An Efficient Approach for Binary Classification in Brain Tumor Detection Using Convolutional Neural Network

by

MD. Arman Islam
19101639
Sheikh Araf Noshin
18101471
MD. Robiul Islam
18101272
MD. Farhan Razy
18101480
Samiha Antara
18101129

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

BRAC
UNIVERSITY

Inspiring Excellence
Department of Computer Science and Engineering
BRAC University
January 2022

Declaration

It is hereby declared that

1. The thesis submitted is our own original work while completing a degree at Brac University.

2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.

3. The thesis does not contain material that has been accepted, or submitted, for any other degree or diploma at a university or other institution.

4. We have acknowledged all the main sources of help.

Student's Full Name & Signature:

MD. Arman Islam
19101639

Sheikh Araf Noshin
18101471

MD. Robiul Islam
18101272

MD. Farhan Razy
18101480

Samiha Antara
18101129

Approval

The thesis/project titled **'An Efficient Approach for Binary Classification in Brain Tumor Detection Using Convolutional Neural Network'** submitted by -

1. MD. Arman Islam   (19101639)

2. Sheikh Araf Noshin   (18101471)

3. MD. Robiul Islam   (18101272)

4. MD. Farhan Razy   (18101480)

5. Samiha Antara   (18101129)

Of January, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science and Engineering on January, 2022.

Examining Committee:

Primary Supervisor (Member):

Co Supervisor (Member):

Zavid Parvez
Digitally signed by Zavid Parvez
DN: cn=Zavid Parvez, o=Brac University, ou=CSE,
email=zavid.parvez@bracu.ac.bd, c=US
Date: 2022.01.20 12:32:03 +11'00'

Dr. Mohammad Zavid Parvez, PhD
Assistant Professor
Dept. of Computer Science and Engineering
BRAC University

Arif Shakil
Lecturer
Dept. of Computer Science and Engineering
BRAC University

Program Coordinator (Member):

Dr. Md. Golam Rabiul Alam, PhD
Associate Professor
Dept. of Computer Science and Engineering
BRAC University

Head of Department (Chair):

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Dept. of Computer Science and Engineering
BRAC University

Abstract

Brain tumor detection using Convolutional Neural Network (CNN) models with binary classification has significantly improved the reliability of medical imaging through Deep Learning. The purpose of this research is to develop a modified CNN model by altering the different layers and weight values of each node to attain similar performance statistics to widely accepted CNN models while maintaining runtime efficiency. The proposed CNN model is trained on a benchmark dataset, Br35H, containing brain tumor MRIs, which incorporates binary cross-entropy to analyze the training data and accurately identifies whether or not a certain structured magnetic resonance imaging (sMRI) picture contains a tumor. In comparison to existing pre-trained CNN models, this study aims to contribute to the computer-aided diagnostic (CAD) system by implementing the proposed model with relatively less computational power. The model achieved an overall classification accuracy of 98.5% after extensive tweaking of the proprietary CNN architecture. The suggested system's performance is also compared with other existing systems, and the study demonstrates that it performs significantly better than some of the other models while having a fraction of their total parameters.

Keywords: Brain Tumor, CNN, Binary Classification, Deep Learning, Binary Cross-entropy, Dataset, Br35H, MRI, sMRI, CAD.

Acknowledgement

To begin with, we express our profound gratitude to Almighty Allah for providing us with the opportunity to work on this thesis and complete it successfully. We have tried our utmost best to achieve what we set out to do within the expected time frame. It's been a gratifying learning opportunity for us. We also express our heartfelt gratitude to our supervisor, Mohammad Zavid Parvez sir, for his guidance and assistance. Without his guidance, we would not have been able to finish our thesis work. He provided us with the required assistance and motivation at every stage of our research. Furthermore, we owe a debt of gratitude to our co-supervisor, Arif Shakil sir, who generously offered his knowledge, experience, and findings to us, which helped us immensely during our research work. Finally, we are also grateful to our family, friends, and loved ones who have helped us with our studies. Last but not least, we are grateful to BRAC University for providing us with a platform upon which we constructed the foundation of our research.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

ADAM Adaptive Moment Estimation

ANN Artificial Neural Network

AUC Under The Area ROC

BBB Blood Brain Barrier

BN   Batch Normalization

CAD Computer Aided Diagnostic

CE   Cross Entropy

CNN Convolutional Neural Network

ConvNet Convolutional Neural Network

DL   Deep Learning

GBD Global Burden of Disease

MRI Magnetic Resonance Imaging

MSE Mean Squared Error

OpenCV Open Source Computer Vision Library

ReLU Rectified Linear Unit

ROC Receiver Operating Characteristics

RSS Residual Sum of Squares

sMRI Structured Magnetic Resonance Imaging

VGG Visual Geometry Group

Chapter 1

Introduction

1.1 Brain Tumor

The brain is regarded as one of the most vital biological structures. The functionalities of the brain are significantly impaired if a tumor grows inside the brain. A brain tumor is a fatal medical disorder that results from the uncontrolled and aberrant growth of cells within the brain [62].

Brain tumor can be categorized as follows:

- **Non-cancerous brain tumor (Benign)**: These are low-grade (grade 1 or 2), indicating that they grow slowly and are less likely to reoccur after treatment.

- **Cancerous brain tumor (malignant)**: These are high-grade (grade 3 or 4) cancers that develop in the brain (secondary tumors); they are more likely to return after treatment.

Toxicologically, it is classified as either a primary brain tumor or a malignant brain tumor that develops in another body location and spreads to the brain [54]. A tumor in the brain is much more challenging to diagnose than a tumor in any other section of the body. As we know, there is a blood-brain barrier (BBB) in the brain, and conventional radioactive markers cannot detect tumor cell hyperactivity [53]. Annually, from 7 to 11 people per 100,000 in various age groups develop brain tumors (grade 2, 3). According to the Global Burden of Disease (GBD), this disease claims almost 227,000 lives each year. Additionally, nearly 7.7 million survivors are adjusting to life with disabilities [57]. Early identification of brain tumors saves lives and prevents disability. Furthermore, early detection decreases the need for brain surgery and other invasive methods of treating brain tumors. Manual brain tumor diagnosis is said to have a consensus rate of 90%–95% among medical specialists. According to an expert study, mixed tumors, such as medulloblastoma and glioblastoma, had an expert disagreement of 77% and 58%, respectively [40].

1.2 CNN in Medical Imaging

A brain tumor is often identified through an MRI scan with specialized equipment. This can be identified as some abnormal growth in the brain that can be visible

in the brain MRI image. The growth of computer vision and computer-aided image identification has led to medical science breakthroughs. Improvement in image categorization and identification has been made possible by using machine learning and neural networks in image recognition. The implementation of CNN in image recognition will ensure almost as much accuracy as any existing identification technology. So if a patient is unable to seek professional help to identify the tumor or the hospital needs automated assistance, that will help identify the tumor's existence in MRI images. The requirement for manually segmenting tumor regions is eliminated by utilizing CNN-based classification systems, resulting in a fully automatic classifier. This deep integration of CNN with medical image detection opens another dimension of advancement opportunities.

1.3 Computer Vision and Implementation of CNN

The ever-growing scope of technological advancement has produced an uncountable scope of computational process-based activities. Computer vision is one of the leading examples. This particular concept trains the machine to understand images and video. Significantly, the introduction of machine learning, "Deep learning," has revolutionized computer vision applications by accomplishing large-scale image and video recognition [27]. This surge of popularity can be attributed to the rise of public image repositories (ImageNet) and progressive performance in computational capabilities in graphics processing units (GPUs) [17]. The application of convolutional neural networks (CNN), a machine learning-based computer vision technology, is quickly becoming the most highly regarded among other traditional machine learning techniques in image recognition. The main reason for this is that CNN works faster and has higher detection accuracy than other approaches, such as Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and Artificial Neural Networks (ANN) [66]. To emphasize this more, CNN can detect altered or tampered images with considerable accuracy, which is the most crucial advantage in real-life scenarios. The recent emergence of different convolutional neural network models is so accurate that, in extreme conditions, CNN can statistically outperform even humans in terms of a particular breed or species identification [66]. The potential of computational power and data collection will only grow exponentially. As a result, the seemingly limitless possibilities for improving CNN and deep learning are expanded even further.

1.4 Research Motives

1.4.1 Research Objectives

Our research primarily focuses on implementing an originally constructed CNN model in the designated brain MRI data set to understand how those particular models process those values and predict the output. Then they compare the proposed model with some recognized pre-trained models and compare the accuracy, loss, and other data to demonstrate whether the proposed model is up to the standard or not. Additionally, the purpose of this research is to familiarize ourselves with the intricacies of each model and analyze those models to compare their results and

identify detection errors to exploit the weaknesses. Furthermore, this will help us mitigate those issues and arrive at an even more accurate and time-efficient version of our proposed model.

To achieve that, we aim to develop a sophisticated algorithm that incorporates image classification and deep learning to establish an open-source algorithm that may one day become the new revolution in convolutional neural networks with scope for future improvements. We intend to create a model with fewer parameters and a lower level of complexity that can be easily implemented in any system.

The objectives of this research are:

- To conduct extensive research on deep learning, convolutional neural networks (CNNs), and other well-established models for implementing deep understanding in image recognition.

- To comprehend the workings of a neural network, including the layers and mathematical functions that go into it.

- Research about different image-based datasets such as ImageNet, BR35h and the properties of those datasets, i.e., how many entries are there, labelling, missing values.

- Study different models (ResNet50, VGG16, MobileNet, InceptionV3, Xception, DenseNet121) and implement those models in our selected dataset through transfer learning.

- Pre-process the data to fit the model and divide it into test and training data.

- To construct classification reports for those datasets which the CNN models process.

- To create a confusion matrix that evaluates the performance of those classification models.

- Develop our own model and run it on the selected dataset, tweaking the parameters and weight distribution to make the model more accurate.

- To measure the accuracy of those models to identify the best model for relationships and patterns between dependent variables identification.

- Calculate the model loss from the loss function to understand if the model is predicting correctly as it should.

1.4.2 Problem Statement

Image identification in medical imaging has been a developing process for a long time. Computer vision and the introduction of neural networks have increased the accuracy of image identification by a significant margin. This research intends to demonstrate the image processing and identification of brain tumor images to identify whether they have a tumor or not by observing the abnormality. This research presented a convolutional neural network model with different convolutional layers

to determine whether a brain MRI image has a tumor. However, this problem must still be validated by other standard models to judge whether this proposed model is on par with others. In order to accomplish that, an in-depth analysis of the proposed model and other models were conducted so that the brain tumor identification through MRI imaging could be justified and assert the model's reliability.

1.4.3 Thesis Structure

In Chapter 1, an overview and introduction to a brain tumor and how it is detected using CNN are provided. Also, study goals and how to use MRI pictures to detect brain tumors are discussed. In chapter 2, research and related work on this topic are mentioned. Later, in Chapter 3, the paper reviewed the design of ANN and CNN and their functions. It also addressed the suggested model, dataset, and pre-processing technique in Chapter 4. Then the details of the model and how it functions are shown. Data set extraction and description, along with preprocessing the dataset to fit the models, are instructed here also. Then, in Chapter 5, the paper demonstrated the research methodology and the workflow of the suggested model. The architecture of CNN models is mentioned here. The discussion of the results and analysis is in Chapter 6. Finally, in chapter 7, all challenges and difficulties experienced throughout the thesis while compiling the models, along with plans for the future, are stated.

Chapter 2

Literature Review

The brain tumor is one of the most widely discussed diseases worldwide because of its uniqueness in nature and form from person to person. Brain tumor and its diagnosis using CNN are extensively discussed by Asma Naseer, Tahreem Yasir, Arifah Azhar, Tanzeela Shakeel, and Kashif Zafar [62]. It is known, the human brain is the most vital part of our body as it is essential for an infinite number of functions. Globally, brain tumor is considered a life-threatening disease, as, without early detection, they can even claim a person's life. To mitigate this issue, the authors have proposed a CNN-based computer-aided diagnosis model performed over the BR35H dataset, which is the intended dataset for our research. The BR35H dataset has divided MRIs into two groups of over a thousand images, where one is considered a positive sample, labeled as "Yes" and the other is a negative sample, labeled as "No". Moreover, they also performed their proposed model over other datasets like BTI, BTS, and others and gained a remarkable result. The primary focus of this paper is to indicate how manual brain tumor detection is less successful (90 to 95%) [40] than the proposed CNN model as the author's CNN models perform a more in-depth diagnosis with a higher success rate.

A brain tumor can be classified as cancerous or non-cancerous. The growth rate of a tumor formed in the brain might vary substantially. The nervous system's ability to operate appropriately is influenced by the rate at which a tumor grows and locates inside the brain. According to the article written by A. Rehman et al.(2021), [63] one can incorporate a convolutional neural network with detection and categorization of brain tumors. Three BraTS datasets from 2015, 2017, and 2018 were used for experiments and validation, achieving 98.32%, 96.97%, and 92.67% accuracy, respectively. Comparing the suggested design with existing methodologies demonstrates that it achieves equal accuracy. Here in the proposed method, a new CNN architecture is used to extract brain tumors, pre-trained VGG19 is used to reduce deep attributes, and Pearson correlation and FNN features are used to classify tumors at the end. The only shortcoming of this paper is that it could not show its results with other pre-trained architectures like ResNet50, Xception, and others.

In contrast, authors S. Deepak and P. Ameer [52] have managed to differentiate among glioma, meningioma, and pituitary tumors using a hybrid model. This hybrid model implements the pre-existing GoogleNet architecture and transfers learning to obtain MRI traits. The acquired traits are classified using established classifica-

tion methods. This experiment was conducted using Figshare's MRI dataset and a unique technique of 5-fold cross-validation at the patient level. It is estimated that the suggested approach has a recognition accuracy of 98%, which is better than any existing approach. The other performance metrics employed in the study are the AUC, precision, recall, F1-score, and specificity. The paper evaluates the system with less training data for practicality. Observations from the study suggest that transfer learning can be effective when medical images are scarce. An examination of classification errors is also provided. Some aspects can be improved as the transfer learned model performs poorly as an independent classifier. In addition, a significant number of samples from the meningioma class were incorrectly classified. Thirdly, they faced an overfitting issue as their dataset was small.

MRI pictures are used to detect brain tumors. The MRI scan generates so much data that manual classification of tumor vs non-tumor is almost impossible. However, it only provides reliable quantitative measurements for a restricted number of photos. An automatic and dependable classification technique is required for early brain tumor detection. This work has proven to be difficult because of the brain tumor's surrounding region's high geographical and structural heterogeneity. In this paper, J. Seetha and S. S. Raj [47] propose automatic brain tumor detection using CNN classification; the conventional categorization of brain tumors is carried out by utilizing Fuzzy C Means (FCM) [46] segmentation, texture, and shape feature extraction, and SVM and DNN-based classification [47]. The fundamental objective of this research is to collect data in order to construct an automated brain tumor diagnosis system with high efficiency, high performance, and simpler fabrication [47]. The researchers used an "ImageNet" dataset for the overall classification, using a pre-trained dataset. Lastly, completing the training for the last layer is required. Additionally, CNN provides the raw pixel value, as well as the depth, width, and height feature values [47]. To attain high accuracy, they used a gradient descent-based loss function; they also calculated training accuracy, validation accuracy, and validation loss; and lastly, they discovered that training accuracy is 97.5%, and the probability of a false positive during validation is extremely low [47].

Brain tumor classification is necessary for classifying tumors and deciding on treatment options. Brain tumors can be visualized in a variety of ways. However, the increased picture quality and lack of exposure to ionizing radiation make MRI a popular choice [58]. Deep learning (DL) has recently demonstrated exceptional performance, particularly in classification and segmentation challenges. Hossam H. Sultan et al.(2019) conducted a study using a vast number of patients and images [58]. In one experiment, they exhibited a CAD system for categorizing brain tumor MR images divided into three categories which are meningioma, glioma, and pituitary; also classified gliomas into distinct grades using a bespoke deep neural network structure. The suggested network has sixteen levels, the first layer being the input layer which holds pre-processed images while passing them to convolutional layers and implementing specific activation functions. Two dropout layers prevent the model from overfitting [58]. Their proposed design attained the highest levels of accuracy for the two datasets studied in this work, 96.13% and 98.7%, respectively. However, the dataset is relatively small (because of the range of imaging angles), data augmentation enabled the presentation of more accurate results.

Since the rise of machine learning, the Convolutional Neural Network (CNN) has become the most extensively used neural network model for classifying images. Back-propagation is used to identify features in CNN models utilizing a variety of layers. In their work, S. Das et al.(2019) concentrated on creating a CNN model for diagnosing brain cancers in T1-weighted contrast-enhanced MRI images [51]. The suggested system is comprised of two critical stages. First, pre-process the photos using various image processing techniques, and then use CNN to classify the pre-processed images. To achieve that, the experiment utilizes a collection of 3064 photos containing three distinct forms of brain tumors (glioma, meningioma, pituitary). The suggested system begins by pre-processing the image data. Preprocessing entails filtering photos with a Gaussian filter and equalizing the histograms of the filtered images. The machine then uses the CNN model to classify the photos. Dropout, regularization is used to prevent overfitting in the system [51]. During the training phase, it assists the model in focusing on the most apparent patterns and therefore improving performance. As a result, the model has a more significant possibility of generalization, which keeps it stable. Their proposed model got the model's accuracy of 94.39%, with an average precision of 93.33%.

In recent times, neural networks have gained massive recognition for classifying visual inputs arising from documents. Neural networks have different algorithms for literature and document analysis, which creates more significant confusion from time to time. Among many types of research that have been conducted on visual document analysis using convolutional neural networks, one considerable research was conducted by P.Y.Simard, et al.(2003) [13]. The data set used in this paper to show the result is an MNIST set of English digit images. This article proposes a straightforward "do-it-yourself" application of convolution with a dynamic layout for various visual document issues. When utilizing a simple convolutional neural network, sophisticated approaches such as momentum, weight decay, structure-dependent learning rates, intermediate layers, tangent props, or even tuning the layout are not required to produce the best results. Here, the peak results from the MNIST set are obtained by training a new set of elastic distortion and CNN. More importantly, this research paper proposes novel ways of building such networks that are significantly easier to construct than earlier methods and allow easy troubleshooting. The techniques used here are simple loops for convolution and modular debugging. This paper has tremendously noteworthy results of having only 0.4% error using the simple conv (CE) algorithm, the lowest error in an MNIST dataset. Research has shown that only two error functions, cross-entropy (CE) and mean squared error (MSE), were tested in the study. Aside from that, they did not use momentum, weight decay, structure-dependent learning rates, or additional padding around the inputs, which eventually turned ineffective. In conclusion, though CNN's research is tough to process, they claim to propose an easy visual document analysis using convolutional neural networks.

The pooling operation is an inseparable part of the convolutional architecture. In the paper [19], D. Scherer et al.(2010) talked about pooling methods and their results. This study compares aggression functions for several common item recognition tasks on a fixed architecture to acquire insight into unique functions. According to

empirical evidence, maximum pooling outperforms subsampling strategies greatly. Despite their shift-invariant features, occupying pooling windows are no better than non-overlapping pooling windows. Using this information, they were able to achieve error rates of 4.57% [19] on the NORB normalized-uniform dataset and 5.6% [19] on the NORB jittered-cluttered dataset. Zhang et al.(2020) used a Convolutional Neural Network (CNN) framework to perform their research in this study they described the tasks of the convolutional layer, pooling layer, and backpropagation. The lag here is that they have not discussed the flattening layer. They tested various pooling operations on the Caltech-101 [15] and NORB [16] datasets. By comparing the result between subsampling and max-pooling, they have distinguished their datasets accordingly. They were able to identify their datasets by comparing the results of subsampling and max-pooling. They showed that a max-pooling procedure outperformed a subsampling strategy for capturing invariances in image-like data. Using an otherwise comparable architecture, recognition performance outperformed subsampling techniques for several datasets. The total recognition rate is unaffected by pooling windows that are more smoothly overlapping.

In order to understand representational data in abstractions of different levels, deep learning can be used with several processing layers. Speech recognition, object detection, pharmaceutical research, and genetics have benefited from these enormous advancements. Deep learning makes use of the backpropagation technique to demonstrate how a machine's internal parameters should be adjusted in order to compute each layer's representation from the preceding layer's representation, therefore uncovering intricate structures in enormous data sets. The journal "Deep Learning", published by Yann LeCun, Yoshua Bengio, and Geoffrey Hinton [31] explained the essential aspects of deep learning and its necessity. It has been decades since typical machine-learning approaches could manage vast quantities of real-world data. Using non-linear but specific modules, deep-learning approaches move the representation from a lower, more concrete level to a higher, more abstract one. They are representation-learning procedures. Very complex aggression functions can be learned using deep learning methods. Moreover, the topics discussed in these papers are CNN, image understanding with a deep convolutional network, Recurrent Neural Networks (RNN), and their respective algorithms like CovNet and LSTM (Long Short Term Memory) [26], unsupervised learning [8] sparked renewed interest in deep learning, but the triumphs of strictly supervised learning have since eclipsed it. Nevertheless, it is not discussed in this paper. Finally, systems that merge representation learning and complex reasoning will substantially contribute to artificial intelligence advancements. To replace rule-based symbolic expression manipulation with operations on huge vectors, new paradigms must be utilized instead of deep learning and fundamental reasoning, which have long been used to identify speech and handwriting [22].

Furthermore, CNN is a popular approach for contextual categorization. It can understand contextual signals and, as a result, overcome the difficulties associated with a pixel-wise categorization. It significantly reduces the number of parameters that must be considered. For example, CNN is widely utilized in remote sensing [35], oceanfront identification [41], high-resolution data [24], audio scene [42], and MR brain image segmentation [36]. In this context, we discussed a study conducted

by Indolia et al.(2018) on the conceptual knowledge of CNN as well as its three most frequent architectures and learning methods [44]. To impart knowledge and comprehension of CNN's various elements, such as models, convolutional layers, pooling layers, fully connected layers, and activation functions, as well as to discuss the various CNN architectures, such as LeNet [11], AlexNet [20], GoogleNet [32] lastly two critical CNN learning algorithms which are Gradient Descent [11]. Adaptive Moment Estimation (ADAM) [25] Optimization has been detailed in this paper.

One of the most notable research projects has been conducted by A. Krizhevsky, I. Sutskever, and G. E. Hinton, [20] over image classification with Deep Convolutional Neural Networks. ImageNet, which is notoriously difficult to work with, had a test error rate of 15.3 % in their study, which shows that a massive convolutional neural network (AlexNet) can deliver record-breaking performance by utilizing solely supervised learning techniques and reaching that result. Five convolutional layers, three max-pooling layers, and three fully-connected layers were used, with a final 1000-way softmax layer at the network's finish. The network had almost 60 million parameters and 650,000 neurons. The nonlinearity functions were computed using ReLU [18]. In addition, we supplemented the data with methods including picture translation, horizontal reflection, and patch extractions. The model was trained on two GTX 580 GPUs for five to six days, utilizing dropout layers to battle the problem of overfitting to the training data and batch stochastic gradient descent with specified settings for momentum and weight decay to combat overfitting to the training data. This specially constructed network was utilized to classify data into 1000 different classes. When a single convolutional layer is removed from their network, performance suffers. The removal of any middle layers results in around 2% of the network's top-1 performance when the middle layers are removed.

E. Walach and L. Wolf [37] look into the problem of item counting in pictures from several perspectives. They use CNNs and integrate two key advancements over existing methods: a gradient boosting and selective sampling, which estimates a density map straight from the input picture. They do this by simultaneously increasing counting accuracy and decreasing processing time. This study shows that the suggested approach is successful even when labeling mistakes exist. Extensive tests on five distinct datasets illustrate our approach's effectiveness and resilience. The mean absolute error was lowered from 40% to 35% due to this process. Simultaneously, the training time for each CNN has been cut by 50 %. Additionally, the recommended technique is straightforward. They outperform trailblazing systems developed for each application while using the same basic architecture for three separate counting applications (microscopy, interior crowd, and exterior crowd). Finally, they want to expand the scope of the suggested approaches to include other CNN regression applications. Problems like these emerge in various domains, from age estimation in facial images to human posture estimates, which are fundamentally different from counting.

In this paper, N. Srivastava et al.(2014) conducted some of the most significant research in terms of preventing overfitting, As overfitting slows the network and makes it impossible to combine predictions from multiple huge neural nets at test time, which is a significant issue with extensive networks [29]. To avoid this sort of

difficulty, they employ the dropout method. The basic idea is to randomly remove units (and their connections to other units) from the neural network as you train it. Dropout prevents units from collaborating too extensively. During testing, a single unthinned network with lighter weights can replicate the impact of averaging the predictions from all of these thinned networks. Additionally, it considerably reduces overfitting and gives significant benefits over conventional regularization strategies. Using back propagation learning alone yields rigid co-adaptations that benefit training data but are useless for new data. In the presence of a hidden unit, these co-adaptations are rendered less efficient due to their unpredictable nature. When this approach is applied, neural nets outperform other classifiers in various application domains, including object classification, digit identification, speech recognition, and document categorization in computational biology. It also shows that dropping out is a generic method not tied to any industry or field. Dropout adds much noise to gradients compared to "Standard Stochastic Gradient Descent." Thus, a large number of gradients tend to cancel one another out. In terms of SVHN, ImageNet, CIFAR-100, and MNIST performance, dropout techniques are at the cutting edge of the field. Furthermore, dropout considerably improved the performance of conventional neural networks on extra data sets when used in conjunction with dropout. This study offers practical suggestions for employing dropout that is easy to practice. Not only does it provide reliable data in support of dropout, but it also explores an intriguing and enlightening idea from a completely different field.

In contrast, moving object identification algorithms perform poorly in complicated situations because of the dynamic background, light fluctuation, and shadows. Using ResNet-18 and an encoder-decoder structure, a method for segmenting moving objects in complicated scenarios has been suggested by X.Ou, P.Yan, Y. Zhang et al.(2019) [56] to address this issue. The encoder-decoder structure of ResNet-18 enables it to categorize pixels on a pixel-by-pixel basis, splitting them into foreground and background. It excels at feature extraction due to the shallowness of its layers, which retain a more significant number of small-scale properties. The proposed technique applies to scenarios with a dynamic backdrop, fluctuating lighting, and shadows, as evidenced by qualitative and quantitative findings on the open-source CDnet2014 and I2R datasets. In quantitative comparisons, the proposed approach significantly performs better than other algorithms, boosting the mean F-measure from 1.99 to 29.17%. The proposed technique is supervised training, and the recommended network structure is encoder-decoder. The network is fed the object frame and the artwork labels associated with it. The decoder then turns the encoder's feature vectors into segmentation maps. Additionally, using the Euclidean distance, the foreground mask is accurately located. Experiments on the I2R and CDnet2014 datasets show that our proposed solution is superior to established methods.

The paper "The CNN Paradigm" by L. O. Chua, and T. Roska (1993) demonstrated the inherent richness of the CNN paradigm with the help of some examples [5]. With time, the CNN paradigm grew to include various topics and contexts while maintaining the two fundamental principles of local connection and analog circuit dynamics. An informative overview of CNN's major class types is included in this book, in addition to a complete taxonomy. The cellular neural network (CNN) innovation, which is a dynamic processor array made of analog processing units that interact di-

rectly within a finite local area, demonstrates this capability [2]. In this paper, CNN is classified into four types, 1) In artificial systems [4] when performing detection tasks and several grid sizes [3] (e.g., coarse and fine grid) may be beneficial, the CNN architecture may be more cost-effective because of the variable grid size in terms of area complexity. 2) There are diverse kinds of processing activity in the linear region. Several nonlinear sigmoid and non-sigmoid functions are depicted in this example. It is best to use processor types [6] that are slowly and regularly altered when utilizing less precise components for highly accurate detection tasks. The small-signal operations in the linear section of the processor constitute a fundamentally new kind of processing. It shows a large number of nonlinear sigmoid and nonsigmoid functions. When using less accurate components for high-accuracy detection jobs, it is preferable to use processor types that are slightly and regularly altered. 3) As an alternative to the typical fixed-point operation, transient, oscillating, and chaotic modes are becoming increasingly relevant. 4) Interaction (connection) kinds: There are many different forms of interaction. The use of cloning templates to represent interactions is a critical concept in CNN. In today's world, templates are no longer limited to translation-invariant ones. Adaptive time-varying templates and associative memory both use templates that fluctuate regularly. The downside of this paper is that it does not contain a clear conclusion.

Chapter 3

Background Information

3.1 General Architecture of Neural Network

Neural Networks, often regarded as ANN, which is short for Artificial Neural Network, is a network of artificial neurons known as nodes developed in the early 1940s to solve Artificial Intelligence (AI) problems by predictions based on pre-processed data. The network takes input from its environment through neurons, passes the data in the form of weights to the subsequent layers, and ends up in the output layer by some functions. The layers can be broken down into three categories: input, hidden, and output. Here, neuron serves as a transfer function, meaning it takes the input values $x_1 + x_2 + x_3 + ... + x_m$ and weights $w_1 + w_2 + w_3 + ... + w_m$ and activation function, ϕ receives the output. Thus, the output y we get is,

$$y = \phi \left(\sum_{i=1}^{m} w_i x_i \right) \tag{3.1}$$

The output can be continuous, binary, categorical and many others depending on the types of input.
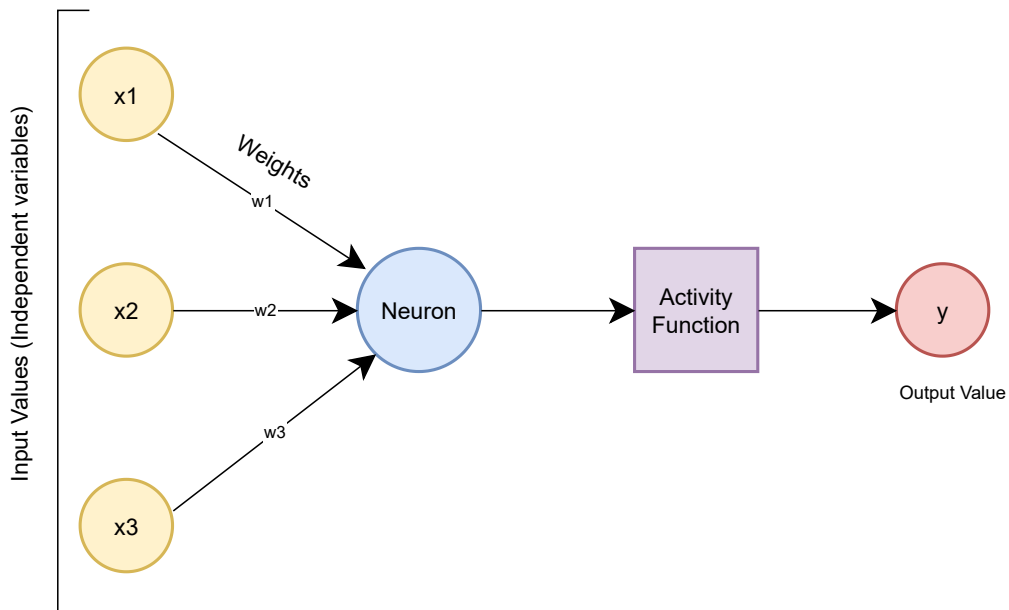

Figure 3.1: Basic Structure of ANN.

3.1.1 Neuron

The concept of the neural network is adapted from the neurons of our brain cells. Thus, neuron function gave birth to the neural network system. Similar to brain cells, the neurons act as each node for some layers, and they are interconnected with each other so they can transfer value and parameters for an optimum solution. The neurons turn on or fire when some information is passed. A specific condition must be reached in a neural network system to fire the neuron. This threshold value is known as bias. A set of neurons are situated and distributed in each layer. Usually, the first layer that takes input has a wide range of neurons. The last layer, or what is called the "black box," contains considerably fewer neurons. This connection throughout the neurons helps the deep learning process by altering and recognizing patterns to predict the expected output. Each neuron behaves like a node, and the connections between each node are considered edges. The edges here are the weighted values that contribute to different functions embedded in the system. It helps determine the accuracy and effectiveness of the machine.
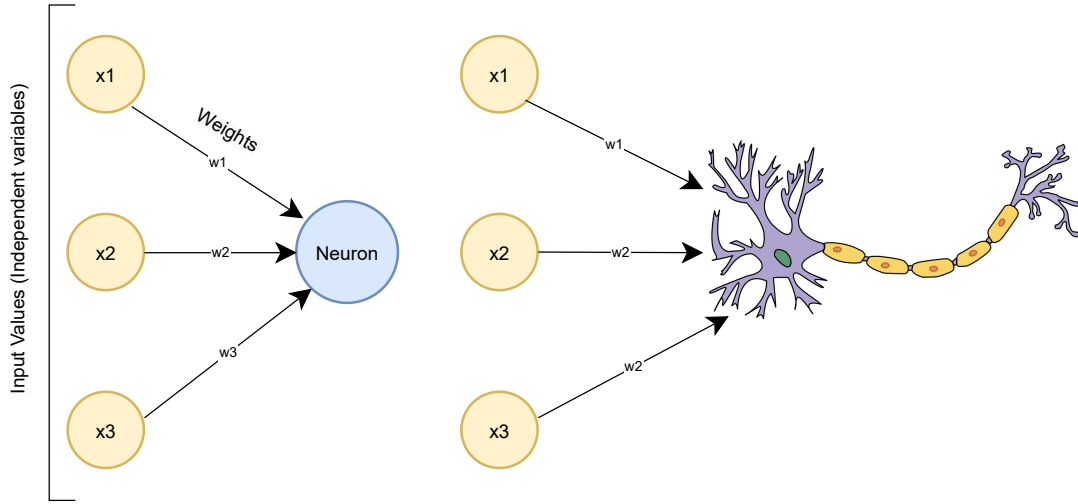

Figure 3.2: Artificial Neuron vs. Biological Neuron.

3.1.2 Activation Function

A neural network's activation function represents how the nodes in a layer's weighted sum input are converted into output. It is also referred to as a 'Transfer Function' [55]. Activation functions can be differentiated based on linearity and non-linearity. The activation function selection is vital as the performance and accuracy depend on it. The activation function is the same for all hidden layers, but the function may differ in the output layer depending on the total classes of output. For each node, the activation function defines the output of that node for the set of inputs given. This concept was introduced to direct the "black box" layers in a deterministic way. Due to constant research and mathematical modifications done to the system, many activation functions are discarded, and some produce higher accuracy [55]. The activation function is determined according to the output value that a specific structure needs for the next neuron to activate. It's done by giving a threshold value that will be regarded as the pivot point for a neuron to activate or not. Generally, the activation function is denoted by $\phi(x)$.
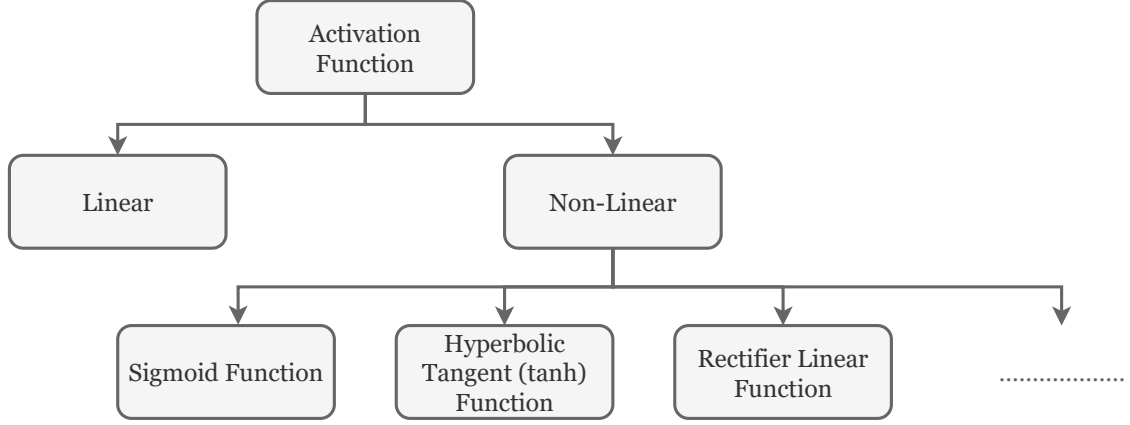
Figure 3.3: Types of Activation Functions.

For the proposed model in this paper, the binary classification is done by implementing 'Sigmoid' as the output activation function and 'ReLU' as the intermediate hidden layer's activation function. They are both part of the non-linear activation function category.

Non-Linear Activation Function

The integration of non-linear functions has significantly improved the accuracy of deep learning. Additionally, it allows the model to generate a complex mapping between the node's input and output [9]. This helps the model determine and learn to accurately decide the output for complex data inputs such as image, video, and audio files. The main advantage of using these functions is that, the differential helps determine the upgrading factor known as gradient based on input changes.

Sigmoid Activation Function

It is an S-shaped activation function. The characteristic of this function is that it is continuously differential and smooth. This function was widely used in neural networks at the early stage. This function congests the value of the input in such a way that output will always range between 0 and 1. This helps the neural network immensely as the range is minimal [21].

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

The derivation of this function ranges from 0 to 0.25. The benefit of using this function is that judging by the slope of the particular point, it can be decided if the value should move to its right or left region. This helps to improve the accuracy of the successive layers. This function has two problems named 'Vanishing Gradient' and 'Exploding Gradient' problem. It happens when the function's derivatives saturate around the x=0 mark and become very small. This is a computation-heavy function.
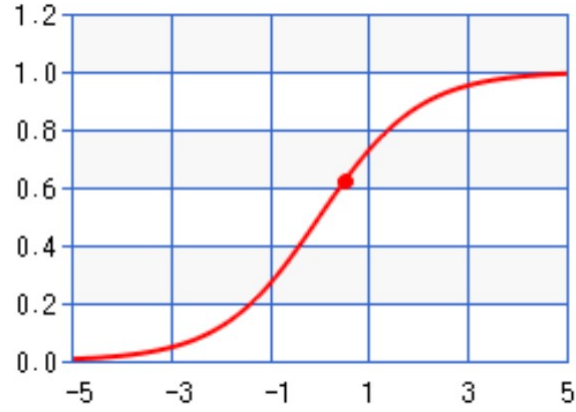
Figure 3.4: Graph of Sigmoid Function.

Function	Equation	Range	Derivative equation
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	0,1	$f'(x) = f(x)(1 - f(x))$

Table 3.1: Equation of Sigmoid Function.

ReLU Activation Function

Rectified linear units or ReLU is the most advanced and commonly used functions in neural network. ReLU is a non-linear function that acts as a linear function. This helps the computational process time [55]. Generally, ReLU can be classified as a piecewise linear function. This is different from the other two functions as those are continuously differential. One of the critical features for the ReLU function is that it mitigates one of the main issues, which is dealing with the negative weighted numbers [18]. The function that denotes ReLU is :

$$f(x) = \phi(x) = max(x, 0) \tag{3.3}$$

Figure 3.5: Graph of ReLU Function.

15

Function	Equation	Range	Derivative equation
ReLU	$f(x) = \begin{cases} 0; & x < 0 \\ x; & x \geq 0 \end{cases}$	$0, +\infty$	$f'(x) = \begin{cases} 0; & x < 0 \\ 1; & x \geq 0 \end{cases}$

Table 3.2: Equation of ReLU Function.

From the figure: 3.5 and equation above, it is evident that the output will be zero(0), if the input is any negative number ($x < 0$). For the positive inputs, the output will produce a linear output proportional to the input. Which is given as "x" in this equation. Effectively making the slope/gradient = 1. For half of the input domain, this function is linear; for the other half, it is non-linear. This results in a far more efficient computation. The negative values are neglected in the process and at the same time, it stays a non-linear function for the positive values. If the system has enough positive values, then this function is the most preferable. Furthermore, ReLU is the only function that can output a true zero (0) value. The hidden layer benefits from this function because of its simplistic approach.

3.1.3 Cost Function

In deep learning, there are hidden layers named as "black box," which process the data given in neurons and act accordingly to predict the output finally [14]. In each layer, different nodes process different data segments and give outputs. However, the whole neural network cluster, which contains numerous inputs, will have a singular output in terms of the cost function. This cost function is the value that determines whether the machine is working as expected or not. The cost function is generated by accumulating all the machine's steps and how the error is produced. In a sense, it is a feedback system for the computer. As what is happening on the hidden layer is unknown, the cost function is the most efficient way to know what variables to alter to make the system more accurate and precise. However, this cost function can only show the system's errors but cannot determine how to improve them. So, we need correctional functions or algorithms of such to improve the system's accuracy based on the cost function [14]. The equation of cost function,

$$C = \frac{1}{n} \sum_{i=0}^{n} (y^i - (mx^i + b))^2 \tag{3.4}$$

Costs are calculated by adding the expected value and subtracting the actual value of a node's forecast. The more accurate the neural model is, the lesser becomes the cost function. Here y denotes the expected value, and $(mx + b)$ means the value that a particular node produces. The summation of those values is calculated for each node to generate the function.

3.1.4 Gradient Descent

In the previous topic, one can only evaluate the cost function to measure whether the machine works poorly or accurately. Later, to improve the layers by tweaking the parameters such as the slope (m), which is also denoted by w in the neural network, and constant (c) also represented as bias (b) to make the machine more accurate for the next iteration. The gradient function [11] aims to make the cost function as low as possible. In each iteration, the gradient function is the differentiate term of the constant function. The function either moves to its right or left by examining the value, depending on the slope values sign.
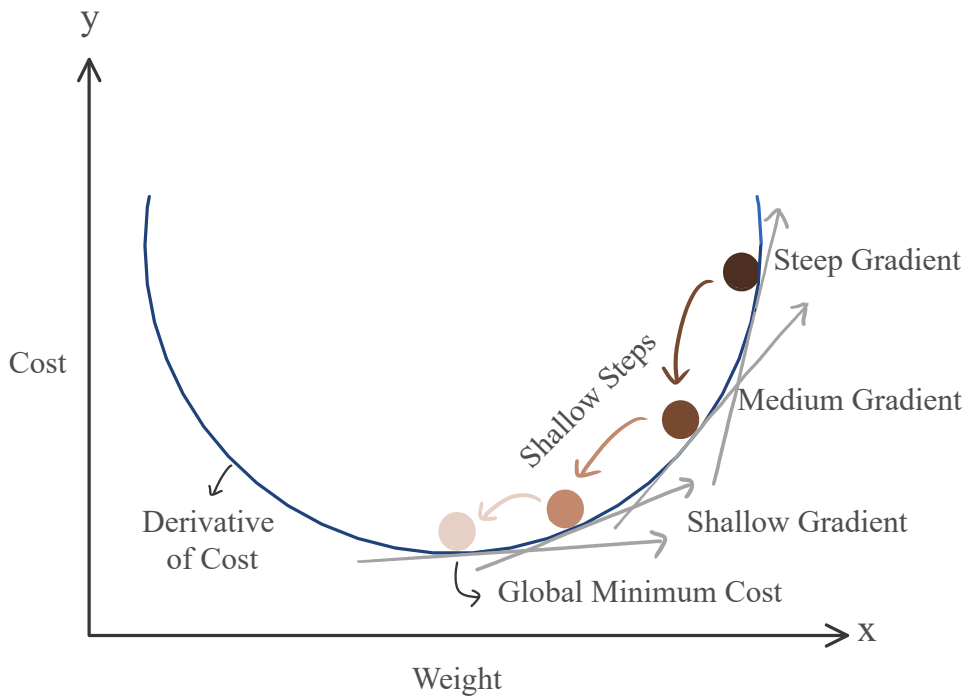

Figure 3.6: One Dimensional Gradient Descent.

3.1.5 Back Propagation

Through the process of back propagation, the network's connections are re-weighted to reduce the cost function's smallest possible value while also maximizing the network's actual output. That is, it will adjust the weight after an iteration [1]. It is a standard method of training the artificial neural network. After back-propagating multiple times, finding the set of weights will make sure that, the output vector is identical with the preferred output for each of the input vectors [1]. The goal of back propagation is to calculate the partial derivative $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of the cost function concerning any weight 'w' or bias 'b'. Therefore, after getting a predicted result, the error is back propagated, and the weights are updated according to how much they are responsible for the error. This tuning of weights of a neural network will improve the model's accuracy. The concept of 'Learning Rate' can show how much it should update the weights. Back propagation's actual performance is data-dependent. The back propagation algorithm implements the chain rule to determine the gradient of

a weight's loss function. It is of two types which are 'Static Back-propagation' and 'Recurrent Back-propagation.'
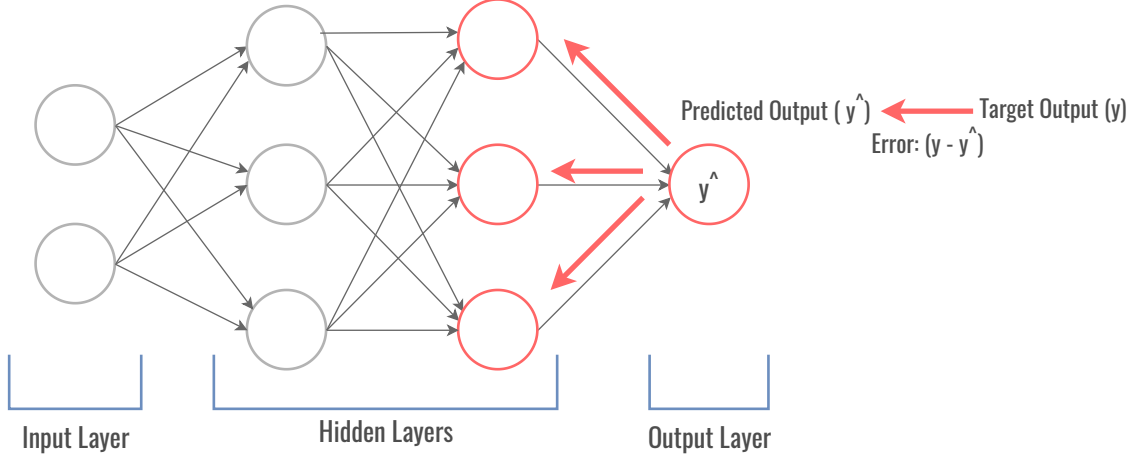

Figure 3.7: Back Propagation Inside Neural Network.

Here, it is figuring out the hidden units based on the actual output and the input value that were given. Each of the corners carries its own weight.

3.2 Convolutional Neural Network (CNN)

3.2.1 Convolution

ConvNet, abbreviated as CNN, is a deep neural network designed for grid-like topology identification and 2D matrix-like image recognition [49]. ConvNet is more than just a deep neural network with many layers disguised underneath the surface. It is a large network that resembles the brain's visual cortex in terms of how it analyzes and discerns images. This technique demonstrates the importance of profound layer improvements for information; basically, images processing [33]. The images are composed of a two-dimensional matrix of pixels on which the CNN algorithm is conducted [50]. The equation of the convolution operation is:

$$s[t] = (x * w)[t] = \sum_{a=-\infty}^{a=\infty} x[a]w[a+t] \tag{3.5}$$

Here, $s=$ feature map, $x=$ input image, and $w=$ feature detector or kernel. Convolution is a mathematical term that refers to a function obtained through the integration of two other functions. It explains how another function can alter a function's structure. For example, assume an image that is denoted by "x." The image is a two-dimensional array of pixels with distinct color channels. Here, the kernel is used "w," which is essentially our feature detector, to extract the output following the application of the feature map. A feature map is a technique that determines how similar two signals are, and this is a result of the convolution layer. A feature detector or filter is used to identify the edges of a picture. The whole convolution operation is responsible for calculating the image's edges.

How CNN Works

Let's consider a (4x4) pixel image that is illustrated in figure 3.5. The convolution filter operation on this image will be used to create a feature map.

Figure 3.8: 4x4 Pixel Image.

Here, we use two convolution filters.

Figure 3.9: Convolutional Filters or Feature Detector.

The actual ConvNet's filters are selected by the training process rather than by a manual decision. Let's start with the process. The convolution operation starts at the upper-left corner, here we use sub-matrix as the same size as the convolution filter and in every step it moves forward by one grid.

Figure 3.10: The First Convolution Operation.

The convolution operation is the sum of the products of the elements in the two matrices that are in the same location. Here the result 12 is on feature map is

calculated as: $(1x1)+(2x0)+(25x0)+(11*1)=12$.

Figure 3.11: Second Convolution Operation.

Here the result 18 on the feature map is calculated as: $(2*1)+(3*0)+(11*0)+(16*1)=18$.

Figure 3.12: The Completion of Feature Map.

This is how the feature map can be produced using convolution filer from a given image pixels. In the same manner, we can calculate the feature map for the another convolution filter.

Figure 3.13: The Feature Map of Second Filter.

The values present in the elements of the above feature map, like the first convolution operation, are determined by whether the image matrix cross-matches with the convolution filter.

In summary, the convolution layer applies convolution filters to input image to generate the feature maps. The trained convolution filters determine which features are extracted in the convolution layer. As a result, the features extracted by the

convolution layer differ based on the specific convolution filter used in the model. The feature map that creates is processed through activation function prior to the layer yields the output. The convolution layer's activation function is identical to that of a regular neural network.

Significance of ReLU in Convolution

ReLU function behaves as a non-linear function. All the negative numbers in that function will be converted into zero(0). And the rest of the domain ($x >= 0$), the value will remain the same as input x. Thus, creating a mixture of linear and non linear relation due to it's nature being piece wise function. If we take a look at the graphs representing the function sigmoid [21], and ReLU in figure: 3.4, 3.5 in other functions the derivative value at some point comes close to zero which is also called gradient value. If the gradient value becomes zero the weight distribution will not change. This mathematical notation is the key to improve the accuracy to the system by telling which values to alter in the future layers. If we look at the ReLU function, if the weighted value ever reaches negative, the value is immediately converted to zero which discards those negative values. This can cause the system to be less accurate. However, It's observed that all the positive values are creating a positive slope linear line which will produce the gradient values as non-zeroes. On the top of that discarding those negative values means the system will perform even faster than before. Moreover, computation power will be kept at minimum as it won't have to deal with exponential functions rather ReLU just uses a max function which is considerably simpler. This function can output a true zero (0). It has the unique feature where it works as linear function. This makes the calculations in the neural network cluster significantly easier. The linear nature of the function effectively mitigates the "Vanishing Gradient" problem because the gradient remains the same for all positive numbers. For multiple hidden layer architecture, this is easily the most preferable function that will produce an accurate output within shorter time.

1	2	3	4
25	11	16	0
0	1	0	1
0	2	4	1

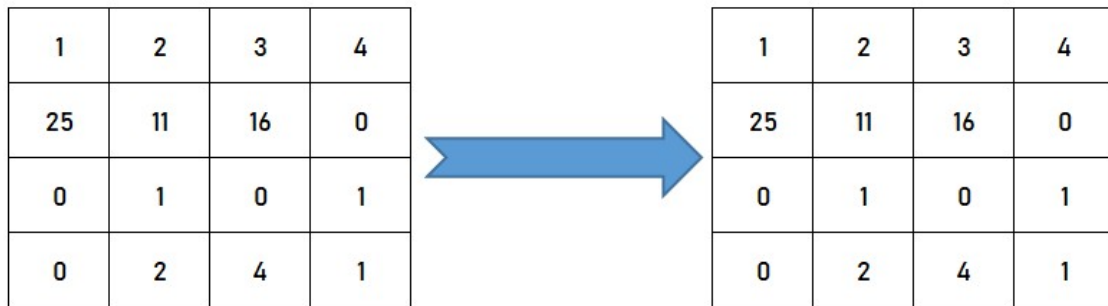| 1	2	3	4
25	11	16	0
0	1	0	1
0	2	4	1

Figure 3.14: ReLU example-01 using only +(ve) pixel Values.

Here, in two different cases operations of ReLU in a 4x4 pixel grid matrix structure is demonstrated. In the first matrix, all the values of each entry is either zero or greater than zero which means all the values are positive numbers so as the node gets the input such as that, the output stays the same as seen in the output matrix given in figure: 3.14.

Figure 3.15: ReLU example-02 using +(ve) and -(ve) pixel values.

But when some negative numbers are given as input in the matrix, the function converts them to zero. In this case, 4 entries were found (highlighted with blue marks) to be negative and in the output the values are converted to zero and the previous values are discarded. That's the illustration of how a basic ReLU operation works.

(a) Before Discarding

(b) After Discarding

Figure 3.16: Before and after discarding negative values [65].

In these two pictures, the ReLU function's work is demonstrated. In the first picture, while the object detection neural network cluster is running, the black pixels are unfavourable, and whites are of positive value. After the function is being applied, the value of the black pixels is converted to zeroes, thus leaving only the white pixels to work with. This process speeds up the computational time. Because the system now has a smaller matrix with fewer values to calculate. This helps the neural network to train efficiently and detect the object more accurately.

3.2.2 Pooling

CNNs use two different types of layers: convolutional (which resembles primary cells) and pooling (which resembles complicated cell behavior). Each convolutional layer applies a non-linear transfer function to the source picture and executes a discrete 2D convolution operation with an altered kernel. By aggregating neurons from a local spatial vicinity, the pooling layers lower the size of the input.

Due to its ability to reduce the dimensionality of feature maps, pooling is an essential step in convolutional systems. A group of values is combined into a smaller number of values, decreasing the dimensionality of the feature. By retaining relevant information, it turns the joint feature representation into useful information. By removing certain connections between convolutional layers, pooling operators provide a spatial transformation in variance while lowering the computational cost for upper layers. The pooling layer is mainly used for two beneficial purposes. The first is to minimize the number of parameters or weights, lowering computing costs, and the second is to prevent over-fitting. Only useful information should be extracted from a pooling method, and irrelevant details should be discarded. Pooling is mainly divided into 1) Max Pooling and 2) Average Pooling. There also exists another form of Pooling which is known as subsampling.
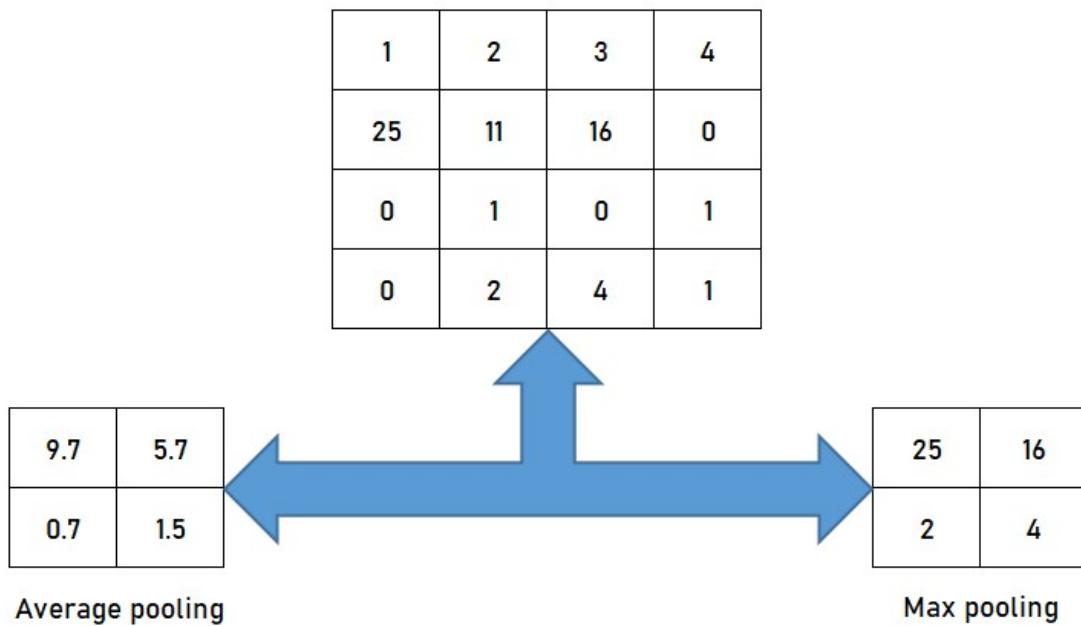

Figure 3.17: Finding Avg and Max pooling from a given image pixel.

Average Pooling: Down-sampling is performed via an intermediate pooling layer, with which rectangular pooling regions are divided and the average values of each cell are computed [59].

$$a_j = tanh(\beta \sum_{N*N} a_i^{n*n} + b) \tag{3.6}$$

Max Pooling: The maximum value inside a group of selected cells is passed forward by the max-pooling operator.

$$a_j = \max_{N*N}(a_i^{n*n} u(n,n)) \tag{3.7}$$

3.2.3 Flattening

It is a very crucial layer for feeding the data. Some artificial neural networks have dense layers as the last layer, which expects data in a one-dimensional system. In a CNN model, the final stage is a classifier, which is a dense layer. The output of the pooling layer must be converted into a one-dimensional feature vector before it can be used by the ANN. This process is called flattening. This feature is a must to flatten the output side of the convolutional or pooling layer and create a 1 dimensional feature vector that the dense layer can utilize to conduct the final classification. The long vector that we obtain after flattening will be the input layer for an artificial neural network [23].
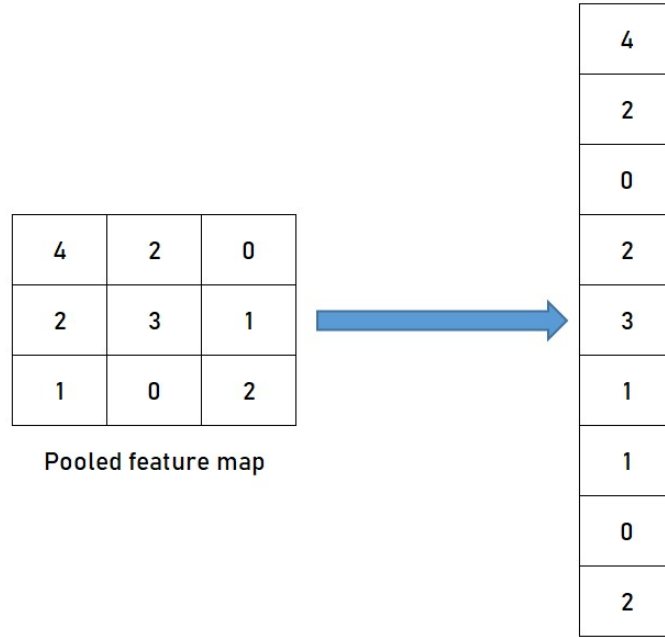

Figure 3.18: Flattening.

In this figure, the pooled feature map is a 3x3 matrix which is converted into a one dimensional matrix.

3.2.4 Full Connection

Due to the fact that our matrix had been converted to a vector and fed into a fully-connected layers in a neural network, we chose to call this the FC (Fully Connected) layer. Here, the qualities extracted from the prior layers are transformed into the final layer. Equation 3.8 establishes the relationship between consecutive levels [43].

$$v_i^j = \delta(\sum_k v_k^{j-1} w_{k,i}^{j-1}) \tag{3.8}$$

Here in equation 3.8, V_i^j denotes the value of neuron i at layer j, δ is the activation function and weight of connection between neuron k from layer $j-1$ and neuron i
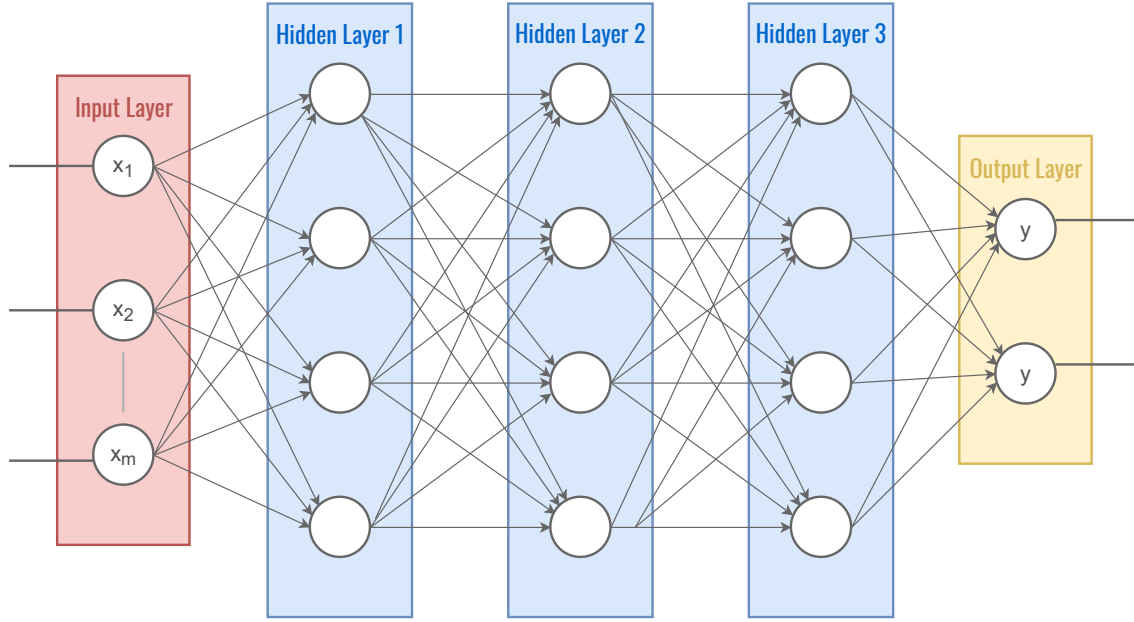
from layer j are shown by $w_{k,i}^{j-1}$ [43].

Figure 3.19: Fully Connected Layers.

The feature map matrix will be transformed to a vector $(x_1, x_2, ...x_m)$ which is shown in the figure: 3.19. These characteristics are integrated into a model using the fully connected layers. The output from one neuron is then fitted into an activation function and passed to another neuron [21].

3.2.5 Output Layer

The output layer of a CNN is the final layer. This layer estimates the classes based on the trained images. The output layer contains a neuron for each conceivable class, i.e., one for unaltered images and another for each possible alteration. This layer implements the activation function "Sigmoid," which gives an output as binary, which is crucial for binary classification.
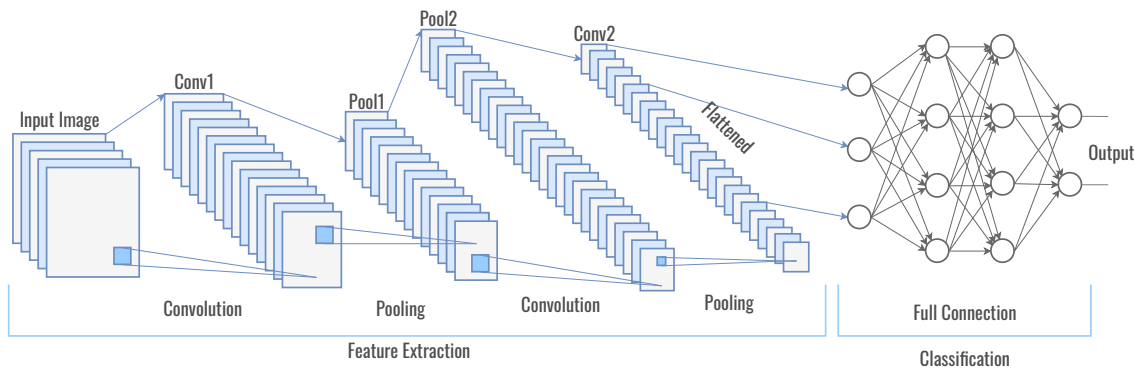

Figure 3.20: Full Conv Architecture.

3.3   Transfer Learning

The pre-trained network's architecture and weights are retained once trained on a large dataset, such as an image classification dataset. In this case, the ImageNet is used to pre-train the model. There are two approaches to make use of the transfer learning from a previously trained network:

- **Feature Extraction**: It is performed by pulling the features of the dataset using the convolutional base which was previously trained in different network and then training a new classifier with the outputs.

- **Fine-Tuning**: The fine-tuning procedure is complementary to the feature extraction method, as it entails unfreezing the final layers of the frozen convolutional base used for feature extraction.

3.4   Batch Normalization

Occasionally, data span an extensive range and do not belong on the same scale; also, each of our samples' characteristics can vary significantly. The more significant inputs in non-normalized datasets can cascade across the layers of neural networks, causing an unbalanced gradient and an inflating gradient problem. This can affect our training speed. So, the solution to these issues is the normalization of the dataset, where we put the data on the same scale. During training, one of the weights can become drastically more extensive than the other weights. This problem will keep on cascading onto the other layer, which will cause instability. This is resolved using batch normalization. Here in batch normalization, we can choose the layer we want to normalize. Batch normalization makes the activation function's output more uniform, and the equation is given below:

$$z = \frac{x - m}{s} \tag{3.9}$$

Then it multiplies this normalized output by an arbitrary parameter, g, the equation is given below:

$$z * g \tag{3.10}$$

There is an additional parameter, b, added to the final product after the multiplication.
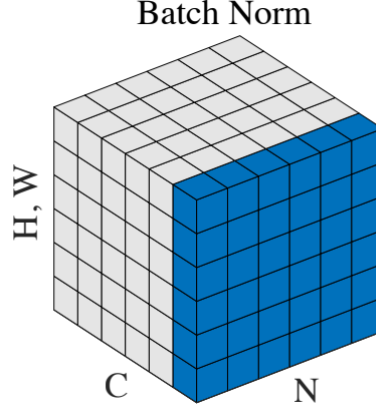
$$(z * g) + b \tag{3.11}$$

Figure 3.21: Batch Norm [48].

As shown, N is the batch axis, C is the channel axis, and W is the spatial height. All of these variables will be optimized throughout training. It has been known that normalizing neural network input data to zero means and the constant standard deviation is helpful in neural network training [10]. A batch normalization layer's input and output are both four-dimensional tensors known as $I_{b,c,x,y}$ and $O_{b,c,x,y}$ respectively. The channels in the input images represent the RGB channels. Batch normalization applies the same normalizing to all activation in a given channel [30].

$$O_{b,c,x,y} \leftarrow \gamma_c \frac{I_{b,c,x,y} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c \quad \forall b, c, x, y \qquad (3.12)$$

Batch normalization subtracts all input activation in channel c from the mean activation $c = 1$, $\mu_c = \frac{1}{|\mathcal{B}|} \sum_{b,x,y} I_{b,c,x,y}$ where B comprises all activation in channel c across all features b in the whole mini-batch and all spatial x; y locations. The centered activation is then divided by the standard deviation c (plus for numerical stability).

3.5 Regularization

Regularization is used to prevent the model from over-fitting by supplementing it with additional data also reduces the coefficients towards zero [7]. Sometimes, machine does not perform as good in the test data compared to the trained data. If the model cannot anticipate the result while dealing with unknown data, it is over-fitted because of the excess noise it produces in the output [12]. A regularization will help to resolve this problem. Regularization works by supplementing the complicated model with a penalty or complexity term. Consider the following equation for linear regression:

$$y = \beta 0 + \beta 1 x 1 + \beta 2 x 2 + \beta 3 x 3 + ... + \beta n x n + b \qquad (3.13)$$

y denotes the value to be predicted in the preceding equation. y has the attributes $x1, x2, ..., xn$. The values $\beta 1, ..., \beta n$ represent the weights or magnitudes assigned to

the characteristics. The bias of the model is denoted by $\beta 0$, while the intercept is denoted by b. Linear regression models attempt to minimize the cost function by optimizing $\beta 0$ and b. To construct a model that properly predicts the value of y, loss function and optimal parameters are needed, such as bias and weights. The term "Residual Sum of Squares" refers to the loss function that is frequently employed in linear regression (RSS).

$$RSS = \sum_{i=1}^{M}(y_i - y_i')^2 = \sum_{i=1}^{M}(y_i - \sum_{j=0}^{n}\beta_j * Xij)^2 \qquad (3.14)$$

There are two primary approaches for regularization: Ridge Regression and Lasso Regression. They are both different in penalizing coefficients. L2 regularization is used in the ridge regularization approach. It changes the RSS by adding a penalty (diminishing quantity) equal to the square of the coefficient magnitude. However, lasso regularization accomplishes L1 regularization, which changes the RSS by adding a penalty (amount of shrinkage) equal to the sum of the coefficients absolute value [12].

3.5.1 Dropout

In machine learning, the term "Dropout" refers to the process of randomly omitting some nodes from a layer during training [29]. The dropout's functionality is that it randomly turns of specified percentage of neurons inside the hidden layers so that model over-fitting is prevented.
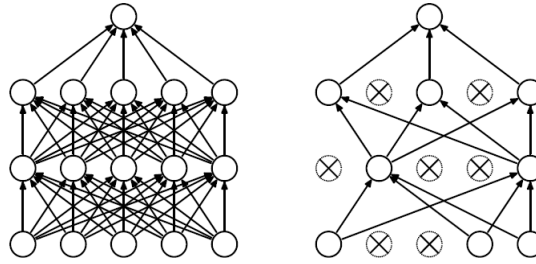

Figure 3.22: Dropout Neural Net Model [29].

On the left hand side of the figure: 3.22 is a representation of a convolutional neural network with two hidden layers, while the right shows a reduced network formed by nullifying some neurons of the left network. Thus, the crossed units have been turned off or nullified.

During training, fewer weights and biases are considered for the crossed teams led on the right side. Dropping a unit from the network involves disconnecting it from all incoming and outgoing connections. It is chosen at random. In the most basic instance, each unit is kept with a fixed probability 'P' independent of other units, where 'P' may be selected using a validation set or set at 0:5, which is optimum for most of the networks. In contrast, the ideal probability for the input units is usually closer to 1 than to 0.5 [29].

Chapter 4

Dataset Extraction

4.1   Dataset Description: Br35H

The research is targeted towards the solution of an image classification problem. The data set chosen for this research is an MRI dataset known as Br35H, containing images of the human brain [61].
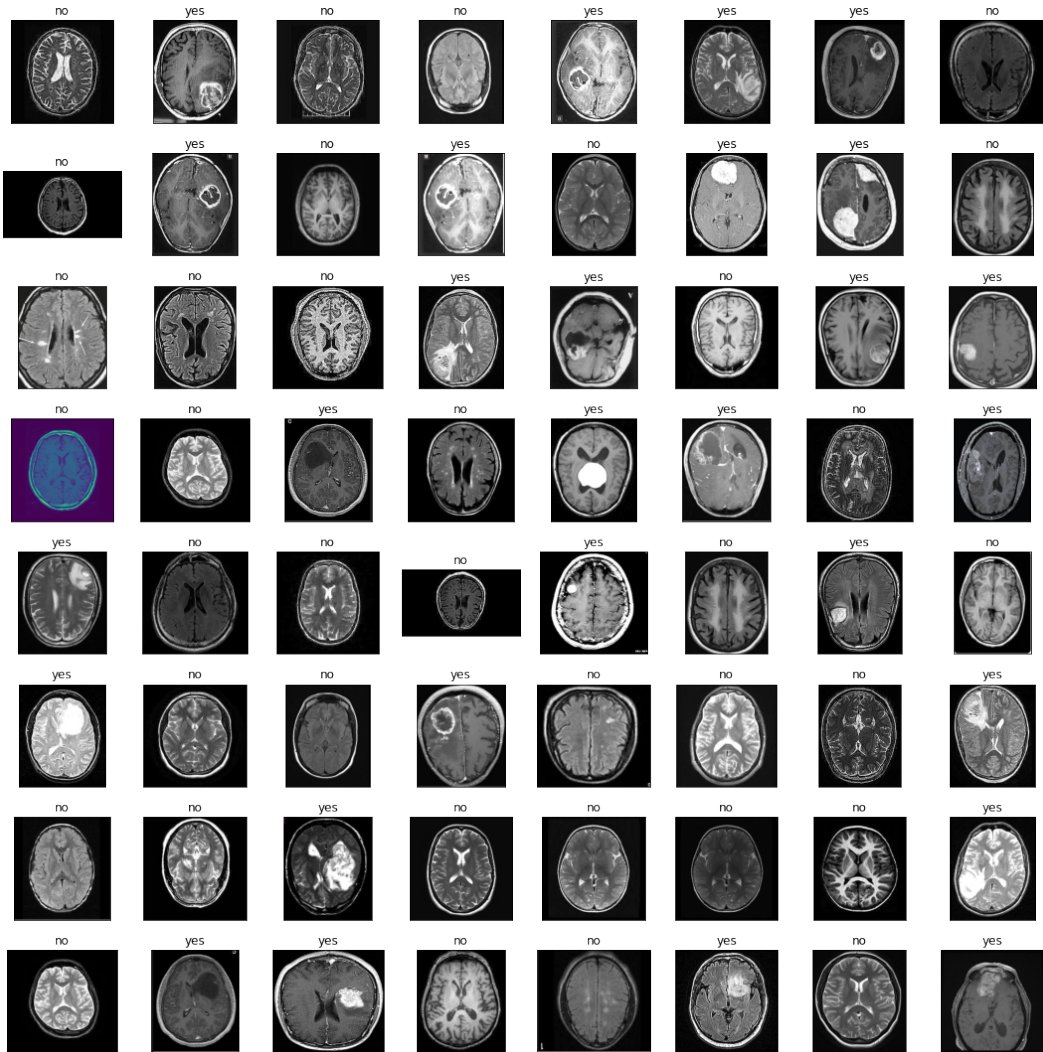

Figure 4.1: MRI Images of Human Brain from Br35H Dataset [61].

It is an abbreviation of "Br35H: Brain Tumor Detection 2020 dataset" [61]. This data set consists of 3000 jpg files of brain tumor MRIs. These images are divided into two categories of 1500 MRIs each. They are categorized as "Yes" and "No," where "Yes" means there is the existence of a brain tumor and "No" means there is not. These are denoted with 0's (no) and 1's (yes). It has a 7.5 usability rating. The data usability score is calculated using the following criteria: license, tagging, data overview and description, simplicity of use, maintainability assurance, machine-readable file formats, metadata, and the presence of a public kernel. According to this rating, licensing, tagging, data overview, and description are considered [62].

The proposed CNN models are divided into two halves. Here, Br35H data was chosen because it has a large amount of brain MRI. It helps to train the model and accurately detect the presence of a brain tumor. This data set is publicly available on Kaggle [61]. The data set was extracted from that online site and then mounted on the programming platform to alter the data and train the model.

4.2 Data Preprocessing

4.2.1 Importing Libraries

Firstly, we imported all the libraries and modules needed to run our model. These libraries include CV2, OS, NumPy, TensorFlow, Keras, Sklearn, Matplotlib, and Pyplot. For data preprocessing, the libraries CV2, OS, and NumPy are used. The OS library is used to check the dataset directory, the CV2 library is used to read images from different folders in the dataset and convert these images to an array, and the NumPy library is used to convert the array into an np array, which is a matrix conversion. The Br35H dataset comprises brain tumor images that are grayscaled by using cmap functions built into Python. Lastly, we uploaded these images separately into two classes, where one class contains images of a brain tumor and the other one does not.

4.2.2 Image Processing

Each dataset has its own complexity and layout, which we must preprocess to enhance accuracy while minimizing loss. In general, image processing can be divided into two categories: analog and digital image processing. Analog image processing is used on hard copies such as scanned or printed images, whereas digital image processing is used to alter digital images such as their properties, attributes, bounding boxes, or masks using a computer. The Br35H dataset comprises 3,000 digital MRI pictures, which we should denoise and resize to feed the model appropriately.

Image Denoising

MRI images are frequently subjected to "Gaussian noise," "Salt and Pepper noise," and "Speckle noise." As a result, obtaining an accurate brain image is challenging. A precise brain picture is critical to obtain for the future diagnosis process. There are numerous filtering methods for image denoising, such as the Gaussian Filtering

Method, the Median Filtering Method, and many others.

$$G_{(x,y)} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.1}$$

After gathering the data, "Gaussian filter" is applied to the images in dataset Br35H to reduce the random noise, as MRI images only contain random noise. Firstly, gaussian filters are added to make the images smoother and denoise the grainy and noisy images. In the code, the skimage library was used to denoise the image where a $sigma = 0.5$ value was assigned to it.
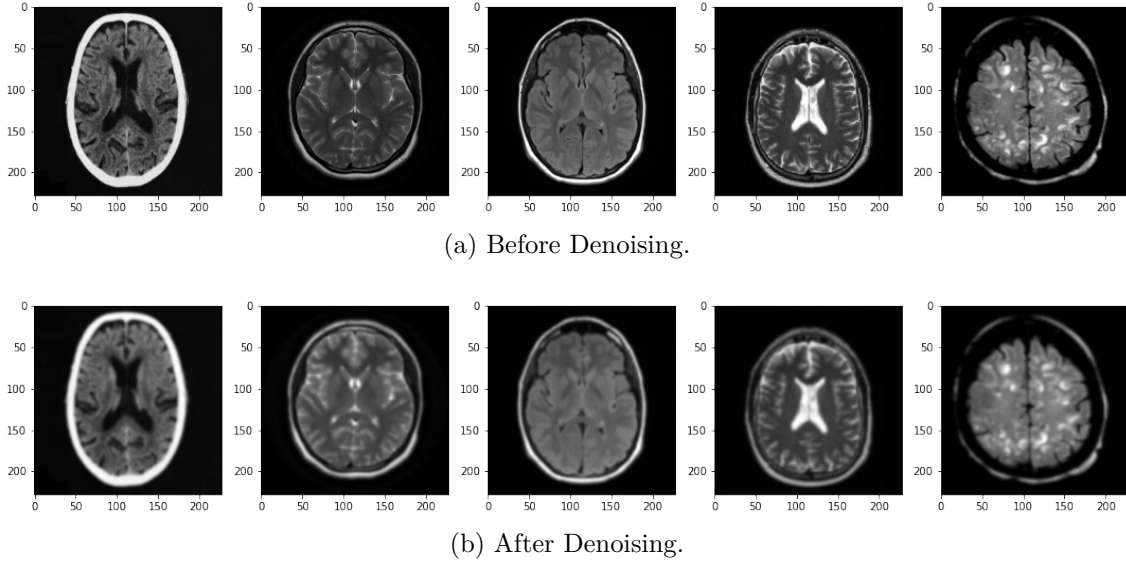

(a) Before Denoising.

(b) After Denoising.

Figure 4.2: Before and After Denoising Image using Gaussian Filter.

Image Resizing

To get optimal performance, each type of CNN architecture necessitates a different image size to resize or distort our image from the one-pixel grid to another. Image resizing is applied based on the model architecture. For example, in the VGG16 architecture, the input size of the image is resized to 224x224. For ResNet50, the input size is resized to 224x224, whereas for Xception and InceptionV3, it is resized to 299x299. The input shape is set as (224,224) in our proposed model.
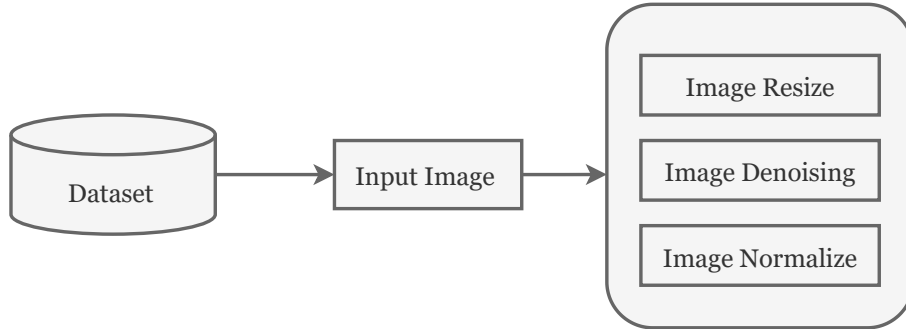

Figure 4.3: Data Pre-processing.

4.2.3   Splitting Train and Test Set

This data set is divided into two sections: a training set and a testing set. Our CNN models will be trained using batch size and epochs on the training set and, after that, evaluated on the test set to determine their accuracy. Precision, recall, f1-score, and accuracy will be used to compare models. The model's accuracy was tested on a subset of the 3000 MRI images in our Br35H dataset, which was separated into two sets: 80% for training and the other 20% for testing. For our model to learn the pattern, we have shuffled the data, which implies that we have mixed up 1's and 0's in each batch. It is done to avoid making the trained model biased. Because in some cases, if we continuously show the same labeled picture for a continuous period, the model starts to memorize it and does not train properly.
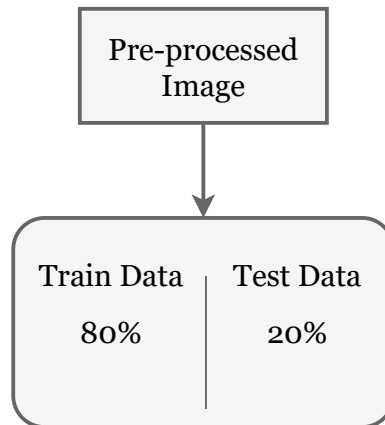
Figure 4.4: Data Splitting.

Chapter 5

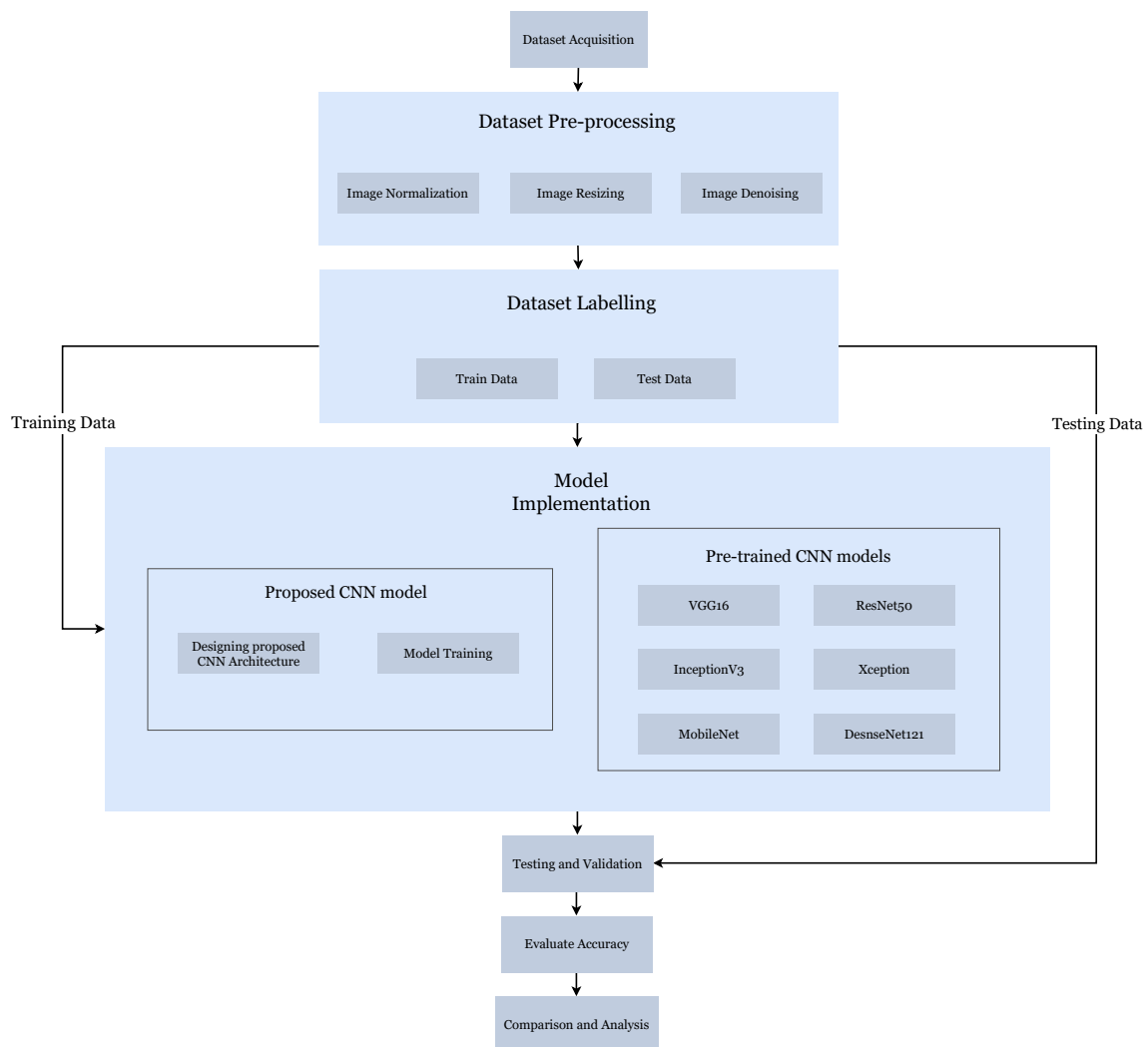# Research Methodology

5.1   Proposed Methodology

Figure 5.1: Workflow of the proposed methodology.

5.2 Used Architectures
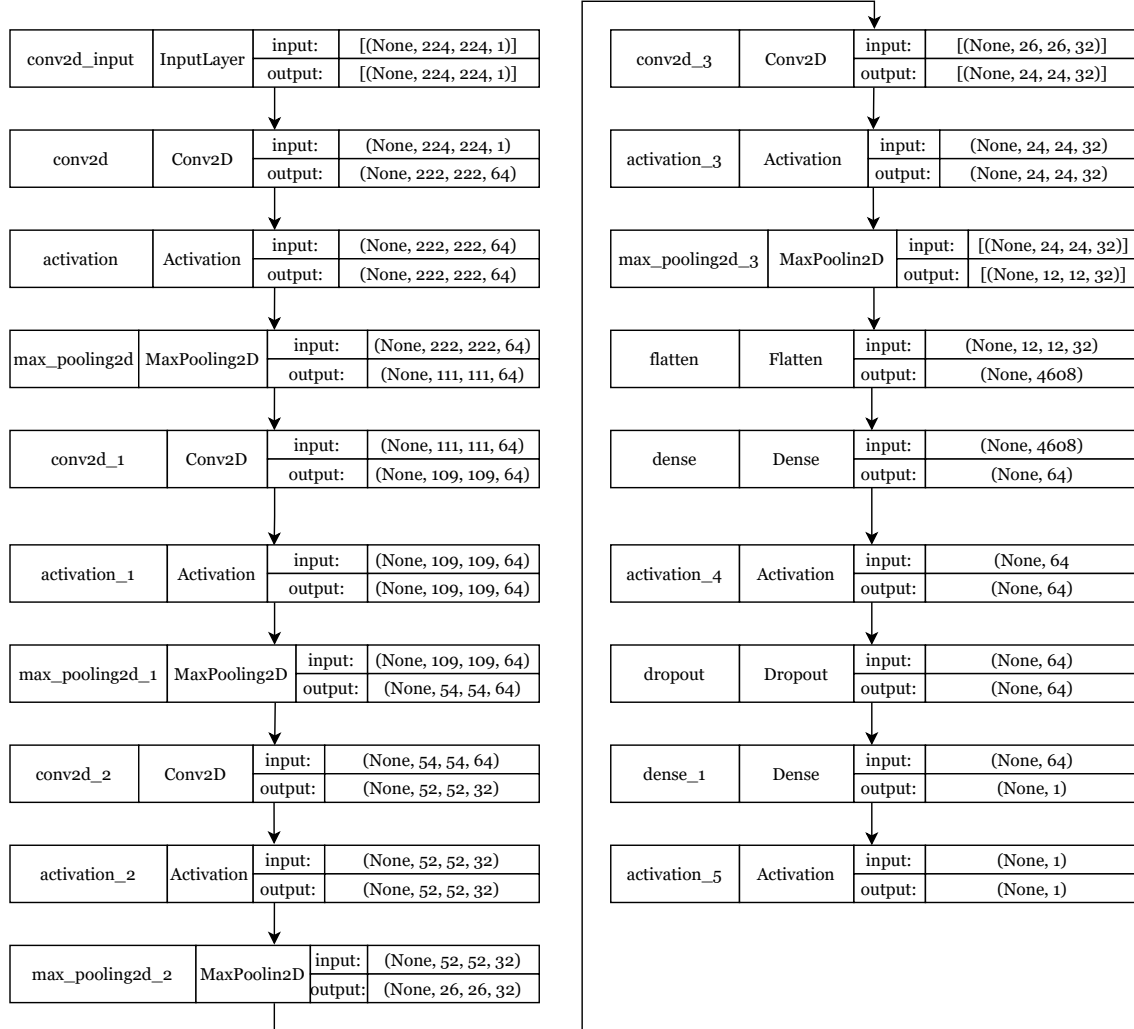
5.2.1 Proposed Model

Figure 5.2: Proposed Model Architecture.

After pre-processing our data, we decided to use Keras as our initial Application Performance Interface (API). This was imported from Google's own open source framework, which is used for machine learning and primarily for image processing data called TensorFlow. TensorFlow made it really accessible for developers like us to implement their in built libraries. We can easily import the Keras API from TensorFlow and implement its various attributes. To name a few, we used the utility named "Normalize" which helped us normalize our dataset in pre-processing. As our model is a sequential model, we used the Keras model library and imported it sequentially. Basically, a sequential model is used when the architecture is built in a multiple-layer stacked way. This is a more simplistic version that works very fast and efficiently for data inputs like our dataset. A sequential model is usually used when the input is given as sequential data. Furthermore, we added different hidden layers with each specified filter for each layer. For that, we have imported conv2D from

the built-in layers in the Keras library. This layer creates a "Convolution Kernel" which is convolved with the specified layer input and generates a tensor of outputs. This attribute also has a bias function, which is initially set to false for unbiased output.

For the pooling function, we imported maxpooling2D from the keras.layers library. It basically down samples the input format along its dimensions. It takes the highest value over an input window, which is predefined by the user by declaring 'pullsize' which is used for each channel of inputs. We can observe the maxpooling2D in action in our model architecture, which downsizes the input frame by half, i.e., if the input is taken as (224,224) dimension, the output will be (112,112) dimension after the maxpooling2D is done. This eases the computational power requirements and can also be a great solution for reducing over-fitting for a neural network.

We used an activation function that helps neural networks deeply understand 'complex patterns' in the data usually after a neuron layer has completed its operation, there is an activation function built-in at the end that decides the next neuron to be fired. In binary classification, the output layer generates binary output (0 or 1). So, in the last layer, we used 'sigmoid' as our activation function to ensure binary classification. In between the input and output layers, all the hidden layers have 'ReLU' as the activation function because it is more complex and advanced at neural firing within the hidden layers. ReLU ensures that the computational power doesn't increase exponentially within each layer, thus making it more efficient. We decided to use ReLU as our preferred activation function to avoid a major problem called "Vanishing Gradient" that occurs while using the sigmoid function.

Another crucial part of the neural network is 'dropout' which is also imported from the same library implemented in our model. It takes an input in percentage and randomly shuts down the same percentage of neurons in each layer to ensure that all neurons are learning. It's basically a nullifying agent for some neurons towards the upcoming layers while leaving the other neurons unchanged. Dropout is the main reason that a CNN model is prevented from over-fitting on the training data. In our case, we used a dropout of 0.7 or 70% right before the output layer.

Moreover, we used the "Flattening" layer imported from the same Keras library, which is used to convert data from multiple dimensions to a one-dimensional array, which is used as the input for the next layer. It is a crucial part of the process because it is used as a medium for inputs to go from one layer to another. It is generally connected to the output layer, also known as the final classification layer, to make it a fully connected layer that is used for generating output.

Finally, we used 'dense' layers, which are used to feed all the outputs gathered from the previous layer to all the neurons, where each neuron provides one single output to the last layer. In the final layer, it is necessary to use a dense value set of 1 because we need binary classification. So, in our proposed model, the layer takes an input of 64, and because of this dense, the output is only 1, representing binary outputs.

The proposed model consists of 4 hidden convolutional layers where the first takes (222, 222) size as input, and then, after max pooling, the next conv2d layer takes (109, 109) as input. The last layers take half of the previous layers using the same process. In addition to the convolutional layers, there are 4 max-pooling 2d layers and two dense layers, making the proposed model of 11 layers excluding the output layer. The proposed model has a total of 360,321 parameters, all of which are trainable.

5.2.2 VGG16

VGG is short for "Visual Geometry Group". This model emphasizes the depth of CNN, which was not plausible in the earlier CNN models. VGG specializes in image classification, usually working with an extensive image dataset also known as ImageNet [28].

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 x 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 5.1: VGG Model Architecture [28].

The proposed model used here is VGG16, which has 16 layers of the convolutional network. This model is more efficient and capable than AlexNet. The VGG-16 network has good accuracy even when prolonged training has minimal image data

sets. The VGG model inputs a 224x224 fixed-size image in the training phase. The input taken is an RGB image. The photo is divided into pixels upon input, and each pixel is passed through a stack of CNN layers. The convolution filter size of each CNN layer is fixed at (3x3) with the same padding as the first two layers. Then, following a stride (2, 2) max pool layer, two layers with 256-layer convolution filters and filters are applied (3, 3). There is a max-pooling layer of stride (2, 2). Two hundred fifty-six filters are used in the second convolution layer (3, 3). Two sets of three convolution layers and a maximum pool layer follow. The final layer is a softmax classifier. All hidden layers are activated with ReLU [60].

5.2.3 MobileNet

Convolutional neural networks, such as MobileNet, are ideal for mobile and embedded application domains because of their portability. In order to develop lightweight deep neural networks, MobileNets employ a simplified architecture that leverages depth-wise separable convolutions. The latency-accuracy tradeoff is simplified to just two simple global hyper-parameters. Model developers can use several different hyper-parameters to select the appropriate model size for every given problem. MobileNets can be used for item detection, fine-grain categorization, facial characteristics, and large-scale geo-localization [39].
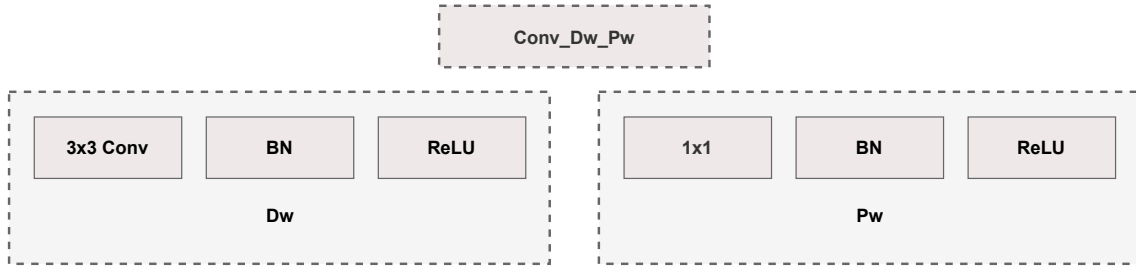

Figure 5.3: MobileNet Model Architecture.

This network can reduce the number of parameters without sacrificing precision. MobileNet requires only 1/33 of the visual geometry group -16 (VGG-16) parameters to achieve the same classification accuracy as ImageNet-1000. It is structured both depth-wise (Dw) and point-wise (Pw). The Dw represents convolutional layers with 3x3 kernels, whereas the Pw represents convolutional layers with 1x1 kernels. Each convolution output is normalized using the batch normalization approach and the rectified linear unit activation function (ReLU)(Li et al., 2018) [45].

5.2.4 ResNet50

In a conventional neural network structure, after processing the input through various layers of neurons, a singular output is generated, and along with it, the loss function is calculated. This loss function is then observed, and some internal changes are made via modifying variables. A backpropagation algorithm is a must in order to do that. The algorithm starts from the end node in backpropagation and goes back to the beginning node. In this backtracking process, the function is the derivative of that particular loss function multiplied with its predecessor function. In that

way, the more backward propagation goes, the lesser the gradient values become.

In the process illustrated above, it is observed that if there are more layers, eventually traversing backwards will produce an increasingly smaller value of gradient. Each iteration will be the derivative of that function multiplied by the previous function. At some point, the gradient value becomes so tiny that it almost counts as 0. That creates the phenomenon named the "Vanishing Gradient Problem." This process hinders the optimization of the neural network layers. So, the accuracy decreases significantly. In order to mitigate this problem, a model called "ResNet" which is short for residual network, was created. This takes advantage of a unique feature known as the "skip function," which effectively connects the inputs to the convolution block's output while skipping intermediate layers for more remarkable performance. In other words, the input will be connected to both the convolution layer for processing the function and the output of the convolution block.
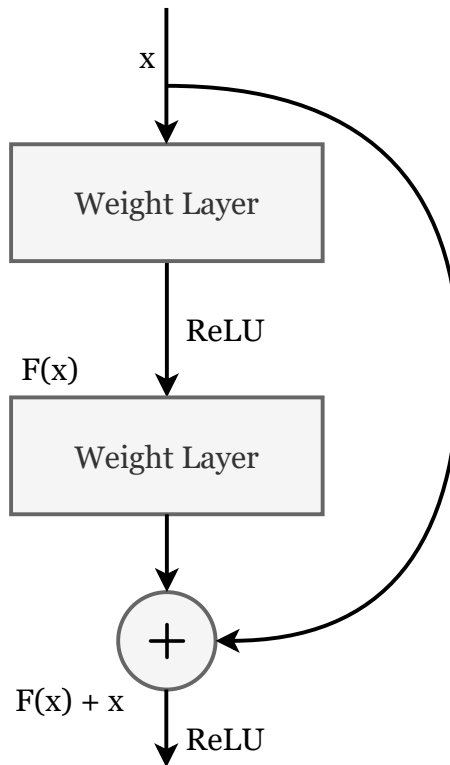

Figure 5.4: The Residual Block.

The ResNet50 model is a 50-layer convolutional neural network (CNN). While the ResNet50 architecture is based on the ResNet34 architecture, one significant change exists. That is, the construction block was redesigned as a bottleneck because of issues about the training time necessary for the layers. This time, a three-layer stack was used rather than the prior two[60]. As a result, ResNet34's two-layer blocks were replaced with a three-layer bottleneck block, resulting in the ResNet50 architecture. This model is far more accurate than the 34-layer ResNet model. The 50-layer ResNet delivers 3.8 billion FLOPS of performance.
As depicted in the figure, the output function for ResNet y is the sum of the func-

tions generated from the convolution layers and the input itself. The neural network aims to provide the most accurate output as close as feasible to the input. In order to make the model as accurate as possible, the goal of this structure is to make the F(x) value 0. That is how the output (y) will be equal to the input (x). Thus, the model predicts correctly.

This ResNet architecture is implemented by various neural network models for image classification. Due to the nature of different problems and accuracy expectations, there are different iterations of ResNet, which are defined by the number of layers each model has. Such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, and so on. Each of them is designed with different layers and designed with different dimensions of matrices to process the data and generate output.

Additionally, there are more versions of Residual Networks which are given with their layer distribution in the figure below:

Layer Name	Output Size	18-layer	34-layer	50-layer	101-layer
conv1	112x112	7x7, 64, stride 2			
conv2_x	56x56	3x3 max pool, stride 2			
conv2_x	56x56	$\begin{bmatrix} 3*3, 64 \\ 3*3, 64 \end{bmatrix}$ x2	$\begin{bmatrix} 3*3, 64 \\ 3*3, 64 \end{bmatrix}$ x3	$\begin{bmatrix} 1*1, 64 \\ 3*3, 64 \\ 1*1, 256 \end{bmatrix}$ x	$\begin{bmatrix} 1*1, 64 \\ 3*3, 64 \\ 1*1, 256 \end{bmatrix}$ x3
conv3_x	28x28	$\begin{bmatrix} 3*3, 128 \\ 3*3, 128 \end{bmatrix}$ x2	$\begin{bmatrix} 3*3, 128 \\ 3*3, 128 \end{bmatrix}$ x4	$\begin{bmatrix} 1*1, 128 \\ 3*3, 128 \\ 1*1, 512 \end{bmatrix}$ x4	$\begin{bmatrix} 1*1, 128 \\ 3*3, 128 \\ 1*1, 512 \end{bmatrix}$ x4
conv4_x	14x14	$\begin{bmatrix} 3*3, 256 \\ 3*3, 256 \end{bmatrix}$ x2	$\begin{bmatrix} 3*3, 256 \\ 3*3, 256 \end{bmatrix}$ x6	$\begin{bmatrix} 1*1, 256 \\ 3*3, 256 \\ 1*1, 1024 \end{bmatrix}$ x6	$\begin{bmatrix} 1*1, 256 \\ 3*3, 256 \\ 1*1, 1024 \end{bmatrix}$ x23
conv5_x	7x7	$\begin{bmatrix} 3*3, 512 \\ 3*3, 512 \end{bmatrix}$ x2	$\begin{bmatrix} 3*3, 512 \\ 3*3, 512 \end{bmatrix}$ x3	$\begin{bmatrix} 1*1, 512 \\ 3*3, 512 \\ 1*1, 2048 \end{bmatrix}$ x3	$\begin{bmatrix} 1*1, 512 \\ 3*3, 512 \\ 1*1, 2048 \end{bmatrix}$ x3
	1x1	average pool, 1000-d fc, softmax			
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9

Table 5.2: ResNet Model Architecture [34].

In the proposed research, the chosen dataset was Br35h. This image classification problem is implemented in ResNet50.

5.2.5 Xception

Xception is an abbreviation for "Extreme Inception". It is an architecture for convolutional neural networks that is solely composed of depthwise separable convolution layers. The network's feature extraction is based on thirty-six convolutional layers of the Xception architecture. A convolutional base will be followed by a logistic regression layer in this case, which is focused on image categorization. The logistic regression layer can be preceded by completely linked layers. In total, there are 36 convolutional layers, with the exception of the first and last layers, organized into 14 modules by linear residual connections [38]. The Xception model architecture is shown below:
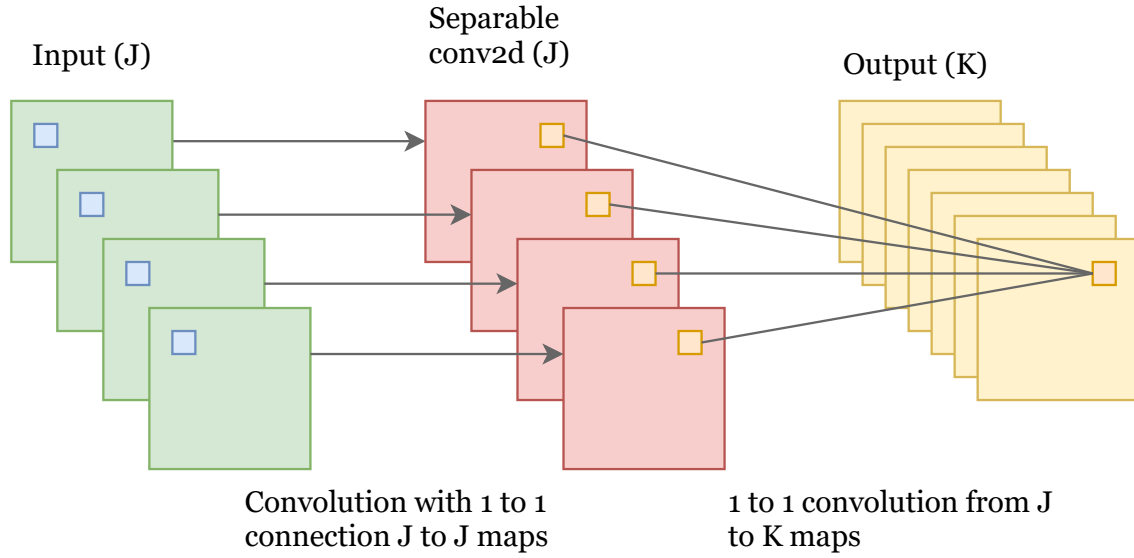
Figure 5.5: Xception Model Architecture.

For the most part, the Xception architecture comprises a vertically stacked linear stack of depth-separable convolution layers linked together by residuals. This concept makes the definition and modification of the architecture much simpler; when a high-level library such as Keras is used, it only requires 30 to 40 lines of code to accomplish this.

5.2.6 InceptionV3

InceptionV3 is a widely renowned convolutional neural network architecture acquired from the Inception architecture that includes label smoothing, factorized 7 x 7 convolutions, and an auxiliary classifier to carry label information more profoundly into the network as batch normalization for sidehead layers. For more significant model adaption, the Inception V3 model (with 42 layers) uses several approaches to optimize the task, resulting in higher efficiency and less computationally expensive than InceptionV1 and InceptionV2. There are some major modifications made to Inception V3:

- **Factorization into Smaller Convolutions**: Assume we have a 5x5 convolutional layer with a high computational cost. As a result, the 5x5 convolutional layer was replaced with 3x3 convolutional layers to lower the computational cost.
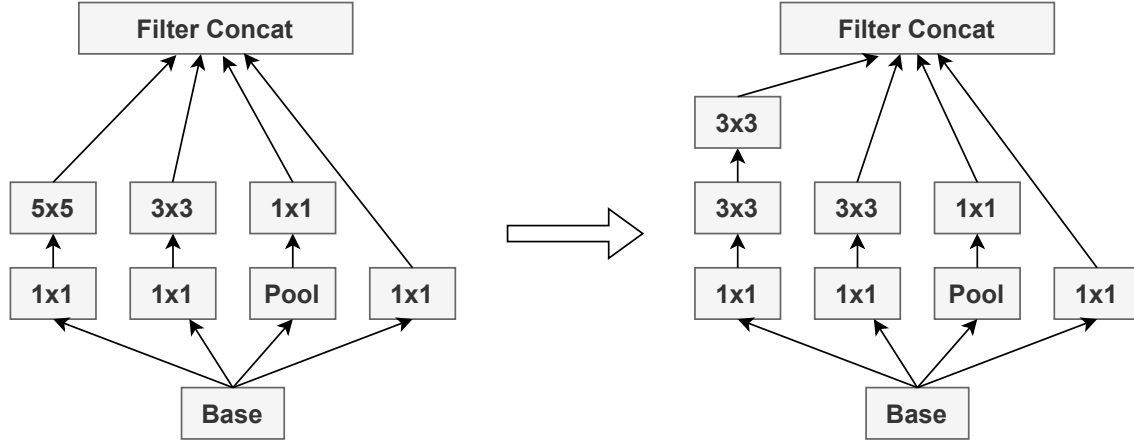

Figure 5.6: After factorization into smaller convolutions.

- **Spatial Factorization into Asymmetric Convolutions**: Asymmetric convolutions are a superior alternative to factorization into smaller groups for making the model more efficient. As a result, we use a 1x3 convolution followed by a 3x1 convolution to substitute the 3x3 convolutions.
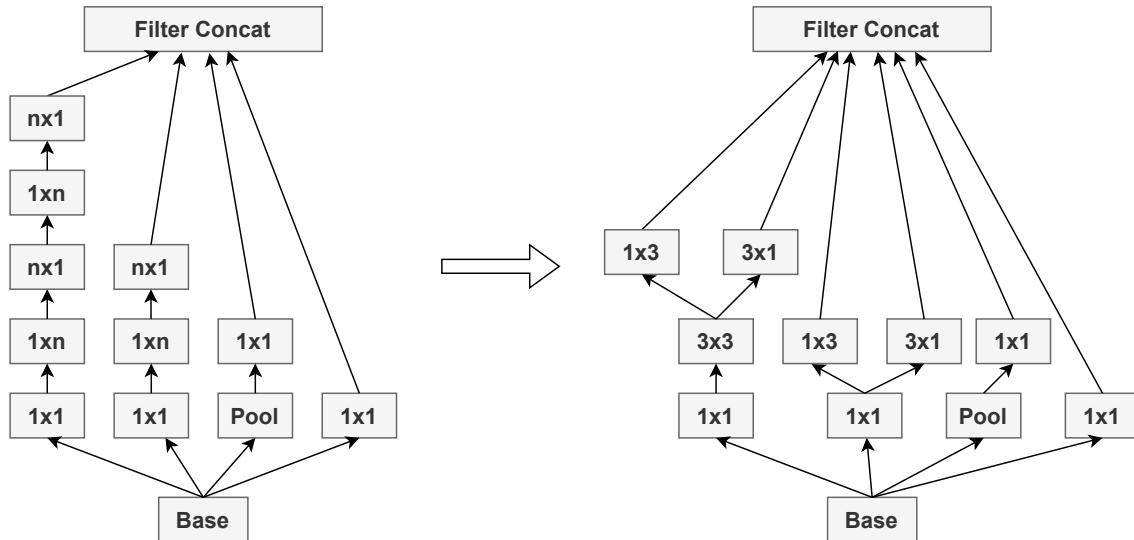
Figure 5.7: Asymmetric Convolutions.

- **Utility of Auxiliary Classifiers**: Using an Auxiliary classifier to enhance the convergence of very deep learning models is the goal. In very deep networks, the auxiliary classifier is primarily used to overcome the vanishing gradient problem. In the initial stages of the training, the auxiliary classifiers made no difference. However, in the end, the system with auxiliary classifiers

outperformed the network without them in terms of accuracy. As a result, the auxiliary classifiers in the Inception V3 model architecture operate as a regularizer.

- **Efficient Grid Size Reduction**: The activation parameter of the networking filters is enhanced in the inception V3 model in order to lower the grid size efficiently. Suppose, if we have a $(d \cdot d)$ grid with k filters, after reduction, we get a $(\frac{d}{2} \cdot \frac{d}{2})$ grid with 2k filters. This is accomplished by concatenating two concurrent blocks of convolution and pooling.
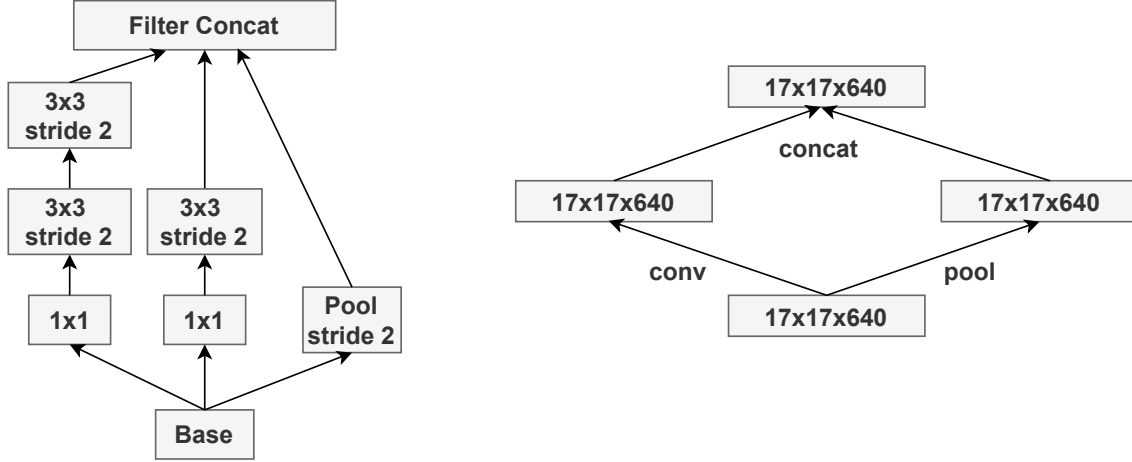
Figure 5.8: After Reducing Grid Size.

The inception V3 model has 42 layers in total, which is slightly more than the preceding inception V1 and V2 models. However, this model's efficiency is truly remarkable.

5.2.7 DenseNet121

DenseNet is a relatively recent development in visual object recognition using neural networks. DenseNet is comparatively similar to ResNet, with a few key distinctions.ResNet uses an additive technique (+) to combine the previous layer's output with the output of the subsequent layer, whereas DenseNet uses a concatenative approach to combine the output of the previous layer with the output of the subsequent layer. DenseNet was initially developed to solve the vanishing gradient phenomenon in high-level neural networks. In the simplest terms, the path between the input and output layers is longer, resulting in data loss along the way. DenseNet is a classic network type. When the composite function operation is used, the previous layer's output becomes an input for the second layer. This composite process comprises the convolution layer, the pooling layer, the batch normalization layer, and the non-linear activation layer. According to this approach, the network has L(L+1)/2 direct connections. L represents the number of layers in the architecture. DenseNet is offered in a number of configurations, including the DenseNet-121, the DenseNet-160, and the DenseNet-201. The numbers denote the number of layers in the neural network. DenseNet121 comprises 121 layers, 120 of which are convolutional and four of which are AvgPool. Since DenseNets require fewer parameters

and allow for feature reuse, they result in more compact models and have exhibited greater performance and results when compared to their classic CNN or ResNet counterparts.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112x112	7x7 conv, stride 2			
Pooling	56x56	3x3 max pool, stride 2			
Dense Block (1)	56x56	$\begin{bmatrix} 1*1conv \\ 3*3, conv \end{bmatrix}$ x6	$\begin{bmatrix} 1*1conv \\ 3*3, conv \end{bmatrix}$ x6	$\begin{bmatrix} 1*1conv \\ 3*3, conv \end{bmatrix}$ x6	$\begin{bmatrix} 1*1conv \\ 3*3, conv \end{bmatrix}$ x6
Transition Layer (1)	56x56	1x1 conv			
	28x28	1x1 conv			
Dense Block (2)	14x14	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x12	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x12	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x12	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x12
Transition Layer (2)	28x28	1x1 conv			
	14x14	1x1 conv			
Dense Block (3)	14x14	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x24	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x32	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x48	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x64
Transition Layer (3)	14x14	1x1 conv			
	7x7	1x1 conv			
Dense Block (3)	14x14	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x16	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x32	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x32	$\begin{bmatrix} 1*1conv \\ 3*3conv \end{bmatrix}$ x48
Classification Layer		average pool, 1000-d fc, softmax			
		$1.8\text{x}10^9$	$3.6\text{x}10^9$	$3.8\text{x}10^9$	$7.6\text{x}10^9$

Table 5.3: DenseNet Models Architecture.

5.3 Evaluation Method

5.3.1 Graphical Analysis

To compare among the 7 models, including the proposed one, we will take the accuracy curve, loss curve, and receiver operating characteristics (ROC) curve under consideration. The model accuracy will be represented by "Train Accuracy Curve vs. Validation Accuracy Curve," whereas the model loss will be represented by "Train Loss Curve vs. Validation Loss Curve" to better understand how accurately the models are training and predicting. This curve is based on the rate of change with increasing epochs. On a graph, we will plot the accuracy and loss for each epoch we set during training and testing the model, with the number of epochs on the x-axis and accuracy and loss on the y-axis. Here, 10 epochs were set for MobileNet,

VGG16, InceptionV3, and 20 epochs for ResNet50, Xception, DenseNet121, and the proposed model in our model implementation.

5.3.2 Confusion Matrix

In order to evaluate a classifier, several metrics can be used to visualize it. In that case, accuracy is used most widely. However, it only works if the test dataset contains an equal number of samples from each class of individuals. However, while working with a dataset where the distribution is uneven, more precise performance metrics can be used, such as 'Confusion Matrices' [64]. In our case, we will try to distinguish all the true and false predicted results through a confusion matrix to get a better visual understanding of how the machine is responding to certain classes.

Predicted Values

	Negative	Positive
Negative	TN	FP
Positive	FN	TP

Figure 5.9: Basic 2x2 Confusion Matrix.

Here,

- **TP (True Positive)**: The predicted value matches with the actual value where the actual value was positive (P) and the predicted value was also positive (P).

- **TN (True Negative)**: The predicted value matches with the actual value where the actual value was negative (N) and the predicted value was also negative (N).

- **FP (False Positive)**: The predicted value does not match with the actual value where the actual value was negative (N) and the predicted value was positive (P).

- **FN (False Negative)**: The predicted value does not match with the actual value where the actual value was positive (P) and the predicted value was negative (N).

Using this confusion matrix, we will get our classification report which is a great way to compare several models. In the Br35h dataset, there are two classes, yes and

no, with 3,000 images. In that, we will use a 2x2 confusion matrix to classify the images into 2 classes and find their classification report. The classification report will have the following attributes:

- **Classification Accuracy**: It is the ratio of the number of correctly classified data samples to the total amount of data.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad (5.1)$$

- **Recall or Sensitivity**: It shows how many of the actual positive cases the models were able to predict. It is useful when the number of false negatives (FN) is higher than false positives (FP). It is the ratio of true positive values to the actual result.

$$Recall = \frac{TP}{TP + FN} \qquad (5.2)$$

- **Precision**: It shows how many of the correctly predicted results are positive. It is useful when the number of false positives (FP) is higher than false negatives (FN). It is the ratio of true positive values to the predicted results.

$$Precision = \frac{TP}{TP + FP} \qquad (5.3)$$

- **F1-Score**: It's the harmonic mean of precision and recall that gives a composite picture of the two measures. When precision equals recall, the F1-score reaches its maximum value.

$$F1 - Score = \frac{2 * precision * recall}{precision + recall} \qquad (5.4)$$

Chapter 6

Experimental Results and Analysis

For performance evaluation, we have taken model accuracy, model loss, ROC, recall, precision, f1-score, and support under consideration along with the confusion matrix. We have used 10 epochs for VGG16, InceptionV3, and MobileNet and 20 epochs for ResNet50, Xception, and DenseNet121 models. An epoch here denotes one complete pass for the training data in the context of machine learning. And finally, for the proposed model, after thoroughly tweaking and observing, we have decided to run the model for 20 epochs.

6.1 Result Analysis

6.1.1 Graphical Analysis

This paper conducted a total of three different graphical analyses on each CNN model. Here, the first graph represents two distinctive curves, where one indicates 'Validation accuracy' and the other one is 'Training Accuracy.' Training accuracy means when a model is trained on a partial part of the dataset. After the model starts training, the percentage of training data can be identified. This number is relatively close to 100 percent because the model has already seen those data whilst training. On the other hand, validation accuracy is purely generated from the test dataset, which has not been used while training. After the training is completed, the test dataset is then fed into the model to identify and classify properly. Furthermore, the percentage of the corrected prediction is represented in the validation accuracy.

In the same way, the second graphical analysis shows two lines differentiated by two colors. 'Training Loss' is indicated by the red curve, and 'Validation Loss' is represented by the blue curve. A loss is a representation of how badly a model is predicting individual data. With gradual epochs, the model loss systematically comes down as the model becomes more accurate.

The ROC (Receiver Operating Characteristic) curve is the graphical representation of a CNN model that indicates the performance of an individual classification model at all possible classification thresholds. The X-axis represents the false positive (FP) rate and the Y-axis represents the true positive (TP) rate. The slope of TP vs FP represents how accurately the model classifies every individual sample as

positive where the result is actually positive versus when the model predicts positive when the actual value is negative. The ROC curve must be upwards, meaning the slope should be greater than 1 to indicate that the model is predicting accurately throughout the whole process. In most of the models, we see a sharp rise in the initial values, closing in on almost 1.0 and becoming stagnant from there to the end of the graph. The dotted line represents how much area is under the curve, which is the deciding value that's used to measure whether the model is producing an effective result or not.

VGG16
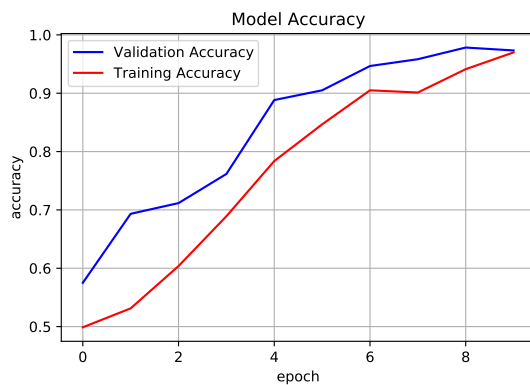
Figure 6.1: VGG16 Model Accuracy.   Figure 6.2: VGG16 Model Loss.
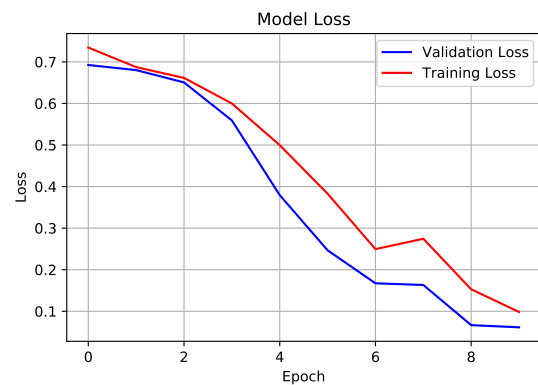
Figure 6.3: VGG16 Model Accuracy and Model Loss Curve.

With 10 epochs, the VGG16 model's accuracy rose from 50% training accuracy at epoch 1 to 97% training accuracy at epoch 10. Although the validation accuracy started a bit higher at 58% and rose to almost the same as the training accuracy. The loss curve of both training and validation starts at around 70% and drops down to almost 5% after.
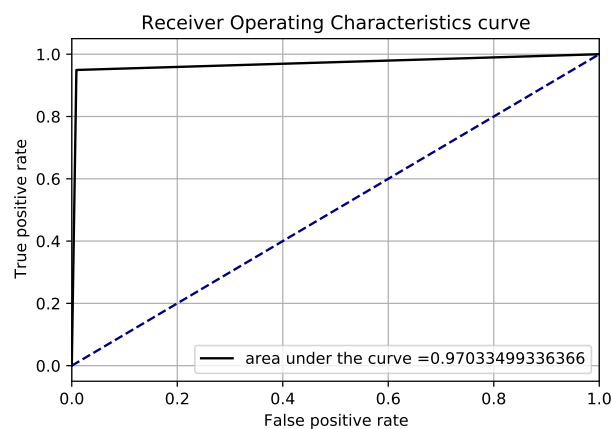
Figure 6.4: VGG16 ROC Curve.

The ROC curve of VGG16 starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating

the area under the curve to 0.97 which means 97% of the total surface available, indicating the model is very accurate.
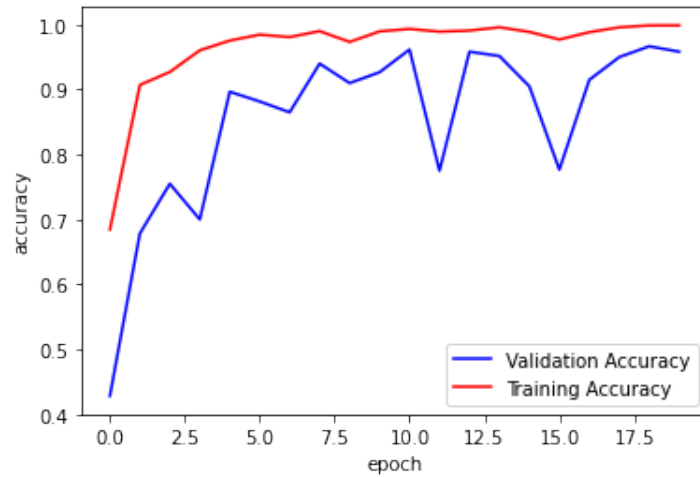
ResNet50

Figure 6.5: ResNet50 Model Accuracy.

With 20 epochs, the ResNet50 model's accuracy rose from 77% training accuracy at epoch 1 to almost 98.1% training accuracy at epoch 10. But the validation accuracy started at 43% and rose to almost the same as the training accuracy, that is 94.5%.
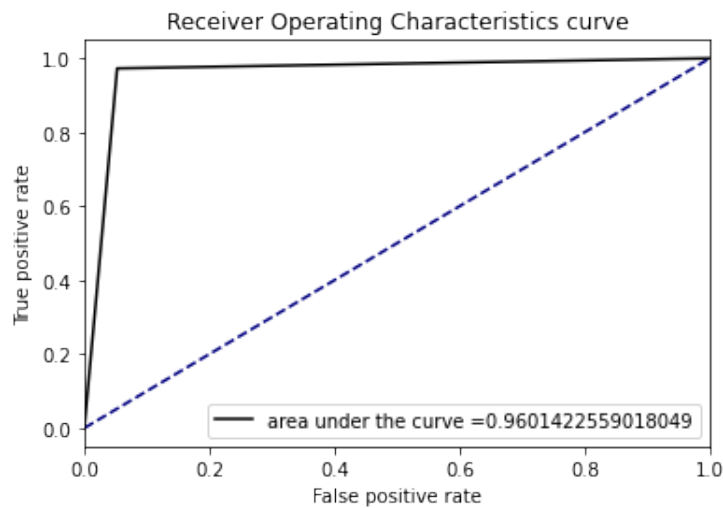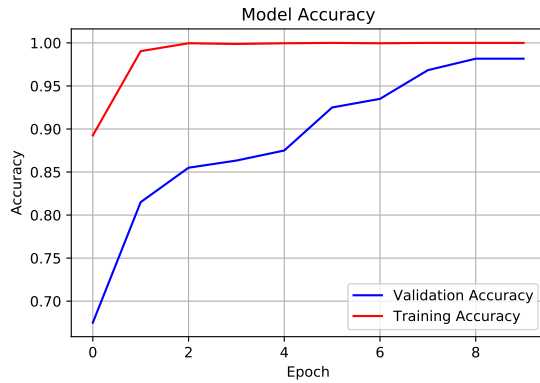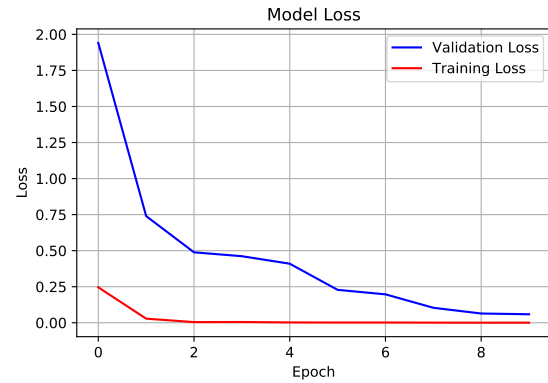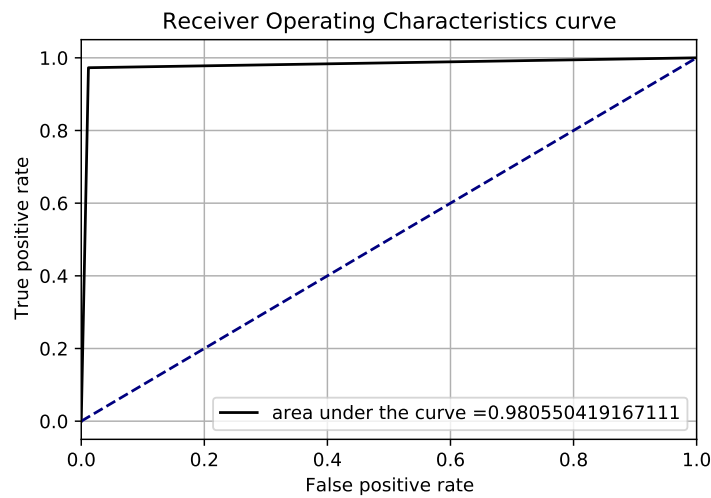

Figure 6.6: ResNet50 ROC Curve.

The ROC curve of ResNet50 starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating the area under the curve to 0.96 which means 96% of the total surface available, indicating the model is very accurate.

InceptionV3

Figure 6.7: InceptionV3 Model Accuracy.   Figure 6.8: InceptionV3 Model Loss.

Figure 6.9: InceptionV3 Accuracy and Model Loss Curve.

With 10 epochs and 38 steps per epoch, the InceptionV3 model's accuracy rose from 88% training accuracy at epoch 1 to 99% training accuracy at epoch 10. But the validation accuracy started at 65% and rose up to almost 98%. The loss curve of training starts at around 25% and drops to almost 1%. The loss validation curve starts at almost 200% and rapidly drops to 11%.

Figure 6.10: InceptionV3 ROC Curve.

The ROC curve of InceptionV3 starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating the area under the curve to 0.98 which means 98% of the total surface available, indicating the model is very accurate.
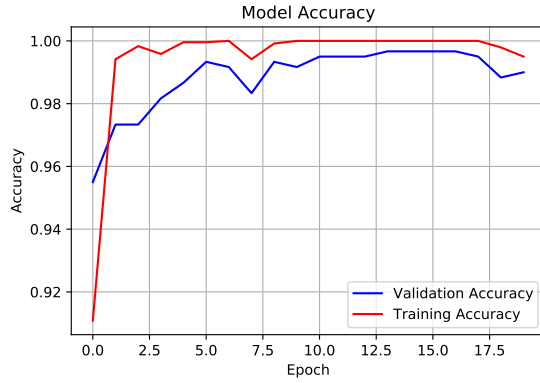
Xception
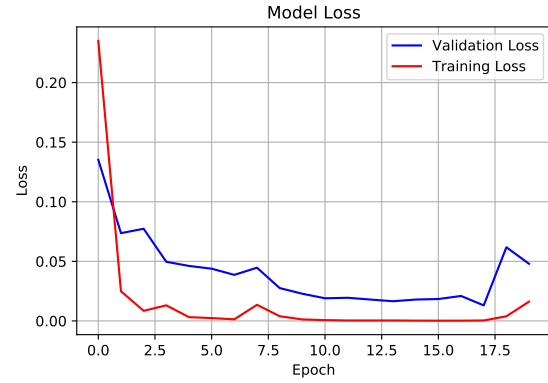

Figure 6.11: Xception Model Accuracy.

Figure 6.12: Xception Model Loss.

Figure 6.13: Xception Model Accuracy and Model Loss Curve.

With 20 epochs and 75 steps per epoch, the Xception model's accuracy rose from 90% training accuracy at epoch 1 to 99% training accuracy at epoch 20 whereas the validation accuracy started at 95.7% and rose up to almost the same as the training accuracy, that is 99%. The loss curve of training starts at around 25% and drops to almost 2% after 20 epochs. The loss validation curve starts at almost 14% and drops down to just over 5%.
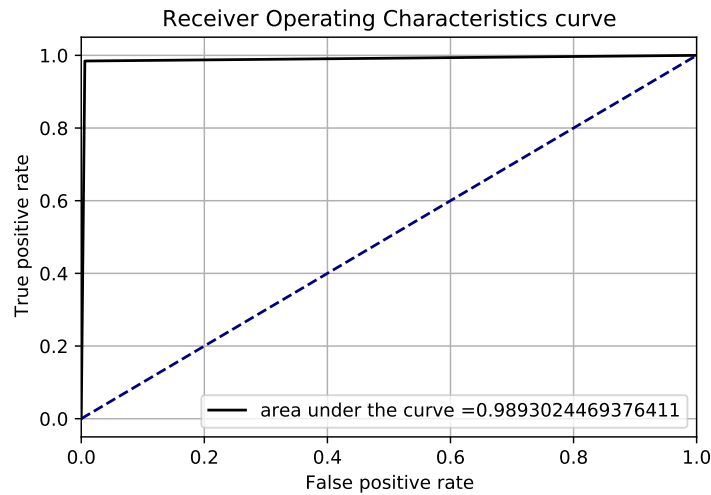

Figure 6.14: Xception ROC Curve.

The ROC curve of Xception starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating the area under the curve to 0.9893 which means 98.93% of the total surface available, indicating the model is very accurate.
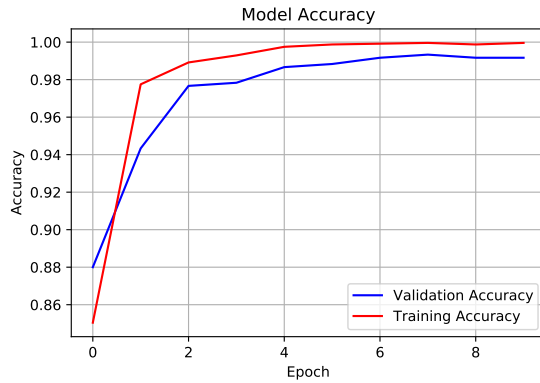
MobileNet
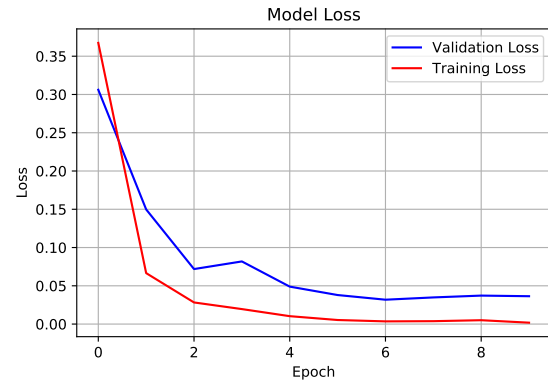

Figure 6.15: MobileNet Model Accuracy.   Figure 6.16: MobileNet Model Loss.

Figure 6.17: MobileNet Accuracy and Model Loss Curve.

With 10 epochs and 75 steps per epoch, the MobileNet model's accuracy rose from 83% training accuracy at epoch 1 to 99% training accuracy at epoch 10. Although, the validation accuracy started a bit higher at 88% and rose up to almost the same as the training accuracy, that is, 99%. The loss curve of training starts at around 36% and drops down to almost 1%. The loss validation curve starts at almost 31% and drops down to 4.5%.
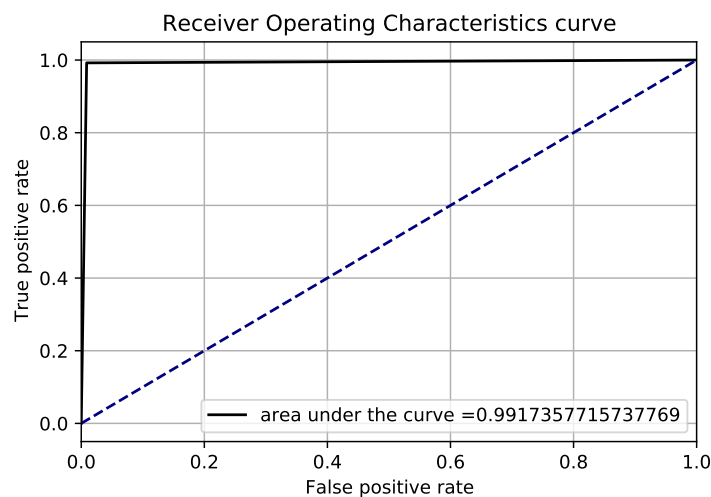

Figure 6.18: MobileNet ROC Curve.

The ROC curve of MobileNet starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating the area under the curve to 0.9917 which means 99.17% of the total surface available, indicating the model is very accurate.
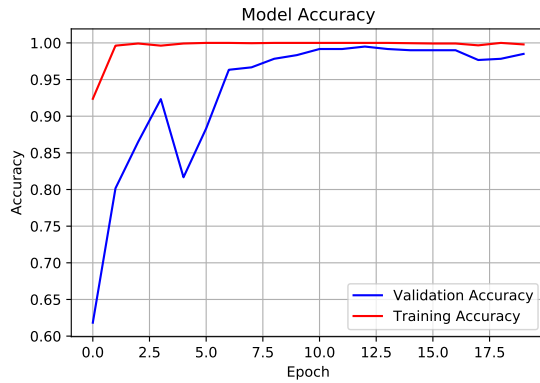
51

DenseNet121

Figure 6.19: DenseNet121
Model Accuracy Curve.

Figure 6.20: DenseNet121
Model Loss Curve.

Figure 6.21: DenseNet121 Model Accuracy and Model Loss Curve.

With 20 epochs and 75 steps per epoch, the DenseNet121 model's accuracy rose from 93% training accuracy at epoch 1 to 99% training accuracy at epoch 20. But the validation accuracy started at 63% and rose up to almost 97.5%. The loss curve of training starts at around 20% and drops to almost 1% after 20 epochs. The loss validation curve starts at almost 15% and drops to 1%.
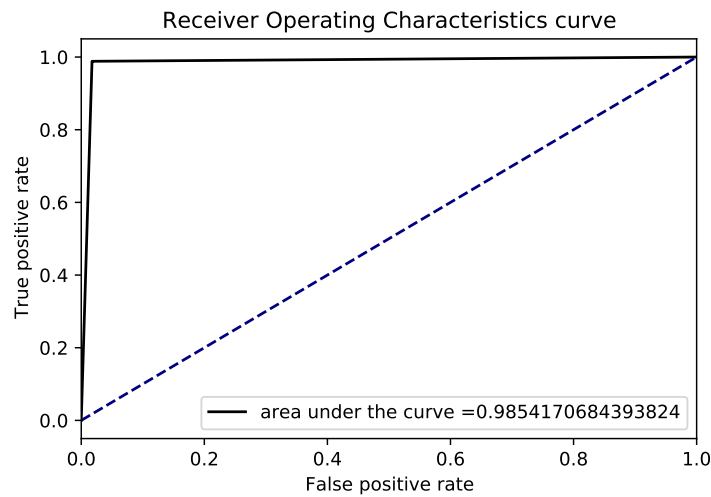

Figure 6.22: DenseNet121 ROC Curve.

The ROC curve of DenseNet121 starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 TP at the end, thus creating the area under the curve to 0.9854 which means 98.54% of the total surface available, indicating the model is very accurate.
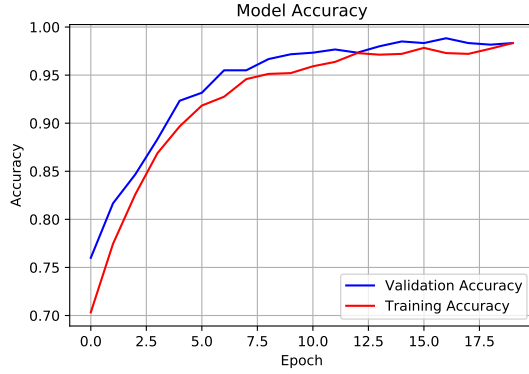
Proposed Model

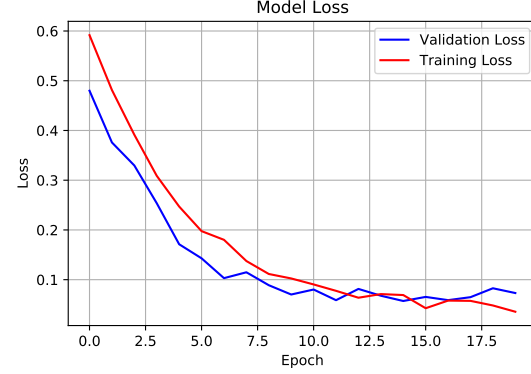
Figure 6.23: Proposed Model Accuracy Curve.

Figure 6.24: Proposed Model Loss Curve.

Figure 6.25: Proposed Model's Accuracy and Model Loss Curve.

With 20 epochs and 150 steps per epoch, the proposed model's accuracy rose from 70.04% training accuracy at epoch 1 to 98.71% training accuracy at epoch 20. The batch size was set to 16. Although, the validation accuracy started a bit higher at 72.67% and rose up to almost 98.5%. The loss curve of training starts at around 58.95% and drops down to almost 2.93%. The loss validation curve starts at almost 52.01% and drops down to 9.76%.
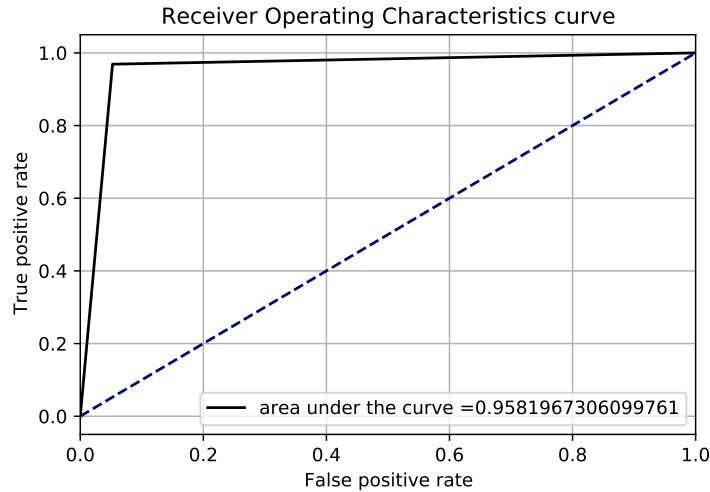

Figure 6.26: Proposed Model's ROC Curve.

The ROC curve of the proposed model starts with a sharp rise because of the high TP vs FP value and the TP value steadily increases to almost 1.0 at the end, thus creating the area under the curve to 0.9581 which means 95.81% of the total surface available, indicating the model is very accurate.

6.1.2 Confusion Matrix

The performance of CNN models among our proposed ones, ResNet50, MobileNet, VGG16, Xception, InceptionV3, and DenseNet121 is compared in detail using confusion matrices for two binary classes, 0 and 1, where 0 means no and 1 means yes. The diagonal elements reflect the correctly categorized classes, whereas anything off the diagonal indicates an erroneous classification. The confusion matrix is divided into three axes: (i) prediction label (class), (ii) true label, and (iii) the value of the heat map (color). The prediction label and true label indicate the prediction class with which we are working. The diagonal of the matrix denotes areas in the matrix where the forecast and the truth are identical; this is where we want to darken the heat map.

VGG16

Figure 6.27: VGG16 Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 240 and the true negative (TN) value is almost 340, with some slight errors counted as total 16.

ResNet50

Figure 6.28: ResNet50 Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 250, and the true negative (TN) value is almost 320, with some slight errors counted as a total of 25.

Inception V3

Figure 6.29: InceptionV3 Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 250 and the true negative (TN) value is almost 340, with some slight errors counted as a total of 11.
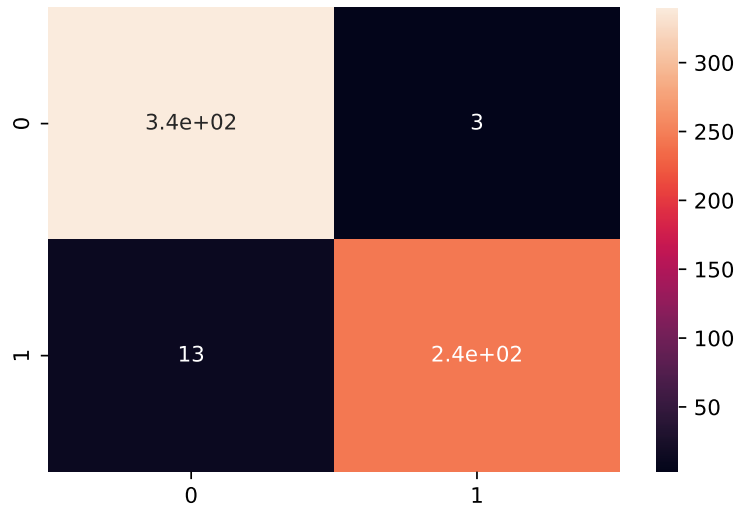
Xception

Figure 6.30: Xception Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 250 and the true negative (TN) value is almost 340, with some slight errors counted as a total of 6.

MobileNet

Figure 6.31: MobileNet Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 260 and the true negative (TN) value is almost 340, with some slight errors counted as a total of 5.
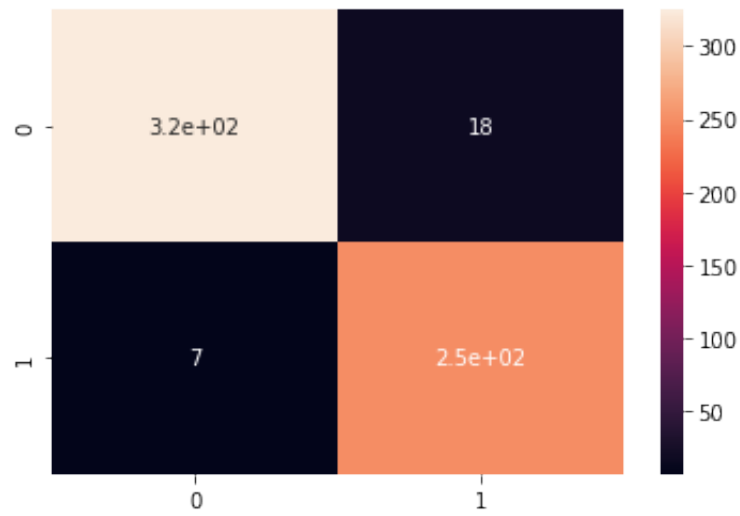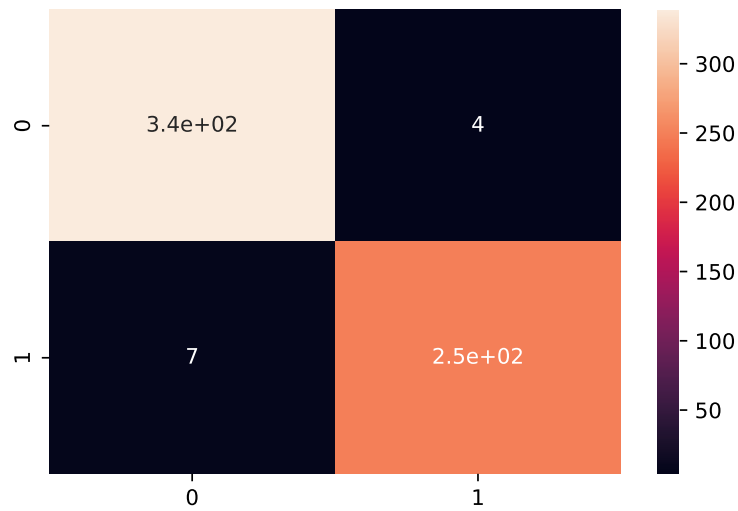
DenseNet121

Figure 6.32: DenseNet121 Confusion Matrix.

Here, it is observed that the true positive (TP) value is almost 250 and the true negative (TN) value is almost 340, with some slight errors counted as a total of 9.

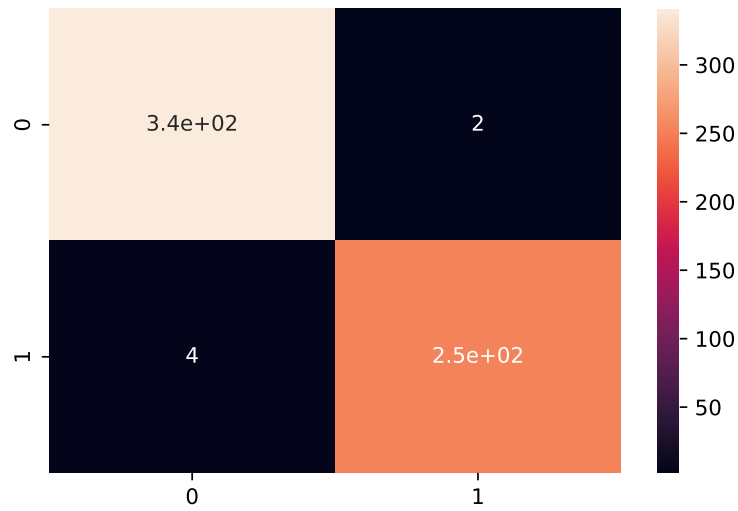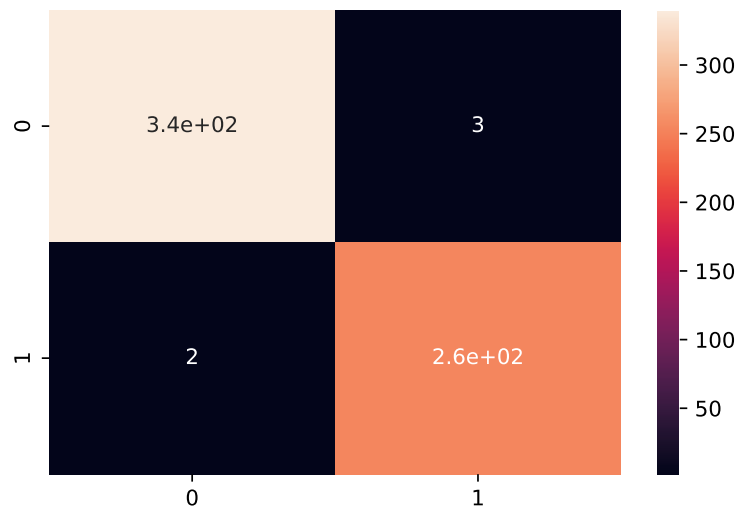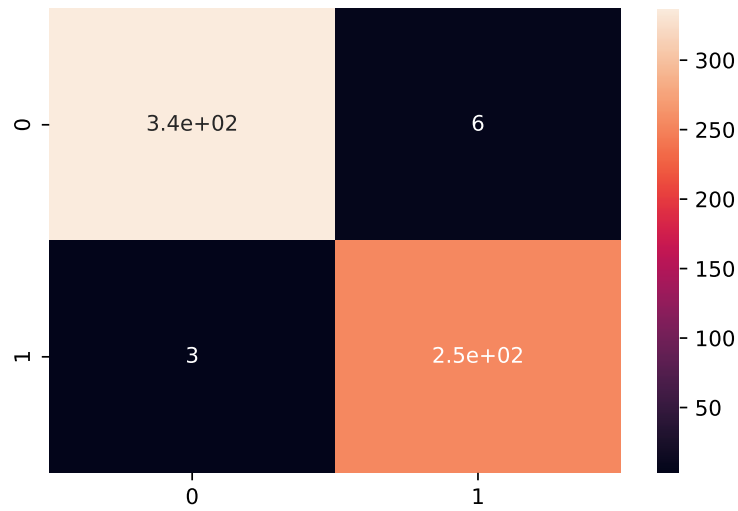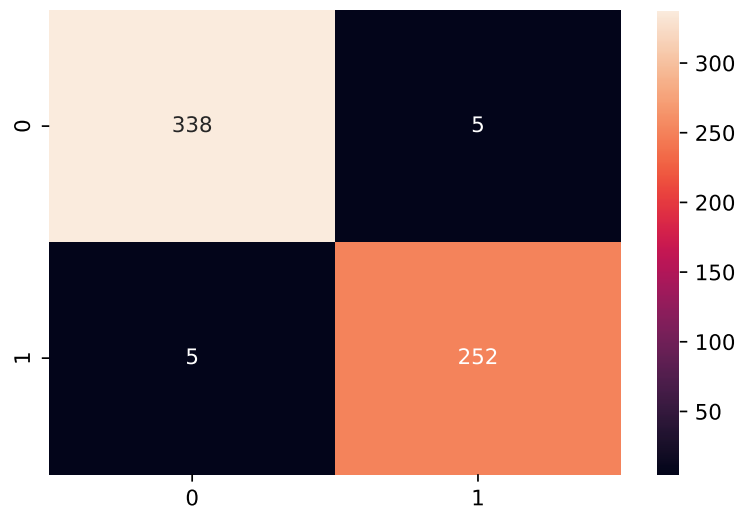**Proposed CNN Model**

Figure 6.33: Proposed Model's Confusion Matrix.

Here, it is observed that the true positive (TP) value is 252, whereas the true negative (TN) value is 338, and some slight errors are counted as a total of 10.

6.2   Result Comparison

Architectures	Precision	Recall	F1-score	Support (0/1 out of 600)	Accuracy
Proposed Model	98%	98%	98%	343/257	98.5%
ResNet50	98%	97%	98%	343/257	94.5%
MobileNet	99%	99%	99%	343/257	99%
VGG16	97%	97%	98%	343/257	97%
Xception	99%	98%	99%	343/257	99%
InceptionV3	98%	97%	98%	343/257	98%
DenseNet121	98%	99%	98%	343/257	98%

Table 6.1: Comparison table among the Proposed Model, ResNet50, MobileNet, VGG16, Xception, InceptionV3 and DenseNet Models.
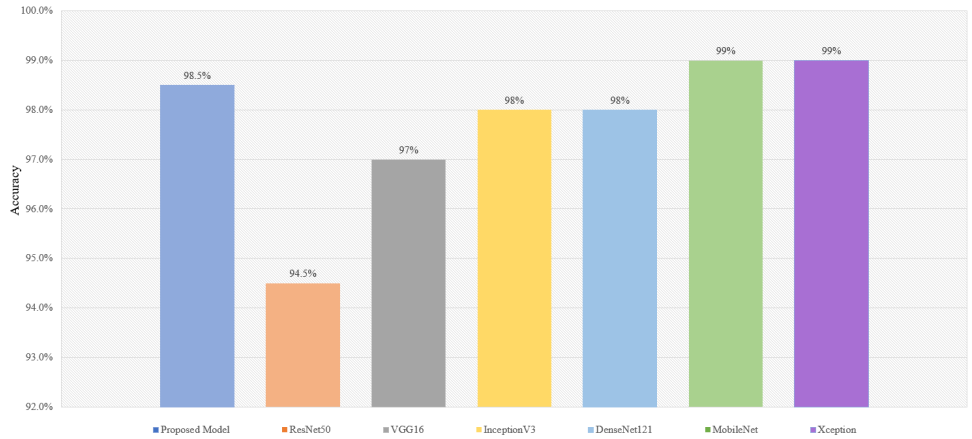
Figure 6.34: Accuracy Analysis of the Implemented Models.

As the table 6.1 shows, we compared the performance of the proposed model against the six renowned pre-trained models. The dataset used for each model is Br35H [62]. It is observed that our proposed model has achieved an accuracy of 98.5% which is comparable with other models. Our precision and recall percentage are also on par with other models. During the testing and tweaking phase, the proposed model peaked at 98.83% in some epochs. Thus, it is solidifying our proposed model as a standard model. To be noted, our constructed model took almost half the time to train and test compared to the other models. This proves that our intention to make a simplistic yet accurate model is paving the path for future improvements.

Chapter 7

Conclusion and Future Works

7.1 Conclusion

Among medical professionals, brain tumors are a constant topic of conversation. The sooner a patient is diagnosed, the better their prognosis. The Br35H dataset has been successfully used in this paper to build a CNN model to address this issue. By implementing this technique, we built computer vision without spending much time or money on computational resources. In order to make data training as simple as possible for systems with less than optimal processing capability, this is a smaller depth model with fewer parameters. We filtered the image dataset using the Gaussian filtering method to make it more accessible in our model, then trained our model on the dataset. To ensure the suggested model's integrity and accuracy, a total of six different CNN models, including VGG16, ResNet50, MobileNet, DenseNet121, InceptionV3 and Xception, were trained via transfer learning. After the training period was done, it was found that the proposed model had achieved an accuracy of around 98.71% accuracy, which was almost on par with the other pre-trained models.

Furthermore, with some extensive modification, the model reached over 98.5% accuracy in between 15 to 20 epochs. The model's run time was considerably faster than the other model, which was our initial goal. Thus, by altering some weight distribution and adding different layers, we hope to achieve an even more accurate and reliable model that can be widely used in different datasets, further broadening the scope of implementations of our proposed model.

7.2 Challenges

7.2.1 Computational Power

Due to the lack of power and being closer to the on-campus computer laboratory, the available computational power to process this data set with 3000 images was very challenging. Available graphics computational power was insufficient to run specific algorithms such as VGG19, ResNet101, and other models. While executing the VGG16 model, right after the training phase was done, the model crashed. So, the prediction run was not successful. Then we used Google's provided computational server using "Google Colab", and finally the expected results were retrieved.

7.2.2 Excessive Training Time

It is a consequential effect of the computational power issues. As the computational power available to us was not up to the mark, the run time for each model to train and test was very long. In some cases, each epoch took over an hour to complete. Due to this slow training time, the research progress for the team was hindered. We hope to resolve these two issues by running other CNN models for future comparisons and analysis.

7.3   Future Works

We are currently able to fabricate a CNN model by implementing convolutional layers, activation functions, max pooling, dense, and various other methods to make our model more accurate and ensure that the training time is very short. Our future goal is to add more layers with different parameters tweaked to increase the model's accuracy. Another future implementation of this model will be to test its accuracy by fitting it into a more challenging dataset and modifying it accordingly. Furthermore, this model is based on binary classification. Later, with this approach, we plan to use transfer learning to address similar complex problems, which will aid others in solving categorical classification difficulties in medical imaging datasets. This research hopes to develop a user application that will be accessible to the general public in the near future, allowing anyone to determine if MRI images show a brain tumor or not.

Bibliography

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[2] L. O. Chua and T. Roska, "Stability of a class of nonreciprocal cellular neural networks," *IEEE Transactions on Circuits and Systems*, vol. 37, no. 12, pp. 1520–1527, 1990.

[3] L. Chua and T. Roska, "Cellular neural networks with nonlinear and delay-type template elements," in *Proc. 1990 IEEE int. workshop on cellular neural networks and their applications*, 1990, pp. 12–25.

[4] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, "Principles of neural science (3d edition)," *Appleton & Lange Norwalk, CT*, 1991.

[5] L. O. Chua and T. Roska, "The cnn paradigm," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 3, pp. 147–156, 1993.

[6] W. Heiligenberg and T. Roska, "On biological sensory information processing principles relevant to cellular neural networks," *Cellular neural networks*, pp. 201–210, 1993.

[7] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural computation*, vol. 7, no. 2, pp. 219–269, 1995.

[8] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, "The" wake-sleep" algorithm for unsupervised neural networks," *Science*, vol. 268, no. 5214, pp. 1158–1161, 1995.

[9] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE transactions on neural networks*, vol. 9, no. 1, pp. 224–229, 1998.

[10] Y. Le, L. Bottou, G. Orr, and K. Muller, "Lecun y. efficient backprop in neural networks: Tricks of the trade," 1998.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] B. Schölkopf, A. J. Smola, F. Bach, *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[13] P. Y. Simard, D. Steinkraus, J. C. Platt, *et al.*, "Best practices for convolutional neural networks applied to visual document analysis.," in *Icdar*, vol. 3, 2003.

[14]   S.-C. Wang, "Artificial neural network," in *Interdisciplinary computing in java programming*, Springer, 2003, pp. 81–100.

[15]   L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*, IEEE, 2004, pp. 178–178.

[16]   Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, IEEE, vol. 2, 2004, pp. II–104.

[17]   J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[18]   V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[19]   D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, Springer, 2010, pp. 92–101.

[20]   A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[21]   D. Costarelli and R. Spigler, "Multivariate neural network operators with sigmoidal activation functions," *Neural Networks*, vol. 48, pp. 72–77, 2013.

[22]   L. Bottou, "From machine learning to machine reasoning," *Machine learning*, vol. 94, no. 2, pp. 133–149, 2014.

[23]   J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," *arXiv preprint arXiv:1412.5474*, 2014.

[24]   J. Jin, K. Fu, and C. Zhang, "Traffic sign recognition with hinge loss trained convolutional neural networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 1991–2000, 2014.

[25]   D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26]   D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Advances in neural information processing systems*, 2014, pp. 3581–3589.

[27]   A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[28]   K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[29]   N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[32] C. Szegedy, W. Liu, Y. Jia, *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning.* MIT press, 2016.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[35] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Convolutional neural networks for large-scale remote-sensing image classification," *IEEE Transactions on geoscience and remote sensing*, vol. 55, no. 2, pp. 645–657, 2016.

[36] P. Moeskops, M. A. Viergever, A. M. Mendrik, L. S. De Vries, M. J. Benders, and I. Išgum, "Automatic segmentation of mr brain images with a convolutional neural network," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1252–1261, 2016.

[37] E. Walach and L. Wolf, "Learning to count with cnn boosting," in *European conference on computer vision*, Springer, 2016, pp. 660–676.

[38] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1251–1258.

[39] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[40] K. J. Johnson, J. Schwartzbaum, C. Kruchko, *et al.*, "Brain tumor epidemiology in the era of precision medicine: The 2017 brain tumor epidemiology consortium meeting report," *Clinical neuropathology*, vol. 36, no. 6, p. 255, 2017.

[41] E. Lima, X. Sun, J. Dong, H. Wang, Y. Yang, and L. Liu, "Learning and transferring convolutional neural network knowledge to ocean front recognition," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 3, pp. 354–358, 2017.

[42] H. Phan, L. Hertel, M. Maass, P. Koch, R. Mazur, and A. Mertins, "Improved audio scene classification based on label-tree embeddings and convolutional neural networks," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1278–1290, 2017.

[43] E. Hoseinzade and S. Haratizadeh, "Cnnpred: Cnn-based stock market prediction using several data sources," *arXiv preprint arXiv:1810.08923*, 2018.

[44] S. Indolia, A. K. Goswami, S. P. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network-a deep learning approach," *Procedia computer science*, vol. 132, pp. 679–688, 2018.

[45] Y. Li, H. Huang, Q. Xie, L. Yao, and Q. Chen, "Research on a surface defect detection algorithm based on mobilenet-ssd," *Applied Sciences*, vol. 8, no. 9, p. 1678, 2018.

[46] H. Mohsen, E.-S. A. El-Dahshan, E.-S. M. El-Horbaty, and A.-B. M. Salem, "Classification using deep learning neural networks for brain tumors," *Future Computing and Informatics Journal*, vol. 3, no. 1, pp. 68–71, 2018, ISSN: 2314-7288. DOI: https://doi.org/10.1016/j.fcij.2017.12.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2314728817300636.

[47] J. Seetha and S. S. Raja, "Brain tumor classification using convolutional neural networks," *Biomedical & Pharmacology Journal*, vol. 11, no. 3, p. 1457, 2018.

[48] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.

[49] M. T. Abed, S. A. Nabil, U. Fatema, *et al.*, "Early prediction of alzheimer's disease using convolutional neural network," Ph.D. dissertation, Brac University, 2019.

[50] A. B. Amir, U. H. Nisa, A. A. Shafi, M. Reza, *et al.*, "Traffic sign recognition using deep learning," Ph.D. dissertation, Brac University, 2019.

[51] S. Das, O. F. M. R. R. Aranya, and N. N. Labiba, "Brain tumor classification using convolutional neural network," in *2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, 2019, pp. 1–5. DOI: 10.1109/ICASERT.2019.8934603.

[52] S. Deepak and P. Ameer, "Brain tumor classification using deep cnn features via transfer learning," *Computers in biology and medicine*, vol. 111, p. 103 345, 2019.

[53] J. J. Graber, C. S. Cobbs, and J. J. Olson, "Congress of neurological surgeons systematic review and evidence-based guidelines on the use of stereotactic radiosurgery in the treatment of adults with metastatic brain tumors," *Neurosurgery*, vol. 84, no. 3, E168–E170, 2019.

[54] A. Khanna, D. Gupta, S. Bhattacharyya, V. Snasel, J. Platos, and A. E. Hassanien, "International conference on innovative computing and communications," *Proceedings of ICICC*, vol. 2, 2019.

[55] Y. Liu, J. Zhang, C. Gao, J. Qu, and L. Ji, "Natural-logarithm-rectified activation function in convolutional neural networks," in *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*, IEEE, 2019, pp. 2000–2008.

[56] X. Ou, P. Yan, Y. Zhang, *et al.*, "Moving object detection method via resnet-18 with encoder–decoder structure in complex scenes," *IEEE Access*, vol. 7, pp. 108 152–108 160, 2019.

[57] A. P. Patel, J. L. Fisher, E. Nichols, *et al.*, "Global, regional, and national burden of brain and other cns cancer, 1990–2016: A systematic analysis for the global burden of disease study 2016," *The Lancet Neurology*, vol. 18, no. 4, pp. 376–393, 2019.

[58] H. H. Sultan, N. M. Salem, and W. Al-Atabany, "Multi-classification of brain tumor images using deep neural network," *IEEE Access*, vol. 7, pp. 69 215–69 225, 2019. DOI: 10.1109/ACCESS.2019.2919122.

[59] H. Gholamalinezhad and H. Khosravi, "Pooling methods in deep neural networks, a review," *arXiv preprint arXiv:2009.07485*, 2020.

[60] D. Theckedath and R. Sedamkar, "Detecting affect states using vgg16, resnet50 and se-resnet50 networks," *SN Computer Science*, vol. 1, no. 2, pp. 1–7, 2020.

[61] A. Hamada, *Br35h :: Brain tumor detection 2020*, Nov. 2021. [Online]. Available: https://www.kaggle.com/ahmedhamada0/brain-tumor-detection.

[62] A. Naseer, T. Yasir, A. Azhar, T. Shakeel, and K. Zafar, "Computer-aided brain tumor diagnosis: Performance evaluation of deep learner cnn using augmented brain mri," *International Journal of Biomedical Imaging*, vol. 2021, 2021.

[63] A. Rehman, M. A. Khan, T. Saba, Z. Mehmood, U. Tariq, and N. Ayesha, "Microscopic brain tumor detection and classification using 3d cnn and feature selection architecture," *Microscopy Research and Technique*, vol. 84, no. 1, pp. 133–149, 2021.

[64] O. Sevli, "Performance comparison of different pre-trained deep learning models in classifying brain mri images," *Acta Infologica*, vol. 5, no. 1, pp. 141–154, 2021.

[65] [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-1b-relu-layer/.

[66] *Cnn image classification | image classification using cnn*, https://tinyurl.com/27x8rxwd.