Gravitational Wave Detection: Find gravitational wave signals from binary black hole collisions

Caroline Sodré

December 2022

1 Objetivo

O objetivo é propor uma solução "simples" porém eficiente para o desafio apresentado em https://www.kaggle.com/competitions/g2net-gravitational-wave-detection/data. O problema consiste em calcular a probabilidade de um conjunto contendo medições simuladas de ondas gravitacionais de 3 interferômetros de ondas gravitacionais (LIGO Hanford, LIGO Livingston e Virgo) conter apenas ruído (target =0) ou conter ondas gravitacionais e ruído (target=1).

2 Introdução

As ondas gravitacionais, previstas por Albert Einstein em 1916 como consequência de sua teoria da relatividade, são "ondulações" invisíveis no tecido do espaço-tempo. Viajando na velocidade de luz, essas ondas "espremem" e "esticam" qualquer coisa em seu caminho enquanto passam. Ondas gravitacionais detectáveis são criadas principalmente pelo movimento em altas velocidades de objetos muito massivos, como como buracos negros ou estrelas de nêutrons orbitando uns aos outros. Ondas gravitacionais criadas por dois buracos negros em colisão foram detectadas pela primeira vez em 2015 pelo LIGO (Laser Interferometer Gravitational-Wave Observatory), e abriram uma nova janela para o Universo nos permitindo aprender coisas novas sobre fenômenos como buracos negros, estrelas de nêutrons, cosmologia etc. Apesar de termos acesso a detectores extremamente sensíveis, os sinais acabam se misturando com ruídos do detector, fazendo com que a análise e tratamento desses dados sejam de grande importância.

3 Estratégia adotada

A estratégia adotada foi a seguinte: primeiramente, por existem algumas diferenças evidentes entre um sinal de ondas gravitacionais e os ruídos do detector no espaço de frequência (arXiv:1712.00356), utilizamos o algoritmo Constant Q

Transform oferecido pelo módulo nnAudio (CQT1992v2) para transformar os dados disponibilizados para o espaço de frequência para que fosse possível o fazer o Feature Extraction de tais imagens. Após isso, um modelo simples de Fine-Tuning (uma técnica de Transfer Learning) foi criado utilizando o Keras e o EfficientNet.

3.1 Feature Extraction

O método de Feature Extraction em imagens se baseia no processo de identificar as características mais importantes dessas imagens e transformá-las em features numéricos compactos que podem ser processados mais facilmente. Como estamos lidando com um grande conjunto de dados, o Feature Extraction ajuda a reduzir efetivamente a quantidade de dados e aumenta a velocidade das etapas de aprendizado.

3.2 Transfer Learning e Fine-Tuning

A ideia por trás do Transfer Learning para classificação de imagens é a de que, se um modelo for treinado em um conjunto de dados grande e geral o suficiente - extraindo features que podem ajudar a identificar arestas, texturas, formas e composição de objetos - esse modelo servirá efetivamente como um modelo genérico, podendo então ser utilizados em outros conjuntos de dados - o que é justamente o caso do EfficientNet, como veremos a seguir.

O Fine-Tuning consiste em construir um modelo seguindo os seguintes passos: Pegue camadas de um modelo previamente treinado excluindo o output layer. Congele-as, para evitar a destruição de qualquer informação que eles contenham durante as próximas rodadas de treinamento. Adicione um novo output layer sobre o modelo pré-treinado para que o novo modelo possa aprender a transformar os features antigos em previsões em para um novo conjunto de dados. Treine o novo modelo em seu conjunto de dados. Após isso, Descongele todo o modelo obtido acima (ou parte dele) e treine-o novamente nos novos dados. Como deseja-se readaptar os pesos pré-treinados, é importante utilizar uma taxa de aprendizado menor nesta etapa. Caso contrário, seu modelo pode sofrer overfitting.

4 Ferramentas utilizadas

4.1 nnAudio.Spectrogram.CQT1992v2

O nnAudio é uma ferramenta PyTorch (uma biblioteca Python para computação científica capaz de realizar cálculos utilizando tensores e construir redes neurais) para processamento de áudio que pode calcular diferentes tipos de espectrogramas em tempo real. Atualmente, o nnAudio suporta o cálculo de diferentes espectrogramas, contando com a função nnAudio.Spectrogram.CQT1992v2, que calcula o Constant Q Transform (CQT) de um dado sinal no espaço do tempo,

gerando uma representação de tempo-frequência em que os bins de frequência são espaçados geometricamente.

4.2 Keras

Keras é um neural network API (API é uma interface que possibilita a comunicação e troca de informações entre diferentes sistemas, permitindo assim o cruzamento de dados e gatilhos entre os softwares) desenvolvido pelo Google para construir modelos de machine learning. Escrito em Python, seu objetivo é tornar mais fácil o processo de implementação de redes neurais. O Keras tem forte adoção tanto na indústria quanto na comunidade de pesquisadores, sendo considerável extremamente beginner-friendly. Além disso, ele é incorporado ao TensorFlow e pode ser usado para realizar deep learning rapidamente, pois fornece módulos embutidos para todos os cálculos de redes neurais.

4.3 EfficientNet

Ao lidar com problemas de classificação desafiadores e havendo recursos computacionais limitados, é necessário escalar adequadamente as dimensões envolvidas no projeto da arquitetura da rede neural convolucional. Nesses cenários de limitação de recursos, quanto maior a dimensão que a rede neural convolucional puder atingir, maior tende a ser o seu desempenho. Ao escalar as dimensões envolvidas no projeto da arquitetura da rede neural convolucional, é necessário levar em conta a existência de uma interdependência entre as três as dimensões da rede neural: largura (número de neurônios do layer), profundidade (número de layers) e resolução (densidade de pixels da imagem de input).

A família de modelos EfficientNet é um conjunto de redes neurais convolucionais do GoogleAI que podem ser usadas como base para uma variedade de tarefas de visuais, embora tenham sido inicialmente projetadas para classificação de imagens. Essa família de modelos foi projetada de forma a balancear com eficiência todas as dimensões da rede utilizando um compound coefficient, sendo assim capaz de alcançar maior precisão e eficiência. Isto é feito da seguinte maneira: a primeira etapa é realizar um grid search (método baseado na procura pelos hiperparâmetros que irão gerar os melhores resultados - para modelos de redes neurais, alguns parâmetros são taxa de aprendizado, número de camadas, quantidade de nó em cada camada entre outros) para encontrar a relação entre as diferentes dimensões da rede base sob uma capacidade de processamento fixa. Com isso, somos então capazes de determinar o coeficiente de escala apropriado para cada uma das dimensões mencionadas acima. Com base nesta estratégia de escalonamento das redes foram criadas oito versões desta arquitetura intituladas EfficientNetB0 (arquitetura-base) a EfficientNetB7 (obtidas escalando a arquitetura-base).

O EfficientNet nos permite formar features a partir de imagens que podem ser passadas posteriormente para um classificador (um tipo de algoritmo de machine learning usado para atribuir um label a um input), permitindo assim ser usada como um feature extractor network. Podemos implementar esse modelo

de classificação de imagem de maneira simples utilizando o Keras, onde temos a opção de carregar os pesos do modelo que já foram pré-treinados utilizando o ImageNet (um grande banco de dados ou conjunto de dados com mais de 14 milhões de imagens organizadas e rotuladas em uma hierarquia).

5 Workflow

1. Training Data Analysis: Aqui, entenderemos com que tipo de dados estamos lidando;

2. Spectrogram: Em seguida, construiremos uma função para obter o espectrograma dos dados nos certificando de os gerarmos no formato correto (3D);

3. DataGenerator: Para lidar com o grande volume de dados, construiremos uma função capaz de dividi-los em batches, os carregando na RAM quando necessário;

4. Model: Com os dados prontos para serem passados à rede, passaremos para a construção do modelo. Aqui, definiremos a arquitetura do modelo, assim como seu optimizador, loss function etc. Após o compilarmos, o treinaremos utilizando um train dataset e um validation dataset (slice do training data original) de acordo com o método de Fine-Tuning;

5. Predicting: Finalmente, caso o treinamento retorne bons resultados, seremos capazes de prever a probabilidade de cada file pertencente ao test data conter ou não o sinal simulado de ondas gravitacionais.

6 Conclusão