@Odemir Depieri Jr

Identificando Tumor Cerebral

com Deep Learning

TensorFlow

OpenCV

pandas

NumPy

seaborn

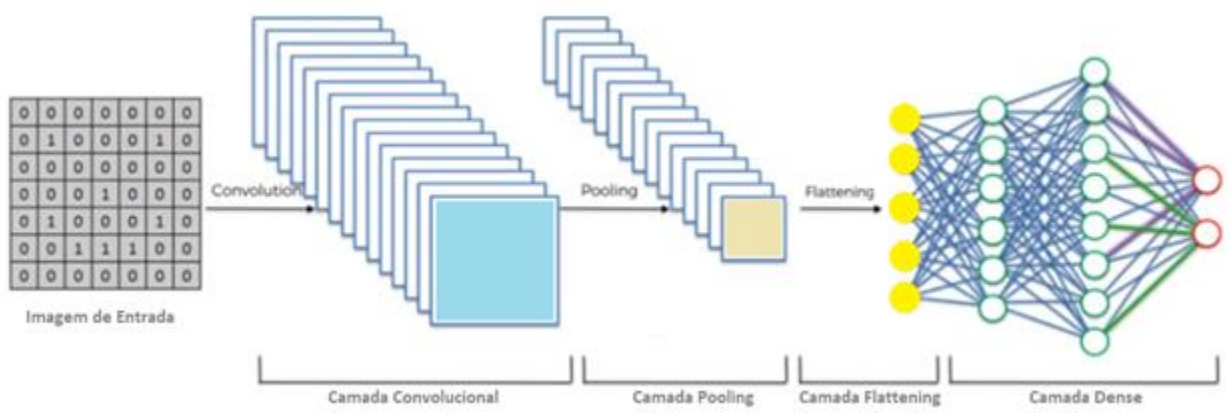
matplotlib

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/denizkavi1/brain-tumor

@Odemir Depieri Jr

Vamos importar as Lib necessárias

In [ ]:
```
# --------------- Instalar Libs no Jupyter --------------- #
# pip install opencv-python
# pip install tensorflow
```

In [20]:
```
# Lib para acessar recursos do sistema operacional
import os
# Lib para trabalhar com imagens
import cv2
# Lib para matrizes
import numpy as np
# Lib para gráficos
import matplotlib.pyplot as plt
# Lib para gráficos
import seaborn as sns

# Lib para Machine Learning
import tensorflow as tf
# Função das camadas e pesos da rede neural
from tensorflow.keras import layers, optimizers
# Função para criar a rede convolucional
from tensorflow.keras.applications import ResNet50
# Função para as camadas da Rede Neurak
from tensorflow.keras.layers import Input, Dense, AveragePooling2D, Dropout, Flatten
# Função para criar a rede neural
from tensorflow.keras.models import Model
# Função para acessar local para direcionar as imagens
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Função para salvar os pesos da Rede Neural
from tensorflow.keras.callbacks import ModelCheckpoint
```

Definindo o local das imagens

In [22]:
```
# Diretorio com as imagens
Diretorio_Imagens = 'C:/Users/datav/Desktop/Imagens/'

# Verificando as pastas
print('Pastas no caminho:', os.listdir( Diretorio_Imagens ) )
```

```
Pastas no caminho: ['1', '2', '3']
```

Preparando as imagens

In [23]:
```
# Fazendo as escalas da imagens em RGB
Image_Generator = ImageDataGenerator( rescale=1./255 )
```

In [26]:
```
# Preparando as imagens para serem passadas na rede neural

# Função para acessar as imagens do diretorio e preparar as imagens
Treino_Generator = Image_Generator.flow_from_directory(
    # Definindo um batch para rodar 50 imagens
    batch_size=50,

    # Passando o local das imagens
    directory=Diretorio_Imagens,

    # Habilitando a opção para embalharar as imagens
    shuffle=True,

    # Definindo o tamanho da imagem
    target_size=(256,256),

    # Definindo um problema de classificação
    class_mode='categorical',

    # Definindo como base de treino
    subset='training'
)
```

```
Found 3064 images belonging to 3 classes.
```

In [27]:
```
# Verificando quantos batchs serão executando
print('Temos 3064 imagens')
print('Serão executados', 3064/50, 'batchs')
```

```
Temos 3064 imagens
Serão executados 61.28 batchs
```

Temos ao todo + 3k de imagens.
Vamos dividir essas imagens em batchs para serem processadas na rede neural.

Aqui vamos usar batches de 50 imagens.

Verificando os dados das imagens

In [28]:
```
# Criando os dados de Treino
Imagens_Treino, Classe_Treino = next(Treino_Generator)
```

In [30]:
```
# Verificando a dimensão
# 50 imagens
# 256 x 256 dimensão das imanges
# 3 escala de RGB
Imagens_Treino.shape
```

Out[30]:
```
(50, 256, 256, 3)
```

In [33]:
```
# Verificando a saida da Rede Neural
Classe_Treino[0:5]
```

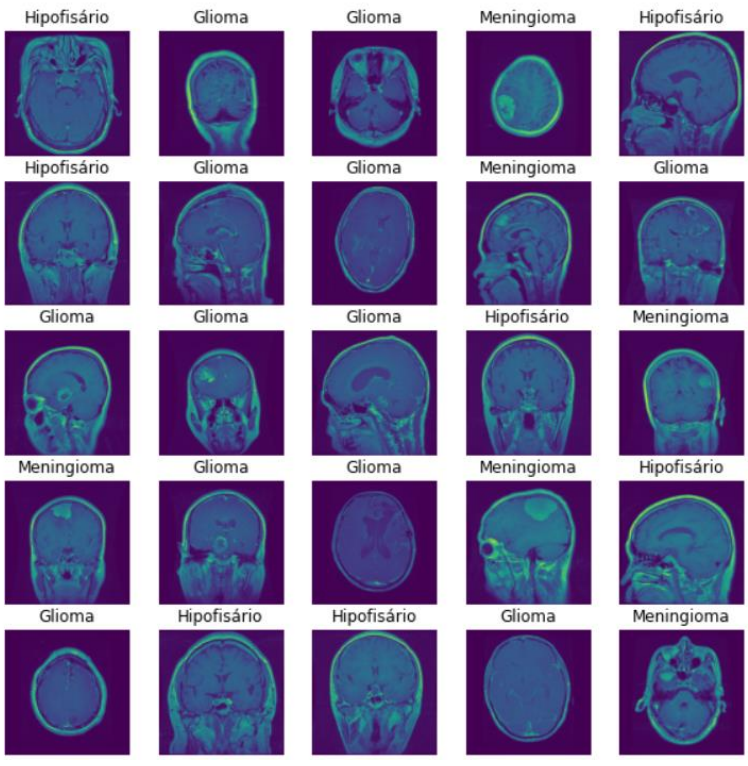
Out[33]:
```
array([[0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.]], dtype=float32)
```

Definindo as classes para serem classificadas

In [34]:
```
# Dicionario com as clases
Classes = {
    0 : 'Meningioma',
    1 : 'Glioma',
    2 : 'Hipofisário',
}
```

Plotando algumas imagens

In [46]:
```
# Plotar as imagens em uma grade de Gráficos

# Definindo o tamanho da Grade
Figuras, Eixos = plt.subplots(5, 5, figsize=(10,10) )
# Transformar uma matriz em um vetor
Eixos = Eixos.ravel()

# Loop pra plotar as imagens
for Loop in np.arange(0,25):

    # Plotando as imagens no gráfico
    Eixos[Loop].imshow( Imagens_Treino[Loop] )

    # Colocando o Titulo nas Imanges
    Eixos[Loop].set_title( Classes[ np.argmax( Classe_Treino[Loop] ) ] )

    # Retirando o eixo da escala
    Eixos[Loop].axis('off')

# Colocar um espaço entre as imagens
plt.subplots_adjust( wspace=0.2 )
```



Criando a rede Neural

In [49]:
```
# Criando a rede neural convolucional com uma estrutura pré-treinada
Modelo_Basico = ResNet50(
    # Definindo o treino
    weights='imagenet',

    # Definindo False para podemos criar a estrutura da rede
    include_top=False,

    # Definindo o formato de entrada dos dados
    input_tensor= Input( shape=(256,256, 3) )
)
```

In [50]:
```
# Verificando os dados da Rede Neural
Modelo_Basico.summary()
```

```
conv2_block1_3_bn (BatchNormali (None, 64, 64, 256)  1024        conv2_block1_3_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_add (Add)          (None, 64, 64, 256)  0           conv2_block1_0_bn[0][0]
                                                                 conv2_block1_3_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_out (Activation)   (None, 64, 64, 256)  0           conv2_block1_add[0][0]
__________________________________________________________________________________________________
conv2_block2_1_conv (Conv2D)    (None, 64, 64, 64)   16448       conv2_block1_out[0][0]
__________________________________________________________________________________________________
conv2_block2_1_bn (BatchNormali (None, 64, 64, 64)   256         conv2_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_1_relu (Activation (None, 64, 64, 64)   0           conv2_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_2_conv (Conv2D)    (None, 64, 64, 64)   36928       conv2_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_2_bn (BatchNormali (None, 64, 64, 64)   256         conv2_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_2_relu (Activation (None, 64, 64, 64)   0           conv2_block2_2_bn[0][0]
__________________________________________________________________________________________________
```

Vamos utilizar a rede **ResNet50**, essa rede já possui parâmetros **pré-configurada.**
Como estamos trabalhando com redes neurais convolucionais, é desgastante treinar uma rede do zero, há inúmeros tentativas e treinos a serem feitas.
A ResNet50 já possui toda uma estrutura configurada.

No print vemos **cada camada** da rede e **seus detalhes**.

In [52]:
```
# Verificando as camadas da Rede
len( Modelo_Basico.layers )
```

Out[52]: 175

Vamos ajustar os últimas camadas da rede

In [54]:
```
# Congelando os pesos da Rede Neural
# Vamos deixar as ultimas camdas livres para serem treinadas
for Camada in Modelo_Basico.layers[:-10]:
    # Cancelando o ajuste dos pesos
    Camada.trainable = False
```

Ajustando a rede neural

In [67]:
```
# Definindo a camada de inicio
Modelo_Inicio = Modelo_Basico.output

# Reduzindo as deminensão
Modelo_Inicio = AveragePooling2D()( Modelo_Inicio )

# Adicionando uma camada
Modelo_Inicio = Flatten()( Modelo_Inicio )

# Adicionando os neuronoios -- 1º
Modelo_Inicio = Dense( 256, activation='relu')( Modelo_Inicio )

# Adicionando o Dropout para zerar os neuronios e evitar overfiting
Modelo_Inicio = Dropout(0.2)( Modelo_Inicio )

# Adicionando os neuronoios -- 2º
Modelo_Inicio = Dense( 128, activation='relu')( Modelo_Inicio )

# Adicionando o Dropout para zerar os neuronios e evitar overfiting
Modelo_Inicio = Dropout(0.2)( Modelo_Inicio )

# Definindo os neuronios de saida
Modelo_Inicio = Dense(3, activation='softmax')( Modelo_Inicio )
```

Ajustando o modelo da rede

In [68]:
```
# Criando o Modelo da Rede Neural
Modelo = Model( inputs=Modelo_Basico.input, outputs=Modelo_Inicio )

# Configurando alguns paramentros da Rede
Modelo.compile(
    # Função de Erro
    loss='categorical_crossentropy',

    # Algoritimo de ajustes dos Pesos
    optimizer=optimizers.RMSprop( lr=1e-4, decay=1e-6),

    # Metrica de avaliação
    metrics=['accuracy']
)
```

Vamos diminuir as imagens de batchs para rodar o treinamento da Rede

In [72]:
```
# Treinamento da Rede Neural

# Salvar os dados da Rede Neural
Salvando_Rede = ModelCheckpoint( filepath='weigts.hdf5', save_best_only=True)

# Função para acessar as imagens do diretorio e preparar as imagens
Treino_Generator = Image_Generator.flow_from_directory(
    # Definindo um batch para rodar 50 imagens
    batch_size=5,

    # Passando o local das imagens
    directory=Diretorio_Imagens,

    # Habilitando a opção para embalharar as imagens
    shuffle=True,

    # Definindo o tamanho da imagem
    target_size=(256,256),

    # Definindo um problema de classificação
    class_mode='categorical',

    # Definindo como base de treino
    subset='training',
)
```

```
Found 3064 images belonging to 3 classes.
```

Rodando a Rede

In [73]:
```
# Rodar o Modelo
Historico = Modelo.fit_generator(
    # Passando o Treino Generator
    Treino_Generator,

    # Definindo as epocas - bacthes
    epochs=25,

    # Definido para salvar os pesos da Redes
    callbacks=[Salvando_Rede]
)

# ---------------- Isso vai demorar !!!!!!!!!!!!!! ---------------- #
```

```
Epoch 1/25
613/613 [==============================] - 624s 1s/step - loss: 1.0709 - accuracy: 0.4612
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 2/25
613/613 [==============================] - 668s 1s/step - loss: 1.0559 - accuracy: 0.4654
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 3/25
613/613 [==============================] - 636s 1s/step - loss: 1.0304 - accuracy: 0.4654
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 4/25
613/613 [==============================] - 576s 939ms/step - loss: 0.9887 - accuracy: 0.4654
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 5/25
613/613 [==============================] - 591s 965ms/step - loss: 0.9569 - accuracy: 0.4883
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 6/25
613/613 [==============================] - 582s 949ms/step - loss: 0.9353 - accuracy: 0.5607
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
Epoch 7/25
613/613 [==============================] - 566s 924ms/step - loss: 0.9128 - accuracy: 0.5858
```

Atenção !

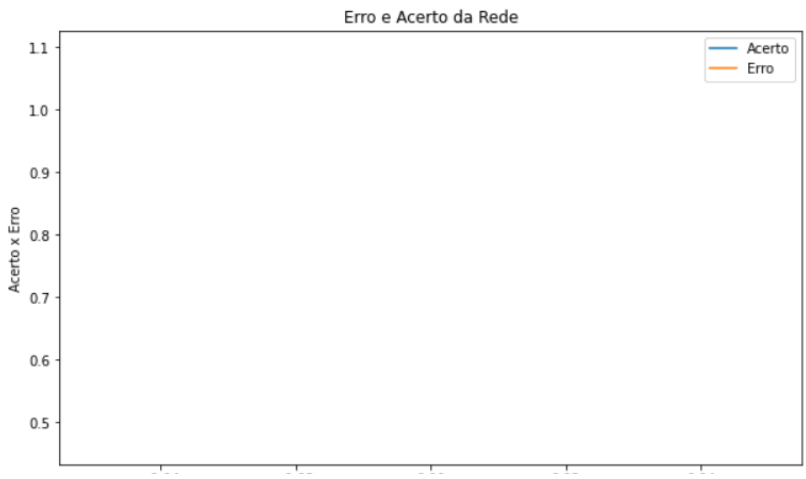
Interrompi o treinamento da Rede porque já estava 1:30 minutos.
Caso tenha paciência, deixe rodar 100%.

Avaliando o Modelo

In [90]:
```
# Acessando os dados da Rede
Historico.history.keys()
```

Out[90]:
```
dict_keys(['loss', 'accuracy'])
```

In [91]:
```
# Plotar a Acuracia do Modelo

# Definindo uma figura
plt.figure( figsize=(10,6) )
# Plotando a Acuracia
plt.plot( Historico.history['accuracy'] )
# Plotando o erro
plt.plot( Historico.history['loss'] )
# Definindo o Tituli
plt.title('Erro e Acerto da Rede')
# Definindo os Labels
plt.xlabel('Épocas')
plt.ylabel('Acerto x Erro')
# Definindo a Legenda
plt.legend(['Acerto', 'Erro']);
```



In [93]:
```
Historico.history['accuracy']
```

Out[93]:
```
[0.4647519588470459]
```

In [94]:
```
Historico.history['loss']
```

Out[94]:
```
[1.0951128005981445]
```

Atenção !

Havia interrompido o processamento, com isso o Histórico da rede se perdeu. Assim rodei apenas 1 batch.

Mas **antes de encerrar o processamento anterior**, a rede já esta com **+ de 60%** de acurácia e faltava metade. Bem provável que a acurácia seria muito bom do modelo.

Final

Esse guia foi elaborada para demostrar o de uma rede convolucional

Link do Script para download
https://drive.google.com/file/d/1601cvMRuTEP-mJPOAmRkuU1N-Ju6TqUE/view?usp=sharing

Odemir Depieri Jr

Data Intelligence Analyst Sr
Tech Lead
Specialization AI