**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA**
**FACULDADE DE TECNOLOGIA DE BOTUCATU**
**CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**AIME GIOVANNA PEREIRA**

# DESENVOLVIMENTO DE UM PROGRAMA COMPUTACIONAL PARA RECONHECIMENTO DE LESÕES DE ESCLEROSE MÚLTIPLA EM IMAGENS DE RESSONÂNCIA MAGNÉTICA

Botucatu-SP
Outubro– 2024

**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA**
**FACULDADE DE TECNOLOGIA DE BOTUCATU**
**CURSO SUPERIOR DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**AIME GIOVANNA PEREIRA**

# DESENVOLVIMENTO DE UM PROGRAMA COMPUTACIONAL PARA RECONHECIMENTO DE LESÕES DE ESCLEROSE MÚLTIPLA EM IMAGENS DE RESSONÂNCIA MAGNÉTICA

Orientador: Prof. Me. Renato Luiz Gambarato

Projeto de Iniciação Científica apresentado à FATEC - Faculdade de Tecnologia de Botucatu para o Curso Superior de Análise e Desenvolvimento de Sistemas.

Botucatu-SP
Outubro- 2024

## AGRADECIMENTOS

Agradeço à Fatec-BT pelo apoio no desenvolvimento desta pesquisa, e a todas as pessoas que direta ou indiretamente colaboram e acreditam neste trabalho.

## RESUMO

A Esclerose Multipla é uma doença neurodegenerativa, que acomete o sistema imunológico e faz com que ele ataque o sistema nervoso central (cérebro e medula óssea). É uma doença que acomete principalmente pessoas jovens do sexo feminino e seus sintomas variam de acordo com o paciente. Não possui causa nem cura e seu diagnóstico e monitoramento é realizado através de imagens de ressonância magnética. Como o diagnostico por ressonância pode ser demorado e complexo, novos estudos mostram novas técnicas para aprimorar esse diagnóstico precoce e consequentemente o tratamento precoce, através de aprendizado de máquina. O objetivo desse trabalho é identificar possíveis lesões em imagens de Ressonância Magnética, utilizando uma rede neural. Para isso utilizaremos a linguagem de programação Phyton com TensorFlow, queras, imagens e nuvem de armazenamento e o google colab com imagens sendo armazenadas no google drive.

# Sumário

# 1 INTRODUÇÃO

A esclerose múltipla (EM) é uma doença neurodegenerativa autoimune que afeta o sistema nervoso central, caracterizada pela ocorrência de surtos inflamatórios que causam lesões no cérebro e na medula espinhal. Afetando principalmente mulheres entre 20 e 45 anos, a EM apresenta um impacto significativo na qualidade de vida dos pacientes e representa um desafio diagnóstico, devido à semelhança de seus sintomas com outras doenças neurológicas (Dobson, Giovannoni, 2019).

Apesar dos avanços no diagnóstico por ressonância magnética, o processo ainda pode ser demorado e sujeito a erros, levando a um diagnóstico tardio ou impreciso. Com isso, novas tecnologias, como a inteligência artificial (IA), têm ganhado relevância como ferramentas de apoio aos médicos para melhorar a acurácia e velocidade no diagnóstico de lesões causadas pela esclerose múltipla (Filippi et al, 2016; Filippi *et* al, 2019).

Este trabalho tem como objetivo aplicar uma rede neural para identificar possíveis lesões em imagens de ressonância magnética de pacientes com esclerose múltipla, utilizando técnicas de aprendizado de máquina, visando melhorar a precisão do diagnóstico e contribuir para um tratamento mais eficaz.

## 2 REVISÃO BIBLIOGRÁFICA

A esclerose múltipla (EM) é uma doença neurodegenerativa inflamatória do Sistema Nervoso Central (SNC), que pode causar inflamação, desmielinização, gliose e perda neuronal, caracterizada por uma resposta imunológica equivocada que ataca o cérebro e a medula espinhal. Afetando principalmente mulheres caucasianas entre 20 e 45 anos (Dobson, Giovannoni, 2019; Pinto et al, 2020; Shoeibi et al, 2021; Haki *et al,* 2024).

Estudos indicam uma crescente prevalência da esclerose múltipla em países desenvolvidos e em desenvolvimento, embora suas causas permaneçam incertas. Fatores como predisposição genética, baixa exposição à luz solar (UVB), níveis insuficientes de vitamina D, infecção pelo vírus Epstein-Barr, obesidade e tabagismo têm sido associados ao aumento da suscetibilidade à doença (Dobson, Giovannoni, 2019; Filippi *et al*, 2019)

É uma doença caracterizada como doença autoimune mediada por células T órgão-especificas, que se tornam auto reativas, causando inflamação e quebra da barreira hematoencefálica, o que por ocasiona o ataque à mielina (Haki *et al,* 2024). É caracterizada por dois estágios: com surtos inflamatórios iniciais e recaídas que podem ser reversíveis. Quando os surtos são recorrentes podem ocorrer lesões que causam alteração graves na medula espinhal ou cérebro. Devido à multiplicidade de áreas afetadas, a doença recebe o nome de esclerose múltipla. Seu curso é heterogêneo, variando amplamente entre os pacientes, e ainda não possui uma cura (Dobson, Giovannoni, 2019; Pinto *et al*, 2020; Shoeibi *et al*, 2021).

As técnicas convencionais de ressonância magnética, como spin-eco ponderado em T1 e T2, além de imagens de recuperação de inversão atenuadas por fluidos, são usadas para avaliar lesões visíveis e a atrofia do sistema nervoso central. Em contraste, métodos avançados de ressonância magnética, como imagens ponderadas por difusão, imagens de transferência de magnetização, espectroscopia de ressonância magnética e ressonância magnética funcional, são essenciais para entender a patogênese da esclerose múltipla (Haki *et al,* 2024). Os critérios para diagnostico e monitoramento da progressão e os efeitos do tratamento são baseados na presença de leões focais na substância branca do Sistema Nervoso Central (Filippi et al, 2016; Filippi *et* al, 2019).

O diagnóstico por ressonância magnética, embora essencial para a esclerose múltipla (EM), pode ser demorado, desafiador e suscetível a erros manuais (Aslam et al, 2022). Diante disso, novas pesquisas têm proposto métodos mais precisos para identificar a doença e diferenciar casos

complexos. Entre esses avanços, as técnicas de inteligência artificial (IA) vêm ganhando destaque como ferramentas importantes para auxiliar médicos no diagnóstico de doenças. Essas técnicas podem ser divididas em métodos convencionais de aprendizado de máquina (ML) e técnicas de aprendizado profundo (DL) (Shoeibi et al, 2021).

O aprendizado de máquina oferece aos computadores a capacidade de resolver problemas aprendendo com a experiência, sem a necessidade de programação explícita. Isso ocorre através de modelos matemáticos que são treinados com dados e ajustados para fazer previsões precisas por meio de algoritmos de otimização (Lundervold e Lundervold, 2018). Já o aprendizado profundo, uma subcategoria do ML, utiliza redes neurais multicamadas para analisar grandes volumes de dados, tornando-se uma técnica eficaz para o diagnóstico automatizado da EM (Aslam *et al.* 2022*).*

Estudos, como os de Arani *et al.* (2018), revisaram o uso de IA no diagnóstico de EM, destacando técnicas como lógica fuzzy (FL) e redes neurais artificiais (ANN) como as mais eficazes, embora ainda tenham limitações. Seccia *et al.* (2021) e Aslam *et* al (2022) também ressaltaram a importância de uma maior colaboração entre clínicos e cientistas da computação, visto que ainda não há um modelo prognóstico amplamente aplicável para a EM.

Tanto DL quanto ML têm o potencial de auxiliar os clínicos na previsão de suscetibilidade à EM, no diagnóstico precoce e preciso, no acompanhamento da progressão da doença e na personalização de tratamentos. Este trabalho revisa a literatura sobre o uso dessas técnicas no diagnóstico de EM, organizando e analisando estudos desde 2011, com foco em modelos, tamanho de conjuntos de dados e desempenho (Aslam *et al.* 2022*).*

Diante desses avanços tecnológicos e da importância do diagnóstico preciso da esclerose múltipla, este trabalho tem como objetivo aplicar uma rede neural para identificar possíveis lesões em imagens de ressonância magnética.

## 3 METODOLOGIA

Neste trabalho será utilizado a técnica de aprendizado profundo (deep leaninrg) para classificar imagens de ressonância magnética junto a uma rede neural. Uma rede neural é um tipo de modelo treinado para reconhecer padrões. Ela é composta por camadas de entrada e finais, além de pelo menos uma camada escondida. Os neurônios de cada uma delas aprendem representações cada vez mais abstratas dos dados (TENSORFLOW, 2023), tendo como objetivo a distinção de imagens com pontos de inflamação e imagens sem pontos de inflamação, a fim de auxiliar no diagnóstico da Esclerose Múltipla. Esse tipo de aplicação usualmente requer pesados processamentos em dataset massivos (CARNEIRO, *et al*).

Redes neurais são treinadas por gradiente descendente os pesos de cada camada começam com valores aleatórios, que melhoram por iteração com o tempo. Isso aumenta a acurácia da rede. Uma função de perda é usada para quantificar essa acurácia. Um processo chamado retropropagação é usado para determinar se o peso aumentará ou diminuirá a fim de reduzir a perda (TENSORFLOW, 2023), serão utilizados datasets públicos obtidos na web que contém imagens de ressonância magnética do cérebro, para seu treinamento.

Essas imagens foram utilizadas em pesquisas com foco similar, classificados e organizados junto ao seus metadados, disponibilizadas por Poldrack *et al*. (2013); Poldrack & Gorgolewski (2015), Muslin, et al (2000) e pelos dataset nitrc[1], dataset medinfo[2], que serão armazenados junto ao Google Drive.

Segundo Carneiro et al. (2018), a maioria dos frameworks de aprendizagem profunda utiliza GPUs como seu processamento principal, envolvendo riscos junto ao hardware, como super utilização, depreciação do hardware e ele pode estar sujeito a falhas. Tornando muito difícil e custoso manter um ambiente computacional físico que seja robusto o suficiente. Por conta disso será utilizada uma ferramenta disponibilizada pelo Google, de forma gratuita, com um ambiente virtual, que providencia interpretadores de Python 2 e 3, e de fácil configuração para instalação de bibliotecas como TensorFlow, Matplotlib e Keras (Carneiro, *et al*).

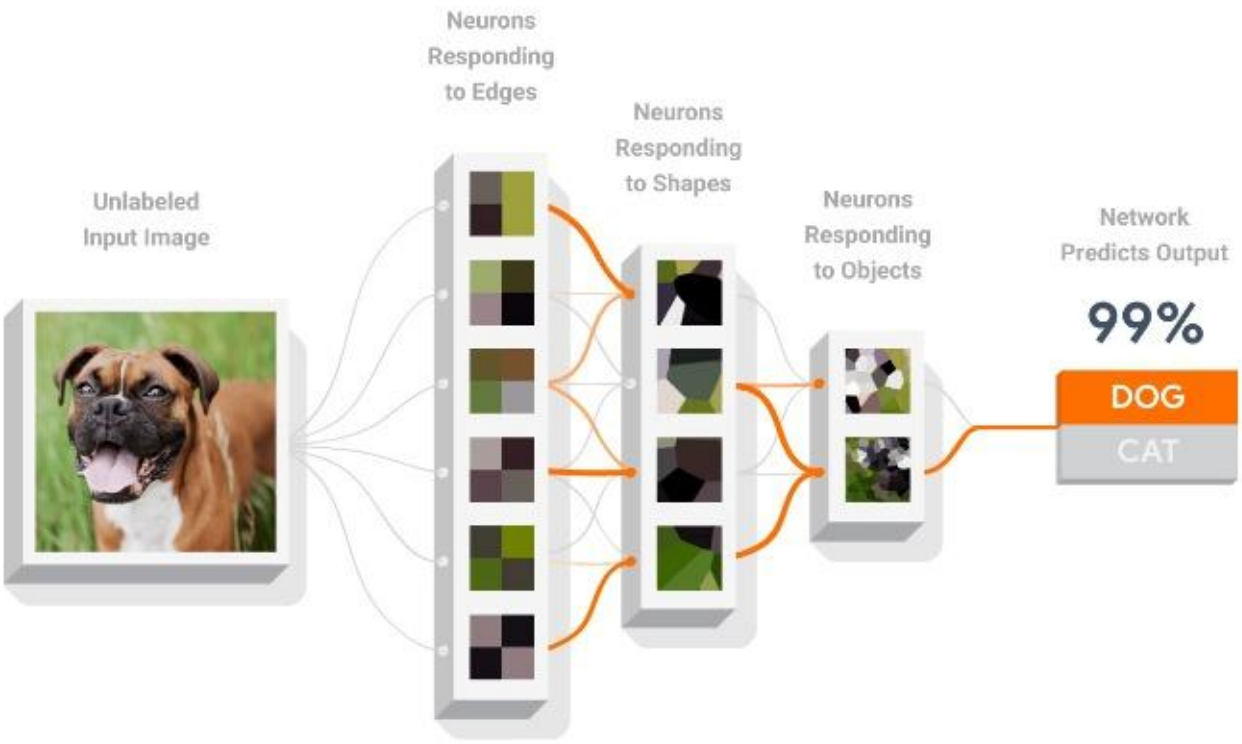



Figura 1. https://www.tensorflow.org/about?hl=pt-br

## 4 RESULTADOS E DISCUSSÃO

Para iniciar o treinamento do modelo as imagens foram divididas em Axial e Sagital a partir de imagens do kaggle, foram importadas algumas bibliotecas que seriam fundamentais para o projeto.

Após importar as *libs* foi feito a estruturação dos dados, criando diretórios, redimensionando as imagens. O conjunto de validação utilizou do conjunto de treinamento por hold-out, que consiste na segmentação das imagens em validação e treinamento (Leon-Sannchez, *et al.,* 2023). Para esse modelo foi considerado 80%  dos dados para treino e teste,  e 20% para validação, no treinamento do modelo foi utilizada uma técnica de autotune para auto regulagem do modelo (Figura 1).

```
################################################
# PARAMETROS PARA TUNING
################################################
btch_size = 32
img_size  = (256, 256)
seed_     = 1337

version = 'v1'
path_refa = f'/content/drive/MyDrive/IC RENATO/data/ref/{version}/axial/'

valid_dir = path_refa + 'valid/'
train_dir = path_refa + 'treino/'


train_ds = tf.keras.utils.image_dataset_from_directory(
  train_dir,
  validation_split=0.2,
  subset="training",
  seed=seed_,
  image_size=(img_size[0], img_size[1]),
  batch_size=btch_size
)

valid_ds = tf.keras.utils.image_dataset_from_directory(
  train_dir,
  validation_split=0.2,
  subset="validation",
  seed=seed_,
  image_size=(img_size[0], img_size[1]),
  batch_size=btch_size
)
```

Figura 1- Início da estruturação dos dados

```
[ ]  AUTOTUNE = tf.data.AUTOTUNE

     train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
     val_ds = valid_ds.cache().prefetch(buffer_size=AUTOTUNE)
     normalization_layer = layers.Rescaling(1./255)

     normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
     image_batch, labels_batch = next(iter(normalized_ds))
     first_image = image_batch[0]
     # Notice the pixel values are now in `[0,1]`.
     print(np.min(first_image), np.max(first_image))

     0.0 0.64197206
```

Figura 2- Configuração Autotune

```
[ ]  num_classes = 2

     model = Sequential([
       layers.Rescaling(1./255, input_shape=(img_size[0], img_size[1], 3)),
       layers.Conv2D(16, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(32, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(64, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Flatten(),
       layers.Dense(128, activation='relu'),
       layers.Dense(num_classes)
     ])


     model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                   metrics=['accuracy'])


     model.summary()
```

Figura 3- Modelo de dados convolucional

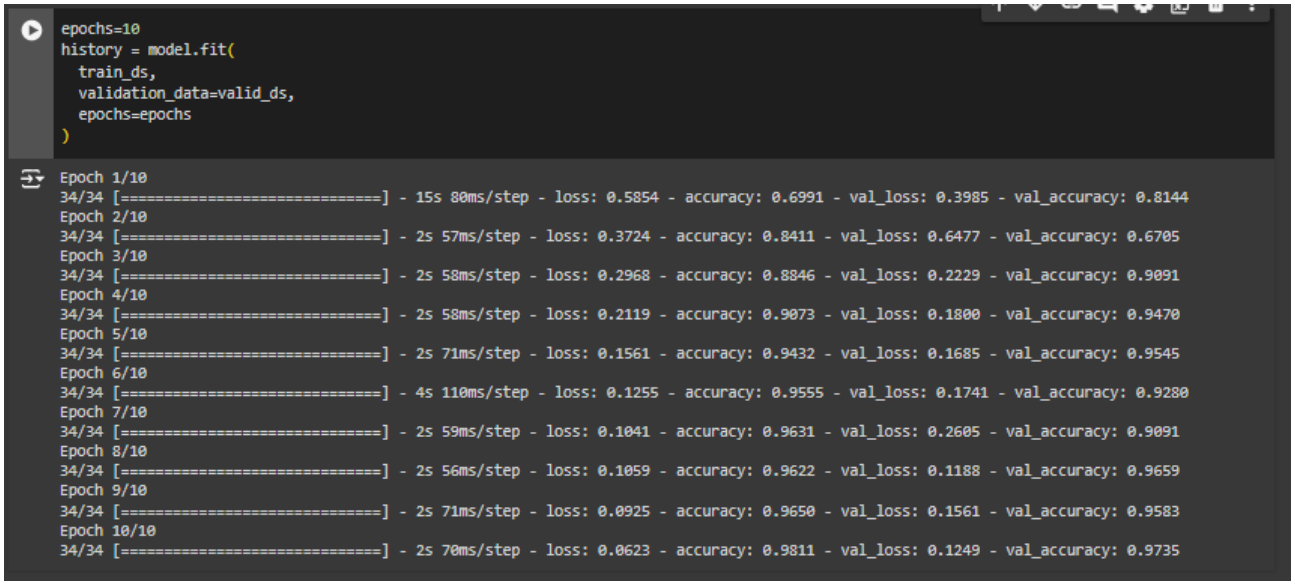
Como podemos ver na figura 3 o código define um modelo de rede neural convolucional (CNN) para classificação de imagens em duas classes. Em seguida, o modelo é construído usando a API Sequential do Keras. A primeira camada é uma camada de normalização (Rescaling), que ajusta os valores dos pixels das imagens para o intervalo [0, 1] (Abadi, *et al.*,2015).Depois, são adicionadas

três camadas de convolução (Conv2D) com 16, 32 filtros, respectivamente, cada uma usando filtros 3x3 e padding ‘same’ para manter as dimensões da entrada. Após cada camada de convolução, há uma camada de pooling (MaxPooling2D) que é um tensor 3D de forma (altura, largura, canais) para reduzir a dimensionalidade espacial dos mapas de características (Abadi, *et al.*,2015)

Após as camadas de convolução e pooling, os mapas de características são achatados em um vetor 1D usando a camada Flatten. Em seguida, uma camada totalmente conectada (Dense) com 128 neurônios e ativação ReLU é adicionada. Como foi citado acima as camadas densas recebem um vetor 1D de entrada, porém sua saida é um tensor 3D. Primeiro ocorre o achatamento da saída 3D para 1D e logo após adiciona uma ou mais camadas densas (Abadi, *et al.*,2015). Finalmente, a camada de saída (Dense) tem dois neurônios, correspondendo ao número de classes, e não usa função de ativação, pois a função de perda SparseCategoricalCrossentropy será usada com from_logits=True. (Abadi, *et al.*,2015)

O modelo é então compilado usando o otimizador Adam (figura 2), a função de perda Sparse Categorical Crossentropy, adequada para problemas de classificação com rótulos inteiros, e a métrica de acurácia para monitorar o desempenho do modelo. Por fim, o resumo do modelo é exibido, mostrando a arquitetura e o número de parâmetros treináveis em cada camadacount.

Foi então configurado o treinamento do modelo de rede neural definida anteriormente. Epochs=10 define que o modelo será treinado por 10 épocas. Uma época é um ciclo completo através de todo o conjunto de dados de treinamento. O conjunto de dados de treinamento train_ds será usado para ajustar os pesos do modelo. Já o conjunto de dados validation_data=valid_ds é referente a validação que será usado para avaliar o desempenho do modelo após cada época de treinamento. O método fit treina o modelo usando os dados de treinamento e validação, e retorna um objeto history que contém informações sobre o treinamento, como a perda e a acurácia em cada época. Permitindo a visualização e analise o desempenho do modelo ao longo do tempo.



```
epochs=10
history = model.fit(
  train_ds,
  validation_data=valid_ds,
  epochs=epochs
)
```

```
Epoch 1/10
34/34 [==============================] - 15s 80ms/step - loss: 0.5854 - accuracy: 0.6991 - val_loss: 0.3985 - val_accuracy: 0.8144
Epoch 2/10
34/34 [==============================] - 2s 57ms/step - loss: 0.3724 - accuracy: 0.8411 - val_loss: 0.6477 - val_accuracy: 0.6705
Epoch 3/10
34/34 [==============================] - 2s 58ms/step - loss: 0.2968 - accuracy: 0.8846 - val_loss: 0.2229 - val_accuracy: 0.9091
Epoch 4/10
34/34 [==============================] - 2s 58ms/step - loss: 0.2119 - accuracy: 0.9073 - val_loss: 0.1800 - val_accuracy: 0.9470
Epoch 5/10
34/34 [==============================] - 2s 71ms/step - loss: 0.1561 - accuracy: 0.9432 - val_loss: 0.1685 - val_accuracy: 0.9545
Epoch 6/10
34/34 [==============================] - 4s 110ms/step - loss: 0.1255 - accuracy: 0.9555 - val_loss: 0.1741 - val_accuracy: 0.9280
Epoch 7/10
34/34 [==============================] - 2s 59ms/step - loss: 0.1041 - accuracy: 0.9631 - val_loss: 0.2605 - val_accuracy: 0.9091
Epoch 8/10
34/34 [==============================] - 2s 56ms/step - loss: 0.1059 - accuracy: 0.9622 - val_loss: 0.1188 - val_accuracy: 0.9659
Epoch 9/10
34/34 [==============================] - 2s 71ms/step - loss: 0.0925 - accuracy: 0.9650 - val_loss: 0.1561 - val_accuracy: 0.9583
Epoch 10/10
34/34 [==============================] - 2s 70ms/step - loss: 0.0623 - accuracy: 0.9811 - val_loss: 0.1249 - val_accuracy: 0.9735
```

Figura 4 - Épocas do modelo

Os gráficos abaixo (Figura 5) que são provenientes dos códigos da figura 4, mostram a acurácia e a perda do treinamento e da validação ao longo das épocas. O gráfico da esquerda mostra a acurácia, com a linha azul representando a acurácia do treinamento e a linha laranja representando a acurácia da validação. O gráfico da direita mostra a perda, com a linha azul representando a perda do treinamento e a linha laranja representando a perda da validação.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Figura 5 - Parametros para o gráfico

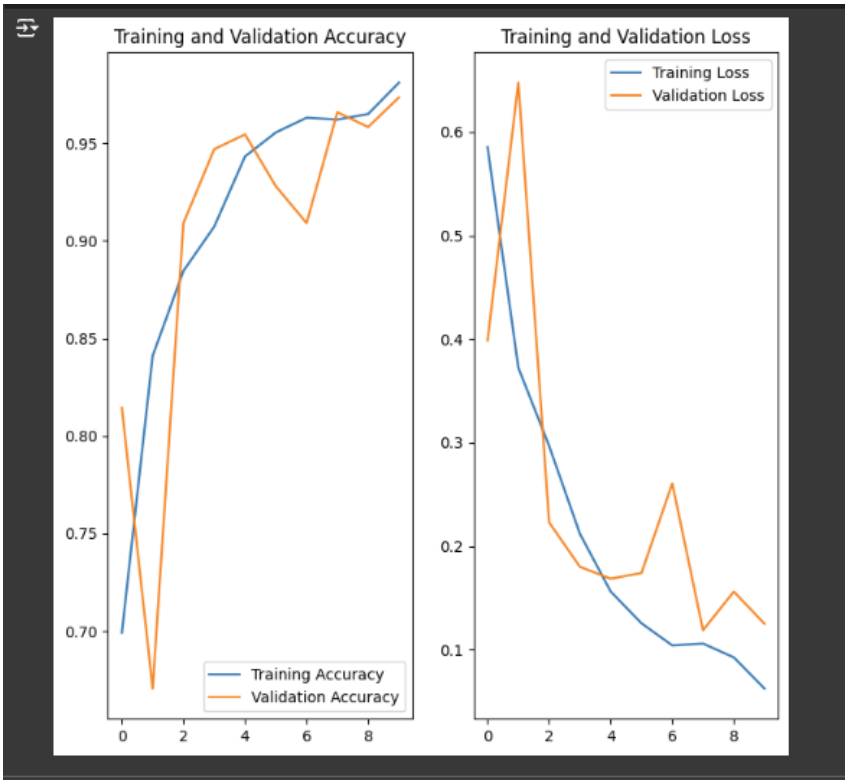



Figura 6 - Resultado Gráfico do Modelo

Foi então realizado o carregamento do modelo treinado para validação, e realizando uma previsão das imagens que não foram utilizadas na etapa de treino. (Figura 7)

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np

# Carregar o modelo treinado
modelo_carregado = load_model(model_path)

# Carregar e preparar a imagem de entrada
imagem_path = '/content/drive/MyDrive/IC RENATO/data/ref/v1/axial/valid/1/MS-A (578).png'
img = image.load_img(imagem_path, target_size=(256, 256))  # Ajuste a altura e largura conforme necessário
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = img_array / 255.0  # Normalização dos valores de pixel para o intervalo [0, 1]

# Fazer a previsão usando o modelo carregado
previsao = modelo_carregado.predict(img_array)

# Exibir a classe prevista ou as probabilidades para cada classe, dependendo do tipo de tarefa
# (classificação binária, classificação multiclasse, regressão, etc.)
print(previsao)
# Valores das saídas do modelo
saida_modelo = np.array(previsao)


print("Probabilidades:", saida_modelo)
# [[-1.1300071  1.0530198]]
# [[-1.1305603  1.0539908]]
# [[-1.1334138  1.0570738]]
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7ce39c20ac20> triggered
1/1 [==============================] - 0s 79ms/step
[[-1.1334138  1.0570738]]
Probabilidades: [[-1.1334138  1.0570738]]
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def softmax(x):
    e_x = np.exp(x - np.max(x))  # Lidando com possíveis problemas de estabilidade numérica
    return e_x / e_x.sum(axis=0)
```

Figura 7 - Validação do Modelo

Um resultado muito parecido foi obtido na parte Sagital

```
Epoch 1/10
36/36 [==============================] - 32s 873ms/step - loss: 0.4647 - accuracy: 0.7905 - val_loss: 0.2687 - val_accuracy: 0.9046
Epoch 2/10
36/36 [==============================] - 4s 103ms/step - loss: 0.2779 - accuracy: 0.8759 - val_loss: 0.2184 - val_accuracy: 0.9081
Epoch 3/10
36/36 [==============================] - 4s 104ms/step - loss: 0.1659 - accuracy: 0.9313 - val_loss: 0.1403 - val_accuracy: 0.9576
Epoch 4/10
36/36 [==============================] - 5s 140ms/step - loss: 0.0923 - accuracy: 0.9630 - val_loss: 0.1014 - val_accuracy: 0.9682
Epoch 5/10
36/36 [==============================] - 4s 106ms/step - loss: 0.0535 - accuracy: 0.9806 - val_loss: 0.1898 - val_accuracy: 0.9329
Epoch 6/10
36/36 [==============================] - 4s 104ms/step - loss: 0.0497 - accuracy: 0.9798 - val_loss: 0.0900 - val_accuracy: 0.9753
Epoch 7/10
36/36 [==============================] - 5s 150ms/step - loss: 0.0215 - accuracy: 0.9930 - val_loss: 0.0568 - val_accuracy: 0.9753
Epoch 8/10
36/36 [==============================] - 4s 104ms/step - loss: 0.0251 - accuracy: 0.9912 - val_loss: 0.1047 - val_accuracy: 0.9717
Epoch 9/10
36/36 [==============================] - 4s 109ms/step - loss: 0.0093 - accuracy: 0.9982 - val_loss: 0.0804 - val_accuracy: 0.9753
Epoch 10/10
36/36 [==============================] - 5s 153ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.1081 - val_accuracy: 0.9717
```

Figura 8 - Treinamento parte Sagital

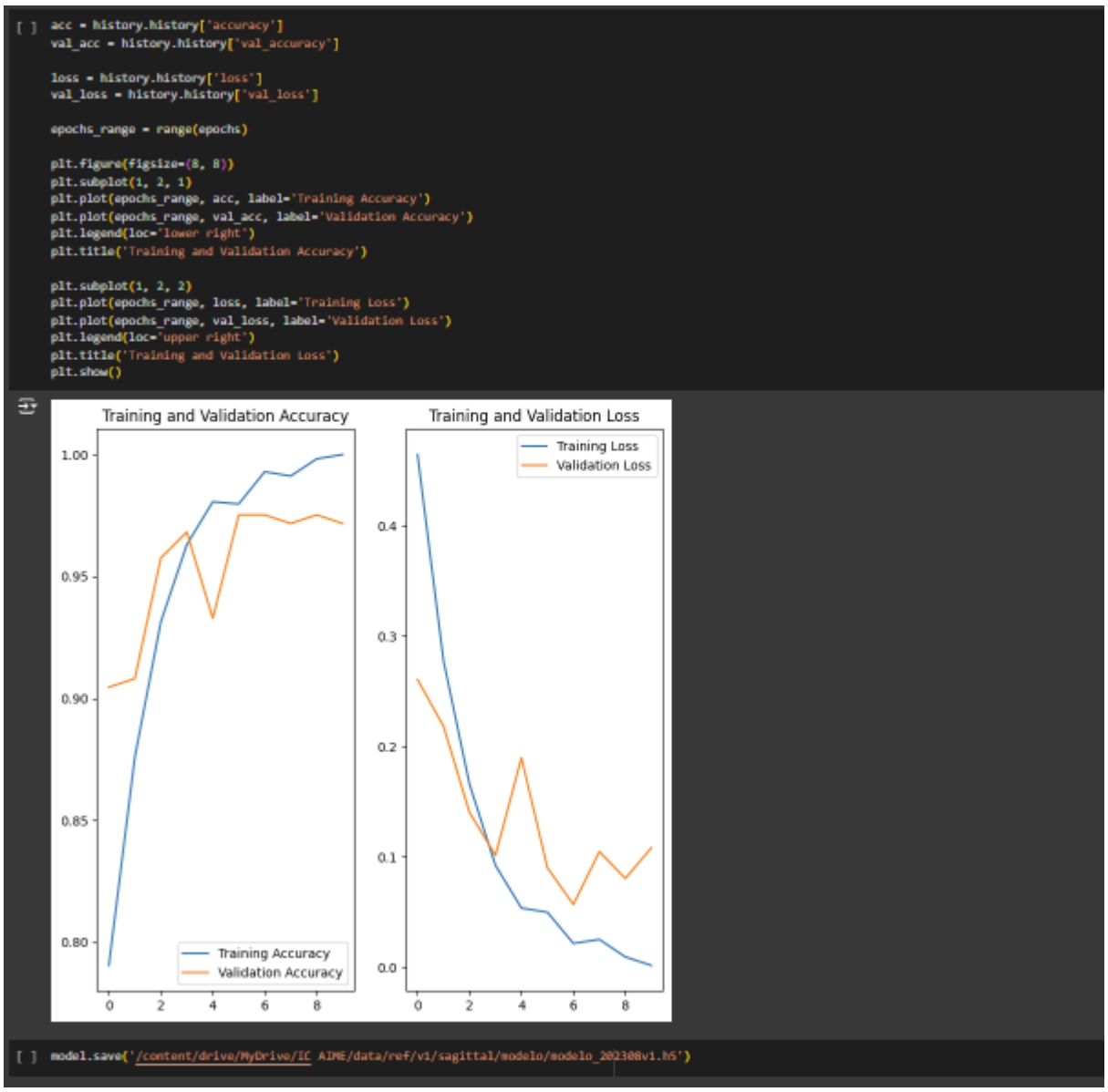



Figura 9 - Resultado Gráfico parte Sagital

Como podemos perceber nos gráficos o modelo apresentou *overfitting*. O *overfitting* acontece quando um modelo apresenta um ótimo desempenho nos dados de treinamento, mas falha em generalizar bem para novos dados. Isso pode ocorrer devido a uma série de fatores, como o desequilíbrio dos dados de treinamento, complexidade exacerbada do modelo, quantidade de imagens utilizadas para criação do modelo, entre outras (Xu,Pirani, Jiang, 2023). Quando ocorre *overfitting* o modelo aprende não apenas os padrões relevantes, mas também o “ruído” ou as variações aleatórias

presentes nos dados de treinamento. Como resultado, o modelo tem um desempenho excelente nos dados de treinamento, mas seu desempenho em dados novos ou de teste é insatisfatório.

Para a segunda tentativa foi criado um único modelo para prever axial e sagital com algumas mudanças na seleção do dataset e parâmetros do modelo de RNN. Assim como no primeiro modelo foi realizado a preparação e pré-processamento de dados de imagens de ressonância magnética, com o objetivo de alimentar um modelo de machine learning para a detecção de lesões relacionadas à esclerose múltipla. Inicialmente, ele organiza os caminhos dos dados brutos, separando as imagens de pacientes com esclerose múltipla e de indivíduos do grupo de controle. Em seguida, utiliza a função glob para coletar todas as imagens com extensão .png dessas categorias (Axial e Sagittal). As imagens coletadas são armazenadas na variável dataset, enquanto as etiquetas correspondentes (1 para esclerose múltipla e 0 para controle) são adicionadas à variável label. Figura10.

Além disso, o código faz o redimensionamento das imagens para um tamanho padronizado de 255x255 pixels, o que facilita o processamento pelo modelo de redes neurais. Esse redimensionamento é crucial para garantir consistência nos dados de entrada, o que, por sua vez, melhora a eficiência e precisão do modelo. Todo esse processo de organização, rotulação e normalização dos dados visa preparar um conjunto de treino adequado para o desenvolvimento de um modelo capaz de identificar anomalias em imagens de ressonância magnética de pacientes com esclerose múltipla. Fig.10

```
[ ]  path_raw = '/content/drive/MyDrive/IC RENATO/data/raw/Multiple Sclerosis/'

     dataset = []
     label   = []

     ##########################################
     # Separacao randomica de treino / teste
     # Normalizacao dos dados
     ##########################################
     path_target = '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/1/'

     for img in list(pathlib.Path(path_raw + '1/MS-Axial/').glob('*.png')) + list(pathlib.Path(path_raw + '1/MS-Sagittal/').glob('*.png')):
       # cria_dir(path_target)
       # shutil.copy(img, path_target)
       dataset.append(path_target + str(img).split('/')[-1])
       label.append(1)

     path_control = '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/0/'
     for img in list(pathlib.Path(path_raw + '0/Control-Axial/').glob('*.png')) + list(pathlib.Path(path_raw + '0/Control-Sagittal/').glob('*.png')):
       # cria_dir(path_control)
       # shutil.copy(img, path_control)
       dataset.append(path_control + str(img).split('/')[-1])
       label.append(0)

[ ]  # redimensionando as imagens
     paths_redimensionar = [
         '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/0/',
         '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/1/'
     ]

     for paths in paths_redimensionar:
       re_size(path=paths, new_size=(255,255), tp_img='png')
```

Figura 10 – Início Segundo Modelo

Após é realizada a implementação um modelo de rede neural convolucional (CNN) para classificar imagens de ressonância magnética entre pacientes com esclerose múltipla e controles. Primeiramente, as imagens são carregadas, rotuladas e divididas em conjuntos de treino e teste. Em seguida, elas são normalizadas e as classes são transformadas em categorias (one-hot encoding). O modelo é construído com camadas convolucionais para extrair características das imagens, seguidas por camadas de pooling para reduzir a dimensionalidade e camadas de dropout para evitar overfitting. Após as camadas convolucionais, o modelo é achatado e passa por camadas densas para realizar a classificação final. A função de perda usada é a categorical_crossentropy, e o otimizador escolhido é o Adam. O treinamento é feito com base nos dados normalizados e divididos, buscando maximizar a acurácia da classificação das imagens entre esclerose múltipla e controle. Fig 11

```
[ ]  dataset = []
     label   = []

     path_control = '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/0/'
     path_target = '/content/drive/MyDrive/IC RENATO/data/ref/v3/train/1/'

     for img in list(pathlib.Path(path_target).glob('*.png')) :
       dataset.append(cv2.imread(str(img)))
       label.append(1)

     for img in list(pathlib.Path(path_control).glob('*.png')) :
       dataset.append(cv2.imread(str(img)))
       label.append(0)

     X_train, X_test, y_train, y_test = train_test_split(dataset, label, test_size = 0.20, random_state = 1337)
```

```
[ ]  X_train = np.array(X_train) / 255.
     X_test = np.array(X_test) /255.

     y_train = tf.keras.utils.to_categorical(y_train)
     y_test = tf.keras.utils.to_categorical(y_test)
```

```
     num_classes = 2

     model = Sequential([
       layers.Conv2D(16, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Conv2D(32, 3, padding='same', activation='relu'),
       layers.MaxPooling2D(),
       layers.Dropout(0.2),
       layers.Conv2D(64, 3, padding='same', activation='relu'),
       layers.Dropout(0.2),
       layers.MaxPooling2D(),
       layers.Flatten(),
       layers.Dense(128, activation='relu'),
       layers.Dense(num_classes)
     ])

     model.compile(loss = "categorical_crossentropy",
                   optimizer = Adam(learning_rate=0.0001), metrics=["accuracy"])

     history = model.fit(X_train, y_train, batch_size=8, epochs=10, verbose = 1,
                         validation_data=(X_test,y_test))
```

Figura 11 – Carregamento imagens e implementação do modelo

Como resultado dois gráficos são gerados que comparam o desempenho do modelo durante o treinamento e a validação. O gráfico de **acurácia** mostra que o modelo está aprendendo bem, com as linhas de treino e validação subindo de forma consistente e permanecendo próximas, indicando uma boa generalização. O gráfico de **perda** começa com uma variação maior, mas depois ambas as linhas (treino e validação) diminuem gradualmente, o que é esperado conforme o modelo ajusta seus pesos para melhorar suas previsões e reduzir o erro. Fig 12.

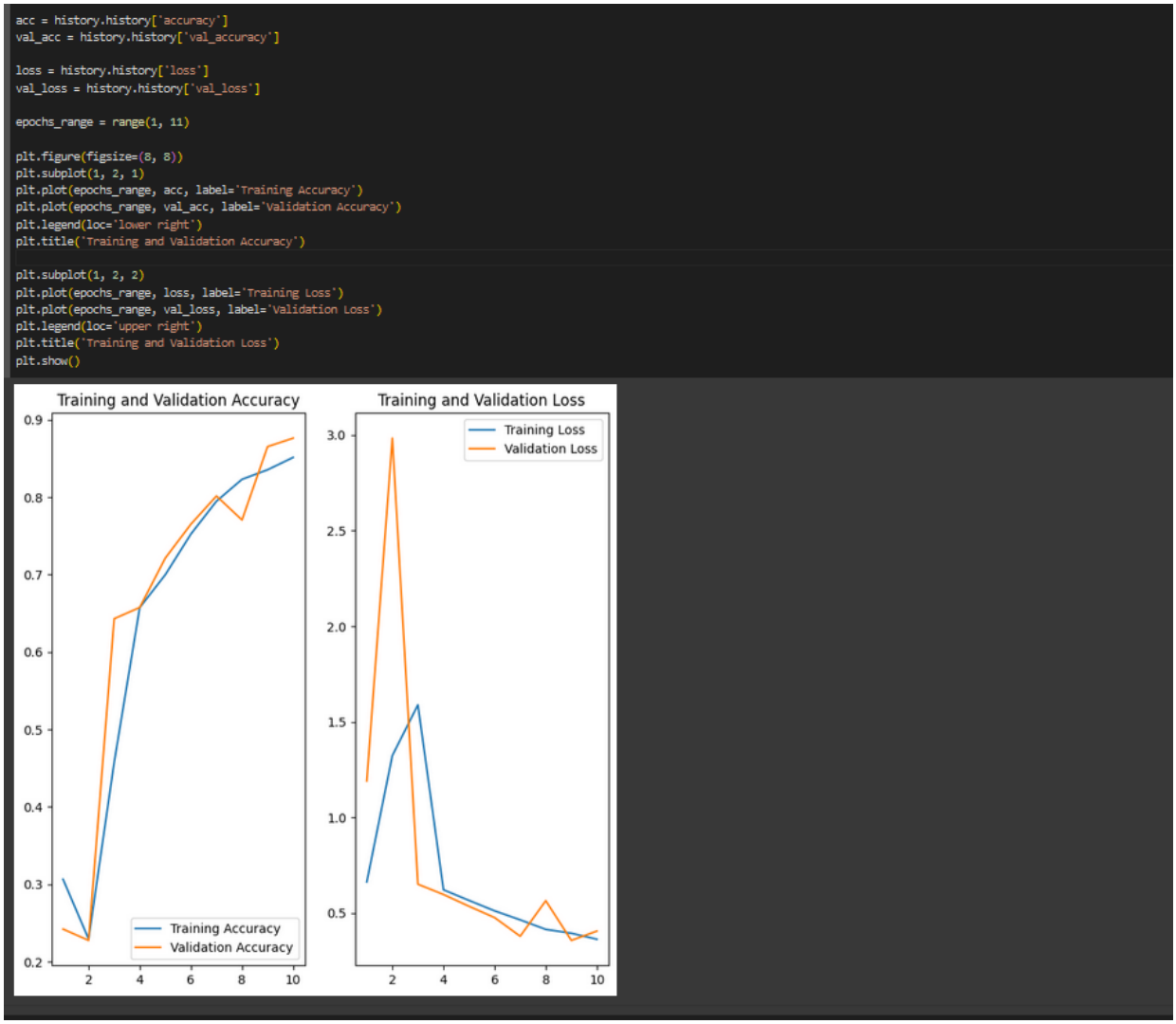



Figura 12 – Resultado Gráfico

O modelo treinado é salvo em um arquivo para reutilização futura, e então uma imagem é escolhida manualmente do conjunto de validação para comparar a previsão do modelo com o rótulo real, que não foi utilizada durante o treinamento e teste, para verificar a capacidade do modelo de

fazer previsões em novos dados. A imagem é normalizada para que os valores de pixel fiquem no intervalo de 0 a 1, e a previsão é feita utilizando o modelo salvo. Além disso, é realizada uma avaliação do modelo no conjunto de dados de teste, obtendo uma acurácia de **87,61%**. O modelo conseguiu prever corretamente a classe dessa imagem, mostrando que ele está funcionando de forma eficaz para identificar a classe correta, tanto em novos exemplos quanto no conjunto de teste. Fig 13.

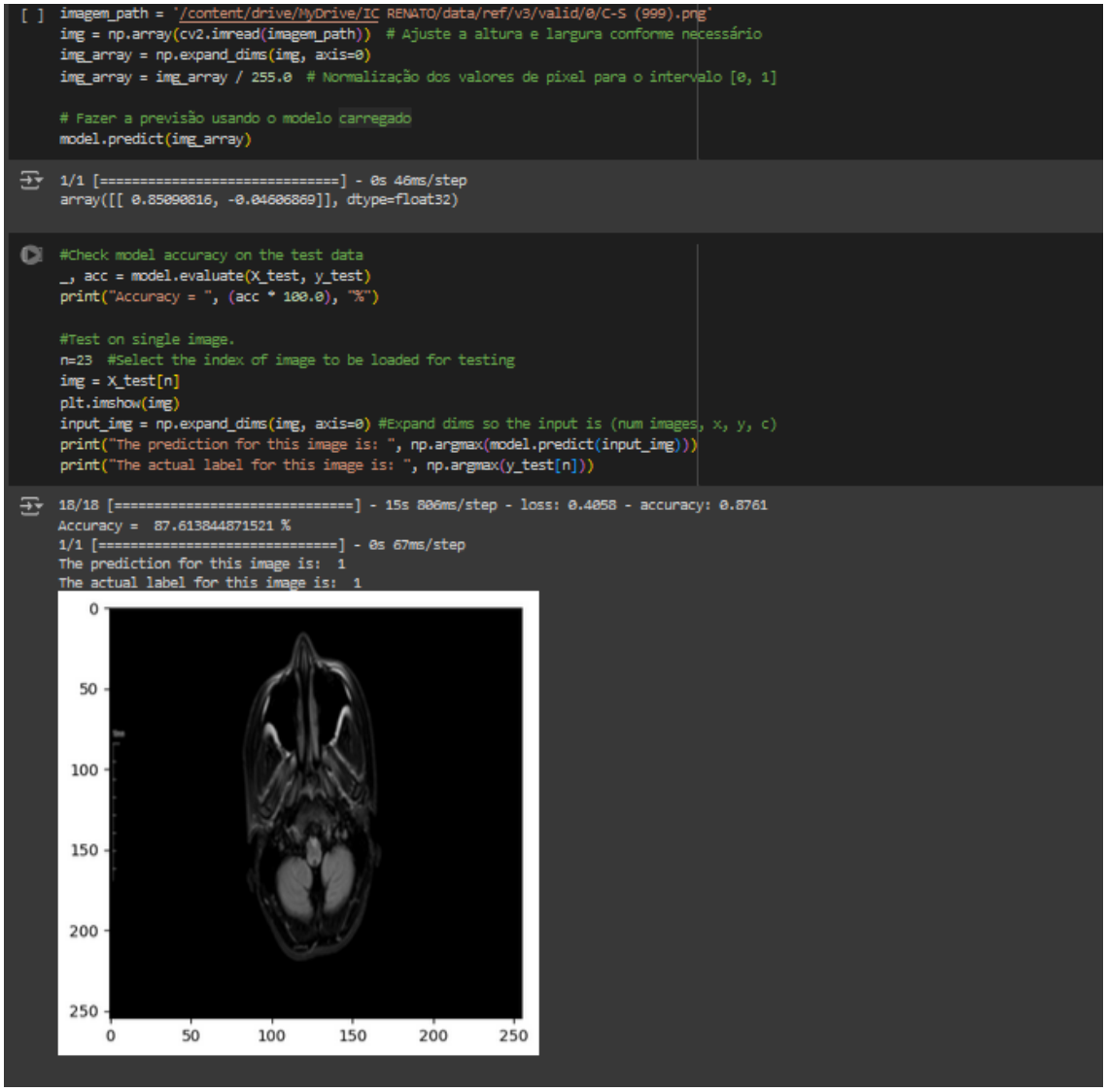



Figura 13 – Teste modelo com imagem do grupo de validação

Além de avaliar o modelo por meio de acurácia, também foi gerada uma matriz de confusão para entender melhor o desempenho do modelo na classificação de cada classe. A matriz de confusão

indica o número de previsões corretas e incorretas para cada classe, fornecendo mais detalhes sobre onde o modelo está acertando ou errando. Fig. 14.

No caso, a matriz de confusão gerada apresentou os seguintes valores:

- **3102** verdadeiros negativos (00): o modelo classificou corretamente 3102 imagens como não tendo lesões de esclerose múltipla.
- **48** falsos negativos (10): o modelo classificou 48 imagens como não tendo lesões, mas elas realmente tinham.
- **20** falsos positivos (01): o modelo classificou 20 imagens como tendo lesões, mas elas não tinham.
- **1702** verdadeiros positivos (11): o modelo classificou corretamente 1702 imagens como tendo lesões de esclerose múltipla.

A visualização da matriz de confusão foi feita usando o *Seaborn*, que facilita a interpretação dos resultados ao gerar um mapa de calor. Isso ajuda a identificar com mais clareza os acertos e erros do modelo, proporcionando uma análise mais detalhada do seu desempenho.

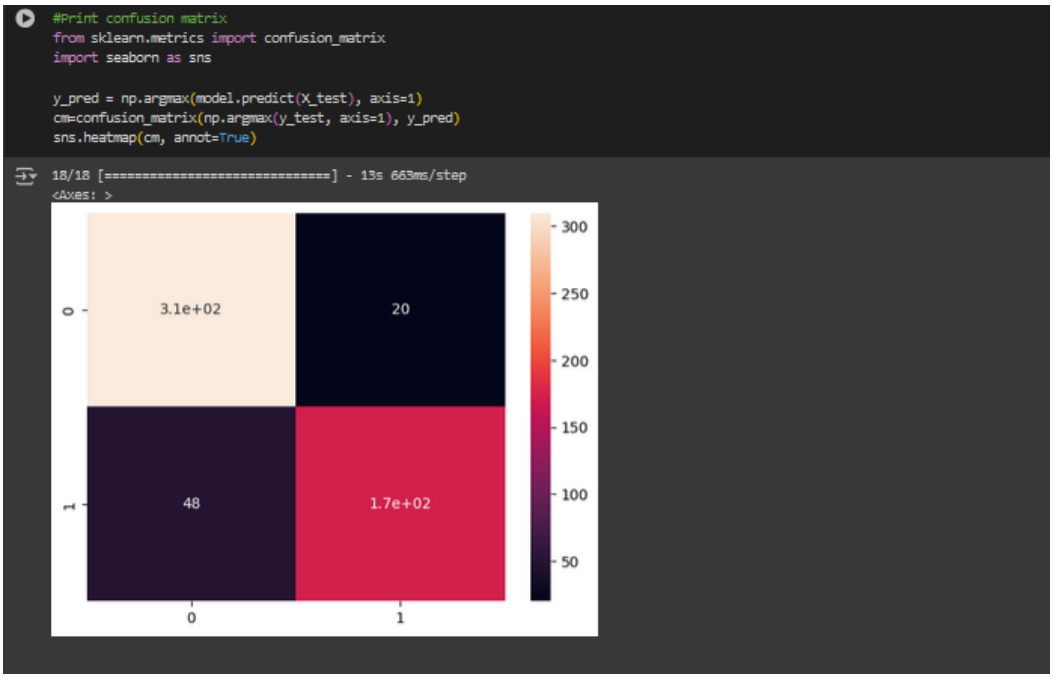



Figura 14 – Matriz de Confusão

Neste trecho final, os dados de treinamento e validação são carregados a partir de um diretório contendo as imagens de ressonância magnética classificadas em duas classes (lesões de esclerose múltipla e controle). O dataset é dividido em 80% para treinamento e 20% para validação, resultando em 2193 imagens para o treinamento e 548 para a validação.

O modelo utilizado é uma rede neural convolucional (*CNN*), que inclui camadas de normalização, convolução, pooling, *dropout* e camadas densas. A primeira camada rescale as imagens, ajustando os valores dos pixels para o intervalo [0, 1]. A rede tem três blocos de convolução e pooling, seguidos de uma camada densa de 128 unidades e uma última camada que realiza a classificação em duas classes (lesões e controle).

O otimizador utilizado é o Adam com uma taxa de aprendizado de 0,0001. A função de perda usada é a Sparse Categorical Crossentropy, apropriada para classificação multi-classe, e a métrica monitorada é a acurácia.

O modelo é treinado por 10 épocas, utilizando o dataset de treinamento e validando o desempenho a cada época com o conjunto de validação. A saída final do código exibe o progresso do treinamento, mostrando como o modelo está aprendendo a classificar as imagens corretamente com o passar das épocas. Fig.15.

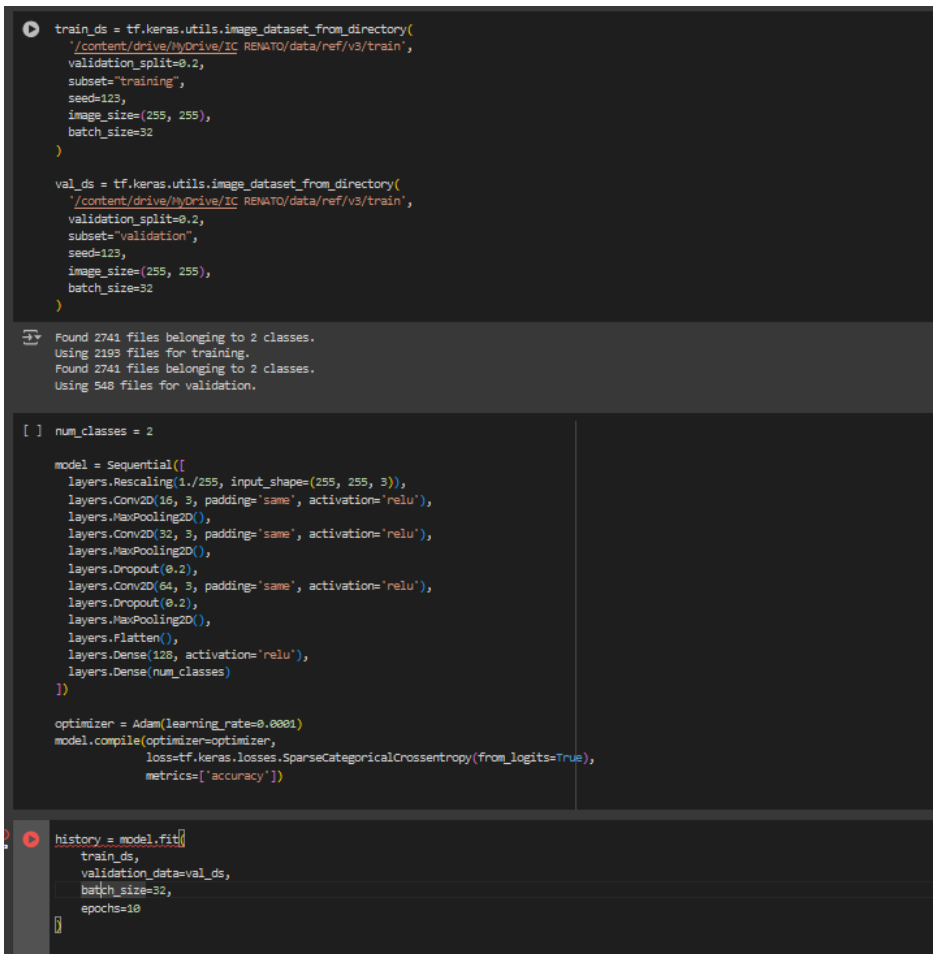



Figura 15 – Finalização do Modelo

# 5 CONCLUSÃO

Com base nos experimentos realizados, conclui-se que o uso de redes neurais convolucionais (CNNs) para a classificação de lesões de esclerose múltipla em imagens de ressonância magnética apresentou resultados promissores. O modelo alcançou uma acurácia de aproximadamente 87,6% no conjunto de teste, demonstrando uma boa capacidade de generalização. Isso indica que o modelo foi capaz de identificar corretamente a presença ou ausência de lesões na maioria das imagens não vistas, o que é importante em um contexto de diagnóstico médico.

A matriz de confusão revelou que o modelo cometeu poucos erros, com apenas 20 falsos positivos e 48 falsos negativos. A baixa taxa de falsos negativos é crucial em diagnósticos, pois garante que poucos casos de esclerose múltipla passariam despercebidos. O uso de técnicas de normalização, redimensionamento das imagens e camadas convolucionais permitiu ao modelo extrair características relevantes das imagens, tornando a classificação eficaz. Além disso, o uso de camadas de *dropout* foi importante para reduzir o risco de *overfitting*, o que foi confirmado pela proximidade entre as curvas de acurácia de treino e validação, demonstrando um aprendizado estável.

O treinamento do modelo foi realizado de maneira eficiente, e os gráficos de acurácia e perda ao longo das épocas indicam que o modelo manteve um bom desempenho sem superajustar aos dados de treino, o que é um ponto positivo. A quantidade de dados utilizada foi moderada (2741 imagens no total), e o modelo se mostrou eficaz com esse conjunto de dados. Isso sugere que, com mais dados e ajustes finos, o desempenho poderia ser ainda mais aprimorado.

Embora os resultados sejam encorajadores, ainda há espaço para melhorias antes que o modelo possa ser utilizado em um ambiente clínico. O aumento do conjunto de dados, o ajuste de hiperparâmetros e o uso de arquiteturas mais complexas poderiam aumentar a precisão e reduzir ainda mais os erros. De forma geral, este projeto demonstrou que é possível desenvolver um modelo de CNN eficaz para identificar lesões de esclerose múltipla, com potencial para auxiliar diagnósticos médicos, complementando a avaliação de especialistas e ajudando na detecção precoce da doença.

# REFERÊNCIAS


ABADI, Martin *et al*. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Software available from tensorflow.org.

ARANI, L.A.; HOSSEINI, A.; ASADI, F.; MASOUD, S.A.; NAZEMI, E. **Intelligent computer systems for multiple sclerosis diagnosis: A systematic review of reasoning techniques and methods.** Acta Inform. Med. 26, 258–264, 2018.

ASLAM, N; KHAN, I.U; BASHAMAKH, A.; ALGHOOL, F.A.; ABOULNOUR, M.; ALSUWAYAN, N.M.; ALTURAIF, R.A.; BRAHIMI, S.; ALJAMEEL, S.S.; GHAMDI, K.A. **Multiple Sclerosis Diagnosis Using Machine Learning and Deep Learning: Challenges and Opportunities.** MPDI, *22*(20), 7856, 2022. Disponível em https://doi.org/10.3390/s22207856 Acesso em 21 de Outubro de 2024.

CARNEIRO, T.; NÓBREGA, R.V.M; NEPOMUCENO, T.; BIAN, G.B.; ALBUQUERQUE, V, H, C.; FILHO, P,P,R., **Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications**. IEEE**,** 2018. Disponível em https://ieeexplore.ieee.org/document/8485684 Acesso em: 01 fev. 2023.

DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF CYPRUS. **Datasets**. Disponível em: https://www.medinfo.cs.ucy.ac.cy/index.php/facilities/32-software/218-datasets. Acesso em: 01 fev. 2023.

DOBSON, R.; GIOVANNONI, G.. **Multiple sclerosis – a review**. **E**uropean Journal Of Neurology**.** Londres, p. 27-40. out. 2018.

FILIPPI, Massimo *et al*. **MRI criteria for the diagnosis of multiple sclerosis: MAGNIMS consensus guidelines**. **Lancet Neurol,** Londres, v. 16, n. , p. 1474-4422, jan. 2016.

FILIPPI, Massimo *et al*. **Association between pathological and MRI findings in multiple sclerosis**. Lancet Neurol, Londres, v. 18, n. , p. 198-210, fev. 2019.

LEON-SANCHES, Edgar Rafael Ponce, *et al*. **A Deep Le.arning Approach for Predicting Multiple Sclerosis.** 29;14(4):749. doi: 10.3390/mi14040749, 2023

LONGITUDINAL Multiple Sclerosis Lesion Imaging Archive. 2014. Disponível em: https://www.nitrc.org/projects/longitudinal_ms/. Acesso em: 01 fev. 2023

LUNDERVOLD, Alexander, S; LUNDERVOLD, **Arvid. An overview of deep learning in medical imagingfocusing on MRI**. ZEMEDI-10775**,** 2018**.**

MUSLIM, Ali**.** *et al*. **Brain MRI dataset of multiple sclerosis with consensus manual lesion segmentation and patient meta information**, Data in Brief**,** 2022**.**

PINTO, Mauro F. *et al*. **Prediction of disease progression and outcomes in multiple sclerosis with machine learning**. Scientific Reports, [*s. l*], v. 1038, n. 10, p. 1-13, fev. 2020.

SECCIA, R.; ROMANO, S.; SALVETTI, M.; CRISANTI, A.; PALAGI, L.; GRASSI, F. **Machine learning use for prognostic purposes in multiple sclerosis**. Life. 11, 122, 2021

SHATTE, Adrian,B,R; HUTCHINSON, Delyse M; TEAGUE, Samantha J. **Machine learning in mental health: a scoping review of methods and applications**. Psychological Medicine**,** 2019.

SHOEIBI, Afshin *et al.* **Applications of Deep Learning Techniques for Automated Multiple Sclerosis Detection Using Magnetic Resonance Imaging: A Review**, 2021

TENSORFLOW. **Por que usar o TensorFlow**. Disponível em: https://www.tensorflow.org/about?hl=pt-br. Acesso em: 01 fev. 2023.

XU, Pablo, PIRANI ,Coen , JIANG ,Xia. **Empirical Study of Overfitting in Deep Learning for Predicting Breast Cancer Metastasis Chuhan.** Cancers (Basel), 15(7). doi: 10.3390/cancers15071969, 2023.

ZHANG, Chiyuan, *et al*. **Understanding Deep Learning (Still) Requires Rethinking Generalization.** Communications of the acm, 2021.