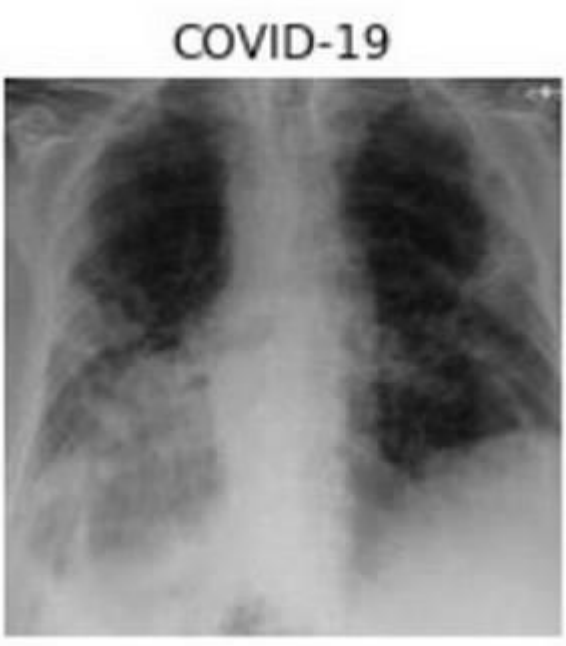
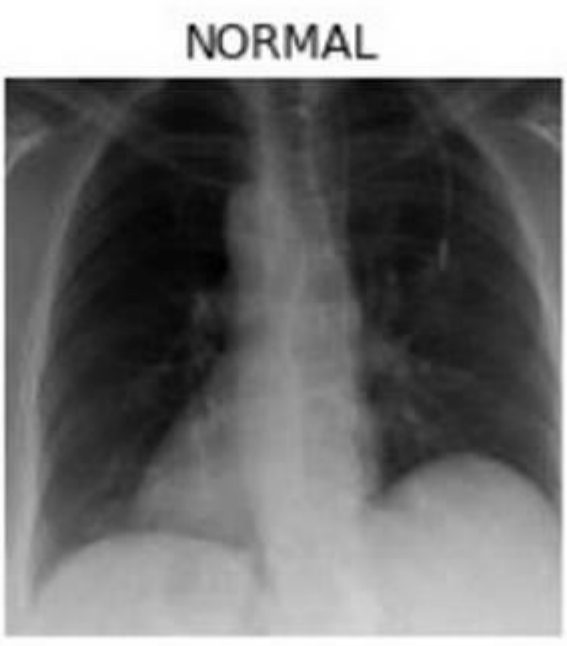
@Odemir Depieri Jr

Prevendo doença Pulmonar





com Deep Learning

TensorFlow

pillow

pandas

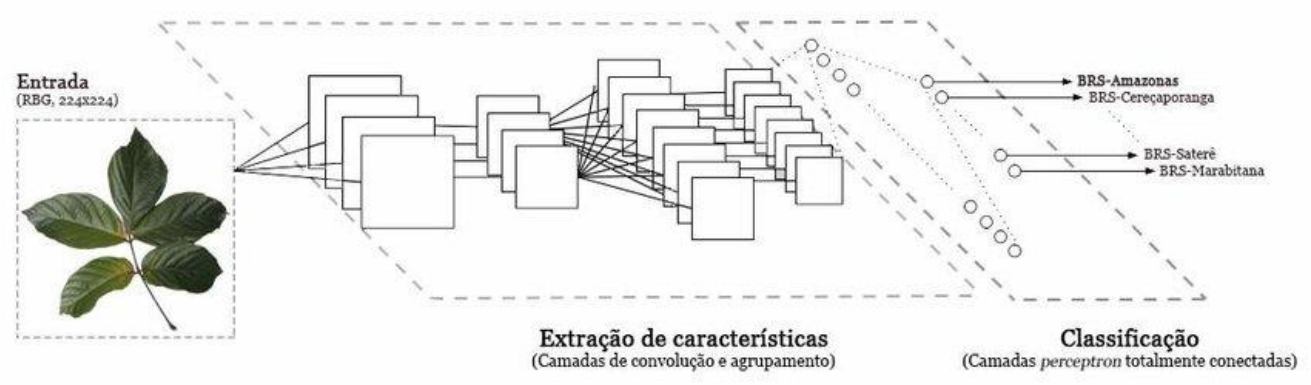
matplotlib

Deep Learning

Redes Convolucionais

Uma **Rede Neural Convolucional** (ConvNet) é um algoritmo de **aprendizado profundo** que pode captar uma imagem de entrada e atribuir importâncias como pesos e vieses a vários aspectos e objetos da imagem e ser capaz de diferenciar umas das outras.

https://www.aliger.com.br/blog/as-redes-neuronais-convolutivas-no-deep-learning/



Artigos para leitura

https://www.deeplearningbook.com.br/introducao-as-redes-neurais-convolucionais/

https://www.ime.unicamp.br/~jbflorindo/Teaching/2018/MT530/T10.pdf

Vídeos

https://www.youtube.com/watch?v=7dsDHb6qKYI

https://www.youtube.com/watch?v=yN9L9mnPyBA

https://www.youtube.com/watch?v=DXnyuUZcAAI

Vamos utilizar os dados da Kaggle

https://www.kaggle.com/praveengovi/coronahack-chest-xraydataset

Nesse caso eu fiz o upload das imagens para google driver.
Você pode fazer usando o diretório local da sua maquina.

@Odemir Depieri Jr

Vamos importar as Lib necessárias

```
[ ] # Lib para modelagem de dados
    import pandas as pd
    # Lib para recursos de matrizes
    import numpy as np
    # Lib para plotagem gráfica
    import matplotlib.pyplot as plt
    # Lib para plotagem gráfica
    import seaborn as sns
    # Lib para carregar imagens pelo MatplotLib
    import matplotlib.pyplot as mpimg

    # Lib para trabalhar com imagens
    import PIL.Image
    # Lib para acessar recursos do Sistema Operacional
    import os

    # Lib para utilizar as funções do TensorFlow
    import tensorflow as tf

    # Funções  para recursos do Tensor Flow
    from tensorflow import keras
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import  *
    from tensorflow.keras.optimizers import Adam, SGD, RMSprop
    from tensorflow.keras.applications import DenseNet121, VGG19, ResNet50

    # Lib para veririfcar o progresso do Loop
    from tqdm import tqdm

    # Funções para extrair os recursos de escala de cores
    from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array

    # Função para carregar imagem pelo Keras
    from tensorflow.keras.preprocessing import image as Image_
```

Vou conectar no meu Driver para pegar as imagens

```
[ ] # Função para conectar no driver
    from google.colab import drive
    # Fazendo a conexão com o google driver
    drive.mount('/content/arquivos_driver')
```

```
Drive already mounted at /content/arquivos_driver; to attempt to forcibly remount, call drive.mount("/content/arquivos_driver", force_remount=Tru
```

Lendo a base de dados

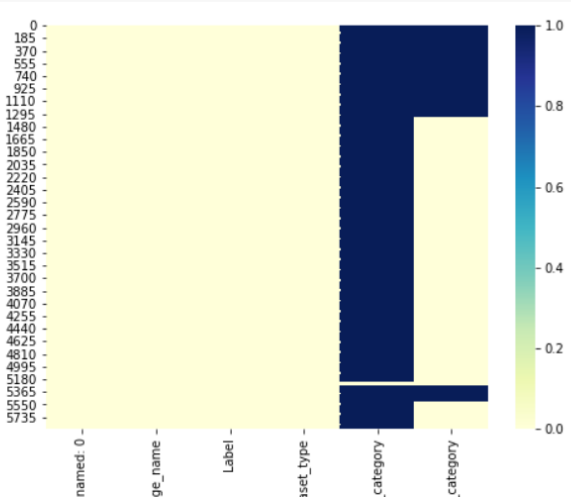
```
[ ] # Lendo a base de dados de treino
    Base_Dados = pd.read_csv('/content/arquivos_driver/MyDrive/Dados_Modelo/Chest_xray_Corona_Metadata.csv')

    # Verificando as primeiras linhas
    Base_Dados.head()
```

	Unnamed: 0	X_ray_image_name	Label	Dataset_type	Label_2_Virus_category	Label_1_Virus_category
0	0	IM-0128-0001.jpeg	Normal	TRAIN	NaN	NaN
1	1	IM-0127-0001.jpeg	Normal	TRAIN	NaN	NaN
2	2	IM-0125-0001.jpeg	Normal	TRAIN	NaN	NaN
3	3	IM-0122-0001.jpeg	Normal	TRAIN	NaN	NaN
4	4	IM-0119-0001.jpeg	Normal	TRAIN	NaN	NaN

```
[ ] # Verificando a Dimensão
    Base_Dados.shape
```

```
(5910, 6)
```

```
[ ] # Verificando se há um valor nulo na base de dados
    # Caso exista haverá linhas no gráfico

    # Definindo o Tamanho da Imagem
    plt.figure( figsize=(8,6) )

    # Plotando o Gráfico
    sns.heatmap( Base_Dados.isnull(), cmap="YlGnBu", cbar=True  );
```



Vamos analisar as classes

```
[ ]  # Analisando as Colunas com a Classificação da Doença

     # 1 Categoria Viral
     print( Base_Dados['Label_1_Virus_category'].value_counts(), '\n' )

     # 2 Categoria Viral
     print( Base_Dados['Label_2_Virus_category'].value_counts(), '\n' )
```

```
bacteria          2777
Virus             1555
Stress-Smoking       2
Name: Label_1_Virus_category, dtype: int64

COVID-19         58
Streptococcus     5
SARS              4
ARDS              2
Name: Label_2_Virus_category, dtype: int64
```

Existe uma amostra **muito pequena de casos de Covid**.

Assim vamos prever apenas **se existe ou não doença pulmonar**.

```
[ ]  # Preenchendo os valores em branco que são casos sem doença
     # Mostrar incluir um "-" no lugar dos valores em branco
     Base_Dados.fillna('-', inplace=True)

     # Calcuando se há campos vazio novamente
     Base_Dados.isnull().sum()
```

```
Unnamed: 0                0
X_ray_image_name          0
Label                     0
Dataset_type              0
Label_2_Virus_category    0
Label_1_Virus_category    0
dtype: int64
```

Vamos gerar a classe que queremos prever

```
[ ]  # Vamos separar as classes que vamos prever

     # Criando a função
     def Classe(row):

       # Filtrando a Linha
       Linha = row

       # Casos sem Doença
       if Linha == '-':
         return 0
       # Casos com Doença Pulmonar
       else:
         return 1

     # Aplicando a Função
     Base_Dados['Classe'] = Base_Dados['Label_1_Virus_category'].apply( Classe )

     # Verificando as Classes que vamos prever
     Base_Dados['Classe'].value_counts()
```

```
1    4334
0    1576
Name: Classe, dtype: int64
```

Selecionando os dados de treino e teste

```
[ ]  # Separando os Dados de Treino e Teste

     # Definindo a Base de Treino
     Base_Treino = Base_Dados[Base_Dados['Dataset_type'] == 'TRAIN'].sample(100)

     # Definindo a Base de Teste
     Base_Teste = Base_Dados[Base_Dados['Dataset_type'] == 'TEST'].sample(20)

     # Verificando o tamanho dos dados de treino e teste
     print('Base de Treino:', Base_Treino.shape , '\n')
     print('Base de Teste:', Base_Teste.shape , '\n')
```

```
Base de Treino: (100, 7)

Base de Teste: (20, 7)
```

Atenção

Para esse case tive que selecionar **apenas uma amostra** das imagens devido o Colab **interromper o processo** devido o uso de memoria.

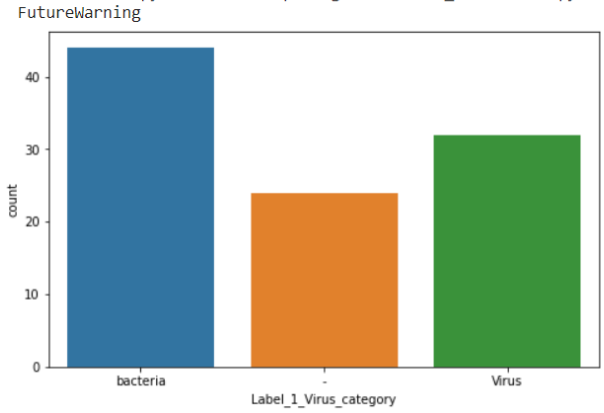
Ideal desse case é fazer na sua própria maquina para pode processar toda essa massa de dados.

O **treinamento da rede neural** estava demorando **+ de 30 minutos** e fora o tempo para extrair todas as escalas de RGB da imagem.

Vamos usar uma amostra nosso do nosso case

```
[ ]  # Plotando a divisão das classes

     # Definindo o Tamanho da Figura
     plt.figure( figsize=(8,5) )

     # Plotando o Gráfico
     sns.countplot(Base_Treino['Label_1_Virus_category']);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.
  FutureWarning
```

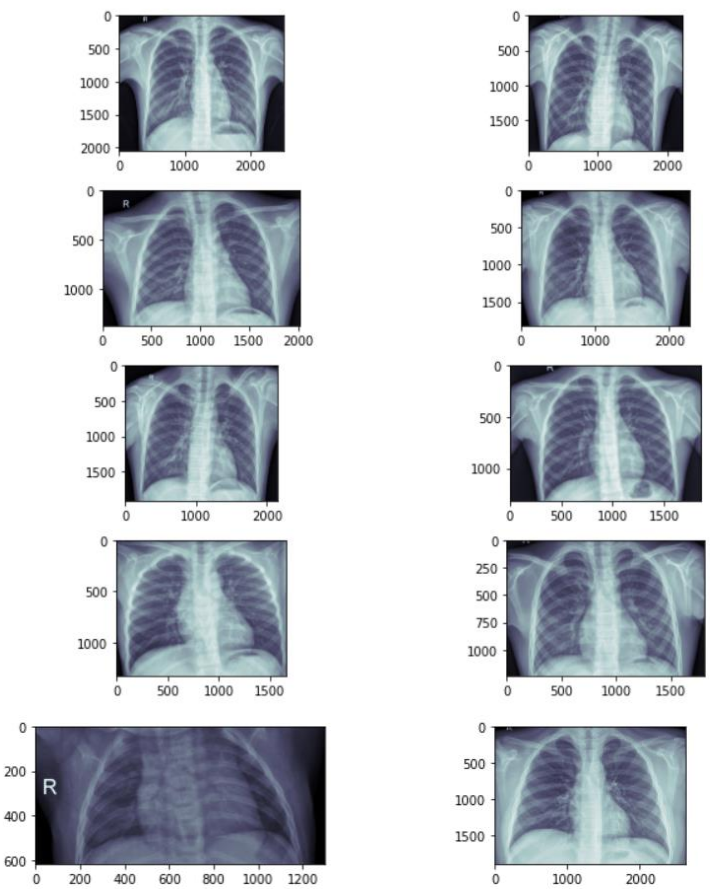


Definindo o caminho das imagens de teste e treino

```
[ ]  # Definindo o caminho das imagens
     Imagens_Teste = '/content/arquivos_driver/MyDrive/Dados_Modelo/Coronahack-Chest-XRay-Dataset/Coronahack-Chest-XRay-Dataset/test'
     Imagens_Treino = '/content/arquivos_driver/MyDrive/Dados_Modelo/Coronahack-Chest-XRay-Dataset/Coronahack-Chest-XRay-Dataset/train'

     # Selecionando uma amostra de Imagens

     # Vamos utilizar o OS.Walk
     # Função OS.Walk gera os nomes dos arquivos em uma árvore de diretórios percorrendo
     # a árvore de cima para baixo ou de baixo para cima.
     # Vamos incluir essa função da OS em uma lista e selecionar apenas algumas imagens
     # Vamos Selecionar apenas 10 Imagens
     Amostra_Imagens = list(os.walk(Imagens_Teste))[0][2][:10]

     # Formartar o caminho da imagem
     # Vamos usar uma função para formartar o caminho da imagem para ser passada na PIL
     Amostra_Imagens = list(map(lambda x: os.path.join(Imagens_Teste, x), Amostra_Imagens))
```

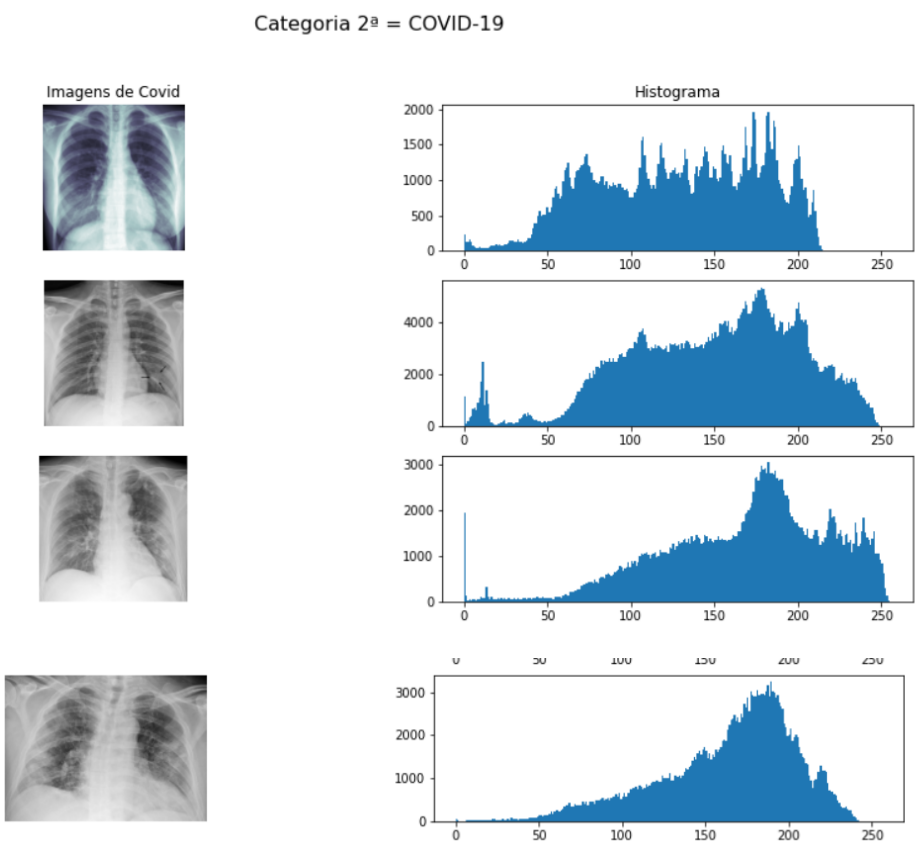
Plotando algumas imagens

```
[ ]  # Plotar algumas imagens da Base de Dados

     # Definindo o tamanho da Imagem
     plt.figure(figsize = (10,10))

     # Loop para plotar as imagens
     for iterator, Caminho_Imagem in enumerate(Amostra_Imagens):

         # Carregando a Imagem usando a Lib PIL
         Imagem = PIL.Image.open(Caminho_Imagem)

         # Definindo a Grade dos Gráficos
         plt.subplot(5, 2, iterator+1)

         # Plotando a Imagem dentro de um gráfico
         plt.imshow(Imagem, cmap=plt.cm.bone)

     # Função para ajustar a grade
     plt.tight_layout()
```



Verificando imagens com covid

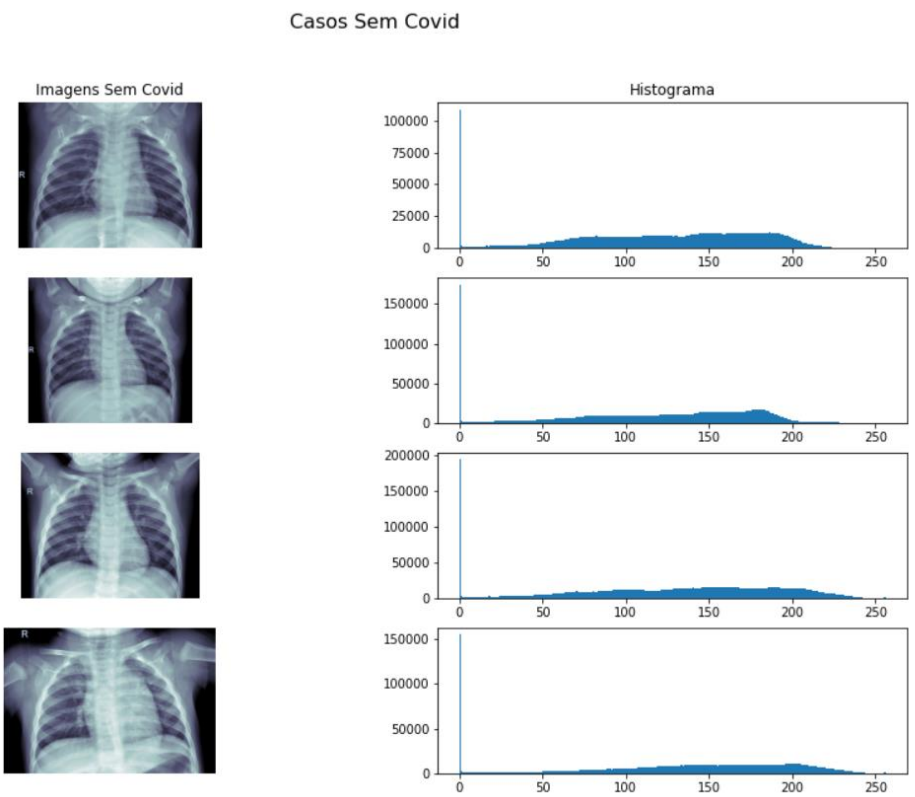
```
[ ]  # Extraindo informações das imagens com COVID

     # Definindo a estrutura da Grade dos gráficos
     fig, ax = plt.subplots(4, 2, figsize=(15, 10))

     # Filtrando apenas os Casos de Covid
     # Pegangando a Coluna dos dados de raio-x
     Casos_Covid = Base_Dados[Base_Dados['Label_2_Virus_category']=='COVID-19']['X_ray_image_name'].values

     # Filtrando apenas alguns casos
     Amostra_Casos_Covid = Casos_Covid[:4]

     # Ajustando o caminho para passar as imagens para a Lib PIL
     Amostra_Casos_Covid = list(map(lambda x: os.path.join(Imagens_Treino, x), Amostra_Casos_Covid))

     # Loop para plotar as imagens e o gráfico
     for Linha, Caminho in enumerate(Amostra_Casos_Covid):

         # Carregando a Imagem no gráfico
         Imagem = plt.imread(Caminho)

         # Plotando a Imagem no gráfico
         ax[Linha, 0].imshow(Imagem, cmap=plt.cm.bone)

         # Plotando o histograma da imagem
         # Função 'ravel' irá retornar uma lista com os valores tonal da Imagem
         # Em outras palavras será extraindo as escalas de cor da imagem
         ax[Linha, 1].hist(Imagem.ravel(), 256, [0,256])

         # Desligando as escalas de valores do gráfico da imagem
         ax[Linha, 0].axis('off')

         # Condição para o primeiro loop
         # Inserindo os titulos nas grades
         if Linha == 0:
             ax[Linha, 0].set_title('Imagens de Covid')
             ax[Linha, 1].set_title('Histograma')

     # Adicionando o Titulo
     fig.suptitle('Categoria 2ª = COVID-19', size=16);
```



Verificando imagens sem covid

```
[ ]  # Extraindo informações das imagens sem COVID

     # Definindo a estrutura da Grade dos gráficos
     fig, ax = plt.subplots(4, 2, figsize=(15, 10))

     # Filtrando apenas os Casos sem COVID
     # Pegangando a Coluna dos dados de raio-x
     Sem_Covid = Base_Treino[Base_Treino['Label']=='Normal']['X_ray_image_name'].values

     # Filtrando apenas alguns casos
     Amostra_Sem_Covid = Sem_Covid[:4]

     # Ajustando o caminho para passar as imagens para a Lib PIL
     Amostra_Sem_Covid = list(map(lambda x: os.path.join(Imagens_Treino, x), Amostra_Sem_Covid))

     # Loop para plotar as imagens e o gráfico
     for Linha, Caminho in enumerate(Amostra_Sem_Covid):

         # Carregando a Imagem no gráfico
         Imagem = plt.imread(Caminho)

         # Plotando a Imagem no gráfico
         ax[Linha, 0].imshow(Imagem, cmap=plt.cm.bone)

         # Plotando o histograma da imagem
         # Função 'ravel' irá retornar uma lista com os valores tonal da Imagem
         # Em outras palavras será extraindo as escalas de cor da imagem
         ax[Linha, 1].hist(Imagem.ravel(), 256, [0,256])

         # Desligando as escalas de valores do gráfico da imagem
         ax[Linha, 0].axis('off')
         # Desligando as escalas de valores do gráfico da imagem
         ax[Linha, 0].axis('off')

         # Condição para o primeiro loop
         # Inserindo os titulos nas grades
         if Linha == 0:
             ax[Linha, 0].set_title('Imagens Sem Covid')
             ax[Linha, 1].set_title('Histograma')

     # Adicionando o Titulo
     fig.suptitle('Casos Sem Covid', size=16);
```

[ ]



Vamos preparar os dados para o modelo

```
[ ]  # Criar a Função para ajustar a imagem
     Função_Pre_Processamento_Imagem =  ImageDataGenerator(
       # 1º Cisamento da Imagem
       # 'Cisalhamento' significa que a imagem será distorcida ao longo de um eixo,
       # principalmente para criar ou retificar os ângulos de percepção.
       # Geralmente é usado para aumentar imagens para que os computadores,
       # possam ver como os humanos veem as coisas de diferentes ângulos.
       shear_range=0.2,

       # Aumentar o zoom na imagem
       zoom_range=0.2,
     )
```

```
[ ]  # Função para ler Imagem e converter a imagem em um array com as escalas de cores
     def Ler_Imagem(Arquivo, Tamanho, Local):

         # Carregar a imagem
         Imagem = Image_.load_img(os.path.join(Local, Arquivo), target_size=Tamanho)

         # Converter a Imagem em Array
         Imagem = img_to_array(Imagem) / 255

         # Retornar a Imagem
         return Imagem

     Covid_Aumentada = []

     # Função para aumentar a imagem
     def Aumentando(Nome):

         # Carregando a imagem com a função 'Ler_Imagem'
         Image = Ler_Imagem(Nome, (255,255), Imagens_Treino)

         # Variavel de apoio para o Loop
         Loop = 0

         # Lopp para modificar as imagens
         # Vamos utilizar o 'tqdm' para ver o processo
         # Chamando a função Pre-Processamento da imagem
         # Chamando o metodo 'Flow' da Função --> vamos extrair os valores RGB da imagem
         for batch in tqdm(Função_Pre_Processamento_Imagem.flow(tf.expand_dims(Image, 0), batch_size=30)):

             # Salvando os valores das escalas da Imagem
             # Retirnando as dimensões do tensor com o 'squeeze'
             Covid_Aumentada.append(tf.squeeze(batch).numpy())

             # Vamos freiar o Loop no 20
             if Loop == 20:
                 break
             Loop = Loop + 1

     # Aplicando a Função
     Base_Treino['X_ray_image_name'].apply(Aumentando)
```

```
20it [00:00, 61.66it/s]
20it [00:00, 61.44it/s]
20it [00:00, 60.62it/s]
20it [00:00, 58.84it/s]
20it [00:00, 59.63it/s]
20it [00:00, 23.64it/s]
20it [00:00, 62.61it/s]
20it [00:00, 58.95it/s]
20it [00:00, 60.88it/s]
20it [00:00, 59.48it/s]
20it [00:00, 60.73it/s]
20it [00:00, 59.00it/s]
20it [00:00, 57.73it/s]
20it [00:00, 62.48it/s]
20it [00:00, 58.29it/s]
20it [00:00, 59.36it/s]
20it [00:00, 58.44it/s]
20it [00:00, 62.75it/s]
20it [00:00, 60.12it/s]
20it [00:00, 62.58it/s]
20it [00:00, 62.63it/s]
20it [00:00, 59.72it/s]
20it [00:00, 61.94it/s]
20it [00:00, 57.80it/s]
20it [00:00, 60.41it/s]
20it [00:00, 61.93it/s]
20it [00:00, 61.08it/s]
20it [00:00, 61.43it/s]
20it [00:00, 59.58it/s]
20it [00:00, 60.05it/s]
```

Preparando os dados para o modelo

```
[ ] # Vamos extrair os dados de escalas da imagens

    # Parte do treino
    # Lista para salvar os valores
    Lista_Treino = []

    # Aplicando a função para ler as imagens e extrair os dados
    Base_Treino['X_ray_image_name'].apply(lambda x: Lista_Treino.append(Ler_Imagem(x, (255,255), Imagens_Treino)))

    # Parte de Teste
    Lista_Teste = []

    # Aplicando a função para ler as imagens e extrair os dados
    Base_Teste['X_ray_image_name'].apply(lambda x: Lista_Teste.append(Ler_Imagem(x, (255,255), Imagens_Teste)))

    # ------- Essa função demora aproximadamente 10 Minutos para rodar ------- #
```

```
5573    None
5627    None
5893    None
5378    None
5330    None
5732    None
5658    None
5791    None
5618    None
5507    None
5531    None
5364    None
5509    None
5339    None
5862    None
5551    None
5764    None
```

```
[ ] # Juntando os rotulos de treinamento com os rotulo das imagens aumentada
    Treinamento_Y = np.concatenate((
        np.int64(Base_Treino['Classe'].values),
        np.ones(len(Covid_Aumentada), dtype=np.int64)))
```

Dividindo os tensores

```
[ ] # Ajustando os dados para tensores
    Tensor_Treino = tf.convert_to_tensor(np.concatenate((np.array(Lista_Treino), np.array(Covid_Aumentada))))
    Tensor_Teste  = tf.convert_to_tensor(np.array(Lista_Teste))
    Tensor_Treino_Y = tf.convert_to_tensor( Treinamento_Y )
    Tensor_Teste_Y = tf.convert_to_tensor(Base_Teste['Classe'].values)
```

```
[ ] # Verificando se todos os tensores estão na mesma dimensão
    print(len( Tensor_Treino ))
    print(len( Tensor_Teste ))
    print(len( Tensor_Treino_Y ))
    print(len( Tensor_Teste_Y ))
```

```
2200
20
2200
20
```

```
[ ] # Definindo os dados de Treino e Teste
    Dados_Treino = tf.data.Dataset.from_tensor_slices((Tensor_Treino, Tensor_Treino_Y))
    Dados_Teste = tf.data.Dataset.from_tensor_slices((Tensor_Teste, Tensor_Teste_Y))
```

Construção do modelo

```
[ ] # Batchs para embalhalar os dados
    Quantidade_Batchs = 4
    Buffer = 1000

    Batchs_Treino = Dados_Treino.shuffle(Buffer).batch(Quantidade_Batchs)
    Batchs_Teste = Dados_Teste.batch(Quantidade_Batchs)
```

```
[ ] # Definir a dimensão das imagens
    Dimensao_Imagens = (255,255,3)

    # Vamos utilizar um rede neural pré treinada
    Base_Modelo = tf.keras.applications.ResNet50(
        input_shape= Dimensao_Imagens,
        include_top=False,
        weights='imagenet')

    # Definindo como falso para não bagunçar os pesos do modelo pré-treinado
    Base_Modelo.trainable = False
```

Criando a rede convolucional

```
[ ] # Definindo os Ajustes da Rede Neural para treinar o modelo

    # Essa função é uma pilha linear de camadas
    Modelo = Sequential()

    # Atribuindo a rede pré treinada
    Modelo.add(Base_Modelo)

    # Adicionando camada Global
    # AveragePooling2D aplica o agrupamento médio nas dimensões espaciais até que cada dimensão espacial seja uma,
    # e deixa as outras dimensões inalteradas.
    Modelo.add( GlobalAveragePooling2D() )

    # Adicionando as camadas na rede neural
    Modelo.add(Dense(128))

    # Adicionando a tecnica 'Dropout'
    # Dropout é uma técnica em que neurônios selecionados aleatoriamente são ignorados durante o treinamento.
    # Eles são “descartados” aleatoriamente.
    # Isso significa que sua contribuição para a ativação dos neurônios a jusante é temporariamente removida na passagem para _
    # frente e quaisquer atualizações de peso não são aplicadas ao neurônio na passagem para trás.
    Modelo.add( Dropout(0.2) )

    # Adicionando tipo de ativação da rede neural
    Modelo.add(Dense(1, activation = 'sigmoid'))
```

```
[ ] # Função para parar o treinamento caso a rede não esteja aprendendoo mais nada
    Funcao_Parada = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2)

    # Vamos definir outros parametros da rede.
    Modelo.compile(optimizer='adam',
                   loss = 'binary_crossentropy',
                   metrics=['accuracy'])
```

```
[ ] # Treinamento do Modelo

    # ----------- Esse processo vai demorar -_-'

    # Função do Modelo
    Modelo.fit(
        # Dados de Treino
        Batchs_Treino,
        # Definindo as Epocas [ Ajustes de Pessos ]
        epochs=10,
        # Dados de Teste
        validation_data=Batchs_Teste,
        # Chamando a função Parada
        callbacks=[Funcao_Parada])
```

```
Epoch 1/10
550/550 [==============================] - 521s 946ms/step - loss: 0.0918 - accuracy: 0.9891 - val_loss: 1.6376 - val_accuracy: 0.7000
Epoch 2/10
550/550 [==============================] - 534s 970ms/step - loss: 0.0750 - accuracy: 0.9891 - val_loss: 1.3749 - val_accuracy: 0.7000
Epoch 3/10
550/550 [==============================] - 542s 985ms/step - loss: 0.0769 - accuracy: 0.9868 - val_loss: 2.0329 - val_accuracy: 0.7000
Epoch 4/10
550/550 [==============================] - 546s 992ms/step - loss: 0.0701 - accuracy: 0.9886 - val_loss: 1.5492 - val_accuracy: 0.7000
```

Essa etapa do treinamento da Rede provavelmente vai demorar muito tempo.
A Rede neural terá mais de milhões de linhas de registros com as escalas das imagens.

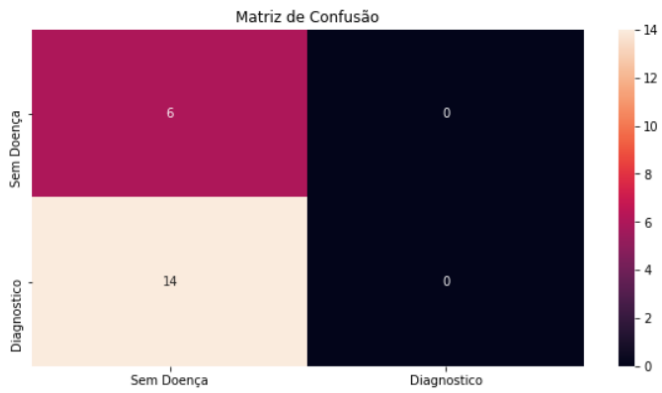
Coloca para rodar e vai tomar um café.
Se estiver o Colab provavelmente irá interromper o fluxo.

Verificando a acurácia do modelo

```
[ ] # Realizando as previsões nas imagens de Testes
    Previsoes = Modelo.predict( np.array(Lista_Teste) )

    # Ajustando a previsão para ser arredondada 0 ou 1
    Ajustes_Numerico = np.argmax( Previsoes, axis=1)

    # Função para medir a acuracia do Modelo
    from sklearn.metrics import classification_report

    # Verificando a acuracia do Modelo
    print( classification_report( Base_Teste['Classe'], Ajustes_Numerico.flatten() ) )
```

```
              precision    recall  f1-score   support

           0       0.30      1.00      0.46         6
           1       0.00      0.00      0.00        14

    accuracy                           0.30        20
   macro avg       0.15      0.50      0.23        20
weighted avg       0.09      0.30      0.14        20
```

```
[ ] # Função para medir a acuracia do Modelo
    from sklearn.metrics import confusion_matrix

    # Calculando a Matriz
    Matriz = confusion_matrix( Base_Teste['Classe'], Ajustes_Numerico.flatten() )

    # Definindo o tamanho da Figura
    plt.figure(figsize = (10,5))

    # Definindo o titulo
    plt.title('Matriz de Confusão')

    # Plotando o Gráfico
    sns.heatmap(Matriz,
                yticklabels=['Sem Doença', 'Diagnostico'],
                xticklabels=['Sem Doença', 'Diagnostico'],
                annot=True);
```



A Acurácia não foi satisfatória, pelo fato de diminuir a amostra de imagens e diminuir alguns parâmetros de processamento.

Final

Esse guia foi elaborada para demostrar o de uma rede convolucional

Link do Colab
https://colab.research.google.com/drive/1NxRswSjPfjNKNnfUBB0ZEzERW4y30BbT?usp=sharing

Odemir Depieri Jr

Data Intelligence Analyst Sr
Tech Lead
Specialization AI