Brain Tumor Classification using Deep Learning Frameworks: An Investigative Project

A Major Project Report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Akash Kumar Singh (201460)

Ritick (201357)

Under the guidance & supervision of

Dr. Rakesh Kanji

Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

CERTIFICATE

This is to certify that the work which is being presented in the project report titled "Brain Tumor Classification using Deep Learning Frameworks: An Investigative Project" in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering and submitted to the Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by "Akash Kumar Singh, 201460" and "Ritick, 201357" during the period from August 2023 to May 2024 under the supervision of Dr. Rakesh Kanji, Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat.

Akash Kumar Singh	Ritick
(201460)	(201357)

The above statement made is correct to the best of my knowledge.

(Dr. Rakesh Kanji)
Assistant Professor (SG)
Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

CANDIDATE’S DECLARATION

We hereby declare that the work presented in this report entitled **‘Brain Tumor Classification using Deep Learning Frameworks: An Investigative Project’** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** submitted in the Department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology, Waknaghat is an authentic record of our own work carried out over the period from August 2023 to May 2024 under the supervision of **Dr. Rakesh Kanji** (Assistant Professor - SG, Department of Computer Science & Engineering and Information Technology).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature with Date)

Student Name: Ritick	Akash Kumar Singh
Roll No.: 201357	201460

This is to certify that the above statement made by the candidates is true to the best of my knowledge.

(Supervisor Signature with Date)
Supervisor Name: Dr. Rakesh Kanji
Designation: Assistant Professor (SG)
Department: CSE & IT
Dated:

ACKNOWLEDGEMENT

We express our heartiest thanks and gratefulness to almighty God for his divine blessing that made it possible to complete the project work successfully.

We are really grateful and wish our profound indebtedness to our Supervisor Dr. Rakesh Kanji, Assistant Professor (SG), Department of CSE & IT, Jaypee University of Information Technology, Waknaghat. Deep knowledge & keen interest of our supervisor in the field of Machine Learning and Artificial Intelligence helped us to carry out this project successfully. His endless patience, scholarly guidance, continuous encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would also generously welcome each one of those individuals who have helped us straight forwardly or in a roundabout way in making this project a win. In this unique situation, we might want to thank the various staff individuals, both educating and non-instructing, who have facilitated our undertaking.

Finally, we must acknowledge with due respect the constant support and patience of our parents.

Ritick	Akash Kumar Singh
(201357)	(201460)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Abnormal formation and growth of cell within the brain gives rise to the Brain Tumors. Tumors can be Benign i.e. Non-cancerous or Malignant i.e. Cancerous. Tumors may also be classified on the basis of their origin into Primary and Secondary Tumors. Primary tumors are those which start to originate from within the brain itself, whereas Secondary are those that start to develop outside the brain and then move into the brain.

Primary Brain tumors are further classified into Meningioma which is generally benign, Glioma that is malignant 80% of the times and Pituitary tumors.

In the present world, Computer Aided Diagnosis systems are powerful means that assist the doctors and other experts in the detection and identification of various kinds of diseases that include heart diseases, tumors, cancer, etc. using medical images or other data. There have been several path breaking researches in this domain concerning different diseases, tools, aspects and specifications. This project is motivated and aimed to create a deep learning based model using Light-weight architecture to detect the presence of brain tumors from MRI images and classify the tumor into Meningioma, Glioma and Pituitary. In the recent past, there have been several works for the same task but most of the works have focused on using models having huge number of parameters, learning layers and large size for storage as well as high time consumption. This project aimed at efficiently performing the said task in less time and by using less storage space.

To achieve the proposed task, light-weight CNN architectures have been used in this project. These include MobileNet, EfficientNet, NASNetMobile, InceptionV3 and DenseNet121. Apart from using these pre-trained models, fine-tuning of these models has also been done to increase the performance. Hence, the best result was obtained by the fine-tuned version of EfficientNet, which achieved an accuracy of 94.25%.

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

Development of unwanted mass lesions and abnormal growth of cells in the brain results into brain tumors. Under the scope of this project work, the classification of brain tumors into Meningioma (generally benign), Glioma (malignant 80% of the times) and Pituitary tumors has been dealt with. When a tumor is said to be benign, it means that the tumor is not cancerous, whereas malignant tumor refers to cancerous tumor. It is noteworthy to mention here that the mentioned classification falls under the category of Primary Tumors, i.e. the tumors that are developed in the brain itself [1]. Apart from these, there are Secondary Tumors, which originate from some other body organ and then enter the brain [2]. The malignant tumors range across Grade 1 to 4 with Grade 4 being the deadliest. The cancerous tumor, starting from Grade 1, gradually get transformed into higher grade tumors. In the grade 4 survival of the patient becomes quite difficult [3]. Thus, it becomes increasingly important to detect brain tumor at the earliest. [4]

The role of doctors, radiologists and other experts definitely cannot be ruled out in the detection, classification and all the other subsequent actions in the cases of any disease. However, apart from conventional methods used for the purpose, which are very time consuming, there is a need of such a system which is very fast, is accessible, viable in terms of economy and scalability and is accurate enough to support the doctors and experts in taking the decisions related to the subject. In the normal course, doctors and experts detect and classify brain tumors with the help of MRI scans and CT scans that come under the domain of medical imaging. Thus, Computer Aided Diagnosis (CAD) is the system which can serve the purpose of classification of brain tumors in an effective manner. CAD makes use of the medical images and using the modern computerized techniques to serve the purpose of detection and classification of the diseases. The insights obtained from CAD are then referred to by the doctors and experts to take further calls related to the disease and necessary treatment. Under the domain of Artificial Intelligence, Deep Learning plays a very significant role in detection and classification of diseases. Deep convolutional neural network is a very adapt tool for classification of images [5].

This project uses the dataset of Brain MRI Scans and applies light-weight convolutional neural network architectures for detection and classification of brain tumors. Unlike the very well-known CNN models that consume a lot of time and need huge amount of disk space [6], the models used in this project are quite fast and reliable at the same time. The models are robust, give desired results and have very less storage requirements. Hence, the result of this project quite robustly serves the purpose of Brain Tumor Classification at minimal cost as well as very good accuracy.

1.2 PROBLEM STATEMENT

The process which doctors and experts use is the examination of the medical images like MRI scans of the brains to detect and classify brain tumors. This process consumes significant amount of time. There is a need to develop some automated system that can assist the doctors and experts in their work.

The task of this project is to develop a Computer Aided Diagnosis system using Light-Weight architectures of the deep learning domain to effectively detect and classify Brain Tumors. Such system, while consuming very less time and requiring very less amount of storage due to the light-weight architecture, can in turn assist the doctors and experts in decision making process. Thus, the task of Brain Tumor Detection and Classification can be performed at much enhanced pace.

1.3 OBJECTIVES

a) To develop a Computer Aided Diagnostics (CAD) system to detect and classify brain tumors from the brain MRI scans.

b) The Computer Aided Diagnostics system should be built in such a way that it costs way less than the present systems in place.

c) To use deep learning techniques and architectures to build a predictive model for the said CAD system with high accuracy and low cost.

1.4 SIGNIFICANCE AND MOTIVATION OF THE PROJECT WORK

The gravity of the problem can be understood by the fact that the average five year survival rate is just 33% in the US in case of brain cancers. Even benign tumors can be fatal depending upon their size and location. The best possible way to deal is to detect the brain tumor and classify it at the earliest possible stage, i.e. minimum grade possible. This warrants the detection and classification system to be very fast and viable in all terms. As a result, there is a need of a system to assist the doctors and experts in their work, so that time frame can be reduced as much as possible. This provides the Motivation for this project work.

This project work will make immense contribution to the medical domain of image based brain tumor detection and classification. The significance will be in terms of a system which will consume less time and occupy less space as compared to the present system without compromising on the performance part.

1.5 ORGANIZATION OF THE PROJECT REPORT

The rest of the project report is organized in such a way that Chapter 2 represents the Literature Survey, Chapter 3 corresponds to System Development, Chapter 4 is dedicated to Testing, Chapter 5 is of Results and Evaluation and Chapter 6 gives the Conclusions and Future Scope.

CHAPTER 2: LITERATURE SURVEY

2.1 OVERVIEW OF RELEVANT LITERATURE

Abhiwinanda et al. [7] implemented simple architecture of CNN that consisted of one layer each of convolution, max pooling and flattening followed by full connection. The objective of brain tumor classification into Meningioma, Glioma and Pituitary was achieved with training accuracy of 98.51% and validation accuracy of 84.19%. This work was implemented using Figshare dataset [8].

Ahmad et al. [9] did rigorous augmentation of brain MRI scans using Generative Adversarial Networks coupled with Variational Autoencoders to increase the size of the dataset. ResNet50 model was applied. Without augmentation, the classification accuracy was 72.63% and after augmentation, the same improved to 96.25%. For most severe class of brain tumor, glioma, results were 0.769, 0.837, 0.833 and 0.80 corresponding to recall, precision, specificity and F1 score respectively.

Afshar et al. [10] proposed a CapsNet architecture that uses both, raw MRI brain images as well as tumor course boundaries. With this method, the need of annotation of tumor is eliminated and architecture is able to focus on main area. The proposed model achieved an accuracy of 90.89%.

Chelghoum et al. [11] used nine different pre-trained models of deep learning domain for the purpose. AlexNet, VGG19, GoogleNet, ResNet18, VGG16, ResNet50, ResNet-Inception-v2, ResNet101 and SENet were used for the task and an accuracy of 98% was achieved. In all the pre-trained models, the end three layers were modified after which a fully connected layer was added according to the size of required output.

Kokkalla et al. [12] used Inception Resnet v2 with customised output layer on a brain tumor dataset of 3064 images.

Gumaei et al [13] proposed a Regularized Extreme Learning Machine (RELM) model along with feature extraction in a hybrid manner to classify the brain tumors. An accuracy of 94.233% was achieved in the said approach. RELM can be used for both classification as well as regression and in advantageous in terms of speed of training and low complexity. It also tends to overcome some disadvantages of backpropagation method [18]. Guamei et al. [16], [17] proposed Normalized GIST descriptor for feature extraction, which is an improved version of traditional GIST descriptor proposed by Oliva and Torralba [15]. NGIST solves the issues related to illumination and shadowing by normalizing the data using L2 norm. The NGIST was coupled with Principal Component Analysis (PCA) to form PCA-NGIST [13] that was efficiently used to extract features from brain images.

Ari and Hanbay [14] used ELM-LRF i.e. Extreme Learning Machine Local Receptive Fields to classify brain tumor into cancerous or non-cancerous. The concept of ELM was introduced by Huang et al. [19] for multiclass classification by making use of RELM.

Anaraki et al. [20] proposed a CNN architecture consisting of six convolutional and max pooling layers and then FC (fully connected) layer. The model achieved an accuracy of 94%. Sajjad et al. [21] developed a deep CNN with data augmentation for multi-grade tumor classification. The pre-trained CNN model is refined for classification. Similarly, Swati et al. [22] proposed a method for brain tumor image classification using fine-tuned transfer learning. These deep transfer learning methods have shown a slight increase in accuracy. On the other hand, Deepak and Ameer [23] proposed deep transfer learning and a support vector machine (SVM) for three-class classification. The authors used a pre-trained GoogleNet to extract features from brain MR images. The SVM classifier was then used for classification. Afshar et al. [24] came out with an approach to ascertain uncertain predictions as well, so as to improve the performance. For this task, the authors used Bayesian Capsule Network, also called BayesCap. Togacar et al. [25] proposed a new model called BrainMRNet for brain tumor classification. The model worked better than the pre-trained models like VGG-16, GoogleNet and AlexNet. The classification accuracy was 96%.

2.2 KEY GAPS IN THE LITERATURE

Ahmad et al [9] have performed extensive preprocessing of the data by rigorous augmentation. Also, in the form of ResNet50, a heavy-weight model has been used that consumes fairly good amount of time. These factors have been adequately addressed in our project work.

The work carried out Afshar et al [10] leaves behind a future scope interpretability of CapsNet architecture for brain tumor classification.

In [7], [9], [10], there is much scope of performance improvement in terms of validation accuracy.

Majority of the works carried out largely leave behind a scope to experiment with light-weight CNN architectures to save the cost. The same has been considered the prime objective of this project work.

CHAPTER 3: SYSTEM DEVELOPMENT

3.1 REQUIREMENTS AND ANALYSIS

The proposed brain tumor classification system aims to accurately classify brain tumors from magnetic resonance imaging (MRI) scans. The system should meet the following requirements:

- Accuracy:
  The system should achieve high accuracy in classifying brain tumors into different types, i.e. glioma, meningioma, and pituitary tumor.

- Generalizability:
  The system should be able to generalize well to unseen data, ensuring its effectiveness in real-world clinical settings.

- Efficiency:
  The system should be computationally efficient, allowing for real-time or near-real-time classification.

- Ease of use:
  The system should have a user-friendly interface that is accessible to medical professionals with varying levels of technical expertise.

- Data Requirements:
  The performance of deep learning models is highly dependent on the quality and quantity of training data. The proposed system will require a large dataset of labeled MRI scans of brain tumors, with accurate annotations for tumor type and grade. The dataset should be diverse and representative of the real-world distribution of brain tumors.

- Performance:

  The system should be able to handle large volumes of data and provide predictions in a timely manner. It should also be able to scale up to accommodate increasing data volumes as needed.

- Reliability:

  The system should be reliable and able to provide accurate predictions consistently.

- Scalability:

  The system should be scalable and able to handle increasing data volumes.

- Maintainability:

  The system should be easy to maintain and update, with clear documentation and modular design.

- Compatibility:

  The system should be compatible with a wide range of operating systems and web browsers.

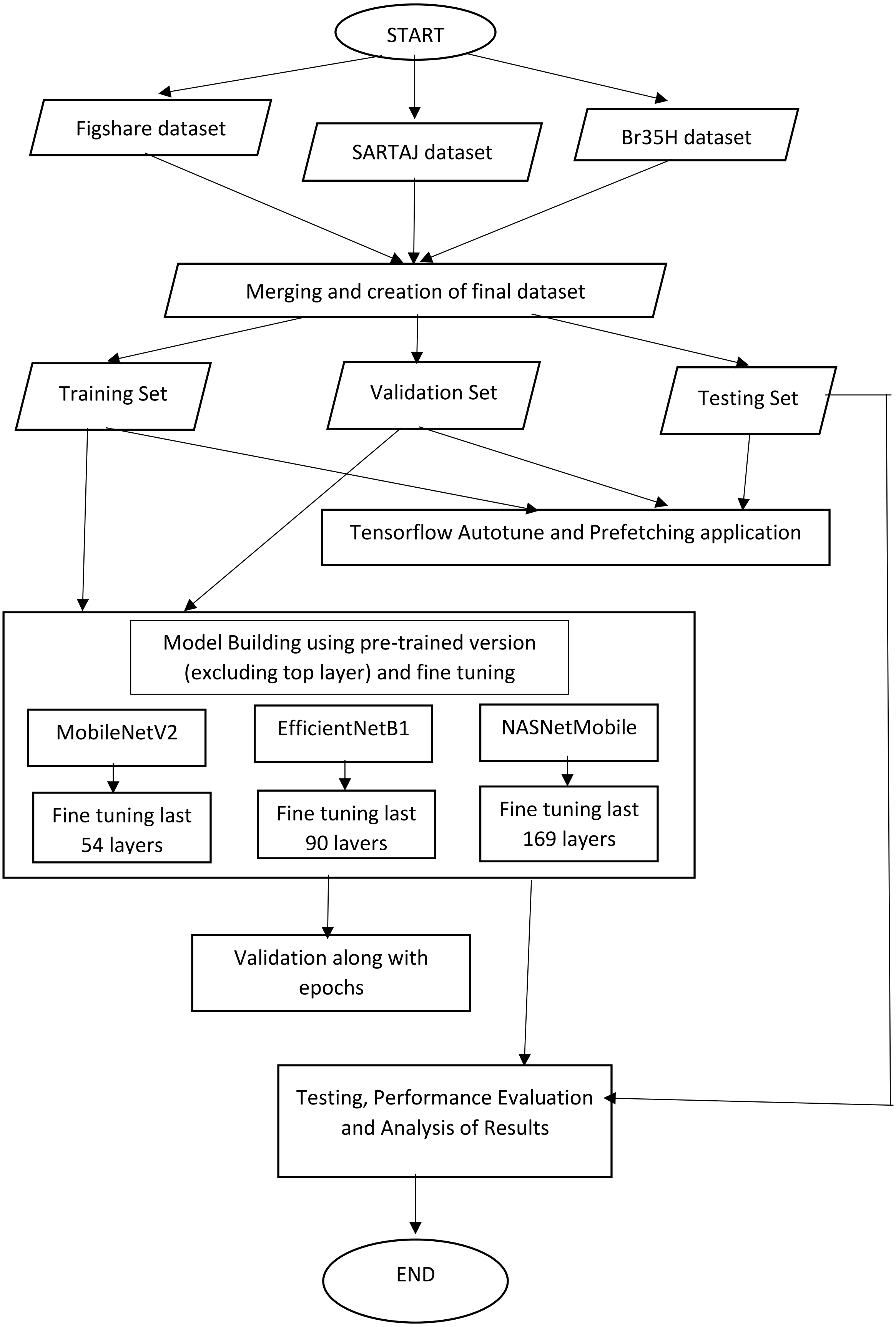
- Performance Efficiency:

  The system should use system resources efficiently and minimize response times.

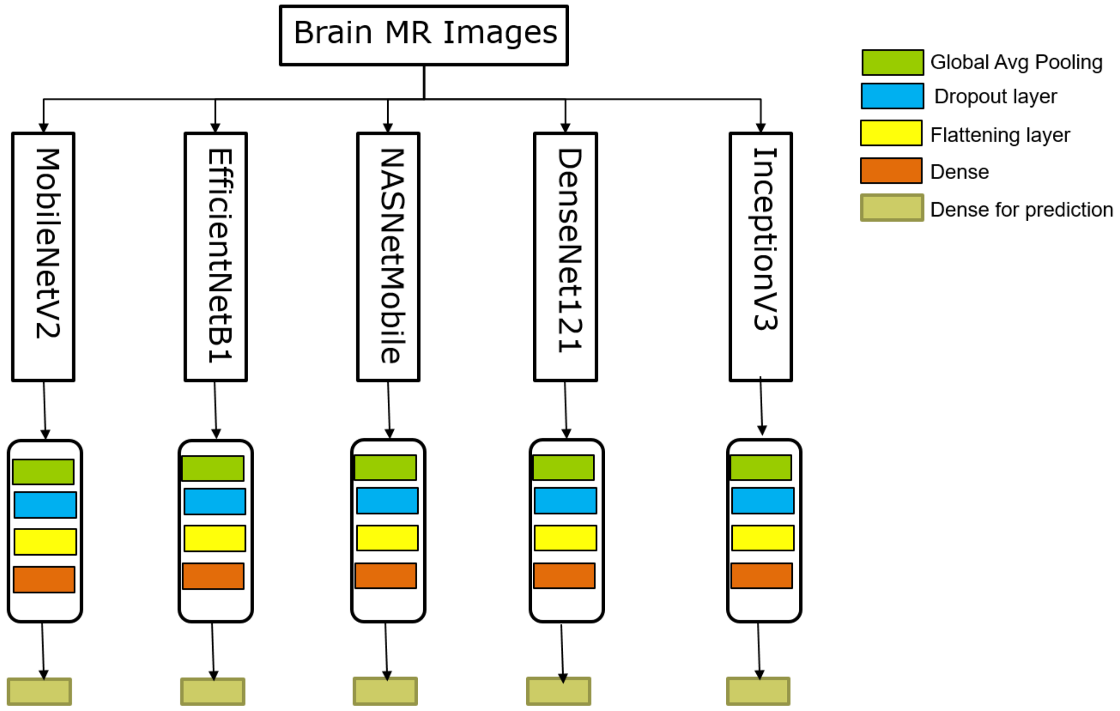
- Interoperability:

  The system should be able to integrate with other systems and tools as needed, such as data visualization tools or data analytics platforms.

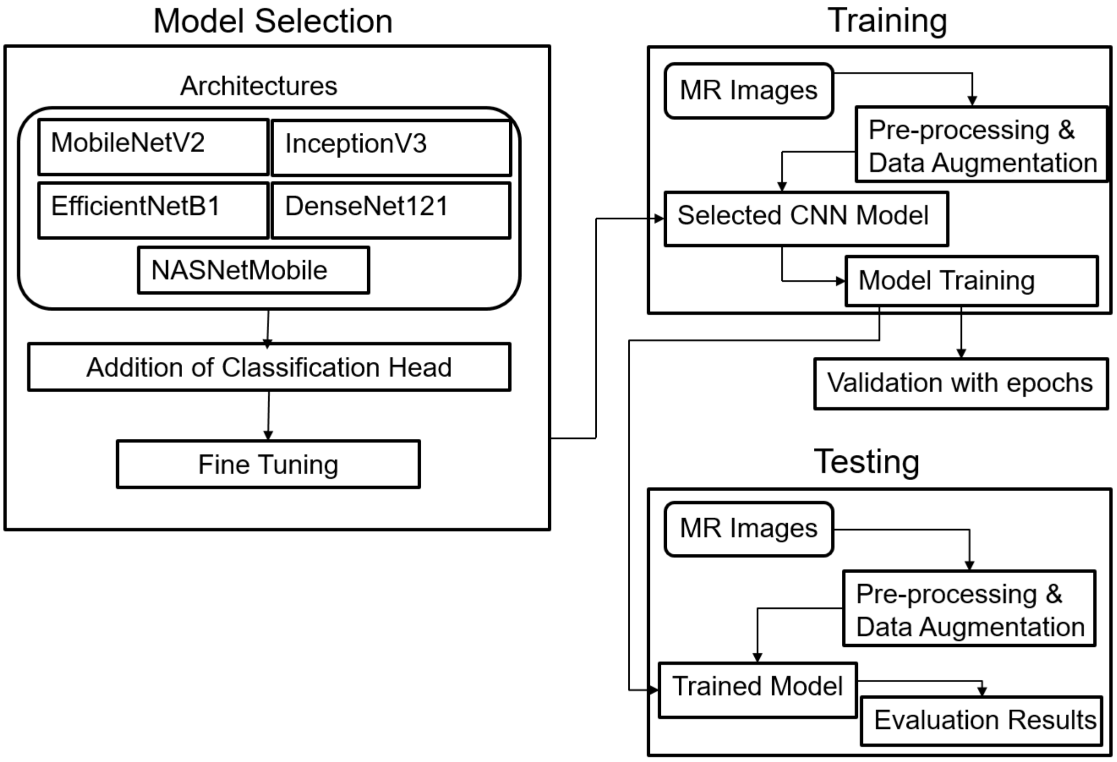
3.2 PROJECT DESIGN AND ARCHITECTURE


START
Figshare dataset
SARTAJ dataset
Br35H dataset
Merging and creation of final dataset
Training Set
Validation Set
Testing Set
Tensorflow Autotune and Prefetching application
Model Building using pre-trained version (excluding top layer) and fine tuning
MobileNetV2
EfficientNetB1
NASNetMobile
Fine tuning last 54 layers
Fine tuning last 90 layers
Fine tuning last 169 layers
Validation along with epochs
Testing, Performance Evaluation and Analysis of Results
END

The structuring of the pre-trained models with custom layers added by replacing the top layer is shown below. Global Average Pooling, Dropout Layer, Flattening Layer and Dense layers have been used.



A general view of the Project Design is shown below.



3.3 DATA PREPARATION

For our project, we needed dataset of Brain MR Images. Some popular publicly available datasets of MRI scans are Figshare dataset [8], SARTAJ dataset [26], Br35H dataset [27]. Figshare dataset consists of 3064 images of MRI scans, SARTAJ dataset consists of 3264 images and Br35H dataset consists 3959 images. For our task, since the CNN architectures are data dependent, there was a need of ample quantity of data so that model training could be carried out effectively. Thus, with a view of obtaining increased performance, to avoid overfitting and to efficiently achieve our goal, all the three mentioned datasets were merged. In this way, our final dataset consisted of total 10287 files. This combined dataset existed with images divided into Training Data and Testing Data. Each folder i.e. Training and Testing further consisted of 4 sub-folders each- meningioma, glioma, pituitary and no_tumor. Going forward, this division of training and testing was expanded into three sets- Training set, Validation set and Testing set, during the course of implementation by the way of programming.

Table 3.1 shows the number of images in each subfolder.

Table 3. 1. Dataset Details

	Training	Testing	**Total**
Meningioma	2161	421	637
Glioma	2147	400	2547
Pituitary	2284	374	2658
No Tumor	1990	510	2500
Total	8582	1705	10287
Percentage	83.42%	16.5%	

The snapshots of the dataset are as follows-

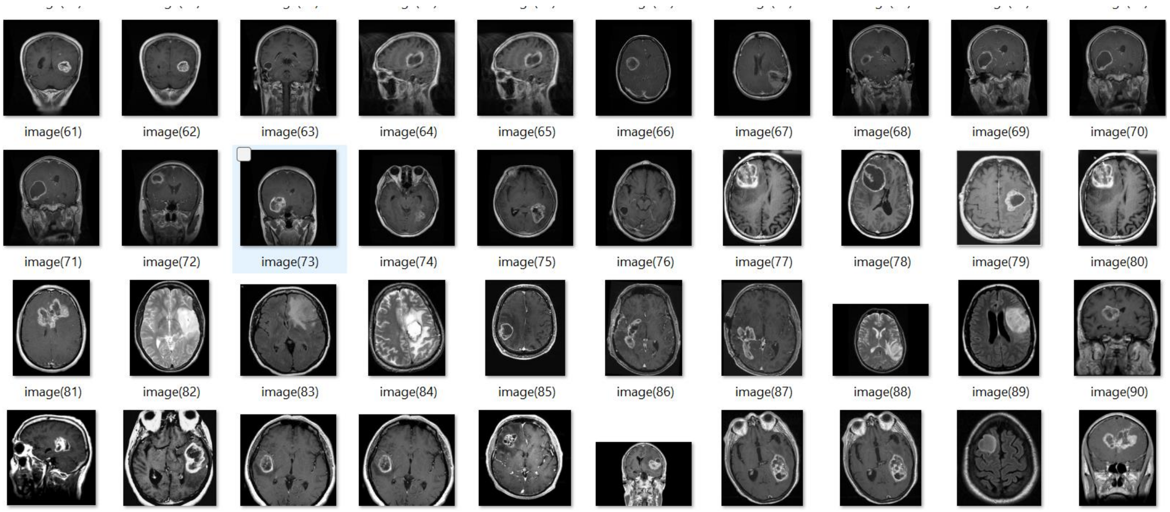


Figure 3. 1. Glioma - Test Set

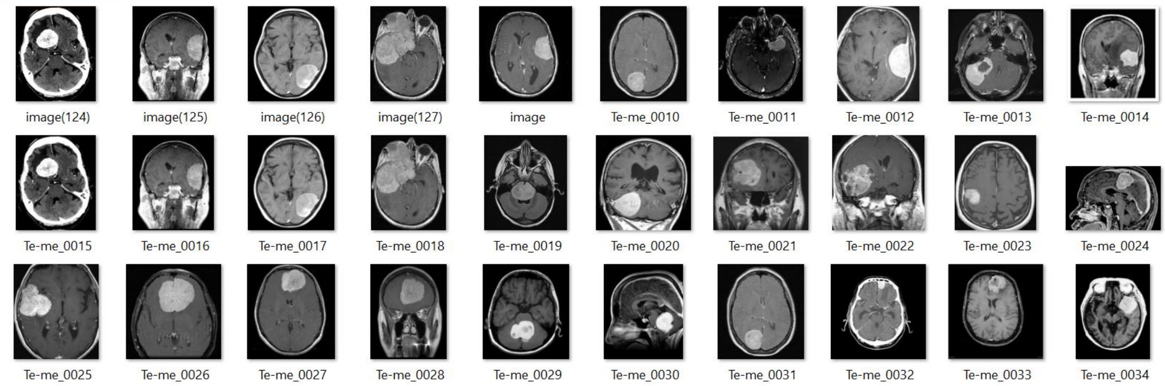


Figure 3. 2. Meningioma – Test Set

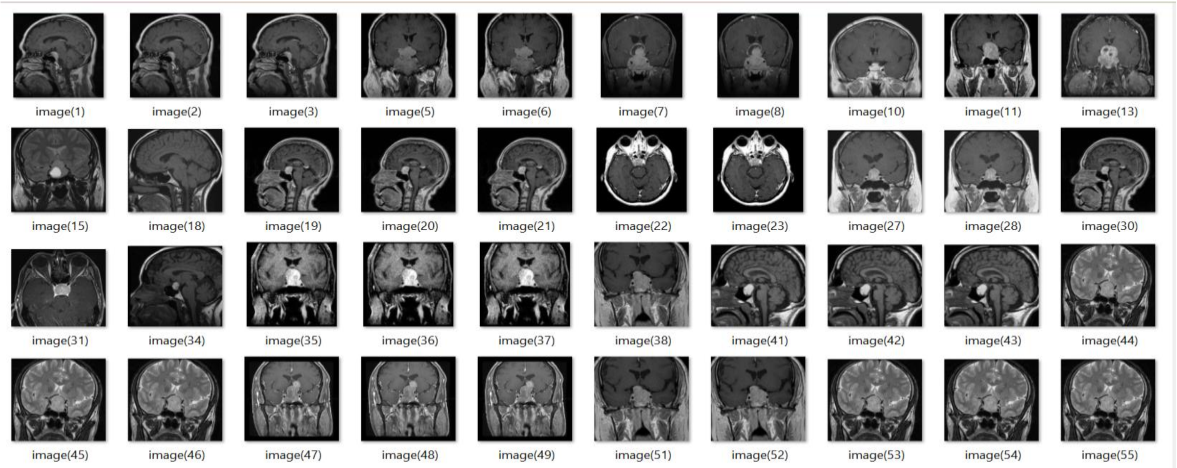


Figure 3. 3. Pituitary tumor – Test Set

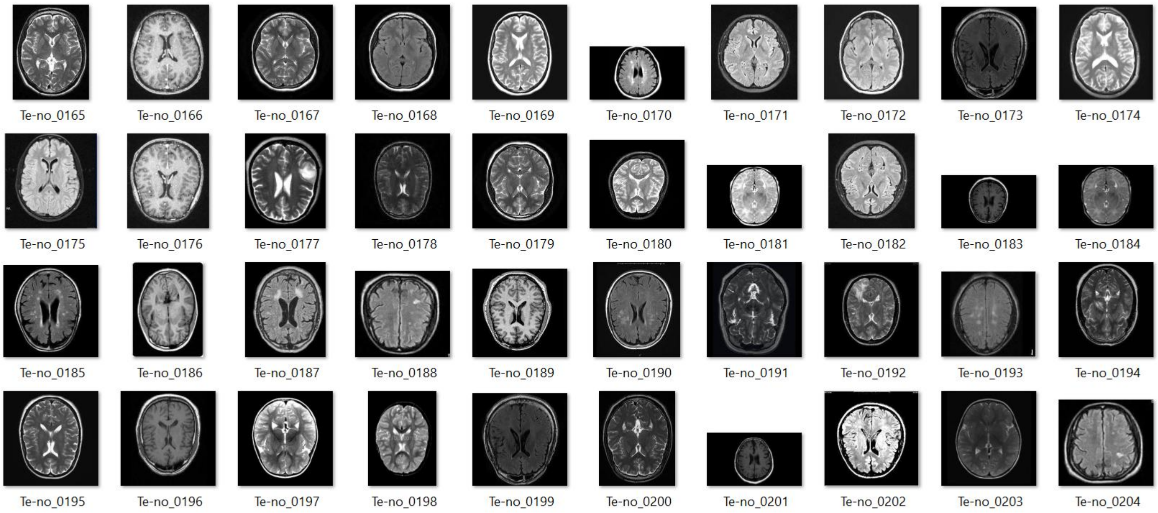


Figure 3. 4. No tumor – Test Set

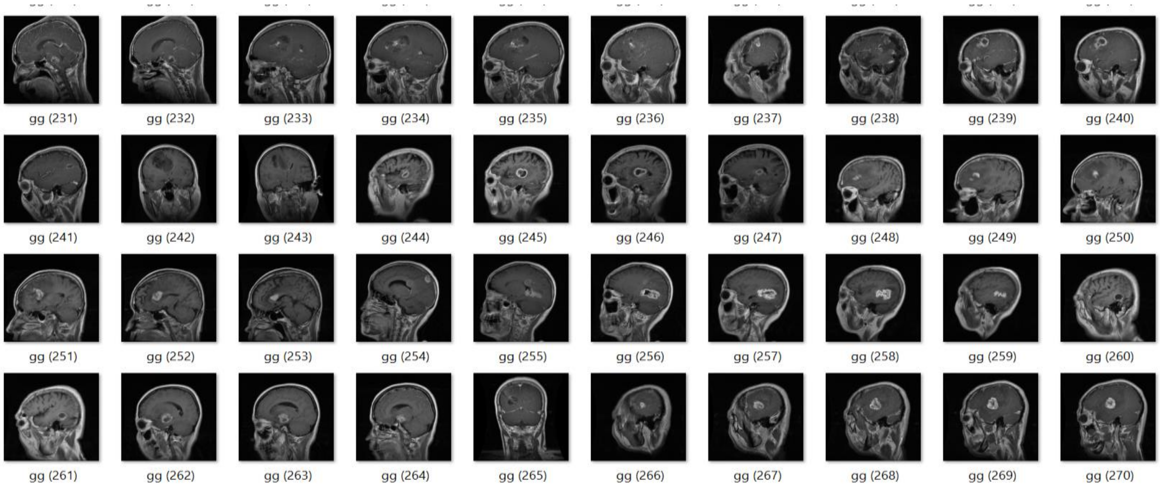


Figure 3. 5. Glioma – Training Set

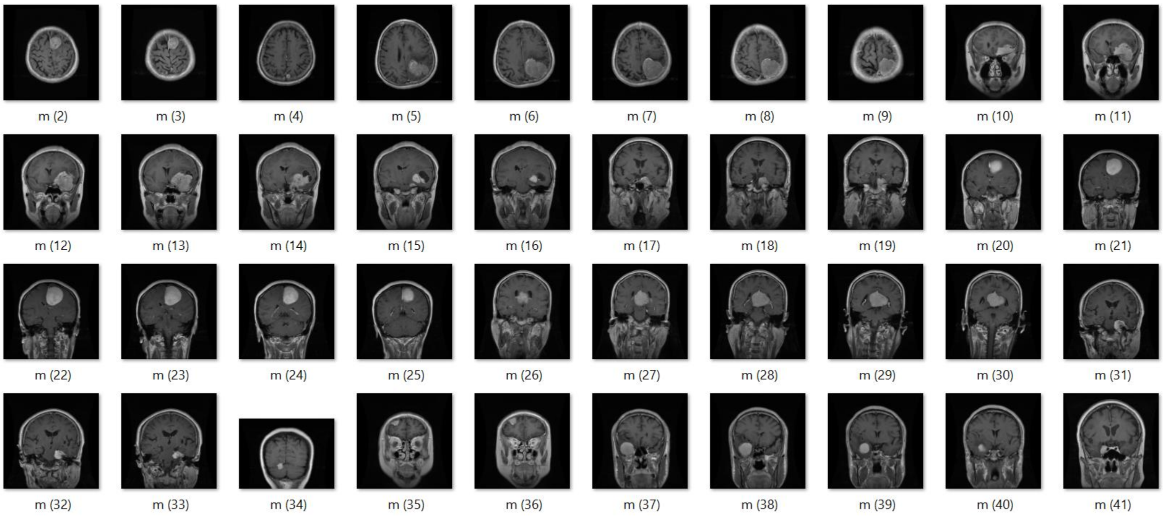


Figure 3. 6. Meningioma – Training Set

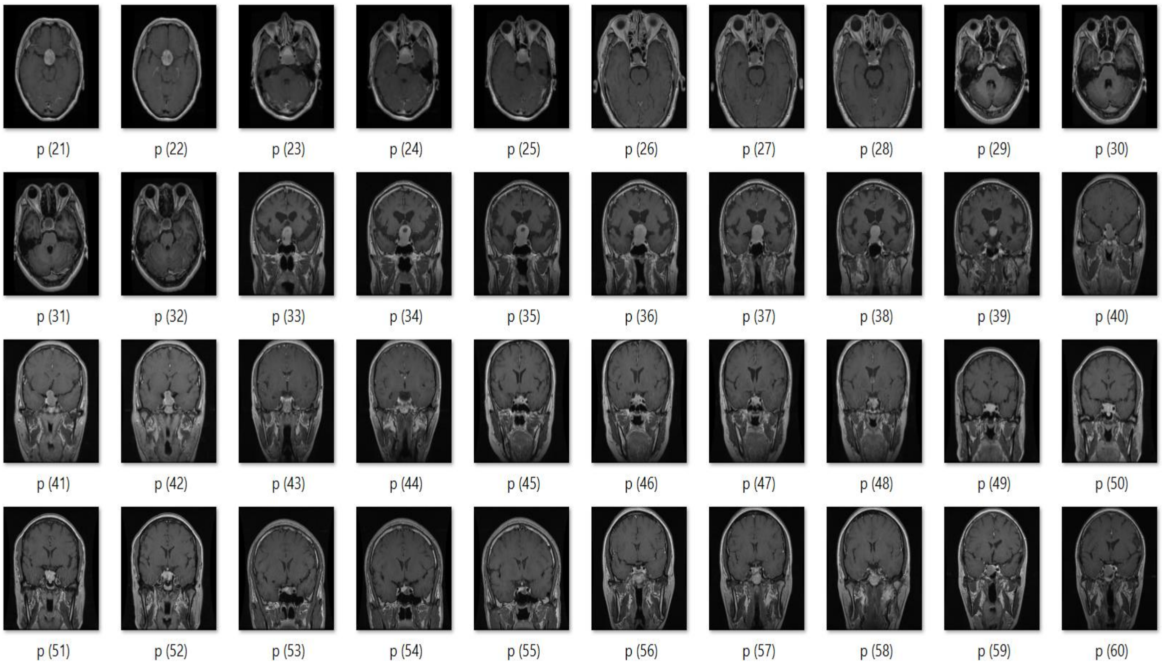


Figure 3. 7. Pituitary Tumor – Training Set

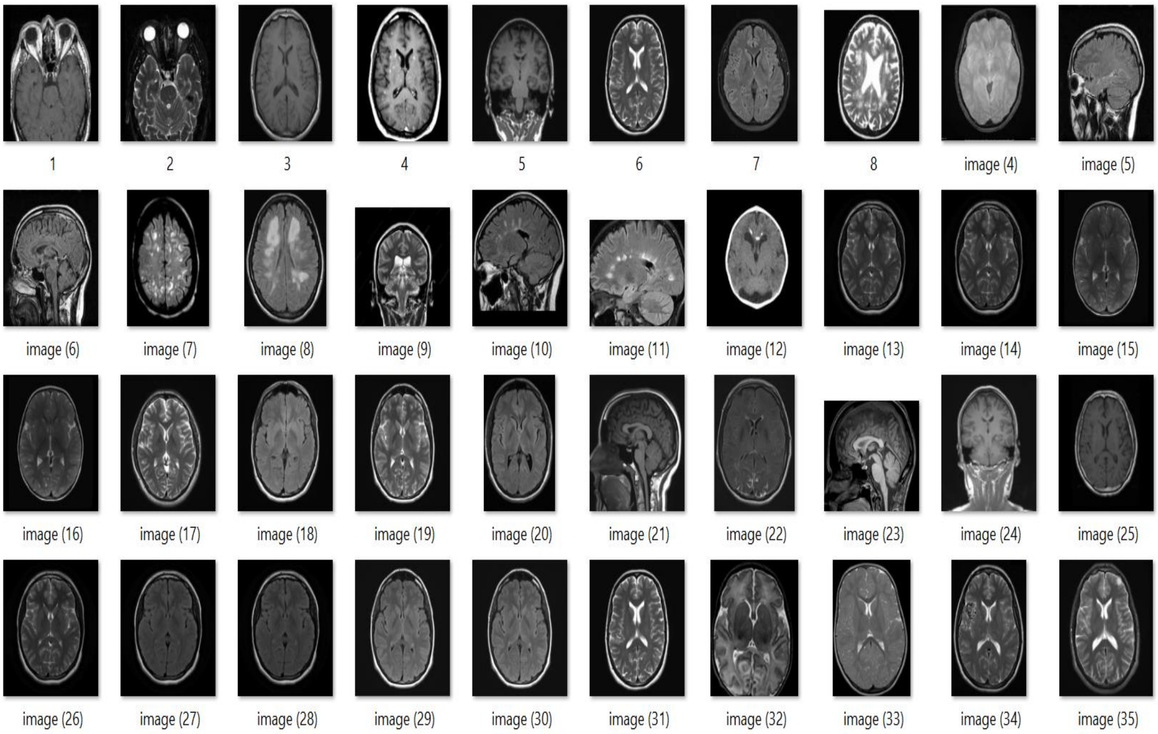


Figure 3. 8. No tumor – Training Set

3.4 IMPLEMENTATION

The aim of the project work was to use light-weight deep learning models to efficiently classify brain tumors into Meningioma, Glioma, Pituitary and No tumor. Hence, after carefully evaluating all the possible methods and architectures, it was decided to implement and test five architectures – MobileNetV2 [28], EfficientNet [29], NASNetMobile [30], InceptionV3 [32] and DenseNet121 [33].

These architectures have been chosen due to their low time and space requirements. The time and space requirements and other details of these architectures have been shown in Table 3.2. For implementation of these models, keras open-source library which is a part of tensorflow was used in python language. The details have been obtained from Keras Applications documentation [31].

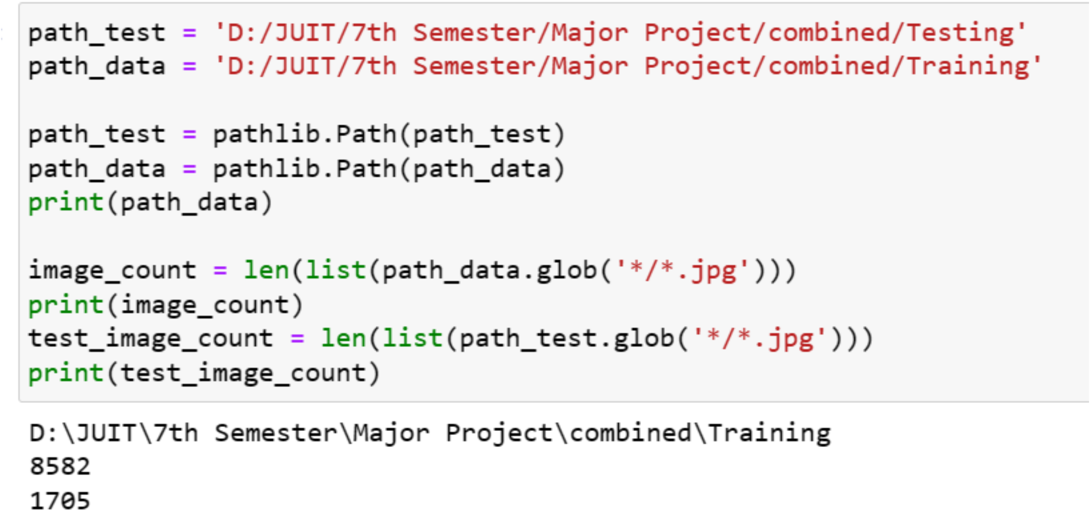
Table 3. 2. Details of the architectures used

Models	Size (MB)	Parameters	Depth	Time (ms) per inference step (CPU)
MobileNetV2	14	3.5M	105	25.9
EfficientNetB1	31	7.9M	186	60.2
NASNetMobile	23	5.3M	389	27
Inception V3	92	23.9M	189	42.2
DenseNet121	33	8.1M	242	77.1

These models have been trained on ImageNet validation dataset.

The network's topological depth is referred to as Depth in the Table 3.2. This covers layers for batch normalization, activation, etc. Depth keeps track of the number of parameterized layers.

The average time for 30 batches and 10 repeats is used for each inference step where each batch is of 32 size, CPU is AMD EPYC Processor (with IBPB) (92 core) and RAM is 1.7T.

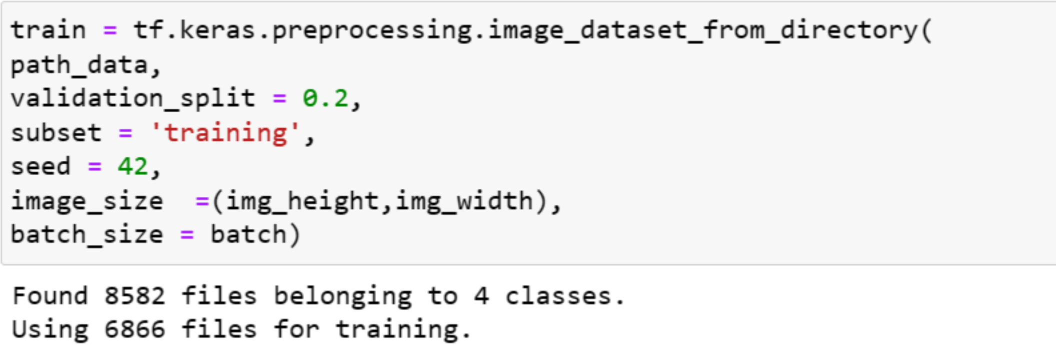


```
path_test = 'D:/JUIT/7th Semester/Major Project/combined/Testing'
path_data = 'D:/JUIT/7th Semester/Major Project/combined/Training'

path_test = pathlib.Path(path_test)
path_data = pathlib.Path(path_data)
print(path_data)

image_count = len(list(path_data.glob('*/*.jpg')))
print(image_count)
test_image_count = len(list(path_test.glob('*/*.jpg')))
print(test_image_count)
```

```
D:\JUIT\7th Semester\Major Project\combined\Training
8582
1705
```

Figure 3. 9. Setting of path for fetching data

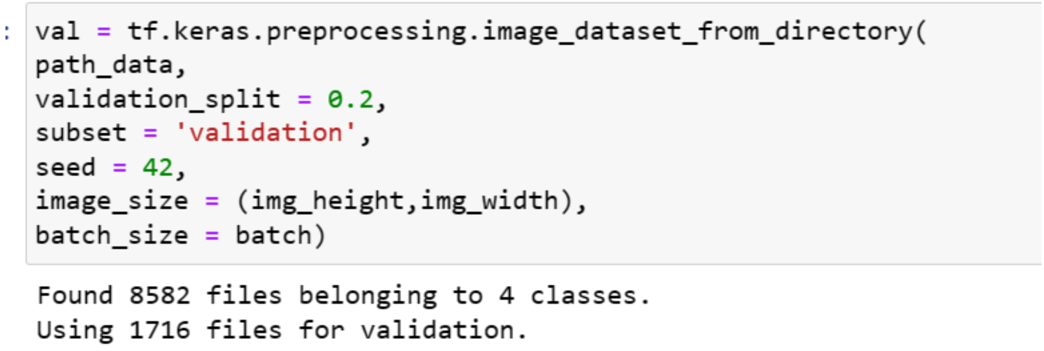
As already shown earlier, the dataset existed divided into Training and Testing data. Now, while implementation, 20% out of Training data was carved out and a Validation set was created. So, after this action, 66.74% i.e. 6866 files of the total dataset of 10287 files were used for Training, 16.68% i.e. 1716 files were used for Validation and 16.5% i.e. 1705 files were used for Testing. All these details have been shown in below figures.



```
train = tf.keras.preprocessing.image_dataset_from_directory(
path_data,
validation_split = 0.2,
subset = 'training',
seed = 42,
image_size  =(img_height,img_width),
batch_size = batch)
```

```
Found 8582 files belonging to 4 classes.
Using 6866 files for training.
```

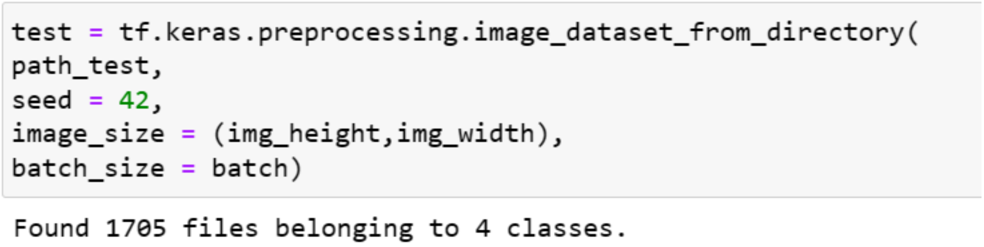
Figure 3. 10. Carving out of Validation set – Showing number of files in Training set



```
val = tf.keras.preprocessing.image_dataset_from_directory(
path_data,
validation_split = 0.2,
subset = 'validation',
seed = 42,
image_size = (img_height,img_width),
batch_size = batch)
```

```
Found 8582 files belonging to 4 classes.
Using 1716 files for validation.
```

Figure 3. 11. Carving out of Validation set – Showing number of files in Validation set

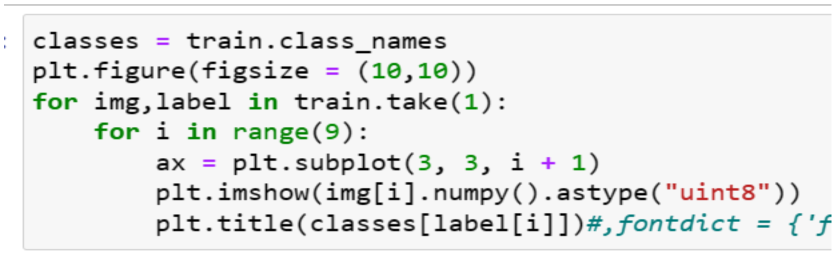


```
test = tf.keras.preprocessing.image_dataset_from_directory(
path_test,
seed = 42,
image_size = (img_height,img_width),
batch_size = batch)
```

```
Found 1705 files belonging to 4 classes.
```

Figure 3. 12. Testing set

Before proceeding for model creation, the dataset was visualized as shown in Figure 3.13 and 3.14. The visualization was done along with the label names of the images. Batch sizes, image dimensions and number of channels were checked, Autotune was applied and pre-fetching was used to optimize the performance as shown in Figure 3.15. Pre-fetching loads next batch of items parallel to the current execution of predecessor batch.



```
classes = train.class_names
plt.figure(figsize = (10,10))
for img,label in train.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(img[i].numpy().astype("uint8"))
        plt.title(classes[label[i]])#,fontdict = {'f
```

Figure 3. 13. Code for visualization of Training set

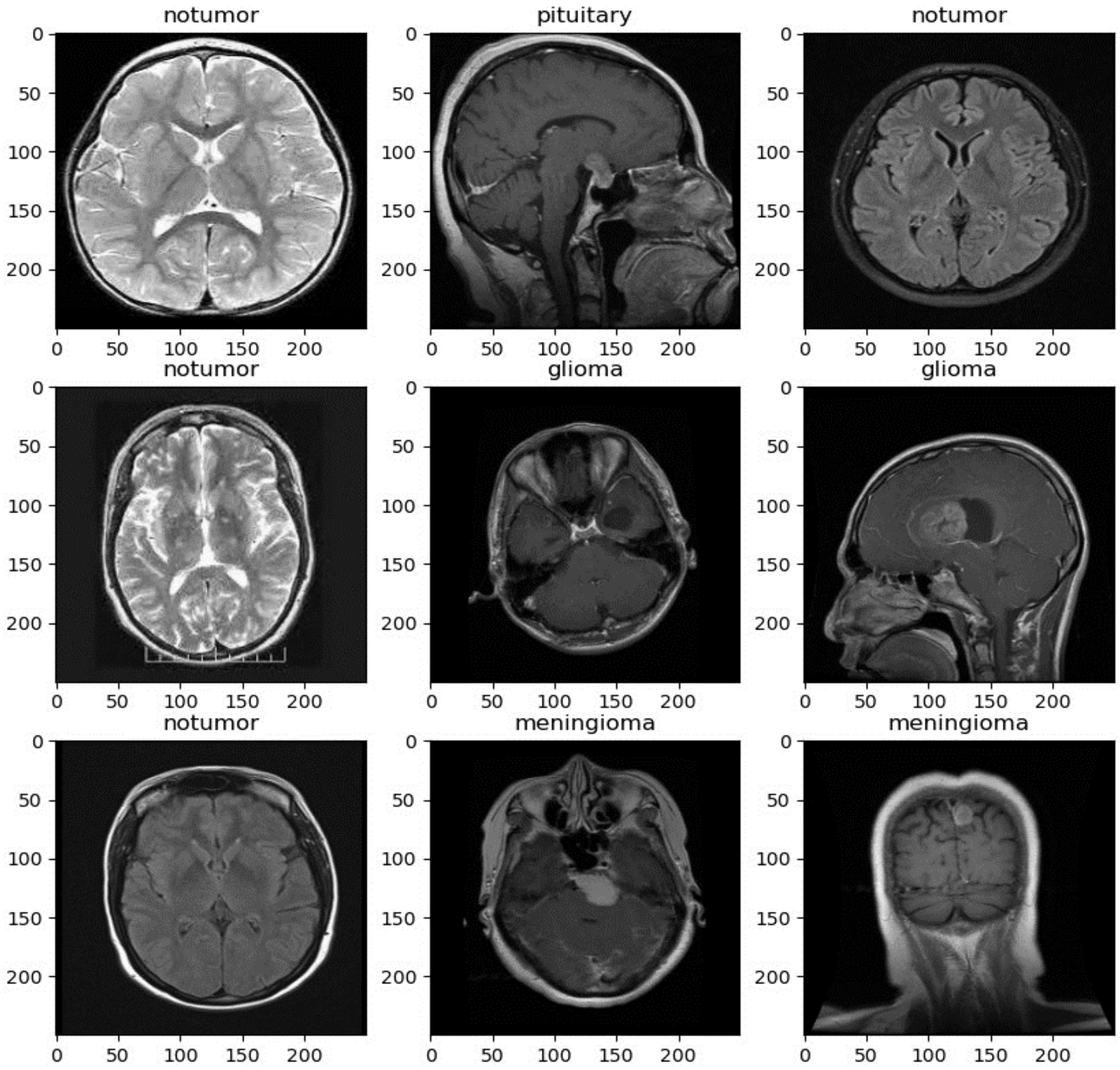
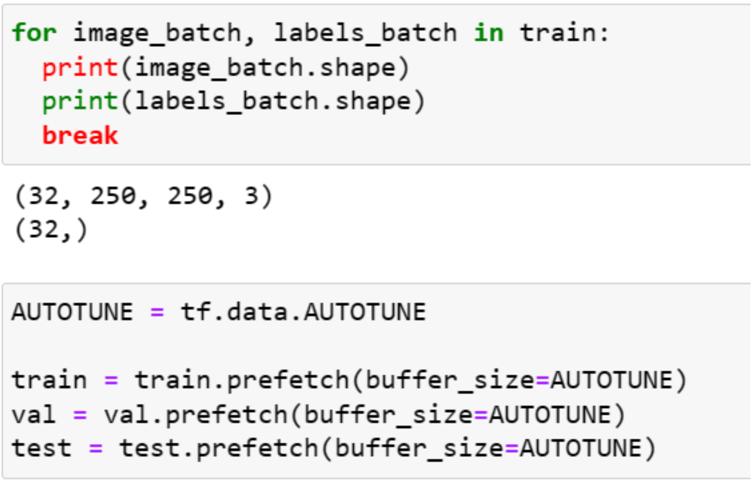


Figure 3. 14. Dataset visualization output



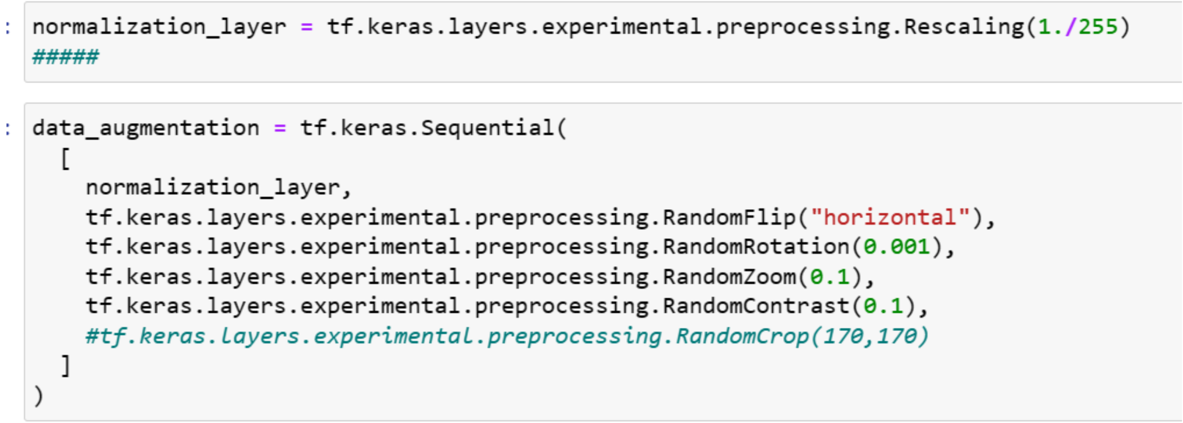
```
for image_batch, labels_batch in train:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 250, 250, 3)
(32,)
```

```
AUTOTUNE = tf.data.AUTOTUNE

train = train.prefetch(buffer_size=AUTOTUNE)
val = val.prefetch(buffer_size=AUTOTUNE)
test = test.prefetch(buffer_size=AUTOTUNE)
```

Figure 3. 15. Pre-fetching applied

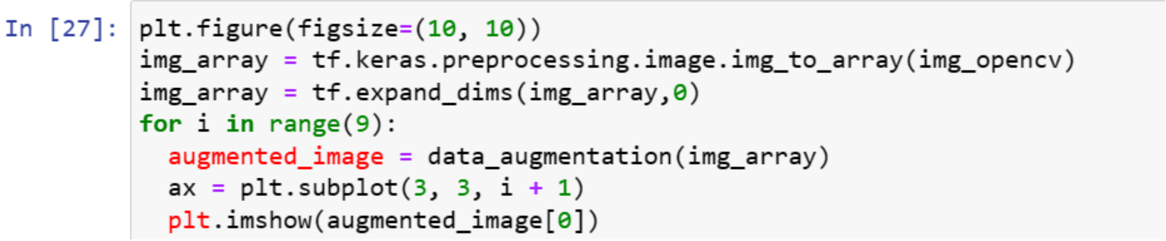
An important requirement of deep learning models is enormous amount of data. In the medical domain, this cannot be ensured automatically or inherently. Thus, Data Augmentation is the tool widely used to increase the size of the dataset. Using Augmentation, artificial images of the existing images are created by introducing features like flipping, rotation, cropping, hazing and de-hazing, changing the contrast, zooming, etc. So, the Training data was augmented before its use. Rescaling was done, horizontal flips were introduced, random rotations were introduced, and zoom and contrast were carried out. The code for such augmentation is shown in Figure 3.16. To display a few augmented images, code is shown in Figure 3.17 and images have been shown in Figure 3.18.



```
: normalization_layer = tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
  #####
```

```
: data_augmentation = tf.keras.Sequential(
    [
      normalization_layer,
      tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
      tf.keras.layers.experimental.preprocessing.RandomRotation(0.001),
      tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
      tf.keras.layers.experimental.preprocessing.RandomContrast(0.1),
      #tf.keras.layers.experimental.preprocessing.RandomCrop(170,170)
    ]
  )
```

Figure 3. 16. Code for data augmentation



```
In [27]: plt.figure(figsize=(10, 10))
         img_array = tf.keras.preprocessing.image.img_to_array(img_opencv)
         img_array = tf.expand_dims(img_array,0)
         for i in range(9):
           augmented_image = data_augmentation(img_array)
           ax = plt.subplot(3, 3, i + 1)
           plt.imshow(augmented_image[0])
```

Figure 3. 17. Code to display some augmented images

Now, the data pre-processing stage was complete. Further, the stage was to create and train CNN architectures for the classification of brain tumors. In line with our objective, pre-trained version of MoblieNetV2 on ImageNet dataset was initialized as our Base model as shown in 3.19. The top layer of the pre-trained version was excluded and the base model was set to non-trainable.

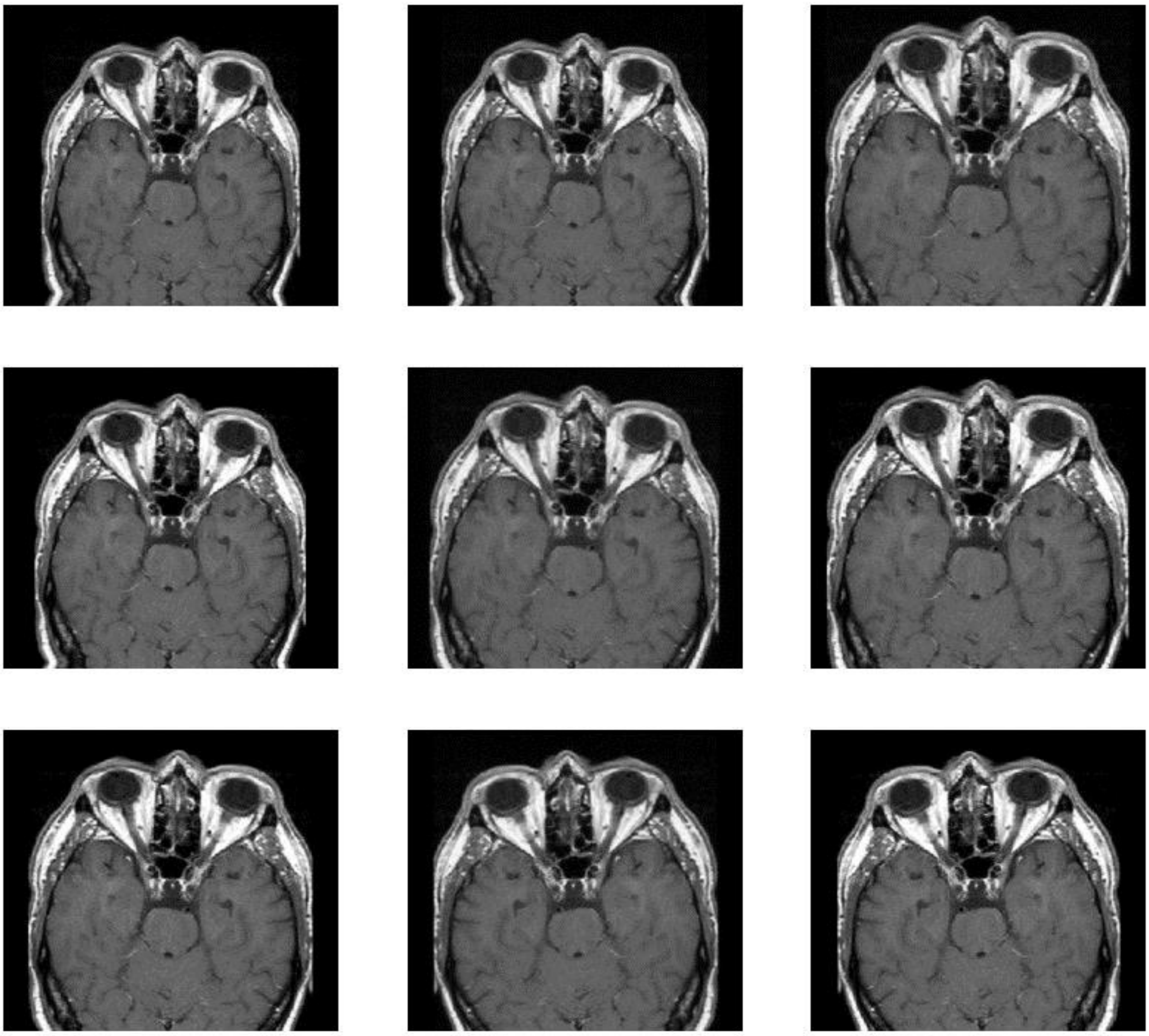
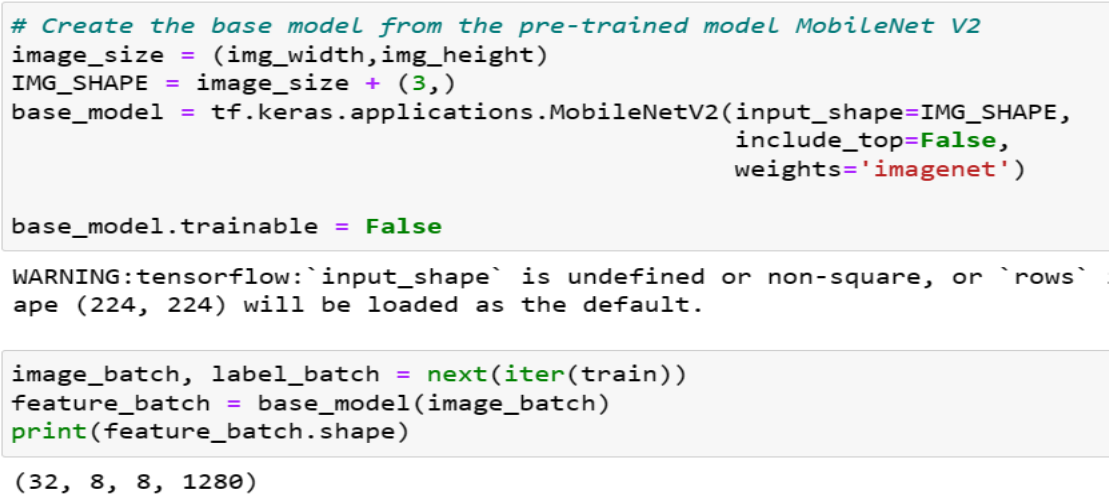


Figure 3. 18. Augmented images



```
# Create the base model from the pre-trained model MobileNet V2
image_size = (img_width,img_height)
IMG_SHAPE = image_size + (3,)
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')

base_model.trainable = False
```

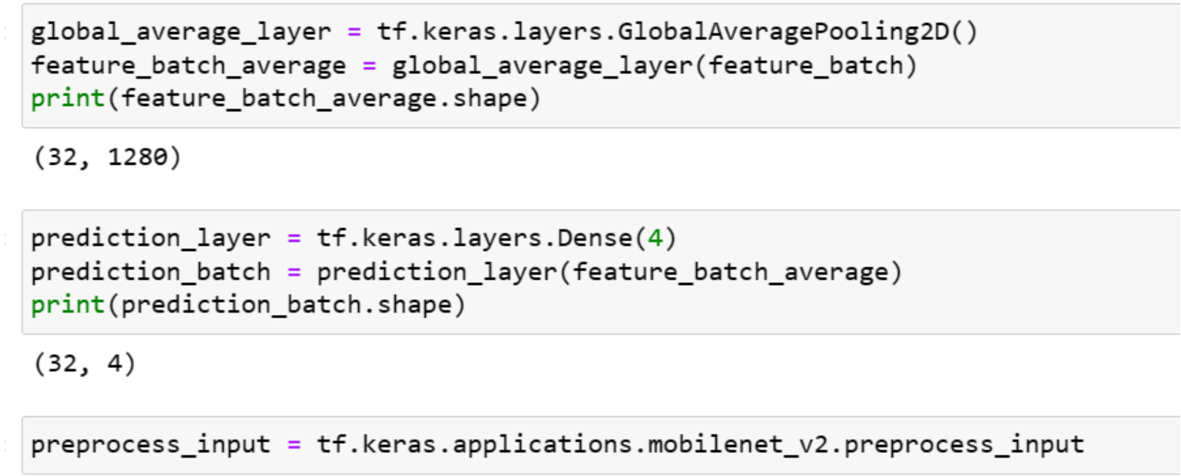
```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows`
ape (224, 224) will be loaded as the default.
```

```
image_batch, label_batch = next(iter(train))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 8, 8, 1280)
```

Figure 3. 19. Initialization of pre-trained version of MobileNetV2

On the top of the pre-trained base model, an average pooling layer was added, dropout was included, flattening layer was added and then a dense layer was added with ReLU activation function. Before passing inputs to the model, pre-processing specific to MobileNetV2 was done.



```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

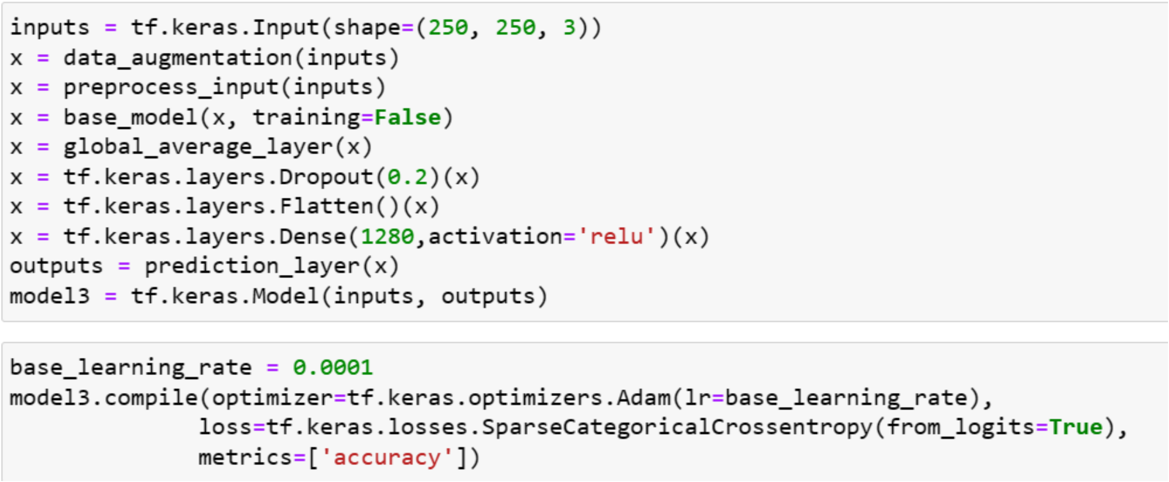
```
prediction_layer = tf.keras.layers.Dense(4)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 4)
```

```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

Figure 3. 20. Initialization of Average Pooling Layer and Dense Layer and Pre-processing of Inputs for MobileNetV2 model

The learning rate was set to 0.0001, optimizer used was Adam and Sparse Categorical Cross Entropy loss function was used. Prediction layer of 4 classes was used since the number of classes of output is 4.

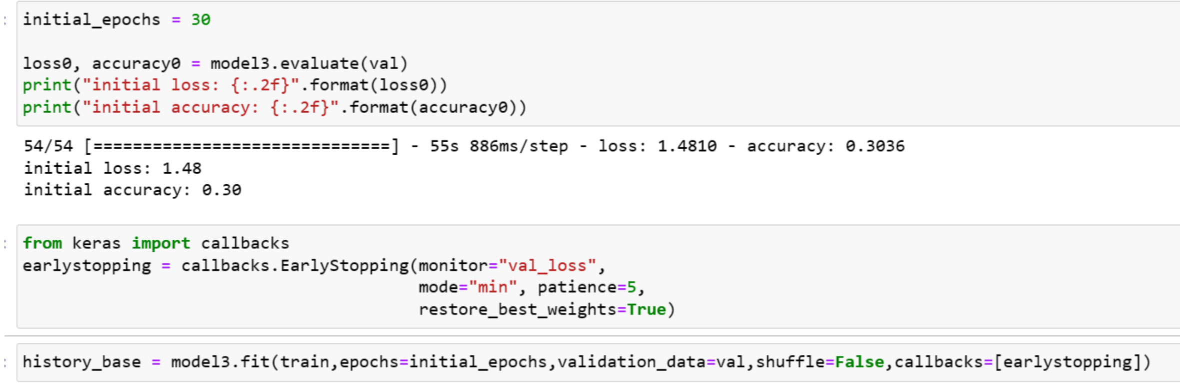


```
inputs = tf.keras.Input(shape=(250, 250, 3))
x = data_augmentation(inputs)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(1280,activation='relu')(x)
outputs = prediction_layer(x)
model3 = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model3.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 3. 21. Adding of layers, pre-processing the input, initializing the model and compiling the model

For avoiding overfitting as well as for saving time, Early Stopping was included. Under this, validation loss was monitored and patience was set to 5. So, after a particular epoch, if for 5 continuous epochs validations would be greater, the model training would stop.



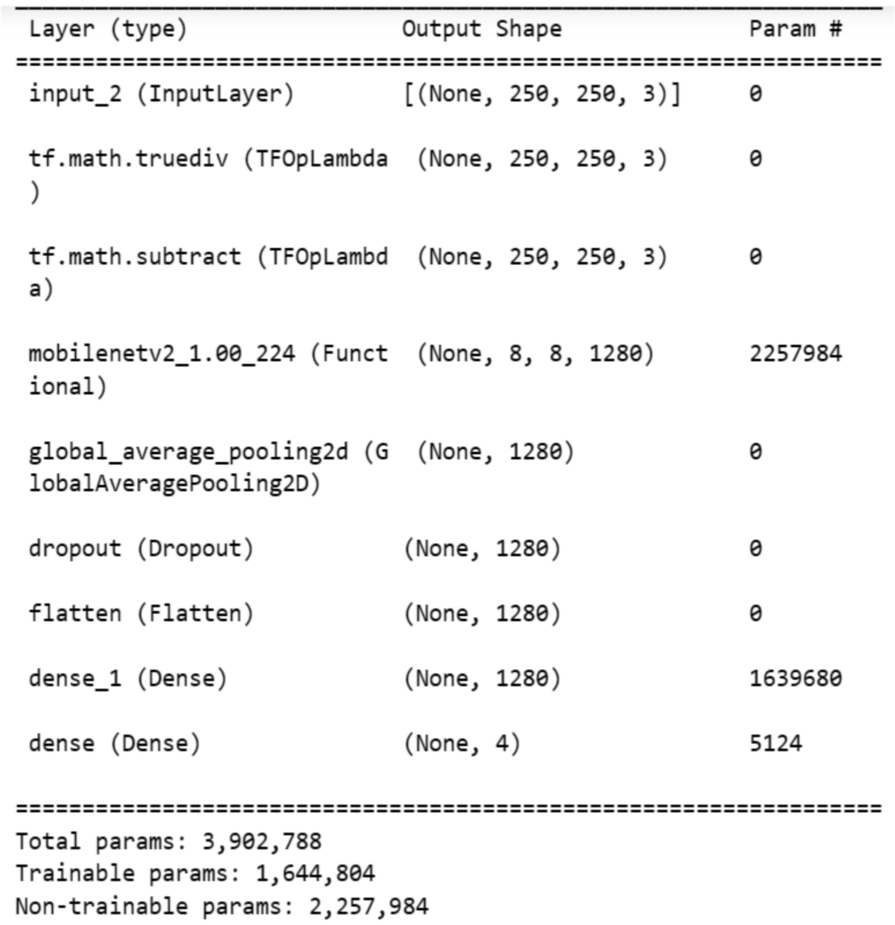
```
initial_epochs = 30

loss0, accuracy0 = model3.evaluate(val)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
54/54 [==============================] - 55s 886ms/step - loss: 1.4810 - accuracy: 0.3036
initial loss: 1.48
initial accuracy: 0.30
```

```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                        mode="min", patience=5,
                                        restore_best_weights=True)
```

```
history_base = model3.fit(train,epochs=initial_epochs,validation_data=val,shuffle=False,callbacks=[earlystopping])
```

Figure 3. 22. Inclusion of Early Stopping and Model Fitting

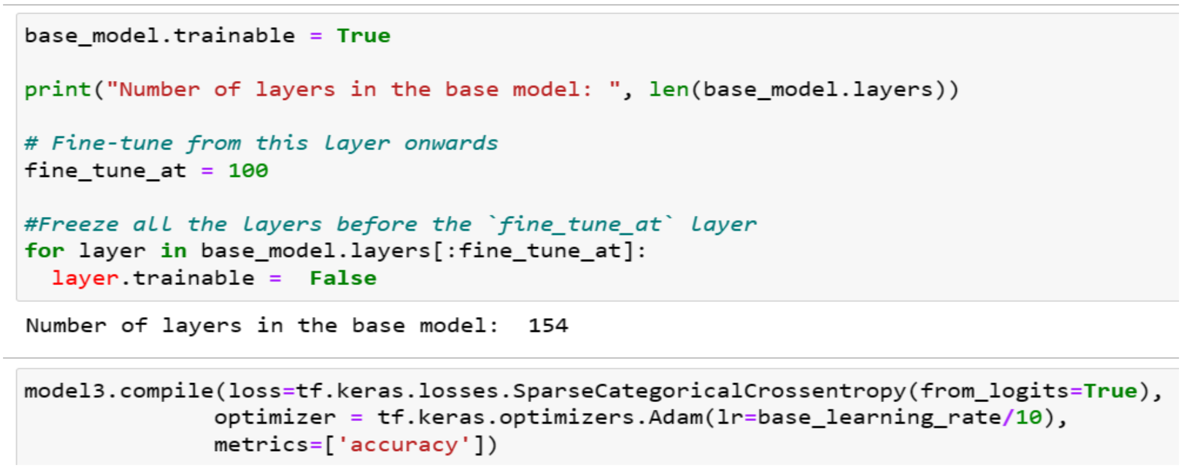


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 250, 250, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 250, 250, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 flatten (Flatten)           (None, 1280)              0

 dense_1 (Dense)             (None, 1280)              1639680

 dense (Dense)               (None, 4)                 5124

=================================================================
Total params: 3,902,788
Trainable params: 1,644,804
Non-trainable params: 2,257,984
```

Figure 3. 23. Summary of final model including MobileNetV2 as well as manually added layers

Now, after using pre-trained model of MobileNetV2, it was decided to fine-tune the model. Fine tuning is always a good idea to enhance the performance of a deep learning model. Fine tuning means some layers are made trainable, i.e. pre-defined weights are not used. Some layers are freshly trained. This sometimes fires back because model gets susceptible to difficulties of training.

Out of 154 layers of the MobileNetV2 model, last 54 layers were made trainable and were fine-tuned.

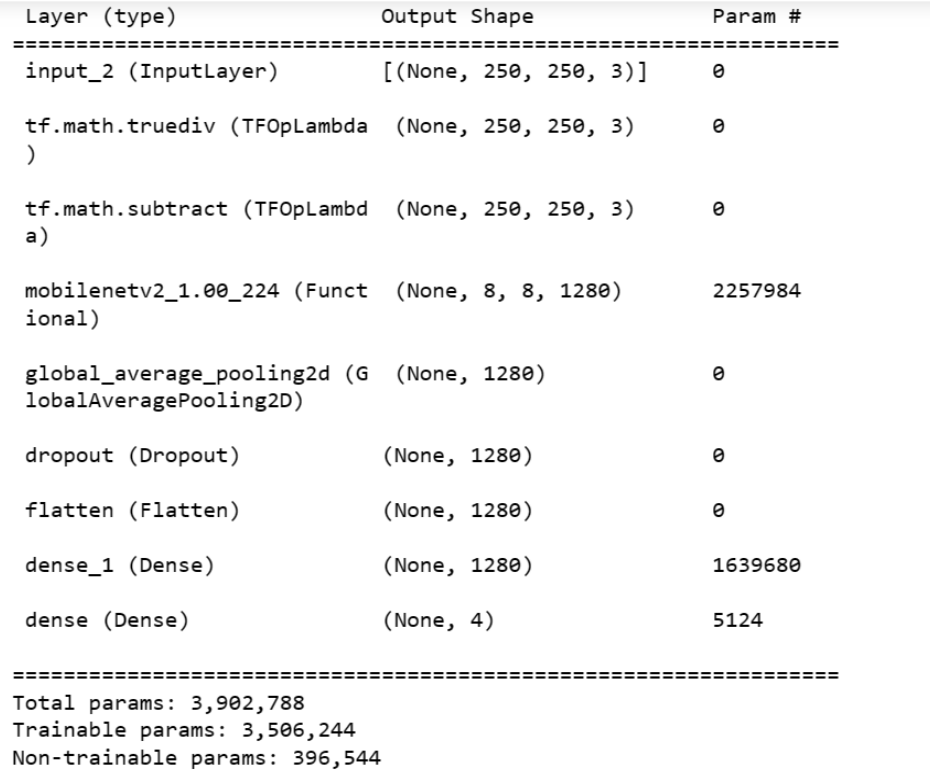


```
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

#Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  154
```

```
model3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               optimizer = tf.keras.optimizers.Adam(lr=base_learning_rate/10),
               metrics=['accuracy'])
```

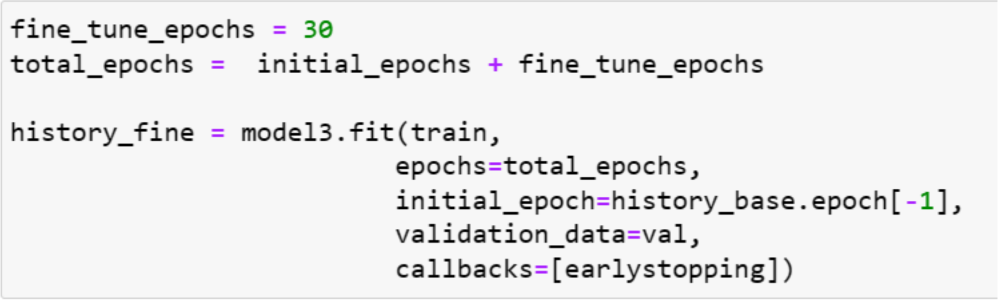
Figure 3. 24. Fine tuning of MobileNet model



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 250, 250, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 250, 250, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 flatten (Flatten)           (None, 1280)              0

 dense_1 (Dense)             (None, 1280)              1639680

 dense (Dense)               (None, 4)                 5124

=================================================================
Total params: 3,902,788
Trainable params: 3,506,244
Non-trainable params: 396,544
```

Figure 3. 25. Summary of fine-tuned model

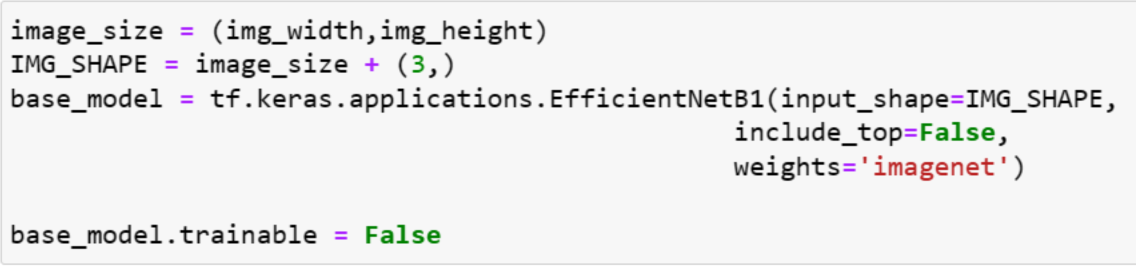
For fine-tuning 30 additional epochs were added. Due to early stopping feature included, fine tuning stopped after total 42 epochs which means 12 epochs of fine tuning ran.



```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model3.fit(train,
                         epochs=total_epochs,
                         initial_epoch=history_base.epoch[-1],
                         validation_data=val,
                         callbacks=[earlystopping])
```

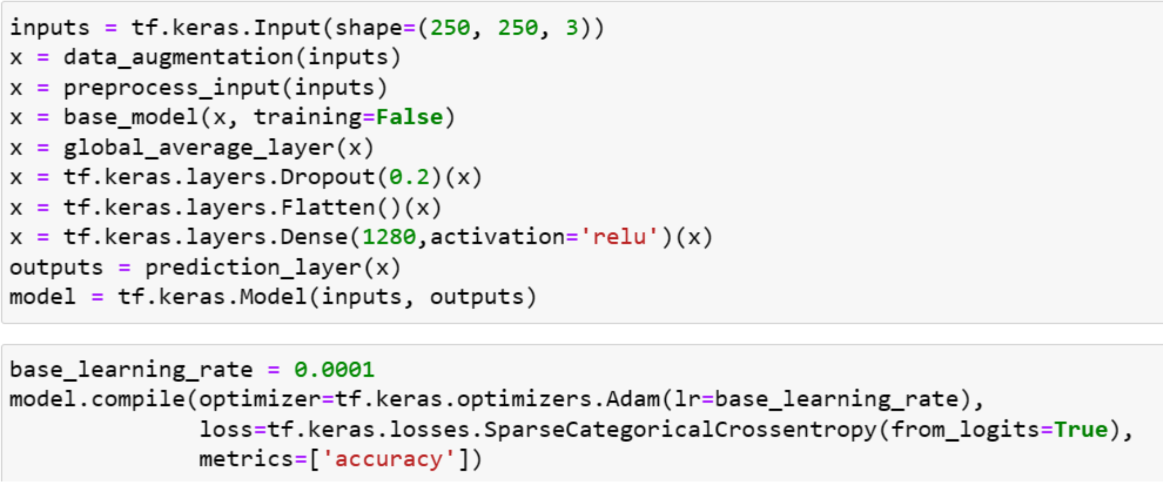
Figure 3. 26. Epochs running for fine-tuned model

Now, EfficientNetB1, another light-weight architecture was chosen to be implemented. Its implementation details and snapshots follow.



```
image_size = (img_width,img_height)
IMG_SHAPE = image_size + (3,)
base_model = tf.keras.applications.EfficientNetB1(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')

base_model.trainable = False
```

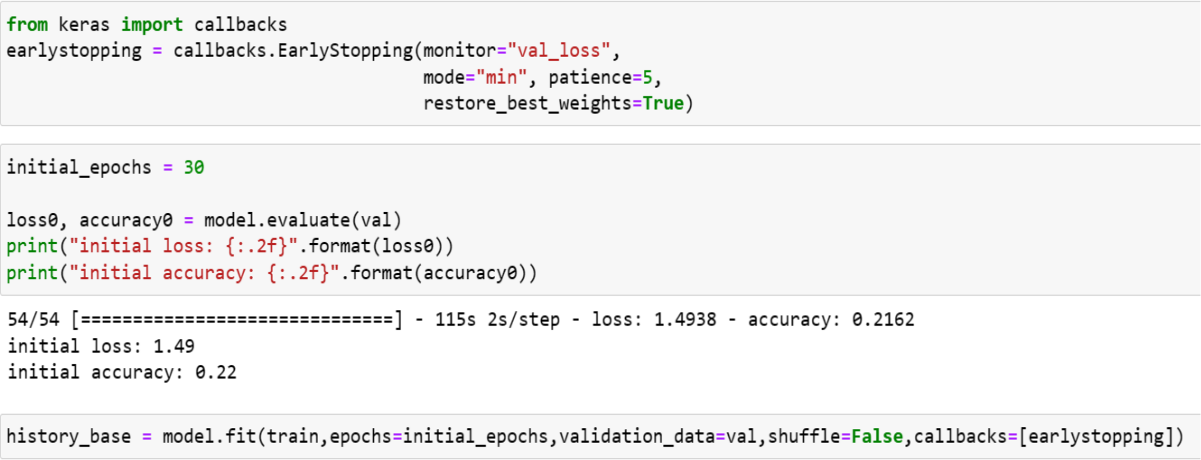
Figure 3. 27. Loading of pre-trained EfficientNetB1



```
inputs = tf.keras.Input(shape=(250, 250, 3))
x = data_augmentation(inputs)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(1280,activation='relu')(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 3. 28. Adding of layers and Model compilation



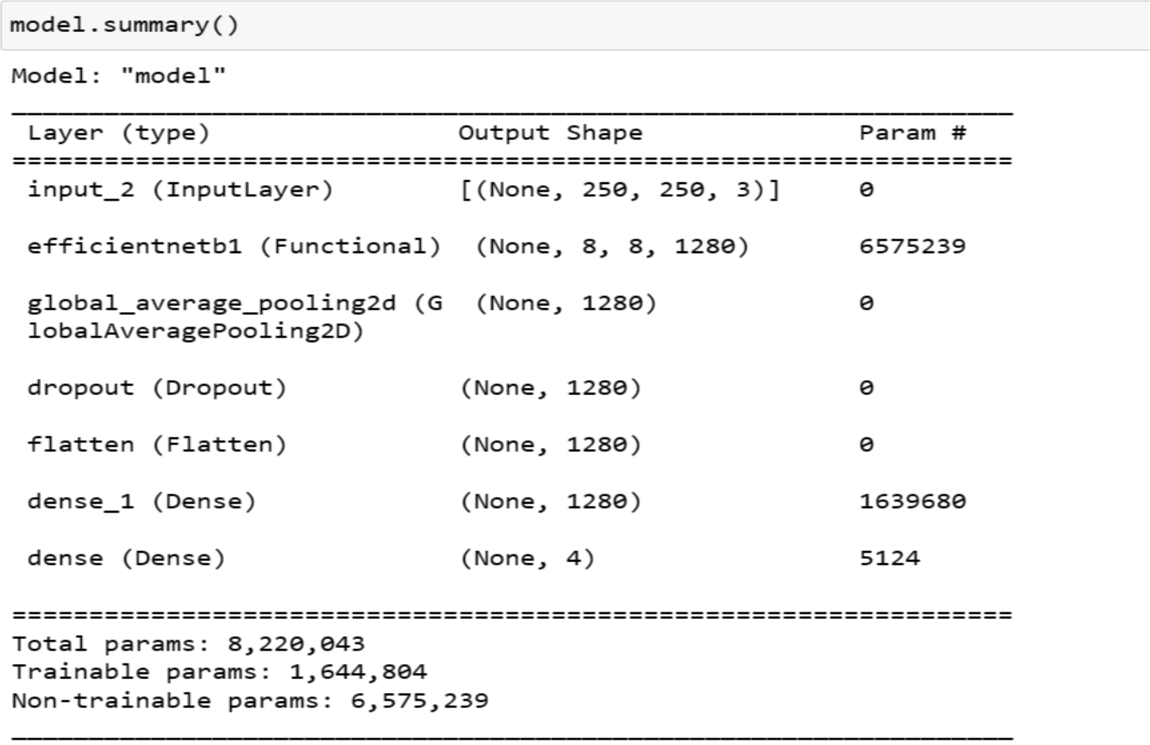
```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                        mode="min", patience=5,
                                        restore_best_weights=True)
```

```
initial_epochs = 30

loss0, accuracy0 = model.evaluate(val)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
54/54 [==============================] - 115s 2s/step - loss: 1.4938 - accuracy: 0.2162
initial loss: 1.49
initial accuracy: 0.22
```

```
history_base = model.fit(train,epochs=initial_epochs,validation_data=val,shuffle=False,callbacks=[earlystopping])
```

Figure 3.29. Training of EfficientNet model

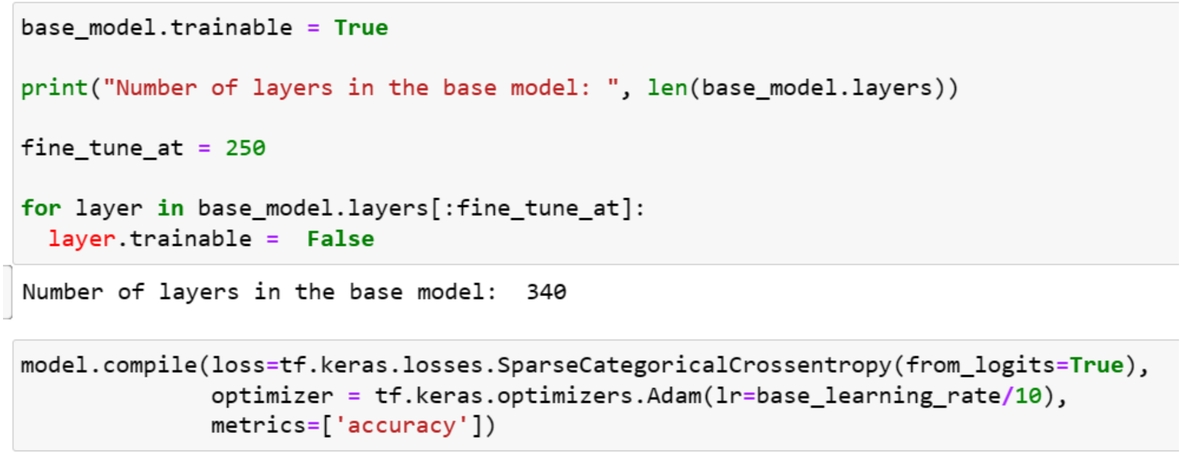


```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0         
                                                                 
 efficientnetb1 (Functional)  (None, 8, 8, 1280)       6575239   
                                                                 
 global_average_pooling2d (G  (None, 1280)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout (Dropout)           (None, 1280)              0         
                                                                 
 flatten (Flatten)           (None, 1280)              0         
                                                                 
 dense_1 (Dense)             (None, 1280)              1639680   
                                                                 
 dense (Dense)               (None, 4)                 5124      
                                                                 
=================================================================
Total params: 8,220,043
Trainable params: 1,644,804
Non-trainable params: 6,575,239
_________________________________________________________________
```

Figure 3.30. EfficientNetB1 model summary

Out of 340 layers of the EfficientNetB1 pre-trained model, the last 90 layers were fine-tuned. For fine-tuning 30 additional epochs were added. Due to early stopping feature included, fine tuning stopped after total 39 epochs which means 9 epochs of fine tuning ran.

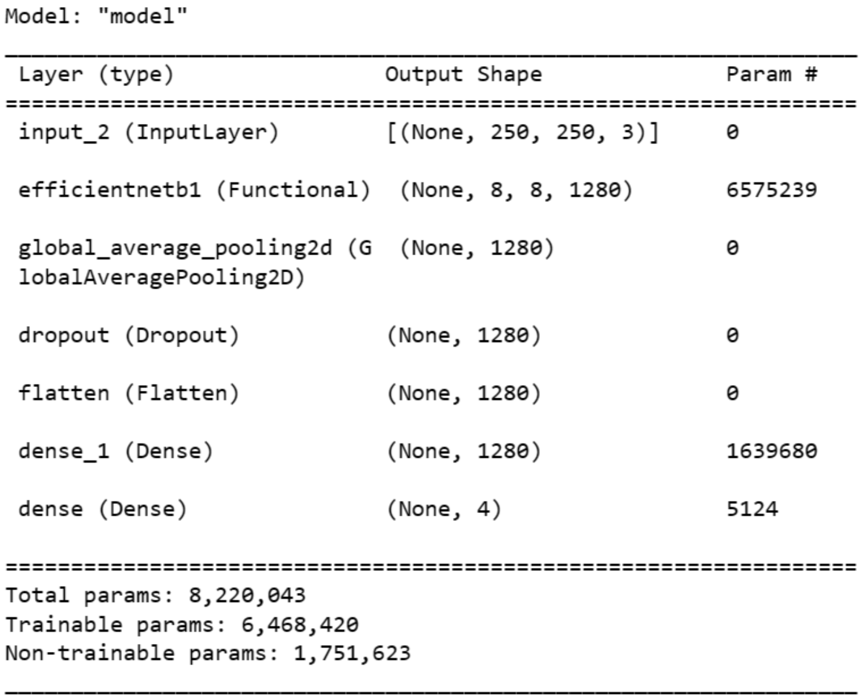


```
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))

fine_tune_at = 250

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  340
```

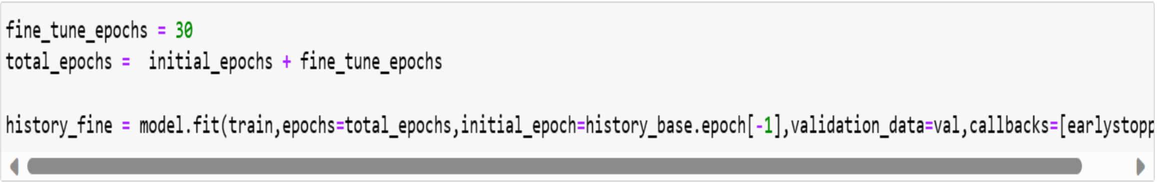
```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.Adam(lr=base_learning_rate/10),
              metrics=['accuracy'])
```

Figure 3.31. Fine tuning of EfficientNetB1



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0

 efficientnetb1 (Functional)  (None, 8, 8, 1280)       6575239

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 flatten (Flatten)           (None, 1280)              0

 dense_1 (Dense)             (None, 1280)              1639680

 dense (Dense)               (None, 4)                 5124

=================================================================
Total params: 8,220,043
Trainable params: 6,468,420
Non-trainable params: 1,751,623
_________________________________________________________________
```

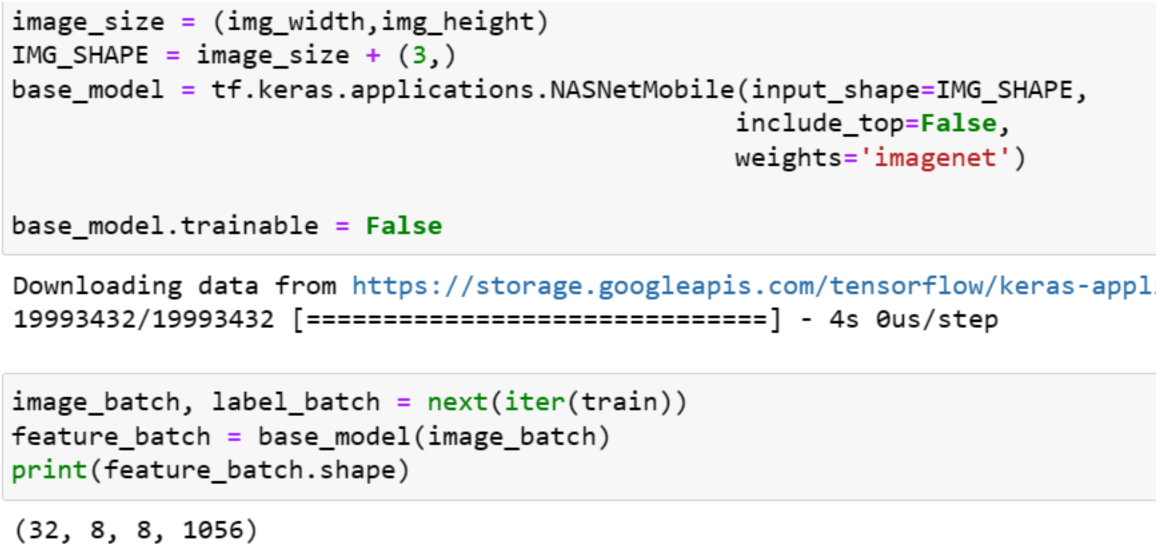
Figure 3.32. Summary of fine-tuned model



```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train,epochs=total_epochs,initial_epoch=history_base.epoch[-1],validation_data=val,callbacks=[earlystopp
```

Figure 3.33. Fine tuning epochs

A very light-weight architecture, NASNetMobile, was chosen to be utilized for the purpose of brain tumor classification. Its pre-trained version was downloaded without the top layer. Also, all other details remained same as in the previously implemented two models.



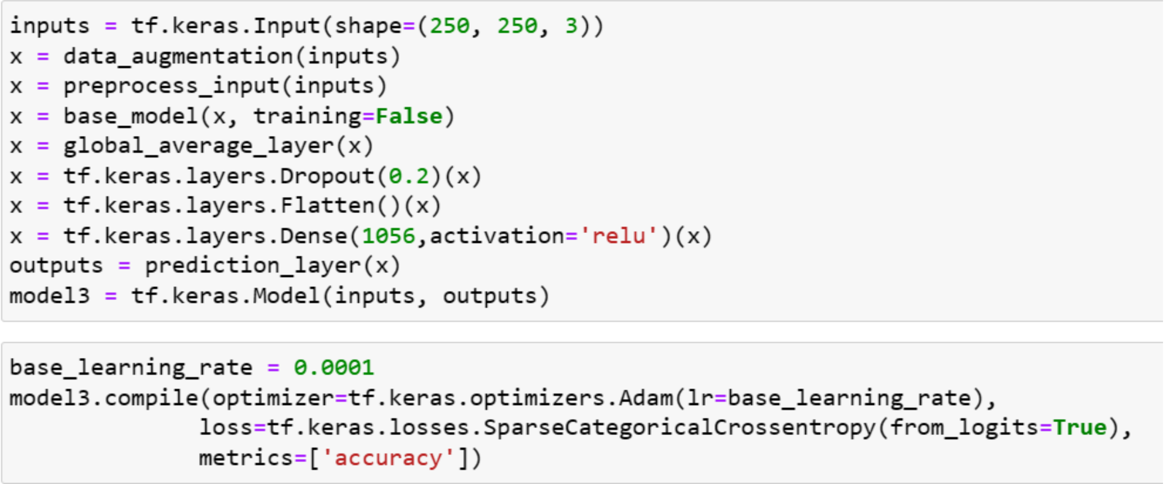
```
image_size = (img_width,img_height)
IMG_SHAPE = image_size + (3,)
base_model = tf.keras.applications.NASNetMobile(input_shape=IMG_SHAPE,
                                                include_top=False,
                                                weights='imagenet')

base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appl
19993432/19993432 [==============================] - 4s 0us/step
```

```
image_batch, label_batch = next(iter(train))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 8, 8, 1056)
```

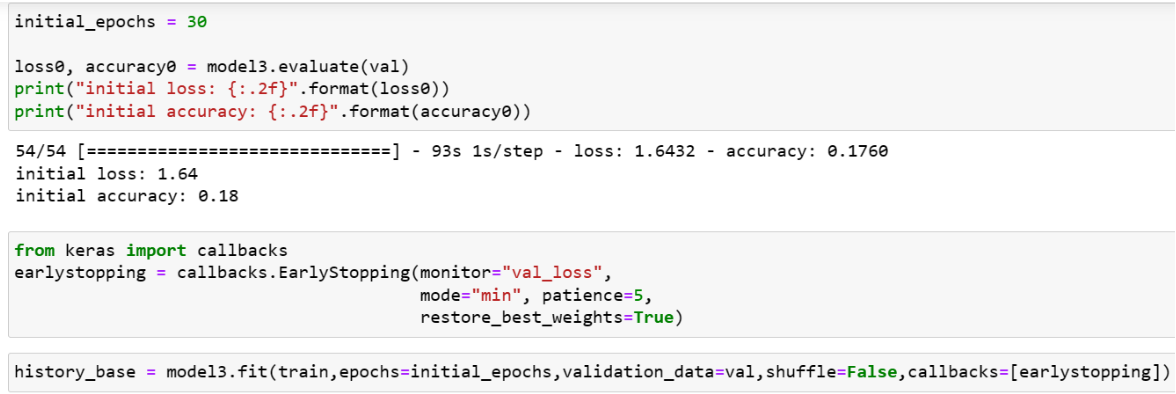
Figure 3.34. Loading pre-trained NASNetMobile



```
inputs = tf.keras.Input(shape=(250, 250, 3))
x = data_augmentation(inputs)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(1056,activation='relu')(x)
outputs = prediction_layer(x)
model3 = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model3.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 3.35. Adding of layers and NASNetMobile model compilation



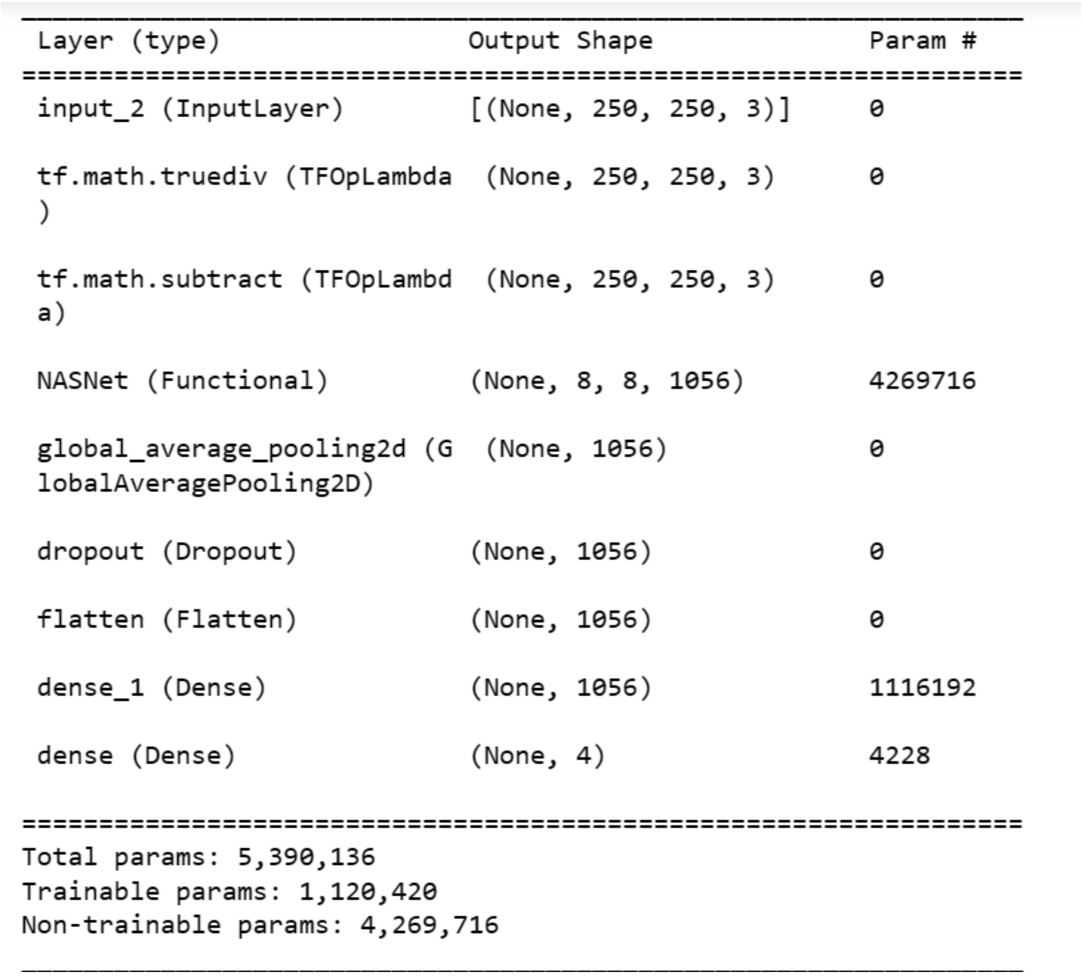
```
initial_epochs = 30

loss0, accuracy0 = model3.evaluate(val)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
54/54 [==============================] - 93s 1s/step - loss: 1.6432 - accuracy: 0.1760
initial loss: 1.64
initial accuracy: 0.18
```

```
from keras import callbacks
earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                        mode="min", patience=5,
                                        restore_best_weights=True)
```

```
history_base = model3.fit(train,epochs=initial_epochs,validation_data=val,shuffle=False,callbacks=[earlystopping])
```

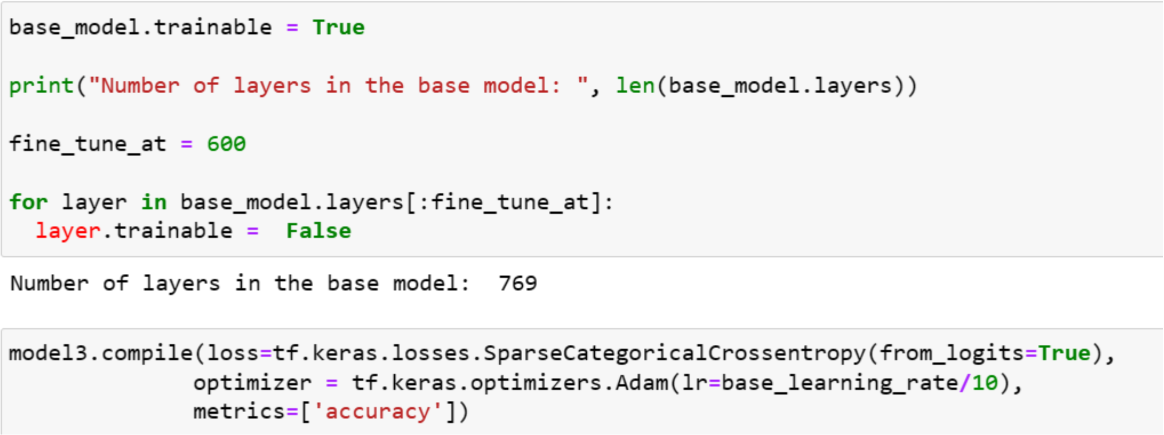
Figure 3.36. Fitting of the NASNetMobile model



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 250, 250, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 250, 250, 3)      0
 a)

 NASNet (Functional)         (None, 8, 8, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 flatten (Flatten)           (None, 1056)              0

 dense_1 (Dense)             (None, 1056)              1116192

 dense (Dense)               (None, 4)                 4228

=================================================================
Total params: 5,390,136
Trainable params: 1,120,420
Non-trainable params: 4,269,716
_________________________________________________________________
```

Figure 3.37. Summary of the NASNetMobile model

The total number of layers in the NASNetMobile model are 769. Out of this, the last 169 layers were made trainable and were fine tuned.

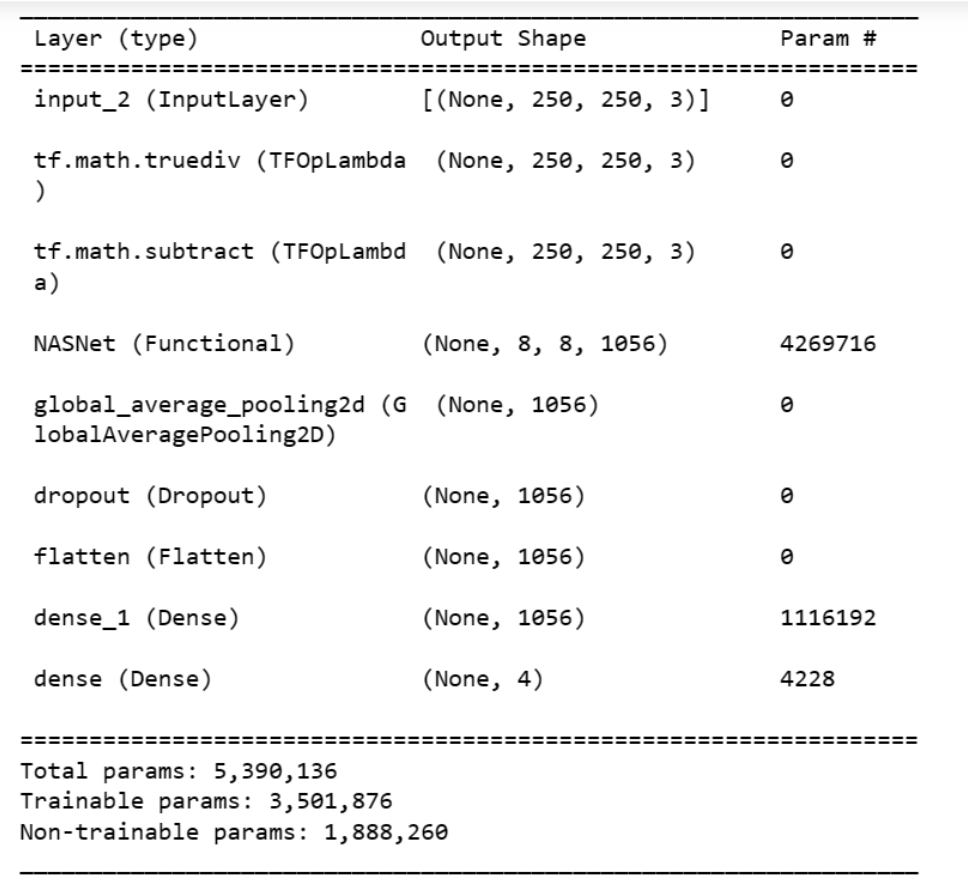


```
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))

fine_tune_at = 600

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  769
```

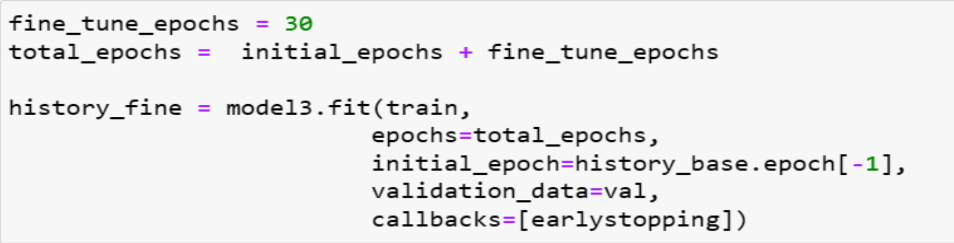
```
model3.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.Adam(lr=base_learning_rate/10),
              metrics=['accuracy'])
```

Figure 3.38. Fine tuning and compilation of NASNetMobile



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 250, 250, 3)]     0

 tf.math.truediv (TFOpLambda  (None, 250, 250, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 250, 250, 3)      0
 a)

 NASNet (Functional)         (None, 8, 8, 1056)        4269716

 global_average_pooling2d (G  (None, 1056)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1056)              0

 flatten (Flatten)           (None, 1056)              0

 dense_1 (Dense)             (None, 1056)              1116192

 dense (Dense)               (None, 4)                 4228

=================================================================
Total params: 5,390,136
Trainable params: 3,501,876
Non-trainable params: 1,888,260
_________________________________________________________________
```

Figure 3.39. Summary of the fine-tuned model



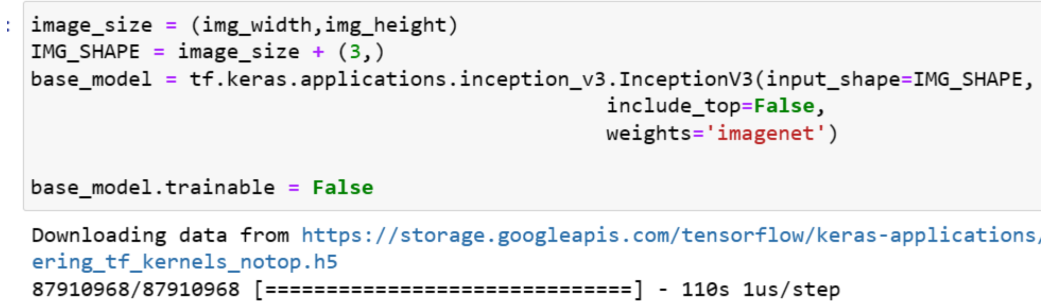
```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model3.fit(train,
                         epochs=total_epochs,
                         initial_epoch=history_base.epoch[-1],
                         validation_data=val,
                         callbacks=[earlystopping])
```

Figure 3.40. Epochs of fine tuning

For fine-tuning 30 additional epochs were added. Due to early stopping feature included, fine tuning stopped after total 43 epochs which means 13 epochs of fine tuning ran.

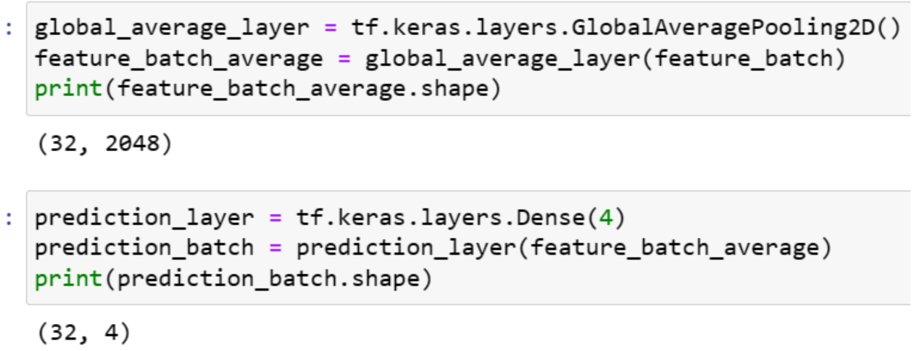
Next, while exploring the documentation for some light-weight architectures, Inception V3 was found to be applicable for the purpose of this project. Although its size may seem to be a bit larger as compared to other architectures used in this project, its time consumption is quite good and lesser than some other models used in this project. Also, from the previous works, it was clear that Inception V3 can be of good use for this project.

Below are the screenshots for the implementation of Inception V3 model, which was added with additional layers after replacement of the top layer of the pre-trained model.



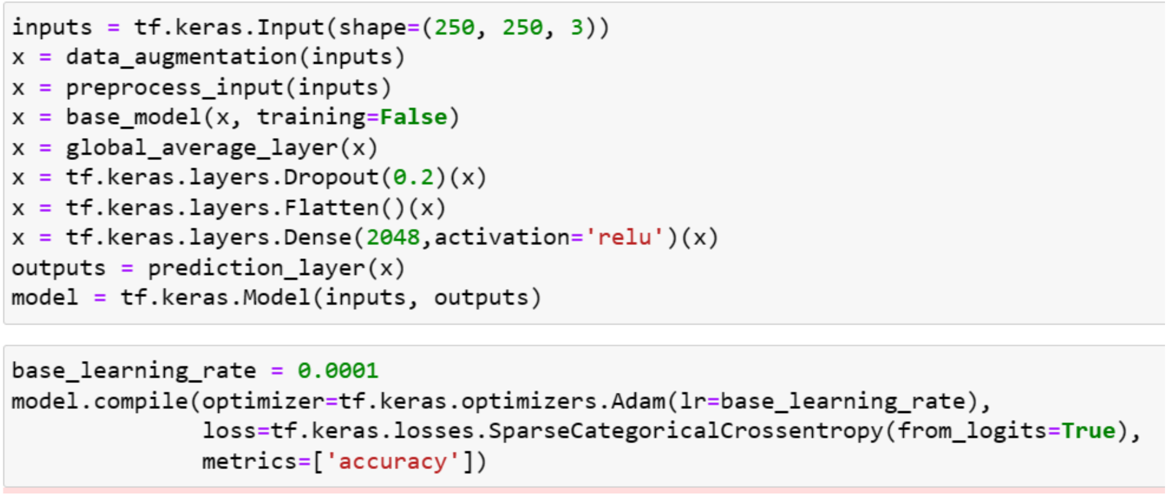
```
: image_size = (img_width,img_height)
  IMG_SHAPE = image_size + (3,)
  base_model = tf.keras.applications.inception_v3.InceptionV3(input_shape=IMG_SHAPE,
                                                             include_top=False,
                                                             weights='imagenet')


  base_model.trainable = False

  Downloading data from https://storage.googleapis.com/tensorflow/keras-applications,
  ering_tf_kernels_notop.h5
  87910968/87910968 [==============================] - 110s 1us/step
```

Figure 3. 41. Loading of InceptionV3 model



```
: global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
  feature_batch_average = global_average_layer(feature_batch)
  print(feature_batch_average.shape)

  (32, 2048)

: prediction_layer = tf.keras.layers.Dense(4)
  prediction_batch = prediction_layer(feature_batch_average)
  print(prediction_batch.shape)

  (32, 4)
```

Figure 3.42. Addition of Custom layers on top of pre-trained model



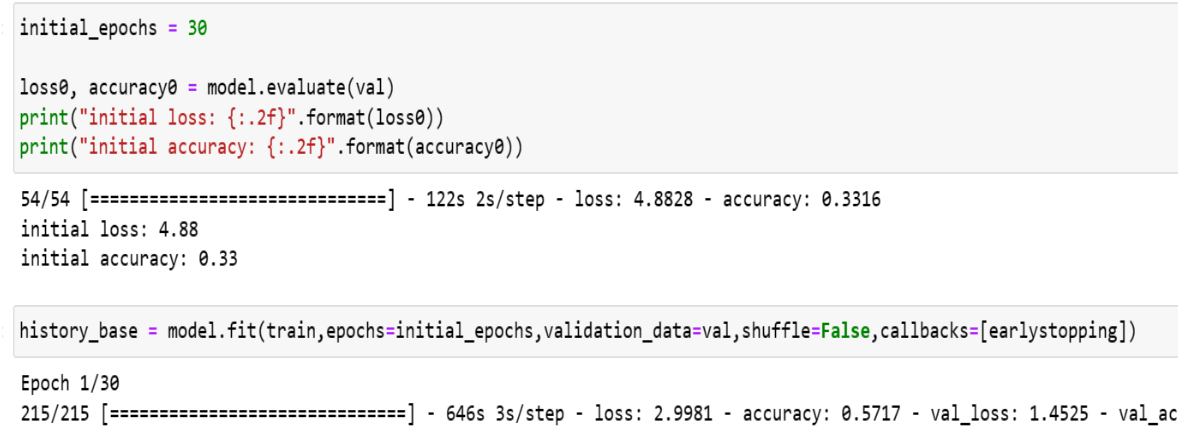
```
inputs = tf.keras.Input(shape=(250, 250, 3))
x = data_augmentation(inputs)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(2048,activation='relu')(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 3. 43. Compilation of the model using Adam cost function and Sparse Categorical Cross Entropy loss function

30 initial epochs were initialized for the training of Inception V3 model. Early Stopping feature was also included on the basis of Validation Loss. This means, if validation loss increases for 5 consecutive epochs, training stops at the fifth consecutive step and the training gets reverted to the step before the start of increasing validation loss.

Inception V3 was the only case in which the initial training stopped before the completion of 30 epochs at 20 epochs.



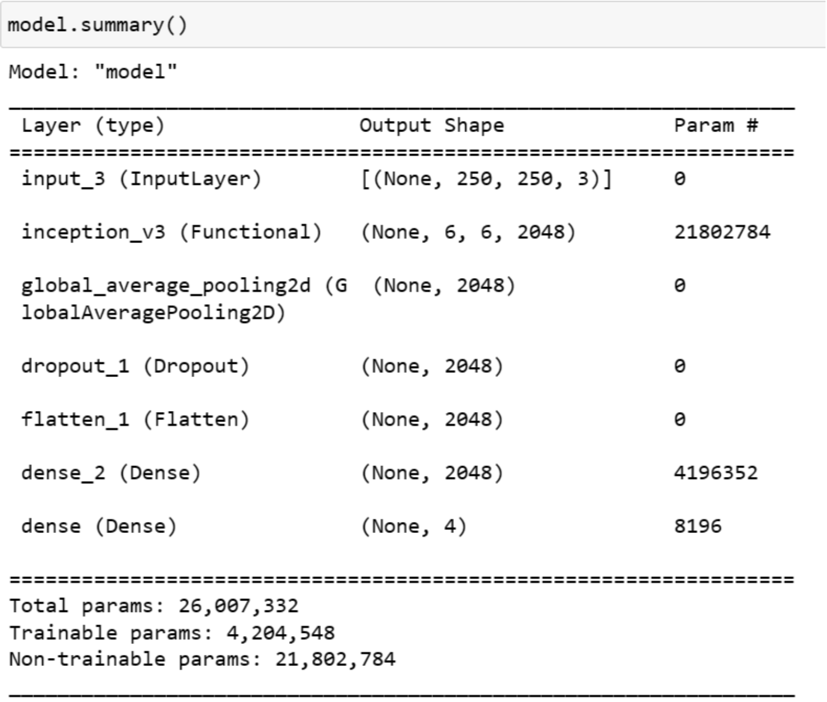
```
initial_epochs = 30

loss0, accuracy0 = model.evaluate(val)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
54/54 [==============================] - 122s 2s/step - loss: 4.8828 - accuracy: 0.3316
initial loss: 4.88
initial accuracy: 0.33
```

```
history_base = model.fit(train,epochs=initial_epochs,validation_data=val,shuffle=False,callbacks=[earlystopping])
```

```
Epoch 1/30
215/215 [==============================] - 646s 3s/step - loss: 2.9981 - accuracy: 0.5717 - val_loss: 1.4525 - val_ac
```

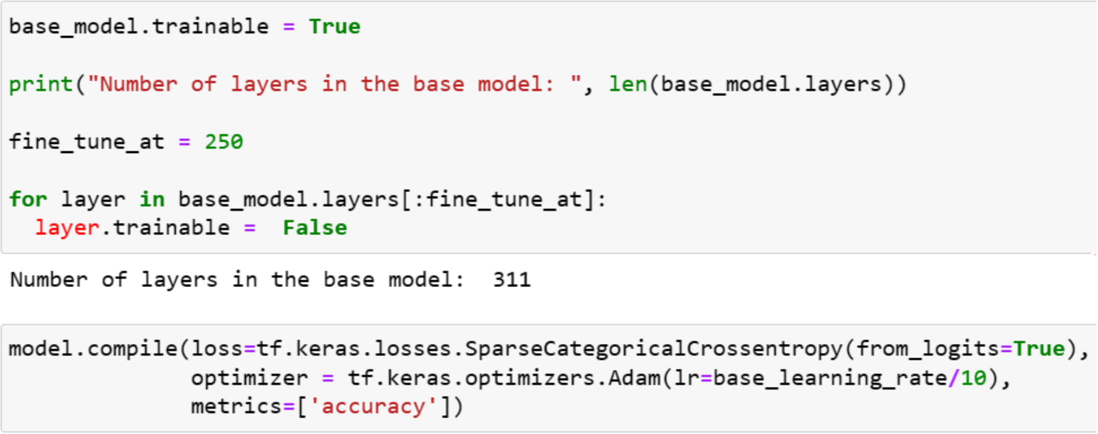
Figure 3.44. Training of 30 epochs of InceptionV3



```
model.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_3 (InputLayer)        [(None, 250, 250, 3)]     0         
                                                                 
 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784  
                                                                 
 global_average_pooling2d (G  (None, 2048)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout_1 (Dropout)         (None, 2048)              0         
                                                                 
 flatten_1 (Flatten)         (None, 2048)              0         
                                                                 
 dense_2 (Dense)             (None, 2048)              4196352   
                                                                 
 dense (Dense)               (None, 4)                 8196      
                                                                 
=================================================================
Total params: 26,007,332
Trainable params: 4,204,548
Non-trainable params: 21,802,784
_________________________________________________________________
```

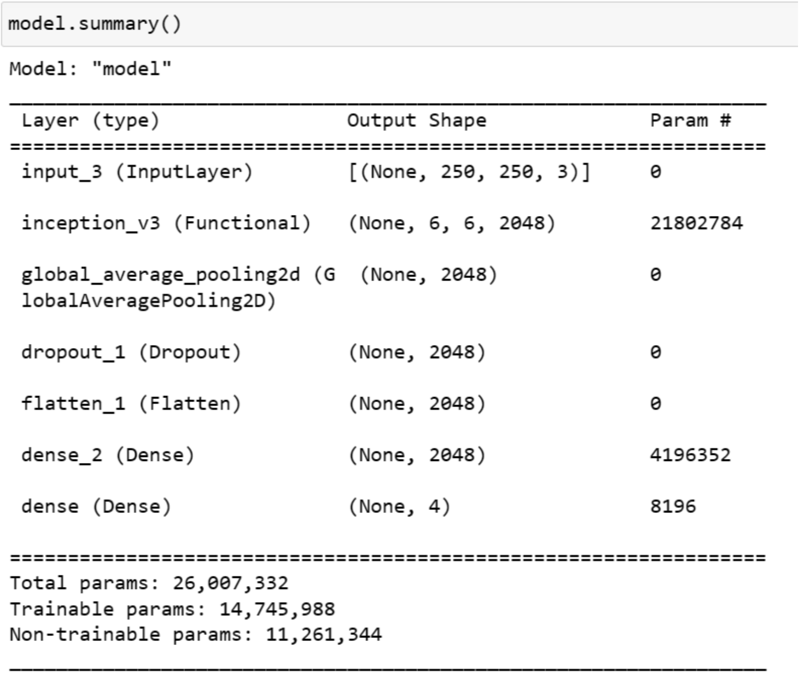
Figure 3. 45. Summary of the InceptionV3 model

There were total 311 layers in the model. For fine tuning, last 61 layers were made trainable. 20 additional epochs were initialized for fine tuning. Since initial training had stopped at 20 epochs itself, the fine tuning began at 21st epoch and continued till 34 epochs were completed until getting stopped due to early stopping feature.



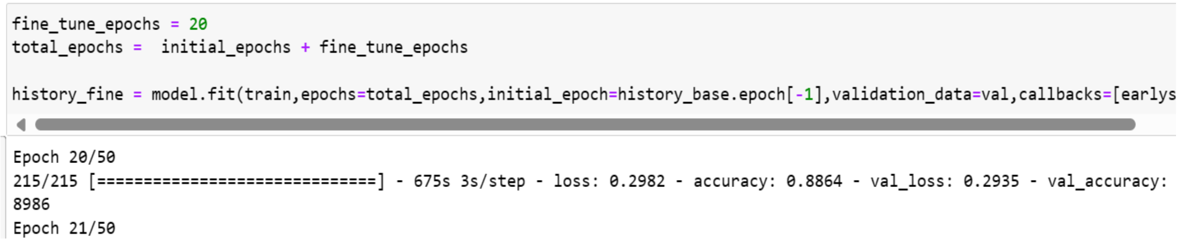
```
base_model.trainable = True

print("Number of layers in the base model: ", len(base_model.layers))

fine_tune_at = 250

for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False

Number of layers in the base model:  311

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.Adam(lr=base_learning_rate/10),
              metrics=['accuracy'])
```

Figure 3.46. Fine tuning initialization of last 61 layers of InceptionV3



```
model.summary()

Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 250, 250, 3)]     0

 inception_v3 (Functional)   (None, 6, 6, 2048)        21802784

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 2048)              0

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 2048)              4196352

 dense (Dense)               (None, 4)                 8196

=================================================================
Total params: 26,007,332
Trainable params: 14,745,988
Non-trainable params: 11,261,344
_________________________________________________________________
```

Figure 3.47. Summary of the fine-tuned version of InceptionV3 model

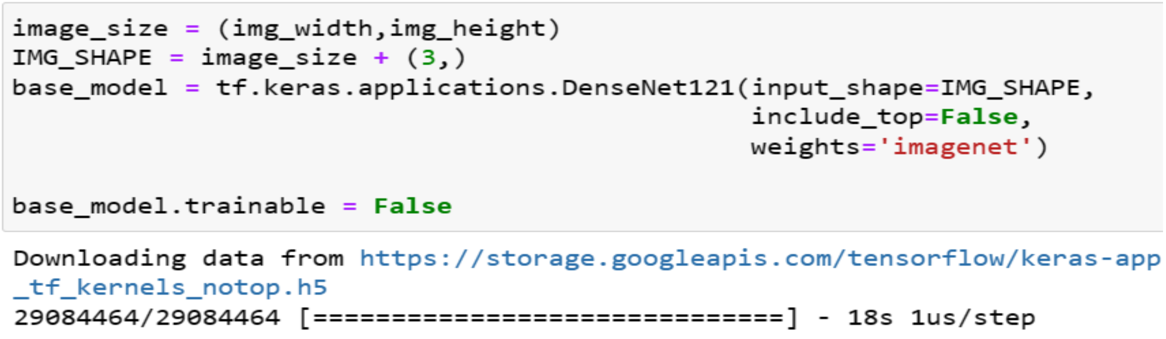


```
fine_tune_epochs = 20
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train,epochs=total_epochs,initial_epoch=history_base.epoch[-1],validation_data=val,callbacks=[earlys
```

```
Epoch 20/50
215/215 [==============================] - 675s 3s/step - loss: 0.2982 - accuracy: 0.8864 - val_loss: 0.2935 - val_accuracy:
8986
Epoch 21/50
```

Figure 3.48. Fine tuning epochs of InceptionV3

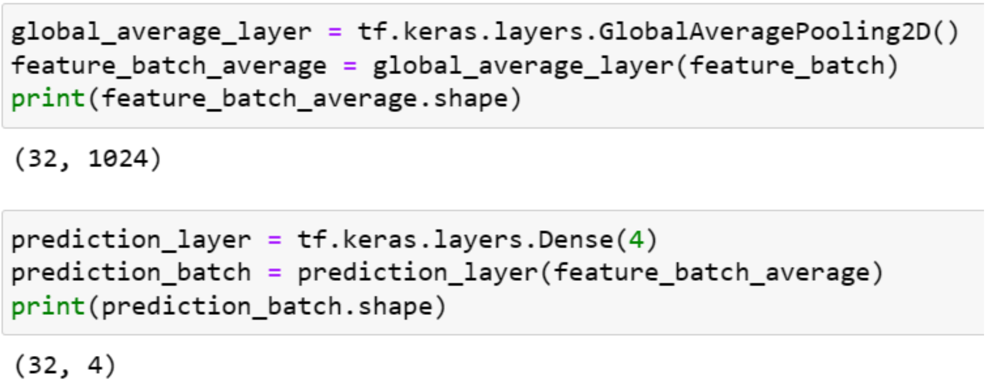
After getting encouraging results of using light-weight architectures, a fifth light-weight model, DenseNet121 was chosen to be implemented for the purpose of this project. This architecture is specifically much optimized in terms of space consumption as well as performance for various deep learning applications.



```
image_size = (img_width,img_height)
IMG_SHAPE = image_size + (3,)
base_model = tf.keras.applications.DenseNet121(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')


base_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-app
_tf_kernels_notop.h5
29084464/29084464 [==============================] - 18s 1us/step
```

Figure 3.49. Loading of DenseNet121 model



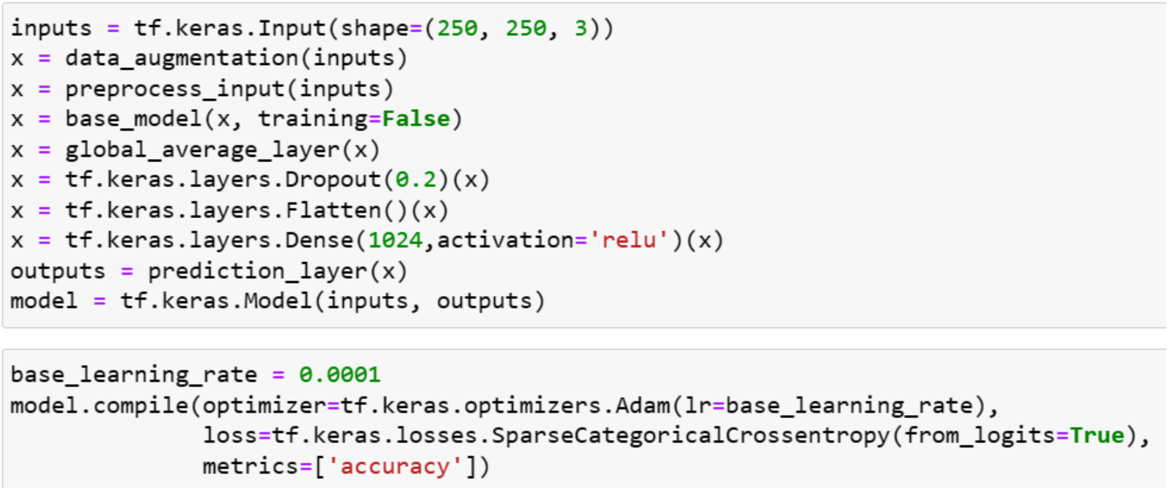
```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1024)
```

```
prediction_layer = tf.keras.layers.Dense(4)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 4)
```

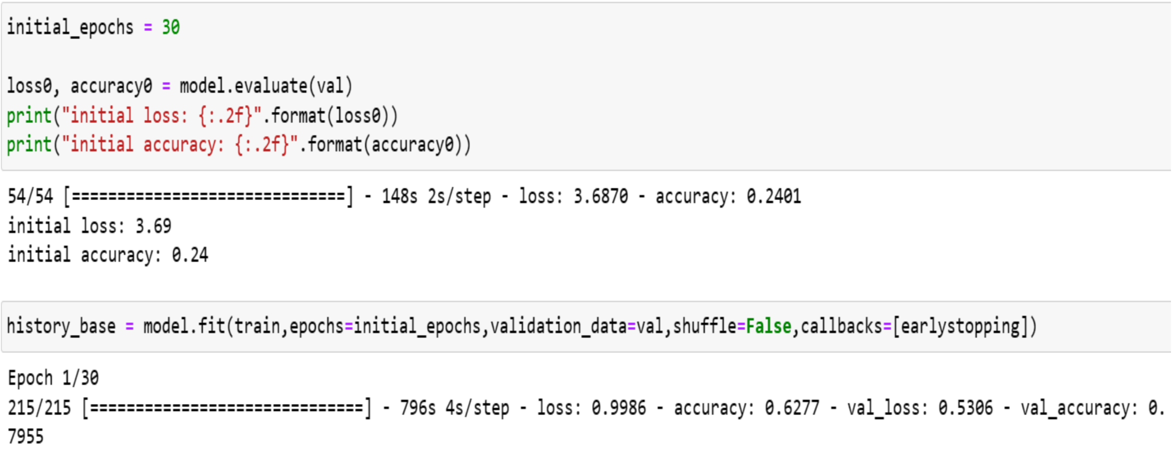
Figure 3.50. Adding of custom layers on DenseNet121



```
inputs = tf.keras.Input(shape=(250, 250, 3))
x = data_augmentation(inputs)
x = preprocess_input(inputs)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(1024,activation='relu')(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)
```

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

Figure 3.51. Compilation of DenseNet121 model



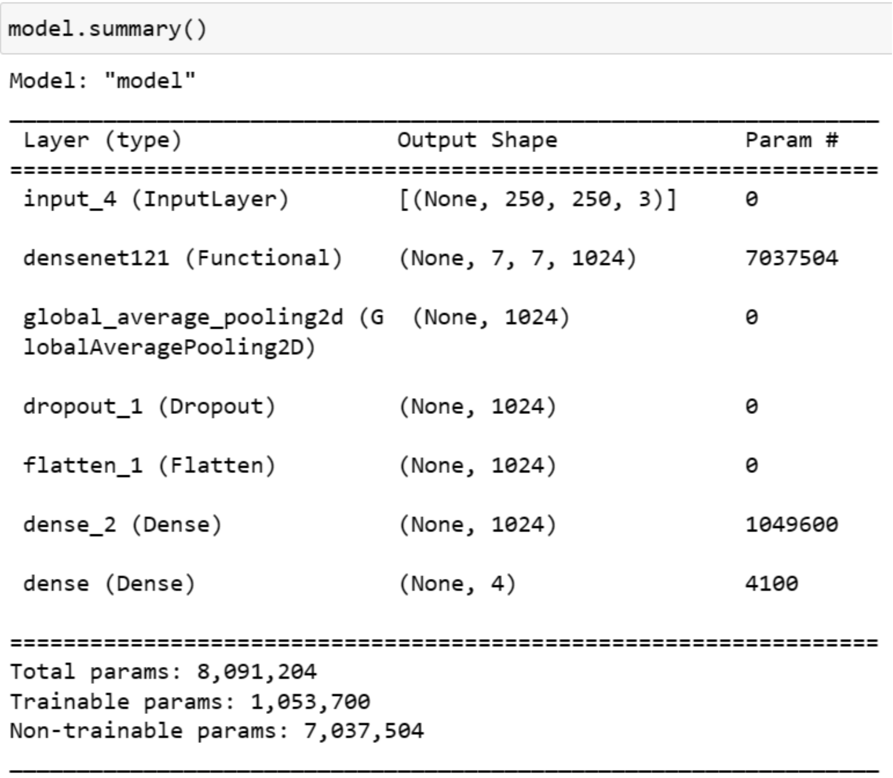
```
initial_epochs = 30

loss0, accuracy0 = model.evaluate(val)
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
54/54 [==============================] - 148s 2s/step - loss: 3.6870 - accuracy: 0.2401
initial loss: 3.69
initial accuracy: 0.24
```

```
history_base = model.fit(train,epochs=initial_epochs,validation_data=val,shuffle=False,callbacks=[earlystopping])
```

```
Epoch 1/30
215/215 [==============================] - 796s 4s/step - loss: 0.9986 - accuracy: 0.6277 - val_loss: 0.5306 - val_accuracy: 0.
7955
```

Figure 3.52. Training of DenseNet121 model using 30 epochs

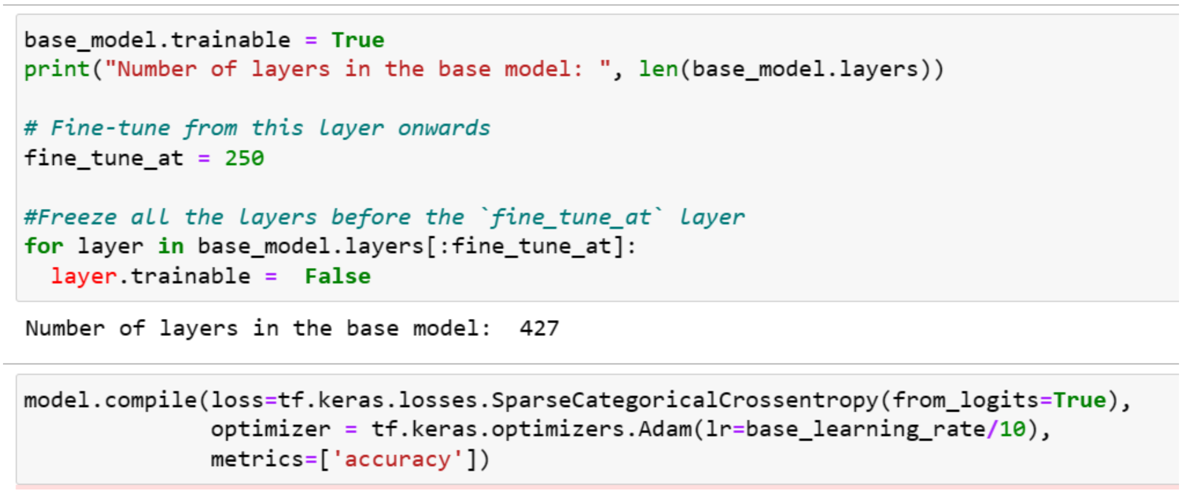


```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_4 (InputLayer)        [(None, 250, 250, 3)]     0         
                                                                 
 densenet121 (Functional)    (None, 7, 7, 1024)        7037504   
                                                                 
 global_average_pooling2d (G  (None, 1024)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout_1 (Dropout)         (None, 1024)              0         
                                                                 
 flatten_1 (Flatten)         (None, 1024)              0         
                                                                 
 dense_2 (Dense)             (None, 1024)              1049600   
                                                                 
 dense (Dense)               (None, 4)                 4100      
                                                                 
=================================================================
Total params: 8,091,204
Trainable params: 1,053,700
Non-trainable params: 7,037,504
_________________________________________________________________
```

Figure 3.53. Summary of DenseNet121 model

There were total 427 layers in the DenseNet121 architecture. Fine tuning was done from the layer 250 onwards. This means, the last 177 layers were fine tuned to enhance the performance of the model. Out of additional 30 epochs initialized for fine tuning, 17 epochs ran after which training stopped due to continuous increase in validation loss for 5 epochs starting from 13th epoch.

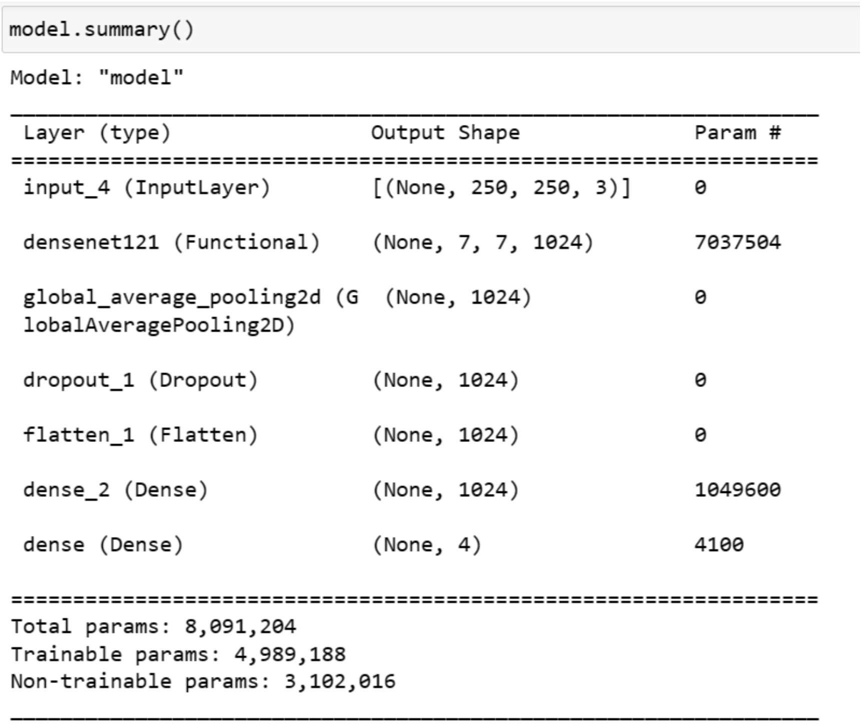


```
base_model.trainable = True
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 250

#Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable =  False
```

```
Number of layers in the base model:  427
```

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.Adam(lr=base_learning_rate/10),
              metrics=['accuracy'])
```

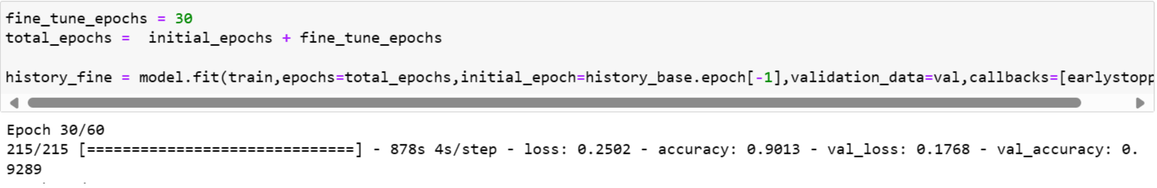
Figure 3. 54. Fine tuning initialization of DenseNet121 model



```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_4 (InputLayer)        [(None, 250, 250, 3)]     0         
                                                                 
 densenet121 (Functional)    (None, 7, 7, 1024)        7037504   
                                                                 
 global_average_pooling2d (G  (None, 1024)             0         
 lobalAveragePooling2D)                                          
                                                                 
 dropout_1 (Dropout)         (None, 1024)              0         
                                                                 
 flatten_1 (Flatten)         (None, 1024)              0         
                                                                 
 dense_2 (Dense)             (None, 1024)              1049600   
                                                                 
 dense (Dense)               (None, 4)                 4100      
                                                                 
=================================================================
Total params: 8,091,204
Trainable params: 4,989,188
Non-trainable params: 3,102,016
_________________________________________________________________
```

Figure 3.55. Summary of Fine-tuned DenseNet121 model



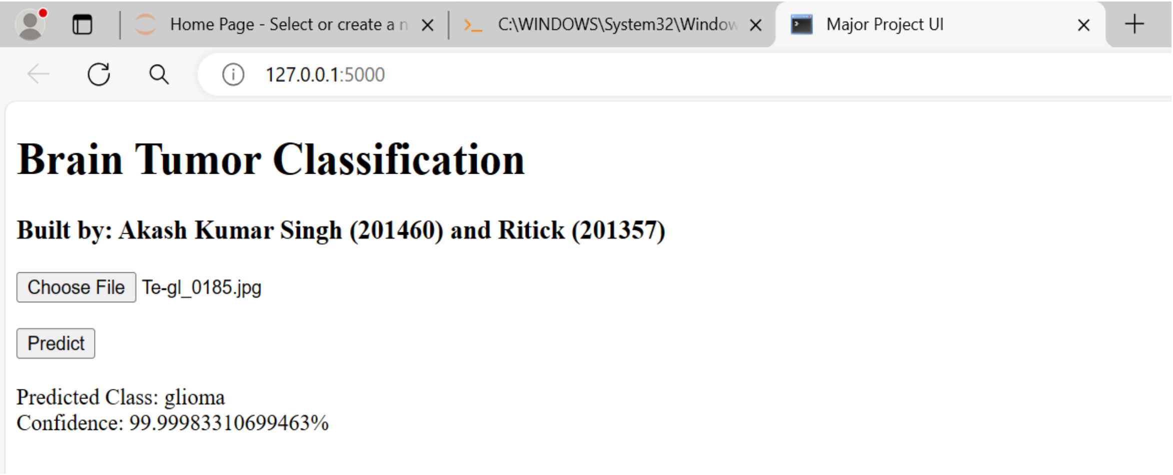
```
fine_tune_epochs = 30
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train,epochs=total_epochs,initial_epoch=history_base.epoch[-1],validation_data=val,callbacks=[earlystopp
```

```
Epoch 30/60
215/215 [==============================] - 878s 4s/step - loss: 0.2502 - accuracy: 0.9013 - val_loss: 0.1768 - val_accuracy: 0.
9289
```

Figure 3.56. Fine tuning epochs for DenseNet121 model

At this point, the model creation and fitting part of the project was complete. Three different architectures were implemented and their training was done on our dataset of brain MR images. All these architectures were pre-trained state of the art light-weight models that consumed very less time and space and at the same time, ensured an uncompromised performance.

For ease of classification and enhanced user experience, a User Interface has been built using Flask application in Python programming language. The UI and prediction using the UI has been shown in Figure 3.57. A UI was also built using tkinter library in Python which is shown in Figure 3.58.



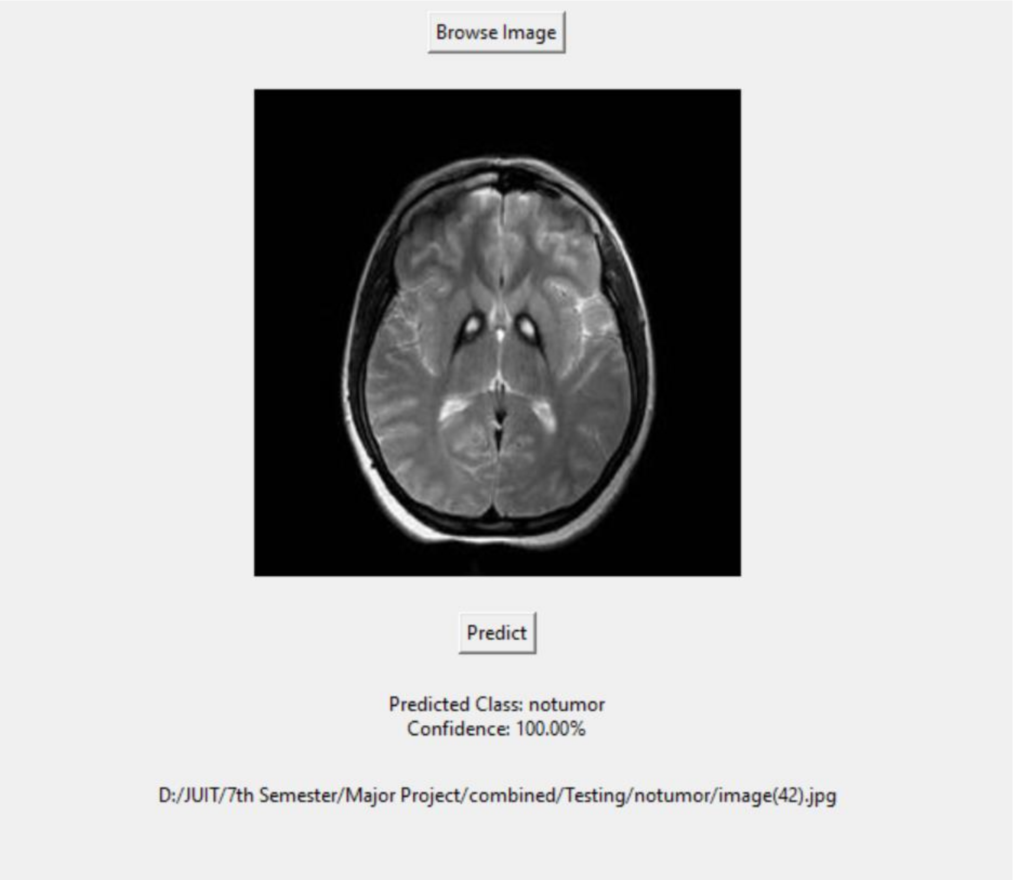
Brain Tumor Classification

Built by: Akash Kumar Singh (201460) and Ritick (201357)

Te-gl_0185.jpg

Predicted Class: glioma
Confidence: 99.99983310699463%

Figure 3.57. Prediction using Flask UI



Predicted Class: notumor
Confidence: 100.00%

D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(42).jpg

Figure 3.58. Prediction using Tkinter UI

3.5 KEY CHALLENGES

The key challenges faced during the course of implementation were majorly related to the difference between the dimensions of the input and the acceptable dimensions of the model's layers. For addressing this issue, the documentation was read carefully for all the architectures and the needful was done accordingly.

Another type of problem that was faced was fair validation accuracy but poor testing accuracy. This problem was addressed by enhancing the dataset and improving the augmentation.

CHAPTER 4: TESTING

4.1 TESTING STRATEGY

Multiple approaches were used for testing the models built for brain tumor classification. Specifically for better testing strategy, the dataset was split into three subsets. The advantage of using both, validation and testing set is that validation set is repeatedly used for checking the performance during multiple experiments. During the course of different experiments, testing set stays isolated. This avoids the case of overfitting because testing set is used only once at the end. If we use testing set repeatedly while model building, it becomes prone to overfitting and we do not get accurate results regarding the performance.

At first, with the pre-trained version of the model, 30 epochs were run and at each epoch, training accuracy, loss, validation loss and validation accuracy were monitored.

Table 4.1 shows the details of monitored parameters for each model at the end of 30 epochs.

Table 4.1. Validation results at the end of 30 epochs

	Loss	Accuracy	Validation Loss	Validation Accuracy
MobileNetV2	0.0248	0.9946	0.0687	0.9808
EfficientNet	0.0263	0.9942	0.0493	0.9825
NASNetMobile	0.0968	0.9681	0.1255	0.9470
InceptionV3	0.6333	0.8024	0.4713	0.8520
DenseNet121	0.2923	0.8870	0.2332	0.8986

Training and Validation Accuracies for all the models were plotted for better visualization. Also, Training loss and Validation loss were plotted for each model. Then, testing set was used to obtain predictions using the model built. The Testing Accuracy was obtained for all the models. Some random images of the test set were used to obtain their predictions for brain tumor. The possibility of correct prediction in terms of percentage confidence was also obtained.

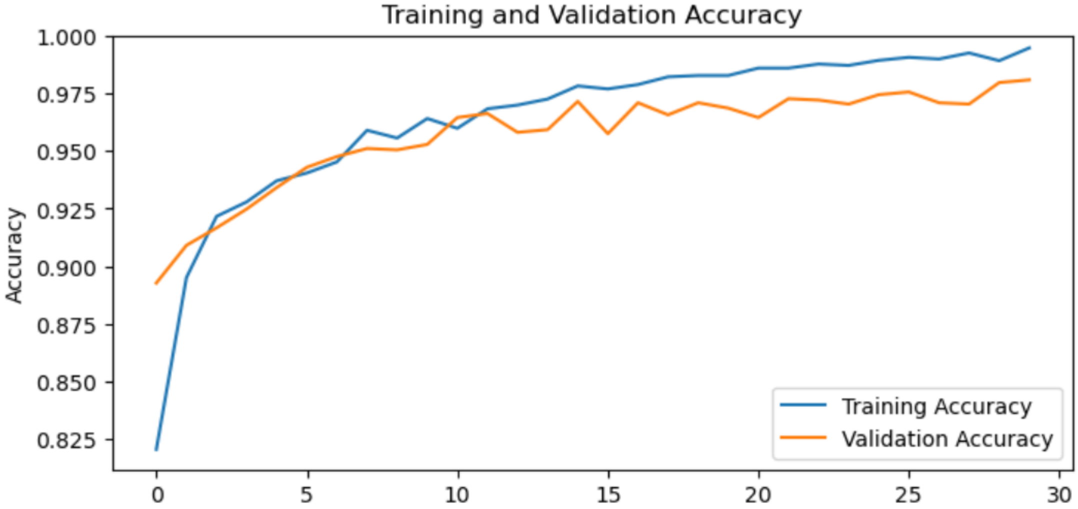


Figure 4.1. Training and Validation Accuracy for MobileNetV2 after 30 epochs

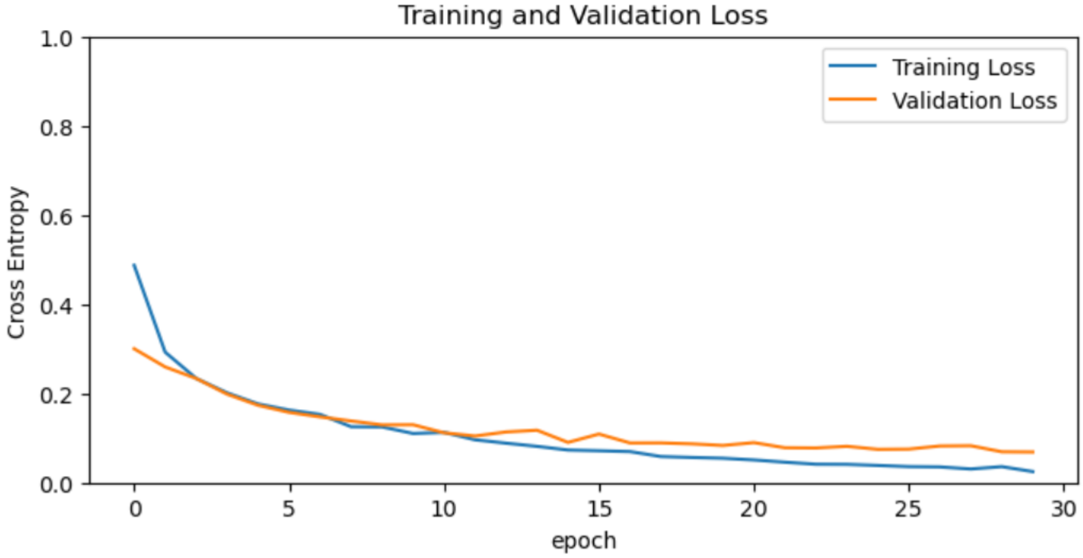
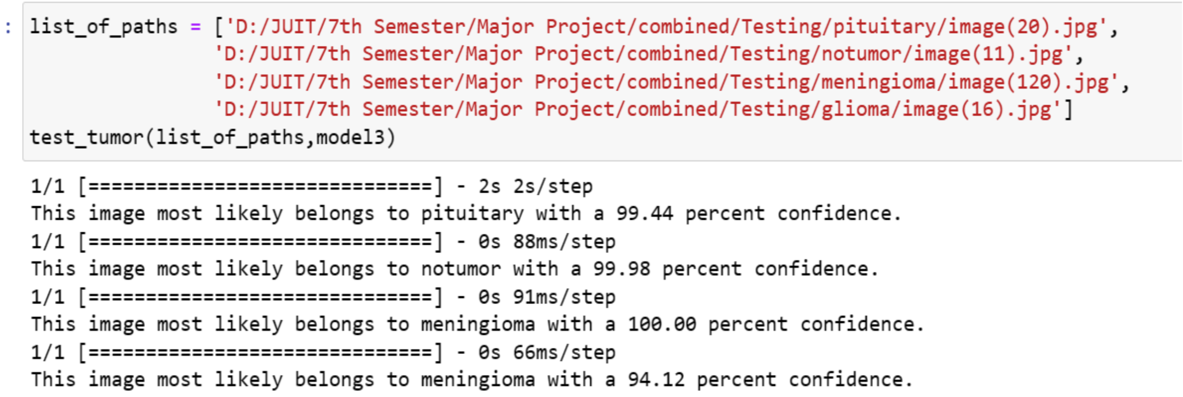


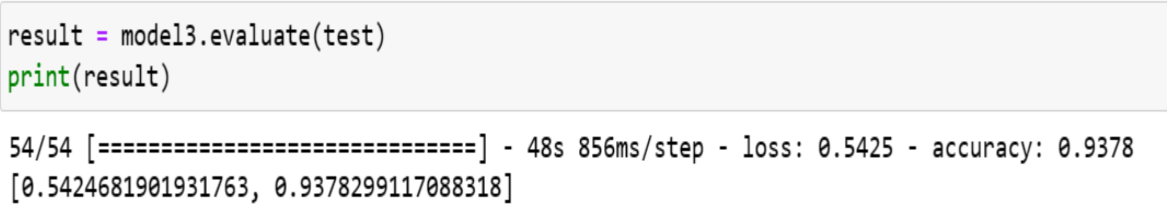
Figure 4.2. Training and Validation Loss for MobileNetV2 after 30 epochs



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model3)
```

```
1/1 [==============================] - 2s 2s/step
This image most likely belongs to pituitary with a 99.44 percent confidence.
1/1 [==============================] - 0s 88ms/step
This image most likely belongs to notumor with a 99.98 percent confidence.
1/1 [==============================] - 0s 91ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 66ms/step
This image most likely belongs to meningioma with a 94.12 percent confidence.
```

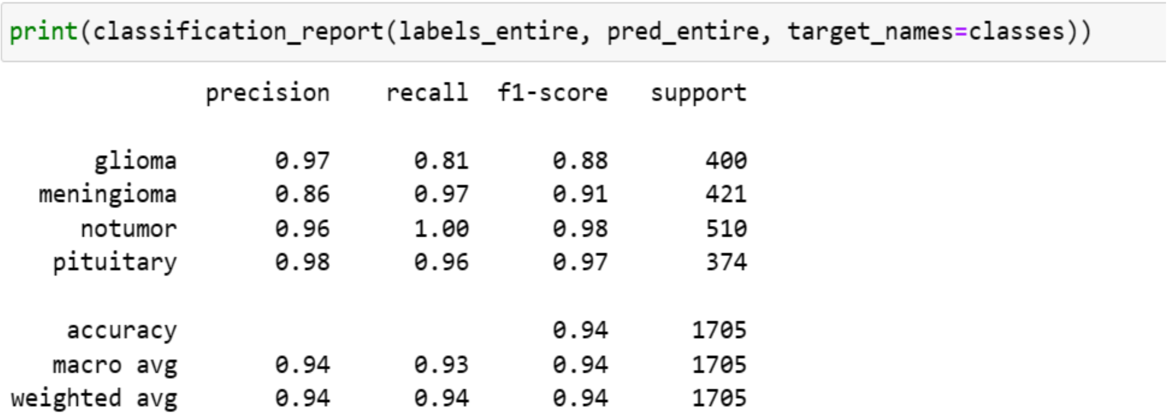
Figure 4.3. Predictions for random images of testing set using MobileNetV2 model



```
result = model3.evaluate(test)
print(result)
```

```
54/54 [==============================] - 48s 856ms/step - loss: 0.5425 - accuracy: 0.9378
[0.5424681901931763, 0.9378299117088318]
```

Figure 4.4. Test Accuracy for MobileNetV2 model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.97      0.81      0.88       400
  meningioma       0.86      0.97      0.91       421
     notumor       0.96      1.00      0.98       510
   pituitary       0.98      0.96      0.97       374

    accuracy                           0.94      1705
   macro avg       0.94      0.93      0.94      1705
weighted avg       0.94      0.94      0.94      1705
```

Figure 4.5. Classification report for MobileNetV2 model

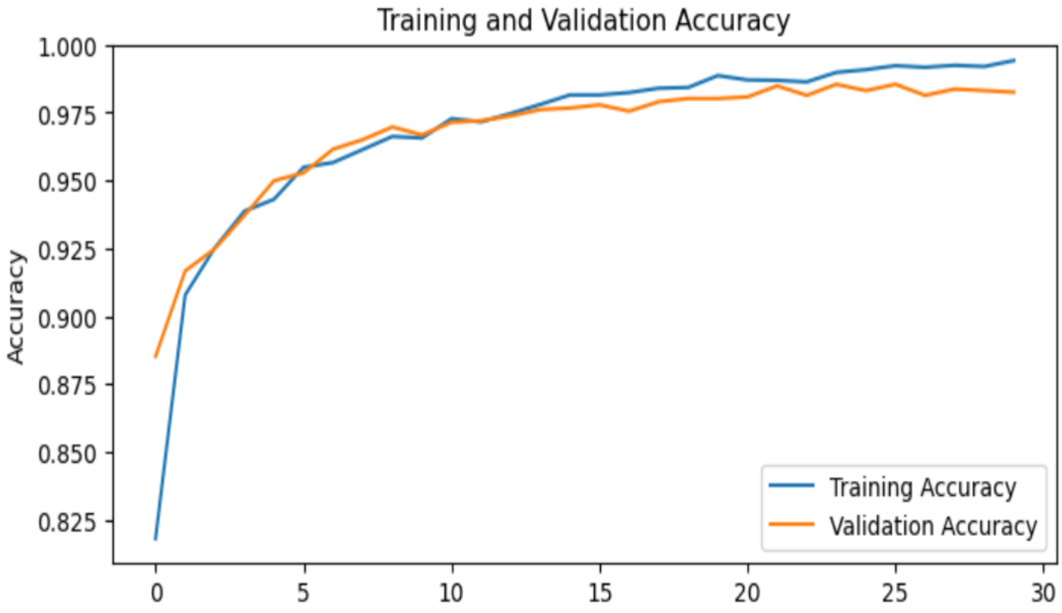


Figure 4.6. Training and Validation Accuracy for EfficientNetB1 model

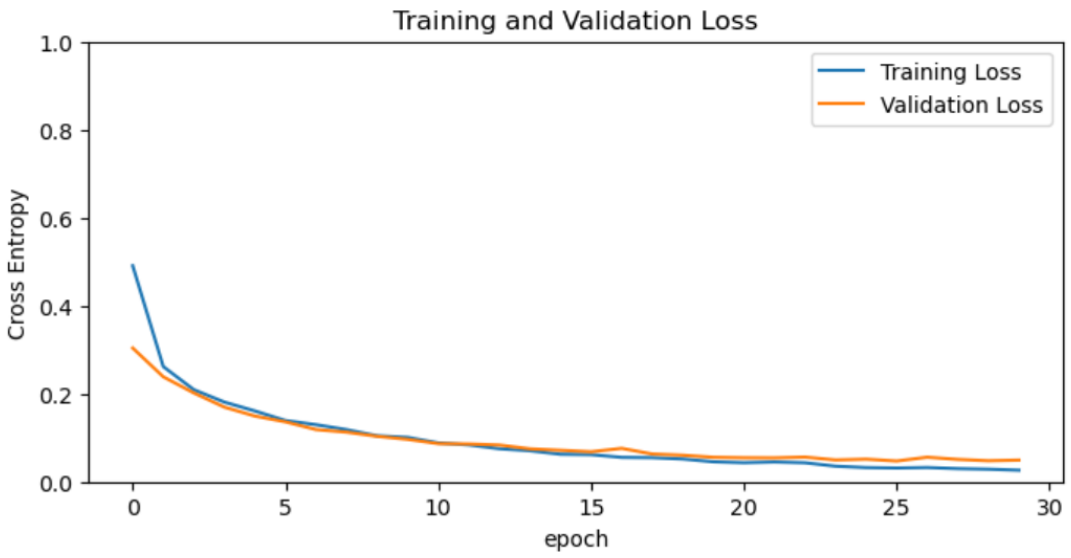
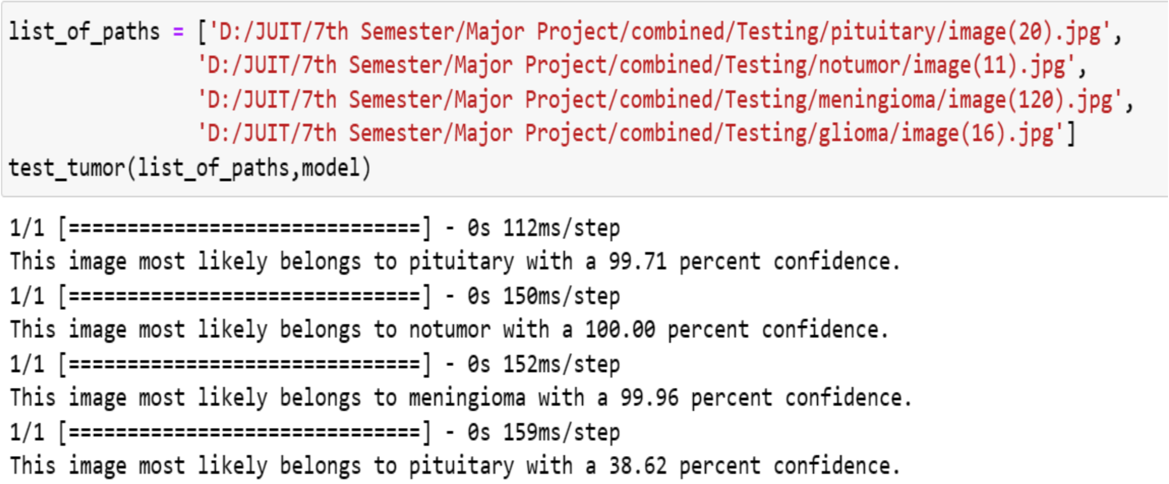


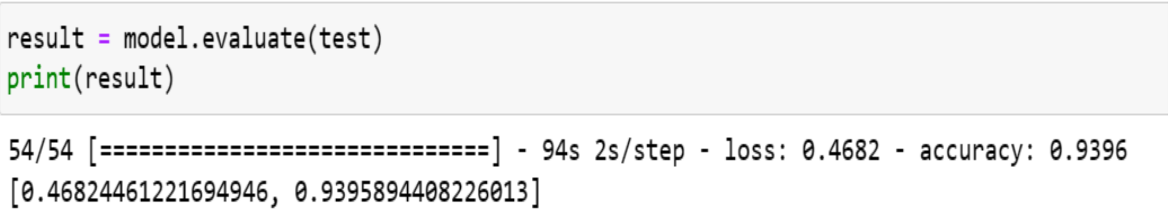
Figure 4.7. Training and Validation Loss for EfficientNetB1 model



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model)
```

```
1/1 [==============================] - 0s 112ms/step
This image most likely belongs to pituitary with a 99.71 percent confidence.
1/1 [==============================] - 0s 150ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 152ms/step
This image most likely belongs to meningioma with a 99.96 percent confidence.
1/1 [==============================] - 0s 159ms/step
This image most likely belongs to pituitary with a 38.62 percent confidence.
```

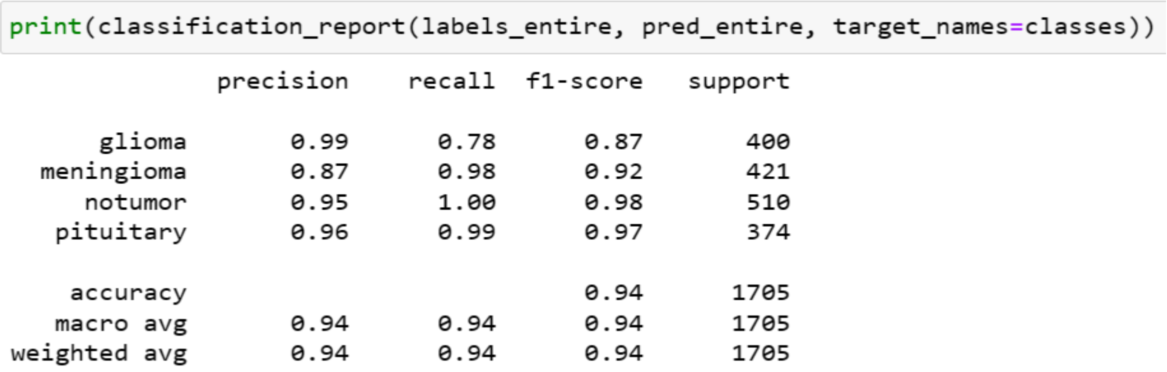
Figure 4.8. Predictions for random test images with confidence



```
result = model.evaluate(test)
print(result)
```

```
54/54 [==============================] - 94s 2s/step - loss: 0.4682 - accuracy: 0.9396
[0.46824461221694946, 0.9395894408226013]
```

Figure 4.9. Test Accuracy for EfficientNetB1 model



```python
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.99      0.78      0.87       400
  meningioma       0.87      0.98      0.92       421
     notumor       0.95      1.00      0.98       510
   pituitary       0.96      0.99      0.97       374

    accuracy                           0.94      1705
   macro avg       0.94      0.94      0.94      1705
weighted avg       0.94      0.94      0.94      1705
```

Figure 4.10. Classification Report for EfficientNetB1 model

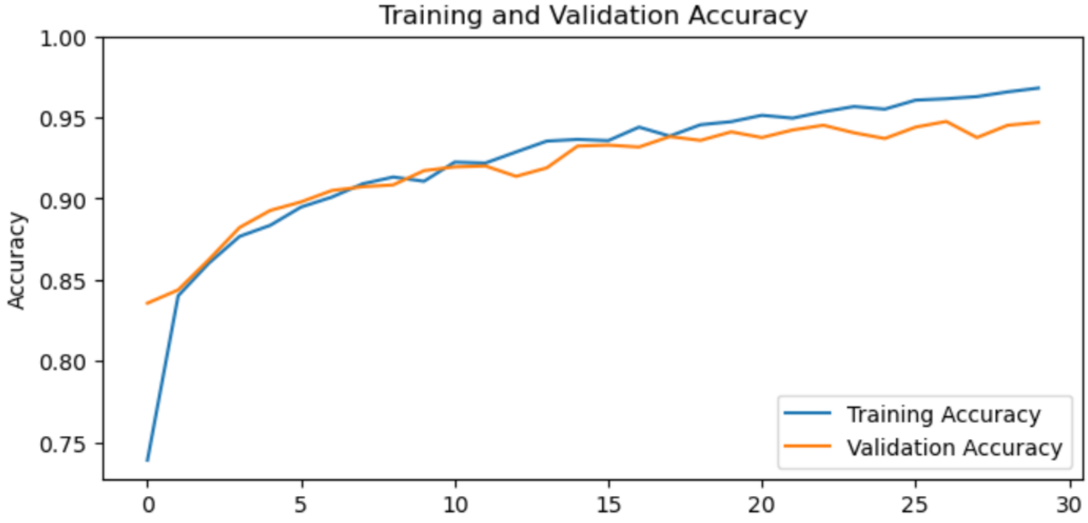


Figure 4.11. Training and Validation Accuracy for NASNetMobile model

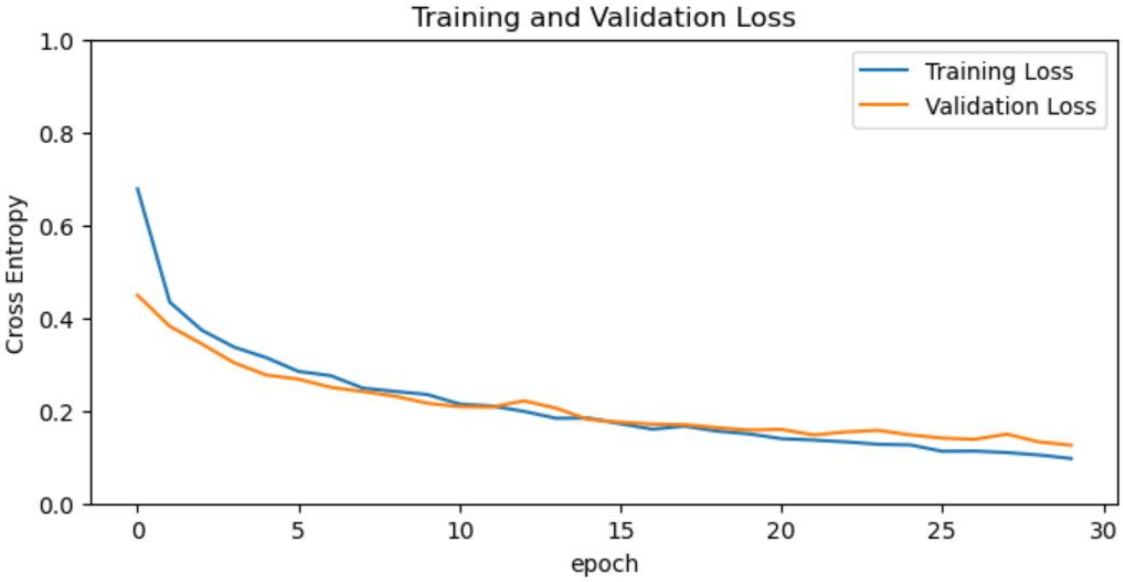
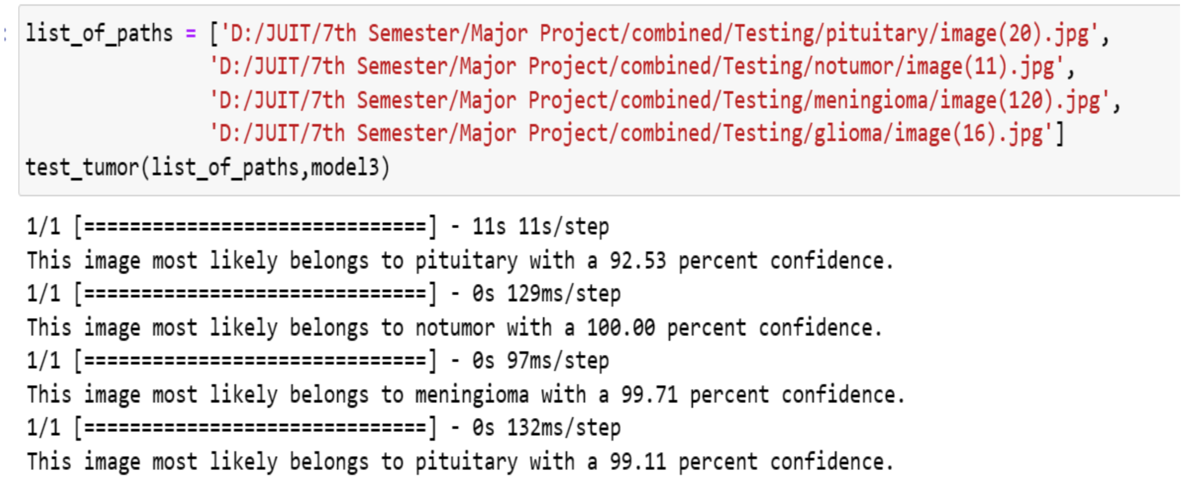


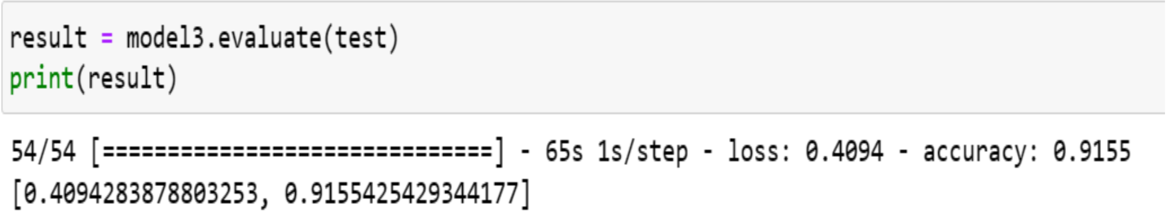
Figure 4.12. Training and Validation Loss for NASNetMobile model



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model3)
```

```
1/1 [==============================] - 11s 11s/step
This image most likely belongs to pituitary with a 92.53 percent confidence.
1/1 [==============================] - 0s 129ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 97ms/step
This image most likely belongs to meningioma with a 99.71 percent confidence.
1/1 [==============================] - 0s 132ms/step
This image most likely belongs to pituitary with a 99.11 percent confidence.
```

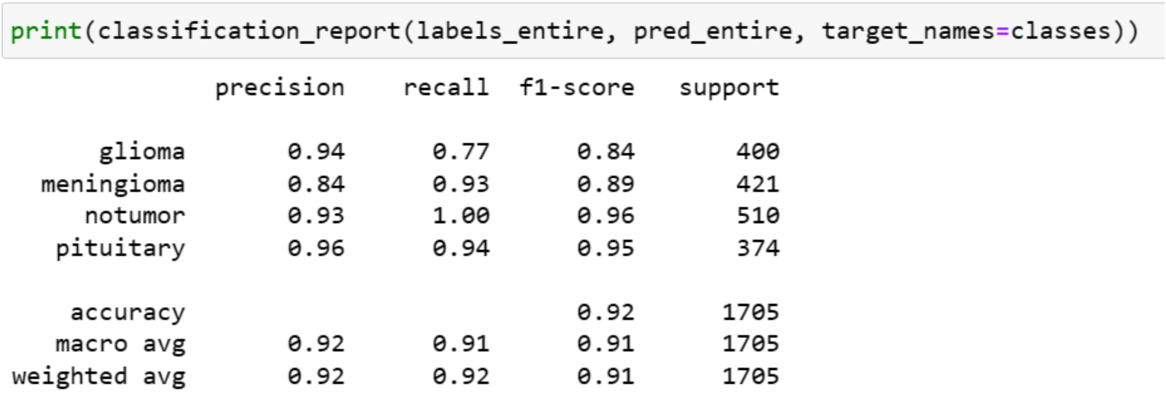
Figure 4.13. Predictions for test images with confidence



```
result = model3.evaluate(test)
print(result)
```

```
54/54 [==============================] - 65s 1s/step - loss: 0.4094 - accuracy: 0.9155
[0.4094283878803253, 0.9155425429344177]
```

Figure 4.14. Test Accuracy for NASNetMobile model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.94      0.77      0.84       400
  meningioma       0.84      0.93      0.89       421
     notumor       0.93      1.00      0.96       510
   pituitary       0.96      0.94      0.95       374

    accuracy                           0.92      1705
   macro avg       0.92      0.91      0.91      1705
weighted avg       0.92      0.92      0.91      1705
```

Figure 4.15. Classification report for NASNetMobile model

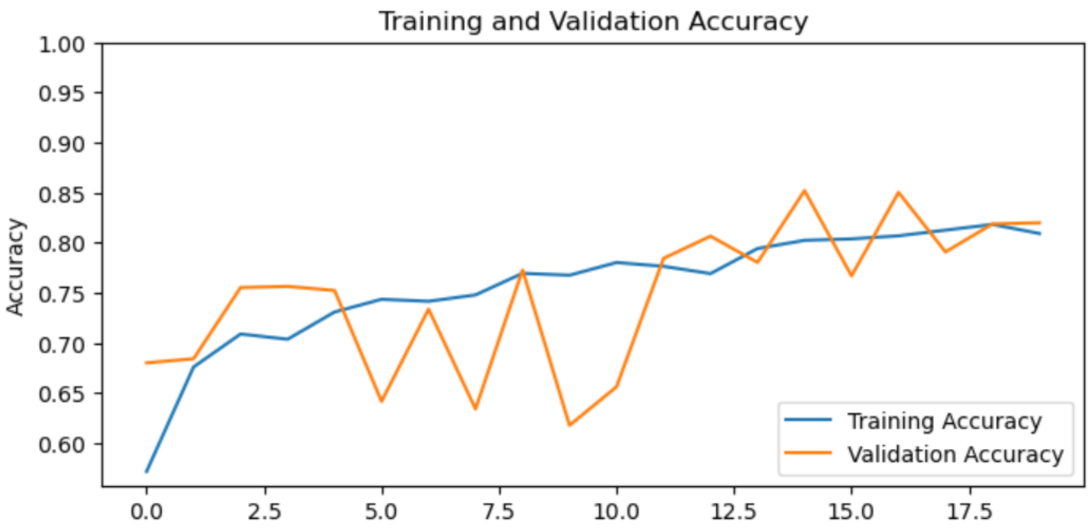


Figure 4. 16. Training and Validation Accuracy for InceptionV3

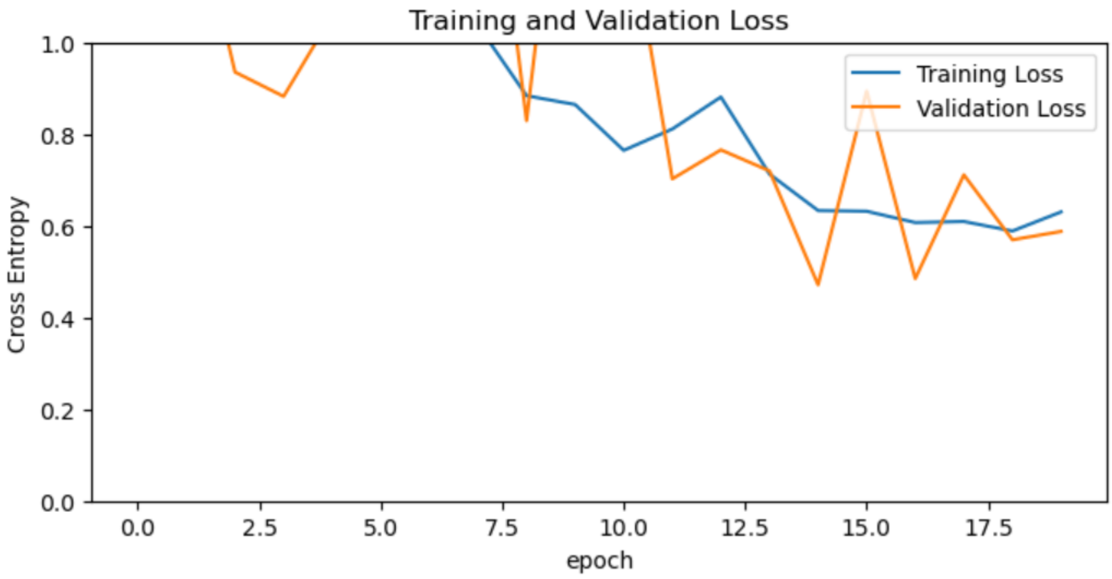
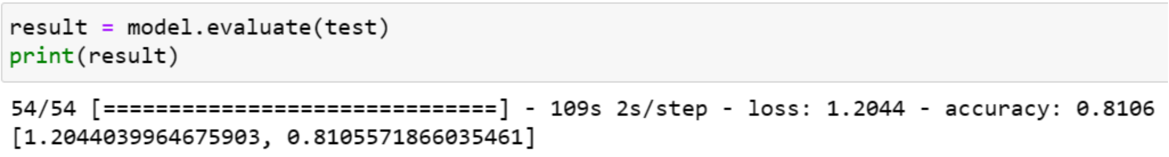


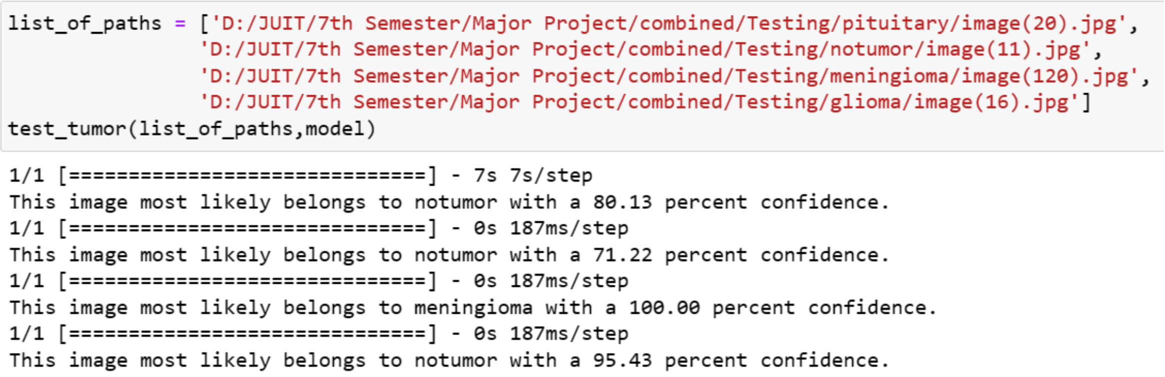
Figure 4. 17. Training and Validation Loss for InceptionV3



```
result = model.evaluate(test)
print(result)
```

```
54/54 [==============================] - 109s 2s/step - loss: 1.2044 - accuracy: 0.8106
[1.2044039964675903, 0.8105571866035461]
```

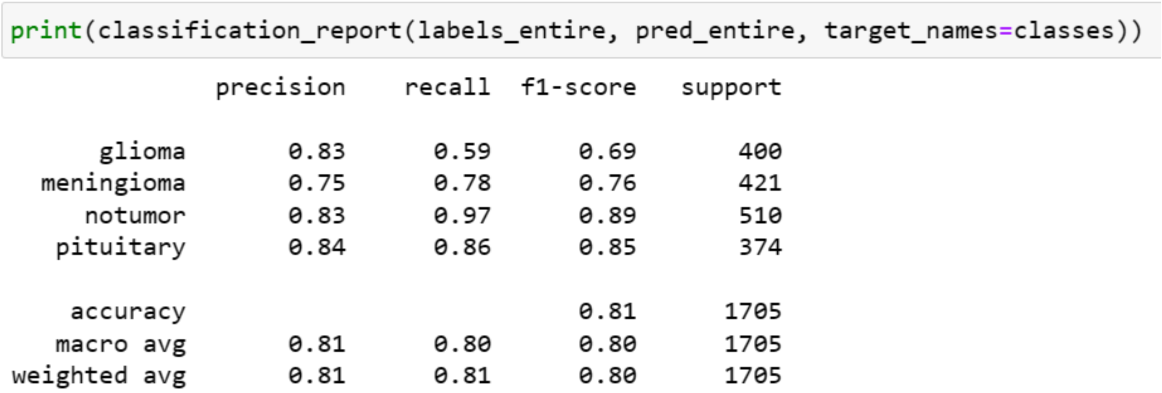
Figure 4. 18. Test accuracy of InceptionV3



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model)
```

```
1/1 [==============================] - 7s 7s/step
This image most likely belongs to notumor with a 80.13 percent confidence.
1/1 [==============================] - 0s 187ms/step
This image most likely belongs to notumor with a 71.22 percent confidence.
1/1 [==============================] - 0s 187ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 187ms/step
This image most likely belongs to notumor with a 95.43 percent confidence.
```

Figure 4. 19. Prediction of random MRIs using InceptionV3



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.83      0.59      0.69       400
  meningioma       0.75      0.78      0.76       421
     notumor       0.83      0.97      0.89       510
   pituitary       0.84      0.86      0.85       374

    accuracy                           0.81      1705
   macro avg       0.81      0.80      0.80      1705
weighted avg       0.81      0.81      0.80      1705
```

Figure 4. 20. Classification Report of InceptionV3

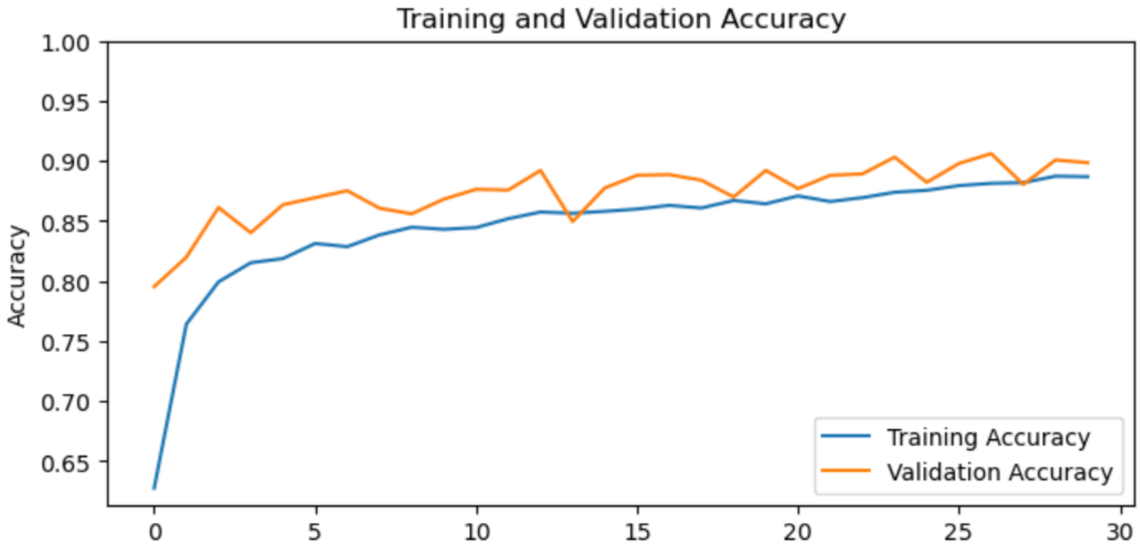


Figure 4. 21. Training And Validation Accuracy of DenseNet121 model

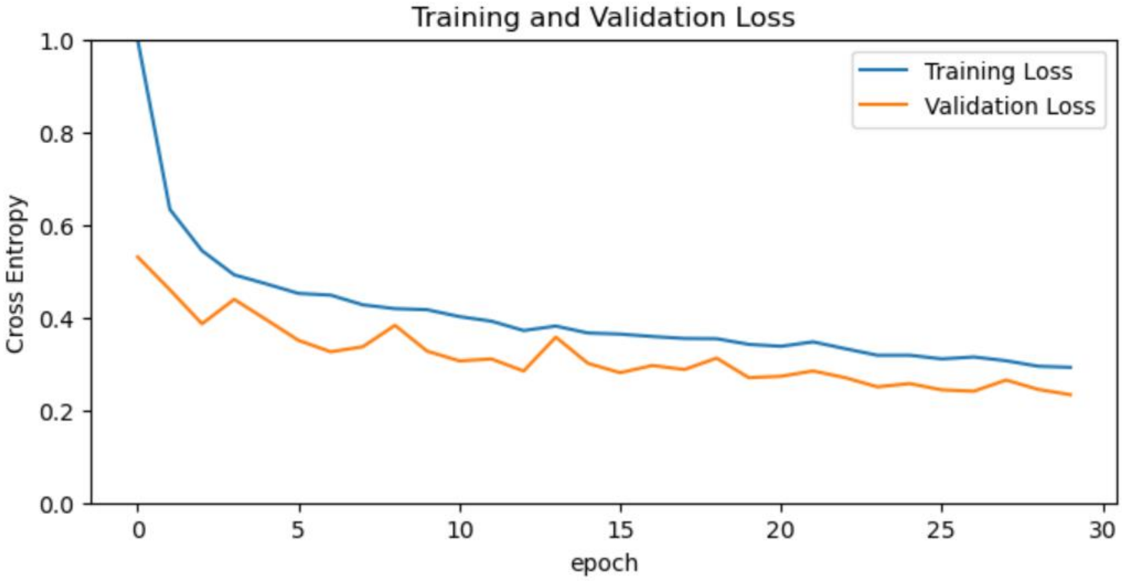
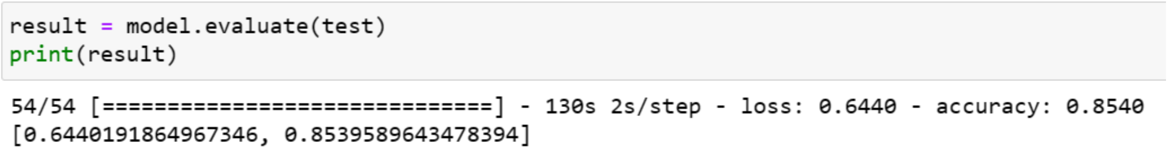


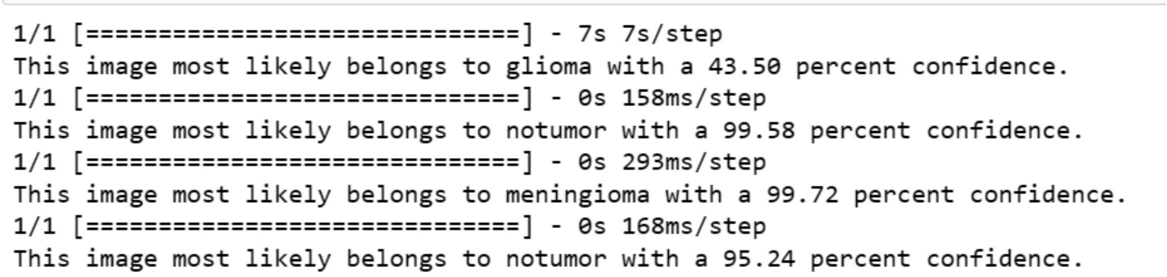
Figure 4. 22. Training and Validation Loss for DenseNet121 model



```
result = model.evaluate(test)
print(result)
```

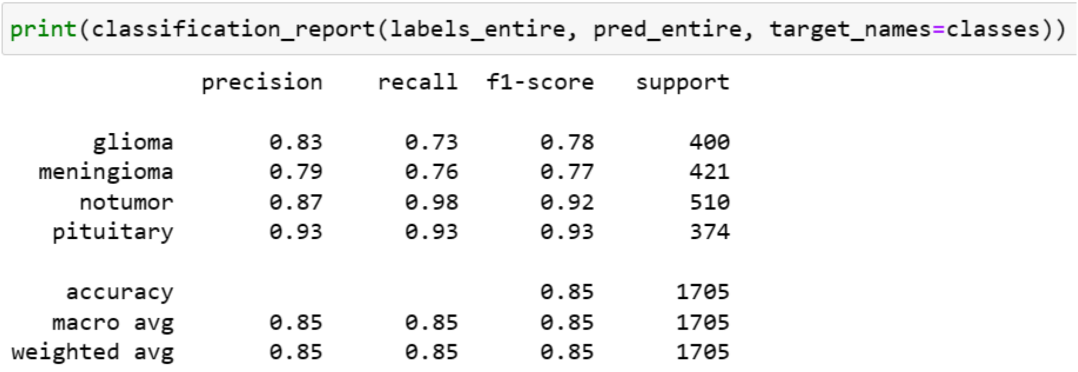
```
54/54 [==============================] - 130s 2s/step - loss: 0.6440 - accuracy: 0.8540
[0.6440191864967346, 0.8539589643478394]
```

Figure 4. 23. Test Accuracy for DenseNet121 model



```
1/1 [==============================] - 7s 7s/step
This image most likely belongs to glioma with a 43.50 percent confidence.
1/1 [==============================] - 0s 158ms/step
This image most likely belongs to notumor with a 99.58 percent confidence.
1/1 [==============================] - 0s 293ms/step
This image most likely belongs to meningioma with a 99.72 percent confidence.
1/1 [==============================] - 0s 168ms/step
This image most likely belongs to notumor with a 95.24 percent confidence.
```

Figure 4. 24. Predictions using DenseNet121 model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.83      0.73      0.78       400
  meningioma       0.79      0.76      0.77       421
     notumor       0.87      0.98      0.92       510
   pituitary       0.93      0.93      0.93       374

    accuracy                           0.85      1705
   macro avg       0.85      0.85      0.85      1705
weighted avg       0.85      0.85      0.85      1705
```

Figure 4. 25. Classification Report of DenseNet121 model

Test Accuracies of all the models have been summarized in Table 4.2. Later, all the three architectures were fine-tuned and a similar strategy for testing was used. Classification Reports for all the models have been summarized in Table 4.3. The training and validation details of the fine-tuned models have been summarized in Table 4.4.

Table 4.2. Test Accuracies after 30 epochs of each model

	Test Accuracy
MobileNetV2	0.9378
EfficientNetB1	0.9396
NASNetMobile	0.9155
InceptionV3	0.8105
DenseNet121	0.8539

Table 4.3. F-1 Scores for all the models at the end of 30 epochs

	MobileNetV2	EfficientNetB1	NASNetMobile	InceptionV3	DenseNet121
Glioma	0.88	0.87	0.84	0.69	0.78
Meningioma	0.91	0.92	0.89	0.76	0.77
No Tumor	0.98	0.98	0.96	0.89	0.92
Pituitary	0.97	0.97	0.95	0.85	0.93

The plots of training and validation accuracy and training and validation loss for fine-tuned models have been shown in the following figures.

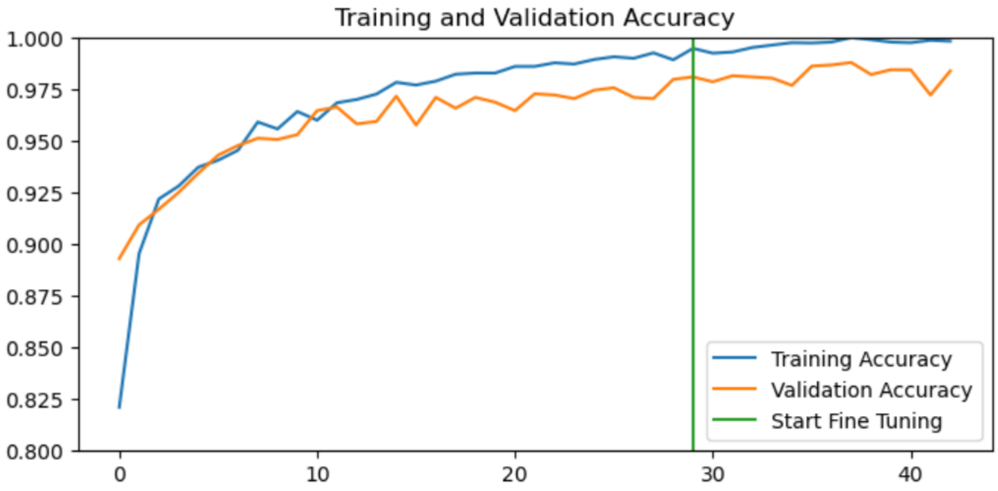


Figure 4.26. Training and Validation Accuracy after Fine-Tuning of MobileNetV2

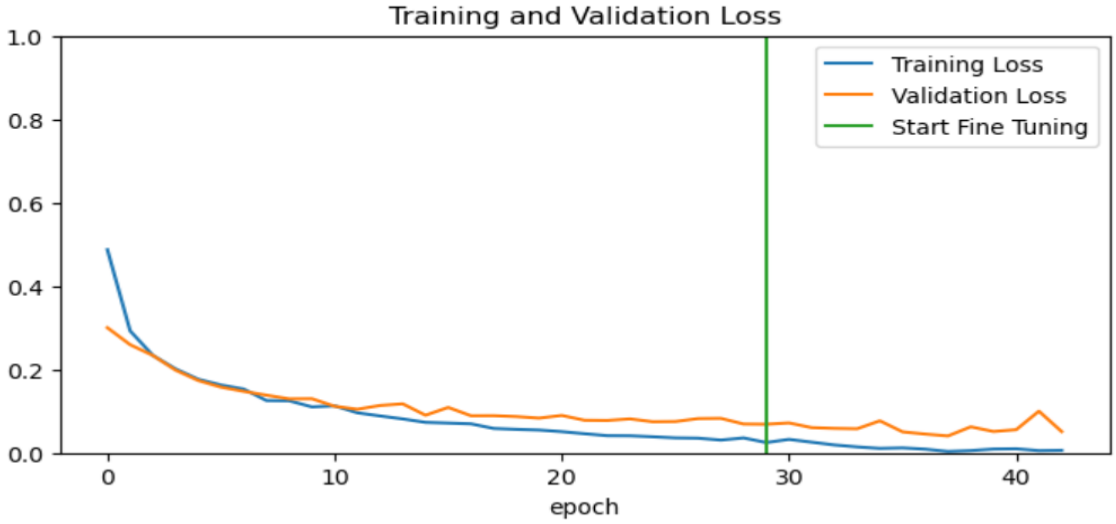


Figure 4.27. Training and Validation Loss after Fine-Tuning of MobileNetV2

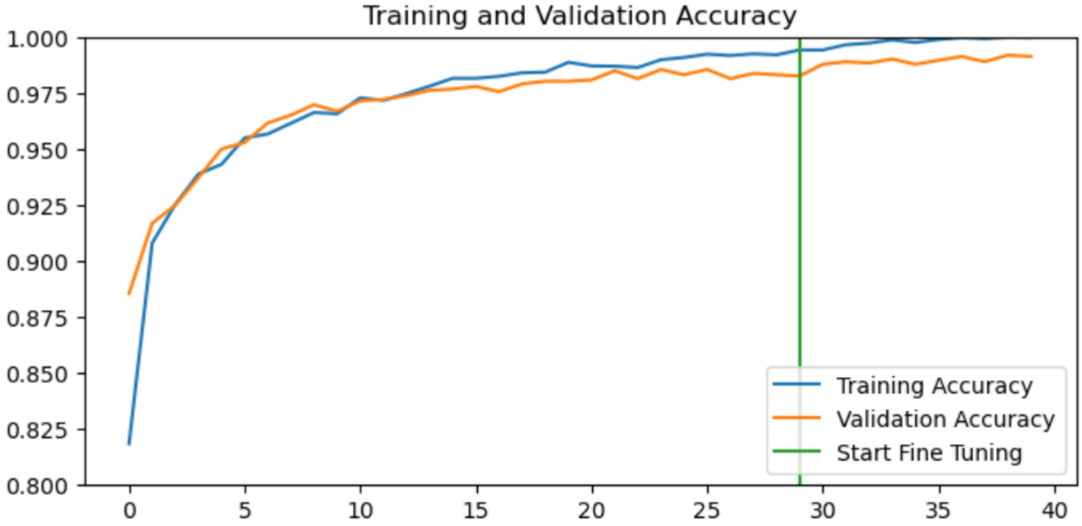


Figure 4.28. Training and Validation Accuracy after Fine-Tuning of EfficientNetB1

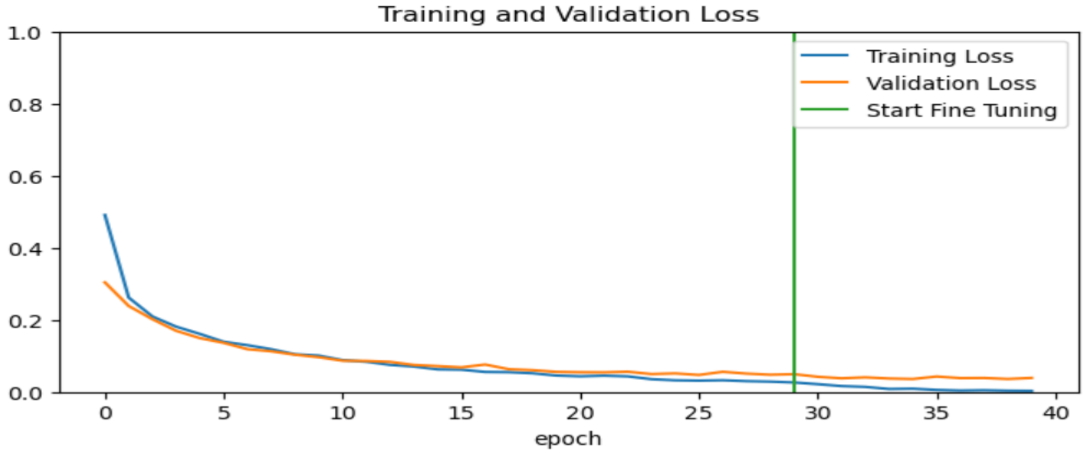


Figure 4.29. Training and Validation Accuracy after Fine-Tuning of EfficientNetB1

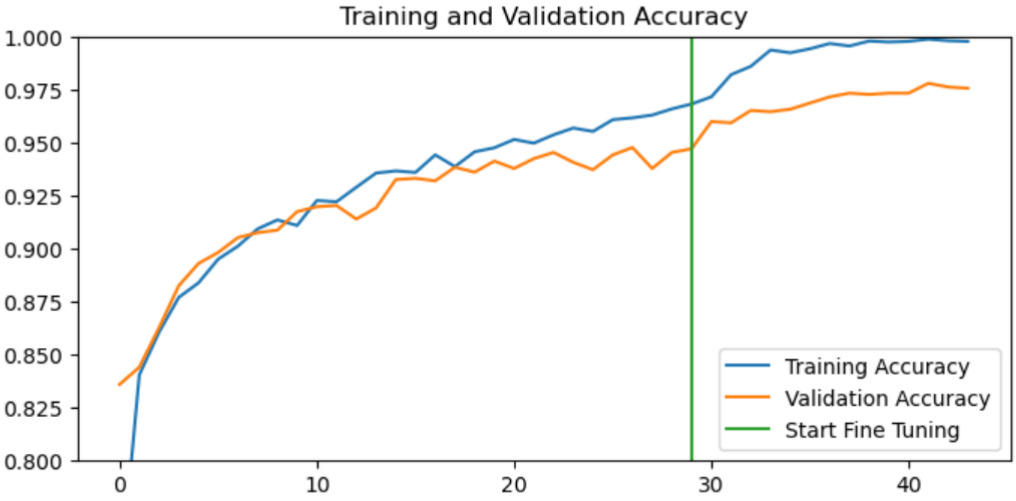


Figure 4.30. Training and Validation Accuracy after Fine-Tuning of NASNetMobile

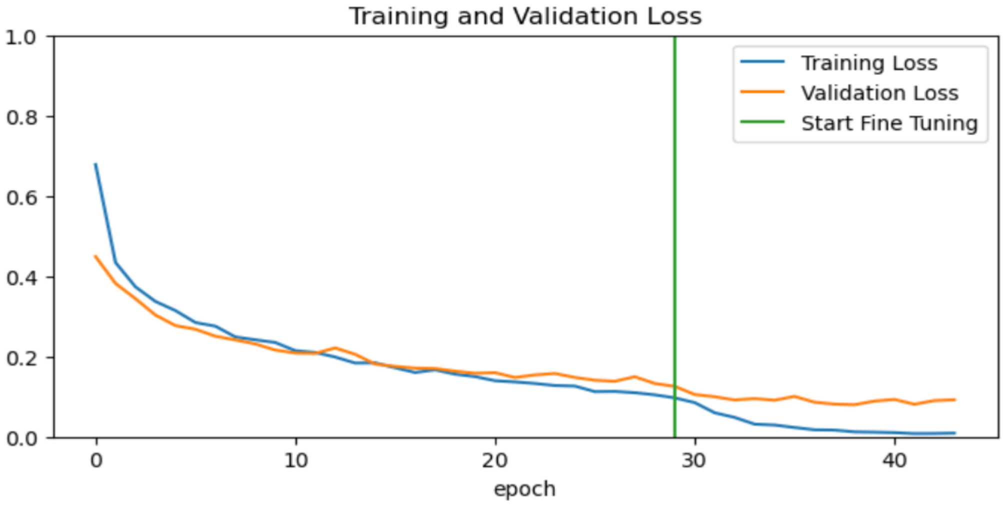


Figure 4.31. Training and Validation Loss after Fine-Tuning of NASNetMobile

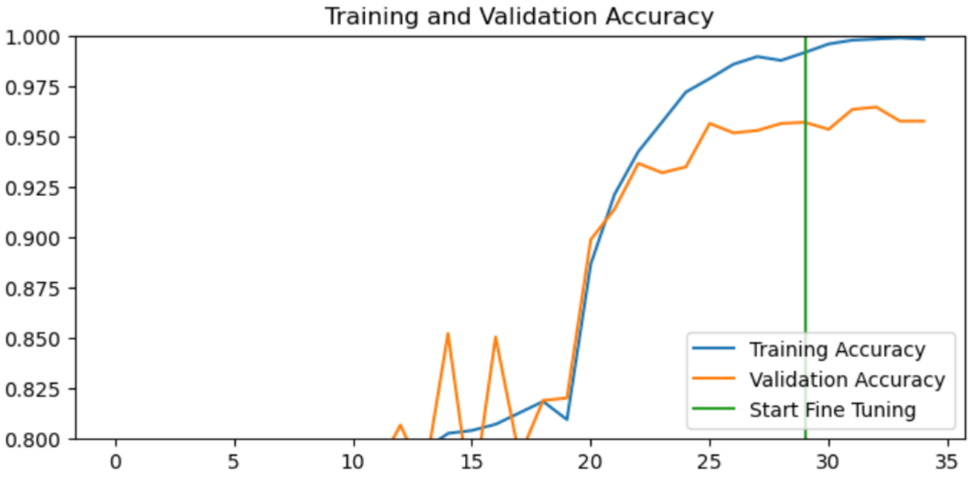


Figure 4. 32. Training and Validation Accuracy after fine-tuning of InceptionV3

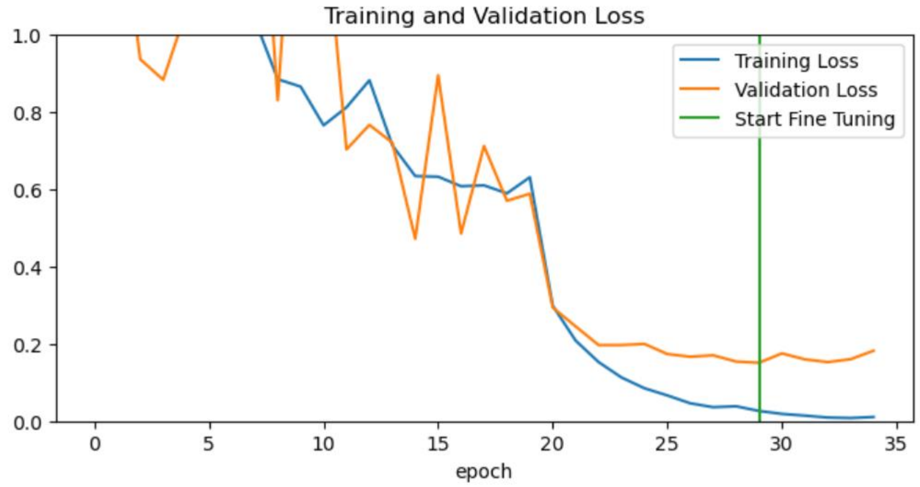


Figure 4. 33. Training and Validation Loss after fine-tuning InceptionV3

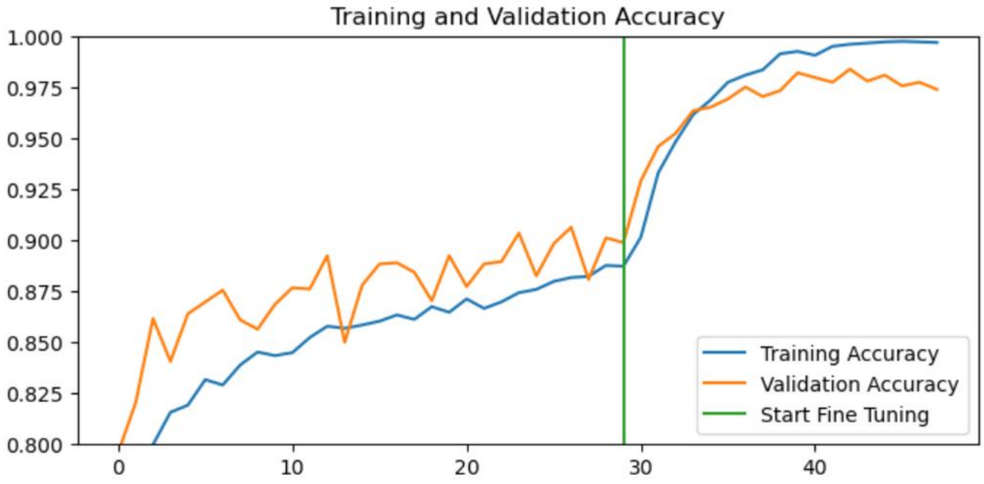


Figure 4. 34. Training and Validation Accuracy after fine-tuning DenseNet121

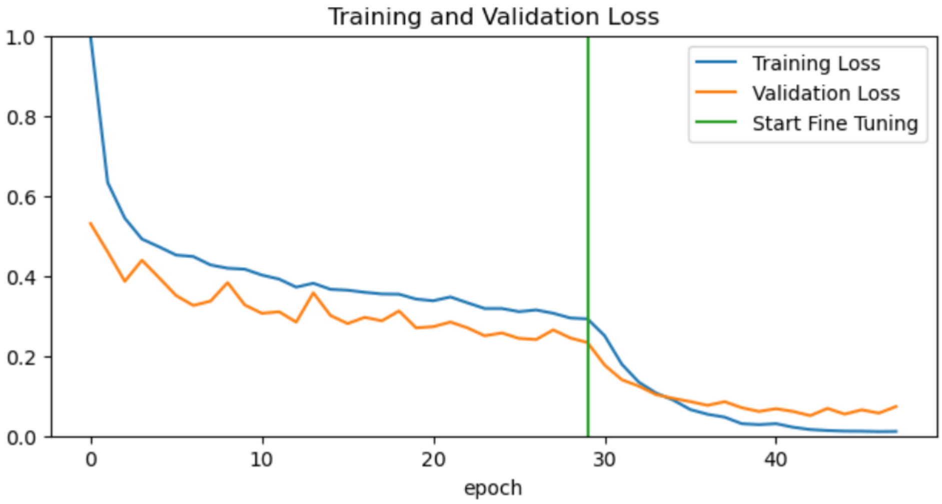
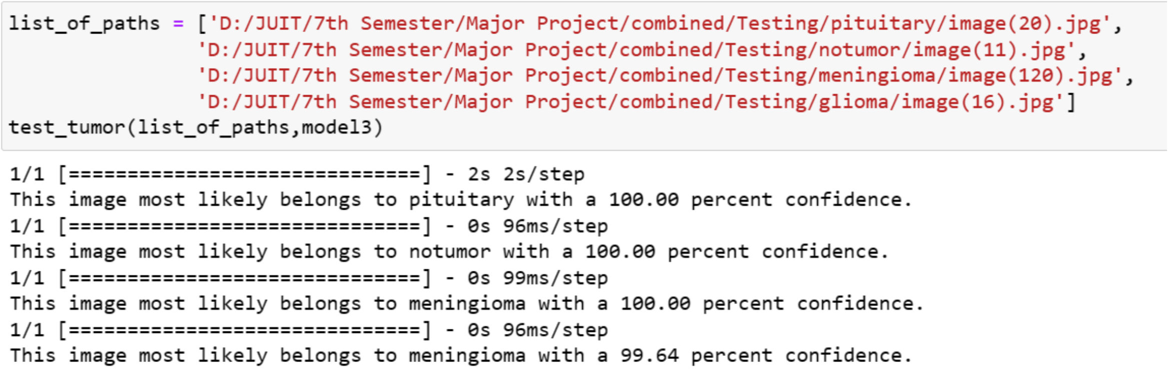


Figure 4. 35. Training and Validation Loss after fine-tuning DenseNet121

Additional 30 epochs were initialized for fine tuning of the models. All the models were already equipped with the early stopping feature with validation loss being monitored with patience of 5.

Table 4.4. Training and Validation after fine tuning

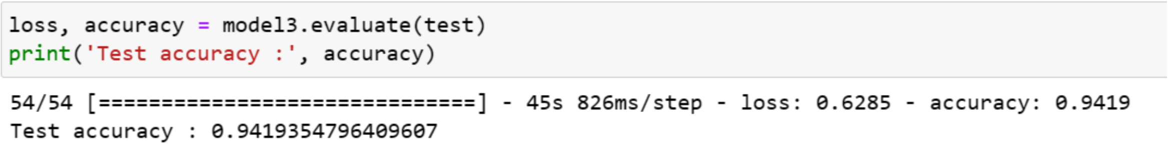
	Total no. of layers	Last Layers fine-tuned	Loss	Accuracy	Validation Loss	Validation Accuracy	Fine tune epochs
MobileNetV2	154	54	0.0034	0.9981	0.0406	0.9878	12
EfficientNetB1	340	90	0.0009	0.9975	0.0359	0.9878	9
NASNetMobile	769	169	0.0122	0.9978	0.0798	0.97526	13
InceptionV3	311	61	0.0262	0.9916	0.1508	0.9569	14
DenseNet121	427	177	0.0159	0.9959	0.0506	0.9837	17



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model3)
```

```
1/1 [==============================] - 2s 2s/step
This image most likely belongs to pituitary with a 100.00 percent confidence.
1/1 [==============================] - 0s 96ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 99ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 96ms/step
This image most likely belongs to meningioma with a 99.64 percent confidence.
```

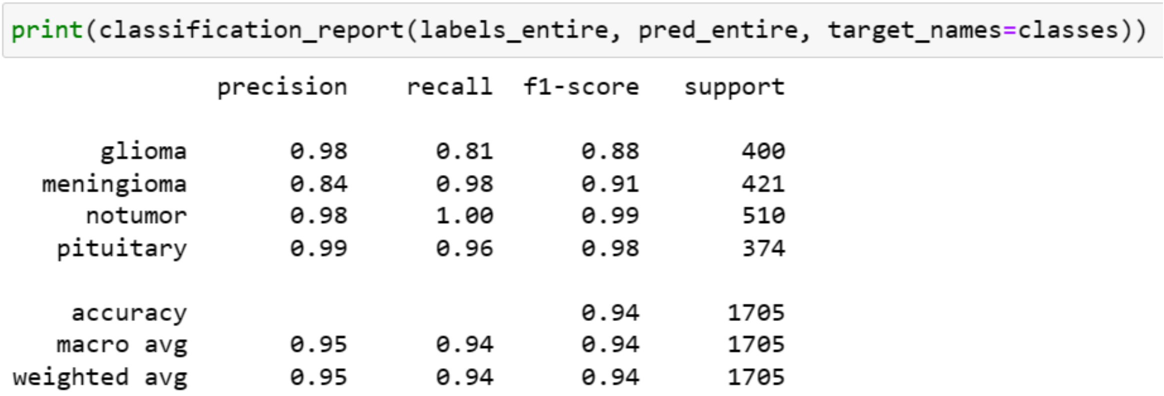
Figure 4.36. Prediction of test images using fine-tuned MobileNetV2 model



```
loss, accuracy = model3.evaluate(test)
print('Test accuracy :', accuracy)
```

```
54/54 [==============================] - 45s 826ms/step - loss: 0.6285 - accuracy: 0.9419
Test accuracy : 0.9419354796409607
```

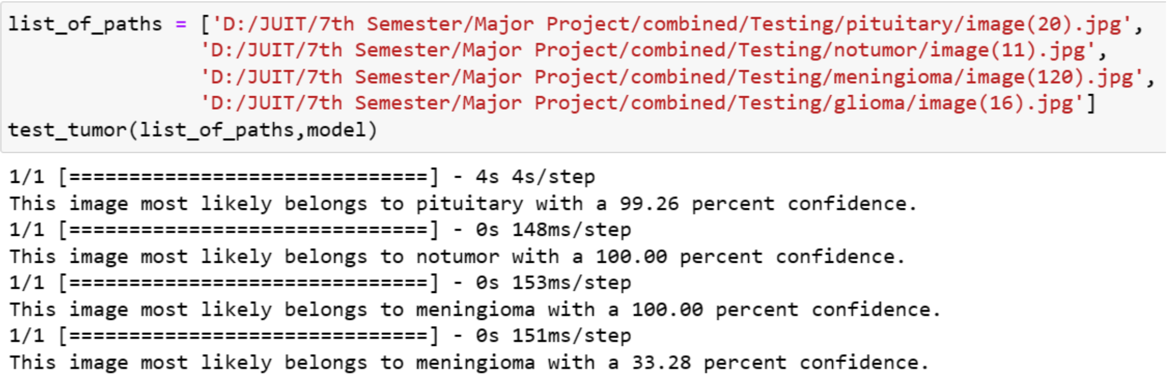
Figure 4.37. Test Accuracy for fine-tuned MobileNetV2 model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.98      0.81      0.88       400
  meningioma       0.84      0.98      0.91       421
     notumor       0.98      1.00      0.99       510
   pituitary       0.99      0.96      0.98       374

    accuracy                           0.94      1705
   macro avg       0.95      0.94      0.94      1705
weighted avg       0.95      0.94      0.94      1705
```

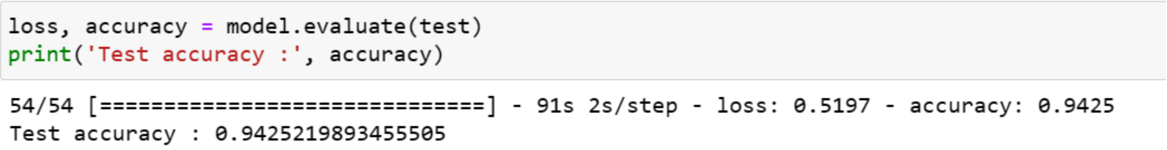
Figure 4.38. Classification report for MobileNetV2 fine-tuned model



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model)
```

```
1/1 [==============================] - 4s 4s/step
This image most likely belongs to pituitary with a 99.26 percent confidence.
1/1 [==============================] - 0s 148ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 153ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 151ms/step
This image most likely belongs to meningioma with a 33.28 percent confidence.
```

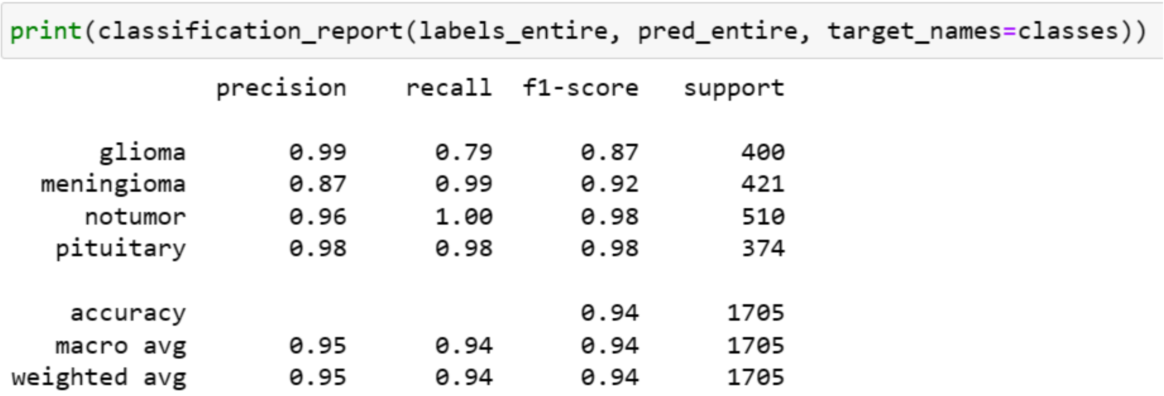
Figure 4.39. Prediction for test images with confidence



```
loss, accuracy = model.evaluate(test)
print('Test accuracy :', accuracy)
```

```
54/54 [==============================] - 91s 2s/step - loss: 0.5197 - accuracy: 0.9425
Test accuracy : 0.9425219893455505
```

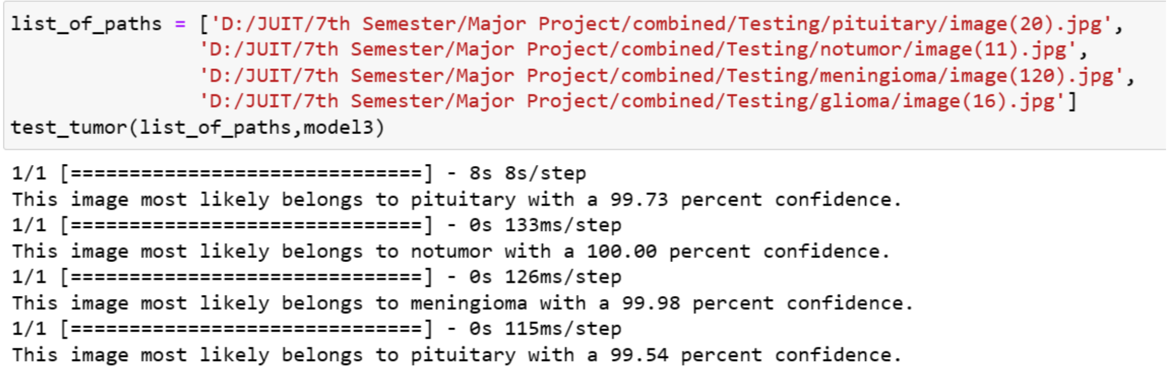
Figure 4.40. Test Accuracy of fine-tuned EfficientNetB1 model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.99      0.79      0.87       400
  meningioma       0.87      0.99      0.92       421
     notumor       0.96      1.00      0.98       510
   pituitary       0.98      0.98      0.98       374

    accuracy                           0.94      1705
   macro avg       0.95      0.94      0.94      1705
weighted avg       0.95      0.94      0.94      1705
```

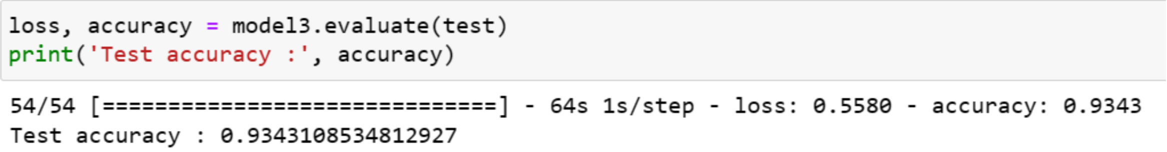
Figure 4.41. Classification report for fine-tuned EfficientNetB1 model



```
list_of_paths = ['D:/JUIT/7th Semester/Major Project/combined/Testing/pituitary/image(20).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/notumor/image(11).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/meningioma/image(120).jpg',
                 'D:/JUIT/7th Semester/Major Project/combined/Testing/glioma/image(16).jpg']
test_tumor(list_of_paths,model3)
```

```
1/1 [==============================] - 8s 8s/step
This image most likely belongs to pituitary with a 99.73 percent confidence.
1/1 [==============================] - 0s 133ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 126ms/step
This image most likely belongs to meningioma with a 99.98 percent confidence.
1/1 [==============================] - 0s 115ms/step
This image most likely belongs to pituitary with a 99.54 percent confidence.
```

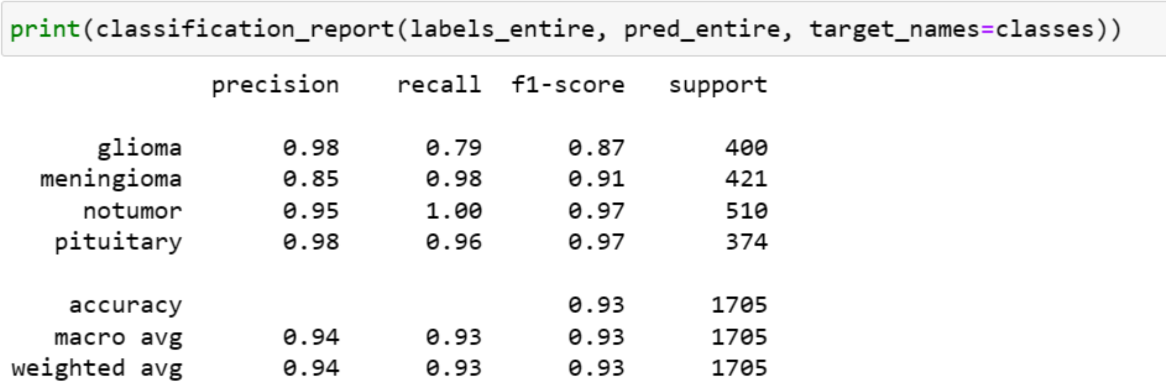
Figure 4.42. Predictions for test images with confidence



```
loss, accuracy = model3.evaluate(test)
print('Test accuracy :', accuracy)
```

```
54/54 [==============================] - 64s 1s/step - loss: 0.5580 - accuracy: 0.9343
Test accuracy : 0.9343108534812927
```

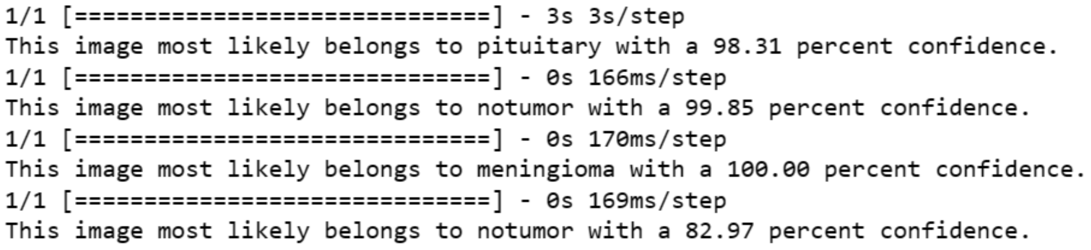
Figure 4.43. Test Accuracy of fine-tuned NASNetMobile model



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

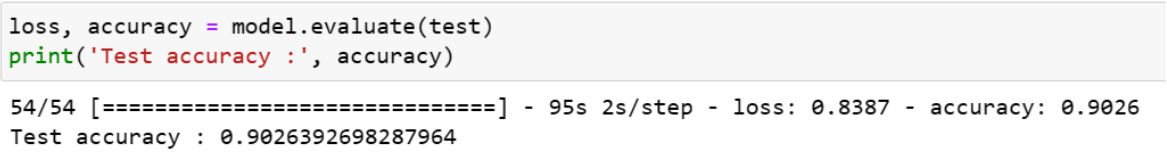
```
              precision    recall  f1-score   support

      glioma       0.98      0.79      0.87       400
  meningioma       0.85      0.98      0.91       421
     notumor       0.95      1.00      0.97       510
   pituitary       0.98      0.96      0.97       374

    accuracy                           0.93      1705
   macro avg       0.94      0.93      0.93      1705
weighted avg       0.94      0.93      0.93      1705
```

Figure 4.44. Classification report of fine-tuned NASNetMobile model



```
1/1 [==============================] - 3s 3s/step
This image most likely belongs to pituitary with a 98.31 percent confidence.
1/1 [==============================] - 0s 166ms/step
This image most likely belongs to notumor with a 99.85 percent confidence.
1/1 [==============================] - 0s 170ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 169ms/step
This image most likely belongs to notumor with a 82.97 percent confidence.
```

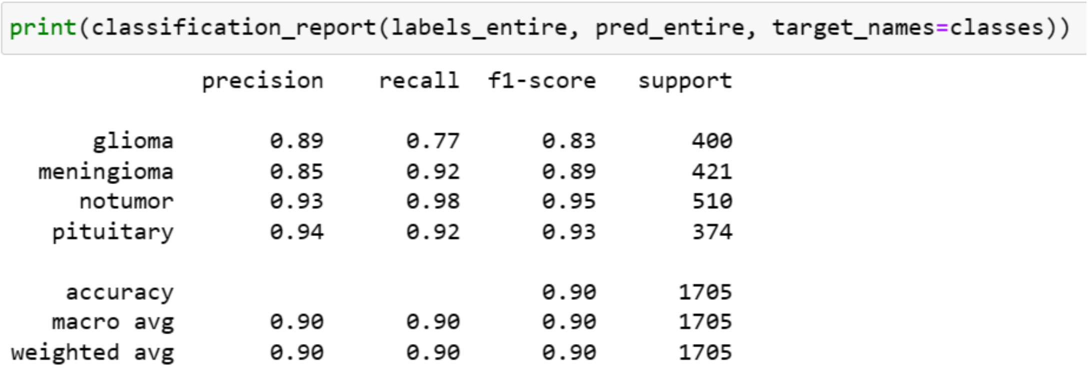
Figure 4. 45. Predictions using fine-tuned InceptionV3



```
loss, accuracy = model.evaluate(test)
print('Test accuracy :', accuracy)
```

```
54/54 [==============================] - 95s 2s/step - loss: 0.8387 - accuracy: 0.9026
Test accuracy : 0.9026392698287964
```

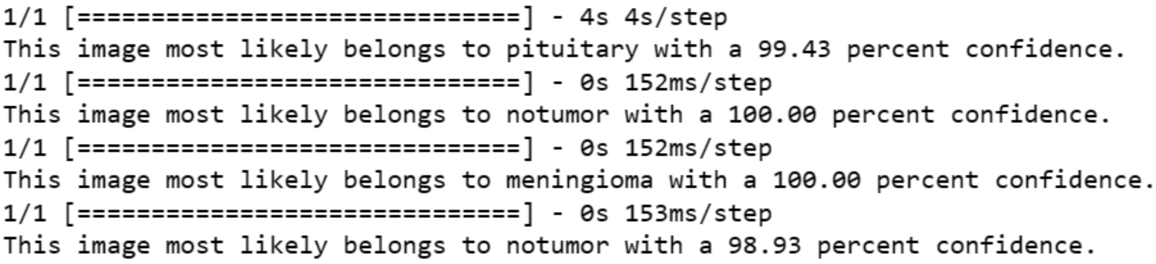
Figure 4. 46. Test Accuracy of fine-tuned InceptionV3



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

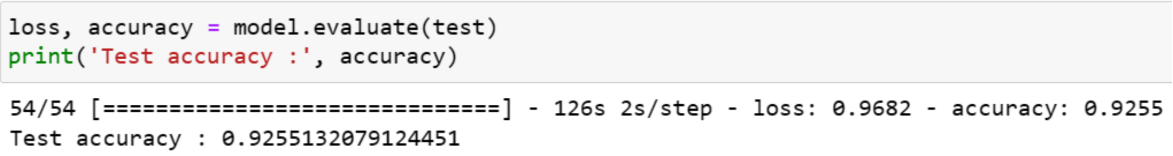
```
              precision    recall  f1-score   support

      glioma       0.89      0.77      0.83       400
  meningioma       0.85      0.92      0.89       421
     notumor       0.93      0.98      0.95       510
   pituitary       0.94      0.92      0.93       374

    accuracy                           0.90      1705
   macro avg       0.90      0.90      0.90      1705
weighted avg       0.90      0.90      0.90      1705
```

Figure 4. 47. Classification Report of fine-tuned InceptionV3



```
1/1 [==============================] - 4s 4s/step
This image most likely belongs to pituitary with a 99.43 percent confidence.
1/1 [==============================] - 0s 152ms/step
This image most likely belongs to notumor with a 100.00 percent confidence.
1/1 [==============================] - 0s 152ms/step
This image most likely belongs to meningioma with a 100.00 percent confidence.
1/1 [==============================] - 0s 153ms/step
This image most likely belongs to notumor with a 98.93 percent confidence.
```

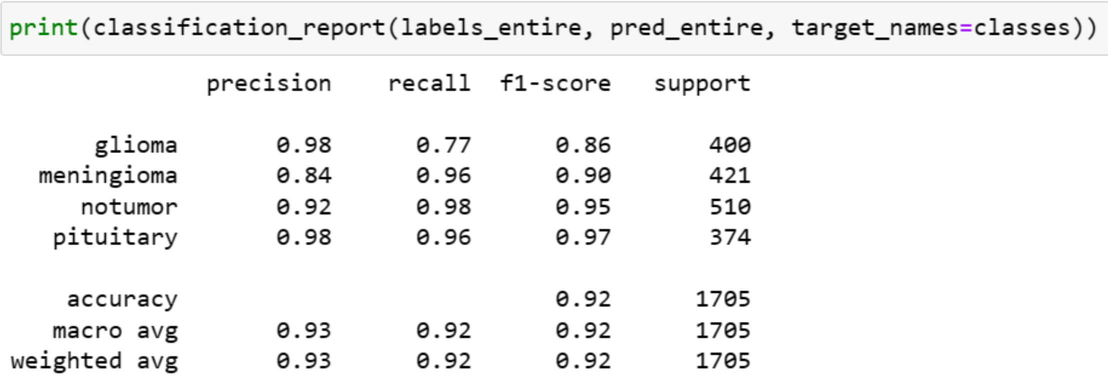
Figure 4. 48. Predictions using fine-tuned DenseNet121



```
loss, accuracy = model.evaluate(test)
print('Test accuracy :', accuracy)
```

```
54/54 [==============================] - 126s 2s/step - loss: 0.9682 - accuracy: 0.9255
Test accuracy : 0.9255132079124451
```

Figure 4. 49. Test Accuracy of fine-tuned DenseNet121



```
print(classification_report(labels_entire, pred_entire, target_names=classes))
```

```
              precision    recall  f1-score   support

      glioma       0.98      0.77      0.86       400
  meningioma       0.84      0.96      0.90       421
     notumor       0.92      0.98      0.95       510
   pituitary       0.98      0.96      0.97       374

    accuracy                           0.92      1705
   macro avg       0.93      0.92      0.92      1705
weighted avg       0.93      0.92      0.92      1705
```

Figure 4. 50. Classification report of fine-tuned DenseNet121

The results after fine-tuning of each model have been summarized in Table 4.5 and 4.6.

Table 4.5. Test Accuracies after fine-tuning of each model

	Test Accuracy
MobileNetV2	0.9419
EfficientNetB1	0.9425
NASNetMobile	0.9343
InceptionV3	0.9026
DenseNet121	0.9255

Table 4.6. F-1 Scores for all the models after fine-tuning

	MobileNetV2	EfficientNetB1	NASNetMobile	InceptionV3	DenseNet121
Glioma	0.88	0.87	0.87	0.83	0.86
Meningioma	0.91	0.92	0.91	0.89	0.90
No Tumor	0.99	0.98	0.97	0.95	0.95
Pituitary	0.98	0.98	0.97	0.93	0.97

4.2 TEST CASES AND OUTCOMES

Each model was analyzed and tested on the following parameters-

- Training Accuracy and Loss
- Validation Accuracy and Loss
- Testing Accuracy
- Precision, Recall and F-1 Score

The details of the results on the basis of the above parameters have been shown in the previous section.

On the basis of Test Accuracy. EfficientNetB1 was found to be the best model for classification of brain tumors. The test accuracy of the pre-trained model was 93.96% and after fine-tuning it increased to 94.25%.

On the basis of F-1 Score, best models for different classes of tumors were MobileNetV2 and EfficientNetB1 as these both models had competitive F-1 scores.

As a part of the testing process, 4 random images were taken from the test set and predictions for those were obtained. The predictions and their confidence have been summarized in Table 4.7 and after fine tuning the models, the summary has been presented in Table 4.8.

Table 4.7. Predictions after 30 epochs

Sr. No.	Original Class	MobileNetV2		EfficientNetB1		NASNetMobile		InceptionV3		DenseNet121	
		Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)
1.	Pituitary	Pituitary	99.44	Pituitary	99.71	Pituitary	92.53	No tumor	80.13	Glioma	43.50
2.	No tumor	No tumor	99.98	No tumor	100	Glioma	100	No tumor	71.22	No tumor	99.58
3.	Mening.	Mening.	100	Mening.	99.96	Mening.	99.71	Mening.	100	Mening.	99.72
4.	Glioma	Mening.	94.12	Pituitary	38.62	Pituitary	99.11	No tumor	95.43	No tumor	95.24

Table 4.8. Predictions after fine tuning the models

Sr. No.	Original Class	MobileNetV2		EfficientNetB1		NASNetMobile		InceptionV3		DenseNet121	
		Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)	Predicted Class	Conf. (%)
1.	Pituitary	Pituitary	100	Pituitary	99.26	Pituitary	99.73	Pituitary	98.31	Pituitary	99.43
2.	No tumor	No tumor	100	No tumor	100	Glioma	100	No tumor	99.85	No tumor	100
3.	Mening.	Mening.	100	Mening.	100	Mening.	99.98	Mening.	100	Mening.	100
4.	Glioma	Mening.	99.64	Mening.	33.28	Pituitary	99.54	No tumor	82.97	No tumor	98.93

Images used for the above purpose have been shown through the following figures.

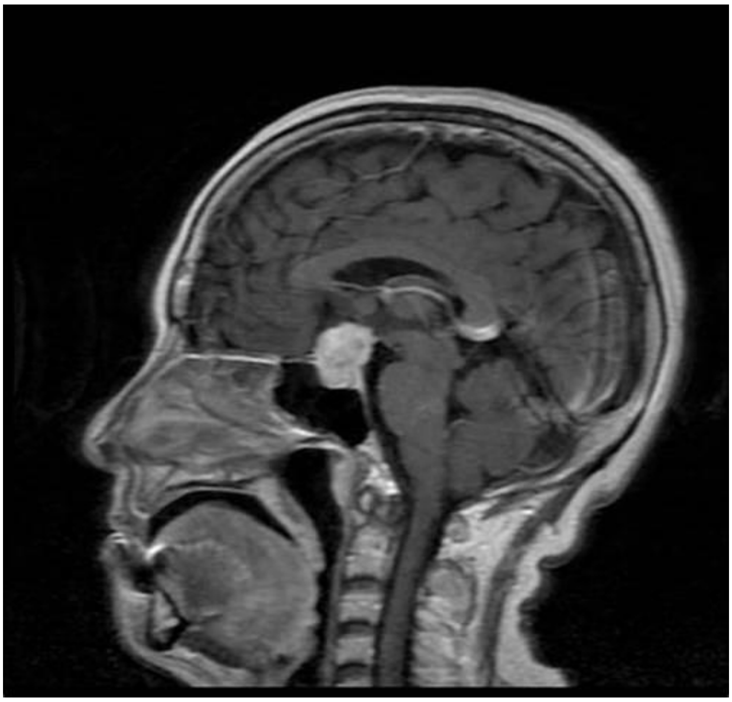


Figure 4.51. Test image – Pituitary class

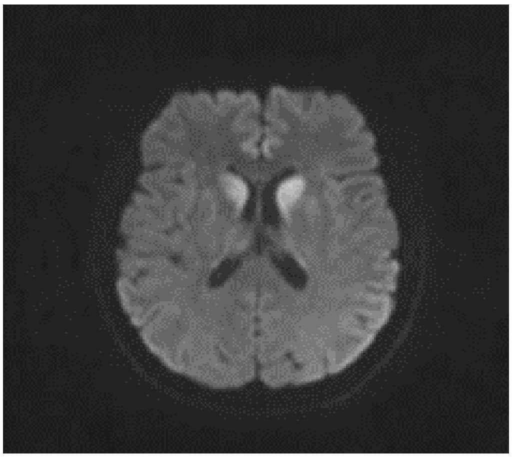


Figure 4.52. Test image – No Tumor class

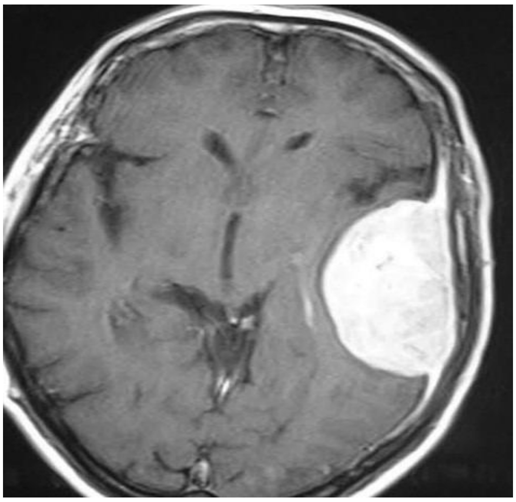


Figure 4.53. Test image – Meningioma Class

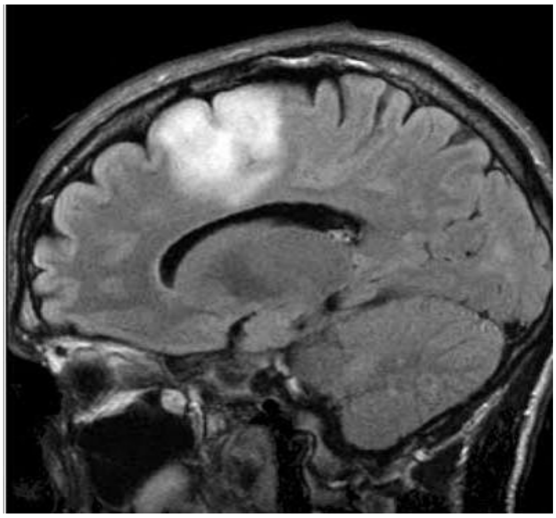


Figure 4.54. Test image – Glioma class

CHAPTER 5: RESULTS AND EVALUATION

5.1 RESULTS

On the basis of Testing Accuracy, EfficientNetB1 model proved to be the best out of all the three models implemented and tested.

The testing was carried out in a rigorous manner and thus, evaluation was not just based on the accuracy. In terms of class of tumors, i.e. Glioma, Meningioma, Pituitary and No Tumor, class specific performance analysis was done with the help of Classification Reports. The summary is presented below in the Table 5.1.

Table 5.1. Analysis based on specific class of tumors

	Best Model w.r.t. F-1 Score
Glioma	EfficientNetB1
Meningioma	MobileNetV2
No Tumor	EfficientNetB1/NASNetMobile
Pituitary Tumor	MobileNetV2

The prime objective of this project work was to use light-weight architectures and complete the task in minimum possible time. Also, it is clear by the findings presented that all the three models have very competitive performance. So in such a case, it also becomes important to compare the consumption of time by all the models. This has been presented in Table 5.2.

Table 5.2. Comparison of Time Consumption

	Time per epoch (s)
MobileNetV2	270
EfficientNetB1	600
NASNetMobile	390
InceptionV3	650
DenseNet121	810

CHAPTER 6: CONCLUSIONS AND FUTURE SCOPE

6.1 CONCLUSION

The key findings of this project work are based on two major parameters – Performance and Cost.

Hence, on the basis of performance in terms of accuracy, EfficientNetB1 has been found to be the best model for classification of brain tumors. On the other hand, in terms of different classes and their corresponding F-1 scores, MobileNetV2 and EfficientNetB1 are comparable.

From the angle of cost, MobileNetV2 were easily outperforms other models in terms of time as well as space consumption. Time and space requirements of MobileNetV2 are almost near to 30-35% less than the corresponding highest figures.

Since the MobileNetV2 also gives a tough competition in terms of performance, it can be considered an appropriate model for deployment if the performance requirements are not very stringent. Else, EfficientNetB1 is the best model in any case.

This project work has made an immense contribution to the field of medical diagnosis with the help of CAD systems. Until now, the systems used for such purpose were very costly in terms of time and space requirements, but this project work eliminates this drawback of the current systems in place.

A possible limitation of this project works is that the performance of the models can be enhanced further to the best possible accuracy.

6.2 FUTURE SCOPE

In the presented work through this project, while fine tuning, 54 last layers out of 154 were made trainable in MobileNetV2, 90 last layers out of 340 in EfficientNetB1, 169 last layers out of 769 in NASNetMobile, 61 last layers out of 311 in InceptionV3 and 177 last layers out of 427 in DenseNet121. Future Scope can be to increase the number of trainable layers and try to enhance the performance. The concept of layers freezing, according to which only specific layers are freezed to use pre-trained weights and others are trainable, can be leveraged, provided the availability of requisite computational power. Also, as a part of the future scope, the extensiveness of the dataset can further be enhanced. This particular project has faced difficulties in the classification of Glioma class of tumors. The dataset related enhancements can cater to the increased correctness and accuracy in the predictions.

REFERENCES

[1] https://www.mayoclinic.org/diseases-conditions/brain-tumor/symptoms-causes/syc-20350084#:~:text=A%20brain%20tumor%20can%20form,headaches%2C%20nausea%20and%20balance%20problems.

[2] https://www.hopkinsmedicine.org/health/conditions-and-diseases/brain-tumor

[3] https://en.wikipedia.org/wiki/Brain_tumor

[4] E. Dandıl, M. Çakıroglu, and Z. Ekşi, ‘‘Computer-aided diagnosis of malign and benign brain tumors on MR images,’’ in ICT Innovations: Advances in Intelligent Systems and Computing, vol. 311. Cham, Switzerland: Springer, 2015, pp. 157–166

[5] R. Kumar, J. Kakarla, B. Isunuri, M. Singh, “Multi-class brain tumor classification using residual network and global average pooling” in Multimedia Tools and Applications, 2019, doi: 80. 10.1007/s11042-020-10335-4.

[6] K. Simonyan, A. Zisserman, “Very deep convolutional networks for large-scale image recognition” in arXiv:1409.1556v6, 2014, https://doi.org/10.48550/arXiv.1409.1556

[7] N. Abhiwinanda, M. Hanif, S.T. Hesaputra, A. Handayani, T.R. Mengko, “Brain Tumor Classification Using Covolutional Neural Network” in World College on Medical Physics and Biomedical Engineering IFMBE Proceedings, vol 68/1, Springer, Singapore, 2019, doi: https://doi.org/10.1007/978-981-10-9035-6_33

[8] J. Cheng. “Brain Tumor Dataset (Version 5)”, retrieved from https://doi.org/10.6084/m9.figshare.1512427.v5

[9] B. Ahmad, J. Sun, Q. You, V. Palade, Z. Mao, “Brain Tumor Classification Using a Combination of Variational Autoencoders and Generative Adversarial Networks” in Biomedicines, 2022, 10, 223. https://doi.org/10.3390/biomedicines10020223

[10] P. Afshar, K. N. Plataniotis and A. Mohammadi, "Capsule Networks for Brain Tumor Classification Based on MRI Images and Coarse Tumor Boundaries," ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2019, pp. 1368-1372, doi: 10.1109/ICASSP.2019.8683759.

[11] R. Chelghoum, A. Ikhlef, A. Hameurlaine, and S. Jacquir, "Transfer learning using convolutional neural network architectures for brain tumor classification from MRI images," in 2020 17th International Conference on Information Technology and Applications (ICITA), 2020, pp. 1-5, doi: 10.1007/978-3-030-49161-1_17.

[12] S. Kokkalla, J. Kakarla, I. B. Venkateswarlu, M. Singh, “Three-class brain tumor classification using deep dense inception residual network” in Soft Computing, 2021, https://doi.org/10.1007/s00500-021-05748-8

[13] A. Gumaei, M. Hassan, M. Hassan, A. Alelaiwi, G. Fortino, “A Hybrid Feature Extraction Method With Regularized Extreme Learning Machine for Brain Tumor Classification” in IEEE Access, 2019, 10.1109/ACCESS.2019.2904145.

[14] A. Ari and D. Hanbay, ‘‘Deep learning based brain tumor classification and detection system’’ in Turkish J. Elect. Eng. Comput. Sci., vol. 26, no. 5, pp. 2275–2286, 2018.

[15] A. Oliva and A. Torralba, ‘‘Modeling the shape of the scene: A holistic representation of the spatial envelope,’’ in Int. J. Comput. Vis., vol. 42, no. 3, pp. 145–175, 2001.

[16] A. Gumaei, R. Sammouda, A. Malik, S. Al-Salman, and A. Alsanad, ‘‘An improved multispectral palmprint recognition system using autoencoder with regularized extreme learning machine,’’ in Comput. Intell. Neurosci., vol. 2018, May 2018, Art. no. 8041609.

[17] A. Gumaei, R. Sammouda, A. Malik, S. Al-Salman, and A. Alsanad, ‘‘Antispoofing cloud-based multi-spectral biometric identification system for enterprise security and privacy-preservation,’’ in J. Parallel Distrib. Comput., vol. 124, pp. 27–40, Feb. 2019.

[18] G. B. Huang, H. Zhou, X. Ding, and R. Zhang, ‘‘Extreme learning machine for regression and multiclass classification,’’ in IEEE Trans. Syst., Man, Cybern. B. Cybern., vol. 42, no. 2, pp. 513–529, Apr. 2012

[19] G. B. Huang, Q. Y. Zhu, and C. K. Siew, ‘‘Extreme learning machine: Theory and applications,’’ in Neurocomputing, vol. 70, pp. 489–501, Dec. 2006

[20] Anaraki, A. Kabir, M. Ayati and F. Kazemi. “Magnetic resonance imaging-based brain tumor grades classification and grading via convolutional neural networks and genetic algorithms.” in Biocybernetics and Biomedical Engineering (2019).

[21] M. Sajjad, S. Khan, K. Muhammad, W. Wu, A. Ullah, S. W. Baik, “Multi-grade brain tumor classification using deep CNN with extensive data augmentation” in Journal of Computational Science, Volume 30, 2019, Pages 174-182, ISSN 1877-7503, https://doi.org/10.1016/j.jocs.2018.12.003.

[22] Z. N. K. Swati, Q. Zhao, M. Kabir, F. Ali, Z. Ali, S. Ahmed and J. Lu, “Brain tumor classification for MR images using transfer learning and fine-tuning” in Computer Methods in Medicine and Imaging, 2019, 1-13.

[23] S. Deepak, P.M. Ameer, “Brain tumor classification using deep CNN features via transfer learning”, in Computers in Biology and Medicine, Volume 111, 2019, 103345, ISSN 0010-4825, https://doi.org/10.1016/j.compbiomed.2019.103345.

[24] P. Afshar, A. Mohammadi and K. N. Plataniotis, "BayesCap: A Bayesian Approach to Brain Tumor Classification Using Capsule Networks," in IEEE Signal Processing Letters, vol. 27, pp. 2024-2028, 2020, doi: 10.1109/LSP.2020.3034858.

[25] M. Toğaçar, B. Ergen, Z. Cömert, “BrainMRNet: Brain tumor detection using magnetic resonance images with a novel convolutional neural network model” in Medical Hypotheses, Volume 134, 2020, 109531, ISSN 0306-9877, https://doi.org/10.1016/j.mehy.2019.109531.

[26] S. Bhuvaji, A. Kadam, P. Bhumkar, S. Dedge, S. Kanchan, “Brain Tumor Classification (MRI)” [Dataset]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/1183165

[27] Br35H dataset link: https://www.kaggle.com/datasets/ahmedhamada0/brain-tumor-detection?select=no

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.

[29] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks” in Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, 9-15 June 2019, 6105-6114.

[30] B. Zoph, V. Vasudevan, J. Shlens and Q. V. Le, "Learning Transferable Architectures for Scalable Image Recognition," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 8697-8710, doi: 10.1109/CVPR.2018.00907.

[31] Keras API reference - https://keras.io/api/applications/

[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, June 2016, pp. 2818-2826.

[33] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, July 2017, pp. 2261-2269.