Brain Tumor Detection and Classification using Convolutional Neural Networks

DSE 309 Advanced Programming in Python
Final Project

By

Name: Sambhav Agrawal

Roll No.: 19264

Department: Data Science and Engineering

BRAIN TUMOR DETECTION AND CLASSIFICATION USING NEURAL NETWORKS

Abstract

According to the World Health Organization (WHO), a brain tumor is a type of tumor that affects the central nervous system. There are basically two types of brain tumors: Cancerous and Non-Cancerous. Brain Tumors are the second most common malignancy in children and it remains one of the major causes of cancer in young and middle age. Brain tumors result from an uncontrolled proliferation of cells derived from neural tissue within the brain. They are the second most common paediatric malignancy and most common paediatric solid tumors. Brain tumors account for 85 to 90 percent of all primary Central Nervous System (CNS) tumors. Every year, around 11,700 people are diagnosed with a brain tumor. Brain tumor cells grow in such a way that they take up all nutrients that are meant for the healthy cells and result in brain failure. In this, detection plays a very crucial role for treatment. If the tumor gets detected, then there are chances to cure it or at least reduce the risk of it. Various image processing techniques are used for the detection. Doctors basically find the position of tumor in brain by looking at MRI images of brain manually. By doing this manually, it takes a lot of time and could also be inaccurate as well.

We can use Deep Learning Architecture Neural Network like VGG16, ResNet50 and many more pretrained models as well for detecting the brain tumor.

Motivation

The motivation behind the brain tumor classification is to automate the task of detecting the brain tumors so that to make sure whether a person has tumor or not. And if the person has a tumor, then it could also detect which type of tumor it is. It could help in early cure of the tumor and could safe many lives. Also, in the developing and poor countries, there is a shortage of trained doctors. With the help of this brain tumor classification, this process could be automated and patients need not go to the doctor to get to know about whether he/she has a tumor or not. They could easily check by uploading their MRI image on the cloud and it would predict about the tumor with excellent accuracy.

Dataset

I have used the following dataset for my project of the brain tumor classification using neural networks:

https://www.kaggle.com/sartajbhuvaji/brain-tumor-classification-mri

This dataset contains 2 folders of Training and Testing Dataset and has 4 classes glioma tumor, meningioma tumor, no tumor and pituitary tumor.

The training folder contains 2870 images and testing folder contains 394 images.

I have appended all the images from the directories into a Python list and then converted them into NumPy arrays after resizing it.

I have given the following labels to the different types of tumors:

0 for Glioma Tumor

1 for No Tumor

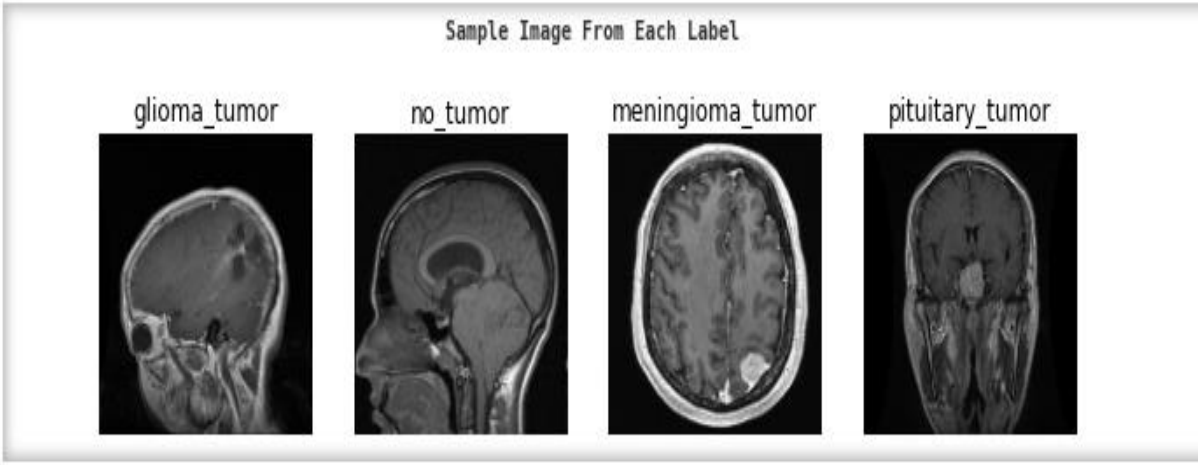
2 for Meningioma Tumor

3 for Pituitary Tumor

Summary of the Dataset

Class	**Number of Images**
Glioma	926
No Tumor	500
Meningioma	937
Pituitary	901
Total	**3264**

Here you can see a sample image from each label of the above mentioned 4 labels.



Dividing the dataset into **Training** and **Testing** sets.

+ Code + Markdown

```
X_train,X_test,y_train,y_test = train_test_split(X_train,y_train, test_size=0.1,random_state=123)
```

Divided the dataset into training and testing set.

One Hot Encoding

Categorical variables are variables whose values takes on the value of labels. Deep Learning neural networks require that both of the input and output to be numbers which means that the categorical data must be encoded to numbers before we can use it to fit to the model and evaluate it further.

Performing **One Hot Encoding** on the labels after converting it into numerical values:

[9]:

```
y_train_new = []
for i in y_train:
    y_train_new.append(labels.index(i))
y_train = y_train_new
y_train = tf.keras.utils.to_categorical(y_train)


y_test_new = []
for i in y_test:
    y_test_new.append(labels.index(i))
y_test = y_test_new
y_test = tf.keras.utils.to_categorical(y_test)
```

Deep Learning Model

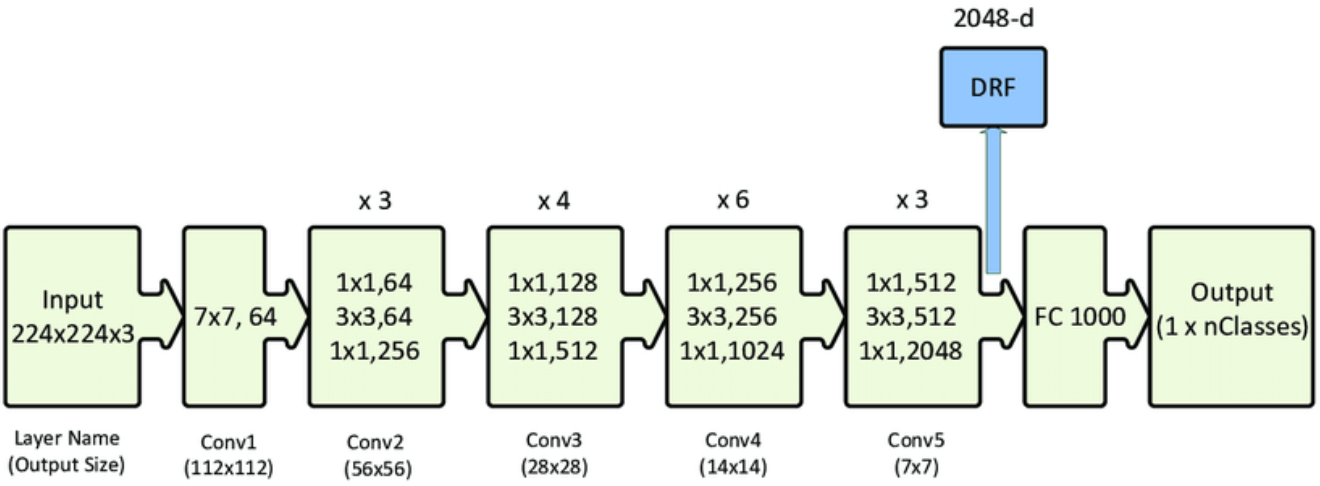
I have used the ResNet50 model for this project. As training these deep learning models could take a lot of computational time and power, so I have used the Transfer Learning technique and have used pretrained ResNet50 model on the ImageNet dataset.

ResNet50 architecture with residual units, size of filters and the outputs of each convolutional layer.

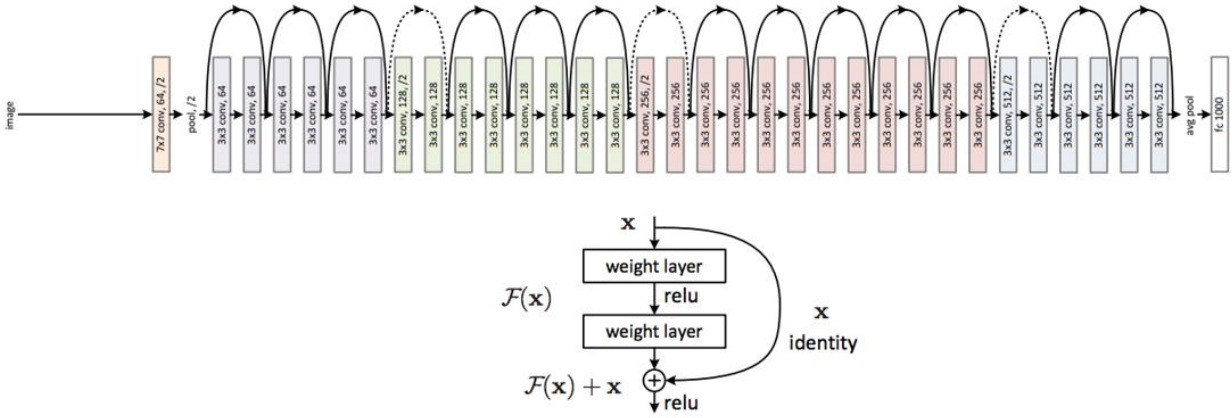
The k x k , n in the convolutional layer block denotes filter of size k and channels n.

The number on top of the convolutional layer block represents the repetition of each unit.

nClasses represents the number of output classes.



Residual Networks (ResNet50)



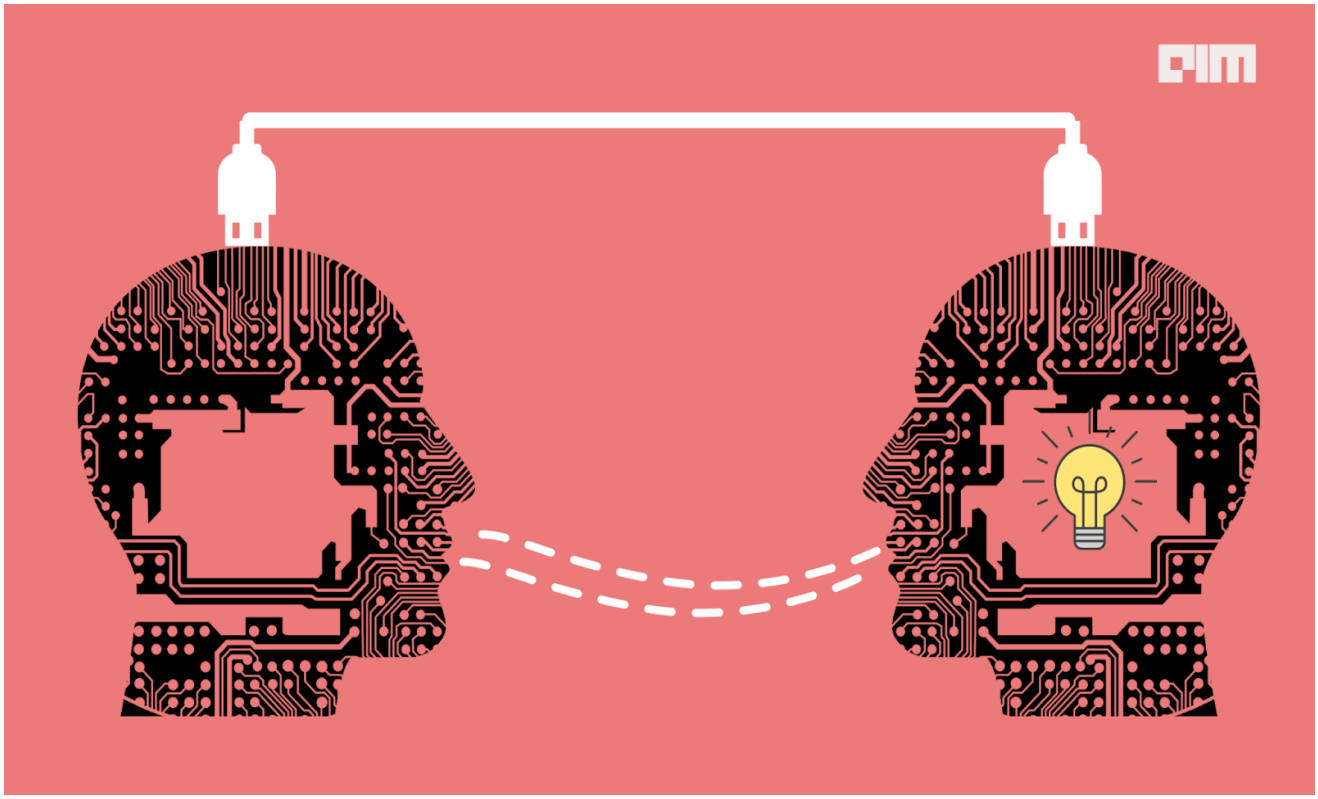
[11]:
```
resnet = ResNet50(weights='imagenet',include_top=False,input_shape=(image_size,image_size,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 0s 0us/step
```

Here, I have used the technique of Transfer Learning meaning using pretrained models on the ImageNet dataset which is an image database designed for use in visual object recognition. It contains more than 14 million images being hand-annotated by project to indicate the objects in the image. It contains more than 20,000 categories.

Here I have supplied weights with ‘imagenet’ as to use the pretrained ImageNet weights for the ResNet50 model. ImageNet is the standard for all image classification tasks.

I have kept include_top = False so as to exclude the fully-connected layer at the top of the network and the output of the last layer get passed through a GlobalAveragePooling2D() which extracts most of the important features.



[12]:

```
model = resnet.output
model = tf.keras.layers.GlobalAveragePooling2D()(model)
model = tf.keras.layers.Dropout(rate=0.5)(model)
model = tf.keras.layers.Dense(4,activation='softmax')(model)
model = tf.keras.models.Model(inputs=resnet.input, outputs = model)
```

GlobalAveragePooling is pooling operation designed to replace fully connected layers in neural networks. Its main work is to generate one feature map for each corresponding category of classification task. It takes the average of each feature map and resulting vector is fed into the SoftMax layer.

The Dropout layer randomly set the input units to 0 with frequency of rate (= 0.5 in our case) at each step during the training time which helps us in preventing overfitting of the model.

[13]:
```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 128, 128, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 134, 134, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 64, 64, 64)   9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 64, 64, 64)   256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 64, 64, 64)   0           conv1_bn[0][0]
```

With the help of model. Summary (), you can see the entire working method of the model along with all types of the convolutional layers, activation functions, etc.

We finally compile our model.

+ Code + Markdown

[14]:
```
model.compile(loss='categorical_crossentropy',optimizer = 'Adam', metrics= ['accuracy'])
```

The compile method requires several parameters. The loss parameter is specified to have type 'categorical_crossentropy'. The metrics parameter is set to 'accuracy' and finally we use the Adam optimizer for training the network. The output at this stage is shown above.

Call Back Functions

I have also used some callback functions to enhance the model as they help to see the training of model better and helps to prevent overfitting by implementing early stopping and can also variate the learning rate on each iteration.

A callback is basically a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

You define and use a callback when you want to automate some tasks after every training/epoch that help you have controls over the training process. This includes stopping training when you reach a certain accuracy/loss score, saving your model

as a checkpoint after each successful epoch, adjusting the learning rates over time, and much more.

Model Checkpoint

This callback saves the model after every epoch. Here are some relevant metrics:

- **filepath**: the file path you want to save your model in
- **monitor**: the value being monitored
- **save_best_only**: set this to True if you do not want to overwrite the latest best model
- **mode**: auto, min, or max. For example, you set mode='min' if the monitored value is val_loss and you want to minimize it.

LearningRateScheduler

It adjusts the learning rate over time using a schedule that you already write beforehand. This function returns the desired learning rate (output) based on the current epoch (epoch index as input).

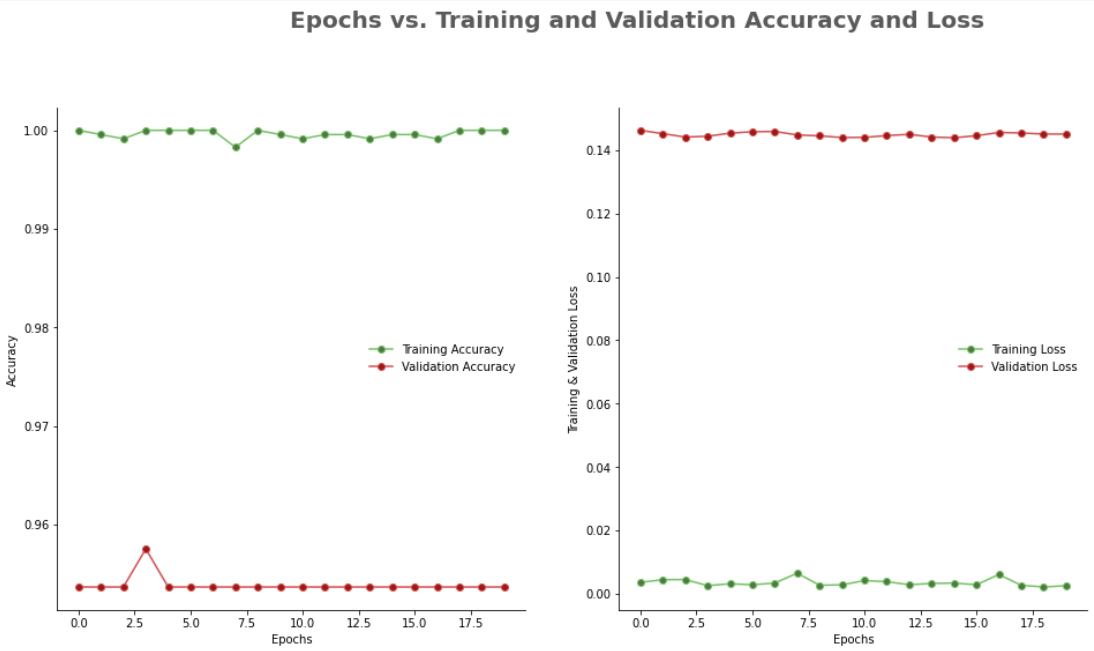
TensorBoard

This callback writes a log for TensorBoard, which is the TensorFlow's excellent visualization tool. It is quite useful for visualising and understanding the runs and graphs.

It also helps to track metrics like loss and accuracy as well.

[33]:

```
tensorboard = TensorBoard(log_dir = 'logs')
checkpoint = ModelCheckpoint("resnet.h5",monitor="val_accuracy",save_best_only=True,mode="auto",verbose=1)
reduce_lr = ReduceLROnPlateau(monitor = 'val_accuracy', factor = 0.3, patience = 1, min_delta = 0.001,
                              mode='auto',verbose=1)
```



Here, I have used the argmax function because in each row of the prediction array, it consists of 4 values of the 4 respective labels of tumors. So the maximum value in each of the row depict the predicted output out of the 4 possible outcomes.

So, the np.argmax() function returns the indices of the max element of the array in a particular axis.

```
pred = model.predict(X_test)
pred = np.argmax(pred,axis=1)
y_test_new = np.argmax(y_test,axis=1)
```

```
print(classification_report(y_test_new,pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        76
           1       0.97      0.95      0.96        41
           2       0.93      0.94      0.93        96
           3       0.95      0.95      0.95        74

    accuracy                           0.95       287
   macro avg       0.95      0.95      0.95       287
weighted avg       0.95      0.95      0.95       287
```

Finally, I have used the classification report from sklearn.metrics. It is used to measure the quality of predictions from a classification problem like True Positives (TP), False Positives (FP) and True Negatives (TN) and False Negatives (FN) are also used for the metrics predictions.

Precision

So basically, precision is the ability of a classifier not to label the instance positive which actually is negative. So, it is defined as the ratio of true positives to sum of true positives and false positives.

Precision = Accuracy of positive predictions.

Precision = TP/(TP+FP)

Recall

Recall is the ability of classifier to find all of the positive instances. It is defined as ratio of true positives to the sum of true positives and false negatives.

Recall = Fraction of the positives that were correctly identified.

Recall = TP/TP+FN

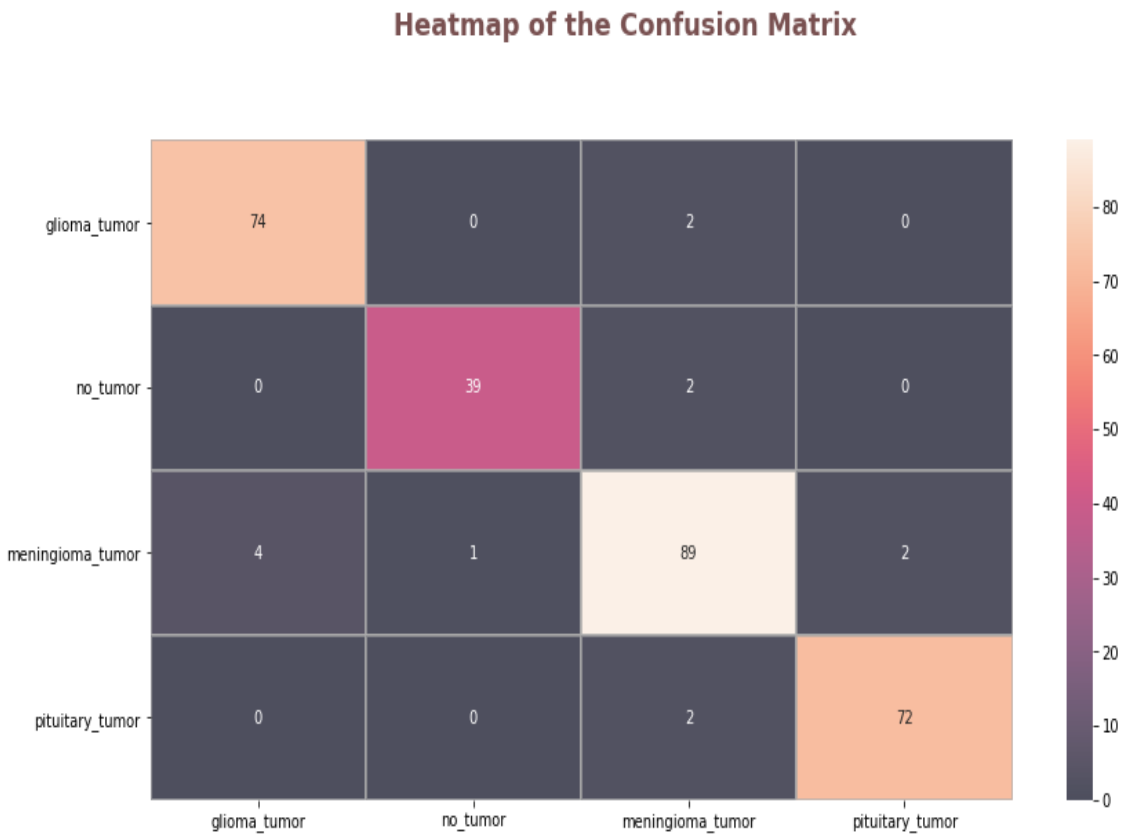
F1-Score

F1-Score is the weighted harmonic mean of precision and recall. It varies between 0 and 1.

F1-Score = 2*(Recall*Precision)/(Recall + Precision)

Support

Support is the number of actual occurrences of the class in the specified dataset.



Confusion Matrix is a performance measurement for the classification problem where the output could be of 2 or more than 2 classes. It is basically a table of True Positive, True Negative and False Positive and False Negative.

In the above image, the diagonal of the table is all correctly predicted values and the rest are incorrectly predicted values.

We could see the variation of the colour according to the numerical values in the above image due to the fact that we have made the heatmap of the confusion matrix itself. We can easily identify the correctly predicted values just by looking at the colour.

Using the Colorama module, we can print coloured text in python. It helps in making our code more readable. It offers three main formatting options: Fore, Back and Style. The colours available are red, green, black, blue, yellow, cyan and white.

```
import colorama
from colorama import Fore



for i in range(100):
    if pred[i] == y_test_new[i] == 0:
        print(Fore.GREEN + 'Correct Prediction Glioma')

    elif pred[i] == y_test_new[i] == 1:
        print(Fore.GREEN + 'Correct Prediction No Tumor')

    elif pred[i] == y_test_new[i] == 2:
        print(Fore.GREEN + 'Correct Prediction Meningioma ')

    elif pred[i] == y_test_new[i] == 3:
        print(Fore.GREEN + 'Correct Prediction Pituitary ')

    else:
        print(Fore.RED + 'InCorrect Prediction')
```

```
Correct Prediction Glioma
Correct Prediction No Tumor
Correct Prediction Pituitary
Correct Prediction No Tumor
Correct Prediction No Tumor
Correct Prediction Glioma
Correct Prediction Meningioma
Correct Prediction Meningioma
Correct Prediction Glioma
Correct Prediction Meningioma
Correct Prediction Glioma
Correct Prediction Pituitary
InCorrect Prediction
Correct Prediction No Tumor
Correct Prediction Glioma
Correct Prediction Glioma
Correct Prediction Meningioma
Correct Prediction Glioma
Correct Prediction Glioma
Correct Prediction Glioma
Correct Prediction Glioma
Correct Prediction Glioma
Correct Prediction Meningioma
Correct Prediction Meningioma
Correct Prediction Glioma
Correct Prediction Pituitary
Correct Prediction Meningioma
```

As we can see that the correct predictions are coloured in green and incorrect predictions in red colour which makes it more readable and clearer.

```
score = model.evaluate(X_test, y_test, verbose = 1)
score[1]

print('The above ResNet50 model gave an accuracy of around 95% which is quite nice. I have modified the ResNet50 model further to get even more accuracy.')
```

```
9/9 [==============================] - 0s 26ms/step - loss: 0.1004 - accuracy: 0.9547
The above ResNet50 model gave an accuracy of around 95% which is quite nice. I have modified the ResNet50 model further to get even more accuracy.
```

Finally, we have used model.evaluate for getting the final accuracy of the model. As we can see that the model is giving an accuracy of around 95% which is excellent.

Conclusion

In brain tumor detection and classification, we got to learn about deep learning techniques and also about the ResNet50. In this project, we successfully detected the presence of tumor in the body and as well as which type of tumor it is also.

However, not everything is said to be perfect in this data science and deep learning domain. More improvement is definitely possible in this project. I got to learn so many new things and concepts in this domain by doing this project.

Acknowledgement

Doing this project would have been very difficult without the constant support of our DSE 309 Professor Parthiban Srinivasan and thanks to the owner of the Brain Tumor MRI images dataset to make the data public.

References

1) Brain Tumor Classification Using Deep Learning Technique - A Comparison between Cropped, Uncropped, and Segmented Lesion Images with Different Sizes
Ali Mohammad Alqudah, Hiam Alquraan , Isam Abu Qasmieh , Amin Alqudah , Wafaa Al-Sharu https://arxiv.org/ftp/arxiv/papers/2001/2001.08844.pdf

2) Three-class brain tumor classification using deep dense inception residual network
Srinath Kokkalla, Jagadeesh Kakarla, Isunuri B. Venkateswarlu and Munesh Singh
https://link.springer.com/article/10.1007/s00500-021-05748-8

3) Muhammad Attique Khan, Imran Ashraf, Majed Alhaisoni, Robertas Damaseviicius, Rafal Scherer, Amjad Rehman, Syed Ahmad Chan BukhariMultimodal Brain Tumor Classification Using Deep Learning and Robust Feature Selection: A Machine Learning Application for Radiologists https://www.mdpi.com/2075-4418/10/8/565/pdf

4) Gupta, Gaurav and Vinay Singh. “Brain Tumor segmentation and classification using Fcm and support vector machine.” (2017).

5) Seetha, J & Selvakumar Raja, S. (2018). “Brain Tumor Classification Using Convolutional Neural Networks. Biomedical and Pharmacology Journal”. 11. 1457-1461. 10.13005/bpj/1511.

6) Mariam Saii, Zaid Kraitem, “Automatic Brain tumor detection in MRI using image processing techniques”, Biomedical Statistics and Informatics, Vol. 2, No. 2, pp. 73-76, 2017.