Order N°……/Faculty/UMBB/2023

People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific

University M'Hamed BOUGARA – Boumerdes

Faculty of Hydrocarbons and Chemistry

Final Year Thesis Presented in Partial Fulfillment of the Requirements for theDegree of:

MASTER

Automation of Industrial Processes

Option: **Instrumentation of Petroleum Installations**

Presented by:

Mohammed Nadjib Allah SALHI

Title

Detection and Localization of Brain Tumor by Deep Learning Models

Jury Members:

Mr. BENDJEGHABA Omar	PROF	FHC	President
Mr. KHEBLI Abdelmalek	MCA	FHC	Examiner
Mr. YOUSSEF Tewfik	MCB	FHC	Examiner
Ms. LACHEKHAB Fadhila	MCB	FHC	Supervisor

2023-2024

People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Faculty of Hydrocarbons and Chemistry

Department: Automation of Industrial Processes

Option: Instrumentation of Petroleum Installations

Final Year Thesis Presented in Partial Fulfillment of the Requirements for the Degree of:

MASTER

Title

Detection and Localization of Brain Tumor by Deep Learning Models

Presented by: Mohammed Nadjib Allah SALHI

Favorable assessment of the Supervisor: F. LACHEKHAB signature

Favorable opinion of the President of the Jury:

Name **signature**

2023-2024

ACKNOWLEDGMENTS

First and foremost, I thank Allah for letting me live to see my project through and finish this work.

I would like to express my deepest gratitude to my supervisor **Dr. Fadhila LACHEKHAB (Kahoul)** for his patience and support during this long journey and learning process, I am very thankful for all of his information, remarks, and encouraging words. I had the honor to share my project with her.

Also, I would like to express my great and special gratitude to the **Jury Members.**

Last but not least, I would like to thank all people who helped me during this journey. Their names are too numerous to be mentioned, though, I wish them the best of luck.

“**Med Nadjib**”

Table of content

List of Figures

List of Tables

List of abbreviation

AI:	Artificial Intelligence
ML:	Machine Learning
DL:	Deep Learning
ANN:	Artificial Neural Network
MLP:	Multi Layer Perceprton
MRI:	Magnetic resonance imaging
CNN:	Convolution Neural Network

ABSTRACT

Healthcare MRI for brain tumor is a critical aspect of modern medicine, particularly in diagnosing and treating neurological disorders. Brain tumors pose significant health risks, and early detection is key to successful treatment outcomes. Traditional diagnostic methods often involve manual interpretation of MRI images by skilled radiologists, which can be time-consuming and subject to human error.

Recent advancements in medical imaging and AI have paved the way for more efficient and accurate diagnosis of brain tumors using Deep Learning algorithms. This study proposes a Deep Learning-powered MRI-based system for automated detection and localization of brain tumors.

Utilize Convolutional Neural Networks (CNNs) to analyze MRI scans and classify them into two classes: "Tumor" and "No tumor." To train and evaluate the four models, a dataset comprising of MRI images with corresponding labels indicating the presence or absence of tumors is utilized and then localization of a tumor if it exists.

Evaluation metrics such as accuracy, F1-score, Precision and Confusion Matrix are employed to assess the performance of the models in distinguishing between tumor and non-tumor cases. The results demonstrate the efficacy of the proposed approach in accurately identifying brain tumors from MRI scans.

Keywords: Brain tumor, Deep Learning, MRI , Convolutional Neural Network, Localization , Detection, VGG19 , Xception , Resnet50

General introduction

General Introduction

The detection and localization of brain tumors are pivotal tasks in medical image analysis, crucial for timely diagnosis and treatment planning. Traditional methods often rely on manual interpretation by medical experts, which can be time-consuming and prone to human error. However, recent advancements in deep learning offer promising avenues for automating these processes with high accuracy and efficiency.

In this study, we present a comprehensive approach to the detection and localization of brain tumors using deep learning models. Specifically, we leverage the capabilities of well-established architectures such as VGG19, ResNet50, Xception, and EfficientB0, which have demonstrated remarkable performance across various computer vision tasks. By harnessing the power of these models, we aim to enhance the precision and reliability of brain tumor detection from medical imaging data.

Moreover, we integrate a specialized architecture, ResUNet, tailored for semantic segmentation tasks, to precisely localize detected tumors within brain images. This hybrid approach combines the strengths of both classification and segmentation models, enabling not only the identification but also the precise delineation of tumor boundaries. Such accurate localization is essential for guiding subsequent medical interventions, including surgery and radiation therapy.

Through this thesis project, we seek to contribute to the advancement of automated brain tumor diagnosis, offering a scalable and efficient solution that holds great potential for improving patient care and outcomes. Our methodology not only demonstrates the effectiveness of deep learning in medical image analysis but also underscores the importance of interdisciplinary collaboration between computer science and healthcare domains for addressing complex healthcare challenges.

Chapter I

Introduction to Artificial Intelligence

Chapter I
Introduction to Artificial Intelligence

I.1 Introduction

First of all, the cutting edge of technology is artificial intelligence (AI), which has the potential to drastically change the way we interact with the outside world. Artificial intelligence (AI) has the unquestionable potential to transform a wide range of industries, from improving decision-making skills to optimizing industrial processes [1]. Machine Learning (ML), a branch of AI that focuses on creating algorithms that can learn from and improve upon data, is one of the most dynamic and promising areas in the field. In-depth examinations of supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning are among the key areas and subfields of artificial intelligence and machine learning (AI and ML) covered in this work. Additionally, it covers the fundamental ideas of Artificial Neural Networks (ANNs) and the groundbreaking effects of Deep Learning (DL), especially with regard to architectures like Convolutional Neural Networks (CNNs) and Recurrent [1].

I.2 Artificial Intelligence

Artificial Intelligence (AI) is a rapidly evolving field that has the potential to revolutionize the way we live and work. In recent years, it has been applied to a wide range of industries and has shown promising results in improving efficiency, accuracy, and decision-making. According to a recent report by McKinsey, AI could contribute up to $13 trillion to the global economy by 2030 [2].

At its core, AI is a discipline of study that focuses on creating intelligent machines that can perform tasks that typically require human intelligence, such as learning, problem-solving, decision-making, and perception. The goal of AI is to develop systems that can operate autonomously, adapt to new situations, and interact with humans and the environment in a natural and seamless way [3].

From an industrial perspective, AI can be defined as the brain that allows a system to detect its environment, interpret the data it collects, solve complicated issues, and learn from experience [4].

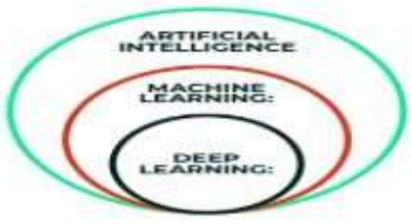


Figure I.1: Representation of Artificial Intelligence and its subfields [5]

I.2.1 Machine Learning

One of the key drivers of Artificial Intelligence is machine learning (ML), a subset of AI that focuses on developing algorithms that can learn from data and improve their performance over time.

Machine learning algorithms organize the data, learn from it, gather insights, and make predictions based on the information it analyzed without the need for additional explicit programming. Training a model with data and after that using the model to predict any new data is the concern of Machine Learning [5].

One of the significant advantages of machine learning is its ability to handle complex and large-scale datasets. By processing vast amounts of data, machine learning algorithms can uncover intricate patterns and relationships that may not be apparent to humans. This enables applications in various domains, such as image and speech recognition, natural language processing, recommendation systems, fraud detection, and autonomous vehicles.

ML is a powerful field of study that enables computers to learn from data, discover patterns, and make predictions or decisions. Through the use of sophisticated algorithms and statistical techniques, machine learning has the potential to transform industries and solve complex problems. As technology continues to advance, machine learning will play a crucial role in shaping the future of artificial intelligence and driving innovation in various domains.

Machine learning techniques can be subdivided into supervised, unsupervised, semi-supervised, and reinforcement learning [6, 7].

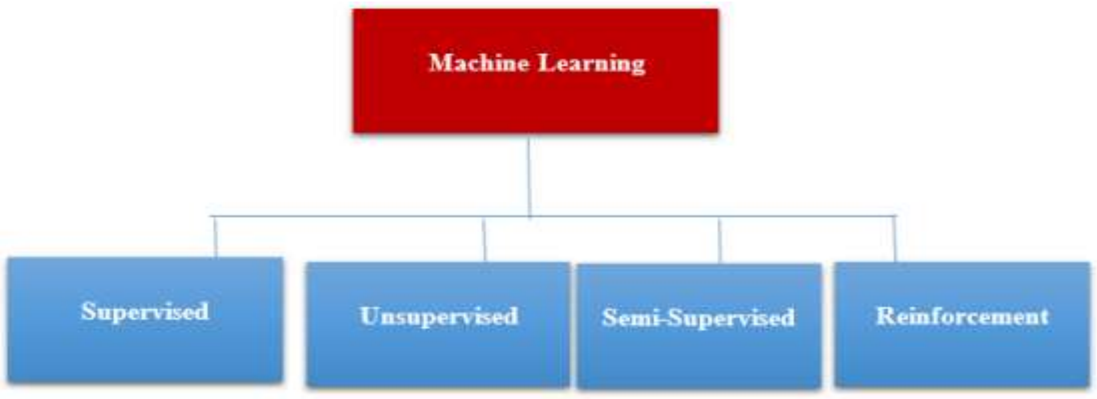


Figure I.2: Components of Machine Learning [5]

I.2.2 Supervised Learning

Supervised learning is a machine learning technique where the algorithm learns from labeled data, with each data point having corresponding input features and known output labels. The goal of supervised learning is to build a predictive model that can accurately map input data to the correct output labels based on the provided training examples [47].

The process of supervised learning begins with the collection of a labeled dataset, where each data instance is associated with a known output value. This dataset is then divided into two parts: the training set and the test set. The training set is used to train the model by presenting it with input features and their corresponding labels. The model learns from the training data by adjusting its internal parameters or weights based on the observed input-output relationships. The objective is to minimize the difference between the predicted output and the actual label for each training example.

Once the model is trained, it is evaluated using the test set, which consists of unseen data with known labels. The model's performance is assessed by comparing its predicted outputs with the true labels.

The main advantage of supervised learning is its ability to make accurate predictions or classifications based on labeled data. It is widely used in various applications, including spam detection, sentiment analysis, image recognition, speech recognition, and medical diagnosis. Supervised learning models can also be extended to handle multiclass classification problems and support probabilistic predictions, providing valuable insights for decision-making.

However, supervised learning also has limitations. It heavily relies on the availability of labeled data, which can be expensive and time-consuming to obtain. An insufficient or biased dataset may lead to inaccurate models and poor generalization of unseen data. Additionally, supervised learning models may struggle when faced with data that falls outside the range of the training examples, making them sensitive to outliers and noise [47].

In summary, supervised learning is a powerful machine learning approach that leverages labeled data to build predictive models. It enables accurate predictions or classifications by learning from observed input-output relationships. While it has its limitations, supervised learning has proven to be valuable in solving a wide range of real-world problems and continues to be a fundamental technique in the field of machine learning.

I.2.3 Unsupervised Learning

In unsupervised learning, the data is not labelled, which means that the ML model aims to discover unknown patterns in the data, by searching for similarities between the data points for example. Algorithms are therefore formulated such that they can find patterns and structures in the data on their own [48].

The process of unsupervised learning begins with collecting a dataset consisting of input features without corresponding output labels. The goal is to find meaningful representations or groupings within the data. Clustering is one common technique in unsupervised learning, where similar data points are grouped based on their inherent similarities. Clustering algorithms, such as k-means, hierarchical clustering, and DBSCAN, are used to identify clusters and partition the data accordingly.

Another key approach in unsupervised learning is dimensionality reduction, which aims to reduce the number of input features while preserving important information. This helps in visualizing high-dimensional data and extracting relevant features. Autoencoders are commonly used methods for dimensionality reduction.

Unsupervised learning algorithms can also be used for anomaly detection, where the goal is to identify unusual or abnormal data points that deviate significantly from the norm. By learning the regular patterns in the data, unsupervised algorithms can detect outliers or anomalies that may indicate potential fraud, errors, or unusual behavior.

Evaluation in unsupervised learning is more challenging than in supervised learning since there are no predefined output labels to compare against. Instead, the quality of unsupervised learning algorithms is assessed based on the coherence and meaningfulness of the discovered patterns, the compactness of clusters, or the ability to separate anomalies from normal data.

However, unsupervised learning has its challenges. Since there are no ground truth labels, evaluating the performance of unsupervised algorithms can be subjective and depend on domain knowledge. The algorithms heavily rely on the quality and representativeness of the data, making it crucial to preprocess and clean the data appropriately. Additionally, unsupervised learning algorithms can be computationally expensive, especially when dealing with large-scale datasets or complex structures.

In summary, unsupervised learning is a valuable approach in machine learning that allows for exploring and extracting patterns from unlabeled data. By uncovering hidden structures and relationships, unsupervised learning algorithms provide insights, aid in data exploration, and serve as a foundation for various downstream tasks.

I.2.4 Semi-Supervised Learning

Semi-supervised learning is a branch of machine learning that combines elements of both supervised and unsupervised learning. It deals with datasets that contain a small portion of labeled data and a larger portion of unlabeled data. The goal of semi-supervised learning is to leverage the limited labeled data together with the unlabeled data to improve the model's performance and generalization [49].

The process of semi-supervised learning begins by partitioning the available data into labeled and unlabeled subsets. The labeled data consists of input features along with their corresponding output labels. The unlabeled data, on the other hand, contains input features without any associated labels. The labeled data is used to train a model using supervised learning techniques, while the unlabeled data is leveraged to enhance the model's performance. Semi-supervised learning algorithms often incorporate unsupervised learning methods to exploit the unlabeled data. By leveraging the inherent structure and patterns within the unlabeled data, the algorithms aim to improve the model's ability to generalize to unseen data.

Unsupervised learning techniques such as clustering, dimensionality reduction, or generative models can be used to extract additional information from the unlabeled data.

One common approach in semi-supervised learning is to use the unlabeled data to create a smoother decision boundary or to estimate the underlying data distribution. By considering the relationships and similarities among the unlabeled data points, the model can make more informed predictions for new, unseen instances.

Semi-supervised learning is particularly useful in scenarios where obtaining labeled data is costly, time-consuming, or difficult. By making effective use of a small labeled dataset in conjunction with a larger unlabeled dataset, semi-supervised learning can achieve comparable or even superior performance to supervised learning approaches that rely solely on labeled data. In conclusion, semi-supervised learning is a powerful approach that combines elements of supervised and unsupervised learning to leverage both labeled and unlabeled data. By effectively utilizing the unlabeled data, semi-supervised learning algorithms can improve the model's performance and generalization, particularly in scenarios where obtaining labeled data is limited or expensive. Despite its challenges, semi-supervised learning continues to be an active area of research, driving advancements in machine learning and expanding the range of problems that can be addressed.

I.2.5 Reinforcement Learning

In a reinforcement learning (RL) system, instead of providing input and output pairs, we describe the current state of the system, specify a goal, provide a list of allowable actions and their environmental constraints for their outcomes, and let the ML model experience the process of achieving the goal by itself using the principle of trial and error to maximize a reward [8].

An agent interacts with an environment sequentially. At each step, the agent observes the current state of the environment and takes action. The environment responds by transitioning to a new state and providing feedback in the form of a reward signal, which indicates the desirability of the agent's action. The goal of the agent is to learn a policy (a mapping from states to actions) that maximizes the expected cumulative reward over time [41].

One key aspect of reinforcement learning is the trade-off between exploration and exploitation. Initially, the agent explores different actions and learns about the environment. As it gathers more knowledge, it shifts towards exploiting its current knowledge to maximize rewards.

Reinforcement learning has been successfully applied to various domains, such as robotics, game playing, autonomous vehicles, recommendation systems, and resource management.

RL encompasses a wide range of algorithms that can be used depending on the problem at hand, the most common ones are Q-Learning and Actor-Critic Learning (ACL).

- **Q-Learning** is a model-free algorithm used in reinforcement learning to learn the optimal action-value function, often referred to as the Q-function. The Q-function represents the expected cumulative reward for taking a particular action in a given state and following a specific policy. The Q-Learning algorithm iteratively updates the Q-values based on the observed rewards and the estimated future rewards. It uses a technique called Temporal

Difference learning, which calculates the difference between the estimated Q-value and the observed reward to update the Q-value. By repeatedly interacting with the environment and updating the Q-values, the agent can learn the optimal policy that maximizes the cumulative reward [45].

- **Actor-Critic Learning (ACL)** is an approach of RL that combines elements of both policy-based and value-based methods. It utilizes two components: an actor and a critic. The actor is responsible for learning and selecting actions based on the current policy. It explores the environment, take actions, and gathers experiences. On the other hand, the critic, evaluates the actions taken by the actor and provides feedback in the form of a value function or Q-values. The actor-critic architecture allows for continuous learning and policy improvement. The actor uses the feedback from the critic to update its policy, while the critic uses the observed rewards to update its value estimates. This way, the actor-critic algorithm can learn both the best actions to take and the value of those actions [46].

In summary, Q-Learning is a value-based algorithm that learns the optimal action-value function, while Actor-Critic Learning combines policy-based and value-based methods to learn both the policy and the value function simultaneously. Both approaches have been widely used in reinforcement learning and have contributed to many successful applications.

I.2.6 Artificial Neural Network

Artificial Neural Network (ANN) is a type of ML inspired by the principle of information processing in biological systems, ANNs consist of mathematical representations of connected processing units called artificial neurons [8].

Like synapses in a brain, each connection between neurons transmits signals whose strength can be amplified or attenuated by a weight that is continuously adjusted during the learning process. Signals are only processed by subsequent neurons if a certain threshold is exceeded as determined by an activation function.

Typically, neurons are organized into networks with different layers. An input layer usually receives the data input and an output layer produces the ultimate result. In between, there are zero or more hidden layers that are responsible for learning a non-linear mapping between input and output.

“Feed-forward” is the first, most common and simplest architecture. It is formed by stacked neurons creating layers, where all the neurons of a layer are connected to all the neurons of the next layer by feeding their output to others' input. However, there are no connections to neurons of previous layers or among neurons of the same layer [11].

Artificial neural networks are of particular interest since their flexible structure allows them to be modified for a wide variety of contexts across all types of ML, therefore, ANNs can be referred to as "Shallow" or "Deep" depending on the number of hidden layers it contains

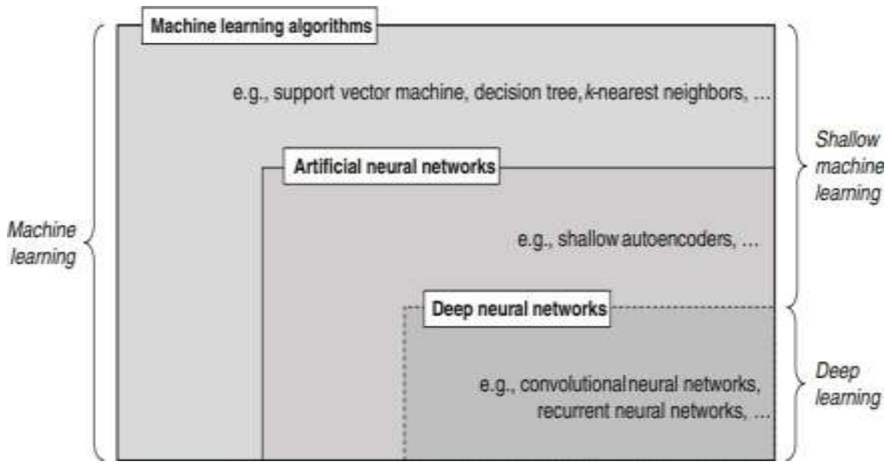

Machine learning algorithms
e.g., support vector machine, decision tree, k-nearest neighbors, ...
Artificial neural networks
e.g., shallow autoencoders, ...
Deep neural networks
e.g., convolutional neural networks, recurrent neural networks, ...
Machine learning
Shallow machine learning
Deep learning

Figure I.3: Diagram of ML classes [8]

I.3 Deep Learning

Deep learning is a subset of machine learning that focuses on the development and application of artificial neural networks with multiple layers, known as deep neural networks. It aims to enable computers to learn and make predictions or decisions by mimicking the structure and function of the human brain. Deep learning has gained significant attention and popularity due to its remarkable ability to automatically learn hierarchical representations from raw data, leading to a state-of-the-art performance in various domains [50].

At the core of deep learning are artificial neural networks, which consist of interconnected nodes, called neurons, organized in layers. The neurons receive input signals, apply mathematical transformations, and produce output signals that are passed on to the next layer. The layers are stacked hierarchically, with each layer learning increasingly complex features or representations of the input data.

One of the key advantages of deep learning is its ability to handle and extract meaningful features from large-scale datasets. Deep neural networks can learn intricate representations of images, text, audio, and other forms of data, leading to breakthroughs in computer vision, natural language processing, predictive maintenance, and many other fields. Convolutional neural networks (CNNs) are widely used in image-related tasks, while recurrent neural networks (RNNs) are commonly employed for sequential and language-based data.

Deep learning has also benefited from advancements in hardware and computational resources, as training deep neural networks often requires significant computational power. Graphics processing units (GPUs) and specialized hardware accelerators, such as tensor processing units (TPUs), have enabled faster training and inference of deep learning models.

In conclusion, deep learning is a powerful branch of machine learning that uses deep neural networks to learn hierarchical representations from data. Its ability to automatically learn features from raw data has revolutionized numerous fields and led to breakthroughs in various applications. As hardware and algorithms continue to advance, DL is poised to drive further innovation and impact a wide range of industries, shaping the future of artificial intelligence.

DL involves training artificial neural networks in order to detect patterns in large unstructured data sets. These ANNs, containing several hidden layers and performing complex tasks with minimal human interference, are mostly called Deep Neural Networks (DNNs).

I.3.1 Deep Neural Networks

A Deep Neural Network (DNN) is simply an artificial neural network containing a large number of hidden layers, which explains the term “deep”.

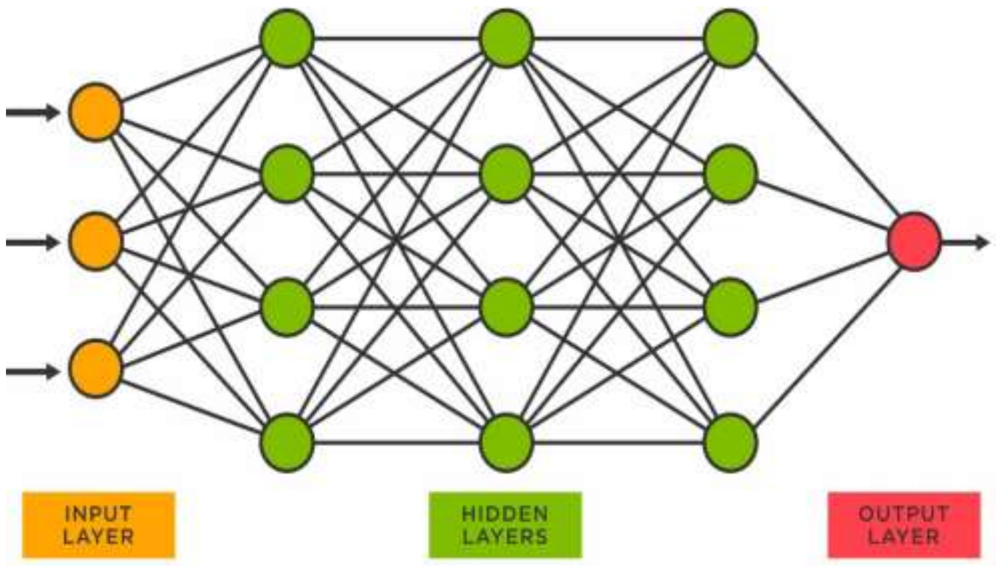


Figure I.4: Structure of a deep neural network

Deep neural networks typically consist of more than one hidden layer, organized in deeply nested network architectures. Furthermore, they usually contain advanced neurons in contrast to simple ANNs.

Therefore, DNNs may use multiple advanced operations in one neuron rather than using a simple activation function. These characteristics allow deep neural networks to be fed with raw input data and automatically discover a representation or output that is needed for the corresponding learning task. This is the networks' core capability, which is commonly known as deep learning [8].

While there are numerous types of DNNs, the most widely known are Convolutional Neural Networks and Recurrent Neural Networks.

I.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural networks commonly used for image and video processing tasks, such as image classification, object detection, and image segmentation. CNNs consist of multiple layers of convolutional and pooling operations that learn to extract features from images. In other words, CNNs have the human-like ability to recognize and classify objects based on their appearance [9].

One of the significant advantages of CNNs is their ability to automatically learn hierarchical representations from raw image data. By employing multiple layers, CNNs can capture increasingly complex and abstract features, enabling them to perform tasks such as image classification, object detection, and semantic segmentation. CNN architectures, such as AlexNet, VGGNet, ResNet, and InceptionNet, have achieved remarkable performance in various computer vision benchmarks and competitions.

Based on the dimension of the training data, CNNs can be devised into 1D CNNs and 2D CNNs. Deep 2D CNNs with many hidden layers and millions of parameters have the ability to learn complex objects and patterns providing that they can be trained on a massive size visual database with ground-truth labels. With proper training, this unique ability makes them the primary tool for various engineering applications for 2D signals such as images and video frames.

This may not be a viable option in numerous applications over 1D signals especially when the training data is scarce or application specific. To address this issue, 1D CNNs have recently been proposed and immediately achieved state-of-the-art performance levels in several applications such as anomaly detection and identification in power electronics and electrical motor fault detection. Another major advantage is that a real-time and low-cost hardware implementation is feasible due to the simple and compact configuration of 1D CNNs that perform only 1D convolutions (scalar multiplications and additions) [10].

In summary, CNNs are a key architecture in deep learning, particularly for computer vision tasks. Their ability to automatically learn and extract features from images has led to significant advancements in various applications. By leveraging convolutional and pooling layers, CNNs can effectively capture spatial patterns and hierarchical representations, enabling them to achieve state-of-the-art performance in image recognition, object detection, and other computer vision tasks.

I.3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural networks that excel in processing sequential data, such as time series, text, and speech. RNNs contain feedback connections that allow information to flow in both directions, unlike feedforward neural networks which only flow in one direction. The feedback connections in RNNs allow the network to process sequential data and capture temporal dependencies, by passing information from one step to the next. At each step, the current input is combined with the previous hidden state to produce a new hidden state and output. This process is repeated for each step in the sequence, allowing the network to capture the context and dependencies of the input data [12].

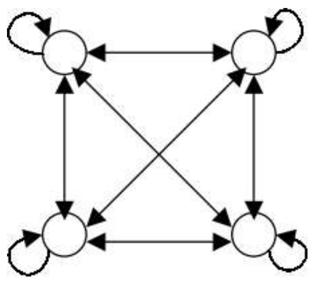


Figure I.5: An example of a fully connected RNN [13]

In the last years, several RNN architectures have been developed to meet the industries' standards, some of the most popular ones are the fully-connected RNN (FRNN), the long short- term memory (LSTM), and the gated recurrent unit (GRU).

- **FRNNs** connect the output of the previous time step with the additional input of the next time step, preserving important information about different time steps in the network [13].

- **LSTM** architecture has one cell state and three gates: an input gate, an output gate, and a forget gate. The cell state acts as a memory, while each gate functions like a conventional neuron, providing a weighted sum of its inputs. The forget gate decides what information to retain from previous steps. The input gate decides what information to add from the current step. The output gate decides what the next hidden state should be. Hence, only relevant information can pass through the hierarchy of the network. Thus, the LSTM has mechanisms to process both short-term and long-term memory components [14].
- **GRU** is similar to LSTM but has only two gates: an update gate and a reset gate. The update gate works similarly to the forget gate and the input gate of LSTM. It decides what information to throw away and what to add. The reset gate decides how much past information to forget. GRU has fewer parameters, uses less memory, and is faster to train than LSTM [14].

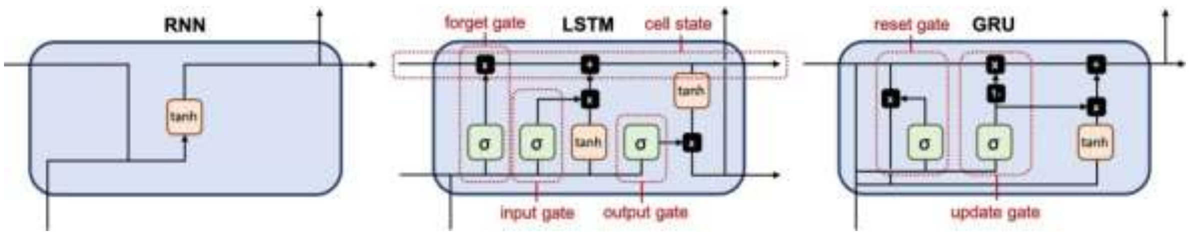


Figure I.6 : Diagram of different recurrent units [14]

In conclusion, RNNs are powerful neural network architectures designed for processing sequential data. Their ability to capture dependencies across time steps enables them to model complex temporal patterns and make predictions or generate outputs based on sequential inputs. With variants like LSTM and GRU, RNNs have achieved state-of-the-art results in various sequential tasks, making them a fundamental tool in fields such as natural language processing, speech recognition, and time series analysis.

I.4 Conclusion

Artificial Intelligence, particularly through its subset Machine Learning, has emerged as a beacon of innovation, offering solutions to complex problems across various domains. From supervised learning's ability to make accurate predictions based on labeled data to unsupervised learning's knack for discovering hidden patterns in unlabeled data, the spectrum of AI techniques presents a formidable arsenal for tackling real-world challenges. Moreover, the integration of Deep Learning, epitomized by architectures like CNNs and RNNs, has propelled AI into uncharted territories, enabling it to decipher intricate relationships in vast datasets and deliver unprecedented insights. As AI continues to evolve, fueled by advancements in hardware and algorithms, its impact is poised to expand, ushering in a new era of technological sophistication and societal transformation. In this landscape of innovation and possibility, the journey towards realizing the full potential of AI is ongoing, promising boundless opportunities for progress and discovery.

Chaptre II

Convolution Neural Networks

Chaptre II

Convolutional Neural Networks

II.1 Introduction

This chapter is dedicated to presenting activation functions and how train convolution neural networks (CNN)

II.2 Related Work

A. N. Sayed, Y. Himeur. (2023). Presented a discussion paper on how the integration of artificial intelligence (AI) and image analysis is transforming different realms within the healthcare sector. Deep learning empowers computers to analyze medical images, such as X-rays and MRI scans, providing automated detection of conditions like tumors and fractures [3], [4].

Yingying Fanga , X.Xing ,. (2023). Published a paper offering a comprehensive analysis of diverse AI techniques that have emerged for the swift diagnosis of this new disease during this public health emergency [5]. Magnetic Resonance Imaging (MRI) scans are frequently employed as a standard medical screening procedure for identifying primary brain tumors [6].

Zhang, C., & Lu, Y. (2021) presented paper highlights that Magnetic Resonance Imaging (MRI) is a widely employed diagnostic tool for the detection and diagnosis of brain tumors. Nonetheless, the precise interpretation of MRI images can pose a challenge for human experts due to the intricate and variable nature of both anatomy and tumor characteristics. In recent years, deep learning algorithms, particularly convolutional neural networks (CNNs), have demonstrated remarkable success in computer vision tasks, including medical image analysis. [7].

N. A. Samee et al (2022) However, the quick development of deep learning methods, particularly in the area of computer vision, has created new opportunities for the automatic and precise diagnosis of brain tumors [8]. Researchers have been investigating the possibility of these methods for identifying and categorizing brain tumor’s from magnetic resonance imaging (MRI) data by using the capabilities of deep learning [9].

J. Amin, M. A. Anjum. (2020) Researchers have published an article outlining their significant efforts in developing Convolutional Neural Networks (CNNs) capable of accurately identifying and classifying brain tumors, along with other types of medical imaging. This endeavor aims to improve medical diagnostic and treatment outcomes. Fueled by this potential, researchers are diligently working to construct advanced CNNs. [10].

E. Lynch et al. (2023). provided the benefit of deep learning is that it can learn intricate, hierarchical features directly from unprocessed data, eliminating the need for explicitly rule-based methods or hand-crafted features [11].

M. M. Taye, et al. (2023). Specifically, Convolutional Neural Networks (CNNs) are designed to capture spatial relationships and local patterns within images, rendering them well-suited for tasks related to medical image processing. Consequently, deep learning has emerged as a highly valuable technique for interpreting medical images. Its application enables more precise diagnosis of illnesses compared to conventional approaches, and it facilitates the detection of abnormalities in imaging data. Furthermore, it can be employed to automate medical diagnoses, thereby reducing the workload for medical experts. [12].

J. Ker, L. Wang, et al. (2018). article aiming at image classification and segmentation have shown rapid growth during the past two decades with the introduction of machine learning and computer vision techniques. Deep learning has found applications in medical imaging, such as identifying local anatomical characters, detecting organs and body parts, and identifying cells of different shapes and sizes.[13].

Multimodal Brain Tumor Segmentation (BRATS) Challengeis the main competition on brain tumor classification which is organized by the Perelman School of Medicine at the University of Pennsylvania, Centre for Biomedical Image Computing & Analytics (CBICA) from 2012 onwards. The BRATS challenge focuses on automating the brain tumor detection and the survival rate estimation techniques and algorithms. Each year the dataset is updated and the overall performance of the proposed algorithms have shown a tremendous improvement over time. On the whole, the accuracy of the algorithms proposed using BRATS dataset falls around 90% [14][15][16].

M. Soltaninejad et al. (2017). Some of these algorithms were developed using classical CNN architecture whereas some are developed using improved CNN algorithms like U-net[11],super pixel-based extremely randomized trees [17]. Another popular and publicly available brain tumor dataset is the Figshare MRI dataset [18] [19] which is the dataset employed in this paper. Due to the easy accessibility and the readily availability, Figshare MRI brain tumor dataset also has been used in many Brain tumor classification and segmentation related research [20][21][22][23]

However, A deep network was enhanced by employing cross channel normalization (CCN) and parametric rectified linear unit (PRELU) in [23] for brain tumor segmentation, In the previous work by the authors, a region proposal algorithm is proposed to address the problem of selecting a random number of objects in a single region [24][25].

In the proposed method, instead of searching the entire image for number of objects, the algorithm search for objects in several selective areas of the image, while treating each sub-region as an independent sub-image. In [24], a fully autonomous learning algorithm was constructed using Region-based Faster Convolutional Neural Network (Faster R-CNN) to localize the meningioma tumor regions in MRI. Once the tumor is segmented, Prewitt and Sobel edge detection algorithms are applied to the segmentation output, with the expectation of detecting the exact tumor boundary. Both of these techniques compute an approximate tumor boundary using the gradient intensity function of the image [26].

A study was done on a subset of the BRATS 2018 dataset that contained 1,992 Brain MRI scans. The YOLOv5 model achieved an accuracy of 85.95% and the Fast Ai classification model achieved an accuracy of 95.78%. These two models can be applied in real-time brain tumor detection for early diagnosis of brain cancer [27]

Another work was presented is prepared with the 29-layer YOLO Tiny and fine-tuned to work efficiently and perform task productively and accurately in most cases with solid execution. The outcome of the model is the highest like precision, recall and F1-score beating other previous results of earlier versions of YOLO and other studies like Fast R-CNN [28].

The darknet yolov4 is used, to perform the classification, and region of interest detection with the best accuracy scores. The model is trained with the Tesla GPU and obtained the results of the existing techniques in the field of fetal brain classification and localization. The accuracy of 97.92% and precision percentage of 96.70 is achieved in the research work [29].

Many research areas are being explored in medical image analysis. It includes medical imaging domains like identification, detection, and segmentation [30]–[31]

Recently, deep learning (DL) methods have frequently been employed for brain MRI categorization [32]. While feature mining and classification were integrated into self-learning, deep learning methods do not necessitate a manual process for feature extraction. The DL approach requires a dataset, and minimal pre-processing is required for selecting salient features in a self-learning way [33].

Recently, in many studies, CNNs have been widely employed to classify brain MRI and validated on a different dataset of brain tumors [34]– [35]. A deep CNN-based model was proposed in [36] for brain MRI images categorization into distinct classes. The authors used brain MRI images from a publicly available dataset to prevent model ambiguity. The suggested model has a classification accuracy of 91.4%. Deepak and Ameer [37] employed a pre-train deep CNN, GoogLeNet, to extract key attributes using brain MR images and classify tumors into three classes with 98% accuracy. Ahmet and Muhammad [38] categorized brain MR images using various CNN models and attained satisfactory accuracy. They modify a pre-trained ResNet-50 DCNN by excluding the final five layers and introducing additional eight layers. The model achieved the highest among all pre-trained model's accuracy of 97.2 %. Sultan et al. [39] suggested a CNNbased deep learning model utilizing two publicly accessible datasets have 3064 (glioma, meningioma, and pituitary tumors) and 516 (Grade II, Grade III, and Grade IV) brain medical scans. The proposed method has the best accuracy of 96.13 % and 98.7 %. Khwaldeh et al. [40] used several CNNs to classify brain MRI images and achieved good results. Using modified pre-trained Alexnet CNN, they achieved a higher accuracy of 97.2 %. Khan, M.A. et al. [41] developed a multi-model-based technique to differentiate brain tumors with DL. The presented system involves multiple stages, employing partial least squares (PLS) for feature concatenation and Extreme Learning Machine (ELM) for classification. Their methodology resulted in significant improvements of 97.8%, 96.9%, and 92.5% on BraTs-2015, BraTs-2017, and BraTs-2018 datasets, respectively.

In a different approach, Özyurt et al. [42] proposed a technique for detecting brain tumors. They initiated the process with MRI tumor image segmentation using the NS-EMFSE algorithm. Features were then extracted from the segmented image using AlexNet. Subsequently, using Support Vector Machine (SVM), they successfully detected and classified brain tumor images as benign or malignant, achieving an impressive accuracy of 95.62%.

A widely adopted solution involves integrating information obtained from multimodal paired MRI. This approach is grounded in the recognition that different pulse sequences or modalities of MRI offer complementary information about brain tumors from various perspectives [43]–[44]. By leveraging data from multiple modalities, researchers and practitioners aim to enhance the overall understanding and characterization of brain tumors, facilitating more comprehensive and accurate analyses.

In [45], a Convolutional Neural Network (CNN) was employed for tumor detection through MRI. The images underwent processing by the CNN, and a Softmax layer yielded an impressive accuracy result of 98.67%. Additionally, precision analysis of the CNN was conducted using the Radial Basis Function (RBF) classifier, achieving a high precision rate of 97.34%. The Decision Tree (DT) classifier was also utilized, providing a respectable accuracy of 94.24%. Beyond accuracy, the evaluation of the network's performance included criteria such as sensitivity, specificity, and precision.

In [46], MRIs were organized using CNNs in a public dataset to classify tumors as benign or malignant. This was done to extract characteristics with enhanced precision. The proposed hybrid model combined CNN and Support Vector Machine (SVM). The CNN-SVM hybrid, as well as the SVM alone for comparison, demonstrated notable efficiency. The hybrid model achieved an accuracy of 98.6702%, showcasing the effectiveness of the combined CNN-SVM approach in tumor classification.

In [47], MRI data was utilized to train a hybrid paradigm that combined Neural Autoregressive Distribution Estimation (NADE) and a Convolutional Neural Network (CNN). The model was subsequently tested with 3064 images representing three types of brain tumors. The results demonstrated that the NADE-CNN hybrid achieved a high level of classification performance, boasting an accuracy of 94.49%.

In [48], the authors proposed a CNN method with the addition of clustering to extract features, enabling classification into two categories: healthy patients and sick patients. Their dataset consisted of 1892 images, with 1666 used for training and 256 for testing. To reduce computational times, a preprocessing phase was conducted by resizing the images to 227×227 pixels. The authors compared their proposed method against other types of CNNs with different activation functions, all sharing the same structure. The CNN with the Softmax activation function achieved the highest accuracy at 98.67%. They then implemented their clustering method into the same network, further enhancing the network's performance to an impressive 99.12% accuracy.

In [49], the authors aimed to classify MRI images of individuals with brain tumors and those without this condition. They employed a Convolutional Neural Network (CNN) with a method of their own design. Additionally, they compared the accuracy of their proposed CNN method against Support Vector Machine (SVM) and Deep Neural Network (DNN). To enhance accuracy during training, a loss layer was added at the end of the training phase to provide feedback to the neural network. The achieved accuracies were 83% for SVM, 97% for DNN, and 97.5% for their proposed CNN method.

In [50], NMR images were classified using an ACNN (presumably an Advanced Convolutional Neural Network). The authors utilized 253 NMR images, transforming them into grayscale and resizing to 256×256 pixels. The dataset was augmented to 2912 images, with 50% representing healthy patients and 50% representing unhealthy patients. The ACNN, with Softmax as the activation function, achieved an impressive accuracy of 96.7%.

II.3 Activation Functions:

An activation function determines the output of each node given its input and is used to compute the output of a neural network in each layer. When a neuron receives n inputs (X1, X2,..., Xn), its output, or activation, is denoted by A, which has the following definition [51]:

$$A = G(W1 * X1 + W2 * X2 + W3 * X3 + \cdots Wn * Xn + B) \ldots\ldots\ldots\ldots\ldots \text{(II.1)}$$

Where W is a vector of real-valued weights, B is the bias, and G is the activation function.

II.3.1 Linear activation function

The input is multiplied by the weights assigned to each neuron by the linear activation function, which is a straight line that produces a signal proportionate to the input. This formula can be used to describe a linear activation function: A signal that is proportionate to the input is produced by the linear activation function, which is a straight line that multiplies the input by the weights of each neuron. The formula below describes a linear activation function:

$$y = mX \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(II.2)}$$

where Y is the output signal, X is the input, and m is the weight of each neuron. As learning is not aided by neurons doing linear regression or classification, linear activations are not used in the learning process of any model.

II.3.2 Non-linear activation functions:

Activation functions are typically utilized in neural networks, where their primary role is to introduce nonlinearity to the network. A few nonlinear activation functions that are frequently employed in neural networks are briefly described in the following sections:

- **ReLU (Rectified Linear Unit):** The rectified linear activation function is defined as the positive part of its variable; it is given by the equation:

$$F(x) = \max(x, 0) \quad \text{(II.3)}$$

The rectified linear unit (ReLU) activation function takes an input, denoted as x, to a neuron. It directly outputs the input value if it's positive; otherwise, it outputs zero. This function is commonly employed in various neural network architectures by default due to its facilitation of easy model training and improved performance. Figure II.1 depicts the graph of the ReLU activation function.

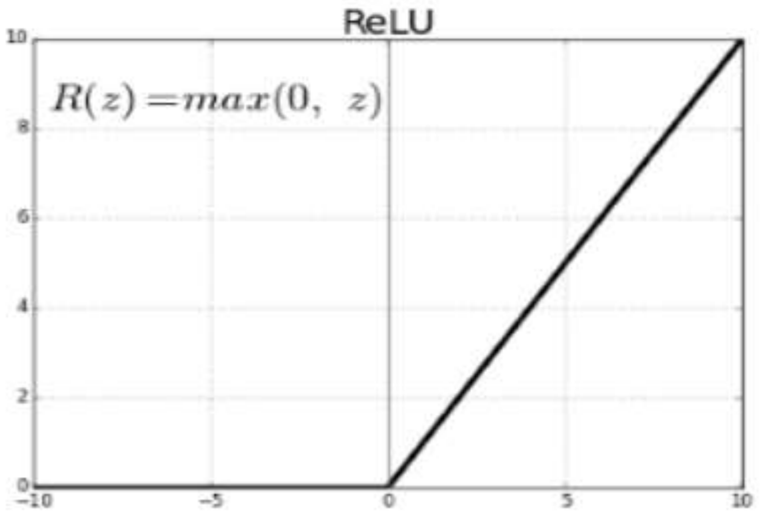


Figure II.1: ReLU activation function [52]

- **Sigmoid:** The sigmoid activation function, also called the logistic function, is a function with a characteristic “S”-shaped curve used in neural networks. The input to the sigmoid function is transformed into a value between 0.0 and 1.0 as shown in Fig II.2, and therefore, it is used for models that require a binary classification task. The equation for the sigmoid function is as follows:

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{(II.4)}$$

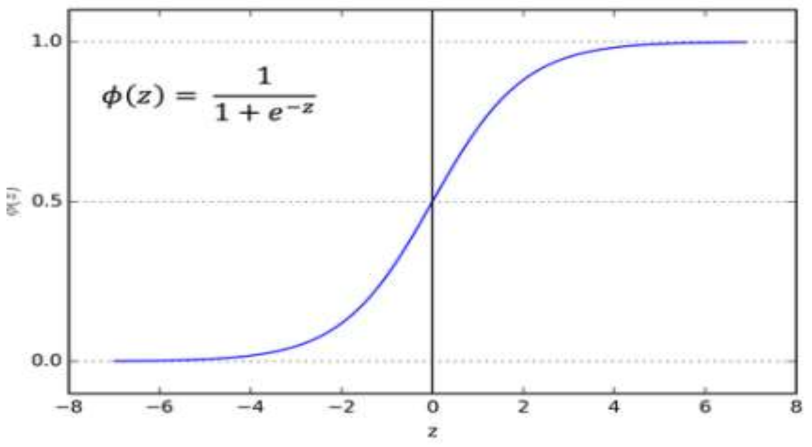


Figure II.2: Sigmoid activation function [53]

- **Softmax :** The softmax activation function serves as an extension of the logistic function (sigmoid) into multiple dimensions. It finds application in multi-class classification tasks and is typically employed as the final activation function in neural networks, typically at the last layer. Below is the formula for the softmax function:

$$f(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \quad \text{……………………………….…..(II.5)}$$

Where : *f* **:** Softmax Function

x **:** Input vector.

$e^{x}{}_{i}$ **:** Standard exponential function f or input vector.

K : Number of classes in the multi-class classifier

$e^{x}{}_{j}$: Standard exponential function f or output vector.

II.4 Convolutional Neural Network Structure:

A Convolutional Neural Network (CNN) comprises an input layer, hidden layers, and an output layer. CNNs process input images or feature vectors by passing them through hidden blocks using nonlinear activation functions. These hidden layers include convolutional layers, which perform convolutions. CNNs are constructed by stacking convolutional layers, pooling layers, and fully-connected (FC) layers together. The architecture of CNNs is depicted in Figure II.3, illustrating the arrangement of these layers.

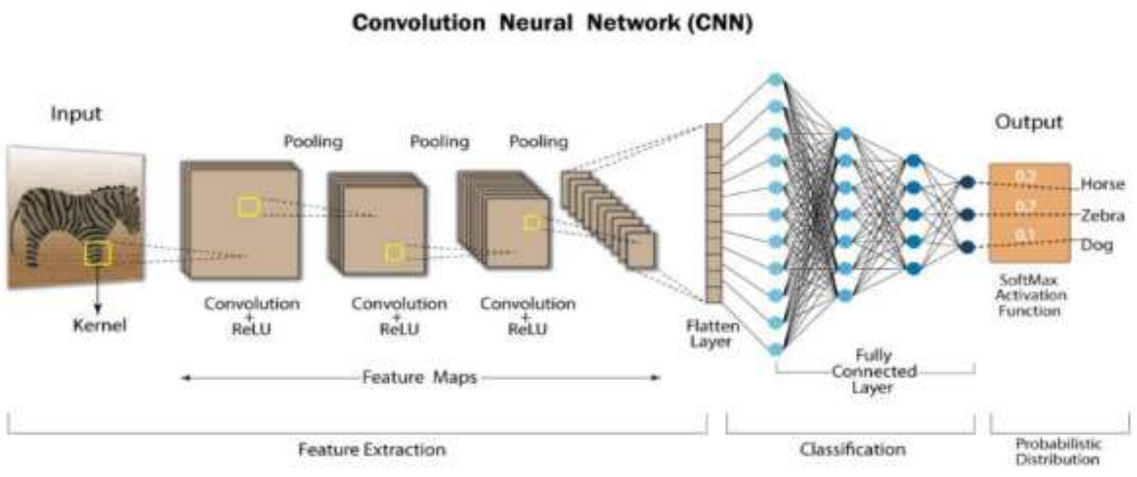


Figure II.3 : Convolutional Neural Network layers [54]

Before the advent of CNNs, experts in various fields manually performed feature extraction. In contrast, CNNs offer a significant advantage by automating the feature extraction process. In CNNs, convolutional and pooling layers serve as automatic feature extractors from the input image, while the fully connected layer acts as a classifier. This means that instead of relying on domain experts to handcraft features, CNNs can learn relevant features directly from the input data, streamlining the overall process and potentially improving performance.
The following are the typical building blocks of a Convolutional Neural Network:

- **Input layer:** The input layer serves as the entry point for the entire network, typically containing the pixel matrix of the input image. For instance, consider an RGB input image with dimensions of 64 pixels in height, 64 pixels in width, and a depth of 3 for the red, green, and blue color channels. In this case, the input layer would have dimensions of 64x64x3, representing the height, width, and depth of the input image, respectively.
- **Convolutional layer:** The convolutional layer is fundamental to a Convolutional Neural Network (CNN) as it handles the majority of computational tasks. This layer applies the convolution operation to the input data and passes the output to the subsequent layer. Illustrated in Figure II.4, convolution involves sliding a filter (also known as a kernel) across the input image, computing the dot product between the filter and the overlapped region of the input image. This process generates feature maps, where each pixel represents a single value obtained from convolving the filter with the input data.

  These feature maps capture different levels of abstraction: lower-level convolutional layers extract basic features such as edges and lines, while higher-level layers learn more complex and abstract features. This hierarchical representation allows CNNs to automatically learn and extract meaningful features from the input data, facilitating tasks such as image recognition and classification.

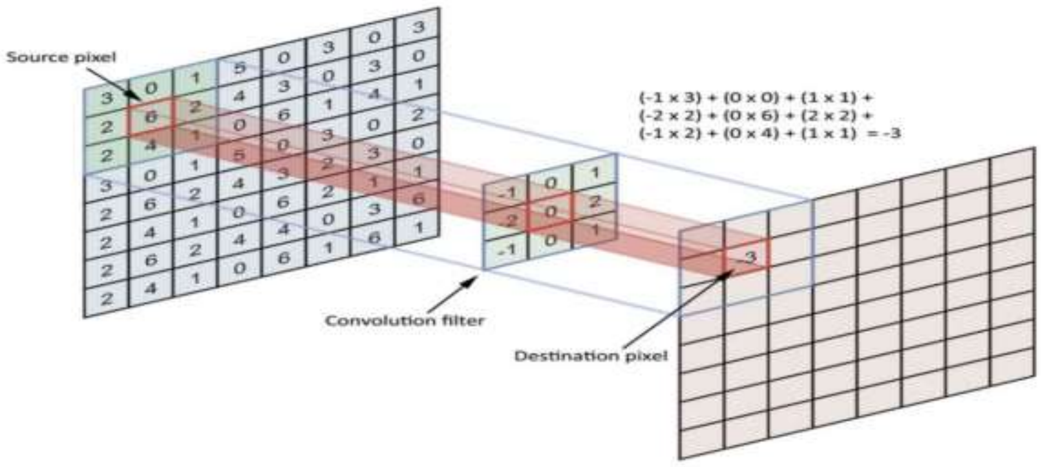

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

Figure II.4: 3x3 kernel convolution operation in a CNN [55]

The size of the feature map is controlled by the three parameters described below:

- **Stride:** The number of pixels by which the filter matrix slides over the input matrix.
- **Depth:** The number of channels of filters used for the convolution operation (for example RGB filters have a depth of 3).
- **Padding:** The number of pixels added to an image when it is being processed by the kernel. An example of padding is shown in figure II.4

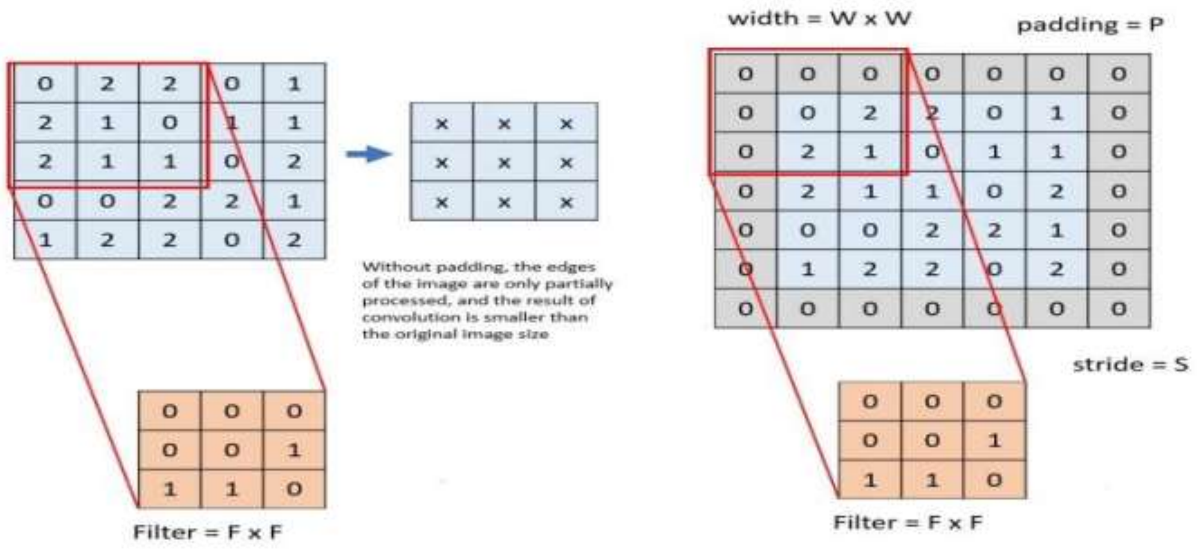

width = W x W
padding = P
Without padding, the edges of the image are only partially processed, and the result of convolution is smaller than the original image size
stride = S
Filter = F x F
Filter = F x F

Figure II.5: Convolution operation with zero-padding [56]

- **Rectified Linear Unit (ReLU) layer:** Following each convolution operation, this layer applies an elementwise activation function. One commonly used activation function is ReLU (Rectified Linear Unit), which is applied independently to each pixel in the feature map. ReLU replaces all negative values in the feature map with zero, while leaving positive values unchanged. This non-linear operation introduces non-linearity into the network, enabling it to learn complex patterns and representations. Importantly, ReLU does not alter the size of the feature map, preserving its spatial dimensions throughout the network.
- **Pooling Layer:** The pooling layer, also known as subsampling or downsampling, is typically added after the nonlinearity (ReLU) in a Convolutional Neural Network (CNN). Its primary function is to reduce the dimensions of the feature maps, thereby decreasing the number of network parameters. This reduction helps control overfitting and computational complexity.
  The pooling operation is defined by a pooling function, often max pooling or average pooling. Similar to the convolution operation, the pooling layer operates independently on each feature map. It spatially resizes each feature map by sliding a filter over each channel of the feature map.
  In the case of max pooling, which is illustrated in Figure II.6, a commonly used configuration is a pooling layer with filters of size 2x2 connected with a stride of 2. This means that the pooling filter moves across the feature map in steps of 2 pixels at a time. In each step, the maximum element from the selected block of the feature map is retained, effectively reducing the spatial dimensions of the feature map while preserving important features.

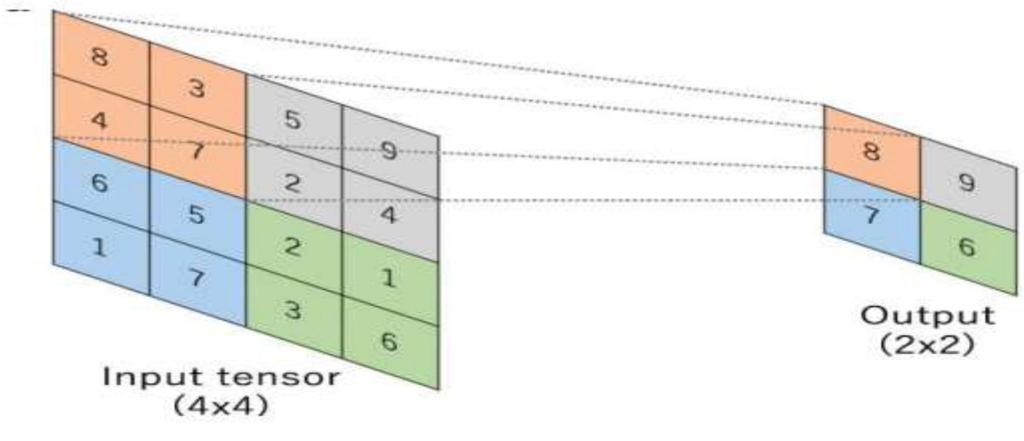


Figure II.6 : Max-pooling layer operation [57]

- **Flatten Layer:** Flattening is a crucial step in a Convolutional Neural Network (CNN) that transforms the multidimensional output from the previous layers into a one-dimensional vector. This transformation, as depicted in Figure II.7, is necessary to prepare the data for connection to the final classification layer, typically a fully connected layer.

By flattening the data, the spatial structure of the feature maps is removed, and the resulting one-dimensional vector contains all the extracted features concatenated together. This vector is then passed as input to the fully connected layer, which performs classification or regression tasks based on the learned features. Flattening thus plays a vital role in enabling CNNs to effectively process and analyze complex data, such as images, for various applications.

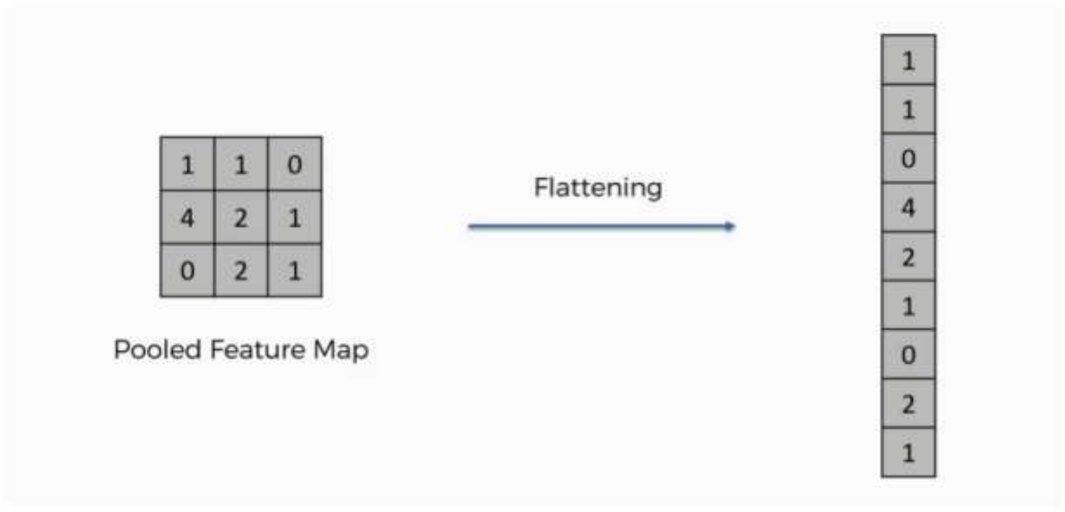


Figure II.7: Flattening for a three-dimensional data [58]

- **Fully connected layer:** A fully connected (FC) layer is similar to a convolution layer that uses the multilayer perceptron principle shown in Figure II.3. In most CNN models, the last few layers are fully connected that compile the data extracted by previous layers to form the final output. The FC layer computes the input image’s class probability using the softmax activation function as presented in Figure 06 Training a Convolutional Neural Network.

II.4.1 Forward Propagation

Forward propagation, or the forward pass, involves computing and storing intermediate variables and the final output of a neural network from the input layer to the output layer. In Convolutional Neural Networks (CNNs), this process includes passing input data through various layers such as convolutional, pooling, activation, and fully connected layers. During forward propagation, weights, biases, and filters are randomly initialized and treated as parameters of the CNN algorithm. These parameters are adjusted during training through backpropagation to minimize a predefined loss function. Overall, forward propagation is essential for neural networks to process input data and generate predictions for tasks like image classification and natural language processing.

II.4.2 Cost function:

The cost function, also called the loss function, evaluates a Deep Learning model's performance by measuring the error between predicted and expected values, producing a single real number. Its form varies based on the problem. The model's goal is to minimize this function by adjusting parameters and weights. Common cost functions include mean squared error (MSE) and cross-entropy, chosen based on the problem's characteristics. The model iteratively updates parameters to minimize the chosen cost function during training, enhancing predictive accuracy.

- **Mean Square Error (MSE):**

The Mean Squared Error (MSE) is a measure of error in mathematical models, calculated as the average squared distance between predicted and observed values. A perfect model yields an MSE of zero, while increasing errors result in higher MSE values. The MSE quantifies model performance and can be computed using a specific equation.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(yi - \widehat{yi}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(II.6)}$$

In the case of neural networks :

$\hat{\hat{Y}}_I$: is the predicted value

Y_i : is the actual value

N : is the number of data points.

- **Cross Entropy**

Cross entropy is often referred to as “cross-entropy”, “logarithmic loss”, “logistic loss”, or “log loss” for short. It is the average number of bits required to send a message from distribution A to distribution B. The notion of entropy can also be useful as a cost function in classification problems, producing faster learning results than MSE. The cross-entropy equation is also known as a negative loglikelihood [58]. The cross-entropy cost can be calculated using the following equation:

$$Cross - entropy = \frac{1}{N}\sum_{1}^{N}(y * log\hat{y} - (1-y)\log(1-\hat{y}))\text{..........} \quad \text{(II.7)}$$

- **Gradient descent:**

Gradient descent is an iterative optimization technique used to locate a local minimum of a differentiable function, as depicted in Figure II.07. In this approach, steps are taken in proportion to the negative gradient of the function at the current point to move towards a local minimum. Conversely, steps proportional to the positive gradient lead towards a local maximum of the function. Neural networks often employ gradient-based learning, utilizing gradients of the cost function for training [59]. Like other machine learning domains, neural networks aim to minimize the cost function. However, due to the non-linear nature of neural networks, many cost functions are non-convex, making it more beneficial to move in the direction opposite to the gradient for better results [59].

Utilizing the cost function C, expressed in terms of the weights w, biases b, and the learning rate ŋ of the neural network, gradient descent can be applied to update these parameters efficiently towards minimizing the cost function [58]. The weights and biases are iteratively adjusted using equations, respectively.

$$W' = W - ŋ\frac{dc(W,b)}{dW} \text{..} \quad \text{(II.8)}$$

$$b' = b - ŋ\frac{dc(W,b)}{dW} \text{.......................................} \quad \text{(II.9)}$$

Due to the often-large data volumes involved in training and testing neural networks, computing an average change for the entire dataset can be infeasible. An alternative approach is stochastic gradient descent. In stochastic gradient descent, a batch of data, or a subset thereof, is selected, and the average change for that batch is computed. This batch can comprise any number of data points, with a trade-off between noisy fluctuations, computation time, and memory [59].

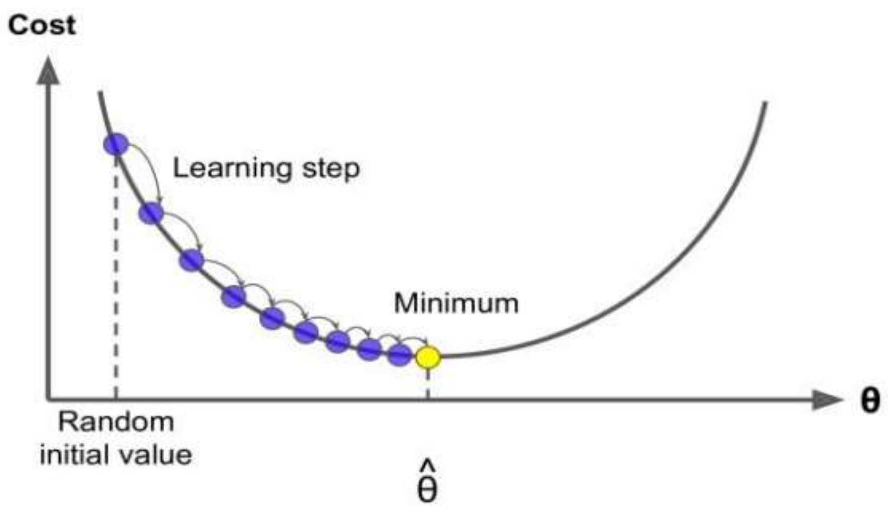


Figure II.8 : Schematic of gradient descent [60]

- **Backward propagation:** The neural network adjusts the weights of its neurons to approximate desired outputs based on provided inputs. Analytically updating neuron weights in a multi-layer network poses challenges, but the backpropagation algorithm offers a straightforward and efficient iterative solution. Backpropagation is extensively employed in training feed-forward neural networks for supervised learning. Although strictly speaking, "backpropagation" refers solely to the gradient computation algorithm and not its application, it's commonly used more broadly to encompass the entire learning process, including methods like stochastic gradient descent.

The backpropagation algorithm operates by computing the gradient of the loss function with respect to each weight, employing the chain rule. It calculates the gradient layer by layer, moving backward from the final layer to prevent redundant intermediate term calculations. In the initial stage, an input vector undergoes forward propagation through the neural network, producing an output which is then compared to the desired output using a cost function. Subsequently, the gradient of the cost function, representing error values, is computed. These error values are then propagated backward through the network to determine the error values of the hidden layer neurons. Finally, the weights are adjusted accordingly. This process is reiterated with different inputs until the weights converge. The relationship between gradient descent and backpropagation is illustrated in fig 08

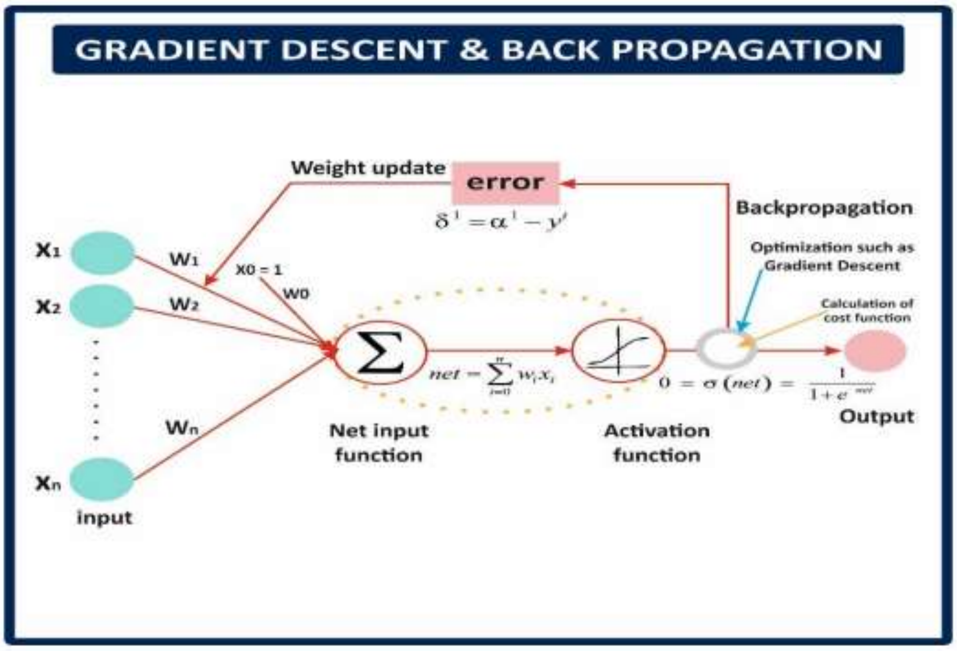

GRADIENT DESCENT & BACK PROPAGATION
Weight update
error
$\delta^1 = \alpha^1 - y^i$
Backpropagation
Optimization such as Gradient Descent
Calculation of cost function
X1
X2
Xn
W1
W2
Wn
X0 = 1
W0
input
$net = \sum_{i=0}^{n} w_i x_i$
$0 = \sigma(net) = \frac{1}{1+e^{-net}}$
Net input function
Activation function
Output

Figure II.9 : Propagation and weights update in backpropagation algorithm [61]

- **Overfitting**

. Overfitting arises when an analysis closely matches a specific dataset, potentially leading to a failure in generalization to new data or accurate prediction of future observations. This phenomenon is common in Deep Learning algorithms, where the model attempts to capture all nuances within the training data, including noise and random fluctuations. Overfitting often occurs due to the utilization of overly complex networks to account for anomalies in the dataset under examination. It manifests when a model becomes overly tailored to the original data, resulting in potentially unreliable outcomes when applied to new data and subsequently leading to suboptimal decision-making. Refer to Figure II.10 for a detailed depiction of this concept.

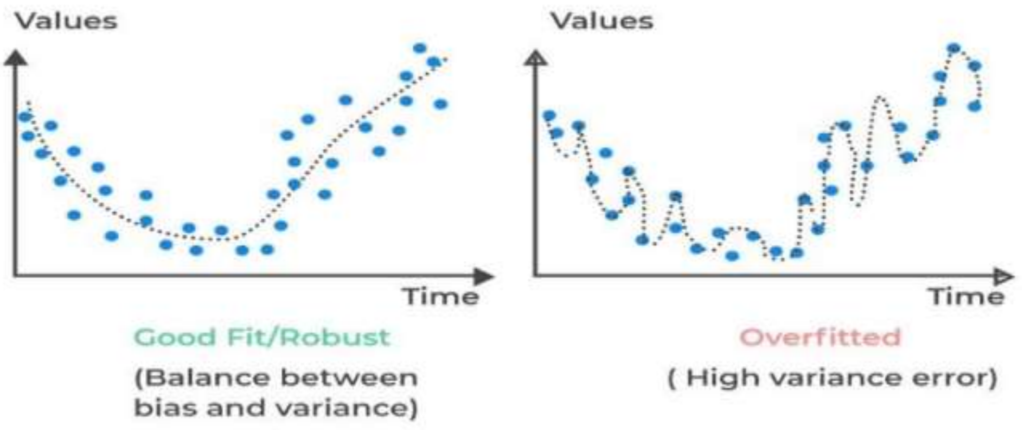


Figure II.10: The difference between a good fit (left) and overfitted function (right) [62]

Overfitting can be mitigated through various techniques such as reducing the network size, increasing the size of the training dataset, and implementing Dropout regularization. By decreasing the network size, the model's capacity to fit noise and irrelevant patterns is limited, promoting better generalization to unseen data. A larger training dataset provides the model with a more diverse range of examples, helping it learn more robust and generalizable patterns. Dropout regularization randomly deactivates a fraction of neurons during training, forcing the network to learn redundant representations and reducing its reliance on specific features, thus enhancing its ability to generalize.

- **Dropout regularization**

  Dropout serves as a regularization technique employed to combat overfitting in artificial neural networks by discouraging the development of complex co-adaptations during training. During its application, a subset of neurons is randomly deactivated (turned off or ignored) during training, as illustrated in Fig II.11. This temporary elimination of their contribution to subsequent neuron activations occurs during the forward pass, with no weight adjustments applied during the backward pass. Dropout is typically implemented per layer in the neural network, including dense fully connected layers and convolutional layers, but it's not applied to the output layer.

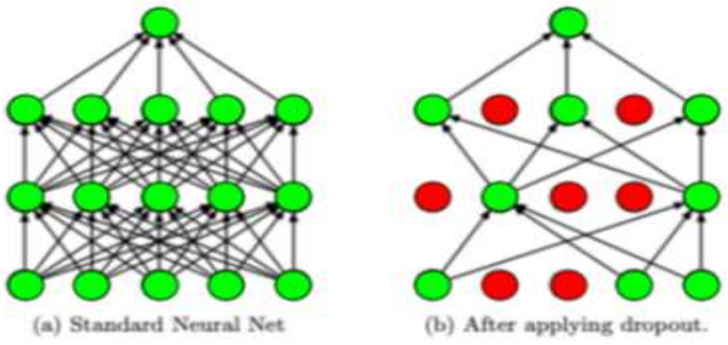


Figure II.11 : Dropout regularization [63]

- **Batch normalization**

  Batch normalization, often referred to as Batch norm, is a training method designed for intricate Deep Neural Networks, which involves standardizing the inputs of a layer for each mini-batch. This technique aims to enhance the speed, stability, and efficiency of Neural Networks while notably diminishing the number of training epochs needed to effectively train deep networks.

II.5 Conclusion

In this chapter, we have discussed the theory of Convolutions neural networks, starting with a general introduction to Neural Networks going through CNN, we introduced activation functions, Convolutional Neural Networks techniques, and how they are trained.

Chaptre III

Models Explanations

Chapter III

Models Explanations

III.1 Introduction

In the landscape of deep learning architectures, Xception, VGG19, ResNet50 EfficientNet80 emerge as notable contenders, each presenting distinctive approaches to image recognition tasks. ResNet50, a variant of the ResNet model, boasts a deep convolutional neural network with 50 layers, specifically trained on ImageNet data. It excels in image classification across a diverse array of object categories, leveraging rich feature representations learned from extensive training. In contrast, EfficientNet80 embodies a systematic scaling approach, optimizing convolutional neural networks with a structured increase in depth, width, and resolution. By incorporating innovative building blocks like MBConv with squeeze-and-excitation optimization, EfficientNet80 aims for superior accuracy and efficiency in image recognition.

III.2 Xception Algorithm

Brain tumors pose a significant threat to human health, requiring accurate and timely diagnosis for effective treatment and management. While medical imaging techniques, such as magnetic resonance imaging (MRI), provide valuable insights into brain abnormalities, the manual interpretation of these images is labor-intensive, prone to error, and often relies on limited expert availability. As a result, there is a pressing need for automated and reliable methods for brain tumor classification to enhance diagnostic accuracy, streamline clinical workflows, and improve patient outcomes.

In this project, we aim to develop and evaluate convolutional neural network (CNN) models for the classification of brain tumors using MRI data. Specifically, we will explore the performance of four CNN architectures: VGG19, Xception, ResNet50, and EfficientNET80. These models offer varying complexities and capabilities, allowing for comprehensive analysis of their effectiveness in brain tumor classification tasks.

III.2.1 Xception Model Explanation:

The Xception architecture figure III.1 is an advanced deep learning model that is an extension of the Inception architecture. It employs depthwise separable convolutions which factorize a standard convolution into two separate operations: depthwise and pointwise convolutions. This design reduces the computational cost and the number of parameters, leading to a more efficient and often more effective model.

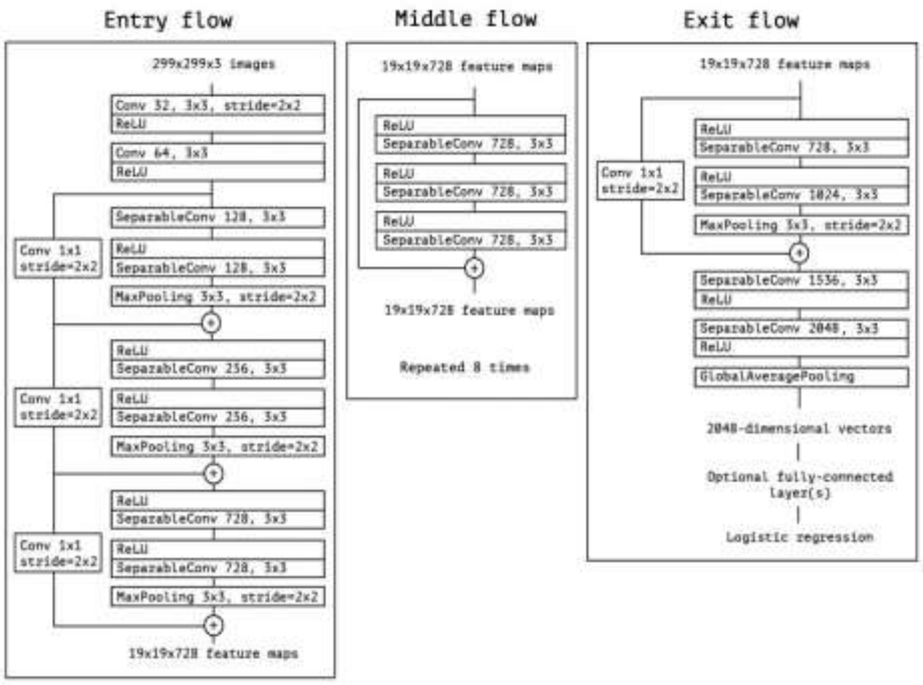


Figure III.1: Xception Model Architecture

III.2.2 Components of the Xception Architecture

- **Inception Module:** A building block that applies convolutions of different sizes and concatenates the results.

- **Depthwise Separable Convolutions:** These convolutions include a depthwise convolution that filters input data followed by a pointwise convolution that combines these outputs.

- **Entry Flow:** The initial layers of the network which process the input data with standard convolutions.

- **Middle Flow:** Consists of repeated depthwise separable convolutions which form the core of the network.

- **Exit Flow:** The final layers which prepare the data for output, typically including pooling, fully connected layers, and a softmax activation for classification.

The Xception model is widely used for tasks such as image classification,

III.3 VGG19 Algorithm

III.3.1 VGG19 Architecture:

VGG19 is a model that was developed figure III.2 by the Visual Graphics Group (VGG) at the University of Oxford and was introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by K. Simonyan and A. Zisserman. The '19' in VGG19 indicates that it has 19 layers that have weights; this depth is one of the key attributes that made it notable at the time of its creation.

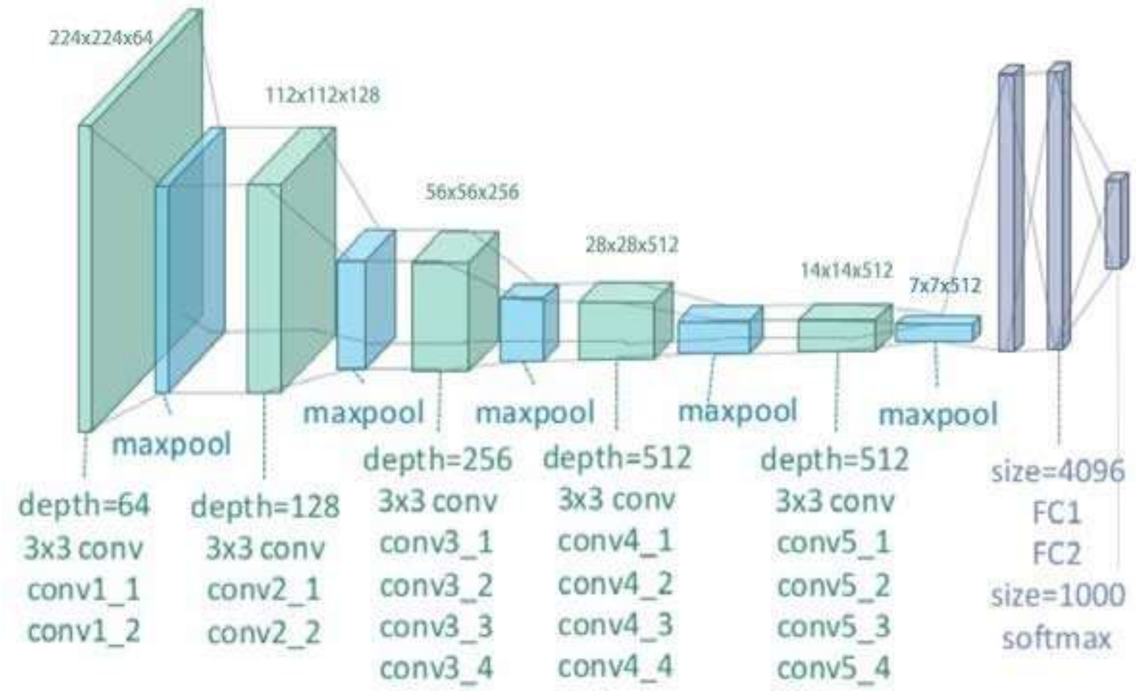


Figure III.2: VGG19 Model Architecture.

III.3.2 Components of the VGG19 Architecture

The architecture is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully connected layers, each with 4096 nodes are then followed by a softmax classifier. Figure III.2 gives a brief explanation of his architecture.

- **Input Layer:** The network takes an input image of size 224x224 pixels with 3 color channels (RGB)
- **Convolutional Layers (conv):** There are several convolutional layers in VGG19, each performing convolutions on the input data with filters to extract features. These are denoted as `conv1_1`, `conv1_2`, etc. The number after 'conv' refers to the stage in the network, and the second number refers to the layer within that stage. For example, `conv3_3` is the third convolutional layer in the third stage. The notation `3x3` refers to the size of the filters (3 pixels by 3 pixels), and the depth (e.g., 64, 128, 256, 512) refers to the number of filters used at that stage.
- **Max Pooling Layers (maxpool):** These layers are interspersed between the convolutional layers and are used to reduce the spatial dimensions (width and height) of the input volume for the next convolutional layer. It helps to reduce the computation required and also helps in making some of the features detected by the convolutions more robust.
- **Fully Connected Layers (FC):** Towards the end of the network, there are three fully connected layers. The first two have 4096 channels each, and the third performs classification and has 1000 channels (one for each class in the dataset for which VGG was originally trained, which is ImageNet).
- **Softmax Function:** This is the last layer in the network and is used to convert the output of the last fully connected layer into probability distributions - each number from this layer will represent the probability that the input image belongs to one of the 1000 classes.

As for equations, in a CNN like VGG19, several operations are performed, the most notable are:

- **Convolution:** For a given input image I and a filter F , the convolution (I * F) at a location (x, y) is given by:

$$(I * F)(x,y) = \sum ai = -a \sum aj = -a\, I(x+i, y+j).F(i,j) \text{..........(III.1)}$$

- **ReLU Activation:** This function is applied after each convolution operation to introduce non-linearity:

$$F(x) = \max(0,x) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(III.2)}$$

- **Max Pooling:** This operation reduces the spatial size of the input volume. For a 2x2 pooling window, the operation can be defined as:

$$M(x,y) = \max\big(I(x,y), I(x+1,y), I(x,y+1), I(x+1,y+1)\big) \ldots\ldots \text{(III.3)}$$

III.4 Resnet 50 Model Explanation

III.4.1 Resnet50 Model Architecture:

The ResNet50 model is a variant of the ResNet model which has 50 layers deep as figure III.3. It is a convolutional neural network that is 50 layers deep and is trained on a large number of images from the ImageNet database. The model is designed to recognize images with minimal preprocessing. It can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the model has learned rich feature representations for a wide range of images. The model has an image input size of 224x224.

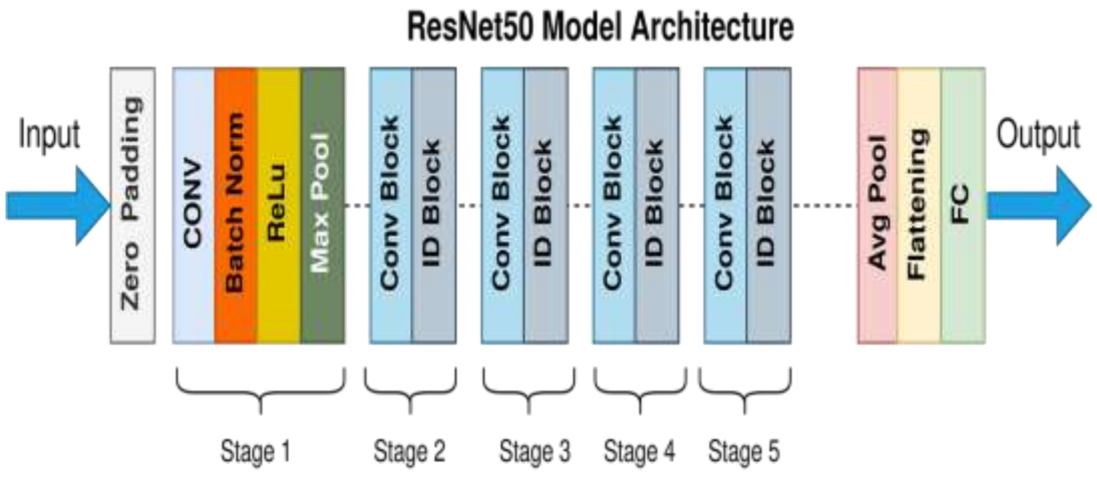


Figure III.3: Resnet50 Model Architecture

III.4.2 Components of the Resnet50 Architecture

- **Input Layer**

The input for ResNet50 is an image of size 224x224x3 (height, width, channels), where the image is preprocessed as per the model's requirements.

- **Zero Padding**

Zero Padding is applied to the input image to preserve the spatial dimensions of the output after convolution. For a kernel size of 7x7, a typical padding of 3 pixels is applied on all sides.

- **Initial Convolutional Block**

This block consists of a convolutional layer with filters of size 7x7, a stride of 2, followed by batch normalization and ReLU activation. A max pooling layer with a stride of 2 is then applied to reduce the spatial dimensions.

- **Convolutional Blocks**

Each convolutional block consists of three main operations in sequence: a convolutional layer, batch normalization, and a ReLU activation. These blocks are repeated multiple times, with varying numbers of filters and strides to capture features at different scales.

- **Identity Blocks**

Identity blocks are the blocks that have a shortcut connection that skips one or more layers. The shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers.

- **Average Pooling**

Average pooling is used to reduce the spatial dimensions of the feature maps by taking the average of elements in the pooling window.

- **Flattening**

The flattened layer is used to convert the final feature maps into a single vector of values, which is then fed into the fully connected (FC) layer.

- **Fully Connected (FC) Layer**

The fully connected layer uses the features from the flattened layer to classify the input image into one of 1000 classes. It is usually implemented with a softmax activation function.

The ResNet50 model utilizes residual connections to enable training of deep neural networks without the vanishing gradient problem. By incorporating identity blocks with shortcut connections, it allows the model to learn identity functions that are crucial for preserving the learned features through the depth of the network.

III.5 EfficientB0 Model Explanation

III.5.1 EfficientNET80 Model Architecture

EfficientNET80 is a scaling of baseline model architecture that uses a compound coefficient to scale up CNNs in a more structured manner. Unlike conventional scaling methods that arbitrarily scale these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. The architecture figure III.4 is designed to achieve better accuracy and efficiency.

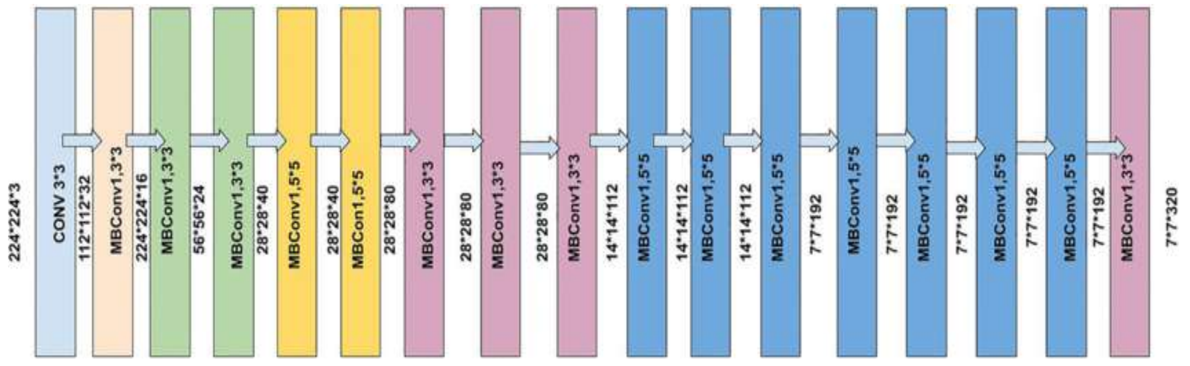


Figure III.4*:* EfficientNET80 Model Architecture

III.5.2 Components of the EfficientNET80 Architecture

- **Input Layer**

The input for EfficientNet80 is an image of size 224x224x3 (height, width, channels). The model employs a systematic approach to scaling the dimensions of the depth, width, and resolution of the network.

- **Stem**

The stem of the network consists of an initial convolutional layer with a 3x3 kernel size, followed by batch normalization and a SiLU (Swish) activation function.

- **MBConv Blocks**

MBConv blocks, also known as inverted residuals with linear bottleneck, are the main building blocks of the EfficientNet80. Each block typically starts with an expansion phase (using 1x1 convolutions), followed by a depthwise convolution (3x3 or 5x5), and then a projection phase that reduces the number of channels.

III.5.3 MB Conv Block Descriptions:

- **Block 1**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k3x3. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 2**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k3x3. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 3**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k5x5. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 4**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k3x3. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 5**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k5x5. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 6**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k5x5. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 7**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k3x3. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 8**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k5x5. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Block 9**

The block uses a mobile inverted bottleneck convolution (MBConv) with a kernel size of k5x5. It includes a squeeze-and-excitation optimization that recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels.

- **Top Layer**

After the final MBConv block, the network includes a convolutional layer with a 1x1 kernel size, followed by batch normalization and a SiLU activation. This is followed by a global average pooling layer and a fully connected layer that outputs the probabilities for each class.

The EfficientNet80 model utilizes compound scaling and a carefully balanced design of network depth, width, and resolution, which contributes to its efficiency and effectiveness. The inclusion of MBConv blocks with squeeze-and-excitation optimization further enhances the model's performance.

III.6 Conclusion

In conclusion, the utilization of advanced CNN architectures such as VGG19, Xception, ResNet50, and EfficientNET80 holds immense promise in revolutionizing brain tumor classification through MRI analysis. These models offer varying complexities and capabilities, allowing for a comprehensive assessment of their effectiveness. By automating the classification process, these models have the potential to significantly enhance diagnostic accuracy, alleviate the burden on healthcare professionals, and ultimately improve patient outcomes. As technology continues to advance, further refinement and optimization of these CNN models will undoubtedly continue to propel the field of medical imaging towards more efficient and reliable diagnostic methodologies.

Chapter IV

Experiments and Results

Chapter IV
Experiments & Results

IV.1 Introduction

This chapter will cover the training and the evaluation of Brain tumor identification and models using several DL classification architectures and localization. Considering the importance of choosing the right dataset, we decided to use the "MRI for brain tumor

"Dataset" datasets available on Kaggle platform that are described in the following sections.

IV.2 Datasets

To Provide an accurate model that can be used for plant disease identification, it is essential to use a dataset with a large number and good quality of images. After an intense search, we decided to use two different datasets (with the same classes); one dataset to train the model and the other one to evaluate its performance. Fortunately, we found the two available datasets on Kaggle that are described below

IV.3 MRI For Brain Tumor Dataset

The "MRI For Brain Tumor" contains around 3929 RGB (Red, Green, Blue) images of tumor classified into 2 distinct classes. Out of 2 classes 2556 are no tumor (No Mask), and 1373 are diseased with tumor (with mask) classes. The entire data is divided into 85 percent and 15 percent for training and validation respectively preserving the directory structure. This dataset was used to train our models. Sample images from the MRI for Brain tumor dataset are shown:

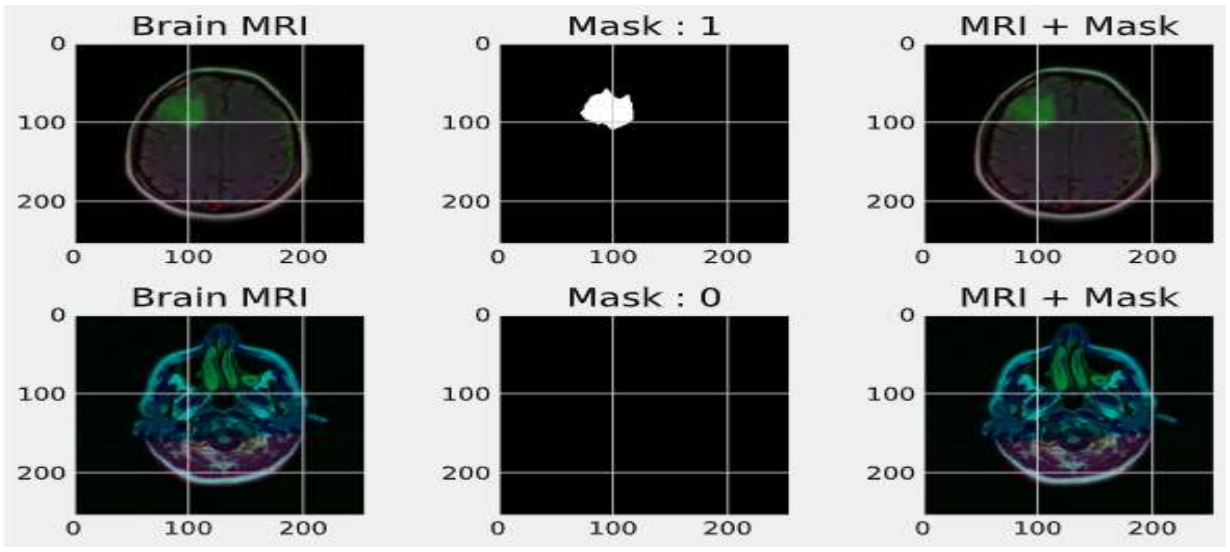


Figure IV.1: Some MRI brain tumors images

IV.4 Dataset image Distribution

Table 1 summarizes the datasets presented in this section along with the number of images for each class of the training, validation, and testing directories.

Classes	Training Images	Testing Images	Validation Images
With Mask	2190	336	109
Without Mask	1149	224	57

Table IV.1 : Dataset Distribution

IV.5 Data Augmentation

Data augmentation serves as a method for enhancing the resilience of Deep Learning models by generating altered versions of existing training data, thereby obviating the need for fresh data collection. When training Deep Neural Networks with extensive datasets, a notable outcome is the refinement of model accuracy and the generation of diversified image representations. These variations contribute to the models' adeptness in extrapolating learned patterns to novel data. Predominantly, common techniques employed in data augmentation include cropping, zooming, and horizontal flipping.

IV.6 Tools

This segment will delve into the ecosystem and various tools leveraged in our model training process. Our algorithms were developed utilizing the Python programming language and trained through the TensorFlow framework in conjunction with the Keras library. Furthermore, our models underwent training on the Kaggle platform.

IV.6.1 Python Programming Language

Python stands as an open-source, interpreted, object-oriented, and high-level programming language. Rooted in a design philosophy that prioritizes code readability to alleviate the burden of program maintenance [42], Python offers support for modules and packages. This versatility fosters its utilization across a spectrum of applications, spanning web development (server-side), software engineering, mathematical computations, and system scripting.

IV.6.2 TensorFlow

TensorFlow (TF) emerges as a freely available, open-source software framework designed for facilitating data flow within programming environments. Its primary function encompasses serving as a mathematical library for computations, accommodating diverse platforms such as CPUs and GPUs. Originating from the collaborative efforts of researchers and engineers affiliated with the Google Brain team, TensorFlow made its debut on November 9, 2015. Renowned for its prowess in Machine Learning and Deep Learning applications, this library boasts an extensive array of pre-trained models, facilitating the training of additional models through transfer learning techniques.

IV.6.3 Keras

Keras stands as an open-source, high-level library crafted in Python, designed to seamlessly execute algorithms atop frameworks like TensorFlow, Theano, PlaidML, and various other machine learning libraries. Boasting Python as its programming language of choice, Keras aims to expedite Deep Neural Network processing. Its appeal lies in its simplicity and ease of use, offering a user-friendly, high-level Python interface while affording the flexibility of selecting different back-ends for computations. Additionally, Keras prioritizes modularity and extensibility, further enhancing its appeal to developers.

IV.6.4 Kaggle

Kaggle is the world’s largest data scientist and Machine Learning engineer’s community platform. Kaggle allows users to find and publish datasets in many fields, collaborate with other users, compete with other data scientists to solve data science challenges, and build models in a web-based data-science environment using GPU-integrated notebooks. Kaggle CHAPTER 3 EXPERIMENTS AND RESULTS 24 started by offering Machine Learning challenges and has now expanded to include a public data platform, a cloud-based data science workbench.

IV.6.5 Hyperparameters

Hyperparameters represent the parameters defined prior to the training of neural networks. These parameters, crucial in the realm of Deep Learning, encompass a broad spectrum, ranging from a few to potentially hundreds in complex models like Convolutional Neural Networks (CNNs). They exert influence over the learning process throughout training and ultimately impact the performance of the final model.

IV.6.6 Batch size

Batch size is a term used in machine learning that represents the number of training samples used in one iteration.

IV.6.7 Learning Rate

The learning rate serves as a pivotal tuning parameter within optimization methods, regulating the step size undertaken at each iteration as the model progresses towards minimizing a loss function during the training of Deep Neural Networks. The task of selecting an appropriate learning rate presents a challenge, as opting for a value that is too small may prolong the learning process, while one that is too high may lead to rapid convergence towards suboptimal weight configurations or even result in an unstable training regimen. Hence, an optimal learning rate strikes a delicate balance, being sufficiently low to foster accuracy improvements within the network while simultaneously ensuring that the training process remains expedient.

IV.6.8 Optimizer

During the training of a Deep Learning model, it is imperative to iteratively adjust the weights across each epoch to minimize the loss function. This pivotal task is facilitated by optimizers, which are algorithms tasked with dynamically modifying the attributes of the neural network, including weights and learning rates. By iteratively refining these parameters, optimizers contribute to the overarching objective of reducing the overall loss while concurrently enhancing the model's accuracy.

IV.6.9 Epoch

Within the domain of Artificial Neural Networks, an epoch denotes a single iteration through the entirety of the training dataset. Typically, training a neural network necessitates multiple epochs to effectively learn from the data. In essence, conducting training over multiple epochs, encompassing various patterns within the data, contributes to improved generalization when the model is subsequently evaluated on unseen data.

IV.6.10 Evaluation Metrics

Evaluation metrics play a crucial role in assessing and quantifying the performance of neural networks, with specific metrics tailored to the unique requirements of each use case. These metrics vary based on the nature of the problem being addressed and are instrumental in gauging the effectiveness of classifiers. Some commun evaluation metrics include:

- **True Positive (TP):** The test result that correctly indicates the presence of a class or a characteristic.
- **True negative (TN):** The test result that correctly indicated the absence of a class or a characteristic
- **False Positive (FP):** The test result that wrongly indicates the presence of a class or a characteristic.
- **False Negative (FN):** The test result that wrongly indicates the absence of a class or a characteristic

Based on the four indicators defined above, all the evaluation metrics used for classification are as follows:

IV.6.11 Confusion matrix

Indeed, a confusion matrix serves as a widely used tool for evaluating the performance of classification models and addressing associated challenges. It proves valuable for both binary and multiclass classification tasks. Particularly in scenarios involving unbalanced datasets, where there may be unequal observations across classes, traditional classification metrics can be misleading.

By contrast, a confusion matrix provides a more granular understanding of the model's performance by explicitly detailing the types of errors made. It categorizes predictions into true positives, true negatives, false positives, and false negatives, enabling a deeper analysis of the model's strengths and weaknesses across different classes.

		Predicted class	
		Normal	**Attack**
Actual class		True Negative (TN)	False positive (FP)
		False Negative (FN)	True positive (TP)

Table IV.2: Confusion Matrix for multiclassification

This nuanced perspective afforded by the confusion matrix enhances the interpretability of classification results, making it an indispensable asset for evaluating and refining classification models, especially in the face of imbalanced datasets.

IV.6.12 Accuracy

Classification accuracy is the proportion of correct predictions to the total number of input samples,

$$accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions made}} \text{……………………..(IV.1)}$$

IV.6.13 Sensitivity (Recall):

Sensitivity, also known as recall, indeed measures a model's capability to correctly predict true positives in each class. It quantifies the proportion of true positives identified by the model out of all actual positive instances. Mathematically, sensitivity can be calculated using the following formula:

$$Sensitivity(Recall) = \frac{TP}{TP+FN} \text{…………………....(IV.2)}$$

This metric is particularly valuable in scenarios where correctly identifying positive instances is crucial, as it highlights the model's ability to capture all positive cases, minimizing false negatives.

IV.6.14 Precision:

Precision serves as a key metric for evaluating the performance of a classification model, focusing on its ability to accurately identify relevant data points while minimizing false positives. Mathematically, precision can be calculated using the formula:

$$Precision = \frac{TP}{TP+FP} \text{……………………………………..(IV.3)}$$

This metric quantifies the proportion of true positive predictions made by the model out of all instances predicted as positive. Precision is particularly valuable when the cost of false positives is high, as it ensures that the model's positive predictions are highly accurate and reliable.

IV.6.15 F1-Score

The F1-score serves as a composite metric that harmoniously combines a classifier's precision and recall into a single value by computing their harmonic mean. This metric is particularly useful for comparing the overall performance of two classifiers.

The used equation to calculate the F1-score is as follows:

$$F1 - score = 2 * \frac{(precision * recall)}{(precision + recall)} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\text{(IV.4)}$$

The F1-score reaches its best value at 1 (perfect precision and recall) and worst at 0. It strikes a balance between precision and recall, providing a holistic assessment of a classifier's performance. This makes it a valuable metric for comparing the effectiveness of different classifiers, especially in scenarios were achieving both high precision and high recall is equally important.

IV.7 Experiments and Results

This section outlines the training and evaluation procedures for Four models, each employing architectures detailed in section 2.6 and assessed using metrics defined in section 3.5. The dataset underwent an 85/15 split, with 80% allocated for training and 15% for validation. To enrich the training data, we employed data augmentation techniques via the ImageDataGenerator class from the Keras library.

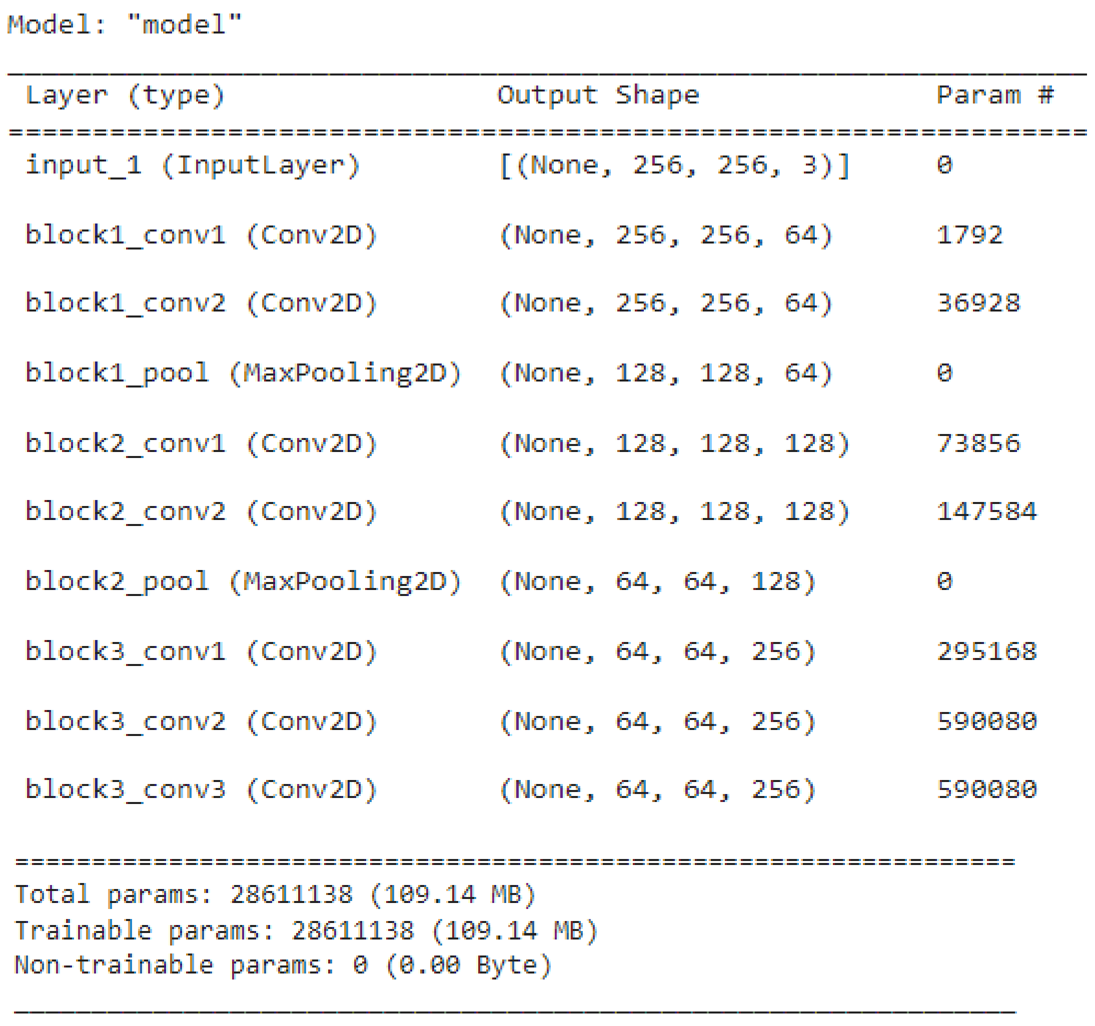
For model training, we adopted transfer learning methodology. The process commenced with loading the respective architectures from the Keras library, followed by sequential layers including batch normalization, a dense layer, dropout (with a rate of 0.5 to mitigate overfitting), a flatten layer, and finally, a dense layer featuring 38 neurons and a softmax activation function, serving as the output layer.

After iterative experimentation, the following hyperparameters were determined optimal for training our models:

- **Batch size:** 187 (default value)
- **Epochs:** 40 epochs (ceasing when training loss and accuracy plateaued)
- **Optimizer:** Adam (widely utilized for training CNNs)
- **Learning rate:** 0.002

IV.7.1 Experiment 1 : VGG19

TensorFlow, the Keras library, and the VGG19 architecture were used to train the initial model in our project. Fig. III.2 shows the number of trainable parameters and the layers employed to fine-tune this architecture.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

=================================================================
Total params: 28611138 (109.14 MB)
Trainable params: 28611138 (109.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure IV.2: The architecture of the model using VGG19

We trained the model and plotted the training and validation loss and accuracy. The graphs are shown by the Following figures.

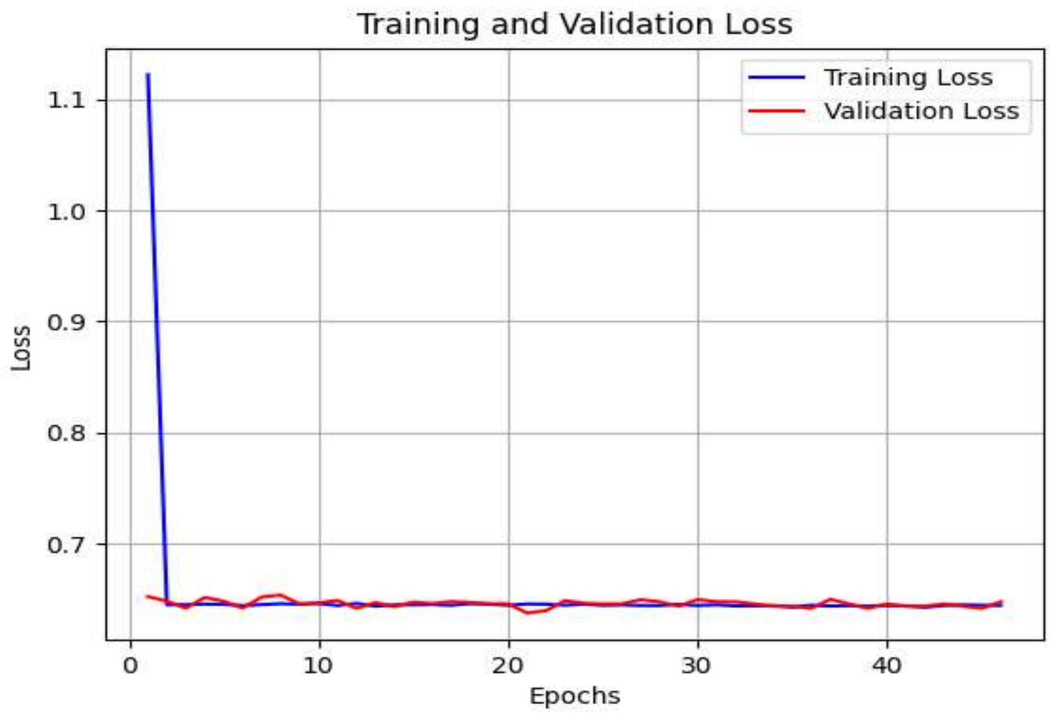


Figure IV.3: Training and validation loss

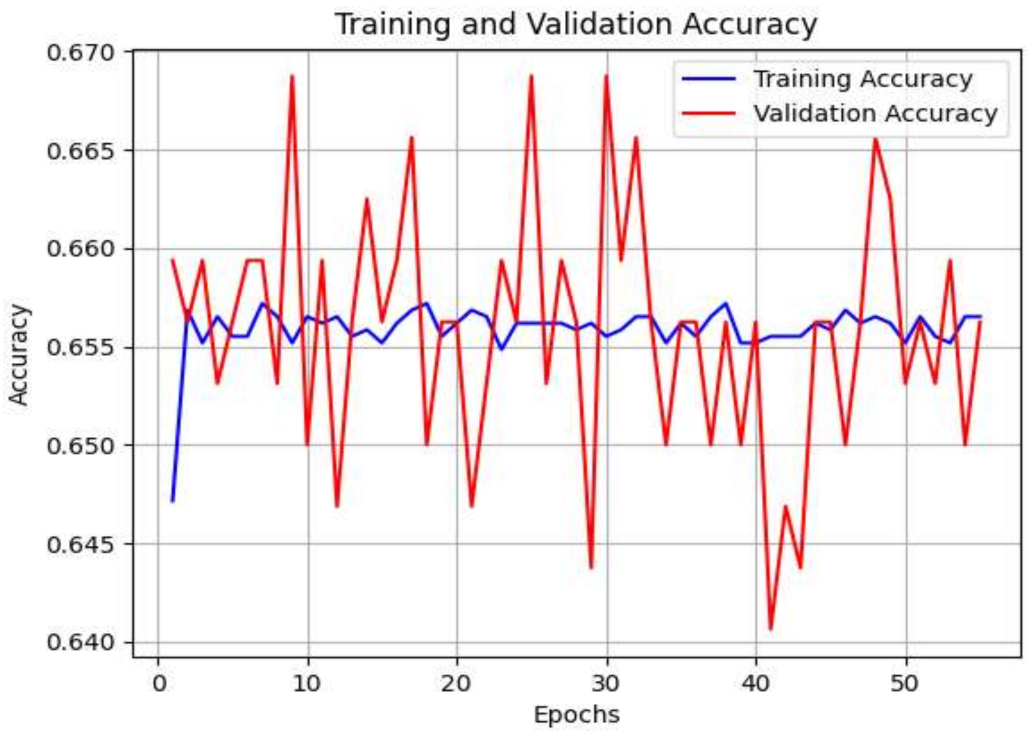


Figure IV.4: Training and Validation accuracy

In the previous analysis, it appears that there was a hint of underfitting observed in the graph for both accuracy and loss, despite employing a dropout of 0.3 for regularization. Subsequent to assessing the model's performance on the test set, we generated a confusion matrix illustrated in Fig IV.5 and computed various evaluation metrics for individual classes along with their average.

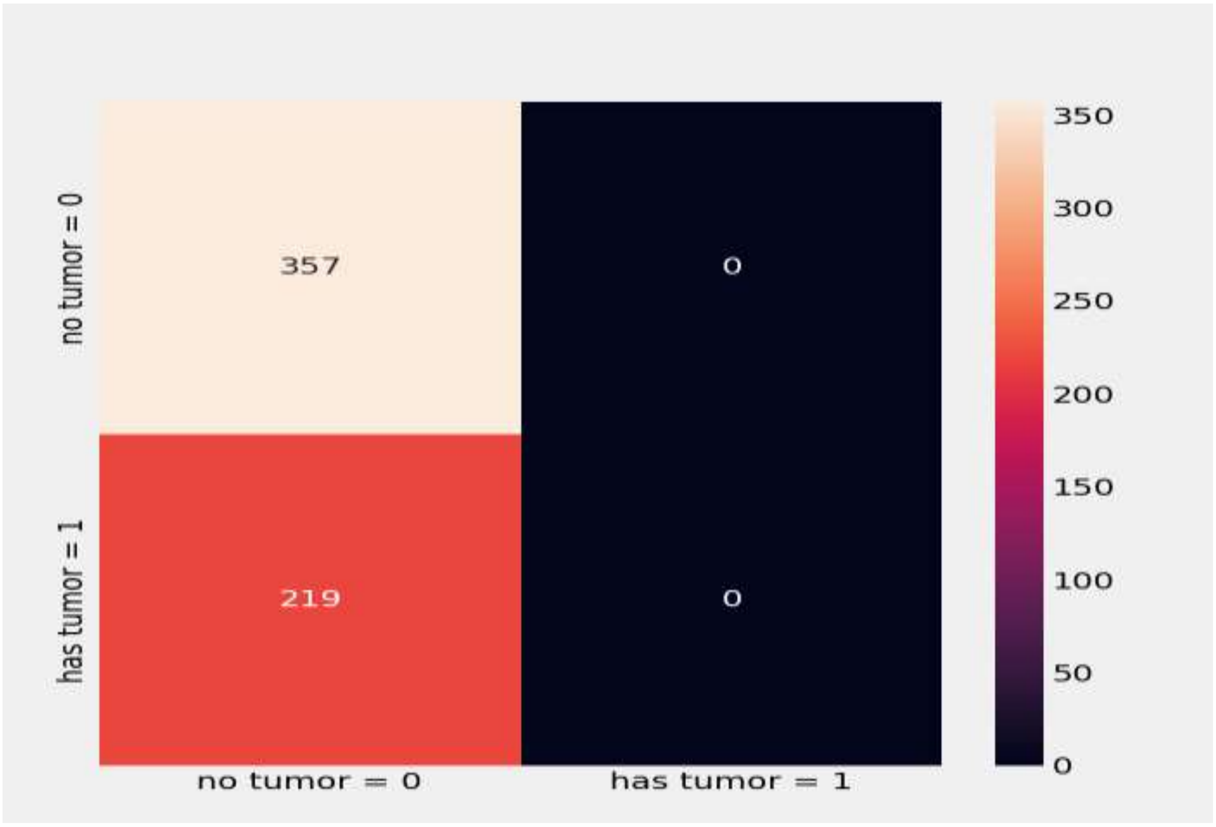


Figure IV.5: Confusion matrix of the VGG19 model

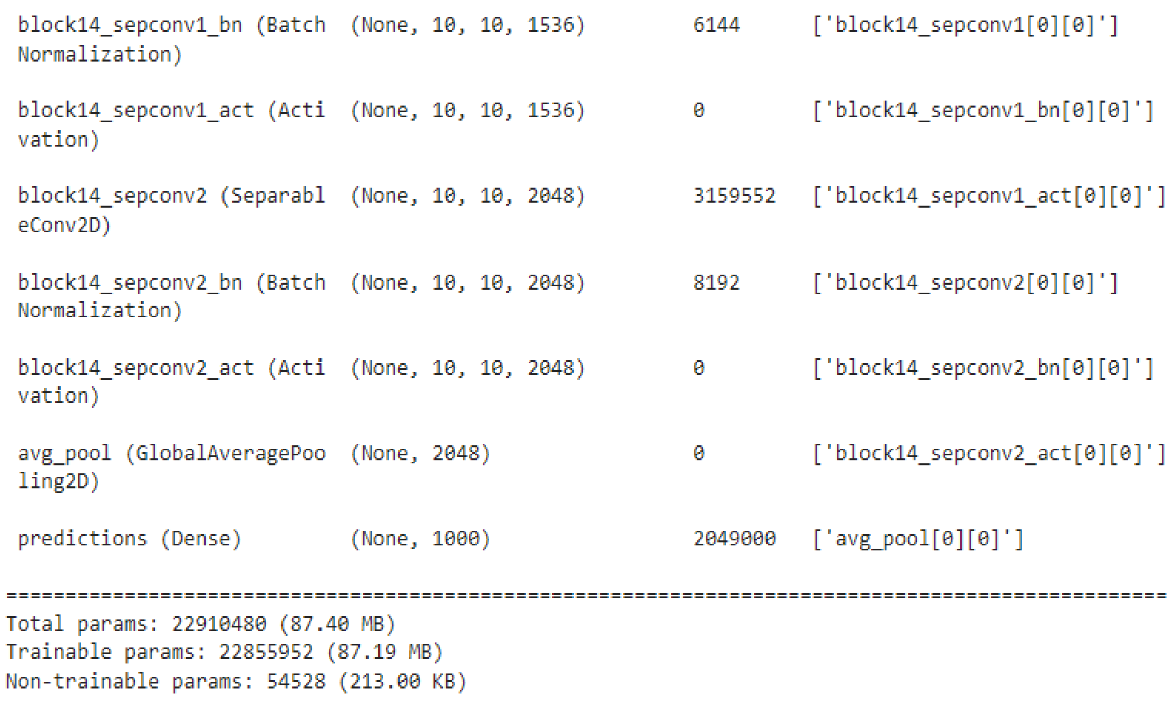
Classification report of VGG19 model:

	Precision	Recall	f1-score	Support
0	0.63	1.00	0.78	497
1	0.00	0.00	0.00	287
Micro avg	0.63	0.63	0.63	784
Macro avg	0.32	0.50	0.39	784
Weighted avg	0.40	0.63	0.49	784

Table IV.3 Classification report of the VGG19 model

IV.7.2 Experiment 2: Xception

The second model was trained using the Xception architecture following the same procedures as the VGG19 architecture. The architecture of the entire model is provided in Fig IV.6.



```
 block14_sepconv1_bn (Batch  (None, 10, 10, 1536)        6144      ['block14_sepconv1[0][0]']
 Normalization)

 block14_sepconv1_act (Acti  (None, 10, 10, 1536)        0         ['block14_sepconv1_bn[0][0]']
 vation)

 block14_sepconv2 (Separabl  (None, 10, 10, 2048)        3159552   ['block14_sepconv1_act[0][0]']
 eConv2D)

 block14_sepconv2_bn (Batch  (None, 10, 10, 2048)        8192      ['block14_sepconv2[0][0]']
 Normalization)

 block14_sepconv2_act (Acti  (None, 10, 10, 2048)        0         ['block14_sepconv2_bn[0][0]']
 vation)

 avg_pool (GlobalAveragePoo  (None, 2048)                0         ['block14_sepconv2_act[0][0]']
 ling2D)

 predictions (Dense)         (None, 1000)                2049000   ['avg_pool[0][0]']

==================================================================================================
Total params: 22910480 (87.40 MB)
Trainable params: 22855952 (87.19 MB)
Non-trainable params: 54528 (213.00 KB)
```

Figure IV.6 : The architecture of the model using Xception

We trained the model and plotted the training and validation loss and accuracy. The graphs are shown in Figure IV.7. Moreover

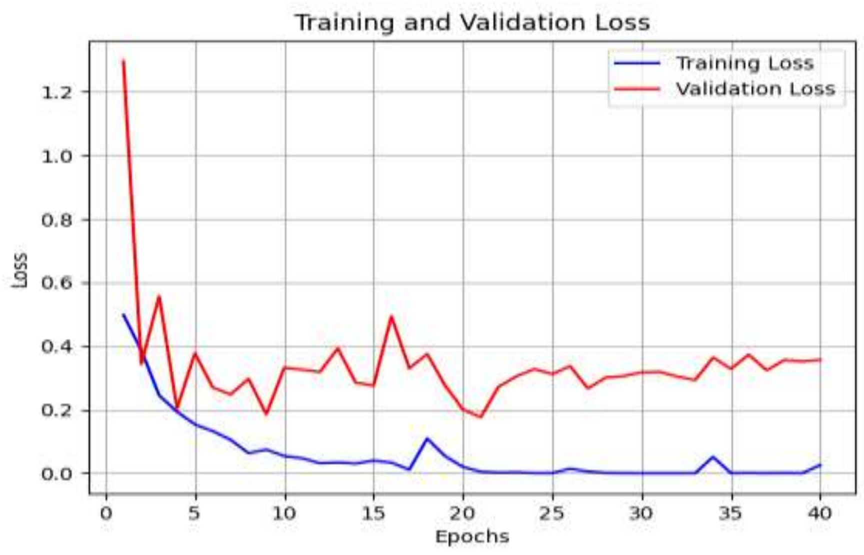


Figure IV.7: Xception model training and validation loss

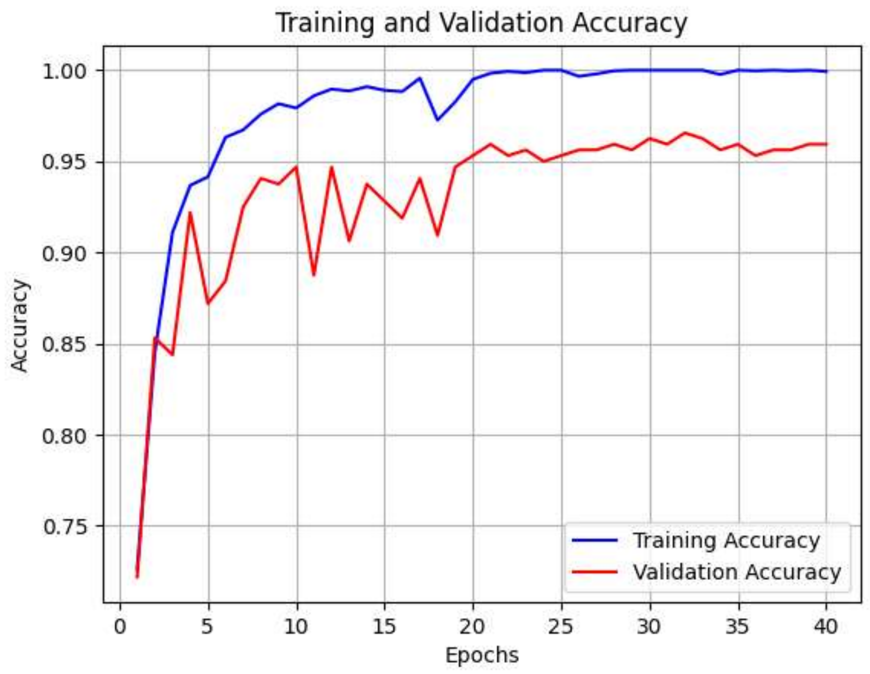


Figure IV. 8 : Xception model training and validation accuracy

From the previous graphs, we notice better performance in terms of training and validation accuracies. To evaluate the performance of this model we need to display the confusion matrix as given in Fig 10 and compute the evaluation metrics.

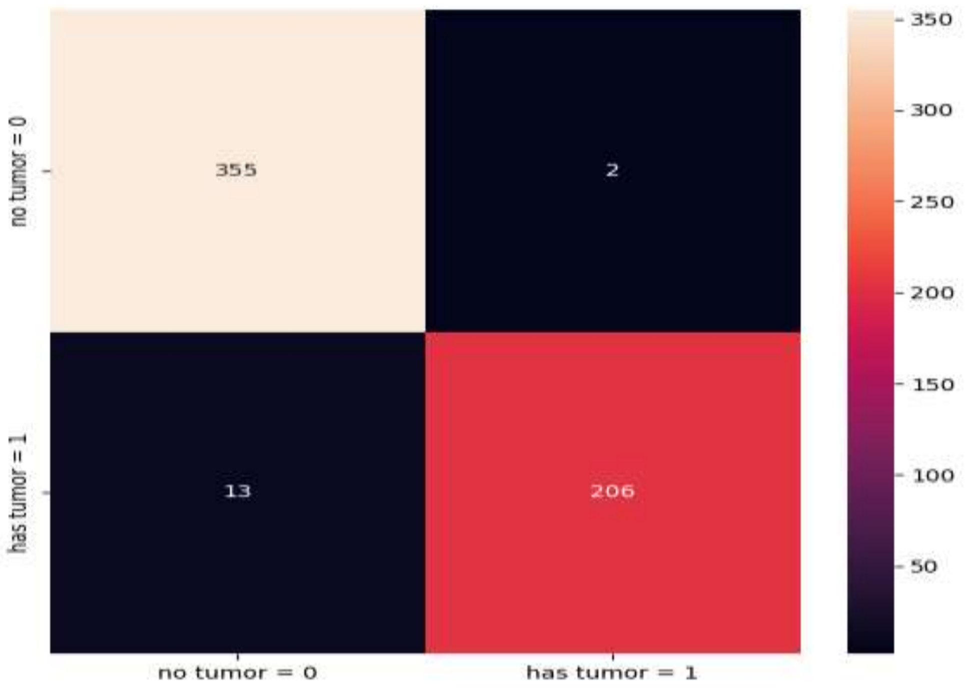


Figure IV.9 : Confusion matrix of the Xception model

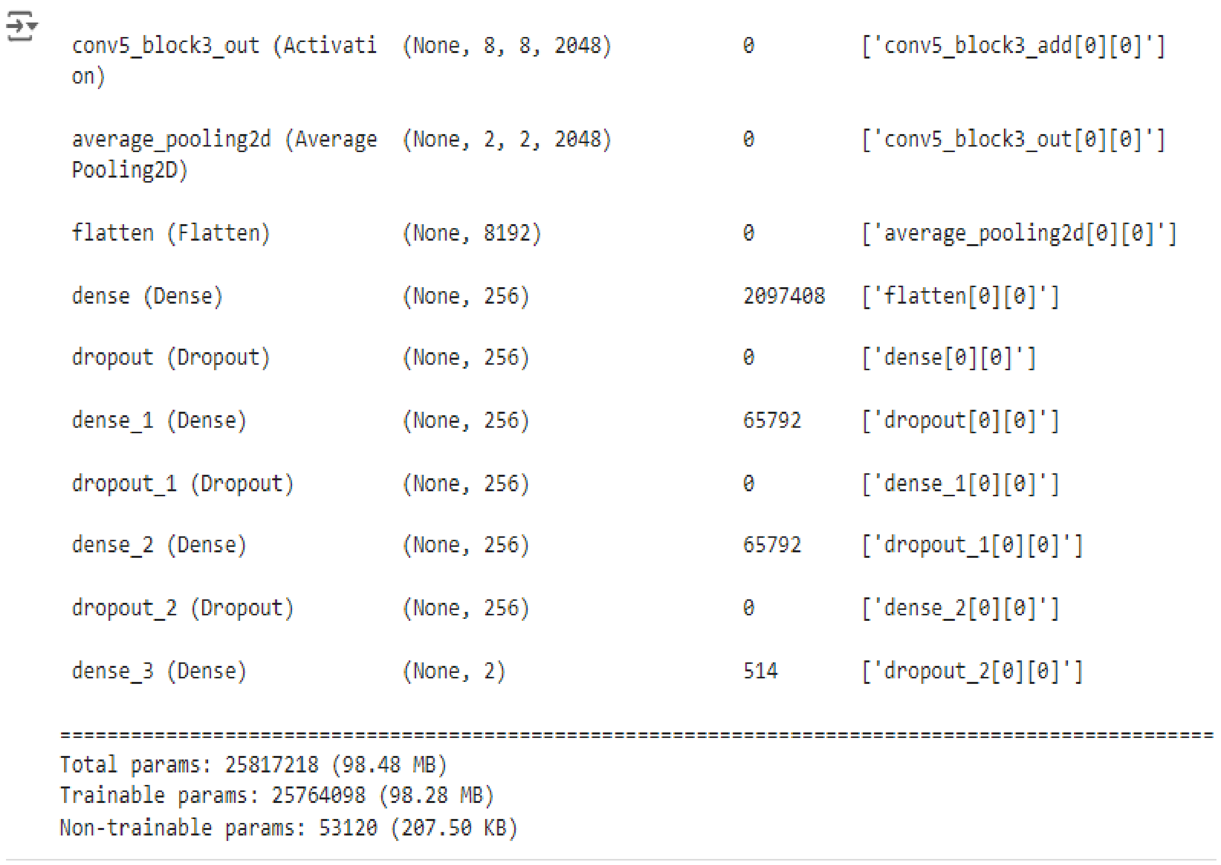
Classification report of Xception model:

	Precision	Recall	f1-score	Support
0	0.95	1.00	0.97	357
1	0.99	0.91	0.95	219
Micro avg	0.96	0.96	0.96	576
Macro avg	0.97	0.95	0.96	576
Weighted avg	0.97	0.96	0.96	576

Table IV.3: Classification report of the Xception model

IV.7.3 Experiment 3: Resnet50

In the third experiment of our project, the model was trained using the first version of Resnet50 architecture. The architecture of the entire model is provided in Fig IV.10.



```
 conv5_block3_out (Activati  (None, 8, 8, 2048)           0         ['conv5_block3_add[0][0]']
 on)

 average_pooling2d (Average  (None, 2, 2, 2048)           0         ['conv5_block3_out[0][0]']
 Pooling2D)

 flatten (Flatten)           (None, 8192)                 0         ['average_pooling2d[0][0]']

 dense (Dense)               (None, 256)                  2097408   ['flatten[0][0]']

 dropout (Dropout)           (None, 256)                  0         ['dense[0][0]']

 dense_1 (Dense)             (None, 256)                  65792     ['dropout[0][0]']

 dropout_1 (Dropout)         (None, 256)                  0         ['dense_1[0][0]']

 dense_2 (Dense)             (None, 256)                  65792     ['dropout_1[0][0]']

 dropout_2 (Dropout)         (None, 256)                  0         ['dense_2[0][0]']

 dense_3 (Dense)             (None, 2)                    514       ['dropout_2[0][0]']

==================================================================================================
Total params: 25817218 (98.48 MB)
Trainable params: 25764098 (98.28 MB)
Non-trainable params: 53120 (207.50 KB)
```

Figure IV.10: The architecture of the model using Resnet50

The training and validation performance is shown in the graphs below:

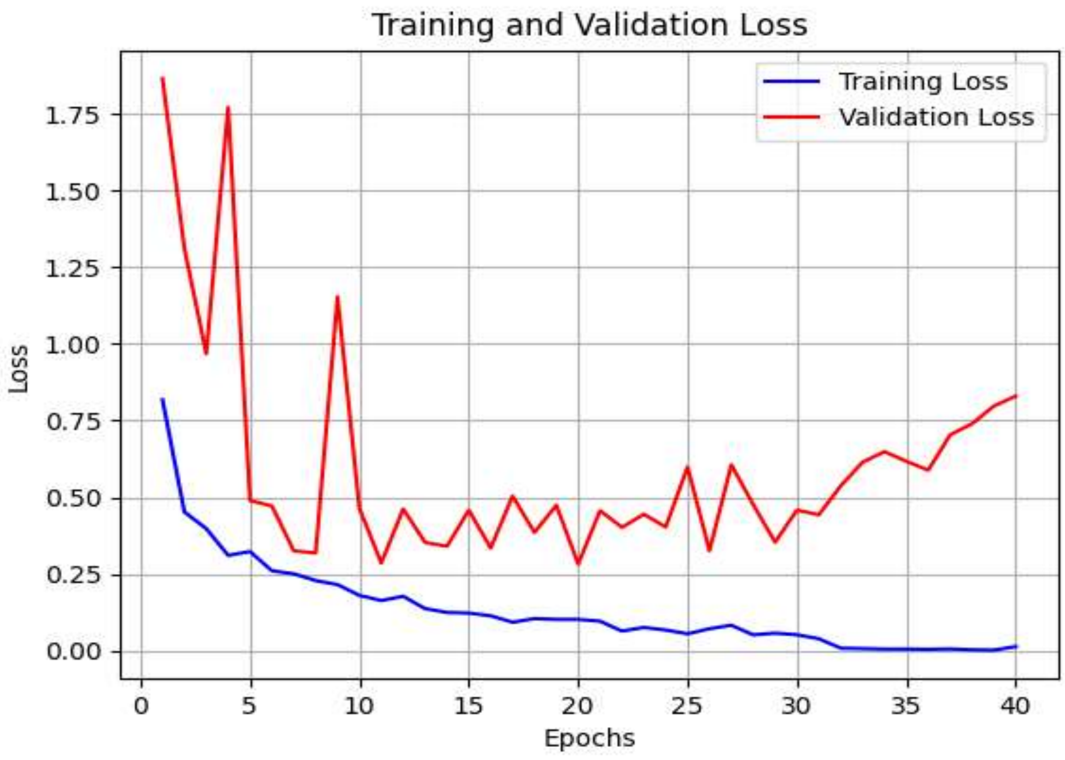


Figure IV.11: Resnet50 training and validation loss

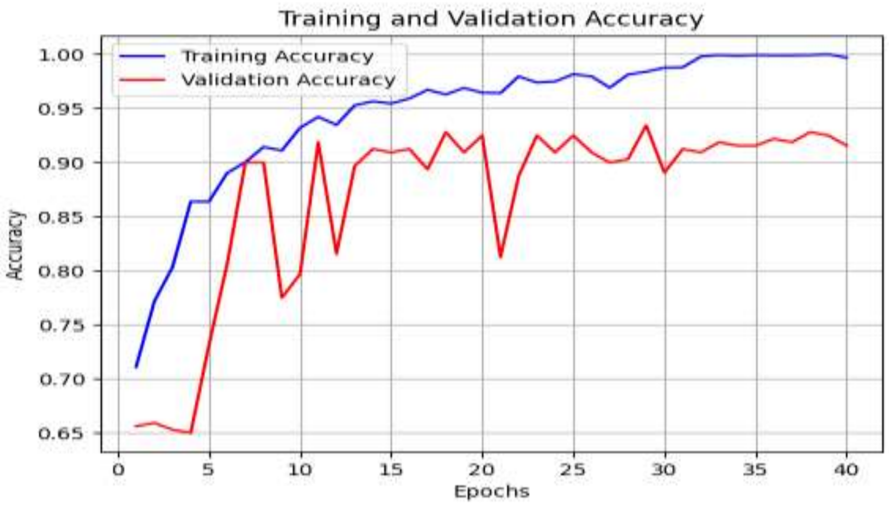


Figure IV.12: Resnet50 training and validation accuracy

The performance of this model is somehow similar to Xception. Likewise, we plotted the confusion matrix to better evaluate the model.

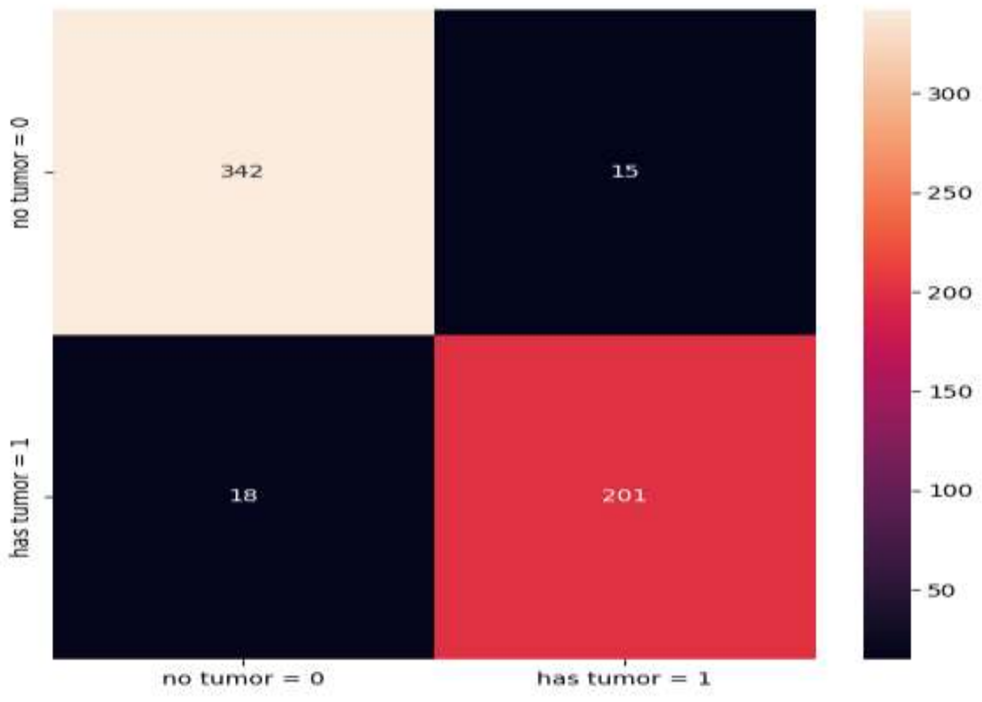


Figure IV.13: Confusion matrix of the Resnet50 Model

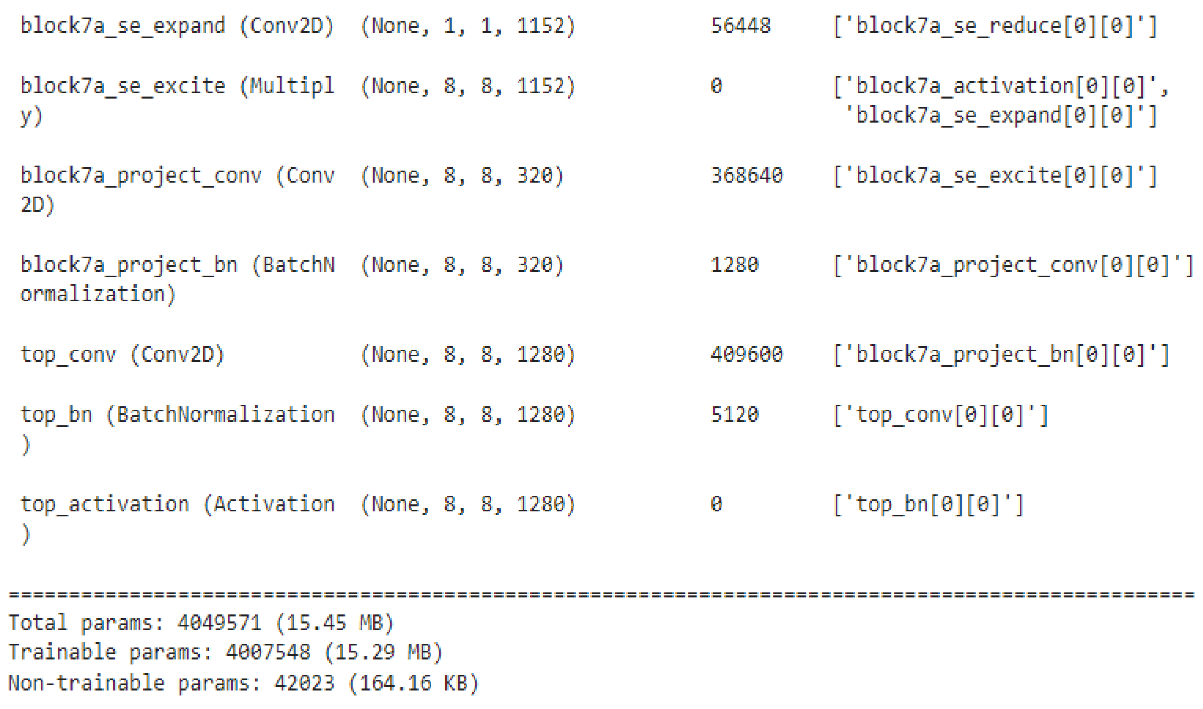
Classification report of Resnet50 model :

	Precision	Recall	f1-score	Support
0	0.95	0.96	0.95	357
1	0.93	0.92	0.92	219
Micro avg	0.94	0.94	0.94	576
Macro avg	0.94	0.94	0.94	576
Weighted avg	0.94	0.94	0.94	576

Table IV.4: Classification report of the Resnet50 model

IV.7.4 Experiment 4: EfficientB0

The last model of this project was trained using the EfficientB0 architecture. The architecture of the last model is provided in Fig IV.14.



```
 block7a_se_expand (Conv2D)  (None, 1, 1, 1152)          56448      ['block7a_se_reduce[0][0]']

 block7a_se_excite (Multipl  (None, 8, 8, 1152)          0          ['block7a_activation[0][0]',
 y)                                                                  'block7a_se_expand[0][0]']

 block7a_project_conv (Conv  (None, 8, 8, 320)           368640     ['block7a_se_excite[0][0]']
 2D)

 block7a_project_bn (BatchN  (None, 8, 8, 320)           1280       ['block7a_project_conv[0][0]']
 ormalization)

 top_conv (Conv2D)           (None, 8, 8, 1280)          409600     ['block7a_project_bn[0][0]']

 top_bn (BatchNormalization  (None, 8, 8, 1280)          5120       ['top_conv[0][0]']
 )

 top_activation (Activation  (None, 8, 8, 1280)          0          ['top_bn[0][0]']
 )

==================================================================================================
Total params: 4049571 (15.45 MB)
Trainable params: 4007548 (15.29 MB)
Non-trainable params: 42023 (164.16 KB)
```

Figure IV.14 The architecture of the model using Efficient B0

Training and validation accuracy and loss are shown below.

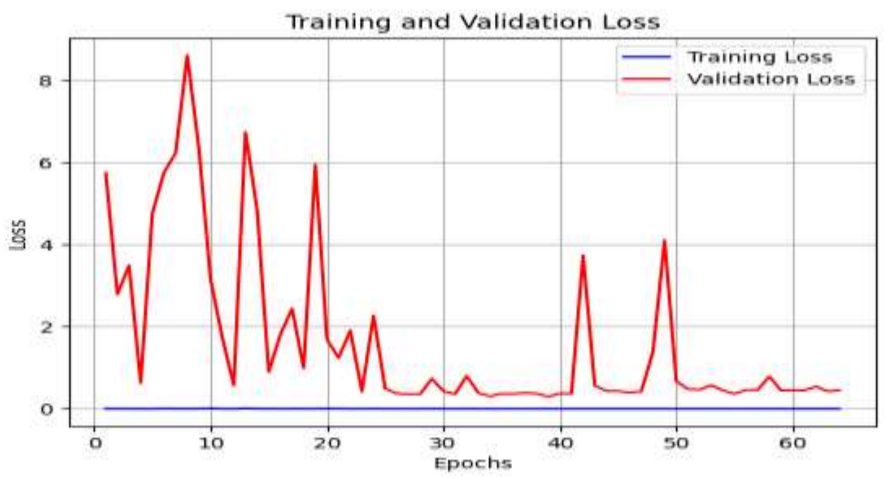


Figure IV.15: Training and Validation loss

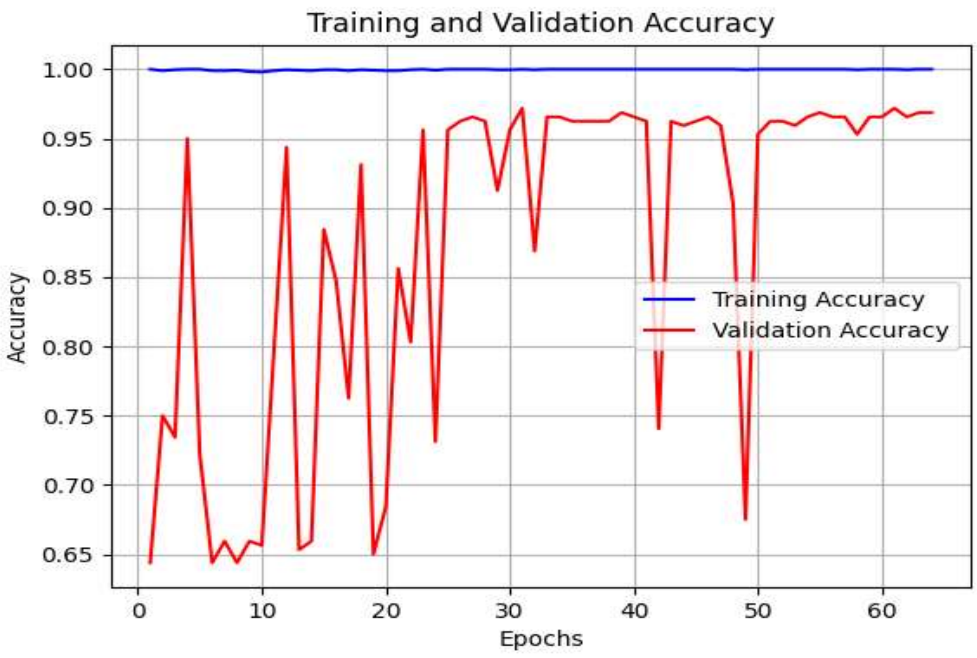


Figure IV.16: Training and Validation accuracy

In the last our model the validation loss remains relatively stable despite oscillations in the training loss, it suggests that our model is generalizing well to unseen data

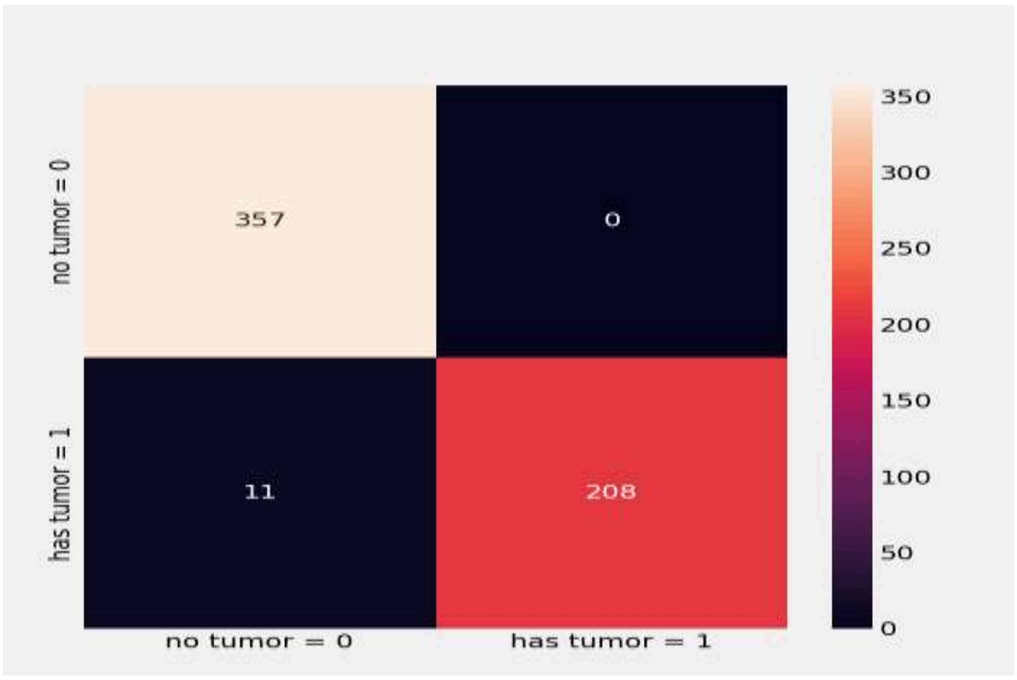


Figure IV.17 : Confusion matrix of the EfficientB0 Model

	Precision	**Recall**	**f1-score**	**Support**
0	0.96	0.99	0.98	357
1	0.98	0.94	0.96	219
Micro avg	0.97	0.97	0.97	576
Macro avg	0.97	0.97	0.97	576
Weighted avg	0.97	0.97	0.97	576

Table IV.5 : Classification report of the EfficientB0 model

IV.8 Summary of Results :

The following table summarizes the results obtained from the previous experiments with an overall comparison between the architectures

Architecture	Overall Accuracy	Balanced accuracy	Average PRE	Average REC	Average F1-Score	Model Size
VGG19	62%	50%	0%	0%	39%	109,14 MB
Xception	97%	96%	99%	91%	96%	87,40 MB
Resnet50	94%	94%	93%	92%	93,50%	98,48 MB
EfficientB0	98%	96%	98%	94%	97%	15,45 MB

Table IV.6: Summary of the results and an overall comparison between experiments

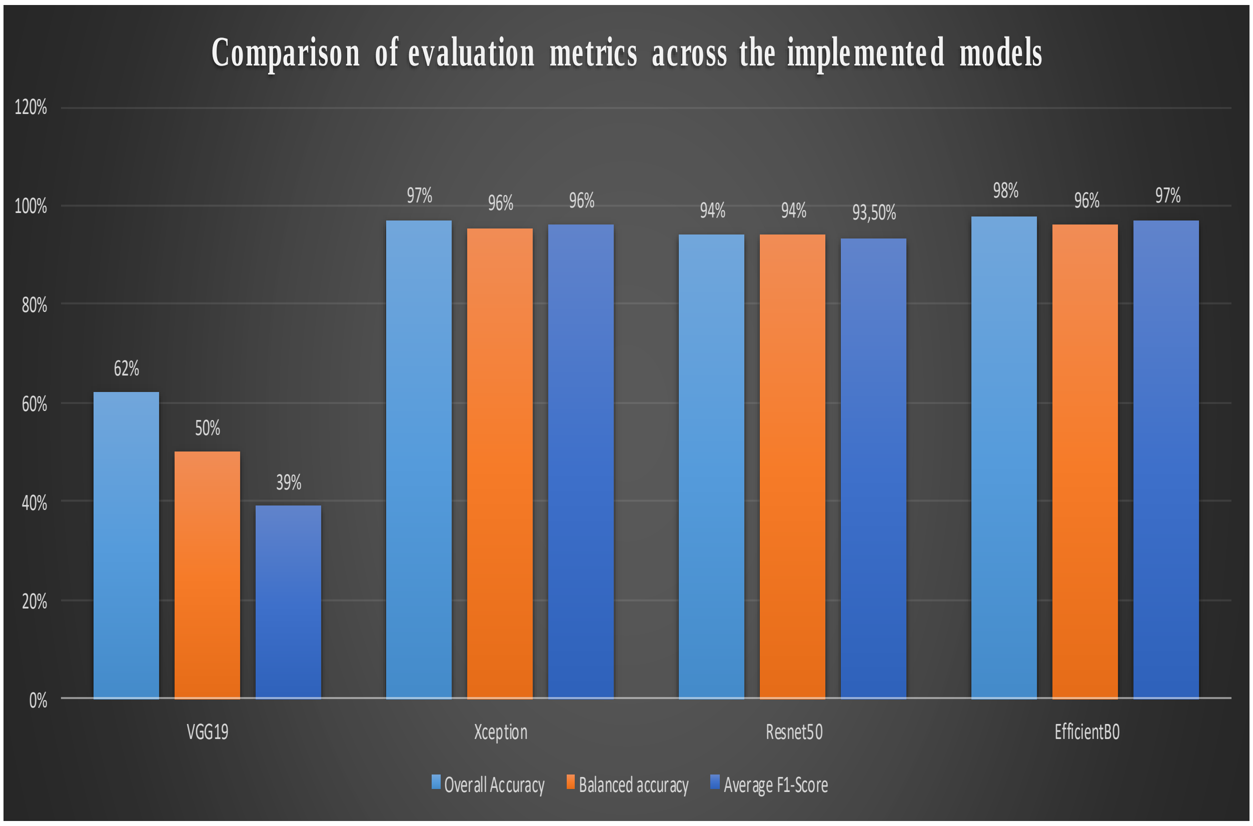


Figure IV.18: Comparison of evaluation metrics across the implemented models

In the following table, an overall comparison between our study and some of the related works is presented:

Study	**Method**	**Training Images**	**Testing Images**	**PRE**	**REC**	**F1-Score**	**ACC**
Jonayet M (2021)	**CNN+RBF**	1151	170	97,59%	89,28%	100%	97,59%
	CNN+Softmax			99.28%	99.27%	99.27%	99.28%
Jayanthi Vajiram (2020)	**Resnet50**	2839	240	95,09%	68,75%	99%	96%
	VGG16			89,61%	84,19%	97%	96%
	Resnet			97.11%	97.04%	97.07%	97.04%
	DNN			94,66 %	89,19%	98 %	97%
Rahul Sharma et al (2022)	**VGG16**	1651	412	98.76%	98.74%	98.74%	98.74%
	ResNet50			98.85%	98.84%	98.83%	98.84%
Our Study	**VGG19**	3390	566	62%	39%	50%	0%
	Xception			97%	96%	91%	96%
	EfficientB0			85%	92%	87%	86%
	Resnet50			94%	94%	93%	92%

Table IV.7: Comparison of deep learning methods for MRI Classification

IV.9 Tumor Localization

IV.9.1 Context and Experiment

ResUNet is an advanced deep learning model that combines the strengths of the U-Net architecture with residual connections. U-Net is renowned for its effectiveness in biomedical image segmentation due to its encoder-decoder structure, which allows for precise localization and context understanding. By integrating residual connections, ResUNet addresses the vanishing gradient problem and improves training efficiency, leading to better performance in segmentation tasks

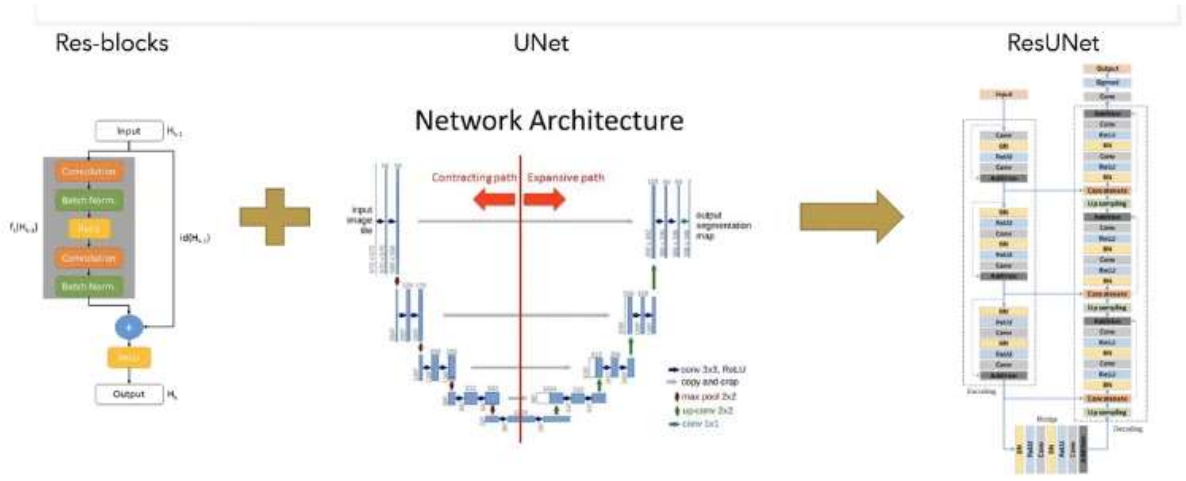


Figure IV.19 : UNet with Res-Blocks

The training and validation performance is shown in the graphs below:

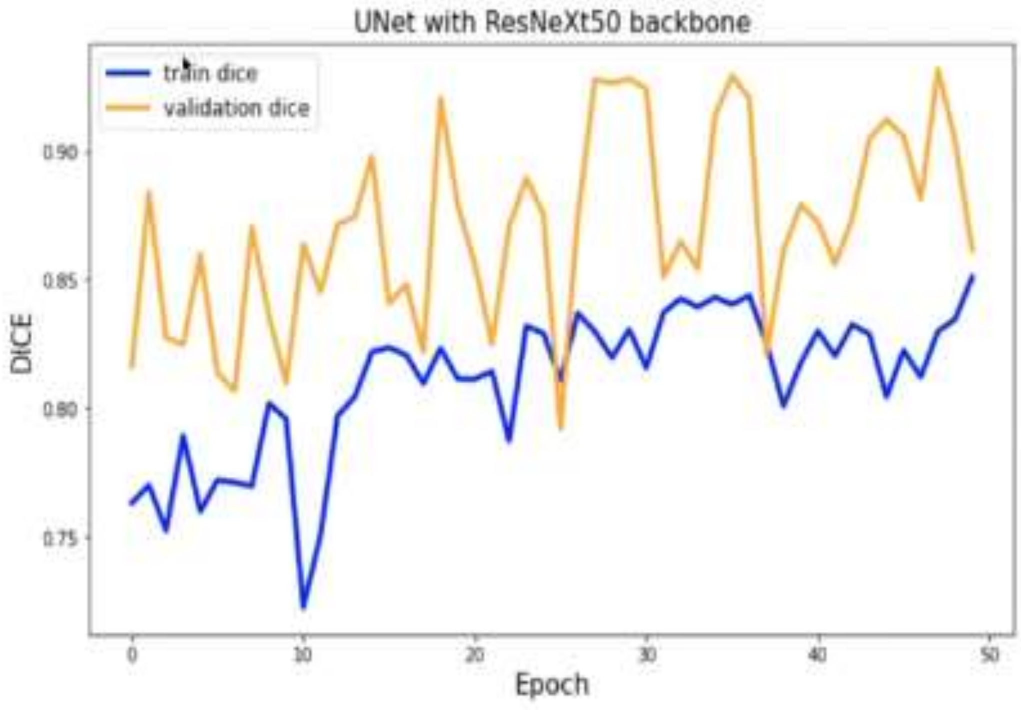


Figure IV.20 : Training and validation accuracy

The fluctuations in both the training and validation scores indicate variability in the model's performance per epoch. - The validation score being consistently higher than the training score could suggest that the model might not be overfitting. In a typical scenario, overfitting would show higher training scores compared to validation scores. - The consistent peaks and troughs in the validation score could indicate that the model's performance is sensitive to the specific batches of data used in each epoch, or there could

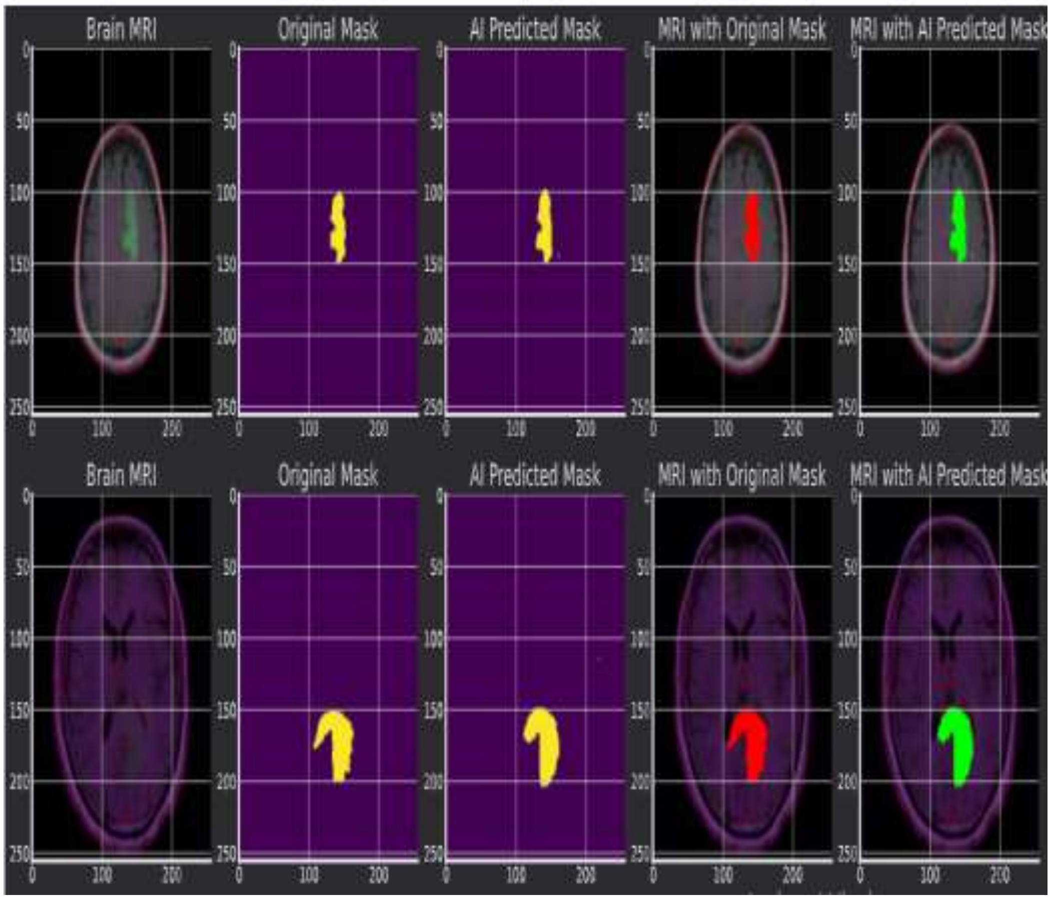


Figure IV.21 : Detection and Localization of Brain Tumor [61]

IV.9 Dicussion

Recently, there have been significant advancements in utilizing Deep Learning algorithms for the early detection of Tumor in human Brain. Our proposed methodology revolves around employing a deep-learning-based algorithm for diagnosing Brains diseases by tumor. We trained four models using the "Healthcare MRI for brain tumor " dataset from Kaggle, which comprises approximately 3929 images, and evaluated their performance on 560 images from the "MRI for Brain Tumor" dataset, also available on Kaggle. As anticipated, due to the extensive dataset used for training, the model training process took approximately 3 hours.

Upon individual evaluation of the models, it became apparent that the VGG19 model with high-size (109,14 MB) architecture performed the poorest compared to other architectures, yielding an overall accuracy of 62%, an overall loss of 50%, and a balanced accuracy of 50%. This discrepancy between balanced and overall accuracies suggests class imbalance issues. Conversely, the EfficientB0 architecture demonstrated superior performance metrics, striking a better balance with an overall accuracy of 98%, balanced accuracy of 96%, and an overall loss of 30%, albeit at the expense of larger model size.

Additionally, other architectures exhibited acceptable performance, maintaining a good balance between overall and balanced accuracy metrics relative to their model sizes. Ultimately, when considering deploying DL models on mobile applications, it is essential to weigh both performance and model size. Hence, we opted for Resnet50 and Xception due to its optimal performances across evaluation metrics while maintaining a manageable model size.

IV.10 Conclusion

In this section, diverse Deep Learning models were employed to categorize which human brain has a tumor or not. The assessment outcomes differed for each model. Consequently, it was observed that the VGG19 architecture exhibited the least satisfactory performance, whereas the Resnet50 and Xception models achieved a more equitable distribution across evaluation criteria. Nonetheless, we designated the EfficientB0 model as the optimal choice.

General conclusion

General Conclusion

Brain tumor have long been a significant concern in medicine for years. Precision healthcare has offered early disease identification and loss minimization by providing effective decisions based on DL methods. This work presented the design and implementation of a DL-powered for Brain tumor identification that allows fast and accurate diagnosis diseases in 2 categories. We used four different DL architectures (VGG19, Resnet50, EfficientB0, Xception) to train four CNN models using transfer learning using an imagery dataset of about 3929 pictures of healthy and diseased Brains. After evaluating the models, we found promising results except for some classes which had a poor evaluation performance compared to the other classes. To increase the system's usability, and using the Flutter framework.

We implemented the EfficientB0 model on the application as it showed the best assessment results compared with the remaining architectures. In less than a second, our algorithm could interpret most of other architectures.

This proves that our system is capable of doing real-time predictions by detection and localization a tumor while maintaining promising prediction accuracy and response time. However, because of the DL limitations,

The main contributions of this work can be summarized below:

➢ The choice of the best DL architectures and their corresponding hyperparameters that suit our field of application.

➢ Two asks of deep learning in the real time (Detection and Localization of tumor).

➢ Compensating for the lack of infrastructure medical instruments.

Bibliography

BIBLIOGRAPHY

[1]. N. Sayed, Y. Himeur, and F. Bensaali, "From time-series to 2d images for building occupancy prediction using deep transfer learning," Engineering Applications of Artificial Intelligence, vol. 119, p. 105786, 2023.

[2]. J. Ker, L. Wang, J. Rao, and T. Lim, "Deep Learning Applications in Medical Image Analysis," IEEE Access, vol. 6, pp. 9375–9389, 2018.

[3]. A. Copiaco, Y. Himeur, A. Amira, W. Mansoor, F. Fadli, S. Atalla, and S. S. Sohail, "An innovative deep anomaly detection of building energy consumption using energy time-series images," Engineering Applications of Artificial Intelligence, vol. 119, p. 105775, 2023.

[4]. Y. Habchi, Y. Himeur, H. Kheddar et al., "Ai in thyroid cancer diagnosis: Techniques, trends, and future directions," arXiv preprint, vol. arXiv:2308.13592, 2023.

[5]. Y.Fang , X.Xing , S.Wanga , S.Walsh, " Post-COVID Highlights: Challenges and Solutions of AI Techniques for Swift Identification of COVID-19"

[6]. Amin, J., Sharif, M., Yasmin, M. and Fernandes, S. L. (2020b). A distinctive approach in brain tumor detection and classification using MRI. Pattern Recognition Letters, 139 118–127.

[7]. Jonayet.M , Duc M Cao, Md Abu Sayed " Advancing Brain Tumor Detection: A Thorough Investigation of CNNs, Clustering, and SoftMax Classification in the Analysis of MRI Images.".

[8]. N. A. Samee et al., "Classification Framework for Medical Diagnosis of Brain Tumor with an Effective Hybrid Transfer Learning Model," Diagnostics, no. 10, p. 2541, Oct. 2022, doi: 10.3390/diagnostics12102541.

[9]. E. Irmak, "Multi-Classification of Brain Tumor MRI Images Using Deep Convolutional Neural Network with Fully Optimized Framework," Iranian Journal of Science and Technology, Transactions of Electrical Engineering, no. 3, pp. 1015–1036, Apr. 2021, doi: 10.1007/s40998-021-00426-9

[10]. J. Amin, M. A. Anjum, M. Sharif, S. Jabeen, S. Kadry, and P. Moreno Ger, "A New Model for Brain Tumor Detection Using Ensemble Transfer Learning and Quantum Variational Classifier," Computational Intelligence and Neuroscience, pp. 1–13, Apr. 2022, doi: 10.1155/2022/3236305.

[11]. E. Lynch, "What Is Deep Learning? A Guide to Deep Learning Use Cases, Applications, and Benefits," ClearML, Feb. 14, 2023. https://clear.ml/blog/what-is-deep-learning/ (accessed Jul. 25, 2023)

[12]. M. M. Taye, "Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions," Computers, no. 5, p. 91, Apr. 2023, doi: 10.3390/computers12050091.

[13]. J. Ker, L. Wang, J. Rao, and T. Lim, "Deep Learning Applications in Medical Image Analysis," IEEE Access, vol. 6, pp. 9375–9389, 2018.

[14]. P. Mlynarski, H. Delingette, A. Criminisi, and N. Ayache, "Deep learning with mixed supervision for brain tumor segmentation," J. Med. Imaging, vol. 6, no. 03,

p. 1, 2019.

[15]. C. G. Madamombe, “Deep Learning Techniques to Classify and Analyze Medical Imaging Data,” Int. J. Comput. Sci. Eng., vol. 7, no. 04, pp. 109–113, 2018.

[16]. M. Soltaninejad et al., “Supervised learning based multimodal MRI brain tumour segmentation using texture features from supervoxels,” Comput. Methods Programs Biomed., vol. 157, pp. 69–84, 2018.

[17]. M. Soltaninejad et al., “Automated brain tumour detection and segmentation using superpixel-based extremely randomized trees in FLAIR MRI,” Int. J. Comput. Assist. Radiol. Surg., vol. 12, no. 2, pp. 183–203, 2017.

[18]. Cheng Jun, “brain tumor dataset”, figshare. Dataset, 2017, Available: https://doi.org/10.6084/m9. figshare.1512427.v5

[19]. J. Cheng et al., “Enhanced Performance of Brain Tumor Classification via Tumor Region Augmentation and Partition,” PLoS One, vol. 10, no. 10, p. e0140381, Oct. 2015.

[20]. Z. N. K. Swati et al., “Content-Based Brain Tumor Retrieval for MR Images Using Transfer Learning,” IEEE Access, vol. 7, pp. 17809–17822, 2019.

[21]. M. M. Badža and M. C. Barjaktarović, “Classification of brain tumors from mri images using a convolutional neural network,” Appl. Sci., vol. 10, no. 6, 2020.

[22]. N. Noreen, S. Palaniappan, A. Qayyum, I. Ahmad, M. Imran, and M. Shoaib, “A Deep Learning Model Based on Concatenation Approach for the Diagnosis of Brain Tumor,” IEEE Access, vol. 8, pp. 55135–55144, 2020.

[23]. S. Tripathi, A. Verma, and N. Sharma, “Automatic segmentation of brain tumour in MR images using an enhanced deep learning approach,” Comput. Methods Biomech. Biomed. Eng. Imaging Vis., vol. 00, no. 00, pp. 1–10, 2020.

[24]. H. N. T. K. Kaldera, S. R. Gunasekara, and M. B. DIssanayake, “Brain tumor Classification and Segmentation using Faster R-CNN,” 2019 Adv. Sci. Eng. Technol. Int. Conf. ASET 2019, 2019.

[25]. H. N. T. K. Kaldera, S. R. Gunasekara and M. B. Dissanayake, "MRI based Glioma segmentation using Deep Learning algorithms," International Research Conference on Smart Computing and Systems Engineering (SCSE), Colombo, Sri Lanka, pp. 51-56, 2019.

[26]. Rafael C. Gonzalez , Richard E. Woods, Digital Image Processing, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001, pp 714-735 .

[27]. N. M. Dipu, S. A. Shohan and K. M. A. Salam, "Deep Learning Based Brain Tumor Detection and Classification," 2021 International Conference on Intelligent Technologies (CONIT), 2021, pp. 1-6, doi: 10.1109/CONIT51480.2021.9498384.

[28]. Dixit, A., Singh, P. (2023). Brain Tumor Detection Using Fine-Tuned YOLO Model with Transfer Learning. In: Gupta, M., Ghatak, S., Gupta, A., Mukherjee, A.L. (eds) Artificial Intelligence on Medical Data. Lecture Notes in Computational Vision and Biomechanics, vol 37. Springer, Singapore. https://doi.org/10.1007/978-981-19-0151- 5_30.

[29]. N. S. Kumar, A. K. Goel and S. Jayanthi, "A Scrupulous Approach to Perform Classification and Detection of Fetal Brain using Darknet YOLO v4," 2021

International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), 2021, pp. 578- 581, doi: 10.1109/ICACITE51222.2021.9404656.

[30]. M. M. Zahoor et al., “A New Deep Hybrid Boosted and Ensemble Learning-Based Brain Tumor Analysis Using MRI,” Sensors 2022, Vol. 22, Page 2726, vol. 22, no. 7, p. 2726, Apr. 2022, doi: 10.3390/S22072726.

[31]. S. H. Khan, A. Sohail, and A. Khan, “COVID-19 Detection in Chest X-Ray Images using a New Channel Boosted CNN,” arXiv. 2020.

[32]. I. Domingues, G. Pereira, P. Martins, H. Duarte, J. Santos, and P. H. Abreu, “Using deep learning techniques in medical imaging: a systematic review of applications on CT and PET,” Artif. Intell. Rev., 2020, doi: 10.1007/s10462-019-09788-3.

[33]. C. A. Goodfellow lan, Bengio Yoshua, “Deep Learning - Ian Goodfellow, Yoshua Bengio, Aaron Courville - Google Books,” MIT Press. p. 800, 2016.

[34]. W. Gómez-Flores et al., “A comparative study of pre-trained convolutional neural networks for semantic segmentation of breast tumors in ultrasound,” Comput. Biol. Med., vol. 126, no. January, pp. 2419–2428, Jun. 2020, doi: 10.1016/j.compbiomed.2020.104036.

[35]. P. Papanagiotou et al., “Convolutional Neural Network Techniques for Brain Tumor Classification (from 2015 to 2022): Review, Challenges, and Future Perspectives,” Diagnostics 2022, Vol. 12, Page 1850, vol. 12, no. 8, p. 1850, Jul. 2022, doi: 10.3390/DIAGNOSTICS12081850.

[36]. J. S. Paul, A. J. Plassard, B. A. Landman, and D. Fabbri, “Deep learning for brain tumor classification,” Med. Imaging 2017 Biomed. Appl. Mol. Struct. Funct. Imaging, vol. 10137, p. 1013710, Mar. 2017, doi: 10.1117/12.2254195.

[37]. S. Deepak and P. M. Ameer, “Brain tumor classification using deep CNN features via transfer learning,” Comput. Biol. Med., vol. 111, pp. 103345 %@ 0010–4825, 2019.

[38]. A. Çinar and M. Yildirim, “Detection of tumors on brain MRI images using the hybrid convolutional neural network architecture,” Med. Hypotheses, vol. 139, Jun. 2020, doi: 10.1016/j.mehy.2020.109684.

[39].] 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. 63536–69215, 2019.

[40]. S. Khawaldeh, U. Pervaiz, A. Rafiq, and R. S. Alkhawaldeh, “Noninvasive grading of glioma tumor using magnetic resonance imaging with convolutional neural networks,” Appl. Sci., vol. 8, no. 1, p. 27, 2018.

[41]. M. A. Khan et al., “Multimodal Brain Tumor Classification Using Deep Learning and Robust Feature Selection: A Machine Learning Application for Radiologists,” Diagnostics (Basel, Switzerland), vol. 10, no. 8, p. 565, 2020, doi: 10.3390/diagnostics10080565.

[42]. F. Özyurt, E. Sert, E. Avci, and E. Dogantekin, “Brain tumor detection based on Convolutional Neural Network with neutrosophic expert maximum fuzzy sure entropy,” Measurement, vol. 147, p. 106830, Dec. 2019, doi: 10.1016/J.MEASUREMENT.2019.07.058.

[43]. Z. Guo, X. Li, H. Huang, N. Guo, and Q. Li, “Deep learning-based image segmentation on multimodal medical imaging,” IEEE TRPMS, vol. 3, no. 2, pp. 162–169, 2019.

[44]. D. Zhang, G. Huang, Q. Zhang, J. Han, J. Han, and Y. Yu, “Crossmodality deep feature learning for brain tumor segmentation,” Pattern Recognition, vol. 110, p. 107562, 2021.

[45]. P. G. Brindha, M. Kavinraj, P. Manivasakam, and P. Prasanth, “Brain tumor detection from mri images using deep learning techniques,” in IOP Conference Series: Materials Science and Engineering, vol. 1055, no. 1. IOP Publishing, 2021, p. 012115.

[46]. M. Khairandish, M. Sharma, V. Jain, J. Chatterjee, and N. Jhanjhi, “A hybrid cnn-svm threshold segmentation approach for tumor detection and classification of mri brain images,” IRBM, 2021.

[47]. R. Hashemzehi, S. J. S. Mahdavi, M. Kheirabadi, and S. R. Kamel, “Detection of brain tumors from mri images base on deep learning using hybrid model cnn and nade,” biocybernetics and biomedical engineering, vol. 40, no. 3, pp. 1225–1232, 2020.

[48]. M. Siar and M. Teshnehlab, “Brain tumor detection using deep neural network and machine learning algorithm,” in 2019 9th International Conference on Computer and Knowledge Engineering (ICCKE). IEEE, 2019, pp. 363–368.

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

[50]. S. Irsheidat and R. Duwairi, “Brain tumor detection using artificial convolutional neural networks,” in 2020 11th International Conference on Information and Communication Systems (ICICS). IEEE, 2020, pp. 197–203.

[51]. Wikipedia, “Activation function”, [Online]. Available: https://en.wikipedia.org/wiki/Activation_function. [Accessed April 2022]

[52]. Kanchan Sarkar, “ReLU: Not a Differentiable Function: Why used in Gradient Based Optimization? and Other Generalizations of ReLU.”, [Online]. Available: https://medium.com/@kanchansarkar/relu-not-a-differentiable-function-why-used-in-gradientbased-optimization-7fef3a4cecec. [Accessed April 2022]

[53]. Sagar Sharma, “Activation Functions in Neural Networks”. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Accessed [April 2022]

[54]. Developers Breach, “Convolutional Neural Network (CNN)”. [Online]. Available: https://developersbreach.com/convolution-neural-network-deep-learning/. [Accessed May 2022]

[55]. Matthew Stewart, “Simple Introduction to Convolutional Neural Networks”. [Online]. Available: https://towardsdatascience.com/simple-introduction-to-convolutional-neuralnetworks-cdf8d3077bac. [Accessed May 2022]

[56]. James D. McCaffrey, “Convolution Image Size, Filter Size, Padding and Stride”. [Online].Available: https://jamesmccaffrey.wordpress.com/2018/05/30/convolution-imagesize-filter-size-padding-and-stride/. [Accessed May 2022]

[57]. Maruthi Srinivas Mudaragadda, “Max Pooling in Convolutional Neural Network and its Features”. [Online]. Available: https://analyticsindiamag.com/max-pooling-inconvolutional-neural-network-and-its-features/. [Accessed May 2022]

[58]. SuperDataScience Team, “Convolutional Neural Networks (CNN): Step 3 – Flattening”, [Online]. Available: https://www.superdatascience.com/blogs/convolutionalneural-networks-cnn-step-3-flattening. [Accessed May 2022]

[59]. M. Nielsen, “Neural Networks and Deep Learning”. Determination Press, 2015

[60]. Goodfellow, Ian, Yoshua, Bengio, Aaron, and Courville. “Deep learning Vol.1”. Cambridge: MIT Press, 2016.

[61]. Varshitha Gudimalla, “Concept of Gradient Descent in Machine Learning”. [Online]. Available: https://varshithagudimalla.medium.com/concept-of-gradient-descent-algorithm-inmachine-learning-44f587ac16ac. [Accessed May 2022]

[62]. Goodfellow, Ian, Bengio, Yoshua, Courville, Aaron, “6.5 Back-Propagation and Other Differentiation Algorithms”. Deep Learning. MIT Press (pp. 200-220), 2016.

[63]. Wilson, D. Randall, and Tony R. Martinez. "The general inefficiency of batch training for gradient descent learning." Neural networks 16, no. 10 (2003): 1429-1451