Universidade Federal de São Carlos
Ciência da Computação
Profº Murilo Coelho Naldi

Classificação de Sons de Sirenes Usando CNN e LSTM

Ana Ellen Deodato P Silva 800206

Vinicius de Oliveira Guimarães 802431

Vinícius Gonçalves Perillo 800219

São Carlos, 02 de maio de 2035

Universidade Federal de São Carlos
Ciência da Computação
Profº Murilo Coelho Naldi

1 - O PROBLEMA

O nosso desafio consiste na aplicação de uma rede neural para diferenciação entre tipos de sirenes, incluindo exemplos de sons de ambulâncias, caminhões de bombeiros e tráfego de automóveis no geral. Para isso, utilizamos uma rede neural convolucional para identificação de padrões em espectrogramas (Imagem do som), com uso de várias estratégias de processamento na busca do melhor desempenho de classificação.

Colab: clique aqui

2 - SOBRE A BASE DE DADOS

A base de dados que será trabalhada consiste de vários arquivos de áudio (wav) das classes “ambulance”, “firetruck” e “traffic”. Mais especificamente, a base original contém 200 arquivos de áudio para cada uma das classes descritas acima.

3 - PRÉ-PROCESSAMENTO

3.0 Divisão de dados

Foram divididos 75% dos dados de cada classe para treino (150) e 25% para teste (50).

3.1 Espectrogramas Mel

Uma técnica famosa de processamento de som é o Mel Spectrograms (MS); ela é muito usada para faixas de som que os humanos normalmente escutam.

O primeiro passo para o processamento do som é transformar o arquivo de áudio em uma diagrama de ondas, que é um mapeamento do som para uma onda representada no plano tempo(s) × amplitude(dB). A partir da onda é usada a transformação de Fourier para gerar o diagrama densidade espectral de potência,

que mapeia a onda no plano da frequência(Hz) × amplitude(dB), desta forma é possível decompor o som em sua diferentes frequências. Integrando o diagrama de densidade espectral de potência ao longo do tempo consegue-se gerar um espectrograma, uma imagem capaz de representar três dimensões, sendo uma no eixo x, outra no eixo y e uma terceira na escala de cor, nesse caso tempo(s) × frequência(Hz) × amplitude(dB).

Para que um espectrograma se transforme em um Mel Spectrogram é preciso alterar a escala linear de frequência para a Escala Mel. Tal escala aproxima a variação da frequência a forma como os humanos percebem (os seres humanos distinguem os sons de maneira logarítmica).

Na imagem abaixo conseguimos observar características marcantes tanto para aúdio da classe

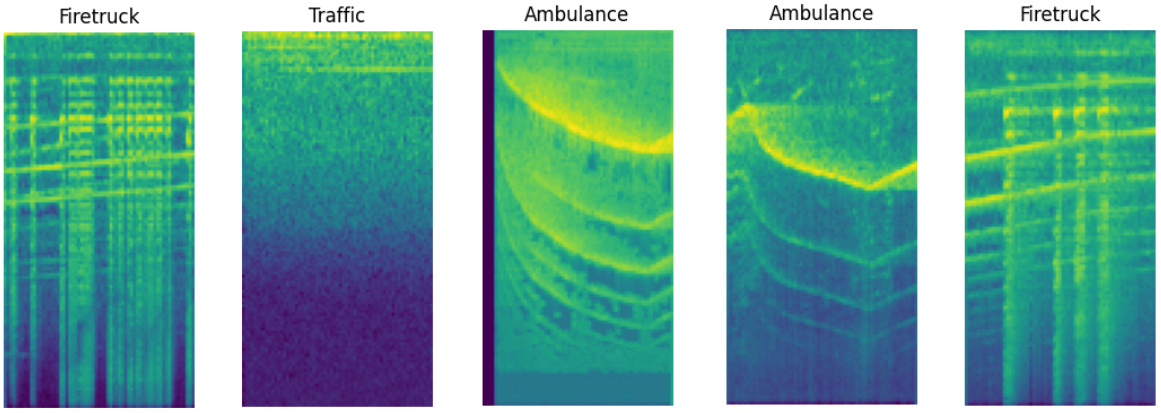

Imagem 1 - Espectrogramas criados para áudios de Firetruck e Ambulance

3.2 Aumento de Dados

Usamos algumas técnicas de aumento de dados para que a rede compreenda a estrutura de cada classe de maneira mais genérica:

- **Gaussian-noise e Pink Noise:** Adiciona ruído Gaussiano e ruído Rosa de pesos aleatórios em áudios aleatórios. Objetivo: Auxiliar no melhor reconhecimento e separação de sons de fundo ou não relacionados às sirenes, para que estes não atrapalhem na classificação e também auxilia na generalização do problema.

  Para o Gaussian-noise, escolhemos para cada classe (ambulance, fire truck e traffic) 50 sons aleatórios para ser aplicado. Essa aplicação considera uma média de 0 e desvio padrão de 0.005, de forma que não haja um viés da geração do ruído, fazendo com que ao ser aplicado o ruído fique mais difícil a compreensão do áudio, gerando maior variação.

  Para o Pink Noise fizemos a separação de 50 sons aleatórios para ser aplicado o ruído. O interessante do Pink Noise é que este, favorece a intensidade de sons com frequências menores, ou seja, no geral tendemos a ter um som com partes graves mais perceptíveis. Isso não significa que as partes com frequências maiores serão desconsideradas, mas sim que teremos uma harmonização maior na intensidade sonora, favorecendo sons mais graves que agudos.

- **Gain/random power:** Multiplica áudios por amplitudes aleatórias para aumentar ou diminuir seu volume. Objetivos: Tornar o algoritmo imparcial ao volume dos áudios, tornando mais fácil o reconhecimento de sirenes mais distantes ou mais próximas.

  Para os 50 áudios selecionados aleatoriamente, estamos sorteando um número entre 0.5 e 1.5, onde valores menores do que 1 irão diminuir o volume do áudio e maiores do que 1 irão aumentar o áudio.

4 - APLICAÇÃO DO MODELO

Como as etapas de pré-processamento resultaram num conjunto de imagens optou-se por analisar o desempenho das CNNs no problema, além disso como as LSTMs lidam bem com séries temporais optou-se por testá-las também resultando em 4 redes neurais.

4.1 CNN sobre os Mel Espectrogramas

A princípio modelamos uma CNN para lidar melhor com o padrão dos Mel Espectrogramas. Ela foi modelada utilizando 16 camadas, sendo que a região de Extração de características foi estruturada em quatro porções, cada uma contendo uma camada de convolução 2D (Com *stride* igual a 1 e função de ativação ReLU), uma camada de *MaxPooling* 2D (Usando *pool* de 2x2 e *stride* de 2) e uma camada de *Batch Normalization*.

Para as 4 camadas de convolução existentes, temos:

- A primeira e a segunda com *kernel* de tamanho 5 (Número de filtros sendo 32 e 64 respectivamente).
- A terceira e a quarta com *kernel* de tamanho 3 (Número de filtros sendo 128 e 256 respectivamente)

A região de classificação foi feita com 3 camadas:

- Uma camada que planifica o espaço tridimensional em unidimensional.
- Uma camada de 1024 neurônios densamente conectada com função de ativação ReLU e dropout de 33%.
- Por fim, uma camada de saída de 3 neurônios com função de ativação softmax, que representam as classes “*ambulance*”, “*firetruck*” e “*traffic*”.

Além disso usamos o otimizador Adam e função de custo *Sparse Categorical Cross Entropy*.

Tudo isso pode ser visto na imagem 2 abaixo:

```
model = keras.Sequential()

model.add(keras.layers.Conv2D(filters=32, kernel_size=5, padding='same', activation='relu', input_shape=(128,65,1)))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())


model.add(keras.layers.Conv2D(filters=64, kernel_size=5, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())


model.add(keras.layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())

model.add(keras.layers.Conv2D(filters=256, kernel_size=3, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(1024, activation='relu'))
model.add(Dropout(0.33))
model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Imagem 2 - Arquitetura da rede neural convolucional 2D

4.2 LSTM sobre os áudios

A segunda rede que se testou foi um rede simples de 6 camadas diretamente sobre os áudios sendo composta por: uma camada LSTM com 64 neurônios e dropout de 20%, duas camadas densamente conectadas com função de ativação ReLU e dropout de 40%, sendo a primeira de 64 neurônios e a segunda de 32, e uma camada de saída igual a da anterior. Pode-se ver na imagem 3 abaixo:

```
model = keras.Sequential()

model.add(keras.layers.LSTM(64, activation= 'tanh',  input_shape=(66150, 1), implementation=2))
model.add(Dropout(0.2))

model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(32, activation='relu'))
model.add(Dropout(0.4))

model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Imagem 3 - Arquitetura da rede neural LSTM

4.3 CNN + LSTM sobre os áudios

Vendo o desempenho da segunda rede decidiu-se por testar a combinação das duas técnicas em uma única rede diretamente sobre os áudios, assim usou-se uma rede de 12 camadas, sendo 3 porções de extração de *features* e uma de LSTM. As 3 partições consistem em:

- Uma camada de Convolução 1D, cada uma com stride 1, função de ativação ReLU, kernel de tamanho 23 e dobrando a quantidade de filtros em cada uma (32, 64, 128) ;
- Uma camada de MaxPooling 1D com pool de tamanho 16 e stride 16;
- Uma Camada de Batch Normalization

Após essas porções tem-se uma camada de LSTM com 64 neurônios com dropout de 20% e por fim a camada de saída igual a das anteriores como mostra a imagem 4 abaixo:

```
model = keras.Sequential()

model.add(keras.layers.Conv1D(filters=32, kernel_size=23, padding='same', activation='relu', input_shape=(66150, 1)))
model.add(keras.layers.MaxPooling1D(pool_size=16, strides=16))
model.add(BatchNormalization())


model.add(keras.layers.Conv1D(filters=64, kernel_size=23, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling1D(pool_size=16, strides=16))
model.add(BatchNormalization())


model.add(keras.layers.Conv1D(filters=128, kernel_size=23, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling1D(pool_size=16, strides=16))
model.add(BatchNormalization())

model.add(keras.layers.LSTM(64, activation= 'tanh'))
model.add(Dropout(0.2))

model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Imagem 4 - Arquitetura da rede neural convolucional 1D + LSTM

4.4 CNN + LSTM sobre os Mel Espectrogramas

Por fim, vendo a possibilidade de utilizar-se as duas técnicas optou-se por testar a junção das técnicas sobre os espectrogramas obtendo uma rede de 12 camadas, semelhante à anterior. A rede consiste em 3 porções de extração de *features* e uma de LSTM. As 3 porções consistem em:

- Uma camada de Convolução 2D, cada uma com stride 1, função de ativação ReLU, kernel de tamanho 5, 5 e 3 e dobrando a quantidade de filtros em cada uma (32, 64, 128) ;
- Uma camada de MaxPooling2D com pool de tamanho 2x2 nos dois primeiros e o último com 32 x 2 para neutralizar uma dimensão e stride 2;
- Uma Camada de Batch Normalization

Após essas porções tem-se uma camada de LSTM com 64 neurônios com dropout de 20% e por fim a camada de saída igual a das anteriores como mostra a imagem 4 abaixo:

```
model = keras.Sequential()

model.add(keras.layers.Conv2D(filters=32, kernel_size=5, padding='same', activation='relu', input_shape=(128,65,1)))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())

model.add(keras.layers.Conv2D(filters=64, kernel_size=5, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=[2,2], strides=2))
model.add(BatchNormalization())

model.add(keras.layers.Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'))
model.add(keras.layers.MaxPooling2D(pool_size=[32,2], strides=2))
model.add(BatchNormalization())

model.add(keras.layers.Reshape([8,128]))

model.add(keras.layers.LSTM(64, activation= 'tanh'))
model.add(Dropout(0.2))

model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Imagem 5 - Arquitetura da rede neural convolucional 2D + LSTM

5 - RESULTADOS

Após o processo de treinamento e validação obtivemos os resultados abaixo para os modelos utilizando CNN e LSTM:

Estratégia Utilizada	**Acurácia**
CNN 2D sobre Mel Espectrogramas	96.00%
LSTM sobre os áudios	33.33%
CNN 1D + LSTM com áudio	95.33%
CNN 2D + LSTM com Mel Espectrogramas	98.00%

A acurácia da primeira rede foi muito boa, 96%, mostrando que a CCN lidou bem com os espectrogramas. Já a segunda rede teve uma acurácia igual a de um algoritmo aleatório, 33,33%, o que desanimou bastante, a partir desse resultado decidiu-se combinar as técnicas e aplicar a CNN 1D com a LSTM surpreendeu bastante com a acurácia de 95.33% o que motivou a testar com os espectrogramas o

que demonstrou que a LSTM melhora a análise dos dados devido sua dependência temporal e teve a melhor acurácia com 98% de acerto.

6 - CONCLUSÃO

Através do trabalho pode-se perceber a eficácia das CNNs, ela tem um papel muito importante para a detecção de padrões nos dados antes do aprendizado. Quando utilizada diretamente sobre o áudio teve uma acurácia muito boa, suprindo a etapa de extração de features do pré processamento. Entretanto a LSTM pura deixou a desejar, seu treino foi muito lento e teve uma acurácia equivalente de um algoritmo aleatório, entretanto uma etapa de extração de features pode ter sido o que faltou já que o uso dela junto da CNN melhorou a acurácia das redes. Usando os espectrogramas como entrada da LSTM pode ser que ela obtenha uma boa acurácia. O problema foi simples, conseguiu-se boas acurácias sem muita dificuldade, mas existem problemas complexos como o da BirdCLEF que exigiriam mais atenção ao arquitetarmos as redes e escolher seus parâmetros.

7 - REFERÊNCIAS

Dataset utilizado:
https://www.kaggle.com/datasets/vishnu0399/emergency-vehicle-siren-sounds

Artigo sobre a BirdClef e aplicação de Mel Spectrogram e Data Augmentation:
https://ceur-ws.org/Vol-3180/paper-174.pdf