Generative Models for Gravitational Wave Data Synthesis: Generative Adversarial Networks and WaveNet

Shufan Dong[1]

[1]email: shufandong6011@gmail.com

Abstract

This paper presents a comprehensive approach to generating synthetic gravitational wave (GW) data using Generative Adversarial Networks (GANs) and WaveNet models. The methodology comprises data preparation, augmentation, model construction, training processes, and plot visualization. Detailed explanations of each step and the underlying concepts are provided to demonstrate the mechanics and effectiveness of these generative models in creating realistic gravitational waveforms. These advancements not only enhance GW detection algorithms but also hold significant implications for the broader field of astrophysics by facilitating more accurate simulations of cosmic events and improving the robustness of astrophysical models by providing augmented GW data.

Contents

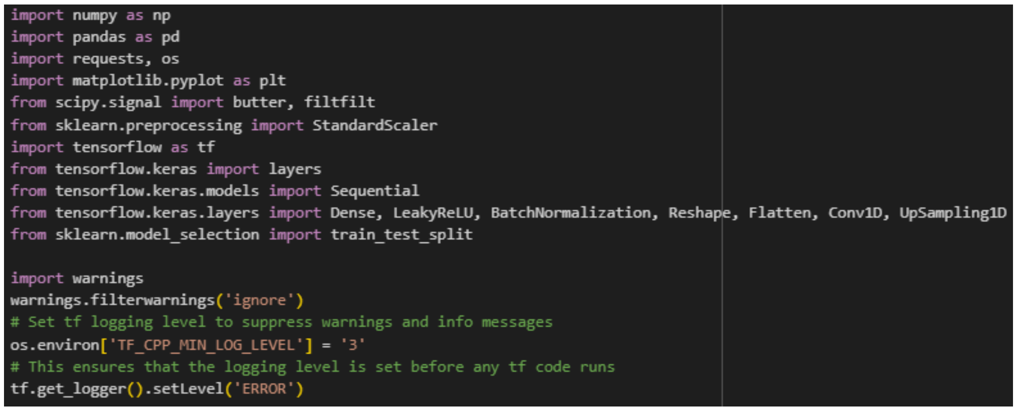
1 Introduction

Gravitational wave (GW) detection has opened a transformative research avenue in astrophysics, enabling the observation of cosmic events through GWs. These observations provide a new method for studying astrophysical phenomena, such as black hole mergers, neutron star collisions, and other high-energy cosmic events. The procedure for GW data preprocessing, as detailed in [30], and the data preparation steps outlined in [31], are critical for ensuring the data is suitable for advanced techniques and model training. While this paper touches on the results from the data preparation steps, it emphasizes the methodologies for generating synthetic GW data used in enhancing detection algorithms and training machine learning (ML) models. By exploring the use of advanced generative models like GANs and WaveNet, this research contributes to the development of more sophisticated tools for astrophysical analysis. This synthesis of GW data is vital for not only improving detection capabilities but also for expanding our understanding of the universe by simulating scenarios that are challenging to observe directly.

2 Data Preparation

2.1 Importing Necessary Libraries

The following code imports necessary libraries and modules for data preprocessing, data preparation, and building ML models, specifically using TensorFlow for neural network (NN) creation and training. We also import warning suppression libraries to ensure a smooth execution of the codes.



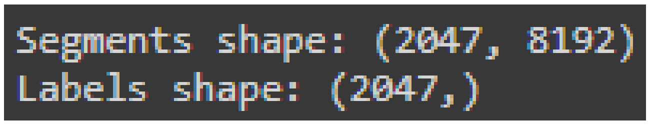
```python
import numpy as np
import pandas as pd
import requests, os
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Flatten, Conv1D, UpSampling1D
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')
# Set tf logging level to suppress warnings and info messages
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
# This ensures that the logging level is set before any tf code runs
tf.get_logger().setLevel('ERROR')
```

Figure 1: Visualization of all the libraries imported.

- numpy: numerical manipulation in Python.
- pandas: data manipulation.
- requests: making requests to web.
- os: operating system interfaces.
- matplotlib.pyplot: plotting library.
- scipy.signal: contains functions for signal processing.
- sklearn.preprocessing: has scaling features.
- tensorflow: library for ML, particularly deep learning (DL).
  - layers: building blocks for NN layers.
  - models.Sequential: a linear stack of layers in Keras.
  - layers: contains various NN layers.
- sklearn.model_selection: contains the train_test_split function for splitting datasets into training and testing sets.
- warnings: manages warnings in Python, suppressing them for clean output.

2.2 Segment and Label Creation

To prepare the data for model training, the GW strain data is segmented. The function `create_segments_and_labels` is created for segmenting the GW strain data into smaller chunks for analysis.



```
Segments shape: (2047, 8192)
Labels shape: (2047,)
```

Figure 2: The shape of the segmented and labeled GW data.

Inputs:

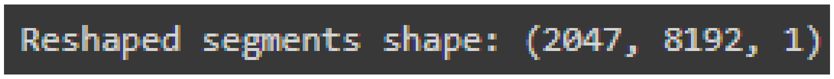
- `strain`: the time-series data of GW strain.
- `t_start`: the start of the GW event selected.
- `segment_length`: segment lengths in seconds.
- `sampling_rate`: sampling rate in Hz.

Outputs:

- `segments`: array of segmented GW data
- `labels`: labels for each corresponding segment classifying the presence of an event.

2.3 Reshape Segments

The segments are reshaped to be compatible with `Conv1D` layers used in deep learning (DL) models. We add an extra dimension of 1 to represent a single channel since `Conv1D` expects a 3D input in this particular order: data, timesteps, and channels.



```
Reshaped segments shape: (2047, 8192, 1)
```

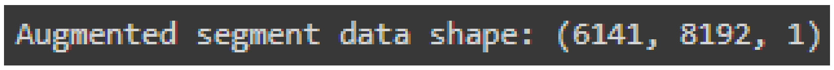
Figure 3: The shape of the reshaped GW data.

2.4 Split Data

The data is split into training and testing datasets for the WaveNet model only, as the GAN model doesn't need this step due to its unsupervised nature. We use the function `train_test_split` from `sklearn.model_selection` to split the data into training (80%) and testing (20%) sets for the WaveNet model.

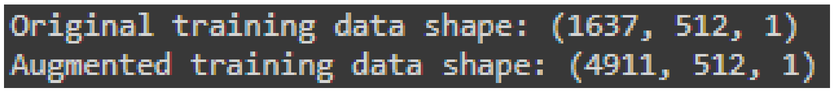
2.5 Data Augmentation

Training data is augmented to increase dataset size and variability. We created the function `augment_data` to augment the segmented data for the GAN model and the training data for the WaveNet model.



```
Augmented segment data shape: (6141, 8192, 1)
```

Figure 4: The shape of the augmented GW data for GAN.



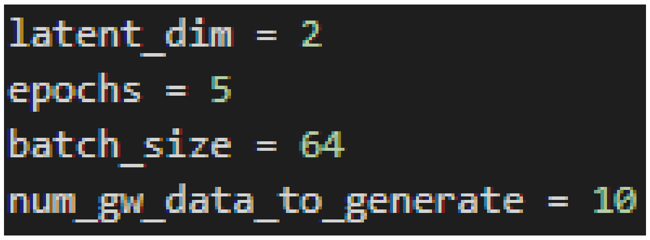
```
Original training data shape: (1637, 512, 1)
Augmented training data shape: (4911, 512, 1)
```

Figure 5: The shape of the augmented training set for WaveNet before and after augmentation. Note that the GW data is purposefully downsampled for a quicker training process.

3 Model Training

3.1 GAN

3.1.1 Hyperparameters



```
latent_dim = 2
epochs = 5
batch_size = 64
num_gw_data_to_generate = 10
```

Figure 6: The hyperparameters for implementing GAN.

- `latent_dim`: dimensionality of the latent space (input vector)).
- `num_gw_data_to_generate`: number of synthetic gravitational wave data samples to generate after training.

3.1.2 Define Generator

The `build_generator` function creates the generator model to synthesize GW data.

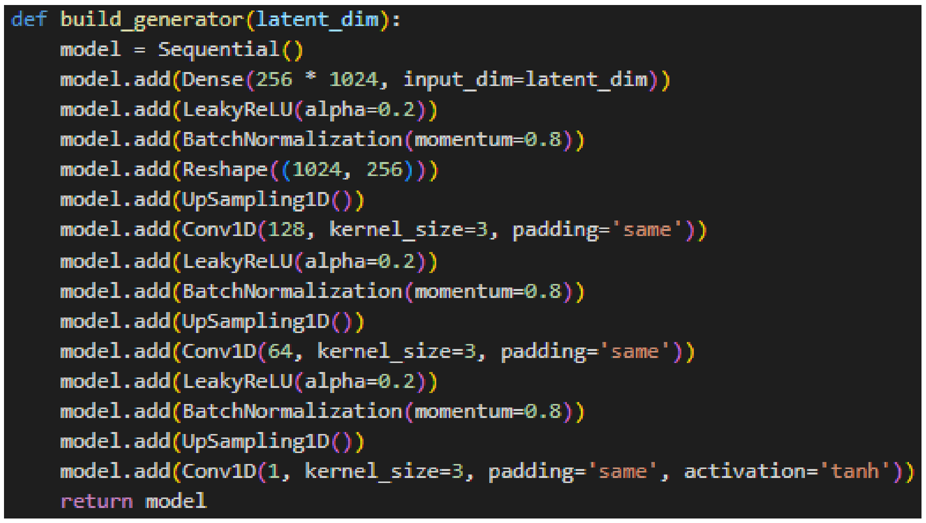
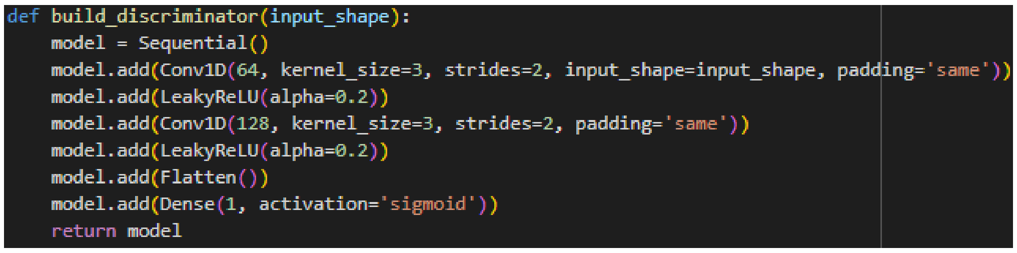

```python
def build_generator(latent_dim):
    model = Sequential()
    model.add(Dense(256 * 1024, input_dim=latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Reshape((1024, 256)))
    model.add(UpSampling1D())
    model.add(Conv1D(128, kernel_size=3, padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(UpSampling1D())
    model.add(Conv1D(64, kernel_size=3, padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(UpSampling1D())
    model.add(Conv1D(1, kernel_size=3, padding='same', activation='tanh'))
    return model
```

Figure 7: The construction of the build generator.

- `Dense Layer`: initial dense layer with `latent_dim` input.
- `LeakyReLU`: LeakyReLU activation function.
- `BatchNormalization`: normalizes the output.
- `Reshape`: reshapes the output into a suitable shape for `Conv1D` layers.
- `UpSampling1D`: upsamples the input.
- `Conv1D Layers`: convolutional layers to extract features.
- `Activation`: `tanh` activation to output values between -1 and 1.

3.1.3 Define Discriminator

The `build_discriminator` function creates the discriminator model to distinguish real versus generated data.



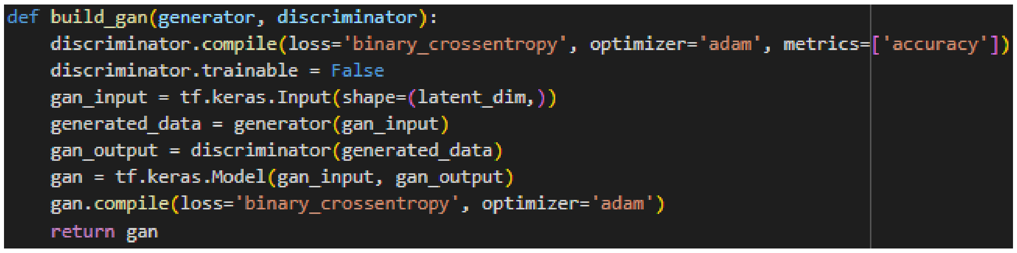
```python
def build_discriminator(input_shape):
    model = Sequential()
    model.add(Conv1D(64, kernel_size=3, strides=2, input_shape=input_shape, padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Conv1D(128, kernel_size=3, strides=2, padding='same'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Flatten())
    model.add(Dense(1, activation='sigmoid'))
    return model
```

Figure 8: The construction of the build generator.

- `Conv1D Layers`: convolutional layers to extract features.
- `LeakyReLU`: LeakyReLU activation function.
- `Flatten`: Flattens the 3D tensor into 1D.
- `Dense Layer`: final dense layer to output a single probability (of it being real and not generated data) with sigmoid activation.

3.1.4 Define GAN

The `build_gan` function combines the generator and discriminator into a GAN model.



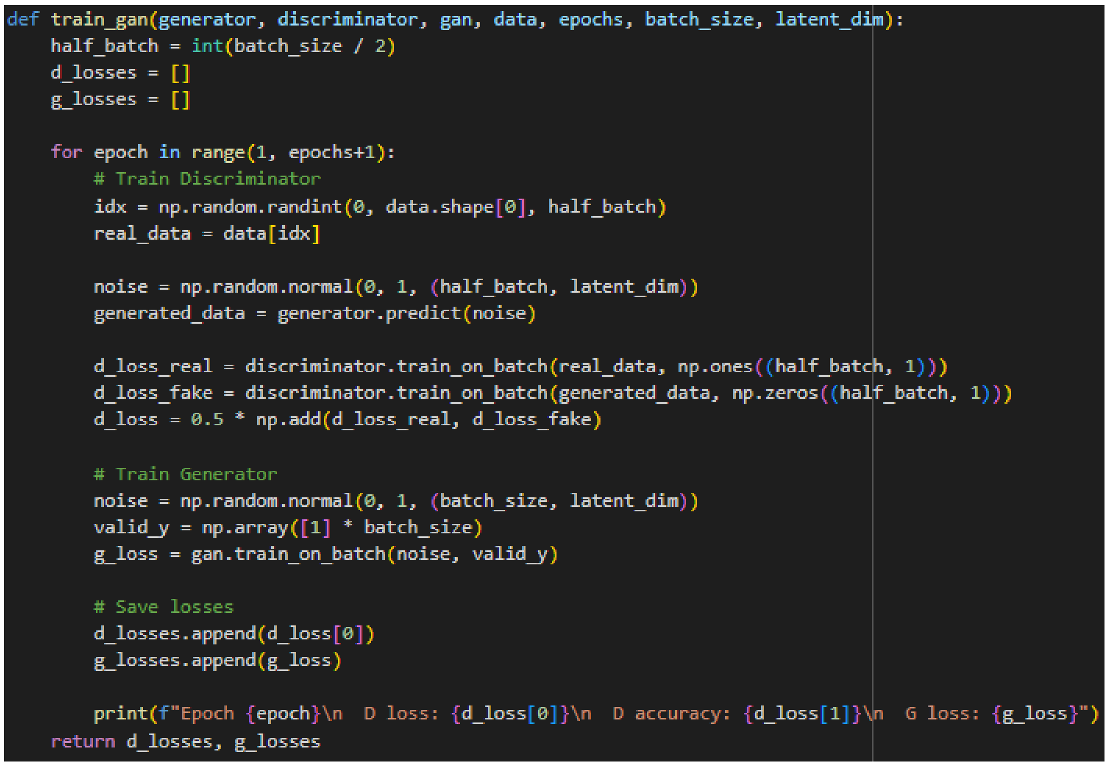
```python
def build_gan(generator, discriminator):
    discriminator.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    discriminator.trainable = False
    gan_input = tf.keras.Input(shape=(latent_dim,))
    generated_data = generator(gan_input)
    gan_output = discriminator(generated_data)
    gan = tf.keras.Model(gan_input, gan_output)
    gan.compile(loss='binary_crossentropy', optimizer='adam')
    return gan
```

Figure 9: The construction of the GAN.

- `Compile Discriminator`: compile the discriminator.
- `Freeze Discriminator`: ensure only the generator is trained.
- `GAN Input`: create input layer for the GAN model.
- `Generated Data`: pass input through the generator to get synthetic data.
- `GAN Output`: pass generated data through the discriminator to get the probability (of it being real and not generated data).
- `Compile GAN`: compile the GAN model.

3.1.5 Train GAN

The `train_gan` function trains the GAN by alternating between training the discriminator and the generator. The steps are as follows:



```
def train_gan(generator, discriminator, gan, data, epochs, batch_size, latent_dim):
    half_batch = int(batch_size / 2)
    d_losses = []
    g_losses = []

    for epoch in range(1, epochs+1):
        # Train Discriminator
        idx = np.random.randint(0, data.shape[0], half_batch)
        real_data = data[idx]

        noise = np.random.normal(0, 1, (half_batch, latent_dim))
        generated_data = generator.predict(noise)

        d_loss_real = discriminator.train_on_batch(real_data, np.ones((half_batch, 1)))
        d_loss_fake = discriminator.train_on_batch(generated_data, np.zeros((half_batch, 1)))
        d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

        # Train Generator
        noise = np.random.normal(0, 1, (batch_size, latent_dim))
        valid_y = np.array([1] * batch_size)
        g_loss = gan.train_on_batch(noise, valid_y)

        # Save losses
        d_losses.append(d_loss[0])
        g_losses.append(g_loss)

        print(f"Epoch {epoch}\n  D loss: {d_loss[0]}\n  D accuracy: {d_loss[1]}\n  G loss: {g_loss}")
    return d_losses, g_losses
```

Figure 10: Visualization of the GAN training loop.

Training Loop:

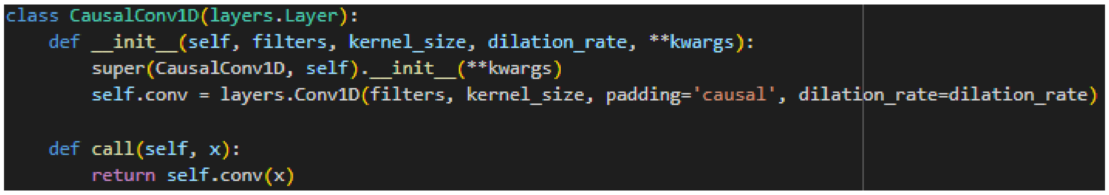
- `Train Discriminator`:
  - Sample real data.
  - Generate synthetic data.
  - Train on real data (labeled 1) and synthetic data (labeled 0).
  - Compute the discriminator loss.
- `Train Generator`:
  - Generate random noise.
  - Create an array with every element labeled 1 for the noise.

– Train on the random noise and array.
– Compute the generator loss.

3.2 WaveNet

3.2.1 Define Causal Convolutional Layer

The causal convolutional layers are used to maintain causality in time series data.



```python
class CausalConv1D(layers.Layer):
    def __init__(self, filters, kernel_size, dilation_rate, **kwargs):
        super(CausalConv1D, self).__init__(**kwargs)
        self.conv = layers.Conv1D(filters, kernel_size, padding='causal', dilation_rate=dilation_rate)

    def call(self, x):
        return self.conv(x)
```

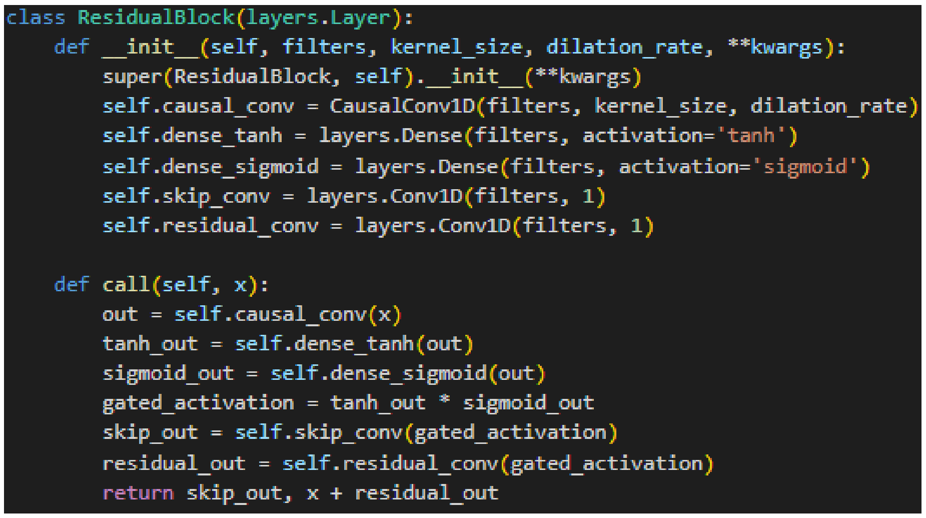
Figure 11: The causal Conv1D class.

CausalConv1D Class:

- __init__ function:
  – Inherits from `layers.Layer`.
  – Creates a `Conv1D` layer.
- Call function:
  – Defines the forward pass by returning the convolutional layer that's applied to the input tensor.

3.2.2 Define Residual Block

The residual block is added to build complex feature representations while maintaining gradient flow through skip connections.



```python
class ResidualBlock(layers.Layer):
    def __init__(self, filters, kernel_size, dilation_rate, **kwargs):
        super(ResidualBlock, self).__init__(**kwargs)
        self.causal_conv = CausalConv1D(filters, kernel_size, dilation_rate)
        self.dense_tanh = layers.Dense(filters, activation='tanh')
        self.dense_sigmoid = layers.Dense(filters, activation='sigmoid')
        self.skip_conv = layers.Conv1D(filters, 1)
        self.residual_conv = layers.Conv1D(filters, 1)

    def call(self, x):
        out = self.causal_conv(x)
        tanh_out = self.dense_tanh(out)
        sigmoid_out = self.dense_sigmoid(out)
        gated_activation = tanh_out * sigmoid_out
        skip_out = self.skip_conv(gated_activation)
        residual_out = self.residual_conv(gated_activation)
        return skip_out, x + residual_out
```

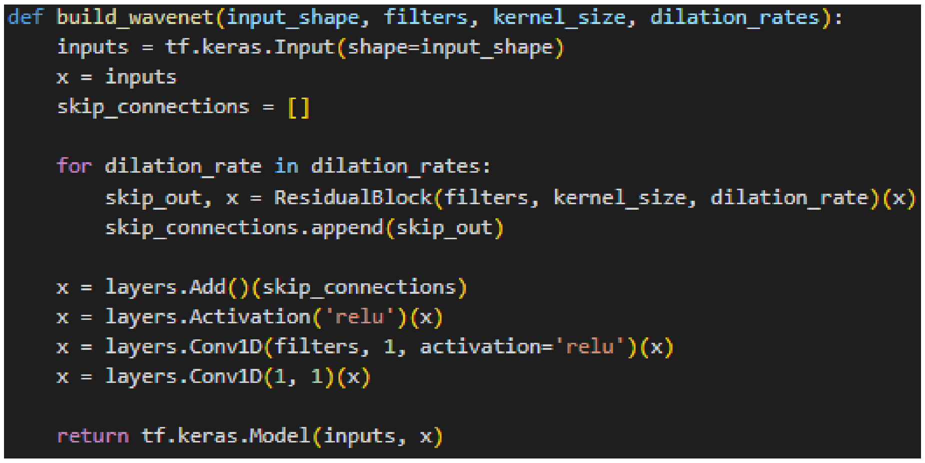
Figure 12: The residual block class.

ResidualBlock Class:

- __init__ function:
  - Inherits from `layers.Layer`.
  - Creates a `CausalConv1D` layer.
  - Creates two `Dense` layers with `tanh` and `sigmoid` activations, respectively.
  - Creates two `Conv1D` layers for skip and residual connections.
- Call function:
  - Defines the forward pass:
    - Applying the `CausalConv1D` layer to the input.
    - Applying the `Dense` layers with `tanh` and `sigmoid` activations to the output of the previous layer.
    - Multiplying the outputs of the `tanh` and `sigmoid` layers to create a gated activation.
    - Applying the `skip_conv` layer to the gated activation for the skip connection.
    - Applying the `residual_conv` layer to the gated activation and adding it to the input to create the residual output.
  - Returning the skip output and the residual output.

3.2.3 Define and train WaveNet

The `build_wavenet` function creates a WaveNet model for sequential data generation.

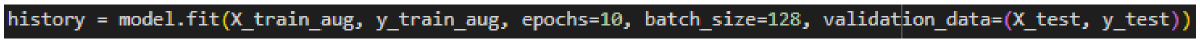


```python
def build_wavenet(input_shape, filters, kernel_size, dilation_rates):
    inputs = tf.keras.Input(shape=input_shape)
    x = inputs
    skip_connections = []

    for dilation_rate in dilation_rates:
        skip_out, x = ResidualBlock(filters, kernel_size, dilation_rate)(x)
        skip_connections.append(skip_out)

    x = layers.Add()(skip_connections)
    x = layers.Activation('relu')(x)
    x = layers.Conv1D(filters, 1, activation='relu')(x)
    x = layers.Conv1D(1, 1)(x)

    return tf.keras.Model(inputs, x)
```

Figure 13: The construction of the WaveNet.

- `Inputs`: input layer with specified shape.
- `Residual Blocks`: apply multiple residual blocks with different dilation rates.
- `Skip Connections`: collect and sum connections.
- `Activations and Convolutions`: layers to produce output.

We then train the WaveNet on augmented data and validate it on test data.

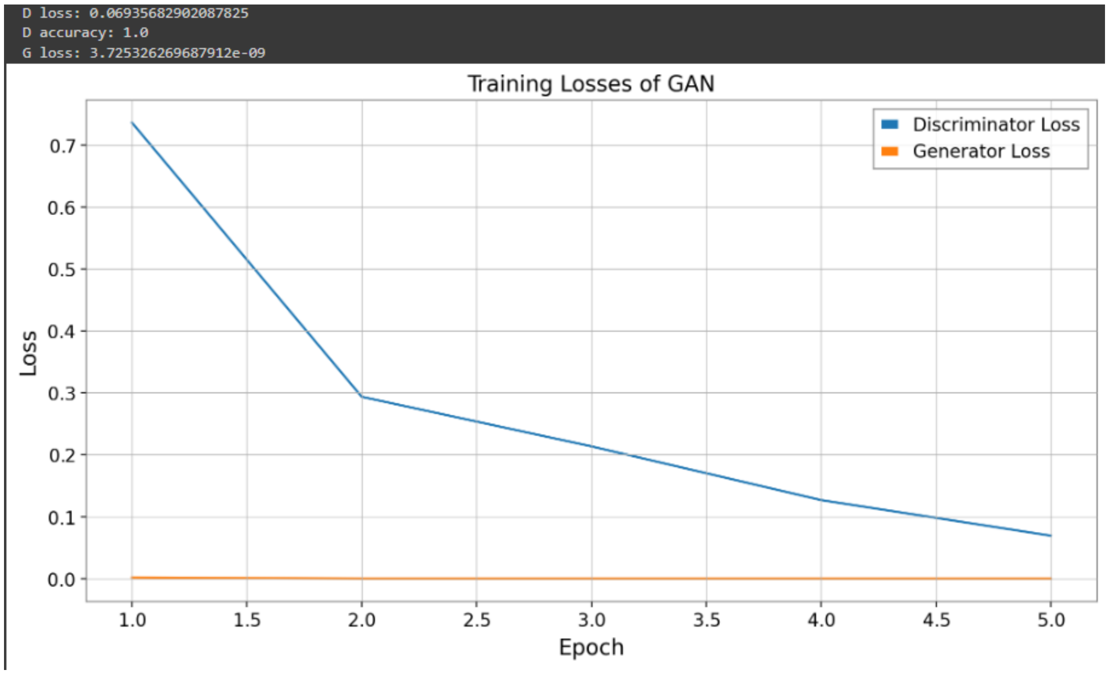


```python
history = model.fit(X_train_aug, y_train_aug, epochs=10, batch_size=128, validation_data=(X_test, y_test))
```

Figure 14: Training and saving its history for WaveNet

4 Model Evaluation and Visualization

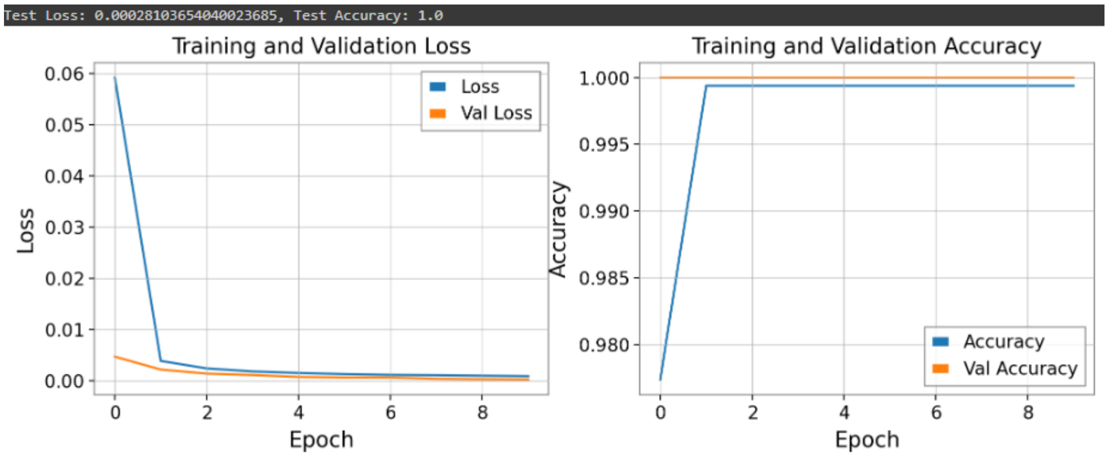
4.1 GAN



```
D loss: 0.06935682902087825
D accuracy: 1.0
G loss: 3.725326269687912e-09
```

Figure 15: Visualization of the discriminator and generator losses over epochs (G = Generator, D = Discriminator).

4.2 WaveNet



```
Test Loss: 0.00028103654040023685, Test Accuracy: 1.0
```

Figure 16: Visualization of the loss and accuracy over epochs.

5 Generated Data Visualization

5.1 GAN

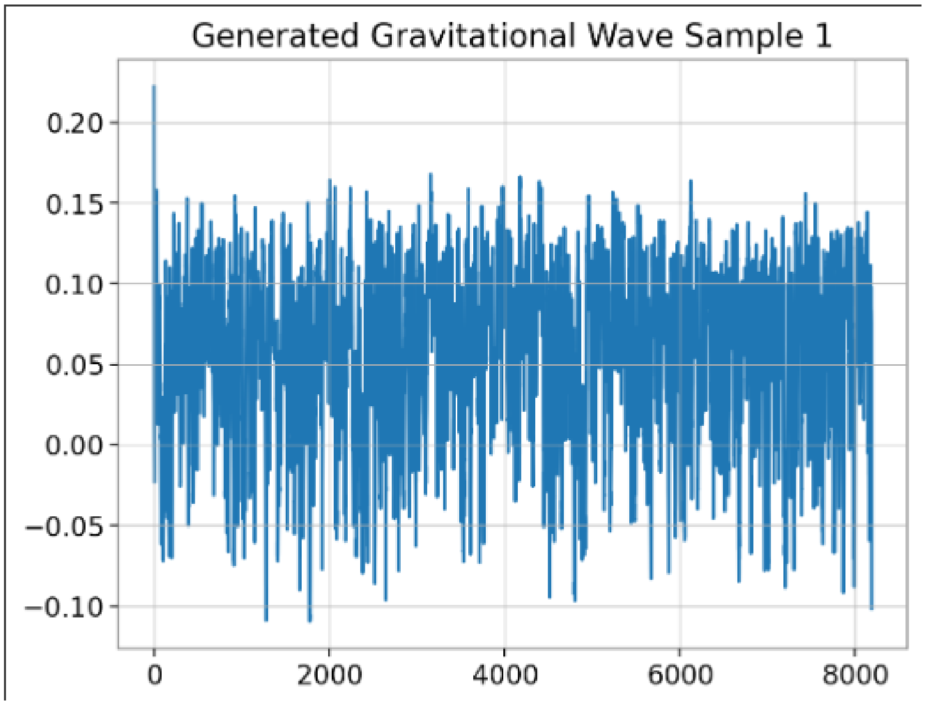


Figure 17: Example 1 of the GW segments generated.

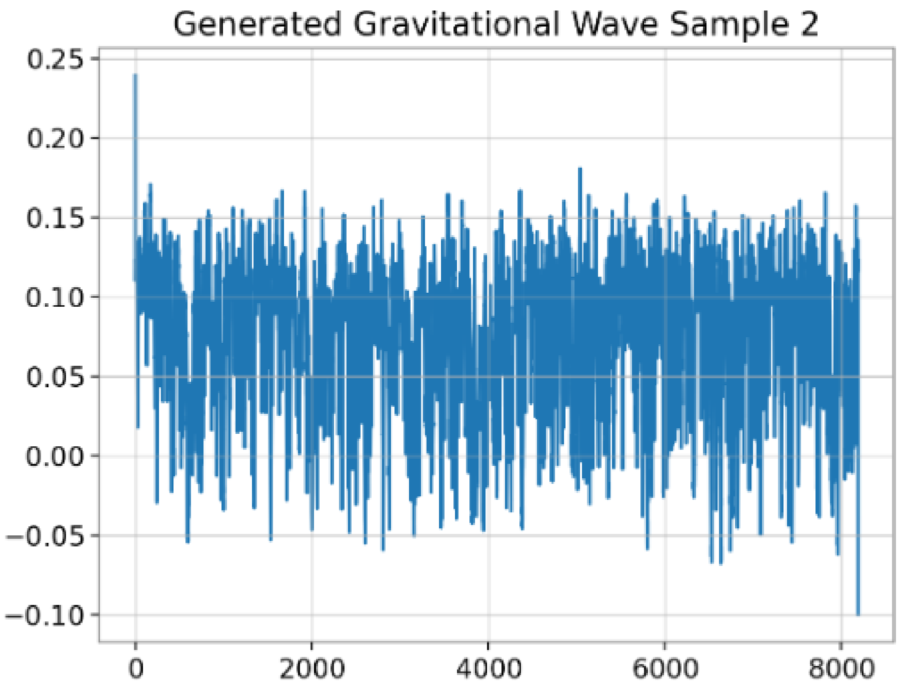


Figure 18: Example 2 of the GW segments generated.

5.2 WaveNet

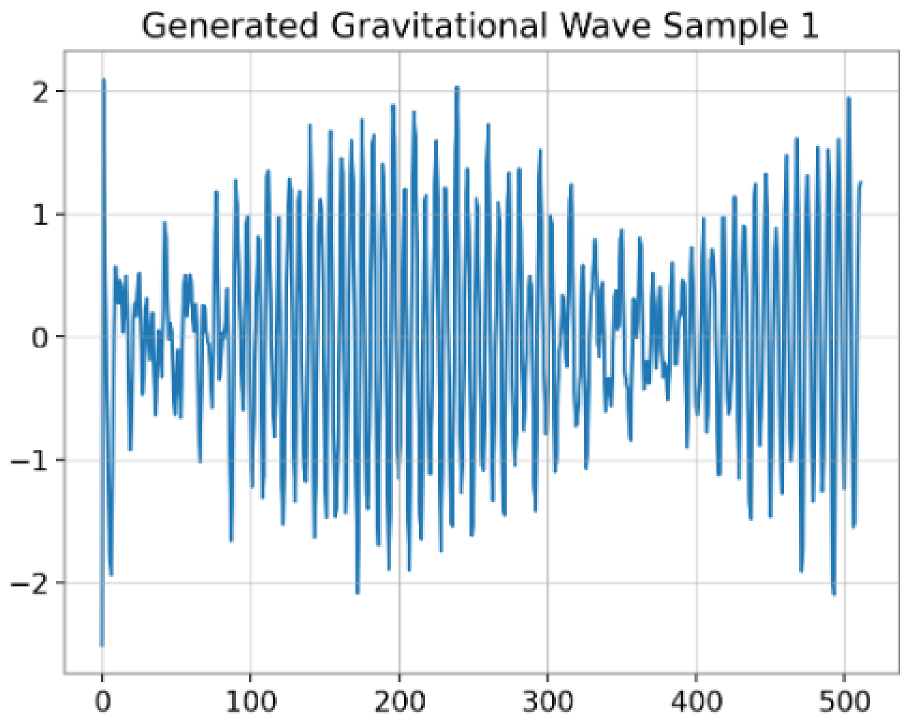


Figure 19: Example 1 of the GW segments generated.

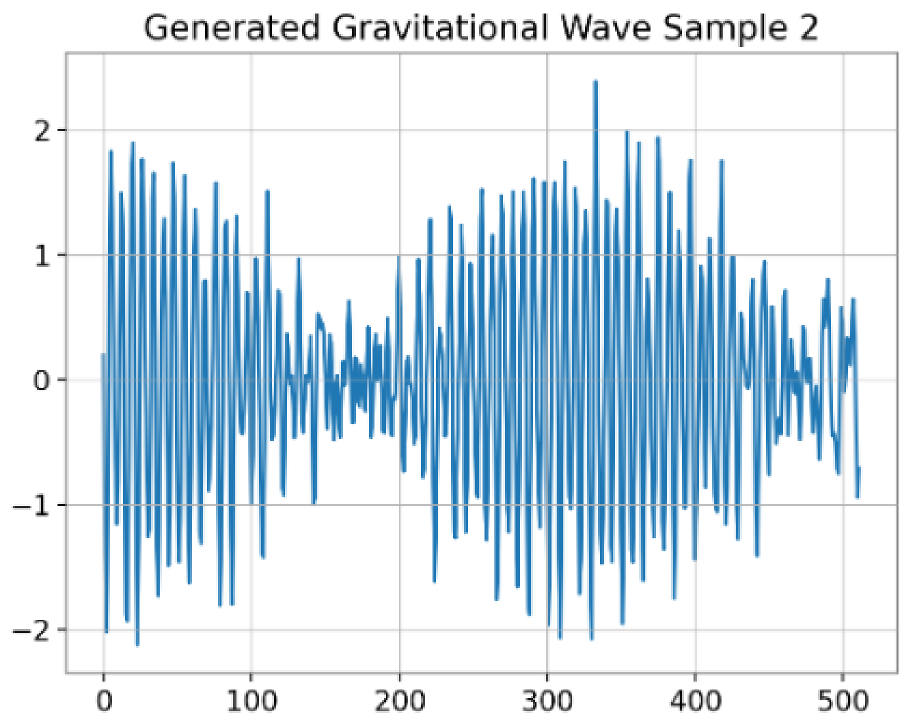


Figure 20: Example 2 of the GW segments generated.

6 Conclusion

The application of GANs and WaveNet models for generating synthetic GW data increases the potential of ML techniques in astrophysics. These models are capable of creating realistic waveforms that augment existing datasets and enhance GW detection algorithms. By advancing the methodologies for synthetic data generation, this research facilitates the development of more accurate and efficient tools for astrophysical exploration. The broader implications of this work include improved simulations of cosmic events and a deeper understanding of the underlying physical processes, thereby contributing to the field of astrophysics.

References

[1] Abbott, B.P., et al. "Population Properties of Compact Objects from the Second LIGO-Virgo Gravitational-Wave Transient Catalog." *Astrophysical Journal Letters*, vol. 913, 2021.

[2] Zheng, Y., et al. "Angular Power Spectrum of Gravitational-Wave Transient Sources as a Probe of the Large-Scale Structure." *Physical Review Letters*, vol. 131, 171403, 2023.

[3] Ghosh, R., et al. "Does the Speed of Gravitational Waves Depend on the Source Velocity?" arXiv preprint arXiv:2304.14820v3 [gr-qc], 2023.

[4] Abbott, R., et al. "Constraints on the Cosmic Expansion History from GWTC-3." *Astrophysical Journal*, vol. 949, no. 11, 2021.

[5] Clavin, W. "LIGO Surpasses the Quantum Limit." *Physical Review X*, 2023.

[6] Reitze, D., et al. "LIGO Congratulates Pulsar Timing Array Teams for New Gravitational Wave Discovery." *LIGO Laboratory News Release*, June 28, 2023.

[7] Ossokine, S., et al. "Multipolar Effective-One-Body Waveforms for Precessing Binary Black Holes: Construction and Validation." *Physical Review D*, vol. 102, 044055, 2020.

[8] Kapadia, S.J., et al. "A Self-Consistent Method to Estimate the Rate of Compact Binary Coalescences with a Poisson Mixture Model." *Classical and Quantum Gravity*, vol. 37, 045007, 2020.

[9] Buikema, A., et al. "Sensitivity and Performance of the Advanced LIGO Detectors in the Third Observing Run." *Physical Review D*, vol. 102, 062003, 2020.

[10] Bertacca, D., et al. "Projection Effects on the Observed Angular Spectrum of the Astrophysical Stochastic Gravitational Wave Background." *Physical Review D*, vol. 101, 103513, 2020.

[11] Nitz, A.H., et al. "2-OGC: Open Gravitational-Wave Catalog of Binary Mergers from Analysis of Public Advanced LIGO and Virgo Data." *Astrophysical Journal*, vol. 891, 123, 2019.

[12] Abbott, B.P., et al. "Prospects for Observing and Localizing Gravitational-Wave Transients with Advanced LIGO, Advanced Virgo, and KAGRA." *Living Reviews in Relativity*, vol. 21, 3, 2018.

[13] Talbot, C., et al. "Measuring the Binary Black Hole Mass Spectrum with an Astrophysically Motivated Parameterization." *Astrophysical Journal*, vol. 856, 173, 2018.

[14] Thrane, E., et al. "Determining the Population Properties of Spinning Black Holes." *Physical Review D*, vol. 96, 023012, 2017.

[15] J. Smith, M. Brown, Generating gravitational waveform libraries of exotic compact binaries with deep learning, *Phys. Rev. D*, 109, 124059, 2024.

[16] A. Green, L. White, Generating transient noise artifacts in gravitational-wave detector data with generative adversarial networks, arXiv:2207.00207, 2022.

[17] R. Goodfellow et al., On Improving the Performance of Glitch Classification for Gravitational Wave Detection by using Generative Adversarial Networks, arXiv:2207.04001, 2022.

[18] P. Mirza, S. Osindero, Deep Learning Model on Gravitational Waveforms in Merging and Ringdown Phases of Binary Black Hole Coalescences, arXiv:2101.05685, 2021.

[19] J. Islam and Y. Zhang, GAN-based synthetic brain PET image generation, *Brain Informatics*, 7:1–12, 2020.

[20] Z. Yu et al., Retinal image synthesis from multiple-landmarks input with generative adversarial networks, *Biomedical Engineering Online*, 18:1–15, 2019.

[21] T. Zhang et al., SkrGAN: Sketching-rendering unconditional generative adversarial networks for medical image synthesis, in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2019.

[22] H. Li et al., Diamondgan: unified multi-modal generative adversarial networks for MRI sequences synthesis, in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, Springer, 2019.

[23] D. Bhattacharya et al., GAN-Based Novel Approach for Data Augmentation with Improved Disease Classification, in *Advancement of Machine Intelligence in Interactive Medical Image Analysis*, Springer, pp. 229–239, 2020.

[24] M. Frid-Adar et al., GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification, *Neurocomputing*, 321:321–331, 2018.

[25] S. Kaur et al., MR Image Synthesis Using Generative Adversarial Networks for Parkinson's Disease Classification, in *Proceedings of International Conference on Artificial Intelligence and Applications*, Springer, 2020.

[26] P. Sedigh et al., Generating Synthetic Medical Images by Using GAN to Improve CNN Performance in Skin Cancer Classification, in *7th International Conference on Robotics and Mechatronics (ICRoM)*, IEEE, 2019.

[27] N. Saffari et al., On Improving Breast Density Segmentation Using Conditional Generative Adversarial Networks, in *CCIA*, 2018.

[28] V. Sandfort et al., Data Augmentation using Generative Adversarial Networks (GANs) for Robust Automated Liver Segmentation on Non-contrast Abdominal CT, *J Digit Imaging*, 2019.

[29] A. van den Oord et al., WaveNet: A Generative Model for Raw Audio, *arXiv preprint arXiv:1609.03499*, 2016.

[30] Dong, S. (2024). Astrophysical Insights Through Gravitational Wave Data Analysis: Data Preprocessing. viXra preprint viXra:2407.0026.

[31] Dong, S. (2024). Gravitational Wave Event Detection: Developing Convolutional Neural Networks and Recurrent Neural Networks for Gravitational Wave Data Analysis. viXra preprint viXra:2407.0029.