Audio Classification - UrbanSound8k

Neste relatório, descreveremos a abordagem adotada para classificar dados sonoros urbanos usando técnicas de Deep Learning.

O problema de classificação consiste em categorizar dados sonoros urbanos em uma das dez classes pré-definidas, sendo essas dez classes: ar condicionado, buzina de um carro, crianças a brincar, ladrar de um cão, perfuração, motor a trabalhar, tiro de uma arma, martelo pneumático, sirene e música de rua.

A primeira abordagem passou por treinar uma CNN de duas dimensões. Achamos que esta seria a melhor solução já que a os mfccs obtidos com a ajuda da biblioteca Librosa têm uma shape que se assemelha ao de uma imagem e para problemas com imagens é comum usar-se CNNs com duas dimensões. A segunda abordagem foi uma RNN, mais precisamente um LSTM unidirecional. Optamos por esta solução porque, apesar de usarmos como input os mfccs e não o som em si, as LSTMs são muito usadas para fazer speech recognition, que é um problema que pensamos que poderia ser semelhante ao que nos foi proposto.

No que toca ao pre-processing começamos por verificar que apenas duas (gun_shot e car_horn) das dez classes não têm cerca de 11% de representatividade no dataset, tendo 4,3% e 4,9% respetivamente, como podemos ver na figura 1. Devido à diferença destes valores não ser muito elevada decidimos não usar nenhuma técnica para tratar de datasets desbalanceados, como resampling ou loss functions com pesos diferentes para cada classe. De seguida notamos que, apesar de grande parte dos sons terem quatro segundos de duração, havia ainda um número razoável dos sons que não tinham essa duração, como podemos ver na figura 2. Para resolver este problema resolvemos repetir o som até que ficasse com os 4 segundos. Isto pode ser bom pois dá mais oportunidades ao modelo de tirar ilações sobre aquele som visto que ele se repete várias vezes nos quatro segundos, mas também pode levar o modelo a tirar conclusões erradas, como por exemplo, como há muitos sons da classe gun_shot com menos que 4 segundos, o modelo pode considerar que deve classificar um som como gun_shot se a certa altura ele se começar a repetir.

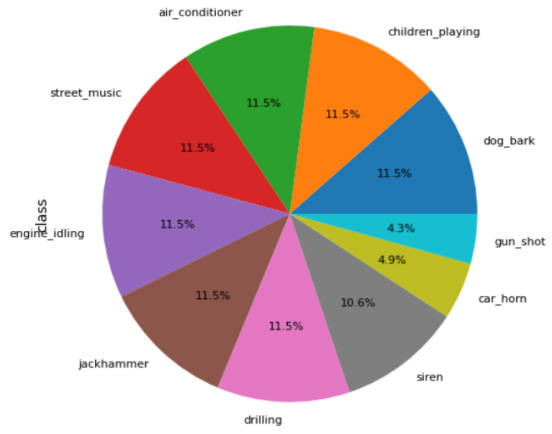

Figura 1

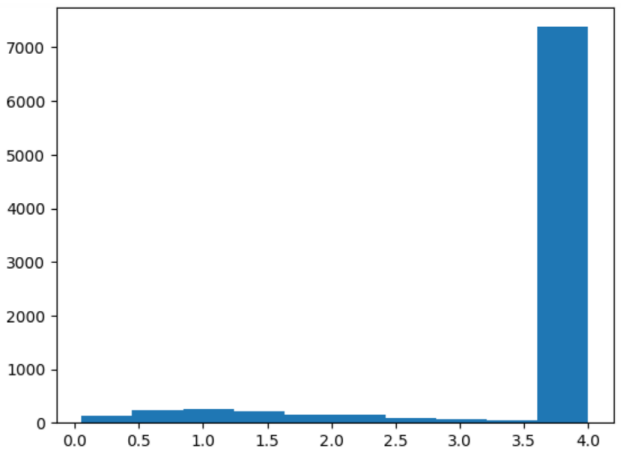

Figura 2

Analisamos também uma onda de um exemplo de cada classe para termos uma ideia de que classes teriam ondas parecidas e poderiam ser confundidas pelo modelo.

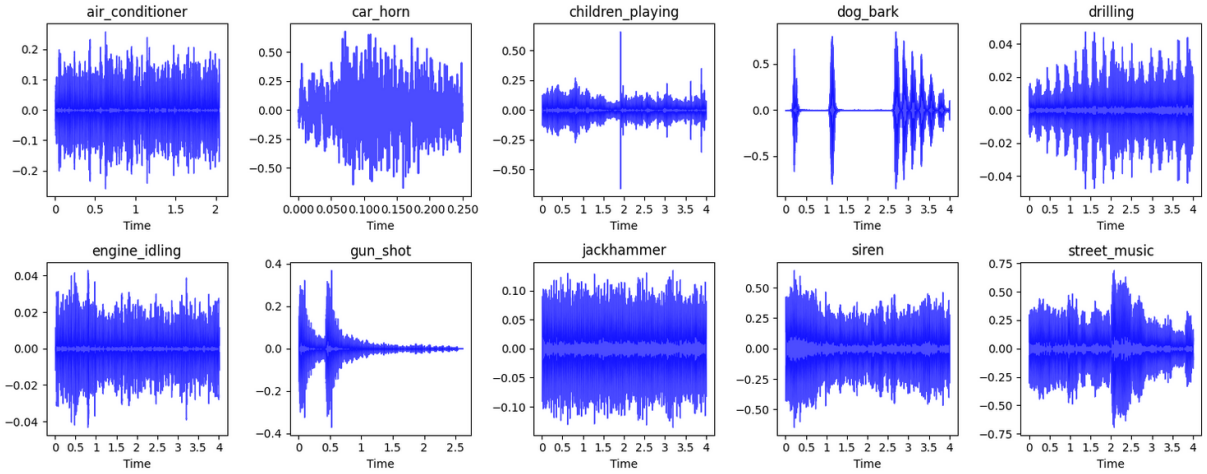

Figura 3

Antes de passarmos à obtenção dos mfccs adicionamos um ruído com um desvio padrão de 0.008 a todos os sons. Após vários testes concluímos que este valor para o desvio padrão alterava o som de maneira suficiente sem que o tornasse imperceptível, como se vê nas figuras 4 e 5. As figuras mostram o som antes e depois de ser acrescentado o ruído.

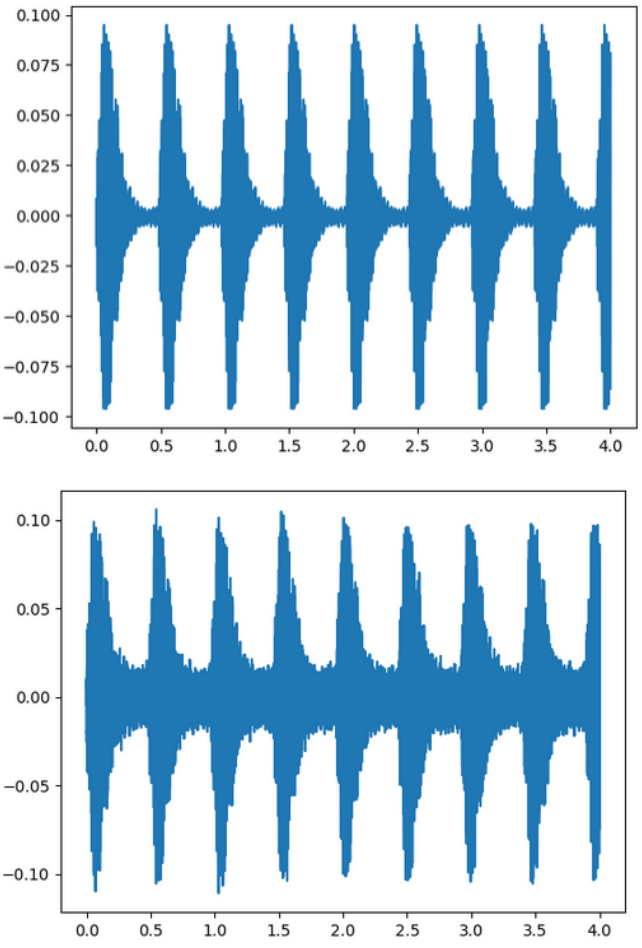

Figura 4

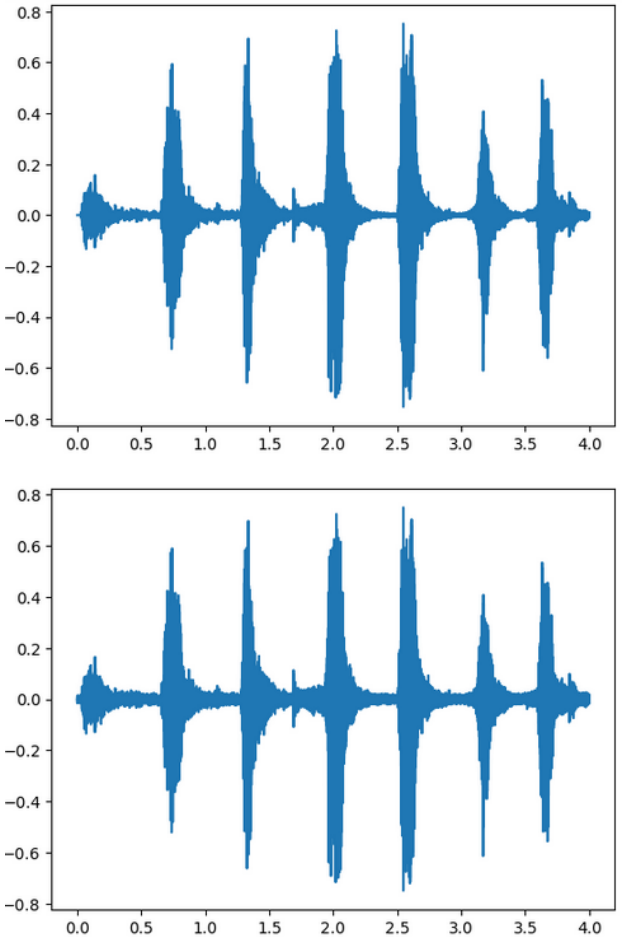

Figura 5

No que toca à extração dos mfccs começamos por alterar a frequência de todos os sons para 44100 Hz. Depois obtivemos os mfccs com a ajuda da biblioteca Librosa e guardamos um array com os mfccs e outro com as classes nas respectivas pastas.

Já no treino dos classificadores, começamos por definir que iríamos usar a sparse_categorical_crossentropy como a função de loss e a sparse_categorical_accuracy como métrica como métrica de avaliação. Para avaliar a performance dos classificadores usamos o sistema de cross validation sugerido no website do dataset UrbanSound.

Quanto ao otimizador utilizamos o adam para tambèm para ambos os modelos, apesar de termos explorado a possibilidade de o stochastic gradient descent para a LSTM tentando descobrir a learning rate ótima para treinar o modelo. Como pode ser visto na figura 6, a alteração da learning rate (eixo x) não alterou a loss (eixo y) o que nos fez usar o otimizador adam.

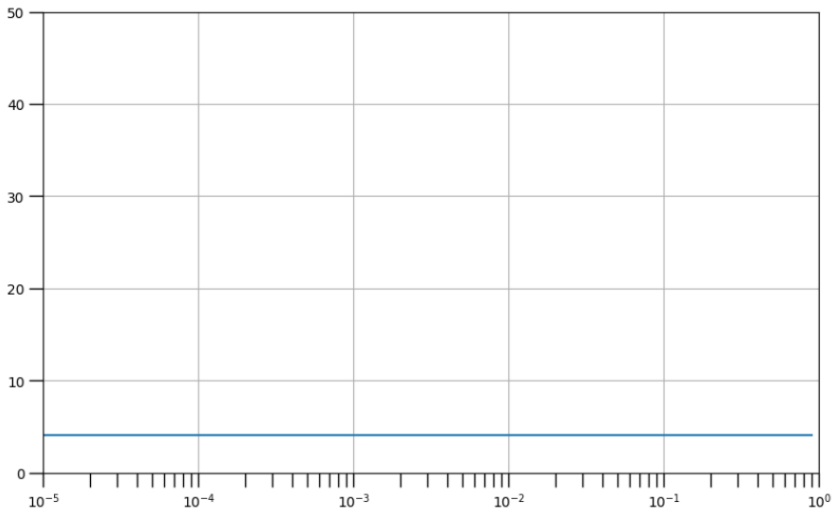

Figura 6

Optamos pelo uso da função EarlyStoping para monitorizar a accuracy do teste para tornar o treino mais rápido. Esta função para o treino se houver 10 epochs em que a accuracy do teste não aumentou pelo menos 0.001.

Foram treinados modelos com os sons que estavam no foreground, no background e um modelo com todos os sons. O número de epochs foi 50, embora os treinos tenham acabado antes devido ao early stopping, e o tamanho das mini batches foi 64.

Não foi usado nenhum tipo de data augmentation por considerarmos não ser produtivo usar os mesmos métodos que são usados nas imagens, como rodar a imagem ou fazer zoo visto que uma onda sonora tem sempre o mesmo aspeto e está sempre na mesma posição.

Ambos os modelos têm layers de Dropout para tentar minimizar o overfitting e de Batch Normalization para agilizar o tempo de treino e melhorar a sua estabilidade.

No caso da CNN usamos como input os vetores com os mfccs com um shape de (40, 321) que tínhamos obtido previamente.

Obtivemos os seguintes resultados para cada treino.

	2D-CNN	2D-CNN_FG	2D-CNN_BG
0	63.573885	73.794210	50.597608
1	64.189190	77.956986	43.333334
2	54.378378	60.508472	51.044774
3	63.737375	67.741936	54.861110
4	74.252135	77.677226	61.172163
5	62.940460	74.174756	54.220778
6	69.928402	81.558937	58.333331
7	67.617863	75.543481	54.724407
8	68.014705	83.818179	41.353384
9	71.087217	85.387325	48.698884

Figura 7

Como pode ser visto na figura 7, onde cada linha corresponde a uma fold de teste, os melhores resultados foram no modelo que foi treinado com os sons no foreground e os piores no modelo treinado com os sons no background. Isto é expectável pois, tal como o ser humano, o modelo dá mais ênfase aos sons que estão perto de si, neste caso mais perto do microfone que gravou o áudio.

Pegando no melhor resultado para cada dataset, podemos verificar com a ajuda dos gráficos das figuras 8, 9 e 10 que, apesar dos nossos esforços, acabou por haver um pouco de overfitting.

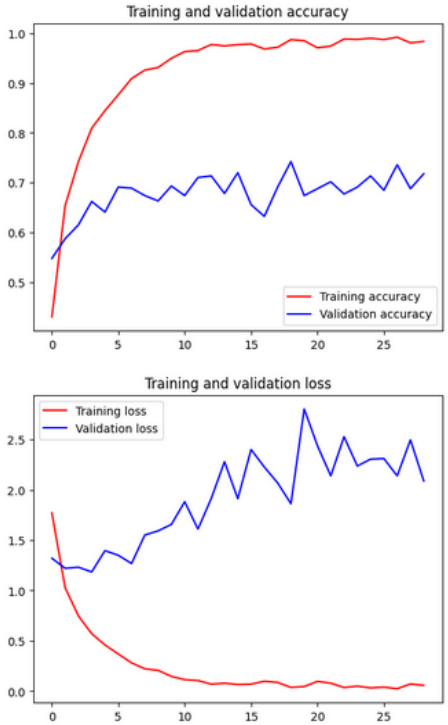

Figura 8 - All Sounds

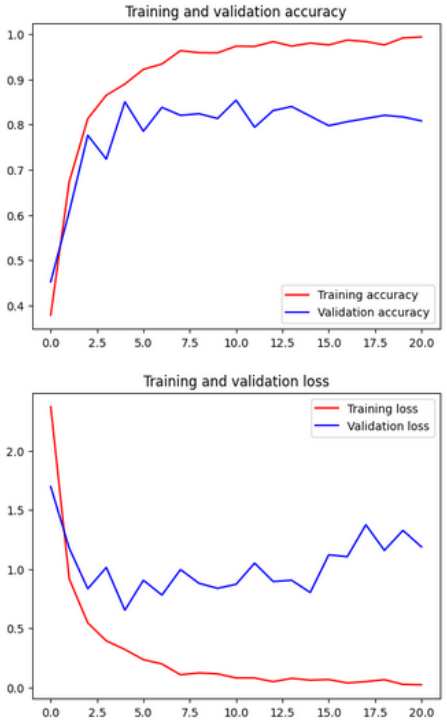

Figura 9- Foreground

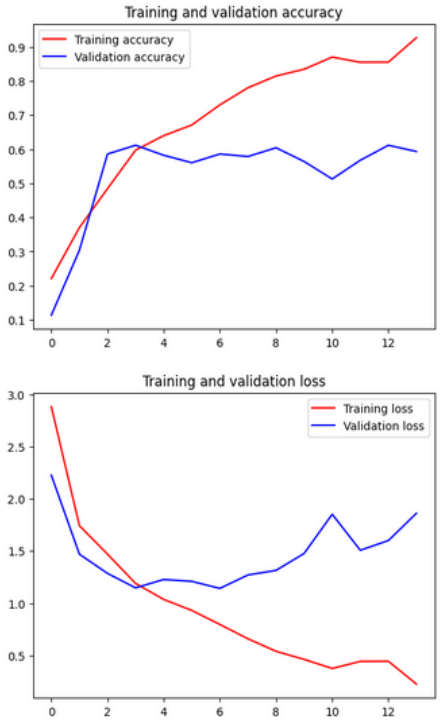

Figura 10 - Background

Analisando as matrizes confusão dos resultados dos treinos podemos verificar que treinar o modelo com todos os sons teve uma accuracy de 65.97% e um desvio padrão de 5.24%, treinar com os sons no foreground teve uma accuracy de 75.82% e um desvio padrão de 7.09% e treinar com os sons no background teve uma accuracy de 51.82% e um desvio padrão de 5.89%.

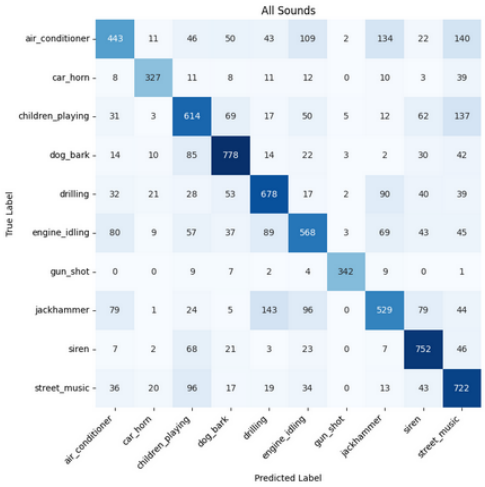

Figura 11

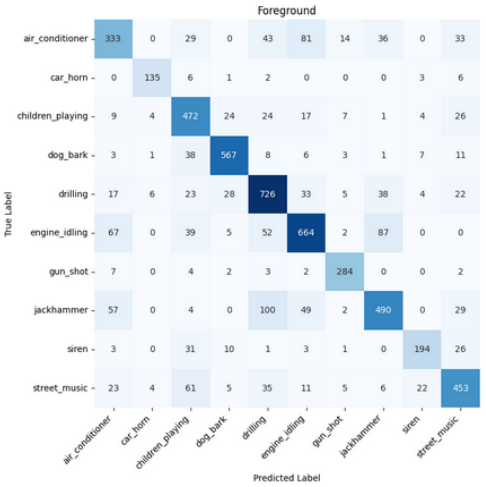

Figura 12

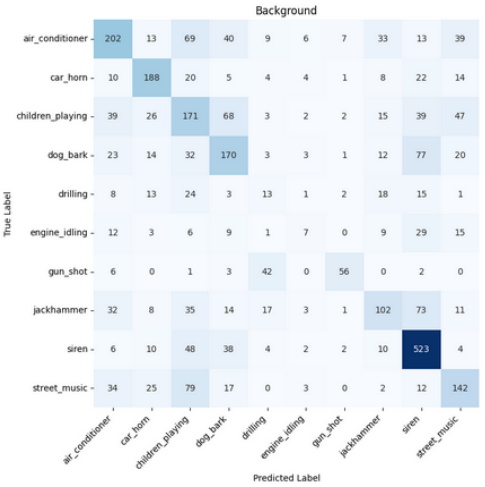

Figura 13

Comparando a matriz confusão do treino com todos os sons com a soma das matrizes de confusão dos outros dois treinos podemos ver que o primeiro caso acertou 5753 vezes enquanto que o segundo acertou 5892 vezes.

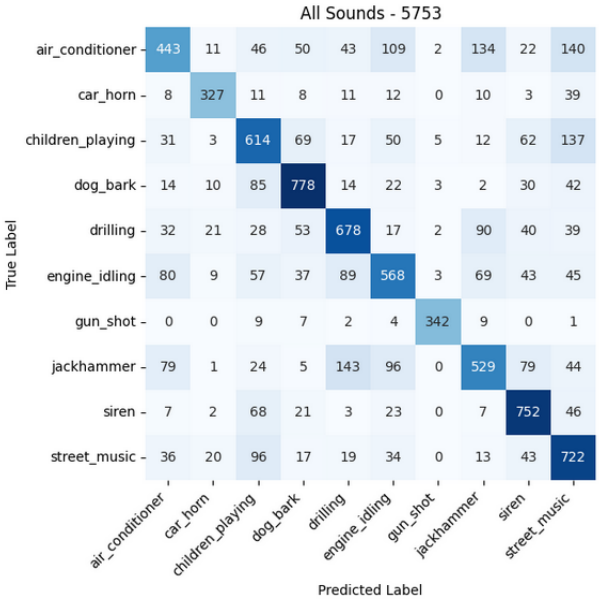

Figura 14

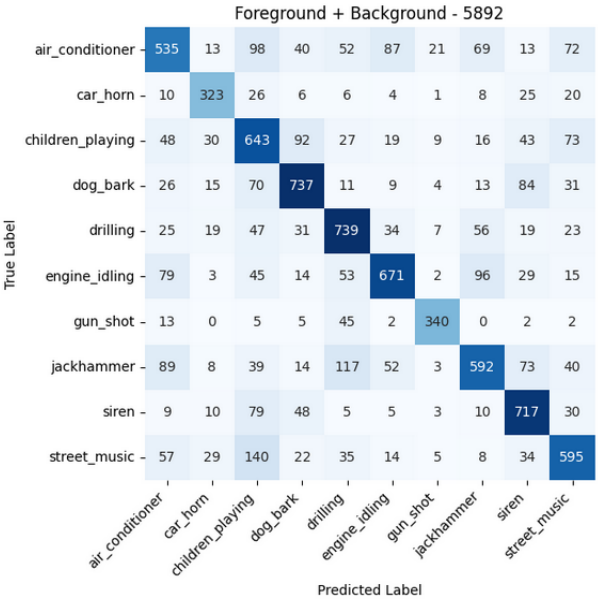

Figura 15

Já na LSTM usamos como input um vetor com um shape de (40) em que cada ponto é a média dos valores ao longo do segundo eixo do vetor dos mfccs. Apesar de poder parecer pouco intuitivo, após alguns testes este foi o formato que nos deu os resultados menos maus (Figura 13).

	LSTM	LSTM_FG	LSTM_BG
0	48.797250	56.591642	35.458168
1	41.328830	46.594983	33.030304
2	41.621622	42.881355	43.582091
3	40.808082	40.143371	41.203704
4	51.923078	54.147816	45.054945
5	39.854193	53.398061	31.493506
6	53.937948	54.182512	41.987181
7	45.657569	50.905800	34.251967
8	54.534316	67.090911	32.330826
9	42.771804	54.929578	29.739776

Figura 16

Analisando os gráficos equivalentes aos do treino com a CNN, vemos que houve mais overfitting do que o treino anterior.

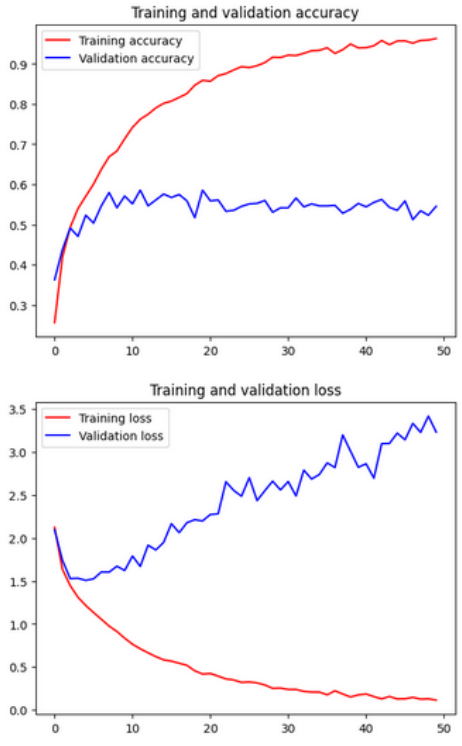

Figura 17 - All Sounds

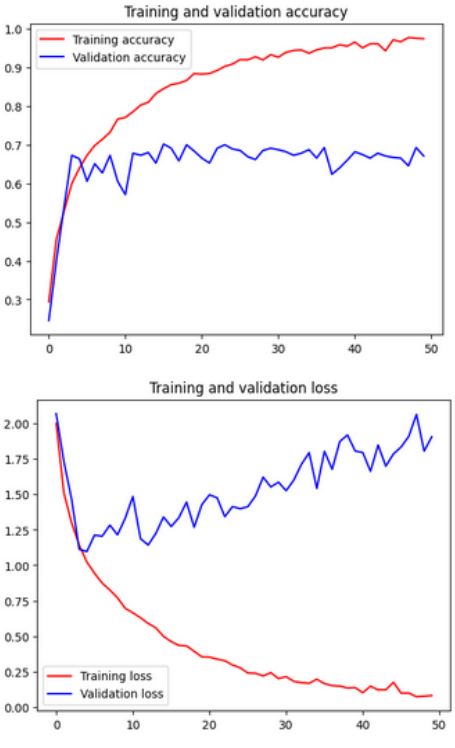

Figura 18 - Foreground

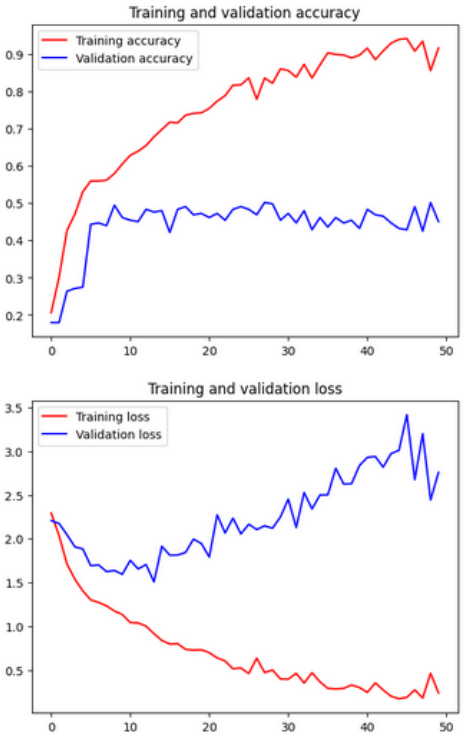

Figura 19 - Background

Analisando as matrizes confusão dos resultados dos treinos podemos verificar que treinar o modelo com todos os sons teve uma accuracy de 46.12% e um desvio padrão de 5.43%, treinar com os sons no foreground teve uma accuracy de 52.09% e um desvio padrão de 7.23% e treinar com os sons no background teve uma accuracy de 36.81% e um desvio padrão de 5.3%.

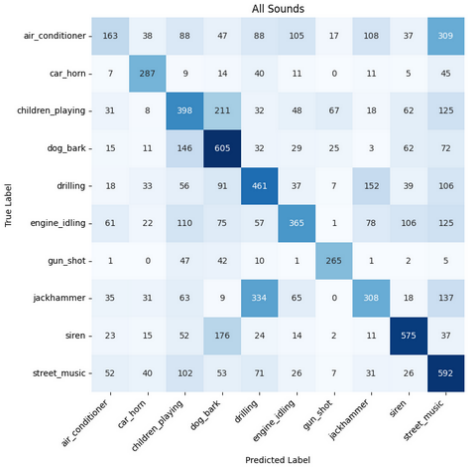

Figura 20

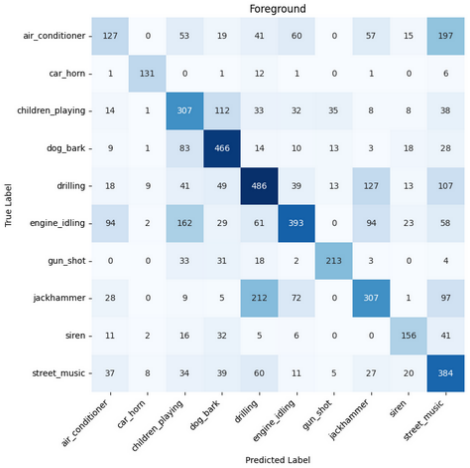

Figura 21

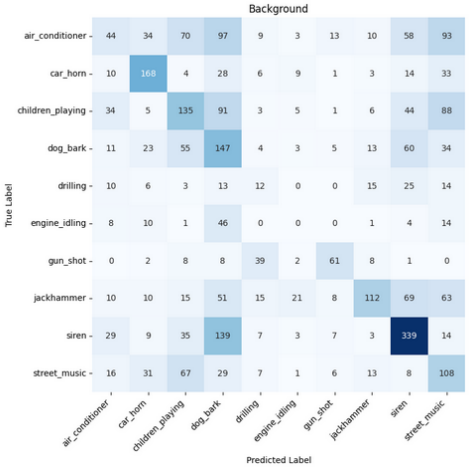

Figura 22

Comparando a matriz confusão do treino com todos os sons com a soma das matrizes de confusão dos outros dois treinos podemos ver que o primeiro caso acertou 4019 vezes enquanto que o segundo acertou 4096 vezes.

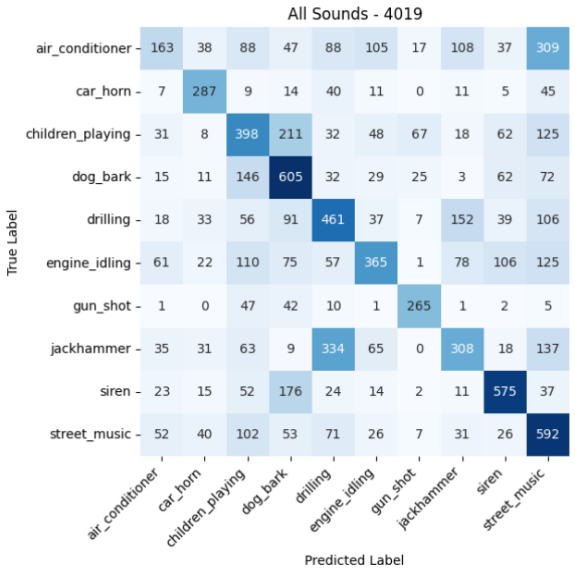

Figura 23

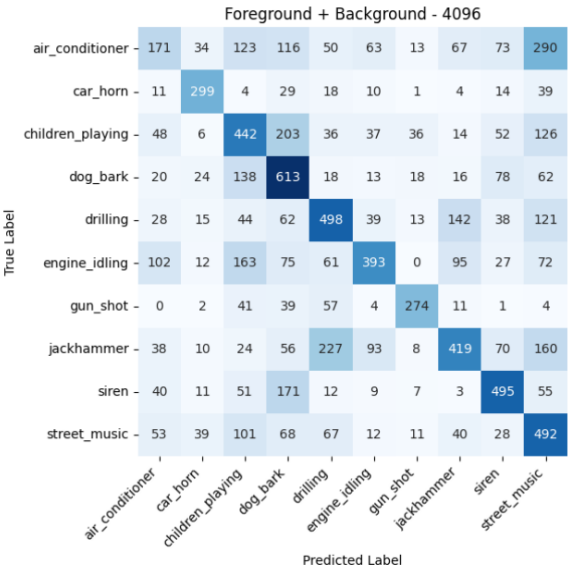

Figura 24

Tendo em conta os nossos testes, concluímos que uma CNN resolve com alguma qualidade este problema enquanto uma LSTM pode ter mais dificuldades. Isto pode ter acontecido também devido a termos usado uma média dos valores dos mfccs e termos perdido informação no processo. Outra explicação pode ser a rede LSTM que usamos não ser a mais adequada, precisando por exemplo de mais camadas e mais unidades dentro de cada camada. Infelizmente, não temos ao nosso dispor poder computacional suficiente para executarmos testes com redes neuronais de maior calibre que exijam mais tempo de execução.

Concluímos também que, havendo a possibilidade, é mais proveitoso treinar um modelo para classificar os sons que estão no foreground e outro para os sons que estão no background pois obtivemos mais acertos neste caso do que no modelo treinado com todos os sons. Outra vantagem que teria, que não foi explorada, seria usar técnicas de pre-processing diferentes para os dois tipos de sons e até mesmo modelos diferentes já que pode haver soluções melhores para um caso que para outro.

Todas as figuras podem ser vistas nos notebooks com mais detalhe.