A PROJECT REPORT

ON

BRAIN TUMOUR DETECTION USING GAN AND CNN

Submitted in partial fulfilment of the requirement

for the award of the degree

of

BACHELOR OF TECHNOLOGY (B.Tech)

in

Information Technology

by

Rajat Goyal

189302034

Information Technology

MANIPAL UNIVERSITY JAIPUR

JAIPUR – 303007

RAJASTHAN, INDIA

JUNE 2022

DEPARTMENT OF INFORMATION TECHNOLOGY

MANIPAL UNIVERSITY JAIPUR, JAIPUR – 303 007 (RAJASTHAN), INDIA

Date: 13/06/2022

CERTIFICATE

This is to certify that the project titled **BRAIN TUMOUR DETECTION USING GAN AND CNN** is a record of the bonafide word done by **RAJAT GOYAL** (189302034) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **Information Technology of Manipal University Jaipur, during the academic year 2021-22**.

Mr. Ravinder Kumar

Project Guide, Dept of Information Technology

Manipal University Jaipur

Dr. Pankaj Vyas

HOD, Dept of Information Technology

Manipal University Jaipur

ACKNOWLEDGEMENTS

I acknowledge my sincere thanks to Dr. Rajveer Singh Shekhawat, Director of School of Computing and Information Technology at Manipal University Jaipur, and Dr. Pankaj Vyas, HOD of Department of Information Technology for their contributions to the completion of my project. It is a pleasure to extend my deep gratitude to my internal guide **Mr. Ravinder Kumar**, for his valuable guidance and support in completing my project. I thank him for the valuable suggestions towards my project, which helped me in making the project more efficient. We brain stormed multiple project ideas in the beginning of semester, and he helped me come up with a project which not only has an application in the welfare of people but enabled me to learn and expand my technological sphere. He has been readily available wherever I got stuck amidst the project and pulled me through various bugs. I am truly thankful to his efforts and concern towards me.

I am very grateful to my peers as well for their cooperation and insights. I discussed a lot of different concepts and we talked rather it is reasonable and possible to apply those or not. Their suggestions really helped me clear out my mind.

I thank and acknowledge each and everyone's efforts that helped me in some or other way in small and significant things.

My heartiest thanks to all.

ABSTRACT

Brain tumour is occurrence of formation of abnormal cells inside the brain. There are different types of symptoms and pain to the patient because of these tumours making their lives miserable. As a result, developing a fast, reliable and automated tool for brain tumour detection is a crucial and urgent activity that could aid in detecting it in early stages and saving many precious lives.

I analyzed the images of MRI scans of brain using **convolutional neural networks**. It is widely recognized and implemented for image classification. I implemented my own made architecture as well as pre-built CNN models via transfer learning. I also used the concept of **Generative Adversarial Networks** to make the dataset more extensive and robust.

The model ended up securing a validation accuracy **of 94.7%.** The GAN also generated **indistinguishable images** of the original dataset as intended and couldn't be told apart by eyes of a commoner.

This project encompasses around concepts like deep learning, artificial neural network, convolutional neural network, transfer learning, parameters & hyperparameters, generative adversarial networks, image processing, computer vision and medical imaging.

LIST OF TABLES

LIST OF FIGURES

Contents

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

Brain tumour is occurrence of formation of abnormal cells inside the brain. There are two types of tumours: malignant and benign. There are two types of cancerous tumours: one originates inside the brain called primary tumours and secondary tumours which originate somewhere else in the body and spread across. There are different types of symptoms and pain to the patient because of these tumours making their lives miserable. As a result, developing a fast, reliable and automated tool for brain tumour detection is a crucial and urgent activity that could aid in detecting it in early stages and saving many precious lives.

Brain tumour is a life-threatening condition. There are many symptoms to it like headaches, seizures, anxiety, loss in vision and balance, etc. Sometimes there are no symptoms making it harder to detect. There are various forms of surgical treatments like chemotherapy, but it is needed to be done in earlier stages. Hence this is where I extract my motivation as I aspire to successfully implement a model which will automatically process the MRI scans and classify whether the brain has tumour or not. All it will need is MRI scans of the brain as input potentially fastening the process of diagnosis and a huge help to not just the patient but the doctors as well. It will require knowledge of neural networks and computer vision. I aim to extensively improve my intellectual skills by learning these rigorous concepts through this project and hopefully contribute towards the cause.

1.2 OBJECTIVES OF THE PROJECT

The project aims to detect whether a person has malignant or benign tumour in their brain. The following is a summary of the planned study's innovation and originality:

- Data scraping and splitting
- Image pre-processing
- Image augmentation
- Implementing GAN to make the dataset robust and extensive
- Building CNN architecture for the proposed dataset and applying various pre-build CNN models
- Making sense of the metrics and result obtained from the proposed methodology

CHAPTER 2: BACKGROUND MATERIAL

2.1 CONCEPTUAL OVERVIEW

2.1.1 Machine Learning

Machine Learning is a branch of research concerned with comprehending and developing methods that 'learn,' that is, ways that use data to enhance performance on a set of tasks. Machine Learning algorithms construct a model using training data in order to make predictions without being manually administered to do so. Machine Learning algorithms are utilised in a wide range of applications, including computer vision, speech recognition, email filtering, spam detection, etc.

Components used in Machine Learning:

- **Model:** A model is a precise representation that a machine learning approach discovers from data. A model is also known as a hypothesis.
- **Feature:** A unique quantifiable attribute of our data is referred to as a feature. A feature vector is a convenient way to specify a set of numeric attributes. Feature vectors are fed into the model as input.
- **Target:** The value that our model is designed to predict is called a target variable, often known as a label.
- **Training:** The purpose is to offer a set of inputs and projected outputs such that after training, we can use the model to map new data to one of the trained categories.
- **Prediction:** We can feed our model a set of inputs and it will predict the outcome after it is complete.

2.1.2 Deep Learning

Deep learning is a subtype of machine learning that consists of three or more layers of a neural network. These neural networks seek to mimic the activity of the human brain, although they fall far short of its capabilities in terms of learning from massive amounts of data. While a single-layer neural network may produce approximate predictions, additional hidden layers can help to optimise and improve for accuracy.

2.1.3 Neural Network

Artificial neural networks (ANNs) are a subset of deep learning. They are at the foundation of deep learning techniques. Their name and structure are based on the human brain, and they function similarly to organic neurons.

2.1.4 Multi-Layered Perceptron

A multi-layered perceptron is made up of interconnected layers of perceptrons (MLP). The input layer is responsible for collecting input patterns. In the output layer, input patterns can be mapped to classifications or output signals.

Hidden layers fine-tune the input weightings until the neural network's margin of error is as little as possible. Hidden layers are supposed to deduce significant elements from the input data that can predict the outputs. This is how feature extraction works, and it's similar to how statistical techniques such as principal component analysis function.

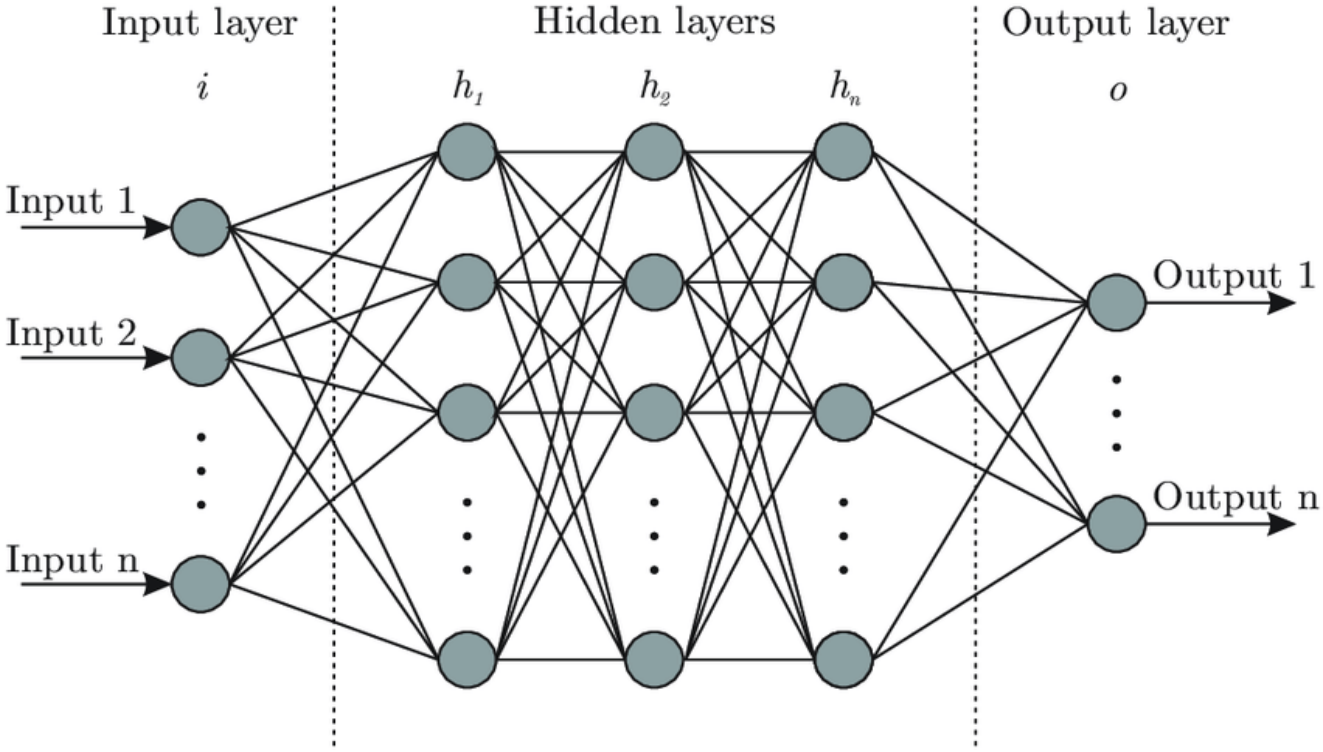


Figure 2.1 Multi-layered Perceptron

2.1.5 COMPUTER VISION

Computer vision is used in Machine Learning and AI to train the model to detect particular patterns and store the data in their artificial memory, which they may then use to predict the consequences in real-life situations.

The goal of using computer vision technologies in machine learning and artificial intelligence is to construct a model that can work without human involvement. The entire procedure include obtaining data, processing, analysing, and comprehending digital images in order to use them in a real-world context.

2.1.6 MEDICAL IMAGING

Medical imaging refers to a group of technologies that are used to visualise the human body in order to diagnose, monitor, and treat medical conditions. Each type of technology provides different types of information on the physiological part being studied or treated, such as sickness, injury, or the effectiveness of medical treatment.

2.2 TECHNOLOGIES INVOLVED

- **Python:** It's a high-level programming language that's dynamically semantic, interpreted, and object-oriented. Its high-level built-in data structures, as well as dynamic type and dynamic binding, make it exceptionally trustworthy.

- **NumPy:** NumPy is a Python package that makes it possible to work with arrays. There are additionally matrices, linear algebra and fourier transforms included.

- **Matplotlib:** Matplotlib is a cross-platform data visualisation and graphical charting package for Python and its numerical extension NumPy.

- **OpenCV:** It's a computer vision and machine learning software library. It was intended to provide a common framework for computer vision applications and to make machine perception easier to understand.

- **Scikit Learn:** For the Python programming language, Scikit-learn is a popular machine learning package. Scikit-learn is a collection of machine learning tools that comprises mathematical, statistical, and general-purpose algorithms that may be used to build a variety of machine learning technologies. Scikit-learn is a free programme that may be used to build many different sorts of algorithms for machine learning and related technologies.

- **TensorFlow:** It's a free, end-to-end platform for creating Machine Learning applications. It's a symbolic math toolbox for deep neural network training and inference that uses dataflow and differentiable programming to address a variety of applications. Machine learning applications can be created with a range of tools, libraries, and community resources.

- **Keras:** It's a deep learning API written in Python that works on top of the TensorFlow machine learning platform. It was made with the intention of facilitating speedy experimentation. It provides key abstractions and building blocks for constructing and launching machine learning systems with high iteration rates. The scalability and cross-platform capabilities of TensorFlow are fully utilised.

- **JupyeterLab:** It's the most up-to-date interactive development environment on the web for notebooks, code, and data. Its flexible interface allows users to construct and organise workflows in data science, scientific computing, computational journalism, and machine learning. A modular architecture encourages extensions to extend and enrich functionality.

- **Google Colab:** It's a product of Google Research. Colab is a Python editor for the web that allows anyone to develop and run Python programmes. Machine learning, data analysis, and education are all areas where it comes in handy.

CHAPTER 3: METHODOLOGY

3.1 ARTIFICIAL NEURAL NETWORKS (ANNs)

3.1.1 WHAT IS ARTIFICAL NEURAL NETWORK?

Artificial neural networks (ANNs) are computer systems that are modelled after biological neural networks seen in animal brains. Artificial neurons, which are made up of a collection of connected units or nodes that loosely resemble the neurons in a biological brain, make up an ANN. Each link, like synapses in the brain, can send a signal to other neurons. An artificial neuron receives and analyses impulses before sending them to other neurons. The output of each neuron is generated by a non-linear function of the total of the inputs, and each link's "signal" is a real number.

3.1.2 COMPONENTS IN ANN

- **Nodes**: Each node has a weight and a threshold connected with it. If a node's output exceeds a certain threshold, the node is activated, and data is sent to the next layer.
- **Input layer:** The input layer is where information from the outside world enters the artificial neural network. The data is processed by input nodes, which analyse or categorise it before passing it on to the next layer.
- **Hidden layer:** The input layer or other hidden layers provide input to hidden layers. A considerable number of hidden layers can be found in artificial neural networks. Each hidden layer examines the previous layer's output, processes it further, and then passes it on to the next layer.
- **Output layer:** The output layer displays the ultimate outcome of the artificial neural network's data processing. It can be made up of one or more nodes.

3.1.3 WORKING OF ANN

There is a multi-neuron input layer in our system. The neuron receives an input of x with a weight of w. The intrinsic parameter is the one over which the model has control in order to improve the output fit. When we give an input to a neuron, we multiply it by the weight of the neuron, which gives us x*w.

The bias is the second parameter in the input. Since the node's value is 1, the bias is exclusively dictated by the value b. When using testing data, the bias provides an element of unpredictability to our model, allowing it to generalise and adapt to many unforeseen inputs.

Our output y is produced by combining the bias and the input, giving us the formula w.x+b=y.

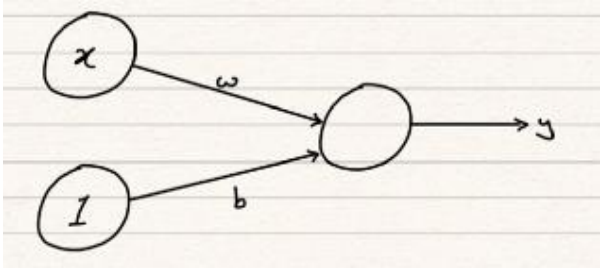


Figure 3.1.1 Neuron

The weights of a neural network are allocated at random when it is first created. The network adjusts weights and measures performance iteratively, repeating the process until the predictions are precise enough. A cost function determines the accuracy of our forecasts. The function compares the model output to actual outputs to see how well our model estimates the dataset. Essentially, we're giving our model a function to minimise. Weights are updated to do this. The optimizer is in charge of determining the best set of weights.

ANN employs two approaches to bring non-linearity into our model. The first capability is to add hidden layers between the input and output levels of our network. The number of nodes in each of these hidden layers will be predetermined, and this added complexity will begin to distinguish the neural network from its regression equivalent.

The introduction of activation functions is the second ability. It uses a non-linear approach to change our input data. After multiplying the weight and adding bias to the input value of a neuron's output, it travels through an activation function. And by using these strategies, we can improve our model's adaptability by removing the linear decision bounds.

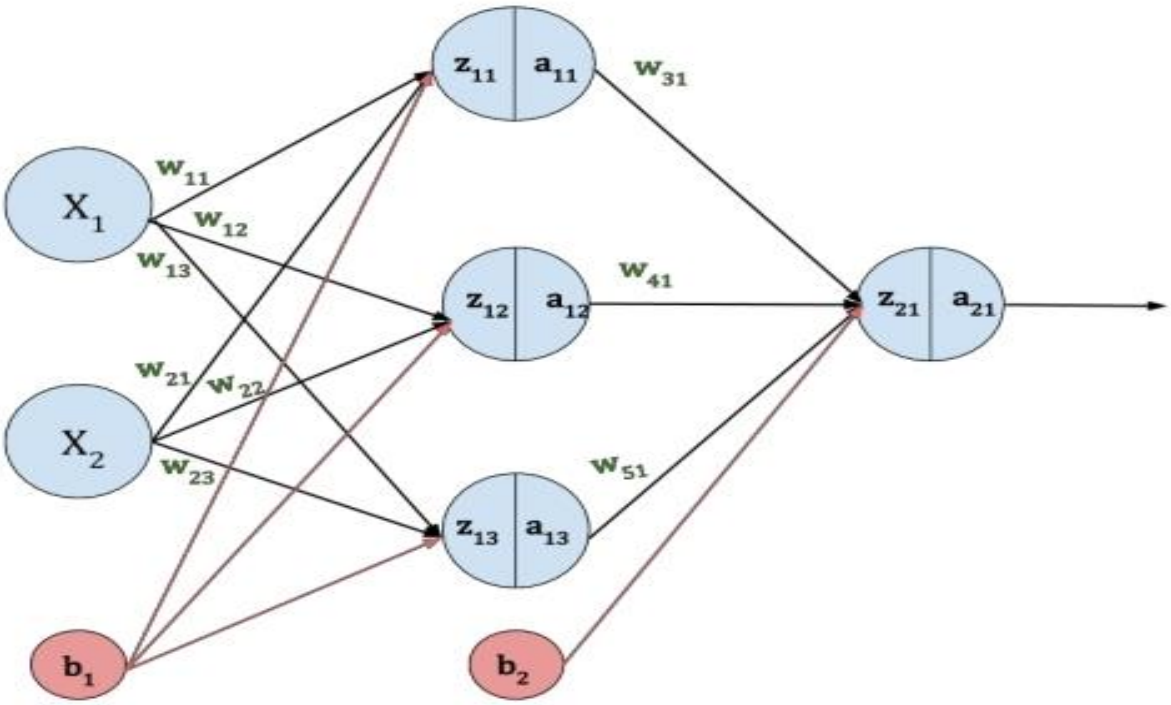


Figure 3.1.2 ANN Architecture

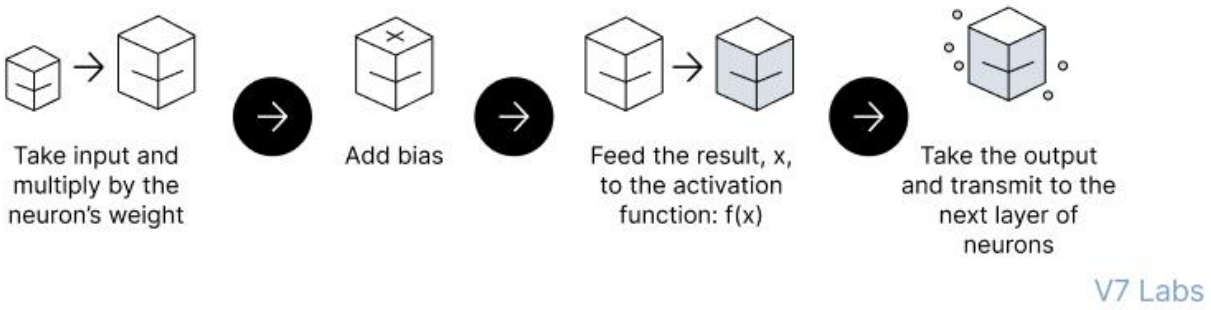


Figure 3.1.3 Computation done in an ANN layer

3.2 CONVOLUTIONAL NEURAL NETWORKS

3.2.1 WHAT IS CNN

A convolutional neural network is a deep learning system that can take an input image and give importance (learnable weights and biases) to various aspects/objects in the image, allowing it to distinguish between them. When compared to other classification techniques, the amount of pre-processing required by a Convolutional neural network is significantly less. While traditional approaches require hand-engineering of filters, convolutional neural networks can learn these filters/characteristics with adequate training.

3.2.2 WHY CHOOSE CNN OVER ANN / MLP

- When working with medium/large sized photos, MLP causes overfitting due to too many parameters (~millions).
- MLP is unable to handle image variations such as translation, rotation, lighting, and size.
- MLP considers each pixel to be a feature, whereas CNN considers regions and recognises that not all characteristics are relevant, and that the ones that are concentrated in the local region.

3.2.3 IMPORTANT TERMINOLOGIES OF CNN

- **Padding:** It refers to the number of pixels added to an image when it is being processed by the kernel of CNN. Padding aids kernel in image processing, allowing for more accurate image interpretation.

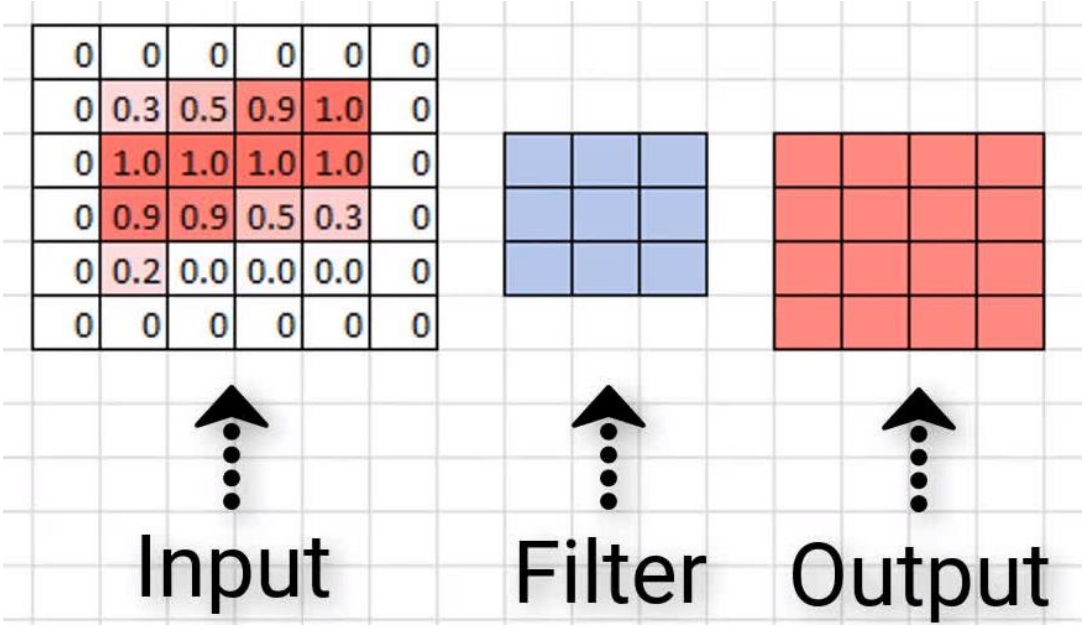


Figure 3.2.1 Padding

$$n_H = \lfloor \frac{n_{H_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$
$$n_W = \lfloor \frac{n_{W_{prev}} - f + 2 \times pad}{stride} \rfloor + 1$$

Figure 3.2.2 Dimensions of layer after padding

- **Strides:** It specifies how a kernel should traverse an image. Each time the kernel moves, the stride is the number of pixels we skip. * The height and width of the output activation map are changed by stride, but not the channel.

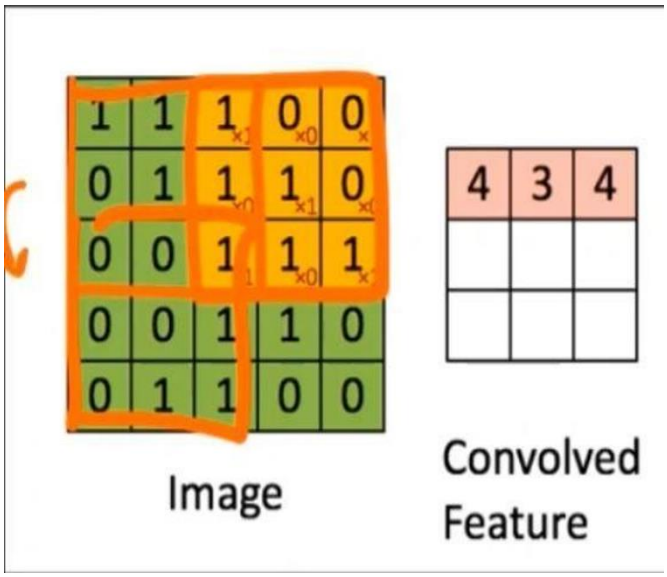


Figure 3.2.3 Stride

$$n_H = \lfloor \frac{n_{H_{prev}} - f}{stride} \rfloor + 1$$
$$n_W = \lfloor \frac{n_{W_{prev}} - f}{stride} \rfloor + 1$$
$$n_C = n_{C_{prev}}$$

Figure 3.2.4 Dimensions of layer after strides

- **Pooling layer:** After the convolution layer, the pooling layer is applied. It aids in the reduction of computation time and improves the robustness of feature detectors. There are two types of pooling layers: average and max.
    - **Avg pooling layer:** It slides an (f,f) kernel over the input and stores the average value of the kernel in the output.

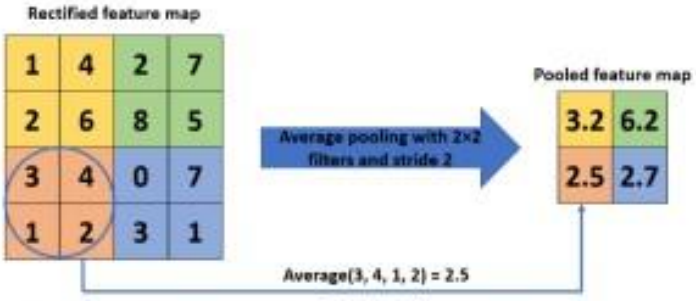


Figure 3.2.5 Avg pooling layer

- **Max pooling layer:** It slides an (f,f) kernel over the input and stores the maximum value of the kernel in the output.

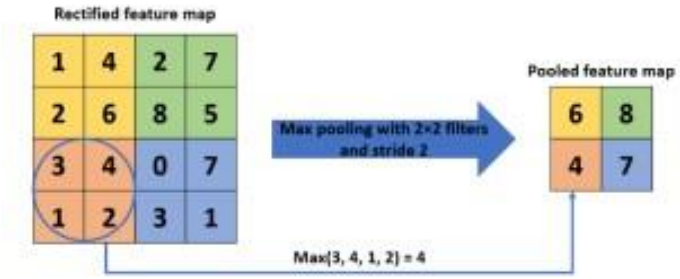


Figure 3.2.6 Max Pooling layer

- **Regularization (Dropout):** We use a technique called dropouts for regularization where randomly selected neurons are turned off during training. This means that their contribution is temporarily removed on the forward pass and any weight updates are not applied to the neuron on the backward pass so it eliminates the dependency of model on a particular neuron by randomizing the process.
- **Convolution layer:** It's the layer that performs a linear operation by multiplying a set of weights with input images represented by metrics similar to those used in classic neural networks. This layer comprises all of the arguments stated above, and the architecture of convolutional neural networks is a network of this layer.

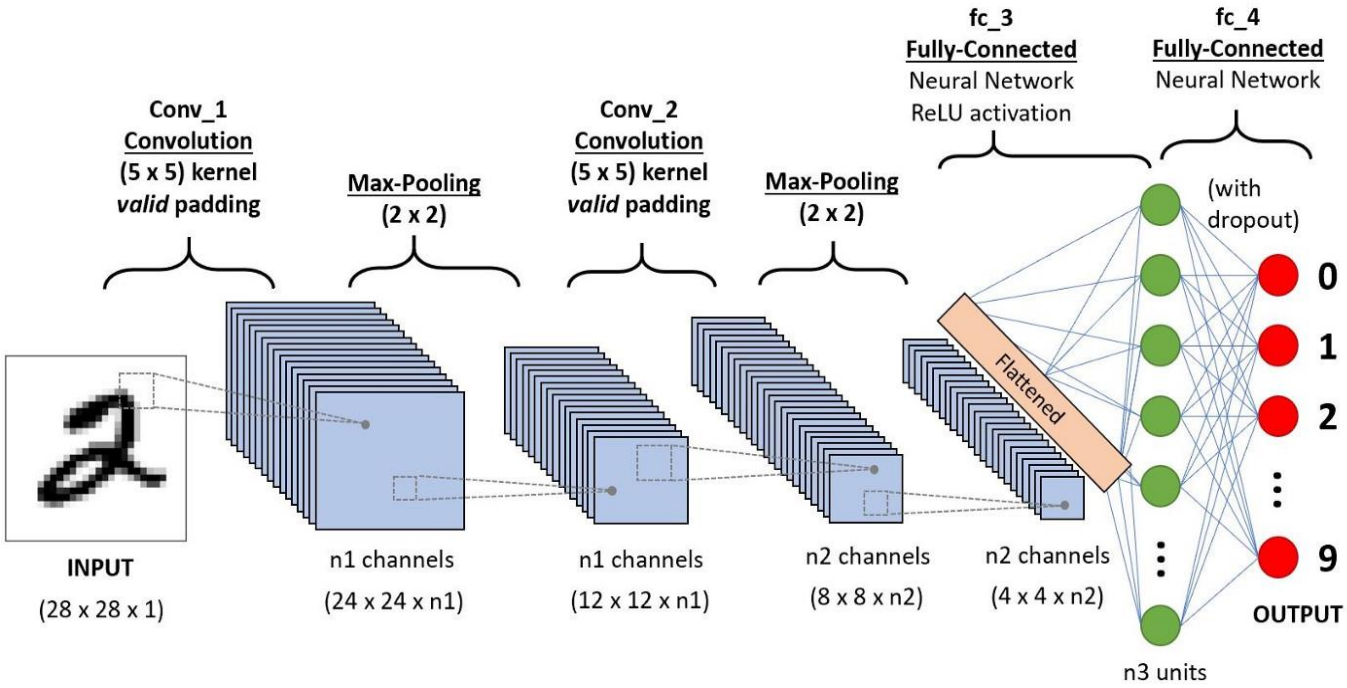


Figure 3.2.7 CNN Architecture

- **Fully connected layer:** Simply put, the fully connected layer is a feed-forward neural network. The final output of the convolution/pooling layer is supplied to it, which is flattened and then fed to the FC layer. It uses a matrix of weights to apply a linear transformation to the input vector. The output is then subjected to a non-linear transformation utilising activation functions.

- **Parameters:** They are the configuration model, which is internal to the model. The count of learnable elements for a filter is the number of parameters in a given layer. These are determined or specified during the model's training and by the model itself.

  There are two parameters in convolutional neural network:
  - Weight
  - Bias

- **Hyperparameters:** They are the parameters that are expressly stated to influence the training process. Hyperparameters are necessary for model optimization. These aren't part of the model. A practitioner performs these manually.

  Hyperparameters in convolutional neural network:
  - Number of convolutional layers
  - Number of kernels of each convolutional layer
  - Architecture of the convolutional layer
  - Learning rate: It's a tuning parameter in an optimization algorithm that determines the step size at each iteration as the algorithm moves toward the loss function's minimum.

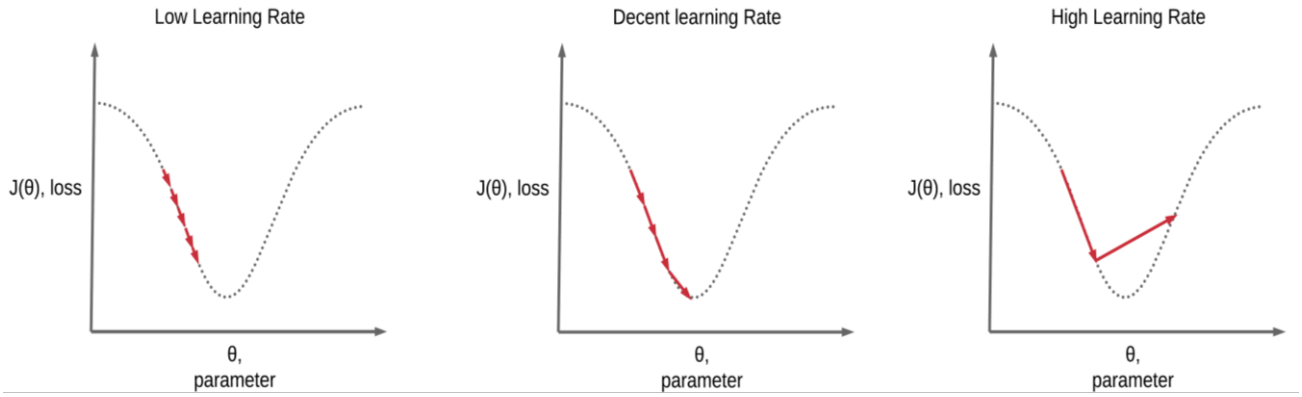


Figure 3.2.8 Learning rate convergence

  - Activation function: It specifies how a node in a network layer transforms the weighted sum of input into an output. In the process of mathematical prediction, it determines whether the input from a neuron to the network is essential or not. It's included to give the neural network some non-linearity.
    - Some common activation functions:
      - **Sigmoid:** $f(x) = \frac{1}{1+e^{-x}}$
      - **Tanh:** $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

- **Softmax:** $z_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$
- **ReLU:** $f(x) = \max(0, x)$
- **Leaky ReLU:** $f(x) = \max(0.1x, x)$

- **Backpropagation:** It is a mathematical weight function fine-tuning approach used to increase the accuracy of an artificial neural network. The loss value generated by each layer determines its accuracy. The algorithm makes a backward pass after each forward pass of the network to update the model's weight based on the loss error.
  There are two components in backpropagation:
  - **Loss functions**: It quantifies the difference between the expected outcome and the outcome produced by the machine learning model.
    - **Cross entropy:** $\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i . \log \hat{y}_i$
    - **Mean square error:** $\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y_i})^2$
    - **Hinge:** $L = \max(0, 1 - y * f(x))$
  - **Optimizers**: It is a function that modifies the attributes of the neural network such as weights and learning rate.
    - Gradient descent
    - Stochastic gradient descent
    - RMSprop
    - Adam

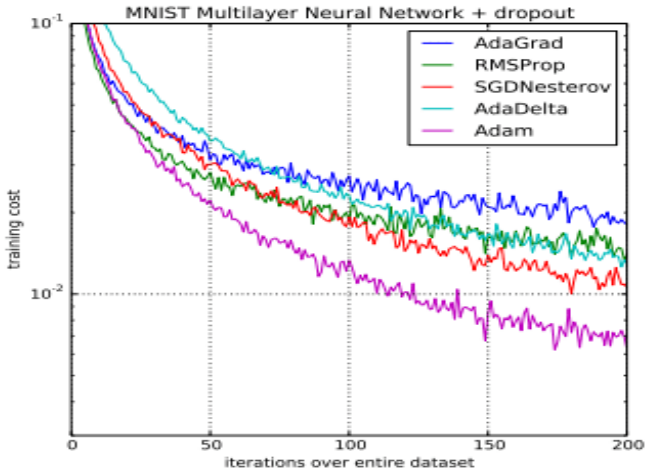


Figure 3.2.9 Optimizers convergence

- **Early stopping:** Early stopping is a model optimization strategy that reduces overfitting while maintaining model correctness. Early stopping's major goal is to terminate training before a model becomes overfit.

3.2.4 WORKING OF CNN

The model takes a picture as input and resizes it to make computations easier. Then we choose a kernel of the practitioner's choosing and slide it across the input image, multiplying its own value by the image element's value. The multiplications are added together to get a single output. When these quantities of output are added together, they form a volume known as a feature map. This feature map is provided as an input to the next layer of the architecture, and the process is continued until the completely connected layer is reached. The fully connected layer receives a flattened vector, which, like any feed-forward neural network, performs computations and produces an output at the very end.

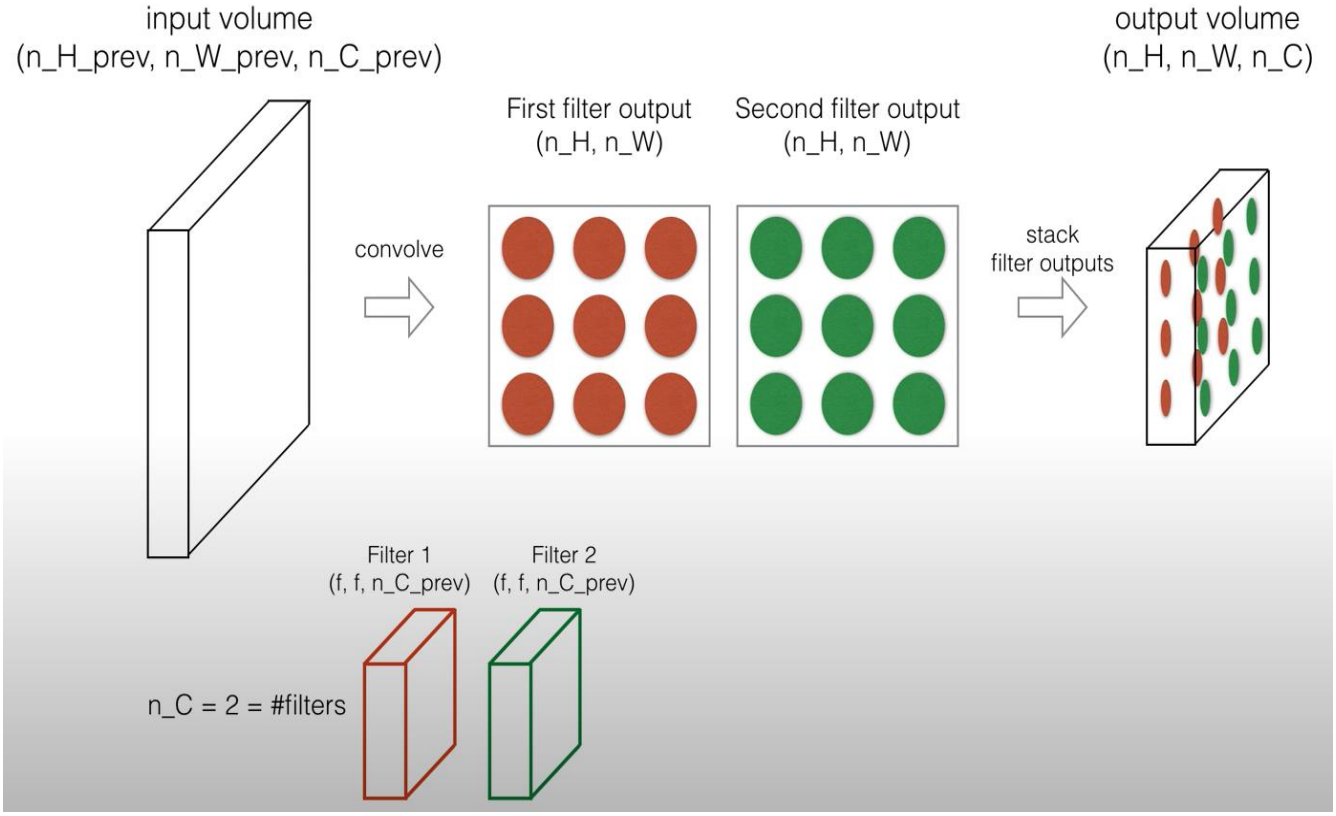


Figure 3.2.10 Working of CNN

3.3 TRANSFER LEARNING

Transfer learning is a machine learning technique in which a model that has been trained for one job is repurposed for a different task. When modelling the second task, it is an optimization that allows for faster progress or better performance. We train a base network on a base dataset and task, then repurpose or transfer the learnt features to a second target network that will be trained on a target dataset and task. This method is more likely to succeed if the characteristics are generic, i.e., applicable to both the base and target tasks, rather than being specific to the base job.

3.3 GENERATIVE ADVERSARIL NETWORKS

3.3.1 WHAT IS GAN

In machine learning, generative adversarial network is an unsupervised learning algorithm that involves automatically detecting and learning regularities or patterns in input data so that the model may be used to produce new non-distinguishable copies of the original dataset. Like any generative model, this contrasts with a discriminator as well. Generative part of GAN creates the desired image from noise and the discriminator part simply labels it.

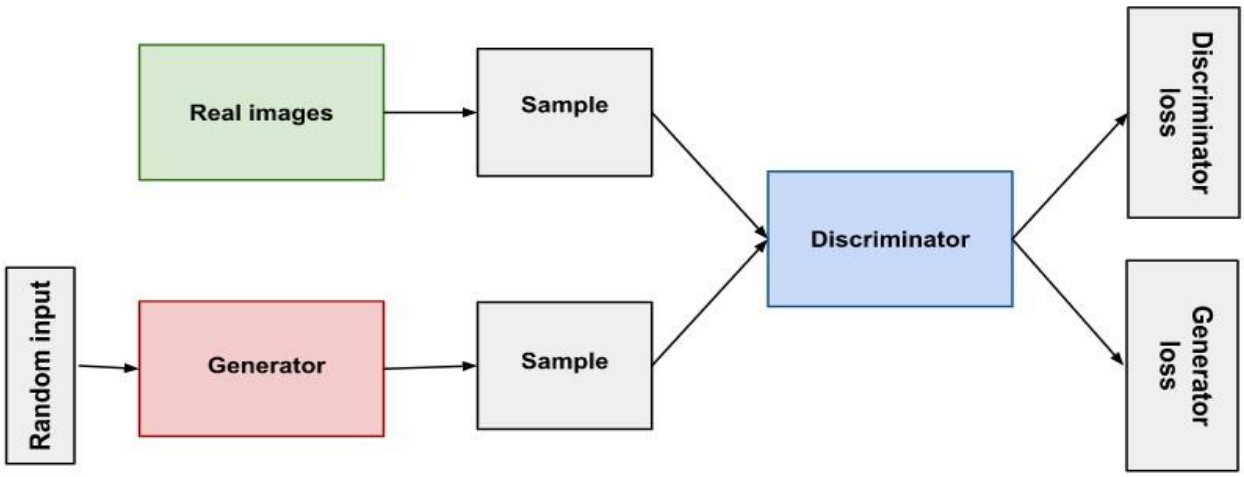


Figure 3.3.1 GAN

3.3.2 COMPONENTS OF GAN

- **Discriminator:** It is simply a classifier. It tries to differentiate between real data and data generated by the generator. Any network architecture relevant to the type of data it's classifying could be used.
    - The discriminator separates true and fake data from the generator.
    - The discriminator is penalised if a real instance is misclassified as fake or a fake instance is misclassified as real.
    - The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

- **Generator:** By incorporating input from the discriminator, it learns to generate fake data. It figures out how to get the discriminator to identify its result as real. Its training necessitates a greater degree of integration between the generator and the discriminator than discriminator training. The architecture of generator includes:
    - Random input
    - Generator network that transforms random input into relevant data
    - Discriminator network, which categorises the data generated
    - Discriminator input
    - Generator loss, which is a penalty imposed on the generator if it fails to deceive the discriminator.

As training progresses, the generator gets closer to producing output that can fool the discriminator:

Figure 3.3.2 Real life example of GAN

Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

Figure 3.3.2 Real life example of GAN

3.3.3 WORKING OF GAN

The generator samples random noise and provides an output from it as it is being taught. The output is then passed via the discriminator, which classifies it as "Real" or "Fake" based on the discriminator's ability to distinguish one from the other. The generator loss is then determined using the discriminator's categorization; if it successfully fools the discriminator, it is rewarded; otherwise, it is penalised.

The discriminator classifies both the genuine and fake data from the generator while it is being taught. By maximising a function, it penalises itself for misclassifying a real instance as fake or a fake instance (made by the generator) as real.

3.3.4 PROBLEMS WITH GAN

- **Mode collapse:** The generator collapse which produces limited varieties of samples.

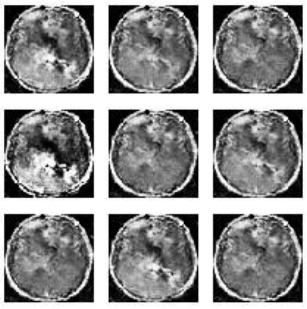


Figure 3.3.3 Mode collapse

- **Diminished gradient:** The discriminator gets too successful that the generator gradient vanishes and learns nothing.

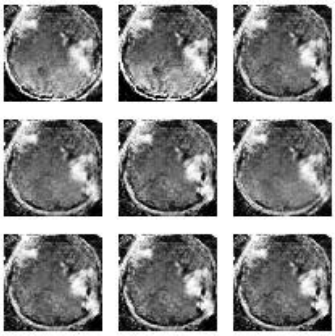


Figure 3.3.4 Diminished gradient

- **Non-Convergence:** The model parameters oscillate, destabilize and never converge.

CHAPTER 4: IMPLEMENTATION

4.1 DATASET DESCRIPTION

The image data that was used for this problem is Brain MRI images for Brain Tumour Classification. It consists of two MRI classes:

0 – Benignant

1 – Malign

Figure 4.1.1 Dataset description

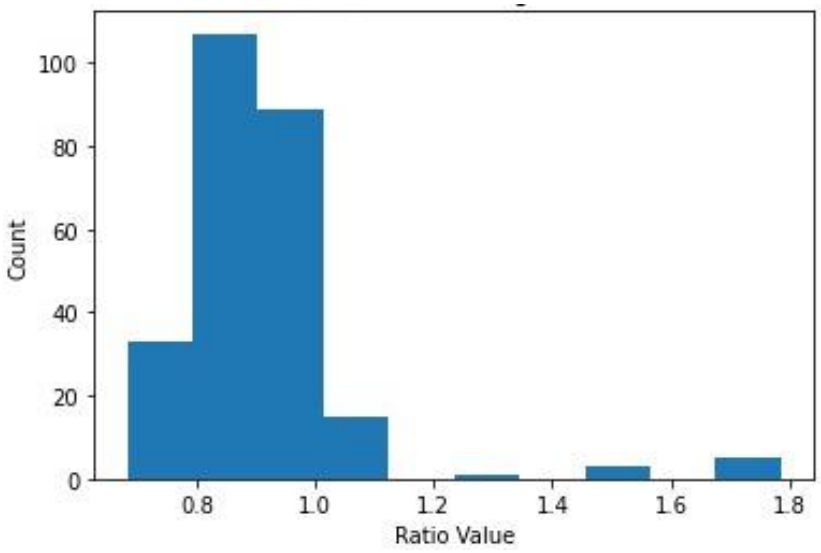


Figure 4.1.2 Distribution of ratio values

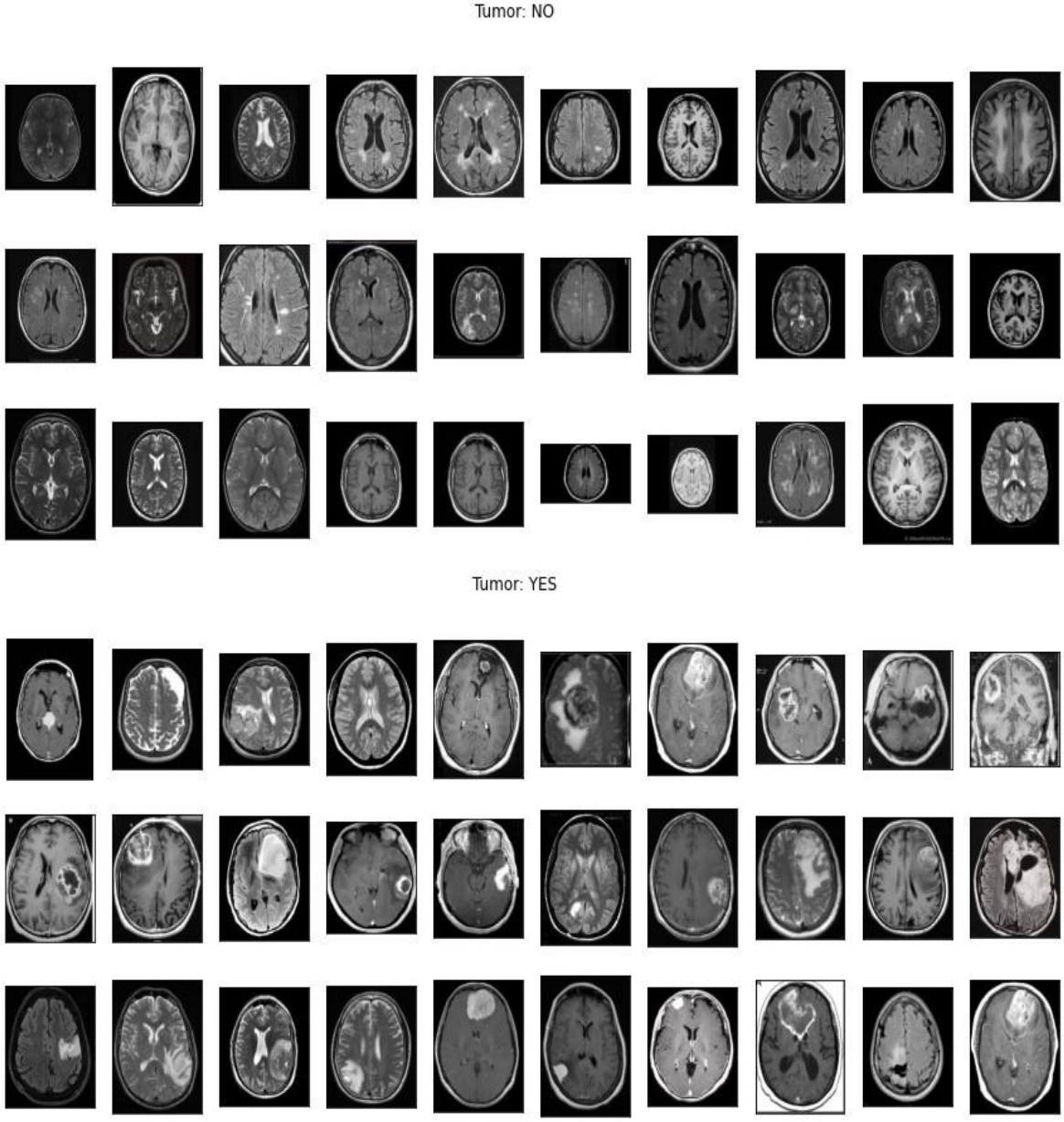


Figure 4.1.3 Dataset illustraion

4.2 IMAGE PREPROCESSING

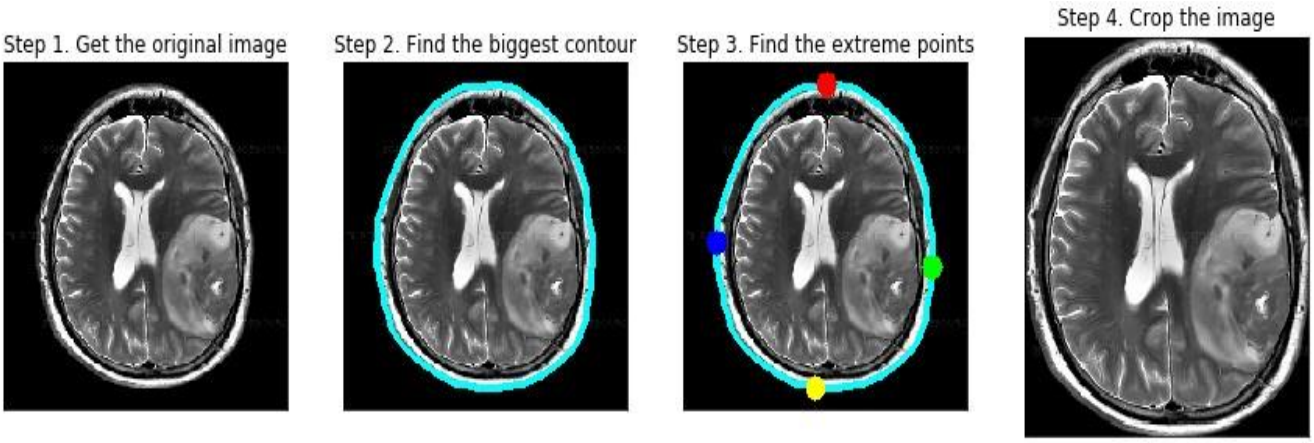


Figure 4.2.1 Cropping of images

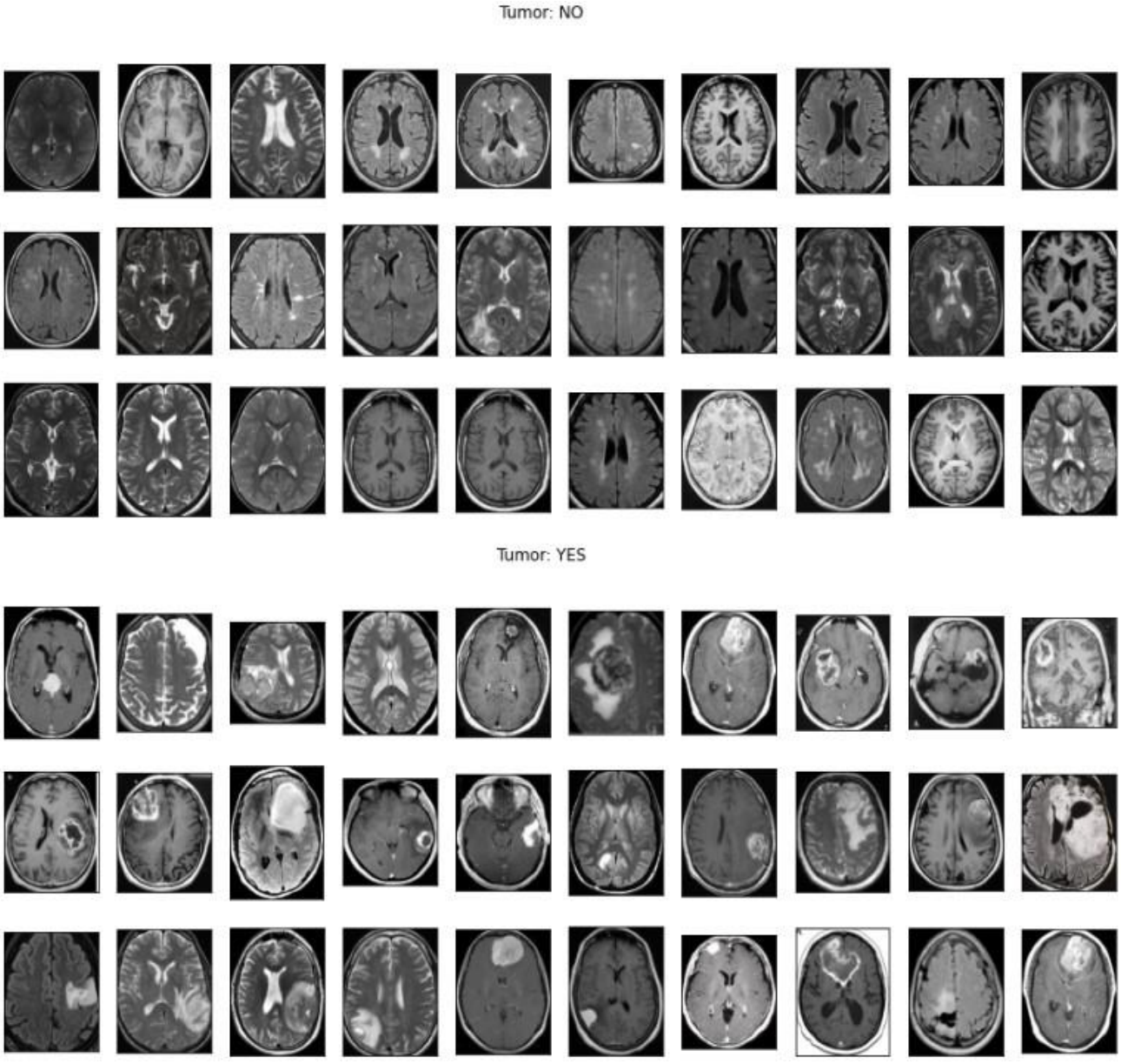


Figure 4.2.2 Cropped images

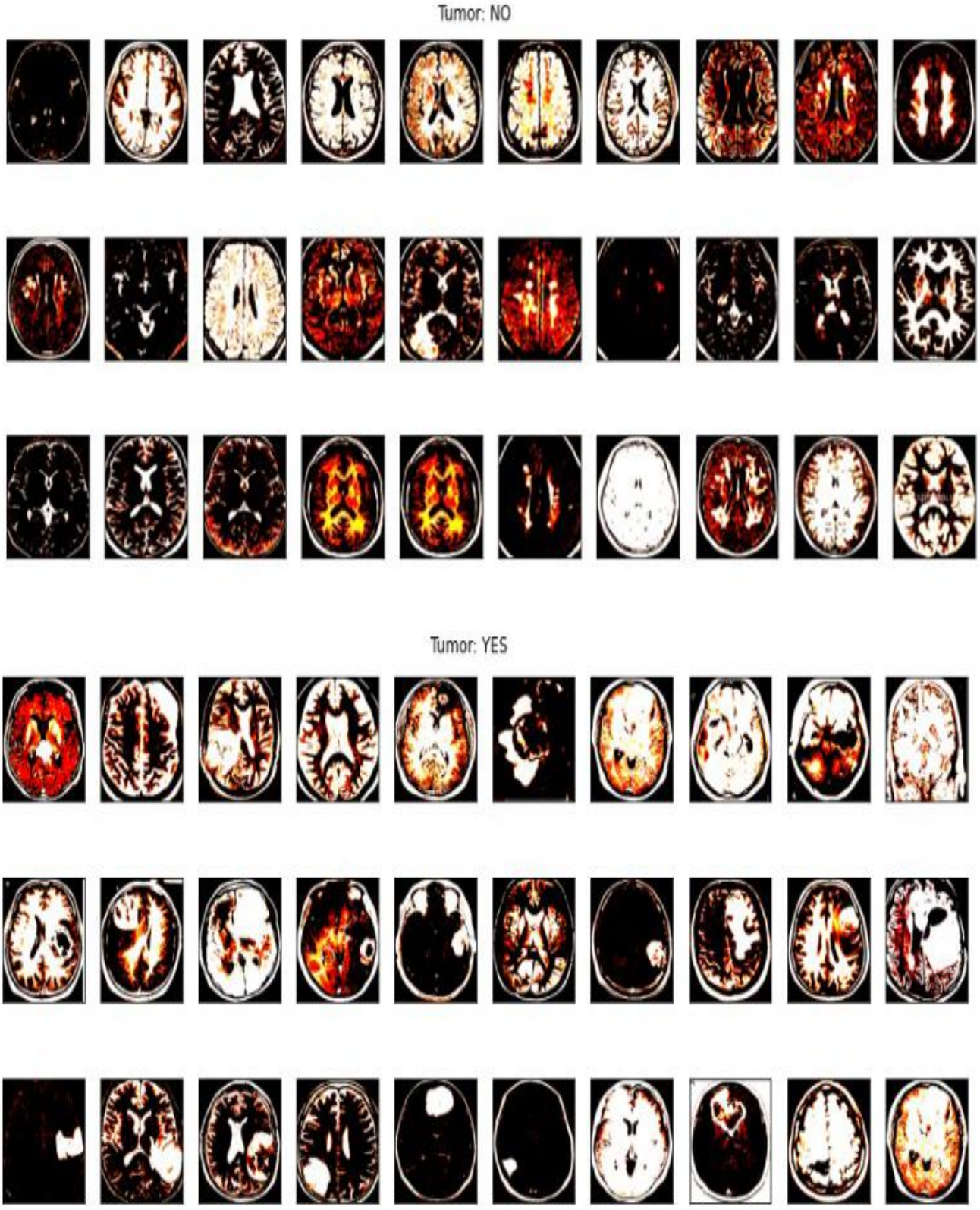


Figure 4.2.3 Preprocessing of images

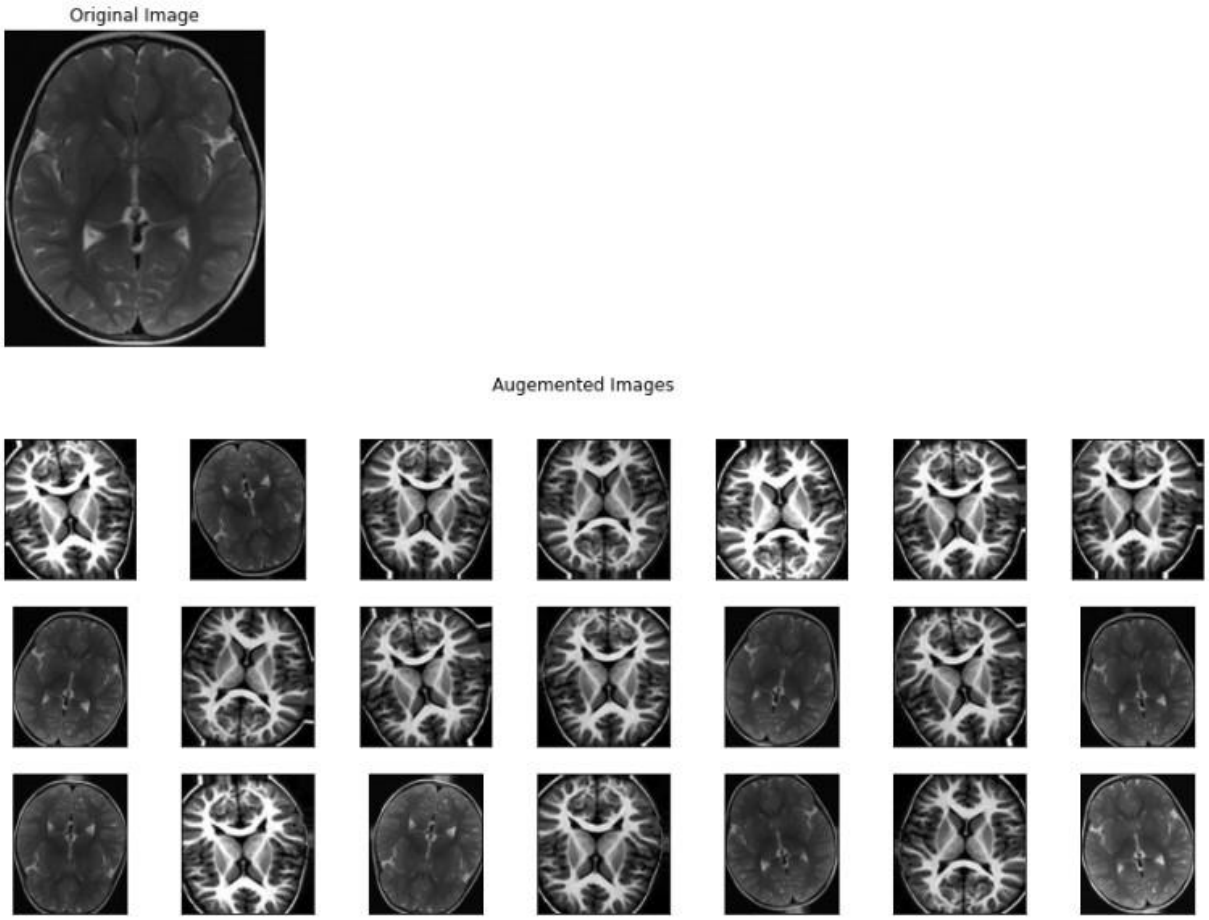


Figure 4.2.4 Augmented images

4.3 CNN ARCHITECTURE

4.3.1 SCRATCH MODEL

```
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(224,224,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
model.summary()
```

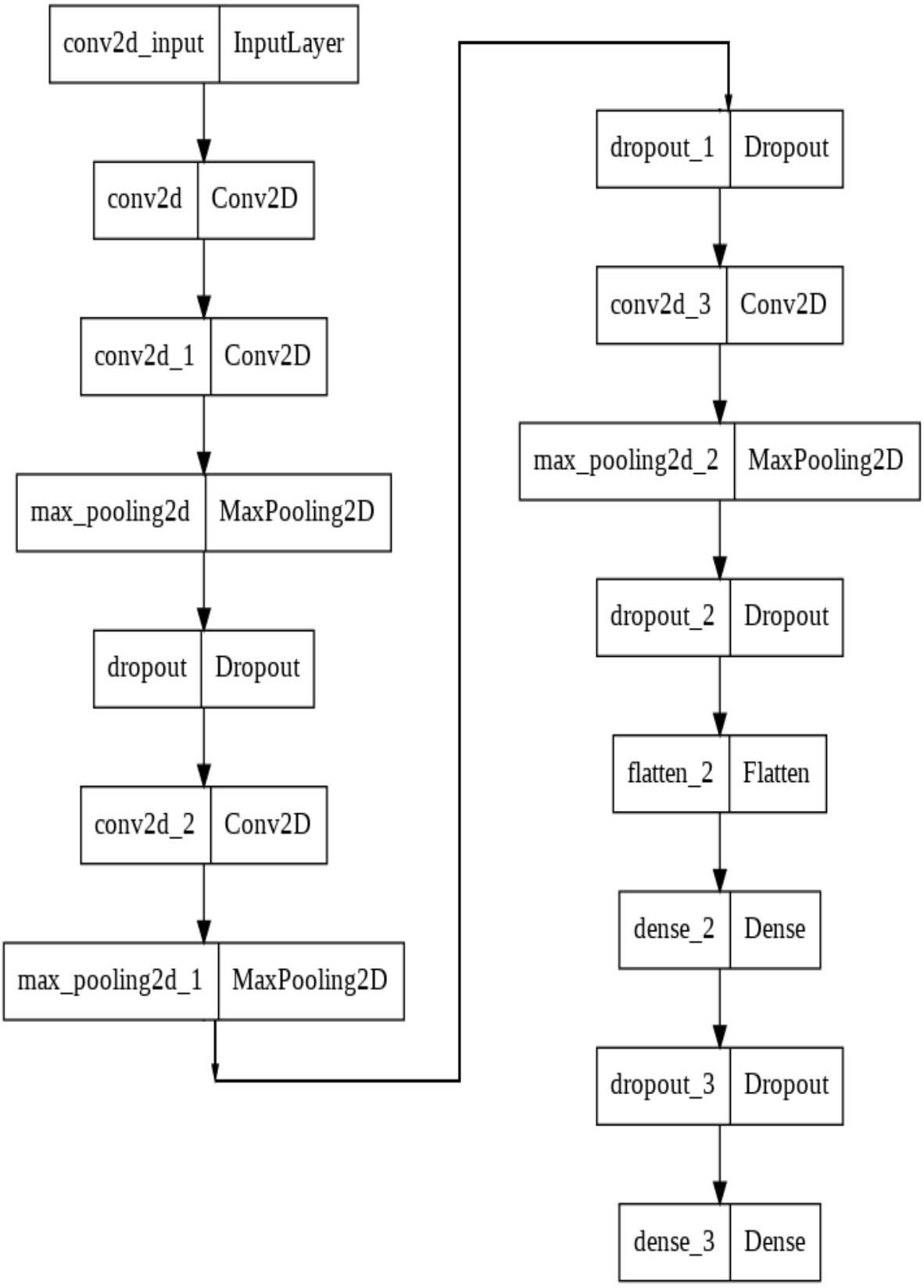


Figure 4.3.1 Scratch CNN Architecture

4.3.2 VGG-16 MODEL

```
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of resnet
vgg = VGG16(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False

l1 = Flatten()(vgg.output)
l2 = Dense(4096, activation='relu')(l1)
l3 = Dense(512, activation='relu')(l2)
prediction = Dense(1, activation='sigmoid')(l3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

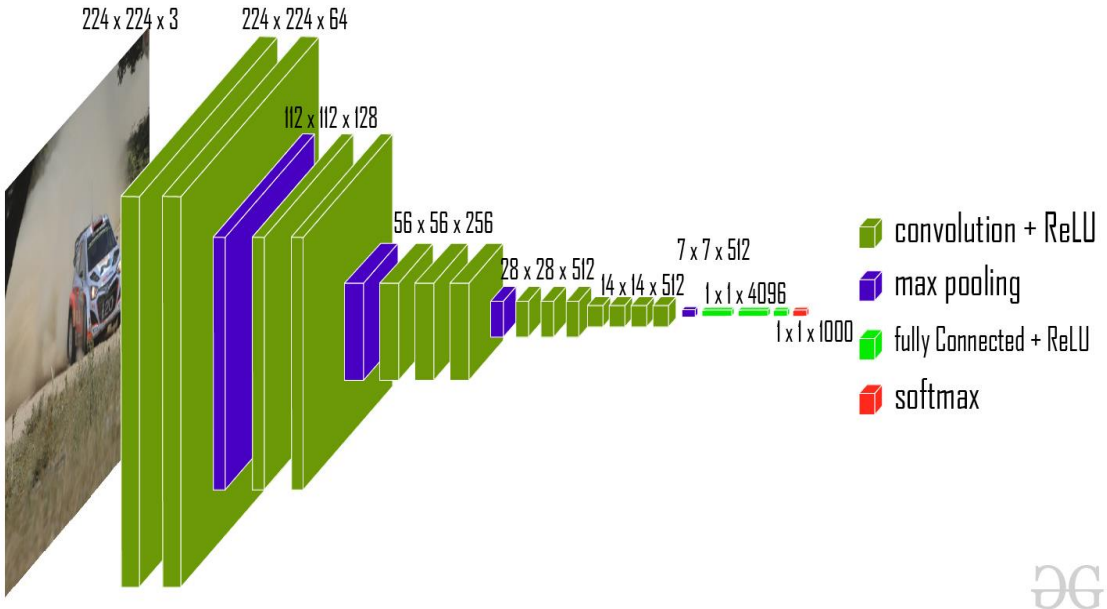


Figure 4.3.2 VGG-16 Architecture

4.3.3 ResNet-50

```
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of VGG
resnet = ResNet50(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in resnet.layers:
    layer.trainable = False

l1 = Flatten()(resnet.output)
l2 = Dense(4096, activation='relu')(l1)
l3 = Dense(512, activation='relu')(l2)
prediction = Dense(1, activation='sigmoid')(l3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

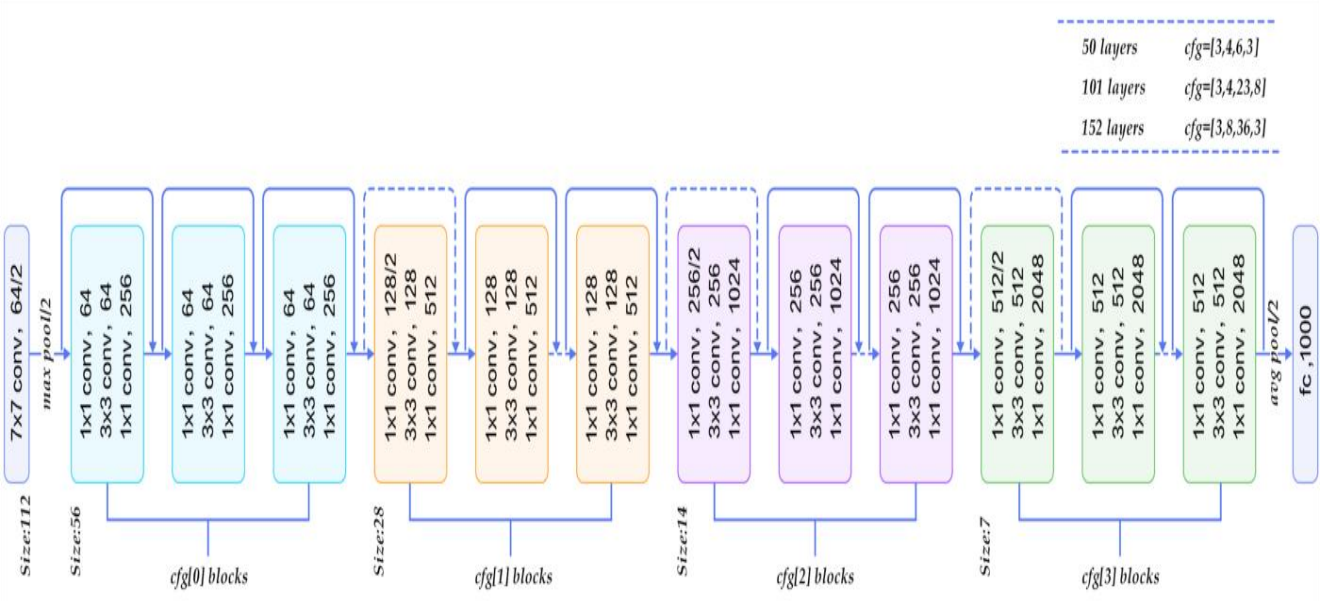


Figure 4.3.3 ResNet-50 Architecture

4.3.4 XCEPTION

```
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of VGG
xception = Xception(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in xception.layers:
    layer.trainable = False

l1 = Flatten()(xception.output)
l2 = Dense(4096, activation='relu')(l1)
l3 = Dense(512, activation='relu')(l2)
prediction = Dense(1, activation='sigmoid')(l3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

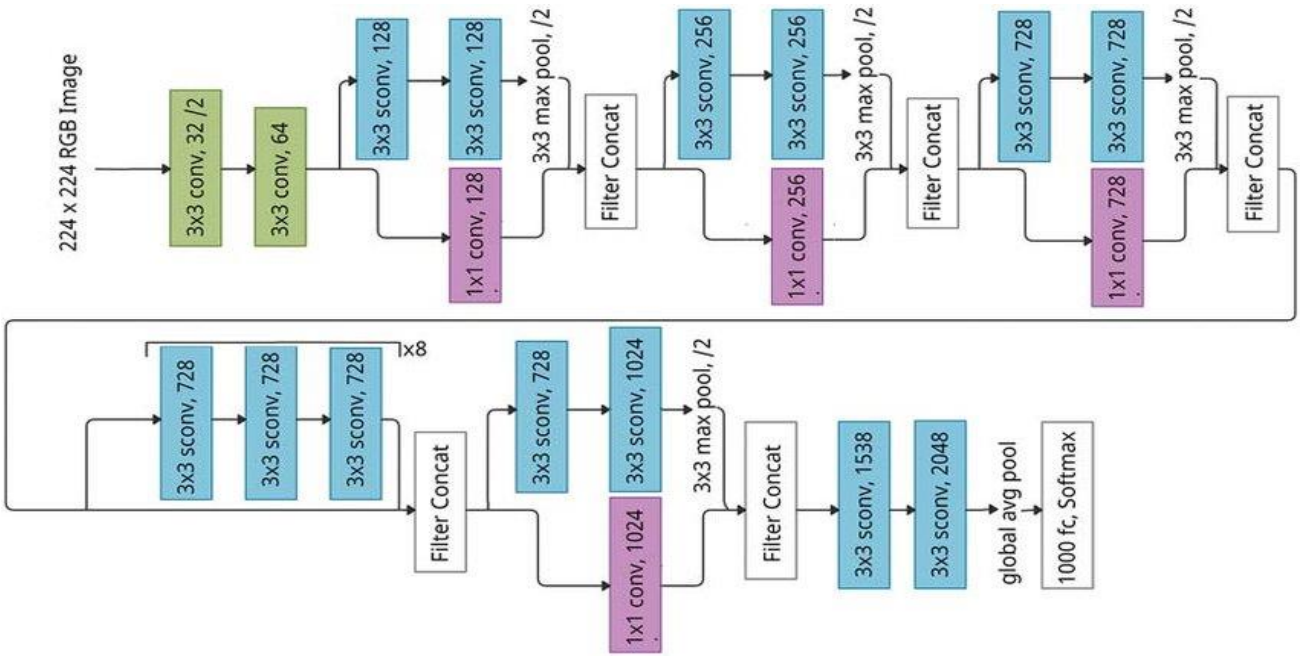


Figure 4.3.4 Xception Architecture

4.3.5 EFFICIENT NET

```
from tensorflow.keras.applications.efficientnet_v2 import EfficientNetV2L, preprocess_input
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of VGG
efficient = EfficientNetV2L(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in efficient.layers:
    layer.trainable = False

l1 = Flatten()(efficient.output)
l2 = Dense(4096, activation='relu')(l1)
l3 = Dense(512, activation='relu')(l2)
prediction = Dense(1, activation='sigmoid')(l3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

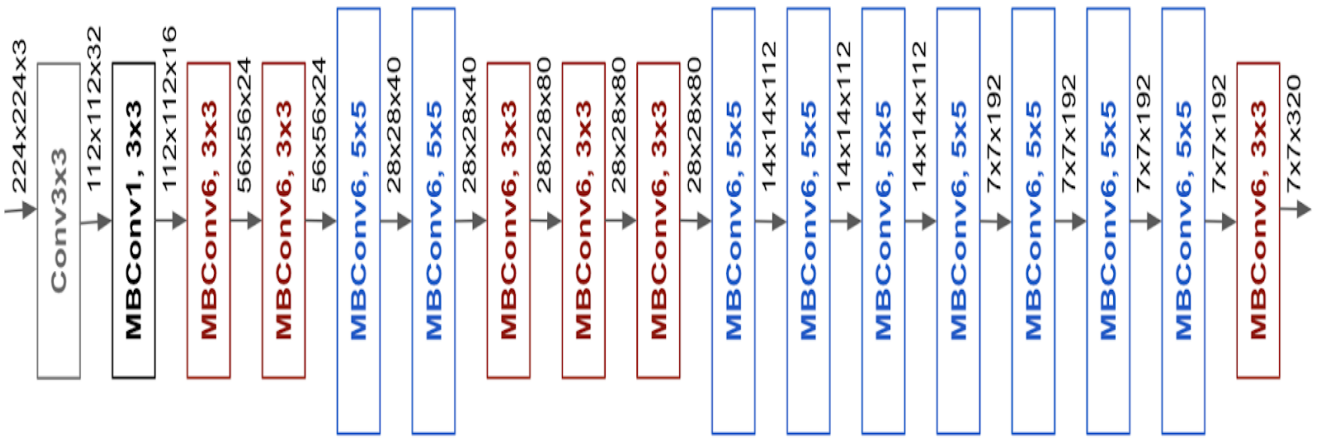


Figure 4.3.5 Efficient Net Architecture

4.3.6 ENSEMBLE MODEL (VGG16 AND RESNET50)

```
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False

resnet = ResNet50(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in resnet.layers:
    layer.trainable = False


# our layers - you can add more if you want
x1 = Flatten()(vgg.output)
x2 = Flatten()(resnet.output)

concatted = Concatenate()([x1,x2])

x3 = Dense(512, activation='relu')(concatted)
#x4 = Dense(512, activation='relu')(x3)
prediction = Dense(1, activation='sigmoid')(x3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

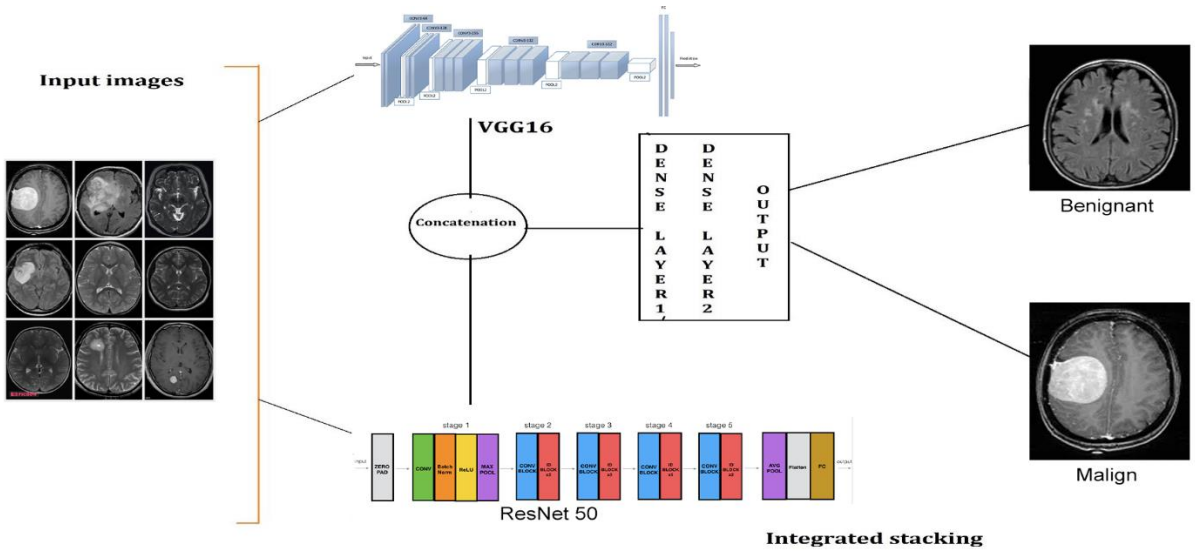


Figure 4.3.6 Ensemble (VGG16 & ResNet50) Architecture

4.3.7 ENSEMBLE MODEL (VGG16 AND EFFICIENT NET)

```
inputImage = Input(shape=(224,224,3))
# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False

efficient = EfficientNetV2L(input_shape=(224,224,3),input_tensor=inputImage,weights='imagenet', include_top=False)

# don't train existing weights
for layer in efficient.layers:
    layer.trainable = False


# our layers - you can add more if you want
x1 = Flatten()(vgg.output)
x2 = Flatten()(efficient.output)

concatted = Concatenate()([x1,x2])

x3 = Dense(512, activation='relu')(concatted)
#x4 = Dense(512, activation='relu')(x3)
prediction = Dense(1, activation='sigmoid')(x3)

# create a model object
model = Model(inputs=inputImage , outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

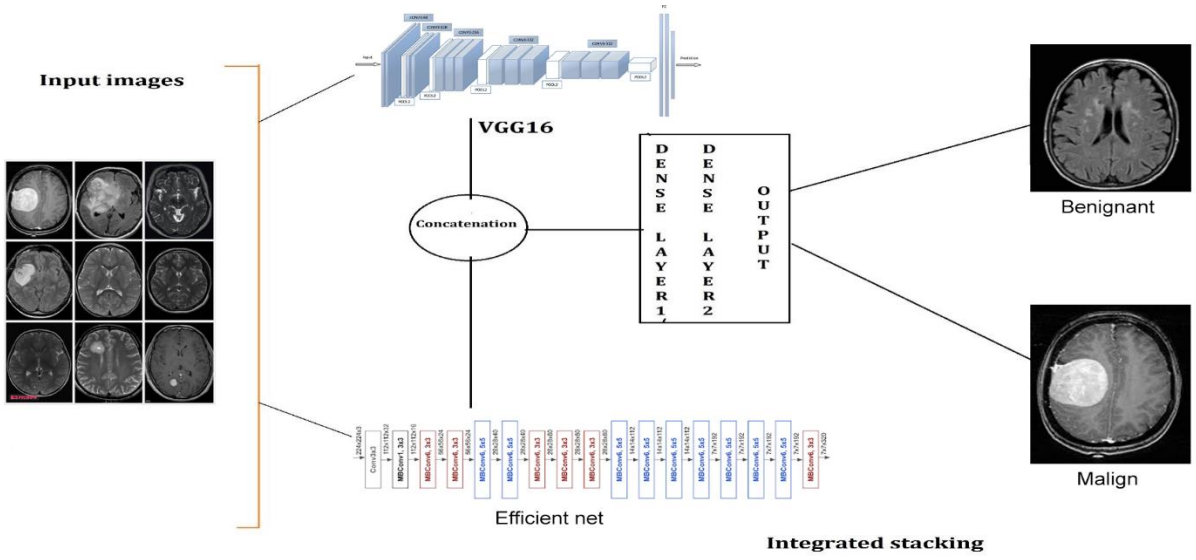


Figure 4.3.7 Ensemble (VGG16 & Efficient Net) Architecture

4.4 GENERATIVE ADVERSARIAL NETWORK ARCHITECTURE

4.4.1 GENERATOR ARCHITECTURE

```
#define the generator
generator = Sequential()
generator.add(Dense(8*8*128,input_shape=(NOISE_DIM,)))
generator.add(Reshape((8,8,128)))
generator.add(LeakyReLU(0.2))
generator.add(BatchNormalization())

#double activation size 16*16*64
generator.add(UpSampling2D())
generator.add(Conv2D(64,kernel_size=(4,4),padding='same'))
generator.add(LeakyReLU(0.2))
generator.add(BatchNormalization())

#double activation size 32*32*32
generator.add(UpSampling2D())
generator.add(Conv2D(32,kernel_size=(4,4),padding='same'))
generator.add(LeakyReLU(0.2))
generator.add(BatchNormalization())

#double activation size 64*64*1
generator.add(UpSampling2D())
generator.add(Conv2D(1,kernel_size=(4,4),padding='same',activation='tanh'))

generator.compile(loss='binary_crossentropy', optimizer='adam')
generator.summary()
```

4.4.2 DISCRIMINATOR ARCHITECTURE

```
# Downsampling
discriminator = Sequential()
discriminator.add(Conv2D(32,(4,4),strides=(2,2),padding='same',input_shape=(64,64,1)))
discriminator.add(LeakyReLU(0.2))

discriminator.add(Conv2D(64,(4,4),strides=(2,2),padding='same'))
discriminator.add(LeakyReLU(0.2))

discriminator.add(Conv2D(128,(4,4),strides=(2,2),padding='same'))
discriminator.add(LeakyReLU(0.2))

#flatten
discriminator.add(Flatten())
discriminator.add(Dense(1,activation='sigmoid'))

discriminator.compile(loss='binary_crossentropy',optimizer=adam)
discriminator.summary()
```

4.4.3 LOSS CALCULATION AND WEIGHT UPDATION

```
for epoch in range(NUM_EPOCHS):
    epoch_d_loss = 0.
    epoch_g_loss = 0.

    for step in range(NO_OF_BATCHES):
        #randomly select 50% real images
        idx = np.random.randint(0,X_Train.shape[0],HALF_BATCH_SIZE)
        real_imgs = X_Train[idx]

        # generate 50% random images
        noise = np.random.normal(0,1,size=(HALF_BATCH_SIZE,NOISE_DIM))
        fake_imgs = generator.predict(noise)


        # one sided label smoothing
        real_y = np.ones((HALF_BATCH_SIZE,1))*0.9 #Label Smoothing, Works well in practice
        fake_y = np.zeros((HALF_BATCH_SIZE,1))

        # train on real and fake images
        d_loss_real = discriminator.train_on_batch(real_imgs,real_y) #updates the weights of discriminator
        d_loss_fake = discriminator.train_on_batch(fake_imgs,fake_y)
        d_loss = 0.5*d_loss_real + 0.5*d_loss_fake

        epoch_d_loss += d_loss


        #Train Generator (Complete Model Generator + Frozen Discriminator)

        noise = np.random.normal(0,1,size=(BATCH_SIZE,NOISE_DIM))
        real_y = np.ones((BATCH_SIZE,1))
        g_loss = model.train_on_batch(noise,real_y)
        epoch_g_loss += g_loss

    print("Epoch %d D loss: %f G loss: %f" % ((epoch + 1), epoch_d_loss, epoch_g_loss))

    if (epoch+1)%1000==0:
        generator.save('/content/drive/MyDrive/Colab Notebooks/brain tumor/models2/gan_generator_{0}.h5'.format(epoch+1))
        save_imgs(epoch)
```

CHAPTER 5: RESULTS AND ANALYSIS

5.1 SCRATCH MODEL

Validation Set Measures	
Accuracy	0.614
Precision	0.000
Sensitivity	nan
Specificity	0.614
F1 score	nan

Table 5.1.1

Test set Measures	
Accuracy	0.615
Precision	0.000
Sensitivity	nan
Specificity	0.615
F1 score	nan

Table 5.1.2

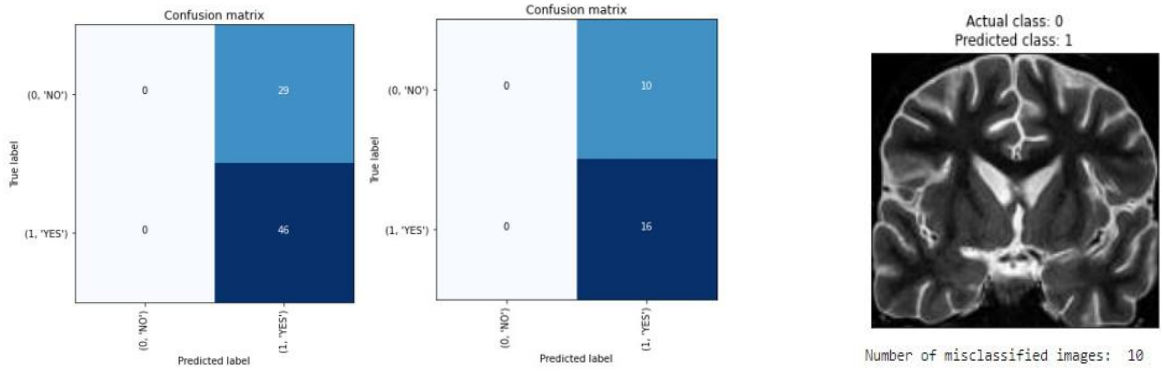
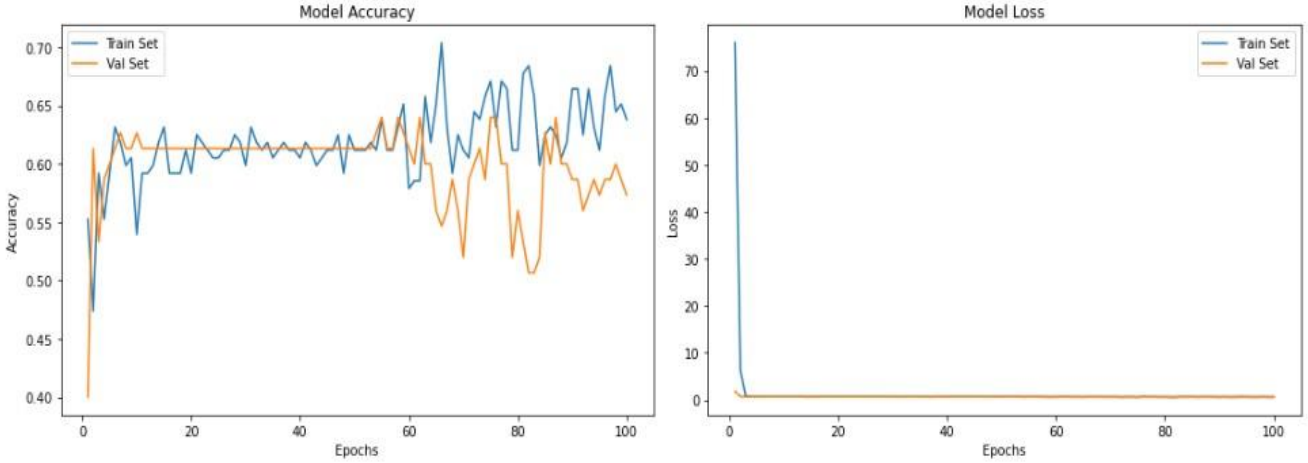


Figure 5.1 Scratch CNN no. of misclassified images



5.2 VGG-16 MODEL

Validation Set Measures	
Accuracy	0.840
Precision	0.931
Sensitivity	0.730
Specificity	0.947
F1 score	0.818

Table 5.2.1

Test Set Measures	
Accuracy	0.962
Precision	1.000
Sensitivity	0.909
Specificity	1.0
F1 score	0.952

Table 5.2.2

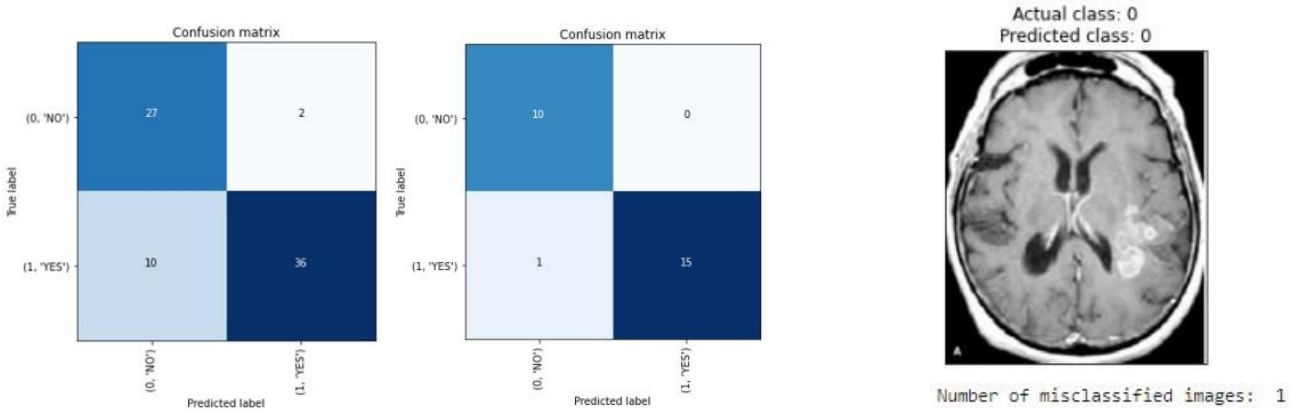
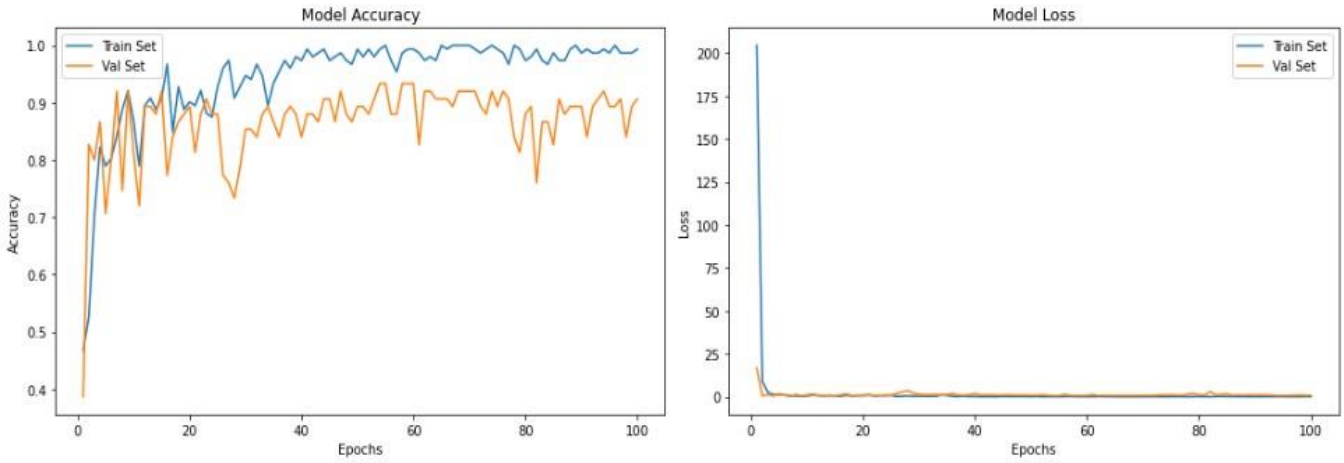


Figure 5.2 VGG-16 no. of misclassified images



5.3 ResNet-50 MODEL

Validation Set Measures	
Accuracy	0.827
Precision	0.586
Sensitivity	0.945
Specificity	0.789
F1 score	0.723

Table 5.3.1

Test Set Measures	
Accuracy	0.962
Precision	0.9
Sensitivity	1.0
Specificity	0.941
F1 score	0.947

Table 5.3.2

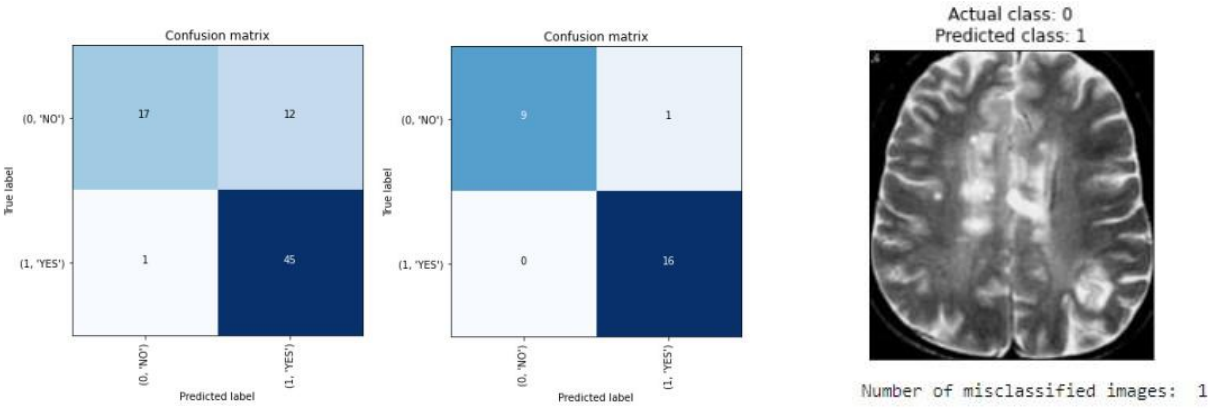
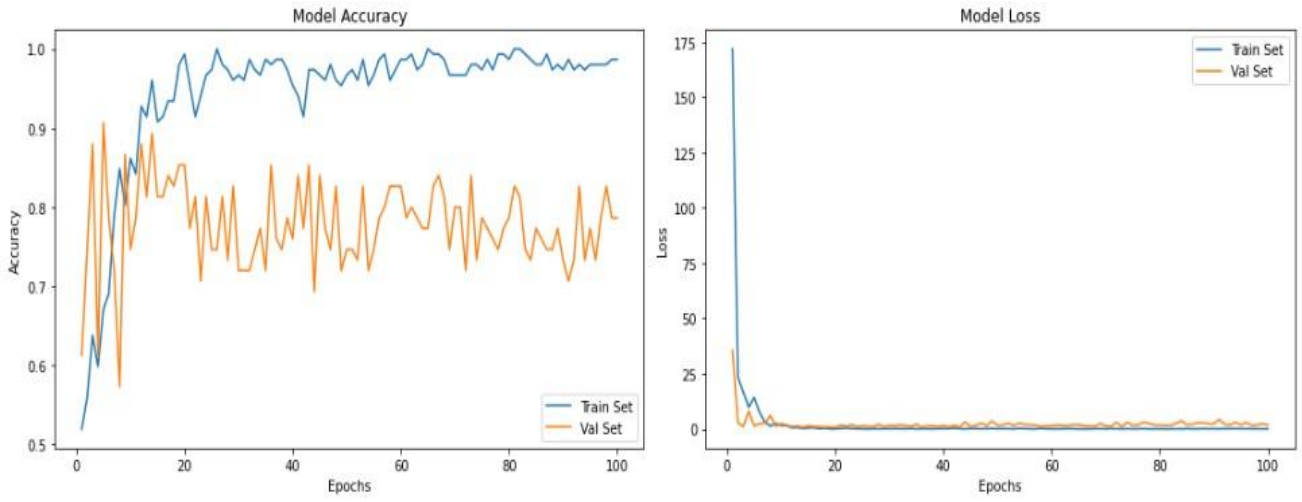


Figure 5.3 ResNet-50 no. of misclassified images



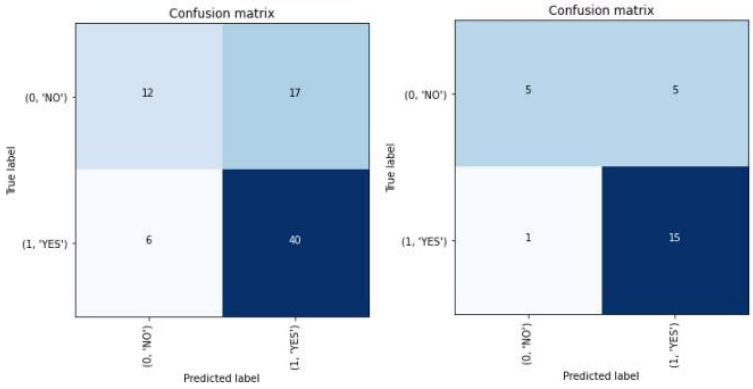
5.4 Xception MODEL

Validation Set Measures	
Accuracy	0.694
Precision	0.413
Sensitivity	0.667
Specificity	0.712
F1 score	0.511

Table 5.4.1

Test Set Measures	
Accuracy	0.769
Precision	0.500
Sensitivity	0.834
Specificity	0.750
F1 score	0.625

Table 5.4.2



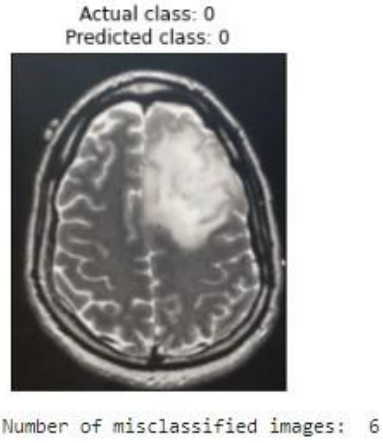
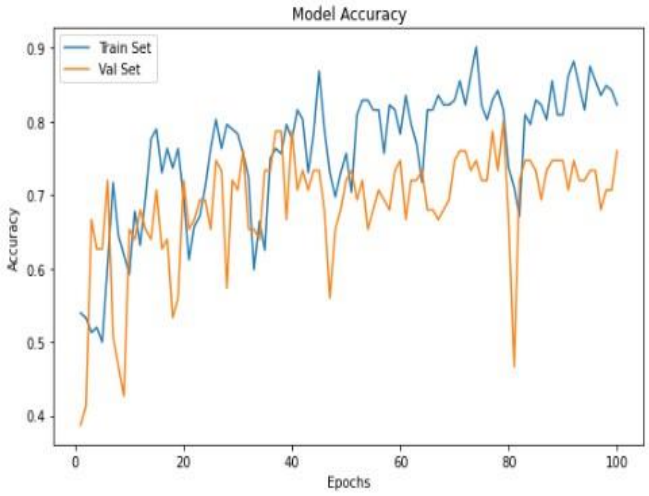
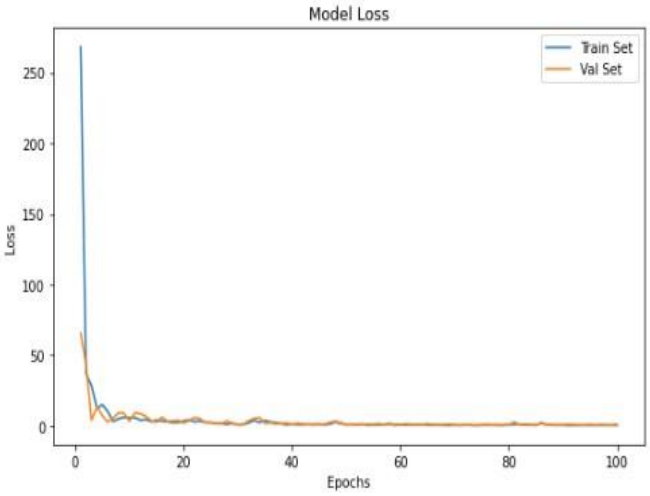


Figure 5.4 Xception no. of misclassified images





5.5 Efficient Net MODEL

Validation Set Measures	
Accuracy	0.92
Precision	0.966
Sensitivity	0.848
Specificity	0.976
F1 score	0.903

Table 5.5.1

Test Set Measures	
Accuracy	0.923
Precision	0.900
Sensitivity	0.900
Specificity	0.9375
F1 score	0.900

Table 5.5.2

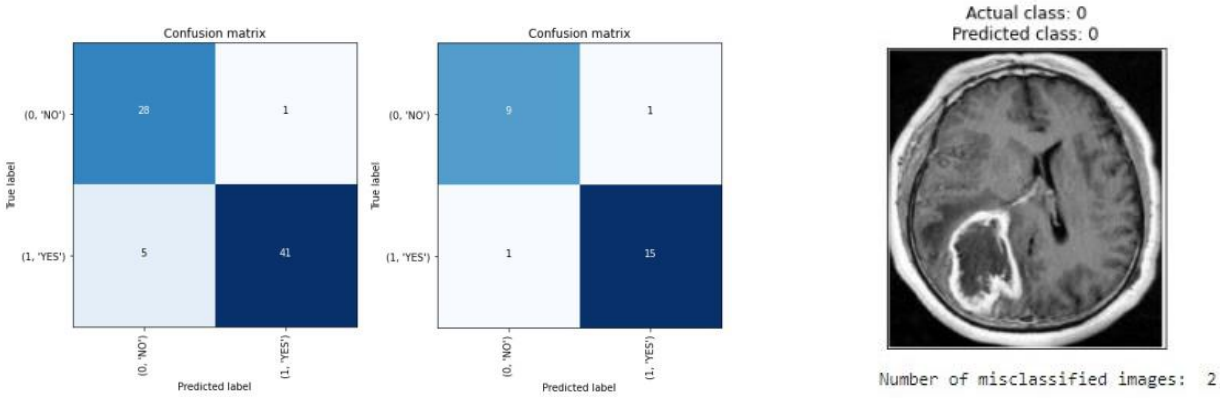
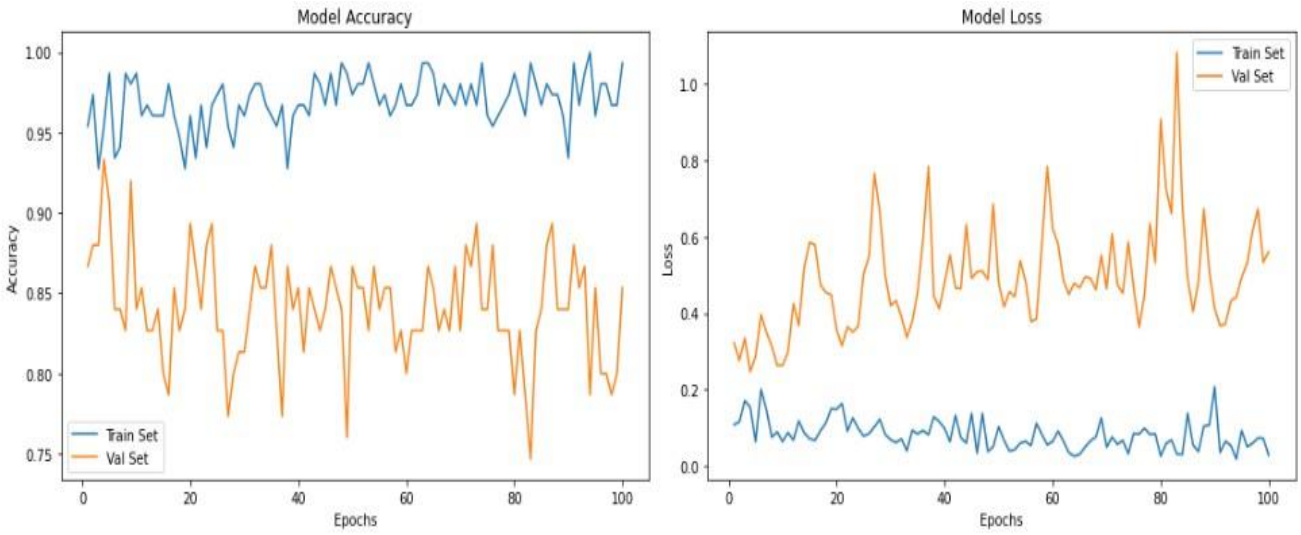


Figure 5.5 Efficient Net no. of misclassified images



5.6 Ensemble Learning (VGG-16 & ResNet-50 MODEL)

Validation Set Measures	
Accuracy	0.907
Precision	0.759
Sensitivity	1.000
Specificity	0.868
F1 score	0.863

Table 5.6.1

Test Set Measures	
Accuracy	0.962
Precision	0.900
Sensitivity	1.000
Specificity	0.941
F1 score	0.947

Table 5.6.2

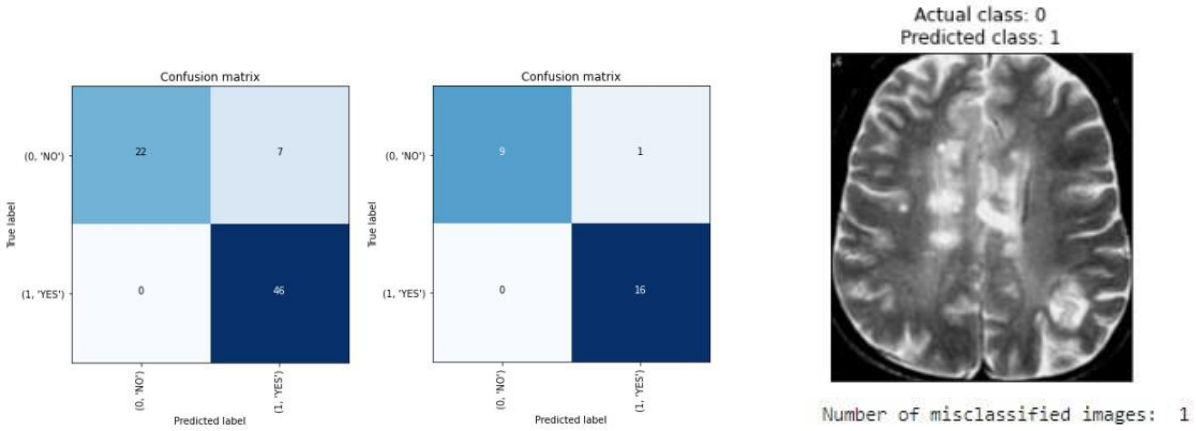
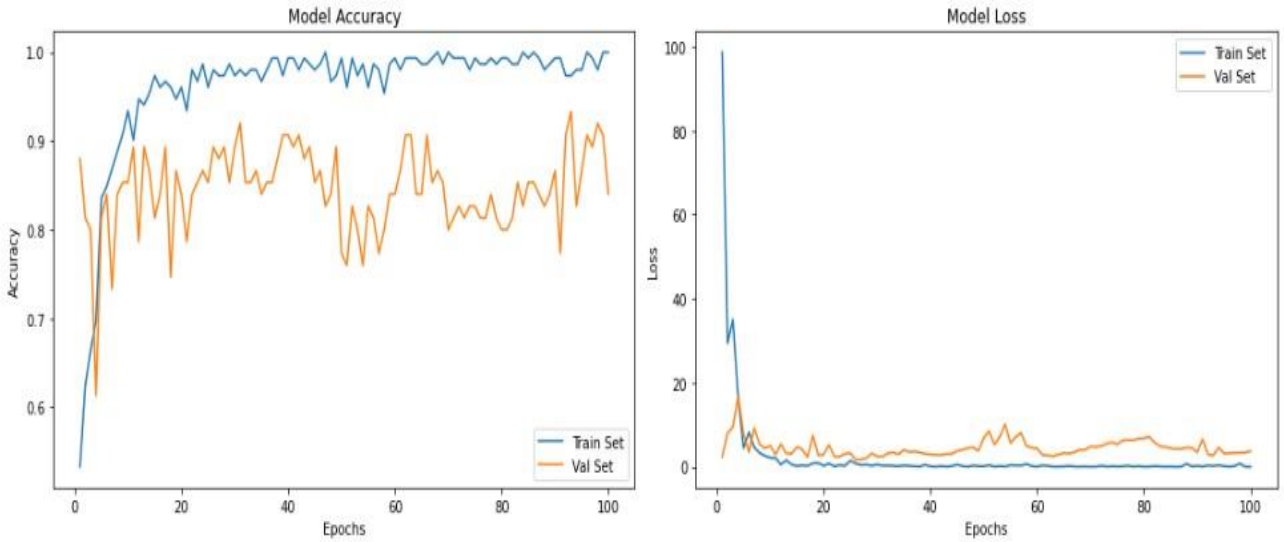


Figure 5.6 Ensemble (VGG-16 & ResNet-50) no. of misclassified images



5.7 Ensemble Learning (VGG-16 & Efficient Net MODEL)

Validation Set Measures	
Accuracy	0.947
Precision	1.000
Sensitivity	0.879
Specificity	1.000
F1 score	0.935

Table 5.7.1

Test Set Measures	
Accuracy	0.885
Precision	1.000
Sensitivity	0.769
Specificity	1.000
F1 score	0.870

Table 5.7.2

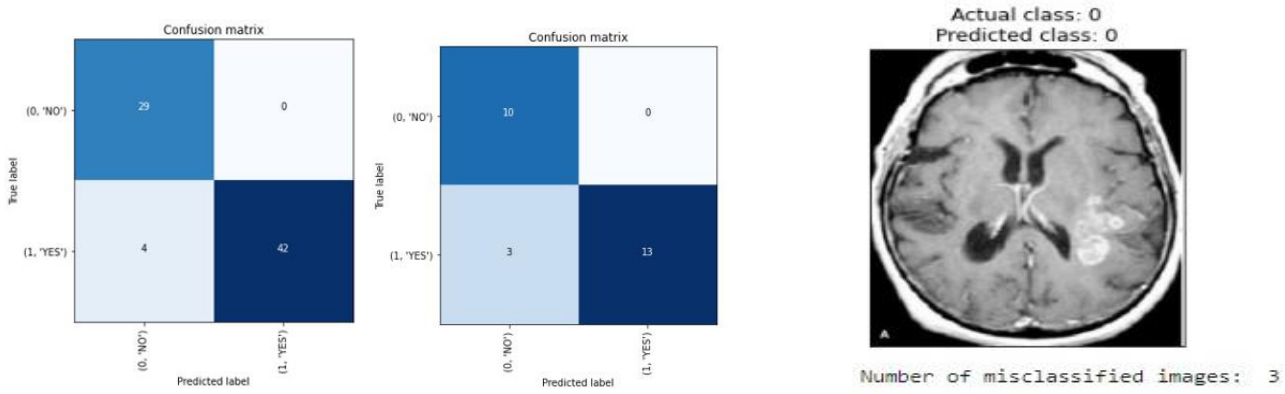
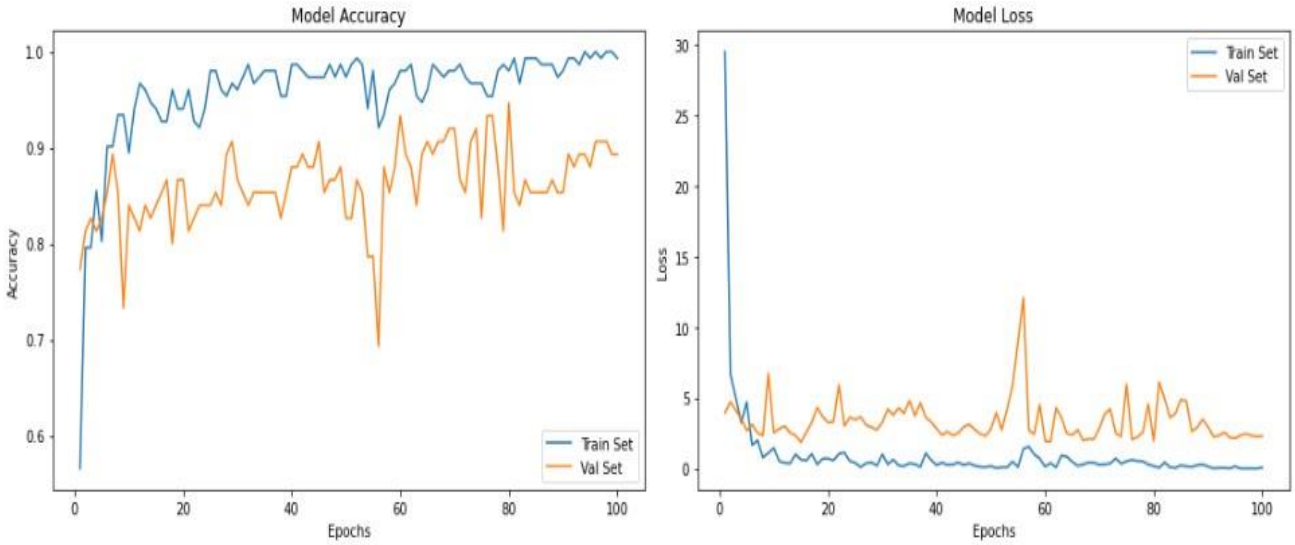


Figure 5.7 Ensemble (VGG-16 & Efficient Net) no. of misclassified images



5.8 GENERATIVE ADVERSARIAL NETWORK

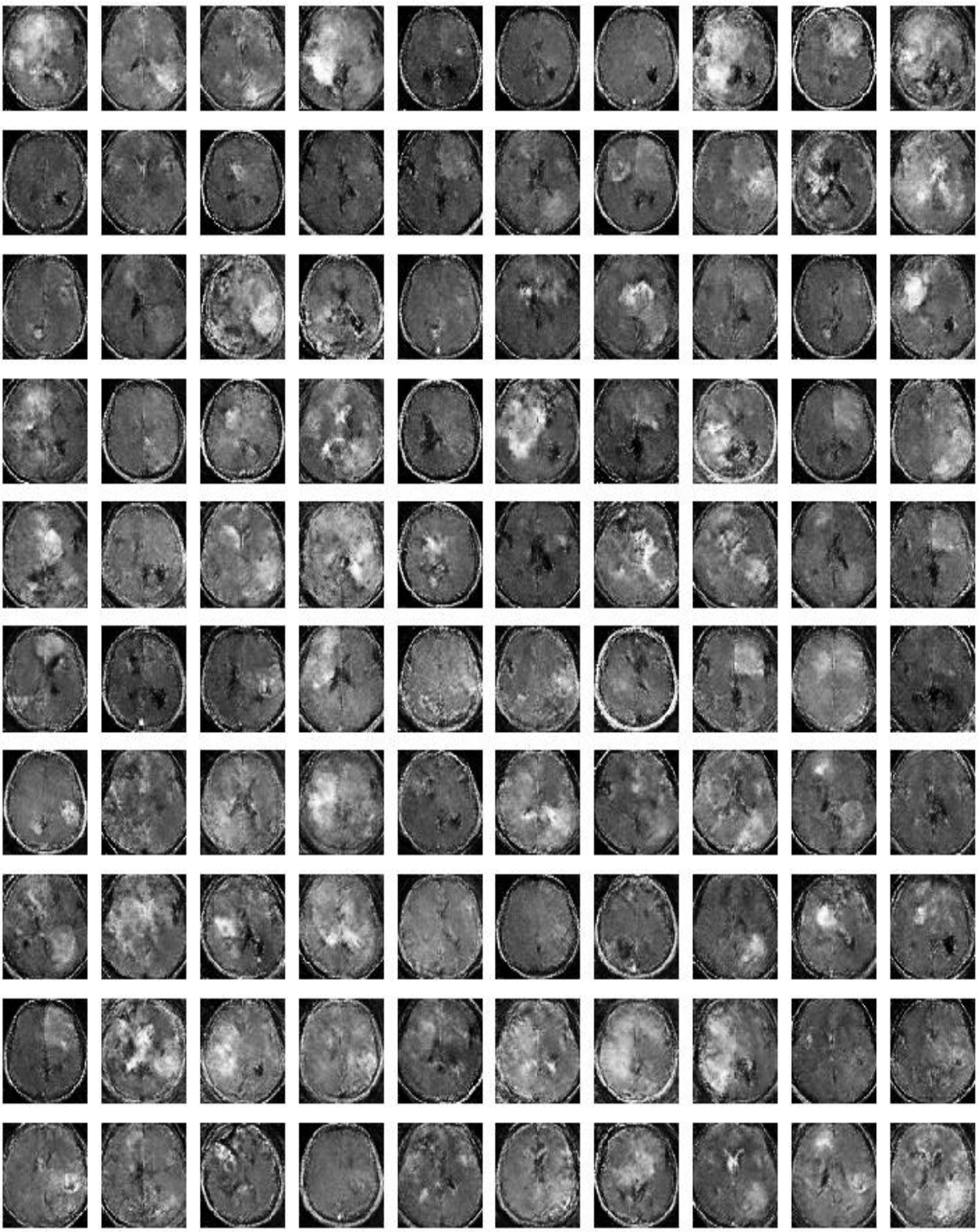


Figure 5.8 GAN Output

CHAPTER 6: CONCLUSION

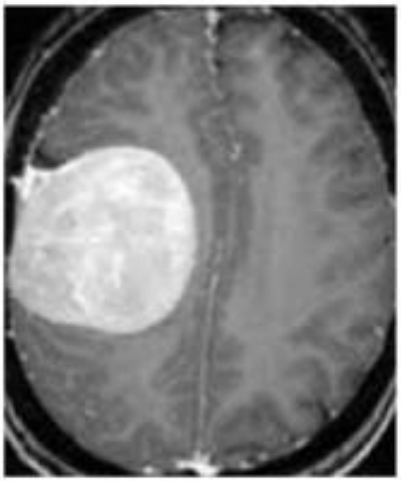
6.1 CONCLUSION

The **Ensemble Learning (VGG-16 & Efficient Net) Model** gave the best validation accuracy of **94.7%** among the 7 different CNN architectures tested.

The Efficient Net CNN Model individually resulted in an validation accuracy of 92%. The efficient net was relatively much faster to train compared to the ensemble of VGG-16 and efficient net which might give away the idea that its better to use it alone as it saves a lot of computation power and time for a mere drop of 1% in accuracy. But as we are dealing with medical imaging and in terms of medical diagnosis, mere decimal of percentages can play a huge role in saving a patient's life so I would propose the **Ensemble Stack Model (VGG16 & Efficient Net) to be chosen** among all models I tested in this report.

```
image = cv2.imread('TEST_CROP/YES/10.JPG',cv2.IMREAD_COLOR)
image2 = cv2.resize(image, (224, 224))
plt.imshow(image, cmap='gray')
plt.axis('off')
plt.show()
image3 = np.reshape(image2, (1, 224, 224, 3))
image3 = image3/255.

y_prob = new_model.predict(image3)
y_classes = np.where(y_prob >=0.5,1,0)
y_classes_labels = np.where(y_prob>=0.5,'MALIGN','BENIGNANT')
print(y_classes_labels)
```



```
[['MALIGN']]
```

Figure 6.1 Classification output

As for GAN, it **successfully generated indistinguishable MRI scans of the brain** originating from noise so it did a satisfactory job which could be visually compared to the actual dataset.

6.2 FUTURE SCOPE

- Collect a more robust and extensive dataset
- Tuning parameters and hyperparameters of CNN
- Ensemble different pre-built CNN for better accuracies
- Finding a pre-built CNN model specifically made for medical images/MRI scans
- Visualizing tumours in the images using object detection and image segmentation
- Experimenting more GAN architectures to demolish the mode collapse and vanishing gradient problem
- Implementing Wasserstein Generative Adversarial Network
- Creating a frontend using Flask or Django as an end-product for user

REFERENCES

Journal/Conference papers:

- S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
- Hussain, Mahbub & Bird, Jordan & Faria, Diego. (2018). A Study on CNN Transfer Learning for Image Classification.
- Radford, Alec and Metz, Luke and Chintala, Soumith, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR, 2015,
- Changee Han, Hideaki Hayashi, Leonardo Rundo, Ryosuke Araki, Wataru Shimoda, Shinichi Muramatsu, Yujiro Furukawa, Giancarlo Mauri, Hideki Nakayama, "
- C. Han et al., "GAN-based synthetic brain MR image generation," 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), 2018, pp. 734-738, doi: 10.1109/ISBI.2018.8363678.

Web:

- Dataset, Kaggle, kaggle/datasets/brain-mri-images-for-brain-tumor-detection
- Artificial Neural Network, Wikipedia, Artificial_neural_network
- Convolutional Neural Network, Towards Data Science, a-comprehensive-guide-to-convolutional-neural-networks
- Machine learning and Deep Learning, CodingBlocks, classroom_course_data_science
- Neural Networks, Tensorflow, tensorflow/api_docs
- Neural Networks, Keras, keras.io/api
- Transfer Learning, Tensorflow, tensorflow/transfer_learning
- Image Classification, Tensorflow, tensorflow/images_classification
- GAN, Google Developers, https://developers.google.com/machine-learning/gan
- GAN, Towards Data Science, towardsdatascience/generating-mri-images-of-brain-tumors-with-gan
- GAN, Coursera, coursera/generative-adversarial-networks