MINIPROJECT

Brain Tumor Classification of MRI Images using Convolution Neural Networks

Course Code: AL3391

Course Title: Artificial Intelligence

Submitted by:

Geetha.A

953621243015

**Objective:**

To create a model that classifies the brain tumor using MRI images using CNN

**Description of the dataset:**

This dataset contains the MRI images. There are two categories in the dataset test and train

* Train:

The train data set contains the images used to train the model. It consists of almost 200 MRI images proves that there is a tumor and 200 MRI images that there is no tumor in the brain

* Test:

The test dataset contains the images used to test the model created. It consists of almost 200 images

**Program:**

import os

import keras

from keras.models import Sequential

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

from PIL import Image

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()

encoder.fit([[0], [1]])

# 0 - Tumor

# 1 – Normal

**Output:**

OneHotEncoder()

**Creating 3 Important Lists**

1. data list for storing image data in numpy array form
2. paths list for storing paths of all images
3. result list for storing one hot encoded form of target class whether normal or tumor

# This cell updates result list for images with tumor

data = []

paths = []

result = []

for r, d, f in os.walk(r'../input/brain-mri-images-for-brain-tumor-detection/yes'):

for file in f:

if '.jpg' in file:

paths.append(os.path.join(r, file)

for path in paths:

img = Image.open(path)

img = img.resize((128,128))

img = np.array(img)

if(img.shape == (128,128,3)):

data.append(np.array(img))

result.append(encoder.transform([[0]]).toarray())

# This cell updates result list for images without tumor

paths = []

for r, d, f in os.walk(r"../input/brain-mri-images-for-brain-tumor-detection/no"):

for file in f:

if '.jpg' in file:

paths.append(os.path.join(r, file))

for path in paths:

img = Image.open(path)

img = img.resize((128,128))

img = np.array(img)

if(img.shape == (128,128,3)):

data.append(np.array(img))

result.append(encoder.transform([[1]]).toarray())

data = np.array(data)

data.shape

**Output:**

(139, 128, 128, 3)

result = np.array(result)

result = result.reshape(139,2)

# Splitting the Data into Training & Testing

x\_train,x\_test,y\_train,y\_test = train\_test\_split(data, result, test\_size=0.2, shuffle=True, random\_state=0)

# Model Building

Batch normalization is a technique for training very deep neural networks that standardizes the inputs to a layer for each mini-batch. This has the effect of stabilizing the learning process and dramatically reducing the number of training epochs required to train deep networks.

model = Sequential()

model.add(Conv2D(32, kernel\_size=(2, 2), input\_shape=(128, 128, 3), padding = 'Same'))

model.add(Conv2D(32, kernel\_size=(2, 2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size = (2,2), activation ='relu', padding = 'Same'))

model.add(Conv2D(64, kernel\_size = (2,2), activation ='relu', padding = 'Same'))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

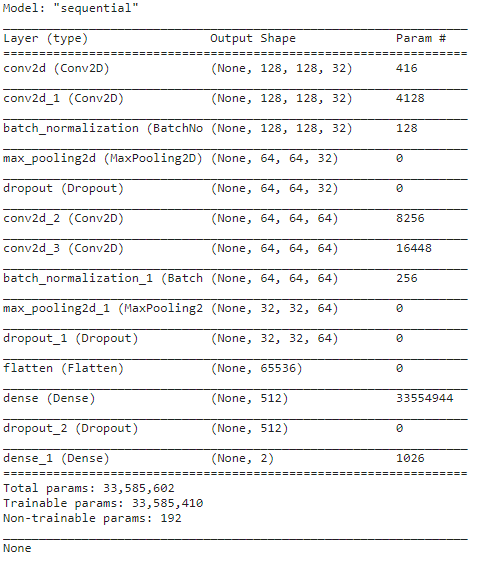
model.add(Dropout(0.5))

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

model.compile(loss = "categorical\_crossentropy", optimizer='Adamax')

print(model.summary())

**Output:**



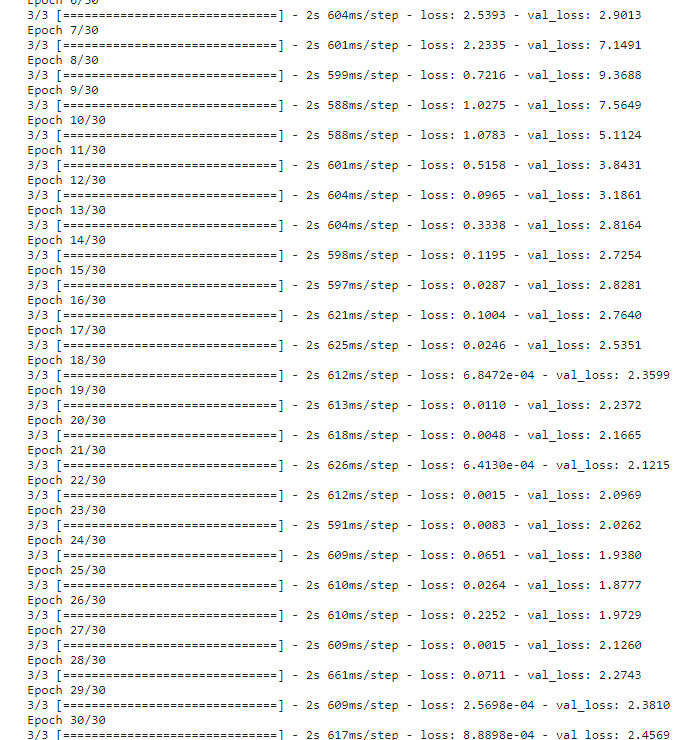
y\_train.shape

**Output:**

(111, 2)

history = model.fit(x\_train, y\_train, epochs = 30, batch\_size = 40, verbose = 1,validation\_data = (x\_test, y\_test))

**Output:**



# Plotting Losses

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

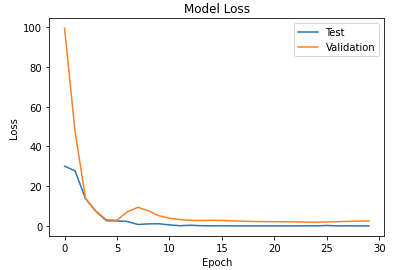
plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Test', 'Validation'], loc='upper right')

plt.show()



# Checking the Model

def names(number):

if number==0:

return 'Its a Tumor'

else:

return 'No, Its not a tumor'

from matplotlib.pyplot import imshow

img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/no/N17.jpg")

x = np.array(img.resize((128,128)))

x = x.reshape(1,128,128,3)

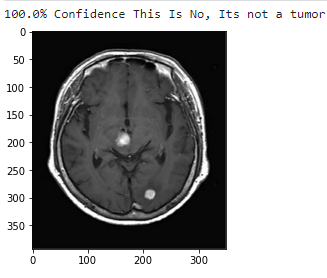
res = model.predict\_on\_batch(x)

classification = np.where(res == np.amax(res))[1][0]

imshow(img)

print(str(res[0][classification]\*100) + '% Confidence This Is ' + names(classification))

**Output:**



from matplotlib.pyplot import imshow

img = Image.open(r"../input/brain-mri-images-for-brain-tumor-detection/yes/Y3.jpg")

x = np.array(img.resize((128,128)))

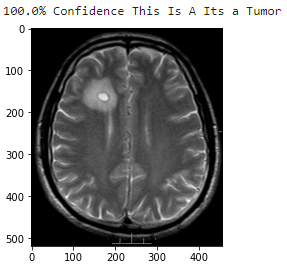
x = x.reshape(1,128,128,3)

res = model.predict\_on\_batch(x)

classification = np.where(res == np.amax(res))[1][0]

imshow(img)

print(str(res[0][classification]\*100) + '% Confidence This Is A ' + names(classification))



**Observations and Conclusion**

* It is observed that the model has 0.93 accuracy
* The model is able to recognize the brain tumor and has 100 % accuracy
* The model has very least loss about 2.34