NANODEGREE ENGENHEIRO DE MACHINE LEARNING

Projeto Final

Marielen Marins Ferreira

30 de abril de 2018

Sumário

Capítulo 1

Definição

1.1 Visão geral do projeto

Existe um *podcast* chamado Mamilos [1] que está disponível nas plataformas iTunes, Spotify, entre outros. Os episódios sempre trazem assuntos polêmicos com o diferencial de colocar à mesa, pessoas especialistas no tema com diferentes pontos de vista, elevando o nível da discussão.

No episódio #58 [2] sobre acessibilidade foi lido um comentário de uma pessoa que colocou num *post* do *podcast*, o seguinte:

"Sou surdo, não sei como posso ouvir esse programa"[2]

Essa frase abalou a equipe e começaram a pensar em como tornar o *podcast* mais acessível para surdos. Assim, a partir desse programa decidiram criar uma rede de colaboradores para transcrever o áudio e não restringir mais a audiência que eles poderiam atingir.

1.2 Descrição do problema

No episódio #128 [3], que celebrava os três anos do *podcast* Mamilos [1], foi revelado diversas curiosidades sobre como era produzido cada episódio. Disseram que a cada cinco minutos de áudio demorava-se 30 para transcrever o conteúdo. Considerando um episódio do Mamilos de 90 minutos, levaria 540 para ser transcrito e mais 150 para ser revisado antes da publicação, segundo o episódio.

A equipe de transcrição, carinhosamente chamada de Mamilândia, já teve

mais de 70 pessoas e entregavam uma transcrição por semana nessa época. Entretanto, nos dias atuais, conta com um grupo de 22 voluntários que tentam encaixar essa tarefa em suas vidas atribuladas.

Sendo assim, o problema que será atacado por este projeto é a redução do tempo de transcrição dos episódios do Mamilos [1].

O conjunto de dados não foi retirado dos episódios do *podcast* Mamilos, pois para uma pesquisa inicial foi mais viável começar com uma base de dados já estruturada. Assim, a base foi retirada da competição *TensorFlow Speech Recognition Challenge* [4, 5] na plataforma kaggle [6].

O conjunto de áudios contém diversas pastas cujo nome era a palavra dita nos arquivos de áudio dentro dela. Escolhi trabalhar com todas, exceto *background noise* que não tem relações com as palavras ditas no áudio, apenas simular sons que não são produzidos em estúdio.

A solução será feita a partir do pré-processamento do áudio para digitalizar as ondas sonoras e do desenvolvimento do algoritmo para associar o áudio às palavras ditas.

O pré-processamento será feito utilizando *Mel Frequency Spectrum* [7], pois é um método simples e já existe biblioteca pronta na linguagem de programação python [8], que foi a escolhida para escrever o código por ser a do Nanodegree Engenheiro de Machine Learning da Udacity.

Após de ter assistido às aulas do módulo Deep Learning [9], achei interessante o uso de Redes Neurais Convolucionais (CNN) como algoritmo para reconhecimento de imagens e resolvi aplicar no meu projeto. Sendo assim, após o pré-processamento dos áudios transformo os dados que são vetores em matrizes para a aplicação de CNN's.

1.3 Métricas

Utilizarei a acurácia como métrica de avaliação por acreditar que é a mais adequada ao problema. Ela é calculada através da taxa de acerto do modelo de *machine learning* em uma base de dados que não foi utilizada para o treinamento do mesmo.

Capítulo 2

Análise

2.1 Exploração dos dados

A base de dados contém subpastas com comandos de voz de um segundo, com o nome da pasta sendo a palavra dita no clipe de áudio. As palavras no conjunto de áudio são: *bed*, *bird*, *cat*, *dog*, *down*, *eight*, *five*, *four*, *go*, *happy*, *house*, *left*, *marvin*, *nine*, *no*, *off*, *on*, *one*, *right*, *seven*, *sheila*, *six*, *stop*, *three*, *two*, *up*, *wow*, *yes* e *zero*. Escolhi trabalhar com todas, exceto a subpasta *background noise* que não tem relação com as palavras ditas no áudio, apenas simular sons que não são produzidos em estúdio.

Como são muitas palavras, decidi escolher aleatoriamente três classes que eu iria focar a minha análise. São elas: *bed*, *cat* e *happy*.

Defini arbitrariamente um arquivo de áudio de cada classe para ouvir. São eles:

Clique e ouça bed/00176480_nohash_0.wav

Clique e ouça cat/00f0204f_nohash_0.wav

Clique e ouça happy/0ac15fe9_nohash_0.wav

2.2 Visualização exploratória

Nas imagens 2.1, 2.2 e 2.3 estão plotadas as ondas sonoras da reprodução das palavras *bed*, *cat* e *happy* respectivamente.

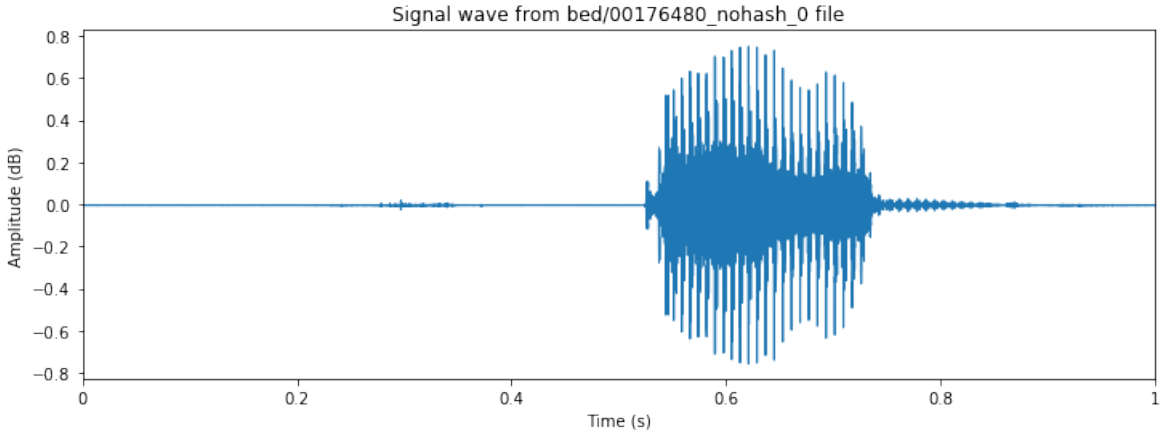


Figura 2.1: Onda sonora da palavra *bed*

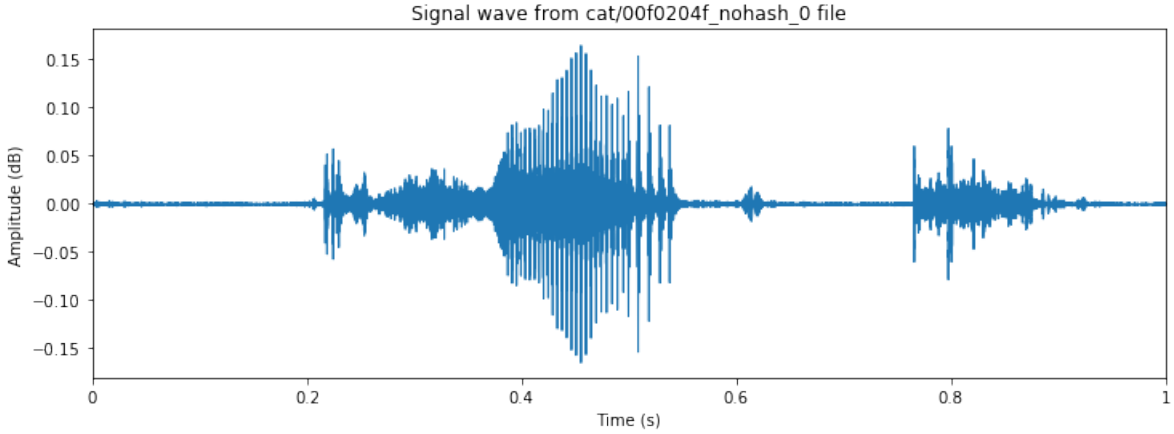


Figura 2.2: Onda sonora da palavra *cat*

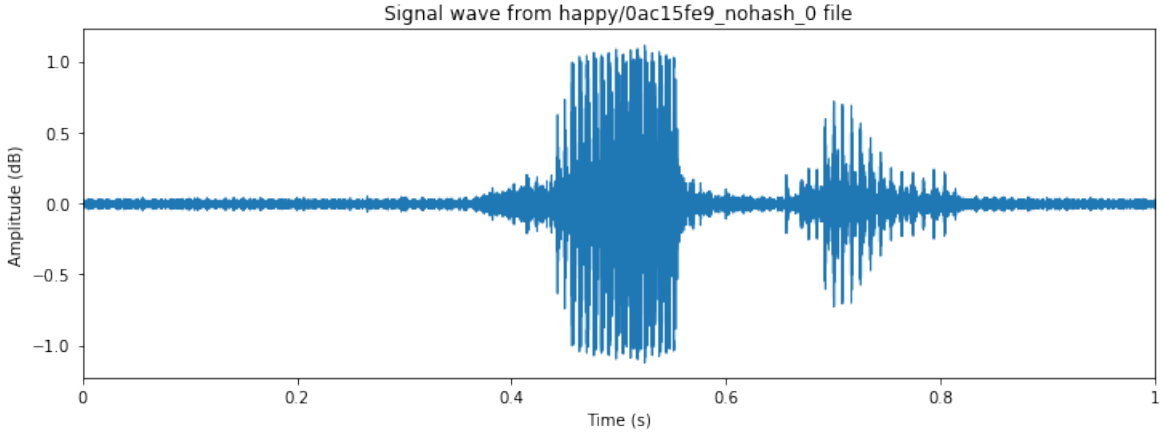


Figura 2.3: Onda sonora da palavra *happy*

2.3 Algoritmos e técnicas

Existem diversas formas de realizar a digitalização de áudios. Entretanto, como o foco desse projeto é o desenvolvimento de algoritmos de *machine learning*, foi utilizado a técnica Mel Frequency Cepstrum [7] para transformação dos arquivos de áudio no formato *.wav em vetores numéricos.

Para o algoritmo foi utilizado uma arquitetura simples de redes neurais e para melhores resultados relacionados a acurácia, existe a opção de redes neurais convencionais.

2.4 Benchmark

Por ser um desafio do kaggle [6], diversas pessoas desenvolveram algoritmos para resolver a competição com métodos distintos. Durante a minha busca sobre soluções referente ao tema, percebi que a solução mais executada foi utilizando Mel-Frequency Cepstrum para o pré-processamento do áudio e Convolutional Neural Networks para o algoritmo de machine learning. Chegando em alguns casos, a 86% de acurácia para configurações semelhantes à descrita.

Capítulo 3

Projeto do Sistema

3.1 Pré-processamento de dados

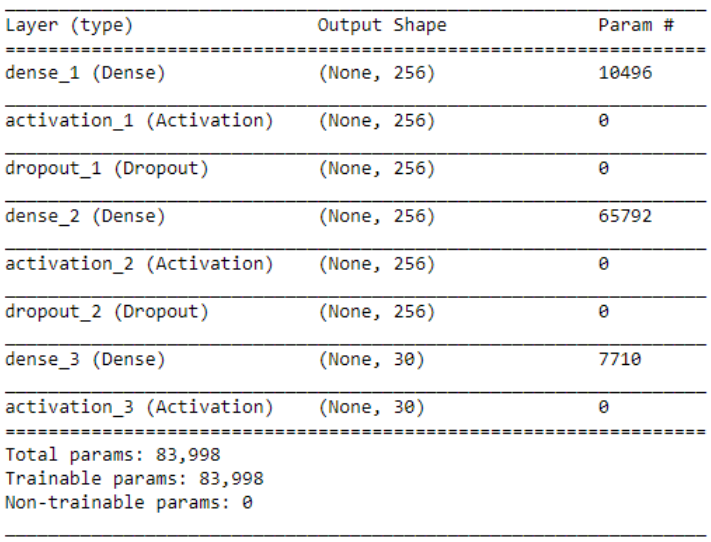
Para fazer a digitalização dos dados foi utilizado a função *mfcc* da biblioteca *librosa.feature* para ser calculado os coeficientes Mel Frequency Cepstrum [7] e assim transformar o arquivo de áudio em um vetor.

3.2 Implementação

A arquitetura de redes neurais para este problema utilizada foi a [9], no qual é utilizado uma arquitetura simples de Deep Learning, como ilustrado na imagem 3.1.

3.3 Refinamento

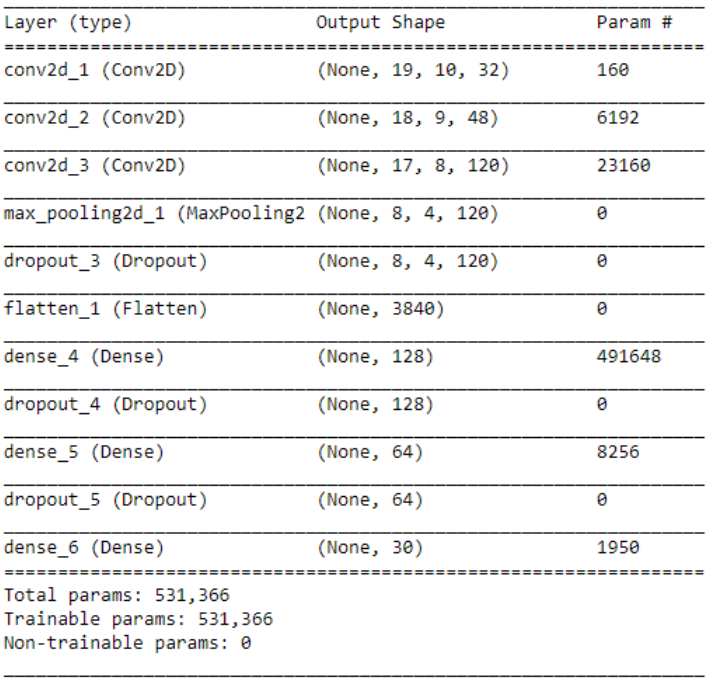
Para o refinamento do modelo é utilizado uma arquitetura de Redes Neurais Convolucionais [10], como ilustrado na imagem 3.2. Com o objetivo da acurácia aumentar devido à complexidade da estrutura.



Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 256)	10496
activation_1 (Activation)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65792
activation_2 (Activation)	(None, 256)	0
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 30)	7710
activation_3 (Activation)	(None, 30)	0

Total params: 83,998
Trainable params: 83,998
Non-trainable params: 0

Figura 3.1: Arquitetura da Rede Neural utilizada [9]



Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 19, 10, 32)	160
conv2d_2 (Conv2D)	(None, 18, 9, 48)	6192
conv2d_3 (Conv2D)	(None, 17, 8, 120)	23160
max_pooling2d_1 (MaxPooling2	(None, 8, 4, 120)	0
dropout_3 (Dropout)	(None, 8, 4, 120)	0
flatten_1 (Flatten)	(None, 3840)	0
dense_4 (Dense)	(None, 128)	491648
dropout_4 (Dropout)	(None, 128)	0
dense_5 (Dense)	(None, 64)	8256
dropout_5 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 30)	1950

Total params: 531,366
Trainable params: 531,366
Non-trainable params: 0

Figura 3.2: Arquitetura da Rede Neural Convolucional utilizada [10]

Capítulo 4

Resultado

4.1 Modelo de avaliação e validação

A utilização da arquitetura mais simples de rede neural, teve um resultado muito baixo. A acurácia avaliada foi de 28%.

Já, a arquitetura mais complexa, utilizando Redes Neurais Convolucionais, teve uma acurácia de 81%.

4.2 Justificativa

Na aplicação da arquitetura mais simples [9] se obteve uma acurácia de 66% muito diferentes do que eu obtive, acredito que muito devido a entrada de dados ser diferente. Na fonte [9] foi utilizado uma base de dados contendo sons diversos, não especificamente palavras.

Contudo, na aplicação de Redes Neurais Convolucionais, foi utilizado a mesma base de dados, entretanto com apenas 3 palavras e assim se obteve 94% de acurácia. Entretanto, nos meus resultados consegui uma acurácia menor, muito devido ao maior número de palavras possíveis.

Capítulo 5

Conclusão

(aprox. 1-2 páginas)

5.1 Reflexão

Neste projeto foi utilizado duas abordagens, uma mais simples e outra mais complexa. Mesmo achando que o uso de áudio como entrada do algoritmo já não era simples por si só, foi comprovado que apenas uma arquitetura mais complexa de redes neurais conseguia resolver o problema e fazer predições mais corretas.

5.2 Melhorias

Alguns passos que podem ser incrementados para que se evolua ainda mais este projeto:

- Vi em algumas referências, a utilização de Redes Neurais Convolucionais junto com *Long Short Term Memory* (LSTM). Poderia ser uma tentativa de aumentar ainda mais a acurácia;
- Poderia ser aumentado a dimensão dos vetores do *Mel Frequency Cepstrum* para ter uma resolução maior quando digitalizar o áudio;
- Incluir uma base maior de palavras para o treinamento do algoritmo.

Referências Bibliográficas

[1] Mamilos b9. http://www.b9.com.br/podcasts/mamilos/. Acesso em 26 de Abril de 2018.

[2] Transcrição mamilos 58. http://www.b9.com.br/63412/transcricao-mamilos-58/, 2016. Acesso em 26 de Abril de 2018.

[3] Transcrição mamilos 58. http://www.b9.com.br/81226/mamilos-128-especial-3-anos/, 2017. Acesso em 26 de Abril de 2018.

[4] Tensorflow speech recognition challenge. https://www.kaggle.com/c/tensorflow-speech-recognition-challenge, 2017. Acesso em 26 de Abril de 2018.

[5] Pete Warden. Speech commands: A public dataset for single-word speech recognition. http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz, 2017.

[6] kaggle. https://www.kaggle.com/, 2017. Acesso em 26 de Abril de 2018.

[7] Beth Logan et al. Mel frequency cepstral coefficients for music modeling. In *ISMIR*, volume 270, pages 1–11, 2000.

[8] python. https://www.python.org/. Acesso em 26 de Abril de 2018.

[9] Faizan Shaikh. Speech commands: Getting started with audio data analysis using deep learning (with case study). https://www.analyticsvidhya.com/blog/2017/08/audio-voice-processing-deep-learning/, 2017.

[10] Manash Mandal. Speech commands: Getting started with audio data analysis using deep learning (with case study). https://blog.manash.me/

building-a-dead-simple-word-recognition-engine-using-convnet-in-keras-25e72c19
2017.