Universidade Federal do Rio de Janeiro

Classificação de Instrumentos Musicais Usando Redes LSTM

Alunos
Cainã Figueiredo Pereira
Felipe Martins Fernandes de Assis
Felipe Schreiber Fernandes
Raul Baptista de Souza

Projeto da disciplina de Processamento de Voz
2020/PLE

Professor
Fernando Gil Vianna Resende Junior

Rio de Janeiro, 12 de novembro de 2020

1 - INTRODUÇÃO:

No campo de Music Information Retrieval (MIR), o reconhecimento dos instrumentos presentes em um áudios musicais tem sido intensamente pesquisado. Uma motivação para essa busca constante por métodos mais eficientes de reconhecimento de instrumentos musicais é que, a cada dia, um número crescente de músicas têm se tornado disponíveis e, por consequência, torna-se cada vez mais necessário haver mecanismos de busca que permitam filtrar as músicas por alguma informação de alto nível, como gênero ou instrumentos presentes, por exemplo. Além disso, quando o assunto é recomendação de músicas, a informação dos instrumentos presentes pode ser de grande valor na hora de determinar quais músicas são do gosto de um usuário.

Neste documento, estaremos relatando os passos pelos quais tivemos de passar a fim de obter a classificação do instrumento predominante em áudios musicais de um conjunto de dados denominado IRMAS utilizando o modelo de redes neurais recorrentes LSTM (Long Short-Term Memory) [4].

Este relatório está organizado da seguinte maneira. Nós começamos descrevendo, na seção 2, as tecnologias que optamos utilizar neste projeto. Na seção 3, nós descrevemos algumas características do conjunto de dados usado. Em seguida, na seção 4, nós apresentamos algumas propostas de features a serem extraídas a partir dos áudios do conjunto de dados IRMAS para posterior utilização pelo modelo de aprendizado. Com a base construída a partir dessas features, partimos para a seção 5, onde veremos a metodologia e os resultados de nossos experimentos. Finalizamos este relatório com uma discussão na seção 6, onde fazemos as devidas considerações sobre os resultados e, eventualmente, algumas comparações com outros métodos e apresentamos uma breve conclusão na seção 7.

2 - TECNOLOGIAS UTILIZADAS:

Para este projeto, optamos por utilizar principalmente bibliotecas do Python. A primeira biblioteca se chama Librosa, a qual foi usada para extração de features e eventuais visualizações do áudio. A biblioteca Keras possui, dentre outras coisas, uma implementação de redes LSTM. Devido a isso, essa foi integrada ao nosso projeto. Outras bibliotecas foram utilizadas, mas se tratam de bibliotecas bastante conhecidas, tais como Pandas, Scikit-Learn e Seaborn, e por isso não consideramos que seja interessante descrever cada uma. Também usamos scripts do website *Audio Content Analysis [3]*.

Optamos por utilizar o Google Colaboratory, uma espécie de Jupyter onde os programas são escritos em forma de notebooks e diversos usuários podem interagir simultaneamente. Além de possibilitar a coordenação entre os componentes do grupo, essa ferramenta ainda disponibiliza recursos, tais como GPU.

3 - DESCRIÇÃO DO DATASET:

Para este projeto, especificamente, nós optamos por usar um conjunto de dados denominado “Instrument Recognition in Musical Audio Signals”, ou simplesmente IRMAS. Esse conjunto contém diversos trechos musicais com anotações sobre o instrumento predominante. Esses trechos consistem de arquivos .wav com duração de 3 segundos de 11 instrumentos, tais como clarinete, piano, violino e flauta, por exemplo. Além disso, todos esses áudios foram obtidos a partir de músicas de diversos gêneros com qualidade profissional, o que permite que modelos robustos sejam produzidos. Nós optamos por reduzir o problema ao reconhecimento de apenas dois instrumentos, sendo eles o piano e violino. Logo, nem todo o conjunto IRMAS foi utilizado, mas apenas aqueles áudios musicais com anotações de instrumento predominante sendo um desses dois instrumentos mencionados anteriormente, totalizando 1301 faixas de áudio, sendo 721 exemplos de piano e 580 de violino.

4 - EXTRAÇÃO DE FEATURES:

Para esta etapa, consideramos a extração de algumas features para cada um dos áudios musicais, dentre as quais algumas podem ser obtidas diretamente a partir do espectro de frequências do áudio ou mesmo a partir do próprio áudio no domínio do tempo. Outras, por sua vez, não são tão diretas, como é o caso dos coeficientes MFCC. As seguintes features foram extraídas com o auxílio da biblioteca Librosa para dois tamanhos de janela temporal diferentes, sendo 23ms e 93ms, com uma frequência de amostragem de 22050Hz:

- **Spectral Centroid:** Essa métrica representa o “centro de massa” de uma distribuição de energia espectral. Ela é calculada como a média ponderada das frequências presentes no sinal com os pesos sendo as suas respectivas magnitudes (ver Equação 1). Essa métrica pode ser determinada a partir do espectro obtido após a aplicação da transformada de Fourier no áudio.

$$Centroid, \mu = \frac{\sum_{i=1}^{N} f_i . m_i}{\sum_{i=1}^{N} m_i} \quad \text{(Equação 1)}$$

onde $\mathbf{m_i}$ representa a magnitude do i-ésimo intervalo (bin) de frequências e $\mathbf{f_i}$ representa a frequência que ocupa o centro desse intervalo.

- **Spectral Bandwidth:** Essa métrica é útil para capturar informações sobre a largura do espectro. Ela é obtida conforme mostrado na equação 2.

$$(\sum_{i} m_i (f_i - \mu)^p) \frac{1}{p} \quad \text{(Equação 2)}$$

onde **mi** representa a magnitude espectral do i-ésimo intervalo (bin) de frequências, **fi** representa a frequência que ocupa o centro desse intervalo, μ é o centróide e p > 0 é a força com que o desvio em relação ao centróide será contabilizado.

- **Spectral Rolloff:** Nessa métrica, queremos determinar a frequência do espectro tal que uma certa fração da energia esteja contida nas frequências abaixo dela. Matematicamente, esta frequência pode ser determinada da seguinte maneira:

$$\underset{f_c \in \{1,\ldots,N\}}{argmin} \sum_{i=1}^{f_c} m_i \geq c \sum_{i=1}^{N} m_i \qquad \text{(Equação 3)}$$

onde **mi** representa a magnitude espectral do i-ésimo intervalo (bin) de frequências, **fc** representa a frequência de "rolloff", **N** é o número total de intervalos de frequências e **c** é um valor entre 0 e 1 representando a fração da energia total do espectro que deve estar abaixo da frequência de "rolloff".

- **Zero-Crossing Rate:** Indica o número de vezes que um sinal cruza o eixo horizontal. Essa métrica é obtida a partir do sinal no domínio do tempo e a equação 4 mostra uma das diversas formas de se obter essa métrica.

$$zcr = \frac{1}{T-1} \sum_{t=1}^{T-1} \mathbb{I}\{s_t s_{t-1} < 0\} \qquad \text{(Equação 4)}$$

onde **st** é um sinal de tamanho **T** e
II é a função indicadora (1 se X = True, 0 caso contrário)

- **RMS Entropy (Root Mean Square Energy):**
  A cada frame calcula-se a raiz da média quadrática da amplitude do sinal no tempo. Isso representa a potência média do sinal naquele frame.

$$RMSE = \sqrt{\frac{1}{N} \sum_{n} |x(n)|^2} \qquad \text{(Equação 5)}$$

- **MFCC (Mel-Frequency Cepstral Coefficients):**

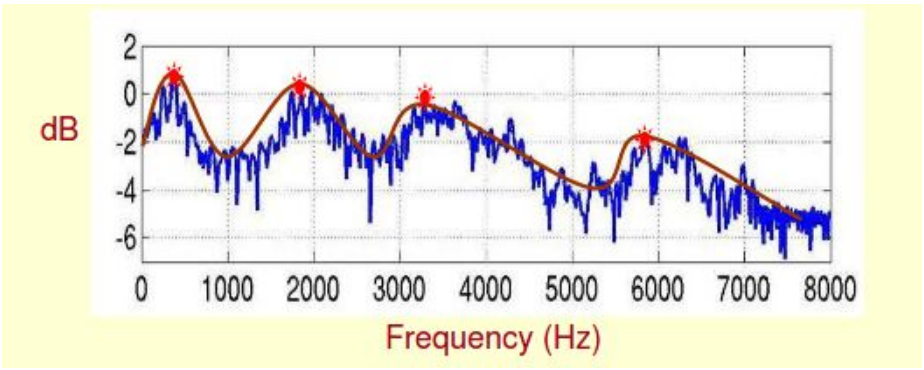


Figura 1

A figura 1 mostra o espectro de um sinal**.** O que nós queremos com o MFCC é capturar a envoltória em marrom conforme mostrado acima. Para tal, separaremos o espectro em duas componentes, conforme a figura 2:

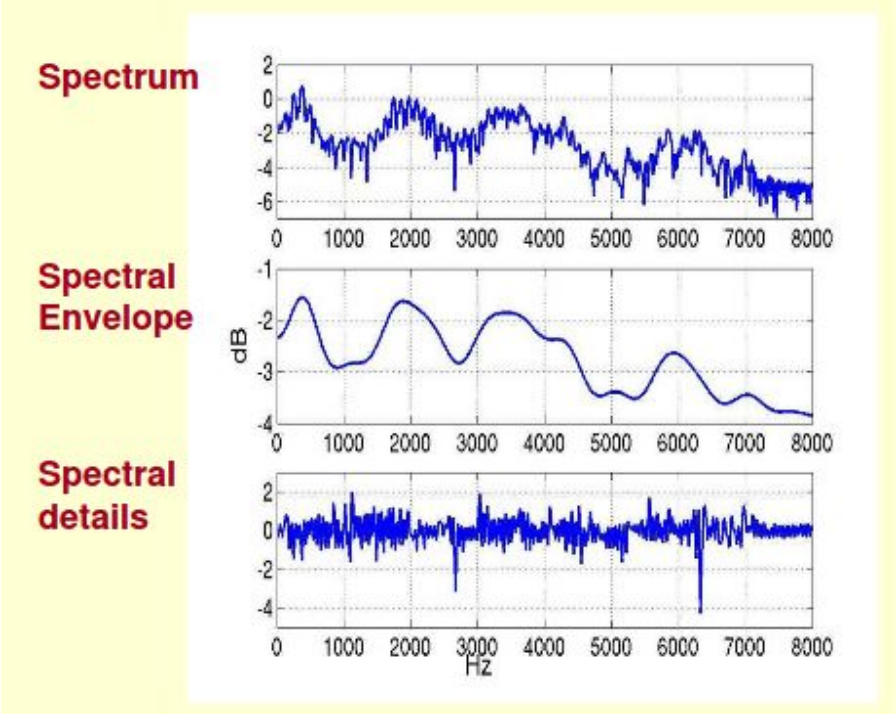


Figura 2

Essa separação pode ser obtida se aplicarmos log no espectro original X[k]. Seja H[k] o espectro da envoltória e E[k] os detalhes do espectro. Então log(X[k]) = log(H[k]) + log(E[k]).

Como efetuar essa separação?

Na prática nós não temos o H[k] nem E[k], apenas X[k]. Uma vez aplicado o log em X[k], faremos a transformada inversa, levando a um domínio de pseudo-frequências (cepstrum).

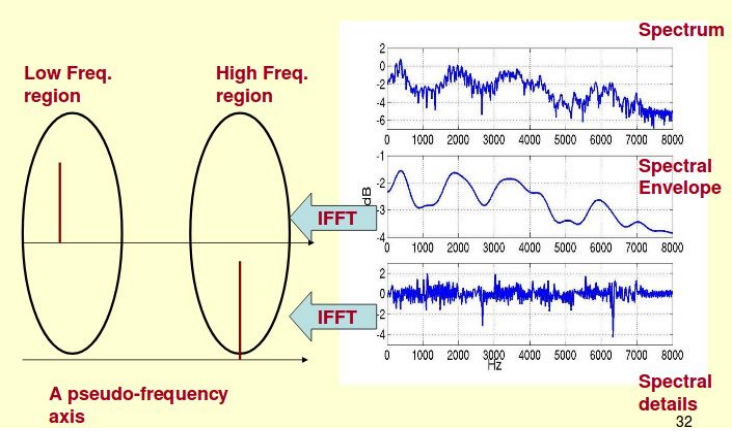


Figura 3

Conforme notamos da figura acima, o espectro X[k] pode ser representado por duas componentes, uma com baixa frequências e outra com altas frequências. Queremos obter aquela resultante da envoltória espectral, portanto basta aplicar um filtro passa baixas e aplicar a transformada para voltar ao domínio original. Agora obtemos os coeficientes que representam a envoltória (os pontos vermelhos da 1a figura).

Importante notar que, como a audição humana percebe melhor variações de baixa frequência, aplica-se um filtro não uniformemente distribuído, denominado Mel. Assim, com o espectro filtrado, obteremos apenas parte da envoltória original.

- **Delta MFCC:**

  Essa característica captura como os coeficientes MFCC variam no tempo, isto é, a sua derivada. Para tal, usamos a função delta() da biblioteca librosa que faz essa estimativa aplicando a filtragem Savitsky-Golay. Essa filtragem tem por objetivo primordial reduzir o ruído associado ao sinal, tornando mais suaves os picos e vales - pontos onde a derivada é inexata. Com isso, esperamos capturar a essência do sinal, com menos ruído, e em uma segunda etapa obter a derivada do sinal.

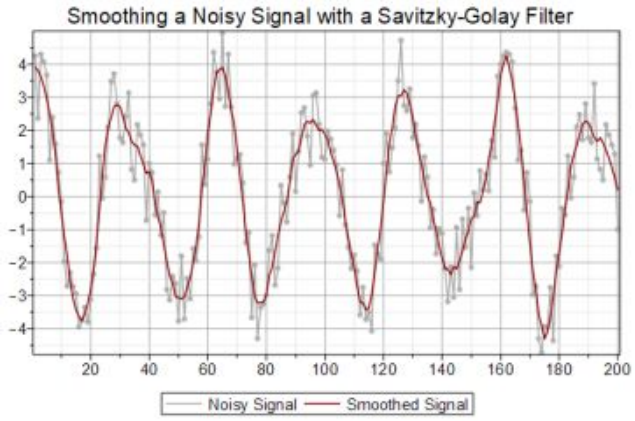


Figura 4

Note na figura 4 o sinal com ruído dado pela curva em cinza e o mesmo sinal após a aplicação do filtro em vermelho.

Para chegar em tal resultado é feita uma convolução do sinal, fazendo aproximações polinomiais de baixa ordem em cada conjunto de pontos por meio do método de mínimos quadrados. Dado um tamanho de janela com m pontos, no máximo podemos utilizar um polinômio de ordem m-1, pois caso contrário não teremos solução única. Com isso, podemos substituir cada ponto no sinal pela sua aproximação dada pelo polinômio obtido, obtendo a curva em vermelho.

Uma vez obtido o sinal suavizado, podemos simplesmente pegar a diferença para cada dois pontos consecutivos e obter assim a primeira derivada. Se quiséssemos pegar a derivada de segunda ordem, por exemplo, bastaria pegar o resultado da derivada de 1 ordem, aplicar o filtro e pegar novamente a diferença entre valores consecutivos. Esse processo pode ser repetido quantas vezes necessário até obtermos a derivada de ordem desejada.

4.1 - NOVAS FEATURES:

Após usarmos as features destacadas acima, também procuramos um conjunto extra para ser adicionado ao conjunto anterior. Assim, poderíamos comparar a resultado com e sem as features adicionais, de forma a tentar melhorar a eficiência da classificação. A escolha das features foi feita com a pesquisa em *[2]*. Os novas features foram as seguintes:

- **Spectral kurtosis**: indica a "planicidade" (*flatness*) ou o "pico" (*peakedness*) da distribuição da energia

$$\gamma_2 = \frac{1}{\sigma^4} \int (f-\mu)^4 . p(f) df$$

(Equação 6 (retirado de [2]))

- **Spectral slope**: indica o quão rápido o espectro vai em direção à altas frequências. Também dá uma indicação da taxa de decrescimento do amplitude

$$m = \frac{1}{\sum_f A(f)} \frac{N \sum_f f.A(f) - \sum_f f \times \sum_f A(f)}{N \sum_f f^2 - \left(\sum_f f \right)^2}$$

(Equação 7 (retirado de [2]))

Para a extração dessas features, não foram usadas ferramentas da Librosa, mas sim scripts python open source que podem ser encontrados em [3].

5 - METODOLOGIA EXPERIMENTAL E RESULTADOS:

Nesta seção, nós começamos apresentando nossa metodologia experimental e depois apresentamos os resultados. Nós escolhemos usar o modelo RNN também a fim de comparar com os resultados do modelo LSTM. Para cada tipo de rede neural, RNN e LSTM, consideramos tanto a alteração no tamanho das janelas quanto a utilização de novas features, conforme apresentadas na seção 4.1. Para os tamanhos das janelas, consideramos usar 23ms e 93ms, que são obtidos com os valores do parâmetro n_ftt iguais a 512 e 2048 e valores do parâmetro hop_length iguais a 128 e 512, respectivamente. Ao todo, oito modelos diferentes foram treinados e seus desempenhos foram expressos em termos da métrica de avaliação AUC ROC. Todos os resultados foram obtidos através de uma divisão dos dados em dois conjuntos, treinamento e teste, onde a proporção de cada conjunto foi 67% e 33%, respectivamente. Não foi utilizada a técnica de validação cruzada aqui neste trabalho devido ao tempo necessário para treinar as redes ser muito longo.

5.1 - ARQUITETURA DAS REDES NEURAIS UTILIZADAS:

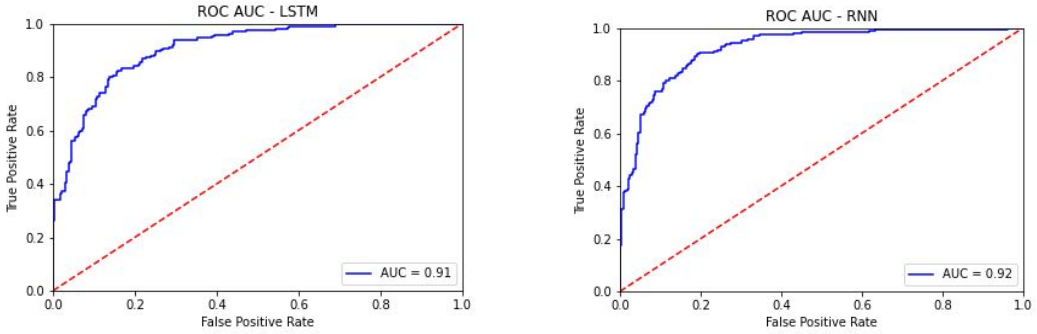
Primeiramente vamos entender o formato de entrado dos dados. Temos uma matriz tridimensional onde a primeira dimensão diz respeito ao tamanho do batch, isto é, quantas sequências (ou arquivos de áudio) serão utilizados para obter o gradiente e assim atualizar os parâmetros. Optamos por deixar com valor 32. Como temos 1301 arquivos e estamos utilizado 66% para treino, teremos (1-0.33)x1301 = 871 arquivos de treino e assim ceil(871/32) = 28 batches. Isso significa que a cada epoch teremos 28 gradientes calculados e 28 atualizações de parâmetros. A segunda dimensão diz respeito ao tamanho da sequência, isto é, quantos blocos são extraídos de cada arquivo de áudio. Aqui temos duas opções, caso o tamanho de janela seja 23ms teremos 517 para essa dimensão, e caso 93ms teremos 130. Essa será nossa dimensão de tempo. Por fim temos a dimensão de características que pode ser 32 ou 34 se considerarmos ou não as novas características calculadas.

Utilizando a biblioteca Keras, nós dispomos diversas camadas em sequência da seguinte maneira: primeiramente, colocamos 256 unidades LSTM, seguidas por uma camada de dropout para evitar overfitting. Essa última, a cada etapa de aprendizado, ignora a saída de uma fração dos neurônios. Dessa forma cada um poderá aprender características mais relevantes de forma independente. Imediatamente após essa camada, colocamos algumas camadas do tipo Time Distributed. Dessa forma aplica-se a rede em cada uma das fatias de tempo, ou seja em cada uma das 130 ou 517 linhas, ao invés de aplicar sobre a matriz toda. Segue-se então uma camada do tipo Flatten para transformar a sequência temporal num único vetor de características. Se ao final da camada TimeDistributed tínhamos 8 neurônios e o tamanho da sequência é de 130, teremos agora um único vetor de 8x130=1040 características. Por fim aplica-se várias camadas

do tipo dense sobre esse vetor, de forma que na última camada temos um único neurônio com função de ativação sigmoid que nos dará uma interpretação de probabilidade entre fazer parte de uma classe ou outra. A função de erro foi a entropia cruzada e o otimizador foi o Adam. Para a rede RNN colocamos a mesma quantidade de camadas, apenas alterando as duas primeiras, que são do tipo SimpleRNN, também com 128 células cada. A quantidade parâmetros para o LSTM foi de 357,561, enquanto que para o RNN foi de 197,433.

5.2 - TAMANHO DE JANELA DE 93ms - SEM NOVAS FEATURES:

Nessa configuração, as features da seção 4 foram extraídas para cada bloco de 93ms, gerando assim 130 janelas, mas as features da subseção 4.1 não foram incluídas. Observa-se pela figura 5 que os resultados foram bastante semelhantes para ambos os modelos.

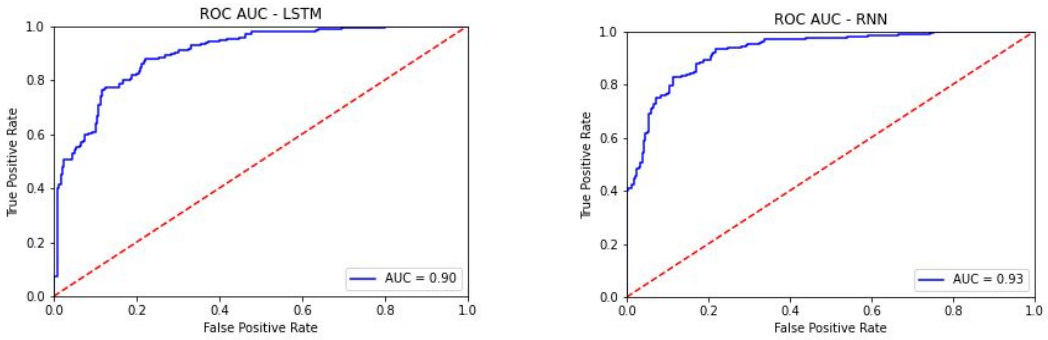


(a) (b)

Figura 5: Desempenho da LSTM e da RNN quando treinadas utilizando features extraídas para blocos de 93ms sem incluir as features da seção 4.1. À esquerda, a curva ROC e a sua respectiva área sob a curva para o modelo LSTM. À direita, a curva ROC e a sua respectiva área sob a curva para o modelo RNN.

5.3 - TAMANHO DE JANELA DE 93ms - COM NOVAS FEATURES:

Semelhante à configuração anterior, o tamanho da janela aqui considerado foi de 93ms, porém agora consideramos todas as features, inclusive as da seção 4.1. Os resultados são apresentados na figura 6.

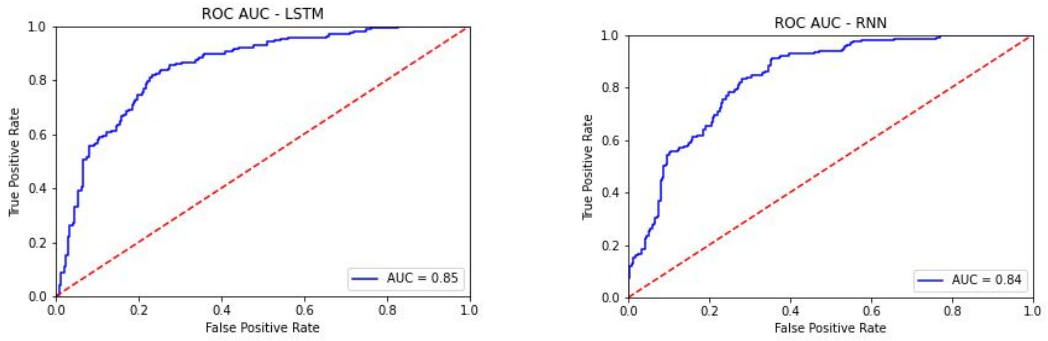


(a) (b)

Figura 6: Desempenho da LSTM e da RNN quando treinadas utilizando features extraídas para blocos de 93ms considerando todas as features. À esquerda, a curva ROC e a sua respectiva área sob a curva para o modelo LSTM. À direita, a curva ROC e a sua respectiva área sob a curva para o modelo RNN.

5.4 - TAMANHO DE JANELA DE 23ms - SEM NOVAS FEATURES:

Nessa etapa e na próxima etapa, nós estaremos estudando os efeitos da mudança no tamanho das janelas, a qual era 93ms nos experimentos anteriores e agora foi reduzida a 23ms, o que dá origem a 517 janelas por áudio, cerca de 4 vezes mais janelas se comparada às configurações anteriores. Novamente, não consideramos aqui a inclusão das features adicionais que propomos na seção 4.1. Os resultados podem ser encontrados na figura 7.

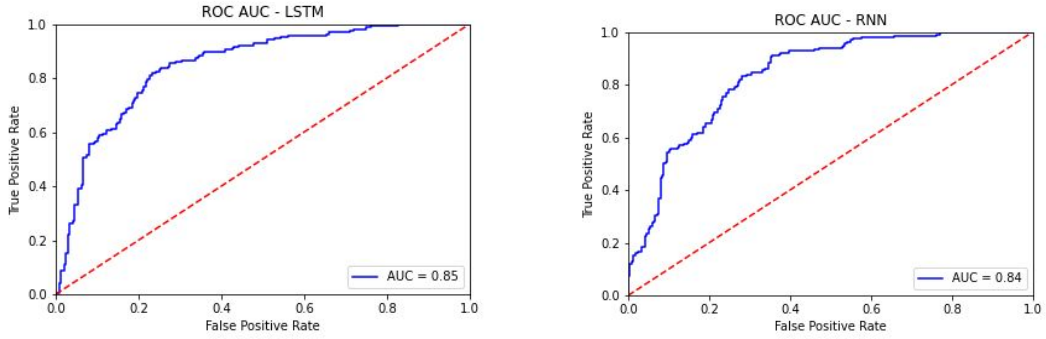


(a) (b)

Figura 7: Desempenho da LSTM e da RNN quando treinadas utilizando features extraídas para blocos de 23ms sem considerar as features adicionais da seção 4.1. (a) Curva ROC e a sua respectiva área sob a curva para o modelo LSTM. (b) Curva ROC e a sua respectiva área sob a curva para o modelo RNN.

5.5 - TAMANHO DE JANELA DE 23ms - COM NOVAS FEATURES:

Chegamos à nossa última configuração proposta: usar todas as features propostas, extraindo-as para cada janela de 23ms. Os resultados estão disponíveis na figura 8.



(a) (b)

Figura 8: Desempenho da LSTM e da RNN quando treinadas utilizando features extraídas para blocos de 23ms sem considerar as features adicionais da seção 4.1. (a) Curva ROC e a sua respectiva área sob a curva para o modelo LSTM. (b) Curva ROC e a sua respectiva área sob a curva para o modelo RNN.

6 - DISCUSSÃO:

Rede Neural	SF / 93 ms	SF / 23 ms	CF / 93 ms	CF / 23 ms
LSTM	**0.91**	0.85	0.90	0.86
RNN	0.92	0.84	**0.93**	0.80

Tabela 1: Tabela comparativa entre desempenhos (AUC ROC) obtidos para diferentes configurações. Nas linhas, temos os modelos LSTM e RNN, respectivamente. Nas colunas, encontramos as quatro combinações obtidas ao considerar os dois tamanhos de janela propostos, 23ms e 93ms, e a inclusão ou não inclusão das features descritas na seção 4.1. SF significa "Sem novas Features" e CF significa "Com novas Features".

Nós concentramos os resultados obtidos na seção anterior em uma única tabela para facilitar a discussão. Na tabela 1, portanto, é possível observar que para um mesmo conjunto de features, ambos os modelos LSTM e RNN têm o desempenho reduzido quando o tamanho das sequências aumentam, ou seja, quando o tamanho da janela é menor. No entanto, isso se torna mais evidente com as RNNs, o que pode ter relação com o fato de que elas possuem memória curta. Além disso, pode-se observar que em ambas as configurações em que o tamanho de bloco é 23ms, ou seja, a sequência é maior, o modelo LSTM obteve um resultado melhor que o modelo RNN. Por outro lado, em ambas as configurações em que o tamanho da janela foi 93ms, o modelo RNN obteve um desempenho melhor que o modelo LSTM em termos da métrica AUC ROC.

Com relação à adição das novas features, porém mantendo o tamanho das janelas, nós percebemos que as métricas sofrem mudanças mais suaves se comparadas à mudança no tamanho das janelas.

Finalmente, nós obtemos a média e o desvio padrão dos desempenhos de ambos os modelos LSTM e RNN, chegando a uma AUC ROC média de 0.8800 ± 0.0007 para o LSTM e uma AUC ROC média de 0.8725 ± 0.0029 para o RNN. Dessa forma, podemos dizer que o modelo LSTM não só foi melhor, na média, mas que também levou a um modelo mais estável devido ao menor desvio padrão.

7 - CONCLUSÃO:

Neste trabalho, nós propomos uma solução para a tarefa de classificação de instrumentos musicais que usa redes LSTM. Mais especificamente, nós restringimos o escopo para abranger somente os instrumentos 'piano' e 'violino'. Vimos que essa é uma etapa importante na área de Music Information Retrieval (MIR) e que diversas aplicações podem lançar mão dessa informação para prover uma melhor organização dos áudios ou mesmo para apoiar a recomendação de músicas para os usuários. Diversas tecnologias foram apresentadas, incluindo um ambiente colaborativo para desenvolvimento, uma biblioteca com implementação de redes RNN e LSTM e uma biblioteca para processamento de áudios. Um conjunto de features foi proposto e o efeito da mudança no tamanho das janelas em que essas features são obtidas foi devidamente documentado. Nós conseguimos modelos que, na média, são melhores para a rede LSTM, atingindo uma AUC ROC média de 0.8800 ± 0.0007.

Vale ressaltar que não conseguimos realizar todas os experimentos extras propostos, devido ao fator temporal. Esses experimentos se tratavam de considerar mais instrumentos e trabalhar com separação de áudios. Estes podem ser tratados como trabalhos futuros.

8 - REFERÊNCIAS BIBLIOGRÁFICAS:

[1] IRMAS: a dataset for instrument recognition in musical audio signals. <https://www.upf.edu/web/mtg/irmas>. Acesso em: 03 de novembro de 2020.

[2]Chandwadkar, D.M; Sutaoneg, M.S. Proper Features and Classifiers for Accurate Identification of Musical Instruments, 2013

[3]Audio Content Analysis. <https://www.audiocontentanalysis.org> Acesso em: 11 de novembro de 2020.

[4] Understanding LSTM. <https://colah.github.io/posts/2015-08-Understanding-LSTMs>. Acesso em: 09 de outubro de 2020.

[5] Audio Data Analysis Using Deep Learning with Python (Part 1). <https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html> Acesso em : 13 de outubro de 2020.