Brain Tumor Detection

* [Part I: Classification](https://www.kaggle.com/code/mosinan10/brain-tumor-detection#section-one)
* [Part II: Activations' Visualization](https://www.kaggle.com/code/mosinan10/brain-tumor-detection#section-two)
* [Part III: Segmentation with U-net](https://www.kaggle.com/code/mosinan10/brain-tumor-detection#section-three)
* [Conclusion](https://www.kaggle.com/code/mosinan10/brain-tumor-detection#section-four)

Part I: Classification

In this first part of the notebook, the main aim is to create a model able to distinguish between MRI images of patients affected by a brain tumor and the ones of healthy subjects. To achieve this aim, instead of using a pre-trained model, as done in the other published notebooks, I decided to build a simple Convolutional Neural Network from scratch. Considering that one of the greatest limitations of Deep learning is the interpretability of the models, I wanted to see if a simpler and more interpretable model could achieve or even overcome the results obtained in the other notebooks with deeper models.

P.s: I'm still almost a novice in the field, so feel free to comment or give suggestions to improve or correct the work!

In [1]:

*#%% IMPORTING LIBRARIES*

import os

import glob

import shutil

import random

import pandas as pd

from PIL import Image

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

In [2]:

*#%% IMPORTING DATA*

def importing\_data(path):

sample = []

for filename **in** glob.glob(path):

*#img = Image.open(filename,'r')*

*#IMG = np.array(img)*

sample.append(filename)

return sample

path1 = '//kaggle/input/brain-tumor-detection/no/\*.jpg'

path2 = '//kaggle/input/brain-tumor-detection/yes/\*.jpg'

path3 = '//kaggle/input/brain-tumor-detection/pred/\*.jpg'

train\_n = importing\_data(path1)

train\_y = importing\_data(path2)

test = importing\_data(path3)

*#%% CREATION OF DATASETS*

df\_train\_n = pd.DataFrame({'image':train\_n, 'label': 'Healthy'})

df\_train\_y = pd.DataFrame({'image':train\_y, 'label': 'Affected'})

df\_test = pd.DataFrame({'image':test})

train\_data = pd.concat([df\_train\_n, df\_train\_y])

train\_data.head()

Out[2]:

|  | image | label |
| --- | --- | --- |
| 0 | //kaggle/input/brain-tumor-detection/no/no26.jpg | Healthy |
| 1 | //kaggle/input/brain-tumor-detection/no/no979.jpg | Healthy |
| 2 | //kaggle/input/brain-tumor-detection/no/no598.jpg | Healthy |
| 3 | //kaggle/input/brain-tumor-detection/no/no141.jpg | Healthy |
| 4 | //kaggle/input/brain-tumor-detection/no/no715.jpg | Healthy |

In [3]:

*#%% TRAIN-VALIDATION SPLIT (90% TRAIN - 10% VALIDATION)*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_val = train\_test\_split(train\_data,

test\_size = 0.1,

shuffle = True,

random\_state = 42)

CREATING THE CNN MODEL

In [4]:

*#%% CREATING THE CNN MODEL*

import keras

from keras.metrics import AUC, Recall, Precision

from keras.models import Sequential

from keras.layers import Dense, GlobalAveragePooling2D, Dropout, Conv2D , MaxPooling2D, Flatten

from keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau

from keras.optimizers import RMSprop

def build\_model():

*'''Sequential Model creation'''*

Cnn = Sequential()

Cnn.add(Conv2D(64,(5,5), activation = 'relu', padding = 'same',

strides=(2,2), input\_shape = [224,224,1]))

Cnn.add(MaxPooling2D(2))

Cnn.add(Conv2D(128,(5,5), activation = 'relu', padding = 'same', strides=(2,2)))

Cnn.add(Conv2D(128,(5,5), activation = 'relu', padding = 'same', strides=(2,2)))

Cnn.add(Conv2D(256,(5,5), activation = 'relu', padding = 'same', strides=(2,2)))

Cnn.add(MaxPooling2D(2))

*#Cnn.add(GlobalAveragePooling2D())*

Cnn.add(Flatten())

Cnn.add(Dense(64, activation = 'relu'))

Cnn.add(Dropout(0.4))

Cnn.add(Dense(32, activation = 'relu'))

Cnn.add(Dropout(0.4))

Cnn.add(Dense(2, activation = 'softmax'))

return Cnn

keras\_model = build\_model()

keras\_model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d (Conv2D) (None, 112, 112, 64) 1664

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d (MaxPooling2D) (None, 56, 56, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_1 (Conv2D) (None, 28, 28, 128) 204928

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 14, 14, 128) 409728

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 7, 7, 256) 819456

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 3, 3, 256) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 2304) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 64) 147520

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 32) 2080

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 2) 66

=================================================================

Total params: 1,585,442

Trainable params: 1,585,442

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

In [5]:

*#%% FITTING THE MODEL*

def Model\_fit(train\_data, val\_data):

keras\_model = None

keras\_model = build\_model()

*'''Compiling the model'''*

keras\_model.compile(optimizer = RMSprop(learning\_rate = 1e-4),

loss='sparse\_categorical\_crossentropy',

metrics =['acc'])

es = EarlyStopping(monitor='val\_loss', mode='min',

patience=2,

restore\_best\_weights=True,

verbose=1)

checkpoint\_cb = ModelCheckpoint("Brain\_model\_best.h5",

save\_best\_only=True)

reduce\_lr = ReduceLROnPlateau(monitor = 'val\_loss',

factor = 0.2,

patience = 3,

min\_lr = 1e-5,

mode = 'min',

verbose=1)

history = keras\_model.fit(train\_data,

validation\_data = val\_data,

epochs= 50,

batch\_size = 10,

callbacks=[es, checkpoint\_cb, reduce\_lr])

return history

In [6]:

from keras.preprocessing.image import ImageDataGenerator

k\_fold = 3

IMG\_SIZE = 224

size = (IMG\_SIZE,IMG\_SIZE)

n\_CLASS = 2

def CV\_training(train\_data, val\_data):

cv\_histories = []

for i **in** range(0,k\_fold):

datagen = ImageDataGenerator(rescale = 1./255)

train\_set = datagen.flow\_from\_dataframe(train\_data,

directory = '//kaggle/input/brain-tumor-detection/\*.jpg',

x\_col = 'image',

y\_col = 'label',

target\_size = size,

color\_mode = 'grayscale',

class\_mode = 'sparse',

batch\_size = 10,

shuffle = True,

interpolation = 'bilinear')

val\_set = datagen.flow\_from\_dataframe(val\_data,

directory = '//kaggle/input/brain-tumor-detection/\*.jpg',

x\_col = 'image',

y\_col = 'label',

target\_size = size,

color\_mode = 'grayscale',

class\_mode = 'sparse',

batch\_size = 10,

shuffle = True,

interpolation = 'bilinear')

print("Training on Fold: ",i+1)

cv\_histories.append(Model\_fit(train\_set, val\_set))

return cv\_histories

cv\_results = CV\_training(X\_train,X\_val)

Found 2700 validated image filenames belonging to 2 classes.

Found 300 validated image filenames belonging to 2 classes.

Training on Fold: 1

Epoch 1/50

270/270 [==============================] - 61s 227ms/step - loss: 0.6150 - acc: 0.6652 - val\_loss: 0.5539 - val\_acc: 0.7200

Epoch 2/50

270/270 [==============================] - 60s 221ms/step - loss: 0.5075 - acc: 0.7607 - val\_loss: 0.4172 - val\_acc: 0.8133

Epoch 3/50

270/270 [==============================] - 59s 219ms/step - loss: 0.4116 - acc: 0.8185 - val\_loss: 0.3231 - val\_acc: 0.8567

Epoch 4/50

270/270 [==============================] - 60s 222ms/step - loss: 0.3079 - acc: 0.8730 - val\_loss: 0.2462 - val\_acc: 0.9000

Epoch 5/50

270/270 [==============================] - 59s 219ms/step - loss: 0.2288 - acc: 0.9130 - val\_loss: 0.1536 - val\_acc: 0.9433

Epoch 6/50

270/270 [==============================] - 60s 223ms/step - loss: 0.1606 - acc: 0.9474 - val\_loss: 0.0916 - val\_acc: 0.9667

Epoch 7/50

270/270 [==============================] - 60s 221ms/step - loss: 0.1031 - acc: 0.9674 - val\_loss: 0.1036 - val\_acc: 0.9667

Epoch 8/50

270/270 [==============================] - 61s 224ms/step - loss: 0.0717 - acc: 0.9767 - val\_loss: 0.0575 - val\_acc: 0.9867

Epoch 9/50

270/270 [==============================] - 60s 224ms/step - loss: 0.0518 - acc: 0.9867 - val\_loss: 0.0754 - val\_acc: 0.9800

Epoch 10/50

270/270 [==============================] - ETA: 0s - loss: 0.0412 - acc: 0.9915Restoring model weights from the end of the best epoch.

270/270 [==============================] - 61s 225ms/step - loss: 0.0412 - acc: 0.9915 - val\_loss: 0.0849 - val\_acc: 0.9833

Epoch 00010: early stopping

Found 2700 validated image filenames belonging to 2 classes.

Found 300 validated image filenames belonging to 2 classes.

Training on Fold: 2

Epoch 1/50

270/270 [==============================] - 60s 222ms/step - loss: 0.5998 - acc: 0.6789 - val\_loss: 0.5502 - val\_acc: 0.7400

Epoch 2/50

270/270 [==============================] - 60s 222ms/step - loss: 0.4913 - acc: 0.7796 - val\_loss: 0.4258 - val\_acc: 0.8167

Epoch 3/50

270/270 [==============================] - 60s 223ms/step - loss: 0.3818 - acc: 0.8363 - val\_loss: 0.2704 - val\_acc: 0.8900

Epoch 4/50

270/270 [==============================] - 60s 223ms/step - loss: 0.2850 - acc: 0.8870 - val\_loss: 0.1875 - val\_acc: 0.9233

Epoch 5/50

270/270 [==============================] - 60s 223ms/step - loss: 0.2041 - acc: 0.9252 - val\_loss: 0.1798 - val\_acc: 0.9100

Epoch 6/50

270/270 [==============================] - 61s 225ms/step - loss: 0.1471 - acc: 0.9533 - val\_loss: 0.1172 - val\_acc: 0.9567

Epoch 7/50

270/270 [==============================] - 61s 225ms/step - loss: 0.0978 - acc: 0.9741 - val\_loss: 0.0703 - val\_acc: 0.9800

Epoch 8/50

270/270 [==============================] - 60s 222ms/step - loss: 0.0726 - acc: 0.9785 - val\_loss: 0.0651 - val\_acc: 0.9833

Epoch 9/50

270/270 [==============================] - 61s 225ms/step - loss: 0.0498 - acc: 0.9863 - val\_loss: 0.0710 - val\_acc: 0.9767

Epoch 10/50

270/270 [==============================] - ETA: 0s - loss: 0.0450 - acc: 0.9900Restoring model weights from the end of the best epoch.

270/270 [==============================] - 61s 226ms/step - loss: 0.0450 - acc: 0.9900 - val\_loss: 0.0772 - val\_acc: 0.9833

Epoch 00010: early stopping

Found 2700 validated image filenames belonging to 2 classes.

Found 300 validated image filenames belonging to 2 classes.

Training on Fold: 3

Epoch 1/50

270/270 [==============================] - 61s 228ms/step - loss: 0.6338 - acc: 0.6337 - val\_loss: 0.5278 - val\_acc: 0.7667

Epoch 2/50

270/270 [==============================] - 61s 227ms/step - loss: 0.5210 - acc: 0.7567 - val\_loss: 0.4681 - val\_acc: 0.7733

Epoch 3/50

270/270 [==============================] - 62s 228ms/step - loss: 0.4332 - acc: 0.8096 - val\_loss: 0.3312 - val\_acc: 0.8467

Epoch 4/50

270/270 [==============================] - 61s 227ms/step - loss: 0.3471 - acc: 0.8507 - val\_loss: 0.2282 - val\_acc: 0.9100

Epoch 5/50

270/270 [==============================] - 61s 224ms/step - loss: 0.2698 - acc: 0.9104 - val\_loss: 0.1362 - val\_acc: 0.9500

Epoch 6/50

270/270 [==============================] - 61s 225ms/step - loss: 0.1809 - acc: 0.9419 - val\_loss: 0.1843 - val\_acc: 0.9267

Epoch 7/50

270/270 [==============================] - 61s 227ms/step - loss: 0.1211 - acc: 0.9604 - val\_loss: 0.0640 - val\_acc: 0.9867

Epoch 8/50

270/270 [==============================] - 61s 225ms/step - loss: 0.0850 - acc: 0.9770 - val\_loss: 0.0557 - val\_acc: 0.9833

Epoch 9/50

270/270 [==============================] - 61s 227ms/step - loss: 0.0618 - acc: 0.9844 - val\_loss: 0.0743 - val\_acc: 0.9867

Epoch 10/50

270/270 [==============================] - ETA: 0s - loss: 0.0483 - acc: 0.9874Restoring model weights from the end of the best epoch.

270/270 [==============================] - 61s 227ms/step - loss: 0.0483 - acc: 0.9874 - val\_loss: 0.0621 - val\_acc: 0.9867

Epoch 00010: early stopping

In [7]:

*#%% CHEKING THE CROSS VALIDATION METRICS*

def acc\_results(results):

i = 0

for fold **in** cv\_results:

print('Val\_Acc Folder '+ str(i) + ' =', max(fold.history['val\_acc']))

i += 1

acc\_results(cv\_results)

Val\_Acc Folder 0 = 0.9866666793823242

Val\_Acc Folder 1 = 0.9833333492279053

Val\_Acc Folder 2 = 0.9866666793823242

In [8]:

*#%% LOOKING AT THE ACCURACY-LOSS PLOTS FOR EACH FOLD*

def Acc\_Loss\_Plot(results):

for fold **in** results:

acc = fold.history['acc']

val\_acc = fold.history['val\_acc']

loss = fold.history['loss']

val\_loss = fold.history['val\_loss']

fig, (ax1, ax2) = plt.subplots(1,2, figsize= (10,5))

fig.suptitle(" MODEL'S METRICS VISUALIZATION ")

ax1.plot(range(1, len(acc) + 1), acc)

ax1.plot(range(1, len(val\_acc) + 1), val\_acc)

ax1.set\_title('History of Accuracy')

ax1.set\_xlabel('Epochs')

ax1.set\_ylabel('Accuracy')

ax1.legend(['training', 'validation'])

ax2.plot(range(1, len(loss) + 1), loss)

ax2.plot(range(1, len(val\_loss) + 1), val\_loss)

ax2.set\_title('History of Loss')

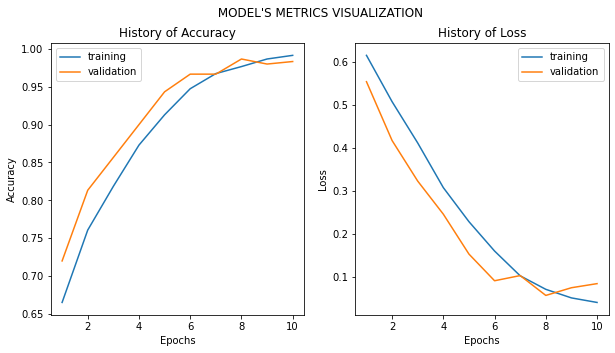
ax2.set\_xlabel('Epochs')

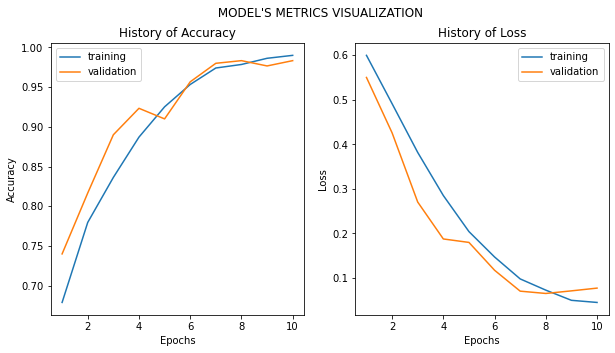
ax2.set\_ylabel('Loss')

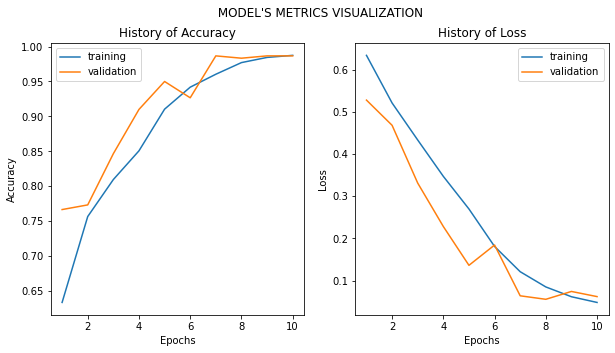
ax2.legend(['training', 'validation'])

plt.show()

Acc\_Loss\_Plot(cv\_results)







In [9]:

*#%% LOADING THE MODEL*

import keras

keras\_model = keras.models.load\_model('Brain\_model\_best.h5')

keras\_model.compile(optimizer = RMSprop(learning\_rate = 1e-4),

loss='sparse\_categorical\_crossentropy', metrics =[ 'acc'])

*# Predictions on the test set*

datagen = ImageDataGenerator(rescale = 1./255)

test\_set = datagen.flow\_from\_dataframe(df\_test,

directory = '//kaggle/input/brain-tumor-detection/\*.jpg',

x\_col = 'image',

y\_col = None,

target\_size = size,

color\_mode = 'grayscale',

class\_mode = None,

batch\_size = 10,

shuffle = False,

interpolation = 'bilinear')

predictions = keras\_model.predict(test\_set)

predictions = predictions.argmax(axis=-1)

print("Where 0 = 'Affected'")

print("Where 1 = 'Healthy'")

print(predictions)

Found 60 validated image filenames.

Where 0 = 'Affected'

Where 1 = 'Healthy'

[1 0 0 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0

0 1 0 1 1 1 0 1 1 0 1 0 1 0 1 1 0 0 1 1 0 0 1]

In [10]:

pred = []

[pred.append('Healthy') if i == 1 else pred.append('Affected') for i **in** predictions]

print(pred)

['Healthy', 'Affected', 'Affected', 'Healthy', 'Affected', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Affected', 'Affected', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Affected', 'Healthy', 'Healthy', 'Healthy', 'Healthy', 'Affected', 'Affected', 'Healthy', 'Healthy', 'Affected', 'Healthy', 'Affected', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Healthy', 'Healthy', 'Affected', 'Healthy', 'Healthy', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Affected', 'Healthy', 'Healthy', 'Affected', 'Affected', 'Healthy', 'Healthy', 'Affected', 'Affected', 'Healthy']

In [11]:

*#%% OBTAINING PREDICTIONS OF THE FIRST BATCH*

images10 = [test\_set[0][0],test\_set[0][1],test\_set[0][2],test\_set[0][3],test\_set[0][4],

test\_set[0][5],test\_set[0][6],test\_set[0][7],test\_set[0][8],test\_set[0][9]]

prediction10 = pred[0:9]

final\_pred = zip(images10,prediction10)

In [12]:

def pre\_visualization(data, predictions):

for image,pred **in** final\_pred:

plt.imshow(image.reshape(224,224), cmap = 'gray')

plt.title("Model's Prediction: " + str(pred))

plt.show()

pre\_visualization(images10,prediction10)

