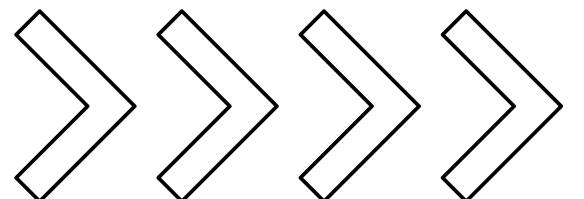


# **APRESENTAÇÃO TRABALHO 2**

**DETECTOR DE EMBARCAÇÃO COM CNN**  
**REDES NEURAIS E DEEP LEARNING**



**Pós-Graduandos:**

- Efrain Marcelo
- Pietro Silva
- Willian Moura

**Docente:** Prof. Dr. Adrião Duarte

**Curso:** PPGEEC

# Índice

**01** Introdução

**02** Objetivo

**03** Metodologia

**04** Desenvolvimento

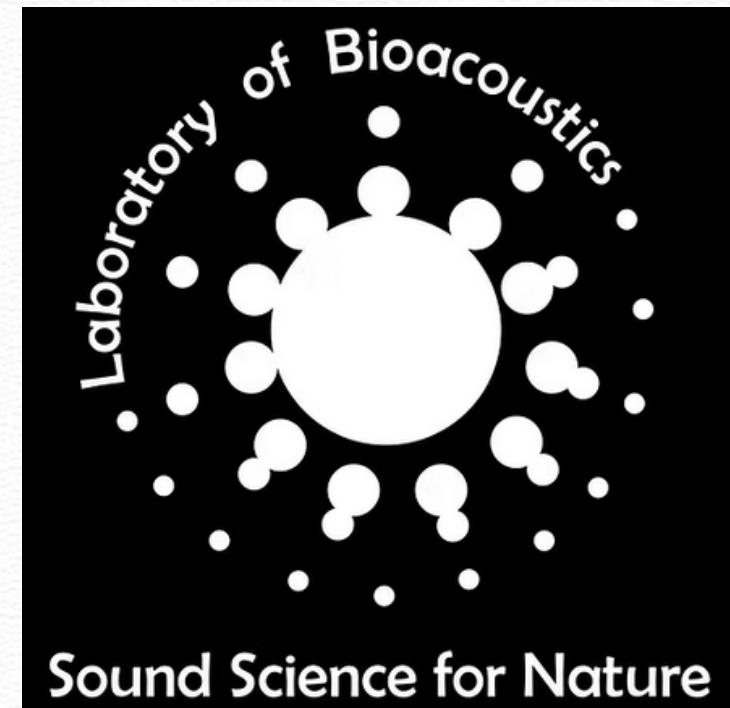
**05** Solução Final

**06** Conclusão

# Introdução

## Embarcações atrapalham processos de análises

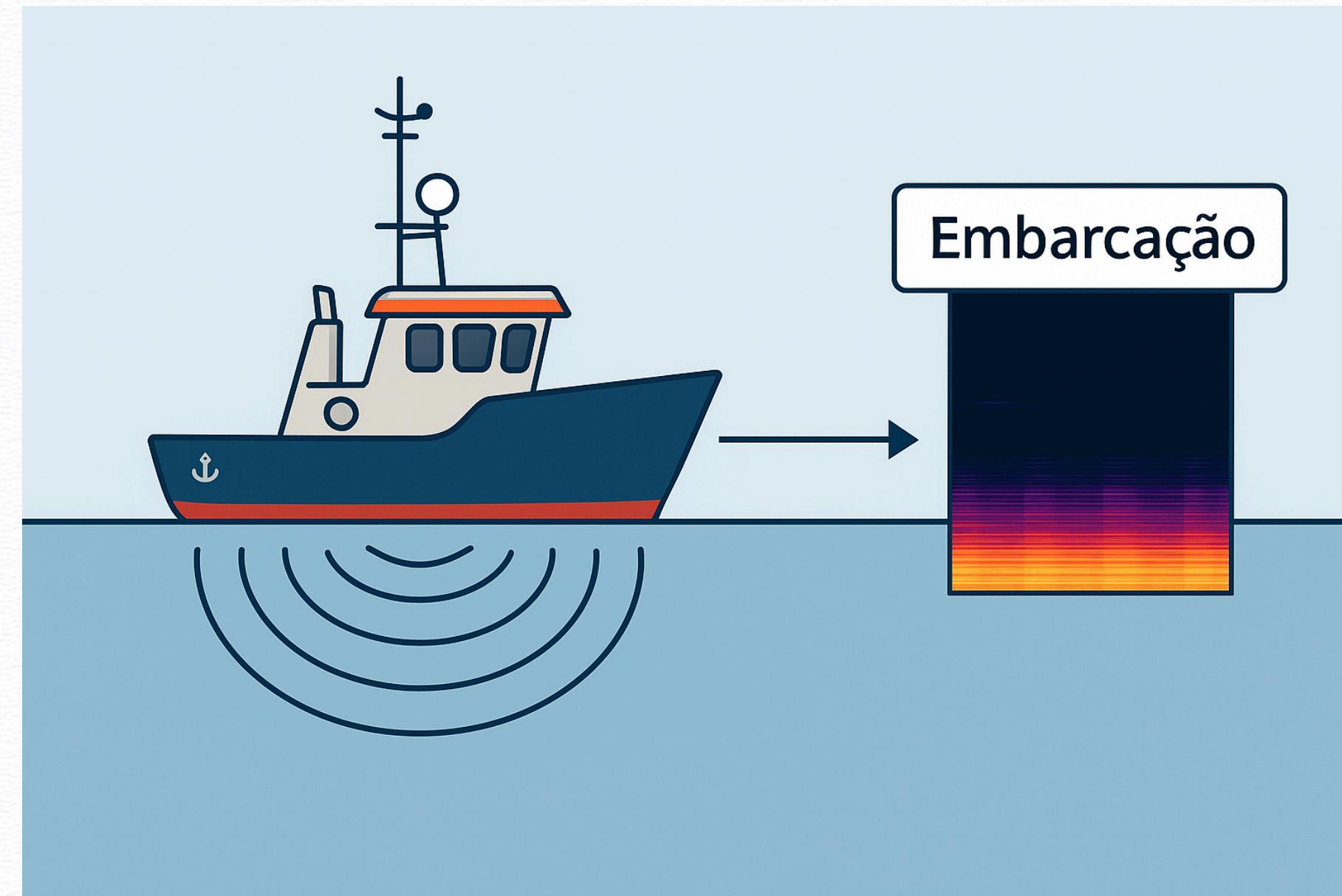
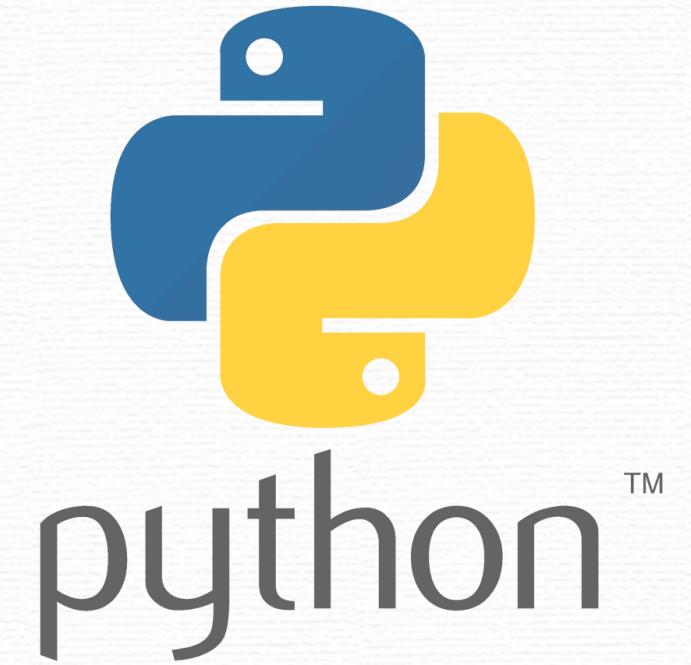
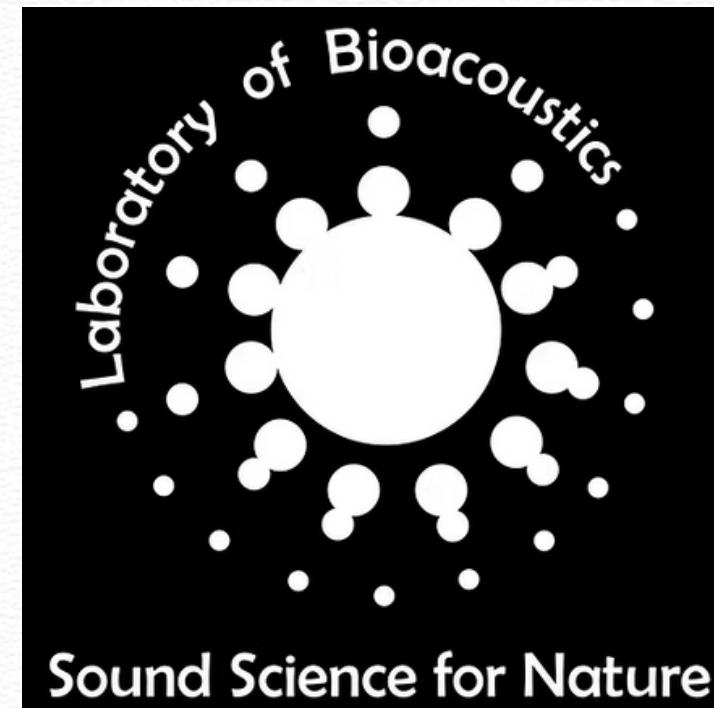
- Demanda proveniente do Laboratório de Pesquisa em Bioacústica.
- Analistas utilizam o software Raven 1.6 para identificação de atividades marinhas.
- Pelo processo atual de análise não consegue se ter uma ideia se o audio contem uma embarcação
- Dificuldade em analisar audios com ruidos de barcos que mancham o espectro
- Interesse em listar eventos de embarcações relacionados a pesquisa



# Objetivo

**Desenvolver um detector automático  
de Embarcações**

- Python oferece maior flexibilidade: aplicações embarcadas, servidores, notebooks pessoais, etc.
- Busca por agilidade, eficiência e acessibilidade no processamento dos dados acústicos.
- Uso de técnicas de DeepLearning para detectar arquivos de audio que podem conter embarcações e dar uma "ideia" ao analista que for trabalhar em cima do audio





# Metodologia

**01** Construir Dataset

**02** EDA

**03** Balanceamento dos Dados

**04** Criação do Modelo CNN

**05** Resultados

**06** Implantação da Solução

- Recebe um áudio bruto como entrada.
- Executa a predição com o modelo treinado.
- Retorna os audios classificados como conteúdo embarcações.

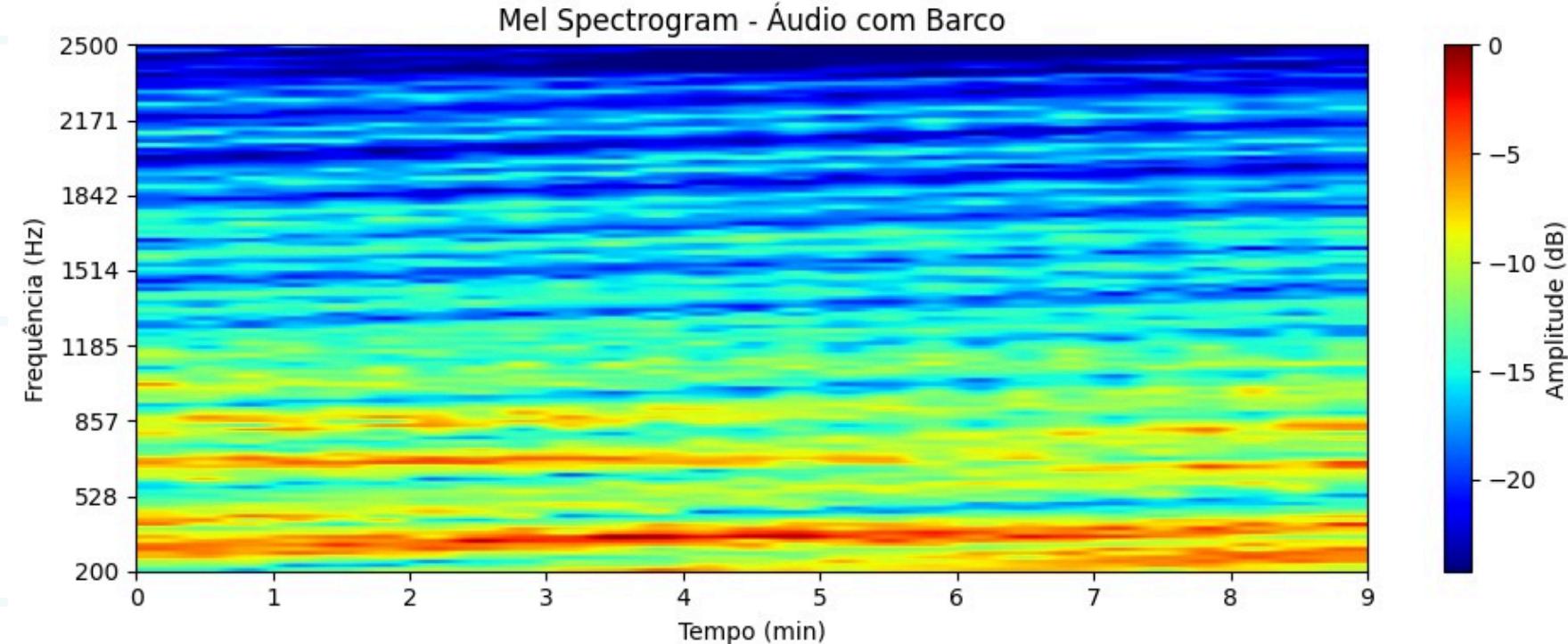
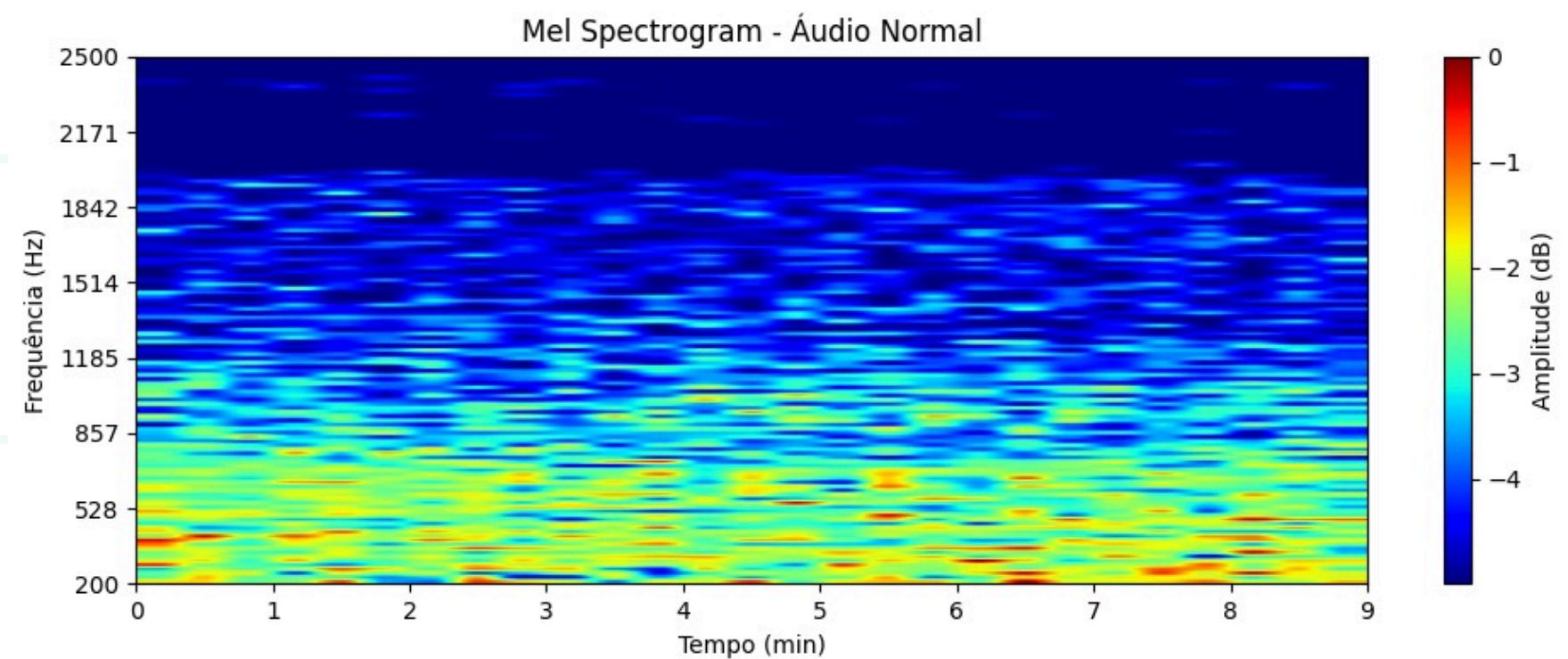
# Desenvolvimento

## Dataset

- Através de anotações feitos por analistas, obtivemos uma lista de audios com embarcações e uma lista de audios "limpos"
- Pxx dos Audios separados em pastas
- Carregamos para o Python transformando para MelSpectogram

```
# === CONFIGURAÇÕES ===  
INPUT_SHAPE = (513, 27, 1)  
PATH_BARCO = '../pxx_barcos'  
PATH_NORMAL = '../pxxs_normais'  
SEED = 42  
[76]
```

```
# Carregamento  
dados_barcos, rotulos_barcos = carregar_mel_h5(PATH_BARCO, label=1, fmin=300, fmax=2500)  
dados_normais, rotulos_normais = carregar_mel_h5(PATH_NORMAL, label=0, fmin=300, fmax=2500)  
print(f"Carregados {len(dados_barcos)} barcos e {len(dados_normais)} normais.")  
[34]  
... Carregados 73 barcos e 359 normais.
```



# Desenvolvimento

## Balanceamento e Data Augmentation

- Poucos dados com embarcações
- Geração de novos dados de embarcação acrescentando aos originais um **Ruído Gaussiano**
- Uso de compute\_class\_weight para garantir iparcialidade do modelo no processo de apredizagem

The screenshot shows a Jupyter Notebook interface with three code cells:

- Aumentando os dados com ruido Gaussiano**:

```
# Aumenta classe barco
dados_barcos_aug = augment_noise(dados_barcos, std=0.01, copies=3)
rotulos_barcos_aug = [1] * len(dados_barcos_aug) "rotulos": Unknown word.

# Junta tudo
X = np.array(dados_normais + dados_barcos + dados_barcos_aug)
y = np.array(rotulos_normais + rotulos_barcos + rotulos_barcos_aug) "rotulos": Unknown word.
```

[82]
- Dividindo dados em Treino e Teste**:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

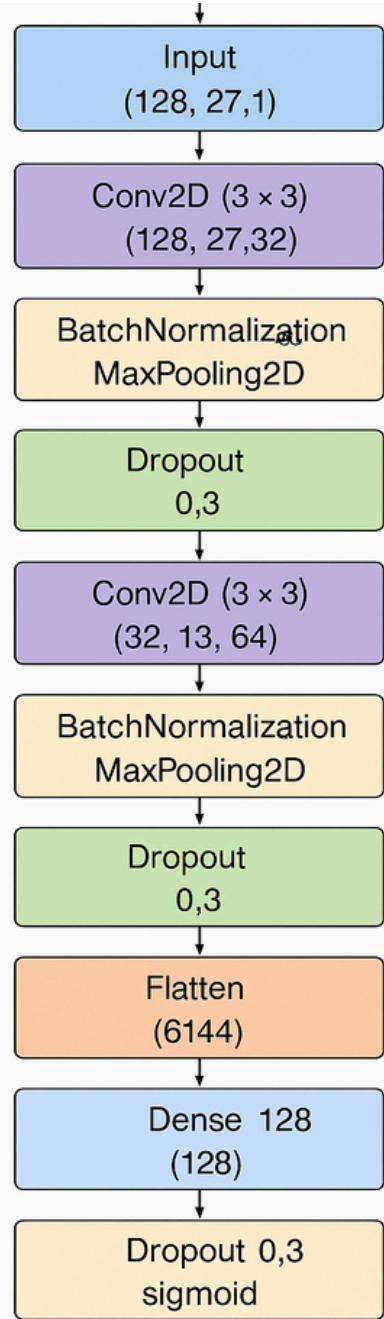
[83]
- Aplicando Peso por Classe**:

```
# Pesos de classe
classes = np.unique(y_train)
class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)
class_weights_dict = dict(zip(classes, class_weights))
```

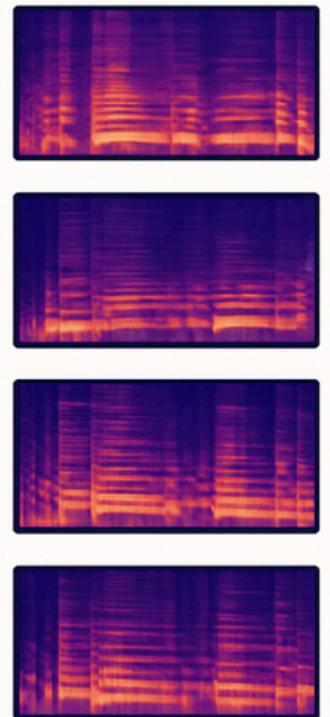
[84]

# Desenvolvimento

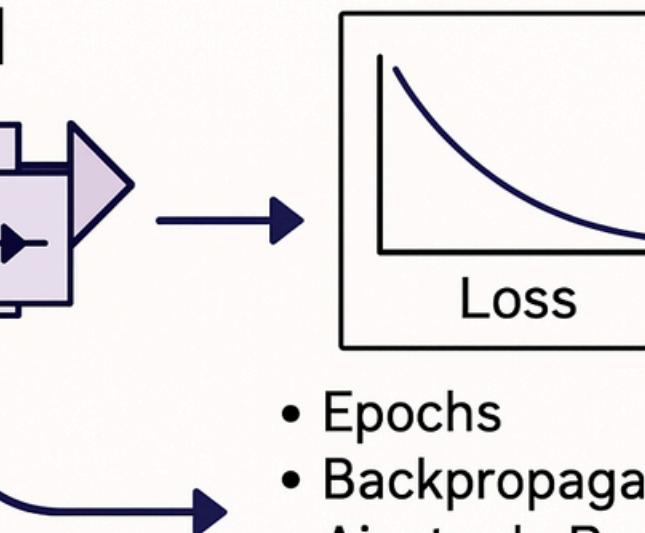
## Modelo e Treino



Mel Spectrograms



CNN



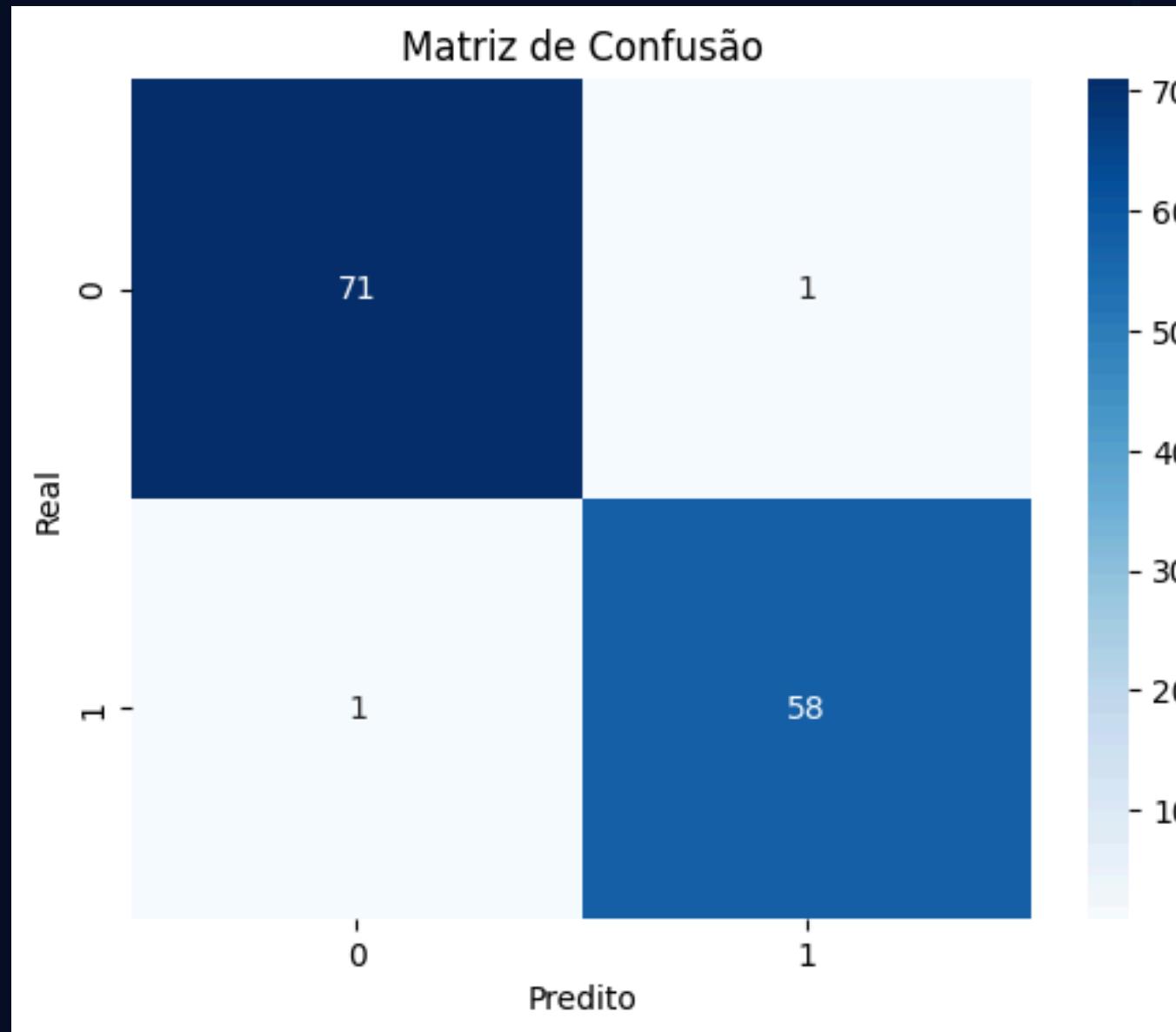
- Epochs
- Backpropagation
- Ajuste de Pesos

Mel Spectrograms

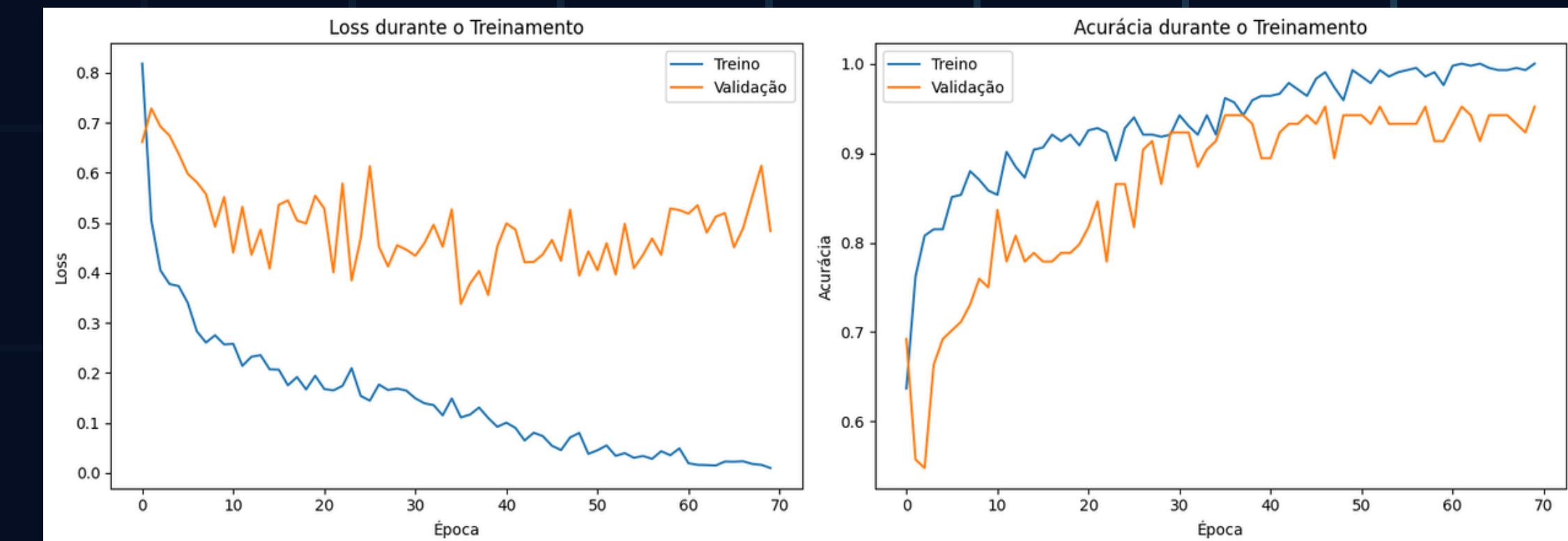
```
history = model.fit(  
    X_train_final, y_train_final,  
    validation_data=(X_val, y_val),  
    epochs=30,  
    batch_size=16,  
    class_weight=class_weights_dict,  
    verbose=1,  
    shuffle=True,  
    callbacks=[early_stop])
```

```
Epoch 1/30  
29/29 2s 39ms/step - accuracy: 0.6239 - loss: 1.0517 - val_accuracy: 0.6897 - val_loss: 0.6589  
Epoch 2/30  
29/29 1s 36ms/step - accuracy: 0.6736 - loss: 0.7402 - val_accuracy: 0.8736 - val_loss: 0.5724  
Epoch 3/30  
29/29 1s 40ms/step - accuracy: 0.6947 - loss: 0.6175 - val_accuracy: 0.8851 - val_loss: 0.4588  
Epoch 4/30  
29/29 1s 39ms/step - accuracy: 0.7599 - loss: 0.6108 - val_accuracy: 0.8736 - val_loss: 0.5391  
Epoch 5/30  
29/29 1s 37ms/step - accuracy: 0.7663 - loss: 0.4397 - val_accuracy: 0.6552 - val_loss: 0.5328  
Epoch 6/30  
29/29 1s 36ms/step - accuracy: 0.7580 - loss: 0.4658 - val_accuracy: 0.8851 - val_loss: 0.4613  
Epoch 7/30  
29/29 1s 36ms/step - accuracy: 0.7891 - loss: 0.4424 - val_accuracy: 0.5172 - val_loss: 0.7650  
Epoch 8/30  
29/29 1s 37ms/step - accuracy: 0.8351 - loss: 0.4124 - val_accuracy: 0.5287 - val_loss: 0.6914  
Epoch 9/30  
29/29 1s 38ms/step - accuracy: 0.8204 - loss: 0.3946 - val_accuracy: 0.5287 - val_loss: 0.5876  
Epoch 10/30  
29/29 1s 36ms/step - accuracy: 0.8114 - loss: 0.3952 - val_accuracy: 0.8506 - val_loss: 0.4260  
Epoch 11/30
```

# Resultados



	precision	recall	f1-score	support
0	0.99	0.99	0.99	72
1	0.98	0.98	0.98	59
accuracy			0.98	131
macro avg	0.98	0.98	0.98	131
weighted avg	0.98	0.98	0.98	131



# Conclusão

## Alto Desempenho no Teste Final:

- Acurácia: 98%
- F1-score: 0.98 para ambas as classes
- Apenas 2 erros em 131 amostras

## Aplicação Eficiente em Bioacústica:

- Detecção confiável de embarcações via áudio
- Pipeline adaptável para outros cenários acústicos

## Melhorias:

- Aplicar regularização L2 (kernel\_regularizer) nas camadas
- Explorar CNNs pré-treinadas adaptadas a espectrogramas
- Ampliar o dataset real (mais gravações)
- Gerar exemplos sintéticos com data augmentation mais variado