NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY, EAST CAMPUS

Pituitary Tumour Detection in Brain using Machine Learning

Semester 06

ECAM 02

Submitted to:

Ms. Somiya Rani

Operating Systems

Submitted by:

Aayush Kumar Singh

Harsh Nagar

Dhruv

Naman

INDEX

ABSTRACT

A Brain tumour is considered as one of the aggressive diseases, among children and adults. Brain tumours account for 85 to 90 percent of all primary Central Nervous System (CNS) tumours. Every year, around 11,700 people are diagnosed with a brain tumour. The 5-year survival rate for people with a cancerous brain or CNS tumour is approximately 34 percent for men and36 percent for women. Brain Tumours are classified as: Benign Tumour, Malignant Tumour, Pituitary Tumour, etc. Proper treatment, planning, and accurate diagnostics should be implemented to improve the life expectancy of the patients.

The best technique to detect brain tumours is Magnetic Resonance Imaging (MRI). A huge amount of image data is generated through the scans. These images are examined by the radiologist. A manual examination can be error-prone due to the level of complexities involved in brain tumours and them properties.
Application of automated classification techniques using Machine Learning (ML) and Artificial Intelligence (AI)has consistently shown higher accuracy than manual classification.

Hence, proposing a system performing detection and classification by using Deep Learning Algorithms using Convolution-Neural Network (CNN), Artificial Neural Network (ANN), and Transfer Learning (TL) would be helpful to doctors all around the world.

INTRODUCTION

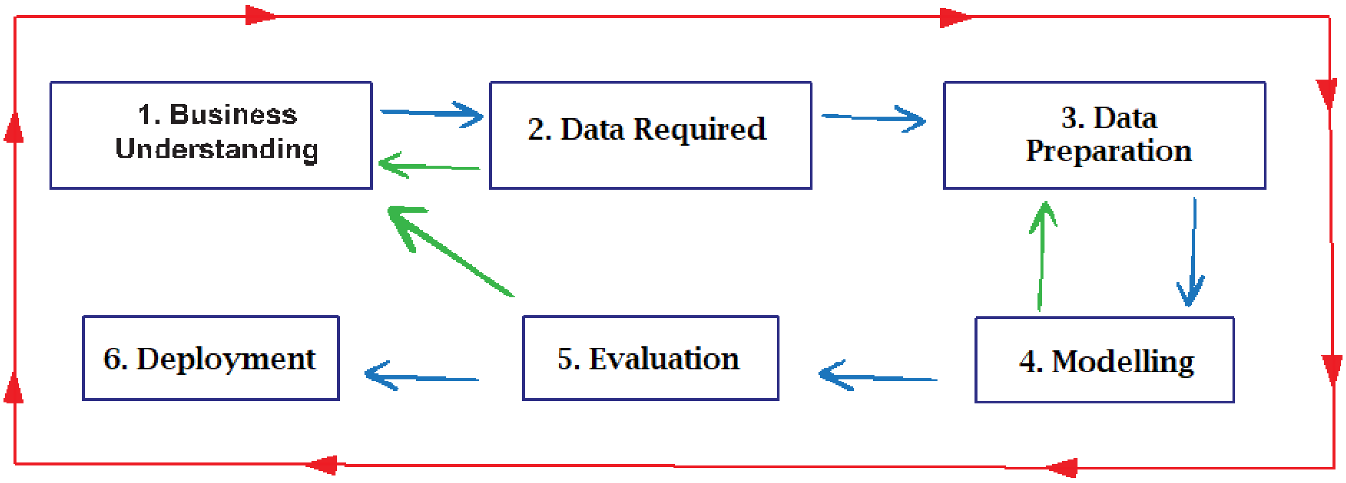
The human body is composed of numerous types of cells. Each cell has a specific function. These cells in the body grow and divide in an arranged manner and form some new cells. These new cells help to keep the human body healthy and ensures proper functioning. When some of the cells lose them ability to control their growth, they grow without any order. The extra cells formed form a mass of tissue which is called tumour. A brain tumour is a collection of abnormal cells in the brain.

Tumours can be benign or malignant. Malignant tumours lead to cancer while benign tumours are not cancerous.

The conventional method for tumour detection in magnetic resonance brain images is human inspection. The observation from human in predicting the tumour may mislead due to noise and distortions found in the images. This method is impractical for large amount of data and also very time consuming.

So, automated tumour detection methods are developed as it would save radiologist time. The MRI brain tumour detection is complex task due to complexity and variant of tumours. Tumour is identified in brain MRI using Machine Learning algorithms. The proposed work is divided into three sections: Pre-processing steps are applied on the brain MRI images, then texture features are extracted using Gray Level Co-occurrence Matrix (GLCM) and finally classification is performed using machine learning algorithm.

METHODOLOGY



1. **Business Understanding:** Intention of our project is outlined from a business perspective which aims to successfully classify brain tumour patients using MRI scans.

2. **Data Required:** We require brain MRI scans of patients diagnosed with pituitary tumour as well as some normal scans, we scraped our data from Kaggle (https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumorclassification-mri) under License CC0: Public Domain. And this data represents our problem.

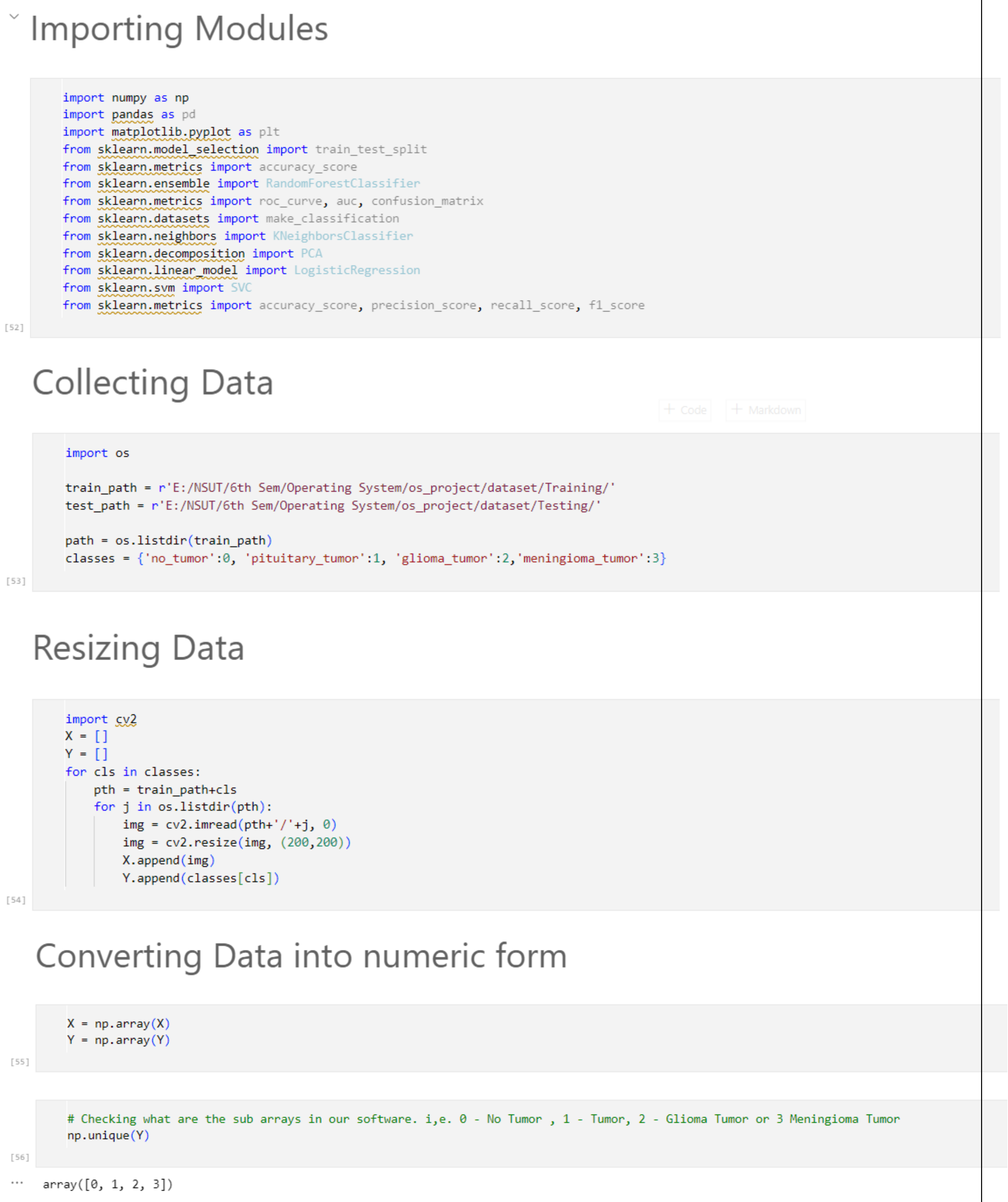
3. **Data Preparation:** Data we are using is much ready to use, for good measures we have resized the images, converted them to suitable dimension arrays, feature scaling and principal component analysis etc before fitting.

4. **Modelling:** Here the data is expressed through apart models to give meaningful insights, new knowledge and patterns. Here we have used Logistic regression as well as support vector machine and then compared their accuracy with svm leading.

5. **Evaluation:** We used. score function of svc and lr to obtain the score for each model.

6. **Deployment:** This tested model is now used on new data outside the scope of dataset, we can now successfully run our model on previously unseen data

Code and Output



Importing Modules

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, auc, confusion_matrix
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```
[52]

Collecting Data

```
import os

train_path = r'E:/NSUT/6th Sem/Operating System/os_project/dataset/Training/'
test_path = r'E:/NSUT/6th Sem/Operating System/os_project/dataset/Testing/'

path = os.listdir(train_path)
classes = {'no_tumor':0, 'pituitary_tumor':1, 'glioma_tumor':2,'meningioma_tumor':3}
```
[53]

Resizing Data

```
import cv2
X = []
Y = []
for cls in classes:
    pth = train_path+cls
    for j in os.listdir(pth):
        img = cv2.imread(pth+'/'+j, 0)
        img = cv2.resize(img, (200,200))
        X.append(img)
        Y.append(classes[cls])
```
[54]

Converting Data into numeric form

```
X = np.array(X)
Y = np.array(Y)
```
[55]

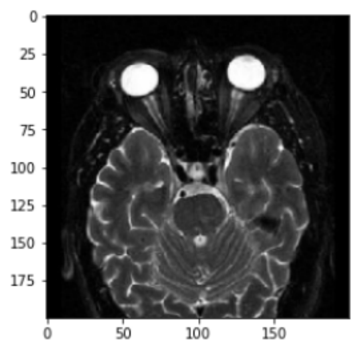
```
# Checking what are the sub arrays in our software. i,e. 0 - No Tumor , 1 - Tumor, 2 - Glioma Tumor or 3 Meningioma Tumor
np.unique(Y)
```
[56]

```
array([0, 1, 2, 3])
```

Visualising Sample Data

```
plt.imshow(X[1], cmap='gray')
```
[59]

```
<matplotlib.image.AxesImage at 0x17e049aee80>
```



Preparing Data

```
# SK Learn works on 2 Dimensional data. But we had 3 Dimensional data above.
# So by reshaping it into 2 dimensional data we can continue
X_updated = X.reshape(len(X), -1)
X_updated.shape
```
[60]

```
(2870, 40000)
```

Using test train split

```
xtrain, xtest, ytrain, ytest = train_test_split(X_updated, Y, random_state=10,
                                                test_size=.20)
```
[61]

```
# 977 sampels for Training the Data
# 245 samples for Testing the Data
xtrain.shape, xtest.shape
```
[62]

```
((2296, 40000), (574, 40000))
```

Feature Scaling

```
# We could have also used standard scaler and min max scaler
# But we used Feature scaler because the RGB value of an image ranges from 0-255
# So we devided the samples by 255
# That is why we are getting all outcomes in 0 or 1
print(xtrain.max(), xtrain.min())
print(xtest.max(), xtest.min())
xtrain = xtrain/255
xtest = xtest/255
print(xtrain.max(), xtrain.min())
print(xtest.max(), xtest.min())
```
[63]

```
255 0
255 0
1.0 0.0
1.0 0.0
```

```
# Principle Component Analisys (Data reduction algo.)
# reduce number of atributes without significant loss in info.

print(xtrain.shape, xtest.shape)

pca = PCA(.98)
# pca_train = pca.fit_transform(xtrain)
# pca_test = pca.transform(xtest)
pca_train = xtrain
pca_test = xtest
```
[64]

```
(2296, 40000) (574, 40000)
```

Logistic Regression

```
# Training Model
# high C means "Trust this training data a lot", while a low value says
#"This data may not be fully representative of the real world data, so if it's telling
#you to make a parameter really large, don't listen to it"
import warnings
warnings.filterwarnings('ignore')
lg = LogisticRegression(C=0.1)
lg.fit(pca_train, ytrain)
```
[65]

```
LogisticRegression(C=0.1)
```

SVM

```
sv = SVC()
sv.fit(pca_train, ytrain)
```
[66]

```
SVC()
```

Random Forest

```
rf = RandomForestClassifier(n_estimators=100, random_state=10)
rf.fit(pca_train, ytrain)
```
[67]

```
RandomForestClassifier(random_state=10)
```

KNN

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(pca_train, ytrain)
```
[68]

```
KNeighborsClassifier()
```

Ridge Classifier

```
from sklearn.linear_model import RidgeClassifier

ridge_clf = RidgeClassifier(alpha=1.0, normalize=True)
ridge_clf.fit(pca_train, ytrain)
```
[76]

```
RidgeClassifier(normalize=True)
```

Naive Bayes

```
# Importing Gaussian Naive Bayes
from sklearn.naive_bayes import GaussianNB

# Creating the Naive Bayes object
nb = GaussianNB()

# Training the Naive Bayes model on the PCA transformed training data
nb.fit(pca_train, ytrain)

# Making predictions
nb_pred = nb.predict(pca_test)
```

[77]

Convolution Neural Network

```
# Load the dataset using ImageDataGenerator
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_path,
    target_size=(200, 200),
    batch_size=32,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    test_path,
    target_size=(200, 200),
    batch_size=32,
    class_mode='categorical')

# Create the architecture of the CNN model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(200, 200, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))
# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train the model
history = model.fit_generator(
    train_generator,
    steps_per_epoch=50,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=50)
```

[79]

```
Found 2870 images belonging to 4 classes.
Found 394 images belonging to 4 classes.
Epoch 1/10
50/50 [==============================] - ETA: 0s - loss: 1.2303 - accuracy: 0.4956WARNING:tensorflow:Your input ran out of data;
50/50 [==============================] - 118s 2s/step - loss: 1.2303 - accuracy: 0.4956 - val_loss: 1.7909 - val_accuracy: 0.2843
Epoch 2/10
50/50 [==============================] - 62s 1s/step - loss: 0.8407 - accuracy: 0.6431
Epoch 3/10
50/50 [==============================] - 64s 1s/step - loss: 0.6683 - accuracy: 0.7157
Epoch 4/10
50/50 [==============================] - 61s 1s/step - loss: 0.5611 - accuracy: 0.7591
Epoch 5/10
50/50 [==============================] - 61s 1s/step - loss: 0.4682 - accuracy: 0.7969
Epoch 6/10
50/50 [==============================] - 65s 1s/step - loss: 0.4118 - accuracy: 0.8352
Epoch 7/10
50/50 [==============================] - 60s 1s/step - loss: 0.3626 - accuracy: 0.8522
Epoch 8/10
50/50 [==============================] - 54s 1s/step - loss: 0.2690 - accuracy: 0.8868
Epoch 9/10
50/50 [==============================] - 56s 1s/step - loss: 0.2612 - accuracy: 0.9006
Epoch 10/10
50/50 [==============================] - 56s 1s/step - loss: 0.2325 - accuracy: 0.9151
```

Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier

# Create a Decision Tree classifier with max_depth = 5
dtc = DecisionTreeClassifier(max_depth=5)

# Train the model
dtc.fit(xtrain,ytrain)
```
[81]

```
DecisionTreeClassifier(max_depth=5)
```

Evaluation Logistic Regression

```python
y_pred = lg.predict(pca_test)
print(classification_report(ytest, y_pred))
lg_train_score = lg.score(pca_train, ytrain)
lg_test_score = lg.score(pca_test, ytest)
print("Training Score: ", lg_train_score)
print("Testing Score: ",lg_test_score)
```
[82]

```
              precision    recall  f1-score   support

           0       0.78      0.69      0.73        91
           1       0.87      0.94      0.91       162
           2       0.68      0.78      0.73       145
           3       0.70      0.61      0.65       176

    accuracy                           0.76       574
   macro avg       0.76      0.76      0.76       574
weighted avg       0.76      0.76      0.76       574

Training Score:  1.0
Testing Score:  0.759581881533101
```

Evaluation SVM

```python
y_pred = sv.predict(pca_test)
print(classification_report(ytest, y_pred))
sv_train_score = sv.score(pca_train, ytrain)
sv_test_score = sv.score(pca_test, ytest)
print("Training Score: ", sv_train_score)
print("Testing Score: ", sv_test_score)
```
[83]

```
              precision    recall  f1-score   support

           0       0.79      0.76      0.78        91
           1       0.86      0.98      0.92       162
           2       0.78      0.83      0.81       145
           3       0.81      0.69      0.75       176

    accuracy                           0.82       574
   macro avg       0.81      0.81      0.81       574
weighted avg       0.82      0.82      0.81       574

Training Score:  0.9390243902439024
Testing Score:  0.8170731707317073
```

Evaluation Random Forest

```python
y_pred = rf.predict(pca_test)
print(classification_report(ytest, y_pred))
rf_train_score = rf.score(pca_train, ytrain)
rf_test_score = rf.score(pca_test, ytest)
print("Training Score: ",rf_train_score )
print("Testing Score: ",rf_test_score  )
```

[84]

```
              precision    recall  f1-score   support

           0       0.81      0.91      0.86        91
           1       0.90      0.98      0.94       162
           2       0.98      0.85      0.91       145
           3       0.85      0.83      0.84       176

    accuracy                           0.89       574
   macro avg       0.89      0.89      0.89       574
weighted avg       0.89      0.89      0.89       574

Training Score:  1.0
Testing Score:  0.8902439024390244
```

Evaluation KNN

```python
y_pred = knn.predict(pca_test)
print(classification_report(ytest, y_pred))
knn_train_score = knn.score(pca_train, ytrain)
knn_test_score = knn.score(pca_test, ytest)
print("Training Score:", knn_train_score)
print("Testing Score:", knn_test_score )
```

[85]

```
              precision    recall  f1-score   support

           0       0.71      0.74      0.72        91
           1       0.83      0.98      0.90       162
           2       0.71      0.93      0.81       145
           3       0.88      0.50      0.64       176

    accuracy                           0.78       574
   macro avg       0.78      0.79      0.77       574
weighted avg       0.80      0.78      0.77       574

Training Score: 0.8732578397212544
Testing Score: 0.7804878048780488
```

Evaluation Ridge Classifier

```python
# Evaluate the model on the test data
y_pred = ridge_clf.predict(pca_test)
print(classification_report(ytest, y_pred))
ridge_clf_train_score = ridge_clf.score(pca_train, ytrain)
ridge_clf_test_score = ridge_clf.score(pca_test, ytest)
print("Training Score: {:.2f}".format(ridge_clf_train_score))
print("Testing Score: {:.2f}".format(ridge_clf_test_score))
```

[86]

```
...              precision    recall  f1-score   support

           0       0.83      0.64      0.72        91
           1       0.88      0.99      0.93       162
           2       0.71      0.86      0.78       145
           3       0.80      0.66      0.72       176

    accuracy                           0.80       574
   macro avg       0.80      0.79      0.79       574
weighted avg       0.80      0.80      0.80       574

Training Score: 1.00
Testing Score: 0.80
```

Evaluation Naive Bayes

```python
y_pred = nb.predict(pca_test)
print(classification_report(ytest, y_pred))

nb_train_score = nb.score(pca_train, ytrain)
nb_test_score = nb.score(pca_test, ytest)
print("Training Score: {:.2f}".format(nb_train_score))
print("Testing Score: {:.2f}".format(nb_test_score))
```
[87]

```
...              precision    recall  f1-score   support

           0       0.62      0.59      0.61        91
           1       0.76      0.78      0.77       162
           2       0.48      0.90      0.63       145
           3       0.40      0.11      0.17       176

    accuracy                           0.58       574
   macro avg       0.57      0.60      0.54       574
weighted avg       0.56      0.58      0.52       574

Training Score: 0.62
Testing Score: 0.58
```

Evaluation CNN

+ Code + Markdown

```python
score = model.evaluate_generator(validation_generator, steps=50)

cnn_train_score = history.history["accuracy"][0]
cnn_test_score = score[1]

print("cnn_train_score: ", cnn_train_score)
print("cnn_test_score: ", cnn_test_score)
```
[88]

```
... WARNING:tensorflow:Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch
cnn_train_score:  0.4955974817276001
```

Evaluation Decision Tree

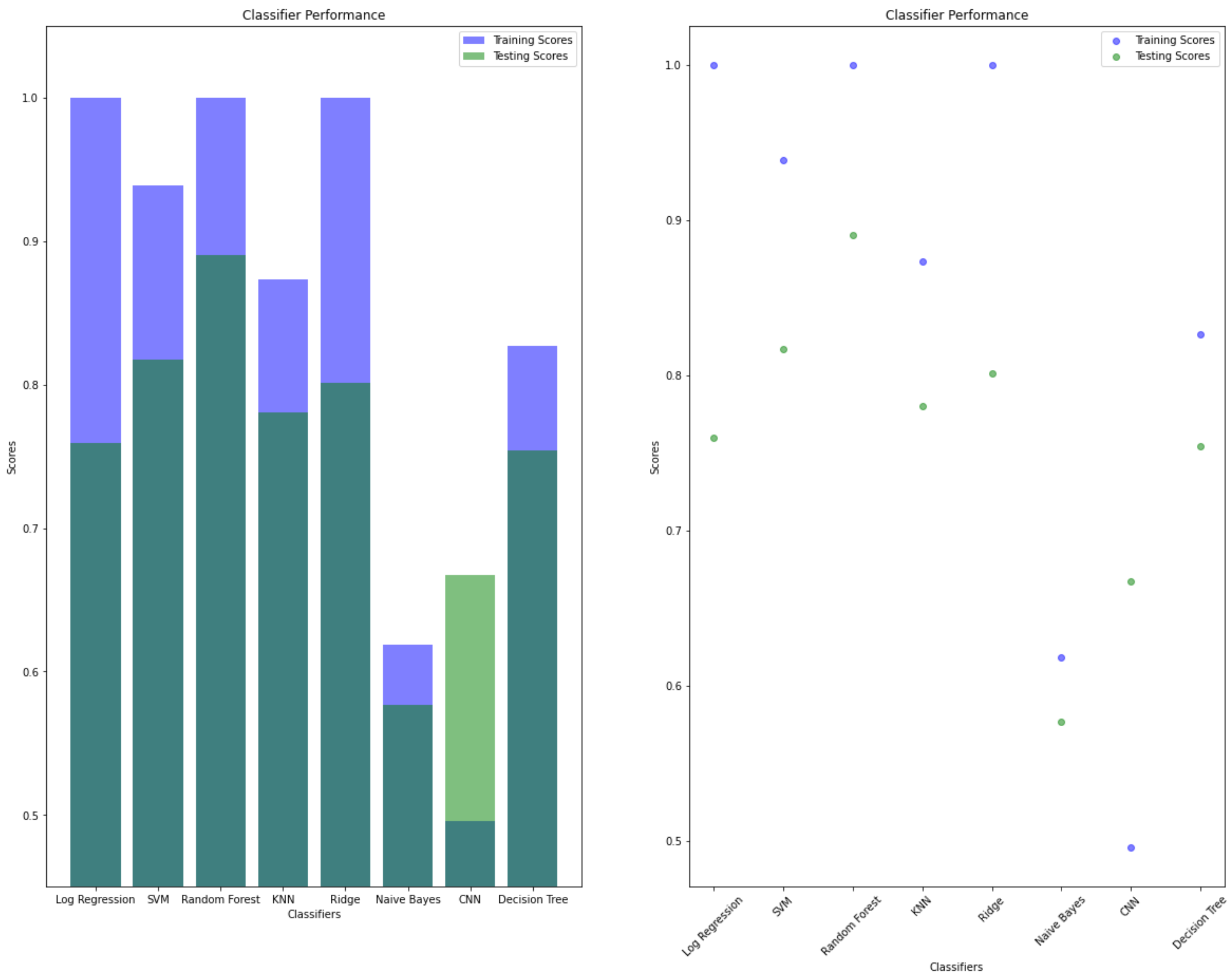
```python
# Evaluate the model
y_pred = dtc.predict(pca_test)
print(classification_report(ytest, y_pred))
dt_train_score = dtc.score(pca_train, ytrain)
dt_test_score = dtc.score(pca_test, ytest)
print("Training Score: {:.2f}".format(dt_train_score))
print("Testing Score: {:.2f}".format(dt_test_score))
```
[89]

```
...              precision    recall  f1-score   support

           0       0.76      0.64      0.69        91
           1       0.77      0.88      0.82       162
           2       0.91      0.68      0.77       145
           3       0.66      0.77      0.71       176

    accuracy                           0.75       574
   macro avg       0.77      0.74      0.75       574
weighted avg       0.77      0.75      0.75       574

Training Score: 0.83
Testing Score: 0.75
```

Deducing the best classifier

```python
classifiers = ['Log Regression', 'SVM', 'Random Forest', 'KNN', 'Ridge', 'Naive Bayes', 'CNN', 'Decision Tree']
training_scores = [lg_train_score, sv_train_score, rf_train_score, knn_train_score, ridge_clf_train_score, nb_train_score, cnn_train_score, dt_train_score]
testing_scores =  [lg_test_score, sv_test_score, rf_test_score, knn_test_score, ridge_clf_test_score, nb_test_score, cnn_test_score, dt_test_score]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 15))

x_pos = [i for i, _ in enumerate(classifiers)]
ax1.bar(x_pos, training_scores, color='blue', alpha=0.5, label='Training Scores')
ax1.bar(x_pos, testing_scores, color='green', alpha=0.5, label='Testing Scores')
ax1.set_xlabel('Classifiers')
ax1.set_ylabel('Scores')
ax1.set_ylim(0.45,1.05)
ax1.set_title('Classifier Performance')
ax1.set_xticks(x_pos)
ax1.set_xticklabels(classifiers)
ax1.legend()

ax2.scatter(classifiers, training_scores, color='blue', alpha=0.5, label='Training Scores')
ax2.scatter(classifiers, testing_scores, color='green', alpha=0.5, label='Testing Scores')
ax2.set_xlabel('Classifiers')
ax2.set_ylabel('Scores')
ax2.set_title('Classifier Performance')
ax2.tick_params(axis='x', rotation=45)
ax2.legend()

plt.show()
```

[104]



```
max_metric = max(zip(training_scores,testing_scores), key=lambda pair: max(pair))
max_index = max_metric.index(max(max_metric))

print("The metric with the maximum value is:", classifiers[max_index])
```
[105]

```
The metric with the maximum value is: Log Regression
```

- Hence, we can see that **Logistic Regression** works with greater accuracy than rest here.

Prediction

```
pred = lg.predict(pca_test)
np.where(ytest!=pred)
```
[107]

```
(array([  1,   4,   8,   9,  14,  19,  21,  25,  32,  35,  40,  44,  48,
        51,  59,  60,  75,  76,  78,  86,  97,  99, 101, 112, 117, 118,
       120, 122, 125, 126, 128, 135, 138, 140, 142, 143, 146, 150, 152,
       159, 163, 164, 165, 171, 174, 177, 181, 182, 183, 184, 187, 200,
       202, 203, 214, 216, 217, 218, 224, 230, 233, 234, 242, 243, 254,
       255, 256, 269, 270, 275, 286, 287, 289, 299, 300, 303, 305, 306,
       317, 322, 324, 326, 329, 335, 337, 338, 342, 359, 369, 370, 372,
       374, 380, 382, 384, 394, 396, 401, 409, 419, 420, 423, 425, 433,
       438, 441, 445, 448, 449, 450, 451, 453, 454, 456, 459, 468, 470,
       471, 482, 493, 501, 505, 512, 513, 517, 520, 522, 525, 530, 535,
       537, 543, 555, 556, 558, 560, 564, 571], dtype=int64),)
```

Testing Model

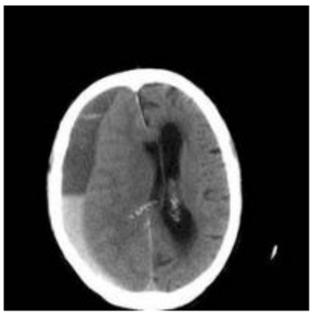
```
dec = {0:'No Tumor', 1:'Pituitary Tumor', 2:'Glioma Tumor', 3:'Meningioma Tumor'}#Classes
classes_keys = list(classes.keys())
```
[108]

```
def pred_func(x):
    plt.figure(figsize=(12,8))
    p = os.listdir(test_path)
    c=1
    for i in os.listdir(test_path + f'/{x}')[:4]:
        plt.subplot(2,2,c)

        img = cv2.imread(test_path + f'/{x}/'+i,0)
        if img is None:
            print('Wrong path:', path)
        else:
            img1 = cv2.resize(img, (200,200))
            img1 = img1.reshape(1,-1)/255
            p = rf.predict(img1)
            plt.title(dec[p[0]])
            plt.imshow(img, cmap='gray')
            plt.axis('off')
            c+=1
```
[109]

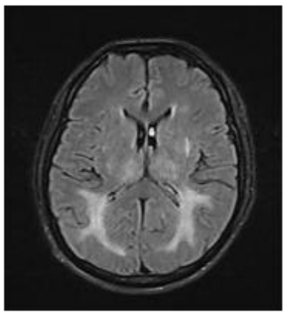
- On No tumor test dataset

```
pred_func(classes_keys[0])
```
[110]

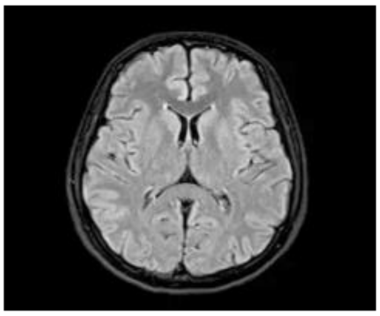
No Tumor



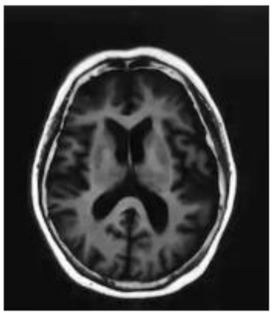
No Tumor



No Tumor



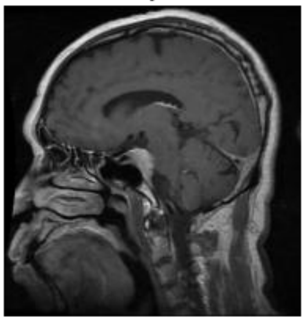
No Tumor



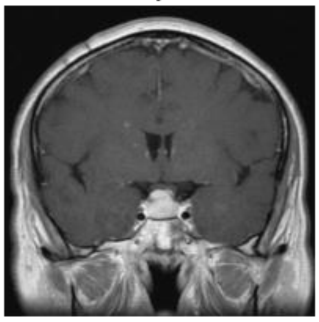
- On Pituitary Tumor test dataset

```
[111]
pred_func(classes_keys[1])
```

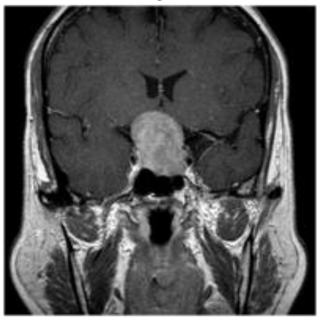
Pituitary Tumor



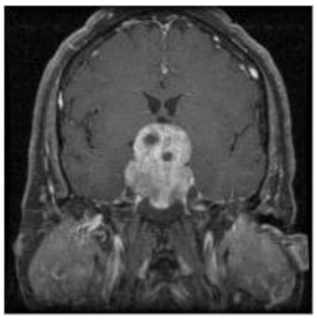
Pituitary Tumor



Pituitary Tumor



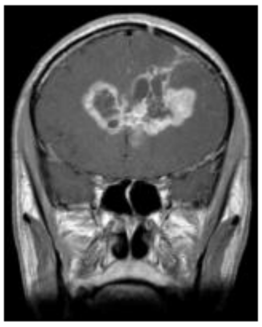
No Tumor



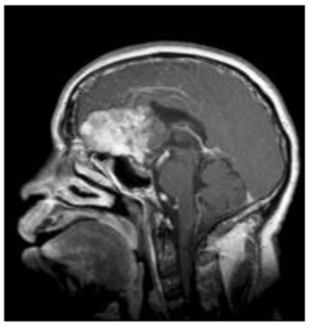
- On Glioma Tumor test dataset

```
[112]
pred_func(classes_keys[2]) #Wrong predictions here
```

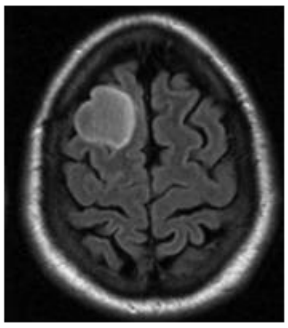
No Tumor



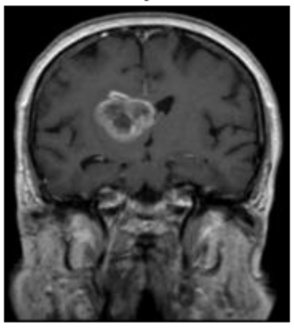
No Tumor



No Tumor



Pituitary Tumor

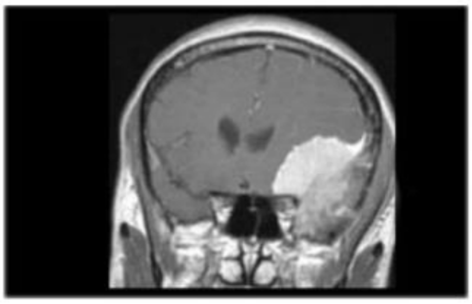


- On Meningioma Tumor test dataset

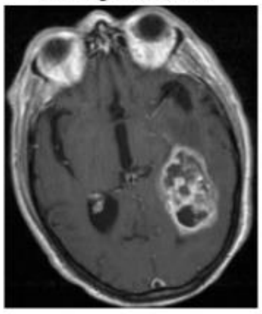
```
pred_func(classes_keys[3])
```

[113]

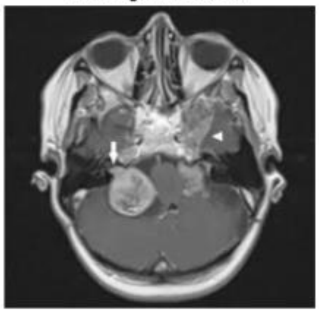
No Tumor



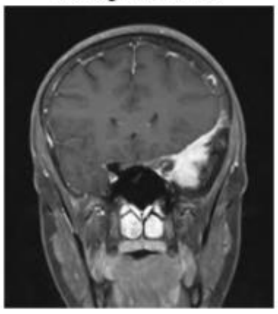
Meningioma Tumor



Meningioma Tumor

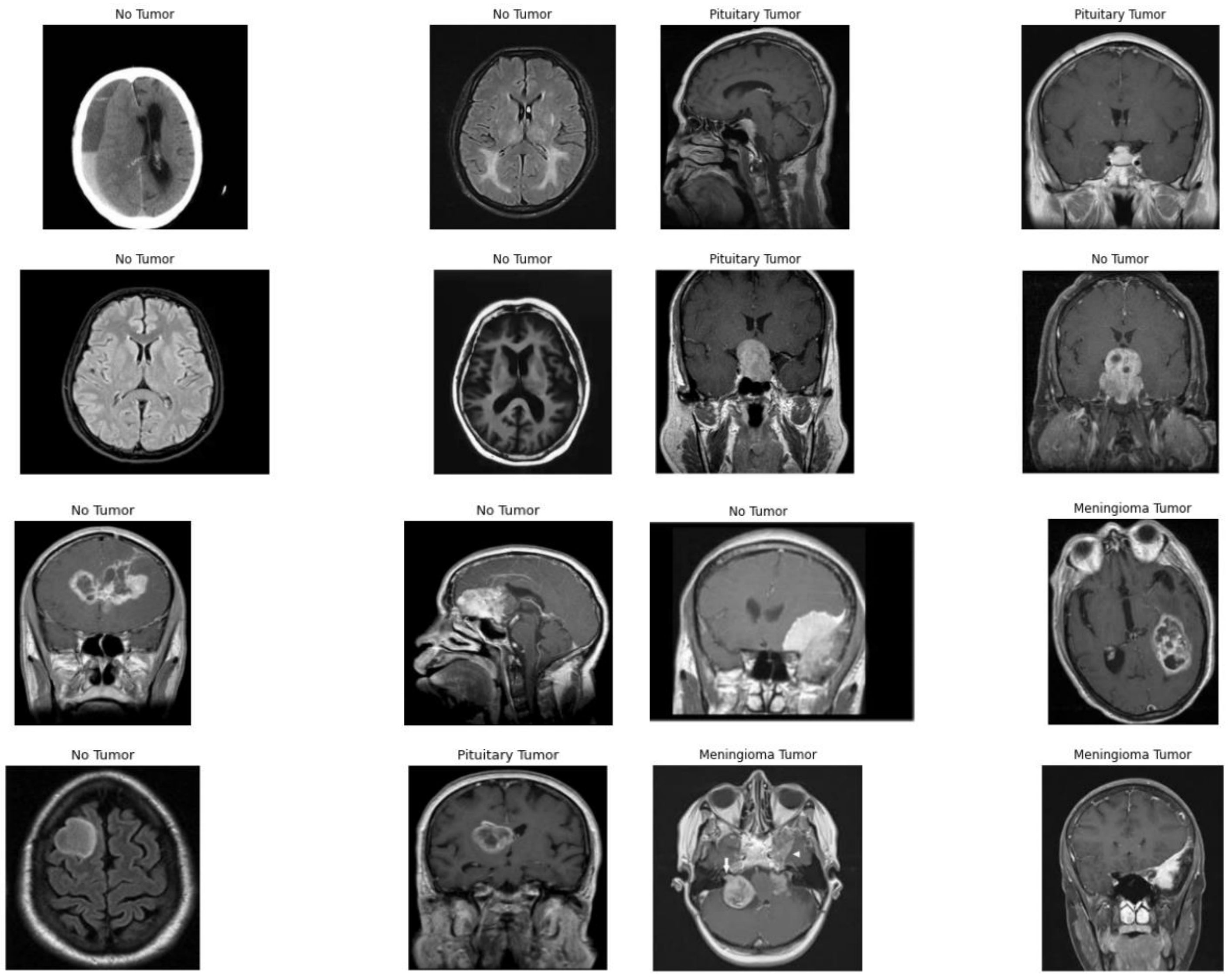


Meningioma Tumor



RESULTS

We have successfully detected whether a patient has pituitary tumor, glioma tumor, meningioma or No tumor at all using Brain MRI scans, with little to false negatives

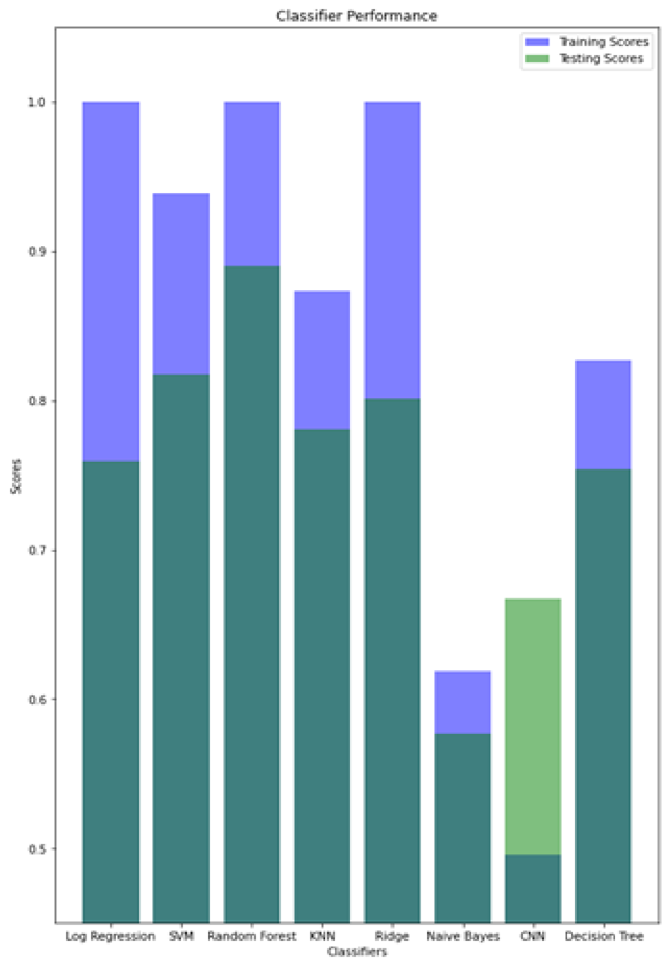


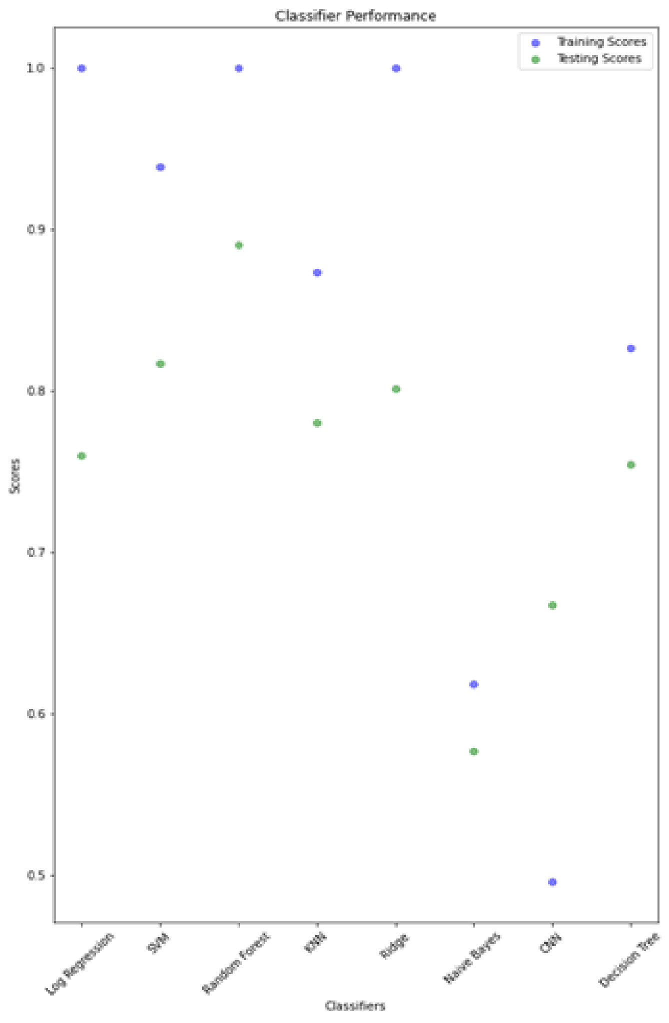
Conclusion

We used the following classifiers from SciKitLearn library of python to conclude our project:

1. Logistic Regression
2. Support Vector Machine
3. Random Forest
4. K-Nearest Neighbour
5. Ridge Classifier
6. Naïve Bayes
7. Decision Tree

Following depicts their respective test and train scores:





- We also included a sequential CNN model using keras Library of Tensorflow.
- Logistic Regression had the highest accuracy in all 8 models.
- Only the CNN model had its test accuracy greater than train since it’s a dl model.
- ML models are suspected to be overfitted since train accuracy is 100%.
- Hence, we concluded **Logistic Regression** to be the Best classifier and carried our project with it.

References

Youtube: https://www.youtube.com/watch?v=5lgrlddp-98

Kaggle:
https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbXFBcFVndU1nM05QUEJ5UHVTMmxjWEZuUHF5QXxBQ3Jtc0trUXdiTUhOS1Jjb0RVNmpPdlpnLTh1bTl2ZnVlaFB1LXJac1huYkUyR19YMEJvUVFNLXNWbC1sUWRSTTBYQ1RRTnY4MlVGRWR2V1V2VkU3N01QZjl5UzRPd0JyaVR4emZoMGdQOXUyaURMVWt0SjhCWQ&q=https%3A%2F%2Fwww.kaggle.com%2Fsartajbhuvaji%2Fbrain-tumor-classification-mri&v=5lgrlddp-98

Geeks for Geeks: https://www.geeksforgeeks.org/tumor-detection-using-classification-machine-learning-and-python/?ref=gcse