UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS

Renato Manoel de Souza

RECONHECIMENTO DE EMOÇÕES ATRAVÉS DA FALA UTILIZANDO REDES NEURAIS

FLORIANÓPOLIS
2020

RENATO MANOEL DE SOUZA

RECONHECIMENTO DE EMOÇÕES ATRAVÉS DA FALA UTILIZANDO REDES NEURAIS

Trabalho de conclusão de curso apresentado ao curso de Ciências da Computação da Universidade Federal de Santa Catarina, como requisito para obtenção do grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Mauro Roisenberg

Florianópolis
2020/1

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Souza, Renato Manoel de
Reconhecimento de emoções através da fala utilizando redes neurais / Renato Manoel de Souza ; orientador, Mauro Roisenberg, 2020.
113 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2020.

Inclui referências.

1. Ciências da Computação. I. Roisenberg, Mauro. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Renato Manoel de Souza

Esta Monografia foi julgada adequada para a obtenção do título de Bacharel em Ciências da Computação, sendo aprovada em sua forma final pela banca examinadora:

Orientador(a): Prof. Dr. Mauro Roisenberg
Universidade Federal de Santa Catarina - UFSC

Profa. Dra. Vania Bogorny
Universidade Federal de Santa Catarina - UFSC

Prof. Dr. Elder Rizzon Santos
Universidade Federal de Santa Catarina - UFSC

Florianópolis, dezembro de 2020

Este trabalho é dedicado ao meu pai Manoel Joaquim de Souza(In Memorian).

Agradecimentos

Agradeço inicialmente a minha mãe, Maria Elizia, meu padrasto Gervásio e a toda a minha família, pela paciência, carinho, amor e apoio durante os mais diversos momentos da vida acadêmica.

Agradeço a minha namorada e parceira de vida, Mariane, que fez do meu sonho o seu sonho do primeiro ao último dia da universidade.

Agradeço aos amigos que acompanharam essa caminhada, em especial, Yuri e Kamila, por ter me apoiado e incentivado durante esses anos na graduação.

Agradeço a todas as amizades que a universidade me proporcionou, com certeza fizeram parte de um momento importante da minha vida e serão levados no coração.

Ao meu orientador e professor Mauro Roisenberg, muito obrigado por ter aceitado me orientar e ter me auxiliado nos momentos de indecisão.

Agradeço a todos os professores do curso de Ciências da Computação e do Departamento de Informática e Estatística, que são responsáveis por construir um curso de excelência.

A Universidade Federal de Santa Catarina que me acolheu como um filho, permitindo uma experiência enriquecedora que jamais será esquecida.

A todos, muito obrigado.

"Para ser grande, sê inteiro: nada
Teu exagera ou exclui.
Sê todo em cada coisa. Põe quanto és
No mínimo que fazes.
Assim em cada lago a lua toda
Brilha, porque alta vive." - Fernando Pessoa

Resumo

A inteligência artificial vem demonstrando nos últimos anos resultados que podem auxiliar na solução de diversos problemas que com as tecnologias existentes ainda não são passíveis de solução. Devido esse fato, uma das áreas que podem utilizar-se da inteligência artificial é o reconhecimento de emoções através da fala, garantindo a aplicação real desses sistemas para facilitar e democratizar o acesso a esse tipo de tecnologia. O atendimento ao cliente será personalizado, com bots identificando o humor do cliente ao fazer um atendimento, podendo redirecionar para um atendimento humano caso notado um estresse na fala. Centrais de atendimento de seguradoras e serviços de emergência, principalmente, serão impactados positivamente com o reconhecimento de emoções. Este trabalho apresenta a Rede Neural Recorrente (RNN)-*Gated Recurrent Unit* (GRU) e a Rede Neural Convolucional (CNN) para classificar emoções através da fala que tiveram o melhor desempenho na etapa de experimentos. Para treinar esses modelos utilizou-se o conjunto de dados *Ryerson Audio-Visual Database of Emotional Speech and Song* (RAVDESS), permitindo construir um ambiente para fazer a avaliação e testes. A avaliação de um modelo treinado na língua inglesa que reconhece áudios da língua portuguesa apresentou uma precisão de aproximadamente 42%, sendo considerado insatisfatório para um classificador. As principais características que foram identificadas como responsável pelo desempenho foram, as características do conjunto de amostras, o viés da classificação sem validação e a falta do tratamento de ruído. A rede neural que apresentou a melhor precisão foi RNN-GRU com 79.69% utilizando uma técnica para aumentar o tamanho do conjunto de dados através da operação de transformação alongamento.

Palavras-chave: redes neurais, inteligência artificial, reconhecimento de emoções, extração de características da fala, aprendizagem de máquina, emoções, fala.

Abstract

Artificial intelligence has been showing results in the last few years that can help in the solution of several problems that with the existing technologies are not yet soluble. Due to this fact, one of the areas that can use artificial intelligence is the recognition of emotions through speech, ensuring the real application of these systems to facilitate and democratize access to this type of technology. The customer service will be personalized, with bots identifying the customer's mood when making a service, and can redirect to human service if stress in speech is noticed. Insurance call centers and emergency services, mainly, will be positively impacted by the recognition of emotions. This paper presents RNN-GRU and CNN to classify emotions through speech that had the best performance in the experiment stage. To train these models, the data set RAVDESS was used, allowing the construction of an environment to carry out the evaluation and tests. The evaluation of a model trained in the English language that recognizes audios in the Portuguese language showed an accuracy of approximately 42%, being considered unsatisfactory for a classifier. The main characteristics that were identified as responsible for the performance were, the characteristics of the sample set, the classification bias without validation and the lack of noise treatment. The neural network that presented the best precision was RNN-GRU with 79.69% using a technique to increase the size of the data set through the elongation transformation operation.

Keywords: neural networks, artificial intelligence, emotion recognition, extraction of speech characteristics, machine learning, emotions, speech.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas

SER *Speech Emotion Recognition*

MFCC *Mel Frequency Cepstral Coefficients*

IA Inteligência Artificial

RNA Redes Neurais Artificiais

ANN Artificial Neural Networks

RN Redes Neurais

ML *Machine Learning*

CNN Rede Neural Convolucional

DBN *Deep Belief Network*

RNN *Recurrent Neural Network*

DNN *Deep Neural Networks*

MLP Perceptron de Várias Camadas

ReLu *Rectified Linear Unit*

GRU *Gated Recurrent Unit*

RNN Rede Neural Recorrente

LSTM *Long Short-Term Memory*

BLSTM *Bidirectional long short-term memory*

Emo-DB *Berlin Database of Emotional Speech*

IEMOCAP *Interactive Emotional Dyadic Motion Capture*

SVM *Support Vector Machine*

KNN *K-Nearest Neighbors*

RAVDESS *Ryerson Audio-Visual Database of Emotional Speech and Song*

RMS *Root Mean Square*

API *Application Programming Interface*

IoT *Internet of Things*

IHM Interação Homem-Máquina

DCT Transformada Discreta de Cosseno

STT *Speech to Text*

Sumário

1 INTRODUÇÃO

Métodos para extração de características da fala são pesquisados há muito tempo, e uma importante pesquisa no começo dos anos 50 permitiu dividir a história em antes e depois. Pesquisadores do laboratório AT&T Bell Lab conseguiram extrair informações do sinal da fala, possibilitando a construção do primeiro sistema de reconhecimento da fala com capacidade de reconhecer 10 palavras em inglês, essas 10 palavras foram os números de 0 a 9 [Yu 2012].

A Precisão do sistema Audrey, como ficou conhecido, era de 90% quando o seu criador falava, passando para 70%/80% de precisão quando outra pessoa falava. Essa característica do sistema Audrey já demonstrava o tamanho do desafio para construir um sistema capaz de reconhecer diversos dialetos, com velocidades e gírias, diferentes entre cada cultura.

Desde então, diversas pesquisas foram realizadas afim de entender como o processo de formação da fala influenciava na extração de características. Outras pesquisas buscaram entender como o ruído do ambiente e a própria qualidade do áudio interferem no reconhecimento da fala. Esse interesse se dá principalmente pela aplicação da tecnologia, como forma de aperfeiçoar a Interação Homem-Máquina (IHM).

Objeto de pesquisa, o reconhecimento de emoções através da fala é fundamental para que as máquinas possam reconhecer as necessidades humanas. Isso permitirá, por exemplo, agilizar atendimentos em sistemas de emergências, melhorar os sistemas de recomendações e claro, a voz terá papel fundamental na iteração com carros autônomos, tratamentos terapêuticos, diagnósticos de depressão e até dispositivos de *Internet of Things* (IoT) [Pandey, Shekhawat e Prasanna 2019].

A evolução da Inteligência Artificial (IA) permitiu criar novas formas de reconhecer emoções através da fala, com base nisso, este trabalho visa realizar uma comparação entre duas redes neurais, redes convolucionais e redes recorrentes.

1.1 MOTIVAÇÃO

Nos últimos anos, a capacidade de processamento dos computadores aumentou a um nível que possibilitou criar novas técnicas para auxiliar na resolução de problemas. A aprendizagem de máquina, também conhecida como *Machine Learning*, ganhou evidencia justamente com a popularização de computadores com maior poder computacional, mas outro fator que também disseminou a aprendizagem de máquina foi a facilidade de acessar ambientes remotos para processar grandes volumes de dados. Esses ambientes

remotos, muitas vezes chamado de *clusters* em nuvem, do inglês *cloud*, facilitou o acesso de pessoas com algum interesse ou de outras áreas que não a computação. Pequenas e médias empresas também fazem uso desse tipo de ambiente para criar e testar novas soluções, democratizando o acesso [MICROSOFT 2017].

Sistemas por comando de voz estão ganhando maior visibilidade e maior atenção da indústria. O mercado automobilístico, com carros autônomos e o mercado de relacionamento com o cliente, com *bots* virtuais, são bons exemplos de mercados que estão apostando nessas tecnologias para criar diferencial competitivo. Assim, se tornou uma necessidade natural evoluir algoritmos e técnicas para aprendizagem de máquina.

Muito popular, os sistemas *Speech to Text* (STT) representam muito bem a evolução das técnicas de reconhecimento de padrões, como os sistemas *Alexa*[1] e *Siri*[2] que utilizam essa técnica para criar produtos e melhorar a experiência do usuário. Se comparado com o STT, o reconhecimento de emoções através da fala ainda não está com pesquisas tão avançadas a ponto de ser utilizado em um produto. Por isso, buscar a evolução é garantir que a interação entre homem e máquina seja ainda mais natural, parte disso passa por entender a relevância da cultura e idioma do falante [SILVA, BARBOSA e ABELIN 2016] para discernir quanto afeta no aprendizado de máquina. Por entender que existem novos modelos de redes neurais que podem permitir avançar nos sistemas de reconhecimento de emoções atuais, é que está sendo proposto uma comparação entre duas arquiteturas de redes onde seja possível treinar e testar em línguas distintas.

A motivação está intrinsecamente ligada ao benefício que sistemas de comunicação trarão para a sociedade, e vislumbrando uma troca de estímulos maior entre homem e máquina, a identificação de emoções é fundamental.

1.2 OBJETIVOS

1.2.1 OBJETIVOS GERAIS

Avaliar o desempenho e a capacidade de classificar áudios da língua portuguesa sobre duas arquiteturas de redes neurais treinadas na língua inglesa para o reconhecimento de emoções através da fala.

1.2.2 OBJETIVOS ESPECÍFICOS

Para cada arquitetura de rede neural, identificar qual modelagem apresenta melhor resultado em relação a todas as emoções. Analisar as características que permitem uma emoção ser melhor representada por uma modelagem do que por outra. Considerando

[1] https://developer.amazon.com/en-US/alexa
[2] https://www.apple.com/br/siri/

o desenvolvimento do trabalho e o objetivo geral apresentado, destacam-se os seguintes objetivos específicos:

- Utilizar o conjunto de dados RAVDESS que contém 1440 arquivos de áudio. As emoções presentes no conjunto de dados são: neutro, calma, alegre, tristeza, raiva, medo, nojo e surpresa;
- Realizar um estudo sobre o método *Mel Frequency Cepstral Coefficients* (MFCC), que permite identificar características importantes sobre a voz.
- Compreender o que são redes neurais e seus tipos CNN e RNN-GRU.
- Desenvolver uma aplicação em *Python*[3] para reconhecer emoções.
- Criar experimentos afim de identificar o melhor modelo para o reconhecimento de emoções.
- Verificar se a rede treinada no idioma inglês funciona para o conjunto de áudios em português.
- Analisar os resultados obtidos.

1.3 METODOLOGIA

A metodologia de pesquisa utilizada para desenvolver essa pesquisa, tem como base livros, teses, dissertações e publicações em periódicos. Priorizou-se os conceitos básicos em livros, mas toda a estrutura de onde estamos no estado da arte, tem base em publicações recentes, visando trazer um conteúdo atualizado para a pesquisa proposta. Uma pesquisa inicial foi realizada para entender o processo de formação do espectro de áudio quando um humano está falando. Após essa primeira etapa, foi definido quais são as características da fala importantes para o reconhecimento de emoções, etapa básica para a escolha de quais emoções foram reconhecidas pelo modelo de aprendizagem de máquina. Para um bom processo de extração de características, foi dado foco no MFCC, que gera um vetor de componentes desse áudio e visa expor apenas características relevantes no processo de reconhecimento de fala. Após a extração, foi utilizado diferentes arquiteturas de *deep learning* para o reconhecimento de emoções. Utilizou-se um conjunto de áudios em português para fazer verificações afim de determinar quanto uma língua influencia na etapa de reconhecimento de outra.

[3] https://www.python.org/

1.4 ORGANIZAÇÃO

O trabalho está organizado da seguinte forma:

- Capítulo 2: apresenta o que entende-se por emoção, os conceitos relacionados ao processo de formação do espectro de áudio, assim como foi apresentado o MFCC para extração de características e pré-processamento. Também foi apresentado os conceitos sobre Redes Neurais Artificiais (RNA).
- Capítulo 3: descreve os trabalhos desenvolvidos e o estado da arte sobre *Speech Emotion Recognition* (SER).
- Capítulo 4: construção e apresentação dos modelos e algoritmos de RNA que foram utilizados.
- Capítulo 5: implementa os modelos construídos para o desenvolvimento desse trabalho.
- Capítulo 6: apresenta os experimentos e resultados obtidos dos treinamentos, das análises de dados e dos testes realizados.
- Capítulo 7: conclusão da pesquisa realizada e apresentação dos trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ANÁLISE DA EMOÇÃO

A emoção está presente em todas as formas de comunicação, garantindo uma boa percepção do ouvinte para interpretação do estado emocional. Essas formas que representam o estado emocional são divididas entre verbais e não verbais [Juslin e Scherer 2008], fluindo conjuntamente uma com a outra para melhor clareza da informação. A subjetividade está atrelada justamente quando se tenta utilizar esses métodos de maneira separada, já que cada parte da comunicação é influenciada pela emoção.

Os parâmetros da voz podem ser medidos através de características que vão além do comportamento vocal, já que os aspectos fisiológicos, como por exemplo a excitação ligada a um estado de raiva que produz alterações na respiração e aumento da tensão muscular, afetam as vibrações das cordas vocais e influenciam o ouvinte na percepção da emoção [Juslin e Scherer 2008].

Segundo [Juslin e Scherer 2008] a voz descreve três níveis diferentes de emoções, são elas:

- Fisiológico
- Fonatório e articulatório
- Acústico

O fisiológico descreve impulsos nervosos envolvidos na produção da voz, o Fonatório-articulatório descreve o movimento das pregas vocais e por último o acústico que descreve as características das formas de onda.

Um sistema de comunicação completo define que o processo emocional também está envolvido com várias camadas da formação vocal. Processo esse que pode ser exposto através de quatro fases [Scherer 2003]:

- A codificação;
- Os sinais acústicos;
- A percepção dos sinais;
- E a decodificação.

A codificação é uma expressão da emoção por parte do falante, os sinais acústicos indicam o sentimento da emoção, a percepção dos sinais pelo lado do ouvinte e a decodificação é a reação/sensação do ouvinte em relação a interpretação pessoal dele.

2.2 CARACTERÍSTICAS ACÚSTICAS

Um sistema automático de reconhecimento de emoções através da fala pode ser dividido em dois principais métodos, um é conhecido como acústico e outro como linguístico [Schuller et al. 2011]. O método linguístico utiliza a semântica e o contexto do que está sendo comunicado, enquanto o acústico utiliza o espectro de áudio e a intonação para determinar e classificar a emoção expressa na conversação.

Uma forma de extrair informações usando o recurso acústico é identificar os descritores, que são chamados de descritores de baixo nível (LLD), ou seja, características que são descritas sem que seja necessário utilizar o contexto semântico para identifica-las, assim, os descritores podem indicar a intonação, por exemplo.

Os descritores são importantes para avaliar um determinado áudio, mas é importante salientar que apesar de existir uma correlação entre os vários descritores, não é consenso que exista um único conjunto de descritores que descrevem as emoções, mesmo as mais básicas, como a raiva e a felicidade. Além disso, depende do tipo de aplicação e qual o ambiente de coleta desses áudios, se será num ambiente com ou sem ruído e com boa ou má qualidade na captação. Quanto maior o ruído gerado, maior será o tempo necessário para realizar a higienização desse áudio, afetando certamente nas etapas de extração de características e classificação.

Existem diversos descritores, tais como entonação, intensidade, predição linear, coeficiente cepstral, formantes, espectro, transformações, harmonicidade e pertubação [Schuller et al. 2011].

Alguns desses descritores tem uma relevância maior quando se fala sobre métodos para extração de características da fala para detecção de emoções, por isso, abaixo será detalhado os principais:

- Entonação: A entonação pode ser vista como sendo o Pitch/altura do som é possível determinar se um som é agudo ou grave. O pitch é determinado pela frequência fundamental(f0), ou seja, quanto maior a altura do som, maior é a frequência (mais agudo) [Seara].
- Intensidade: A intensidade do sinal da fala é a amplitude de um som, que pode ser chamada de volume. Como a percepção da intensidade não é linear, é medido utilizando uma escala logarítmica, conhecido como decibel(dB) [Schuller et al. 2011].

- Espectro: O espectro é formado por um conjunto de frequências que representam o formato da onda.

2.3 MEL FREQUENCY CEPSTRAL COEFFICIENTS

O coeficiente cepstral pertence a classe dos descritores mencionados anteriormente, com isso, se encaixa dentro dos métodos acústicos para extração de características.

MFCC é um método para extrair coeficientes de um espectro de áudio, permitindo criar um vetor de características. Introduzido inicialmente por Davis e Mermelstein na década de 1980 [Davis e Mermelstein 1980], fornecem uma melhor representação do sinal, já que exploram a frequência que os humanos ouvem.

Para que o processo de extração ocorra dentro de um padrão, o método utiliza um janelamento, são instantes de tempo que tem uma duração especifica, como por exemplo de 20ms, garantindo que cada janela de tempo terá esse tamanho de amostra para que seja realizada a analise da fala. Se esse janelamento for menor, pode não haver amostras suficientes, e se for muito grande pode haver muita informação dificultando o filtro.

Uma maneira de examinar as características de amplitude em cada quadro é estimando o periodograma, e é exatamente isso que o MFCC considera nas primeiras etapas da extração dos recursos. Como pode conter muitas informações desnecessárias nessa janela de tempo, é realizado um agrupamento de blocos de periodograma com o filtro Mel.

Para encontrar as várias frequências contidas no espectro da fala, utiliza-se a escala Mel [Stevens, Volkmann e Newman 1937], proposta para compreender como as frequências são analisadas pelo ouvido humano e, como relacionar a frequência de um tom percebida a uma real que foi medida, permitindo uma maior correspondência como a forma que se ouve, mesmo sabendo que os humanos são melhores na percepção de mudanças da fala em frequências menores. Abaixo as formulas que representam a conversão da frequência em escala Mel e vice-versa [Makhoul e Cosell 1976]:

$$M(f) = 1125ln(1 + \frac{f}{700})$$

$$M^{-1}(m) = 700(e^{\frac{m}{1125}} - 1)$$

Não existe uma fórmula única para representar essa conversão, cada autor justifica sua fórmula, como o proposto por Gunnar Fant, em 1949 [Fant 1949] e resumido no seu livro de 1973 [Fant 1973], que menciona uma escala de 1000Hz.

A saída da equação acima são N filtros triangulares, onde a quantidade depende do intervalo da frequência escolhida. Os filtros são ditos triangulares devido a sobreposição da escala Mel, já que são calculados utilizando a média ponderada. Conforme apresenta a figura abaixo:

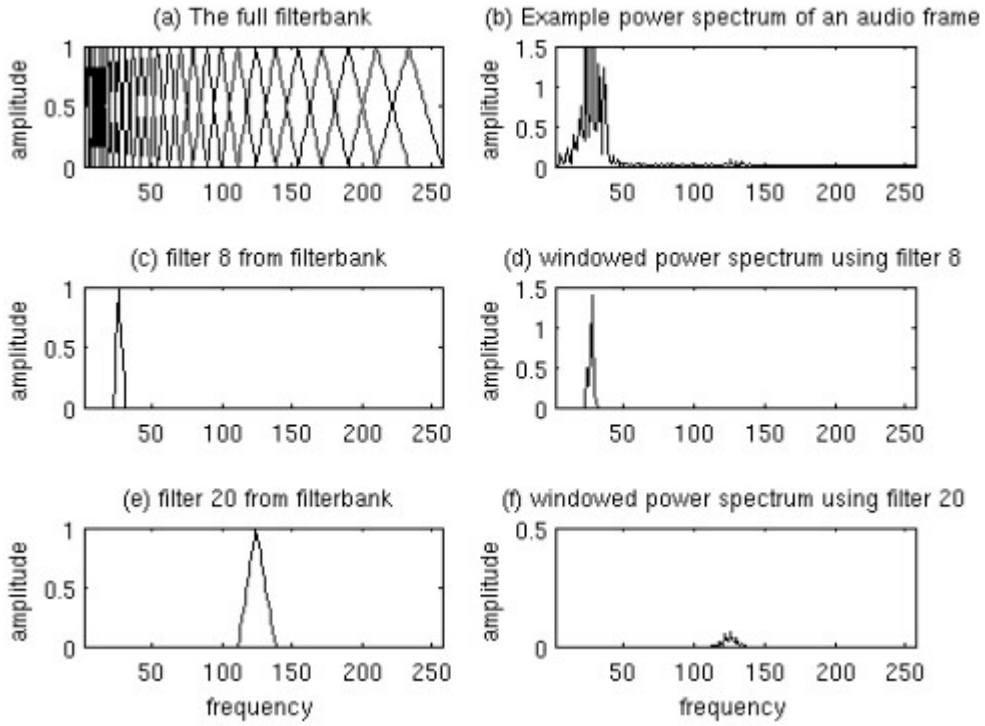


Figura 1 – Banco de filtros numa escala Mel [Lyons 2013]

Após aplicada a conversão do sinal para a escala Mel e a obtenção dos filtros, utiliza-se a Transformada Discreta de Cosseno (DCT) no logaritmo da energia para obter o vetor de características, que também é conhecido como Coeficientes Mel Cepstrais. Esse processo é descrito pela equação abaixo:

$$c_n = \sum_{k=1}^{K} log(S_k) cos[n(k - \frac{1}{2})\frac{\pi}{K}], n = 1, ..., L$$

Figura 2 – Coeficientes Mel Cepstrais [Thiago 2017]

A etapa para gerar vetores de características é considerada fundamental para que se obtenha sucesso nas etapas seguintes. Importante citar que existem outros métodos para essa extração que não foram abordados, assim como algumas combinações de diversas técnicas afim de melhorar a informação de entrada para os classificadores.

2.4 REDES NEURAIS ARTIFICIAIS

RNA podem ser entendidas como uma forma de modelar um problema computacional, problemas que surgiram com o avanço da tecnologia, por isso houve a necessidade de criar uma abstração que permitisse representar o mundo real. Essa abstração foi inspirada nos neurônios biológicos e como esses estão estruturados no cérebro, tendo em vista que o grande volume aumenta a capacidade de processamento. Essa ideia iniciou com um modelo computacional para representar um neurônio, proposto por McCulloch e Pitts (1943). Depois Hebb (1949) propôs um modelo de aprendizado e por último veio o modelo perceptron, proposto por Rosemblatt (1957), que a partir de uma entrada com um determinado peso, uma regra de propagação e uma função de ativação obtêm-se uma saída. Com esses 3 estudos foi possível simular, mesmo que basicamente, o funcionamento de um neurônio.

Um modelo de neurônio é apresentado abaixo, onde observa-se alguns dos elementos citados anteriormente.

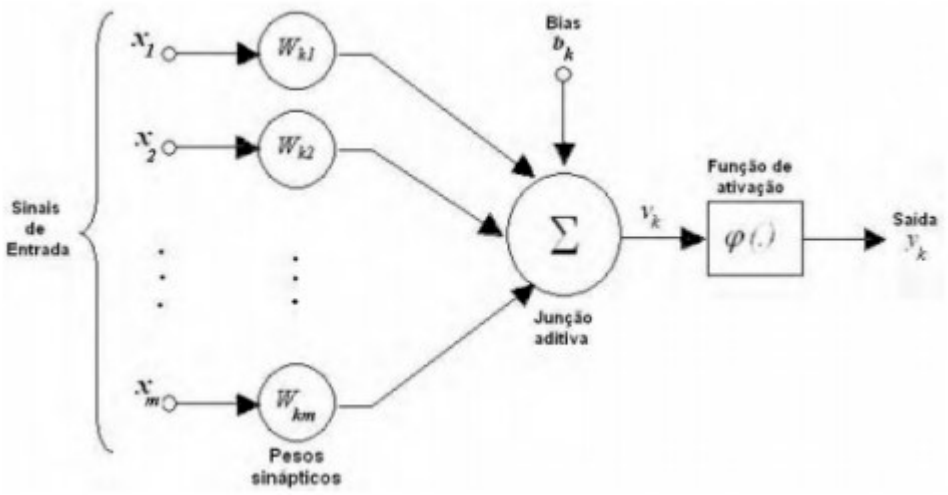


Figura 3 – Modelo de neurônio [Haykin 2000]

A função de ativação dita o comportamento do neurônio, segue alguns exemplos abaixo:

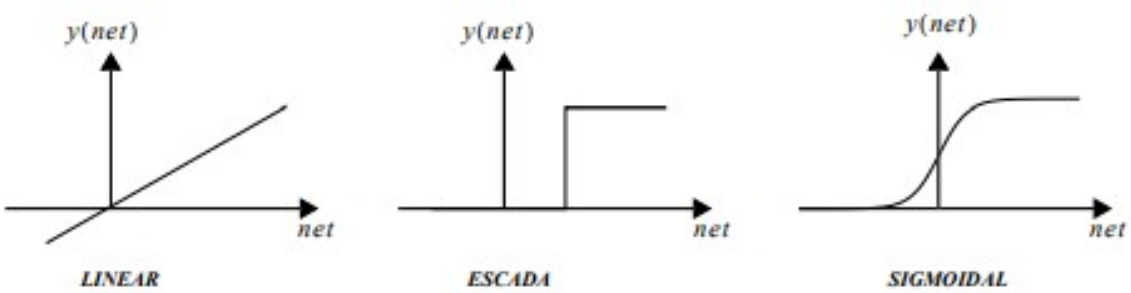


Figura 4 – Funções de ativação

Além dessas funções apresentadas acima, uma função importante e que merece destaque é a ReLu. Essa é uma função simples e de rápida convergência. Outras funções eram aplicadas, entretanto, descobriu-se que a ReLu é mais rápida e atinge o objetivo. Um ponto de atenção sobre essa função de ativação é a ocorrência do gradiente de fuga, mesmo que tenham uma menor influencia [Glorot, Bordes e Bengio 2011], onde valores dos parâmetros de entrada tem pouca influencia no valor de saída, significa, no pior dos casos, que a rede não consegue aprender. A função abaixo representa o comportamento da ReLu.

$$f(x) = max(0, x)$$

2.4.1 *MULTILAYER PERCEPTRON* E *BACKPROPAGATION*

O conceito importante por trás do Perceptron de Várias Camadas (MLP) está na implementação de camadas ocultas, e como elas se conectam umas as outras. Basicamente é uma rede completamente conectada, ou seja, cada nó da rede se conecta a todos os nós da rede seguinte. Não bastou apenas criar camadas, mas sim como cada camada pode ter suas entradas manipuladas para extrair boas informações sobre os dados de entrada. Por isso, o algoritmo *backpropagation* define um processo para calcular várias vezes os pesos da rede. Segundo [Rumelhart, Hinton e Williams 1986] o processo foi definido para minimizar o erro entre os vetores de saída da rede. Com os pesos sendo calculados várias vezes, as camadas intermediária fornecem características valiosas sobre o dado de entrada, que anteriormente, só era possível com problemas lineares.

O fluxo básico do algoritmo consiste em duas fases, a primeira é conhecida como processamento direto, do inglês *forward pass*, onde a entrada é propagada pela rede até o última camada. Nessa primeira fase, pode-se ver que o fluxo de ida da rede não se altera, o que se altera é como os pesos serão atualizados para a próxima passagem. Já a segunda fase é conhecida como processamento reverso, do inglês *backward pass*, para esse passo é calculado o gradiente da função de perda na última camada, ou seja, para cada previsão existirá um erro atrelado com base no valor esperado, conhecido como processo supervisionado, com isso a fase de processamento reverso garante que um sinal será propagado a cada camada da rede onde os valores serão atualizados para a próxima iteração do algoritmo. Esse processo iterativo permite minimizar o erro em função do valor esperado.

2.4.2 REDES NEURAIS CONVOLUCIONAIS

A definição básica de CNN, também chamada de *ConvNet*, é ser um algoritmo que tem a capacidade de ser treinado e aprender padrões. Como é uma variação do MLP, utiliza os mesmos conceitos de pesos e funções de ativação não lineares. Mas alguns novos

conceitos são incluídos como camada convolucional, camada de *Polling*, camada de *SoftMax* e a camada totalmente conectada. Também será apresentada a técnica de regularização *Dropout* e detalhado um pouco a função de ativação ReLu após a etapa de convolução.

2.4.2.1 CAMADA CONVOLUCIONAL

Convolução é uma operação matemática que toma duas funções para gerar uma terceiro função que mede a soma do produto desses sinais ao longo da região submetida. Uma forma fácil de visualizar esse processo é imaginar uma imagem, onde cada pixel da imagem é uma posição que será submetida a um filtro, que neste caso será uma matriz com valores criados afim de conhecer algumas características dessa imagem inicial. Após essa etapa onde tem-se duas informações, é realizada a soma dos produtos entre essas duas matrizes, e aplicando um processo de deslizamento desse filtro sobre toda a imagem, tem-se uma imagem que vai realçar o comportamento da imagem analisada, por exemplo, se predomina alguma cor ou existe uma variedade de cores.

As figuras 5, 6 e 7 apresentam graficamente o que foi descrito acima.

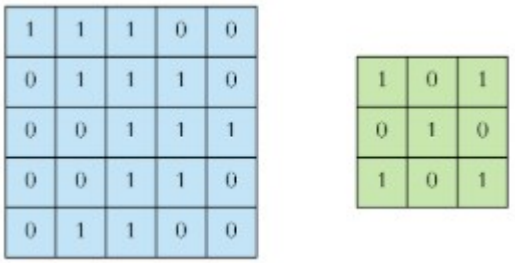


Figura 5 – Matriz de entrada 6x6 e matriz filtro 3x3 [Dertat 2017]

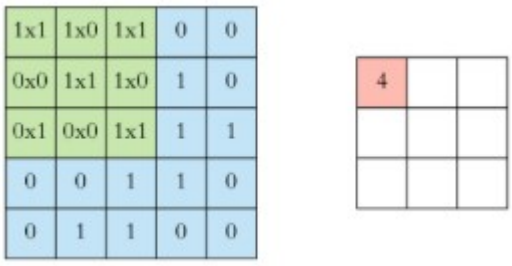


Figura 6 – Matriz de características [Dertat 2017]

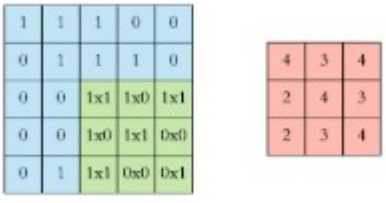


Figura 7 – Matriz de características completa [Dertat 2017]

É possível destacar alguns parâmetros que precisam ser configurados para que o processo de convolução ocorra da maneira desejada, por isso é importante citá-los. Esses parâmetros determinam o formato da saída ao final da convolução, por isso é importante controlar e entender como afetam o resultado final da rede neural. Abaixo os parâmetros que controlam o tamanho da saída da convolução:

- Profundidade: é igual a quantidade de filtros que são aplicados na convolução. Um exemplo seria o número de canais numa imagem.
- *Stride*: determina o salto que será aplicado a cada deslizamento no processo de convolução, esse é um parâmetro que geralmente é 1, onde cada filtro será aplicado pulando uma casa para a direita, como no exemplo nas figuras 6 e 7.
- *Padding*: serve para controlar o número de reduzibilidade da convolução, num cenário onde a entrada passa por muitas convoluções se altera significativamente o tamanho da saída, uma forma de contornar é adicionar zeros nas bordas para manter a dimensionalidade durante as inúmeras operações.

2.4.2.2 CAMADA *POLLING*

Após a camada ReLu é possível inserir uma camada de *polling* que tem a função de reduzir a quantidade de parâmetros e controlar o *overfitting*. Essa função de *polling* é aplicada a uma determinada região de tamanho especifico que é deslocado(*stride*) pela área de entrada aplicando um regra de maximização, minimização ou média, o caso mais comum é maximização, onde numa área reduz à apenas o maior valor ali contido. Esse processo reduz o número de computação, já que diminui o número de informações que serão processadas pelas outras camadas da rede.

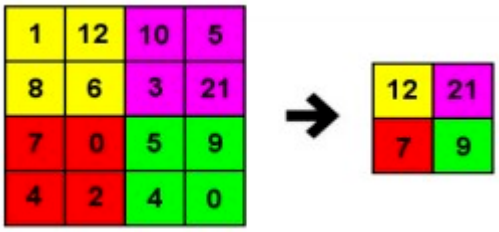


Figura 8 – Exemplo de *MaxPooling* com filtro 2x2 e *Stride* 2 [Ferreira 2017]

Para determinar a dimensão de saída, utiliza-se a equação abaixo, onde W é a dimensão da entrada, P é o *padding*, K é o tamanho do filtro e S é o *stride*.

$$O = \frac{(W - K - 2P}{(S)} + 1$$

2.4.2.3 CAMADA TOTALMENTE CONECTADA

Nessa camada, todos nós são conectados a todos os nós da camada anterior, ou seja, o tamanho dessa camada depende da quantidade de nós na camada anterior. Nessa camada, considerada como pertencente a etapa de classificação, é onde as informações são aprendidas com base no que foi filtrado nas camadas anteriores (convolução, *ReLu* e *polling*)

2.4.2.4 CAMADA *SOFTMAX*

Nessa camada, geralmente tem-se a mesma quantidade de nós que o número de classes de decisão do problema, em geral a *softmax* é uma função de ativação que tem como função definir qual a probabilidade da entrada pertencer a classe. A saída é normalizada, de forma que a saída é um valor pertencente ao intervalo [0,1], portanto, a soma de todos os valores devem ser 1.

$$S(Y_i) = \frac{e^{Y_i}}{\sum_{n=j} e^{Y_j}}$$

Por ter a mesma quantidade de nós da classe de entrada, está é geralmente a última camada, porém, existem diversas arquiteturas que variam conforme o tipo de aplicação. Aqui foi apresentado de maneira geral as principais etapas, podendo haver diferentes formatos.

2.4.2.5 FUNÇÃO DE ATIVAÇÃO ReLu

É indicado que tenha uma função de ativação ReLu após a etapa de convolução, isso garante uma não linearidade, já que o processo de convolução basicamente calcula sistemas lineares.

2.4.2.6 TÉCNICA *DROPOUT*

Para evitar *overfitting* é possível aplicar algumas técnicas de regularização, como L1 e L2, entretanto, no contexto desse trabalho será citado apenas a técnica mais utilizada e que apresenta um bom resultado. O *Dropout* aplica a remoção de nós aleatoriamente junto como todas as suas conexões de entrada e saída, assim evitando que a rede se acostume com os dados de entrada. Essa técnica remove apenas nós ocultos, deixando os nós de entrada e saída inalterados.

2.4.3 REDES NEURAIS RECORRENTES

As RNN foram propostas para poder processar uma sequencia de dados de entrada, onde a informação anterior é utilizada como saída da rede neural. Então, para definir

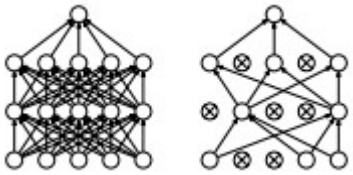


Figura 9 – Exemplo de *Dropout* [Srivastava et al. 2014]

uma sequencial de entrada é necessário definir o tamanho que pode ter, que depende do problema à ser classificado.

Na descrição matemática abaixo o b é o *bias*, o W representa os pesos e x as entradas, onde o tamanho da sequencia é definido por t, com intervalo variando de 1 até t.

$$h_t = f(b_h + X_t W_x + h_{t-1} W_h)$$

$$y_t = b_0 + h_t W_0$$

Com a ideia acima, é possível entender a recorrência da rede e o processo de realimentação, onde cada nó(unidade) da rede pode estar conectado com nós da camada anterior ou até mesmo na mesma camada. Com isso, a saída do nó não depende só do valor principal da entrada, mas também das anteriores. O principio de realimentação caracteriza a ideia de memória, que evidencia o armazenamento da informação.

Na figura abaixo, o processo de dependência entre camadas é visualizado. Após o processamento da entrada s na unidade, a saída apresentará o resultado e uma cópia é enviada como entrada para a próxima iteração. Ou seja, quando a próxima palavra entrar na unidade, a saída anterior será utilizada para processar a próxima entrada e assim sucessivamente.

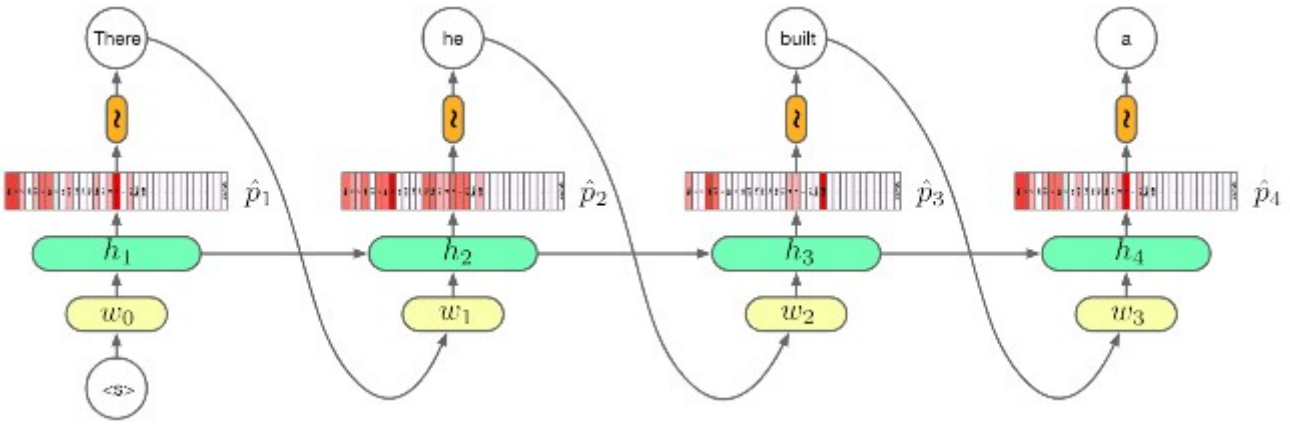


Figura 10 – Exemplo de realimentação numa RNN [Blunsom 2017]

As RNN são apropriadas para processamento de áudio e vídeo, que a informação anterior é importante para a decisão da saída. Contudo, esse modelo apresenta alguns

problemas por conta da dependências de longas sequencias. A lacuna de informação pode ser muito grande, o que degrada o aprendizado da rede fazendo com que a informação do erro seja perdida devido o método de treinamento utilizar o gradiente [Bengio, Simard e Frasconi 1994]. Um gradiente muito pequeno faz com que os pesos do modelo não sejam atualizados, já um gradiente grande tornam a rede instável.

2.4.3.1 *GATED RECURRENT UNIT* - GRU

Proposta por [Cho et al. 2014], as redes *GRU* foram introduzidas para tratar o problema gradiente da RNN. Vista como uma simplificação da *Long Short-Term Memory* (LSTM), ela reduz o número de portões utilizados. Com essa ideia, permite melhorar o desempenho e manter a capacidade de aprendizado de longas sequencias.

A GRU utiliza dois portões(*gates*), um para realizar a atualização(*update*) e outro para a redefinição(*reset*) da informação.

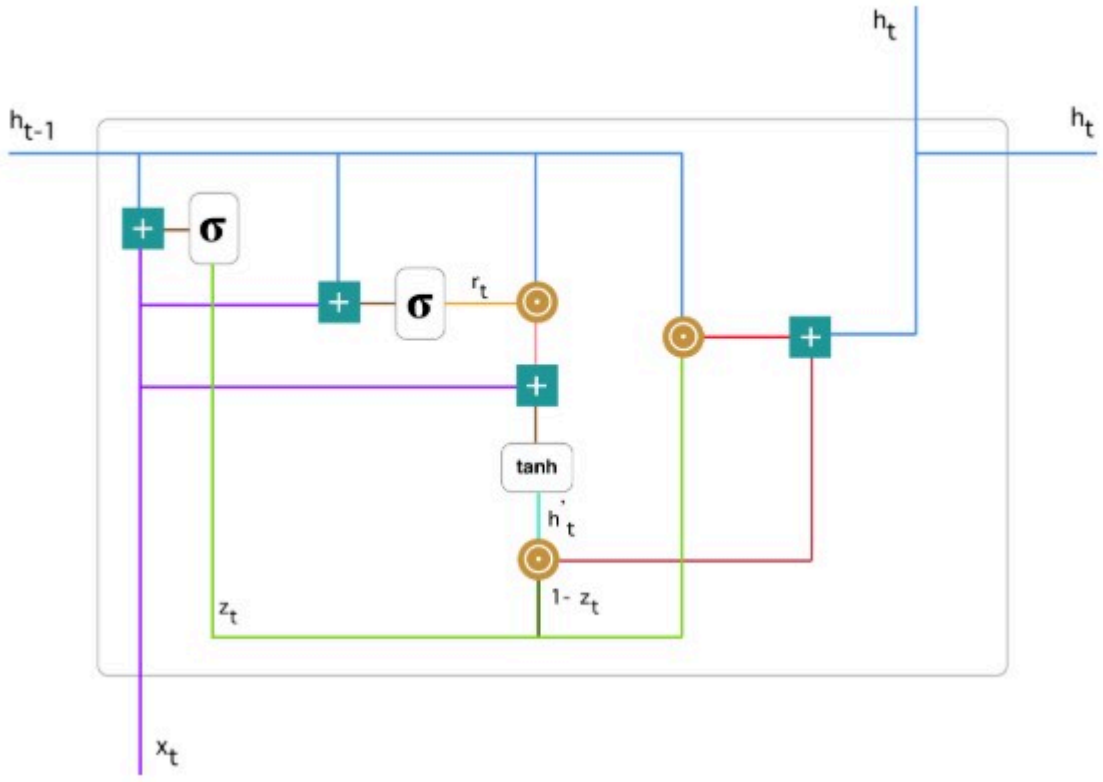


Figura 11 – Uma unidade da RNN-GRU [Kostadinov 2017]

- Atualização do portão: A função dessa etapa é definir quanto das informações anteriores precisa ser passada para a próxima etapa, ou seja, quanto do passado será lembrado no futuro. Isso evita o problema de dissipação do gradiente mencionado na versão inicial da RNN. A atualização ocorre quando a função z_t é aplicada, onde

x_t é a entrada atual, h_{t-1} é a saída anterior e $W^{(z)}$ e $U^{(z)}$ são os pesos de x_t e h_{t-1}, respectivamente.

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

- Redefinição do portão: Nessa etapa é definido quanto das informações anteriores serão esquecidas pela rede. A função é a mesma da etapa de atualização, o que muda são os pesos que influenciam o que deverá ser esquecido.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

- Conteúdo atual da memória: Essa é a etapa que define qual o conteúdo da memória nesse instante, quando utilizará a saída da redefinição para guardar informações relevantes do passado.

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

- Saída da unidade oculta: É a etapa que tem a memória final, definindo o valor que saíra como resultado sobre etapas anteriores. O valor de saída alimenta o próximo nível, que obterá apenas informações relevantes para garantir a memória da rede. Essa visão geral é apresentada na figura 12.

$$h_t = (z_t \odot h_{t-1} + (1 - z_t) \odot h'_t)$$

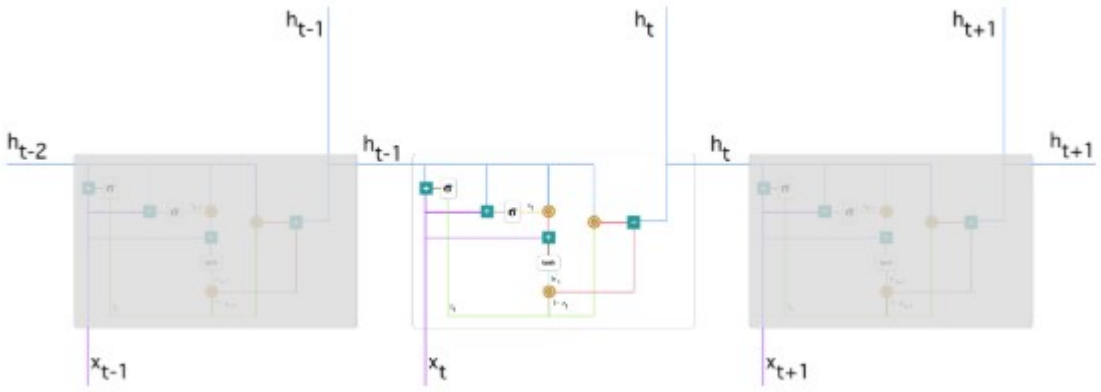


Figura 12 – Visão geral, apresenta a dependência entre cada unidade RNN-GRU [Kostadinov 2017]

3 TRABALHOS CORRELATOS

Nesse capítulo serão apresentados trabalhos que aplicaram técnicas para o reconhecimento de emoções através da fala. Os trabalhos foram escolhidos para permitir uma variação dos conceitos e métodos aplicados, permitindo aumentar o campo de visão sobre o que está sendo pesquisado nessa área do conhecimento.

3.1 *Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network*

No trabalho apresentado por [Badshah et al. 2017], é demonstrado como é possível utilizar CNN para fazer uma análise da imagem do espectrograma. Essa metodologia é apresentada visando fugir do pipeline padrão de reconhecimento de emoções através da fala. O espectrograma é formado por um eixo x que denota o tempo e um eixo y que denota a frequência. Nesse gráfico formado pelo espectrograma a amplitude da frequência é indicada pela intensidade das coras, as mais frias representam menor intensidade e as mais quentes maior.

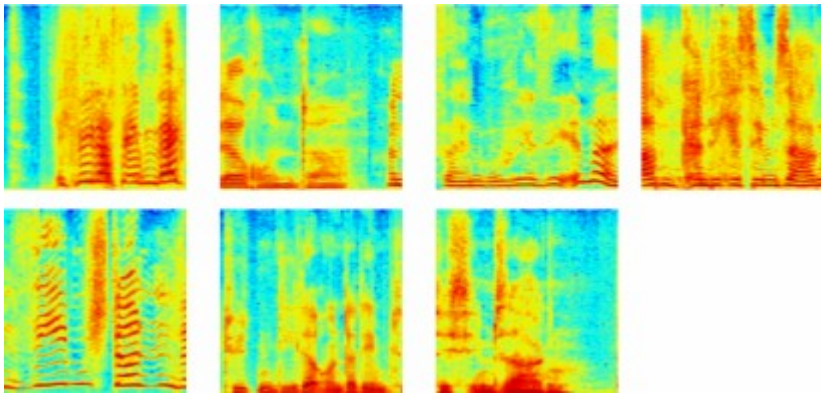


Figura 13 – Exemplo de espectrogramas da fala [Badshah et al. 2017]

Os resultados obtidos pelo modelo proposto foram de 84,3% considerando a média das 7 classes de emoções treinadas, sendo elas raiva, tédio, nojo, tristeza, medo, felicidade e neutra. O autor também cita os problemas de confusão que a rede comete, como por exemplo medo é constantemente confundido com raiva, nojo e felicidade. Para reduzir esse erro, uma forma citada pelo autor é aumentar o conjunto de dados para fornecer mais elementos característicos dessas classes a rede neural.

3.2 *Deep Learning Techniques for Speech Emotion Recognition: A Review*

O trabalho de [Pandey, Shekhawat e Prasanna 2019] faz uma comparação de duas arquiteturas de redes neurais e a combinação dessas duas para realizar o reconhecimento de emoções. Como entrada dessas redes são utilizados três métodos de extração de características, espectrogramas de magnitude, espectrogramas de log-mel e coeficientes mel-cepstrais. Foram utilizados *Berlin Database of Emotional Speech* (Emo-DB) e *Interactive Emotional Dyadic Motion Capture* (IEMOCAP) como conjunto de dados, o primeiro é uma base de dados em alemão e o segundo é um conjunto de dados em inglês. Os dois conjuntos de dados tem as mesmas sete classes, entretanto, foram utilizados apenas as classes raiva, felicidade, tristeza e neutro.

Os métodos foram fornecidos como entrada para uma rede CNN e o método de extração que apresentou melhor resultado como entrada para essa rede foi espectrograma de log-mel com uma classificação de 78.16% para o Emo-DB e 47.04% para IEMOCAP.

Os mesmos métodos também foram fornecidos para a rede *Bidirectional long short-term memory* (BLSTM), que é uma variação bidirecional da rede LSTM, o método MFCC apresentou melhor desempenho, com 66.61% para o conjunto de dados Emo-DB e 46.21% para o IEMOCAP.

Uma última comparação também foi realizada usando a CNN para extrair recursos de alto nível e o LSTM para modelar informações ao longo do tempo. Nesse cenário obteve-se melhor desempenho para o conjunto Emo-DB utilizando MFCC com 82.35%. Para o conjunto IEMOCAP o método espectrograma log-mel obteve 50.05%, o que representa o melhor resultado nesse conjunto de dados.

Para justificar a diferença entre os dois conjuntos de dados, o autor cita a maior naturalidade do IEMOCAP.

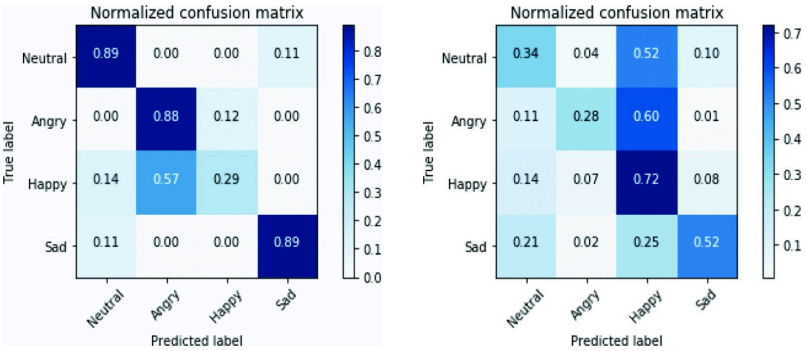


Figura 14 – Matriz de confusão: a esquerda CNN+BLSTM com MFCC e a direita CNN+BLSTM com espectrograma log-mel [Pandey, Shekhawat e Prasanna 2019]

3.3 Reconhecimento Automático de Emoções Através da Voz

No trabalho realizado por [Júnior 2017], foi proposto a utilização de *K-Nearest Neighbors* (KNN) e *Support Vector Machine* (SVM) como forma de comparar qual classificador tem melhor desempenho. Também foi utilizado o conjunto de dados Emo-DB com as sete classes. O autor também utiliza um conjunto de áudios criado por ele, para realizar o reconhecimento de emoções utilizando a língua portuguesa do Brasil.

Quando analisado o desempenho do KNN com as sete classes e K=3 obteve 66.7% de acurácia. Já quando o mesmo algoritmo foi aplicado com quatro classes, neutra, raiva, felicidade e tristeza, e o mesmo k = 3, a acurácia subiu para 82.2%.

Para o SVM com sete classes de emoções e com um parâmetro de custo igual a 10, o SVM apresentou uma acurácia de 71.5% contra 84.1% com apenas quatro classes.

Outro importante teste realizado foi equalizar o número de amostras de áudio em cada classe, como alternativa foram duplicados algumas amostras visando equiparar o conjunto de áudio. Com essa alternativa o desempenho da SVM melhorou, chegando a uma acurácia de 93.6% com quatro classes de emoção. Com o conjunto de áudios equiparado, foi testado um método de força bruta para buscar os melhores parâmetros de janela e passos, os valores encontrados permitiram melhorar a acurácia chegando a 94.3%.

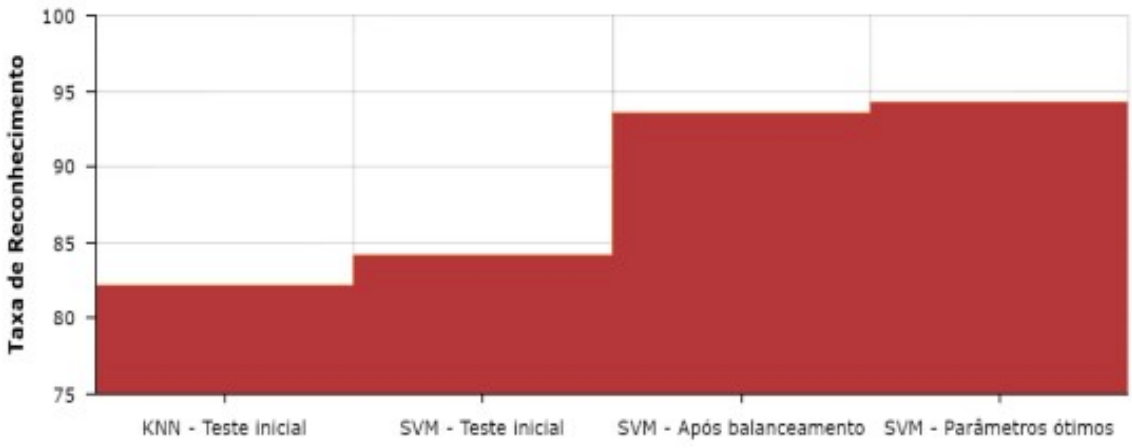


Figura 15 – Resultados obtidos em cada método com 4 classes de emoções [Júnior 2017]

Para o conjunto de áudios em português e aplicando a técnica para equiparar o número de amostras no algoritmo SVM, a acurácia obtida na etapa de validação foi de 82.1%. Quando foi aplicado força bruta, o resultado da acurácia na etapa de validação foi de 84.4%. Quando aplicado sobre as amostras de testes, a acurácia foi de 70.83%.

3.4 *Speech emotion recognition with deep convolutional neural networks*

O trabalho desenvolvido por [Issa, Fatih Demirci e Yazici 2020] faz uso dos conjuntos de dados IEMOCAP, Emo-DB e RAVDESS. O autor utiliza cinco métodos para extração de características que são fornecidos como entrada para a rede neural convolucional proposta, os métodos são MFCC, *Mel-scaled spectrogram*, *Chromagram*, *Spectral contrast feature* e *Tonnetz representation*.

O modelo proposto tem seis camadas convolucionais de uma dimensão, com a primeira camada recebendo um vetor de 193 características, que conforme informado, é um modelo base que sobre alterações para execução dos experimentos. A função de ativação utilizada é ReLu com o otimizador RMSProp a uma taxa de aprendizado de 0.00001.

Falando especificamente do conjunto de dados RAVDESS, o autor utiliza as 8 classes de emoções, totalizando os 1440 arquivos de áudios do conjunto. O resultado apresentado foi uma acúracia de 71.61%, onde o melhor desempenho foi para a classe raiva, com 92.31% e, o pior desempenho foi na classe tristeza com 51.85%.

Figura 16 – Resultados obtidos com RAVDESS e 8 classes de emoções [Issa, Fatih Demirci e Yazici 2020]

O autor ainda comparou o resultado do modelo com o resultado da validação realizada pelos autores do conjunto [Livingstone e Russo 2018], ficando pouco acima do reconhecimento humano, 71.61% do autor contra 67% da validação humana.

4 DESENVOLVIMENTO

4.1 CONJUNTO DE DADOS

O conjunto de dados utilizado para treinamento é o RAVDESS, publicado na revista PLoS ONE, foi produzido como parte de uma pesquisa para desenvolvimento de um conjunto com arquivos de áudio, vídeo e audiovisual [Livingstone e Russo 2018]. Nesse trabalho será utilizado apenas o conjunto de áudio.

A distribuição e características desses áudios se dá da seguinte forma:

- 24 atores profissionais, com 12 homens e 12 mulheres.
- 60 áudios por ator, totalizando 1440 arquivos.
- na língua inglesa norte americana.
- as emoções são: calma, feliz, triste, raiva, medo, surpresa e nojo.
- cada emoção é reproduzida em dois níveis de intensidade, normal e forte.

Os áudios são rotulados com um identificador exclusivo e dividido em sete partes, conforme a seguir:

- Modalidade: 01=áudio-video, 02=video e 03=áudio
- Tipo de canal: 01=fala, 02=música
- Emoção: 01=neutro, 02=calmo, 03=feliz, 04=triste, 05=raiva, 06=medo, 07=nojo, 08=surpreso
- Intensidade da emoção: 01=normal, 02=forte
- Frase: 01=*"Kids are talking by the door"*, 02=*"Dogs are sitting by the door"*
- Repetição: 01=primeira repetição, 02=segunda repetição
- Ator: 01 até 24

4.1.1 CONJUNTO EM PORTUGUÊS

Para a realização dos testes na língua portuguesa foi utilizado um conjunto de amostras criado a partir de partes de vídeos do YouTube[4], com a classificação manual de cada áudio para uma das quatro classes de emoção [Júnior 2017].

A distribuição desses áudios se dá da seguinte forma:

- Raiva (*angry*): 37 áudios.
- Feliz (*happy*): 27 áudios.
- Triste (*sad*): 22 áudios.
- Neutro (*neutral*): 24 áudios.

4.2 FERRAMENTAS

O projeto foi desenvolvido com o auxilio de algumas ferramentas, que serão detalhadas para facilitar o entendimento ao longo do projeto.

4.2.1 LibROSA

Desenvolvida por [McFee et al. 2015], LibRosa é uma biblioteca para realizar análise de áudios e música, fornecendo uma interface fácil para extrair características. Alguns dos recursos que podem ser extraídos de um áudio usando a biblioteca são:

- MFCC
- Espectrograma em escala de mel
- *Root Mean Square* (RMS): Calcula a raiz do valor quadrático médio

Além disso, a biblioteca conta com um módulo para visualização de espectrograma, como pode ser visto na figura .

4.2.2 TENSORFLOW

O *TensorFlow*[5] é uma ferramenta que permite criar, modelar, treinar, validar e testar uma arquitetura de aprendizado de máquina. A ferramenta é amplamente utilizada, principalmente pela fácil implementação quando utilizada a *Application Programming*

[4] https://www.youtube.com/
[5] https://www.tensorflow.org/

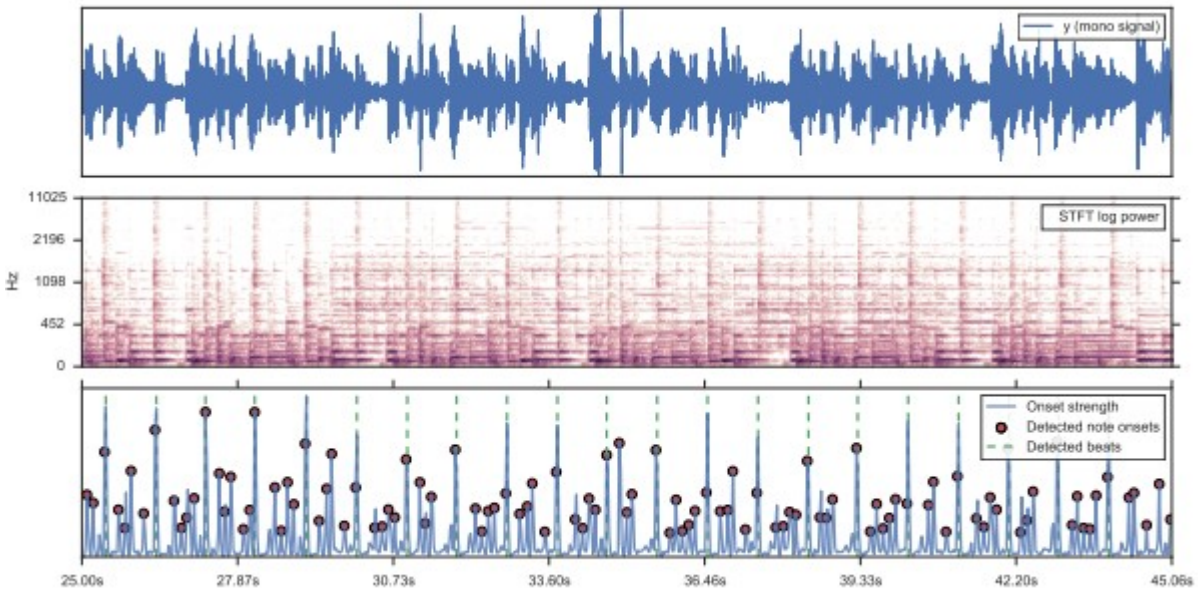


Figura 17 – Apresentação da biblioteca LibRosa [McFee et al. 2015]

Interface (API) do *Keras*[6], que já implementa alguns dos modelos mais utilizados. Permite a definição de qualquer arquitetura utilizando os módulos específicos, tornando o desenvolvimento mais rápido.

4.3 EXPLORAÇÃO DOS DADOS

Antes de iniciar a etapa de extração de características, é necessário explorar o conjunto de dados, permitindo identificar possíveis oportunidades.

A primeira análise a ser feita é contar o número de ocorrência de cada emoção. Abaixo o gráfico apresenta a distribuição do número de emoções no conjunto:

Uma primeira observação a ser realizada é sobre o número de áudios classificado como neutro, é inferior aos demais. O conjunto foi criado considerando duas gravações para a mesma emoção com intensidade diferente, porém, apenas a emoção neutra isso não se aplica. Como serão utilizadas as emoções neutra, feliz, triste e raiva para treinamento e classificação, é importante observar que a quantidade de áudio com a emoção neutro pode afetar o desempenho.

A próxima análise, figura 19, é comparar a forma de onda(*waveform*) de cada emoção, subdividindo entre gênero masculino e feminino.

Os áudios tem uma duração média de 3,5 segundos, onde o primeiro segundo é praticamente vazio, ou seja, não vai auxiliar no treinamento, assim como o final. Outra observação importante é a diferença entre masculino e feminino, o que pode no momento

[6] https://keras.io/

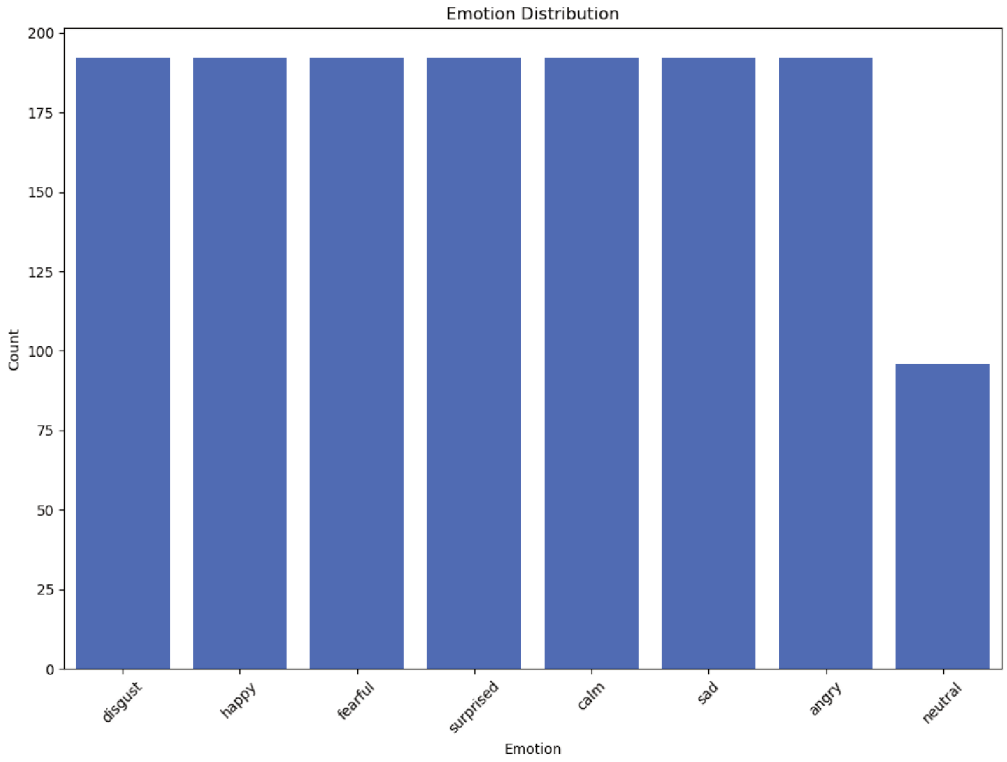


Figura 18 – Distribuição das emoções. Fonte: O autor (2020)

da classificação do conjunto de teste provocar falso-positivo.

4.4 EXTRAÇÃO DE CARACTERÍSTICAS

A etapa para extrair características é muito importante para qualquer desenvolvimento que visa fazer uma classificação, principalmente os que utilizam redes neurais, por isso foi apresentado na seção dois desse projeto o MFCC, uma maneira de analisar as características que melhor representam as percepções auditivas dos humanos, além de captar melhor as baixas frequências presente na voz e as mudanças espectrais mais abruptas.

A biblioteca LibRosa, apresentada na seção 4.2.1, foi utilizada como interface para facilitar a interação com o algoritmo MFCC, que fornece um vetor dos atributos presentes no espectro.

Os valores de entrada para o algoritmo foram personalizados conforme documentação da biblioteca e testes realizados:

- n_mfcc: número de características retornadas do áudio de entrada
- n_fft: tamanho do sinal numa janela de tempo

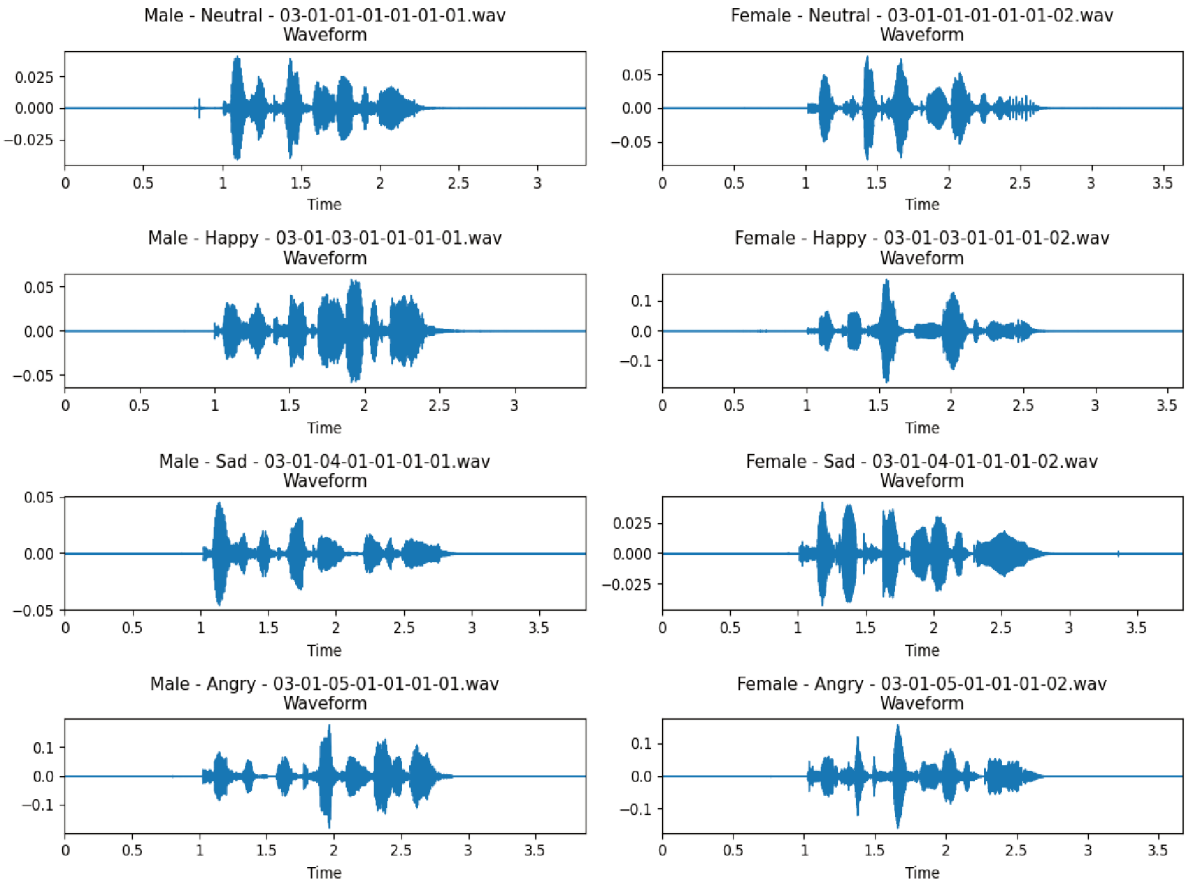


Figura 19 – Forma de onda. Fonte: O autor (2020)

- fmax: frequência máxima da escala Mel

Foram testados alguns valores para o tamanho do vetor de características, sendo que nenhum apresentou diferenças significativas nos testes realizados, por isso foi optado por não padronizar um valor fixo, deixando a critério dos testes de desempenho de cada rede neural. Para determinar o tamanho da janela de tempo, foi utilizado o valor 1024 proposto na documentação da biblioteca como uma boa referência para processamento de voz. Com esse valor a janela é de aproximadamente 23 milissegundos. O valor máximo da frequência para o banco de filtro Mel foi uma proposta realizada por [Lyons 2013].

A figura abaixo apresenta uma comparação dos vetores de características, separado por emoção e gênero:

4.5 ESTRATÉGIA SOBRE OS DADOS

Existem algumas técnicas que podem melhorar o desempenho de uma rede neural de aprendizado supervisionado, e uma delas é aumentar o tamanho do conjunto de dados com base nos dados que se tem, ou seja, gera-se novos dados fazendo pequenas transformações nos dados existentes. Essa técnica, conhecida como *data augmentation* pode melhorar

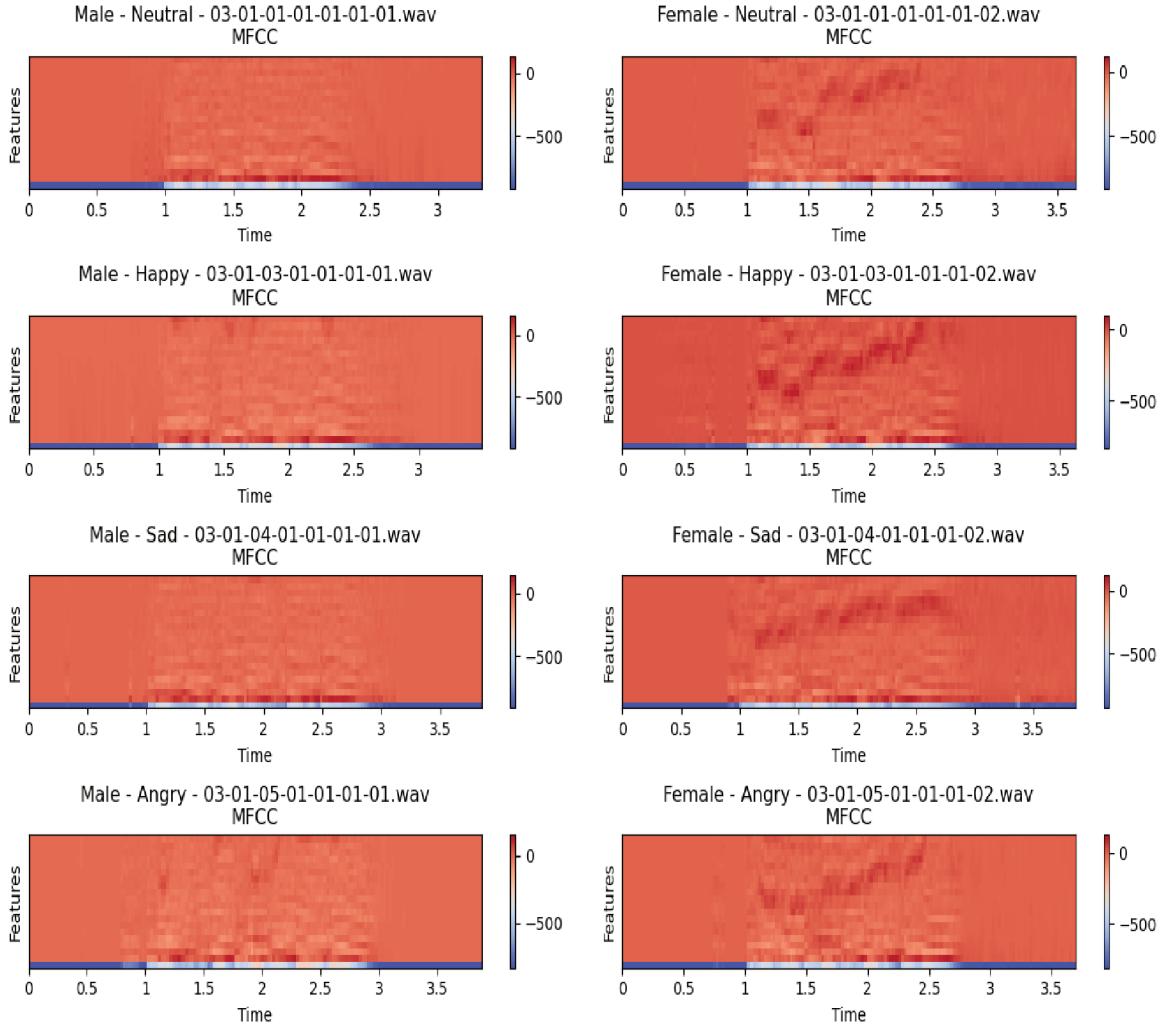


Figura 20 – Vetor de características ao longo do tempo. Fonte: O autor (2020)

a acurácia, já que imagina-se que quanto maior o conjunto de dados, melhor será o desempenho. Outra técnica é o balanceamento do conjunto de dados, garantindo que o número de ocorrências de cada emoção seja igual, garantindo que as classes menos presentes tenham a mesma probabilidade de serem aprendidas.

4.5.1 DATA AUGMENTATION

Para aplicar transformações sobre o conjunto de dados que será utilizado para treinamento da rede neural, aplica-se operações como ruído(*noise*), deslocamento(*shift*), alongamento(*stretch*) e *pitch* [LOK, KIN e DEANE-MAYER 2019]. Essas são as principais operações realizadas para expandir o conjunto de dados com novos áudios.

Essas operações podem ser aplicadas separadamente ou em conjunto, também pode-se combinar algumas operações de forma isolada. Não é possível garantir que apenas aumentando o conjunto de treinamento o resultado será melhor, por isso cada teste deve ser isolado para identificar qual operação contribui para aumentar a precisão do modelo.

São fatores que devem ser levados em consideração quando estratégias como essas são aplicadas.

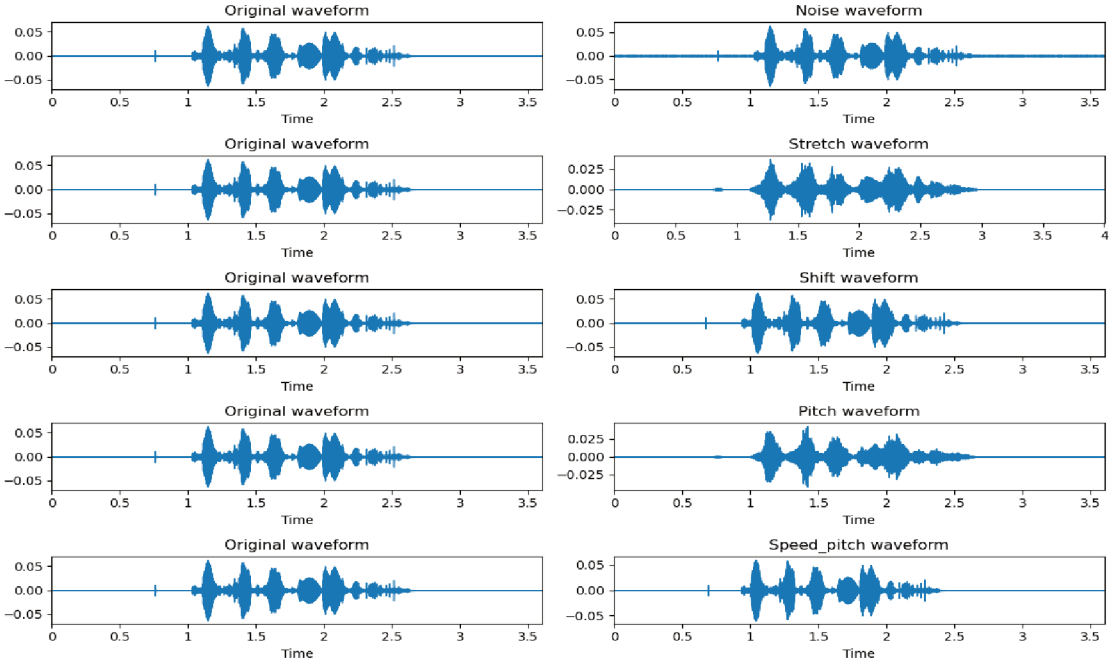


Figura 21 – Comparação das operações de transformação. Fonte: O autor (2020)

4.5.2 BALANCEAMENTO

O conjunto de dados não é equilibrado, a emoção neutra não tem a mesma quantidade de áudios que as outras emoções. Para garantir que o número seja igual e não ocorra impacto no treinamento foi adicionado o dobro de áudios da emoção neutro com a operação de transformação *noise*, permitindo que o conjunto fosse equiparado, já que o conjunto não é grande e retirar áudios poderia afetar o desempenho do modelo.

4.6 ARQUITETURAS PROPOSTAS

A proposta é comparar redes neurais convolucionais e redes neurais recorrentes tipo GRU para identificar qual arquitetura obtém melhor desempenho utilizando a métrica de precisão(*accuracy*). Com base nisso, abaixo serão apresentados os quatro modelos, dois de cada arquitetura, que serão treinados e avaliados.

A tabela 3 apresenta as camadas das duas redes CNN que foram comparadas. A escolha de cada uma foi com base em testes realizados sobre o conjunto de dados original e expandido, e que obtiveram a melhor acurácia.

Os modelos foram construídos utilizando convolução 1D, mais adequado para análise de série temporal, conforme destacado na documentação da API do Keras. A

entrada da camada convolucional tem o formato (*batch_size*, *input_dim*), onde o primeiro parâmetro é o número de vetores (tamanho do conjunto de treinamento) e o segundo é a dimensão do vetor de características.

CNN 1	**CNN 2**
Entrada (*batch_size*, *input_dim*)	
Conv1d 20-64	Conv1d 20-64
Activation ReLu	
Conv1d 20-32	Conv1d 20-32
Activation ReLu	
MaxPool	
Conv1d 20-16	Conv1d 20-4
Activation ReLu	
MaxPool	Flatten
Conv 20-8	Dense
Activation ReLu	Activation Softmax
MaxPool	
Conv 20-4	
Activation ReLu	
Flatten	
Dense	
Activation Softmax	

Tabela 1 – Tabela de comparação dos modelos CNN. Fonte: O autor (2020)

A CNN 1 tem duas camadas de convolução, duas de ReLu e duas de MaxPool a mais do que a CNN 2, além das camadas a mais, a terceira camada de convolução da CNN 2 aplica apenas 4 filtros, enquanto a terceira camada da CNN 1 aplica 16 filtros. O tamanho do *kernel* é o mesmo para as duas redes, vinte.

Abaixo, na tabela 2, são mostradas as duas RNN-GRU que apresentaram a melhor precisão na etapa de testes. Uma característica que pode ser destacada aqui é que na etapa de treinamento geralmente a precisão ficava abaixo do valor quando comparado aplicando o modelo ao conjunto de teste.

Foi aplicado dois filtros(*units*) diferentes, 256 na rede RNN-GRU 1 e 128 na rede RNN-GRU 2. Na rede RNN-GRU 1 foi dobrado o tamanho de *units*, seguido por uma camada completamente conectada e três camadas GRU de 256 *units*, terminando com uma camada completamente conectada do tamanho de classes a serem classificadas e uma função de ativação *softmax*. Já na RNN-GRU 2 a rede é bem menor, com uma camada completamente conectada de tamanho quatro, representando o número de classes para classificação e uma camada *softmax*.

RNN-GRU 1	**RNN-GRU 2**
Entrada (*dim*, *input_ dim*)	
GRU 256	GRU 128
GRU 512	Dense
Dense	Activation Softmax
GRU 256	
GRU 256	
GRU 256	
Dense	
Activation Softmax	

Tabela 2 – Tabela de comparação dos modelos RNN-GRU. Fonte: O autor (2020)

4.6.1 CONFIGURAÇÕES

Em todos os treinamentos foi utilizado um padrão para monitorar as métricas de perda e precisão, evitando assim, desperdiçar tempo aguardando um modelo que não apresenta evolução na taxa de aprendizagem. Exitem algumas técnicas que são utilizadas visando agilizar a conclusão de um treinamento, principalmente de modelos que não estão performando bem. Por isso, apresento as configurações que foram utilizadas em todos os treinamentos.

- ReduceLROnPlateau: reduz a taxa de aprendizado quando atingir um platô. A métrica monitorada para identificar um platô é a perda(*loss*) do conjunto de validação, um fator aplicado de 0.9 com uma paciência(*patience*) de 5, significa que aguarda 5 épocas sem melhora até que a taxa de aprendizagem seja reduzida aplicando o fator, como pode ser visto na figura 22.

- EarlyStopping: interrompe o treinamento quando uma determinada métrica não é atingida. A métrica de desempenho do modelo é a perda do conjunto de validação, com uma paciência de 50 épocas, onde o objetivo é a menor perda, como pode ser visto na figura 22.

- ModelCheckPoint: o melhor modelo treinado, que apresentou a menor taxa de perda para o conjunto de validação após todas as épocas é salvo para que possa ser aplicado no conjunto de testes.

- Epoch: é o número de vezes que os dados passam por todas as camadas da rede. O número de épocas foi atribuído um valor alto já que a função de monitoração EarlyStopping interrompe o modelo quando não há alteração.

- LearningRate: a taxa de aprendizado indica qual a velocidade de atualização dos pesos durante o treinamento da rede, indicando que quando ocorre uma oscilação significativa pode ser necessário reduzir a taxa, entretanto isso pode manter a

conversão do modelo mais lenta. Portanto, o ideal aqui é encontrar uma taxa que seja a maior possível sem grandes oscilações.

- Dropout: aplica-se um fator que remove os nós da rede aleatoriamente, evitando *overfitting*.
- Optimizer: existem alguns algoritmos que otimizam a conversão do modelo. Alguns dos mais utilizados são *Adam*[7], *RMSprop*[8] e *SGD*[9], que vão minimizar a função de perda em relação aos parâmetros da rede.

Como mencionado nos tópicos acima, a figura 22 demonstra algumas das configurações apresentadas de forma que seja possível visualizar cada configuração em ação. Além desse ponto, o gráfico trás pelo menos duas informações, perda e acurácia, a primeira é utilizada para otimizar o modelo, sendo essa informação utilizada em cada iteração para fazer a atualização dos pesos do modelo. Já a acurácia é uma métrica de avaliação que será apresentada no próximo tópico, mas de maneira resumida apresenta o percentual de acerto do modelo sobre o conjunto de treino e de validação. O gráfico também apresenta as informações de treino e validação, onde uma parte do conjunto é separada para treinar o modelo e outra parte para fazer a validação de como o modelo se saiu sobre uma amostra que não foi utilizada durante o treino.

4.7 MÉTRICAS PARA AVALIAÇÃO DOS MODELOS

A qualidade dos modelos será avaliada conforme algumas métricas utilizadas para modelos de classificação, desta forma é possível identificar qual o modelo apresenta a maior capacidade de acerto. Cada modelo treinado e testado apresenta as duas principais informações para avaliação dos modelos, a primeira é acurácia, do inglês *accuracy*, que é o número total de acertos dividido pelo número total de amostras e a segunda é uma matriz de confusão que consolida por classe de classificação, melhorando a visualização e agilizando o processo de identificação da distribuição de acerto por classe. A figura 23 na primeira linha da classe *angry* teve uma taxa de acerto de 62.50%, enquanto a soma da primeira linha deve corresponder aos 100% que é o número total de amostras dessa única classe, sendo estendido as outras classes o mesmo raciocínio. As linhas na matriz de confusão representam a classe real e as colunas representam as classes preditas pelo modelo. Para facilitar a leitura do gráfico foi optado por apresentar a porcentagem dentro de cada classe de emoção.

[7] https://keras.io/api/optimizers/adam/
[8] https://keras.io/api/optimizers/rmsprop/
[9] https://keras.io/api/optimizers/sgd/

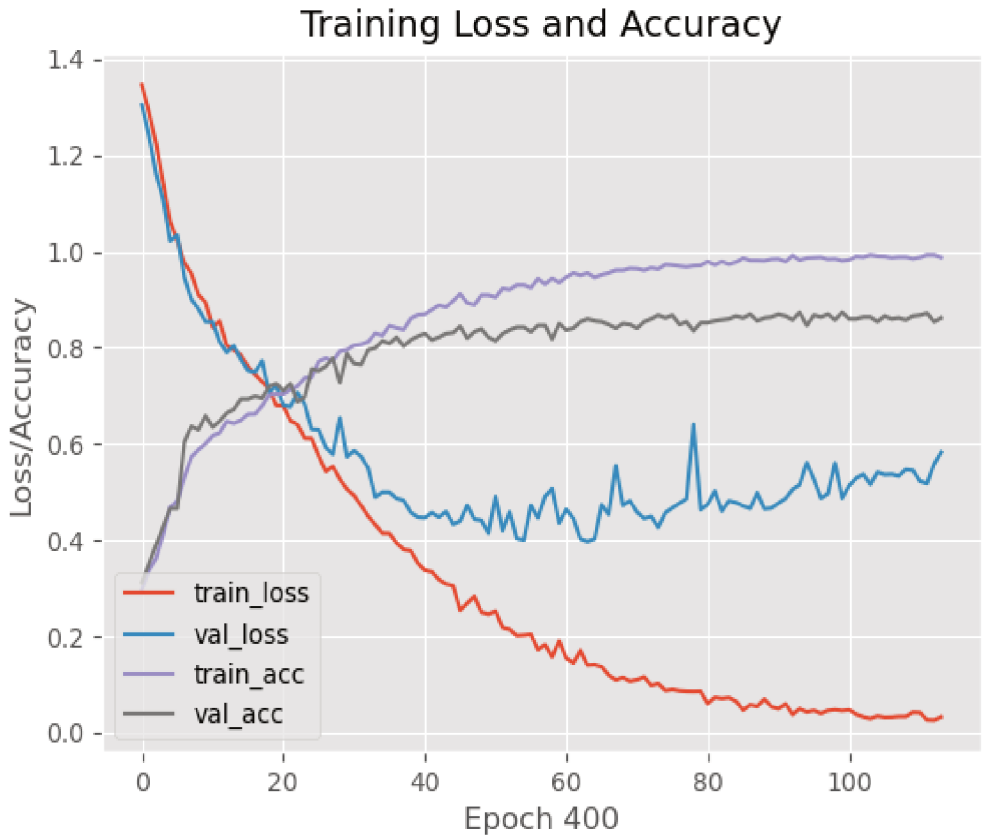


Figura 22 – Demonstração das configurações *ReduceLROnPlateau* e *EarlyStopping*. Fonte: O autor (2020)

4.8 CONJUNTO DE TREINAMENTO

O conjunto de treinamento apresentado na seção 4.1, foi dividido entre treinamento e validação, onde 75% foi utilizado para treinamento e 25% para validação. Para o conjunto de teste foi separado os atores um e dois, o primeiro é masculino e o segundo é feminino, garantindo que o modelo não teve contato com esses áudios anteriormente.

O método de aumento de dados apresentando também foi mensurado, por isso não existe uma quantidade padrão no conjunto de treinamento e validação, o que se mantém é o conjunto de teste com dois atores. Nos testes foi possível identificar que as arquiteturas apresentam desempenho diferentes conforme as operações aplicadas são alteradas.

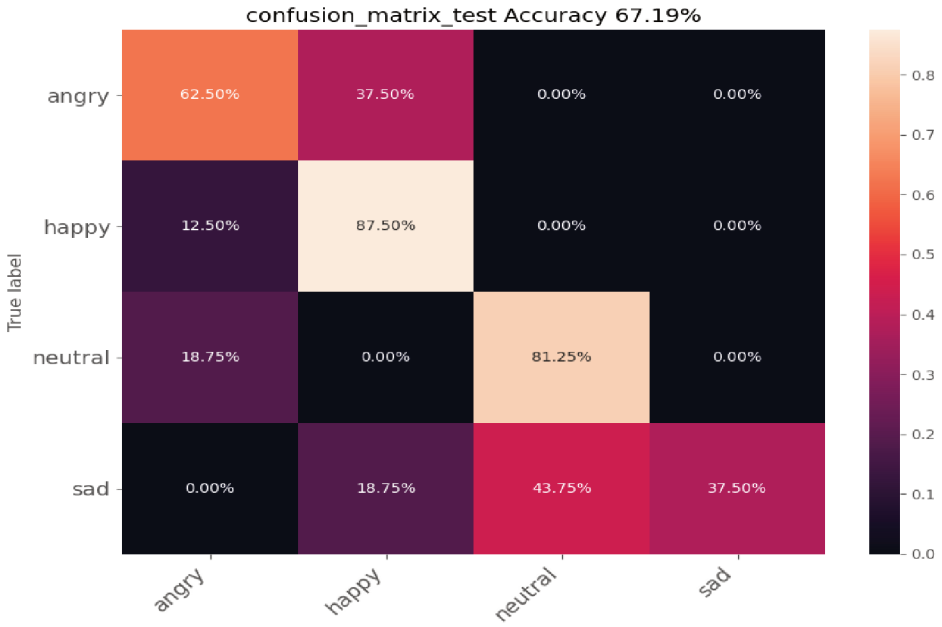


Figura 23 – Demonstração das métricas de avaliação - Fonte: O autor (2020)

5 IMPLEMENTAÇÃO

A implementação realizada durante o desenvolvimento do projeto é apresentada nesta seção, assim como as informações relevantes sobre os parâmetros utilizados e características de cada arquitetura que por ventura ainda não foram demonstradas.

O código é apresentado na seguinte ordem:

- conjunto para predição
- extrator de características
- construir modelo
- treinar modelo
- testar modelo

O *framework* Pandas[10] foi utilizado para indexar as informações do conjunto de áudios dentro de um *dataframe*.

Para testar o modelo treinado, ou seja, fazer a predição de um conjunto que a rede neural não conhece, foi separado dois atores conforme algoritmo abaixo:

Algoritmo 1 - Conjunto para predição

```
def split_dataset_for_predict(data_df):
    actor_1, actor_2 = 1, 2

    df_train = data_df.copy()
    df_train = df_train[df_train.label != "none"].reset_index(drop=True)
    df_train = df_train[df_train.actor != actor_1].reset_index(drop=True)
    df_train = df_train[df_train.actor != actor_2].reset_index(drop=True)

    test_df = data_df.copy()
    test_df = test_df[test_df.label != "none"].reset_index(drop=True)
    tmp1 = test_df[test_df.actor == actor_1]
    tmp2 = test_df[test_df.actor == actor_2]
    df_test = pd.concat([tmp1, tmp2], ignore_index=True)
    df_test.reset_index(drop=True)

    return df_train, df_test
```

[10] https://pandas.pydata.org/

O algoritmo para extrair característica utiliza a função librosa.feature.mfcc(), conforme detalhado na seção 2. O parâmetro *offset* significa que será desconsiderado o primeiro meio segundo e o *duration* considera os próximos 3 segundos. Essa padronização auxilia no treinamento da rede e descarta o inicio e o final dos áudios que têm silêncio, conforme apresentado na figura 21. A saída do algoritmo é um *dataframe* com o vetor de características com tamanho 20 (parâmetro *n_mfcc*) na coluna *feature*.

Algoritmo 2 - Extrator de características

```
def extract_features(data_df):
    data = pd.DataFrame(columns=['feature'])

    for i in tqdm(range(len(data_df))):
        audio, sr = librosa.load(data_df.path[i], sr=44100, offset=0.5,
            duration=3, res_type='kaiser_fast')
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=20, n_fft=1024,
            n_mels=128, fmax=8000)
        mfcc_feature = np.mean(mfcc.T, axis=0)
        data.loc[i] = [mfcc_feature]

    return data
```

O próximo passo é implementar os modelos que serão treinados pela rede neural, seja ela CNN ou RNN-GRU. Todos os modelos apresentados foram implementados usando a API do *Keras*, que facilitou a definição dos parâmetros de cada camada e agilizou o processo de testes de conversão da melhor rede. Tanto que, abaixo são demonstradas apenas quatro redes, entretanto, foram realizados testes com outras camadas até que essas fossem escolhidas.

Algoritmo 3.1 - Construção do modelo CNN 1

```
def build_cnn_5conv1d(x_traincnn):
    dropout = 0.2
    kernel_size = 20
    pool_size = 2
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=kernel_size, padding='same',
    input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(Conv1D(32, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
```

```
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(16, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(8, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(4, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(lr=0.001, decay=1e-6)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])

    return model, opt
```

Algoritmo 3.2 - Construção do modelo CNN 2

```
def build_cnn_3conv1d(x_traincnn):
    dropout = 0.5
    kernel_size = 20
    pool_size = 2
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=kernel_size, padding='same',
    input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(Conv1D(32, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(4, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(lr=0.001, decay=1e-6)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])
```

```
    return model, opt
```

A implementação de uma rede neural recorrente do tipo GRU assemelha-se muito com a rede CNN, a diferença fica na utilização de uma camada GRU ao invés de uma camada Conv1D. Assim como a CNN utilizando Conv1D, a camada GRU é adequada para dados com características temporais, portanto, adequadas para fluxos de áudios.

As RNNs apresentam dois novos parâmetros, o primeiro *recurrent_dropout* aplica *dropout* as unidades de uma rede recorrente e *return_sequences* retorna a última saída da sequencia. O parâmetro *return_sequences* na RNN-GRU 2 é *False* devido ser a única camada *GRU* presente nesse modelo. No modelo RNN-GRU 2 também foi alterado a função de otimização, de *Adam* para *RMSprop*.

Algoritmo 3.3 - Construção do modelo RNN-GRU 1

```
def build_rnn_5gru(x_traingru):
    dropout = 0.2
    rd = 0.1
    n_dim = 256
    input_shape = (x_traingru.shape[1], x_traingru.shape[2])

    model = Sequential()
    model.add(GRU(n_dim, input_shape=input_shape, dropout=dropout,
   recurrent_dropout=rd, return_sequences=True))
    model.add(GRU(n_dim * 2, dropout=dropout, recurrent_dropout=rd,
   return_sequences=True))
    model.add(Dense(n_dim))
    model.add(Dropout(dropout))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd,
   return_sequences=True))
    model.add(Dropout(dropout))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd,
   return_sequences=True))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd))
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.Adam(learning_rate=0.0001)
    model.summary()

    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
   metrics=['acc'])

    return model, opt
```

Algoritmo 3.4 - Construção do modelo RNN-GRU 2

```
def build_rnn_1gru(x_traingru):
    dropout = 0.2
    rd = 0.1
    n_dim = 128
    input_shape = (x_traingru.shape[1], x_traingru.shape[2])

    model = Sequential()
    model.add(GRU(n_dim, input_shape=input_shape, dropout=dropout,
   recurrent_dropout=rd, return_sequences=False))
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=0.0)
    model.summary()

    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
   metrics=['acc'])

    return model, opt
```

O próximo algoritmo vai realizar o treinamento propriamente dito da rede, considerando os parâmetros de configuração apresentados na seção 4.6.1. O melhor modelo será salvo no arquivo *file_weights* que será utilizado na etapa de testes. A função model.fit() inicia o treinamento do modelo, utilizando 25% dos dados para validação de cada *epoch* considerando a lista de *callbacks*. A parada do treinamento ocorrerá quando atingir o número máximo de *epochs* ou a monitoração pelo *callbacks early_stopping*.

Algoritmo 4 - Treinamento do modelo

```
def train_model(x_traincnn, y_train, x_testcnn, y_test, file_weights):
    mcp_save = ModelCheckpoint(file_weights, save_best_only=True, monitor='
   val_loss', mode='min')
    lr_reduce = ReduceLROnPlateau(monitor='val_loss', patience=5, verbose
   =2, factor=0.9)
    early_stopping = EarlyStopping(monitor='val_loss', patience=50, verbose
   =1, mode='min')

    H = model.fit(x_traincnn, y_train, batch_size=64, epochs=400,
                  validation_data=(x_testcnn, y_test), callbacks=[mcp_save,
    lr_reduce, early_stopping])

    return H
```

Conforme mencionado acima, na etapa de treinamento o melhor modelo é salvo afim de utilizar para inferência das amostras de áudios dos dois atores (1 e 2), amostras essas que nunca foram utilizadas durante a etapa de treinamento. Além de salvar o melhor modelo durante o treinamento, o modelo foi serializado em um arquivo model.json, permitindo que não seja necessário construir novamente cada camada para realizar a predição das amostras de teste. Com esses dois arquivos salvos, é necessário apenas abrir o arquivo em formato *json* e carregar dentro dele o modelo salvo na etapa de treino.

Com o modelo pronto e carregado, é necessário extrair o vetor de características das amostras de teste, assim como foi realizado para a etapa de treinamento.

O algoritmo abaixo apresenta a utilização da função predict() do modelo passando o conjunto de testes.

Algoritmo 5 - Predição do conjunto de teste

```
def predict_test(dataset, loaded_model):
    preds = loaded_model.predict(dataset, batch_size=64, verbose=1)
```

6 EXPERIMENTOS E RESULTADOS

Nesta seção serão apresentados os experimentos realizados afim de buscar o melhor resultado dentro do conjunto de teste. A estratégia para realização dos experimentos foi primeiro fazer uma comparação entre cada rede neural convolucional, para isso, os testes foram conduzidos com um primeiro modelo de CNN até que alcançasse a melhor *accuracy*, após obter o primeiro modelo, foi realizado algumas alterações mais significativas no primeiro modelo, por exemplo, reduzir e aumentar o número de camadas completamente conectadas e *polling* até obter um modelo para realizar os testes comparando as duas redes CNN.

Para os experimentos com a RNN-GRU foi aplicada a mesma estratégia da CNN, com a construção de uma rede neural recorrente com a melhor *accuracy* e depois a alteração para uma nova RNN afim de obter duas redes para comparação.

Todos os modelos foram avaliados seguindo a métrica *accuracy*.

Conforme objetivo desde projeto, após os testes e resultados dentre as quatro redes neurais propostas, a que obteve o melhor desempenho foi utilizada para inferir o conjunto de amostras de áudios em português, conjunto esse criado pelo Jair da Rosa Júnior quando realizou seu trabalho de conclusão de curso. Esse conjunto foi utilizado devido a dificuldade de se obter um conjunto em português, e devido ser utilizado apenas para testes nos modelos não é necessariamente obrigatório um grande volume de amostras.

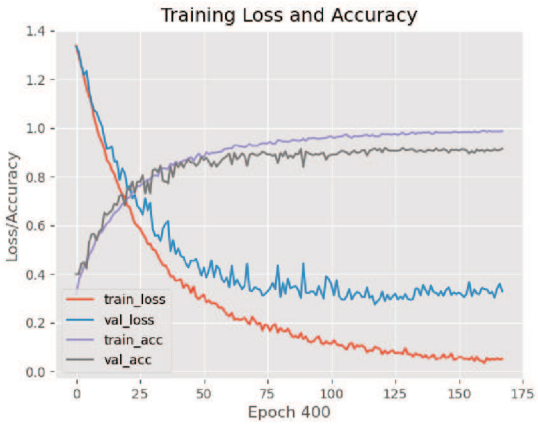
Com esses quatro modelos foram aplicados os experimentos descritos nas subseções abaixo.

6.1 EXPERIMENTOS CNN

Para a CNN 1, foram realizados testes aumentando o conjunto de treino/validação. Como pode-se observar na figura 24, foram aplicados testes sobre cada operação de transformação que aumentasse o conjunto de treino, entretanto, apenas aumentar o conjunto não significa que o resultado será melhor, pelos testes realizados com a CNN 1 a operação de transformação que teve o melhor desempenho foi alongamento(*stretch*), onde a taxa de *loss* e *accuracy* durante o processo de treinamento e validação foi a que teve o melhor desempenho.

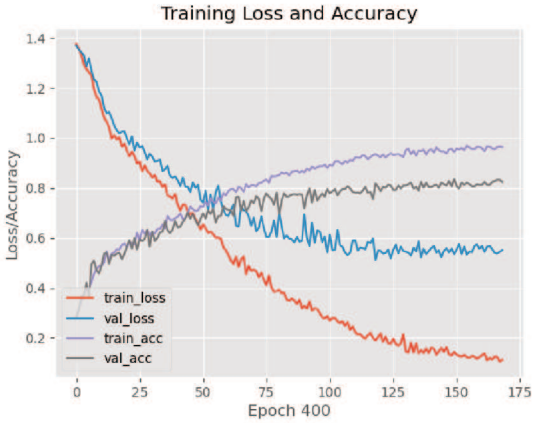
Já a figura 25 apresenta a matriz de confusão dos testes, como esperado, o melhor desempenho da operação *stretch* obteve uma *accuracy* de 67.19%. Além disso, precisaria de menos épocas.

(a) Sem aumento de dados



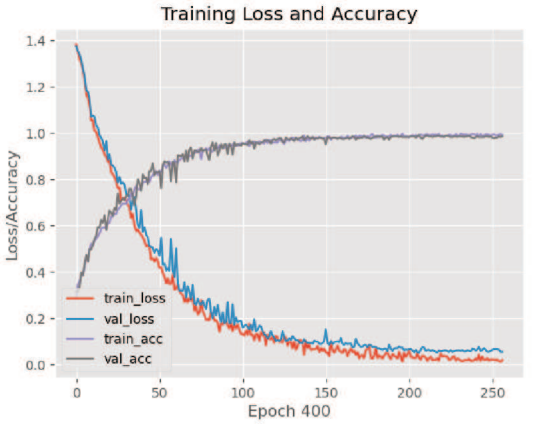
(b) Com todas as operações de aumento

(c) Com aumento de *noise*



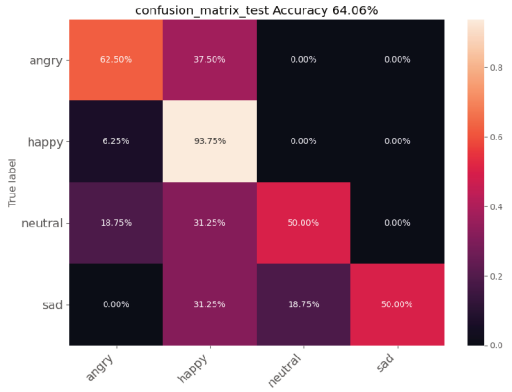
(d) Com aumento de *pitch*

(e) Com aumento de *speed_pitch*

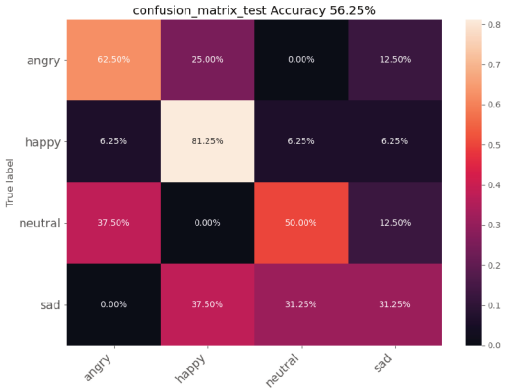


(f) Com aumento de *stretch*

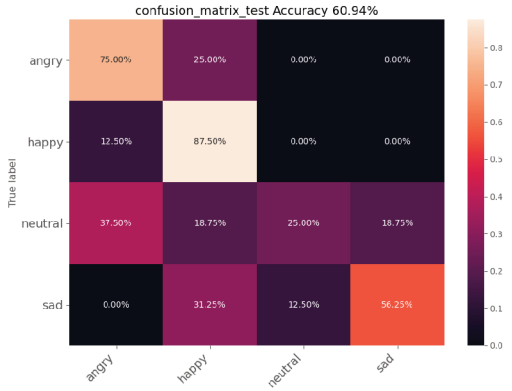
Figura 24 – CNN 1 - Comparação do desempenho das operações de transformação. Fonte: O autor (2020)



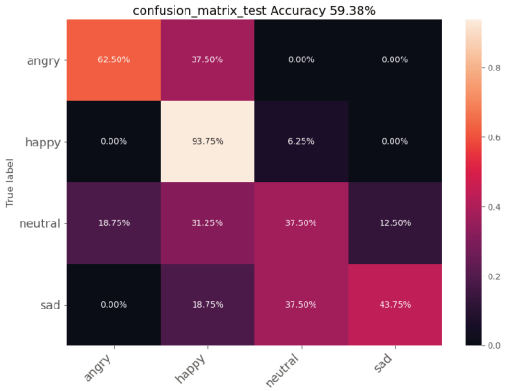
(a) Sem aumento de dados



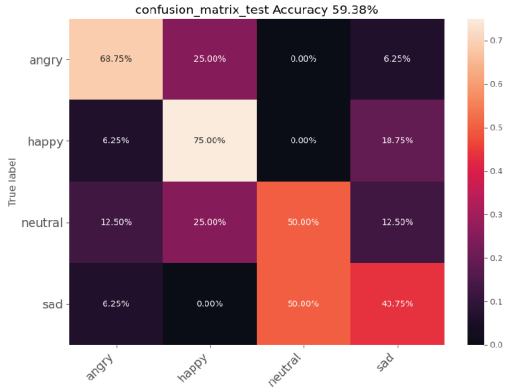
(b) Com todas as operações de aumento



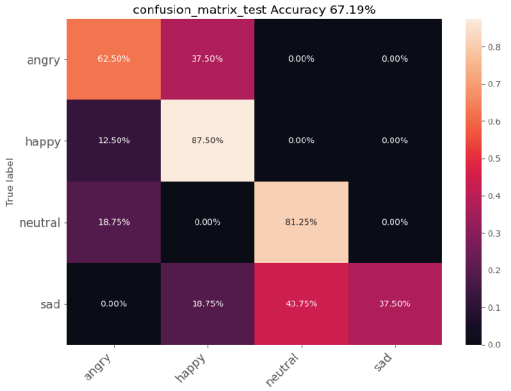
(c) Com aumento de *noise*



(d) Com aumento de *pitch*



(e) Com aumento de *speed_pitch*

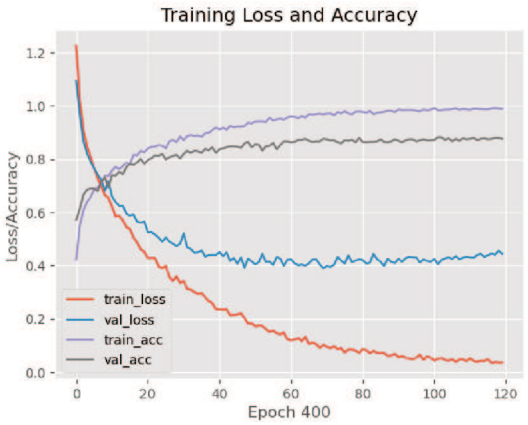


(f) Com aumento de *stretch*

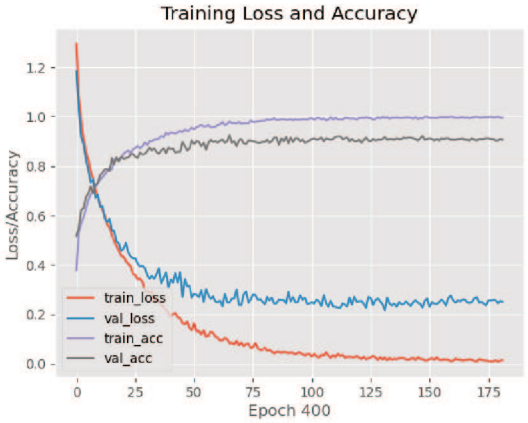
Figura 25 – CNN 1 - Matriz de confusão das operações de transformação. Fonte: O autor (2020)

Os mesmos testes foram aplicados para a rede CNN 2, e o desempenho foi melhor do que a CNN 1 em quase todas as operações, só não foi melhor na operação *stretch*. Outro ponto é o baixo desempenho usando a operação *noise*, por esse motivo, foi realizado mais um teste apenas com as operações *pitch* e *speed_pitch*. O teste dessas duas operações em conjunto foi o que teve o melhor desempenho entre todos, por isso foi adicionado na figura 26 e 27 (item b) no lugar de todas as operações. A *accuracy* foi de 73.44%, representando uma melhora de aproximadamente 6% em relação a CNN 1.

(a) Sem aumento de dados



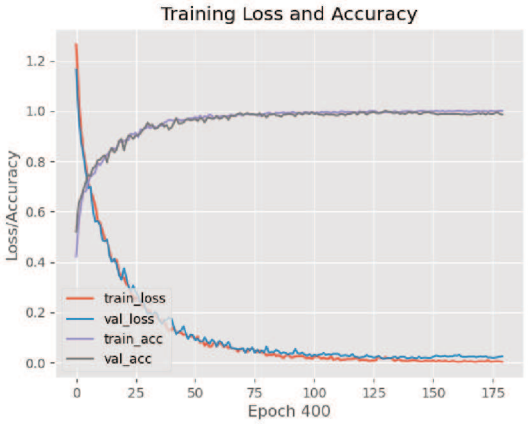
(b) Com as operações *pitch* e *speed_pitch*



(c) Com aumento de *noise*

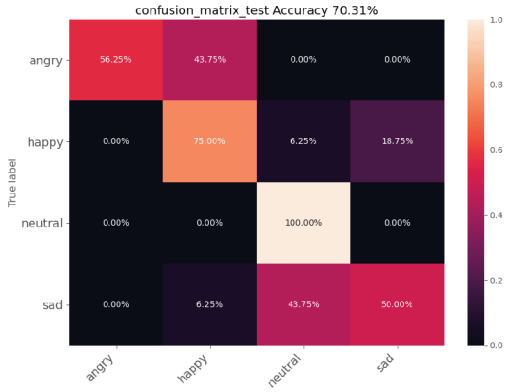
(d) Com aumento de *pitch*

(e) Com aumento de *speed_pitch*

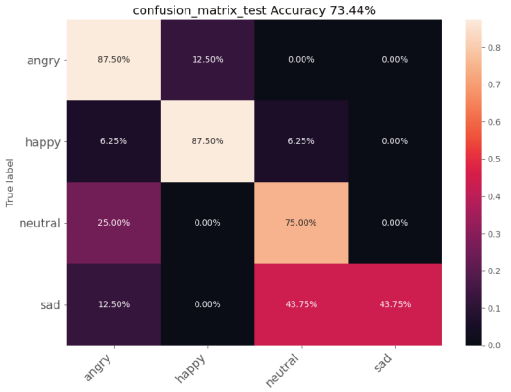


(f) Com aumento de *stretch*

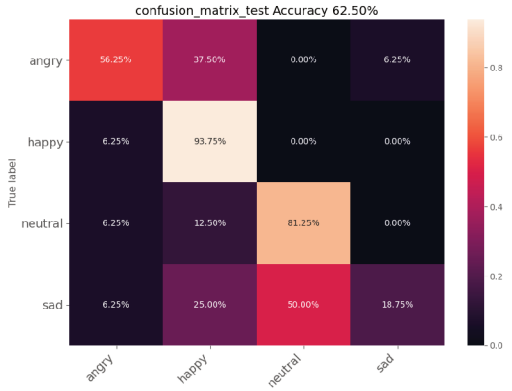
Figura 26 – CNN 2 - Comparação do desempenho das operações de transformação. Fonte: O autor (2020)



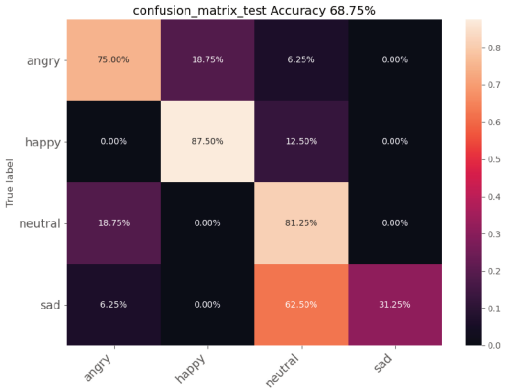
(a) Sem aumento de dados



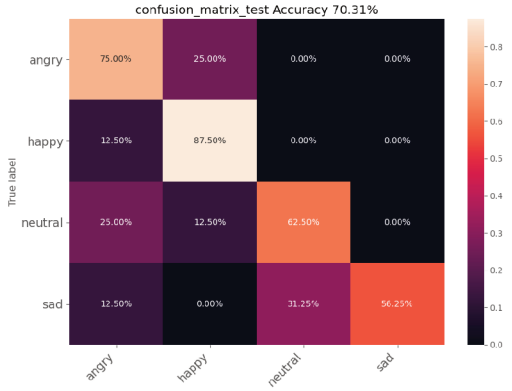
(b) Com as operações *pitch* e *speed_pitch*



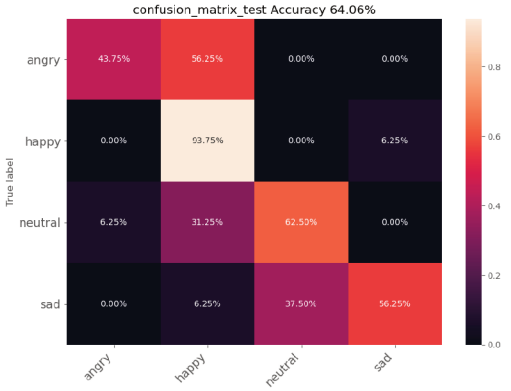
(c) Com aumento de *noise*



(d) Com aumento de *pitch*



(e) Com aumento de *speed_pitch*



(f) Com aumento de *stretch*

Figura 27 – CNN 2 - Matriz de confusão das operações de transformação. Fonte: O autor (2020)

6.2 EXPERIMENTOS RNN-GRU

Os testes de aumento do conjunto de treino/validação foi realizado também nas duas redes RNN. O objetivo é mostrar como esse fator influencia na etapa de testes e se o desempenho das operações é o mesmo das redes CNNs.

A rede RNN-GRU 1 de forma geral apresentou melhor desempenho do que as CNNs, principalmente o desempenho utilizando apenas a operação *stretch*, chegando a 79.69%. Outro ponto importante que deve ser destacado quando a matriz de confusão, figura 29, é analisada, é a taxa de 100% de acerto nas classes *happy* e *neutral*, também não se pode desprezar a taxa de 87.5% da classe *angry*. O ponto negativo é a assertividade quando analisado a classe *sad*, que em todos os testes foram bem abaixo das outras classes. Outra informação relevante é a diferença entre o conjunto de dados sem aumento, ou seja, conjunto original, que ficou com uma *accuracy* de 68.75% contra os 79.69% aumentando o conjunto de dados utilizando a operação *stretch*, um ganho de aproximadamente 11%. Para buscar uma melhor convergência do modelo, tentou-se aumentar a taxa de aprendizado, porém o resultado nas amostras de teste não foram satisfatórios, mesmo a função de perda apresentando um valor menor na etapa de treino.

Os testes utilizando a rede RNN-GRU 2, figuras 30 e 31, não apresentaram diferença aumentando o conjunto de testes/validação, permanecendo com os mesmo 79.69% da rede RNN-GRU 1. A diferença aqui ficou com o melhor desempenho do teste que não utilizou nenhum técnica de aumento de dados, com 71.88% e o pior desempenho utilizando todas as técnicas ao mesmo tempo, com 65.62%. Assim como foi realizado na rede RNN-GRU 1, na rede RNN-GRU 2 também foi aumentando o taxa de aprendizado, e o resultado foi insatisfatório como na rede anterior.

(a) Sem aumento de dados

(b) Com todas as operações de aumento

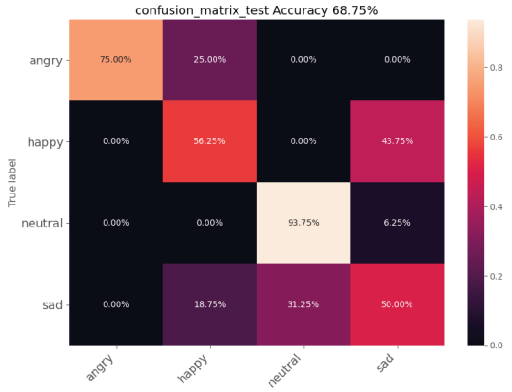
(c) Com aumento de *noise*

(d) Com aumento de *pitch*

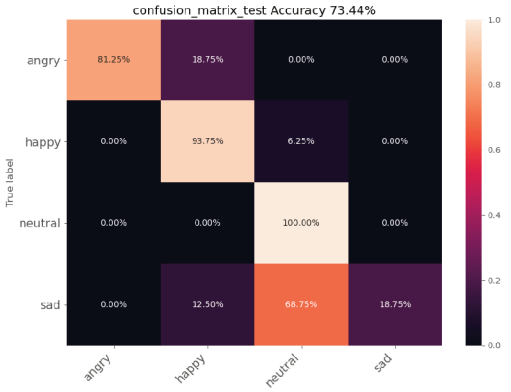
(e) Com aumento de *speed_pitch*

(f) Com aumento de *stretch*

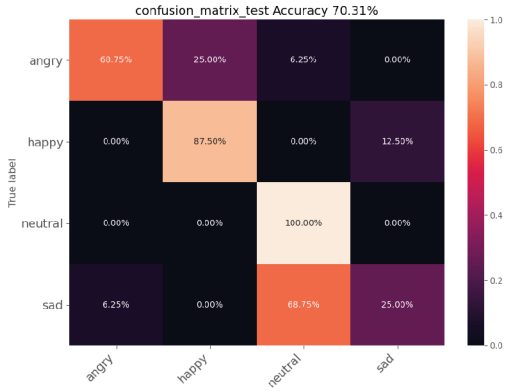
Figura 28 – RNN-GRU 1 - Comparação do desempenho das operações de transformação. Fonte: O autor (2020)



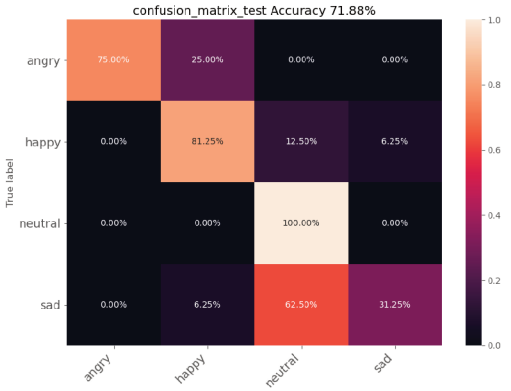
(a) Sem aumento de dados



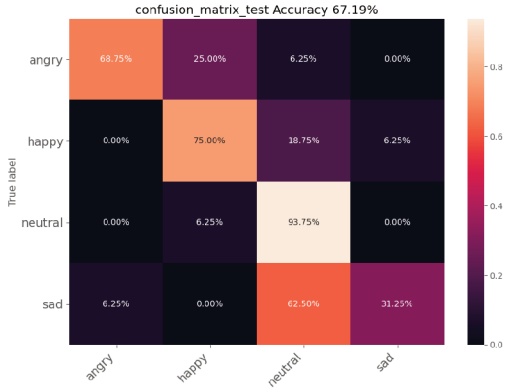
(b) Com todas as operações de aumento



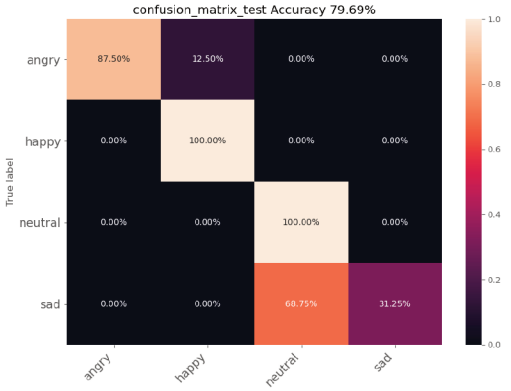
(c) Com aumento de *noise*



(d) Com aumento de *pitch*

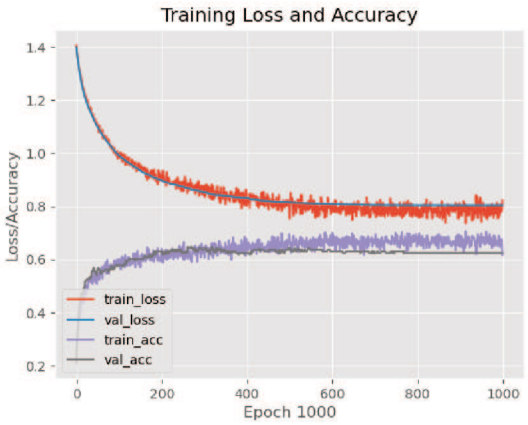


(e) Com aumento de *speed_pitch*

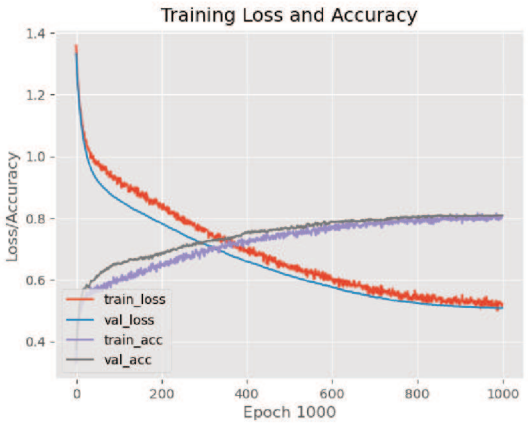


(f) Com aumento de *stretch*

Figura 29 – RNN-GRU 1 - Matriz de confusão das operações de transformação. Fonte: O autor (2020)



(a) Sem aumento de dados



(b) Com todas as operações de aumento

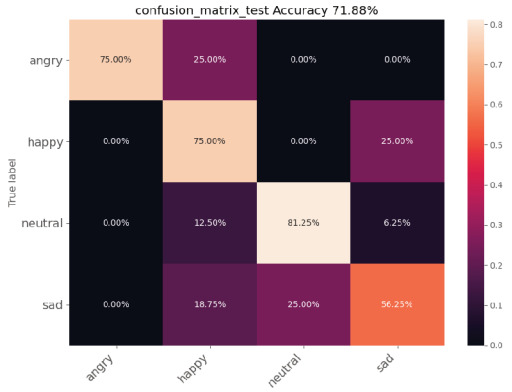
(c) Com aumento de *noise*

(d) Com aumento de *pitch*

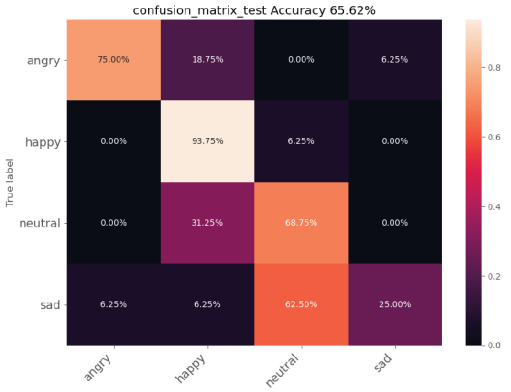
(e) Com aumento de *speed_pitch*

(f) Com aumento de *stretch*

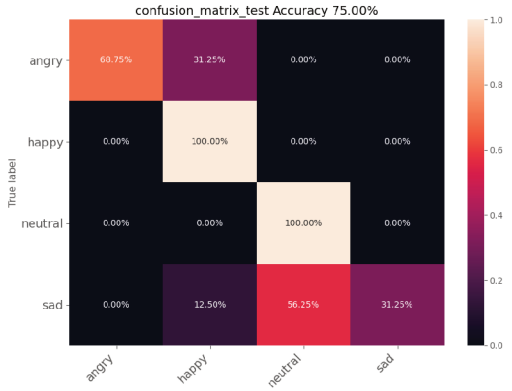
Figura 30 – RNN-GRU 2 - Comparação do desempenho das operações de transformação. Fonte: O autor (2020)



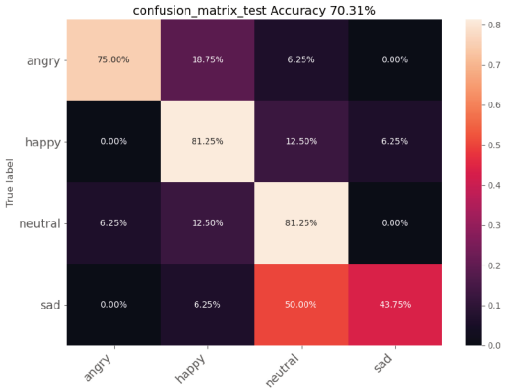
(a) Sem aumento de dados



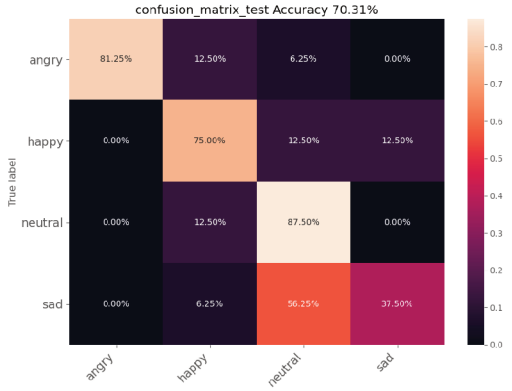
(b) Com todas as operações de aumento



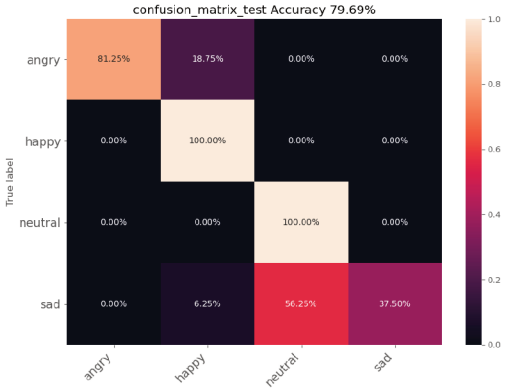
(c) Com aumento de *noise*



(d) Com aumento de *pitch*



(e) Com aumento de *speed_pitch*



(f) Com aumento de *stretch*

Figura 31 – RNN-GRU 2 - Matriz de confusão das operações de transformação. Fonte: O autor (2020)

6.3 TESTES COM CONJUNTO EM PORTUGUÊS

O modelo que apresentou o melhor desempenho foi utilizado para testar um conjunto de áudios em português do Brasil. Esse teste visa entender se a língua falada é um fator que deve ser levado em consideração quando aplicações de reconhecimento de emoções através da fala são desenvolvidas e se, é possível, utilizar um modelo treinado em uma língua predizer áudios de outra. Isso é importante já que os conjuntos de áudios para treino/validação devem ter um tamanho considerável e na sua grande maioria são na língua inglesa, portante, um modelo treinado e validado pode ser uma forma de pular a etapa de construção desses conjuntos, partindo diretamente para a etapa de predição.

Nesta seção, serão apresentados os resultados dos testes realizados junto ao conjunto de amostras em português. Para esse conjunto não foi aplicado nenhuma operação de transformação dos áudios para aumentar o conjunto de amostras, os testes foram realizados com o conjunto original.

Os modelos RNN-GRU 1 e 2 apresentaram a mesma acurácia durante a etapa de experimentos, e por isso os dois foram aplicados ao conjunto de amostras.

O modelo RNN-GRU 1, figura 32, obteve o melhor resultado se comparada com o modelo RNN-GRU 2, figura 33, entretanto o desempenho dos dois modelos não foi satisfatório. O primeiro atingiu uma acurácia de 42.73% e o segundo 40.91%, onde a classe com a maior taxa de acerto foi tristeza com 54.55%.

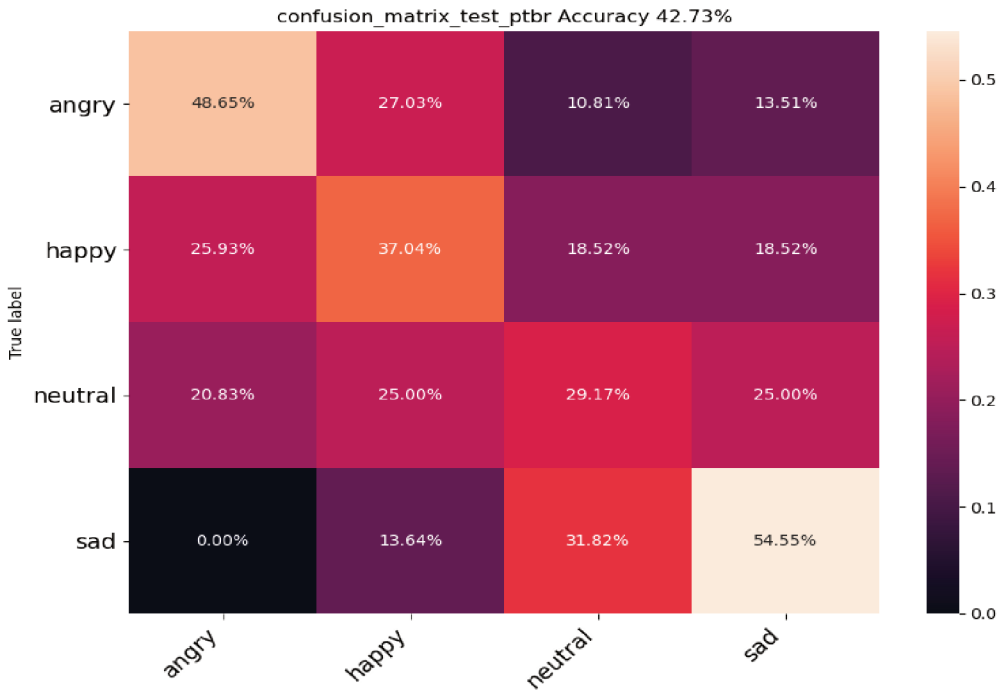


Figura 32 – RNN-GRU 1: Matriz de confusão amostras em Português. Fonte: O autor (2020)

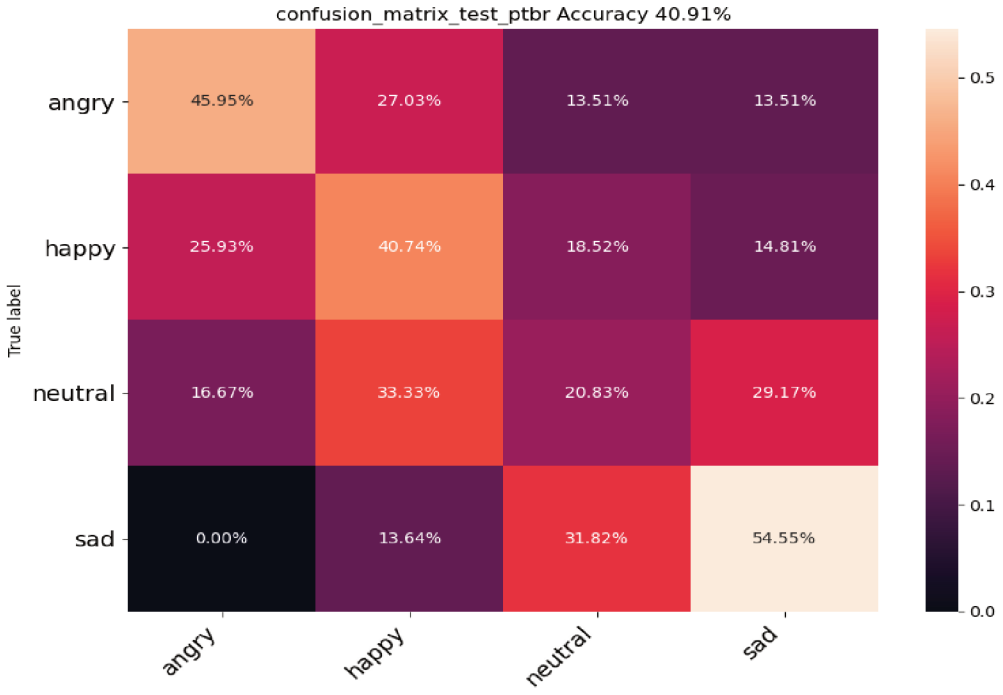


Figura 33 – RNN-GRU 2: Matriz de confusão amostras em Português. Fonte: O autor (2020)

6.4 RESULTADOS

Após todos os experimentos, observou-se que a diferença entre os modelos não foi tão significativa, ainda assim, é importante ressaltar a relevância do tipo RNN-GRU que nos testes alcançou um desempenho melhor se comparado com uma CNN.

Mesmo não tendo encontrado na literatura referências sobre a predição de emoções de uma língua utilizando modelos treinados sobre outras, acreditou-se que poderia encontrar resultados melhores, mas isso não foi comprovado. Importante destacar alguns pontos que também influenciaram para o baixo desempenho, sendo as características do conjunto de amostras em português, o viés da classificação sem validação e a falta do tratamento de ruído, os principais.

Basta lembrar que esse conjunto de amostras representa um contexto real, onde na prática seriam necessários mais de um modelo para fazer a classificação, por exemplo, um para analisar a semântica utilizando STT e processamento de linguagem natural, outro para analisar o espectro do áudio e por fim um modelo para consolidar essas informações e converter em apenas um resultado, sendo mais preciso por considerar fatores fisiológicos e semânticos.

A tabela abaixo procura mostrar o desempenho dos modelos descritos na etapa de trabalhos correlatos, mesmo sabendo que existem diferenças significativas entre esses modelos e o que foi proposto neste trabalho. Para facilitar, é mostrado na tabela as principais diferenças entre os modelos.

	Trabalho [Pandey, Shekhawat e Prasanna 2019]	**Trabalho [Júnior 2017]**	**Trabalho [Issa, Fatih Demirci e Yazici 2020]**	**Modelo proposto**
Acurácia	78.16%	94.3%	71.61%	79.69%
Modelo	CNN	SVM	CNN	RNN-GRU
Dataset	Emo-DB	Emo-DB	RAVDESS	RAVDESS
Classes	4	4	8	4

Tabela 3 – Tabela de comparação de desempenho com outros modelos. Fonte: O autor (2020)

7 CONCLUSÃO

Neste trabalho buscou-se reconhecer emoções através da fala utilizando duas das principais arquiteturas de redes neurais, que são aplicadas as diversas áreas do conhecimento. Tanto redes neurais convolucionais quanto redes neurais recorrentes foram apresentadas como opção e sua literatura foi revisada afim de entender suas especificidades e sua real aplicação dentro do problema proposto.

Através da revisão bibliográfica, foi identificado um tipo recente de rede neural recorrente com pouca pesquisa, principalmente sobre o tema reconhecimento de emoções através da fala. Diferente do tipo de rede neural, a técnica MFCC para extração de características é amplamente utilizada para sistemas automáticos de reconhecimento de emoções pela voz e que consegue captar melhor as intenções por trás da fala.

Ao longo do desenvolvimento, foi identificado que as técnicas para aumentar o tamanho do conjunto de dados para treinamento e validação influenciavam na precisão do modelo, caracterizando um ponto de atenção quando inicia-se a construção desse tipo de aplicação. Tendo em vista que as pesquisas relacionadas sobre reconhecimento de emoções não estão tão avançadas quanto a manipulação de imagens e ainda existe uma insuficiência relacionada a conjuntos de dados prontos para evoluir as pesquisas nessa área, foram expostas algumas técnicas para aumentar o conjunto de amostras e sua eficácia na melhoria do desempenho de cada modelo proposto.

A técnica de aumento de dados melhorou o desempenho da rede neural, mas não foram todas as operações de transformações do áudio que melhoraram o desempenho, a que apresentou o melhor resultado foi alongamento.

A diferença entre a voz masculina e a voz feminina não foi analisada, porém durante os experimentos foi identificado que a classe de emoção tristeza apresentou um baixo desempenho. Uma possível explicação pode estar relacionada com a mesma intensidade da voz da emoção neutra, conforme pode ser visto na figura 19. Na imagem, é possível identificar uma relação entre a intensidade da voz masculina e feminina nas classes neutra e tristeza, corroborando para o baixo desempenho na classe tristeza. Outro ponto que pode ser destacado é a pequena melhora da classe tristeza sem utilizar aumento de dados, conforme figura 31 do melhor resultado obtido, o que deve ser considerado um ponto de atenção.

Os testes realizados sobre o conjunto de amostras de áudios em português apresentaram um baixo desempenho e é relevante que novas pesquisas tenham como foco esse objetivo, já que utilizando apenas o método de extração MFCC não foram suficientes.

7.1 TRABALHOS FUTUROS

Para a realização de trabalhos futuros é importante validar o desempenho da rede GRU utilizando outros métodos de extração, assim como construir uma rede para analise semântica que possa ser utilizada em conjunto com a rede proposta, afim de extrair novos elementos e conclusões. Testar novos parâmetros da rede e do método de extração MFCC também podem ser realizados visando a continuidade deste trabalho.

Uma etapa que pode ser adicionada é diferenciar voz masculina e feminina, partindo para uma segmentação de dois modelos, um para masculino e outro para feminino. Isso criaria modelos mais específicos e menos ambíguos.

Agrupar os diferentes conjuntos de áudios, padronizando e balanceando suas classes para que possam treinar o modelo também é relevante. Testar esses conjuntos individualmente também pode trazer novos elementos que sirvam de base para novas pesquisas. Avaliar as diferentes operações de aumento de dados sobre os conjuntos, procurando identificar relações e padrões.

Explorar todos os pontos levantados acima para avaliar o conjunto de amostras em português. E não menos importante, criar um conjunto de áudios em português nos moldes dos que foram apresentados nesse trabalho.

Testar e avaliar o desempenho da rede sobre vários idiomas, identificando novas possibilidades. No contexto deste trabalho foi utilizado apenas um idioma como fonte de treinamento, o inglês, e um idioma como destino para ser classificado, o português, por isso é importante continuar na busca por um modelo que possa melhor representar o maior número possível de idiomas.

Referências

Badshah, A. M. et al. Speech emotion recognition from spectrograms with deep convolutional neural network. In: *2017 International Conference on Platform Technology and Service (PlatCon)*. [S.l.: s.n.], 2017. p. 1–5. Citado 2 vezes nas páginas 17 e 44.

Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, v. 5, n. 2, p. 157–166, March 1994. Citado na página 41.

BLUNSOM, P. *Recurrent Neural Networks and Language Modelling: Part 1*. 2017. Acesso em: 15 nov. 2019. Disponível em: <https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%203%20-%20Language%20Modelling%20and%20RNNs%20Part%201.pdf>. Citado 2 vezes nas páginas 17 e 40.

CHO, K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. Disponível em: <http://arxiv.org/abs/1406.1078>. Citado na página 41.

Davis, S.; Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, v. 28, n. 4, p. 357–366, August 1980. Citado na página 33.

DERTAT, A. *Applied Deep Learning - Part 4: Convolutional Neural Networks*. 2017. Acesso em: 14 nov. 2019. Disponível em: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>. Citado 2 vezes nas páginas 17 e 37.

FANT, G. Analys av de svenska konsonantljuden. *LM Ericsson protokoll H/P*, v. 1064, n. 194, p. 139, 1949. Citado na página 33.

FANT, G. Speech sounds and features. The MIT Press, 1973. Citado na página 33.

FERREIRA, A. dos S. *Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja*. Dissertação (Mestrado) — Universidade Federal de Mato Grosso do Sul, https://repositorio.ufms.br:8443/jspui/bitstream/123456789/3101/1/Redes2017. Acesso em: 14 nov. 2019. Citado 2 vezes nas páginas 17 e 38.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. In: GORDON, G.; DUNSON, D.; DUDíK, M. (Ed.). *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Fort Lauderdale, FL, USA: PMLR, 2011. (Proceedings of Machine Learning Research, v. 15), p. 315–323. Disponível em: <http://proceedings.mlr.press/v15/glorot11a.html>. Citado na página 36.

HAYKIN, S. S. Redes neurais artificiais: princípio e prática. *2ª Edição, Bookman, São Paulo, Brasil*, 2000. Citado 2 vezes nas páginas 17 e 35.

ISSA, D.; Fatih Demirci, M.; YAZICI, A. Speech emotion recognition with deep convolutional neural networks. *Biomedical Signal Processing and Control*, v. 59, p. 101894,

2020. ISSN 1746-8094. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1746809420300501>. Citado 3 vezes nas páginas 17, 47 e 81.

JUSLIN, P. N.; SCHERER, K. R. Speech emotion analysis. *Scholarpedia*, v. 3, n. 10, p. 4240, 2008. Citado na página 31.

JúNIOR, J. da R. *Reconhecimento automático de emoções através da voz*. 2017. Acesso em: 16 nov. 2019. Disponível em: <https://repositorio.ufsc.br/bitstream/handle/123456789/182186/reconhecimento_automatico_emocoes_voz_final_pdfa.pdf>. Citado 4 vezes nas páginas 17, 46, 50 e 81.

KOSTADINOV, S. *Understanding GRU Networks*. 2017. Acesso em: 16 nov. 2019. Disponível em: <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>. Citado 3 vezes nas páginas 17, 41 e 42.

LIVINGSTONE, S. R.; RUSSO, F. A. The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PLOS ONE*, Public Library of Science, v. 13, n. 5, p. 1–35, 05 2018. Disponível em: <https://doi.org/10.1371/journal.pone.0196391>. Citado 2 vezes nas páginas 47 e 49.

LOK, E. J.; KIN, C. E.; DEANE-MAYER, Z. *Audio Emotion Recognition: part 5 - data augmentation*. 2019. Acesso em: 05 Set. 2020. Disponível em: <https://www.kaggle.com/ejlok1/audio-emotion-part-5-data-augmentation>. Citado na página 54.

LYONS, J. *Mel Frequency Cepstral Coefficient (MFCC) tutorial*. 2013. Acesso em: 08 Novembro 2019. Disponível em: <http://www.practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/>. Citado 3 vezes nas páginas 17, 34 e 53.

Makhoul, J.; Cosell, L. Lpcw: An lpc vocoder with linear predictive spectral warping. In: *ICASSP '76. IEEE International Conference on Acoustics, Speech, and Signal Processing*. [S.l.: s.n.], 1976. v. 1, p. 466–469. Citado na página 33.

MCFEE, B. et al. librosa: Audio and music signal analysis in python. In: . [S.l.: s.n.], 2015. p. 18–24. Citado 3 vezes nas páginas 17, 50 e 51.

MICROSOFT, G. P. *CLOUD Computing: a nuvem está ao alcance de todos*. 2017. Acesso em: 25 out. 2020. Disponível em: <https://revistapegn.globo.com/Publicidade/Microsoft/noticia/2017/09/cloud-computing-nuvem-esta-ao-alcance-de-todos.html>. Citado na página 27.

Pandey, S. K.; Shekhawat, H. S.; Prasanna, S. R. M. Deep learning techniques for speech emotion recognition: A review. In: *2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA)*. [S.l.: s.n.], 2019. p. 1–6. Citado 4 vezes nas páginas 17, 26, 45 e 81.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, p. 533–536, 1986. Citado na página 36.

SCHERER, K. R. Vocal communication of emotion: A review of research paradigms. *Speech Communication*, v. 40, p. 227–256, 2003. Citado na página 31.

SCHULLER, B. et al. Recognising realistic emotions and affect in speech: State of the art and lessons learnt from the first challenge. *Speech Communication*, v. 53, p. 1062–1087, 2011. Citado na página 32.

SEARA, I. C. *Fonética Acústica*. Acesso em: 02 Novembro 2019. Disponível em: <https://moodle.ufsc.br/pluginfile.php/918950/mod_resource/content/1/Manual%20de%20Fon%C3%A9tica%20Ac%C3%BAstica.pdf>. Citado na página 32.

SILVA, W. d.; BARBOSA, P. A.; ABELIN, . A. Cross-cultural and cross-linguistic perception of authentic emotions through speech: An acoustic-phonetic study with Brazilian and Swedish listeners. *DELTA: DocumentaÃ§Ãde Estudos em LingÃÃstica TeÃe Aplicada*, scielo, v. 32, p. 449 – 480, 08 2016. ISSN 0102-4450. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-44502016000200449&nrm=iso>. Citado na página 27.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 06 2014. Citado 2 vezes nas páginas 17 e 40.

STEVENS, S. S.; VOLKMANN, J.; NEWMAN, E. B. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, v. 8, n. 3, p. 185–190, 1937. Disponível em: <https://doi.org/10.1121/1.1915893>. Citado na página 33.

THIAGO, E. R. de S. Reconhecimento de voz utilizando extração de coeficientes mel-cepstrais e redes neurais artificiais. 2017. Citado 2 vezes nas páginas 17 e 34.

Yu, Y. Research on speech recognition technology and its application. In: *2012 International Conference on Computer Science and Electronics Engineering*. [S.l.: s.n.], 2012. v. 1, p. 306–309. Citado na página 26.

Apêndices

APÊNDICE A – Fontes

```
import os
import keras
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import wave
from tqdm import tqdm
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.callbacks import ReduceLROnPlateau
from keras.models import Sequential, model_from_json
from keras.layers import Dense, GRU
from keras.layers import Flatten, Dropout, Activation
from keras.layers import Conv1D, MaxPooling1D
from keras.utils import np_utils
from keras import losses
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

config = {
    'sample_rate': 44100,
    'audio_duration': 3,
    'n_mels': 128,
    'n_fft': 1024,
    'n_mfcc': 20,
    'fmin': None,
    'fmax': 8000,
    'dct_type': 2,
    'n_classes': 4,
    'learning_rate': 0.0001,
    'max_epochs': 100,
    'batch_size': 64,
    'n_splits': 1,
    'datadir': 'Audio_Speech_Actors_01-24\\',
    'actors_tests': [1, 2],
    'dnn_type': 'rnn-gru'  # cnn / rnn-gru
}

def plot_time_series(data):
```

```
    fig = plt.figure(figsize=(14, 8))
    plt.title('Raw wave ')
    plt.ylabel('Amplitude')
    plt.plot(np.linspace(0, 1, len(data)), data)
    plt.show()


def noise(data):
    noise_amp = 0.025 * np.random.uniform() * np.amax(data)
    data = data.astype('float64') + noise_amp * np.random.normal(size=data.
    shape[0])
    return data


def shift(data):
    s_range = int(np.random.uniform(low=-5, high=5) * 500)
    return np.roll(data, s_range)


def stretch(data, rate=0.8):
    return librosa.effects.time_stretch(data, rate)


def pitch(data, sample_rate):
    pitch_pm = 2
    pitch_change = pitch_pm * 2 * (np.random.uniform())
    data = librosa.effects.pitch_shift(data.astype('float64'), sample_rate,
    n_steps=pitch_change)
    return data


def speed_n_pitch(data):
    length_change = np.random.uniform(low=0.8, high=1)
    speed_fac = 1.0 / length_change
    tmp = np.interp(np.arange(0, len(data), speed_fac), np.arange(0, len(
    data)), data)
    minlen = min(data.shape[0], tmp.shape[0])
    data *= 0
    data[0:minlen] = tmp[0:minlen]
    return data


def aug_neutral_with_stretch(audio, path_new_file):
    import soundfile as sf
    if not os.path.exists(path_new_file):
        audio, sr = sf.read(audio)
        audio = noise(audio)
```

```
        sf.write(path_new_file, audio, sr, 'PCM_16')


def create_data_frame():
    dir_list = os.listdir(config['datadir'])
    dir_list.sort()
    for i, dir_actor in enumerate(dir_list):
        file_list = os.listdir(config['datadir'] + dir_actor)
        for j, file in enumerate(file_list):
            nm = file.split('.')[0].split('-')
            path = config['datadir'] + dir_actor + '\\' + file
            if nm[2] == '01':
                new_file = file[:9] + '02' + file[11:]  # replace
   03-01-01-01-01-01-01 by 03-01-01-02-01-01-01
                path_new_file = config['datadir'] + dir_actor + '\\' +
   new_file
                aug_neutral_with_stretch(path, path_new_file)

    data_df = pd.DataFrame(
        columns=['path', 'source', 'actor', 'gender', 'intensity', '
   statement', 'repetition', 'emotion_id',
                 'emotion_name', 'n_frames', 'label'], dtype='float')

    dir_list = os.listdir(config['datadir'])
    dir_list.sort()
    count = 0
    for i in dir_list:
        file_list = os.listdir(config['datadir'] + i)
        for f in file_list:
            nm = f.split('.')[0].split('-')
            path = config['datadir'] + i + '\\' + f
            src = int(nm[1])
            actor = int(nm[-1])
            emotion = int(nm[2])

            label = nm[2]
            if label == '01':
                emotion_name = 'neutral'
                lb = 'neutral'
            elif label == '02':
                emotion_name = 'calm'
                lb = 'none'
            elif label == '03':
                emotion_name = 'happy'
                lb = 'happy'
            elif label == '04':
                emotion_name = 'sad'
```

```
            lb = 'sad'
        elif label == '05':
            emotion_name = 'angry'
            lb = 'angry'
        elif label == '06':
            emotion_name = 'fearful'
            lb = 'none'
        elif label == '07':
            emotion_name = 'disgust'
            lb = 'none'
        elif label == '08':
            emotion_name = 'surprised'
            lb = 'none'
        else:
            emotion_name = None
            lb = 'none'

        n_frames = wave.open(path).getnframes()

        if int(actor) % 2 == 0:
            gender = "female"
        else:
            gender = "male"

        if nm[3] == '01':
            intensity = 0
        else:
            intensity = 1

        if nm[4] == '01':
            statement = 0
        else:
            statement = 1

        if nm[5] == '01':
            repeat = 0
        else:
            repeat = 1

        data_df.loc[count] = [path, src, actor, gender, intensity,
statement, repeat, emotion, emotion_name,
                              n_frames, lb]
        count += 1

    return data_df
```

```
def aug_noise(syn_data1):
    for i in tqdm(range(len(data2_df))):
        audio_time_series, sr = librosa.load(data2_df.path[i], sr=config['
    sample_rate'], res_type='kaiser_fast',
                                             duration=config['
    audio_duration'], offset=0.5)
        audio_time_series = noise(audio_time_series)
        mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
    sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], # passado para
    melspectrogram
                                    n_mels=config['n_mels'], fmax=config['
    fmax']) # passado para mel
        mfcc_feature = np.mean(mfcc.T, axis=0)
        syn_data1.loc[i] = [mfcc_feature, data2_df.label[i]]


def aug_pitch(syn_data2):
    for i in tqdm(range(len(data2_df))):
        audio_time_series, sr = librosa.load(data2_df.path[i], sr=config['
    sample_rate'], res_type='kaiser_fast',
                                             duration=config['
    audio_duration'], offset=0.5)
        audio_time_series = pitch(audio_time_series, sr)
        mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
    sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], # passado para
    melspectrogram
                                    n_mels=config['n_mels'], fmax=config['
    fmax']) # passado para mel
        mfcc_feature = np.mean(mfcc.T, axis=0)
        syn_data2.loc[i] = [mfcc_feature, data2_df.label[i]]


def aug_speed_pitch(syn_data3):
    for i in tqdm(range(len(data2_df))):
        audio_time_series, sr = librosa.load(data2_df.path[i], sr=config['
    sample_rate'], res_type='kaiser_fast',
                                             duration=config['
    audio_duration'], offset=0.5)
        audio_time_series = speed_n_pitch(audio_time_series)
        mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
    sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], # passado para
    melspectrogram
                                    n_mels=config['n_mels'], fmax=config['
    fmax']) # passado para mel
```

```
        mfcc_feature = np.mean(mfcc.T, axis=0)
        syn_data3.loc[i] = [mfcc_feature, data2_df.label[i]]


def aug_stretch(syn_data5):
    for i in tqdm(range(len(data2_df))):
        audio_time_series, sr = librosa.load(data2_df.path[i], sr=config['
    sample_rate'], res_type='kaiser_fast',
                                             duration=config['
    audio_duration'], offset=0.5)
        audio_time_series = stretch(audio_time_series)
        mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
    sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], # passado para
    melspectrogram
                                    n_mels=config['n_mels'], fmax=config['
    fmax']) # passado para mel
        mfcc_feature = np.mean(mfcc.T, axis=0)
        syn_data5.loc[i] = [mfcc_feature, data2_df.label[i]]


def augmented_data(aug_type):
    aug_df = []

    if 'noise' in aug_type:
        syn_data1 = pd.DataFrame(columns=['feature', 'label'])
        aug_noise(syn_data1)
        syn_data1 = syn_data1.reset_index(drop=True)
        df4 = pd.DataFrame(syn_data1['feature'].values.tolist())
        labels4 = syn_data1.label
        syndf1 = pd.concat([df4, labels4], axis=1)
        syndf1 = syndf1.rename(index=str, columns={"0": "label"})
        syndf1 = syndf1.fillna(0)
        aug_df.append(syndf1)

    if 'pitch' in aug_type:
        syn_data2 = pd.DataFrame(columns=['feature', 'label'])
        aug_pitch(syn_data2)
        syn_data2 = syn_data2.reset_index(drop=True)
        df4 = pd.DataFrame(syn_data2['feature'].values.tolist())
        labels4 = syn_data2.label
        syndf2 = pd.concat([df4, labels4], axis=1)
        syndf2 = syndf2.rename(index=str, columns={"0": "label"})
        syndf2 = syndf2.fillna(0)
        aug_df.append(syndf2)

    if 'speed_pitch' in aug_type:
```

```
        syn_data3 = pd.DataFrame(columns=['feature', 'label'])
        aug_speed_pitch(syn_data3)
        syn_data3 = syn_data3.reset_index(drop=True)
        df4 = pd.DataFrame(syn_data3['feature'].values.tolist())
        labels4 = syn_data3.label
        syndf3 = pd.concat([df4, labels4], axis=1)
        syndf3 = syndf3.rename(index=str, columns={"0": "label"})
        syndf3 = syndf3.fillna(0)
        aug_df.append(syndf3)

    if 'stretch' in aug_type:
        syn_data5 = pd.DataFrame(columns=['feature', 'label'])
        aug_stretch(syn_data5)
        syn_data5 = syn_data5.reset_index(drop=True)
        df4 = pd.DataFrame(syn_data5['feature'].values.tolist())
        labels4 = syn_data5.label
        syndf5 = pd.concat([df4, labels4], axis=1)
        syndf5 = syndf5.rename(index=str, columns={"0": "label"})
        syndf5 = syndf5.fillna(0)
        aug_df.append(syndf5)

    return aug_df


def build_cnn_5conv1d(x_traincnn):
    dropout = 0.5
    kernel_size = 20
    pool_size = 2
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=kernel_size, padding='same',
    input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(Conv1D(32, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(16, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(8, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(4, kernel_size=kernel_size, padding='same'))
```

```
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(lr=0.001, decay=1e-6)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])

    return model, opt


def build_cnn_3conv1d(x_traincnn):
    dropout = 0.5
    kernel_size = 20
    pool_size = 2
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])

    model = Sequential()
    model.add(Conv1D(filters=64, kernel_size=kernel_size, padding='same',
    input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(Conv1D(32, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Dropout(dropout))
    model.add(MaxPooling1D(pool_size=pool_size))
    model.add(Conv1D(4, kernel_size=kernel_size, padding='same'))
    model.add(Activation('relu'))
    model.add(Flatten())
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(lr=0.001, decay=1e-6)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])

    return model, opt


def build_rnn_5gru(x_traincnn):
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])
    dropout = 0.2
    rd = 0.1
    n_dim = 256

    model = Sequential()
    model.add(GRU(n_dim, input_shape=input_shape, dropout=dropout,
```

```
    recurrent_dropout=rd, return_sequences=True))
    model.add(GRU(n_dim * 2, dropout=dropout, recurrent_dropout=rd,
    return_sequences=True))
    model.add(Dense(n_dim))
    model.add(Dropout(dropout))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd,
    return_sequences=True))
    model.add(Dropout(dropout))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd,
    return_sequences=True))
    model.add(GRU(n_dim, dropout=dropout, recurrent_dropout=rd))
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.Adam(learning_rate=0.0001)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])

    return model, opt


def build_rnn_1gru(x_traincnn):
    input_shape = (x_traincnn.shape[1], x_traincnn.shape[2])
    dropout = 0.2
    rd = 0.1
    n_dim = 128

    model = Sequential()
    model.add(GRU(n_dim, input_shape=input_shape, dropout=dropout,
    recurrent_dropout=rd, return_sequences=False))
    model.add(Dense(config['n_classes']))
    model.add(Activation('softmax'))
    opt = keras.optimizers.RMSprop(learning_rate=0.0001, decay=0.0)
    model.summary()
    model.compile(loss=losses.categorical_crossentropy, optimizer=opt,
    metrics=['acc'])

    return model, opt


def print_confusion_matrix(confusion_matrix, class_names, acc_score,
    graph_name, aug_type, figsize=(10, 7),
                           fontsize=14):
    confusion_matrix = confusion_matrix.astype('float') / confusion_matrix.
    sum(axis=1)[:, np.newaxis]
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
```

```
    )
    fig = plt.figure(figsize=figsize)
    heatmap = sns.heatmap(df_cm, annot=True, fmt='.2%')
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation
    =0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation
    =45, ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(f'{graph_name} Accuracy {acc_score:.2f}%')
    plt.savefig(f'tests/accuracy/{graph_name}_{acc_score:.2f}_{aug_type}.
    png')
    plt.show()


def predict_test(dataset, loaded_model, y_test, lb, dataset_type, aug_type)
    :
    preds = loaded_model.predict(dataset, batch_size=config['batch_size'],
    verbose=1)
    preds1 = preds.argmax(axis=1)

    abc = preds1.astype(int).flatten()
    predictions = (lb.inverse_transform((abc)))
    preddf = pd.DataFrame({'predictedvalues': predictions})
    actual = y_test.argmax(axis=1)
    abc123 = actual.astype(int).flatten()
    actualvalues = (lb.inverse_transform((abc123)))
    actualdf = pd.DataFrame({'actualvalues': actualvalues})

    finaldf = actualdf.join(preddf)
    finaldf.to_csv(f'tests/accuracy/predictions_{dataset_type}.csv', index=
    False)
    classes = finaldf.actualvalues.unique()
    classes.sort()

    y_true = finaldf.actualvalues
    y_pred = finaldf.predictedvalues
    acc_score = accuracy_score(y_true, y_pred) * 100
    print(f'\nacc_score: {acc_score:.2f}%')
    clf_report = classification_report(y_true, y_pred, output_dict=True)
    print(clf_report)
    sns.heatmap(pd.DataFrame(clf_report).iloc[:-1, :].T, annot=True, fmt='
    .2%')
    plt.title(f'report_{dataset_type}')
    plt.savefig(f'tests/accuracy/report_{dataset_type}_{acc_score:.2f}_{
    aug_type}.png')
    plt.show()
```

```
    cm = confusion_matrix(y_true, y_pred)
    print_confusion_matrix(cm, classes, acc_score, graph_name=f'
    confusion_matrix_{dataset_type}', aug_type=aug_type)


if __name__ == '__main__':
    data_df = create_data_frame()
    data2_df = data_df.copy()
    data2_df = data2_df[data2_df.label != "none"].reset_index(drop=True)
    data2_df = data2_df[data2_df.actor != config['actors_tests'][0]].
    reset_index(drop=True)
    data2_df = data2_df[data2_df.actor != config['actors_tests'][1]].
    reset_index(drop=True)

    data_test_df = data_df.copy()
    data_test_df = data_test_df[data_test_df.label != "none"].reset_index(
    drop=True)
    tmp1 = data_test_df[data_test_df.actor == config['actors_tests'][0]]
    tmp2 = data_test_df[data_test_df.actor == config['actors_tests'][1]]
    data_df_test = pd.concat([tmp1, tmp2], ignore_index=True).reset_index(
    drop=True)

    data = pd.DataFrame(columns=['feature'])
    for i in tqdm(range(len(data2_df))):
        audio_time_series, sr = librosa.load(data2_df.path[i], sr=config['
    sample_rate'], offset=0.5,
                                             duration=config['
    audio_duration'], res_type='kaiser_fast')
        mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
    sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], n_mels=config['
    n_mels'], fmax=config['fmax'])
        mfcc_feature = np.mean(mfcc.T, axis=0)
        data.loc[i] = [mfcc_feature]

    df3 = pd.DataFrame(data['feature'].values.tolist())
    labels = data2_df.label
    newdf = pd.concat([df3, labels], axis=1)
    rnewdf = newdf.rename(index=str, columns={"0": "label"})

    rnewdf = rnewdf.fillna(0)
    aug_type = 'none'
    aug_df = augmented_data(aug_type=[])
    # aug_df = augmented_data(aug_type=['stretch'])
    # aug_df = augmented_data(aug_type=['noise'])
    # aug_df = augmented_data(aug_type=['pitch'])
```

```
# aug_df = augmented_data(aug_type=['speed_pitch'])
# aug_df = augmented_data(aug_type=['noise', 'pitch'])
# aug_df = augmented_data(aug_type=['noise', 'speed_pitch'])
# aug_df = augmented_data(aug_type=['pitch', 'speed_pitch'])
# aug_df = augmented_data(aug_type=['noise', 'pitch', 'speed_pitch'])
# aug_df = augmented_data(aug_type=['noise', 'pitch', 'speed_pitch', '
stretch'])
# aug_df = augmented_data(aug_type=['stretch', 'speed_pitch'])
# aug_df = augmented_data(aug_type=['pitch', 'speed_pitch', 'stretch'])
# aug_df = augmented_data(aug_type=['pitch', 'speed_pitch'])

if aug_df:
    aug_df.append(rnewdf)
    combined_df = pd.concat(aug_df, ignore_index=True, sort=False)
else:
    combined_df = pd.concat([rnewdf], ignore_index=True, sort=False)

combined_df = combined_df.fillna(0)
X = combined_df.drop(['label'], axis=1)
y = combined_df.label

if config['dnn_type'] == 'rnn-gru':
    X = X.values.reshape((X.shape[0], 1, X.shape[1]))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
=0.25, random_state=42)

mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
lb = LabelEncoder()
y_train = np_utils.to_categorical(lb.fit_transform(y_train))
y_test = np_utils.to_categorical(lb.fit_transform(y_test))

if config['dnn_type'] == 'rnn-gru':
    x_traincnn = X_train
    x_testcnn = X_test
else:
    x_traincnn = np.expand_dims(X_train, axis=2)
    x_testcnn = np.expand_dims(X_test, axis=2)

model, opt = build_rnn_1gru(x_traincnn)
```

```
file_weights = 'tests/models/best_train.h5'
mcp_save = ModelCheckpoint(file_weights, save_best_only=True, monitor='
val_loss', mode='min')
lr_reduce = ReduceLROnPlateau(monitor='val_loss', patience=5, verbose
=2, factor=0.9)
early_stopping = EarlyStopping(monitor='val_loss', patience=50, verbose
=1, mode='min')

H = model.fit(x_traincnn, y_train, batch_size=config['batch_size'],
epochs=config['max_epochs'],
              validation_data=(x_testcnn, y_test), callbacks=[mcp_save,
 lr_reduce, early_stopping])

score = model.evaluate(x_testcnn, y_test, verbose=0)
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, len(H.history["loss"])), H.history["loss"], label
="train_loss")
plt.plot(np.arange(0, len(H.history["loss"])), H.history["val_loss"],
label="val_loss")
plt.plot(np.arange(0, len(H.history["loss"])), H.history["acc"], label=
"train_acc")
plt.plot(np.arange(0, len(H.history["loss"])), H.history["val_acc"],
label="val_acc")
plt.xlabel("Epoch␣" + str(config['max_epochs']))
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower␣left")
plt.title(f"Training␣Loss␣and␣Accuracy")
plt.savefig(f'tests/accuracy/graph_train_{aug_type}.png')
plt.show()

file_model = 'tests/models/model.json'
with open(file_model, "w") as json_file:
    json_file.write(model.to_json())

json_file = open(file_model, 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
loaded_model.load_weights(file_weights)
predict_test(x_testcnn, loaded_model, y_test, lb, dataset_type='
validation', aug_type=aug_type)

data_test = pd.DataFrame(columns=['feature'])
for i in tqdm(range(len(data_df_test))):
    audio_time_series, sr = librosa.load(data_df_test.path[i], sr=
config['sample_rate'], res_type='kaiser_fast',
```

```
                                                duration=config['
audio_duration'], offset=0.5)
    mfcc = librosa.feature.mfcc(y=audio_time_series, sr=config['
sample_rate'], n_mfcc=config['n_mfcc'],
                                    n_fft=config['n_fft'], # passado para
melspectrogram
                                    n_mels=config['n_mels'], fmax=config['
fmax']) # passado para mel
    mfcc_feature = np.mean(mfcc.T, axis=0)
    data_test.loc[i] = [mfcc_feature]

test_valid = pd.DataFrame(data_test['feature'].values.tolist())

if config['dnn_type'] == 'rnn-gru':
    test_valid = test_valid.values.reshape((test_valid.shape[0], 1,
test_valid.shape[1]))

mean = np.mean(test_valid, axis=0)
std = np.std(test_valid, axis=0)
test_valid = (test_valid - mean) / std
test_valid = np.array(test_valid)
test_valid_lb = np.array(data_df_test.label)
lb_test = LabelEncoder()
test_valid_lb = np_utils.to_categorical(lb_test.fit_transform(
test_valid_lb))

if config['dnn_type'] == 'rnn-gru':
    test_valid = test_valid
else:
    test_valid = np.expand_dims(test_valid, axis=2)

predict_test(test_valid, loaded_model, test_valid_lb, lb_test,
dataset_type='test', aug_type=aug_type)
```

Reconhecimento de Emoções Através da Fala Utilizando Redes Neurais

Renato M. de Souza[1]

[1]Instituto de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Caixa Postal 476 – 88.040-900-Florianópolis – SC – Brasil

renato.souza@ufsc.br

Abstract. *This paper presents RNN-GRU and CNN to classify emotions through speech that had the best performance in the experiment stage. To train these models, the data set RAVDESS was used, allowing the construction of an environment to carry out the evaluation and tests. The evaluation of a model trained in the English language that recognizes audios in the Portuguese language showed an accuracy of approximately 42%, being considered unsatisfactory for a classifier. The main characteristics that were identified as responsible for the performance were, the characteristics of the sample set, the classification bias without validation and the lack of noise treatment. The neural network that presented the best precision was RNN-GRU with 79.69% using a technique to increase the size of the data set through the elongation transformation operation.*

Resumo. *Este trabalho apresenta a RNN-GRU e a CNN para classificar emoções através da fala que tiveram o melhor desempenho na etapa de experimentos. Para treinar esses modelos utilizou-se o conjunto de dados RAVDESS, permitindo construir um ambiente para fazer a avaliação e testes. A avaliação de um modelo treinado na língua inglesa que reconhece áudios da língua portuguesa apresentou uma precisão de aproximadamente 42%, sendo considerado insatisfatório para um classificador. As principais características que foram identificadas como responsável pelo desempenho foram, as características do conjunto de amostras, o viés da classificação sem validação e a falta do tratamento de ruído. A rede neural que apresentou a melhor precisão foi RNN-GRU com 79.69% utilizando uma técnica para aumentar o tamanho do conjunto de dados através da operação de transformação alongamento.*

1. Introdução

Sistemas por comando de voz estão ganhando maior visibilidade e maior atenção da indústria. O mercado automobilístico, com carros autônomos e o mercado de relacionamento com o cliente, com *bots* virtuais, são bons exemplos de mercados que estão apostando nessas tecnologias para criar diferencial competitivo. Assim, se tornou uma necessidade natural evoluir algoritmos e técnicas para aprendizagem de máquina.

Muito popular, os sistemas STT representam muito bem a evolução das técnicas de reconhecimento de padrões, como os sistemas *Alexa*[1] e *Siri*[2] que utilizam essa técnica

[1]https://developer.amazon.com/en-US/alexa
[2]https://www.apple.com/br/siri/

para criar produtos e melhorar a experiência do usuário. Se comparado com o STT, o reconhecimento de emoções através da fala ainda não está com pesquisas tão avançadas a ponto de ser utilizado em um produto. Por isso, buscar a evolução é garantir que a interação entre homem e máquina seja ainda mais natural, parte disso passa por entender a relevância da cultura e idioma do falante [SILVA et al. 2016] para discernir quanto afeta no aprendizado de máquina. Por entender que existem novos modelos de redes neurais que podem permitir avançar nos sistemas de reconhecimento de emoções atuais, é que está sendo proposto uma comparação entre duas arquiteturas de redes onde seja possível treinar e testar em línguas distintas. A motivação está intrinsecamente ligada ao benefício que sistemas de comunicação trarão para a sociedade, e vislumbrando uma troca de estímulos maior entre homem e máquina, a identificação de emoções é fundamental.

2. Fundamentação Teórica

A fundamentação teórica abordada neste artigo apresenta dois tópicos principais, o primeiro é o método para extração de características e o segundo é sobre as redes neurais.

2.1. Mel Frequency Cepstral Coefficients - MFCC

O MFCC é um método para extrair coeficientes de um espectro de áudio, permitindo criar um vetor de características. Introduzido inicialmente por Davis e Mermelstein na década de 1980 [Davis and Mermelstein 1980], fornecem uma melhor representação do sinal, já que exploram a frequência que os humanos ouvem.

Para que o processo de extração ocorra dentro de um padrão, o método utiliza um janelamento, são instantes de tempo que tem uma duração especifica, como por exemplo de 20ms, garantindo que cada janela de tempo terá esse tamanho de amostra para que seja realizada a analise da fala. Se esse janelamento for menor, pode não haver amostras suficientes, e se for muito grande pode haver muita informação dificultando o filtro.

Para encontrar as várias frequências contidas no espectro da fala, utiliza-se a escala Mel [Stevens et al. 1937], proposta para compreender como as frequências são analisadas pelo ouvido humano e, como relacionar a frequência de um tom percebida a uma real que foi medida, permitindo uma maior correspondência como a forma que se ouve, mesmo sabendo que os humanos são melhores na percepção de mudanças da fala em frequências menores.

2.2. Redes Neurais Artificiais

A RNA podem ser entendidas como uma forma de modelar um problema computacional, problemas que surgiram com o avanço da tecnologia, por isso houve a necessidade de criar uma abstração que permitisse representar o mundo real. Essa abstração foi inspirada nos neurônios biológicos e como esses estão estruturados no cérebro, tendo em vista que o grande volume aumenta a capacidade de processamento. Essa ideia iniciou com um modelo computacional para representar um neurônio, proposto por McCulloch e Pitts (1943). Depois Hebb (1949) propôs um modelo de aprendizado e por último veio o modelo perceptron, proposto por Rosemblatt (1957), que a partir de uma entrada com um determinado peso, uma regra de propagação e uma função de ativação obtêm-se uma saída. Com esses 3 estudos foi possível simular, mesmo que basicamente, o funcionamento de um neurônio.

Redes Neurais Convolucionais: Convolução é uma operação matemática que toma duas funções para gerar uma terceiro função que mede a soma do produto desses sinais ao longo da região submetida.

Redes Neurais Recorrentes: As RNN foram propostas para poder processar uma sequencia de dados de entrada, onde a informação anterior é utilizada como saída da rede neural. Então, para definir uma sequencial de entrada é necessário definir o tamanho que pode ter, que depende do problema à ser classificado. As RNN são apropriadas para processamento de áudio e vídeo, que a informação anterior é importante para a decisão da saída. Contudo, esse modelo apresenta alguns problemas por conta da dependências de longas sequencias. A lacuna de informação pode ser muito grande, o que degrada o aprendizado da rede fazendo com que a informação do erro seja perdida devido o método de treinamento utilizar o gradiente [Bengio et al. 1994]. Um gradiente muito pequeno faz com que os pesos do modelo não sejam atualizados, já um gradiente grande tornam a rede instável.

Gated Recurrent Unit - GRU: Rede proposta por [Cho et al. 2014], as redes GRU foram introduzidas para tratar o problema gradiente da RNN. Vista como uma simplificação da LSTM, ela reduz o número de portões utilizados. Com essa ideia, permite melhorar o desempenho e manter a capacidade de aprendizado de longas sequencias.

3. Desenvolvimento

O desenvolvimento apresenta quais os conjuntos de áudios utilizados, quais as estratégias sobre o conjunto e qual as arquiteturas propostas.

Conjunto Inglês: O conjunto de dados utilizado para treinamento é o RAVDESS, foi produzido como parte de uma pesquisa para desenvolvimento de um conjunto com arquivos de áudio, vídeo e audiovisual [Livingstone and Russo 2018]. Esse conjunto está distribuído em 24 atores profissionais, com 12 homens e 12 mulheres, 60 áudios por ator, totalizando 1440 arquivos, na língua inglesa norte americana, com as emoções calma, feliz, triste, raiva, medo, surpresa e nojo e onde cada emoção é reproduzida em dois níveis de intensidade, normal e forte.

Conjunto Português: Para a realização dos testes na língua portuguesa foi utilizado um conjunto de amostras criado a partir de partes de vídeos do YouTube[3], com a classificação manual de cada áudio para uma das quatro classes de emoção [da Rosa Júnior 2017]. Esse conjunto está distribuído em 37 áudios com a emoção raiva, 27 áudios feliz, 22 áudios triste e neutro são 24 áudios.

Data Augmentation: Existem algumas técnicas que podem melhorar o desempenho de uma rede neural de aprendizado supervisionado, e uma delas é aumentar o tamanho do conjunto de dados com base nos dados que se tem, ou seja, gera-se novos dados fazendo pequenas transformações nos dados existentes. Para aplicar transformações sobre o conjunto de dados que será utilizado para treinamento da rede neural, aplica-se operações como ruído(*noise*), deslocamento(*shift*), alongamento(*stretch*) e *pitch* [LOK et al. 2019]. Essas são as principais operações realizadas para expandir o conjunto de dados com novos áudios.

[3]https://www.youtube.com/

3.1. Arquiteturas Propostas

A proposta é comparar redes neurais convolucionais e redes neurais recorrentes tipo GRU para identificar qual arquitetura obtém melhor desempenho utilizando a métrica de precisão(*accuracy*). Com base nisso, abaixo serão apresentados os quatro modelos, dois de cada arquitetura, que serão treinados e avaliados. A escolha de cada uma foi com base em testes realizados sobre o conjunto de dados original e expandido, e que obtiveram a melhor acurácia. Os modelos da CNN foram construídos utilizando convolução 1D, mais adequado para análise de série temporal, conforme destacado na documentação da API do *Keras*[4].

CNN 1	**CNN 2**
Entrada (*batch_size*, *input_dim*)	
Conv1d 20-64	Conv1d 20-64
Activation ReLu	
Conv1d 20-32	Conv1d 20-32
Activation ReLu	
MaxPool	
Conv1d 20-16	Conv1d 20-4
Activation ReLu	
MaxPool	Flatten
Conv 20-8	Dense
Activation ReLu	Activation Softmax
MaxPool	
Conv 20-4	
Activation ReLu	
Flatten	
Dense	
Activation Softmax	

Table 1. Tabela de comparação dos modelos CNN. Fonte: O autor (2020)

RNN-GRU 1	**RNN-GRU 2**
Entrada (*dim*, *input_dim*)	
GRU 256	GRU 128
GRU 512	Dense
Dense	Activation Softmax
GRU 256	
GRU 256	
GRU 256	
Dense	
Activation Softmax	

Table 2. Tabela de comparação dos modelos RNN-GRU. Fonte: O autor (2020)

[4]https://keras.io/

4. Experimentos e Resultados

A estratégia para realização dos experimentos foi primeiro fazer uma comparação entre cada rede neural convolucional, para isso, os testes foram conduzidos com um primeiro modelo de CNN até que alcançasse a melhor *accuracy*, após obter o primeiro modelo, foi realizado algumas alterações mais significativas no primeiro modelo, por exemplo, reduzir e aumentar o número de camadas completamente conectadas e *polling* até obter um modelo para realizar os testes comparando as duas redes CNN. Para os experimentos com a RNN-GRU foi aplicada a mesma estratégia da CNN, com a construção de uma rede neural recorrente com a melhor *accuracy* e depois a alteração para uma nova RNN afim de obter duas redes para comparação. Conforme objetivo desde projeto, após os testes e resultados dentre as quatro redes neurais propostas, a que obteve o melhor desempenho foi utilizada para inferir o conjunto de amostras de áudios em português, conjunto esse criado pelo Jair da Rosa Júnior quando realizou seu trabalho de conclusão de curso. Esse conjunto foi utilizado devido a dificuldade de se obter um conjunto em português, e devido ser utilizado apenas para testes nos modelos não é necessariamente obrigatório um grande volume de amostras. Todos os modelos foram avaliados seguindo a métrica *accuracy*.

Resultados CNN: A rede CNN que teve o melhor desempenho foi a segunda, que apresentou uma acurácia de 73.44%. Um ponto importante foi a utilização das técnicas de aumento de dados utilizando as operações de transformação *pitch* e *speed_pitch*. Com essa técnica, a melhora foi de aproximadamente 6% a mais do que a CNN 1. A figura 1 apresenta o resultado na forma de uma matriz de confusão.

Resultados RNN-GRU: A rede RNN-GRU de forma geral apresentou melhor desempenho do que as CNNs, principalmente o desempenho utilizando apenas a operação *stretch*, chegando a 79.69%. Outro ponto importante que deve ser destacado analisando a matriz de confusão, figura 2, é a taxa de 100% de acerto nas classes *happy* e *neutral*, também não se pode desprezar a taxa de 87.5% da classe *angry*. O ponto negativo é a assertividade quando analisado a classe *sad*, que em todos os testes foram bem abaixo das outras classes. Outra informação relevante é a diferença entre o conjunto de dados sem aumento, ou seja, conjunto original, que ficou com uma acurácia de 68.75% contra os 79.69% aumentando o conjunto de dados utilizando a operação *stretch*, um ganho de aproximadamente 11%. A 2 apresenta a matriz de confusão da RNN 2, que apresentou o melhor resultado.

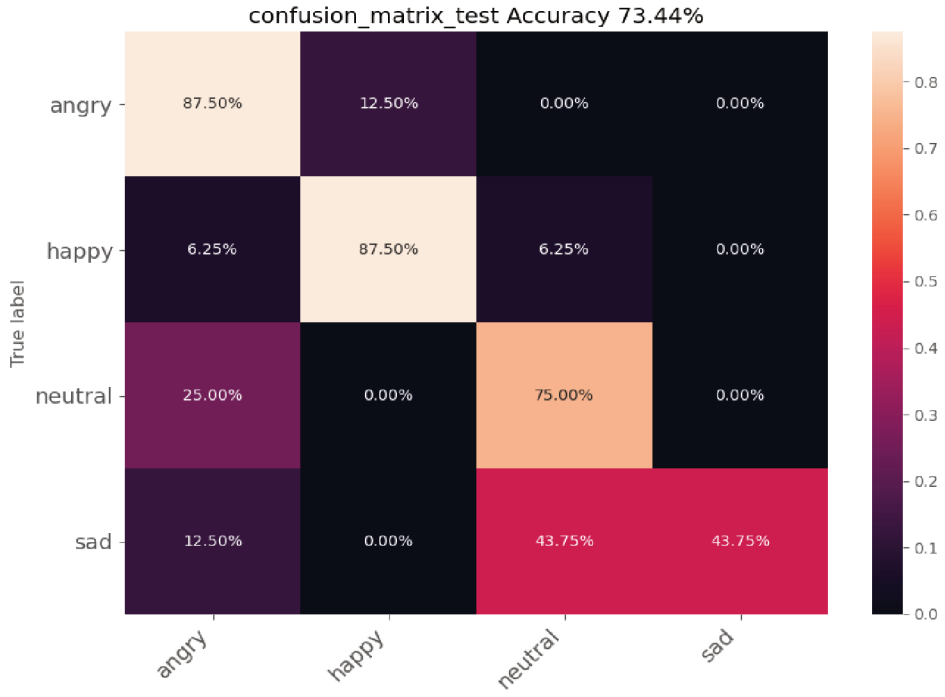


Figure 1. Matriz de confusão da CNN 2. Fonte: O autor (2020)

5. Testes Conjunto em Português

O modelo que apresentou o melhor desempenho foi utilizado para testar um conjunto de áudios em português do Brasil. Esse teste visa entender se a língua falada é um fator que deve ser levado em consideração quando aplicações de reconhecimento de emoções através da fala são desenvolvidas e se, é possível, utilizar um modelo treinado em uma língua predizer áudios de outra. Isso é importante já que os conjuntos de áudios para treino/validação devem ter um tamanho considerável e na sua grande maioria são na língua inglesa, portante, um modelo treinado e validado pode ser uma forma de pular a etapa de construção desses conjuntos, partindo diretamente para a etapa de predição. Para esse conjunto em português não foi aplicado nenhuma operação de transformação dos áudios para aumentar o conjunto de amostras, os testes foram realizados com o conjunto original. O modelo RNN-GRU 1, figura 3, obteve o melhor resultado se comparada com o modelo RNN-GRU 2, entretanto o desempenho dos dois modelos não foi satisfatório. O primeiro atingiu uma acurácia de 42.73% e o segundo 40.91%, onde a classe com a maior taxa de acerto foi tristeza com 54.55%.

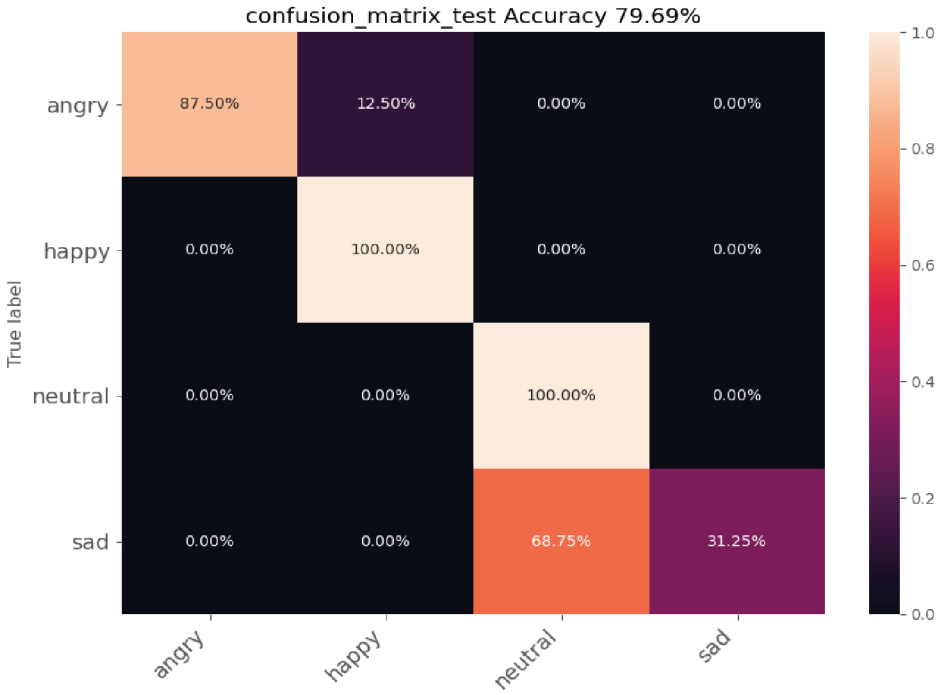


Figure 2. Matriz de confusão da RNN 1. Fonte: O autor (2020)

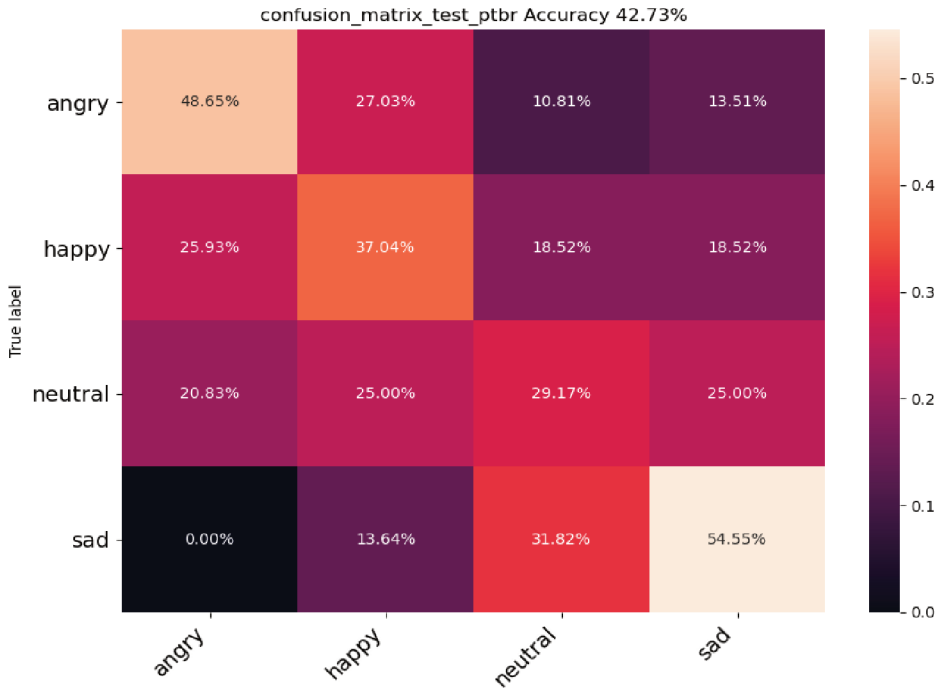


Figure 3. Matriz de confusão RNN 1 - Testes em Português. Fonte: O autor (2020)

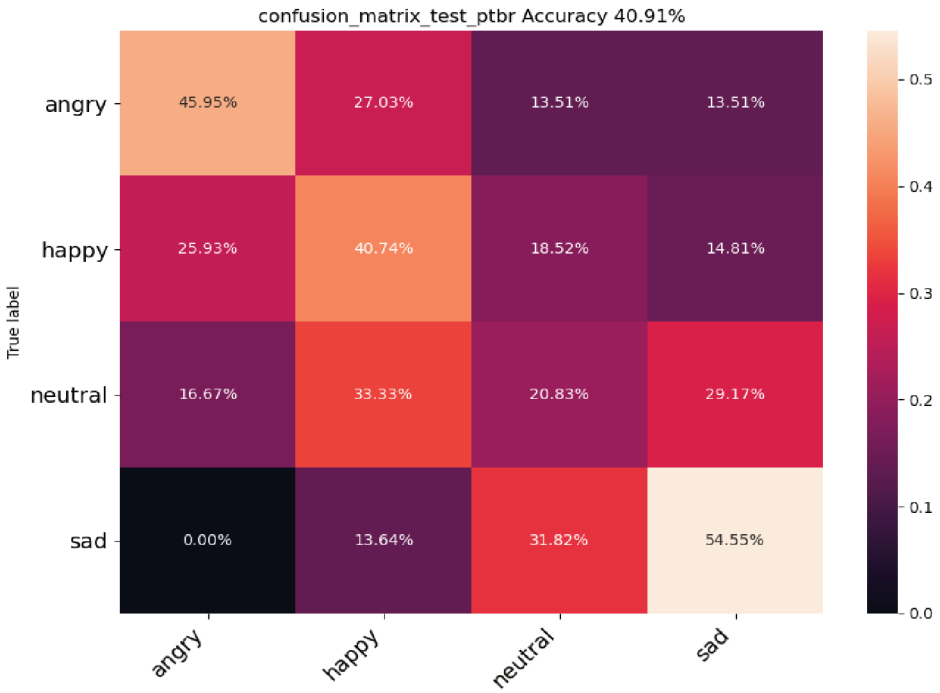


Figure 4. Matriz de confusão RNN 2 - Testes em Português. Fonte: O autor (2020)

6. Conclusão

Neste trabalho buscou-se reconhecer emoções através da fala utilizando duas das principais arquiteturas de redes neurais, que são aplicadas as diversas áreas do conhecimento. Tanto redes neurais convolucionais quanto redes neurais recorrentes foram apresentadas como opção e sua literatura foi revisada afim de entender suas especificidades e sua real aplicação dentro do problema proposto.

Ao longo do desenvolvimento, foi identificado que as técnicas para aumentar o tamanho do conjunto de dados para treinamento e validação influenciavam na precisão do modelo, caracterizando um ponto de atenção quando inicia-se a construção desse tipo de aplicação. Tendo em vista que as pesquisas relacionadas sobre reconhecimento de emoções não estão tão avançadas quanto a manipulação de imagens e ainda existe uma insuficiência relacionada a conjuntos de dados prontos para evoluir as pesquisas nessa área, foram expostas algumas técnicas para aumentar o conjunto de amostras e sua eficácia na melhoria do desempenho de cada modelo proposto. A técnica de aumento de dados melhorou o desempenho da rede neural, mas não foram todas as operações de transformações do áudio que melhoraram o desempenho, a que apresentou o melhor resultado foi alongamento.

Os testes realizados sobre o conjunto de amostras de áudios em português apresentaram um baixo desempenho e é relevante que novas pesquisas tenham como foco esse objetivo, já que utilizando apenas o método de extração MFCC não foram suficientes.

References

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

da Rosa Júnior, J. (2017). Reconhecimento automático de emoções através da voz. Acesso em: 16 nov. 2019.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366.

Livingstone, S. R. and Russo, F. A. (2018). The ryerson audio-visual database of emotional speech and song (ravdess): A dynamic, multimodal set of facial and vocal expressions in north american english. *PLOS ONE*, 13(5):1–35.

LOK, E. J., KIN, C. E., and DEANE-MAYER, Z. (2019). Audio emotion recognition: part 5 - data augmentation. Acesso em: 05 Set. 2020.

SILVA, W. d., BARBOSA, P. A., and ABELIN, . A. (2016). Cross-cultural and cross-linguistic perception of authentic emotions through speech: An acoustic-phonetic study with Brazilian and Swedish listeners. *DELTA: DocumentaÃ§Ãde Estudos em LingÃÃstica TeÃe Aplicada*, 32:449 – 480.

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190.