loss: 0.6552 -
val_auc: 0.5000
Epoch 18/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6471 - auc: 0.5071 - val_loss: 0.6558 -
val_auc: 0.5000
Epoch 19/50
16/16 [==============================] - 1s 73ms/step - loss: 0.6468 - auc: 0.4944 - val_loss: 0.6568 -
val_auc: 0.5000
Epoch 20/50
16/16 [==============================] - 2s 105ms/step - loss: 0.6559 - auc: 0.4508 - val_loss: 0.6575 -
val_auc: 0.5000
Epoch 21/50
```

```
16/16 [==============================] - 1s 70ms/step - loss: 0.6497 - auc: 0.4538 - val_loss: 0.6564 -
val_auc: 0.5000
Epoch 22/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6482 - auc: 0.4768 - val_loss: 0.6573 -
val_auc: 0.5000
Epoch 23/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6494 - auc: 0.4615 - val_loss: 0.6551 -
val_auc: 0.5000
Epoch 24/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6490 - auc: 0.4897 - val_loss: 0.6559 -
val_auc: 0.5000
Epoch 25/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6502 - auc: 0.4527 - val_loss: 0.6554 -
val_auc: 0.5000
Epoch 26/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6468 - auc: 0.5192 - val_loss: 0.6554 -
val_auc: 0.5000
Epoch 27/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6496 - auc: 0.4624 - val_loss: 0.6566 -
val_auc: 0.5000
Epoch 28/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6437 - auc: 0.5202 - val_loss: 0.6555 -
val_auc: 0.5000
Epoch 29/50
16/16 [==============================] - 1s 76ms/step - loss: 0.6426 - auc: 0.5384 - val_loss: 0.6559 -
val_auc: 0.5000
Epoch 30/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6520 - auc: 0.4630 - val_loss: 0.6566 -
val_auc: 0.5000
Epoch 31/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6448 - auc: 0.5163 - val_loss: 0.6564 -
val_auc: 0.5000
Epoch 32/50
16/16 [==============================] - 1s 69ms/step - loss: 0.6457 - auc: 0.5019 - val_loss: 0.6578 -
val_auc: 0.5063
Epoch 33/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6461 - auc: 0.5135 - val_loss: 0.6584 -
val_auc: 0.5000
Epoch 34/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6474 - auc: 0.4861 - val_loss: 0.6560 -
val_auc: 0.5000
Epoch 35/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6491 - auc: 0.4845 - val_loss: 0.6577 -
val_auc: 0.5000
Epoch 36/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6436 - auc: 0.5226 - val_loss: 0.6554 -
val_auc: 0.5000
Epoch 37/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6480 - auc: 0.5076 - val_loss: 0.6563 -
val_auc: 0.5000
Epoch 38/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6475 - auc: 0.5020 - val_loss: 0.6557 -
val_auc: 0.5000
Epoch 39/50
16/16 [==============================] - 1s 75ms/step - loss: 0.6495 - auc: 0.4766 - val_loss: 0.6568 -
val_auc: 0.5000
Epoch 40/50
16/16 [==============================] - 1s 75ms/step - loss: 0.6431 - auc: 0.5372 - val_loss: 0.6568 -
val_auc: 0.5000
Epoch 41/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6436 - auc: 0.5284 - val_loss: 0.6559 -
val_auc: 0.5000
```

```
Epoch 42/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6516 - auc: 0.4531 - val_loss: 0.6556 -
val_auc: 0.5000
Epoch 43/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6522 - auc: 0.4752 - val_loss: 0.6576 -
val_auc: 0.5000
Epoch 44/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6509 - auc: 0.4922 - val_loss: 0.6557 -
val_auc: 0.5000
Epoch 45/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6535 - auc: 0.4259 - val_loss: 0.6556 -
val_auc: 0.5000
Epoch 46/50
16/16 [==============================] - 1s 70ms/step - loss: 0.6462 - auc: 0.5281 - val_loss: 0.6576 -
val_auc: 0.5000
Epoch 47/50
16/16 [==============================] - 1s 88ms/step - loss: 0.6513 - auc: 0.4804 - val_loss: 0.6556 -
val_auc: 0.5000
Epoch 48/50
16/16 [==============================] - 2s 90ms/step - loss: 0.6476 - auc: 0.4903 - val_loss: 0.6577 -
val_auc: 0.5000
Epoch 49/50
16/16 [==============================] - 1s 71ms/step - loss: 0.6477 - auc: 0.4852 - val_loss: 0.6587 -
val_auc: 0.5000
Epoch 50/50
16/16 [==============================] - 1s 72ms/step - loss: 0.6437 - auc: 0.5458 - val_loss: 0.6577 -
```

In [44]:
```
plt.figure(figsize=(12,10))

plt.suptitle('Optimizer : Adam, Loss : Binary CrossEntropy', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(lstmhistory.history['loss'], label='Training Loss')
plt.plot(lstmhistory.history['val_loss'], label='Validatoin Loss')
plt.legend(loc='upper right')
```

Out[44]:
```
<matplotlib.legend.Legend at 0x7f3d59e6a210>
```



Saving LSTM Model weights

In [45]:
```
lstmmodel.save_weights('lstmmodel.h5')
```

Releasing Memory

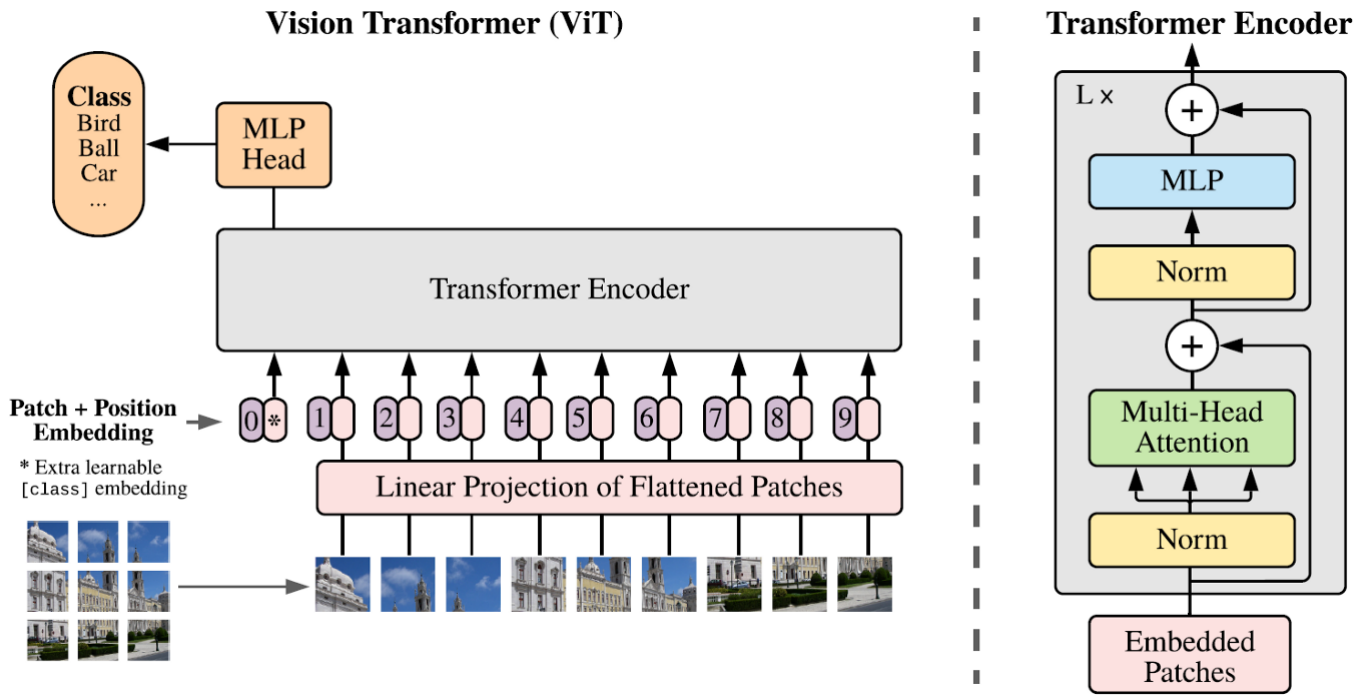
In [46]:
```
del time_df, train_df
```

In [47]:
```
gc.collect()
```

Out[47]:
```
1772
```

Modeling with Vision Transformers(ViT) and Convolutional Neural Network(CNN) in Frequency Domain

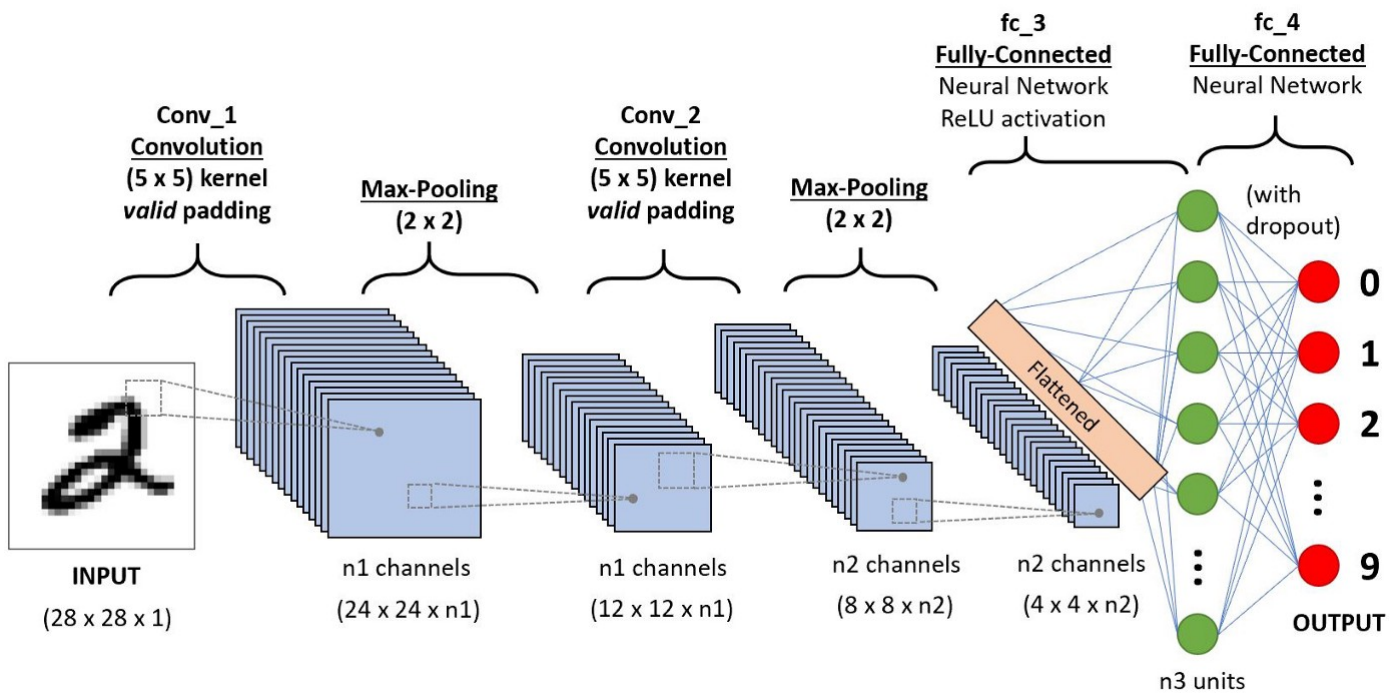
The concept of Vision Transformer (ViT) is an extension of the original concept of Transformer. It is only the application of Transformer in the image domain with slight modification in the implementation in order to handle the different data modality. More specifically, a ViT uses different methods for tokenization and embedding. However, the generic architecture remains the same. An input image is split into a set of image patches, called visual tokens. The visual tokens are embedded into a set of encoded vectors of fixed dimension. The position of a patch in the image is embedded along with the encoded vector and fed into the transformer encoder network which is essentially the same as the one responsible for processing the text input.



There are multiple blocks in the ViT encoder and each block consists of three major processing elements: Layer Norm, Multi-head Attention Network (MSP) and Multi-Layer Perceptrons (MLP). Layer Norm keeps the training process on track and let model adapt to the variations among the training images. MSP is a network responsible for generation of attention maps from the given embedded visual tokens. These attention maps help network focus on most important regions in the image such as object(s).

Convolutional Neural Network

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most commonly applied to analyze visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps. Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the downsampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series.



CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "full connectivity" of these networks make them prone to overfitting data. Typical ways of regularization, or preventing overfitting, include: penalizing parameters during training (such as weight decay) or trimming connectivity (skipped connections, dropout, etc.) CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble patterns of increasing complexity using smaller and simpler patterns embossed in their filters. Therefore, on a scale of connectivity and complexity, CNNs are on the lower extreme.

General CNN vs. ViT talk

The differences between CNNs and Vision Transformers are many and lie mainly in their architectural differences. In fact, CNNs achieve excellent results even with training based on data volumes that are not as large as those required by Vision Transformers. This different behaviour seems to derive from the presence in the CNNs of some inductive biases that can be somehow exploited by these networks to grasp more quickly the particularities of the analysed images even if, on the other hand, they end up limiting them making it more complex to grasp global relations.

On the other hand, the Vision Transformers are free from these biases which leads them to be able to capture also global and wider range relations but at the cost of a more onerous training in terms of data. Vision Transformers also proved to be much more robust to input image distortions such as adversarial patches or permutations. However, choosing one architecture over another is not always the wisest choice, and excellent results have been

Generating 360×360 Dataset by Mark Wijkhuizen (https://www.kaggle.com/code/markwijkhuizen/g2net-eda-360×360-dataset-creation/notebook)

In [48]:
```
# Number of Samples in train dataset
N_SAMPLES = len(labels_df)
# Make 360x360 Patches
TARGET_HEIGHT = 360
TARGET_WIDTH = 360
print(f'TARGET_HEIGHT: {TARGET_HEIGHT}, TARGET_WIDTH: {TARGET_WIDTH}')
IS_INTERACTIVE = os.environ['KAGGLE_KERNEL_RUN_TYPE'] == 'Interactive'
INPUTS = ['x_h_r', 'x_h_i', 'x_l_r', 'x_l_i']
```

```
TARGET_HEIGHT: 360, TARGET_WIDTH: 360
```

In [49]:
```
# Get recording data type
# A handful of complex128 recordings are present which will be ignored
def get_dtype(train_id):
    file = h5py.File(f'{TRAIN_PATH}/{train_id}.hdf5', 'r')[train_id]
    return file['H1']['SFTs'].dtype

labels_df['dtype'] = labels_df['id'].apply(get_dtype)
```

In [50]:
```
N_FLOAT64_SAMPLES = (labels_df['dtype'] == 'complex64').sum()
print(f'N_FLOAT64_SAMPLES: {N_FLOAT64_SAMPLES}')

display(labels_df['dtype'].value_counts().to_frame())
```

```
N_FLOAT64_SAMPLES: 600
```

	dtype
complex64	600
complex128	3

In [51]:
```
train_dimension_rows = []

for train_id in tqdm(labels_df['id']):
    file = h5py.File(f'{TRAIN_PATH}/{train_id}.hdf5', 'r')[train_id]
    SFT_H = file['H1']['SFTs']
    SFT_L = file['L1']['SFTs']
    train_dimension_rows.append({
        'id': train_id,
        'H_height': SFT_H.shape[0],
        'H_width': SFT_H.shape[1],
        'L_height': SFT_L.shape[0],
        'L_width': SFT_L.shape[1],
    })
```

In [52]:
```
labels_df = labels_df.merge(pd.DataFrame(train_dimension_rows), on='id')
display(labels_df.head())
```

	id	target	dtype	H_height	H_width	L_height	L_width
0	001121a05	1	complex64	360	4612	360	4653
1	004f23b2d	1	complex64	360	4546	360	4630
2	00a6db666	1	complex64	360	4525	360	4598
3	00f36a6ac	1	complex64	360	4609	360	4566
4	010a387db	1	complex64	360	4624	360	4527

In [53]:
```
labels_df[['H_height', 'L_height']].value_counts().to_frame(name='Count')
```

Out[53]:

		Count
H_height	L_height	
360	360	603

In [54]:
```
# Create target directories
!rm -rf train_samples
!mkdir -p train_samples/{x,target}
!ls -l train_samples
```

```
total 8
drwxr-xr-x 2 root root 4096 Oct 14 17:25 target
drwxr-xr-x 2 root root 4096 Oct 14 17:25 x
```

In [55]:

```
# This large function actually generates the 360x360 patches
def get_train_stats():
    c = 0
    # Loop over all training samples
    for row_idx, row in tqdm(labels_df.iterrows(), total=N_SAMPLES):
        train_id = row['id']
        # Skip non-complex64 samples
        if row['dtype'] != 'complex64':
            continue

        # Read SFTs as numpy arrays
        with h5py.File(f'{TRAIN_PATH}/{train_id}.hdf5', 'r') as file:
            SFT_H = np.array(file[train_id]['H1']['SFTs'])
            SFT_L = np.array(file[train_id]['L1']['SFTs'])

        # Split into real and imaginary part
        SFT_H_SPLIT = SFT_H.view(np.float32).reshape([*SFT_H.shape, 2])
        SFT_L_SPLIT = SFT_L.view(np.float32).reshape([*SFT_L.shape, 2])
        # Transpose to get channel(real/imaginary) first
        SFT_H_SPLIT = np.transpose(SFT_H_SPLIT, [2,0,1])
        SFT_L_SPLIT = np.transpose(SFT_L_SPLIT, [2,0,1])

        # Create target array
        N = min(row['H_width'], row['L_width']) // TARGET_HEIGHT
        x = np.zeros(shape=[N, len(INPUTS), TARGET_HEIGHT, TARGET_HEIGHT], dtype=np.float32)
        # Get patches
        for offset in range(N):
            x[offset, 0] = SFT_H_SPLIT[0, :, offset * TARGET_HEIGHT:(offset + 1) * TARGET_HEIGHT]
            x[offset, 1] = SFT_H_SPLIT[1, :, offset * TARGET_HEIGHT:(offset + 1) * TARGET_HEIGHT]
            x[offset, 2] = SFT_L_SPLIT[0, :, offset * TARGET_HEIGHT:(offset + 1) * TARGET_HEIGHT]
            x[offset, 3] = SFT_L_SPLIT[1,:, offset * TARGET_HEIGHT:(offset + 1) * TARGET_HEIGHT]



        # Save patches and target
        np.save(f'./train_samples/x/{c}.npy', x)
        np.save(f'./train_samples/target/{c}.npy', np.array(row['target']))
        c += 1

    return c

N_TRAIN_SAMPLES = get_train_stats()
print(f'N_TRAIN_SAMPLES: {N_TRAIN_SAMPLES}')
```

```
N_TRAIN_SAMPLES: 600
```

In [56]:

```
SAMPLE_IDXS = np.arange(len(glob.glob('./train_samples/target/*')))
print(f'SAMPLE_IDXS shape: {SAMPLE_IDXS.shape}')
```

```
SAMPLE_IDXS shape: (600,)
```

In [57]:

```
random.shuffle(SAMPLE_IDXS)
```

In [58]:
```
# Training dataset chooses a random 360x360 patch from a recording
def get_train_dataset(idxs, bs):
    while True:
        X = {
                'x_h_r': np.zeros(shape=[bs,TARGET_HEIGHT, TARGET_WIDTH], dtype=np.float32),
                'x_h_i': np.zeros(shape=[bs,TARGET_HEIGHT, TARGET_WIDTH], dtype=np.float32),
                'x_l_r': np.zeros(shape=[bs,TARGET_HEIGHT, TARGET_WIDTH], dtype=np.float32),
                'x_l_i': np.zeros(shape=[bs,TARGET_HEIGHT, TARGET_WIDTH], dtype=np.float32),
            }
        y = np.zeros(shape=[bs], dtype=np.int8)
        for i in range(bs):
            # Choose random file path
            index = np.random.choice(idxs, 1).squeeze()

            # Load x and choose random frame
            XX = np.load(f'./train_samples/x/{index}.npy')
            # Choose random patch
            patch_index = np.random.choice(len(XX), 1).squeeze()
            X['x_h_r'][i] = XX[patch_index, 0]
            X['x_h_i'][i] = XX[patch_index, 1]
            X['x_l_r'][i] = XX[patch_index, 2]
            X['x_l_i'][i] = XX[patch_index, 3]

            # Load target
            y[i] = np.load(f'./train_samples/target/{index}.npy')

        yield X, y
```

In [59]:
```
# Train batch statistics
def train_dataset_test():
    train_dataset = get_train_dataset(SAMPLE_IDXS, 64)
    X, y = next(train_dataset)
    for k in INPUTS:
        print(f'X {k} shape: {X[k].shape}, dtype: {X[k].dtype}', end=', ')
        print(f'X {k} mean: {X[k].mean():.2E}, std: {X[k].std():.2f}, min: {X[k].min():.2f}, max: {X[k].max
():.2f}')
    print(f'y: {y}')

train_dataset_test()
```

```
X x_h_r shape: (64, 360, 360), dtype: float32, X x_h_r mean: 1.41E-26, std: 0.00, min: -0.00, max: 0.00
X x_h_i shape: (64, 360, 360), dtype: float32, X x_h_i mean: 2.10E-26, std: 0.00, min: -0.00, max: 0.00
X x_l_r shape: (64, 360, 360), dtype: float32, X x_l_r mean: 2.75E-26, std: 0.00, min: -0.00, max: 0.00
X x_l_i shape: (64, 360, 360), dtype: float32, X x_l_i mean: -1.81E-27, std: 0.00, min: -0.00, max: 0.00
y: [0 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1
 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0 1 0 1]
```

Setting up general Parameters

In [60]:
```
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 32
num_epochs = 30
image_size = 360  # We'll resize input images to this size
patch_size = 40  # Size of the patches to be extract from the input images
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
transformer_units = [
    projection_dim * 2,
    projection_dim,
]  # Size of the transformer layers
transformer_layers = 8
mlp_head_units = [2048, 1024]  # Size of the dense layers of the final classifier
```

Implement MLP

In [61]:
```
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

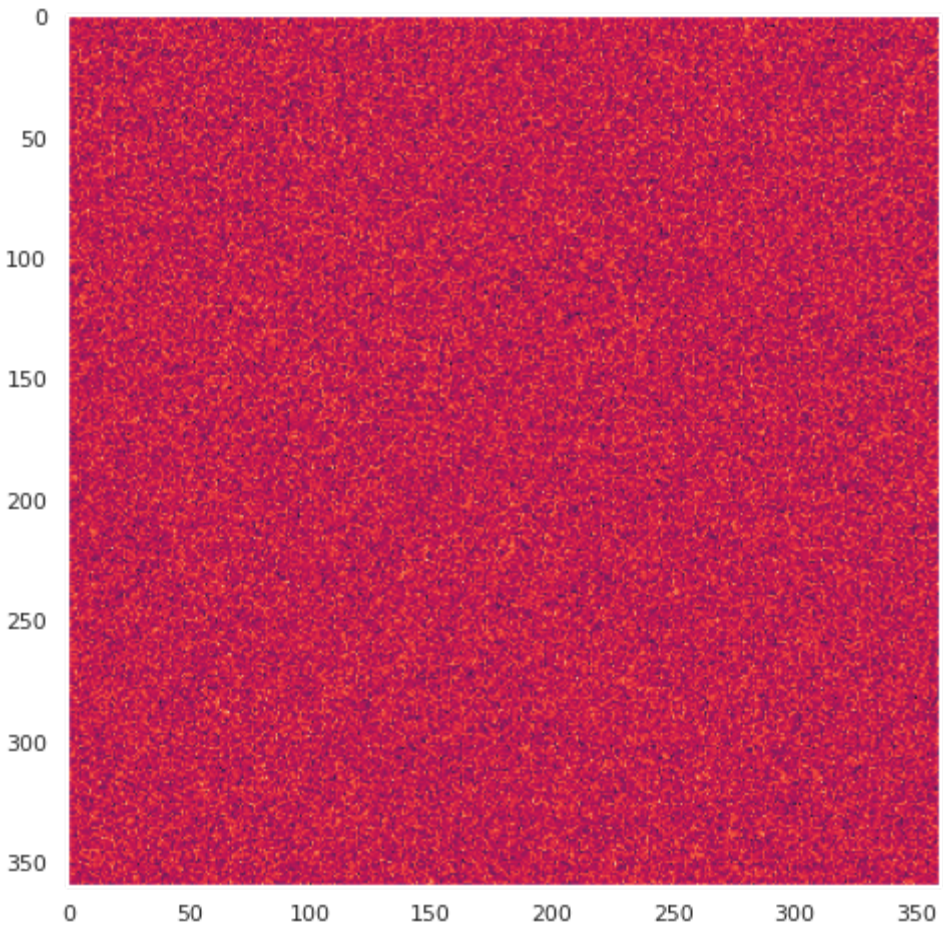
Implement patch creation as a layer

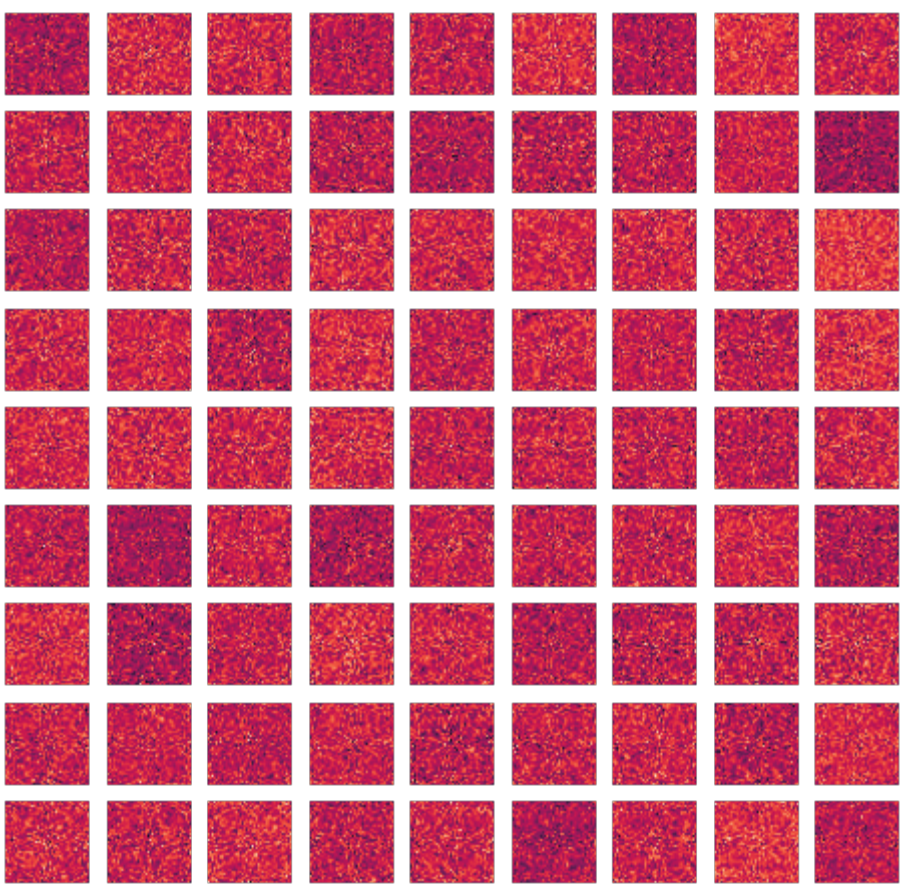
In [62]:
```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        patch_dims = patches.shape[-1]
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
```

In [63]:

```
plt.figure(figsize=(8, 8))
images = np.load(f'./train_samples/x/{np.random.choice(SAMPLE_IDXS, 1)[0]}.npy')[np.random.choice(len(np.lo
ad(f'./train_samples/x/{np.random.choice(SAMPLE_IDXS, 1)[0]}.npy')), 1).squeeze()]
image = images[0]
plt.grid(False)
plt.imshow(image)

img = tf.reshape(image, (1, 360, 360, 1))
resized_image = tf.image.resize(
    tf.convert_to_tensor(img), size=(image_size, image_size)
)
print(resized_image.shape)
patches = Patches(patch_size)(resized_image)
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"Patches per image: {patches.shape[1]}")
print(f"Elements per patch: {patches.shape[-1]}")
n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(8, 8))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 1))
    plt.imshow(patch_img.numpy())
    plt.axis("off")
```

```
(1, 360, 360, 1)
Image size: 360 X 360
Patch size: 40 X 40
Patches per image: 81
Elements per patch: 1600
```





Creating the patch encoder

The PatchEncoder layer will linearly transform a patch by projecting it into a vector of size projection_dim. In addition, it adds a learnable position embedding to the projected vector.

In [64]:
```
class PatchEncoder(tf.keras.layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super(PatchEncoder, self).__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions)
        return encoded
```

Building the ViT

The ViT model consists of multiple Transformer blocks, which use the layers.MultiHeadAttention layer as a self-attention mechanism applied to the sequence of patches. The Transformer blocks produce a [batch_size, num_patches, projection_dim] tensor, which is processed via a Dense head to produce the final output.

Unlike the technique described in the paper, which prepends a learnable embedding to the sequence of encoded patches to serve as the image representation, all the outputs of the final Transformer block are reshaped with layers.Flatten() and used as the image representation input to the classifier head. Note that the layers.GlobalAveragePooling1D layer could also be used instead to aggregate the outputs of the Transformer block, especially when the number of patches and the projection dimensions are large.

In [65]:

```
def create_vit_classifier(x_input):

    inputs = tf.expand_dims(x_input, axis=-1)
    inputs = tf.tile(inputs, [1,1,1,3])
    # Augment data.
    # Create patches.
    patches = Patches(patch_size)(inputs)
    # Encode patches.
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP.
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.5)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate=0.5)
    # Classify outputs.
    x = layers.Dense(256, activation='relu')(features)
    # Create the Keras model.
    return x
```

Building CNN based Model

In [66]:

```
arch = tf.keras.Sequential([

        tf.keras.layers.Conv2D(filters=64,kernel_size=(5,5),strides=(2,2), activation="relu", padding="vali
d",input_shape=(image_size,image_size,3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(filters=32,kernel_size=(5,5),strides=(2,2), activation="relu", padding="vali
d"),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Conv2D(filters=64,kernel_size=(3,3),strides=(2,2), activation="relu", padding="vali
d",input_shape=(image_size,image_size,3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),strides=(2,2), activation="relu", padding="vali
d"),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(units=512, activation='relu',
                              kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3),
                              bias_regularizer=regularizers.L2(1e-2),
                              activity_regularizer=regularizers.L2(1e-3)),
])
def create_cnn(x_input):

    inputs = tf.expand_dims(x_input, axis=-1)
    inputs = tf.tile(inputs, [1,1,1,3])
    cnn = tf.keras.Sequential([

        tf.keras.layers.Conv2D(filters=64,kernel_size=(5,5),strides=(2,2), activation="relu", padding="vali
d",input_shape=(image_size,image_size,3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(filters=32,kernel_size=(5,5),strides=(2,2), activation="relu", padding="vali
d"),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Conv2D(filters=64,kernel_size=(3,3),strides=(2,2), activation="relu", padding="vali
d",input_shape=(image_size,image_size,3)),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Conv2D(filters=32,kernel_size=(3,3),strides=(2,2), activation="relu", padding="vali
d"),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(units=512, activation='relu',
                              kernel_regularizer=regularizers.L1L2(l1=1e-3, l2=1e-3),
                              bias_regularizer=regularizers.L2(1e-2),
                              activity_regularizer=regularizers.L2(1e-3)),
    ])
    features = cnn(inputs)
    x = layers.Dense(256, activation='relu')(features)
    return x
```

Functions to get the vit and cnn models

In [67]:

```
def get_vit_model():
    # 1) Hanford Real
    h_r_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_h_r')
    # 2) Hanford imaginary
    h_i_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_h_i')
    # 3) Livingston Real
    l_r_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_l_r')
    # 4) Livingston Imaginary
    l_i_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_l_i')

    # Get embedding from vit
    h_r_embed = create_vit_classifier(h_r_input)
    h_i_embed = create_vit_classifier(h_i_input)
    l_r_embed = create_vit_classifier(l_r_input)
    l_i_embed = create_vit_classifier(l_i_input)

    # Concatenate embeddings
    x = tf.keras.layers.Concatenate()([h_r_embed, h_i_embed, l_r_embed, l_i_embed])
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(128)(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    # Target prediction in range [0,1] with sigmoid activation
    output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

    # Model
    inputs = [h_r_input, h_i_input, l_r_input, l_i_input]
    model = tf.keras.models.Model(inputs=inputs, outputs=output)

    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    model.compile(
        optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics = [
            tf.keras.metrics.AUC(),
        ]
    )
    return model


def get_cnn_model():
    # 1) Hanford Real
    h_r_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_h_r')
    # 2) Hanford imaginary
    h_i_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_h_i')
    # 3) Livingston Real
    l_r_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_l_r')
    # 4) Livingston Imaginary
    l_i_input = tf.keras.layers.Input(shape=[TARGET_HEIGHT, TARGET_WIDTH], dtype=tf.float32, name='x_l_i')

    # Get embedding from cnn
    h_r_embed = create_cnn(h_r_input)
    h_i_embed = create_cnn(h_i_input)
    l_r_embed = create_cnn(l_r_input)
    l_i_embed = create_cnn(l_i_input)

    # Concatenate embeddings
    x = tf.keras.layers.Concatenate()([h_r_embed, h_i_embed, l_r_embed, l_i_embed])
    x = tf.keras.layers.Dropout(0.5)(x)
```

```
    x = tf.keras.layers.Dense(128)(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    # Target prediction in range [0,1] with sigmoid activation
    output = tf.keras.layers.Dense(1, activation='sigmoid')(x)

    # Model
    inputs = [h_r_input, h_i_input, l_r_input, l_i_input]
    model = tf.keras.models.Model(inputs=inputs, outputs=output)

    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    model.compile(
        optimizer=optimizer,
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics = [
            tf.keras.metrics.AUC(),
        ]
    )
    return model
```

Exploring ViT in action

In [68]:

```
tf.keras.backend.clear_session()
gc.collect()
model = get_vit_model()
```

In [69]:
```
print(model.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
x_h_r (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_h_i (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_l_r (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_l_i (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
tf.expand_dims (TFOpLambda)     (None, 360, 360, 1)  0           x_h_r[0][0]
__________________________________________________________________________________________________
tf.expand_dims_1 (TFOpLambda)   (None, 360, 360, 1)  0           x_h_i[0][0]
__________________________________________________________________________________________________
tf.expand_dims_2 (TFOpLambda)   (None, 360, 360, 1)  0           x_l_r[0][0]
__________________________________________________________________________________________________
tf.expand_dims_3 (TFOpLambda)   (None, 360, 360, 1)  0           x_l_i[0][0]
__________________________________________________________________________________________________
tf.tile (TFOpLambda)            (None, 360, 360, 3)  0           tf.expand_dims[0][0]
__________________________________________________________________________________________________
tf.tile_1 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_1[0][0]
__________________________________________________________________________________________________
tf.tile_2 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_2[0][0]
__________________________________________________________________________________________________
tf.tile_3 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_3[0][0]
__________________________________________________________________________________________________
patches (Patches)               (None, None, 4800)   0           tf.tile[0][0]
__________________________________________________________________________________________________
patches_1 (Patches)             (None, None, 4800)   0           tf.tile_1[0][0]
__________________________________________________________________________________________________
patches_2 (Patches)             (None, None, 4800)   0           tf.tile_2[0][0]
__________________________________________________________________________________________________
patches_3 (Patches)             (None, None, 4800)   0           tf.tile_3[0][0]
__________________________________________________________________________________________________
patch_encoder (PatchEncoder)    (None, 81, 64)       312448      patches[0][0]
__________________________________________________________________________________________________
patch_encoder_1 (PatchEncoder)  (None, 81, 64)       312448      patches_1[0][0]
__________________________________________________________________________________________________
patch_encoder_2 (PatchEncoder)  (None, 81, 64)       312448      patches_2[0][0]
__________________________________________________________________________________________________
patch_encoder_3 (PatchEncoder)  (None, 81, 64)       312448      patches_3[0][0]
__________________________________________________________________________________________________
layer_normalization (LayerNorma (None, 81, 64)       128         patch_encoder[0][0]
__________________________________________________________________________________________________
layer_normalization_17 (LayerNo (None, 81, 64)       128         patch_encoder_1[0][0]
__________________________________________________________________________________________________
layer_normalization_34 (LayerNo (None, 81, 64)       128         patch_encoder_2[0][0]
__________________________________________________________________________________________________
layer_normalization_51 (LayerNo (None, 81, 64)       128         patch_encoder_3[0][0]
__________________________________________________________________________________________________
multi_head_attention (MultiHead (None, 81, 64)       66368       layer_normalization[0][0]
                                                                 layer_normalization[0][0]
__________________________________________________________________________________________________
multi_head_attention_8 (MultiHe (None, 81, 64)       66368       layer_normalization_17[0][0]
                                                                 layer_normalization_17[0][0]
__________________________________________________________________________________________________
multi_head_attention_16 (MultiH (None, 81, 64)       66368       layer_normalization_34[0][0]
                                                                 layer_normalization_34[0][0]
__________________________________________________________________________________________________
```

multi_head_attention_24 (MultiH	(None, 81, 64)	66368	layer_normalization_51[0][0] layer_normalization_51[0][0]
add (Add)	(None, 81, 64)	0	multi_head_attention[0][0] patch_encoder[0][0]
add_16 (Add)	(None, 81, 64)	0	multi_head_attention_8[0][0] patch_encoder_1[0][0]
add_32 (Add)	(None, 81, 64)	0	multi_head_attention_16[0][0] patch_encoder_2[0][0]
add_48 (Add)	(None, 81, 64)	0	multi_head_attention_24[0][0] patch_encoder_3[0][0]
layer_normalization_1 (LayerNor	(None, 81, 64)	128	add[0][0]
layer_normalization_18 (LayerNo	(None, 81, 64)	128	add_16[0][0]
layer_normalization_35 (LayerNo	(None, 81, 64)	128	add_32[0][0]
layer_normalization_52 (LayerNo	(None, 81, 64)	128	add_48[0][0]
dense_1 (Dense)	(None, 81, 128)	8320	layer_normalization_1[0][0]
dense_21 (Dense)	(None, 81, 128)	8320	layer_normalization_18[0][0]
dense_41 (Dense)	(None, 81, 128)	8320	layer_normalization_35[0][0]
dense_61 (Dense)	(None, 81, 128)	8320	layer_normalization_52[0][0]
dropout (Dropout)	(None, 81, 128)	0	dense_1[0][0]
dropout_19 (Dropout)	(None, 81, 128)	0	dense_21[0][0]
dropout_38 (Dropout)	(None, 81, 128)	0	dense_41[0][0]
dropout_57 (Dropout)	(None, 81, 128)	0	dense_61[0][0]
dense_2 (Dense)	(None, 81, 64)	8256	dropout[0][0]
dense_22 (Dense)	(None, 81, 64)	8256	dropout_19[0][0]
dense_42 (Dense)	(None, 81, 64)	8256	dropout_38[0][0]
dense_62 (Dense)	(None, 81, 64)	8256	dropout_57[0][0]
dropout_1 (Dropout)	(None, 81, 64)	0	dense_2[0][0]
dropout_20 (Dropout)	(None, 81, 64)	0	dense_22[0][0]
dropout_39 (Dropout)	(None, 81, 64)	0	dense_42[0][0]
dropout_58 (Dropout)	(None, 81, 64)	0	dense_62[0][0]
add_1 (Add)	(None, 81, 64)	0	dropout_1[0][0] add[0][0]
add_17 (Add)	(None, 81, 64)	0	dropout_20[0][0] add_16[0][0]
add_33 (Add)	(None, 81, 64)	0	dropout_39[0][0]

```
                                                                  add_32[0][0]
__________________________________________________________________________________________________
add_49 (Add)                    (None, 81, 64)       0           dropout_58[0][0]
                                                                  add_48[0][0]
__________________________________________________________________________________________________
layer_normalization_2 (LayerNor (None, 81, 64)       128         add_1[0][0]
__________________________________________________________________________________________________
layer_normalization_19 (LayerNo (None, 81, 64)       128         add_17[0][0]
__________________________________________________________________________________________________
layer_normalization_36 (LayerNo (None, 81, 64)       128         add_33[0][0]
__________________________________________________________________________________________________
layer_normalization_53 (LayerNo (None, 81, 64)       128         add_49[0][0]
__________________________________________________________________________________________________
multi_head_attention_1 (MultiHe (None, 81, 64)       66368       layer_normalization_2[0][0]
                                                                  layer_normalization_2[0][0]
__________________________________________________________________________________________________
multi_head_attention_9 (MultiHe (None, 81, 64)       66368       layer_normalization_19[0][0]
                                                                  layer_normalization_19[0][0]
__________________________________________________________________________________________________
multi_head_attention_17 (MultiH (None, 81, 64)       66368       layer_normalization_36[0][0]
                                                                  layer_normalization_36[0][0]
__________________________________________________________________________________________________
multi_head_attention_25 (MultiH (None, 81, 64)       66368       layer_normalization_53[0][0]
                                                                  layer_normalization_53[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 81, 64)       0           multi_head_attention_1[0][0]
                                                                  add_1[0][0]
__________________________________________________________________________________________________
add_18 (Add)                    (None, 81, 64)       0           multi_head_attention_9[0][0]
                                                                  add_17[0][0]
__________________________________________________________________________________________________
add_34 (Add)                    (None, 81, 64)       0           multi_head_attention_17[0][0]
                                                                  add_33[0][0]
__________________________________________________________________________________________________
add_50 (Add)                    (None, 81, 64)       0           multi_head_attention_25[0][0]
                                                                  add_49[0][0]
__________________________________________________________________________________________________
layer_normalization_3 (LayerNor (None, 81, 64)       128         add_2[0][0]
__________________________________________________________________________________________________
layer_normalization_20 (LayerNo (None, 81, 64)       128         add_18[0][0]
__________________________________________________________________________________________________
layer_normalization_37 (LayerNo (None, 81, 64)       128         add_34[0][0]
__________________________________________________________________________________________________
layer_normalization_54 (LayerNo (None, 81, 64)       128         add_50[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 81, 128)      8320        layer_normalization_3[0][0]
__________________________________________________________________________________________________
dense_23 (Dense)                (None, 81, 128)      8320        layer_normalization_20[0][0]
__________________________________________________________________________________________________
dense_43 (Dense)                (None, 81, 128)      8320        layer_normalization_37[0][0]
__________________________________________________________________________________________________
dense_63 (Dense)                (None, 81, 128)      8320        layer_normalization_54[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 81, 128)      0           dense_3[0][0]
__________________________________________________________________________________________________
dropout_21 (Dropout)            (None, 81, 128)      0           dense_23[0][0]
__________________________________________________________________________________________________
dropout_40 (Dropout)            (None, 81, 128)      0           dense_43[0][0]
__________________________________________________________________________________________________
dropout_59 (Dropout)            (None, 81, 128)      0           dense_63[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 81, 64)       8256        dropout_2[0][0]
```

dense_24 (Dense)	(None, 81, 64)	8256	dropout_21[0][0]
dense_44 (Dense)	(None, 81, 64)	8256	dropout_40[0][0]
dense_64 (Dense)	(None, 81, 64)	8256	dropout_59[0][0]
dropout_3 (Dropout)	(None, 81, 64)	0	dense_4[0][0]
dropout_22 (Dropout)	(None, 81, 64)	0	dense_24[0][0]
dropout_41 (Dropout)	(None, 81, 64)	0	dense_44[0][0]
dropout_60 (Dropout)	(None, 81, 64)	0	dense_64[0][0]
add_3 (Add)	(None, 81, 64)	0	dropout_3[0][0] add_2[0][0]
add_19 (Add)	(None, 81, 64)	0	dropout_22[0][0] add_18[0][0]
add_35 (Add)	(None, 81, 64)	0	dropout_41[0][0] add_34[0][0]
add_51 (Add)	(None, 81, 64)	0	dropout_60[0][0] add_50[0][0]
layer_normalization_4 (LayerNor	(None, 81, 64)	128	add_3[0][0]
layer_normalization_21 (LayerNo	(None, 81, 64)	128	add_19[0][0]
layer_normalization_38 (LayerNo	(None, 81, 64)	128	add_35[0][0]
layer_normalization_55 (LayerNo	(None, 81, 64)	128	add_51[0][0]
multi_head_attention_2 (MultiHe	(None, 81, 64)	66368	layer_normalization_4[0][0] layer_normalization_4[0][0]
multi_head_attention_10 (MultiH	(None, 81, 64)	66368	layer_normalization_21[0][0] layer_normalization_21[0][0]
multi_head_attention_18 (MultiH	(None, 81, 64)	66368	layer_normalization_38[0][0] layer_normalization_38[0][0]
multi_head_attention_26 (MultiH	(None, 81, 64)	66368	layer_normalization_55[0][0] layer_normalization_55[0][0]
add_4 (Add)	(None, 81, 64)	0	multi_head_attention_2[0][0] add_3[0][0]
add_20 (Add)	(None, 81, 64)	0	multi_head_attention_10[0][0] add_19[0][0]
add_36 (Add)	(None, 81, 64)	0	multi_head_attention_18[0][0] add_35[0][0]
add_52 (Add)	(None, 81, 64)	0	multi_head_attention_26[0][0] add_51[0][0]
layer_normalization_5 (LayerNor	(None, 81, 64)	128	add_4[0][0]
layer_normalization_22 (LayerNo	(None, 81, 64)	128	add_20[0][0]

layer_normalization_39 (LayerNo	(None, 81, 64)	128	add_36[0][0]
layer_normalization_56 (LayerNo	(None, 81, 64)	128	add_52[0][0]
dense_5 (Dense)	(None, 81, 128)	8320	layer_normalization_5[0][0]
dense_25 (Dense)	(None, 81, 128)	8320	layer_normalization_22[0][0]
dense_45 (Dense)	(None, 81, 128)	8320	layer_normalization_39[0][0]
dense_65 (Dense)	(None, 81, 128)	8320	layer_normalization_56[0][0]
dropout_4 (Dropout)	(None, 81, 128)	0	dense_5[0][0]
dropout_23 (Dropout)	(None, 81, 128)	0	dense_25[0][0]
dropout_42 (Dropout)	(None, 81, 128)	0	dense_45[0][0]
dropout_61 (Dropout)	(None, 81, 128)	0	dense_65[0][0]
dense_6 (Dense)	(None, 81, 64)	8256	dropout_4[0][0]
dense_26 (Dense)	(None, 81, 64)	8256	dropout_23[0][0]
dense_46 (Dense)	(None, 81, 64)	8256	dropout_42[0][0]
dense_66 (Dense)	(None, 81, 64)	8256	dropout_61[0][0]
dropout_5 (Dropout)	(None, 81, 64)	0	dense_6[0][0]
dropout_24 (Dropout)	(None, 81, 64)	0	dense_26[0][0]
dropout_43 (Dropout)	(None, 81, 64)	0	dense_46[0][0]
dropout_62 (Dropout)	(None, 81, 64)	0	dense_66[0][0]
add_5 (Add)	(None, 81, 64)	0	dropout_5[0][0] add_4[0][0]
add_21 (Add)	(None, 81, 64)	0	dropout_24[0][0] add_20[0][0]
add_37 (Add)	(None, 81, 64)	0	dropout_43[0][0] add_36[0][0]
add_53 (Add)	(None, 81, 64)	0	dropout_62[0][0] add_52[0][0]
layer_normalization_6 (LayerNor	(None, 81, 64)	128	add_5[0][0]
layer_normalization_23 (LayerNo	(None, 81, 64)	128	add_21[0][0]
layer_normalization_40 (LayerNo	(None, 81, 64)	128	add_37[0][0]
layer_normalization_57 (LayerNo	(None, 81, 64)	128	add_53[0][0]
multi_head_attention_3 (MultiHe	(None, 81, 64)	66368	layer_normalization_6[0][0] layer_normalization_6[0][0]
multi_head_attention_11 (MultiH	(None, 81, 64)	66368	layer_normalization_23[0][0] layer_normalization_23[0][0]

multi_head_attention_19 (MultiH	(None, 81, 64)	66368	layer_normalization_40[0][0] layer_normalization_40[0][0]
multi_head_attention_27 (MultiH	(None, 81, 64)	66368	layer_normalization_57[0][0] layer_normalization_57[0][0]
add_6 (Add)	(None, 81, 64)	0	multi_head_attention_3[0][0] add_5[0][0]
add_22 (Add)	(None, 81, 64)	0	multi_head_attention_11[0][0] add_21[0][0]
add_38 (Add)	(None, 81, 64)	0	multi_head_attention_19[0][0] add_37[0][0]
add_54 (Add)	(None, 81, 64)	0	multi_head_attention_27[0][0] add_53[0][0]
layer_normalization_7 (LayerNor	(None, 81, 64)	128	add_6[0][0]
layer_normalization_24 (LayerNo	(None, 81, 64)	128	add_22[0][0]
layer_normalization_41 (LayerNo	(None, 81, 64)	128	add_38[0][0]
layer_normalization_58 (LayerNo	(None, 81, 64)	128	add_54[0][0]
dense_7 (Dense)	(None, 81, 128)	8320	layer_normalization_7[0][0]
dense_27 (Dense)	(None, 81, 128)	8320	layer_normalization_24[0][0]
dense_47 (Dense)	(None, 81, 128)	8320	layer_normalization_41[0][0]
dense_67 (Dense)	(None, 81, 128)	8320	layer_normalization_58[0][0]
dropout_6 (Dropout)	(None, 81, 128)	0	dense_7[0][0]
dropout_25 (Dropout)	(None, 81, 128)	0	dense_27[0][0]
dropout_44 (Dropout)	(None, 81, 128)	0	dense_47[0][0]
dropout_63 (Dropout)	(None, 81, 128)	0	dense_67[0][0]
dense_8 (Dense)	(None, 81, 64)	8256	dropout_6[0][0]
dense_28 (Dense)	(None, 81, 64)	8256	dropout_25[0][0]
dense_48 (Dense)	(None, 81, 64)	8256	dropout_44[0][0]
dense_68 (Dense)	(None, 81, 64)	8256	dropout_63[0][0]
dropout_7 (Dropout)	(None, 81, 64)	0	dense_8[0][0]
dropout_26 (Dropout)	(None, 81, 64)	0	dense_28[0][0]
dropout_45 (Dropout)	(None, 81, 64)	0	dense_48[0][0]
dropout_64 (Dropout)	(None, 81, 64)	0	dense_68[0][0]
add_7 (Add)	(None, 81, 64)	0	dropout_7[0][0] add_6[0][0]

```
add_23 (Add)                    (None, 81, 64)       0           dropout_26[0][0]
                                                                 add_22[0][0]
__________________________________________________________________________________________________
add_39 (Add)                    (None, 81, 64)       0           dropout_45[0][0]
                                                                 add_38[0][0]
__________________________________________________________________________________________________
add_55 (Add)                    (None, 81, 64)       0           dropout_64[0][0]
                                                                 add_54[0][0]
__________________________________________________________________________________________________
layer_normalization_8 (LayerNor (None, 81, 64)       128         add_7[0][0]
__________________________________________________________________________________________________
layer_normalization_25 (LayerNo (None, 81, 64)       128         add_23[0][0]
__________________________________________________________________________________________________
layer_normalization_42 (LayerNo (None, 81, 64)       128         add_39[0][0]
__________________________________________________________________________________________________
layer_normalization_59 (LayerNo (None, 81, 64)       128         add_55[0][0]
__________________________________________________________________________________________________
multi_head_attention_4 (MultiHe (None, 81, 64)       66368       layer_normalization_8[0][0]
                                                                 layer_normalization_8[0][0]
__________________________________________________________________________________________________
multi_head_attention_12 (MultiH (None, 81, 64)       66368       layer_normalization_25[0][0]
                                                                 layer_normalization_25[0][0]
__________________________________________________________________________________________________
multi_head_attention_20 (MultiH (None, 81, 64)       66368       layer_normalization_42[0][0]
                                                                 layer_normalization_42[0][0]
__________________________________________________________________________________________________
multi_head_attention_28 (MultiH (None, 81, 64)       66368       layer_normalization_59[0][0]
                                                                 layer_normalization_59[0][0]
__________________________________________________________________________________________________
add_8 (Add)                     (None, 81, 64)       0           multi_head_attention_4[0][0]
                                                                 add_7[0][0]
__________________________________________________________________________________________________
add_24 (Add)                    (None, 81, 64)       0           multi_head_attention_12[0][0]
                                                                 add_23[0][0]
__________________________________________________________________________________________________
add_40 (Add)                    (None, 81, 64)       0           multi_head_attention_20[0][0]
                                                                 add_39[0][0]
__________________________________________________________________________________________________
add_56 (Add)                    (None, 81, 64)       0           multi_head_attention_28[0][0]
                                                                 add_55[0][0]
__________________________________________________________________________________________________
layer_normalization_9 (LayerNor (None, 81, 64)       128         add_8[0][0]
__________________________________________________________________________________________________
layer_normalization_26 (LayerNo (None, 81, 64)       128         add_24[0][0]
__________________________________________________________________________________________________
layer_normalization_43 (LayerNo (None, 81, 64)       128         add_40[0][0]
__________________________________________________________________________________________________
layer_normalization_60 (LayerNo (None, 81, 64)       128         add_56[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 81, 128)      8320        layer_normalization_9[0][0]
__________________________________________________________________________________________________
dense_29 (Dense)                (None, 81, 128)      8320        layer_normalization_26[0][0]
__________________________________________________________________________________________________
dense_49 (Dense)                (None, 81, 128)      8320        layer_normalization_43[0][0]
__________________________________________________________________________________________________
dense_69 (Dense)                (None, 81, 128)      8320        layer_normalization_60[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 81, 128)      0           dense_9[0][0]
__________________________________________________________________________________________________
dropout_27 (Dropout)            (None, 81, 128)      0           dense_29[0][0]
__________________________________________________________________________________________________
dropout_46 (Dropout)            (None, 81, 128)      0           dense_49[0][0]
```

dropout_65 (Dropout)	(None, 81, 128)	0	dense_69[0][0]
dense_10 (Dense)	(None, 81, 64)	8256	dropout_8[0][0]
dense_30 (Dense)	(None, 81, 64)	8256	dropout_27[0][0]
dense_50 (Dense)	(None, 81, 64)	8256	dropout_46[0][0]
dense_70 (Dense)	(None, 81, 64)	8256	dropout_65[0][0]
dropout_9 (Dropout)	(None, 81, 64)	0	dense_10[0][0]
dropout_28 (Dropout)	(None, 81, 64)	0	dense_30[0][0]
dropout_47 (Dropout)	(None, 81, 64)	0	dense_50[0][0]
dropout_66 (Dropout)	(None, 81, 64)	0	dense_70[0][0]
add_9 (Add)	(None, 81, 64)	0	dropout_9[0][0] add_8[0][0]
add_25 (Add)	(None, 81, 64)	0	dropout_28[0][0] add_24[0][0]
add_41 (Add)	(None, 81, 64)	0	dropout_47[0][0] add_40[0][0]
add_57 (Add)	(None, 81, 64)	0	dropout_66[0][0] add_56[0][0]
layer_normalization_10 (LayerNo	(None, 81, 64)	128	add_9[0][0]
layer_normalization_27 (LayerNo	(None, 81, 64)	128	add_25[0][0]
layer_normalization_44 (LayerNo	(None, 81, 64)	128	add_41[0][0]
layer_normalization_61 (LayerNo	(None, 81, 64)	128	add_57[0][0]
multi_head_attention_5 (MultiHe	(None, 81, 64)	66368	layer_normalization_10[0][0] layer_normalization_10[0][0]
multi_head_attention_13 (MultiH	(None, 81, 64)	66368	layer_normalization_27[0][0] layer_normalization_27[0][0]
multi_head_attention_21 (MultiH	(None, 81, 64)	66368	layer_normalization_44[0][0] layer_normalization_44[0][0]
multi_head_attention_29 (MultiH	(None, 81, 64)	66368	layer_normalization_61[0][0] layer_normalization_61[0][0]
add_10 (Add)	(None, 81, 64)	0	multi_head_attention_5[0][0] add_9[0][0]
add_26 (Add)	(None, 81, 64)	0	multi_head_attention_13[0][0] add_25[0][0]
add_42 (Add)	(None, 81, 64)	0	multi_head_attention_21[0][0] add_41[0][0]
add_58 (Add)	(None, 81, 64)	0	multi_head_attention_29[0][0] add_57[0][0]

layer_normalization_11 (LayerNo	(None, 81, 64)	128	add_10[0][0]
layer_normalization_28 (LayerNo	(None, 81, 64)	128	add_26[0][0]
layer_normalization_45 (LayerNo	(None, 81, 64)	128	add_42[0][0]
layer_normalization_62 (LayerNo	(None, 81, 64)	128	add_58[0][0]
dense_11 (Dense)	(None, 81, 128)	8320	layer_normalization_11[0][0]
dense_31 (Dense)	(None, 81, 128)	8320	layer_normalization_28[0][0]
dense_51 (Dense)	(None, 81, 128)	8320	layer_normalization_45[0][0]
dense_71 (Dense)	(None, 81, 128)	8320	layer_normalization_62[0][0]
dropout_10 (Dropout)	(None, 81, 128)	0	dense_11[0][0]
dropout_29 (Dropout)	(None, 81, 128)	0	dense_31[0][0]
dropout_48 (Dropout)	(None, 81, 128)	0	dense_51[0][0]
dropout_67 (Dropout)	(None, 81, 128)	0	dense_71[0][0]
dense_12 (Dense)	(None, 81, 64)	8256	dropout_10[0][0]
dense_32 (Dense)	(None, 81, 64)	8256	dropout_29[0][0]
dense_52 (Dense)	(None, 81, 64)	8256	dropout_48[0][0]
dense_72 (Dense)	(None, 81, 64)	8256	dropout_67[0][0]
dropout_11 (Dropout)	(None, 81, 64)	0	dense_12[0][0]
dropout_30 (Dropout)	(None, 81, 64)	0	dense_32[0][0]
dropout_49 (Dropout)	(None, 81, 64)	0	dense_52[0][0]
dropout_68 (Dropout)	(None, 81, 64)	0	dense_72[0][0]
add_11 (Add)	(None, 81, 64)	0	dropout_11[0][0] add_10[0][0]
add_27 (Add)	(None, 81, 64)	0	dropout_30[0][0] add_26[0][0]
add_43 (Add)	(None, 81, 64)	0	dropout_49[0][0] add_42[0][0]
add_59 (Add)	(None, 81, 64)	0	dropout_68[0][0] add_58[0][0]
layer_normalization_12 (LayerNo	(None, 81, 64)	128	add_11[0][0]
layer_normalization_29 (LayerNo	(None, 81, 64)	128	add_27[0][0]
layer_normalization_46 (LayerNo	(None, 81, 64)	128	add_43[0][0]
layer_normalization_63 (LayerNo	(None, 81, 64)	128	add_59[0][0]
multi_head_attention_6 (MultiHe	(None, 81, 64)	66368	layer_normalization_12[0][0]

```
                                                                 layer_normalization_12[0][0]
__________________________________________________________________________________________________
multi_head_attention_14 (MultiH (None, 81, 64)       66368       layer_normalization_29[0][0]
                                                                 layer_normalization_29[0][0]
__________________________________________________________________________________________________
multi_head_attention_22 (MultiH (None, 81, 64)       66368       layer_normalization_46[0][0]
                                                                 layer_normalization_46[0][0]
__________________________________________________________________________________________________
multi_head_attention_30 (MultiH (None, 81, 64)       66368       layer_normalization_63[0][0]
                                                                 layer_normalization_63[0][0]
__________________________________________________________________________________________________
add_12 (Add)                    (None, 81, 64)       0           multi_head_attention_6[0][0]
                                                                 add_11[0][0]
__________________________________________________________________________________________________
add_28 (Add)                    (None, 81, 64)       0           multi_head_attention_14[0][0]
                                                                 add_27[0][0]
__________________________________________________________________________________________________
add_44 (Add)                    (None, 81, 64)       0           multi_head_attention_22[0][0]
                                                                 add_43[0][0]
__________________________________________________________________________________________________
add_60 (Add)                    (None, 81, 64)       0           multi_head_attention_30[0][0]
                                                                 add_59[0][0]
__________________________________________________________________________________________________
layer_normalization_13 (LayerNo (None, 81, 64)       128         add_12[0][0]
__________________________________________________________________________________________________
layer_normalization_30 (LayerNo (None, 81, 64)       128         add_28[0][0]
__________________________________________________________________________________________________
layer_normalization_47 (LayerNo (None, 81, 64)       128         add_44[0][0]
__________________________________________________________________________________________________
layer_normalization_64 (LayerNo (None, 81, 64)       128         add_60[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 81, 128)      8320        layer_normalization_13[0][0]
__________________________________________________________________________________________________
dense_33 (Dense)                (None, 81, 128)      8320        layer_normalization_30[0][0]
__________________________________________________________________________________________________
dense_53 (Dense)                (None, 81, 128)      8320        layer_normalization_47[0][0]
__________________________________________________________________________________________________
dense_73 (Dense)                (None, 81, 128)      8320        layer_normalization_64[0][0]
__________________________________________________________________________________________________
dropout_12 (Dropout)            (None, 81, 128)      0           dense_13[0][0]
__________________________________________________________________________________________________
dropout_31 (Dropout)            (None, 81, 128)      0           dense_33[0][0]
__________________________________________________________________________________________________
dropout_50 (Dropout)            (None, 81, 128)      0           dense_53[0][0]
__________________________________________________________________________________________________
dropout_69 (Dropout)            (None, 81, 128)      0           dense_73[0][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 81, 64)       8256        dropout_12[0][0]
__________________________________________________________________________________________________
dense_34 (Dense)                (None, 81, 64)       8256        dropout_31[0][0]
__________________________________________________________________________________________________
dense_54 (Dense)                (None, 81, 64)       8256        dropout_50[0][0]
__________________________________________________________________________________________________
dense_74 (Dense)                (None, 81, 64)       8256        dropout_69[0][0]
__________________________________________________________________________________________________
dropout_13 (Dropout)            (None, 81, 64)       0           dense_14[0][0]
__________________________________________________________________________________________________
dropout_32 (Dropout)            (None, 81, 64)       0           dense_34[0][0]
__________________________________________________________________________________________________
dropout_51 (Dropout)            (None, 81, 64)       0           dense_54[0][0]
__________________________________________________________________________________________________
dropout_70 (Dropout)            (None, 81, 64)       0           dense_74[0][0]
```

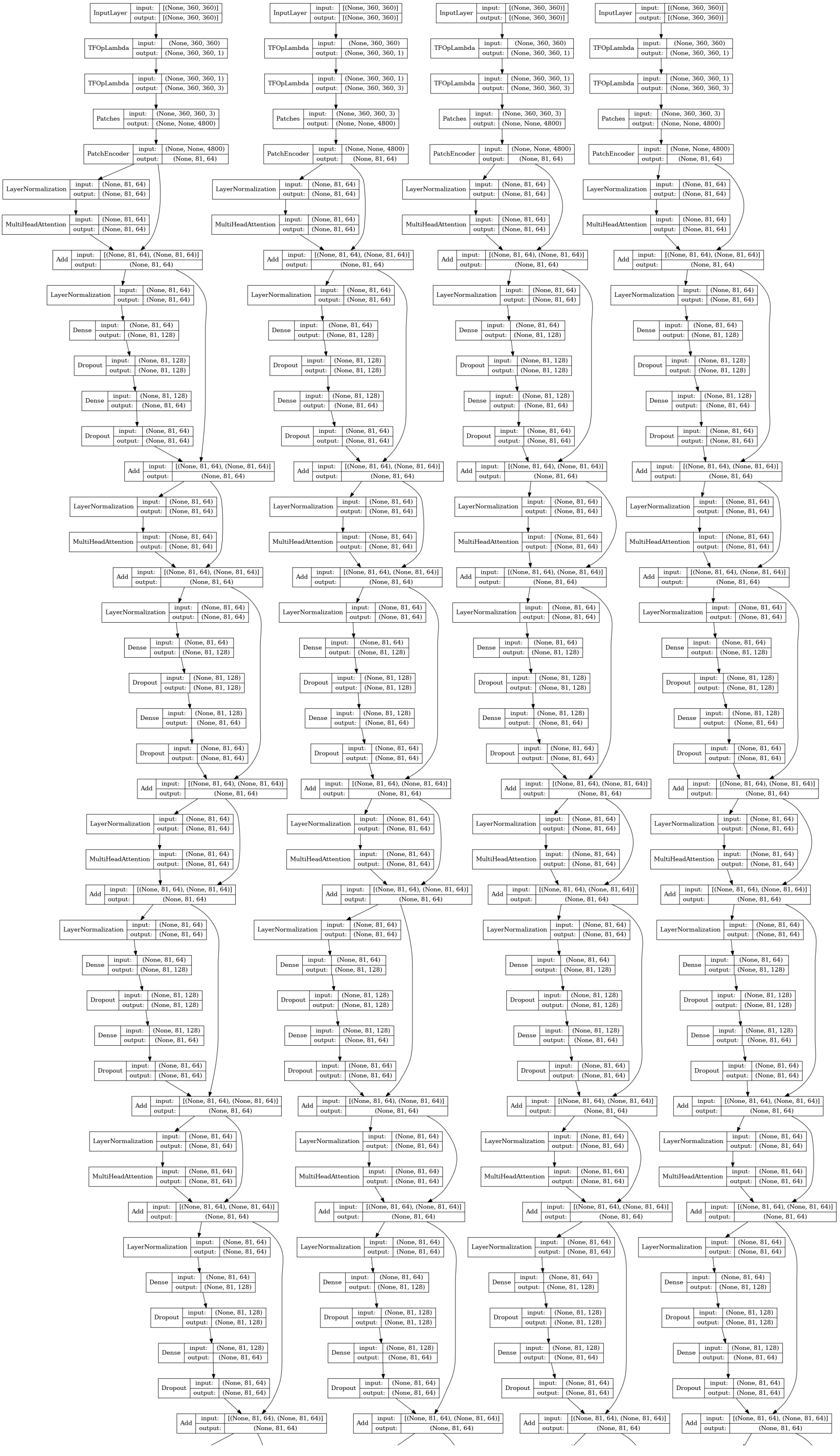
add_13 (Add)	(None, 81, 64)	0	dropout_13[0][0] add_12[0][0]
add_29 (Add)	(None, 81, 64)	0	dropout_32[0][0] add_28[0][0]
add_45 (Add)	(None, 81, 64)	0	dropout_51[0][0] add_44[0][0]
add_61 (Add)	(None, 81, 64)	0	dropout_70[0][0] add_60[0][0]
layer_normalization_14 (LayerNo	(None, 81, 64)	128	add_13[0][0]
layer_normalization_31 (LayerNo	(None, 81, 64)	128	add_29[0][0]
layer_normalization_48 (LayerNo	(None, 81, 64)	128	add_45[0][0]
layer_normalization_65 (LayerNo	(None, 81, 64)	128	add_61[0][0]
multi_head_attention_7 (MultiHe	(None, 81, 64)	66368	layer_normalization_14[0][0] layer_normalization_14[0][0]
multi_head_attention_15 (MultiH	(None, 81, 64)	66368	layer_normalization_31[0][0] layer_normalization_31[0][0]
multi_head_attention_23 (MultiH	(None, 81, 64)	66368	layer_normalization_48[0][0] layer_normalization_48[0][0]
multi_head_attention_31 (MultiH	(None, 81, 64)	66368	layer_normalization_65[0][0] layer_normalization_65[0][0]
add_14 (Add)	(None, 81, 64)	0	multi_head_attention_7[0][0] add_13[0][0]
add_30 (Add)	(None, 81, 64)	0	multi_head_attention_15[0][0] add_29[0][0]
add_46 (Add)	(None, 81, 64)	0	multi_head_attention_23[0][0] add_45[0][0]
add_62 (Add)	(None, 81, 64)	0	multi_head_attention_31[0][0] add_61[0][0]
layer_normalization_15 (LayerNo	(None, 81, 64)	128	add_14[0][0]
layer_normalization_32 (LayerNo	(None, 81, 64)	128	add_30[0][0]
layer_normalization_49 (LayerNo	(None, 81, 64)	128	add_46[0][0]
layer_normalization_66 (LayerNo	(None, 81, 64)	128	add_62[0][0]
dense_15 (Dense)	(None, 81, 128)	8320	layer_normalization_15[0][0]
dense_35 (Dense)	(None, 81, 128)	8320	layer_normalization_32[0][0]
dense_55 (Dense)	(None, 81, 128)	8320	layer_normalization_49[0][0]
dense_75 (Dense)	(None, 81, 128)	8320	layer_normalization_66[0][0]
dropout_14 (Dropout)	(None, 81, 128)	0	dense_15[0][0]

```
dropout_33 (Dropout)            (None, 81, 128)      0           dense_35[0][0]
dropout_52 (Dropout)            (None, 81, 128)      0           dense_55[0][0]
dropout_71 (Dropout)            (None, 81, 128)      0           dense_75[0][0]
dense_16 (Dense)                (None, 81, 64)       8256        dropout_14[0][0]
dense_36 (Dense)                (None, 81, 64)       8256        dropout_33[0][0]
dense_56 (Dense)                (None, 81, 64)       8256        dropout_52[0][0]
dense_76 (Dense)                (None, 81, 64)       8256        dropout_71[0][0]
dropout_15 (Dropout)            (None, 81, 64)       0           dense_16[0][0]
dropout_34 (Dropout)            (None, 81, 64)       0           dense_36[0][0]
dropout_53 (Dropout)            (None, 81, 64)       0           dense_56[0][0]
dropout_72 (Dropout)            (None, 81, 64)       0           dense_76[0][0]
add_15 (Add)                    (None, 81, 64)       0           dropout_15[0][0]
                                                                 add_14[0][0]
add_31 (Add)                    (None, 81, 64)       0           dropout_34[0][0]
                                                                 add_30[0][0]
add_47 (Add)                    (None, 81, 64)       0           dropout_53[0][0]
                                                                 add_46[0][0]
add_63 (Add)                    (None, 81, 64)       0           dropout_72[0][0]
                                                                 add_62[0][0]
layer_normalization_16 (LayerNo (None, 81, 64)       128         add_15[0][0]
layer_normalization_33 (LayerNo (None, 81, 64)       128         add_31[0][0]
layer_normalization_50 (LayerNo (None, 81, 64)       128         add_47[0][0]
layer_normalization_67 (LayerNo (None, 81, 64)       128         add_63[0][0]
flatten (Flatten)               (None, 5184)         0           layer_normalization_16[0][0]
flatten_1 (Flatten)             (None, 5184)         0           layer_normalization_33[0][0]
flatten_2 (Flatten)             (None, 5184)         0           layer_normalization_50[0][0]
flatten_3 (Flatten)             (None, 5184)         0           layer_normalization_67[0][0]
dropout_16 (Dropout)            (None, 5184)         0           flatten[0][0]
dropout_35 (Dropout)            (None, 5184)         0           flatten_1[0][0]
dropout_54 (Dropout)            (None, 5184)         0           flatten_2[0][0]
dropout_73 (Dropout)            (None, 5184)         0           flatten_3[0][0]
dense_17 (Dense)                (None, 2048)         10618880    dropout_16[0][0]
dense_37 (Dense)                (None, 2048)         10618880    dropout_35[0][0]
```

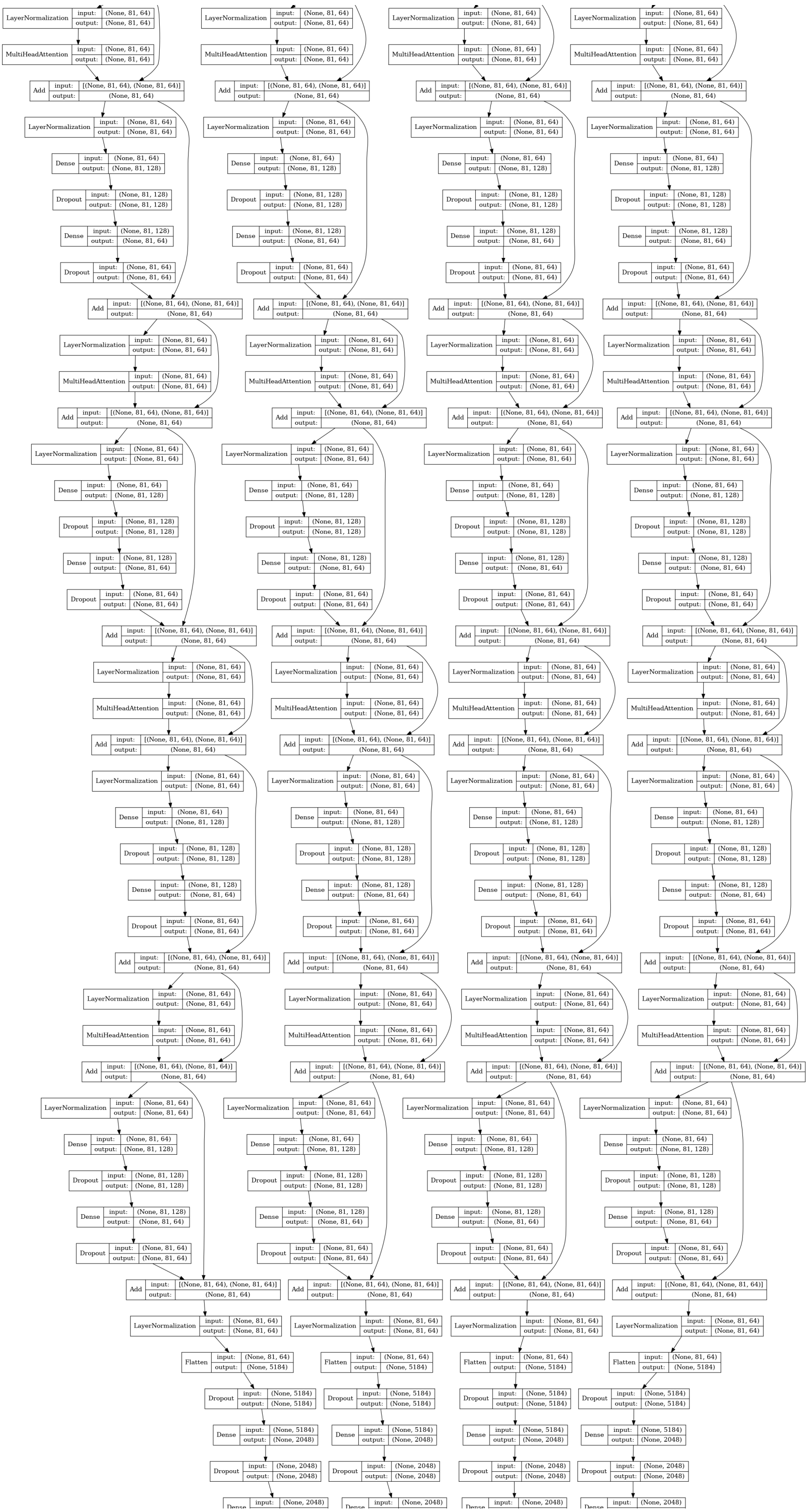
```
dense_57 (Dense)                (None, 2048)         10618880    dropout_54[0][0]
dense_77 (Dense)                (None, 2048)         10618880    dropout_73[0][0]
dropout_17 (Dropout)            (None, 2048)         0           dense_17[0][0]
dropout_36 (Dropout)            (None, 2048)         0           dense_37[0][0]
dropout_55 (Dropout)            (None, 2048)         0           dense_57[0][0]
dropout_74 (Dropout)            (None, 2048)         0           dense_77[0][0]
dense_18 (Dense)                (None, 1024)         2098176     dropout_17[0][0]
dense_38 (Dense)                (None, 1024)         2098176     dropout_36[0][0]
dense_58 (Dense)                (None, 1024)         2098176     dropout_55[0][0]
dense_78 (Dense)                (None, 1024)         2098176     dropout_74[0][0]
dropout_18 (Dropout)            (None, 1024)         0           dense_18[0][0]
dropout_37 (Dropout)            (None, 1024)         0           dense_38[0][0]
dropout_56 (Dropout)            (None, 1024)         0           dense_58[0][0]
dropout_75 (Dropout)            (None, 1024)         0           dense_78[0][0]
dense_19 (Dense)                (None, 256)          262400      dropout_18[0][0]
dense_39 (Dense)                (None, 256)          262400      dropout_37[0][0]
dense_59 (Dense)                (None, 256)          262400      dropout_56[0][0]
dense_79 (Dense)                (None, 256)          262400      dropout_75[0][0]
concatenate (Concatenate)       (None, 1024)         0           dense_19[0][0]
                                                                 dense_39[0][0]
                                                                 dense_59[0][0]
                                                                 dense_79[0][0]
dropout_76 (Dropout)            (None, 1024)         0           concatenate[0][0]
dense_80 (Dense)                (None, 128)          131200      dropout_76[0][0]
dropout_77 (Dropout)            (None, 128)          0           dense_80[0][0]
```

In [70]:
```
plot_model(model, show_shapes=True, show_layer_names=False, to_file='vitmodel.png')
```

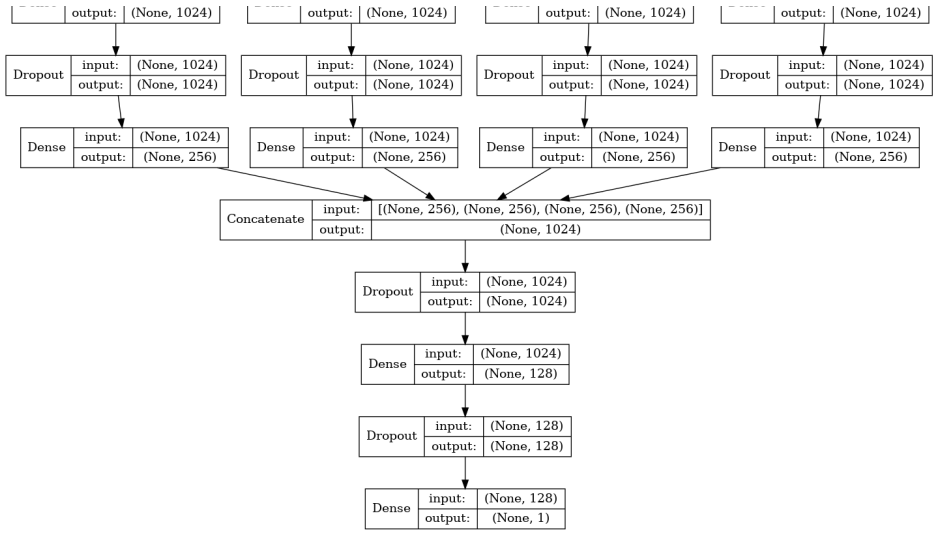
Out[70]:


InputLayer input: [(None, 360, 360)] output: [(None, 360, 360)]
TFOpLambda input: (None, 360, 360) output: (None, 360, 360, 1)
TFOpLambda input: (None, 360, 360, 1) output: (None, 360, 360, 3)
Patches input: (None, 360, 360, 3) output: (None, None, 4800)
PatchEncoder input: (None, None, 4800) output: (None, 81, 64)
LayerNormalization input: (None, 81, 64) output: (None, 81, 64)
MultiHeadAttention input: (None, 81, 64) output: (None, 81, 64)
Add input: [(None, 81, 64), (None, 81, 64)] output: (None, 81, 64)
Dense input: (None, 81, 64) output: (None, 81, 128)
Dropout input: (None, 81, 128) output: (None, 81, 128)
Dense input: (None, 81, 128) output: (None, 81, 64)
Dropout input: (None, 81, 64) output: (None, 81, 64)

Fitting tr


LayerNormalization input: (None, 81, 64) output: (None, 81, 64)
MultiHeadAttention input: (None, 81, 64) output: (None, 81, 64)
Add input: [(None, 81, 64), (None, 81, 64)] output: (None, 81, 64)
Dense input: (None, 81, 64) output: (None, 81, 128)
Dropout input: (None, 81, 128) output: (None, 81, 128)
Dense input: (None, 81, 128) output: (None, 81, 64)
Dropout input: (None, 81, 64) output: (None, 81, 64)
Flatten input: (None, 81, 64) output: (None, 5184)
Dropout input: (None, 5184) output: (None, 5184)
Dense input: (None, 5184) output: (None, 2048)
Dropout input: (None, 2048) output: (None, 2048)
Dense input: (None, 2048)

In [71]:


output: (None, 1024)
Dropout input: (None, 1024) output: (None, 1024)
Dense input: (None, 1024) output: (None, 256)
Concatenate input: [(None, 256), (None, 256), (None, 256), (None, 256)] output: (None, 1024)
Dropout input: (None, 1024) output: (None, 1024)
Dense input: (None, 1024) output: (None, 128)
Dropout input: (None, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 1)

```
Epoch 1/30
18/18 [==============================] - 117s 3s/step - loss: 6.0524 - auc: 0.5040
Epoch 2/30
18/18 [==============================] - 80s 4s/step - loss: 8.0402 - auc: 0.4547
Epoch 3/30
18/18 [==============================] - 84s 5s/step - loss: 4.8567 - auc: 0.5066
Epoch 4/30
18/18 [==============================] - 80s 4s/step - loss: 3.9407 - auc: 0.4964
Epoch 5/30
18/18 [==============================] - 71s 4s/step - loss: 3.0267 - auc: 0.4896
Epoch 6/30
18/18 [==============================] - 71s 4s/step - loss: 2.2414 - auc: 0.4636
Epoch 7/30
18/18 [==============================] - 68s 4s/step - loss: 1.3932 - auc: 0.5169
Epoch 8/30
18/18 [==============================] - 68s 4s/step - loss: 1.2671 - auc: 0.5188
Epoch 9/30
18/18 [==============================] - 65s 4s/step - loss: 1.2454 - auc: 0.5159
Epoch 10/30
18/18 [==============================] - 100s 6s/step - loss: 1.0282 - auc: 0.5225
Epoch 11/30
18/18 [==============================] - 143s 8s/step - loss: 0.9843 - auc: 0.5287
Epoch 12/30
18/18 [==============================] - 134s 7s/step - loss: 0.9696 - auc: 0.5108
Epoch 13/30
18/18 [==============================] - 133s 7s/step - loss: 0.9676 - auc: 0.4848
Epoch 14/30
18/18 [==============================] - 128s 7s/step - loss: 0.8290 - auc: 0.4933
Epoch 15/30
18/18 [==============================] - 129s 7s/step - loss: 0.7973 - auc: 0.4956
Epoch 16/30
18/18 [==============================] - 128s 7s/step - loss: 0.7338 - auc: 0.5204
Epoch 17/30
18/18 [==============================] - 115s 6s/step - loss: 0.7857 - auc: 0.5055
Epoch 18/30
18/18 [==============================] - 117s 6s/step - loss: 0.7843 - auc: 0.5045
Epoch 19/30
18/18 [==============================] - 127s 7s/step - loss: 0.7402 - auc: 0.4880
Epoch 20/30
18/18 [==============================] - 127s 7s/step - loss: 0.7723 - auc: 0.4695
Epoch 21/30
18/18 [==============================] - 135s 7s/step - loss: 0.7311 - auc: 0.5303
Epoch 22/30
18/18 [==============================] - 127s 7s/step - loss: 0.7824 - auc: 0.4961
Epoch 23/30
18/18 [==============================] - 124s 7s/step - loss: 0.7396 - auc: 0.4733
Epoch 24/30
18/18 [==============================] - 121s 7s/step - loss: 0.7539 - auc: 0.4719
Epoch 25/30
18/18 [==============================] - 135s 7s/step - loss: 0.7335 - auc: 0.5132
Epoch 26/30
18/18 [==============================] - 140s 8s/step - loss: 0.7585 - auc: 0.4277
Epoch 27/30
18/18 [==============================] - 127s 7s/step - loss: 0.7353 - auc: 0.5133
Epoch 28/30
18/18 [==============================] - 126s 7s/step - loss: 0.7369 - auc: 0.4871
Epoch 29/30
18/18 [==============================] - 125s 7s/step - loss: 0.7172 - auc: 0.4923
Epoch 30/30
18/18 [==============================] - 146s 8s/step - loss: 0.7295 - auc: 0.5010
```

Saving Model(ViT) Weights

In [72]:
```
model.save_weights('g2net_vit.h5')
```

Exploring CNN in action

In [73]:
```
tf.keras.backend.clear_session()
gc.collect()
cnnmodel = get_cnn_model()
```

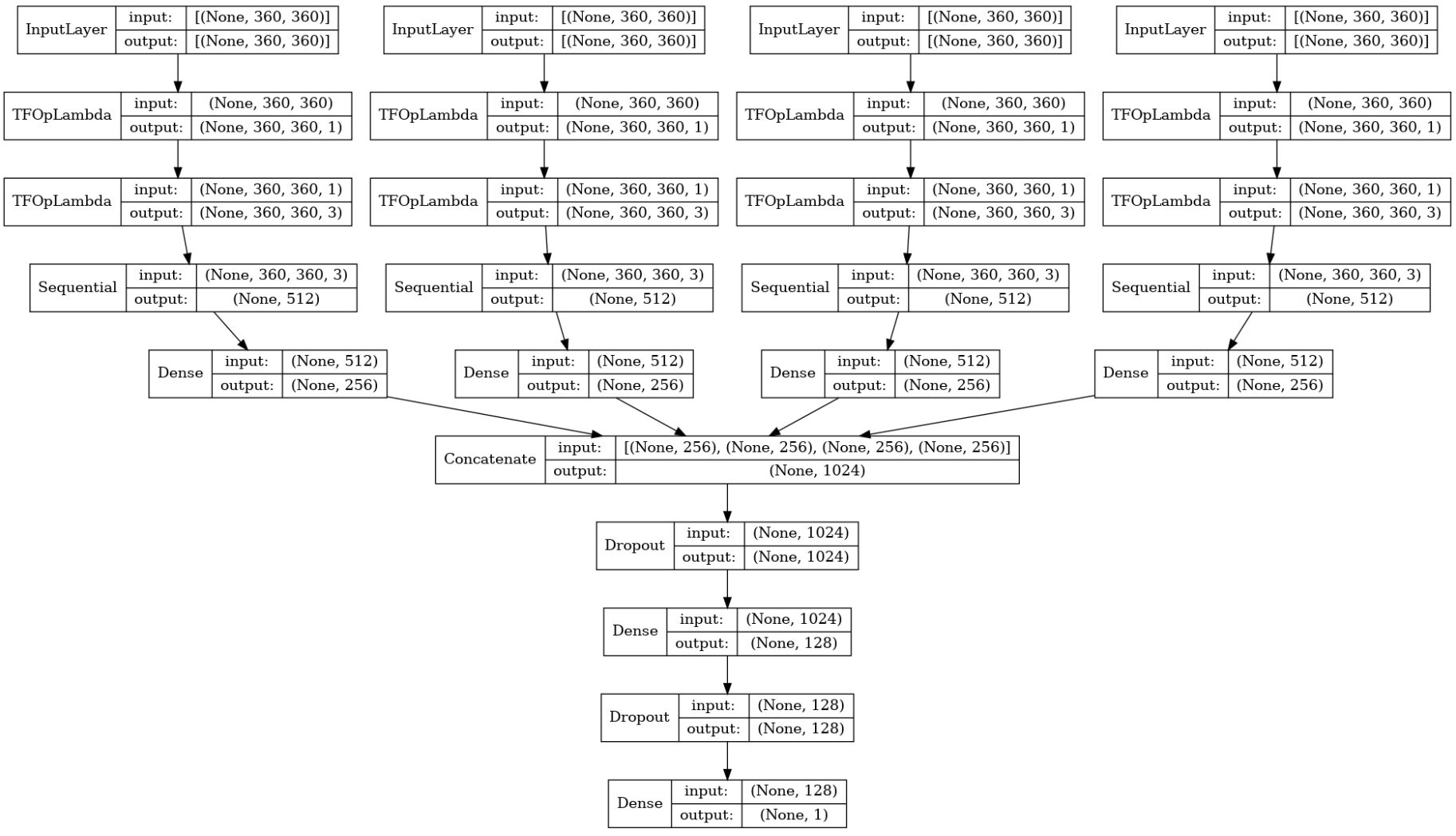
In [74]:
```
print(cnnmodel.summary())
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
x_h_r (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_h_i (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_l_r (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
x_l_i (InputLayer)              [(None, 360, 360)]   0
__________________________________________________________________________________________________
tf.expand_dims (TFOpLambda)     (None, 360, 360, 1)  0           x_h_r[0][0]
__________________________________________________________________________________________________
tf.expand_dims_1 (TFOpLambda)   (None, 360, 360, 1)  0           x_h_i[0][0]
__________________________________________________________________________________________________
tf.expand_dims_2 (TFOpLambda)   (None, 360, 360, 1)  0           x_l_r[0][0]
__________________________________________________________________________________________________
tf.expand_dims_3 (TFOpLambda)   (None, 360, 360, 1)  0           x_l_i[0][0]
__________________________________________________________________________________________________
tf.tile (TFOpLambda)            (None, 360, 360, 3)  0           tf.expand_dims[0][0]
__________________________________________________________________________________________________
tf.tile_1 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_1[0][0]
__________________________________________________________________________________________________
tf.tile_2 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_2[0][0]
__________________________________________________________________________________________________
tf.tile_3 (TFOpLambda)          (None, 360, 360, 3)  0           tf.expand_dims_3[0][0]
__________________________________________________________________________________________________
sequential (Sequential)         (None, 512)          110720      tf.tile[0][0]
__________________________________________________________________________________________________
sequential_1 (Sequential)       (None, 512)          110720      tf.tile_1[0][0]
__________________________________________________________________________________________________
sequential_2 (Sequential)       (None, 512)          110720      tf.tile_2[0][0]
__________________________________________________________________________________________________
sequential_3 (Sequential)       (None, 512)          110720      tf.tile_3[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          131328      sequential[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 256)          131328      sequential_1[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 256)          131328      sequential_2[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 256)          131328      sequential_3[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 1024)         0           dense_1[0][0]
                                                                 dense_3[0][0]
                                                                 dense_5[0][0]
                                                                 dense_7[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 1024)         0           concatenate[0][0]
__________________________________________________________________________________________________
dense_8 (Dense)                 (None, 128)          131200      dropout[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 128)          0           dense_8[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 1)            129         dropout_1[0][0]
==================================================================================================
Total params: 1,099,521
Trainable params: 1,097,985
Non-trainable params: 1,536
__________________________________________________________________________________________________
```

None

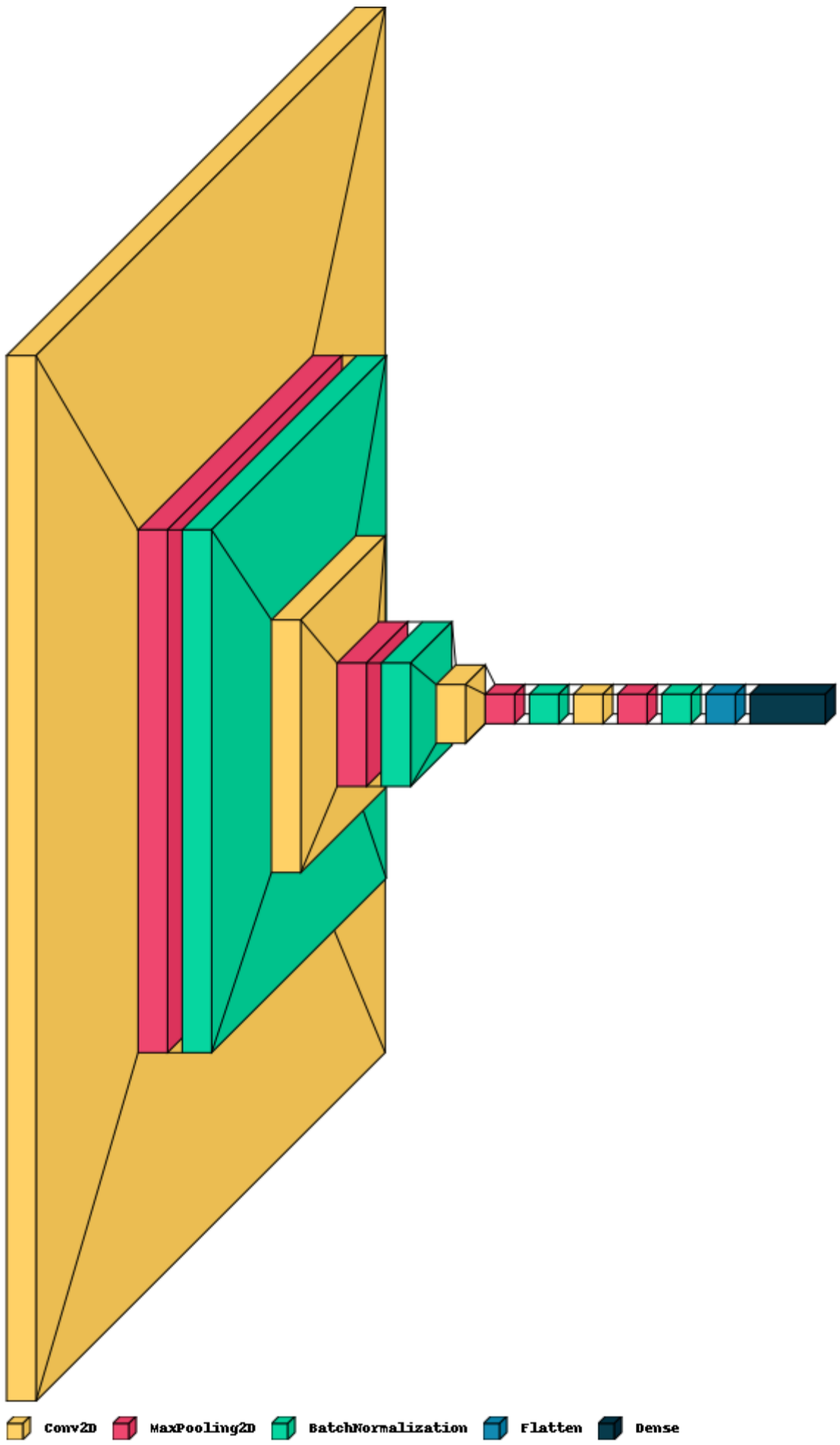
In [75]:
```
plot_model(cnnmodel, show_shapes=True, show_layer_names=False, to_file='cnnmodel.png')
```

Out[75]:



In [76]:
```
visualkeras.layered_view(arch, legend=True)
```

Out[76]:



In [77]:
```
class_weights = {0:1.5, 1:0.75}
cnnhistory = cnnmodel.fit(
        get_train_dataset(SAMPLE_IDXS, batch_size),
        steps_per_epoch=N_TRAIN_SAMPLES // batch_size,
        epochs = 10,
        class_weight=class_weights,
        verbose = 1,
    )
```

```
Epoch 1/10
18/18 [==============================] - 128s 7s/step - loss: 3.7860 - auc: 0.4816
Epoch 2/10
18/18 [==============================] - 135s 8s/step - loss: 2.7443 - auc: 0.4782
Epoch 3/10
18/18 [==============================] - 138s 8s/step - loss: 1.9265 - auc: 0.4585
Epoch 4/10
18/18 [==============================] - 129s 7s/step - loss: 1.3020 - auc: 0.5266
Epoch 5/10
18/18 [==============================] - 146s 8s/step - loss: 0.9517 - auc: 0.4524
Epoch 6/10
18/18 [==============================] - 146s 8s/step - loss: 0.7574 - auc: 0.4858
Epoch 7/10
18/18 [==============================] - 149s 8s/step - loss: 0.7196 - auc: 0.5144
Epoch 8/10
18/18 [==============================] - 140s 8s/step - loss: 0.7017 - auc: 0.4756
Epoch 9/10
18/18 [==============================] - 147s 8s/step - loss: 0.7003 - auc: 0.4371
Epoch 10/10
18/18 [==============================] - 143s 8s/step - loss: 0.7084 - auc: 0.4645
```

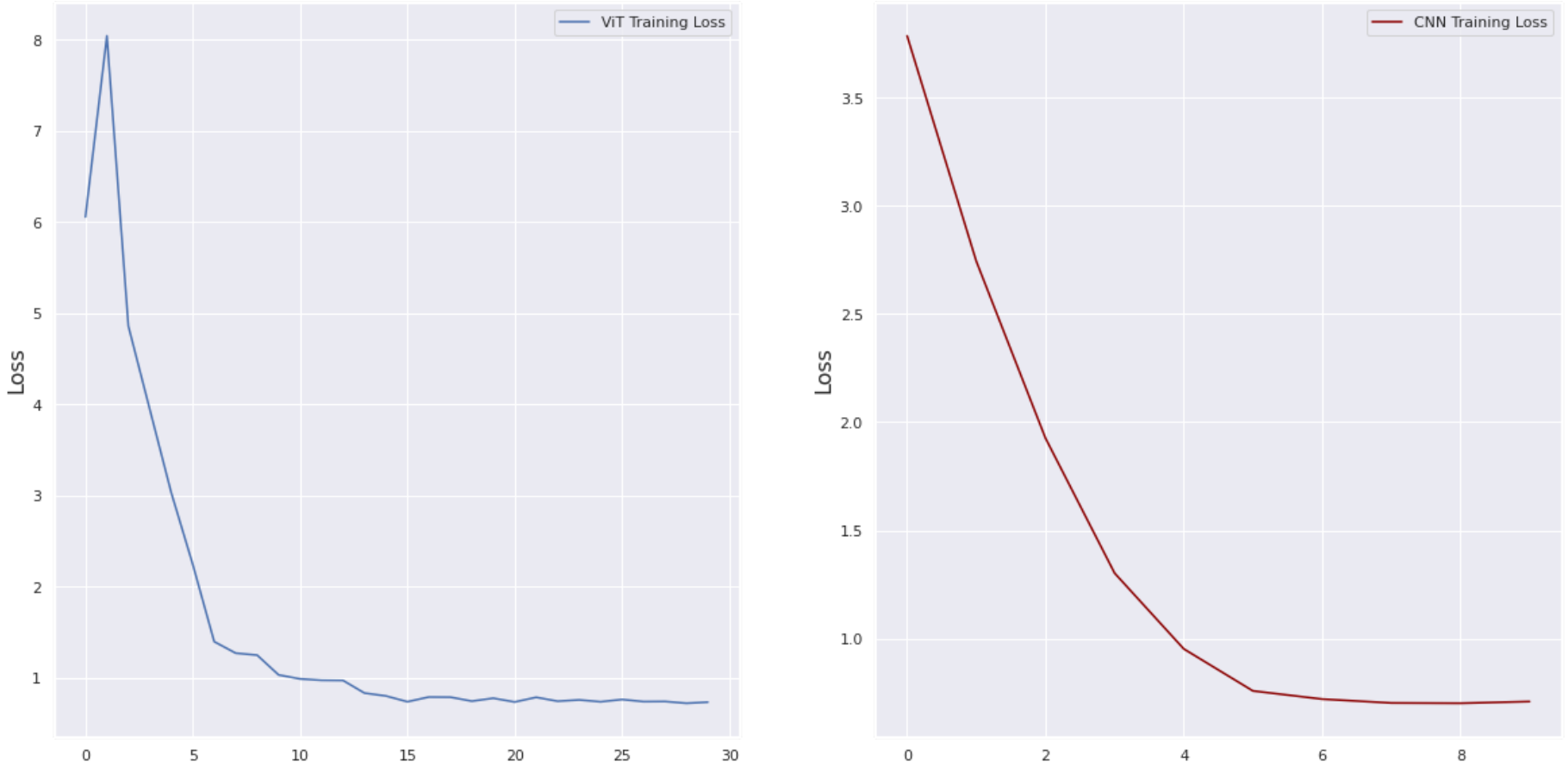
Saving CNN weights

In [78]:
```
cnnmodel.save_weights('g2net_cnn.h5')
```

Binary Cross Entropy Loss Curve

In [79]:
```
plt.figure(figsize=(20,10))

plt.subplot(1, 2, 1)
plt.suptitle('Optimizer : Adam, Loss : Binary CrossEntropy', fontsize=10)
plt.ylabel('Loss', fontsize=16)
plt.plot(history.history['loss'], label='ViT Training Loss')
plt.legend(loc='upper right')

plt.subplot(1, 2, 2)
plt.ylabel('Loss', fontsize=16)
plt.plot(cnnhistory.history['loss'], label='CNN Training Loss', c = 'darkred')

plt.legend(loc='upper right')
```

Out[79]:
```
<matplotlib.legend.Legend at 0x7f3d2da03d10>
```



Notes about Frequency Domain modeling

- As expected the CNN model could converge faster 3x than the ViT model as the available dataset size isn't the ideal for such architecture(we's talking about 600 instances here) which is based on vision transformers which in turn needs more data than CNN to work properly.
- The model loss curve is descending but it doesn't mean that the model is learning well, that's why further INVESTIGATION and IMPROVEMENTS shall be done on this work to find out if the imbalanced classes is the main issue here or if we need to try feature extraction techniques(noise cancelation filters, etc.) other than using the SFT's available. One of the possible improvements here is to use pre-trained model and only fine tune it, this shall overcome data limitation and other issues.