UNIVERSIDADE D
COIMBRA

Gonçalo Ferreira Ferrão Castel-Branco

IDENTIFICAÇÃO DE INSTRUMENTOS MUSICAIS EM MÚSICA POLIFÓNICA

Dissertação no âmbito do Mestrado integrado em Engenharia Eletrotécnica e de Computadores, especialização em computadores orientada pelo Professor Doutor Fernando Manuel dos Santos Perdigão e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2019

Agradecimentos

Quero deixar um sincero agradecimento a quem, de uma forma ou de outra, fez com que a realização deste trabalho se tornasse possível.

Ao Professor Doutor Fernando Manuel dos Santos Perdigão pela disponibilidade, apoio e sobretudo pela confiança depositada em mim.

À minha Mãe pela insistência, constante preocupação e valores transmitidos não só ao longo da realização deste projeto mas também durante todo o meu percurso académico. Ao meu Pai pela tranquilidade, positivismo e interesse constantes.

Aos meus irmãos João e Inês, pela companhia, boa disposição e humor que sempre transmitiram.

À minha namorada, Sofia, por toda a paciência, carinho e incentivo que sempre me deu desde o início.

A todos os meus amigos pelo companheirismo e por todos os momentos partilhados.

Por fim, às duas pessoas que criaram as melhores condições possíveis e fizeram de tudo para que este projeto se concretizasse, ao meu Avô Daniel e à minha Avó Maria Helena.

Resumo

A identificação de instrumentos musicais continua um desafio por resolver na área de investigação em música, geralmente referida como Music Information Retreival (MIR). Este problema, fundamental para campos como a pesquisa por áudio, reconhecimento de género musical, recomendação de música, ou a identificação de plágio, será abordado tendo em conta diversos métodos.

A seguinte dissertação de mestrado apresenta um sistema de identificação de instrumentos que tem por base uma pequena parte da base de dados AudioSet [10] com sons de instrumentos e que propõe o reconhecimento de áudio com base em imagens, neste caso espetrogramas de mel, que representam o som que se pretende classificar.

O OpenMic-2018 (OM18) [17] é uma base de dados (BD) que surge no seguimento do AudioSet e com os mesmos ideais, mas direcionada para 20 classes de instrumentos musicais. Esta base de dados, publicada recentemente, conta ainda com poucos trabalhos que a abordem. Tentar-se-á superar os resultados já apresentados tanto através de abordagens originais como através de abordagens publicadas para o AudioSet. Trabalhos muito recentes utilizam modelos de atenção para classificar os exemplares do AudioSet e revelaram resultados muito positivos, pelo que também serão tidos em conta ao longo do projeto para a BD OM18.

No âmbito do presente trabalho foi criada uma nova base de dados, **PureMic**, que tem por base as duas bases de dados já referenciadas. Esta é uma base de dados cujos exemplares são mais precisos e escolhidos de forma rigorosa, para poder contribuir para o classificador em tempo real e para uma melhoria das etiquetas do OM18, base de dados que ainda tem alguma falta de informação nesse aspeto.

A seguinte dissertação faz então um resumo das abordagens a ser consideradas nomeadamente a implementação de redes neuronais convolucionais, muito utilizadas nesta área. Serão utilizadas as três bases de dados já referidas que providenciarão uma grande e suficiente quantidade de dados devidamente identificados.

Palavras-chave: Identificação de instrumentos Músicais, Music information retreival, redes neuronais convolucionais, OpenMIC-2018, AudioSet, PureMic, Modelos de Atenção, Espectograma Mel.

Abstract

Musical instruments recognition remains an unsolved challenge in Music Information Retreival (MIR). This problem, wich is fundamental for fields such as audio research, music genre recognition or music recommendation will be addressed with a variety of methods.

This Master's dissertation presents an instrument identification system that is based on a small portion of AudioSet dataset [10] with 20 musical instrument classes. This dataset proposes the recognition of audio events based on image inputs wich are log mel spectograms of sound events.

OpenMic-2018 (OM18) [17] is a dataset that extends the reach of AudioSet but targeted to only 20 classes of musical instruments. There are several publications arround AudioSet research. Since OpenMic is similar to AudioSet, some methods used in AudioSet will be aplied in OM18.

In the context of this work, a new dataset was created, **PureMic**, based on the two datasets already referenced. This is a dataset whose audio clips are more accurate and rigorously chosen to be able to contribute to the real-time classifier and to the improvement of OM18 labels.

The following dissertation summarizes the approaches to be considered namely the implementation a convolutional neural networks, widely used in this area. AudioSet, OpenMic-2018 and PureMic, will proveide a large and sufficient amount of properly identified data.

Keywords: Musical instruments recognition, Music information retreival, Concolutional Neural Networks, OpenMIC-2018, AudioSet, PureMic, Atention Models, Mel spectogram.

Conteúdo

Lista de Figuras

Siglas

RF	Random Forest
BD	Base de Dados
CNN	Convolutional Neural Network
DFT	Discret Fourier Transform
DL	Deep Learning
STFT	Short Time Fourier Transform
FFT	Fast Fourier Transform
MIL	Multiple Instance Learning
MIR	Music Information Retreival
ML	Machine Learning
NN	Neural Network (Rede Neuronal)
OM18	OpenMic-2018
PCA	Principal Component Analysis
PCM	Pulse Code Modulation
SED	Sound Event Detection
SLD	Strongly Labelled Dataset
TF	TensorFlow
WLD	Weakly Labaled Dataset

Capítulo 1

Introdução

A música é uma forma de arte omnipresente e essencial para milhões de pessoas por todo o mundo. Não é conhecida nenhuma civilização ou agrupamento que não possua manifestações musicais próprias. Criações e atuações musicais são um dos nossos artefactos culturais mais complexos e o poder emocional da música, pode tocar o ser humano de formas surpreendentemente profundas. Mesmo sem estudar música, o ser humano consegue identificar, naturalmente, timbres, tons, relação harmónica entre notas musicais, emoções, entre muitas outras características. Para isso, o nosso cérebro realiza uma grande quantidade de processos complexos que compilam o áudio ouvido em informação que nos é útil, como por exemplo distinguir fado de rock ou distinguir uma guitarra de um piano.

1.1 Motivação

Entre muitas áreas que abrangem esta arte, a música ao vivo é uma área que requer muito rigor. Para que o público possa ouvir e sentir uma atuação da forma que se pretende, são necessários vários estudos antes e durante o espetáculo tanto por parte do engenheiro de som, responsável pelo controlo de todos os aspetos sonoros, como por parte do artista, que em conjunto decidem os pormenores que aos ouvidos da plateia farão a diferença. O processo responsável pela preparação prévia do som de todo o espetáculo chama-se Sound Check.

O Sound Check é um procedimento necessário antes de qualquer tipo de atuação no qual é decidida a melhor configuração possível de forma a que o som reproduzido para a plateia seja limpo, com o volume adequado e com as misturas e balanços tonais corretos. Todas estas configurações são feitas com uma *mixing console* (mesa de som). Essencial para qualquer artista, o Sound Check é normalmente realizado sem audiência uma vez que é um processo demorado e mais técnico. Todos os instrumentos estão ligados à mesa de mistura, cada instrumento com o seu canal (microfone ou ligado diretamente). Para identificar cada instrumento e o associar ao respetivo canal, o engenheiro de som deve ouvir um instrumento de cada vez para assim, etiquetar todos os canais (Figura 1.1).

O volume é a característica de um canal que o público identifica com mais facilidade (é fácil dizer que a guitarra está baixa face aos outros instrumentos no decorrer de um concerto) mas para que o som ouvido seja de qualidade, existem muitas outras propriedades (Gain, fader, compressão, graves, etc.) que devem ser consideradas e ajustadas antes e se necessário durante uma atuação.

Ajustar todas estas propriedades para cada canal é uma tarefa que pode ser difícil de resolver principalmente quando o rigor é essencial. Para além de dever ajustar em tempo real todas estas propriedades, o engenheiro de som deve ser capaz de as ajustar de acordo com o que a sua experiência lhe sugere mas também deve ter em conta os requisitos do artista que naturalmente varia consoante o seu gosto.

Quando um espetáculo começa, a acústica da sala é alterada devido à sua ocupação por parte do público e consequentemente o som ouvido não é o mesmo. Este é um caso previsível mas pode haver muitas outras causas que implicam a mesma consequência e que podem não ser tão claras aos ouvidos de um engenheiro de som.

Todos estes processos, para além de muito dispendiosos são complexos, demorados e exigem muita experiência por parte do engenheiro de som pelo que resolver este problema com a ajuda de tecnologia traria sempre vantagens para o objetivo final que é sempre garantir o melhor som possível ao público. Para isso, é necessário o estudo de dois temas principais, a **identificação do instrumento de cada canal** e posteriormente, o **ajuste das propriedades de cada canal em tempo real** que variam consoante o instrumento identificado.

A motivação para este projeto parte precisamente da ideia de aperfeiçoar todos estes processos de forma a minimizar erros, diminuir a demora dos processos e aumentar a sua precisão em tempo real.

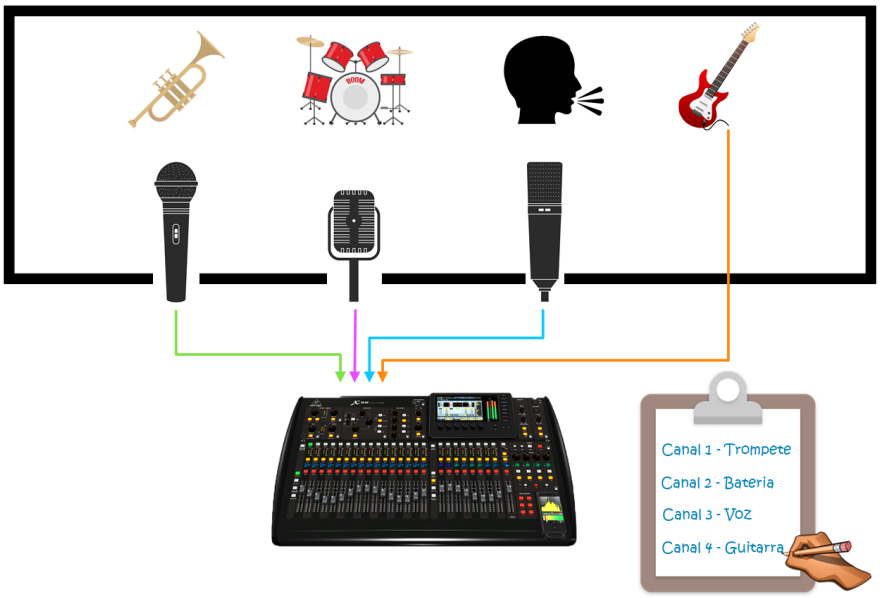

Figura 1.1 Identificação de instrumentos num caso prático

1.2 Objetivos

Esta dissertação consiste no estudo e investigação da identificação de instrumentos em tempo real, primeira das duas partes da ideia que motivou a escolha deste tema. O objetivo principal passa pelo estudo de técnicas computacionais, em particular a aprendizagem automática (Machine Learning), para identificar instrumentos.

Machine Learning (ML) é uma área em crescimento que rapidamente atingiu resultados nunca antes conseguidos em campos como o reconhecimento de imagem ou o reconhecimento da fala. No que diz respeito à identificação de instrumentos também existem alguns trabalhos com abordagens semelhantes mas também existe espaço para os melhorar ou considerar outras abordagens para resolver este problema.

ML envolve diversas técnicas, sendo a escolha da arquitetura certa para uma dada tarefa, uma das tarefas mais complexas. Isso requer o estudo e preparação de todos os dados disponíveis para assim, chegar à arquitetura que melhor satisfaz os requisitos. A utilização deste tipo de técnicas é tal que faz com que apareçam novos e melhores resultados muito regularmente pelo que é difícil estar sempre a par daquela que supostamente será a técnica mais adequada ao problema.

Para atingir o principal objetivo, será então necessário o estudo de muitas das técnicas utilizadas no ML desde a sua raiz até às abordagens mais complexas para no final poder apresentar um processo completo capaz de identificar instrumentos em tempo real.

1.3 Contextualização

Esta dissertação será então escrita no contexto do Music Information Retreieval (MIR), uma área de investigação que tem vindo a crescer ao longo dos anos. Tal como o nome indica, esta área tem como principal objetivo extrair informação da música e para isso são necessários conhecimentos em diversas áreas como musicologia, psicoacústica, psicologia, teoria musical, processamento de sinal, computação, aprendizagem automática entre muitas outras áreas que unem a engenharia e a música. Com o crescimento do MIR surgiu então a sua sociedade internacional, International Society for Music Information Retreival (ISMIR) [20] e a sua conferencia que veio servir de plataforma para trabalhos da área e assim uma maior divulgação tanto para investigação como para a indústria. Esta sociedade veio também facilitar a troca de conhecimentos e a propagação de tecnologia numa área mais específica. Apesar de já existirem conferências mais conceituadas e com muitos temas semelhantes, como o ICASSP [18] (International Conference on Acoustics, Speech, and Signal Processing) ou o Digital Audio Effects [3] (DAFx) estas não têm como tema principal a música.

A identificação de instrumentos é um problema ainda não completamente resolvido no MIR, motivo que torna este projeto ainda mais aliciante. Pretende-se contribuir para a resolução futura deste problema.

1.4 Contribuições

No final deste projeto, espera-se então contribuir com:

- um sistema de identificação de instrumentos musicais em tempo real

- uma base de dados de sons de instrumentos musicais pura e original com base no Audioset [10] e no OpenMic-2018 [17].
- Contribuição para o OpenMic-2018 através da desambiguação das suas etiquetas.

1.5 Estrutura

Os restantes capítulos do documento serão organizados da seguinte forma: o segundo capítulo aborda todos os fundamentos teóricos necessários para o trabalho e também o estado da arte da identificação de instrumentos fazendo um resumo de diversos trabalhos já realizados.

No capítulo 3, é feita uma preparação para o desenvolvimento ou seja, são abordados conceitos mais práticos, que não se enquadram no capítulo 2. Para além disso, é neste capítulo que se fala com mais pormenor das bases de dados utilizadas.

O capítulo 4 apresenta o trabalho desenvolvido em conjunto com alguns resultados que fazem mais sentido ser apresentados durante algumas implementações.

No capítulo 5 é feito um rescaldo e análise aos resultados obtidos de uma forma mais resumida e objetiva.

Por fim, o capítulo 6 conclui este projeto falando sobre o que foi aprendido e o trabalho que ainda se pode realizar em volta desta dissertação.

Capítulo 2

Estado da arte

2.1 Perceção do ouvido humano

Para ensinar uma máquina como ouvir e entender sons ou música, é necessário antes, perceber como é que o ser o humano o faz.

Uma das características mais espantosas do ser humano é a capacidade de criar representações internas através do processamento de diversas fontes de informação provenientes dos nossos sentidos. Este processo de tradução consiste na criação de uma representação da informação que foi recebida (no caso deste projeto será a informação ouvida), que seja capaz de ser interpretada pelo nosso cérebro. Esta representação abstrata para o ser humano, é também essencial para uma máquina poder converter uma gravação em algo que consiga interpretar.

O ouvido humano é sensível a uma vasta extensão de padrões, desde o som de um relógio a trabalhar até ao som produzido durante um concerto de rock por exemplo. É particularmente sensível a sons na extensão de frequências que a voz pode produzir (300 a 3000Hz) apesar de ser capaz de captar frequências até aos 20KHz mas com menos sensibilidade para esta gama. Este aspeto pode ser visto como uma evolução do ser humano uma vez que um dos estímulos mais captados pelo seu ouvido foi e é a voz humana.

Segundo McAdams (1993) [26] o modelo percetual do ouvido humano consiste nos seguintes mecanismos:

1. Transdução sensorial: A primeira etapa consiste na representação do sinal acústico no sistema auditivo. Por outras palavras, nesta fase, as vibrações presentes no ar que são captadas pelo ouvido, são codificadas em impulsos nervosos que serão interpretados por processos percetuais mais complexos.

2. Agrupamento auditivo: Consiste em agrupar diferentes fontes de áudio. Quando ouvimos um determinado som, este pode ser constituído por diversas fontes (como por exemplo numa cidade ouvir pessoas a conversar e carros a trabalhar) que juntas formam o som ouvido. Esta etapa consiste na separação e agrupamento dessas fontes.

3. Propriedades e características auditivas: Uma vez separada a informação sensorial em fontes de som, são acionados uma série de processos que progressivamente analisam as propriedades

consideradas relevantes ao ouvido num dado momento. Esta análise pode extrair informação tanto de curtos (mili-segundos) como de longos (segundos) intervalos de tempo.

4. Associar as propriedades auditivas a representações na memória: Esta etapa consiste então em recorrer à memória para assim decidir e reconhecer o som ouvido. O estímulo é reconhecido tendo em conta a classe que melhor corresponde à representação auditiva, isto é, o som que é ouvido é comparado com os sons conhecidos e a classe com "maior ativação"(mesmo que nunca tenha sido ouvido um som rigorosamente igual) é a classe correspondente ao som.

Tal como vamos poder verificar, existem muitas semelhanças entre o método biológico e o método computacional, facto interessante e que pode ajudar na implementação do sistema de identificação de instrumentos musicais.

Figura 2.1 Modelo percetual do ouvido Humano [27]

2.2 Representação de sinais de áudio

Esta secção explica vários conceitos de processamento de sinal necessários para que um computador possa interpretar um sinal de áudio. O objetivo é obter a melhor representação do sinal de áudio (feature) para que, neste caso, seja possível distinguir instrumentos musicais.

Os sinais de áudio representam a variação da pressão acústica do ar que é captada e interpretada pelo sistema auditivo. Podem ser captados por microfones e convertidos para amostras digitais em sistemas de aquisição de áudio. A informação contida nos sinais de áudio pode ainda ser comprimida com codificadores de áudio, explorando as características do sistema auditivo humano. É o caso dos codificadores MP3, ACC, OGG, etc. No caso de áudio não comprimido é regularmente usado formato WAVE, onde apenas se especifica a frequência de amostragem e o formato das amostras. Do ponto de vista computacional, é difícil de extrair informação capaz de identificar o conteúdo do áudio neste tipo de formato. Um dos desafios do MIR passa por transformar estes sinais não codificados em representações bidimensionais de tempo e frequência.

Short Time Fourier Transform

Para obter uma representação na frequência de um sinal discreto é comum usar a *Discret Fourier Transform* (DFT). Contudo, esta representação é geralmente relativa a todo o sinal. Para identificar o conteúdo espetral do sinal **ao longo do tempo** é usada a *Short time Fourier Transform* (STFT). Para obter a STFT é necessário dividir o sinal em tramas (ou *frames*) com um tamanho suficientemente pequeno (*short time*) que permita assumir que as frequências neste intervalo são estacionárias. Se o sinal de áudio for simplesmente dividido em tramas antes de aplicar a DFT (significa multiplicar por uma janela perfeitamente retangular de amplitude unitária), pode acontecer um fenómeno chamado *leakage* espectral. Este fenómeno, representado na figura 2.2, acontece porque uma trama com um sinal periódico, pode não conter um número inteiro de períodos e a DFT cria uma variação brusca do fim do sinal para o início do novo período espalhando assim, esta variação temporal nas frequências vizinhas à frequência original do sinal.

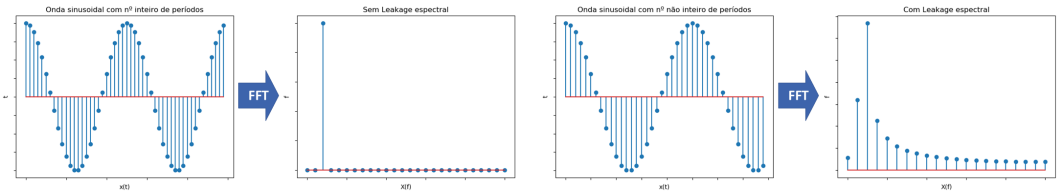

Figura 2.2 Figura da esquerda sem Leakage espectral e figura direita com Leakage espectral

Este fenómeno dificulta a identificação das frequências do sinal e por esse motivo, as tramas são multiplicadas por uma janela que atenua este efeito causado pelos limites de uma janela retangular. Estas janelas, neste caso de Hann (poderiam ser outras como as de Hamming), devem ser sobrepostas tal como mostra a figura 2.3 uma vez que, caso contrário haverá menos suavização entre conteúdos espetrais sucessivos. Depois de multiplicar todo o sinal pelas janelas, é calculada a Fast Fourier transform (FFT[1]) para cada trama. O resultado desta análise é chamado de Short-Time Fourier-Transform (STFT), também conhecido por sonograma e é visualizado usualmente com pseudocores resultando um espetograma linear na frequência. Esta técnica é uma das mais comuns no que diz respeito a representações espetrais.

[1] Algoritmo eficiente para calcular a DFT

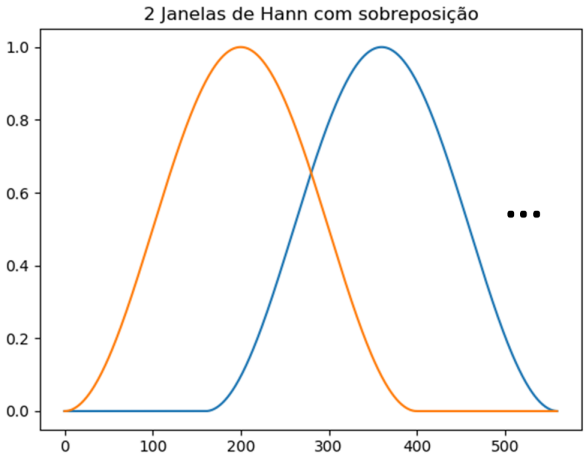

Figura 2.3 Janelas de Hann sobrepostas

Escala Mel

A escala de Mel tenta simular a perceção auditiva humana que é mais sensível a variações nas baixas frequências e portanto as larguras de banda percetuais (filtros) deverão crescer com a frequência (figura 2.4). Para isso, utilizando um banco de M filtros, as frequências da STFT podem ser mapeadas na escala de Mel correspondente. A conversão entre Mel e Hertz pode ser calculada através da equação,

$$m = 2595 \log\left(1 + \frac{f}{700}\right) \tag{2.1}$$

onde m é a frequência em escala Mel e f, a frequência em Hz. Um banco de filtros é um conjunto de filtros passa-banda que separam o sinal de entrada em vários canais, cada um correspondente a uma banda de frequência específica.

Coeficientes Cepstrais em escala Mel (MFCC)

Uma vez que os coeficientes do banco de filtros estão altamente correlacionados, é habitual aplicar uma DCT (Discrete Cosine Transform) ao logaritmo das energias nos vários canais do banco de filtros. O resultado são os coeficientes MFCC (Mel-frequency Cepstral Coefficients) , que descorelacionam e representam de forma comprimida o banco de filtros.

Tipicamente, para o reconhecimento de locutor, os coeficientes considerados são do intervalo [2,13] (figura 2.5) sendo que os restantes são descartados porque representam mudanças demasiado rápidas no espetograma não contribuindo para o reconhecimento de locutor. Apesar dos parâmetros MFCC não serem usados no âmbito desta tese, serão abordados algumas vezes no que diz respeito aos trabalhos desenvolvidos no âmbito da identificação de instrumentos musicais.

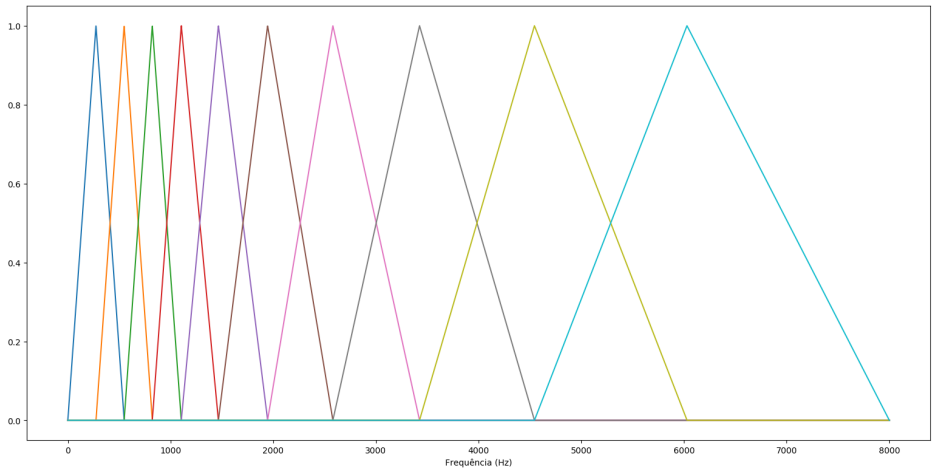

Figura 2.4 Filtros Mel na Frequência em Hz

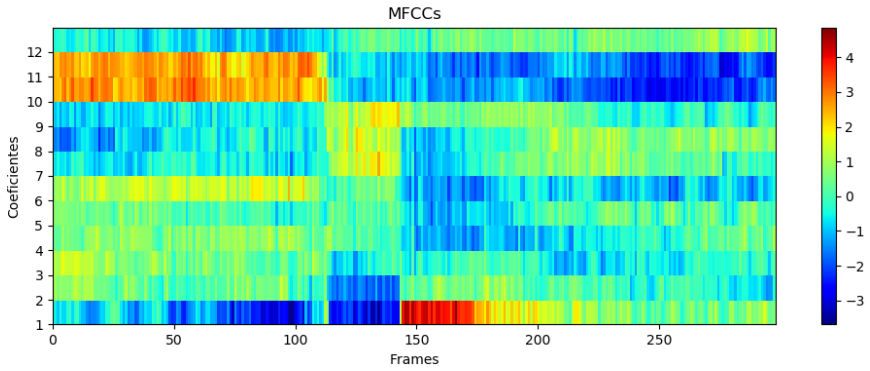

Figura 2.5 Coeficientes MFCC ao longo do tempo

2.3 Aprendizagem Automática

Nas ciências da computação e em campos relacionados, as redes neurais artificiais (ANNs) são modelos computacionais inspirados no sistema nervoso central de um animal (em particular o cérebro) que permitem a uma máquina não só aprender como também reconhecer padrões. O nome "Aprendizagem automática"vem do facto de estas redes (ANNs) terem a capacidade de ser ensinadas. A grande vantagem é que podem aprender a identificar problemas não lineares.

Quando ouvimos um piano, há neurónios no nosso cérebro que são estimulados e enviam sinais para outros neurónios que por sua vez continuam a enviar sinais entre outros neurónios até chegar um sinal ao neurónio responsável por dizer se o som ouvido era ou não um piano. As redes neuronais artificiais tentam simular este processo construindo redes de diversos neurónios com o objetivo de identificar, por exemplo, um instrumento num excerto de áudio. Por outras palavras, a "superfície de decisão"não é uma linha (como é o caso, por exemplo, de uma regressão linear).

2.3.1 Redes Neuronais

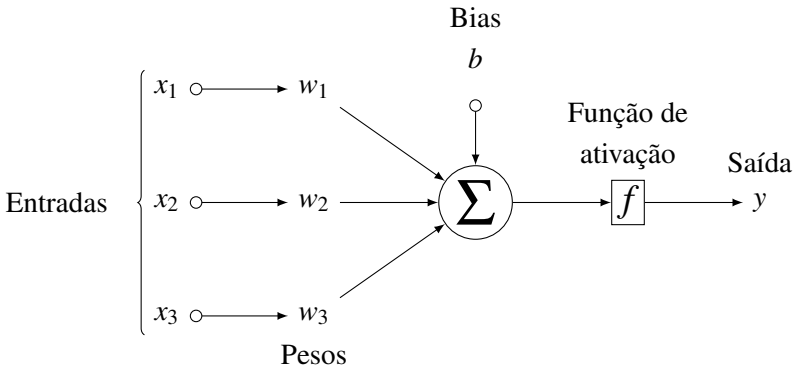

Figura 2.6 Perceptrão

A figura 2.6 representa um perceptrão. w_n representa as entradas e y representa uma soma ponderada dos pesos de cada entrada **caso a função de ativação seja** $f(x) = x$.

Tal como se pode ver na figura 2.7, foi adicionada uma camada a amarelo entre a entrada e a saída, chamada camada escondida (*hidden layer*). Cada um dos nós desta camada corresponde à soma ponderada dos pesos da entrada a verde e a saída corresponde à soma ponderada dos pesos dos neurónios amarelos. As redes neuronais são capazes de resolver problemas com superfícies de separação não lineares e para isso é necessário introduzir uma não linearidade.

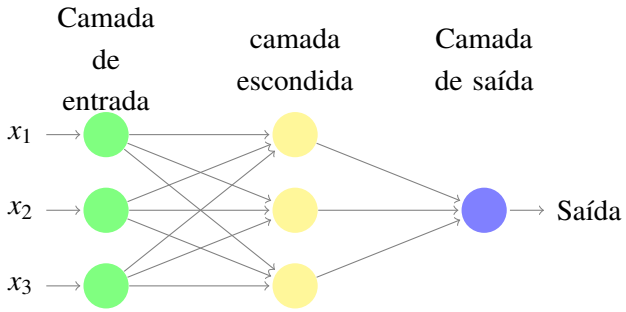

Figura 2.7 Modelo Multi-layer Perceptron (MLP)

A estrutura de um neurónio (figura 2.6) tem na sua saída uma função que no desenho convencional de redes neuronais é omitida (figura 2.7). Esta função, mais conhecida por função de ativação, é responsável por introduzir a não linearidade pretendida. Todos os nós das camadas intermédias são transformados por uma função de ativação antes de passar para as somas ponderadas da camada seguinte.

Existem várias funções não lineares neste tipo de redes sendo a função sigmoide e a função ReLU (rectified linear unit)(figura 2.8) das mais utilizadas. A derivada da função sigmoide, essencial para o algoritmo responsável pela aprendizagem da rede, satura rapidamente nos seus limites tendendo para zero. Por esta razão a rede neuronal pode estagnar na aprendizagem e portanto uma solução é substituir a função sigmoide pela ReLU que tem a sua derivada igual à entrada para entradas positivas e zero no caso de entradas negativas. Para além destas funções de ativação, a função **softmax** também é muito utilizada na camada de saída.

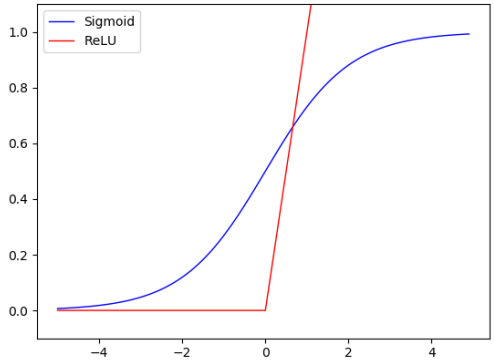

Figura 2.8 Funções de ativação mais comuns

Supondo que $f()$ representa uma função de ativação (ver a seguir), o valor de um neurónio na rede neuronal é dado pela formula,

$$f(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.2}$$

onde w é o peso do neurónio, x a sua entrada e b, mais conhecido por polarização (*bias*) desloca a função de ativação para um dos lados funcionando como um *threshold* ou limiar para obter diferentes valores na saída do nó.

Assumindo agora que todos os neurónios têm a sua função de ativação e adicionando ainda mais *hidden layers* à rede, o modelo começa a ganhar a capacidade de encontrar relações muito complexas e não lineares entre a entrada e a suposta saída. Cada camada tenta chegar ao melhor modelo tendo em conta apenas a sua entrada (camada anterior). Na prática, as camadas iniciais estabelecem relações gerais como picos espetrais ou subidads e descidas no tempo. As camadas mais profundas estabelecem relações mais complexas, de tal forma que na saída é possível obter, neste caso, classes de instrumentos.

Funções de ativação

- **ReLU**: A ativação linear retificada veio revolucionar o machine learning sendo das funções mais utilizadas. Esta função é linear para valores positivos e o que a torna não linear é o facto de valer sempre 0 para valores negativos:

  $$f(x) = max(x, 0) \tag{2.3}$$

  O facto de ter comportamento próximo do linear facilita a otimização da rede neuronal. Esta função de ativação é geralmente usada no fim de cada camada, exceto na saída.

- **Sigmoide**: Esta função é muito utilizada na saída das redes neuronais ou seja na camada utilizada para classificação. A sigmoide transforma a sua entrada em valores entre 0 e 1 o que

pode ser visto como uma probabilidade. No caso de multi etiquetas (ver a seguir), esta função permite a ativação de várias classes em simultâneo ou seja, se a rede for usada para classificar um excerto de música com piano e guitarra, a saída da rede pode ativar os dois neurónios da respetiva classe:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.4}$$

- **Soft-Max**: Semelhante à sigmoide, esta função também transforma um vetor de valores de saída em probabilidades. Neste caso a soma dessas probabilidades deve ser **1**, o que "obriga"a função a decidir apenas uma classe ou então dividir as probabilidades por todas:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}} \tag{2.5}$$

O modelo que serviu de exemplo tem agora as componentes básicas de uma rede neuronal:

- um conjunto de nós ou neurónios organizados por camadas
- um conjunto de pesos (w) representados pelas ligações entre cada camada
- um conjunto de "biases"(b) para cada nó.
- Uma função de ativação que transforma a saída de cada nó (diferentes camadas podem ter diferentes funções de ativação).

É com todos os valores de todos os nós bem calibrados, que uma rede neuronal melhora o seu desempenho. Para isso, deve haver um treino supervisionado. Isto significa que existem pares de entradas x e etiquetas y: (x,y). A uma entrada x podem corresponder várias etiquetas o que leva a dois conceitos também eles essenciais para a aprendizagem automática:

1. **Multi classe**: Quando **não** se pretende um classificador binário, ou seja, que decida se por exemplo um violino está ou não presente num excerto de áudio, o problema passa a ser multi classe. Neste caso existem duas classes ou mais.

2. **Multi etiqueta**: Problemas multi etiqueta só fazem sentido quando o problema é também ele multi classe isto porque um problema multi etiqueta permite a existência de várias etiquetas para a mesma entrada, x. Se um excerto contiver sons de bandolim e ukulele em simultâneo e se o objetivo for detetar os dois instrumentos na mesma classificação então o problema é multi etiqueta.

O objetivo de um sistema de aprendizagem automática é que este consiga classificar x como y. Como se trata de um sistema *feed-forward* ou seja, os dados fluem da entrada para a saída sem nunca formar um ciclo, existe um algoritmo de atribuição de "culpa"a cada peso, denominado *backpropagation*. Neste algoritmo é definido um critério geral de desempenho da rede (custo) onde, a cada iteração, se tenta diminuir o seu valor. No fim, o desempenho da rede é definido com base nas saídas previstas e no valor efetivo dessas saídas.

Inicialização do Modelo

Os pesos e *biases*, são responsáveis pelo comportamento de uma rede neuronal e tendem para o valor ideal à medida que estes algoritmos convergem. Para isso, deve-lhes ser atribuído um valor inicial que deve ser inicializado aleatoriamente. Apesar de ser uma prática comum, os pesos podem também ser inicializados a 0 ou podem tomar valores já conhecidos como quando uma rede já foi treinada e se pretende recuperar os seus parâmetros.

Foward propagation

Depois de inicializar todos os seus parâmetros, a rede faz propagar a sua entrada no sentido da saída, ou seja, calcula a saída (previsão) da rede para uma dada entrada, com base nos pesos e biases atribuídos. Geralmente este processamento é feito em "fornadas"mais conhecidas como ***batches*** que no fundo são pedaços mais pequenos da base de dados de treino. Depois de serem processados todos os batches completa-se uma **época**. É comum repetir este tipo de algoritmos por diversas épocas.

Função de custo

Função de custo ou *Loss function*, C, é a função que determina o quão longe estamos da saída pretendida. Para isso, esta função que no fundo representa o valor de um erro (custo), precisa de conhecer para cada entrada, a saída **estimada** pela rede e o valor **efetivo** dessa saída, (chamado **ground truth**). Quanto mais afastado estiver o valor estimado e o ground truth, maior o custo. Uma função de custo muito utilizada é a entropia cruzada (*cross entropy*),

$$Xent(p(y),y) = -\sum_i y_i \log(p(y)_i) \tag{2.6}$$

onde $p(y)$ é a saída estimada e y o ground truth.

Gradiente

O cálculo diferencial é essencial para a aprendizagem automática. Já que o objetivo é minimizar a função de custo, a sua derivada (em função de qualquer parâmetro da rede) ajuda a perceber em que sentido continuar e se estamos ou não perto do mínimo (idealmente absoluto) da função (é a isto que se chamou "culpa": derivada da função de custo em relação a um parâmetro). Como a função de custo tem geralmente milhares de parâmetros, essa derivada deve ser calculada em função de cada um deles (derivadas parciais).

Nesta etapa o objetivo é tentar encontrar o conjunto de pesos e biases que minimizam a função de custo (gradiente nulo). Este é um processo iterativo, que se repete para cada batch e que pode demorar de poucas até milhares de iterações dependendo do problema e do tipo de dados utilizados. A figura 2.9[2] mostra o valor do custo em função de um parâmetro w. Neste caso, só existe um peso e a solução ótima acontece quando o gradiente é nulo ("Mínimo"da função). Habitualmente as funções de custo são muito mais complexas de tal forma que é difícil visualizá-las com mais de dois parâmetros.

Para resolver um dos maiores desafios da aprendizagem automática, é muito comum utilizar o algoritmo **Gradient Descent**. Este é um algoritmo de otimização iterativo que tenta chegar ao mínimo da função de custo através de pequenos passos. O gradiente indica a direção na qual a função cresce e

[2] https://saugatbhattarai.com.np/wp-content/uploads/2018/06/gradient-descent-1.jpg

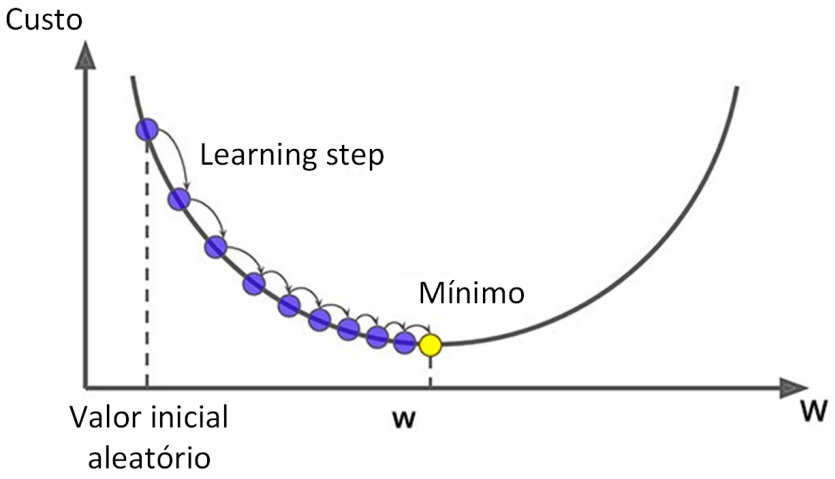

Figura 2.9 Valor do custo em função de um parâmetro, w

como o objetivo é minimizar, é necessário "evoluir"no sentido contrário. Para minimizar o custo, os parâmetros são atualizados na direção "descendente"do gradiente daí o nome, gradiente descendente.

Supondo que só é utilizado um peso w na rede, é necessário saber qual o respectivo valor da função de custo, C. Dando vários valores a w é possível obter uma função x como a da figura 2.9 e calculada a sua derivada em ordem a w, o declive em cada ponto passa também a ser conhecido.

O Gradient decent, começa num determinado ponto aleatório da função C e através de pequenos passos tenta chegar ao seu mínimo. Para isso, é necessário calcular as derivadas parciais da função de custo e o algoritmo responsável por este processo é o **backpropagation**.

Back propagation

Depois de derivar a função de custo, é possível retro-propagar a rede e ir calculando as derivadas parciais de cada parâmetro desde a saída até à entrada, processo que faz jus ao nome deste algoritmo.

Assumindo que estamos perante uma rede neuronal como a da figura 2.10,

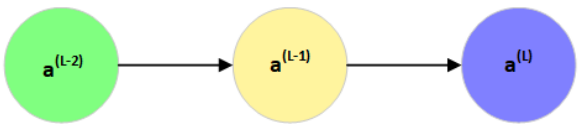

Figura 2.10 Rede Neuronal simples

a ativação do neurónio $a^{(L)}$ é dada por

$$a^{(L)} = \sigma(w^{(L)}.a^{(L-1)} + b^{(L)}) \tag{2.7}$$

sendo w, b e $a^{(L-1)}$ o valor do peso, bias e ativação do camada anterior, respetivamente. σ representa a função de ativação, sigmoide. A soma ponderada, entrada da sigmoide passa a denominar-se Z e simplificando:

$$Z^{(L)} = w^{(L)}.a^{(L-1)} + b^{(L)} \tag{2.8}$$

$$a^{(L)} = \sigma(Z^{(L)}) \tag{2.9}$$

O objetivo do gradient descent é saber o quão sensível é a função de custo, C, a pequenas variações de cada um dos parâmetros e para isso, são calculadas as derivadas parciais de C em ordem a w (2.10), b (2.11), e $a^{(L-1)}$ (2.12):

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial Z^{(L)}}{\partial w^{(L)}}.\frac{\partial a^{(L)}}{\partial Z^{(L)}}.\frac{\partial C^{(L)}}{\partial a^{(L)}} \tag{2.10}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial Z^{(L)}}{\partial b^{(L)}}.\frac{\partial a^{(L)}}{\partial Z^{(L)}}.\frac{\partial C^{(L)}}{\partial a^{(L)}} \tag{2.11}$$

$$\frac{\partial C}{\partial a^{(L-1)}} = \frac{\partial Z^{(L)}}{\partial a^{(L-1)}}.\frac{\partial a^{(L)}}{\partial Z^{(L)}}.\frac{\partial C^{(L)}}{\partial a^{(L)}} \tag{2.12}$$

O vector gradiente (2.14) é composto pelas equações 2.10 e 2.11. Apesar de não ser incluída, a derivada parcial de C em função de $a^{(L-1)}$ (2.12) ou por outras palavras, a sensibilidade de C a variações da ativação $a^{(L-1)}$, é essencial para o algoritmo de backpropagation uma vez que é esta função que "abre caminho"aos pesos e biases camada anterior (2.13). Este processo, cada vez mais complexo com o aumentar dos parâmetros, repete-se até à entrada da rede.

$$a^{(L-1)} = \sigma(w^{(L-1)}.a^{(L-2)} + b^{(L-1)}) \tag{2.13}$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w^{(1)}} \\ \frac{\partial C}{\partial b^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w^{(L)}} \\ \frac{\partial C}{\partial b^{(L)}} \end{bmatrix} \tag{2.14}$$

Assumindo agora uma rede com mais de um neurónio por camada (figura 2.11), o processo é semelhante sendo apenas necessário ter em conta que cada neurónio da saída, Z_j tem mais do que um a influenciá-lo (comparar com 2.8, caso com apenas um neurónio por camada):

$$Z_j^{(L)} = w_{j0}^{(L)}.a_0^{(L-1)} + w_{j1}^{(L)}.a_1^{(L-1)} + w_{j2}^{(L)}.a_2^{(L-1)} + b_j^{(L)} \tag{2.15}$$

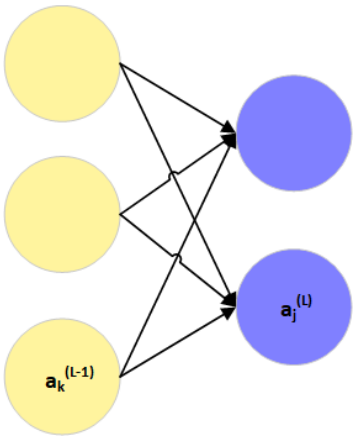

Figura 2.11 Rede Neuronal mais complexa

Tal como anteriormente, é importante ter apenas um valor para o custo, algo que sirva de referência para a performance da rede. Neste caso, com mais do que um neurónio por camada, a função de custo global, C, é dada pelo somatório das funções de custo obtidas para cada neurónio:

$$C = \sum_{j=0}^{n_L-1} C_j \tag{2.16}$$

Por conseguinte, a derivada parcial da função de custo em função da ativação da camada anterior, $L-1$ (outrora 2.12) é agora ligeiramente diferente já que conta com mais do que uma activação na saída, neste caso duas. Para resolver este problema, é feito um somatório da derivada da função de custo de cada neurónio, em função da ativação da camada $L-1$:

$$\frac{\partial C}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial Z_j^{(L)}}{\partial a_k^{(L-1)}} \cdot \frac{\partial a_j^{(L)}}{\partial Z_j^{(L)}} \cdot \frac{\partial C^{(L)}}{\partial a_j^{(L)}} \tag{2.17}$$

Com estes procedimentos, o algoritmo de backpropagation é capaz de calcular todas as derivadas parciais que em muitos casos, atingem milhões de parâmetros. Este cálculo permite então conhecer, para cada parâmetro, qual o seu próximo valor isto é, se o valor deve diminuir ou aumentar muito ou pouco.

Atualização dos pesos

O próximo valor de cada parâmetro,w_{new} (2.19), é obtido através do passo, **Learning step** (2.18), representado na figura 2.9. Este passo é obtido com base no declive, calculado através do backpropagation, e através do ritmo de aprendizagem, **Learning rate**.

$$Learning\ step = declive \times Learning\ rate \tag{2.18}$$

$$W_{new} = W_{old} - Learning\ step \tag{2.19}$$

O facto de o Learning step estar relacionado com o declive da função é útil porque nos diz o quão perto estamos do mínimo isto é, para grandes declives os passos são maiores, mas vão diminuindo com a proximidade ao mínimo da função de custo ou ao zero da sua derivada (2.20),

$$\nabla C = \mathbf{0} \tag{2.20}$$

onde C é o custo e ∇ o vetor gradiente.

Se o passo for negativo, W_{new} aumenta. Por outro lado, se o passo for positivo, W_{new} diminui o que é coerente com a procura do mínimo da função de custo. O Learning rate serve para garantir que a actualização dos pesos é lenta e suave.

A atualização dos pesos é feita com uma determinada frequência que é determinada pelo número de batches, M e pelo tamanho de cada batch N. Existem 3 opções:

1. **stochastic gradient descend**: Neste caso, $N = 1$ e portanto os parâmetros vão ser atualizados a cada exemplar da base de dados de treino. Esta abordagem pode tornar o processo demasiado lento.

2. **batch gradient descend**: Também é possível utilizar um único e grande batch que inclui a base de dados de treino toda. Neste caso, os pesos serão atualizados apenas uma vez por época e portanto é preciso treinar a rede durante várias épocas até esta começar a convergir.

3. **mini-batch gradient descend**: Tenta encontrar um meio termo entre as duas técnicas anteriores. Geralmente $10 < N < 1000$ sendo que com esta abordagem (e também com a anterior) é feita a média dos valores da função de custo de todos os exemplares de treino contidos num determinado batch.

A atualização dos parâmetros é feita através de otimizadores. Existem vários como o Adagrad, Adam, RMSprop [30].

Todo o processo abordado nesta secção está resumido na figura 2.12.

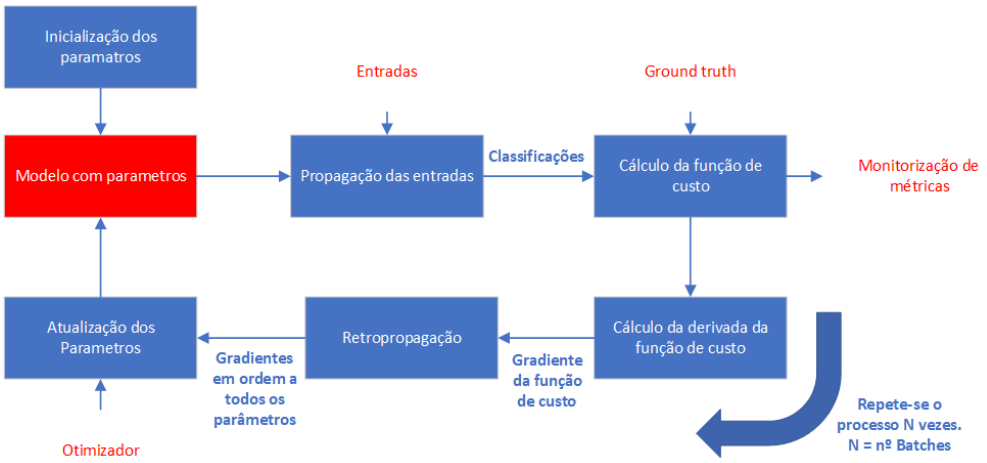

Figura 2.12 Processo de treino de um algoritmo de Aprendizagem Automática

Deep Learning é um termo que se tem tornado cada vez mais popular, este consiste em transformar estas redes com poucas camadas escondidas em redes profundas, quanto mais profunda uma rede for

mais capaz é de reconhecer padrões complexos mas, a exigência computacional também aumenta. Por outro lado, quanto mais profunda for a rede mais dados são necessários uma vez que também é necessário "ensinar"mais parâmetros. Com o evoluir da tecnologia surgiu naturalmente o Deep Learning (DL) tanto pelo crescimento no número de dados como pela evolução a nível computacional.

2.3.2 Redes Neuronais Convolucionais

Uma das classes de redes neuronais mais utilizadas chama-se redes neuronais Convolucionais (CNNs). Ao contrário das redes neuronais artificiais (ANNs) que têm uma cama de neurónios como entrada, uma CNN é uma rede que pode receber como entrada uma imagem e atribuir a eventos que nela possam ocorrer pesos e biases capazes de os distinguir. As CNNs filtram diversas vezes a imagem de entrada e aprendem como chegar aos filtros que melhor distinguem um determinado evento.

Na realidade, uma imagem pode ser representada por N matrizes, sendo N o número de canais que esta contempla. No caso de uma imagem Red Green Blue (RGB) existem 3 canais e supondo que a imagem tem uma dimensão de 4 x 4 é possível então extrair 3 matrizes de dimensões 4 x 4 cada, tal como se pode verificar na figura 2.13.

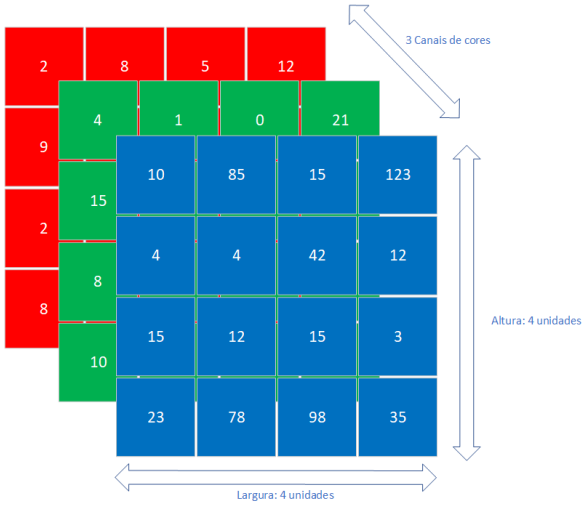

Figura 2.13 Imagem RGB

Claro que se poderia transformar estes $4 \times 4 \times 3 = 48$ valores numa única camada e utilizar uma rede neuronal para a classificação, mas imagens reais têm dimensões muito maiores e o peso computacional tornar-se-ia enorme. As CNNs aplicam filtros à imagem de entrada por convolução 2D (2.21).

Kernel

Uma camada convolucional é composta por um kernel ou filtro, matriz que é responsável por "deslizar"ao longo da imagem efetuando uma multiplicação a cada avanço ou **stride**. Dada uma entrada 2D, I, e um kernel K a convolução 2D é definida pela equação

$$C(j,k) = \sum_p \sum_q I(p,q)K(j-p,k-q) \tag{2.21}$$

onde (j,k) são as coordenadas da entrada e (p,q) as coordenadas do kernel. O kernel desliza pela imagem e os pesos que o compõem são aplicados a toda a área de entrada. Esta partilha de parâmetros reduz significativamente o peso computacional e consequentemente o tempo de treino destas redes.

Figura 2.14 Convolução do kernel com uma matriz. Stride=1, valid Padding

Caso a imagem tenha mais do que um canal, a cada avanço são somados os resultados da multiplicação de cada kernel com o respectivo pedaço da imagem. Também é possível, em redes muito complexas escolher o número de filtros a utilizar ou seja, no caso da figura 2.14 se em vez de ser usado 1 filtro fossem usados 64, a saída da rede seria $2 \times 2 \times 64$.

Padding

Por vezes, para que o kernel possa fazer uma multiplicação corretamente é necessário acrescentar pixeis à imagem. A esse processo chama-se Padding e existem duas opções, *valid* e *same*. Valid Padding mantém a imagem original e a dimensão do resultado reduz sempre. Same Padding acrescenta zeros na moldura da imagem o que faz com que o resultado, tal como o nome indica, tenha as mesmas dimensões que a imagem.

Pooling Layer

Similar às camadas convolucionais, as camadas de pooling também contêm um kernel e o objetivo é reduzir a dimensão da entrada. Esta camada não tem parâmetros ou seja, não é treinada. O pooling é útil para sumarizar algumas *features* detetadas na entrada e para além disso é também capaz de gerar uma representação invariante a pequenos deslocamentos de uma determinada feature detetada. Isto é importante quando queremos verificar se existe ou não um determinado padrão em vez de saber a sua localização.

Existem dois tipos de pooling (figura 2.15):

- Max Pooling - retorna o máximo valor da porção de imagem que está sob o kernel.
- Average Pooling - retorna a média dos valores da porção de imagem que está sob o kernel.

A operação de Max Pooling é mais usada uma vez que tem desempenhos melhores que a alternativa.

Flatten Layer

Depois de reduzir várias vezes as dimensões é comum utilizar esta camada para converter a matriz ou matrizes resultante(s) num único vetor para depois ser classificado (figura 2.16).

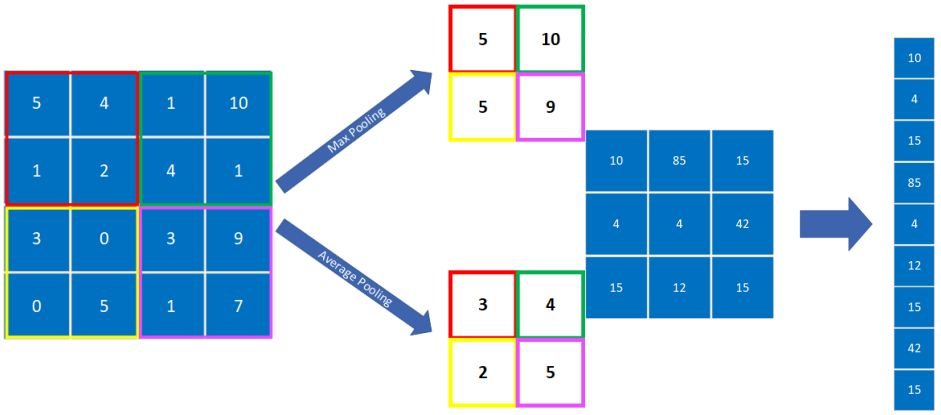

Figura 2.15 Operações de Max Pooling e Average Pooling

Figura 2.16 Camada flatten

Fully connected layer

Uma camada Fully connected ou ANN convencional pode ser vista como a entrada de uma rede neuronal que, a partir desta camada, se comporta tal como explicado anteriormente. A figura 2.17 [3] mostra uma rede com as diferentes camadas.

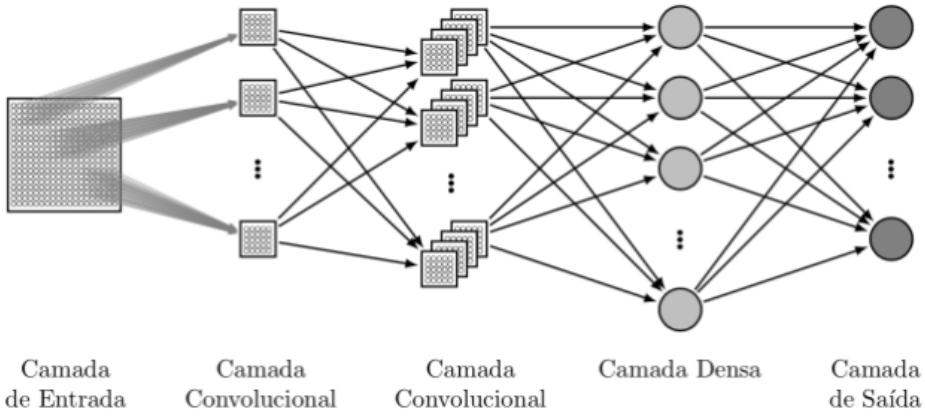

Figura 2.17 Rede Neuronal Convolucional

2.4 Identificação de instrumentos musicais

A identificação de instrumentos é um problema ainda não completamente resolvido no Music Information retreival. Ao longo dos anos um dos principais objetivos foi identificar as melhores *features* para a classificação de instrumentos musicais isto é, a melhor forma de representar o áudio de maneira a identificar corretamente um instrumento. Existem abordagens que usam *features* baseadas no timbre, *feautures* espectrais, temporais e muitas outras formas de olhar para uma gravação de um ponto de vista musical que ajudam a melhor reconhecer instrumentos [29], [4].

[3] http://rafaelsakurai.github.io/images/posts/2017-12-20-cnn-mapreduce.png

Uma vez que é uma tarefa complexa, é importante estudar e perceber como se deve comportar um sistema de identificação de instrumentos ideal. Martin (1999) [25] sugere alguns procedimentos básicos para tarefas de reconhecimento de fontes sonoras. Apesar do trabalho não ser direcionado para o reconhecimento de instrumentos musicais, enquadra-se perfeitamente neste problema. Um modelo ideal deve então cumprir os seguintes requisitos:

1. **apresentar Generalização**: O reconhecimento de um instrumento não deve depender de factores externos como quem é que o toca ou o meio ambiente. Todos estes factores, por mais que influenciem o som ouvido, nunca podem influenciar o reconhecimento do instrumento que está a ser tocado.

2. **Pertencer ao mundo real**: Os dados a utilizar devem pertencer ao mundo real o que implica algum cuidado na escolha das bases de dados. Estas não devem conter sons demasiado artificiais. O objetivo é sempre que o classificador se comporte da melhor forma em casos reais.

3. **Ser escalável**: É importante que o sistema esteja preparado para receber novas classes. A base de dados utilizada pode conter apenas 20 instrumentos e se o sistema funciona para este conjunto, também deve funcionar caso se pretenda aumentar o número de instrumentos a reconhecer. Não só o sistema, as *features* extraídas também devem ser flexíveis a todo o tipo de instrumentos.

4. **Ser robusto à degradação**: Com o aumento do barulho ambiente, reverberação, eco ou o número de fontes de som ouvidas, o ser humano perde gradualmente a capacidade de reconhecer um determinado instrumento. Da mesma forma, os classificadores deixam de funcionar corretamente quando um certo nível de degradação é atingido. Num cenário real, há informação (relativa ao instrumento que se pretende identificar) que se perde. Para isso, o sistema deve ser capaz de resolver o que é chamado o *"missing feature problem"*. Tem de ser capaz de reconhecer o instrumento ainda que não tenha acesso a toda a sua informação.

5. **Ter uma estratégia de aprendizagem flexível**: Muitas vezes o ser humano aprende não só com dados etiquetados (saber que o som que se está a ouvir pertence a um determinado instrumento) mas também com dados desconhecidos. A performance de um sistema melhora sempre quando este é capaz de aprender com os dados que tem etiquetados mas também, posteriormente aprender com os dados cuja presença ou ausência de um instrumento não são conhecidas.

6. **Funcionar em tempo real**: Com o objetivo de simular a forma como a perceção humana funciona, o modelo deve ser capaz de seguir a evolução temporal do som. A extração de features e a inferência de uma rede neuronal são processos que podem demorar e impedir que a identificação ocorra em tempo real. É comum então apresentar resultados a uma taxa inferior para haver tempo para todos os cálculos necessários.

Arquitetura

Independentemente das escolhas que se possam fazer durante a implementação do algoritmo, este tem sempre a mesma arquitetura composta por 4 etapas:

1. **Pré-processamento**: Aplicadas ainda ao áudio, existem estratégias de pre-processamento como a redução do ruído que remove as componentes ruidosas do sinal ou a remoção de momentos de silêncio.

2. **Extração de features**: Tal como já referido, é importante representar o áudio em formatos que sejam mais legíveis para um classificador.

3. **Classificação**: Etapa crucial onde decorre o algoritmo que aprende a identificar instrumentos e por fim faz uma previsão relativamente a um conjunto de teste.

4. **Pós-processamento**: Com base nas previsões da etapa anterior é comum transformar estes dados por forma a obter o melhor resultado para a métrica pretendida. Para este tipo de cálculos é conveniente comparar os valores previstos com os valores verdadeiros (conhecidos previamente).

2.4.1 Identificação de instrumentos em música monofónica

A identificação de instrumentos em música monofónica é feita com base em duas categorias:

- reconhecimento com base em notas únicas de um só instrumento.

- reconhecimento com base em frases de um só instrumento.

Naturalmente, a investigação na identificação de instrumentos começou por esta área, mais simples e com menos problemas para resolver quando comparada com a identificação de instrumentos em música polifónica.

Um dos primeiros estudos realizados nesta área (2000) utilizou classificadores Gaussianos e kNN (k-nearest neighbors) num conjunto de **43 features** temporais e espetrais com base em instrumentos orquestrais [6]. Este trabalho marca o início do reconhecimento de instrumentos musicais. No seguimento do trabalho anterior, um ano depois (2001) a mesma equipa utilizou MFCCs em conjunto com outras features espetrais e utilizou modelos de misturas Gaussianas (GMM) para as classificar. Os resultados, mostravam que as MFCC eram as features que melhores resultados apresentavam e ainda hoje são muito utilizadas.

Essid et al. [8] (2006) utilizou um conjunto de features encontradas com alguns algoritmos de seleção de features e Support Vector Machines (SVM) para classificar 10 instrumentos clássicos tornando-se na altura a abordagem com melhores resultados. Com o objetivo de explorar novas features , Yu et al. [31] (2009) utilizou como entrada de classificadores KNN, espetogramas de clips de áudio como imagens. Esta abordagem é semelhante aquela que será considerada neste projeto.

Diment et al. [5] (2013) utiliza a feature *Modified Group Delay Features* (MODGDF) [15] em conjunto com MFCC em 22 instrumentos da base de dados RWC [11] (2002). Por fim, Han et al. [14] (2016) propôs uma abordagem para aprender *features* provenientes do espetograma de Mel. Este trabalho foi realizado com base em notas únicas de 24 instrumentos novamente da base de dados RWC [11].

Para se aproximar de situações reais, os investigadores começaram a trabalhar na identificação ainda em áudio monofónico mas em frases de instrumentos. Um dos primeiros estudos foi realizado

por Krishna e Sreenivas [22] que utilizaram Line Spectral Features (LSF) como entrada de um modelo GMM implementado com 14 instrumentos diferentes.

Essid et al. [7] utilizou como modelo, novamente o GMM com MFCC em conjunto com Principal Component Analisys (PCA) e testado com base em 5 instrumentos. PCA [1] é uma técnica utilizada para reduzir a dimensão dos dados. Através do PCA é possível transformar dados com muitas dimensões em dados com menos, mas preservar ainda assim a informação relevante. Com esta técnica é normal que se despenda de alguma precisão no modelo implementado, mas ganha-se simplicidade, portanto existe um "trade-off"entre o peso computacional e a precisão que os novos dados proporcionam uma vez que dados com dimensões menores são mais fáceis de analisar e visualizar.

2.4.2 Identificação de instrumentos em música polifónica

Ao longo dos últimos anos, o foco da investigação nesta área recai sobre áudio polifónico uma vez que é a abordagem que mais se aproxima do mundo real. O Deep Learning e o seu crescimento exponencial (figura 2.18[4]), vieram revolucionar a identificação de instrumentos musicais nomeadamente através das Redes Neuronais Convolucionais (figura 2.19[5]).

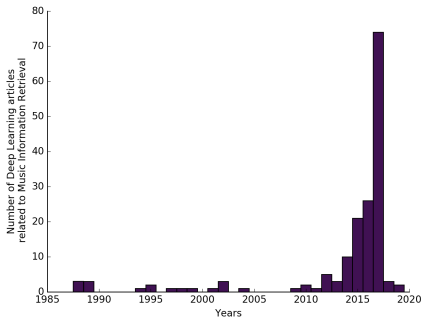

Figura 2.18 Número de artigos publicados no âmbito do MIR, ao longo dos anos

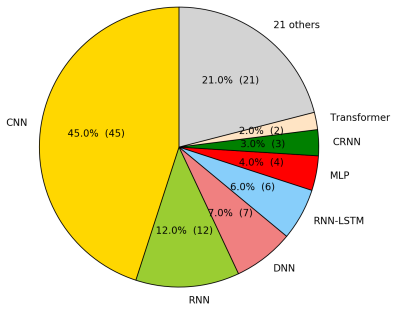

Figura 2.19 Arquiteturas DL mais utilizadas no âmbito do MIR

Em 2015 começaram a surgir alguns trabalhos com CNN [23], [24] e em 2016, Han et al. [13] realizou um dos primeiros trabalhos na identificação de instrumentos musicais com recurso a CNN. Este trabalho foi baseado na base de dados IRMAS [19] e uma vez mais utiliza o espetograma Mel como entrada da CNN implementada.

Esta estratégia começou a ser cada vez mais comum uma vez que a rede ganha a capacidade de analisar tanto a componente temporal como a espetral. Esta "imagem"pode conter mais do que um instrumento que produz timbres únicos identificáveis no espetograma. Através dos filtros convolucionais é possível distinguir os diferentes instrumentos.

[4] https://github.com/ybayle/awesome-deep-learning-music/blob/master/fig/articles_per_year.png
[5] https://github.com/ybayle/awesome-deep-learning-music/blob/master/fig/pie_chart_architecture.png

Em 2017 surge uma nova base de dados publicada pela Google, o AudioSet [10]. Esta base de dados surge com a intenção de ajudar a investigação na deteção de eventos de som (SED). A Google lançou um modelo pre-treinado chamado VGGish [16] que pode ser usado de duas formas:

- **Extrator de features**: O modelo converte a entrada num vetor de features de dimensão 128 que pode servir de entrada para outro modelo.
- **Parte de um modelo maior**: Acrescentando camadas no final da rede até à camada de classificação, obtém-se um modelo que recebe as features de áudio e prevê a saída para as classes pretendidas. Esta técnica é também conhecida por **transfer learning**.

Isto porque o modelo só é disponibilizado até uma camada ***bottleneck***.

Bottleneck ou gargalo de garrafa são palavras que ajudam a entender o que é que significa este conceito. Esta, é uma camada com menos neurónios quando comparada com as anteriores. É uma camada muito utilizada para obter uma representação da entrada com dimensões muito mais reduzidas ou seja, para comprimir e codificar essa informação. O vetor que resulta deste tipo de camadas chama-se **embedding** e é muito usado para visualização dos dados já que a sua dimensão é muito reduzida quando comparada com a da entrada (figura 2.20).

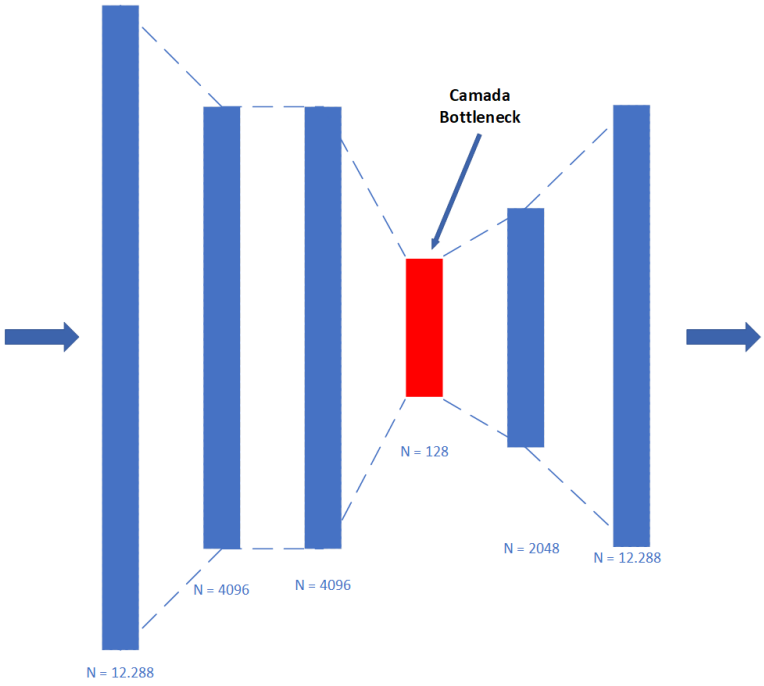

Figura 2.20 Rede Neuronal com camada Bottleneck

O nome do modelo VGGish vem do modelo VGG16 [2] utilizado para reconhecimento de imagens. A arquitetura da rede VGG16 (figura 2.21) é modificada para obter a rede VGGish (figura 3.2): esta rede utiliza sempre convoluções com filtros de 3×3. A dimensão da entrada é alterada para 96×64 (espetograma de Mel) e são usados **4** grupos de camadas convolução-maxpool (que têm menos uma convolução cada) em vez de 5. Na saída, em vez de 1000 unidades, é usada uma camada Fully Connected com 128. Por fim, este vetor é pós processado com uma transformação do tipo Principal Component Analisys (PCA).

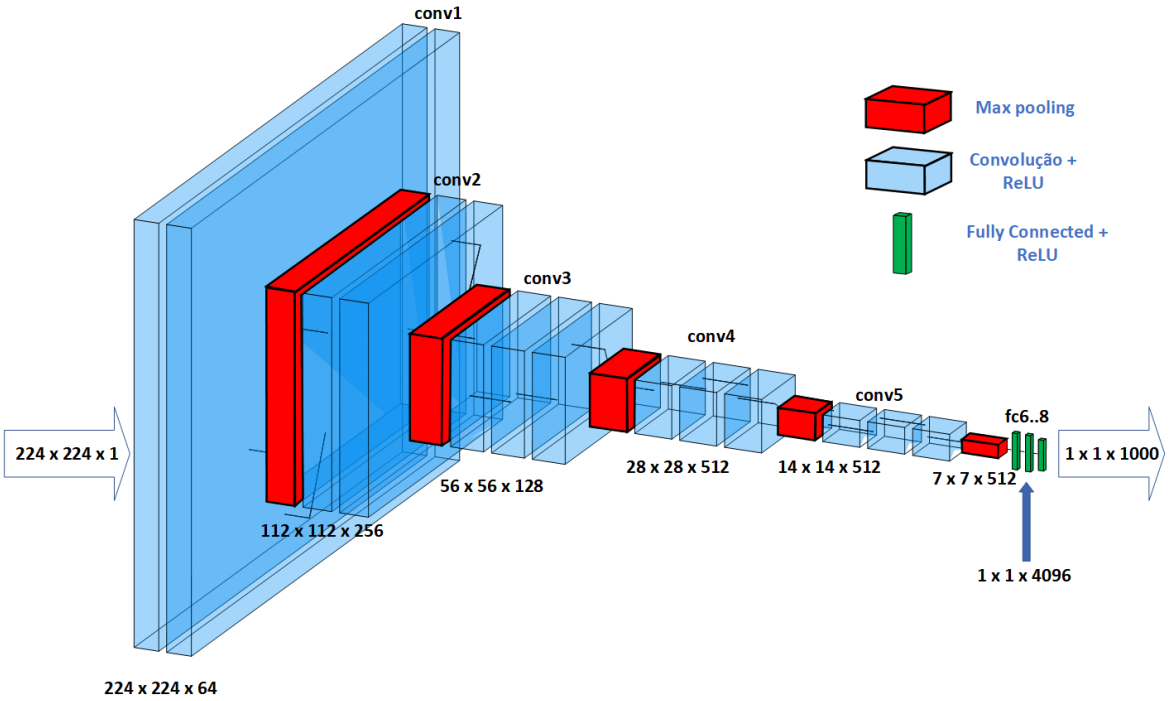

Figura 2.21 VGG16 [2]

Em 2018, surge uma base de dados com 20.000 clips de áudio, o OpenMic-2018 (OM18) [17]. Cada clip tem 10 segundos e é uma base de dados com 20 classes de instrumentos musicais. Anotar um clip segundo a segundo é muito dispendioso e portanto esta é uma base de dados com etiquetas fracas, *Weakly Labelled Dataset* (WLD). Isto significa que um instrumento é anotado como positivo se estiver presente em pelo menos uma parte do clip, caso contrário é anotado como negativo (se não estiver presente em nenhum momento do clip). Cada clip está anotado apenas para 3-4 classes (em 20) sendo as restantes anotadas como desconhecidas. Esta base de dados, segue a mesma arquitetura do AudioSet, tanto na extração de features (espetograma Mel) como na rede a utilizar (VGGish). Um dos trabalhos realizados em torno desta base de dados tem por base um classificador Random Forest (RF) e foi desonvolvido por quem criou o OpenMic-2018. Muito recentemente, foram publicados trabalhos que, tal como projetado para esta dissertação, aplicam modelos de atenção a esta base de dados [12], [28].

Random forest é um algoritmo de aprendizagem automática utilizado (não só) para classificação e que será utilizado ainda que muito pouco, na execução deste projeto. Este algoritmo utiliza várias árvores de decisão e decide a sua saída com base nos resultados de todas elas.

2.5 Dados com etiquetas fracas

Em muitos casos, os dados a que temos acesso não têm "etiquetas fortes"ou seja, a grande parte das bases de dados usadas pertencem à categoria das Weakly labeled Datasets (WLD). Uma base de dados do tipo WLD contém exemplares, como por exemplo um ficheiro de áudio de 10 segundos cuja etiqueta é violino, mas que nada nos garante que este instrumento está presente em todos os 10

segundos. Com este tipo de bases de dados surgiu o **Multiple Instance Learning** (MIL). A ideia do MIL é que um exemplar representa um saco de instâncias. Ao receber previsões das classes de cada uma das instâncias, é necessário juntar todas elas numa única previsão de todo o saco.

Uma das abordagens mais simples para tratar este tipo de problemas consiste no modelo *Bag of Words* (BOW) que assume que todas as instâncias herdam a etiqueta do *bag* respetivo. Tudo isto é necessário para a fase de treino mas na fase de inferência a decisão é tomada agregando as previsões de cada uma das instâncias sendo técnicas como o max pooling (considera a previsão mais alta) ou mean pooling (média de todas as instâncias) das mais comuns.

Em contraste às WLD existem as Strongly Labaled Datasets (SLD) que descriminam temporalmente os ficheiros de áudio indicando a presença ou ausência de, neste caso, instrumentos que possam coexistir em cada segmento temporal.

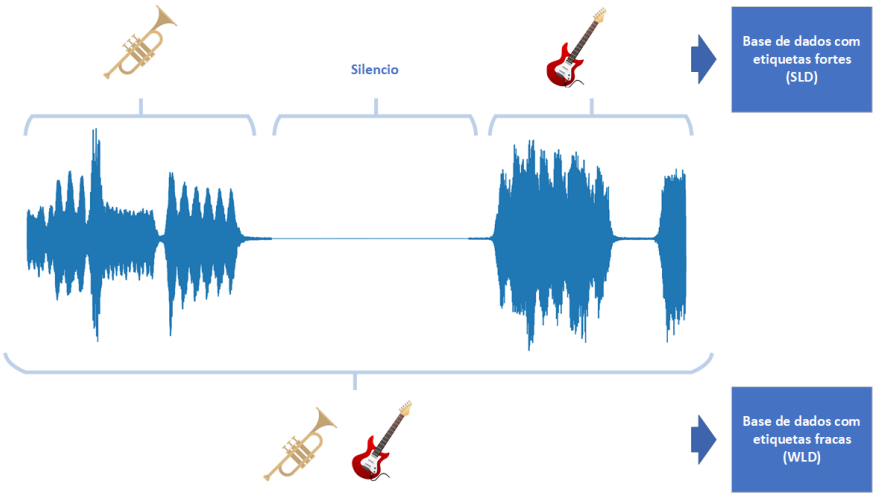

Figura 2.22 Imagem que identifica o problema do MIL

Modelo de Atenção

De uma forma semelhante ao BOW, os modelos de atenção criam um classificador (*instance level classifier*) $f(x)$ para cada instância x mas por outro lado, as instâncias não herdam as etiquetas do *bag*. No fundo a aprendizagem supervisionada ao nível da instância deixa de existir. É então necessário agregar as previsões de cada instância resultando num previsão $F(B)$ de todo o *bag*, B, dada por:

$$F(B)_k = \sum_{x \in B} p(x)_k f(x)_k \tag{2.22}$$

onde $p(x)_k$ corresponde ao peso de $f(x)_k$ para cada classe, k. $p(x)_k$ é então chamado de **função de atenção** e deve satisfazer a seguinte condição:

$$\sum_{x \in B} p(x)_k = 1 \tag{2.23}$$

para que a previsão do *bag* seja vista como uma soma ponderada das previsões das instâncias e para isso a função de atenção pode ser modelada da seguinte forma:

$$p(x)_k = v(x)_k / \sum_{x \in B} v(x)_k \tag{2.24}$$

$v(x)$ deverá ser uma função não negativa que garanta que $p(x)$ é uma probabilidade como por exemplo a **softmax**.

A função de atenção $p(x)_k$ controla o quanto é que uma determinada previsão $f(x)_k$ deve importar.

A figura 2.23 mostra o modelo de atenção descrito.

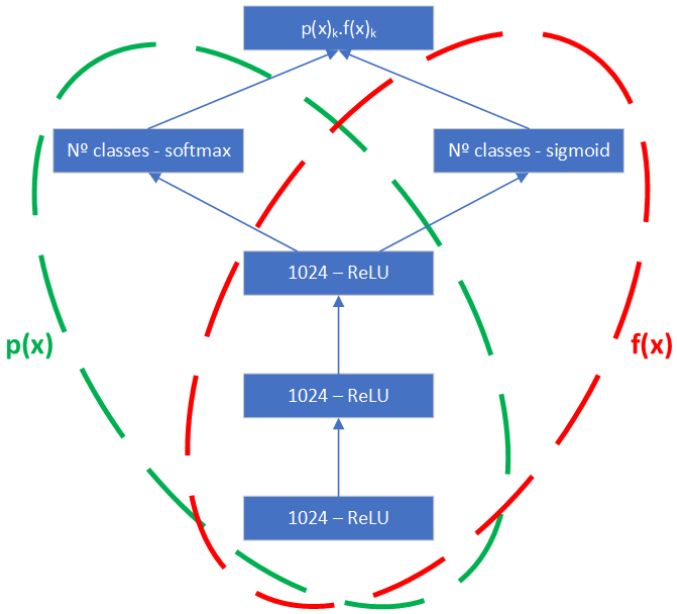

Figura 2.23 Implementação de um modelo de atenção [21]

Tanto o OM18 como o AudioSet são WLD. Investigações recentes, muito em volta do AudioSet, tentaram encontrar alternativas com melhores resultados. O trabalho de Kong et al. [21] (2019) utilizou modelos de atenção para classificar o AudioSet e atingiu resultados nunca antes obtidos noutros trabalhos com esta base de dados.

Capítulo 3

Suporte ao desenvolvimento

Para que o desenvolvimento do trabalho decorra da melhor forma, é necessário conhecer algumas técnicas tanto na preparação dos dados como durante a implementação de uma rede neuronal. Para além disso é essencial conhecer bem as bases de dados e perceber quais as suas limitações. Para poder haver referências que nos indiquem a performance do algoritmo existem também algumas métricas que, dependendo do tipo de problemas, podem ser usadas.

3.1 Bases de Dados

Para que todas as técnicas referenciadas ao longo da dissertação se possam concretizar, é essencial o recurso a bases de dados. Esta secção serve para introduzir as bases de dados utilizadas neste projeto.

3.1.1 Audioset

O Audioset [10] é uma Weakly Labeled Dataset (WLD) (secção 2.5) que conta com mais de 2 milhões de clips de áudio cada um com 10 segundos dos mais variados eventos. Esta base de dados tem 527 classes (multi classe) de eventos acústicos como alarmes, sons emitidos por animais ou instrumentos musicais e foi construída a partir de vídeos do YouTube com o objetivo de contribuir para a identificação de eventos acústicos (SED). É uma base de dados multi-etiqueta e o que é publicado, em vez dos ficheiros de áudio ou conjuntamente com os URLs do YouTube, são vetores de "features"de dimensão 128 para cada segundo de áudio. Para além disso também estão disponíveis o nome de cada segmento (identificador através do qual se chega ao vídeo no YouTube), o tempo inicial e final do clip (os vídeos têm sempre mais de 10 segundos pelo que é preciso selecionar um segmento) e por fim, uma (ou mais) etiqueta à qual correspondem os 10 segundos (figura 3.1).

A equipa que desenvolveu o Audioset, desenvolveu um trabalho onde testou varias redes de classificação de imagem na classificação de áudio [16]. Neste trabalho, a rede de classificação de imagens **VGG** [2] (figura 2.21) foi modificada e depois de algumas adaptações (ver sec. 2.4.2), foi então criada a rede **VGGish** (figura 3.2) que tem como objetivo classificar áudio. Depois de treinar milhões de exemplares verificou-se numa camada **bottleneck** (ver sec. 2.4.2) a meio da CNN, um conjunto de valores que representa, de forma compacta, a informação relevante para a classificação do segmento de áudio. É então por essa razão que o AudioSet disponibiliza apenas os 128 valores desta

saída "bottleneck". Estes representam de uma forma muito eficaz o áudio não sendo necessário (nem permitido neste caso) disponibilizar os ficheiros de áudio.

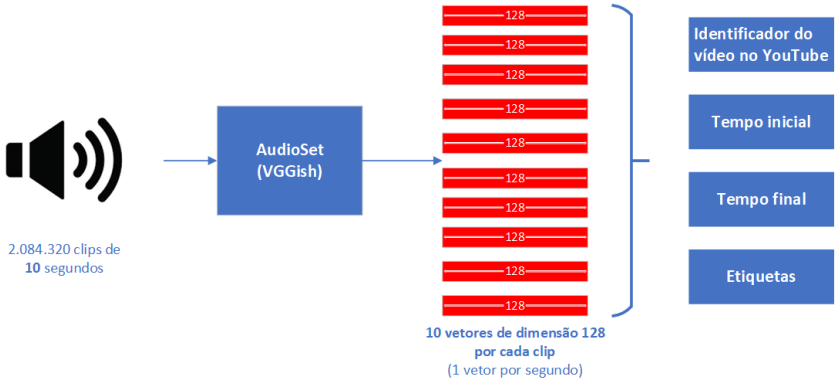

Figura 3.1 Estrutura do AudioSet

A arquitetura disponibilizada por parte da google, representada na figura 3.2 só está implementada até à camada bottleneck pelo que pode ser usada apenas para extrair os 128 valores ou então para fazer parte de uma CNN maior.

Para a entrada desta rede são considerados 64 filtros mel por 96 *frames*, 10ms cada, de áudio (**96 x 64 x 1**). Todos os pesos e biases são inicializados a 0, todas as ativações são ReLU, todas as convoluções são 3x3 com stride=1 (ver sec. 2.3.2) e todos os maxpools são 2x2 com stride=2 (razão pela qual a dimensão é deduzida sempre para metade). Resumindo, a rede recebe matrizes de **96** $\times$ **64** e resulta numa saída (bottleneck) de 128 valores.

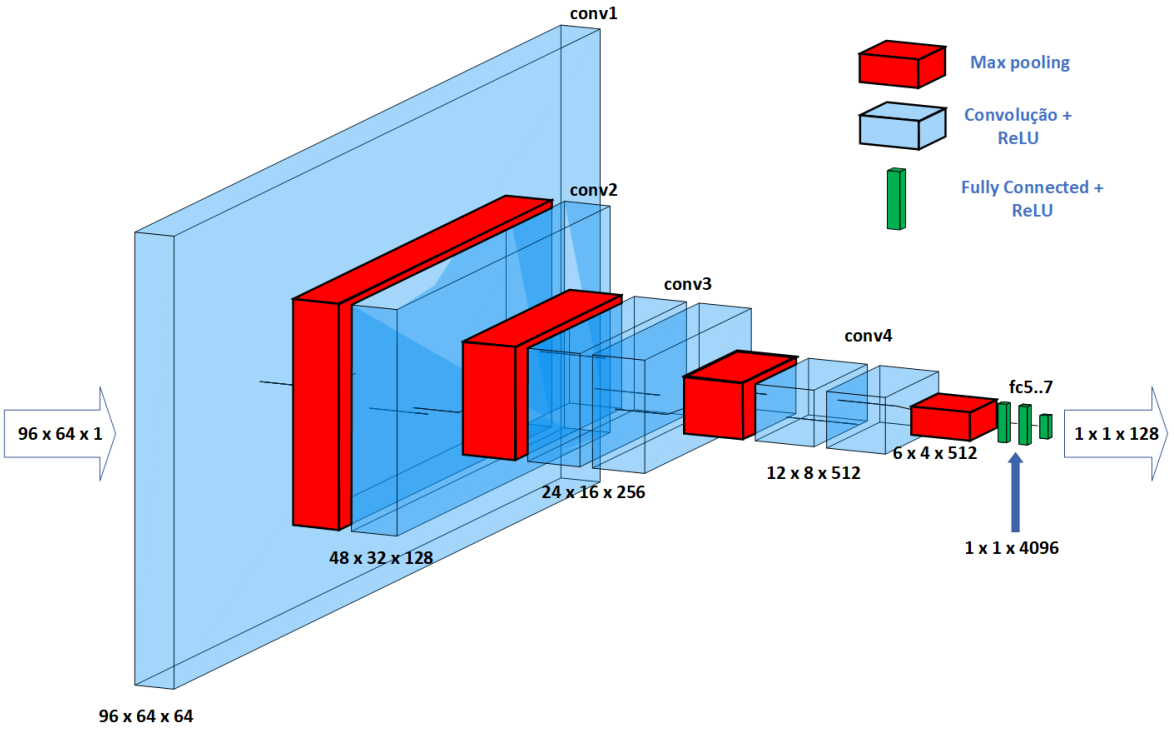

Figura 3.2 Rede VGGish implementada até ao bottleneck layer

Entrada da rede VGGish

Para transformar qualquer ficheiro de áudio nas matrizes de dimensão **96 x 64** é necessário um pré-processamento de todos os clips. O processo utilizado pelo AudioSet tem por base o procedimento referido na secção 2.2 através do qual se obtém o espetrograma de Mel.

A figura 3.3 mostra o módulo da DFT de um segmento do sinal amostrado a 44.1kHz. Tal como se pode verificar, as frequências mais elevadas são da ordem dos 20kHz, mais concretamente 22050 Hertz, metade da frequência de amostragem, ou frequência de Nyquist. É perfeitamente visível que grande parte do conteúdo espectral está nas baixas frequências pelo que não é necessário um sinal com uma frequência de amostragem tão elevada. A equipa do AudioSet optou então por converter todos os sinais para uma frequência de amostragem de 16kHz, cuja frequência de Nyquist é de 8kHz, valor a partir do qual o conteúdo espetral começa a ser irrelevante.

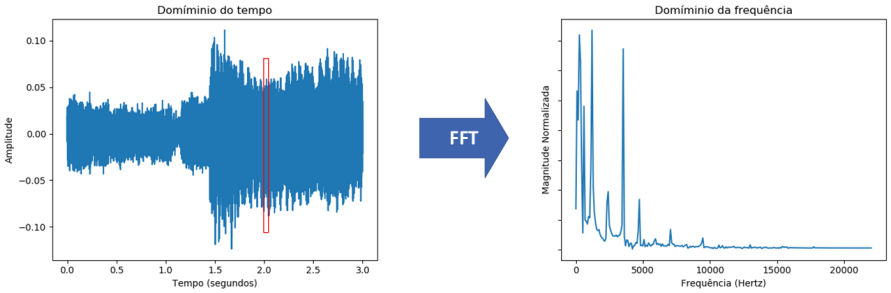

Figura 3.3 FFT do sinal amostrado a 44.1kHz

Depois de mudar a frequência de amostragem do sinal para 16kHz, este é multiplicado pela janela de Hann de duração **25ms (400 amostras) e um avanço (hop) de 10ms (160 amostras)**.

Terminado o processo anterior, a cada segmento, que no caso dos clips de 10s são 998,

$$\left\lfloor \frac{160.000 - 400}{160} \right\rfloor + 1 = 998$$

é aplicada a DFT e o seu módulo quadrado é aplicado ao banco de 64 filtros mel que estão contidos **entre os 125 e os 7500Hz**. É aplicado o logaritmo ao espetograma Mel somado com um $offset$ ($log(mel + 0.01)$) para evitar o logaritmo de zero o que causaria instabilidade. O resultado é o logaritmo do espetograma em escala Mel de dimensão 998 x 64 (figura 3.4 [1]).

Para finalizar, a equipa desenvolvedora desta base de dados optou por dividir estes 998 frames em conjuntos de 96 **sem sobreposição**, resultando assim em **10** matrizes com as mesmas dimensões da entrada da rede VGGIsh, **96 x 64**. De notar que $998 - (96 \times 10) = 38$ frames, são descartados tal como se pode verificar na figura 3.4 que resume todo o processo aqui explicado. O número de frames escolhido é 96 (10ms cada) uma vez que é possível decimar este valor por 2, 5 vezes o que não aconteceria caso o número de frames fosse, como seria esperado, 100.

É interessante referir que a entrada da rede VGGish, matrizes de 96 x 64 (figura 3.5), é interpretada como uma imagem (de um canal) ou seja, a rede tentará reconhecer um espetograma que representa som da mesma forma que tenta distinguir imagens de números manuscritos por exemplo.

[1] https://www.youtube.com/embed/Fu85jL7q6yE?start=70&end=80

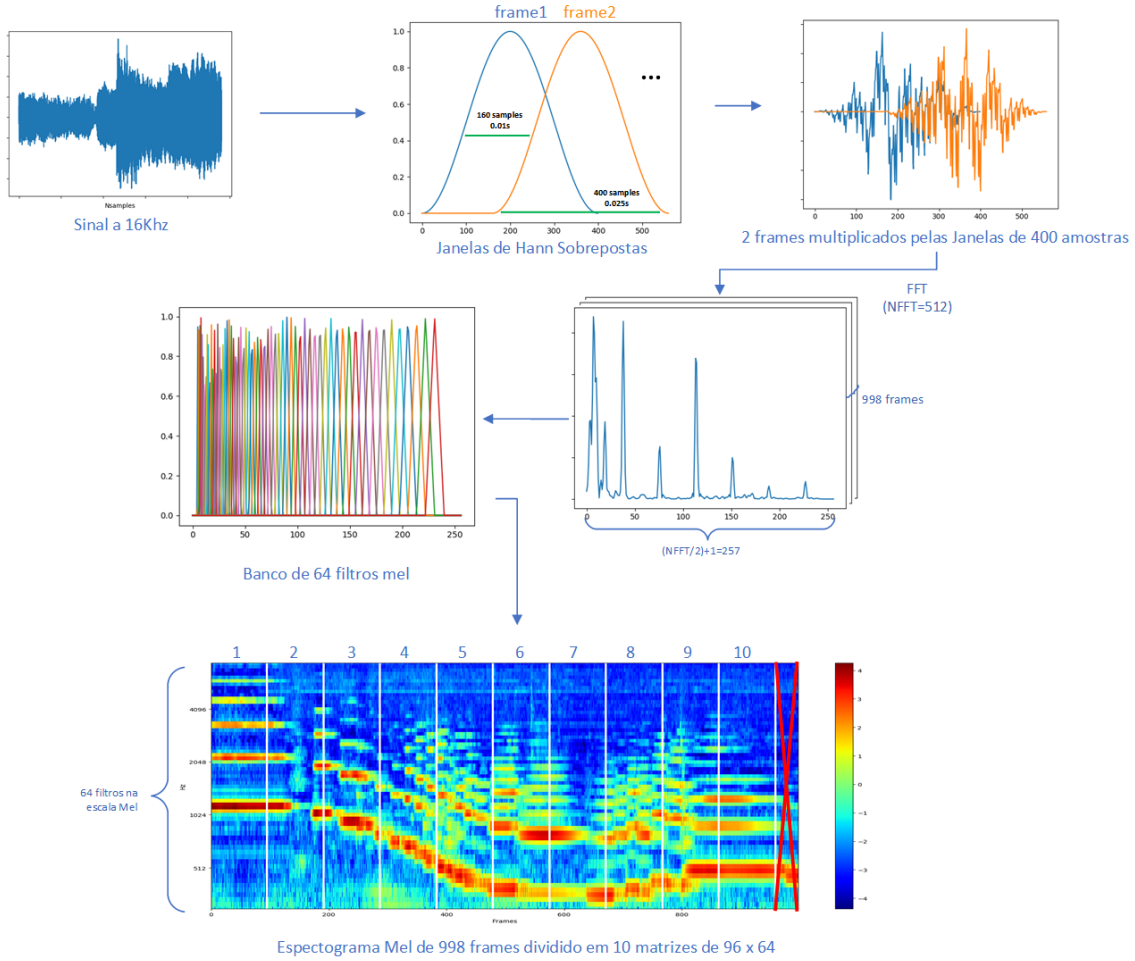

Figura 3.4 Processo de obtenção do espectograma Mel

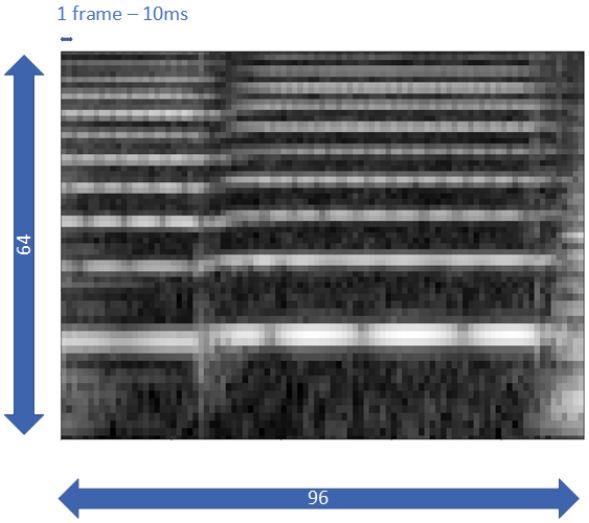

Figura 3.5 Matriz de entrada da rede VGGish (96 x 64)

PCA

A saída da rede, vetor de dimensão 128, é então uma boa representação do clip que foi transformado em espetograma de Mel e posteriormente processado por toda a VGGish. O AudioSet aplica ainda a transformação de **PCA** a este vetor. Apesar de normalmente esta técnica ser utilizada para redução dimensional, neste caso, a dimensão mantém-se. O AudioSet disponibiliza um vetor de médias (128) e uma matriz de vetores próprios (128 x 128). A figura 3.6 mostra o processo implementado para obter o PCA. É mostrado um gráfico (embedding[21],embedding[14]) de 2 dos 128 valores, um em função do outro. A entrada da rede são duas distribuições normais com médias e desvios padrão diferentes de 160.000 pontos cada (10 segundos). Em primeiro lugar os valores são subtraídos à sua média (**A**) com o objetivo de **tentar** descolar os dados para a origem (média = 0). Depois de deslocados, os dados são multiplicados pela matriz de vetores próprios. A multiplicação por esta matriz que foi obtida com base em muitos exemplares, faz com que os dados se desloquem na direção com maior variância (**B**). De seguida é feito o ***cliping***. Esta operação "limita"os dados isto é, dado um intervalo de dois valores, todos aqueles que estiverem fora desse intervalo são "empurrados"até às suas arestas (**C**). Por fim, os valores são normalizados num intervalo, neste caso [0:255], para no final serem convertidos para inteiros de 8 bits (**D**).

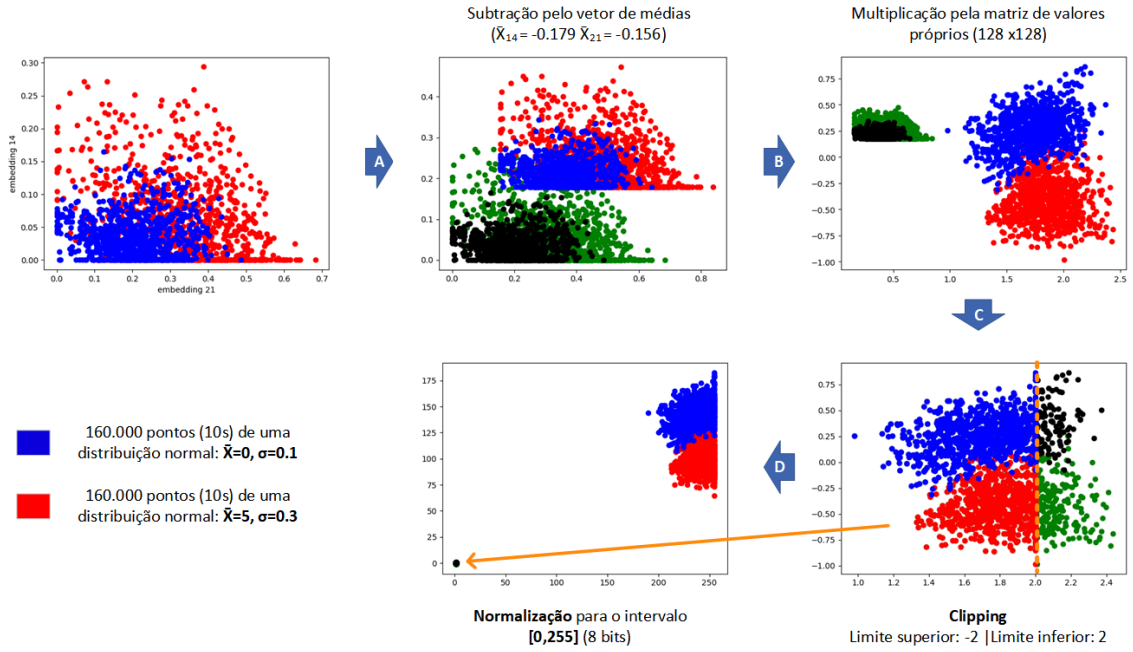

Figura 3.6 Implementação do PCA (AudioSet)

Uma vez que o AudioSet não dispõe dos ficheiros de áudio, foi executado um script que ao longo de várias semanas fez o download dos ficheiros de áudio desta base de dados mas apenas as 20 classes que são utilizadas no OpenMic. Foram selecionados os ficheiros *.csv* das 20 classes pretendidas. Estes ficheiros contêm a informação necessária para fazer o download de um video do YouTube dentro do intervalo de tempo correto. Quer por direitos de autor ou por restrições no que diz respeito à região, não foi possível obter todos os ficheiros que fazem parte da base de dados. Apesar disso, a quantidade de exemplares que não se conseguiram obter é muito reduzida quando comparada com o total de

ficheiros obtidos (**160.400**). A distribuição do número de exemplares por classes está representada na figura 3.7.

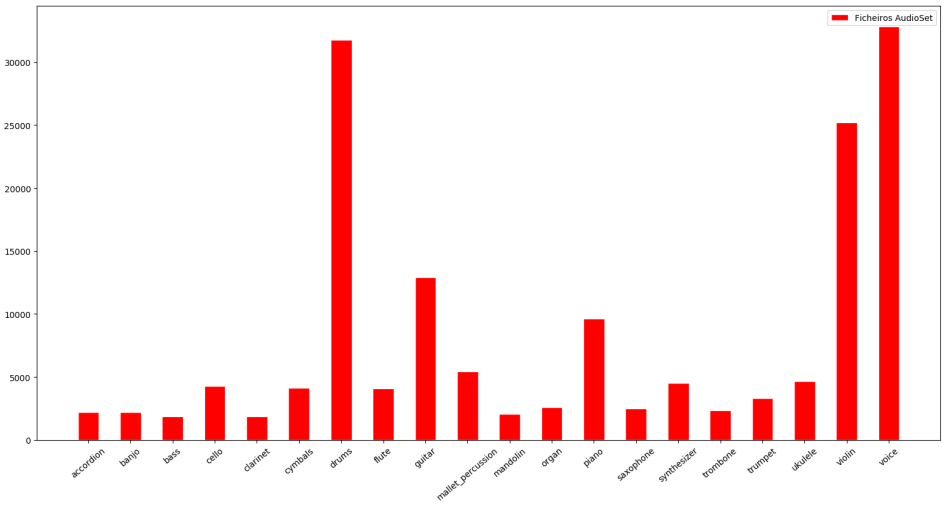

Figura 3.7 Distribuição do número de exemplares descarregados

3.1.2 OpenMic2018

A base de dados OpenMic2018 [17] é uma base de dados que contém 20.000 exemplares de trechos de áudio de 20 instrumentos musicais.

- Acordeão
- Banjo
- Baixo
- Violoncelo
- Clarinete
- Pratos
- Bateria
- Flauta
- Guitarra
- Xilofone
- Bandolim
- Órgão
- Piano
- Saxofone
- Sintetiz.
- Trombone
- Trompete
- Ukulele
- Violino
- Voz

Esta base de dados, também ela multi-etiqueta, é baseada no AudioSet isto é, a informação disponível constitui precisamente a mesma estrutura, 10 vetores de 128 valores para cada clip de áudio obtidos através do mesmo processo. O OM18 tem a vantagem de disponibilizar também os seus ficheiros de áudio. Pela negativa, a informação que existe para cada clip não é relativa a todas as classes mas sim, relativa a apenas algumas isto é, cada clip tem no máximo 4 (das 20) etiquetas anotadas.

Para disponibilizar os ficheiros, é necessário que estes estejam disponíveis para uso livre, ao contrário dos vídeos do YouTube e portanto esta BD foi gerada com clips do Free Music Archive (FMA) [9].

O FMA é um repositório online de gravações musicais disponíveis gratuitamente. Durante o processo de construção do OpenMic foram processados os clips deste arquivo que continham a licença Creative Commons. Posteriormente foi construída uma DNN (*InstrumentDNN*) [17] treinada com o

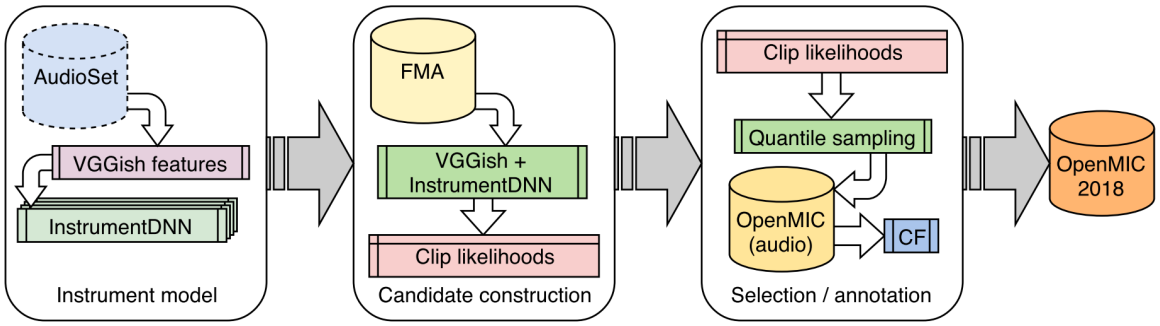

Figura 3.8 Processo de criação do OpenMic-2018 [17]

conteúdo do AudioSet relativo às 20 (das 526) classes correspondentes aos instrumentos musicais do OpenMic ("instrument model", fig. 3.8).

Depois de ter um modelo treinado, fizeram passar os dados do FMA pela *InstrumentDNN* ("candidate construction", fig. 3.8) e seccionaram **para cada classe**, 1000 clips daqueles que continham maior probabilidade e 500 cuja probabilidade era menor ("quantile sampling", fig. 3.8). Posteriormente os clips foram alvo de um processo chamado **croudsourcing** ("CF", fig. 3.8) que consiste na etiquetagem dos clips mas por parte da população. Foram selecionados os clips que eram coerentes entre a InstrumentDNN e o croudsourcing perfazendo o total de **20.000** exemplares.

Tal como o AudioSet, esta é também uma Weakly Labeled Dataset (WLD) e para além disso, conta com muitas mais etiquetas negativas do que positivas. A figura 3.9 mostra a distribuição das anotações para cada classe.

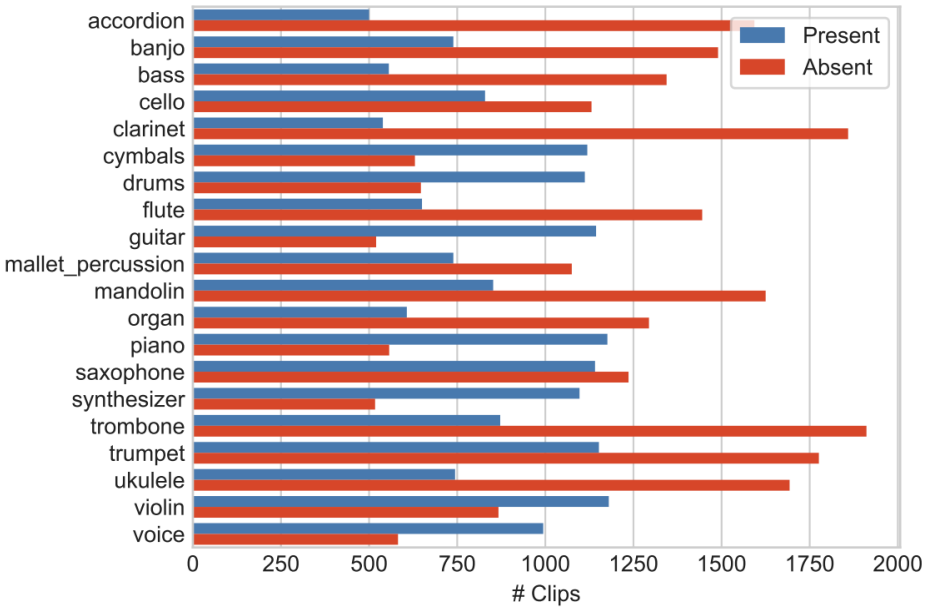

Figura 3.9 Distribuição das anotações por classe [17]

3.1.3 PureMic

Precisamente pelo facto de o AudioSet e o OpenMic terem etiquetas fracas, surgiu a necessidade de, no âmbito do presente trabalho, construir uma base de dados com etiquetas fortes (SLD). O PureMic é uma contribuição para este trabalho que foi construída com base nos exemplares do AudioSet, e tendo em conta as 20 classes do OpenMic2018. Depois de descarregar os ficheiros do AudioSet tornou-se possível a construção desta base de dados.

Esta nova e original base de dados, foi construída de forma a garantir que cada clip tem um e um só instrumento podendo assim garantir que em todos os 10 segundos de áudio um determinado instrumento está presente. Foi necessário ouvir, através de mono-avaliação, vários exemplares do AudioSet até chegar a 50 clips por instrumento resultando num total de 1000 clips de 10 segundos cada. É normal que apesar de conter apenas um instrumento, possam por vezes haver momentos de silêncio e por essa razão foi acrescentada uma 21ª classe que representa precisamente o silêncio que em muitos casos também é útil ser classificado. Com esta nova classe, o PureMic conta com **1058 exemplares** com etiquetas fortes.

Uma vez que foi feita com base no AudioSet, deve seguir os mesmos parâmetros e portanto não é permitido disponibilizar os ficheiros de áudio. Tal como no AudioSet, é possível disponibilizar o identificador do YouTube, o tempo de início e fim do clip e as etiquetas que neste caso é apenas uma por clip. A base de dados será então disponibilizada para uso livre.

3.1.4 Preparação dos dados

Para que os modelos funcionem da melhor forma possível, é necessária uma análise intensiva e preparação cuidada dos dados para assim, o modelo pelo qual se opta, poder tirar o máximo partido da sua arquitetura. Esta secção explica então alguns conceitos relacionados com o tratamento de dados que serão abordados mais à frente em paralelo com a implementação do trabalho prático.

A entrada da rede VGGish é de 96 frames de 10ms cada, por 64 filtros. Cada clip contém 10 entradas para a rede (10 x 96 x 64) portanto são necessárias metodologias para resolver este problema já que só é permitido processar uma de cada vez. O método proposto pela baseline (trabalho de referência) do OpenMic calcula a médias dos 10 embbedings (depois de passarem, cada um, pela VGGish) e assim, obtém apenas um embedding por clip (figura 3.10 [2]).

Esta técnica diminui consideravelmente o número de dados (10 vezes menos) o que em casos como a BD PureMic é desvantajoso já que conta com poucos exemplares quando comparada com as restantes.

Uma outra abordagem consiste em atribuir aos embeddings as etiquetas do seu clip que neste caso, levaria a 10 exemplares por clip. Esta abordagem pode levar a resultados piores se as bases de dados forem WLD. Uma classe pode estar presente apenas no início, mas ao assumir a etiqueta (ou etiquetas) do clip, os instantes finais herdam uma etiqueta que não lhes corresponde. Como a BD PureMic contém apenas um instrumento por clip que foi escolhido com cuidado, esta abordagem é mais vantajosa do que a anterior já que multiplica o número de exemplares por 10.

Desta forma, existem mais duas formas de apresentar os dados:

- Fazendo a média - 1 embedding por clip (1×128).

[2] https://www.youtube.com/embed/0rA7Sr4WfEU?start=60&end=70

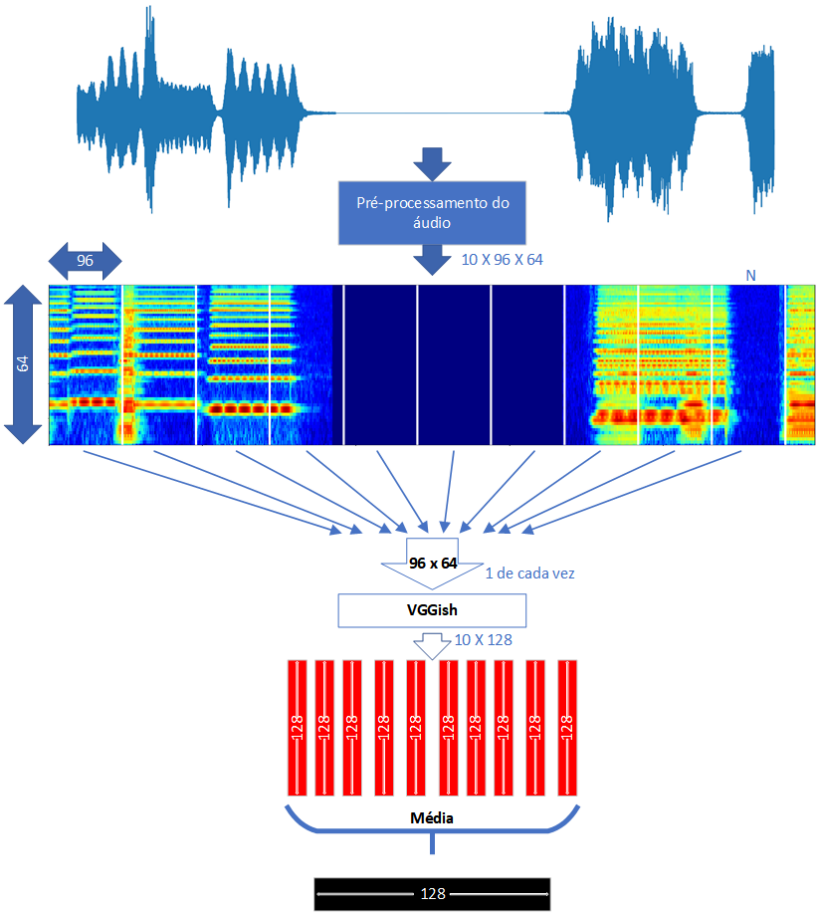

Figura 3.10 Média dos embeddings

- Assumindo que as instâncias têm todas a etiqueta do clip a que correspondem - 10 embeddings por clip (10×128).

Um parâmetro que também se pode alterar quando os ficheiros de áudio estão disponíveis é o avanço para a obtenção das matrizes de entrada da rede VGGish (96 x 64). O Audioset optou por processar estas matrizes sem sobreposição (avanço de 96 frames), razão pela qual os 10 segundos de cada exemplar são representados por uma matriz de dimensões **10** x 96 x 64.

Uma vez que este projeto tenta resolver um problema em tempo real pretende-se diminuir este avanço. Os dados foram pré-processados de forma a obter novos dados que em vez de avançar 0.96s, avançam 10ms (1 frame) ou 100ms (10 frames) o que permite uma maior resolução temporal. Quanto maior a resolução, maior a quantidade de dados. Não é possível aumentar a resolução para além dos 10ms porque esse é o intervalo de tempo a que corresponde cada uma das 96 colunas da matriz. Por outras palavras o avanço utilizado aquando do calculo da STFT é de 10ms.

Acrescem agora mais três formas de apresentar os dados, com três resoluções temporais distintas:

- avanço de 0.96s (sem sobreposição) - 10 matrizes
- avanço de 100ms - 91 matrizes
- avanço de 10ms - 903 matrizes

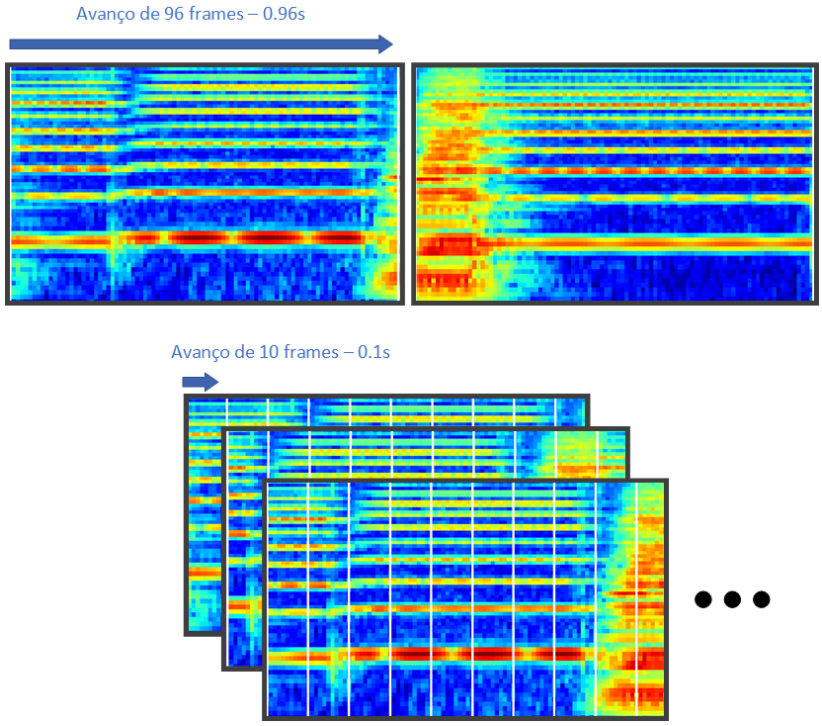

Figura 3.11 Diferentes Resoluções temporais

K-fold Cross Validation

Quando o número de dados não é suficiente, a técnica de K-fold Cross Validation é muito utilizada. Esta técnica, em primeiro lugar, divide os dados em **K** *folds* de tamanho igual. Posteriormente o modelo é treinado com **K-1** folds e testado com o *fold* que foi excluído. Este processo de treino é então repetido **K** vezes sendo que cada uma das novas iterações deve conter um *fold* de teste que ainda não tenha sido utilizado. A figura 3.12 ajuda a clarificar a forma como esta técnica é implementada.

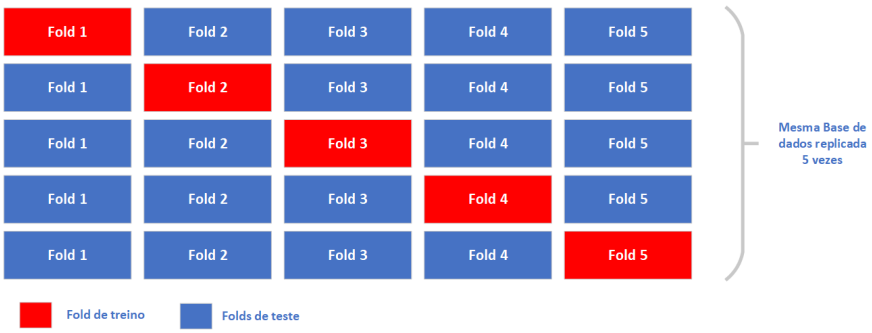

Figura 3.12 Implementação da técnica cross fold validation

No caso da figura 3.12, obtêm-se no fim dos 5 treinos, 5 resultados que juntos, representam melhor a performance do classificador. Com apenas um treino podem por coincidência os dados de teste indicar um resultado que na verdade não é aquele que o modelo representa.

Esta técnica requer que os dados sejam misturados para que a sua distribuição seja aleatória mas deve haver precaução na divisão treino/teste no sentido de garantir que se um determinado instante do clip *P* pertencer ao conjunto de treino os restantes instantes pertencentes ao mesmo clip *P* **devem** pertencer ao mesmo conjunto. Seria muito fácil obter resultados falaciosos se todos os instantes reservados para teste pertencessem a clips que também teriam instantes no conjunto de treino. É suposto neste tipo de técnicas os dados de teste serem completamente novos e desconhecidos para a rede, coisa que não aconteceria neste caso. Assim, a divisão de dados para o *cross fold validation* será feita com base nos clips e só depois de estes estarem atribuídos ao respectivo conjunto, se "desdobram"em 10, 91 ou 903 vetores (de dimensão 128) por clip.

3.2 Otimização

Esta secção retrata algumas técnicas essenciais para que as implementações sejam mais eficientes e menos pesadas a nível computacional. Apesar da utilidade de algumas técnicas, a utilização de uma GPU capaz de processar uma enorme quantidade de dados em paralelo é a forma de otimização que permite uma maior redução do tempo de processamento. Consequentemente, permite uma maior facilidade na execução de todo o trabalho.

Monitorização durante o treino

Geralmente, tem-se sempre acesso ao valor da função de custo ao longo do treino mas é também possível monitorizar outras métricas durante este período. A função de custo, que é usada para a otimização apenas utiliza os dados de treino sendo, normalmente, os dados de teste utilizados no fim do treino para atribuir uma pontuação à rede. É também prática comum, calcular o valor da função de custo **para os dados de teste** ao longo do treino (ao fim de cada época) sem deixar que estes influenciem a otimização. No fundo, serve apenas para ter uma ideia de como a rede se comporta com dados que não conhece.

O valor da função de custo para os dados de treino tende sempre a decrescer mas nem sempre até ao seu mínimo absoluto. Acontece que as redes neuronais começam a **decorar** os dados de treino em vez de os **generalizar** o que leva a um novo aumento do custo relativamente aos dados de teste. O facto de poder seguir a função de custo ao longo do treino mas calculada com base nos dados de teste é excelente para evitar este fenómeno, chamado **overfitting** (figura 3.13[3])

O overfitting (figura 3.13) acontece precisamente no ponto em que o custo dos dados de teste começa a crescer e em contraste o custo dos dados de treino continua a diminuir o que supostamente estaria a melhorar a rede. A partir do momento em que ocorre overfitting, deixa de ser vantajoso continuar a treinar.

Early stop

Para evitar a ocorrência do overfitting é comum utilizar a técnica de early stop. É precisamente no instante assinalado na figura 3.13 que ocorre o early stop isto é, por mais épocas que ainda faltassem treinar, o treino é interrompido já que não era vantajosa a sua continuação. Resulta assim uma rede mais capaz de identificar os padrões pretendidos.

[3] https://commons.wikimedia.org/wiki/File:Overfitting_svg.svg#/media/File:Overfitting_svg.svg

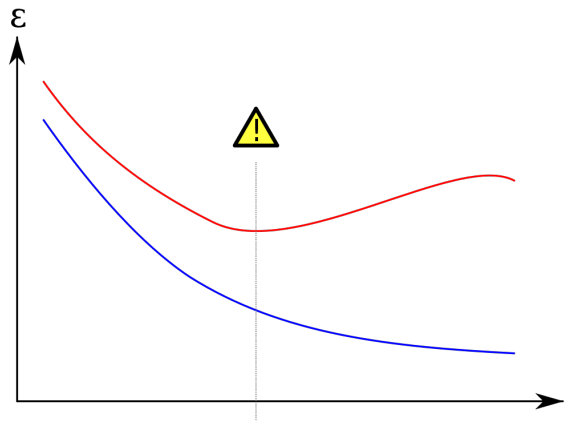

Figura 3.13 Valor do custo dos dados de treino e de teste ao longo de várias épocas

hyperparameter tunning

Um dos principais objetivos quando se implementa um modelo de deep Learning é encontrar os hyperparametros que levam ao melhor resultado. Estes parâmetros podem ser a configuração da arquitetura da rede, o tipo de funções de ativação, o learning rate, entre muitos outros que fazem com que a procura pela melhor configuração seja muito demorada e muito dispendiosa a nível computacional. Hyperparameter tunning é o processo que tenta precisamente encontrar a melhor configuração dos vários parâmetros a considerar. Existem certas configurações que à partida são melhores do que outras mas nada nos garante que não estejamos a considerar certos pormenores que podem fazer a diferença, principalmente quando o objetivo é melhorar a performance da rede por mais pequena que esta melhoraria possa ser.

Grid Search

Grid Search é uma técnica que ajuda a resolver o problema referido. Considerando por exemplo que queremos experimentar 5 parâmetros na rede implementada como por exemplo:

1. Número de layers da rede: 1,2 ou 3
2. Número de neurónios de cada layer: 1024 ou 2048
3. Learning rate: 0.01 ou 0.001
4. Função de ativação: sigmoid ou softmax
5. Número de épocas: 100 ou 150

Através da técnica de Grid Search o modelo é então treinado $3 \times 2 \times 2 \times 2 \times 2 = 48$ vezes, número de todas as configurações possíveis com estes 5 parâmetros. As métricas a registar podem ser várias e a secção seguinte ajuda a clarificar este conceito. A ideia é que se deve registar para cada uma das 48 iterações a métrica pretendida e no fim verificar qual a melhor. Isto indica qual a melhor configuração ou pelo menos dá uma ideia para onde esta tende. A framework **TensorBoard** explicada na secção

4.1 ajuda muito neste processo assim como a utilização de uma GPU. A utilização desta unidade de processamento diminui consideravelmente o tempo de computação.

3.3 Métricas

Para poder comparar os resultados de vários modelos e ter a noção se uma determinada implementação está num bom caminho ou não, é necessário ter uma referência isto é, um ou mais números que avaliem as implementações utilizadas. Geralmente representados em percentagens, este tipo de valores chamam-se métricas e existem várias, capazes de se adaptar ao problema em causa. O tipo de métrica a escolher varia consoante o tipo de problema ou seja, se é multi etiqueta, multi classe, se a base de dados está equilibrada e pode também depender do tipo de finalidade que se pretende. Ao longo desta secção serão clarificados todos os pormenores acerca das métricas a utilizar. Geralmente uma métrica que avalia um determinado modelo é obtida apenas através dos dados que foram guardados para teste já que são dados que a rede desconhece, simulando assim um caso real.

3.3.1 Métricas dependentes de um limiar

Supondo que se pretende identificar uma guitarra em diversos exemplares que contêm sons de vários instrumentos musicais ou seja é necessário decidir entre **guitarra** ou **não guitarra**. Depois de treinar um determinado modelo, os resultados da fase de teste correspondem à probabilidade de cada exemplar conter uma guitarra. Estas probabilidades podem ser representadas pelo histograma representado na figura 3.14[4].

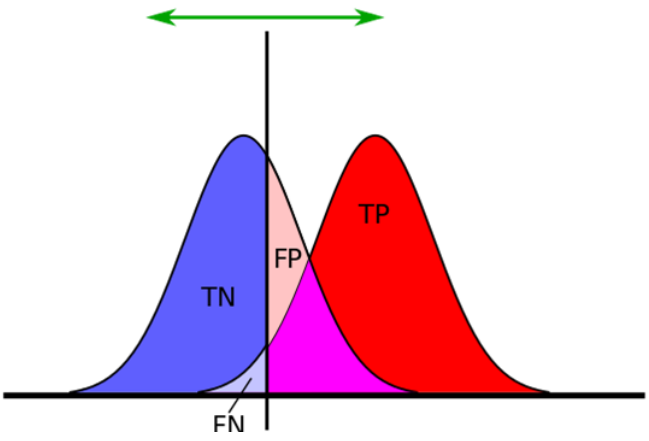

Figura 3.14 Histograma da classificação de uma classe

As **métricas dependentes de um limiar** ou threshold têm em conta um limiar que deverá maximizar a métrica escolhida. Neste caso e geralmente, esse limiar deve ter o valor de $\sigma = 0.5$. Este valor, tipicamente escolhido quando trabalhamos com probabilidades (por ser o valor intermédio) diz-nos que os exemplares com probabilidades a baixo de σ são considerados como **não guitarra** e por outro lado, quando a probabilidade é superior a σ o exemplar é considerado como **guitarra**.

[4] https://commons.wikimedia.org/wiki/File:Receiver_Operating_Characteristic.png

É importante dizer que este limiar serve para decisão já que o valor verdadeiro (**ground truth**) de todos os exemplares é previamente conhecido e inalterável. É normal que probabilidades perto do limiar contenham erros isto é, um exemplar com probabilidade de 0.51 é considerado (**predicted**) positivo mas está tão perto do negativo que podemos estar perante um **falso positivo** ou seja, apesar de a decisão ter sido **guitarra**, o **ground truth** deste exemplar é **não guitarra**. Portanto em algumas aplicações e dependendo do objetivo de quem as implementa pode ser útil deslocar este threshold.

Matrizes de confusão

Falso positivo, palavra utilizada no parágrafo anterior, é um dos quatro termos associados a uma matriz de confusão (figura 3.15). Estes termos, também representados na figura 3.14, dependem, tal como se pode verificar do threshold escolhido e significam o seguinte:

- True Positives (TP): Casos em que a decisão é **guitarra** e o ground truth do exemplar também o é.
- True Negatives (TN): Casos em que a decisão é **não guitarra** e o groundtruth deste exemplar é igual à decisão tomada.
- False Positives (FP): Casos em que a decisão é **guitarra** mas na verdade não corresponde ao ground truth deste exemplar que é **não guitarra**.
- False Positives (FN): Casos em que a decisão é **não guitarra** mas, ao contrário do que foi previsto, o ground truth para este exemplar é **guitarra**.

Como se percebe facilmente, o ideal seria que não houvesse nem **False Positives** nem **False Negatives** o que faria com que o valor de qualquer métrica fosse 100%. Em problemas da vida real este cenário é praticamente impossível de acontecer.

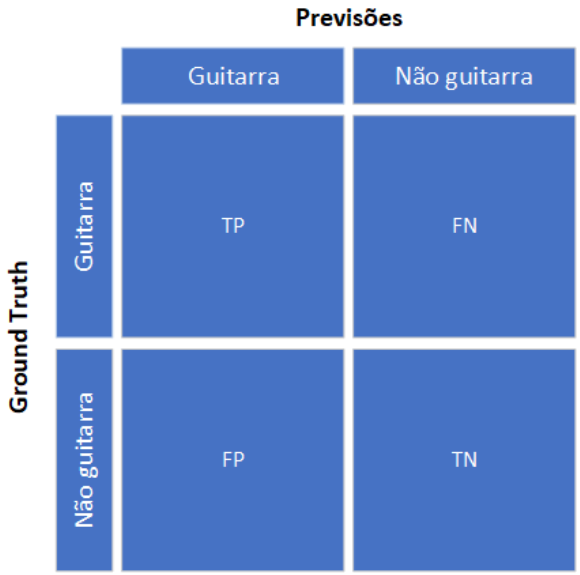

Figura 3.15 Matriz de confusão de uma classe

Precisão

Esta métrica responde à pergunta *Daqueles que se decidiram como **guitarra** quantos é que efectivamente são?* que pode ser representada através da fórmula

$$Precisão = \frac{TP}{TP+FP} \tag{3.1}$$

É uma métrica que dá então importância aos exemplares positivos, descartando os negativos. É possível obter uma precisão muito elevada valor que pode ser enganoso se pretendermos que o classificador, neste caso, seja também capaz de rejeitar exemplares de uma forma eficiente. A próxima métrica explica melhor este caso.

Recall

Recall, sensibilidade ou true positive rate (TPR) são os nome que se podem dar a esta métrica. A pergunta à qual esta métrica responde é *Quantos dos exemplares que são efetivamente **guitarra**, é que foram classificados como tal?* e a formula que a representa é a seguinte:

$$Recall = \frac{TP}{TP+FN} \tag{3.2}$$

Ao contrário da precisão, esta métrica dá valor a quantos dos exemplares **guitarra** foram classificados como tal descartando exemplares cujo ground truth é **não guitarra** mas que também são classificados como **guitarra**. O recall, tal como a métrica anterior também pode então induzir em erro quando é muito elevado. Geralmente quando uma métrica é 100% a outra pode ser próxima dos 0% portanto deve existir um tradeoff entre as duas com o objetivo de as equilibrar. Este equilíbrio é determinado pelo threshold escolhido. A figura 3.16 ajuda a entender quando é que as métricas podem ter valores muito elevados mas que não é propriamente vantajoso para a aplicação que se pretende.

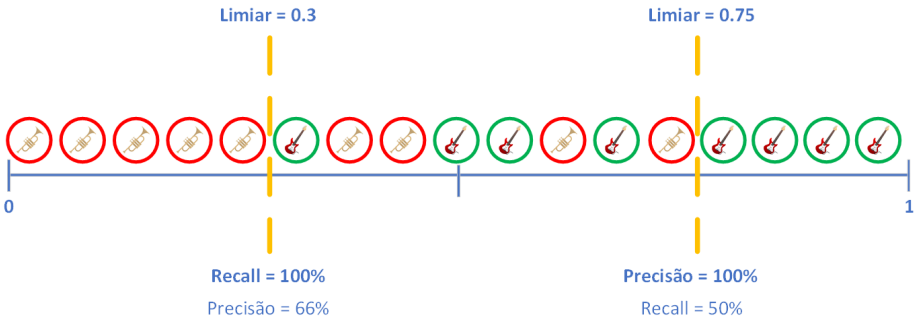

Figura 3.16 Precisão e recall elevados

F1 Score

Finalmente, o F1 Score é uma métrica que combina as duas anteriores num só valor. A sua fórmula corresponde à média harmónica entre a precisão e o recall:

$$F1 = \frac{2 \times precisão \times recall}{precisão + recall} \tag{3.3}$$

É uma métrica que é muito útil quando as bases de dados são desequilibradas isto é, quando o número de exemplares positivos é muito maior ou muito menor que o número de exemplares negativos.

Accuracy
Muitas vezes para simplificar e também quando não se pretende olhar para as métricas anteriormente faladas a accuracy ou exatidão é uma métrica que representa a taxa de acertos total:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.4}$$

Quando o número de classes é superior a 2 esta métrica pode ser representada pela média das *accuracies* de cada classe. Apesar de ser um métrica bastante abrangente, falha quando as bases de dados são desiquilibradas.

Fall-out
Esta métrica também chamada de False positive Rate (FPR) é menos utilizada como referência mas muito utilizada para construir o gráfico AUC (ver a seguir), razão pela qual será aqui abordada. A sua fórmula é a seguinte:

$$FPR = \frac{FP}{FP+TN} \tag{3.5}$$

Ao contrário das métricas anteriores o valor ideal desta é 0 ou seja, o objetivo passa pela sua minimização.

3.3.2 Métricas independentes de um limiar

Uma vez que nem sempre é necessário decidir (escolher o limiar) e é mais vantajoso analisar os modelos para todas as possibilidades existem métricas que através de um número conseguem representar a pontuação de um modelo sem que seja necessário atribuir um thershold para a decisão.

Receiver Operating Characteristic
Receiver Operating Characteristic (ROC) ou ROC Curve, é um gráfico que representa a performance de um determinado classificador. Este gráfico ilustra uma curva do recall em função do Taxa de Falsos Positivos (FPR) para todos os thresholds possíveis. A figura 3.17 representa um exemplo deste tipo de gráficos.

A área por de baixo da curva representada (Area Under Curve) ou AUC é então uma métrica que representa a performance do classificador sem especificar algum threshold. Quando o problema contém mais do que uma classe, é feita a média de todas as AUC's resultando finalmente na mAUC (mean Area Under Curve).

O ponto que maximiza a AUC, assinalado na figura, é ponto cujo recall é 1 mas o FPR é 0.

Precision Recall Curve
Muito semelhante à AUC a curva de precision Vs recall, representa, tal como o nome indica, a precisão em função do recall e um exemplo deste tipo de gráfico está representado na figura 3.18

O nome que se dá à area por de baixo desta curva chama-se, desta vez, Average Precision (AP) e no caso de multi-classes a métrica que representa a performance total do classificador é a média destas áreas chamada mAP (mean Average Precision)

O ponto que maximiza esta área, neste caso e como é espectável por análise à secção anterior, é quando a precisão e o recall são ambos 1.

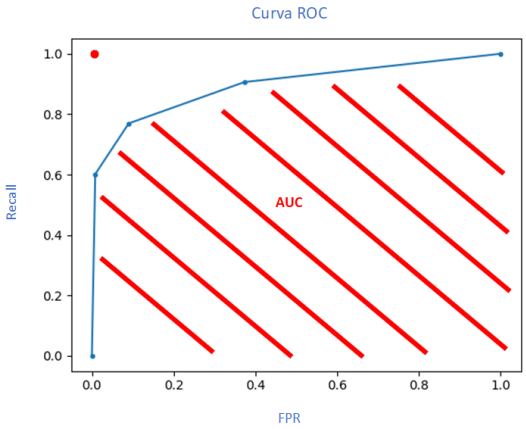

Figura 3.17 Curva ROC

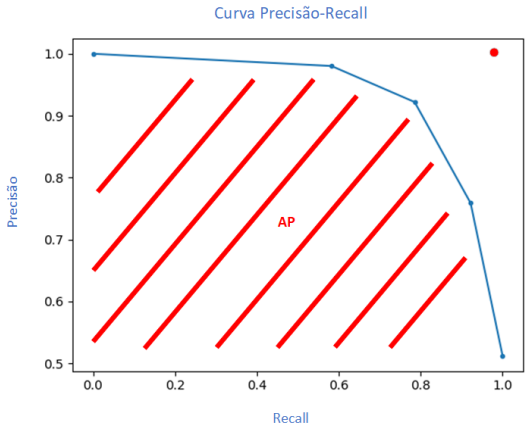

Figura 3.18 Curva Precision Recall

No fundo, ambas as métricas indicam o quanto misturadas as classificações estão (em casos como o da figura 3.16). Idealmente, se houvesse um limiar que conseguisse separar as duas classificações e obter 100% tanto para a precisão como para o recall, a área de ambas as curvas seria 1.

Capítulo 4

Desenvolvimento

Este capítulo é dedicado à explicação de como todo o trabalho foi desenvolvido.

4.1 Ferramentas utilizadas - Frameworks

No âmbito deste projeto será utilizada a linguagem de programação de alto nível, interpretada, o **Python**. Esta linguagem dispõe de diversas bibliotecas essenciais para a realização deste projeto sendo que serão destacadas as mais importantes.

- TensorFlow (TF): É das bibliotecas mais utilizadas para tratar problemas de machine learning. Esta biblioteca foi desenvolvida pela equipa Google Brain para contribuir para a investigação nas áreas de machine learning e Redes Neuronais profundas. Este tipo de redes trabalham com arrays multi-dimensionais, ou por outras palavras, tensores de onde vem o nome desta biblioteca. É uma biblioteca que trabalha com redes neuronais a muito baixo-nível pelo que é possível configurar todos os pormenores por mais evidentes que possam ser. Também serão utilizadas algumas ferramentas disponíveis com a próxima versão, TF2.0.

- Keras: é uma API de alto-nível que funciona "por cima"do TensorFlow. Simplifica os problemas de baixo-nivel do TF sendo ideal para implementações mais rápidas e menos pormenorizadas. Para além disso, está bem documentada o que faz desta API excelente para quem pretende começar com este tipo de bibliotecas.

- TensorBoard: Ainda relacionada com o TensorFlow, esta ferramenta é excelente para fazer debugging das implementações efetuadas. Permite vizualizar gráficos de métricas, histogramas, comparar hyper-parametros entre muitas outras ferramentas de visualização que têm muita utilidade. É esta a ferramenta utilizada para implementar o *grid-search*, técnica já abordada anteriormente.

- scikit-learn: Biblioteca desenvolvida para python que inclui vários algoritmos de classificação, regressão e clustering, alguns deles utilizados para o desenvolvimento do projeto. Para além disso, inclui ferramentas também úteis para o cálculo e apresentação de vários tipos de métricas.

4.2 Otimização dos Modelos

Para obter a arquitectura ideal, foi efetuadada a técnica de grid search (ver sec. 3.2) para obter os hyperparametros ideais. O recurso à *GPU Nvidia Geforce GTX 1080* permitiu acelerar significativamente todo o processo. Os parâmetros considerados no grid search foram os seguintes:

- Número de neurónios na primeira camada: 512, 1024, 2048 ou 4096.
- Número de neurónios na segunda camada: 512, 1024, 2048 ou 4096.
- Otimizador: Adam, SGD, Adagrad, RMSprop
- learning rate: 0.0001, 0.001 ou 0.01
- tamanho do batch: 512, 1024 ou 2048

Antes de efetuar este processo, foi feito um grid search apenas para decidir quantas camadas se usariam, razão pela qual esse não é um parâmetro a explorar dentro dos referidos anteriormente.

4.3 Dados disponíveis

Como ponto de partida tem-se então os seguintes dados:

- AudioSet: 160.400 ficheiros de áudio e os respetivos embeddings (160.400 x 10 x 128) com 20 classes e etiquetas fracas.
- OpenMic-2018: 20000 ficheiros de áudio e os respetivos embeddings (20000 x 10 x 128) com 20 classes e etiquetas fracas.
- PureMic: 1058 ficheiros de áudio e os respetivos embeddings (1058 x 10 x 128) com 21 classes e etiquetas fortes. Um e um só instrumento por clip.

Com base nos procedimentos indicados na secção 3.1.4, a tabela 4.1 indica as abreviaturas das diferentes configurações abordadas tomando como exemplo uma base de dados de **3 exemplares de 10 segundos cada**. Estas abreviaturas farão parte, consoante a configuração, do nome das bases de dados originais. A tabela 4.2 resume todas as bases de dados a ser utilizadas ao longo do desenvolvimento.

Tabela 4.1 Abreviaturas das diferentes configurações

Abreviatura da configuração	Descrição	Dimensão
MEAN	média dos embeddings de cada clip	3 x 128
R096	Resolução temporal de 96 frames (fig. 3.11)	3 x 10 x 128
R10	Resolução temporal de 10 frames (fig. 3.11)	3 x 91 x 128
ALL_R10	juntam-se todos os embeddings (resolução **R_10**)	273 x 128

Seroa de esperar que o suposto número de embeddings (10 ou 91 por clip, consoante a resolução temporal) multiplicado pelo número de clips resulte na dimensão das bases de dados "ALL". Isto acontece com o OpenMic (14.914 x 91 = 1.357.174) uma vez que os ficheiro são disponibilizados

Tabela 4.2 Bases de dados a ser utilizadas

Nome da BD	Descrição	Dimensão
OM_treino_R10	Set de treino da BD OpenMic-2018	14.914 x 91 x 128
OM_treino_ALL_R10	OM_treino_R10 com os embeddinggs juntos	1.357.174 x 128
AS_ALL_R10	embeddings do AudioSet juntos	14.582.011 x 128
PM_MEAN	média dos embeddings do PureMic	1058 x 128
PM_ALL_R10	embeddings do PureMic juntos	96.278 x 128

"oficialmente". No caso do AudioSet não acontece o mesmo porque alguns dos ficheiros que foram descarregados através do script não têm exatamente 10 segundos o que faz diminuir ligeiramente o número de embeddings. Da mesma forma, uma vez que provém do AudioSet, o PureMic também tem alguns exemplares com menos de 10 segundos.

4.4 Estratégia

O repositório do OpenMic disponibiliza uma implementação de um modelo Random Forest (RF) cuja entrada é o vetor de 128 valores. A implementação deste modelo foi utilizada para testar a nova base de dados PureMic. A figura 4.1[1] mostra uma classificação feita a cada 10 ms pelo modelo RF, treinado apenas com a BD **PM_MEAN** 4.2.

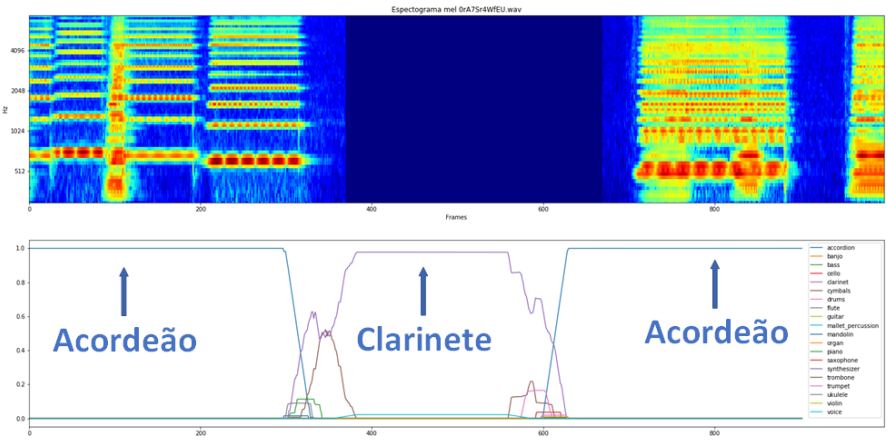

Figura 4.1 Classificação do RF sem a classe de silêncio

O momento de silêncio é claramente visível, mas como a rede não conhece este "som", força a tivação da classe clarinete. Isto também acontece porque a soma das probabilidades a cada instante tem de ser igual a 1 tal como acontece com a função de ativação softmax (ver sec. 2.3).

Ao acrescentar a classe de silêncio o problema anterior fica resolvido: quando recebe momentos de silêncio, o classificador não tem dúvidas relativamente à classe que deve ativar (figura 4.2).

[1] https://www.youtube.com/embed/0rA7Sr4WfEU?start=60&end=70

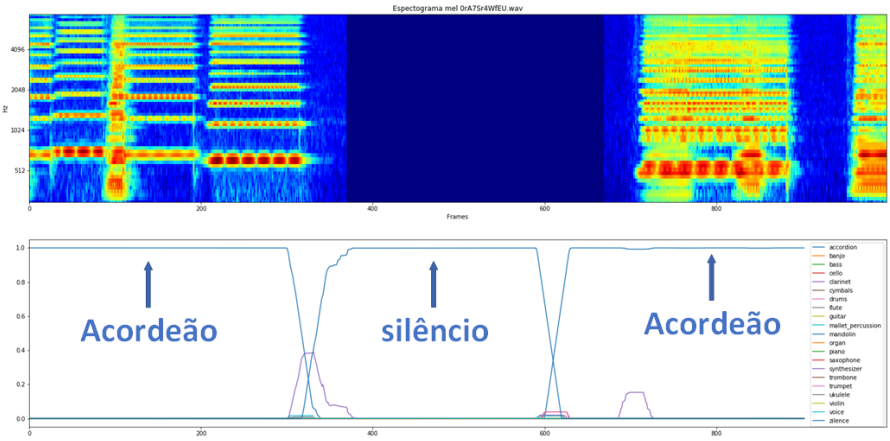

Figura 4.2 Classificação do RF com a classe de silêncio

Tal como já referido, a PureMic é uma SLD ou seja, uma base de dados com etiquetas fortes e para além disso, cada clip tem um e um só instrumento. Assumir desta vez que todas as matrizes herdam a classe do clip a que correspondem, à semelhança do BOW (ver sec. 2.5), é muito mais seguro e também aumenta a quantidade de dados. Apesar disso é inevitável que estes clips tenham momentos de silêncio. É também por esta razão que a classe de silêncio surge uma vez que com ela, é possível identificar os momentos de silêncio que apesar de poucos estavam a herdar etiquetas de um determinado instrumento e que na verdade não as mereciam (como aconteceria no exemplo das figuras 4.1 ou 4.2).

4.4.1 PM1

PM1 é uma base de dados que tem como finalidade purificar as etiquetas do PureMic. Para começar, foi treinada uma rede neuronal, **NN_MEAN**, com a base de dados **PM_MEAN** (tabela 4.2). A média remove qualquer resolução temporal não permitindo distinguir os instantes em que ocorre cada instrumento mas uma vez que a BD contém apenas um instrumento por clip, a média dos 10 segundos contém de certeza informação do instrumento a que a etiqueta corresponde (mesmo tendo alguns momentos de silêncio). Depois de efetuar o grid search, os parâmetros que obtiveram, em conjunto, melhores resultados foram 4096 neurónios para a primeira camada, 2048 para a segunda, com um learning rate de 0.001, tamanho do batch de 512 e o otimizador Adam (figura 4.3). O número de épocas é controlado pela técnica de *early stopping* para evitar o *overfitting* (secção 3.2).

Obtém-se então um classificador que ainda não atinge os objectivos pretendidos mas consegue dar algumas certezas relativamente às 21 classes: 20 instrumentos e a classe de silêncio, que no fundo é a que será mais útil para a etapa seguinte.

A matriz de confusão da figura A.1 mostra o resultado do treino da rede **NN_MEAN** suportado pelos 5 folds do cross fold validation (ver sec. 3.1.4) somados. Resulta uma accuracy (secção 3.3) final de **94.80%**.

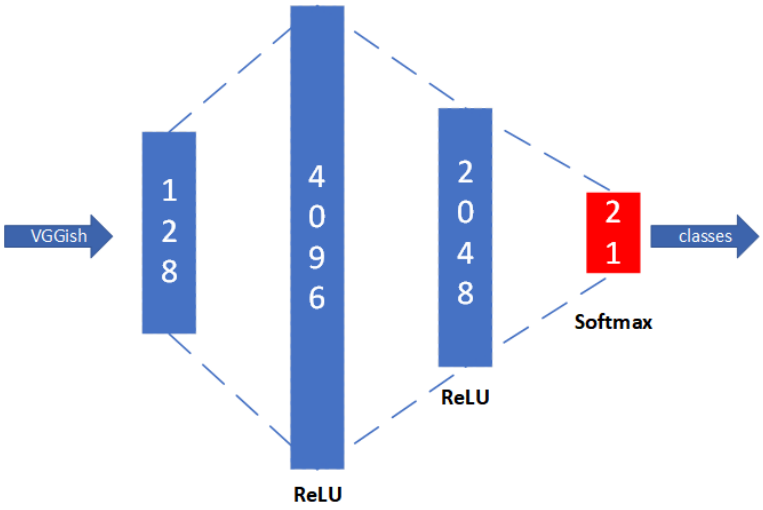

Figura 4.3 Arquitetura da rede *NN_MEAN*

Posteriormente será utilizada a base de dados **PM_ALL_R10** (tabela 4.2). Fazendo passar todos estes dados pela rede neuronal *NN_MEAN* é possível obter classificações **no tempo** para cada uma das 21 classes, tal como acontece na figura 4.2. Isto permite identificar momentos de silêncio que tinham a etiqueta do clip a que correspondem. Quando num instante a classe de silêncio é classificada acima de um limiar, $k1 = 0.5$, a etiqueta desse instante é alterada. Para além disso, os momentos restantes também devem ser superiores a um limiar $k2 = 0.2$ que garante que os frames restantes pertencem **efetivamente** ao instrumento indicado na etiqueta e descarta frames abaixo deste limiar que representam momentos que não são nem de silêncio nem do suposto instrumento. Esta etapa está representada na figura 4.4 com o nome **filtragem**. Esta figura representa o processo de obtenção da base de dados **PM1** que conta agora com **81.032** embeddings.

A rede *NN_MEAN* foi treinada com poucos exemplares e principalmente, treinou com a média dos embeddings (PM_MEAN) o que significa que não é tão sensível a embeddings "puros"saídos directamente da rede VGGish (com PCA). Uma vez que o silêncio é uma classe que se distingue facilmente das restantes, esta rede serviu apenas para purificar ainda mais a BD PureMic através da identificação dos momentos onde esta classe ocorre.

4.4.2 PM2

PM2 é uma base de dados, que contém exemplares de todas as bases de dados deste projeto. Foi então treinada, com a base de dados *PM*1, uma nova rede neuronal **NN_ALL**, com a mesma arquitetura da rede *NN_MEAN* (figura 4.3). Esta rede será agora capaz de fazer classificações ao longo do tempo uma vez que foi treinada com exemplares de 10 frames de resolução temporal e sem média aplicada. A matriz de confusão da figura A.2 mostra o resultado do treino da rede **NN_ALL** suportado pelos 5 folds do cross fold validation somados resultando numa accuracy final de **95.99%**. Este resultado vem confirmar que a base de dados *PM*1 é uma base de dados com etiquetas ainda mais fortes. Apesar de neste caso haver discriminação no tempo (cenário mais suscetível a erros) os resultados, ainda assim, são melhores do que no caso anterior (*NN_MEAN*).

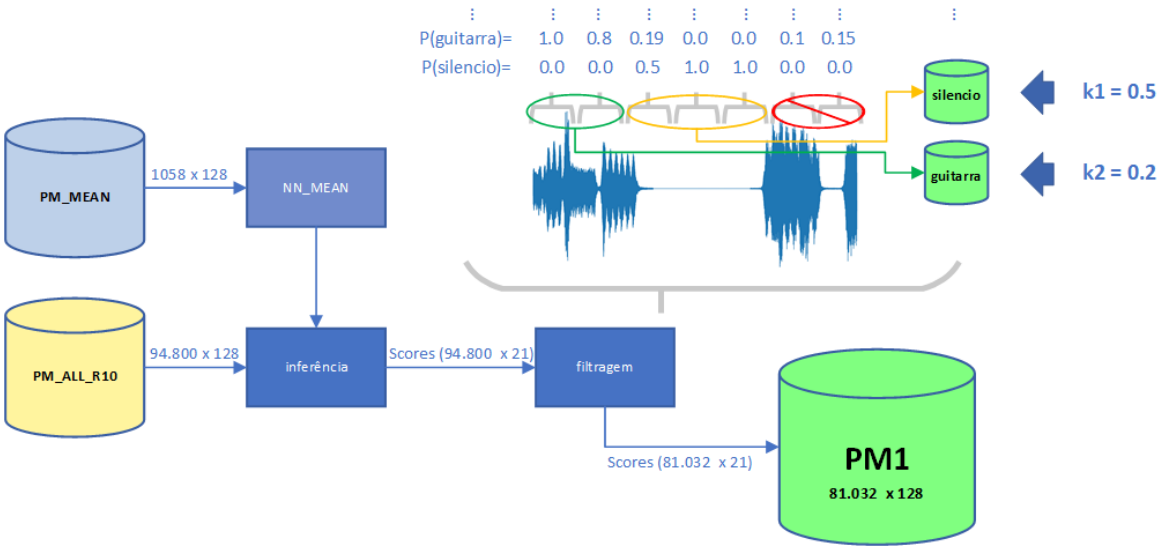

Figura 4.4 Processo de obtenção da BD *PM*1

Com o objetivo de aumentar a base de dados *PM*1, todos os clips das restantes BD's (OM_treino_ALL_R10 e AS_ALL_R10) serão passados pela rede **NN_ALL** e aqueles onde houver grandes certezas (**classificação > 0.85**) são guardados. Antes de pertencer à nova BD, **PM2**, as classificações superiores ao limiar terão também de corresponder à etiqueta do clip (figura 4.5). Resulta numa BD de 6.228.472 de embeddings.

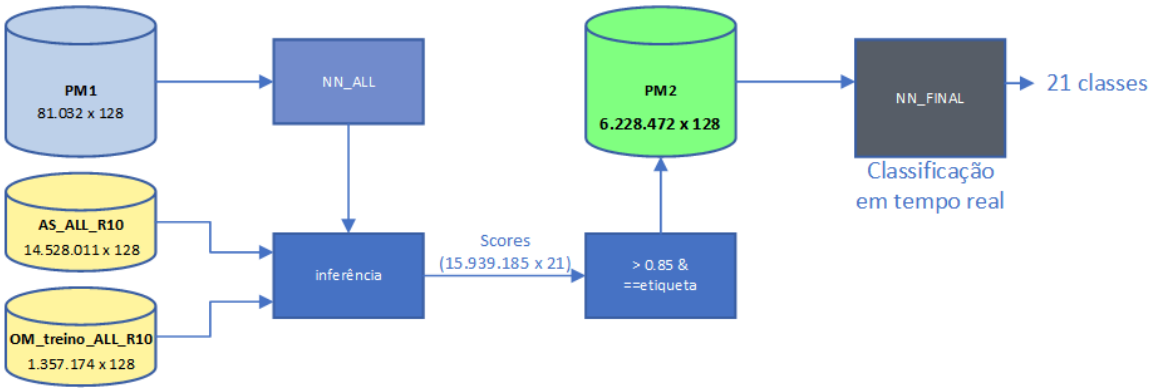

Figura 4.5 Processo de obtenção da BD *PM*2

A rede **NN_FINAL**, com a mesa configuração que as anteriores, será a rede final deste projeto. Esta rede é capaz de identificar instrumentos em tempo real o que permitirá fortalecer as etiquetas da base de dados OpenMic-2018. O facto de poder classificar com uma determinada precisão temporal permite conhecer, a cada instante, a classificação para todas as 21 classes.

4.5 Caso Prático

A figura 4.6 mostra a classificação, com resolução temporal de 10ms, ao longo de 10 segundos de um clip do YouTube [2]. É interessante verificar que com esta abordagem, se notam claramente três classes que se destacam e que são precisamente as classes associadas a este clip, **piano** , **violino** e **violoncelo** tal como se pode verificar na figura 4.7 que representa um instante do vídeo referido.

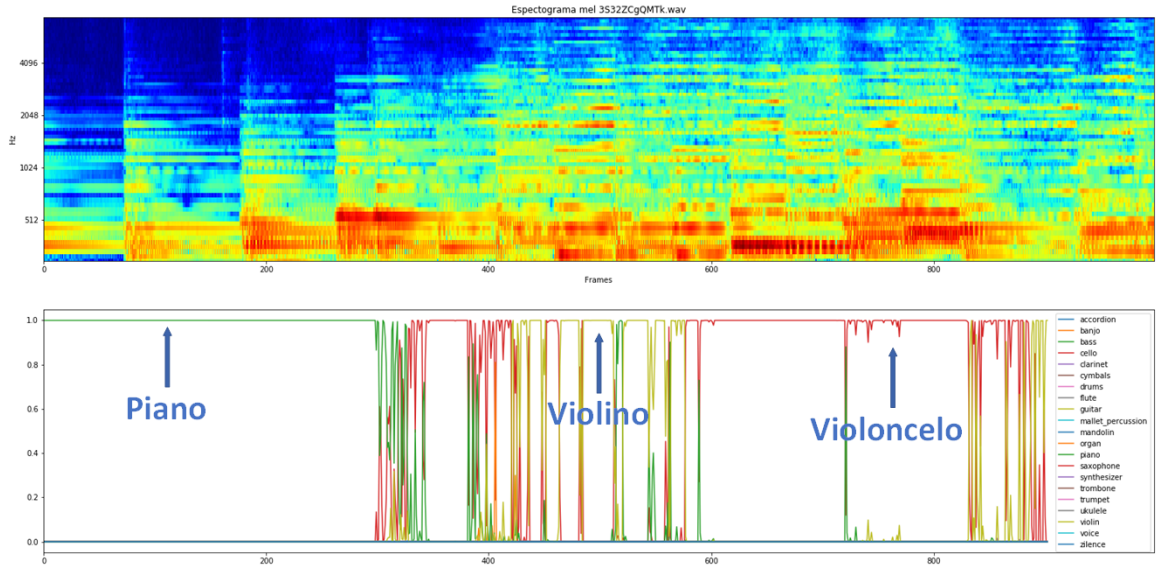

Figura 4.6 Classificação de um clip de 10 segundos com resolução temporal de 1 frame (10 ms)

Isto prova que é possível converter este tipo de classificações no tempo, numa etiqueta que represente os 10 segundos precisamente como as etiquetas disponíveis no AudioSet e OM18. Para isso será considerada a estratégia de ***mean pooling***, que calcula para cada classe, a média de todas as classificações (neste caso 903).

[2] https://www.youtube.com/embed/3S32ZCgQMTk?start=300&end=310

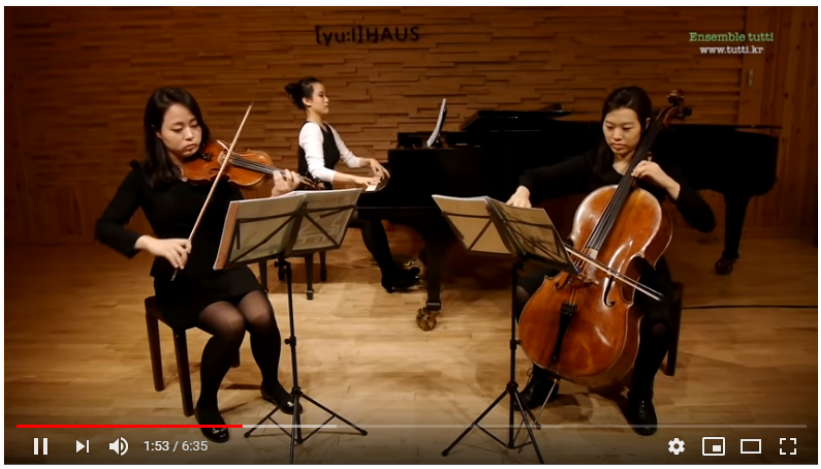

Figura 4.7 Imagem correspondente ao vídeo que contém o áudio classificado

Capítulo 5

Resultados e discussão

Este capítulo serve para apresentar e comparar os resultados tendo em conta diversos modelos e diversas BD para teste. Os modelos a considerar serão os seguintes:

- OM18-Baseline (RF)
- Modelo de atenção treinado com a BD OM_treino_R10 (tabela 4.2)
- NN_FINAL + Mean Pooling

5.1 Modelos

Baseline

Como referência, o OpenMic-2018 disponibiliza uma baseline que consiste num modelo Random Forest (RF) cujas métricas são avaliadas com base na base de dados de teste do OM18. Esta baseline utiliza 1 classificador binário para cada instrumento (20 RF's) e a sua entrada é a média dos 10 embeddings de cada clip. Estes resultados, apresentados na tabela 5.1, são a única referência disponível sendo os resultados restantes, obtidos no decorrer deste projeto.

Modelo de Atenção

O modelo de atenção (ver sec. 2.5), foi replicado do trabalho de Kong et al. [21] que disponibiliza o código para a sua implementação. Os dados utilizados para treino são os da base de dados OM_treino_R10 (tabela 4.2). Os resultados são apresentados na tabela 5.1.

NN_FINAL + Mean pooling

Com esta estratégia, abordada no capítulo anterior, é possível comparar os resultados de uma rede que classifica instrumentos ao longo do tempo, com a baseline que representa resultados para 10 segundos. A *NN_FINAL* classificou todos os embeddings da BD e depois de fazer a média das classificações para cada clip, obteve os resultados apresentados na tabela 5.1.

5.2 Comparação dos resultados

A tabela 5.1 mostra os resultados dos modelos testados com a base de dados de teste do OM18.

Tabela 5.1 Resultados Obtidos para o set de teste do OM18

	mAP	AUC
OpenMic-2018 Baseline	0.79	0.87
Modelo de atenção	**0.81**	**0.88**
NN_FINAL + Mean pooling	0.72	0.82

Uma vez que o PureMic contém áudios simples, apenas com um instrumento, sem ambiguidades, é interessante verificar como é que BD como o OM18, treinada com alguns milhares de exemplares (14.9154), se comporta com este tipo de dados mais "claros". Por essa razão, a tabela 5.1 apresenta os resultados dos mesmos modelos mas aplicados à BD de teste do PureMic.

Tabela 5.2 Resultados Obtidos para o set de teste do PureMic

	mAP	AUC
OpenMic-2018 Baseline	0.75	0.97
Modelo de atenção	0.89	0.99
NN_FINAL + Mean pooling	**0.99**	**0.99**

A tabela 5.1 mostra que o modelo de atenção implementado supera a baseline mas o NN_FINAL + Mean pooling não. Isto deve-se ao facto de o OM conter exemplares muito ruidosos, com muitos eventos em simultâneo ou seja, contém poucos excertos de instrumentos a tocar sozinhos ou com poucas interferências. Por essa mesma razão, a tabela 5.2 mostra que a baseline tem resultados pouco consistentes quando testada em instrumentos puros.

O set de teste do PureMic tem os dados desequilibrados isto porque, havendo 20 classes, cada uma com cerca de 50 exemplares e todos eles etiquetados, uma classe tem apenas 50 dados positivos em contraste com $50 \times 19 = 950$ dados negativos. Com este tipo de bases de dados, a métrica **AUC** não tem tanto significado uma vez que é uma métrica que considera os TN (True Negatives) contemplados na formula do FPR (False Positive Rate) utilizado para a contrução da curva ROC (secção 3.3). Como existem muito mais dados negativos, é normal que o valor desta métrica seja elevado quando na verdade não representa correctamente o desempenho dos classificadores. Desta forma, deve ser mais importante a métrica AP, esta não considera os TN e é mais robusta a dados desequilibrados.

O gráfico da figura 5.1 apresenta os valores da métrica AP para cada instrumento com o set de teste do OM18. O gráfico da figura 5.2 apresenta a mesma métrica para o set de teste do PureMic.

Quando testados com dados puros, o modelo NN_FINAL + Mean pooling tem resultados melhores. Estes resultados são obtidos com base na média das classificações de cada clip portanto, não é por haver alguns frames mal classificados que o resultado final não vai corresponder ao correto. A média atenua este efeito. Claro que o facto de pertencer à mesma base de dados (apesar de não terem contribuído para o treino) pode ajudar a obter resultados melhores tal como também acontece no baseline testado com o seu set de teste.

O objetivo é que se consigam identificar claramente e apenas instrumentos musicais. A ideia que deu origem a este trabalho pretende identificar instrumentos durante um espetáculo. Esses

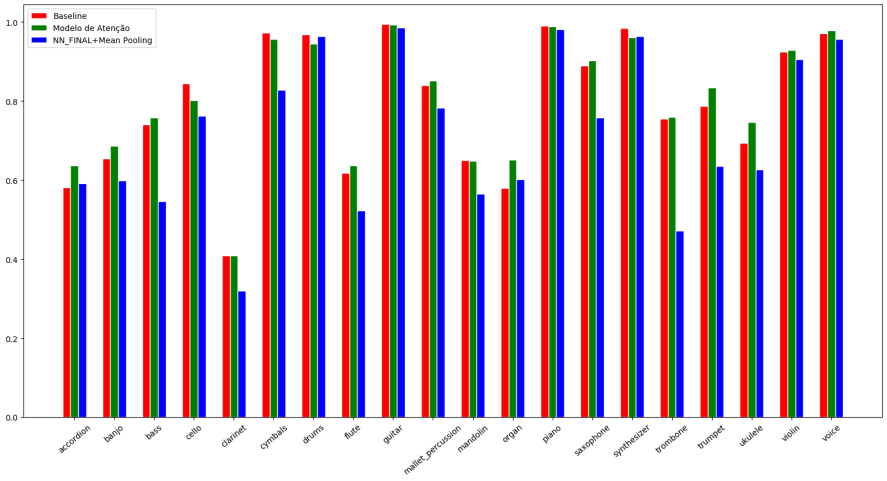

Figura 5.1 Average precision por instrumento para o set de teste do OM

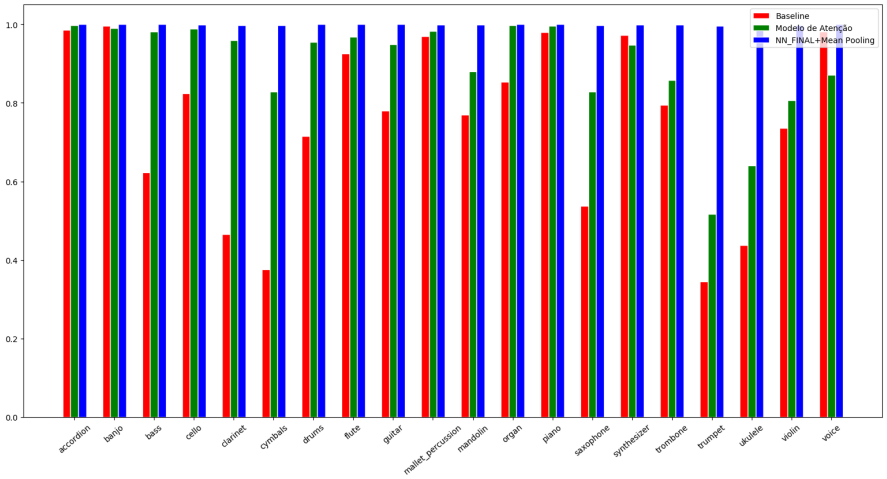

Figura 5.2 Average precision por instrumento para o set de teste do PureMic

instrumentos, por norma, têm o seu sinal acessível de uma forma limpa (ou por microfone ou no caso de instrumentos eletrónicos diretamente ligados) tal como os exemplares escolhidos para o PureMic. Para além disso, na figura 5.1 é possível verificar que o modelo NN_FINAL + Mean pooling chega a superar a baseline em dois casos, acordeão e orgão.

5.3 Etiquetas OpenMic-2018

A base de dados OpenMic-2018, tem um total de 400.000 etiquetas (20×20.000). Das 400.000 etiquetas apenas 41.265 estão etiquetadas, 17.611 positivas e 23.654 negativas.

Ao passar todos os exemplares pelo modelo NN_FINAL + Mean Pooling, obtemos as classificações atribuídas a todos os 20.000 exemplares. Dependendo do grau de certeza que se pretende, pode-se

contribuir com mais ou menos etiquetas. A tabela 5.3 mostra a relação entre a percentagem escolhida e número de etiquetas que se consegue melhorar. A coluna esquerda mostra as etiquetas negativas e a coluna direita as positivas.

Tabela 5.3 Relação entre a percentagem de certeza e o número de etiquetas

percentagem	nr^o de exemplares negativos	percentagem	nr^o de exemplares positivos
1×10^{-10}	28.617	0.99	1275
1×10^{-9}	30874	0.95	2078
1×10^{-8}	33519	0.9	2768
1×10^{-7}	37017	0.8	3939
1×10^{-6}	41213	0.65	6298
1×10^{-5}	46995	0.5	9529

No caso em que se pretende mais rigor (primeira linha da tabela) é possível acrescentar às 41.265 etiquetas, mais 29.892 (cerca de 70% das etiquetas que existiam). No máximo (para os limiares considerados na tabela) é possível contribuir com 56.524 etiquetas novas aumentando assim o número de dados etiquetados para mais de o dobro.

Capítulo 6

Conclusão e trabalho futuro

Neste trabalho foram desenvolvidos 3 classificadores de instrumentos musicais. O número de instrumentos considerado foi de 20 de acordo com a base de dados OpenMic-2018 mais a classe de silêncio que em muitos casos é útil que se classifique. O modelo **NN_MEAN** foi utilizado para melhorar ainda mais as etiquetas da base de dados PureMic, uma das contribuições deste trabalho. Com isto, a base de dados que já tinha etiquetas fortes tornou-se ainda mais "pura"através da remoção de momentos de silêncio. O resultado foi a BD PM1 que foi utilizada para treinar o modelo **NN_ALL**, testado com dados puros com uma resolução temporal de 100ms e obteve uma *accuracy* de 95.99% (matriz A.2). O modelo final **NN_FINAL** classifica clips de áudio também com uma resolução temporal de 100ms e foi usado para contribuir para o aumento das etiquetas da base de dados OpenMic-2018 sendo que é possível aumentar o número de etiquetas para mais de o dobro (de 41.265 para 97.789).

Para além de todo o conhecimento ganho em vários campos como processamento de sinal, machine learning e python e na utilização de várias ferramentas como o TensorFlow, TensorBoard, latex entre outros, é possível agora, através do modelo NN_FINAL, classificar instrumentos musicais a uma taxa de cerca de 1 segundo aplicando este modelo diretamente ao som captado por um microfone.

Como trabalho futuro, existem técnicas para melhorar o trabalho já realizado como a técnica de data augmentation que procura aumentar os dados que já existem através da sua manipulação. Também existem outras metodologias que podem melhorar a performance dos modelos como o treino da rede VGGish que neste trabalho foi usada apenas para extrair *features*. Acima de tudo o objetivo deverá passar por continuar o projeto que motivou a realização desta dissertação.

Bibliografia

[1] R. Bro. Analytical Methods Principal Component Analysis. *Royal Society Of Chemistry*, pages 2812–2831, 2014.

[2] C. Chung, S. Patel, R. Lee, L. Fu, S. Reilly, T. Ho, J. Lionetti, M. D. George, and P. Taylor. Implementation of an integrated computerized prescriber order-entry system for chemotherapy in a multisite safety-net health system. *American Journal of Health-System Pharmacy*, 75(6):398–406, 2018.

[3] Dafx. Digital Audio Effects 2019 (DAFx 2019).

[4] J. D. Deng, C. Simmermacher, and S. Cranefield. A study on feature analysis for musical instrument classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(2):429–438, 2008.

[5] A. Diment, P. Rajan, T. Heittola, and T. Virtanen. Modified Group Delay Feature for Musical Instrument Recognition. Technical report.

[6] A. Eronen. Comparison of features for musical instrument recognition. In *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, pages 19–22. IEEE.

[7] S. Essid, G. Richard, and B. David. Musical instrument recognition on solo performances. *undefined*, 2004.

[8] S. Essid, G. Richard, and B. David. Musical Instrument Recognition by pairwise classification strategies. Technical report, 2006.

[9] FMA. Free Music Archive.

[10] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter. Audio Set: An ontology and human-labeled dataset for audio events. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 776–780, 2017.

[11] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka. RWC Music Database: Popular, Classical, and Jazz Music Databases. Technical report.

[12] S. Gururani, M. Sharma, and A. Lerch. An Attention Mechanism for Musical Instrument Recognition. 2(Miml), 2019.

[13] Y. Han, J. Kim, and K. Lee. Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 25(1):208–221, jan 2017.

[14] Y. Han, S. Lee, J. Nam, and K. Lee. Sparse feature learning for instrument identification: Effects of sampling and pooling methods. 2290.

[15] R. M. Hegde, H. A. Murthy, and G. V. Rao. The modified group delay feature: A new spectral representation of speech. *8th International Conference on Spoken Language Processing, ICSLP 2004*, (May 2014):913–916, 2004.

[16] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. Wilson. CNN architectures for large-scale audio classification. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 131–135, 2017.

[17] E. J. Humphrey, S. Durand, and B. Mcfee. OpenMIC-2018: AN OPEN DATASET FOR MULTIPLE INSTRUMENT RECOGNITION. 2018.

[18] Icassp. Home - ICASSP 2020.

[19] Irmas. IRMAS: a dataset for instrument recognition in musical audio signals - MTG - Music Technology Group (UPF).

[20] Ismir. ISMIR - Home | ISMIR.

[21] Q. Kong, C. Yu, Y. Xu, T. Iqbal, W. Wang, and M. D. Plumbley. Weakly Labelled AudioSet Tagging with Attention Neural Networks. (Mil):1–12.

[22] A. Krishna and T. Sreenivas. Music instrument recognition: from isolated notes to solo phrases. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages iv–265–iv–268. IEEE.

[23] T. Lee, Taejin. Park. MUSICAL INSTRUMENT SOUND CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORK USING FEATURE FUSION APPROACH Taejin Park and Taejin Lee Electronics and Telecommunications Research Institute (ETRI), Republic of Korea. *Electronics and Telecommunications Research Institute (ETRI), Republic of Korea*.

[24] P. Li, J. Qian, and T. Wang. Automatic Instrument Recognition in Polyphonic Music Using Convolutional Neural Networks. 2015.

[25] K. D. Martin. Sound-source recognition : a theory and computational model. 1999.

[26] Mcadams. Stephen McAdams: Recognition of Auditory Sound Sources and Events (Thinking in Sound: The Cognitive Psychology of Human Audition, Oxford University Press, Oxford 1993), 1993.

[27] S. Mcadams. Recognition of sound sources and events. In *Thinking in SoundThe Cognitive Psychology of Human Audition*, pages 146–198. Oxford University Press, apr 1993.

[28] P. D. I. Milano. MUSICAL INSTRUMENTS RECOGNITION : A TRANSFER LEARNING APPROACH. 2018.

[29] M. Müller. *Fundamentals of Music Processing*. 2015.

[30] S. Ruder. An overview of gradient descent optimization algorithms *. Technical report.

[31] G. Yu and J.-J. Slotine. Audio classification from time-frequency texture. pages 1677–1680, 2009.

Apêndice A

Matrizes de Confusão

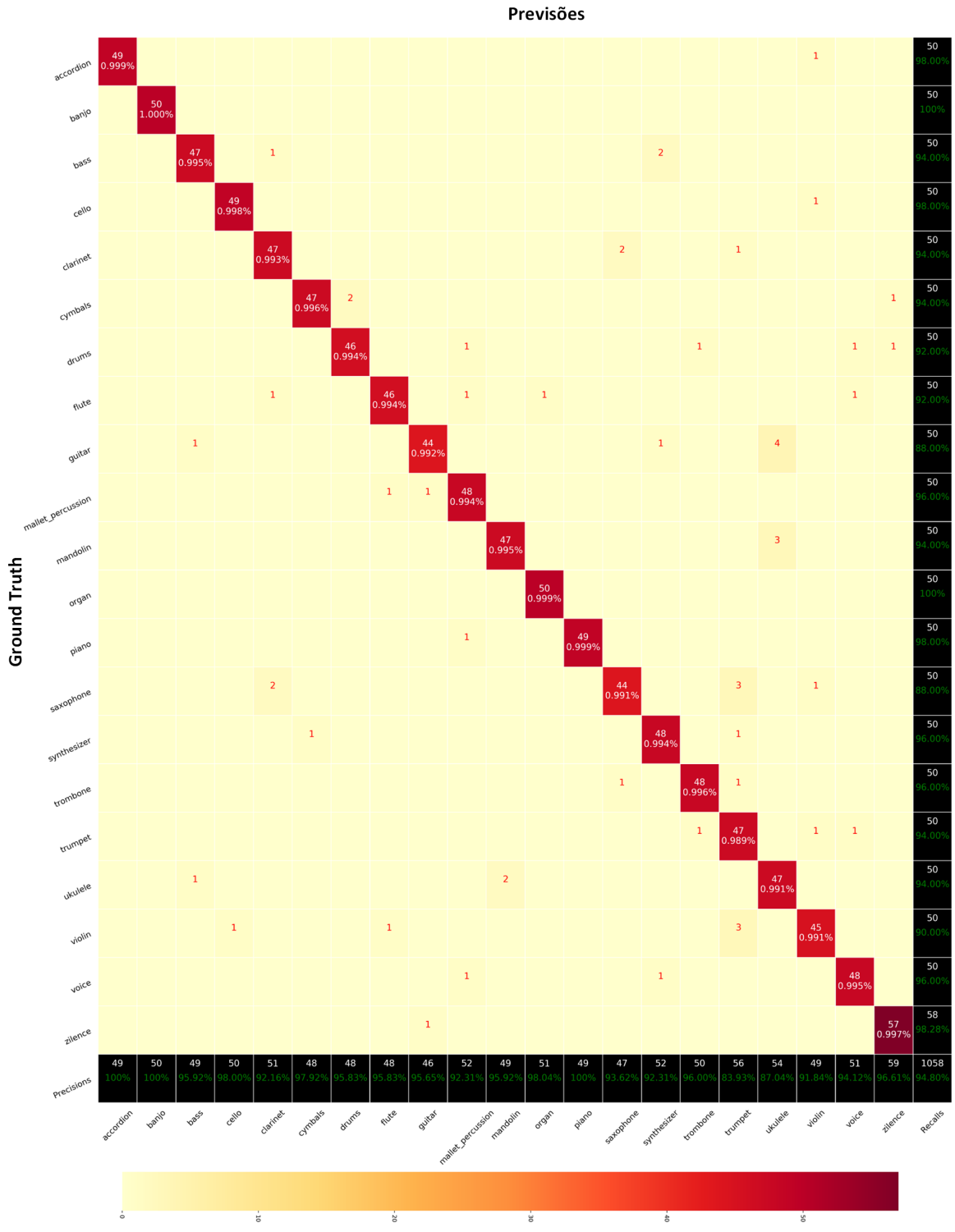

Figura A.1 Matriz de Confusão NN_MEAN

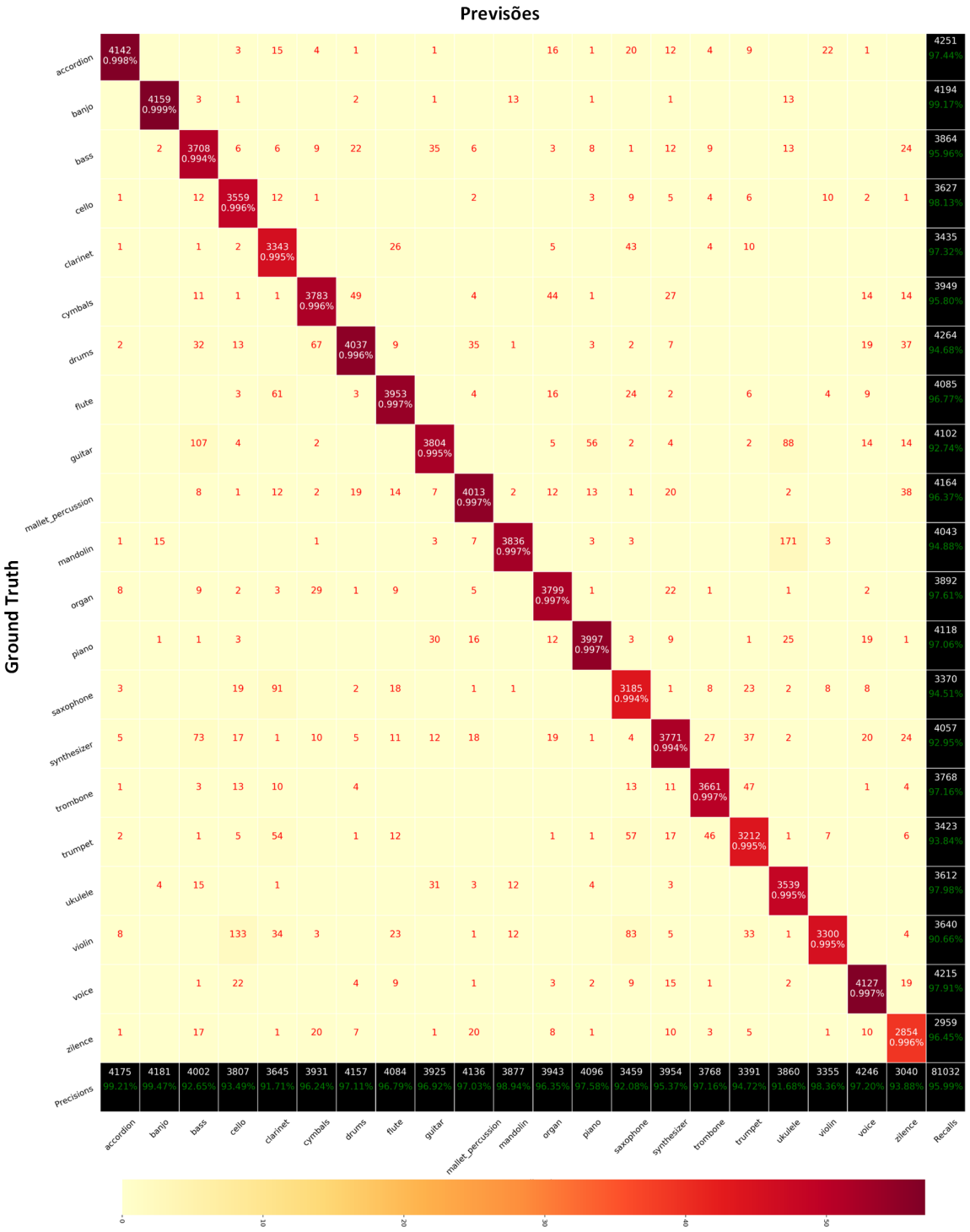

Figura A.2 Matriz de confusão NN_ALL