الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

UNIVERSITÉ BADJI MOKHTAR - ANNABA
BADJI MOKHTAR – ANNABA UNIVERSITY

جامعة باجي مختار – عنابـــــــــة

Faculté : TECHNOLOGIE
Département : ELECTRONIQUE.
Domaine : SCIENCES ET TECHNOLOGIES
Filière : Electronique.
Spécialité : Instrumentation.

Mémoire

Présenté en vue de l'obtention du Diplôme de Master

Thème :

Leveraging Squeeze-and-Excitation Networks for CT Scan Image Classification
Improving CT Scan Image Classification using Squeeze-and-Excitation Networks

Présenté par : ***Selougha Yahia et Khezzane Khalifa Mohamed Ali***
Encadrant : *Neili Zakaria MCB Université Badji Mokhtar Annaba*

Jury de Soutenance :

Dr Bouchareb	MCA	Université d'Annaba	Président
Neili Zakaria	MCB	Université d'Annaba	Encadrant
Dr Bensaoula	MCA	Université d'Annaba	Examinateur

Année Universitaire : 2023/2024

Acknowledgements

Above all, we thank Almighty Allah for giving us the strength and willpower to complete this modest work.

We would first like to thank our thesis supervisor, Dr. Zakaria Neili, for his guidance, patience and commitment. His enlightened advice and valuable suggestions enabled us to develop our work in the right.

Dedications

God be praised first of all.

we dedicate this modest work with great love, sincerity and pride:

to our dear parents, source of tenderness, nobility and affection. May this step be for you a reason for satisfaction.

to our siblings, as a token of brotherhood, with our wishes for happiness, health and success.

and to all the members of our family

to all our friends, all our loved ones

and to all those who compile this modest work.

Khalifa & Yahia

Abstract

Many modern methods have shown the benefit of enhancing spatial encryption by utilizing convolutional neural networks (CNNs). These methods extract features by combining spatial information and channel information within receptive fields to enhance the representational power of the network. Recently, significant progress has been made in classifying COVID-19 pneumonia images and many other health conditions using CNN models. However, in this work, we focus on a newly introduced concept by researchers known as the 'Squeeze-and-Excitation' (SE) block. This block adaptively recalibrates channel-level feature responses by explicitly modelling inter-channel relationships.

In this study, we will integrate this block into various models, including C-CNN, VGG16, and Alex-Net. It is crucial to study and compare the effectiveness of these models to facilitate the early detection and diagnosis of lung diseases using CT-Scan images. The study emphasizes the potential of CNN models with and without the SE block in assisting healthcare professionals in diagnosing and treating lung diseases. The main objective of our experiment is to train and compare regular models with those incorporating the SE block.

An improvement in accuracy by 2% and 1% was achieved in most models with the SE block, along with a reduction in error rates, with differences in some models reaching 6% compared to regular models. These results were obtained using data frequently employed by researchers.

Keywords: COVID-19, pneumonia, deep learning, CT-Scan images, VGG16, Alex-Net, used data, image classification, Convolutional Neural Network (CNN), Squeeze-and-Excitation block, SE-block

Résumé

De nombreuses méthodes modernes ont démontré l'intérêt de renforcer le chiffrement spatial en utilisant des réseaux de neurones convolutifs (CNN). Ces méthodes extraient des caractéristiques en combinant les informations spatiales et les informations de canal au sein des champs récepteurs afin d'améliorer la puissance de représentation du réseau. Récemment, des progrès significatifs ont été réalisés dans la classification des images de pneumonie COVID-19 et de nombreuses autres conditions de santé en utilisant des modèles CNN. Cependant, dans ce travail, nous nous concentrons sur un concept récemment introduit par des chercheurs, connu sous le nom de bloc 'Squeeze-and-Excitation' (SE). Ce bloc réétalonne de manière adaptative les réponses des caractéristiques au niveau des canaux en modélisant explicitement les relations inter-canaux.

Dans cette étude, nous intégrerons ce bloc dans divers modèles, y compris C-CNN, VGG16 et Alex-Net. Il est crucial d'étudier et de comparer l'efficacité de ces modèles pour faciliter la détection et le diagnostic précoces des maladies pulmonaires en utilisant des images CT-Scan. L'étude souligne le potentiel des modèles CNN avec et sans le bloc SE pour aider les professionnels de la santé à diagnostiquer et traiter les maladies pulmonaires. L'objectif principal de notre expérience est de former et de comparer les modèles réguliers avec ceux intégrant le bloc SE.

Une amélioration de la précision de 2% et 1% a été obtenue dans la plupart des modèles avec le bloc SE, ainsi qu'une réduction des taux d'erreur, avec des différences dans certains modèles atteignant 6% par rapport aux modèles réguliers. Ces résultats ont été obtenus en utilisant des données fréquemment utilisées par les chercheurs.

Mots-clés : COVID-19, pneumonie, apprentissage profond, images CT-Scan, VGG16, AlexNet, données utilisées, classification d'images, Réseau de Neurones Convolutifs (CNN), bloc Squeeze-and-Excitation, SE-block.

الملخص

أظهرت العديد من الأساليب الحديثة فائدة تعزيز التشفير المكاني، وذلك من خلال اعتماد الشبكات العصبية التلافيفة على عملية التلافيف من خلال استخراج ميزات عن طريق دمج المعلومات المكانية والمعلومات المتعلقة بالقناة معا داخل المجالات الاستقبالية، وذلك من أجل تعزيز القوة التمثلية للشبكة وكما نعلم أن في الآونة الأخيرة قد تم إحراز تقدم كبير في تصنيف صور الالتهاب الرئة كوفيد 19 COVID-19 والعديد من الحالات الصحية من خلال استخدام نماذج CNN لكن في هذا العمل نركز على علاقة جديدة طُرِحَت مؤخرا من قبل الباحثين، والتي يطلق عليها كتلة 'الضغط والإثارة' والتي تعمل على إعادة معايرة استجابات الميزات على مستوى القناة بشكل تكيفي من خلال نمذجة الترابط بين القنوات بشكل واضح. لذلك في هذه الدراسة سنقوم بإضافة هذه الكتلة إلى كل من النماذج C-CNN و VGG16 و Alex-Netومهم جدا دراسة ومقارنة فعالية هذه النماذج لتسهيل الكشف المبكر عن أمراض الرئة وتشخيصها باستخدام صور Ct-Scan. تؤكد الدراسة على إمكانات كل من نماذج CNNمع وبلا كتلة 'الضغط والإثارة SE-block 'في مساعدة المتخصصين في الرعاية الصحية في تشخيص وعلاج أمراض الرئة . الهدف الرئيسي من تجربتنا هو تدريب ومقارنة بين النماذج العادية والمضاف لها كتلة 'الضغط والإثارة'. تم الوصول إلى تحسن في دقة 2% و 1% في معظم النماذج المضاف إليها كتلة الضغط والإثارة وأيضا تحسين في نسبة الخطأ والذي وصل الفرق في بعض النماذج ب6% مقارنة مع النماذج العادية وهذه النتائج تم الحصول عليها بإستخدام ببيانات تم استخدمها من قبل الباحثين بكثرة .

الكلمات المفتاحية :COVID-19، الالتهاب الرئوي، التعلم العميق، صور CT-Scan، VGG16، Alex-Net، البيانات مستخدمة ، تصنيف الصور، الشبكة العصبية الالتفافية (CNN) ، كتلة ' الضغط والإثارة ' ، SE-block.

List of Figures

List of tables:

Contents:

Introduction

In recent years, advancements in medical imaging and machine learning techniques have driven the development of automated image classification systems to aid in the diagnosis and early detection of pneumonia. Convolutional Neural Networks (CNNs) have been a cornerstone in image classification tasks, demonstrating remarkable performance in various medical applications. CNNs benefit from hierarchical feature extraction and spatial relationships to achieve high accuracy in medical image analysis. Many recent approaches have emerged regarding the network's strength, and a group of researchers at the University of Oxford in the UK proposed a new architectural unit called the "Squeeze-and-Excitation" (SE) module. This module adaptively recalibrates channel feature responses by explicitly modeling inter-channel dependencies.

This study aims to integrate this unit into CNN models and compare them with standard models for classifying chest CT-Scan images of pneumonia, including COVID-19 cases, and healthy individuals. By evaluating and comparing the performance of these models, we seek to highlight their effectiveness and suitability in clinical settings. The investigation includes a variety of cases, including COVID-19, other types of pneumonia, and healthy individuals, to provide a comprehensive understanding of the models' capabilities. The results of this study have significant implications for the field of biomedical engineering and clinical practice. Enhancing the use of CNN models with the new architectural unit Senet can improve accuracy and efficiency in the early detection and diagnosis of pneumonia diseases, including COVID-19.

The findings of this research will inform healthcare professionals and decision-makers, enabling them to make informed choices regarding the implementation of advanced image classification models in clinical settings. Overall, this comparative study contributes to the growing body of knowledge in medical image analysis and highlights the potential of both CNN and CNN+SE models in assisting healthcare professionals in diagnosing and treating pneumonia diseases. By leveraging the power of deep learning and image classification techniques, we can pave the way for improved patient care, better resource allocation, and ultimately, positively impact public health outcomes.

Chapter 1: General Introduction

Chapter 1

Introduction

In the first chapter of the article, we introduce Ai and branches, especially Ml and Dl, and how we can use it easily through the programming languages, platforms, and libraries that helped us in the path of our research.

Artificial intelligence is a technical solution, system, or device that aims to imitate human intelligence to perform tasks while improving itself repeatedly based on the information it collects.

1.1 Artificial Intelligence

AI, or Artificial Intelligence, refers to the development of computer systems that can perform tasks that typically require human intelligence. These tasks include understanding natural language, recognizing patterns, learning from experience, and making decisions. AI technology enables machines to mimic human cognitive functions like reasoning, problem-solving, perception, and language understanding. AI applications range from voice assistants like Siri to self-driving cars and personalized recommendation systems [1].

AI applications span various industries and domains, including healthcare, finance, automotive, manufacturing, entertainment, and more. AI has the potential to revolutionize many aspects of our lives, improving efficiency, productivity, and decision-making across various fields. However, it also raises ethical, societal, and regulatory considerations, such as privacy, bias, transparency, and accountability.

Machine learning & Deep learning is a subset of artificial intelligence that focuses on building a software system that can learn or improve performance based on the data it consumes. This means that every machine learning solution is an AI solution, but not all AI solutions are machine learning solutions [2].

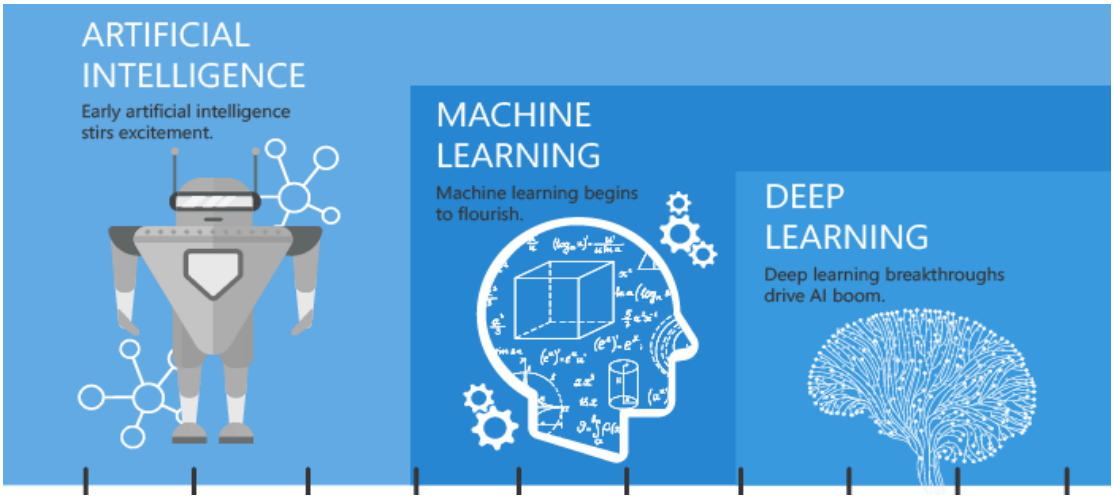


Figure 1: Artificial intelligence and branches.

1.2 Machine Learning

Machine Learning (ML) is a subset of Artificial Intelligence (AI) that focuses on creating systems capable of learning from data and improving their performance over time without being explicitly programmed. ML algorithms enable computers to identify patterns and make data-driven decisions or predictions. Examples of ML applications include recommendation systems, image recognition, spam detection, and medical diagnosis.

Machine learning structures typically start with collecting relevant data, which is then pre-processed to ensure it is usable by machine learning algorithms. Next, an appropriate model is selected and trained on the pre-processed data. The model's performance is evaluated using separate test data. Hyperparameters may be tuned to optimize performance. Once satisfactory, the model is deployed for real-world use. Ongoing monitoring and maintenance ensure the model's continued effectiveness and reliability [3] [4].

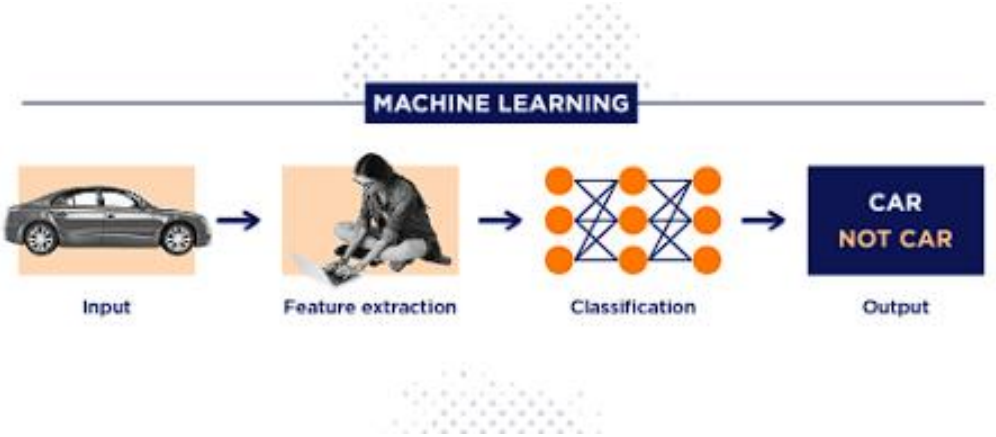


Figure 2: Machine Learning Structure

1.3 Deep Learning

Deep Learning (DL) is a subset of machine learning that utilizes artificial neural networks with multiple layers (hence "deep") to learn representations of data. Unlike traditional machine learning approaches, which may require manual feature extraction, deep learning algorithms automatically learn hierarchical representations of the input data. This makes deep learning particularly effective for tasks such as image and speech recognition, natural language processing, and recommendation systems. DL models, such as convolutional neural networks (CNNs) for image processing and recurrent neural networks (RNNs) for sequential data, have achieved remarkable performance in various domains, often surpassing human-level performance. Training deep learning models typically require large amounts of data and computational resources, but they have demonstrated state-of-the-art results in a wide range of applications [4].

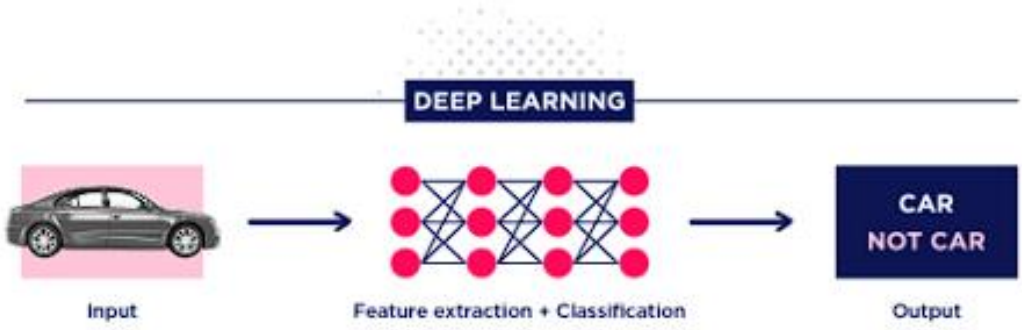


Figure 3: Deep Learning Structure

1.4 Programming Language Used

1.4.1 Python 

The Python language is an open source, cross-platform, object-oriented programming language. Thanks to specialized libraries, Python can be used for many situations such as software development, data analysis, or infrastructure management. It is therefore not, like HTML for example, solely dedicated to web programming.

An interpreted programming language, Python allows code to be executed on any computer. Usable by both beginners and expert programmers, Python allows you to create programs quickly and easily.

A language used for machine learning and data science; Python has significantly contributed to the growth of big data. Thanks to its numerous libraries such as Panda, Bokeh, NumPy, SciPy, Scrapy, Matplotlib, Scikit-Learn or even TensorFlow, Python offers great flexibility in the tasks to be carried out and great compatibility whatever the platform used [5].

1.5 Platform Used

1.5.1 Google Collab 

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.

Google Colab offers several benefits that make it a popular choice among data scientists, researchers, and machine learning practitioners. Key features of Google Collaboratory notebook include:

- **Free Access to GPUs:** Colab offers free GPU access, which is particularly useful for training machine learning models that require significant computational power.

- **No Setup Required:** Colab runs in the cloud, eliminating the need for users to set up and configure their own development environment. This makes it convenient for quick coding and collaboration.
- **Collaborative Editing:** Multiple users can work on the same Colab notebook simultaneously, making it a useful tool for collaborative projects.

Integration with Google Drive: Collab is integrated with Google Drive, allowing users to save their work directly to their Google Drive account. This enables easy sharing and access to notebooks from different devices.

a. Support for Popular Libraries: Colab comes pre-installed with many popular Python libraries for machine learning, data analysis, and visualization, such as TensorFlow, PyTorch, Matplotlib, and more.

b. Easy Sharing: Colab notebooks can be easily shared just like Google Docs or Sheets. Users can provide a link to the notebook, and others can view or edit the code in real-time [6].

1.5.2 Kaggle:

Kaggle is a platform for data science competitions, where data scientists and machine learning engineers can compete to create the best models for solving specific problems or analysing certain data sets. The platform also provides a community where users can collaborate on projects, share code and data sets, and learn from each other's work. Founded in 2010, Google acquired Kaggle in 2017, and the platform is now part of Google Cloud.

Kaggle hosts a variety of competitions sponsored by organizations, ranging from predicting medical outcomes to classifying images or identifying fraudulent transactions. Participants can submit their models and see how they perform on a public leaderboard, as well as receive feedback from other competitors and the community.

In addition to competitions, Kaggle also offers public data sets, machine learning notebooks, and tutorials to help users learn and practice their skills in data science and machine learning. It has become a popular platform for both novice and experienced data scientists to improve their skills, build their portfolios, and connect with others in the industry.

Kaggle is also used for:

Learning: Kaggle provides resources such as public data sets, machine learning tutorials, and code notebooks that allow users to learn and practice data science skills.

Collaboration: Kaggle allows users to form teams and collaborate on submissions, share code and data sets, and provide feedback to each other.

Community building: Kaggle has a large community of data scientists, machine learning engineers, and data enthusiasts, providing a platform for users to connect, share ideas, and collaborate on projects.

Research: Kaggle's data sets and competitions are impactful for research purposes, making it a platform for testing and improving machine learning algorithms.

Overall, Kaggle is a versatile platform that offers a range of opportunities for data scientists and machine learning engineers, from learning and collaboration to research [7].

1.6 Software Used

1.6.1 Anaconda:

Anaconda is a free distribution and open-source programming language in Python and R application to develop new applications in the science of donation and automatic application, which can simplify the delivery of packages and development [8].

1.6.2 Spyder:

Spyder is a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers, and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package [9].

1.7 Python Libraries Used

1.7.1 TensorFlow:

TensorFlow is an open-source machine learning framework developed by Google that is widely used for building and training deep learning models. It provides a comprehensive ecosystem of tools, libraries, and resources to support various machine learning tasks, including neural networks, natural language processing, computer vision, and reinforcement learning [10].

1.7.2 Keras:

Keras is an open-source neural network library written in Python. It is designed to be user-friendly, modular, and extensible, making it a popular choice for building and training deep learning models. Keras provides a high-level API that allows developers to quickly prototype and experiment with neural network architectures without needing to deal with the complexities of low-level implementations [10].

1.7.3 Numpy:

NumPy is a Python library used for working with arrays; It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open-source project, and you can use it freely.

NumPy stands for Numerical Python [11].

1.7.4 Sk-Learn:

Scikit-learn is a Python library that can be used automatically. It is developed by several contributing contributors at the Academy of French Superior Institutes and scholarships as Inria [12].

Chapter 2: Convolutional Neural Network

Chapter 2: Convolutional Neural Network (CNN)

2.1. Introduction

In the second chapter, we will discuss CNN, which is the primary tool for image and video recognition. We will provide you with a definition of its basic concepts and components, its structure, and its applications in computer vision. We will also examine all its layers, and through understanding how CNNs work, we can leverage their power to improve image classification.

2.2. Neural Network

A neural network is a computer algorithm inspired by the human brain. It has made up of interconnected nodes organized into layers. These networks can analyse data, recognize patterns, and make decisions. In applications like social media marketing, they are used for tasks like sentiment analysis, content recommendation, and customer segmentation, helping marketers understand their audience and improve their strategies.

Neural networks are typically trained using a dataset of input-output pairs, called a training set. During training, the network adjusts the weights of its connections between neurons to minimize the difference between the actual output and the desired output, once a neural network is trained, it can be used to make predictions or classifications on new input data [4].

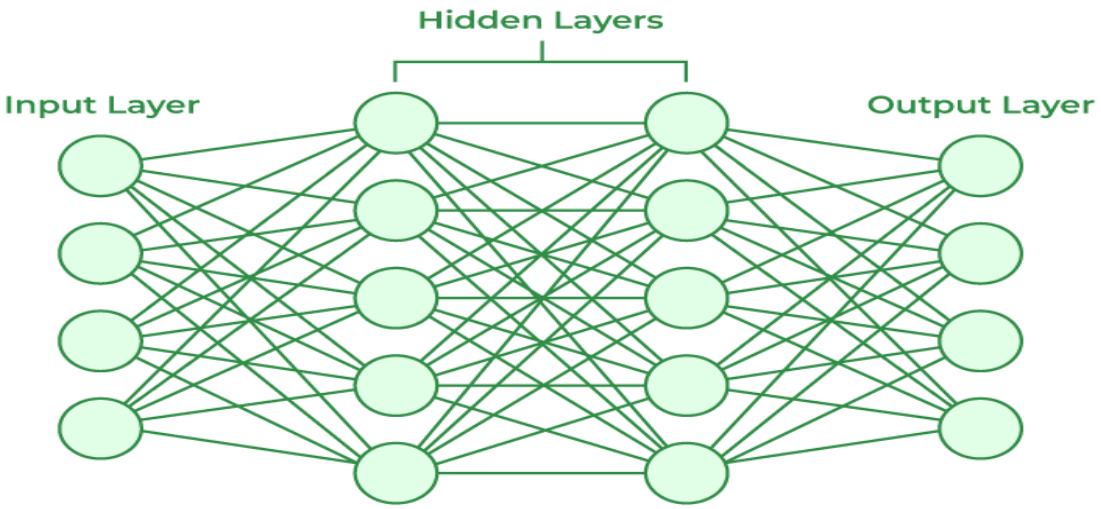


Figure 4: Neural Network Structure

2.3. Image Classification

Image classification using Convolutional Neural Networks (CNN) has revolutionized computer vision tasks by enabling automated and accurate recognition of objects within images. CNN-based image classification algorithms have gained immense popularity due to their ability to learn and extract intricate features from raw image data automatically. This article will explore the principles, techniques, and applications of image classification using CNNs. We will delve into the architecture, training process, and CNN image classification evaluation metrics. By understanding the workings of CNNs for image classification, we can unlock many possibilities for object recognition, scene understanding, and visual data analysis [13].

Image classification using CNN involves the extraction of features from the image to observe some patterns in the dataset. Using an ANN for the purpose of image classification would end up being very costly in terms of computation since the trainable parameters become extremely large.

We use filters when using CNNs. Filters exist of many different types according to their purpose [14].

It has a wide range of practical applications across various domains, including scene understanding, defect detection in manufacturing processes, facial recognition for security purposes, and disease diagnosis through medical imaging, such as X-ray or CT-scan images. [12] In our research study, we aim to perform image classification tasks with multiple classes, specifically focusing on pneumonia diseases, including COVID-19.

By addressing the image classification task involving these specific classes, we aim to contribute to the scientific understanding and development of accurate diagnostic models for pneumonia diseases, including COVID-19.

By automating the process of image classification, it is possible to analyse large volumes of images quickly and accurately, allowing for more efficient and effective decision-making in various industries [15] [16].

2.4. Convolutional Neural Network (CNN)

Chapter 2

A Convolutional Neural Network (CNN) is a specialized type of artificial neural network designed to analyse visual data such as images and videos. CNNs are particularly powerful in tasks like image classification, object detection, and image segmentation.

During training, CNNs learn to automatically extract relevant features from raw pixel data, gradually improving their ability to classify images accurately. This process is often supervised, meaning that the network is trained on labelled data pairs (input images and their corresponding labels) [4].

CNNs have revolutionized the field of computer vision and have been instrumental in achieving state-of-the-art performance in various visual recognition tasks, including image classification, object detection, facial recognition, and medical image analysis [17].

A Convolutional Neural Network (CNN) comprises several key components that work together to process and analyse visual data like images. These components include:

- Input layer.
- convolutional (Conv) layer.
- Pooling layer.
- Fully connected (FC) layer.
- SoftMax.
- Output layer.

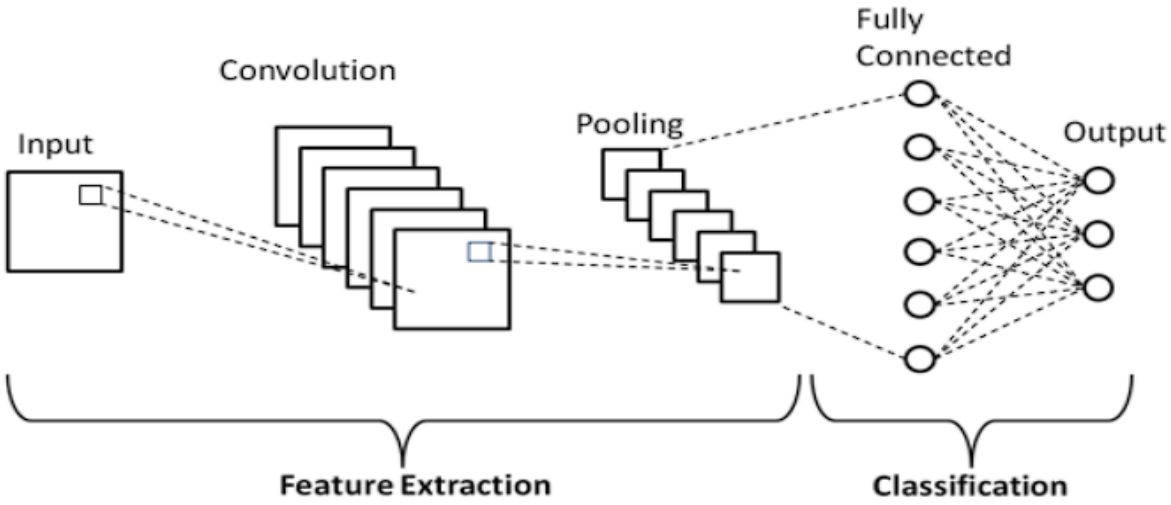


Figure 5: Convolutional Neural Network Model

2.4.1. Input layer

The input layer is the initial stage of a Convolutional Neural Network (CNN) where raw data is fed into the network for processing. In the context of image classification, the input layer receives raw pixel values from the input image.

The input layer serves as the starting point for the flow of information through the network. As the data progresses through subsequent layers, including convolutional layers, pooling layers, and fully connected layers, the network learns hierarchical representations of the input data, enabling tasks such as image classification, object detection, and segmentation.

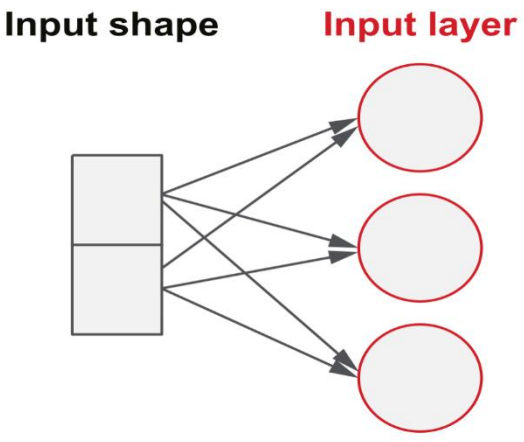


Figure 6: input layer structure

2.4.2. Convolutional (Conv) layer

The convolutional layer is a fundamental building block of a Convolutional Neural Network (CNN) responsible for extracting features from input data, such as images. Here is a detailed explanation of the convolutional layer:

a. Convolution Operation

The core operation of the convolutional layer is convolution. It involves sliding a small window called a filter (or kernel) over the input data (e.g., an image) and performing element-wise multiplication between the filter and the overlapping region of the input. The result is a feature map that highlights spatial patterns and structures in the input.

b. Filters and Feature Maps

A convolutional layer typically consists of multiple filters, each responsible for detecting different features. As the filters slide over the input, they produce corresponding feature maps that capture different aspects of the input data. These feature maps represent learned representations of local patterns, such as edges, textures, or shapes.

c. Shared Weights and Bias

In CNNs, the same filter is applied across the entire input image, allowing the network to learn spatially invariant features. Each filter has its set of learnable parameters, including weights and a bias term, which are adjusted during the training process to minimize the prediction error.

d. Stride and Padding

The stride determines the step size of the filter as it moves across the input. A larger stride reduces the spatial dimensions of the output feature maps, while padding can be added to the input to preserve spatial dimensions. Padding is often used to ensure that the output feature maps have the same spatial dimensions as the input.

e. Non-linear Activation

After the convolution operation, a non-linear activation function, such as ReLU (Rectified Linear Unit), is applied elementwise to the feature maps. This introduces non-linearity into the network, enabling it to learn complex relationships and represent more abstract features.

The convolutional layer plays a crucial role in capturing hierarchical representations of the input data, enabling the network to learn increasingly abstract features as information flows through subsequent layers. It forms the backbone of CNNs and is essential for tasks such as image classification, object detection, and image segmentation.

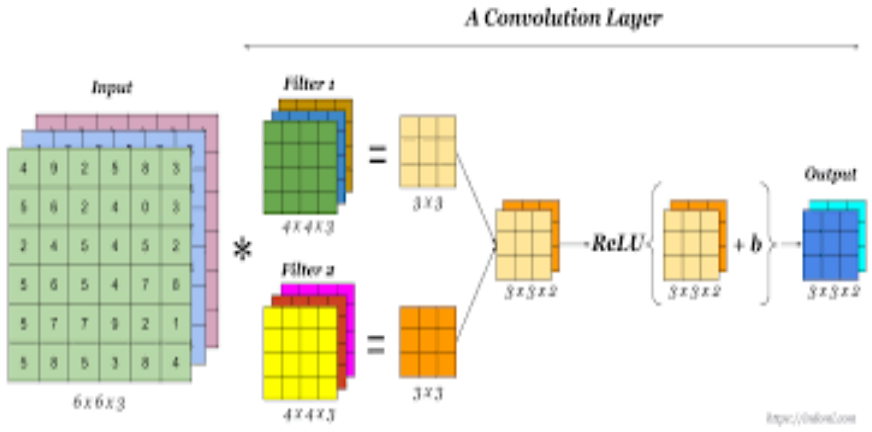


Figure 7: convolution layer

2.4.3. pooling layer

The pooling layer is a crucial component in Convolutional Neural Networks (CNNs) designed to reduce the spatial dimensions of the feature maps generated by the convolutional layers while retaining important information. Here is an overview of the pooling layer:

a. Down sampling

The primary function of the pooling layer is down sampling, which reduces the spatial dimensions (width and height) of the input feature maps. This helps in decreasing the computational complexity of the network and controlling overfitting by providing a form of spatial abstraction.

b. Pooling Operations

The pooling layer performs pooling operations over small spatial regions of the input feature maps. The two most common pooling operations are max pooling and average pooling:

- **Max Pooling:** For each region of the input feature map, the maximum value is retained, discarding the rest. This helps in preserving the most prominent features in each region.
- **Average Pooling:** The average value of the region is computed and retained. This operation can help in reducing noise and capturing more generalized features.

Pooling Size and Stride: The pooling layer operates with a specified pooling size (e.g., 2x2 or 3x3) and a stride that determines the step size for moving the pooling window across the input feature maps. By adjusting the pooling size and stride, the degree of down sampling can be controlled [17].

2.4.4. Fully connected (FC) layer:

The fully connected (FC) layer, also known as the dense layer, is a crucial component in Convolutional Neural Networks (CNNs) that plays a key role in high-level feature representation and classification. Here is an overview of the fully connected layer [17]:

a. Flattening

Before the fully connected layer, the feature maps generated by the convolutional and pooling layers are typically flattened into a one-dimensional vector. This process converts the spatially arranged features into a linear sequence, allowing them to be fed into the fully connected layer.

b. Connection to Neurons

In a fully connected layer, each neuron is connected to every neuron in the preceding layer. This means that every element in the flattened feature vector is connected to every neuron in the fully connected layer.

2.4.5. Learnable Parameters

Each connection between neurons in the flattened input vector and neurons in the fully connected layer has its associated weight parameter. During training, these weights are learned through backpropagation, adjusting their values to minimize the loss function and improve the network's performance on the given task.

2.4.6. Bias Term

In addition to weights, each neuron in the fully connected layer also has a bias term associated with it. The bias term provides the model with additional flexibility to learn complex patterns and relationships in the data.

2.4.7. Activation Function

Like other layers in the network, the fully connected layer typically applies a non-linear activation function to the output of each neuron. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, or tanh, introducing non-linearity into the model and enabling it to learn complex mappings between input and output.

2.4.8. Classification or Regression

The fully connected layer is often the final layer in a CNN architecture and is responsible for generating the output predictions. Depending on the task, such as image classification or regression, the fully connected layer may have a different number of neurons corresponding to the number of classes or output dimensions.

Overall, the fully connected layer in CNNs serves as a powerful tool for learning high-level representations of the input data and making predictions based on these representations, making it a critical component in various machine learning tasks.

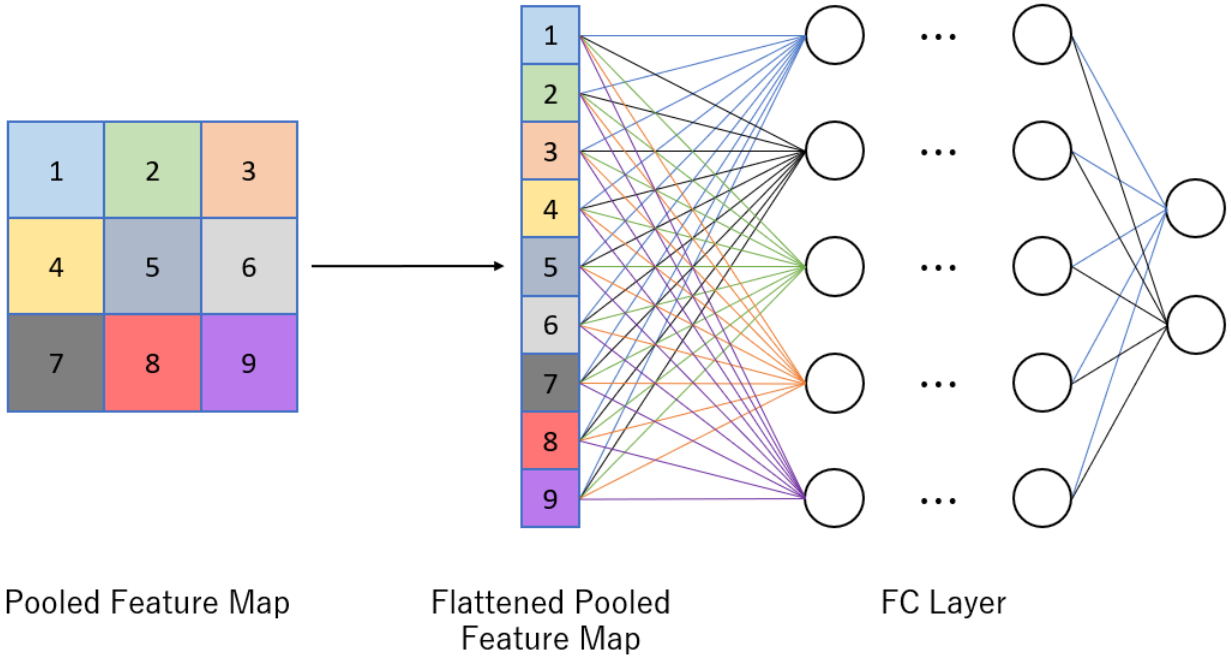


Figure 8: (FC) layer structure

2.4.9. SoftMax

SoftMax is an activation function commonly used in the output layer of neural networks, including Convolutional Neural Networks (CNNs), for multi-class classification tasks. Here is an explanation of SoftMax:

2.4.10. Probability Distribution

SoftMax converts the raw output scores from the previous layer into a probability distribution over multiple class. It ensures that the sum of the probabilities for all classes equals one, making it suitable for multi-class classification problems.

2.4.11 Properties

- SoftMax ensures that the output probabilities are non-negative.

- It emphasizes the high-scoring classes while suppressing the low-scoring ones, effectively highlighting the class predictions.

- SoftMax is differentiable, making it suitable for training neural networks using gradient-based optimization algorithms like backpropagation.

2.4.12. Output Interpretation

After applying SoftMax, the output of the neural network represents the probability of each class. The class with the highest probability is considered the predicted class for the input.

2.4.13. Cross-Entropy Loss

SoftMax is often paired with the cross-entropy loss function for training neural networks. Cross-entropy loss measures the difference between the predicted probability distribution and the true distribution of class labels, guiding the network to minimize this difference during training.

In summary, SoftMax is a valuable component in CNNs for converting raw output scores into meaningful probabilities, enabling the network to make confident predictions for multi-class classification tasks.

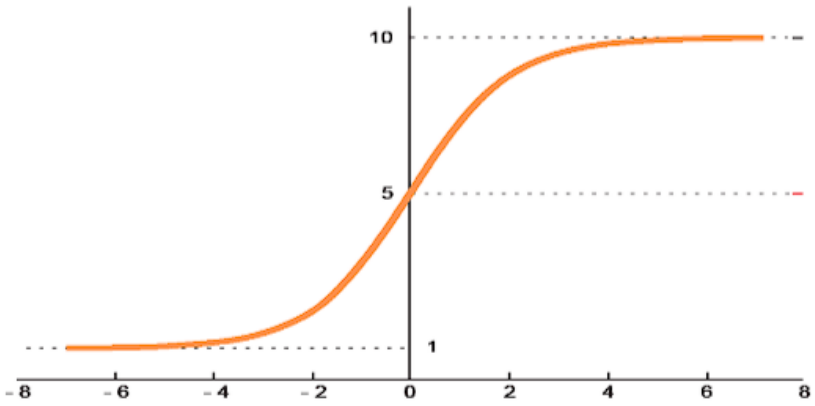


Figure 9: The shape of the SoftMax function

2.4.14. Output layer

The output layer is the final layer in a neural network architecture, including Convolutional Neural Networks (CNNs), responsible for producing the network's predictions or outputs based on the input data.

2.4.15. Activation Function

The activation function applied in the output layer depends on the task and the desired properties of the output. For binary classification tasks, a sigmoid activation function is commonly used to produce outputs in the range [0, 1], representing the probability of belonging to one class. For multi-class classification tasks, a SoftMax activation function is often applied to generate a probability distribution over multiple classes, ensuring that the sum of the probabilities equals one.

2.4.16. Loss Function

The choice of loss function in the output layer is critical for training the neural network. For classification tasks, cross-entropy loss is commonly used with SoftMax activation, measuring the discrepancy between the predicted probabilities and the true class labels. For regression tasks, mean squared error (MSE) or other regression-specific loss functions are used to measure the difference between the predicted and true continuous values.

2.4.17. Training and Inference

During training, the neural network adjusts its parameters (weights and biases) based on the chosen loss function and optimization algorithm to minimize prediction errors. During inference (testing or deployment), the trained network uses the learned parameters to generate predictions for new, unseen data.

2.4.18 Hyperparameters

Hyperparameters are settings or configurations that are set before training a machine learning model and cannot be learned from the data itself. They control the overall behaviour of the algorithm during training, affecting factors like model complexity, learning speed, and generalization performance. Common hyperparameters include the learning rate, batch size, number of layers, number of neurons per layer, and activation functions. Fine-tuning hyperparameters are crucial for optimizing model performance and achieving the best results.

a. Learning Rate

The learning rate is a hyperparameter in machine learning algorithms, determining the step size of parameter updates during training. It influences how quickly or slowly a model learns from the data. A higher learning rate accelerates learning but may cause instability or overshooting, while a lower learning rate may lead to slow convergence or getting stuck in local minima. Finding the right balance for the learning rate is crucial for effective model training and optimization.

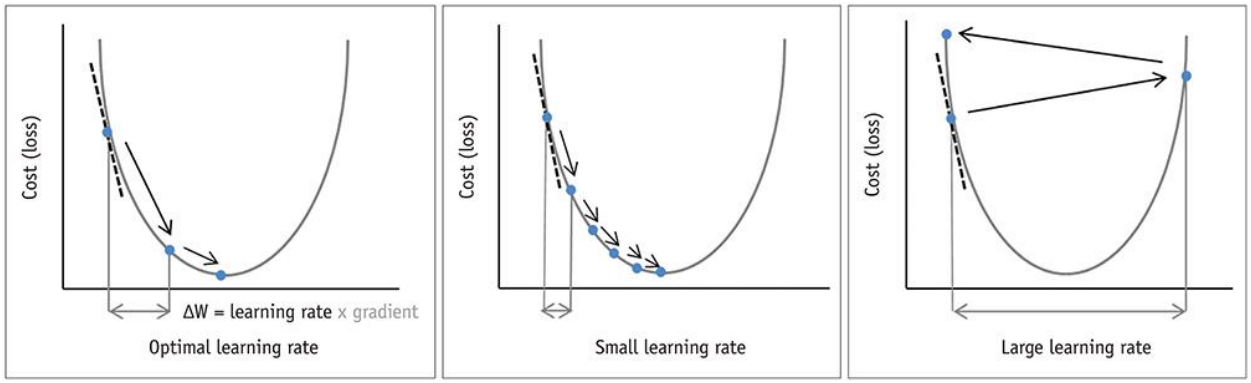


Figure 10: The effect of choosing different learning rates

c. Batch Size

The batch size is a hyperparameter in machine learning that specifies the number of training examples used in one iteration of model training. It affects the speed and stability of training, as well as the memory requirements. A larger batch size leads to faster training but requires more memory and may result in less noisy updates to the model's parameters. Conversely, a smaller batch size consumes less memory but may lead to slower training and more fluctuation in the optimization process. Choosing an appropriate batch size is important for achieving efficient and effective model training.

d. Epochs

Epochs refer to the number of times the entire dataset is passed forward and backward through the neural network during training. Each epoch consists of one forward pass (computing loss) and one backward pass (updating parameters) for all training examples. Increasing the number of epochs allows the model to see the data multiple times, potentially improving its ability to learn from the dataset. However, too many epochs can lead to overfitting, where the model memorizes the training data instead of learning general patterns. Finding the right balance between underfitting and overfitting by adjusting the number of epochs is essential for effective model training.

2.5. CNN Models

2.5.1. VGG16

The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model’s architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity

compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition in computer vision where teams tackle tasks including object localization and image classification. VGG16, proposed by Karen Simonyan and Andrew Zisserman in 2014, achieved top ranks in both tasks, detecting objects from 200 classes and classifying images into 1000 categories [18].

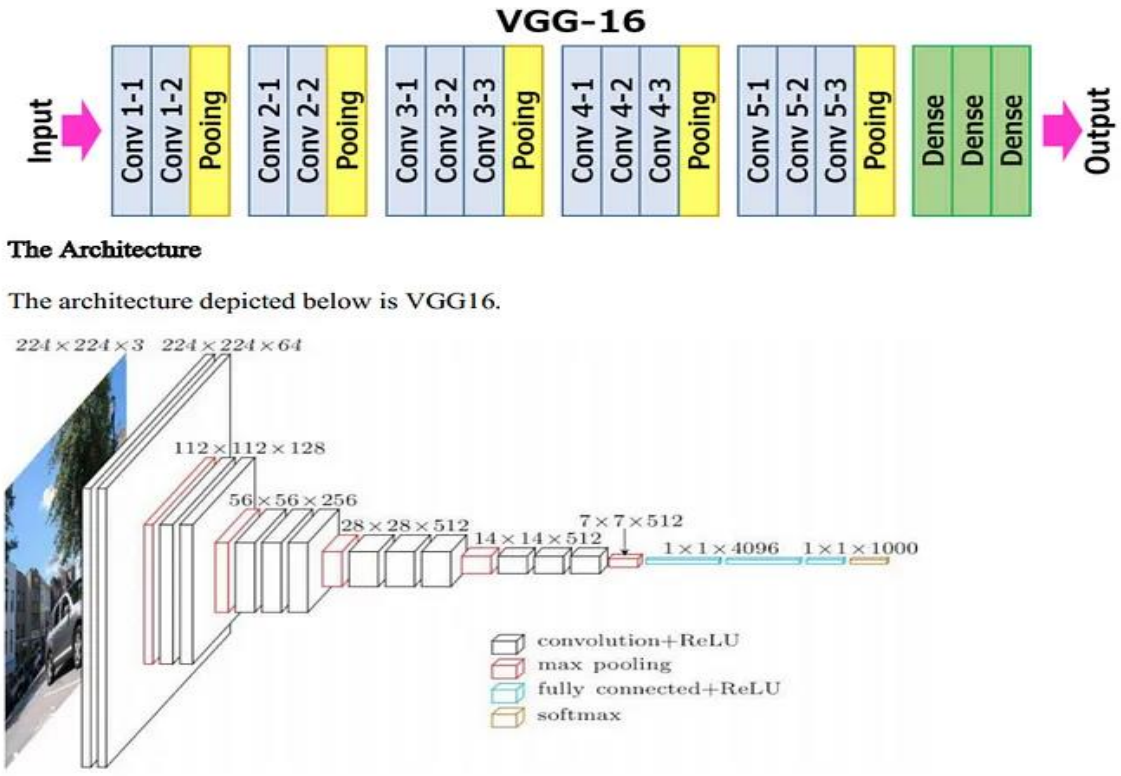


Figure 11: VGG-16 model.

2.5.2. Alxenet

AlexNet is a pioneering deep convolutional neural network architecture that played a significant role in advancing the field of computer vision. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, demonstrating a significant improvement in image classification accuracy over previous methods.

AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers.

AlexNet's success was attributed to several key innovations, including the use of rectified linear units (ReLU) for activation functions, overlapping pooling, dropout regularization, and the utilization of multiple GPUs for training, which significantly accelerated the learning process. These advances helped pave the way for the widespread adoption of deep convolutional neural networks in various computer vision tasks.

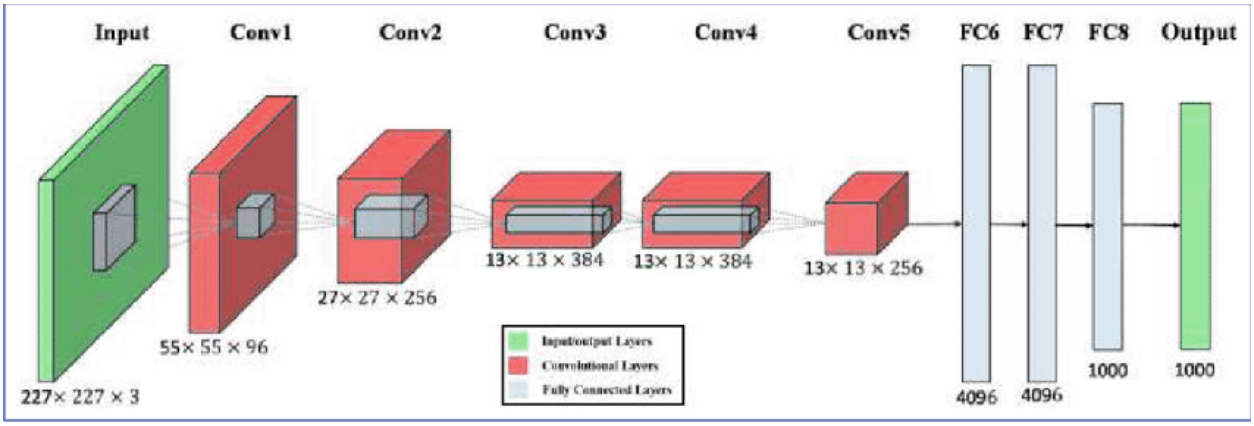


Figure 12: AlexNet model

2.6. Conclusion

In Chapter 2 of our essay, we delve into the significance of Convolutional Neural Networks (CNNs) in computer vision. We discuss their fundamental principles, including composition, The ability to learn complex patterns, and the training process. Emphasis is placed on image Classification as a key application of CNNs, enabling accurate classification based on visual features.

Chapter 3: SENet — Squeeze-and-Excitation Network

Chapter 3

3.1. Introduction:

In this section, we will give you a simple explanation of a novel architectural unit, we demonstrate that by stacking these blocks together, we can construct SENet architectures that generalize extremely well across challenging datasets.

Crucially, we find that SE blocks produce significant performance improvements for existing state-of-the-art deep architectures at minimal additional computational cost.

All this information was taken from the article, and we enhanced it and worked on it.

3.2. squeeze and excitation network:

The Squeeze and Excitation Network introduces a novel channel-wise attention mechanism for CNNs (Convolutional Neural Network) to improve their channel interdependencies. The network adds a parameter that re-weights each channel accordingly so that it becomes more sensitive towards significant features while ignoring the irrelevant features [19].

3.3. The architecture of Squeeze and Excitation Networks:

The author proposes an easy-use module called Squeeze and Excitation block also called SE-block. The SE-block consists of three operations:

- Squeeze
- Excitation
- Scaling

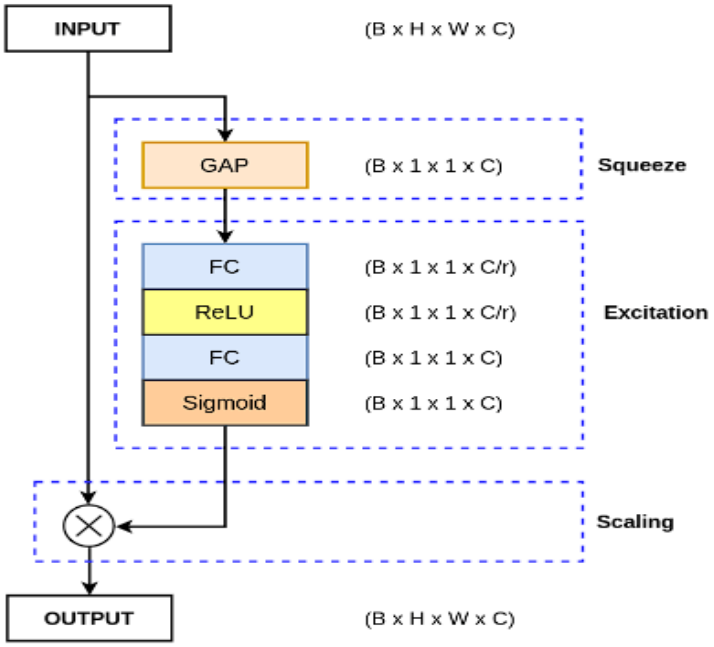


Figure 13: SE-Blocks Structure

3.3.1. Squeeze:

The squeeze operation is mainly used to extract the global information from each channel of the feature map. The feature map is basically the output of the convolution layer, which is a 4D tensor of size B x H x W x C. Here:

B: refers to batch size.

H: refers to the height of each feature map.

W: refers to the width of each feature map.

C: refers to the number of channels in the feature map.

In modern convolutional neural networks, pooling operations are used to reduce the spatial dimensions of the feature maps. The two widely used pooling operations are:

Max Pooling: operation is used to take the maximum pixel value from a defined window.

Average Pooling: operation is used to computes the average pixel values from a defined window [19].

3.2.2. Excitation:

The feature map is now reduced to a smaller dimension (B x 1 x 1 x C), for each channel of size H x W is reduced to a singular vector. For the excitation operation, a fully connected multi-layer perceptron (MLP) with a bottleneck structure is used. The MLP is used to generate the weights to scale each channel of the feature map adaptively [19].

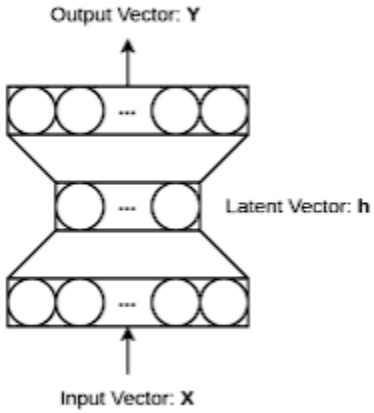


Figure 14: ***Example of a Multi-Layer Perceptron (MLP) structure***.

The MLP consists of three layers, where the hidden layer is used to reduce the number of features by a reduction factor r. The dimensions of the feature maps in the layers are:

The input is of shape (B x 1 x 1 x C), which is reduced to B x C. Thus, the input layer has the C number of neurons.

The hidden layer reduces the number of neurons by a factor of r. Thus, the hidden layer has a C/r number of neurons.

Finally, In the output layer, the number of neurons increased back to C.

Overall, the MLP takes the input as B x 1 x 1 x C as return the output with the same dimensions.

3.2.3. Scaling:

The excitation operation passes the “excited” tensor of shape B x 1 x 1 x C. This tensor is then passed through a sigmoid activation function. The sigmoid activation function converts the tensor values in the range of 0 and 1. We then perform an element-wise multiplication between the output of the sigmoid activation function and the input feature map.

If the value is close to 0 means the channel is less important so, the values of the feature channel would be reduced, and if the value is close to 1, this means that is channel is important.

By integrating SE-blocks into existing CNN architectures, models can achieve better performance with minimal computational overhead, as it enables the network to focus on the most informative features dynamically.

3.3. How Squeeze-and-Excitation Networks Help?

So, till now you have understood the architecture of the Squeeze and Excitation Network. During the scaling operation, we perform an element-wise multiplication between the initial feature map and the output of the sigmoid activation function.

The sigmoid activation function outputs a value between 0 and 1 and each channel is multiplied with it.

So, now imagine, a channel is multiplied with a value that is near 0. It will reduce the pixel values of that feature map, as these pixel values are not much relevant according to the SE-block.

When the channel is multiplied with a value that is near 1, it would not reduce the pixel values that much, if compared with the above case.

So, now we can see that the Squeeze and Excitation Network basically scale each channel information. It reduces the non-relevant channel information, and the relevant channel are not much affected. So, after the whole operation, the feature map only contains the relevant information, which increases the representational power of the entire network [20].

3.4. conclusion:

In the end, we leave you to the next chapter, in which we made a comparison between many models with the addition of the new architecture, SE-block, and the results we obtained, many observations, and a lot of fun a wait you in the next chapter.

Chapter 4: Experiment and result

Chapter 4

4.1 Introduction:

In this study, the main goal is to compare CNN and CNN+SENET in both binary classification and multiple classification using CT-scan images.

First, we explain the data that we used in our study and its source, because it is the most important step before the beginning of the study. After that, we move to the training stage, where we use the custom CNN model for two types of training methods.

First, by classifying an image into Covid-19 or not Covid-19 in binary classification, and the second method is multi classification. We added data for others disease in addition to Covid-19 and Healthy data, and this is to prove the quality of the model in dealing with the distinction between respiratory diseases.

After the training process, we analyse the results.

4.2 Data Description:

-We have 2 classes of CT-scan images in Binary classification: [21]

Table 1: Table showing data distribution for binary classification.

Data/case	Covid-19	Normal	overall
Training	1001	983	1984
Validation	125	123	248
Test	126	123	249
overall	1252	1229	2281

We have 3 classes of CT-scan images in multi classification: [22]

Table 2: Table showing data distribution for multi classification.

Data/case	Covid-19	others	Normal	Overall
Training	606	606	606	1818
Validation	75	75	75	225
Test	76	76	76	228
Overall	757	757	757	2271

4.3 The method used for splitting data (split folders):

Split folders are a widely used approach to divide a dataset into different groups or folders. This technique is commonly used in data analysis, especially when organizing data into training, validation, and testing sets. By applying split folders, the data is properly allocated to each subset, considering predefined proportions or criteria. This ensures that model training, evaluation, and generalization are carried out effectively.

We divided the data into 3 folders (train/test/Val) by using the split folders library from Python to train and evaluate the model effectively, we split the data by the ratio of 80% for training, 10% for validation and 10% for testing. You can see the code source used for splitting the data in the figure below.

- ***Function to Create Data Frame from Dataset in Kaggle:***

```
# Generate data paths with paths
def define_paths(data_dir):
    filepaths = []
    labels = []

    folds = os.listdir(data_dir)
    for fold in folds:
        foldpath = os.path.join(data_dir, fold)
        filelist = os.listdir(foldpath)
        for file in filelist:
            fpath = os.path.join(foldpath, file)
            filepaths.append(fpath)
            labels.append(fold)

    return filepaths, labels

# Concatenate data paths with labels into one dataframe ( to later be fitted into the model )
def define_df(files, classes):
    Fseries = pd.Series(files, name='filepaths')
    Lseries = pd.Series(classes, name='labels')
    return pd.concat([Fseries, Lseries], axis=1)

# Split dataframe to train, valid, and test
def split_data(data_dir):
    # train dataframe
    files, classes = define_paths(data_dir)
    df = define_df(files, classes)
    start = df['labels']
    train_df, dummy_df = train_test_split(df, train_size=0.8, shuffle=True, random_state=123, stra
tify=start)

    # valid and test dataframe
    strat = dummy_df['labels']
    valid_df, test_df = train_test_split(dummy_df,  train_size= 0.5, shuffle= True, random_state=
123, stratify= strat)

    return train_df, valid_df, test_df
```

Figure 15: Function to Create Data Frame from Dataset in Kaggle:

- ***Function to generate images from data frame in Kaggle:***

```
def create_gens(train_df, valid_df, test_df, batch_size):
    '''
    This function takes train, validation, and test dataframes and fit them into image data generato
r, because model takes data from image data generator.
    Image data generator converts images into tensors. '''

    # define model parameters
    img_size = (224, 224)
    channels = 3 # either BGR or Grayscale
    color = 'rgb'
    img_shape = (img_size[0], img_size[1], channels)

    # Recommended: use custom function for test data batch size, else we can use normal batch size.
    ts_length = len(test_df)
    test_batch_size = max(sorted([ts_length // n for n in range(1, ts_length + 1) if ts_length%n =
= 0 and ts_length/n <= 80]))
    test_steps = ts_length // test_batch_size

    # This function which will be used in image data generator for data augmentation, it just takes
the image and return it again.
    def scaler(img):
        return img

    tr_gen = ImageDataGenerator(preprocessing_function=scaler, horizontal_flip=True)
    ts_gen = ImageDataGenerator(preprocessing_function=scaler)

    train_gen = tr_gen.flow_from_dataframe(train_df, x_col='filepaths', y_col='labels', target_siz
e=img_size, class_mode='categorical', color_mode=color, shuffle=True, batch_size=batch_size)
    valid_gen = ts_gen.flow_from_dataframe(valid_df, x_col='filepaths', y_col='labels', target_siz
e=img_size, class_mode='categorical', color_mode=color, shuffle=True, batch_size=batch_size)

    # Note: we will use custom test_batch_size, and make shuffle=false
    test_gen = ts_gen.flow_from_dataframe( test_df, x_col='filepaths', y_col='labels', target_size
=img_size, class_mode='categorical', color_mode=color, shuffle=False, batch_size=test_batch_size)

    return train_gen, valid_gen, test_gen
```

Figure 16: Function to generate images from data frame in Kaggle.

4.1. Criteria of Evaluation and Metrics:

A confusion matrix is a crucial tool in evaluating the performance of a classification algorithm. It provides a summary of the prediction results on a classification problem by comparing the actual target values with those predicted by the machine learning model. Here's a detailed explanation of the confusion matrix components and how to interpret them:

Components of a Confusion Matrix:

A confusion matrix for a binary classification problem typically looks like this:

Table 3: confusion matrix

	P	**N**
P	True Positives (TP)	False Positives (FP)
N	False Negatives (FN)	True Negatives (TN)

True Positives (TP): These are the cases where the actual class is positive, and the model correctly predicts it as positive.

True Negatives (TN): These are the cases where the actual class is negative, and the model correctly predicts it as negative.

False Positives (FP): These are the cases where the actual class is negative, but the model incorrectly predicts it as positive (also known as a Type I error).

False Negatives (FN): These are the cases where the actual class is positive, but the model incorrectly predicts it as negative (also known as a Type II error).

- **Accuracy**

Accuracy is a metric used to evaluate the overall performance of a classification model. It measures the proportion of correctly classified instances (both true positives and true negatives) among the total number of instances, Accuracy can be calculated with the equation given in (2):

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \qquad (2)$$

➢ Precision

Precision, also known as Positive Predictive Value, is a metric used to evaluate the accuracy of positive predictions made by a classification model. It is particularly useful when the cost of false positives is high and we want to ensure that when the model predicts a positive class, it is indeed correct, precision can be calculated with the equation given in (3):

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

➢ Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances out of all actual positive instances in the dataset.

Mathematically, recall can be calculated with the equation given in (4):

$$Recall = \frac{TP}{TP+FN} \quad (4)$$

➢ Specificity

Specificity is a metric used particularly in binary classification problems. It measures the proportion of correctly predicted negative instances out of all actual negative instances in the dataset.

Mathematically, specificity can be calculated with the equation given in (5):

$$Specificity = \frac{TN}{TN+FP} \quad (5)$$

➢ The F1 score

Mathematically, the F1 score can be calculated as the harmonic mean of precision and recall with the equation given in (6):

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall} \qquad (6)$$

By analysing these metrics, you can better understand the strengths and weaknesses of your model and make informed decisions about model improvements or adjustments.

4.4.2 Performance of a system of CT-Scan image classification using C-CNN:

4.4.2.1 Binary classification:

We execute the model in Kaggle fixing the same hyperparameters utilized above we obtained the following results shown in image () and table ().

Table 4: classification report from C-CNN for Binary classification (Batch= 32, LR= 0.001, Epoch=25)

	precision	recall	F1-score	support
COVID	0.91	0.93	0.92	126
non-COVID	0.93	0.91	0.92	123
accuracy			**0.92**	**249**

Table 5: results C-CNN (Binary-classification)

Train Loss	4.4351309043122455e-06
Train accuracy	1.0
Validation Loss	0.39954569935798645
Validation Accuracy	0.9435483813285828
Test loss	0.5965678095817566
Test Accuracy	0.9196786880493164

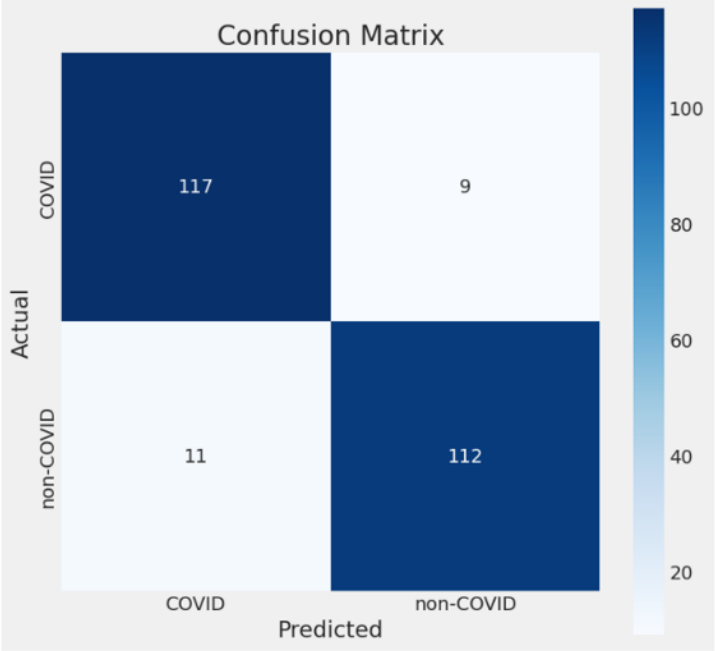


Figure 17: C-CNN confusion matrix for binary classification

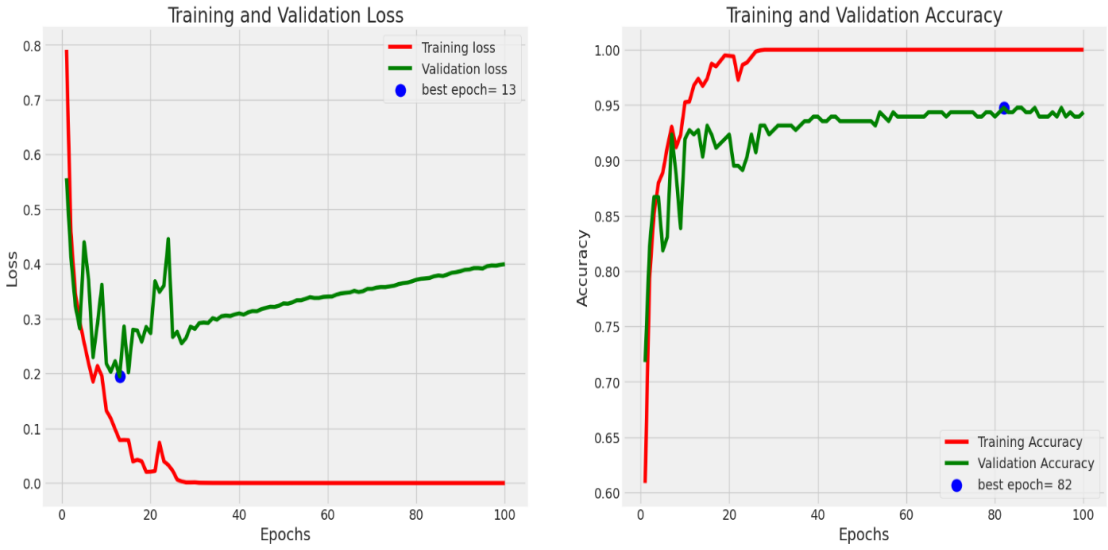


Diagram 1 :Training and validation loss and training and validation accuracy Ep=100

-We notice that Val-loss at EP=25 stopped decreasing and started rising straight up.

Conclusion:

It is better to stop at EP=25 or use an early stop to stop the training process.

- Experiment using Ep= 25, Lr=0.001, Bs=32 the results were as follows:

Table 6: classification report from C-CNN for Binary classification (Batch= 32, LR= 0.001, Epoch=25

	Precision	recall	F1-score	support
COVID	0.90	0.95	0.92	126
non-COVID	0.95	0.89	0.92	123
accuracy			**0.92**	**249**

Table 7: results C-CNN Ep=25(Binary-classification)

Train Loss	0.009414691478013992
Train accuracy	0.9974798560142517
Validation Loss	0.3501361906528473
Validation Accuracy	0.9274193644523621
Test loss	0.3872244358062744
Test Accuracy	0.9196786880493164

Diagram 2 Training and validation loss and training and validation accuracy Ep=25

-We notice that compared to 100 Ep, it gave us the same acc, but in return the loss value became 35% lower.

-The time spent training was 5 minutes (306 seconds).

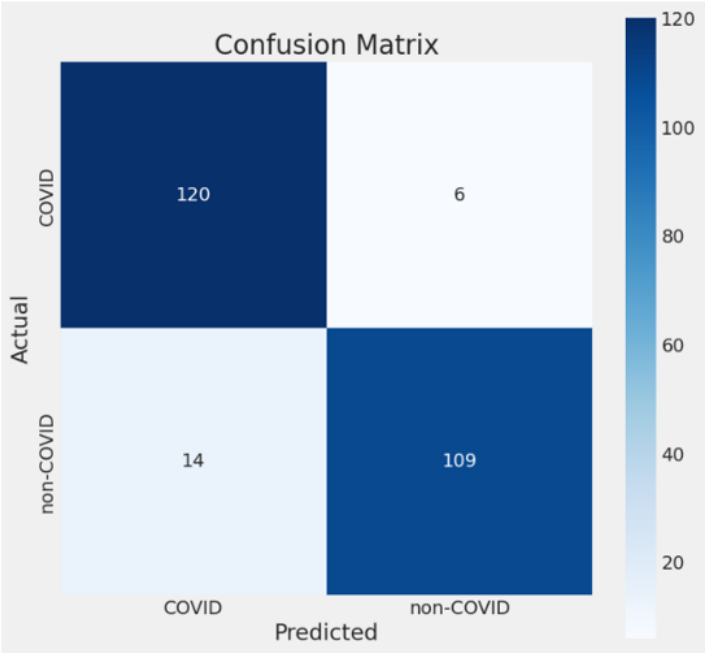


Figure 18: C-CNN confusion matrix for binary classification ep=25

The confusion matrix shown in Figure 23 reveals the performance of the C-CNN model. The matrix is a 2x2 square matrix representing the three classes (Covid, No-Covid).

- **COVID-19 class:** The C-CNN model correctly classified 120 samples as COVID-19. However, it misclassified 6 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Normal class:** The C-CNN model correctly classified 109 samples as Normal. However, it misclassified14 samples as COVID-19 . Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

Conclusion:

with 25=Ep our C-CNN is more suitable because it gives us the same acc with less loss and much less time.

4.4.2.2 Multi classification :

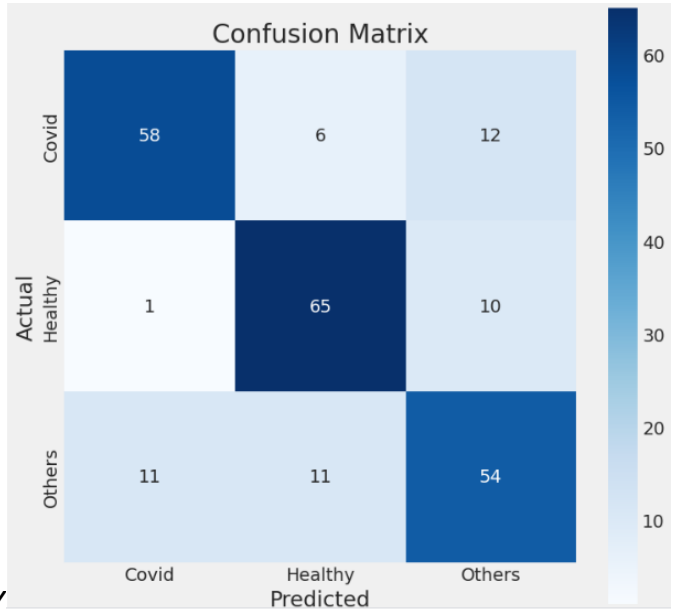
a. Multi classification :

Table 8: classification report from C-CNN for Multi-classification (Batch= 32, LR= 0.001, Epoch=100)

	precision	recall	f1-score	support
0	0.83	0.76	0.79	76
1	0.79	0.86	0.82	76
2	0.71	0.71	0.71	76
accuracy			0.78	228

Table 9: results C-CNN (Multi-classification)

Train Loss	0.00027149805100634694
Train accuracy	1.0
Validation Loss	1.1400889158248901
Validation Accuracy	0.75
Test loss	0.6308616995811462
Test Accuracy	0.7763158082962036



Y

Figure 19: C-CNN confusion matrix for Multi-classification

- **COVID-19 class:** The C-CNN model correctly classified 58 samples as COVID-19. However, it misclassified 6 samples as Healthy and 12 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Healthy class: The** C-CNN model correctly classified 65 samples as Healthy. However, it misclassified 1 sample as COVID-19 and 10 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.
- **Others class:** The C-CNN model correctly classified 54 samples as Others. However, it misclassified 11 samples as COVID-19 and 11 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a reasonably good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

4.4.3 Performance of a system of CT-Scan image classification using C-CNN+SE-Block:

4.4.3.1 Binary classification

We execute the model in Kaggle fixing the same hyperparameters utilized above we obtained the following results shown in image () and Table ():

Table 10: classification report from C-CNN+ SE-Block for multi classification (Batch= 32, LR= 0.001, Epoch=25)

	precision	recall	F1-score	support
COVID	0.93	0.96	0.95	126
non-COVID	0.96	0.93	0.94	123

accuracy	**0.94**	**249**

Table 11: results C-CNN+SE (Binary-classification)

Train Loss	0.009055455215275288
Train accuracy	0.9979838728904724
Validation Loss	0.2048267126083374
Validation Accuracy	0.9556451439857483
Test loss	0.34837666153907776
Test Accuracy	0.9437751173973083

-We notice an improvement in ACC and a decrease in Test Loss.

-The training time was approximately 6min (338s).

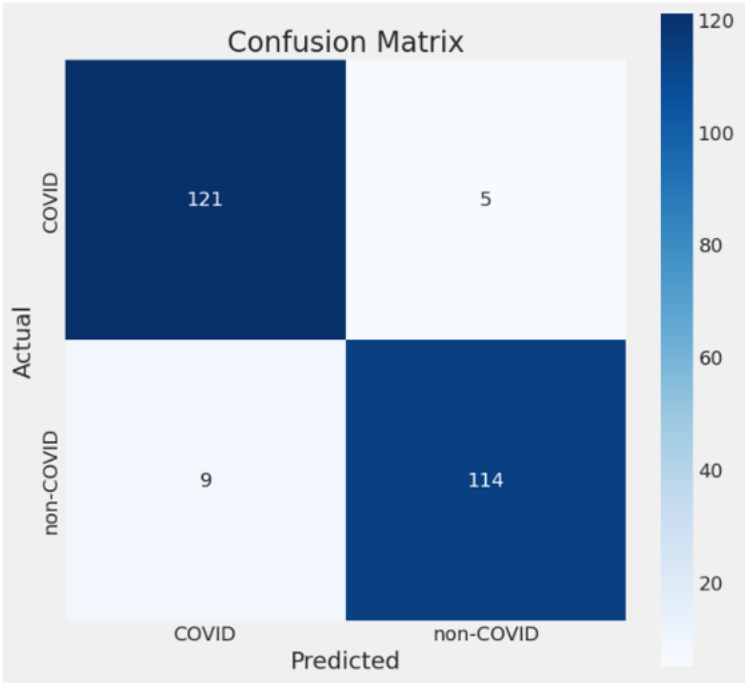


Figure 20: C-CNN +Se-block confusion matrix for binary classification

The confusion matrix shown in Figure 23 reveals the performance of the C-CNN model. The matrix is a 2x2 square matrix representing the three classes (Covid, No-Covid).

- **COVID-19 class:** The C-CNN+SE model correctly classified 121 samples as COVID-19. However, it misclassified 5 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Normal class:** The C-CNN+SE model correctly classified 114 samples as Normal. However, it misclassified 9 samples as COVID-19 . Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

4.4.3.2 Multi classification:

a. Multi classification First data:

Table 12: classification report from C-CNN +SE for Multi-classification (Batch= 32, LR= 0.001, Epoch=100)

	Precision	recall	f1-score	support
0	0.83	0.76	0.79	76
1	0.79	0.86	0.82	76
2	0.71	0.71	0.71	76
accuracy			0.78	228

Table 13: résulte C-CNN+SE (Multi-classification)

Train Loss	0.0005537105607800186
Train accuracy	1.0
Validation Loss	0.711747944355011
Validation Accuracy	0.8229166865348816
Test loss	0.8139581680297852
Test Accuracy	0.7938596606254578

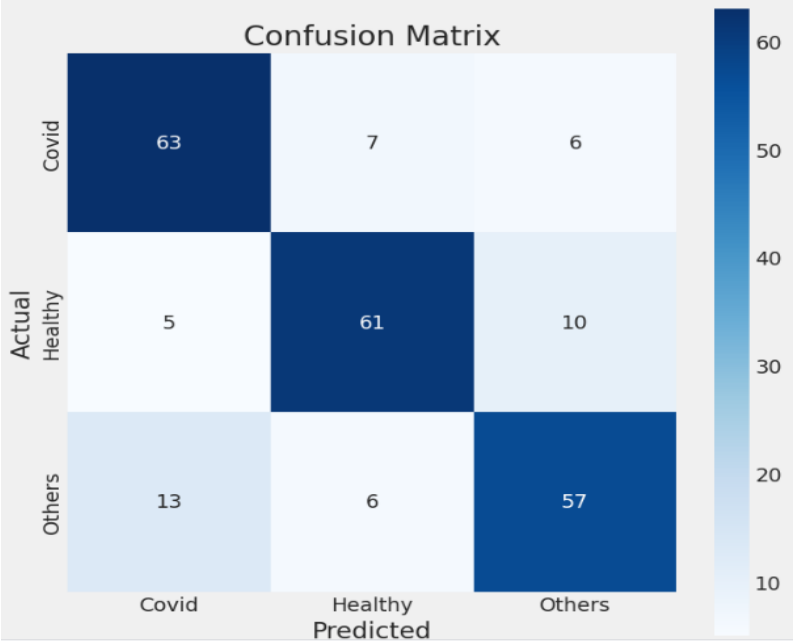


Figure 21 : C-CNN+SE confusion matrix for Multi-classification

The confusion matrix shown in Figure 23 reveals the performance of the

C-CNN+SE model. The matrix is a 3x3 square matrix representing the three classes (Covid, Healthy, and Others).

- **COVID-19 class:** The C-CNN+SE model correctly classified 63 samples as COVID-19. However, it misclassified 7 samples as Healthy and 6 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.

- **Healthy class: The** C-CNN+SE model correctly classified 61 samples as Healthy. However, it misclassified 5 samples as COVID-19 and 10 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.

- **Others class:** The C-CNN+SE model correctly classified 57 samples as Others. However, it misclassified 13 samples as COVID-19 and 6 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a reasonably good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

4.4.4 Performance of a system of CT-Scan image classification using Alex-Net:

4.4.4.1Binary classification

We execute the model in Kaggle fixing the same hyperparameters utilized above we obtained the following results shown in image () and Table:

Table 14: classification report from Alex-Net for Binary classification (Batch= 23, LR= 0.0001, Epoch=100

	Precision	recall	F1-score	support
COVID	0.93	0.94	0.94	126
Non-COVID	0.94	0.93	0.93	123
accuracy			**0.94**	**249**

Table 15: results Alex-Net (Binary-classification)

Train Loss	8.394466931349598e-06
Train accuracy	1.0
Validation Loss	0.2641461193561554
Validation Accuracy	0.9596773982048035
Test loss	0.8253661394119263
Test Accuracy	0.935742974281311

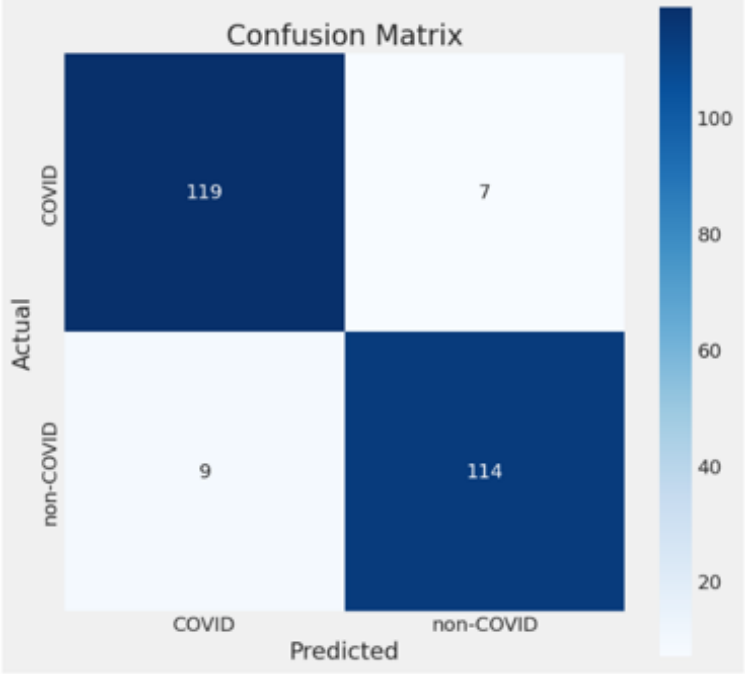


Figure 22: Alex-net confusion matrix for binary classification

The confusion matrix shown in Figure 23 reveals the performance of the Alex-Net model. The matrix is a 2x2 square matrix representing the three classes (Covid, Normal).

- **COVID-19 class:** The Alex net model correctly classified 119 samples as COVID-19. However, it misclassified 7 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Normal class:** The alex net model correctly classified 114 samples as Normal. However, it misclassified 9 samples as COVID-19 . Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

4.4.4.2Multi classification:

a. Multi classification First data:

Table 16: classification report from Alex net for Binary classification (Batch= 32, LR= 0.0001, Epoch=100)

	precision	recall	f1-score	support
0	0.94	0.76	0.84	76
1	0.83	0.86	0.84	76
2	0.75	0.87	0.80	76
accuracy			0.83	228

Table 17: results Alex-net (multi-classification)

Train Loss	8.268122655863408e-06
Train accuracy	1.0
Validation Loss	0.4608541429042816
Validation Accuracy	0.8541666865348816
Test loss	0.6101381778717041
Test Accuracy	0.8289473652839661

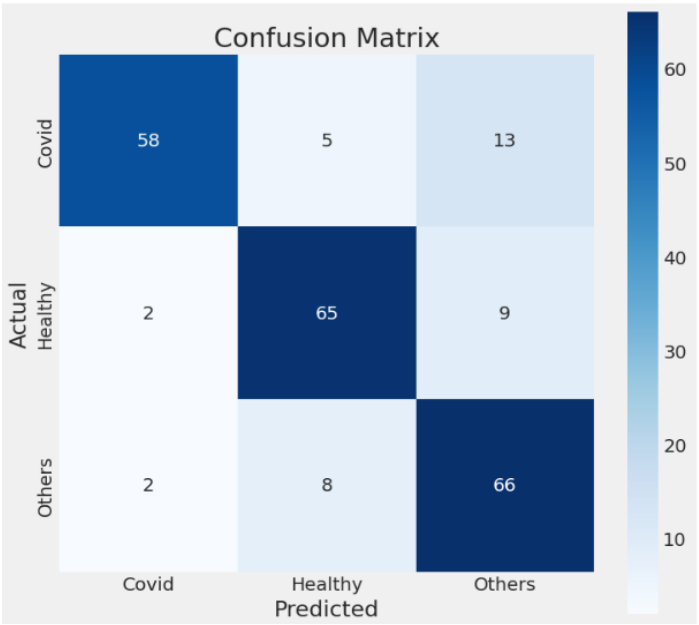


Figure 23: Alxe-Net confusion matrix for Multi-classification

The confusion matrix shown in Figure 23 reveals the performance of the Alex-net model. The matrix is a 3x3 square matrix representing the three classes (Covid, Healthy, and Others).

- **COVID-19 class:** The Alex net model correctly classified 58 samples as COVID-19. However, it misclassified 5 samples as Healthy and 13 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Healthy class: The** Alex net model correctly classified 65 samples as Healthy. However, it misclassified 2 samples as COVID-19 and 9 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.
- **Others class:** The Alex net model correctly classified 66 samples as Others. However, it misclassified 2 samples as COVID-19 and 8 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a reasonably good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

4.4.5 Performance of a system of CT-Scan image classification using Alex-Net +SE-Block:

4.4.5.1 binary classification

We execute the model in Kaggle fixing the same hyperparameters utilized above we obtained the following results shown in image () and Table:

Table 18: classification report from Alex-Net+SE for Binary classification (Batch= 32, LR= 0.001, Epoch=50)

	Precision	recall	F1-score	support
COVID	0.94	0.94	0.94	126
Non-COVID	0.95	0.93	0.93	123
accuracy			**0.94**	**249**

Table 19: results Alxe-Net +SE (Binary-classification)

Train Loss	5.8294426708016545e-06
Train accuracy	1.0
Validation Loss	0.36092373728752136
Validation Accuracy	0.9395161271095276
Test loss	0.6220570802688599
Test Accuracy	0.9437751173973083

We notice a difference when adding S-block, an improvement in Test-acc and a decrease in Test-Loss. Results in the pictures () ().

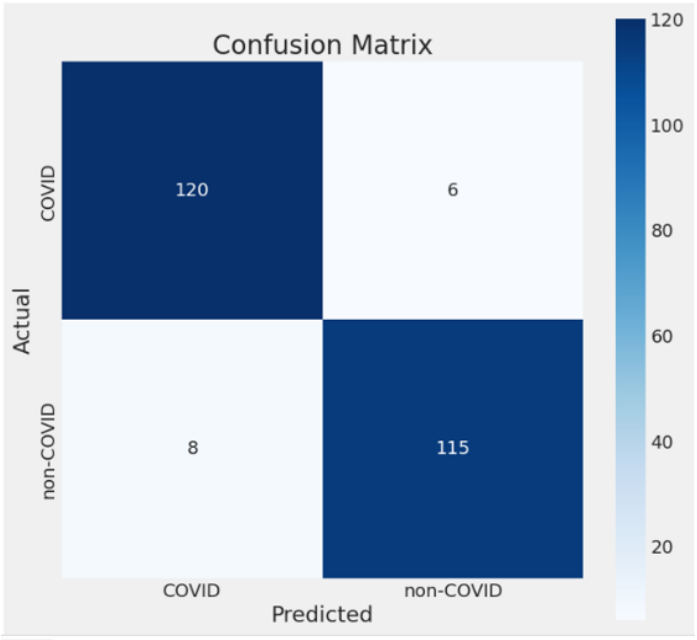


Figure 24: Alxe-Net+SE-block confusion matrix for binary classification

The confusion matrix shown in Figure 23 reveals the performance of the Alex-net+SE model. The matrix is a 2x2 square matrix representing the three classes (Covid, No-Covid).

- **COVID-19 class:** The Alxe-Net+SE model correctly classified 120 samples as COVID-19. However, it misclassified 6 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Normal class:** The Alxe-Net+SE model correctly classified 115 samples as Normal. However, it misclassified 8 samples as COVID-19 . Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

➢

4.4.5.2 Multi classification:

a. Multi classification:

Table 20: classification report from Alxe-Net+SE **for Binary classification (Batch= 32, LR= 0.0001, Epoch=100)**

	precision	recall	f1-score	support
0	0.91	0.80	0.85	76
1	0.87	0.93	0.90	76
2	0.81	0.84	0.83	76
accuracy			0.86	228

Table 21 : results Alxe-Net+SE **(Multi-classification)**

Train Loss	6.792152362322668e-06
Train accuracy	1.0
Validation Loss	0.5303332209587097
Validation Accuracy	0.8854166865348816
Test loss	0.5075471997261047
Test Accuracy	0.859649121761322

Table 3 results

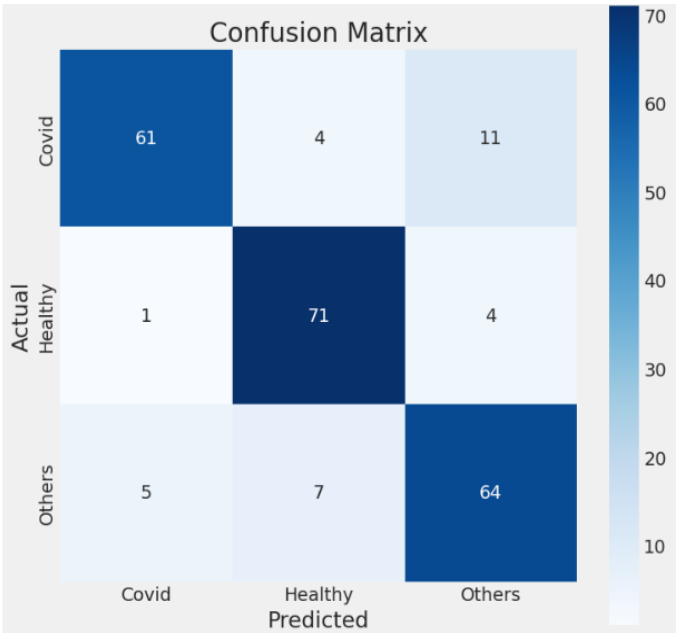


Figure 25: Alxe-Net+SE confusion matrix for binary classification

The confusion matrix shown in Figure 23 reveals the performance of the Alex-net+SE model. The matrix is a 3x3 square matrix representing the three classes (Covid, Healthy, and Others).

- **COVID-19 class:** The Alxe-Net+SE model correctly classified 61 samples as COVID-19. However, it misclassified 4 samples as Healthy and 11 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Healthy class: The** Alxe-Net+SE model correctly classified 71 samples as Healthy. However, it misclassified 1 sample as COVID-19 and 4 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.
- **Others class:** The Alxe-Net+SE model correctly classified 64 samples as Others. However, it misclassified 7 samples as COVID-19 and 5 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a reasonably good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

4.4.6 Performance of a system of CT-Scan image classification using VGG16:

4.4.6.2 Binary classification:

Table 22: classification report from VGG16 for Binary classification (Batch= 32, LR= 0.0001, Epoch=100)

	precision	*recall*	*f1-score*	*support*
COVID	**0.95**	**0.98**	**0.97**	**126**
non-COVID	**0.98**	**0.95**	**0.97**	**123**
accuracy			**0.97**	**249**

Table 23: results VGG16(binary-classification)

Train Loss	1.1631196628059115e-07
Train accuracy	1.0
Validation Loss	0.3169421851634979
Validation Accuracy	0.975806474685669
Test loss	0.5550627112388611
Test Accuracy	0.9678714871406555

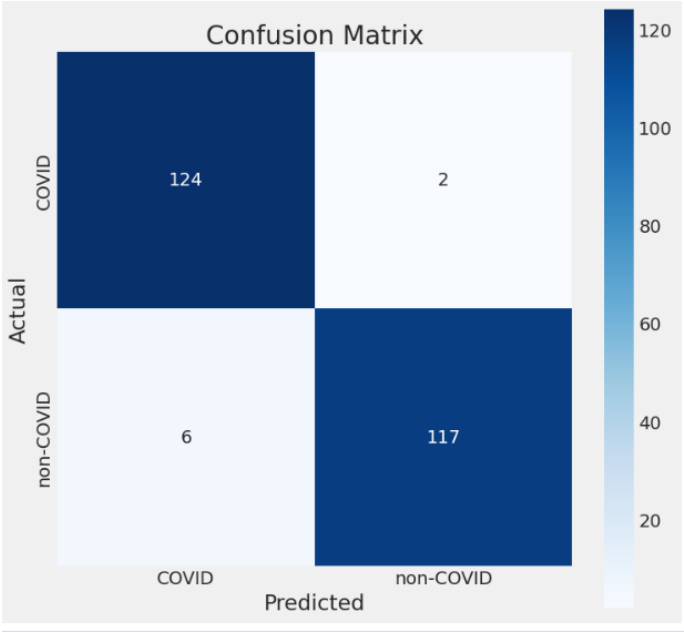


Figure 26: VGG16 confusion matrix for binary classification.

The confusion matrix shown in Figure 23 reveals the performance of the VGG16 model. The matrix is a 2x2 square matrix representing the three classes (Covid, No-Covid).

- **COVID-19 class:** The VGG16 model correctly classified 124 samples as COVID-19. However, it misclassified 2 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.

- **Normal class:** The VGG16 model correctly classified 117 samples as Normal. However, it misclassified 6 samples as COVID-19 . Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

4.4.6.3 Multi classification :

a. Multi classification :

Table 24: classification report from VGG16 for Multi classification (Batch= 32, LR= 0.0001, Epoch=100)

	precision	recall	f1-score	support
0	0.89	0.82	0.85	76
1	0.83	0.89	0.86	76
2	0.80	0.80	0.80	76
accuracy			0.84	228

Table 25: results VGG16(multi-classification)

Train Loss	1.3857239764547558e-06
Train accuracy	1.0
Validation Loss	0.4224267899990082
Validation Accuracy	0.8958333134651184
Test loss	1.267059087753296
Test Accuracy	0.8377193212509155

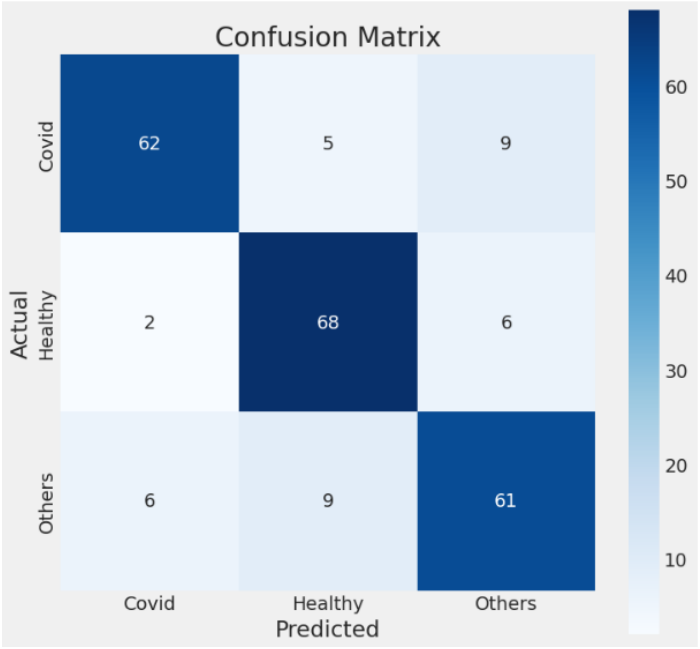


Figure 27: VGG16 confusion matrix for Multi-classification

The confusion matrix shown in Figure 23 reveals the performance of the VGG16 model. The matrix is a 3x3 square matrix representing the three classes (Covid, Healthy, and Others).

- **COVID-19 class:** The VGG16 model correctly classified 62 samples as COVID-19. However, it misclassified 5 samples as Healthy and 9 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Healthy class: The** VGG16 model correctly classified 68 samples as Healthy. However, it misclassified 2 samples as COVID-19 and 6 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.
- **Others class:** The VGG16 model correctly classified 61 samples as Others. However, it misclassified 6 samples as COVID-19 and 9 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a reasonably good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

➢

4.4.7 Performance of a system of CT-Scan image classification using VGG16+SE:

4.4.7.2Binary classification (Batch= 32, LR= 0.0001, Epoch=100):

Table 26: classification report from mini- VGG16 for Binary classification (Batch= 32, LR= 0.0001, Epoch=100)

	precision	recall	f1-score	support
COVID	**0.51**	**1.00**	**0.67**	**126**
non-COVID	**0.00**	**0.00**	**0.00**	**123**
accuracy			**0.51**	**249**

Table 27: results VGG16+SE (binary-classification)

Train Loss	0.693106472492218
Train accuracy	0.5045362710952759
Validation Loss	0.6931147575378418
Validation Accuracy	0.5040322542190552
Test loss	0.6930819153785706
Test Accuracy	0.5060241222381592

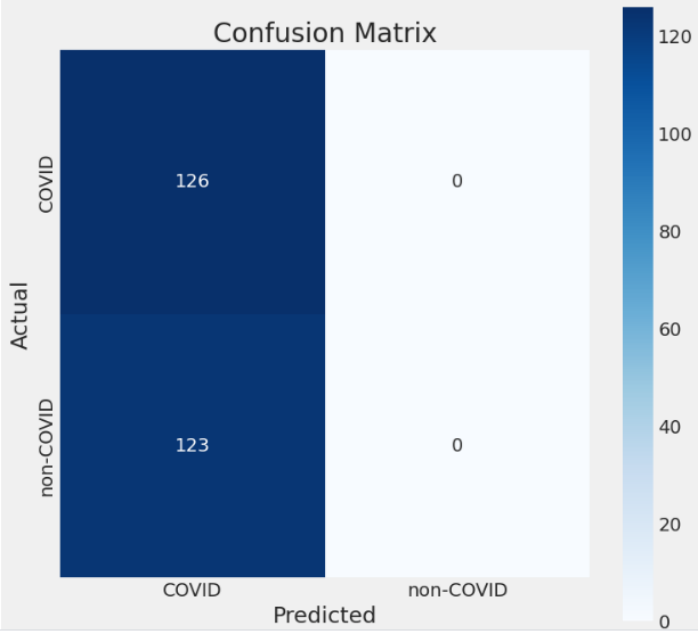


Figure 28: vgg16 confusion matrix for binary classification.

In contrast, the Vgg16 model exhibited different behaviour during training. Overfitting was not a major concern. We would have preferred to continue training for a longer duration, but resource limitations, such as a lack of GPU memory and computational power, prevented us from doing so.

But we solved this problem by reducing the layers of the model, and so we created a new type of model, which is Mini-VGG+SE, but it is still under study, and these are some of the results.

We notice that the model was improved when the number of layers was reduced and, in less time, u can see that in the table ().

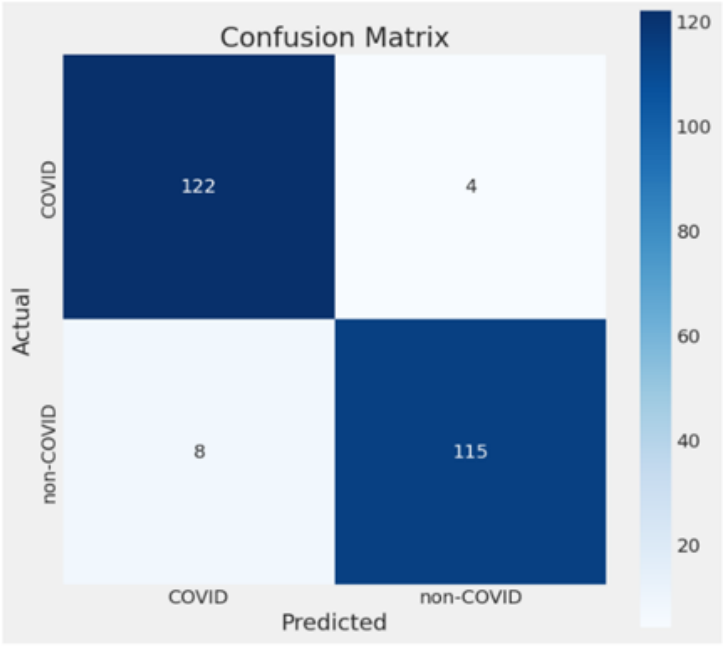


Figure 29: mini-vgg16+SE confusion matrix for binary classification

The confusion matrix shown in Figure 23 reveals the performance of the Mini-VGG+SE model. The matrix is a 2x2 square matrix representing the three classes (Covid, No-Covid).

- **COVID-19 class:** The Mini-VGG16+SE model correctly classified 122 samples as COVID-19. However, it misclassified 4 samples as Normal. Considering the class distribution, with 1252 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Normal class:** The Mini-VGG16+SE model correctly classified 115 samples as No-Covid. However, it misclassified 8 samples as COVID-19 Considering the class distribution, with 1229 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying Normal cases. However, there is a notable number of misclassifications in both directions.

Table 28: Comparison between VGG16+SE and Mini-VGG16+SE:

	VGG16+SE	Mini-VGG16+SE
Acc	50%	95%
Loss	88%	54%

time	29min	50min

4.4.7.3 Multi classification:

a. Multi classification first-data:

Table 29: classification report from VGG16+SE for Multi classification (Batch= 32, LR= 0.0001, Epoch=100)

	precision	recall	f1-score	support
0	0.90	0.86	0.88	76
1	0.87	0.99	0.93	76
2	0.89	0.82	0.85	76
accuracy			0.89	228

Table 30 : results VGG16+SE (Multi-classification)

Train Loss	1.2635294979190803e-06
Train accuracy	1.0
Validation Loss	1.1296567916870117
Validation Accuracy	0.90625
Test loss	0.7284902334213257
Test Accuracy	0.8859649300575256

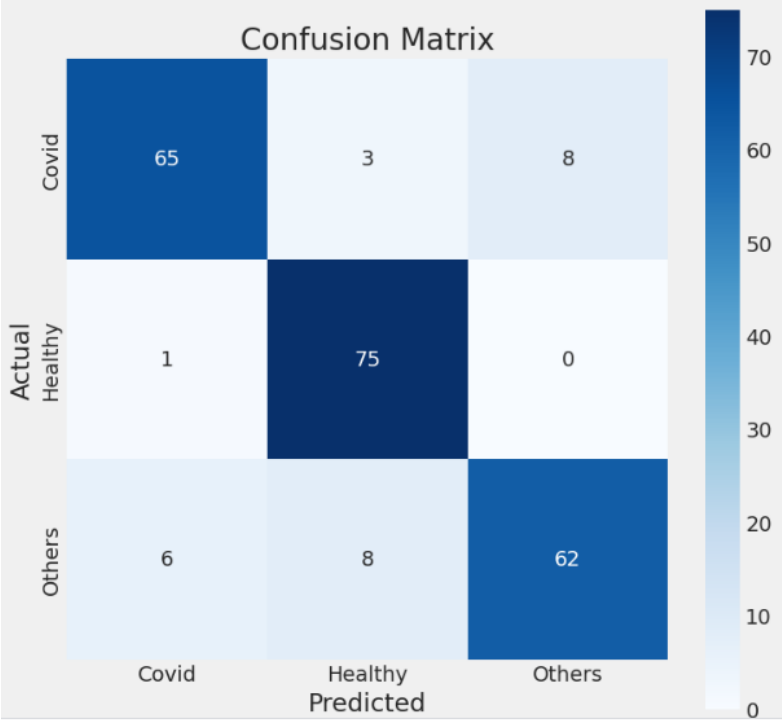


Figure 30: VGG16 confusion matrix for Multi-classification

The confusion matrix shown in Figure 23 reveals the performance of the VGG16+SE model. The matrix is a 3x3 square matrix representing the three classes (Covid, Healthy, and Others).

- **COVID-19 class:** The VGG16+SE model correctly classified 65 samples as COVID-19. However, it misclassified 3 samples as Healthy and 8 samples as others. Considering the class distribution, with 757 samples in the COVID-19 class, the model shows a reasonably good accuracy in identifying COVID-19 cases. However, there is a notable number of misclassifications in both directions.
- **Healthy class: The** VGG16+SE model correctly classified 75 samples as Healthy. However, it misclassified 1 sample as COVID-19 and 0 samples as others. Considering the class distribution, with 757 samples in the Healthy class, the model shows a reasonably good accuracy in identifying Healthy cases. However, there is a notable number of misclassifications in both directions.
- **Others class:** The VGG16+SE model correctly classified 62 samples as Others. However, it misclassified 6 samples as COVID-19 and 9 samples as Healthy. Considering the class distribution, with 757 samples in the others class, the model shows a good accuracy in identifying other cases. However, there is a notable number of misclassifications in both directions.

1. Comparison between models of a system of CT-Scan image classification:

table 31: Comparison between models of a system of CT-Scan image classification

MODELS	HYPER PARMANENT	CLASSIFICATION TYPE	ACC	LOSS	TIME
C-CNN	Bs= 32 Lr= 0.001 EP= 25	Binary	91%	38%	5 min
		Multi	77%	63%	17 min
C-CNN +SE		Binary	**94%**	**34%**	6 min
		Multi	**79%**	**81%**	22 min
ALEX- NET	Bs= 32 Lr= 0.0001 EP= 100	Binary	93%	82%	23min
		Multi	82%	61%	22 min
ALEX- NET +SE		Binary	**94%**	**62%**	26 min
		Multi	**85%**	**50%**	21 min
VGG 16	Bs= 32 Lr= 0.0001 EP= 100	Binary	96%	55%	46 min
		Multi	83%	126%	45 min
Mini-VGG 16 +SE		Binary	**95%**	**54%**	29 min
VGG 16 +SE		Multi	**88%**	**72%**	45 min

1.1. Comparison between CCNN and CNN+SE:

The results of the comparison between the three models as we can see in Table ():

a. binary-classification:

- We note that the results of the CNN+SE model were better than the results of the CNN model, as we see that Test acc and Test loss improved when SE-Block was added.

b. (multi-classification):

- We note that the CNN model may obtain a loss that is less than the CNN +SE model, but we also note that the CNN +SE model may obtain an acc that is approximately 2% better than the CNN model.

1.2. Comparison between Alex-Net and Alex-net:

The results of the comparison between the three models as we can see in Table ():

a. binary-classification:

- We note that the results of the AlexNet's model were better than the results of the Alex-Net model, as we see that Test acc and test loss improved when SE-Block was added.

b. (multi-classification):

- We note that the model was improved when adding the SE-block with an increase in test acc and a decrease in test loss.

1.3. Comparison between VGG16 and VGG16+SE:

The results of the comparison between the three models as we can see in Table ():

a. binary-classification:

- We note that when adding the SE-block, it did not work well and gave random results, due to data imbalance or computational complexity.

b. (multi-classification):

- We notice this time that SE-Block has worked very well by improving both Test loss and Test acc, which were not good in the VGG16 model.

Chapter 5: Web application

Chapter 5

5.1. Introduction:

Web development is the work involved in developing a website for the Internet. Web development can range from developing a single simple static page of plain text to complex web applications, e-businesses, and social networking services, but this time we have integrated deep learning into the web by creating an interface that receives medical images of the lung and classifies them into Covid-19 or not, and all of this via The Python language and a library called Flask has a lot of interesting work that you will see in this research, which is considered the summary of our work.

5.2. Definition of website development

Web development is the building and maintenance of websites; It's the work that happens behind the scenes to make a website look great, work fast and perform well with a seamless user experience.

Web developers, or 'devs', do this by using a variety of coding languages. The languages they use depends on the types of tasks they are performing and the platforms on which they are working.

Web development skills are in high demand worldwide and well paid too – making development a great career option. It is one of the easiest accessible higher paid fields as you do not need a traditional university degree to become qualified.

The field of web development is generally broken down into front-end (the user-facing side) and back-end (the server side).

5.3. Software Used

5.3.1. Visual Studio Code:

Visual Studio Code is an editor of extensible code developed by Microsoft for Windows, Linux and macOS. The functions include the debugging charge, syntax error, intelligent code compilation, snippets, code refactoring and Git integration [23].

5.4. programming Language Used

5.4.1. Python:

The Python language is used in many fields, and our memorandum indicates this. We have used it in machine learning and deep learning to develop many models, and here we are once again using it in website development. It has facilitated the development of a user interface in the medical field easily [5]..

5.4.2. Html:

HyperText Markup Language, commonly abbreviated HTML or, in its latest version, HTML5, is the markup language designed to represent web pages [24].

5.4.3. CSS:

Cascading Style Sheets, generally called CSS, form a computer language that describes the presentation of HTML and XML documents. The standards defining CSS are published by the World Wide Web Consortium [25]

5.5. Python Libraries Used

We used a new library called flask in addition to all of the libraries we used in the first research:

5.5.1. Flask:

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

Flask is part of the categories of the micro-framework. Micro-framework are normally framework with little to no dependencies to external libraries. This has pros and cons. Pros would be that the framework is light, there are little dependencies to update and watch for security bugs, cons is that some time you will have to do more work by yourself or increase yourself the list of dependencies by adding plugins [26]

a. routs:

Modern web applications use meaningful URLs to help users. Users are more likely to like a page and come back if the page uses a meaningful URL they can remember and use to directly visit a page [26].

Use the **route()** decorator to bind a function to a URL.

You can do more! You can make parts of the URL dynamic and attach multiple rules to a function.

b. render template:

Generating HTML from within Python is not fun, and actually pretty cumbersome because you have to do the HTML escaping on your own to keep the application secure. Because of that Flask configures the Jinja2 template engine for you automatically.

Templates can be used to generate any type of text file. For web applications, you'll primarily be generating HTML pages, but you can also generate markdown, plain text for emails, and anything else.

To render a template, you can use the render template method. All you must do is provide the name of the template and the variables you want to pass to the template engine as keyword arguments. Here's a simple example of how to render a template:

Flask will look for templates in the templates folder [26].

c. request:

For web applications it's crucial to react to the data a client sends to the server. In Flask this information is provided by the global request object. If you have some experience with Python, you might be wondering how that object can be global and how Flask manages to still be thread safe [26]

d. Lode model:

Model progress can be saved during and after training. This means a model can resume where it left off and avoid long training times. Saving also means you can share your model and others can recreate your work. When publishing research models and techniques, most machine learning practitioners share:

code to create the model, and

the trained weights, or parameters, for the model

Sharing this data helps others understand how the model works and try it themselves with new data [10].

e. jsonify:

jsonify simplifies the process of creating JSON responses in Flask by taking care of the necessary headers and serialization, allowing you to focus on the data you want to send back to the client [27].

5.6. Code description

5.6.1. Imports and Initializations:

```
from flask import Flask, render_template, request, jsonify
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
```

The required libraries are imported. Flask is used to create the web application, and TensorFlow/Keras for handling the image classification model. NumPy is used for numerical operations.

```
app = Flask(__name__)
senet = load_model('cnn+senet30E.h5')
```

- An instance of the Flask app is created.
- The pre-trained model cnn+senet30E.h5 is loaded.

5.6.2. Image Preparation Function:

```
def prepare_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224,3))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img /= 255.0
    return img
```

- This function takes the path of an image, processes it, and returns it in a format suitable for model prediction.
- The image is resized to the required input size of the model, converted to an array, expanded to match the model's expected input shape, and normalized by scaling pixel values to the range [0, 1].

5.6.3. Homepage Route:

```
@app.route("/")
def homepage():
    return render_template("homepage.html")
```

The root route renders the homepage template, which would likely include an HTML form for uploading images.

5.6.4. Prediction Route:

```
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return 'No file part'

    file = request.files['file']
    if file.filename == '':
        return 'No selected file'

    img_path = 'uploads/' + file.filename
    file.save(img_path)

    img = prepare_image(img_path)
    senet_pred = senet.predict(img)[0][0]
    senet_pred = float(senet_pred)

    # Classification based on threshold
    classification = 'COVID' if senet_pred >= 0.9 else 'No COVID'

    result = {
        'senet_prediction': senet_pred,
        'classification': classification
    }

    return jsonify(result)
```

- This route handles the file upload and prediction.
- Checks if the file part exists in the request and if a file is selected.
- Saves the uploaded file to a directory (uploads/).
- Processes the saved image using the prepare image function.

- Predicts using the loaded model and converts the prediction to a float.
- Classifies the result based on a threshold (0.9 in this case).
- Returns the prediction and classification result as a JSON response.

5.6.5. Running the App:

```
if __name__ == "__main__":
    app.run(debug=True)
```

Starts the Flask application in debug mode.

General Conclusion

general conclusion

In our research study, our goal was to compare the performance of CNN models by adding a new architectural unit, SENet, for classifying chest CT-Scan images. During the first three chapters, we provided comprehensive definitions of CNN and SENet, along with related concepts and components.

We observed several key findings in our study. First, we identified a significant problem with data imbalance in the biomedical field, which poses challenges for CT-Scan image classification. Additionally, we recognized the complexity of this task due to the presence of similar visual patterns shared among CT-Scan images.

Based on our experiments, we found that SENet showed superior performance when added to well-known CNN models, VGG16, Alex-Net, and custom CNN, in both binary and multi-class classification of CT-Scan images.

We attribute the success of SENet in improving the model to its channel-specific control mechanism, which recalibrates each channel by enhancing important features. This includes improving inter-channel correlations and refining a parameterized network that reweights each channel to be more sensitive to relevant features and ignore unrelated ones.

It is important to emphasize that the main objective of our study was to integrate the SENet unit into CNN models and compare them with standard CNN models for classifying chest CT-Scan images using the same datasets. Through this comparison, we demonstrated that SENet improves the model and reduces its error rate. Our findings contribute to the scientific understanding of CT-scan image classification methodologies by improving CNN models when adding the new architectural unit in this field.

Bibliographie

[1] R. V. M. D. N. T. CARNEIRO et N. THIAGO , «Performance analysis of google colaboratory as a tool for accelerating deep learning applications,» *IEEE Access,* 2018.

[2] Z. C. M. Nour et P. Kemal , «A Novel Medical Diagnosis model for COVID-19 infection detection based on Deep Features and Bayesian Optimization,» *Eelsevier,* 2020.

[3] f. chollet, «Deep Learning with Python,» *Second Edition, Simon and Schuster,* 2021.

[4] S. v. d. Walt, C. S. Chris et V. Gael , «The NumPy Array: A Structure for Efficient Numerical Computation,,» *IEEE,* 2011.

[5] J. D. HUNTER, «Matplotlib: A 2D graphics environment,» *Computing in science & engineering,* 2007.

[6] T. S. Albawi, M. Abed et A.-Z. Saad , «Understanding of a convolutional neural network,» *IEEE,* 2017.

[7] «kaggle,» [En ligne]. Available: https://www.kaggle.com/learn.

[8] K. Janocha et M. C. Wojciech , «On Loss Functions for Deep Neural Networks in Classification,» *arXiv preprint,* 2017.

[9] «Spyder,» En ligne, [En ligne]. Available: https://en.wikipedia.org/wiki/Spyder_(software)..

[10] f. chollet, «Deep Learning with Python, Second Edition,» *Simon and Schuster,* 2021.

[11] S. v. d. Walt, C. S. Chris et V. Gael, «The NumPy Array: A Structure for Efficient Numerical Computation,» *IEEE,* 2011.

[12] [En ligne]. Available: https://scikit-learn.org/stable/.

[13] T. B. A. e. I. T. g. S. Toraman, «Convolutional CapsNet: A novel artificial neural network approach to detect COVID-19 disease from X-ray images using capsule networks,» *Eelsevier,* 2020.

[14] S. I. A. e. M. A. T. I. D. Apostolopoulos, «Extracting Possibly Representative COVID 19 Biomarkers from X ray Images with Deep Learning Approach and Image Data Related to Pulmonary Diseases,» *Journal of Medical and Biological Engineering,* 2020.

[15] F. M. e. A. R. L. Brunese, «Explainable Deep Learning for Pulmonary Disease and Coronavirus COVID-19 Detection from X-rays,» *Elsevier,* 2020.

[16] I. D. A. e. T. A. Mpesiana, «Covid 19: automatic detection from X ray images utilizing transfer learning with convolutional neural networks,» *Physical and Engineering Sciences in Medicine,* 2020.

[17] T. A. M. e. S. A.-Z. S. Albawi, «Understanding of a convolutional neural network,» *IEEE,* 2017.

[18] A. K. SIMONYAN et ZISSERMAN, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» *arXiv preprint,* 2014.

[19] L. S. G. S. Jie Hu, «Squeeze-and-Excitation Networks,» *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 2018.

[20] Y. X. ,. X.-S. W. Xin Jin et Z. Bo-Rui , «Delving deep into spatial pooling for squeeze-and-excitation networks,» *elsevier journal,* 2022.

[21] «kaggle,» [En ligne]. Available: https://www.kaggle.com/datasets/plameneduardo/sarscov2-ctscan-dataset.

[22] «kaggle,» 2024. [En ligne]. Available: https://www.kaggle.com/datasets/yahiaselougha/mc-data.

[23] [En ligne]. Available: https://code.visualstudio.com/.

[24] [En ligne]. Available: https://www.codecademy.com/learn/learn-html.

[25] [En ligne]. Available: https://web.dev/learn/css.

[26] [En ligne]. Available: https://flask.palletsprojects.com/en/3.0.x/.

[27] [En ligne]. Available: https://www.w3schools.com/js/js_json_intro.asp.