TABLE OF CONTENTS

ABSTRACT

Classifying images of brain tumours is a crucial aspect of medical image processing. It helps medical professionals diagnose and treat patients accurately. One of the primary imaging techniques for studying brain tissue is magnetic resonance imaging (MRI). Here, we present an approach to classify MR images of brain tumours using CNN ResNet50 and Xception pretrained models.

Although deep learning models have been deployed in many fields, sensitive applications like medical imaging still need to be adjusted for. Because of the time constraints, technology is used in the medical field; yet, reliability is ensured by the accuracy level. Medical machine learning programmes are unable to use medical data due to privacy issues.

For instance, utilising image-based classification to diagnose brain tumours is challenging due to a scarcity of brain MRI pictures. This task was solved with the help of the CNN application. Because of its effective depthwise separable convolutions, Xception—which has 71 layers—often surpasses ResNet-50, achieving an accuracy of 99% compared to 97% for ResNet-50. When pretrained on datasets like as ImageNet, this architecture has improved transfer learning capabilities and captures complex characteristics. Its improved capacity and useful regularisation strategies add to its potential for improved accuracy across a range of jobs.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. Brain Tumour

Impactful solutions are being offered by the context-appropriate deployment of deep learning techniques to enhance medical diagnostics. The World Health Organisation (WHO) states that a correct diagnosis of a brain tumour entails its discovery, location, and classification based on its grade, kind, and malignancy. In this experimental effort, the tumour is detected, its grade and kind are classified, and its position is identified in relation to the use of Magnetic Resonance Imaging (MRI) for brain tumour diagnosis.

Instead of using a distinct model for every classification assignment, this strategy has experimented with using a single model to categorise brain MRI on several tasks. Tumour classification and detection are supported by the Convolutional Neural Network (CNN) based multi-task classification system. By segmenting the brain tumour, a CNN-based model is also used to determine the location of the tumour.

1.2. STATISTICAL INFORMATION

Individuals of all ages, races, nationalities, and genders have been documented to have brain tumours. Currently, more than 1.3 million Americans suffer from a primary or secondary/metastatic brain tumour.3. Meningiomas, pituitary tumours, and gliomas are the three most prevalent kinds of primary brain tumours. Secondary brain tumours, also known as metastatic brain tumours, originate in an organ other than the brain, such as the lung or breast, and then move to the brain. The most typical brain tumours are these ones.

Every year, about 90,000 people receive a primary brain tumour diagnosis. Fifth most frequent cancer is brain and other central nervous system tumours. More than a million individuals are dealing with a primary brain tumour diagnosis. Primary brain and central nervous system tumours come in more than a hundred varieties.

Malignant brain and central nervous system (CNS) tumours account for about one-third (27.0%) of all tumours. The most prevalent cancer in children aged 0-14 to be diagnosed is brain and central nervous system tumours. There are already more than 28,000 youngsters in the US with brain tumour diagnoses. Every year, over 3,400 kids (ages 0 to 14) receive a primary brain tumour diagnosis. Every year, 12,800 teenagers and young adults (ages 15 to 39) receive a primary brain tumour diagnosis which is a lot if considered.

The age group 85 years and above has the highest incidence rate of brain and central nervous system tumours. About 17,200 people lose their lives to malignant brain tumours each year.

The prognosis following a primary brain tumour diagnosis is highly dependent on a few factors, including age, race, location, kind of tumour, molecular markers, and tumour type.

1.3 CURRENT DIAGNOSIS METHODS

Brain magnetic resonance imaging. Strong magnets are used in magnetic resonance imaging, or MRI, to provide images of the inside of the body. Since MRI provides a clearer image of the brain than other imaging procedures, it is frequently used to identify brain-tumours.

Before an MRI, a dye is frequently injected into an arm vein. The dye produces more vivid images. Smaller tumours are simpler to see as a result. It can assist your medical team in distinguishing between healthy brain tissue and a brain tumour.
To get more detailed images, an MRI of a particular kind is occasionally required. Functional MRI is one instance. This unique MRI illustrates which brain regions are responsible for language, movement, and other critical functions. This aids in the planning of surgeries and other treatments by your healthcare professional.

Magnetic resonance spectroscopy is an additional unique MRI exam. MRI is used in this test to quantify specific chemical levels in the tumour cells. Excessive or insufficient levels of these substances may provide information to your medical team on the type of brain tumour you have.

An additional unique kind of MRI is magnetic resonance perfusion. This test measures the blood volume in several brain tumour regions using magnetic resonance imaging. The areas of the tumour with the most blood flow might also be the most active. This information is used by your healthcare staff to schedule your appointments.

2. LITERATURE REVIEW

The literature review highlights the challenges of brain tumor diagnosis and the need for non-invasive computational methods. Existing approaches suffer from low sensitivity and long processing times. This study proposes a novel method combining advanced segmentation (Canny Mayfly), feature selection (Enhanced

Chimpanzee Optimization Algorithm), and deep learning (ResNet-152) for brain tumor classification. Python is used for implementation and evaluation on the Figshare dataset. Results show superior performance with 98.85% accuracy, indicating promise for improving brain tumor diagnosis.

3. OBJECTIVE

To build an efficient deep learning models to classify both binary (normal and abnormal) and multiclass (meningioma, glioma, and pituitary) brain tumors. We use publicly available datasets that include 7023 MRI images. Implement 2 models using Xception and Resnet50 pre-trained models. Finally, we compare our proposed models with those reported in the literature.

4. PROPOSED METHODOLOGY

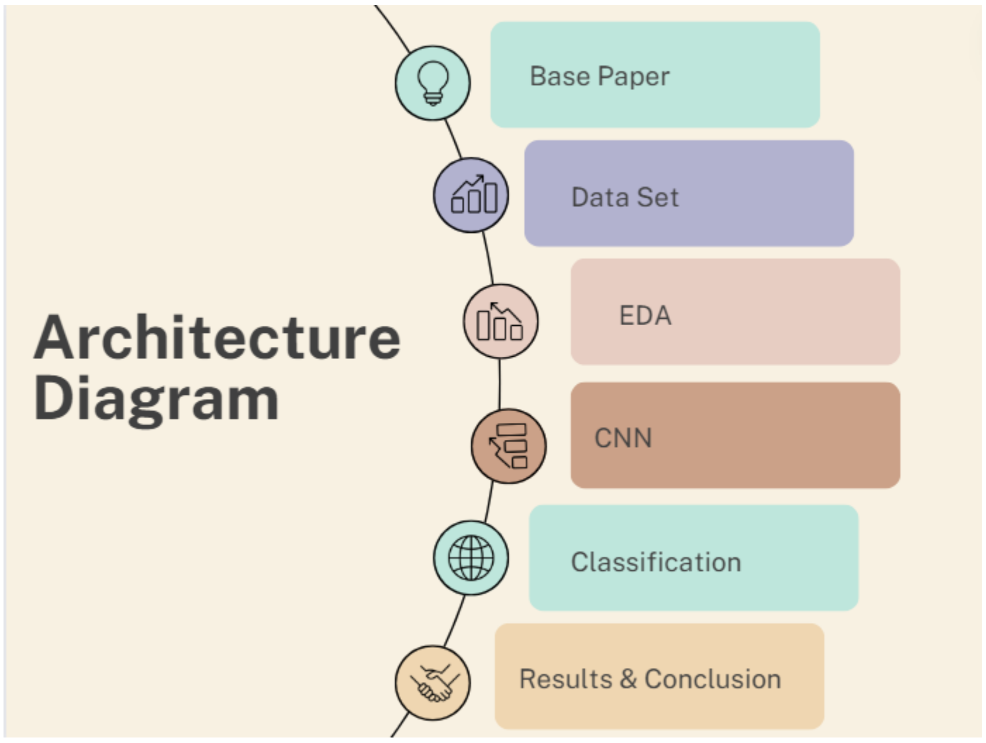


Fig 4.1 Methodology

This project aims to classify MRI brain scans of patients using deep learning techniques while utilizing the machine learning models to predict the type of tumour. The MRI images were sourced from Kaggle which has 7023 images all shaped in 299x299x3.They were divided into 4 classes of images namely; *glioma*, *meningioma*, *pituitary* and no tumour. Pretrained models were utilized to predict the images into the 4 different classes.

While using the MRI scans from Kaggle was extracted and pre-processing techniques like train test split and Standardisation was carried out. Then Restnet50 and Xception deep learning techniques were utilized and accuracy, confusion matrix was used to find the efficiency of the model.

5. TOOLS AND TECHNIQUES

There are several tools commonly used for deep learning classification, depending on the specific application and the level of expertise of the practitioner. Here are some popular tools used for deep learning classification:

1. TensorFlow: TensorFlow is an open-source deep learning framework developed by Google. It is widely used for classification tasks, including image classification, natural language processing, and speech recognition.

2. Keras: Keras is a high-level deep learning framework that can be used with TensorFlow or Theano as the backend. It provides an easy-to-use API for building and training deep learning models, including classification models.

3. Scikit-learn: Scikit-learn is a popular machine learning library for Python that includes many algorithms for classification tasks, including deep learning algorithms such as multi-layer perceptrons (MLPs).

4. MATLAB: It is a programming language and environment that includes many deep learning tools and features, including support for popular deep learning frameworks like TensorFlow and PyTorch.

6 IMPLEMENTATION

6.1 BRAIN TUMOR CLASSIFICATION USING MRI

6.1.1. ABOUT THE DATASET:

The Data is hand collected from various websites with each and every labels verified and sourced Kaggle.

THE 4 CLASSES:

GLIOMA:1621

MENINGIOMA: 1645

PITUITARY: 1895

NO TUMOUR: 1862

IMAGE SHAPE: 299X299X3

NO OF ROWS: 7023

6.1.2. PREPROCESSING

Pre-processing is just as important in deep learning as it is in other areas of machine learning and data analysis. In fact, it can be argued that it is even more critical in deep learning due to the complexity of the models and the large amount of data typically involved.

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data. It includes making minor changes to the dataset or using deep learning to generate new data points. Techniques include resizing, zoom, rotating, cropping, padding, etc. It helps to address issues like overfitting and data scarcity, and it makes the model robust with better performance.

6.1.3. MODELS

Resnet50

Deep residual networks like the popular ResNet-50 model is a convolutional neural network (CNN) that is 50 layers deep. A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that stacks residual blocks on top of each other to form a network. It can work with different input size and designed to overcome problem of vanishing gradient by using residual connections.

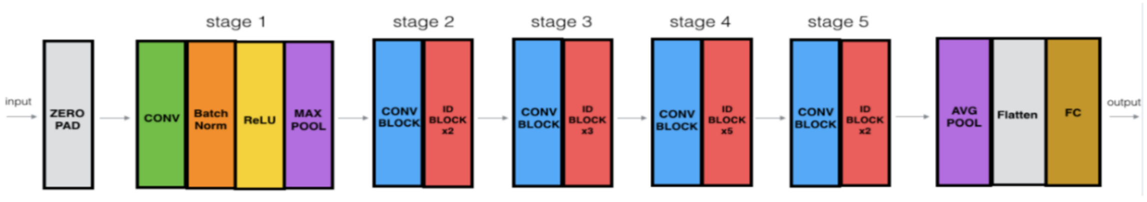


Fig 6.1 Architecture of Resnet50

- Architecture includes
    - It contains an initial convolutional layer, followed by a max pooling layer.
    - Four sets of convolutional layers, with each set containing several residual blocks.
    - A global average pooling layer.
    - A fully connected layer with a softmax activation function.

XCEPTION

- Xception is a pre-trained convolutional neural network that is 71 layers deep, which is a version of the network already trained on more than a million images from the ImageNet database. This pre-trained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.
- As a result, the network has learned rich feature representations for a wide range of images. The model extracts general features from input images in the first part and classifies them based on those features in the second part.

6.1.4. EVALUATION

Evaluation is a critical step in any deep learning project as it helps to assess the performance of the model and identify areas for improvement. There are several evaluation metrics that are commonly used in deep learning, depending on the task and the type of model being evaluated.

1. Accuracy: Accuracy is the most commonly used evaluation metric for classification tasks. It measures the proportion of correct predictions made by the model.

2. Precision and Recall: Precision and Recall are evaluation metrics used in binary classification tasks. Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positive cases.

3. F1 Score: The F1 score is a weighted average of precision and recall that combines the two metrics into a single score. It is often used in binary classification tasks where precision and recall are both important.

4. Area Under the Curve (AUC): The AUC is a metric that measures the overall performance of a classification model across all possible classification thresholds.

6.2. Brain Tumor Data CLASSIFICATION

6.2.1. ABOUT THE DATASET

Source: Kaggle

License: CC0: Public Domain

This dataset is a combination of the following three datasets :
figshare
SARTAJ dataset
Br35H

This dataset contains 7023 images of human brain MRI images which are classified into 4 classes: glioma - meningioma - no tumor and pituitary.

6.2.2. EXPLORATORY DATA ANALYSIS

EDA (Exploratory Data Analysis) is an essential step in any machine learning project, including brain tumor classification. EDA involves exploring and visualizing the data to gain insights and identify patterns that can inform the model building process. Since it is an Images dataset there is no significant information obtained from EDA.

6.2.3. PRE-PROCESSING

Preprocessing is a critical step in any deep learning project, including classification tasks such as brain tumor classification.

Train-test split involves dividing the available data into 3 subsets: a training set, validation set and a testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the trained model. This technique is essential for preventing overfitting and ensuring that the model generalizes well to new data.

Then all the images are resized to 299*299 and rescaled to ensure standardization.

6.2.4. MODELS

After having performed data preprocessing, we apply pretrained CNN models, namely

<u>Resnet50</u>

- Depth and Skip Connections: ResNet-50 is a deep neural network with 50 layers, featuring skip connections that aid in training very deep networks effectively.
- Bottleneck Blocks: It employs bottleneck blocks to reduce computational complexity while maintaining model expressiveness.
- Pretraining and Transfer Learning: ResNet-50 models are often pretrained on ImageNet, facilitating transfer learning for various computer vision tasks with limited data.

<u>Xception</u>

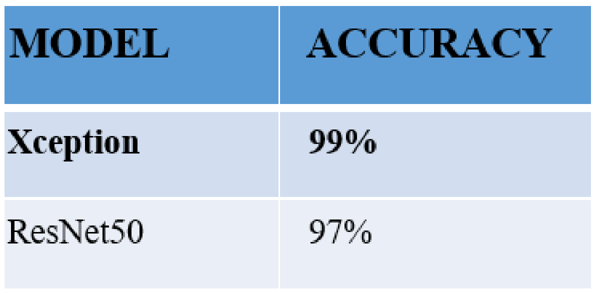
- Depth and Efficiency: Xception is a deep neural network with 71 layers, utilizing depthwise separable convolutions for efficiency.
- Pretrained and Versatile: It's often pretrained on datasets like ImageNet, making it adaptable for transfer learning across different visual tasks.
- Feature Hierarchies and Performance: Xception excels in capturing complex feature hierarchies, leading to high accuracy in image recognition tasks.

6.2.5. EVALUATION

1.Accuracy: Accuracy is the most commonly used evaluation metric for classification tasks. It measures the proportion of correct predictions made by the model.

2. Precision and recall: These metrics are important for evaluating the performance of models in cases where the data is imbalanced. Precision measures the proportion of true positives among all positive predictions, while recall measures the proportion of true positives among all actual positive cases.

3. F1 score: It provides a balanced measure of the model's performance and is particularly useful for evaluating models in cases where both precision and recall are important.



MODEL	**ACCURACY**
Xception	**99%**
ResNet50	97%

Table no 7.1 Models

7. RESULTS AND DISCUSSIONS

For the image dataset, when classifying the MRI scans into 4 classes, the best results are found to be from Xception. Xception often surpasses ResNet-50 due to its efficient depthwise separable convolutions, allowing it to capture intricate features and offer superior transfer learning capabilities when pretrained on datasets like ImageNet. Its increased capacity and effective regularization techniques contribute to its potential for higher accuracy in various tasks.

MODEL	**ACCURACY**
Xception	**99%**
ResNet50	97%

Table no 7.2 Models

8.CONCLUSION

- In conclusion, the classification of Brain Tumor using MRI scans shows promising results. Both modalities provide complementary information for the accurate detection and diagnosis of different types of dementia.
- All 3 types of brain tumors are classified accurately and diagnosed from MRI images. The use of technology in the medical domain is needed because of the time limit, the level of accuracy assures trustworthiness.
- The lack of brain MRI images makes it difficult to classify brain tumors using image-based classification. The solution to this challenge was achieved through the application CNN
- However, further research is needed to improve the specificity and sensitivity of these classification methods, as well as to evaluate their practicality and applicability in clinical settings.

9. FUTURE ENHANCEMENT

- Data Augmentation and Synthesis: Explore advanced data augmentation techniques and synthetic data generation to increase dataset diversity and address data scarcity.
- Model Architecture Optimization: Experiment with optimized CNN architectures, considering deeper models, attention mechanisms, or network pruning techniques to improve classification performance and efficiency.
- Transfer Learning and Fine-Tuning: Investigate transfer learning from pre-trained models on larger datasets like BraTS, followed by fine-tuning on domain-specific datasets to enhance classification accuracy.
- Ensemble Learning: Utilize ensemble learning methods to combine predictions from multiple CNN models or architectures, improving robustness and generalization.

10.APPENDICIES

10.1 FULL CODE

For ResNET model:

```
from keras.callbacks import
EarlyStopping,ReduceLROnPlateau,ModelCheckpoint,TensorBoard,LambdaCallback
from keras.layers import Input,Dropout, Dense,GlobalAveragePooling2D
from keras.models import Sequential,Model
from keras.applications.resnet import ResNet50
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
from tensorflow import keras
import tensorflow as tf
from tqdm import tqdm
import seaborn as sns
import numpy as np
import itertools
import datetime
import cv2
import os
import io
import tensorflow as tf
labels = ['glioma', 'meningioma', 'notumor', 'pituitary']

x_train = [] # training images.
y_train  = [] # training labels.
x_test = [] # testing images.
y_test = [] # testing labels.

image_size = 200


for label in labels:
    trainPath = os.path.join('C:\\Users\\lalit\\Downloads\\jcomp\\Training',
label)
    for file in tqdm(os.listdir(trainPath)):
        image = cv2.imread(os.path.join(trainPath, file),0) # load images in
gray.
        image = cv2.bilateralFilter(image, 2, 50, 50) # remove images noise.
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE) # produce a
pseudocolored image.
        image = cv2.resize(image, (image_size, image_size)) # resize images
into 150*150.
        x_train.append(image)
        y_train.append(labels.index(label))

    testPath = os.path.join('C:\\Users\\lalit\\Downloads\\jcomp\\Testing',
label)
    for file in tqdm(os.listdir(testPath)):
        image = cv2.imread(os.path.join(testPath, file),0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
        image = cv2.resize(image, (image_size, image_size))
        x_test.append(image)
        y_test.append(labels.index(label))


x_train = np.array(x_train) / 255.0 # normalize Images into range 0 to 1.
x_test = np.array(x_test) / 255.0

print(x_train.shape)
print(x_test.shape)
images = [x_train[i] for i in range(15)]
fig, axes = plt.subplots(3, 5, figsize = (10, 10))
axes = axes.flatten()
for img, ax in zip(images, axes):
    ax.imshow(img)
plt.tight_layout()
plt.show()

x_train, y_train = shuffle(x_train,y_train, random_state=42)

y_train = tf.keras.utils.to_categorical(y_train) #One Hot Encoding on the
labels
y_test = tf.keras.utils.to_categorical(y_test)

x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=42) #Dividing the dataset into Training and
Validation sets.

print(x_val.shape)
# ImageDataGenerator transforms each image in the batch by a series of
random translations, rotations, etc.
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.05,
    height_shift_range=0.05,
    horizontal_flip=True)

# After you have created and configured your ImageDataGenerator, you must
fit it on your data.
datagen.fit(x_train)
```

```
net = ResNet50(
    weights='imagenet', # Load weights pre-trained on ImageNet.
     include_top=False, # Do not include the ImageNet classifier at the top.
     input_shape=(image_size,image_size,3))
model = net.output
model = GlobalAveragePooling2D()(model)
model = Dropout(0.4)(model)
model = Dense(4, activation="softmax")(model)
model = Model(inputs= net.input, outputs= model)

#compile our model.
adam = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=adam, loss = 'categorical_crossentropy',
metrics=['accuracy'])
model.summary()
from sklearn.metrics import confusion_matrix
import numpy as np

BATCH_SIZE = 8
EPOCHS = 3

Checkpoint = ModelCheckpoint(filepath='model-{epoch:02d}-{val_accuracy:.2f}-
{val_loss:.2f}.h5', monitor='val_loss', verbose=1, save_best_only=True,
mode='min')

ES = EarlyStopping(monitor='val_loss', min_delta=0.001, patience=5,
mode='min', restore_best_weights=True, verbose=1)

RL = ReduceLROnPlateau(monitor='val_loss', factor=0.3, patience=5,
verbose=1, mode='min')

callbacks = [ES, RL, Checkpoint]

def log_confusion_matrix(epoch, logs):
    # Get predictions on validation set
    y_pred = model.predict(x_val)
    # Convert predictions to classes
    y_pred_classes = np.argmax(y_pred, axis=1)
    # Convert true labels to classes
    y_true_classes = np.argmax(y_val, axis=1)
    # Calculate confusion matrix
    cm = confusion_matrix(y_true_classes, y_pred_classes)
    # Log confusion matrix
    print("Confusion Matrix:")
    print(cm)

callbacks.append(LambdaCallback(on_epoch_end=log_confusion_matrix))
```

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=8),
validation_data=(x_val, y_val), epochs=EPOCHS, callbacks=callbacks)
predicted_classes = np.argmax(model.predict(x_test), axis = 1)
confusionmatrix = confusion_matrix(np.argmax(y_test,axis=1),
predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar = True)
print(classification_report(np.argmax(y_test,axis=1), predicted_classes))
loss,acc = model.evaluate(x_test,y_test)
```

For Xception model:

```
from google.colab import drive
drive.mount('/content/drive')
import os
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from glob import glob
#----------------------------------------
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
#----------------------------------------
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.optimizers import Adamax
from tensorflow.keras.metrics import Precision, Recall
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#----------------------------------------
import warnings
warnings.filterwarnings("ignore")
def train_df(tr_path):
    classes, class_paths = zip(*[(label, os.path.join(tr_path, label,
image))
                                 for label in os.listdir(tr_path) if
os.path.isdir(os.path.join(tr_path, label))
                                 for image in
os.listdir(os.path.join(tr_path, label))])

    tr_df = pd.DataFrame({'Class Path': class_paths, 'Class':
classes})
    return tr_df
def test_df(ts_path):
    classes, class_paths = zip(*[(label, os.path.join(ts_path, label, 
image))
                                 for label in os.listdir(ts_path) if 
os.path.isdir(os.path.join(ts_path, label))
                                 for image in 
os.listdir(os.path.join(ts_path, label))])

    ts_df = pd.DataFrame({'Class Path': class_paths, 'Class': 
classes})
    return ts_df
tr_df = train_df('/content/drive/MyDrive/jcomp_brain/jcomp/Training')

ts_df = train_df('/content/drive/MyDrive/jcomp_brain/jcomp/Testing')
import matplotlib.pyplot as plt
import seaborn as sns

# Create a color palette with different colors for each class
class_colors = sns.color_palette("husl", 
len(tr_df['Class'].unique()))

plt.figure(figsize=(10, 7))
ax = sns.countplot(data=tr_df, x='Class', palette=class_colors)

plt.xlabel('Class', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Count of images in each class', fontsize=16)

# Add labels to the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., 
p.get_height()), ha='center', va='center', xytext=(0, 5), 
textcoords='offset points')

plt.xticks(rotation=45, ha='right')  # Rotate x labels for better 
visibility
plt.tight_layout()
plt.show()
import matplotlib.pyplot as plt
import seaborn as sns

# Create a color palette with different colors for each class
class_colors = sns.color_palette("husl", 
len(ts_df['Class'].unique()))

plt.figure(figsize=(10, 7))
ax = sns.countplot(data=ts_df, x='Class', palette=class_colors)

plt.xlabel('Class', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Count of images in each class (Test Set)', fontsize=16)

# Add labels to the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2.,
p.get_height()), ha='center', va='center', xytext=(0, 5),
textcoords='offset points')

plt.xticks(rotation=45, ha='right')  # Rotate x labels for better
visibility
plt.tight_layout()
plt.show()
valid_df, ts_df = train_test_split(ts_df, train_size=0.5,
random_state=20, stratify=ts_df['Class'])
batch_size = 32
img_size = (299, 299)

_gen = ImageDataGenerator(rescale=1/255,
                          brightness_range=(0.8, 1.2))

ts_gen = ImageDataGenerator(rescale=1/255)


tr_gen = _gen.flow_from_dataframe(tr_df, x_col='Class Path',
                                  y_col='Class',
batch_size=batch_size,
                                  target_size=img_size)

valid_gen = _gen.flow_from_dataframe(valid_df, x_col='Class Path',
                                     y_col='Class',
batch_size=batch_size,
                                     target_size=img_size)

ts_gen = ts_gen.flow_from_dataframe(ts_df, x_col='Class Path',
                                  y_col='Class', batch_size=16,
                                  target_size=img_size,
shuffle=False)
class_dict = tr_gen.class_indices
classes = list(class_dict.keys())
images, labels = next(ts_gen)

plt.figure(figsize=(20, 20))

for i, (image, label) in enumerate(zip(images, labels)):
    plt.subplot(4,4, i + 1)
    plt.imshow(image)
    class_name = classes[np.argmax(label)]
    plt.title(class_name, color='k', fontsize=15)
plt.show()
img_shape=(299,299,3)
base_model = tf.keras.applications.Xception(include_top= False,
weights= "imagenet",
                            input_shape= img_shape, pooling= 'max')

# for layer in base_model.layers:
#     layer.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dropout(rate= 0.3),
    Dense(128, activation= 'relu'),
    Dropout(rate= 0.25),
    Dense(4, activation= 'softmax')
])

model.compile(Adamax(learning_rate= 0.001),
              loss= 'categorical_crossentropy',
              metrics= ['accuracy',
                        Precision(),
                        Recall()])

model.summary()
tf.keras.utils.plot_model(model, show_shapes=True)
hist = model.fit(tr_gen,
                 epochs=10,
                 validation_data=valid_gen,
                 shuffle= False)
hist.history.keys()
tr_acc = hist.history['accuracy']
tr_loss = hist.history['loss']
tr_per = hist.history['precision']
tr_recall = hist.history['recall']
val_acc = hist.history['val_accuracy']
val_loss = hist.history['val_loss']
val_per = hist.history['val_precision']
val_recall = hist.history['val_recall']

index_loss = np.argmin(val_loss)
val_lowest = val_loss[index_loss]
index_acc = np.argmax(val_acc)
acc_highest = val_acc[index_acc]
index_precision = np.argmax(val_per)
per_highest = val_per[index_precision]
index_recall = np.argmax(val_recall)
recall_highest = val_recall[index_recall]
```

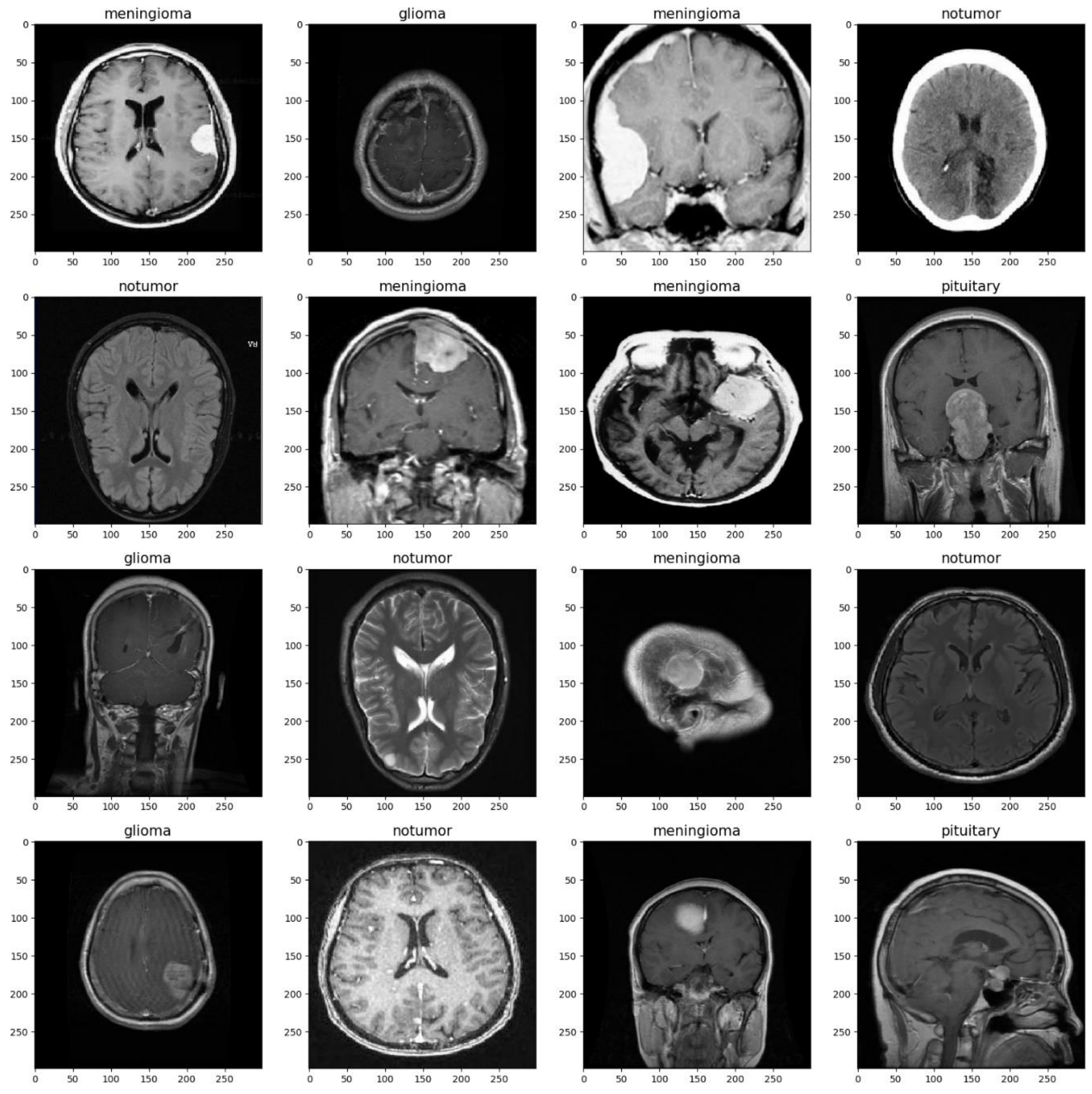
```
Epochs = [i + 1 for i in range(len(tr_acc))]
loss_label = f'Best epoch = {str(index_loss + 1)}'
acc_label = f'Best epoch = {str(index_acc + 1)}'
per_label = f'Best epoch = {str(index_precision + 1)}'
recall_label = f'Best epoch = {str(index_recall + 1)}'


plt.figure(figsize=(20, 12))
plt.style.use('fivethirtyeight')


plt.subplot(2, 2, 1)
plt.plot(Epochs, tr_loss, 'r', label='Training loss')
plt.plot(Epochs, val_loss, 'g', label='Validation loss')
plt.scatter(index_loss + 1, val_lowest, s=150, c='blue',
label=loss_label)
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(Epochs, tr_acc, 'r', label='Training Accuracy')
plt.plot(Epochs, val_acc, 'g', label='Validation Accuracy')
plt.scatter(index_acc + 1, acc_highest, s=150, c='blue',
label=acc_label)
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 3)
plt.plot(Epochs, tr_per, 'r', label='Precision')
plt.plot(Epochs, val_per, 'g', label='Validation Precision')
plt.scatter(index_precision + 1, per_highest, s=150, c='blue',
label=per_label)
plt.title('Precision and Validation Precision')
plt.xlabel('Epochs')
plt.ylabel('Precision')
plt.legend()
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(Epochs, tr_recall, 'r', label='Recall')
plt.plot(Epochs, val_recall, 'g', label='Validation Recall')
plt.scatter(index_recall + 1, recall_highest, s=150, c='blue',
label=recall_label)
```

```python
plt.title('Recall and Validation Recall')
plt.xlabel('Epochs')
plt.ylabel('Recall')
plt.legend()
plt.grid(True)

plt.suptitle('Model Training Metrics Over Epochs', fontsize=16)
plt.show()
train_score = model.evaluate(tr_gen, verbose=1)
valid_score = model.evaluate(valid_gen, verbose=1)
test_score = model.evaluate(ts_gen, verbose=1)

print(f"Train Loss: {train_score[0]:.4f}")
print(f"Train Accuracy: {train_score[1]*100:.2f}%")
print('-' * 20)
print(f"Validation Loss: {valid_score[0]:.4f}")
print(f"Validation Accuracy: {valid_score[1]*100:.2f}%")
print('-' * 20)
print(f"Test Loss: {test_score[0]:.4f}")
print(f"Test Accuracy: {test_score[1]*100:.2f}%")
preds = model.predict(ts_gen)
y_pred = np.argmax(preds, axis=1)
cm = confusion_matrix(ts_gen.classes, y_pred)
labels = list(class_dict.keys())
plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted Label')
plt.ylabel('Truth Label')
plt.show()
clr = classification_report(ts_gen.classes, y_pred)
print(clr)
def predict(img_path):
    import numpy as np
    import matplotlib.pyplot as plt
    from PIL import Image
    label = list(class_dict.keys())
    plt.figure(figsize=(12, 12))
    img = Image.open(img_path)
    resized_img = img.resize((299, 299))
    img = np.asarray(resized_img)
    img = np.expand_dims(img, axis=0)
    img = img / 255
    predictions = model.predict(img)
    probs = list(predictions[0])
    labels = label
    plt.subplot(2, 1, 1)
    plt.imshow(resized_img)
    plt.subplot(2, 1, 2)
    bars = plt.barh(labels, probs)
    plt.xlabel('Probability', fontsize=15)
    ax = plt.gca()
    ax.bar_label(bars, fmt = '%.2f')
    plt.show()
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/meningioma/
Te-meTr_0000.jpg')
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/glioma/Te-
glTr_0007.jpg')
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/notumor/Te-
noTr_0001.jpg')
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/pituitary/T
e-piTr_0001.jpg')
```

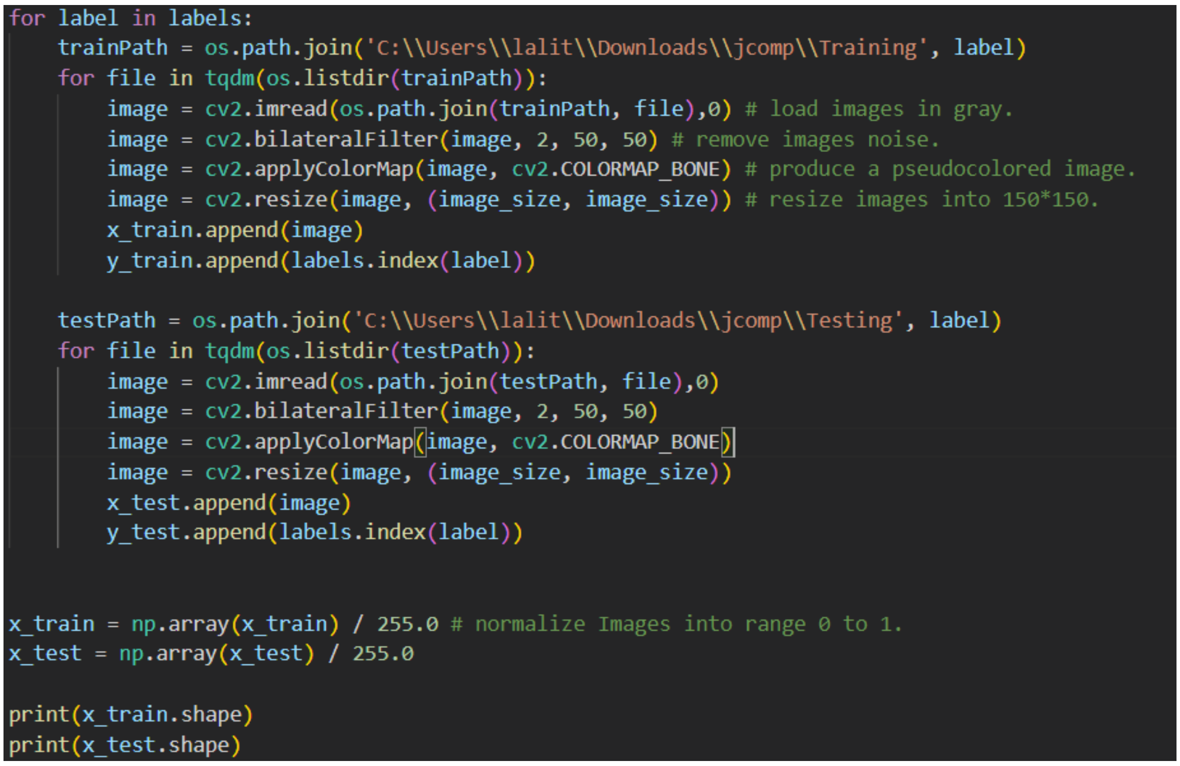
SCREENSHOTS:

For ResNET model:

DataSet:

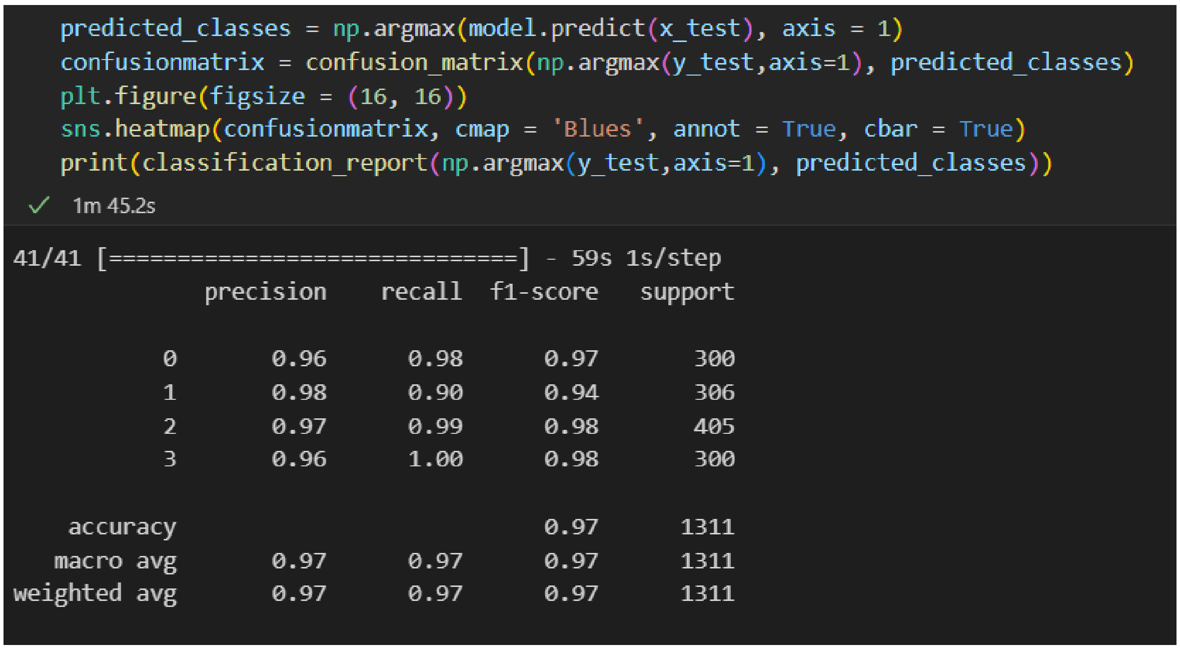


Pre-Processing:



```
for label in labels:
    trainPath = os.path.join('C:\\Users\\lalit\\Downloads\\jcomp\\Training', label)
    for file in tqdm(os.listdir(trainPath)):
        image = cv2.imread(os.path.join(trainPath, file),0) # load images in gray.
        image = cv2.bilateralFilter(image, 2, 50, 50) # remove images noise.
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE) # produce a pseudocolored image.
        image = cv2.resize(image, (image_size, image_size)) # resize images into 150*150.
        x_train.append(image)
        y_train.append(labels.index(label))

    testPath = os.path.join('C:\\Users\\lalit\\Downloads\\jcomp\\Testing', label)
    for file in tqdm(os.listdir(testPath)):
        image = cv2.imread(os.path.join(testPath, file),0)
        image = cv2.bilateralFilter(image, 2, 50, 50)
        image = cv2.applyColorMap(image, cv2.COLORMAP_BONE)
        image = cv2.resize(image, (image_size, image_size))
        x_test.append(image)
        y_test.append(labels.index(label))


x_train = np.array(x_train) / 255.0 # normalize Images into range 0 to 1.
x_test = np.array(x_test) / 255.0

print(x_train.shape)
print(x_test.shape)
```

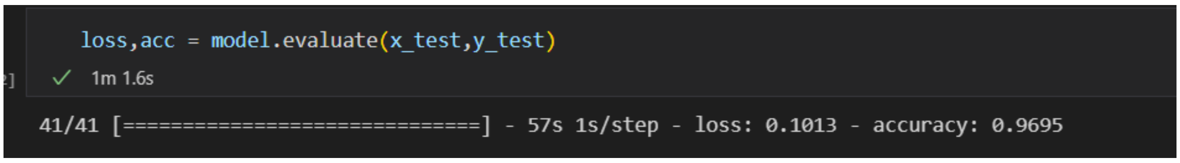
Performance Analysis for ResNET



```
predicted_classes = np.argmax(model.predict(x_test), axis = 1)
confusionmatrix = confusion_matrix(np.argmax(y_test,axis=1), predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar = True)
print(classification_report(np.argmax(y_test,axis=1), predicted_classes))
```

✓ 1m 45.2s

```
41/41 [==============================] - 59s 1s/step
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       300
           1       0.98      0.90      0.94       306
           2       0.97      0.99      0.98       405
           3       0.96      1.00      0.98       300

    accuracy                           0.97      1311
   macro avg       0.97      0.97      0.97      1311
weighted avg       0.97      0.97      0.97      1311
```

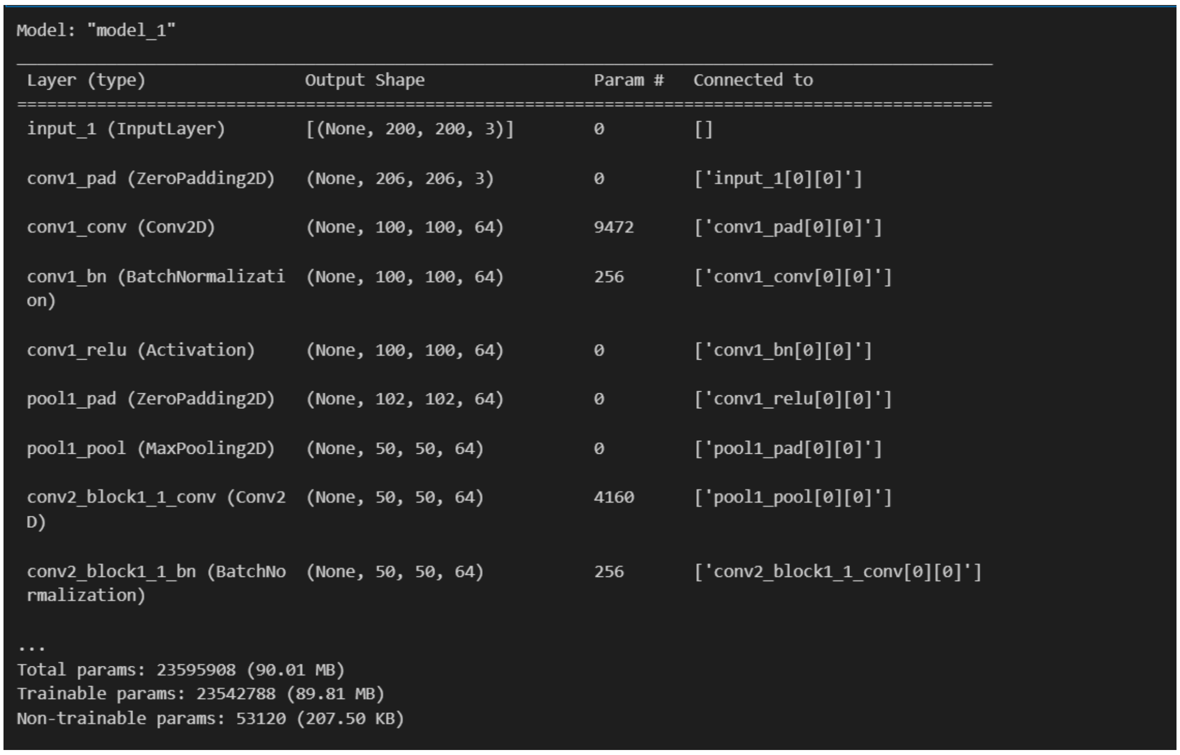


```
loss,acc = model.evaluate(x_test,y_test)
```

✓ 1m 1.6s

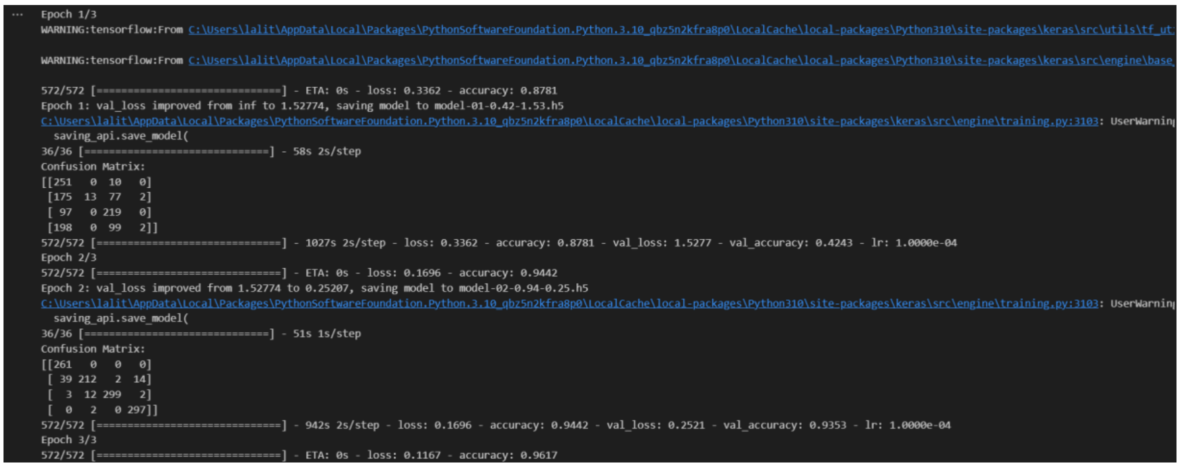
```
41/41 [==============================] - 57s 1s/step - loss: 0.1013 - accuracy: 0.9695
```

Model Layers:

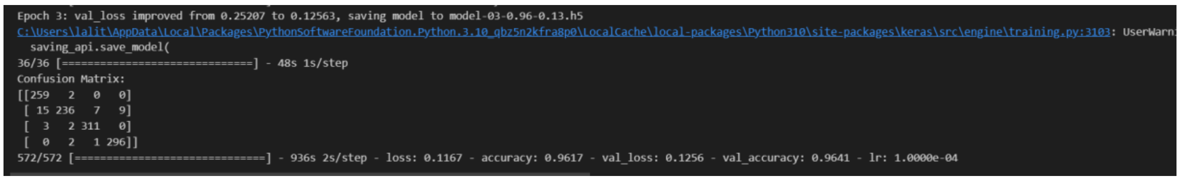


```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 200, 200, 3)]        0         []

 conv1_pad (ZeroPadding2D)   (None, 206, 206, 3)          0         ['input_1[0][0]']

 conv1_conv (Conv2D)         (None, 100, 100, 64)         9472      ['conv1_pad[0][0]']

 conv1_bn (BatchNormalizati  (None, 100, 100, 64)         256       ['conv1_conv[0][0]']
 on)

 conv1_relu (Activation)     (None, 100, 100, 64)         0         ['conv1_bn[0][0]']

 pool1_pad (ZeroPadding2D)   (None, 102, 102, 64)         0         ['conv1_relu[0][0]']

 pool1_pool (MaxPooling2D)   (None, 50, 50, 64)           0         ['pool1_pad[0][0]']

 conv2_block1_1_conv (Conv2  (None, 50, 50, 64)           4160      ['pool1_pool[0][0]']
 D)

 conv2_block1_1_bn (BatchNo  (None, 50, 50, 64)           256       ['conv2_block1_1_conv[0][0]']
 rmalization)

...
Total params: 23595908 (90.01 MB)
Trainable params: 23542788 (89.81 MB)
Non-trainable params: 53120 (207.50 KB)
```

Epochs 1,2&3

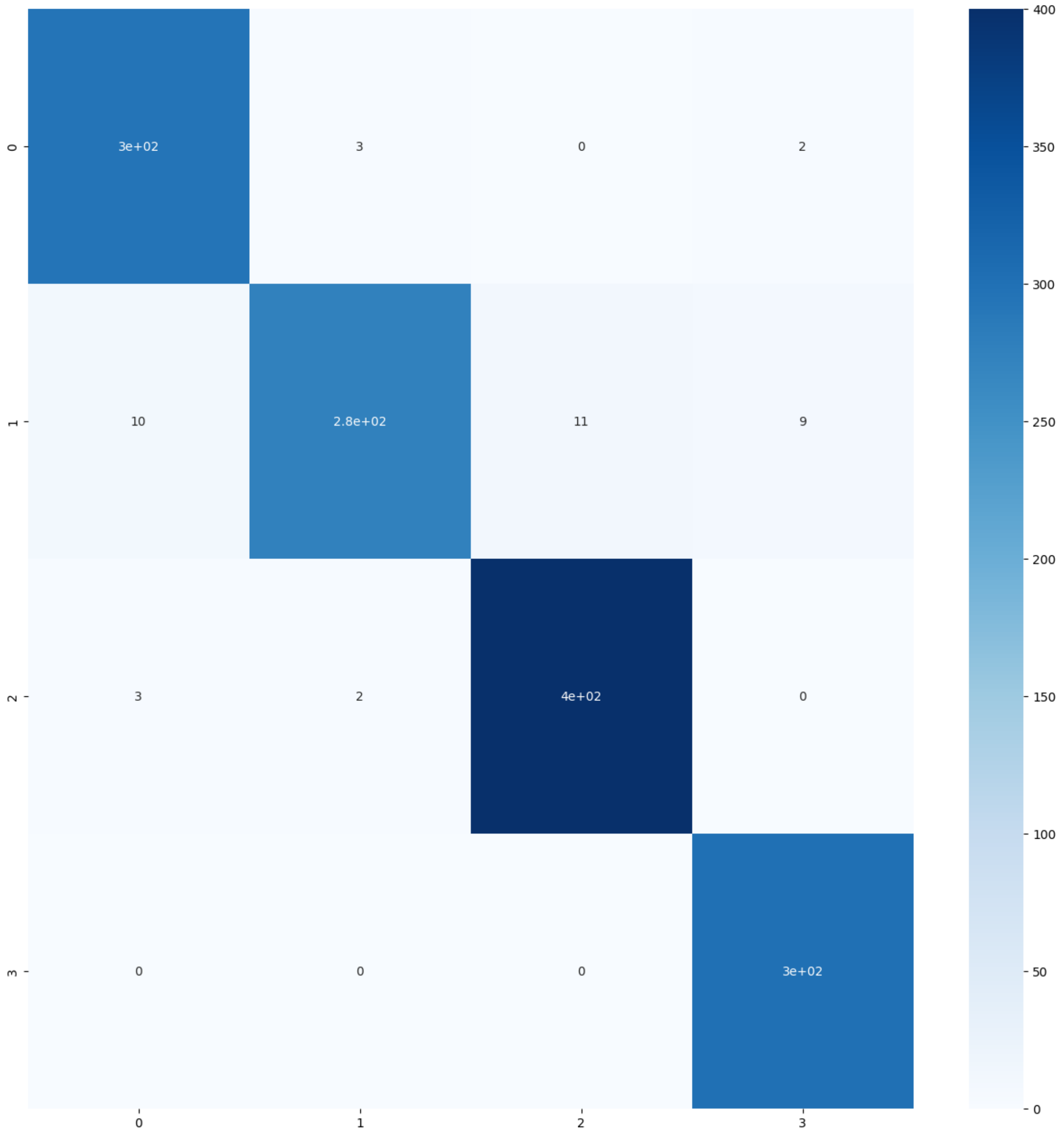


```
...  Epoch 1/3
     WARNING:tensorflow:From C:\Users\lalit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\utils\tf_ut

     WARNING:tensorflow:From C:\Users\lalit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\engine\base_

     572/572 [==============================] - ETA: 0s - loss: 0.3362 - accuracy: 0.8781
     Epoch 1: val_loss improved from inf to 1.52774, saving model to model-01-0.42-1.53.h5
     C:\Users\lalit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\engine\training.py:3103: UserWarning
       saving_api.save_model(
     36/36 [==============================] - 58s 2s/step
     Confusion Matrix:
     [[251   0  10   0]
      [175  13  77   2]
      [ 97   0 219   0]
      [198   0  99   2]]
     572/572 [==============================] - 1027s 2s/step - loss: 0.3362 - accuracy: 0.8781 - val_loss: 1.5277 - val_accuracy: 0.4243 - lr: 1.0000e-04
     Epoch 2/3
     572/572 [==============================] - ETA: 0s - loss: 0.1696 - accuracy: 0.9442
     Epoch 2: val_loss improved from 1.52774 to 0.25207, saving model to model-02-0.94-0.25.h5
     C:\Users\lalit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\engine\training.py:3103: UserWarning
       saving_api.save_model(
     36/36 [==============================] - 51s 1s/step
     Confusion Matrix:
     [[261   0   0   0]
      [ 39 212   2  14]
      [  3  12 299   2]
      [  0   2   0 297]]
     572/572 [==============================] - 942s 2s/step - loss: 0.1696 - accuracy: 0.9442 - val_loss: 0.2521 - val_accuracy: 0.9353 - lr: 1.0000e-04
     Epoch 3/3
     572/572 [==============================] - ETA: 0s - loss: 0.1167 - accuracy: 0.9617
```



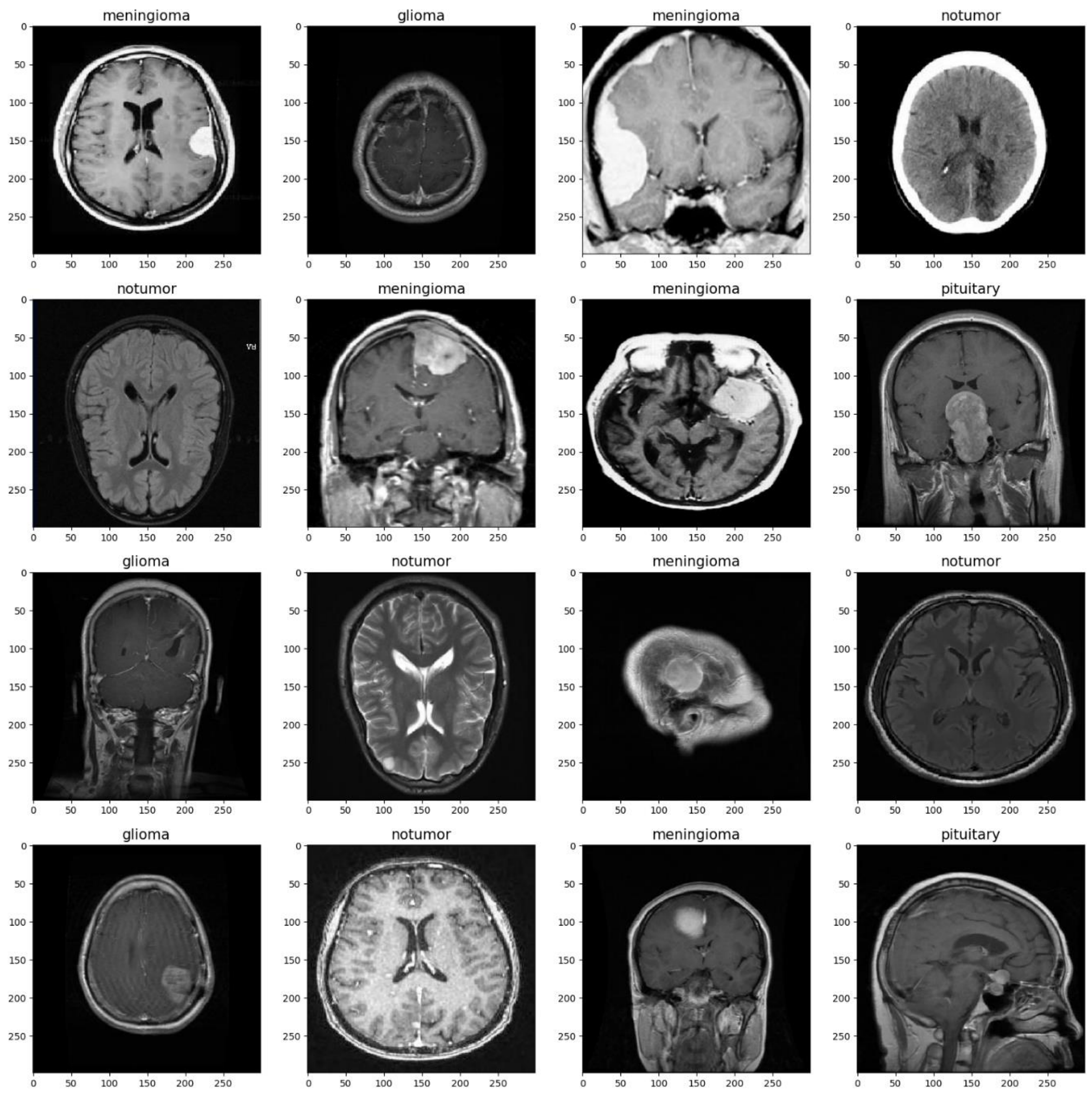
```
Epoch 3: val_loss improved from 0.25207 to 0.12563, saving model to model-03-0.96-0.13.h5
C:\Users\lalit\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\keras\src\engine\training.py:3103: UserWarn
  saving_api.save_model(
36/36 [==============================] - 48s 1s/step
Confusion Matrix:
[[259   2   0   0]
 [ 15 236   7   9]
 [  3   2 311   0]
 [  0   2   1 296]]
572/572 [==============================] - 936s 2s/step - loss: 0.1167 - accuracy: 0.9617 - val_loss: 0.1256 - val_accuracy: 0.9641 - lr: 1.0000e-04
```

Confusion Matrix:

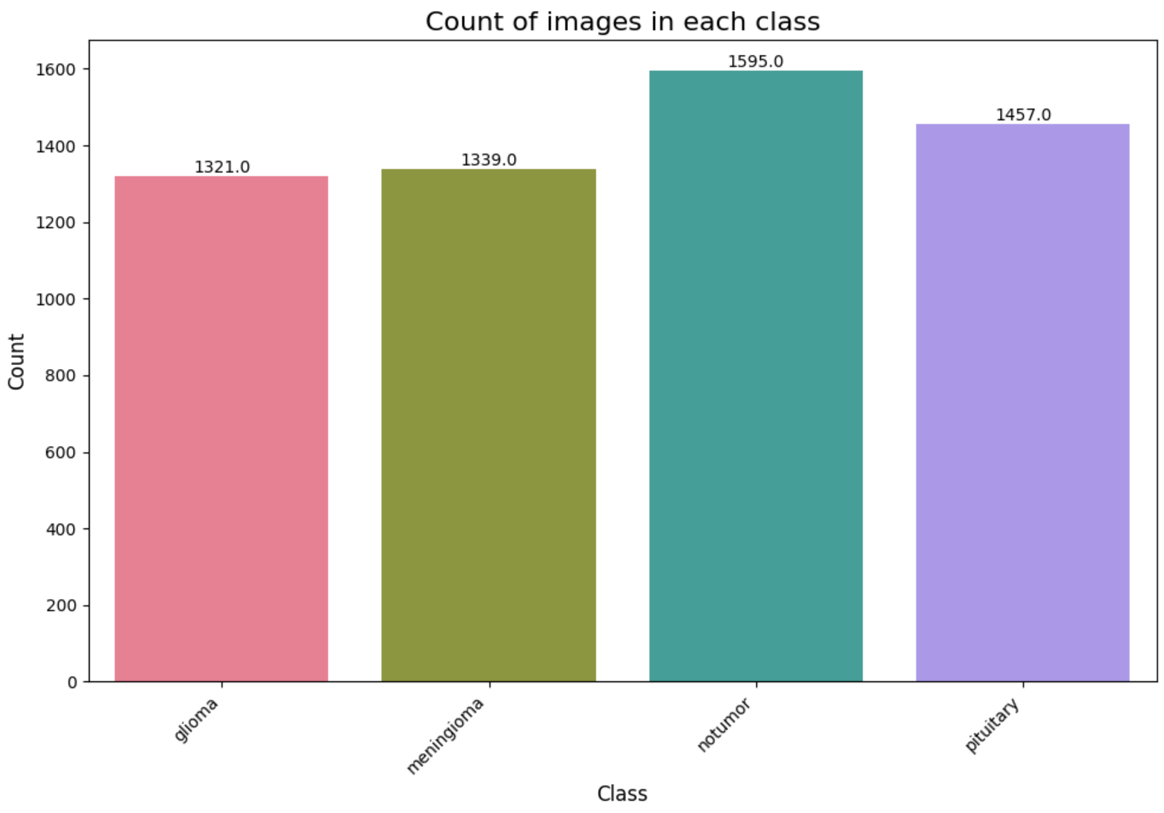


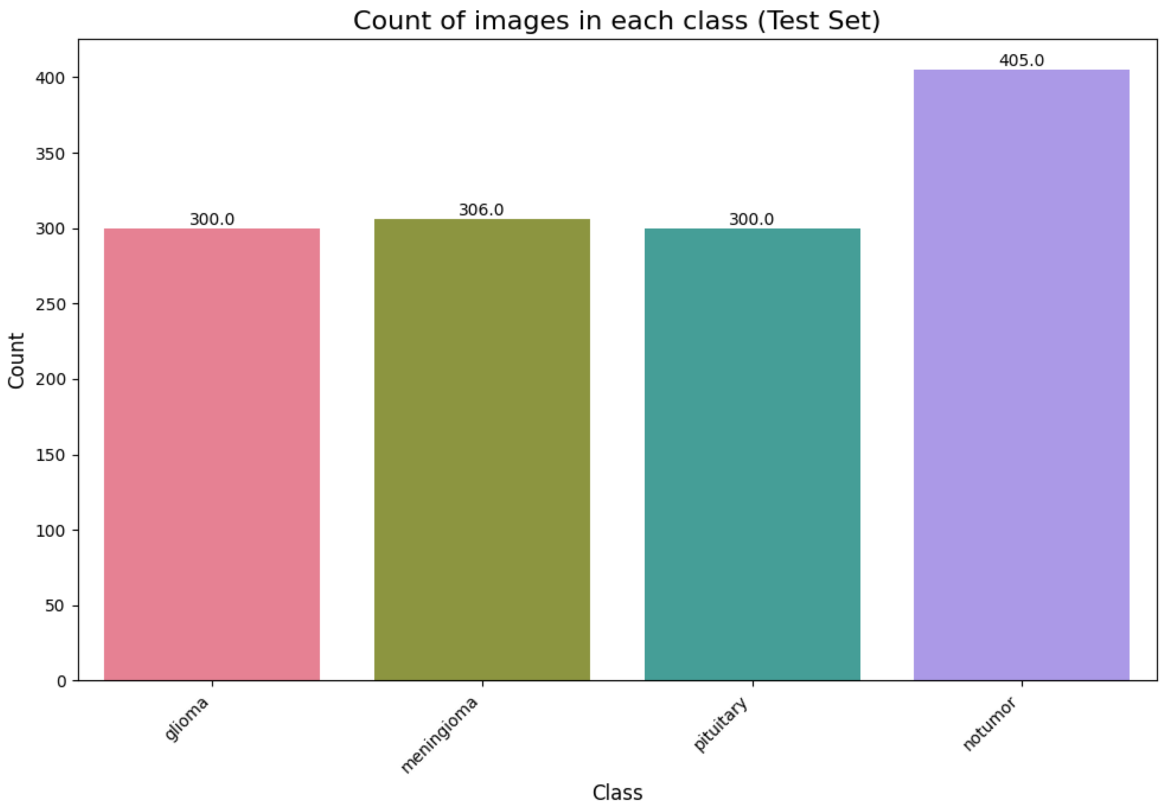
For Xception model:

DataSet:

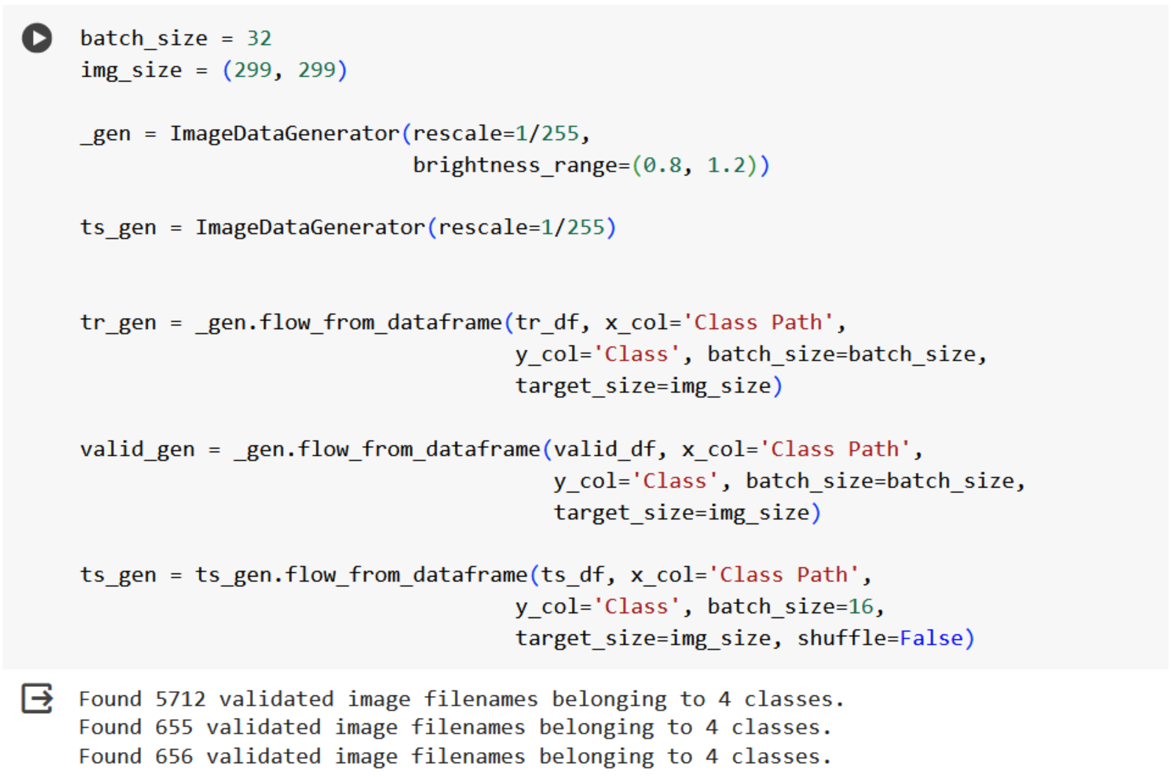


Distribution of data across 4 categories:





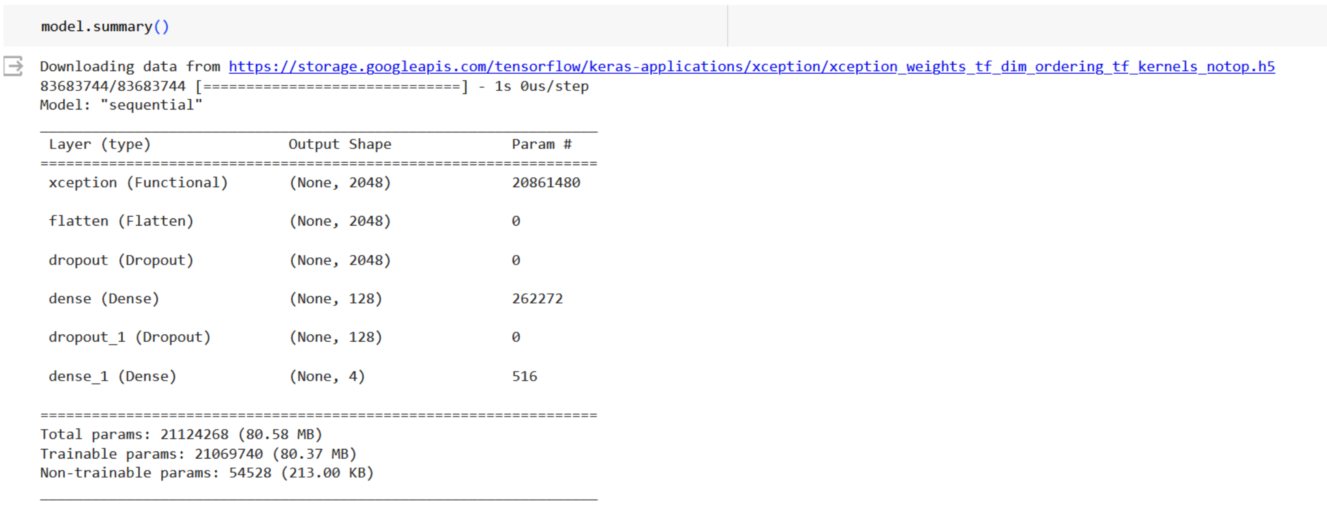
Pre-Processing:



```
batch_size = 32
img_size = (299, 299)

_gen = ImageDataGenerator(rescale=1/255,
                          brightness_range=(0.8, 1.2))

ts_gen = ImageDataGenerator(rescale=1/255)


tr_gen = _gen.flow_from_dataframe(tr_df, x_col='Class Path',
                                  y_col='Class', batch_size=batch_size,
                                  target_size=img_size)

valid_gen = _gen.flow_from_dataframe(valid_df, x_col='Class Path',
                                     y_col='Class', batch_size=batch_size,
                                     target_size=img_size)

ts_gen = ts_gen.flow_from_dataframe(ts_df, x_col='Class Path',
                                  y_col='Class', batch_size=16,
                                  target_size=img_size, shuffle=False)
```

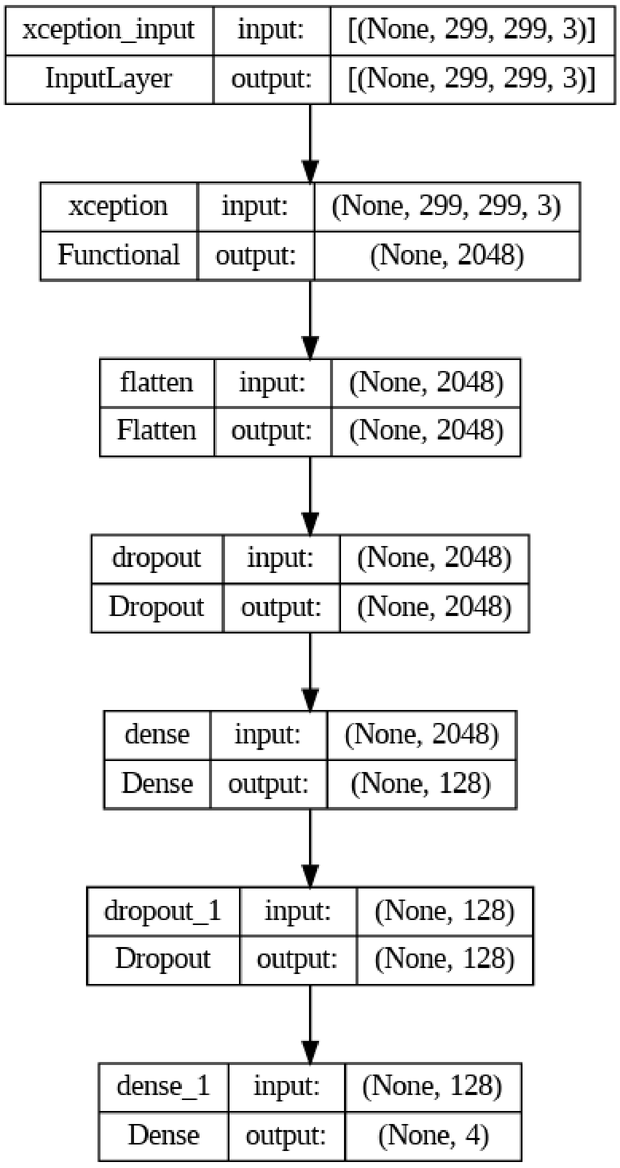
```
Found 5712 validated image filenames belonging to 4 classes.
Found 655 validated image filenames belonging to 4 classes.
Found 656 validated image filenames belonging to 4 classes.
```

Xception Model:

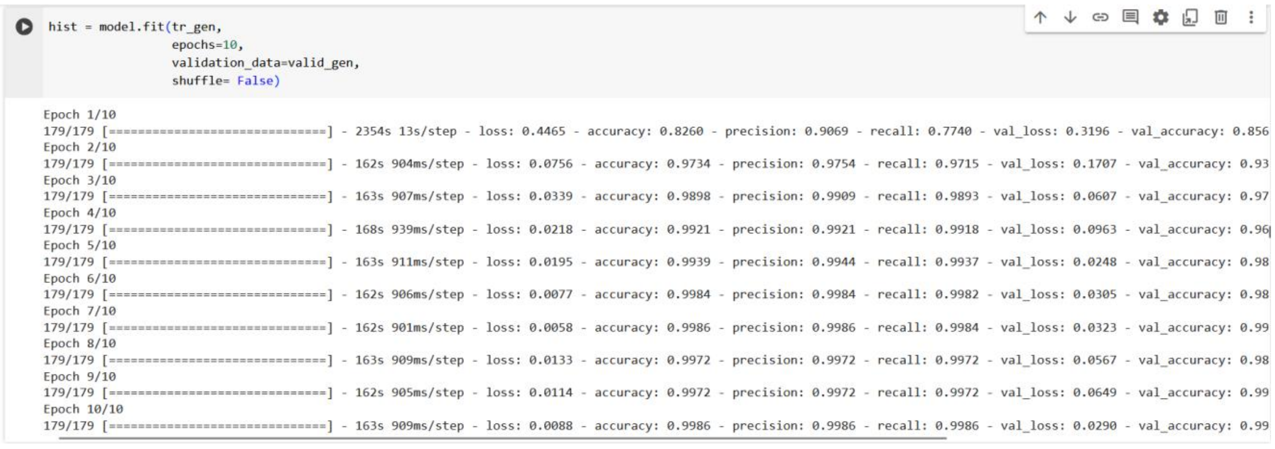


```
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 2048)              20861480

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 4)                 516

=================================================================
Total params: 21124268 (80.58 MB)
Trainable params: 21069740 (80.37 MB)
Non-trainable params: 54528 (213.00 KB)
_________________________________________________________________
```

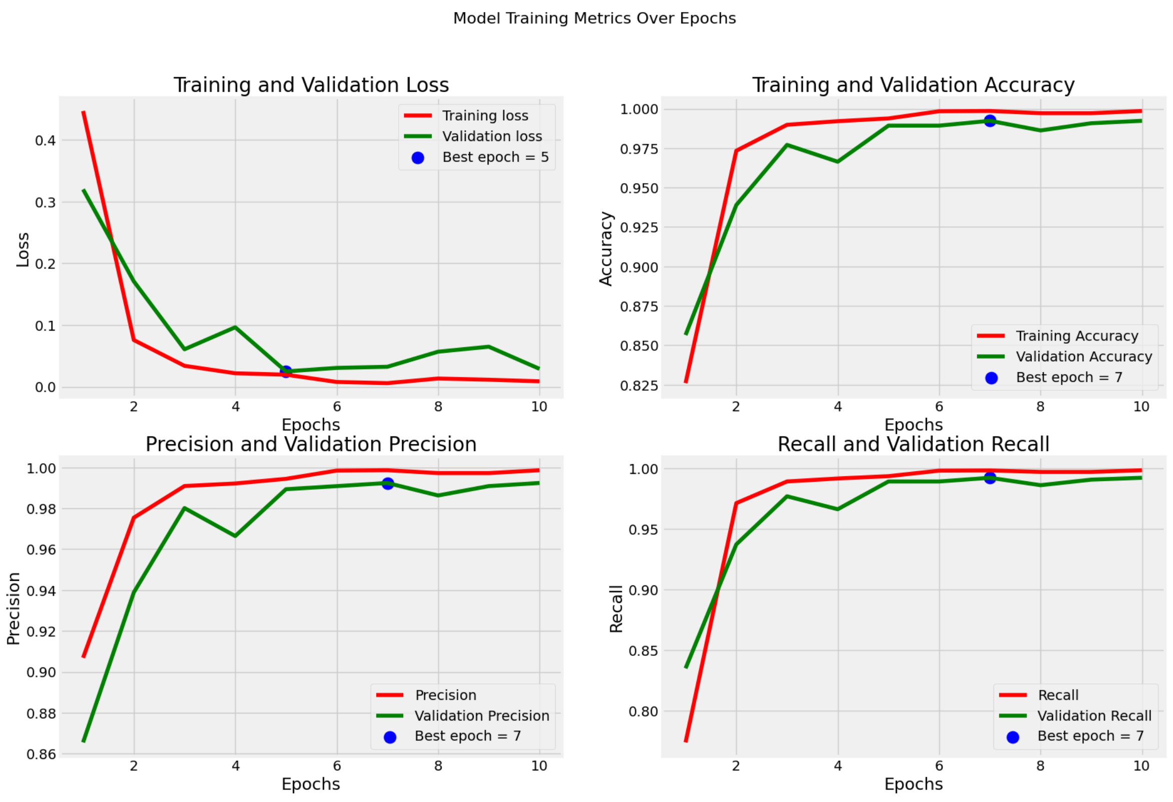

xception_input | input: | [(None, 299, 299, 3)]
InputLayer | output: | [(None, 299, 299, 3)]
xception | input: | (None, 299, 299, 3)
Functional | output: | (None, 2048)
flatten | input: | (None, 2048)
Flatten | output: | (None, 2048)
dropout | input: | (None, 2048)
Dropout | output: | (None, 2048)
dense | input: | (None, 2048)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 4)

Epochs:

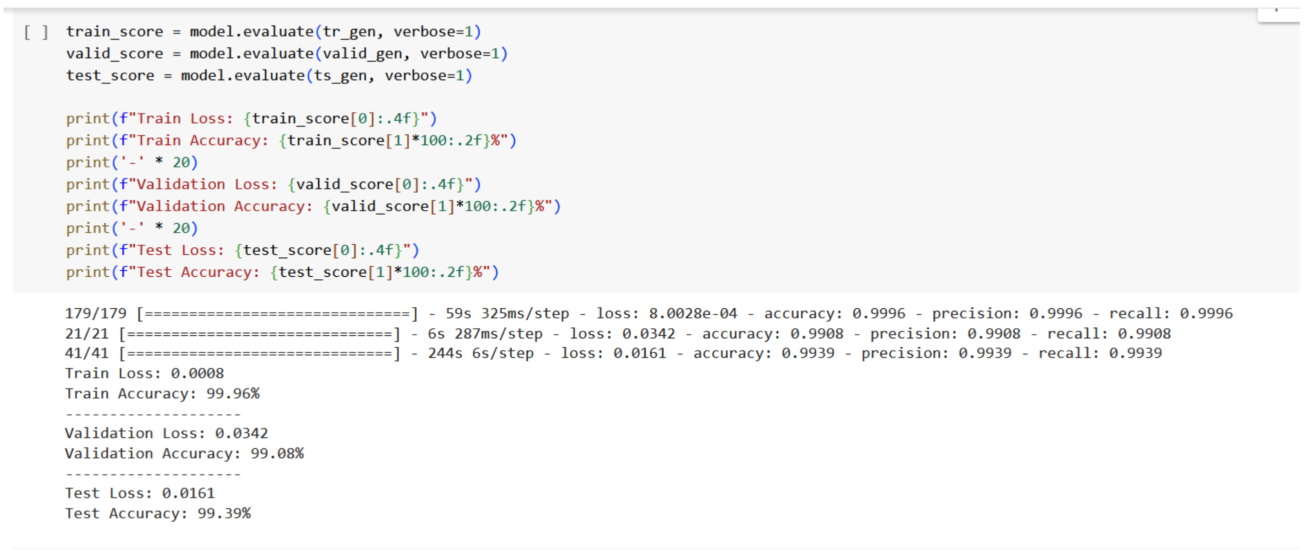

```
hist = model.fit(tr_gen,
                 epochs=10,
                 validation_data=valid_gen,
                 shuffle= False)
```

```
Epoch 1/10
179/179 [==============================] - 2354s 13s/step - loss: 0.4465 - accuracy: 0.8260 - precision: 0.9069 - recall: 0.7740 - val_loss: 0.3196 - val_accuracy: 0.856
Epoch 2/10
179/179 [==============================] - 162s 904ms/step - loss: 0.0756 - accuracy: 0.9734 - precision: 0.9754 - recall: 0.9715 - val_loss: 0.1707 - val_accuracy: 0.93
Epoch 3/10
179/179 [==============================] - 163s 907ms/step - loss: 0.0339 - accuracy: 0.9898 - precision: 0.9909 - recall: 0.9893 - val_loss: 0.0607 - val_accuracy: 0.97
Epoch 4/10
179/179 [==============================] - 168s 939ms/step - loss: 0.0218 - accuracy: 0.9921 - precision: 0.9921 - recall: 0.9918 - val_loss: 0.0963 - val_accuracy: 0.96
Epoch 5/10
179/179 [==============================] - 163s 911ms/step - loss: 0.0195 - accuracy: 0.9939 - precision: 0.9944 - recall: 0.9937 - val_loss: 0.0248 - val_accuracy: 0.98
Epoch 6/10
179/179 [==============================] - 162s 906ms/step - loss: 0.0077 - accuracy: 0.9984 - precision: 0.9984 - recall: 0.9982 - val_loss: 0.0305 - val_accuracy: 0.98
Epoch 7/10
179/179 [==============================] - 162s 901ms/step - loss: 0.0058 - accuracy: 0.9986 - precision: 0.9986 - recall: 0.9984 - val_loss: 0.0323 - val_accuracy: 0.99
Epoch 8/10
179/179 [==============================] - 163s 909ms/step - loss: 0.0133 - accuracy: 0.9972 - precision: 0.9972 - recall: 0.9972 - val_loss: 0.0567 - val_accuracy: 0.98
Epoch 9/10
179/179 [==============================] - 162s 905ms/step - loss: 0.0114 - accuracy: 0.9972 - precision: 0.9972 - recall: 0.9972 - val_loss: 0.0649 - val_accuracy: 0.99
Epoch 10/10
179/179 [==============================] - 163s 909ms/step - loss: 0.0088 - accuracy: 0.9986 - precision: 0.9986 - recall: 0.9986 - val_loss: 0.0290 - val_accuracy: 0.99
```

Model Metrics over Epochs:

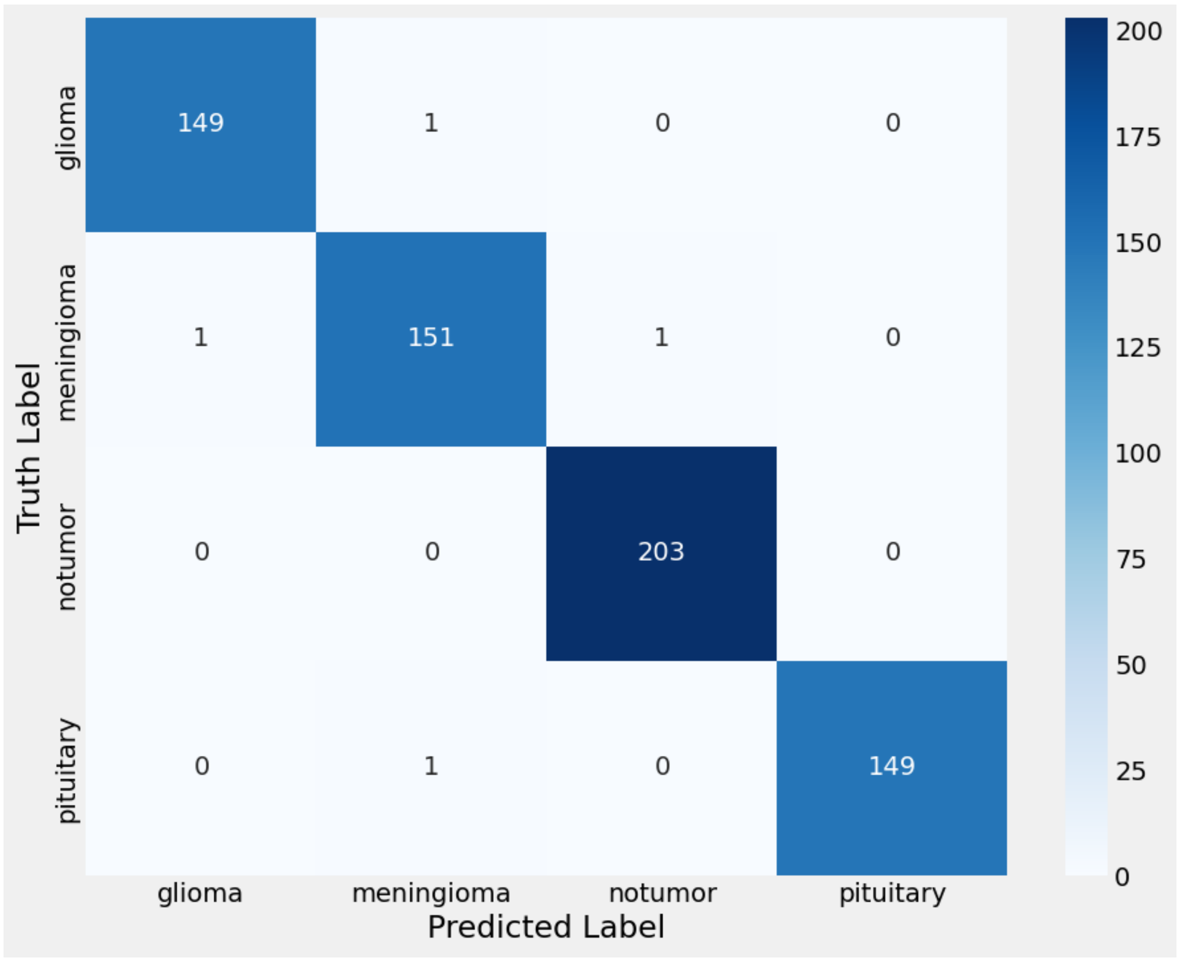


Metrics:



```
[ ] train_score = model.evaluate(tr_gen, verbose=1)
    valid_score = model.evaluate(valid_gen, verbose=1)
    test_score = model.evaluate(ts_gen, verbose=1)

    print(f"Train Loss: {train_score[0]:.4f}")
    print(f"Train Accuracy: {train_score[1]*100:.2f}%")
    print('-' * 20)
    print(f"Validation Loss: {valid_score[0]:.4f}")
    print(f"Validation Accuracy: {valid_score[1]*100:.2f}%")
    print('-' * 20)
    print(f"Test Loss: {test_score[0]:.4f}")
    print(f"Test Accuracy: {test_score[1]*100:.2f}%")
```

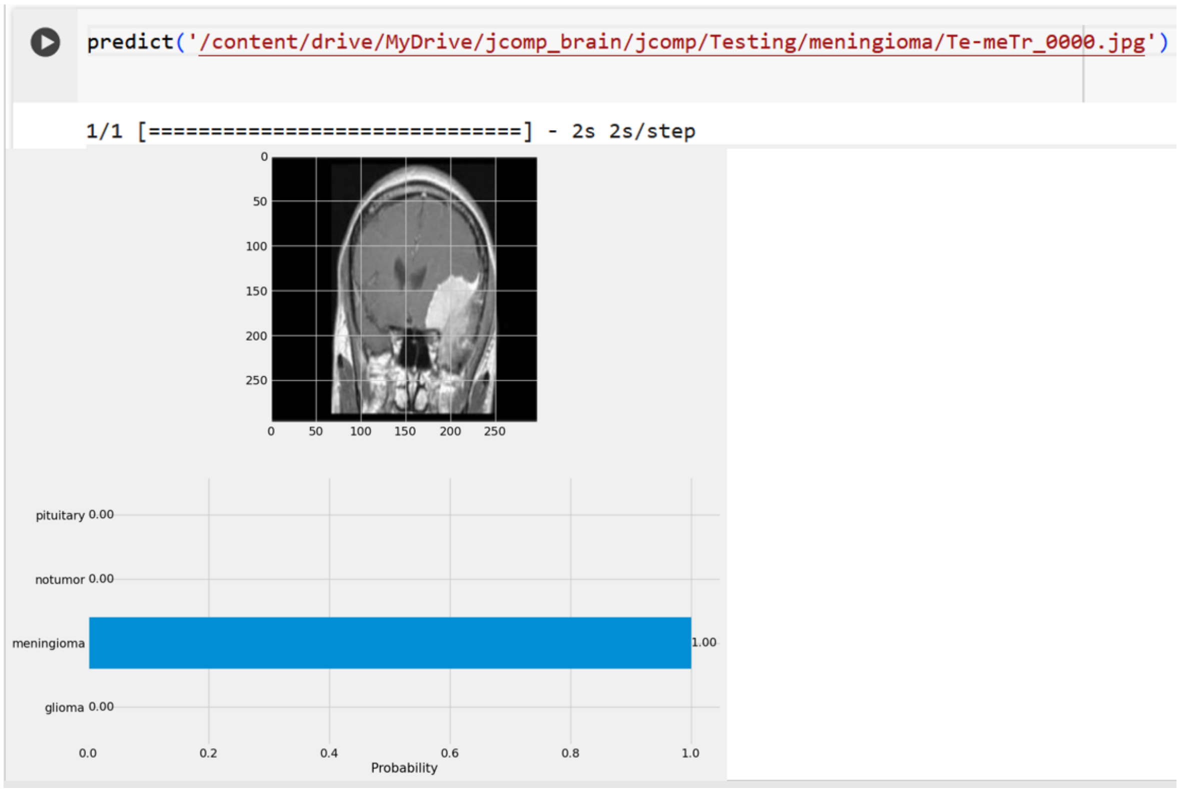
```
179/179 [==============================] - 59s 325ms/step - loss: 8.0028e-04 - accuracy: 0.9996 - precision: 0.9996 - recall: 0.9996
21/21 [==============================] - 6s 287ms/step - loss: 0.0342 - accuracy: 0.9908 - precision: 0.9908 - recall: 0.9908
41/41 [==============================] - 244s 6s/step - loss: 0.0161 - accuracy: 0.9939 - precision: 0.9939 - recall: 0.9939
Train Loss: 0.0008
Train Accuracy: 99.96%
--------------------
Validation Loss: 0.0342
Validation Accuracy: 99.08%
--------------------
Test Loss: 0.0161
Test Accuracy: 99.39%
```



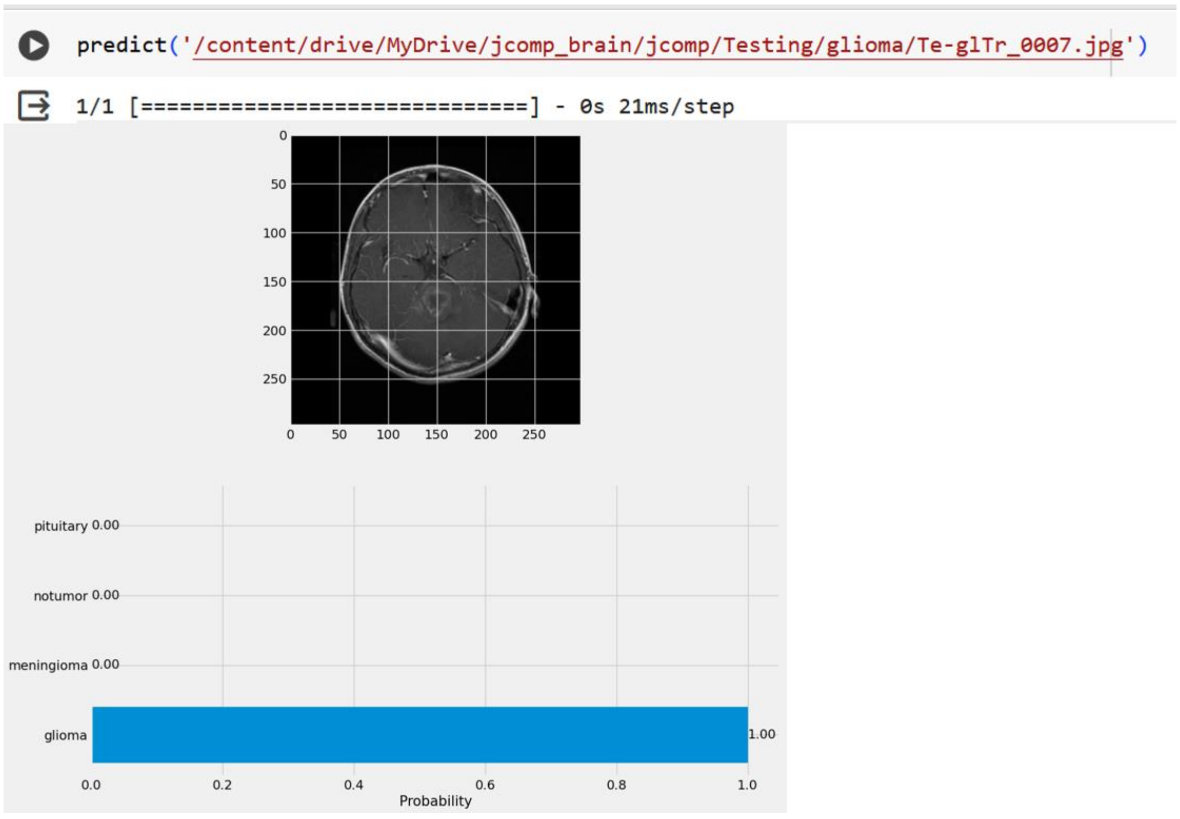
```
[ ]  clr = classification_report(ts_gen.classes, y_pred)
     print(clr)
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       150
           1       0.99      0.99      0.99       153
           2       1.00      1.00      1.00       203
           3       1.00      0.99      1.00       150

    accuracy                           0.99       656
   macro avg       0.99      0.99      0.99       656
weighted avg       0.99      0.99      0.99       656
```

Predicting:

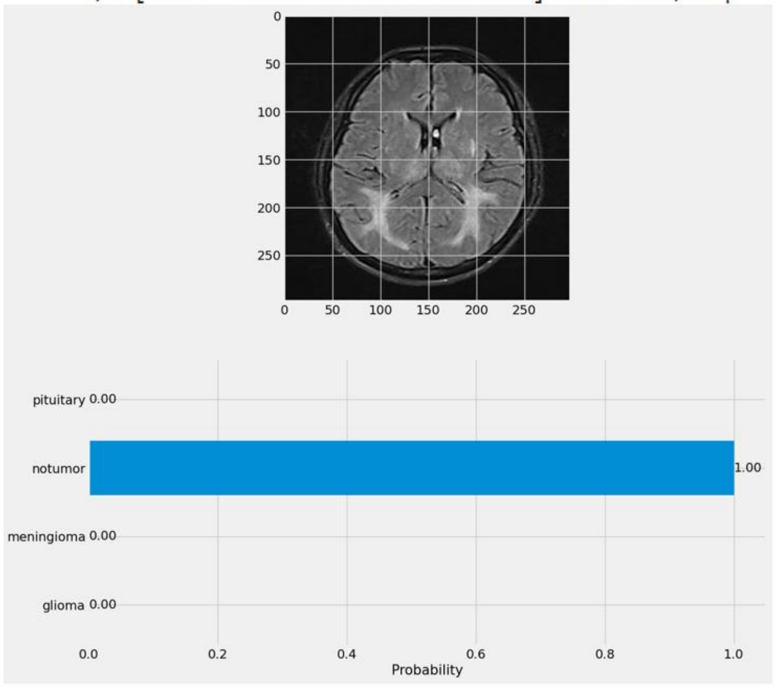


```
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/meningioma/Te-meTr_0000.jpg')

1/1 [==============================] - 2s 2s/step
```

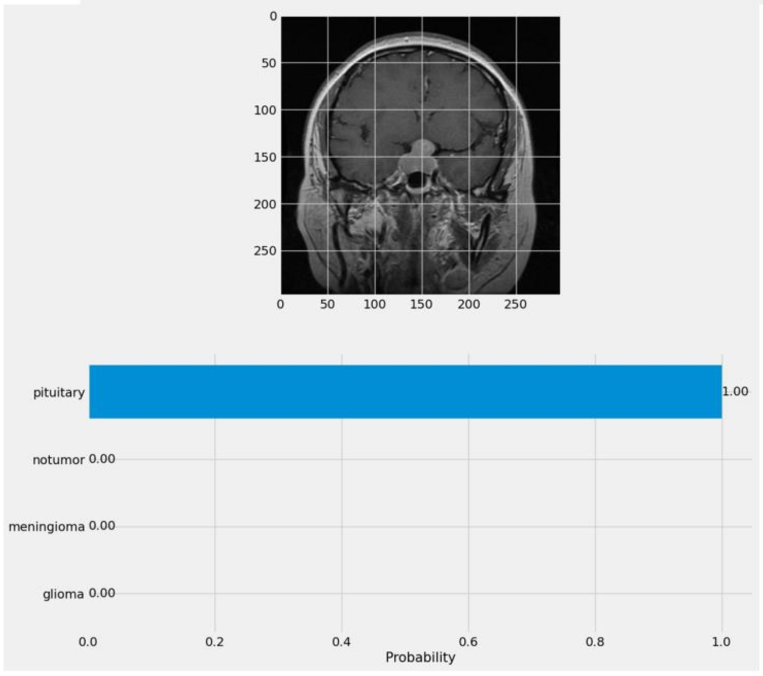


```
predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/glioma/Te-glTr_0007.jpg')

1/1 [==============================] - 0s 21ms/step
```

```
[ ]  predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/notumor/Te-noTr_0001.jpg')

     1/1 [==============================] - 0s 24ms/step
```



```
[ ]  predict('/content/drive/MyDrive/jcomp_brain/jcomp/Testing/pituitary/Te-piTr_0001.jpg')

     1/1 [==============================] - 0s 22ms/step
```



11. REFERENCES

Journal References:

[1] Pernas, F. J. D., Martínez-Zarzuela, M., Antón-Rodríguez, M., & González-Ortega, D. (2021, February 2). A Deep Learning Approach for Brain Tumor Classification and Segmentation Using a Multiscale Convolutional Neural Network. Healthcare. https://doi.org/10.3390/healthcare9020153

[2] Athisayamani, S., Antonyswamy, R. S., Velliangiri, S., Almeshari, M., Alzamil, Y., & Ravi, V. (2023, February 10). Feature Extraction Using a Residual Deep Convolutional Neural Network (ResNet-152) and Optimized Feature Dimension Reduction for MRI Brain Tumor Classification. Diagnostics. https://doi.org/10.3390/diagnostics13040668

[3] Rasool, M., Ismail, N. A., Boulila, W., Ammar, A., Samma, H., Yafooz, W. M. S., & Emara, A. H. M. (2022, June 8). A Hybrid Deep Learning Model for Brain Tumour Classification. Entropy. https://doi.org/10.3390/e24060799

[4] Alrashedy, H. H. N., Almansour, A. F., Ibrahim, D. M., & Hammoudeh, M. A. A. (2022, June 6). BrainGAN: Brain MRI Image Generation and Classification Framework Using GAN Architectures and CNN Models. Sensors. https://doi.org/10.3390/s22114297

Web References:

[1] Building a brain tumor classifier using deeplearning:

https://www.analyticsvidhya.com/blog/2022/07/building-a-brain-tumor-classifier-using-deep-learning/

[2] Hands-on Transfer Learning with Keras and the VGG16 Model:

https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/