A

Report On Capstone Project

Phase - I

**“Cloud-Based Chronic Disease Prediction Using Deep Learning Approach”**

Submitted

in partial fulfillment of the requirements for the degree of

#### Bachelor of Technology

#### in

#### Computer Science and Information Technology

***by***

Miss. Sakshi Rajendra Pawar (2210051)

Mr. Pranav Aviansh Desai (2210054)

Mr. Ajinkya Vikram Bhosale (2210003)

**Under the Guidance of**  
Prof. Savita P. Patil



**Information Technology Department**

K. E. Society’s

**Rajarambapu Institute of Technology, Rajaramnagar**

(An Empowered Autonomous Institute, Affiliated to Shivaji University)

**2024-2025**

K. E. Society’s

**Rajarambapu Institute of Technology, Rajaramnagar**

(An Empowered Autonomous Institute, Affiliated to Shivaji University)

**Department of Information Technology**

**CERTIFICATE**

This is to certify that the project work titled **“Cloud-Based Chronic Disease Prediction Using Deep Learning Approach”** is submitted by the following students, to the Rajarambapu Institute of Technology, Rajaramnagar during the academic year 2024-25, in partial fulfillment for the award of the degree of B. Tech in Computer Science and Information Technology under our supervision. This report is the record of students work carried out under my supervision and guidance.

| Name of Students | Roll Number |
| --- | --- |
| 1. Sakshi Rajendra Pawar 2. Pranav Avinash Desai 3. Ajinkya Vikram Bhosale | 2210051  2210054  2210003 |

Date: 9 April 2025

Place: Rajaramnagar

**Project Guide Project Coordinator Head IT Dept.**

**Prof. Savita P. Patil Prof. M. A. Vhatkar Dr. A. C. Adamuthe**

**DECLARATION**

We declare that this report reflects my thoughts about the subject in my own words. We have sufficiently cited and referenced the original sources, referred or considered in this work. We have not plagiarized or submitted the same work for the award of any other degree. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. We understand that any violation of the above will be cause for disciplinary action by the Institute.

| Sr. No. | Student Name | Roll No | Signature |
| --- | --- | --- | --- |
|  | Sakshi Rajendra Pawar | 2210051 |  |
|  | Pranav Avinash Desai | 2210054 |  |
|  | Ajinkya Vikram Bhosale | 2210003 |  |

Date:

Place: RIT, Rajaramnagar.

**Certificate**





**ACKNOWLEDGEMENT**

It is our foremost duty to express our deep sense of gratitude and respect to the guide Prof. Savita P. Patil for his uplifting tendency and inspiring us for taking up this project work successfully. We express my sincere gratitude towards Dr. A. C. Adamuthe, Head of the Department, Information Technology, for providing necessary facilities, guidance and support.

We are thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staff of the Information Technology Department, which helped us in successfully completing our project work. Also, We would like to extend my sincere esteems to all staff in the laboratory for their timely support.

# **Index**

| **Sr No.** | **Description** | **Page No.** |
| --- | --- | --- |
| **1** | **Introduction**  1.1 Motivation for present Work  1.2 Problem Statement  1.3 Research Gaps  1.4 Objectives of present work | **6-7** |
| **2** | **Literature Review** | **8-11** |
| **3** | **Preliminaries**  3.1 Architecture of Implemented Model  3.2 Model Training Essentials | **12-20** |
| **4** | **Implementation**  4.1 Dataset Information  4.2 Dataset Preprocessing/Preparation  4.3 Implementation of Models | **20-33** |
| **5** | **Result & Discussion**  5.1 Key terms of model comparison  5.2 Comparison of models based on performance  5.3 Comparison of models based on computational cost | **34-40** |
| **6** | **Conclusion** | **40** |
| **7** | **References** | **41-42** |
| **8** | **Plagiarism** | **43** |

# 

1. **Introduction**

Diabetes, heart disease, and kidney disease are among the chronic disorders that contribute the most to severe health outcomes as well as mortality across the world. They have the potential to develop slowly over a long period and when diagnosed late can complicate the health status of an individual. Timely prediction and diagnosis are critical to preventing the progression of these diseases that can impact a person's quality of life and in many cases, lead to premature death.

By identifying intricate patterns in medical data, Machine Learning (ML) and Deep Learning (DL) are essential tools for diagnosing and predicting chronic illnesses. Techniques like Long Short-Term Memory (LSTM), Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN) enhance prediction accuracy by identifying various patterns in extensive medical datasets. By leveraging these approaches, predictive healthcare systems can provide early warnings, enabling timely medical interventions and personalized treatment plans.

Cloud computing is essential in contemporary healthcare applications, providing scalable, secure and real-time data processing capabilities. A cloud-based system guarantees that patient data is stored and processed efficiently, making it accessible to healthcare professionals from any location, regardless of geographical barriers. Cloud platforms provide robust security features designed to safeguard data privacy and help meet healthcare regulations.

This project seeks to solve these problems by building a cloud-based application that employs deep learning technology to predict chronic diseases using information from the patients and their previous health records. Because the system will be cloud based, it will be real-time, scalable and available for use by clients from everywhere, which makes the application highly accessible and convenient to use. The deep learning model will be designed to improve the accuracy and reliability of disease predictions, allowing for earlier intervention and personalized healthcare solutions.

* 1. **Motivation for present work**

Chronic diseases such as diabetes, heart problems, and chronic kidney disease continue to be the top culprits behind premature deaths worldwide.This is mainly because these conditions are often only discovered after serious, sometimes irreversible, damage to organs has already taken place. However, recent breakthroughs in deep learning show that neural networks can pick up on subtle, non-linear patterns that are often hidden in routine lab tests, imaging, and long-term health records patterns that traditional statistical methods and human reviews frequently miss. This capability allows healthcare providers to spot at-risk patients months before any symptoms become serious.

To turn these research advancements into practical applications, the proposed system utilizes a variety of architectures Convolutional Neural Networks (CNN), fully connected Artificial Neural Networks (ANN) and deeper DNN stacks and efficient yet powerful vision models like MobileNet and EfficientNet all hosted on a secure, standards-compliant cloud platform. The use of these compact models is essential; their smaller parameter sizes and optimized operations require significantly less computational power than traditional deep-learning frameworks. This means real-time analysis can happen even on less powerful hardware, making high-quality decision support available in primary care offices, rural clinics, and telehealth settings around the world.

* 1. **Problem Statement**

The project aims to develop a cloud-based deep learning system that predicts chronic diseases, enhancing early diagnosis and personalized healthcare interventions.

* 1. **Research Gaps**

Deep learning for healthcare and clinical prediction has made great strides, but there are still a number of important research gaps that restrict the systems generalizability, real-world applicability, and credibility. The main areas that require more research are listed below:

* + 1. **Small, homogeneous or imbalanced datasets**

A lot of research depends on traditional benchmark datasets like the Pima Indians, UCI Heart, or some limited collections from specific institutions. Many image studies have pointed out issues with class imbalance. These datasets often fall short in terms of demographic diversity, lack the depth of longitudinal data, and don’t reflect the latest clinical practices. This raises some serious concerns about potential bias in models and their reliability when applied to wider populations.

* + 1. **Data quality and handling missing values**

Papers often recognize that there are missing or noisy records, but they rarely put in place systematic data-quality pipelines, probabilistic imputation, or uncertainty quantification. The effect of these data gaps on prediction confidence and the best ways to convey that uncertainty to clinicians has not been thoroughly examined.

* + 1. **Computational efficiency versus accuracy trade‑offs**

Lightweight models such as MobileNet and EfficientNet are often celebrated for their low compute costs. However, there’s a noticeable gap in comprehensive studies that directly compare their performance, latency, and energy consumption against heavier models across various tasks. We still need clear guidelines to help choose the best model when working with limited resources.

* + 1. **Security, privacy, and regulatory compliance**

While there are plenty of proposals for cloud or IoMT architectures, there is often find a lack of solid mechanisms to ensure compliance with the rules established by the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA) as well as for implementing federated learning or differential privacy. This ongoing struggle between the need for large-scale data aggregation and the importance of patient confidentiality remains a significant issue.

* 1. **Objectives of present work**
* To prepare a comprehensive dataset for chronic disease prediction.
* To build a deep learning model for chronic disease prediction.
* To develop a chronic disease prediction model on a cloud platform.

1. **Literature Review**

This section presents the critical analysis of research papers focused on machine learning and deep learning based clinical decision support systems for chronic disease predictions.

C. T. Wu *et al.* [1] created a precision health service aimed at preventing and managing chronic diseases through the use of wearable devices, smartphone apps, AI-assisted telecare platforms, and environmental monitoring. To analyze the gathered data, the authors utilized a variety of Deep Neural Networks (DNNs), AdaBoost, random forests, and decision trees as examples of machine learning and deep learning techniques. The study achieved an average accuracy of 88.46%, showcasing the effectiveness of AI in monitoring chronic diseases. One limitation noted in the study was the reliance on wearable devices and smartphone applications, which may hinder adoption among populations without access to such technologies.

M.A.Reshan *et al.* [2] introduced an innovative Ensemble Deep Learning (EDL) clinical decision support system aimed at predicting diabetes. This system integrates ANN, LSTM, and CNN models to deliver impressive accuracy. The research utilized three datasets: the Pima Indian Diabetes Dataset, the diabetes dataset from Frankfurt Hospital in Germany, along with the Iraqi Diabetes Patient Dataset, resulting in an overall accuracy of 98.45%. The implementation was carried out using Scikit-learn, NumPy, Keras, and TensorFlow.

N. Bhaskar *et al.* [3] introduced an automated medical system designed to detect type 2 diabetes through a deep hybrid architecture. This model utilized a Convolutional Neural Network (CNN) alongside a Correlational Neural Network (CORNN) to analyze exhaled breath samples from individuals diagnosed with diabetes. The research highlighted the significance of non-invasive methods for diabetes detection. The model demonstrated impressive accuracy, showcasing the potential of breath analysis in medical diagnostics.

Both studies [2],[3] recognized that issues related to data quality, such as missing or incomplete medical records, could affect the reliability of the models. The combination of CNN, LSTM, and CORNN facilitated improved feature extraction and robust classification, enhancing the prediction and detection of diabetes. These studies played crucial role in advancing diabetes prediction and detection by utilizing deep learning techniques.

M. M. Ali *et al.* [4] proposed an IoT-based framework for detecting Chronic Kidney Disease (CKD) through a deep learning approach. To find the most pertinent features for classification, the authors used the Anova-F feature selection technique. Using a Multi-layer Perceptron (MLP) classifier, the technique achieved a remarkable 99% accuracy rate. The study also compared the effectiveness of several classifiers, such as Support Vector Machines (SVM), Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), and Gated Recurrent Units (GRU), proving the superiority of their methodology.

J. Chaki and A. Ucar [5] Automated kidney stone diagnosis is done efficiently and robustly as presented in paper by utilizing an inductive transfer ensemble based Deep Neural Network (DNN) model. The study was based on the analysis of Computed Tomography (CT) images of kidney stones and utilized datasets including KD, KD1, KD2, and KD3 which were gathered by the Picture Archiving and Communication System (PACS). The system was able to achieve accuracy rates of 99.8% and 96.7%.

K. Venkatrao and S. Kareemulla [6] presented HDLNET, a hybrid deep learning network model that incorporates intelligent IoMT (Internet of Medical Things) for detecting and classifying Chronic Kidney Disease (CKD). The research presented the Deep Separable Convolution Neural Network (DSCNN), which the Sooty Tern Optimization Algorithm was used to refine. The model demonstrated its potential for early CKD detection by achieving an impressive accuracy of 99.18% by utilizing the Chronic Kidney Disease (CKD) dataset.

S. M. M. Elkholy, et al. [7] developed an intelligent classification and prediction model for Chronic Kidney Disease (CKD) detection at an early stage using a modified Deep Belief Network (DBN). The research utilized UCI Chronic Kidney Disease (CKD) Dataset and had a prediction accuracy of 98.50%. The study was done in MATLAB, and a wide range of variables were extracted that included patient demographics, biochemical, and clinical features, like age, blood pressure, specific gravity, albumin levels, sugar content, red blood cells, pus cells, hemoglobin, and so on

G. Chen et al. [8] introduced an Adaptive Hybridized Deep Convolutional Neural Network (AHDCNN) aimed at the early detection of Chronic Kidney Disease (CKD) within the Internet of Medical Things (IoMT) framework. The research evaluated the performance of different deep learning methods and used sample datasets sourced from the DeepLesion repository (https://nihcc.app.box.com/v/DeepLesion). The model reached an FI-Score of 97.3%, demonstrating its strong predictive abilities.

The above studies commonly faced limitations such as constraints on dataset size, regulatory and privacy issues, noise in medical data, and challenges with integration. Despite these hurdles, the proposed algorithms showed impressive predictive accuracy, robustness, and efficiency in detecting Chronic Kidney Disease (CKD), positioning them as valuable tools for early diagnosis and intervention in healthcare environments.

M. Golec *et al.* [9] introduced an innovative healthcare framework powered by AI, which leverages the Internet of Things (IoT) and a serverless computing environment to tackle the issue of heart disease-related deaths. The data for this research came from the UCI ML Repository and was utilized in the LightGBM model that we set up on Google Cloud Platform (GCP) Cloud Functions. The system delivered an impressive prediction accuracy of 91.80%, highlighting its effectiveness in identifying heart disease.

The study presented by S. S. Sarmah [10] introduced an innovative IoT-based system for monitoring patients and predicting heart disease, utilizing a Deep Learning Modified Neural Network (DLMNN). This system verified heart patients from designated hospitals and uses wearable IoT sensor devices affixed to the patient's body to send real-time health information to the cloud for analysis. By leveraging the Hungarian Heart Disease Dataset, the model achieved an impressive accuracy of 98.25% based on a dataset of 500 records. This research highlights the significant potential of IoT-enabled AI systems in enhancing patient monitoring and decision-making in cardiac care.

A. A. Almazroi et al. [11] presented a Clinical Decision Support System (CDSS) built on deep learning techniques for heart disease prognosis. The research employs both DNN and CNN algorithms which use the Hungarian Heart Disease Dataset. The model achieves 83% prediction accuracy and shows improved overall accuracy, sensitivity, and specificity as compared to single models and other ensemble approaches. This work shows promise while predicting there is room for improvement in heart disease patient data image analysis.

Y. Pan et al. [12] presented an Enhanced Deep Learning Assisted Convolutional Neural Network (EDCNN) system that operates on the Internet of Medical Things (IoMT) platform for predicting heart disease. This system acts as a decision support tool, allowing doctors to effectively diagnose heart-related issues through globally accessible cloud-based platforms. The research utilized the Hungarian Heart Disease Dataset and integrated various deep learning models, such as ANN, DNN, RNN, and EDCNN, achieving an impressive accuracy rate of 99.1%. Age, gender, the type of chest pain, blood pressure, cholesterol, blood sugar, maximum heart rate, exercise habits, OldPeak and the target variable were the primary patient characteristics that we examined.

S. N. Ali et al. [13] presented an end-to-end deep learning framework aimed at real-time denoising of heart sounds (phonocardiograms, PCGs) to improve the detection of cardiac diseases in noisy settings. Bi-LSTM (Bidirectional Long Short-Term Memory) networks were used in the study, and the PASCAL Heart Sound dataset and the 2016 PhysioNet/CinC Heart Sound (PHS) dataset were used to evaluate performance. This framework tackled the challenge of environmental and physiological noise that can distort heart sound signals recorded with digital stethoscopes, potentially affecting their important and critical features.

A common limitation noted in these studies was the absence of adequate security and privacy measures, which could compromise the protection of patient data. The deep learning models used in these studies offered notable benefits compared to traditional machine learning methods. For example, LightGBM and DLMNN showed remarkable efficiency in managing large datasets while still achieving high prediction accuracy.

Chakraborty and Kishor [14] proposed a real time, cloud-based monitoring system, which uses computational health systems to predict heart disease. A variety of machine learning classification algorithms, including K-Nearest Neighbors (K-NN), Support Vector Machine (SVM), Random Forest (RF), and Artificial Neural Networks (ANN), were used in their investigation. The Internet of Medical Things (IoMT) was used to gather the dataset for this study, which concentrated on vital health indicators such as blood pressure, blood sugar, heart rate, temperature, and oxygen levels.

M. A. Khan and F. Algarni [15] proposed a Healthcare Monitoring System for Diagnosing Heart Disease in the IoMT Cloud Environment Using Modified Salp Swarm Optimization (MSSO) and Adaptive Neuro-Fzzy Inference System (ANFIS) introduced a framework based on the Internet of Medical Things (IoMT). The system was trained using the Hungarian Heart Disease Dataset and the Framingham Heart Disease Dataset, achieving an impressive accuracy rate of 99.45%.The study emphasized the advantages of MSSO in both exploration and exploitation, while also noting its limitations, such as slow and premature convergence.

Cloud computing significantly contributed to these studies [14],[15] by offering a flexible infrastructure for handling and storing vast amounts of health data. It facilitated real-time access, remote monitoring, and effective computational analysis, leading to improved prediction accuracy. By incorporating cloud-based IoMT systems, healthcare professionals were able to improve early diagnosis and intervention, ultimately lowering mortality rates associated with heart disease.

A. Sundas et al. [16] created a machine learning-based model for predicting Chronic Kidney Disease (CKD), which is integrated with a user-friendly web application. To identify CKD, they used a variety of machine learning techniques, such as Support Vector Machines (SVM), Artificial Neural Networks (ANN) and K-Nearest Neighbors (KNN). The model was trained on a dataset that included several clinical parameters such as age, blood pressure, random blood glucose levels, hemoglobin, serum creatinine, and other relevant biomarkers for kidney health. To ensure easy access and real-time predictions, the authors utilized the Flask framework to develop the web application.

R. K. Haldera et al. [17] introduced a Smart Patient Monitoring and Recommendation (SPMR) framework that utilizes deep learning and cloud-based analytics for managing chronic diseases. This framework uses Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) to monitor patients with chronic blood pressure problems. The research used a dataset on Chronic Blood Pressure Disorders collected through Ambient Assisted Living (AAL) devices. The system achieved impressive accuracy (99.96%) and an F1-score between 0.91 and 0.97.

The limitation across these studies was the narrow application of the models to specific diseases, highlighting the need for broader validation across various chronic conditions. Additionally, there was a risk of overfitting, which could be addressed through hyperparameter tuning and regularization techniques. The findings indicated that deep learning models, especially CNN and RNN, excelled in feature extraction, while traditional machine learning algorithms like SVM and KNN were effective for classification with lower computational demands.

A. A. Ali Tabtaba and Oguz Ata [18] introduced a new model for detecting Diabetic Retinopathy (DR) that utilizes a hybrid heuristic-aided deep learning approach. Before feeding the fundus photos into a Hybrid Cascaded Multi-scale Deep Convolutional Neural Network (HCMD-CNN), they gathered and pre-processed them. This model combined Region Attention Networks (RAN) and MobileNet to effectively extract features. The training and evaluation were conducted using the IDRiD dataset, and the model achieved an accuracy of 91%.

R. O. Ornelasa *et al.* [19] proposed the use of MobileNetV2 was investigated for the early detection of lung cancer through a thorough transfer learning approach. Using the LC25000 dataset, this model successfully classified lung adenocarcinoma (LAC), lung squamous cell carcinoma (LUSC), and benign lung tissue (BLT). Additional National Cancer Institute histopathological images were added to improve generalization. A high classification accuracy of 98.77% was attained by the model. Notwithstanding this remarkable accuracy, the study recognized certain drawbacks, such as dataset problems.

There are some limitations pointed in both studies, Diabetic Retinopathy (DR) detection model had trouble with optimal feature extraction, while the lung cancer detection model encountered difficulties due to an imbalanced dataset. Nevertheless, the studies showcased the benefits of employing deep learning-based models like MobileNet and HCMD-CNN, which offered high accuracy and efficient computation for analyzing medical images.

1. **Preliminaries**

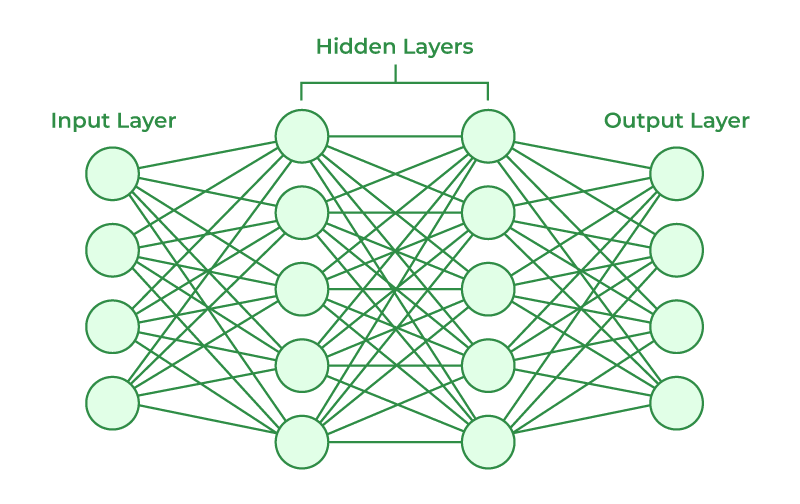
An overview of deep learning models, including Artificial Neural Networks (ANNs), Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), MobileNet, and EfficientNet, is provided in this section. Each model is thoroughly explained, emphasizing its fundamental structure, important layers and working principles to give readers a clear grasp of how these models work and are used in real-world scenarios. Also it described the various implementation and optimization strategies used to enhance model performance in addition to going over the fundamental architecture of each model. Data preprocessing, regularization techniques, hyperparameter tuning, model pruning, transfer learning, and the use of lightweight models for real-time and mobile applications are a few of these.

* 1. **Architecture of Implemented Model**

This section gives the summary of the various kinds of deep learning models and their architectures. MobileNet, EfficientNet, Convolutional Neural Networks (CNNs), Deep Neural Networks (DNNs), and Artificial Neural Networks (ANNs). Each model's architecture comprises various layers that are thoroughly explained to aid in understanding their practical application and underlying principles.

* + 1. **Architecture of ANN model**

The Artificial Neural Network (ANN) is a computational model that is modeled after the biological neural networks that are present in the human brain. It is composed of many neurons, which are linked processing units that work together to solve specific problems.

Fig 3.1 Architecture of ANN 

As shown in Fig 3.1 there are three different kinds of layers that make up an ANN's basic architecture i.e input, hidden and output layers. Raw data is sent to the following layers by the input layer. Weighted connections and activation functions are used by the hidden layers which may consist of one or more to carry out intricate calculations, adding non-linearity to the model. Lastly, based on the calculations, the output layer generates the final classification or prediction. Information must move forward (forward propagation) in order for an ANN to function.

**Layers in ANN**

Artificial Neural Networks (ANNs) are implemented using three types of layers:

1. Input Layer :

* This is a stage where data is entered into model
* Every node in the input layer of structured data represents a distinct feature (e.g., temperature, income, or age).

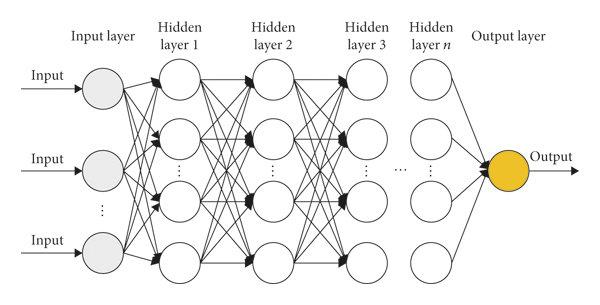
1. Hidden Layer :

* ANN typically consists of one or two hidden layers.
* Every hidden layer (dense layer) is completely connected to the one before it and non-linearity is usually introduced using activation functions like sigmoid or ReLU (Rectified Linear Unit).

1. Output Layer :

* For Binary classification : A single neuron with sigmoid activation is utilized.
* For Multiclass classification : Multiple neurons with softmax activation are used.
* For regression: a single neuron with linear activation.
  + 1. **Architecture of DNN model**

With a greater number of hidden layers, a Deep Neural Network (DNN) is a more advanced kind of Artificial Neural Network (ANN) that can recognize more intricate patterns in data.

Fig 3.2 Architecture of DNN model

The architecture of a DNN is defined by its depth, which means multiple hidden layers and output layers as shown in Fig 3.2. DNN consists of multiple dense layers using different activation functions and incorporated batch normalization to speed up training and enhance stability. The network's depth facilitates hierarchical feature learning, which is essential for uncovering subtle connections in health indicators. DNN proved to be especially effective in achieving higher accuracy with large and varied datasets.

**Layers in DNN model**

Both ANN and DNN follow a similar structure in terms of layer types. Deep Neural Network (DNN) has several hidden layers in between the input and output layers. A DNN layer’s consists:

1. Input Layer

* The data is fed into the network at this initial layer.
* Every node in the input layer for structured data represents a distinct feature (such as blood pressure or age).

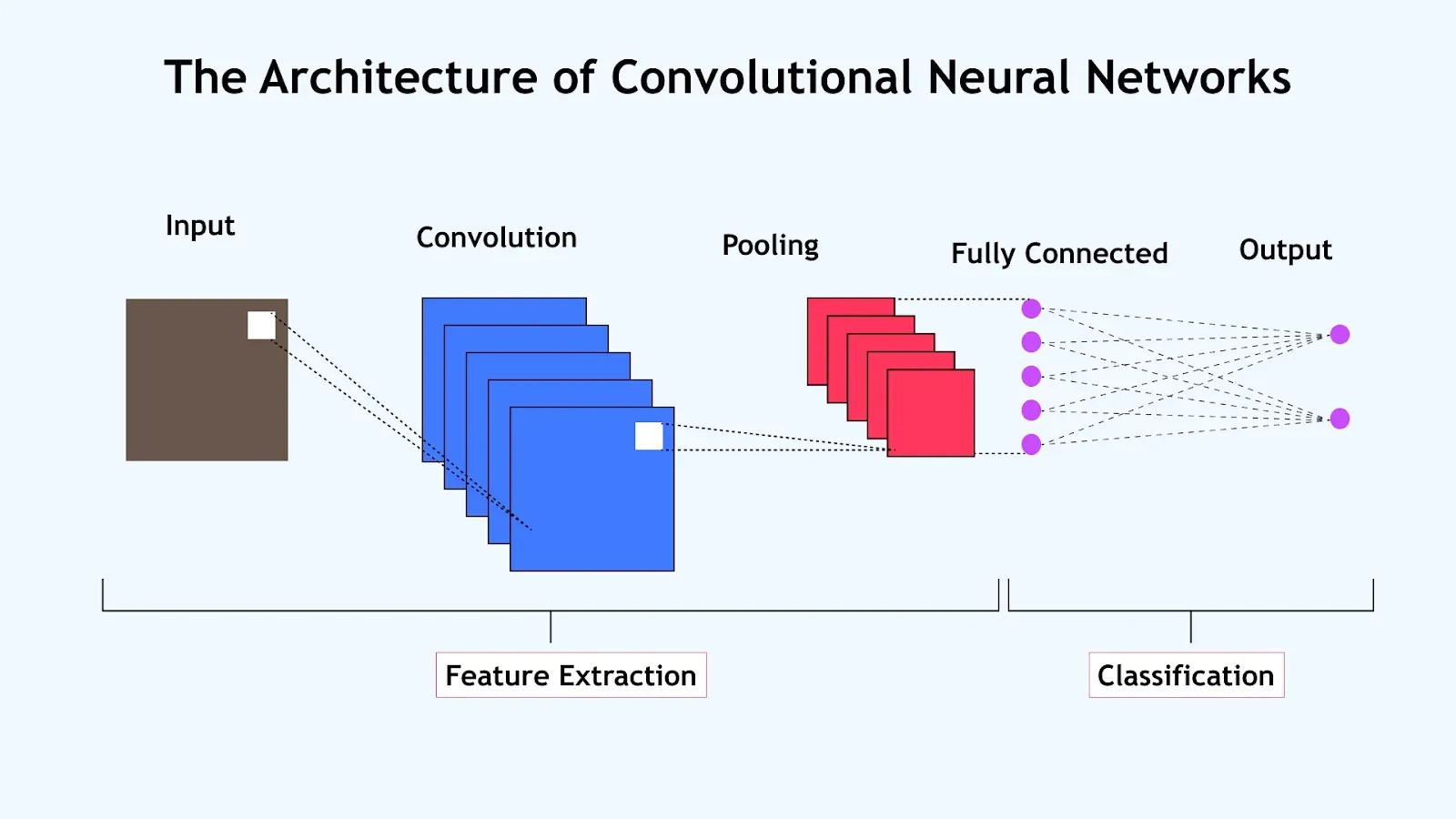
1. Hidden Layers

* DNN Features multiple hidden layers, showcasing a deeper architecture.
* Depending on the complexity of the model, DNNs can have dozens or even hundreds of hidden layers.

1. Output Layer

* The model's final output is provided by the output layer, whose structure is determined by the task's requirements.
* For binary classification: a single neuron with sigmoid activation.
* For multi-class classification: several neurons using softmax activation.
  + 1. **Architecture of CNN model**

Convolutional Neural Networks (CNNs), a special type of deep learning model, are primarily used to process data with a grid-like topology, such as images. CNNs employ specialized layers that are able to automatically identify spatial hierarchies and patterns (such as edges, textures, and shapes) in data, in contrast to traditional DNNs, which mainly rely on fully connected layers. A CNN's architecture typically consists of convolutional layers, pooling layers, activation functions and fully connected layers.

Fig 3.3 Architecture of CNN

As shown in Fig. 3.3, the typical elements of a CNN's architecture are convolutional layers, pooling layers, activation functions and fully connected layers. In the early layers, the CNN initially records low-level features (colors, edges). The CNN architecture began with one or more convolutional layers that applied various filters to the input images, picking up on low-level features like edges and patterns among parameters. It learns increasingly intricate and abstract features (object parts, shapes) as the data passes deeper through the network, ultimately producing precise predictions. In order to modify the filters and weights according to the loss, the model is trained using backpropagation with gradient descent. The output was flattened and funneled through one or more fully connected dense layers, where we could learn deeper associations between features. This structured approach enabled the CNN to effectively grasp spatial dependencies within the health data.

**Layers in CNN model**

1. Input Layer

* The input layer receives images (or other grid-structured data) raw pixel values.
* For instance, a 64x64 image with three RGB color channels would be entered as a 64×64×3 tensor.

1. Convolutional Layer

* The fundamental components of a CNN are convolutional layers.
* In a convolutional layer, a collection of learnable filters, or kernels, traverse the input data to extract important features like edges, corners, or textures.

1. Pooling Layers

* These layers are used to minimize the feature maps' width and height while maintaining the most crucial data.
* Max Pooling, the most popular kind, chooses the highest value from a feature map region.
* This helps to increase the network computational efficiency.

1. Fully Connected Layers (Dense Layer)

* The high-level features that were taken out of the input are compressed into a one-dimensional vector and then sent through one or more fully connected layers following a number of convolutional and pooling layers.\
* These layers use an learned features to carry out the final classification or regression task.

1. Output Layer

* For classification tasks, it typically uses a softmax activation function (for multi-class classification) or sigmoid (for binary classification).
* A linear activation can be applied to regression tasks.
  + 1. **Architecture of MobileNet model**

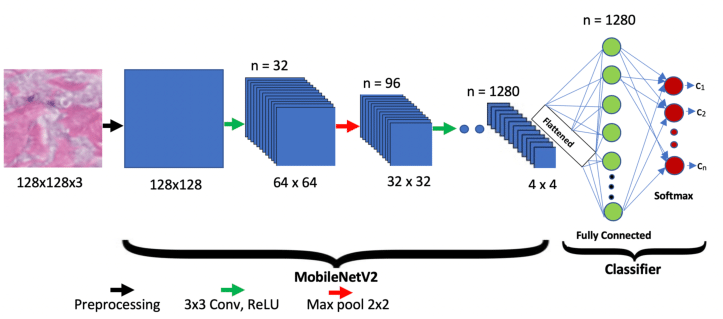
MobileNet is a lightweight, effective deep neural network architecture, created especially for embedded and mobile vision applications with constrained computational resources (such as memory and processing power). 

Fig 3.4 Architecture of MobileNet

Instead of applying a single convolution operation across all input channels, as shown in Fig 3.4 MobileNet first performs a depthwise convolution to filter each input channel individually and a pointwise convolution, or 1×1 convolution, is then performed to combine the outcomes. Toward the end, a global average pooling layer is added before the final fully connected layer, which uses softmax or sigmoid activation for classification. This modular and lightweight design allows MobileNet to strike a great balance between speed, memory efficiency, and accuracy, making it an excellent choice for healthcare applications on low-power devices.

**Layers in MobileNet Model**

1. Input Layer:

* Images or structured grid data provide raw pixel values to the input layer.
* For instance, a 224×224×3 tensor is used to input a 224×224×224 image with three RGB color channels.

1. Depthwise Convolution Layer

* MobileNet employs depthwise separable convolutions in place of conventional convolutions.
* The first step is depthwise convolution, in which a single filter is applied independently to each input channel.

1. Pointwise Convolutional Layer

* A pointwise convolution, also known as a 1x1 convolution, is applied following the depthwise convolution.
* In order to create new feature representations, this step combines outputs from the depthwise layer to mix information across channels.

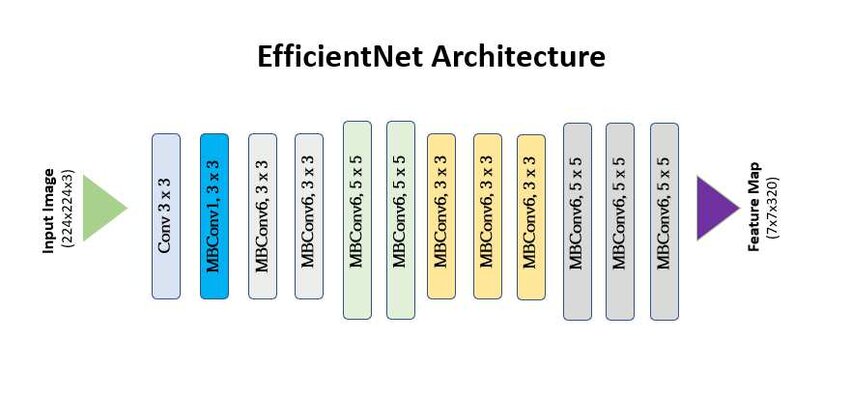
1. Downsampling

* Instead of using separate pooling layers, MobileNet commonly uses strided convolutions to reduce the spatial dimensions (width and height) of the feature maps.
* Global Average Pooling, which reduces each feature map to a single number, is typically used for pooling at the network's end.

1. Dense Layer

* The fully connected layer, also known as the dense layer, flattens (or averages) the high-level extracted features into a one-dimensional vector following a number of depthwise separable convolution blocks.
* A fully connected layer then processes this vector, combining the learned features to produce final predictions.
  + 1. **Architecture of EfficientNet model**

Google AI created the EfficientNet deep learning model, which aims to be both computationally efficient (faster and lighter) and highly accurate. EfficientNet's new scaling method uniformly scales the depth, width, and resolution of a network using a compound coefficient. EfficientNet achieves better performance with fewer parameters and FLOPS (Floating Point Operations) by scaling the network's width (more channels) and depth (more layers) simultaneously.

Fig 3.5 Architecture of EfficientNet

As shown in Fig 3.5 EfficientNet model processes input images through a designed sequence layer each with a distinct function in feature extraction and refinement. The input layer is where it starts, and it takes pictures that are usually 224 x 224 x 3 in size. To prepare the data for further analysis, the first operation is a stem convolution layer, which rapidly decreases the spatial dimensions and increases the number of channels using a 3×3 convolution with a stride of 2. The efficiency-optimized MBConv (Mobile Inverted Bottleneck Convolution) blocks form the foundation of EfficientNet. Following each of these steps, training is stabilized using batch normalization, and the network is able to learn more intricate patterns by introducing non-linearity through the use of a Swish activation function. After a fully connected (dense) layer has interpreted the high-level features, this output is sent to the output layer, which, depending on the task, generates the model's final prediction using a softmax (or sigmoid/linear) activation.

**Layers in EfficientNet**

1. Input Layer :

* Takes in the unprocessed input picture.
* Typical input is an image with dimensions of 224 x 224 x 3 (height, width, and three RGB color channels).
* Function: Transforms the image into a tensor so the network can process it.

1. Stem Convolutional Layer

* It is a 3x3 standard convolution with stride 2
* The goal is to increase the depth (number of channels) while decreasing the input image's spatial dimensions.
* Function: Executes downsampling and initial feature extraction. aids in condensing the data so that it can be processed further.

1. MBConv Blocks

* Expansion Layer: To enable more feature extraction by increasing the number of channels (depth).
* Depthwise Convolution: Independently applies a spatial convolution to every channel.
* Squeeze and Excitation Layer : Draws attention by assigning a weight to each channel's significance.

1. Output Layer

* To generate a final prediction based on the task.
* For classification tasks, it uses either a softmax activation function (for multi-class classification) or a sigmoid (for binary classification).
  1. **Model Training Essentials**

Understanding essential elements such as loss functions, activation functions, data preprocessing, and output transformation techniques is necessary for training a deep learning model that is both accurate and effective. The way the model learns, predicts, and generalizes to new data is determined by these components taken together.

* + 1. **Loss Functions and Error Metrics**

Loss function used to measure how well the model’s prediction aligns with correct values. Reducing this loss is the aim of training.

1. Binary Cross-Entropy:

* Used for binary classification tasks.
* Formula: - [y log(p) + (1 - y) log(1 - p)]
* Output activation: sigmoid

1. Categorical Cross-Entropy:

* Designed for multi-class classification.
* Requires one-hot encoded labels.
* Output activation: softmax

1. Mean Squared Error (MSE):

* Applicable in regression scenarios.
* Formula: MSE = (1/n) \* Σ (y\_true - y\_pred)^2
  + 1. **Output Activation Functions**

Output activation functions determine how the final output of a neural network is transformed, depending on the type of task (classification or regression). These functions map the network's raw output (called logits) to a form suitable for interpreting predictions.

1. Sigmoid Activation (Binary Classification):

* Produces an output between 0 and 1.
* Ideal for scenarios with just two classes.

1. Softmax Activation (Multi-Class Classification):

* Generates probabilities for various classes.
* The class with the highest probability becomes the prediction.

1. Linear Activation (Regression):

* Utilized when the output is a continuous value.

1. Tanh

* Output range: (-1, 1).
* It's an improvement over sigmoid for data that’s centered.
* Yet, it still faces the vanishing gradient challenge.

1. ReLU (Rectified Linear Unit)

* Formula: f(x)=max(0,x)
* This is the go-to choice for hidden layers.
* It's quick, efficient, and helps tackle the vanishing gradient issue.

1. Swish (Especially in EfficientNet):

* 𝑓(𝑥) = 𝑥 ⋅ sigmoid(𝑥)
* It smooths things out and boosts performance.
  + 1. **Multiple Output to Binary Conversion Techniques**

When a model produces multiple classes or labels, you may need to convert them into a binary format. Here are a couple of techniques to do just that:

1. One-vs-Rest (OvR) / One-vs-All Encoding:

This method involves training a separate binary classifier for each class, allowing it to distinguish that class from all the others. It's a popular choice for converting multi-class outputs into binary.

1. Label Binarizer / One-Hot Encoding:

This technique transforms multi-class labels into a binary matrix. For instance, if you have four classes, Class 2 would be represented as [0, 0, 1, 0].

1. Thresholding in Multi-Label:

In cases where you're dealing with multi-label problems and using sigmoid outputs, you can apply a threshold (like 0.5) to turn those predictions into binary values (0 or 1).

* + 1. **Data Normalization and Scaling Techniques**

Normalization and scaling are data preprocessing techniques. They ensure that each feature contributes equally to the training process and improves convergence speed

1. StandardScaler (Z-Score Normalization)

* Formula: z = (x - 𝜇)
* Centers data around mean = 0 and standard deviation = 1.
* Useful when features follow a Gaussian distribution.

1. MinMaxScaler

* Scales data to a fixed range, usually [0, 1].
* Maintains shape of original distribution.

1. MaxAbsScaler

* Scales data between [-1, 1] by dividing by maximum absolute value.
* Useful for sparse data (many zeros).

1. RobustScaler

* Uses median and interquartile range.
* Not sensitive to outliers.
* Best when data contains extreme outliers.

1. Normalizer

* Scales each data point (row-wise) to have unit norm.
* Formula (L2 norm): x/∥x∥2​
* Useful in text classification or cosine similarity-based models.

1. **Development and Implementation**

In this section, in order to predict heart disease, various deep learning models are used. The section also presents data on three datasets: diabetes, kidney disease and heart disease. The preprocessing procedures used on these datasets are also covered in detail, as are the different deep learning models that were employed for prediction.

* 1. **Dataset Information**

Diabetes, kidney disease, and heart disease are the three disease datasets that are described. Comprehensive details about these datasets are provided in this section, including each dataset's size, total number of features, and missing values in each feature.

* + 1. **Heart Disease dataset**

For the Chronic Disease prediction system, the dataset which is used is Heart Disease dataset from the UCI Machine Learning Repository. Based on a number of medical characteristics, this dataset is frequently used to predict whether a patient has heart disease or not. The dataset originates from the Cleveland database and contains 14 attributes related to clinical and personal patient data. Data from four countries Cleveland, USA; Hungary; Switzerland; and the VA Long Beach, USA make up this dataset. Although the original database contains 76 attributes, nearly all published experiments and machine learning research focus on a selected subset of 14 attributes as outlined in Table 4.1, especially from the Cleveland subset, which is the most processed and commonly used in ML studies.

Dataset Overview:

* Total Records: 920 patients
* Total Features: 16 columns
* Missing Values: Present in several columns

Key Features in Dataset:

Table 4.1 Details of key features in Heart Disease Dataset

| Feature Name | Description | Data Type | Missing Values |
| --- | --- | --- | --- |
| id | unique identifier for each patient | Integer | 0 |
| age | Age of patient (years) | Integer | 0 |
| sex | Sex of patient (Male/Female) | Categorical | 0 |
| dataset | Dataset origin/source | Categorical | 0 |
| cp | Type of chest pain (Typical angina, asymptomatic, etc) | Categorical | 0 |
| trestbps | Resting blood pressure (mm Hg) | Float | 59 |
| chol | Serum cholesterol level (mm/dg) | Float | 30 |
| fbs | Fasting blood sugar > 120 mg/dl (True/False) | Categorical | 90 |
| restecg | Resting electrocardiographic results | Categorical | 2 |
| thalch | Maximum heart rate achieved | Float | 55 |
| exang | Exercise-induced angina (Yes/No) | Categorical | 55 |
| oldpeak | ST depression induced by exercise relative to rest | Float | 62 |
| slope | Slope of the peak exercise ST segment | Categorical | 309 |
| ca | Number of major vessels (0–3) colored by fluoroscopy | Float | 611 |
| thal | Type of defect (normal, fixed, reversible) | Categorical | 486 |
| num | **Target variable** – Presence of heart disease (0 = No disease, 1–4 = Level of Disease) | Integer | 0 |

* + 1. **Chronic Kidney Disease dataset**

This dataset was gathered from a hospital setting over the course of about two months and is intended for the classification task of predicting Chronic Kidney Disease (CKD). In addition to a target variable, it comprises data for 400 patients and 26 features, all of which are categorical or real-valued attributes that represent different clinical parameters. Because the dataset is multivariate, it records several variables for every observation. Age, blood pressure (bp), urine specific gravity (sg), albumin (al), sugar levels (su), and a variety of biochemical markers like blood glucose random (bgr), blood urea (bu), and serum creatinine (sc) are all characteristics that correlate to commonly measured indicators during medical assessments as detailed in Table 4.2. Observations like red blood cells (RBC), pus cells (PC), and hypertension (HTN) are examples of categorical features.

Key Information :

* Total records : 400
* Total features: 26
* Target variables: classification (binary classification)

Table 4.2 Details of key features in Kidney Disease Dataset

| Feature Name | Description | Data Type | Missing Values |
| --- | --- | --- | --- |
| id | Patient ID | Integer | 0 |
| age | Age (in years) | Float | 9 |
| bp | Blood Pressure (in mm/Hg) | Float | 12 |
| sg | Specific Gravity | Float | 47 |
| al | Albumin | Float | 46 |
| su | Sugar | Float | 49 |
| rbc | Red Blood Cells (Normal/Abnormal) | Categorical | 152 |
| pc | Pus Cell (Normal/Abnormal) | Categorical | 65 |
| pcc | Pus Cell Clumps (Present/Not Present) | Categorical | 4 |
| ba | Bacteria (Present/Not Present) | Categorical | 4 |
| bgr | Blood Glucose Random (in mgs/dl) | Float | 44 |
| bu | Blood Urea (in mgs/dl) | Float | 19 |
| sc | Serum Creatinine (in mgs/dl) | Float | 17 |
| sod | Sodium (in mEq/L) | Float | 87 |
| pot | Potassium (in mEq/L) | Float | 88 |
| hemo | Hemoglobin (in gms) | Float | 52 |
| pcv | Packed Cell Volume | Categorical | 70 |
| wc | White Blood Cell Count | Categorical | 105 |
| rc | Red Blood Cell Count | Categorical | 130 |
| htn | Hypertension (Yes/No) | Categorical | 2 |
| dm | Diabetes Mellitus (Yes/No) | Categorical | 2 |
| cad | Coronary Artery Disease (Yes/No) | Categorical | 2 |
| appet | Appetite (Good/Poor) | Categorical | 1 |
| pe | Pedal Edema (Yes/No) | Categorical | 1 |
| ane | Anemia (Yes/No) | Categorical | 1 |
| classification | CKD or not (ckd/notckd) | Categorical | 0 |

* + 1. **Pima Indians Diabetes dataset**

The Pima Indians Diabetes Dataset, which is widely used in the medical field for research and predictive modeling, is sourced from the National Institute of Diabetes and Digestive and Kidney Diseases. Using a variety of health-related diagnostic metrics, this dataset focuses on the task of diagnosing diabetes in Pima Indian women who are 21 years of age or older. It contains several predictor variables, such as age, blood pressure, body mass index (BMI), insulin levels, number of pregnancies, and others listed in Table 4.3, in addition to one target variable, outcome, which indicates whether a patient has diabetes (1) or not (0).

Key Information:

* Total Records: 768
* Total Features: 9
* Target Variable: Outcome (Binary classification - 0 indicates no diabetes, 1 indicates diabetes)

Table 4.3 Details of key features in Diabetes Disease dataset

| Feature Name | Data Type | Missing Values |
| --- | --- | --- |
| Age | Integer | 0 |
| BMI | Float | 11 |
| Blood Pressure | Integer | 35 |
| Diabetes Pedigree Function | Float | 0 |
| Glucose | Integer | 5 |
| Insulin | Integer | 374 |
| Outcome | Integer | 0 |
| Pregnancies | Integer | 0 |
| SkinThickness | Integer | 227 |

* 1. **Dataset Preprocessing / Preparation**

Here, the Heart Disease dataset was preprocessed to make sure the dataset was clean, consistent and ready for deep learning models, several preprocessing steps were applied. These steps tackled issues like missing values, inconsistent data types, categorical variables, and the formatting of target labels. The main aim was to boost data quality and enhance the performance of the model.

* + 1. **Analyzing the Dataset**

The first step involved loading and taking a good look at the dataset using functions like head(), info(), and describe(). This step was crucial for understanding the dataset's structure, figuring out types of variables, spotting any missing values, and assessing the overall quality of the data. It also gave us a quick snapshot of the numerical and categorical features, how outliers were distributed, and the balance of classes for our target variable.

* + 1. **Handling Missing Values**

The columns that had missing data like oldpeak, trestbps, thalach, and chol were filled in using the median value for each one. The median was selected over the mean because it’s better for handling skewed distributions or outliers, giving us a more reliable measure of central tendency.

df\_clean['ca'].fillna(df\_clean['ca'].median(), inplace=True)

df\_clean['thal'].fillna(df\_clean['thal'].median(), inplace=True)

df\_clean['slope'].fillna(df\_clean['slope'].median(), inplace=True)

for col in ['oldpeak', 'trestbps', 'thalch', 'chol']:

df\_clean[col].fillna(df\_clean[col].median(), inplace=True)

df\_clean['exang'].fillna(df\_clean['exang'].mode()[0], inplace=True)

for col in ['fbs', 'exang', 'restecg']:

df\_clean[col].fillna(df\_clean[col].mode()[0], inplace=True)

* + 1. **Verifying Clean Data**

After taking care of the missing values and removing any columns that weren't relevant, a final check was performed using isnull().sum() to make sure there were no leftover missing values in the dataset. This step confirmed that our dataset was clean and all set for the next stages of transformation and modeling.

print(" Missing Values:\n")

print(df\_clean.isnull().sum().sort\_values(ascending=False))

* + 1. **Converting Categorical Values into Numerical Format**

Categorical variables such as sex, chest pain type (cp), fasting blood sugar (fbs), and restecg and turned them into numerical formats. Binary variables were kept as 0s and 1s, while multi-class categorical features were label-encoded based on what they are. This transformation is crucial because machine learning & deep learning algorithms usually need numerical input to function properly.

thal\_mapping = {

'normal': 1,

'fixed defect': 2,

'reversable defect': 3

}

df\_clean['thal'] = df\_clean['thal'].map(thal\_mapping)

slope\_mapping = {

'upsloping': 1,

'flat': 2,

'downsloping': 3

}

df\_clean['slope'] = df\_clean['slope'].map(slope\_mapping)

* + 1. **Converting Multi-Class Output into Binary Format**

In certain datasets, the target variable (num) can have several class labels, like 0, 1, 2, 3, and 4, which show how severe the disease is. When it comes to binary classification deciding whether heart disease is present or not this variable is converted into a binary format. A value of ‘0’ means there’s no heart disease and ‘1’ means heart disease is present, regardless of how severe it is. This change makes the modeling process easier and is a common approach in clinical diagnostic predictions.

df\_clean['sex'] = LabelEncoder().fit\_transform(df['sex'])

df\_clean['fbs'] = LabelEncoder().fit\_transform(df['fbs'])

df\_clean['restecg'] = LabelEncoder().fit\_transform(df['restecg'])

df\_clean['target'] = df\_clean['num'].apply(lambda x: 1 if x > 0 else 0)

* + 1. **Creating the Target Column and Dropping the Original Label Column**

A new binary target column was created based on the existing num column to serve as the final output label for our predictions. After this transformation, the original num column was removed from the dataset to keep things tidy and avoid any redundancy.

num\_distribution = df\_clean['num'].value\_counts().sort\_index()

for level, count in num\_distribution.items():

if level == 0:

print(f"{level} → {count} instances (No disease)")

else:

print(f"{level} → {count} instances")

df\_clean['target'] = df\_clean['num'].apply(lambda x: 1 if x > 0 else 0)

df\_clean.drop(columns=['num'], inplace=True)

* 1. **Implementation of models**

The various preprocessing procedures were applied to the datasets will described in the section on the implementation of models. Following that, the post-preprocessing implementation steps that are unique to each deep learning model after the preprocessing will be discussed. These procedures are customized based on the type of input data needed for each model, such as image-based data for CNN, MobileNet, and EfficientNet models, or structured tabular data for models like ANN and DNN. Next, the deep learning architecture of each model used in this project will be examined.

* + 1. **Post-Preprocessing Implementation Steps**

After data cleaning, handling missing values encoding categorical variables, and normalization, followed specific steps tailored to each deep learning model based on what kind of input it needed whether it was structured data for models like ANN and DNN or image-based data for CNN, MobileNet, and EfficientNet.

1. **For ANN and DNN Models (Structured Tabular Data)**
2. **Splitting Data:**

The cleaned dataset was divided into training and testing subsets using an 80:20 ratio. This ensured that the model could be trained using training data while testing data used for performance evaluation.

x = df.drop("target", axis=1).values

y = df["target"].values

1. **Feature Scaling:**

Feature scaling was done using StandardScalar or RobustScalar to normalize the input data so that all features contribute equally.

scaler = RobustScaler()

x\_train\_scaled = scaler.fit\_transform(x\_train)

x\_test\_scaled = scaler.transform(x\_test)

1. **Model Architecture:**

ANN: Implemented using a Sequential model that features dense layers, ReLU activations, and dropout for regularization, topped off with a final sigmoid layer for binary classification.

DNN: It’s like the ANN but goes deeper, incorporating batch normalization and dropout to help stabilize and regularize the learning process.

1. **For CNN, MobileNet and EfficientNet (Image-Based Models)**
2. **Data conversion to Image Format:**

The tabular feature vectors were reshaped into 2D grayscale arrays using NumPy and stored as synthetic images. These images were saved and categorized based on target class.

1. **Image loading and Augmentation**

Images were loaded using ImageDataGenerator in Keras for memory-efficient training. Basic augmentations like rescaling, zoom, and rotation were applied to improve generalization.

1. **Image Resizing:**

Images were resized as per model requirements:

CNN: 32×32

MobileNet: 224×224

EfficientNetB0: 224×224

1. **Model Architecture and Training:**

CNN: Built from scratch with Conv2D, MaxPooling2D, Flatten, and Dense layers.

MobileNet: Used pre-trained base with custom dense layers on top for classification (Transfer Learning).

EfficientNetB0: Loaded as a pre-trained base with GlobalAveragePooling2D and output dense layer added for fine-tuning

1. **Optimization:**

All models were compiled using the Adam optimizer, trained with binary cross-entropy loss, and evaluated using metrics like accuracy, precision, recall, and F1-score.

* + 1. **Deep Learning Architecture of models**

Each model's deep learning architecture used in this project will be examined. The setup and architecture of CNN, MobileNet and EfficientNet which are better suited for image-based data are then explained, after which the ANN and DNN models were created to process structured data.

1. **Artificial Neural Network**

The Artificial Neural Network (ANN) model is a sequential deep learning architecture used for binary classification in the context of heart disease prediction. The model has several hidden layers, each containing 256, 128, 64, and 32 neurons.

model = Sequential([

Dense(256, kernel\_regularizer=l2(0.001),

input\_shape=(x\_train.shape[1],)), # Increased neurons

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.2),

Dense(128, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.2),

Dense(64, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.2),

Dense(32, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.2),

Dense(1, activation='sigmoid') # Output layer for binary classification

])

* Incorporated 4 hidden layers with 256, 128, 64, and 32 neurons.
* To enhance performance and prevent dead neurons, LeakyReLU activation was used.
* After each layer, Batch Normalization was added to stabilize the learning process and speed up training.
* To combat overfitting, dropout layers were included to randomly deactivate some neurons during training.
* L2 regularization was also applied to further minimize overfitting by penalizing large weights.
* Finally, a sigmoid activation function was used in the output layer to make binary predictions (0 or 1).

1. **Deep Neural Network**

With six hidden layers that contain 512, 256, 128, 64, 32, and 16 neurons each, the Deep Neural Network (DNN) model has a deeper architecture than the ANN. The purpose of this model is to identify more intricate patterns in the data.

model = Sequential([

Dense(512, kernel\_regularizer=l2(0.001), input\_shape=(X\_train.shape[1],)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(256, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(128, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(64, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(32, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(16, kernel\_regularizer=l2(0.001)),

LeakyReLU(alpha=0.1),

BatchNormalization(),

Dropout(0.3),

Dense(1, activation='sigmoid')

])

* A deep learning model featuring six hidden layers, with each layer containing 512, 256, 128, 64, 32, and 16 neurons, respectively.
* To tackle the dying neuron issue and ensure a smoother training experience, LeakyReLU activation with an alpha value of 0.1 was used in every hidden layer.
* After each dense layer, Batch Normalization was implemented to standardize the inputs for the next layer, which helps stabilize and speed up the training process.
* To combat overfitting and improve generalization, Dropout layers with a dropout rate of 0.3 were added, which randomly turns off neurons during training.
* L2 regularization (with λ = 0.001) was applied to each dense layer, which discourages large weights and promotes a simpler model that’s less likely to overfit.
* For the final output layer, a single neuron with a sigmoid activation function was used to produce a binary prediction indicating whether the patient is likely to have heart disease (1) or not (0).

1. **Conventional Neural Network**

In order to process structured data, the Conventional Neural Network (CNN) model transforms it into a format that can be used for 1D convolution. The input data is first reshaped, and three convolutional blocks with 64, 128 and 256 filters, respectively, are applied.

x\_scaled = x\_scaled.reshape((x\_scaled.shape[0], x\_scaled.shape[1], 1))

model = Sequential([

Input(shape=(x\_train.shape[1], 1)),

Conv1D(filters=64, kernel\_size=3, activation='relu', padding='same'),

BatchNormalization(),

MaxPooling1D(pool\_size=2),

Dropout(0.3),

Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same'),

BatchNormalization(),

MaxPooling1D(pool\_size=2),

Dropout(0.3),

Conv1D(filters=256, kernel\_size=3, activation='relu', padding='same'),

BatchNormalization(),

GlobalAveragePooling1D(),

Dropout(0.4),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(1, activation='sigmoid') # Binary classification

])

* CNN model consists of three convolutional blocks, followed by two dense layers. The convolutional layers contain 64, 128, and 256 filters.
* The ReLU activation function was used to help the model learn complex patterns and activate neurons in each layer
* After each convolutional layer, Batch Normalization was applied to standardize the output and stabilize the learning process.
* To reduce the spatial dimensions and computation, MaxPooling1D layers with a pool size of 2 were used after the first two convolutional layers.
* Following the convolutional blocks, Global Average Pooling was used, which reduces each feature map to a single value, significantly decreasing the number of parameters and minimizing overfitting.
* For the final output layer, a single neuron with a sigmoid activation function was used to produce a binary prediction indicating whether the condition (e.g., disease presence) is detected (1) or not (0).

1. **MobileNet Lightweight Model**

The MobileNet model is effective at tasks involving image data because of its lightweight architecture, which uses depth wise separable convolutions. This model first reshapes and resizes the tabular data to create an image-like format.

X\_reshaped = x\_scaled.reshape(x\_scaled.shape[0], x\_scaled.shape[1], 1, 1)

feature\_count = x\_scaled.shape[1]

image\_size = int(np.ceil(np.sqrt(feature\_count))) # Create square input

padded\_features = np.zeros((x\_scaled.shape[0], image\_size\*\*2))

padded\_features[:, :feature\_count] = x\_scaled

x\_image = padded\_features.reshape(-1, image\_size, image\_size, 1)

x\_resized = tf.image.resize(x\_image, [224, 224])

x\_resized = tf.image.grayscale\_to\_rgb(x\_resized)

base\_model = MobileNetV2(include\_top=False, weights='imagenet', input\_shape=(224, 224, 3))

base\_model.trainable = False # Freeze base layers

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(64, activation='relu')(x)

output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base\_model.input, outputs=output)

To tailor the model for our binary classification task, we introduced some custom layers on top of the existing framework:

* First off, a Global Average Pooling layer was used to condense the output from the convolutional base into a single vector for each feature map. This step helps cut down on parameters and reduces the risk of overfitting.
* Next, a Dense layer with 64 neurons was incorporated, using the ReLU activation function to capture task-specific features from the spatial representations extracted.
* Finally, a single neuron was added to the output layer with a sigmoid activation function, which gives us a binary prediction showing whether the patient is likely to have heart disease (1) or not (0).

1. **EfficientNet Lightweight Model**

Another lightweight architecture that has been optimized for accuracy and efficiency is the EfficientNet model. The tabular\_to\_image function, which reshapes the data and pads it into a square image format, is used to convert the tabular data into images, much like MobileNet does.

# Convert tabular rows to image

def tabular\_to\_image(x\_row, img\_size=224):

side = int(np.ceil(np.sqrt(x\_row.shape[0])))

padded = np.pad(x\_row, (0, side\*side - x\_row.shape[0]), mode='constant')

image = padded.reshape(side, side)

image = np.stack([image]\*3, axis=-1)

image\_resized = resize(image, (img\_size, img\_size), mode='reflect', anti\_aliasing=True)

return image\_resized

x\_images = np.array([tabular\_to\_image(x) for x in x\_scaled])

#Build EfficientNet Model

base\_model = EfficientNetB0(include\_top=False, input\_shape=(224, 224, 3), weights='imagenet')

base\_model.trainable = False # Freeze base layers

x = base\_model.output

x = GlobalAveragePooling2D()(x)

x = Dense(64, activation='relu')(x)

output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=base\_model.input, outputs=output)

model.compile(optimizer=Adam(), loss='binary\_crossentropy', metrics=['accuracy'])

This model is built with several depthwise separable convolution layers, making it efficient and lightweight for image processing.

* A Dense hidden layer with 64 neurons was incorporated to capture task-specific features from the extracted representations.
* To address the dying neuron problem and ensure smoother training, the ReLU activation function was chosen for this hidden layer.
* Although the original code doesn’t explicitly include Batch Normalization, it can be seamlessly integrated after each dense layer to standardize activations and stabilize the learning process.
* To enhance generalization and minimize overfitting, adding Dropout with a rate of 0.3 following the dense layer, even though this wasn’t part of the original code.
* For the final output, a single neuron with a sigmoid activation function was used to generate a binary prediction indicating whether the patient is likely to have heart disease (1) or not (0).

1. **Result & Discussion**

In the Result & Discussion section, the performance of different models is typically evaluated and compared. This section discussed some key terms that will be used for comparing the performance of the models, utilizing various graphs and tables.

* 1. **Key Terms for model Comparison**

Here are key terms and aspects that are commonly discussed when comparing models, particularly in the context of binary classification.

* + 1. **Train Accuracy**

Train accuracy refers to the percentage of correct predictions made by the model on the training dataset.It shows how effectively the model has absorbed the information from the data it was trained on.

* If train accuracy increases, it usually means the model is doing a good job of picking up the patterns in the training data.
* A drop in train accuracy might suggest that the model is underfitting — in other words, it’s too simplistic to grasp the underlying patterns.
* While high train accuracy is a positive sign, if it’s much higher than the test accuracy, it could indicate overfitting — where the model excels on training data but struggles to generalize to new data.
  + 1. **Test Accuracy**

Test accuracy refers to the percentage of correct predictions that a model makes when it encounters new, unseen data. The purpose of measuring this is to see how well the model can adapt to and generalize from new information.

* When test accuracy goes up, it indicates that the model is doing a great job of generalizing and is performing reliably in real-world situations.
* On the flip side, if test accuracy drops, it could mean that the model is either overfitting to the training data or hasn't been trained enough.
* Ultimately, test accuracy is a vital measure of how well a model performs. Ideally, a strong model should show high accuracy on both training and test data, with only a small difference between the two.
  + 1. **Total Number of Parameters**

This refers to the total count of trainable weights and biases in a neural network model. It’s a key metric that showcases the model's capacity and complexity, which in turn influences memory usage and computation during training.

Formulae:

* For a Dense (Fully Connected) Layer:
  + Parameters = (Input Units × Output Units) + Output Units
  + Here, the first part represents the weights, while the second part accounts for the biases one for each output unit.
* For a Conv2D (Convolutional) Layer:
  + Parameters = (Kernel Height × Kernel Width × Input Channels × Output Channels) + Output Channels
  + In this case, the bias is added for each output channel (or filter).

Usage: Understanding the number of parameters helps gauge the memory needs and training capacity of the model

**When It Increases**: The model becomes more complex and capable of learning intricate patterns.

**When It Decreases**: The model is simpler and may be easier to train but might not capture complex relationships.

Having fewer parameters often results in quicker, smaller models that are easier to deploy. On the flip side, more parameters can enhance learning ability but also increase the chances of overfitting and lead to higher computational costs.

* + 1. **Floating point operation**

FLOPs, or Floating Point Operations per Second, measure how many arithmetic operations—like additions and multiplications—a model carries out during its inference or forward pass. This metric gives us a glimpse into the computational complexity of the model.

For a Conv2D layer, the typical formula looks like this:

FLOPs = 2 × (Kernel Height × Kernel Width × Input Channels) × (Output Height × Output Width × Output Channels)

The factor of 2 here accounts for both the multiplications and additions involved. This calculation applies to each inference sample.

When it comes to Dense layers, the FLOPs are calculated simply as:

FLOPs = 2 × (Input Units × Output Units)

A higher FLOPs count can suggest better performance, but it often comes with slower inference times and increased resource usage. On the flip side, lower FLOPs can lead to quicker inference, making them ideal for real-time applications or embedded systems, though this might come at the cost of performance.

* + 1. **Model Size**

Model size is all about the amount of storage space usually measured in megabytes or gigabytes that's needed to keep the model's structure and the weights it has learned. This size is shaped by how many parameters there are and the precision of those parameters (which refers to the bit width).

Formula: Model Size (in Bytes) = Total Parameters × Precision (in Bytes)

Bigger models can be tricky to use on devices with limited resources. On the flip side, smaller models achieved through techniques like quantization, pruning, or compression—are easier to deploy, but they might not perform as well.

* + 1. **Running Time**

Running time, often referred to as execution time or time complexity, is all about how long an algorithm takes to finish its task based on the size of the input. This metric is super important when it comes to assessing how efficient algorithms are, particularly regarding their scalability and performance.

You can express running time in a couple of ways:

Empirical form: This is measured in real-world units like seconds or milliseconds on actual hardware.

Theoretical form: Here, Big O notation used to illustrate how growth relates to the size of the input.

* 1. **Comparison of models based on performance**

By examining both training and test accuracy, the performance of various models is compared in this section. This allows evaluation of each model's generalization to unknown data through this comparison, which is essential for determining how well the model can predict outcomes on real-world datasets.

* + 1. **Artificial Neural Network**

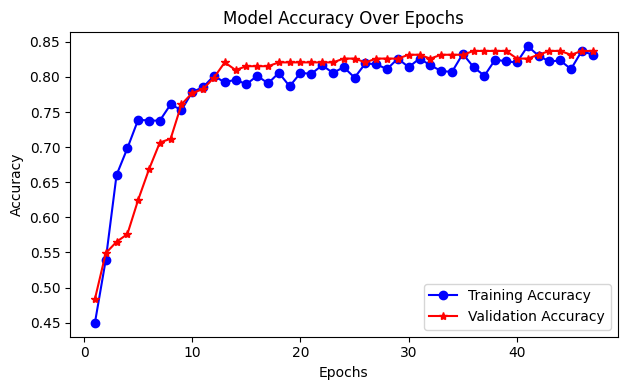
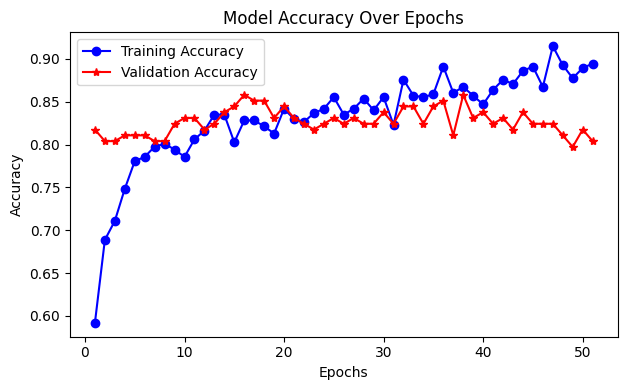
The model was trained for 45 epochs, as illustrated in the graph, using a batch size 50 that remained consistent throughout the training. Both training and validation accuracy were plotted to assess how well the model performed over time.

Fig 5.1 Comparison of Training and Testing accuracy of ANN model

The accuracy graph shown in Fig 5.1 clearly shows the model's journey toward better classification of the target labels. It started off with fairly low accuracy but then experienced a sharp rise during the early epochs. By around epoch 12, both training and validation accuracies were hovering around the 80% mark, which suggests that the model was learning effectively. As the training progressed, both accuracy curves began to level off, with validation accuracy staying consistently high. This is a good sign that the model has learned to generalize well to new, unseen data without falling into the trap of overfitting.

* + 1. **Deep Neural Network**

Training of the model was done for 52 epochs with a batch size of 32, and the training and validation accuracy were plotted to see how well the model was performing over time.

Fig 5.2 Comparison of Training and Testing accuracy of DNN model

The accuracy graph shown in Fig 5.2 shows a clear trend: the model got better at classifying the data, starting with a training accuracy of about 59% and steadily improving from there. The validation accuracy also increased during the early epochs, which is a good sign of effective learning.However, around epoch 15, the validation accuracy started to level off and even fluctuate, while the training accuracy kept climbing. This growing gap between the training and validation curves suggests that the model might be overfitting. It’s great to see the training accuracy surpass 90% by the end of the epochs, but the validation accuracy hovering around 80-85% indicates that the model's ability to generalize didn’t keep improving. To improve this, using a more varied training set that could enhance the testing accuracy.

* + 1. **Conventional Neural Network**

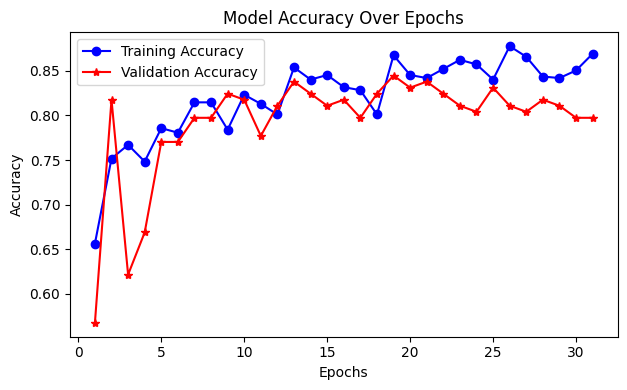
Model trained for 32 epochs with a batch size of 32, and the training and validation accuracy were plotted to evaluate the model’s performance over time.

Fig 5.3 Comparison of Training and Testing accuracy of CNN model

The accuracy graph shown in Fig 5.3 demonstrates that the model quickly learned meaningful features from the data, as seen in the sharp rise in both training and validation accuracy during the initial epochs. The training accuracy steadily improved and reached close to 86%, indicating that the model is effectively fitting the training data. The validation accuracy also improved early on but began to fluctuate after around epoch 10, stabilizing in the 80-84% range. These fluctuations reflect the model's efforts to generalize to new, unseen data, but they also suggest some level of overfitting, as the training accuracy continues to climb while the validation accuracy does not consistently follow.

* + 1. **MobileNet**

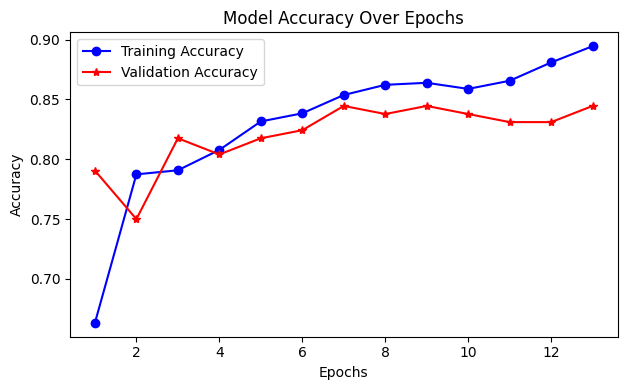
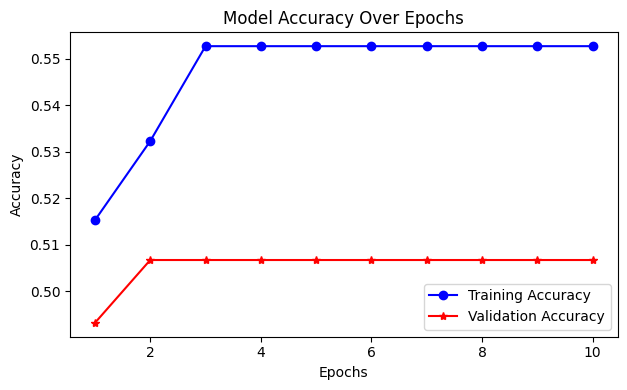
Model is trained for 13 epochs with a batch size of 32, the accuracy graph demonstrates how the model progressively improved its ability to classify the data.

Fig 5.4 Comparison of Training and Testing accuracy of MobileNet Model

Here, graph shown in fig 5.4 showing a sharp increase in training accuracy during the initial epochs from around 67% to nearly 90% by the final epoch. Validation accuracy followed a similar upward trend, peaking around 84-85%, which suggests the model was able to generalize well to unseen data. Although the training accuracy continues to climb, the validation accuracy flattens slightly in the latter epochs, indicating a mild risk of overfitting. This suggests the model is beginning to learn specific patterns in the training data that may not fully translate to the validation set.

* + 1. **EfficientNet**

Training of the model was done for 10 epochs with a batch size of 32, and the training and validation accuracy were plotted, unlike the other models, this accuracy graph shows only a slight improvement during training.

Fig 5.5 Comparison of Training and Testing accuracy of EfficientNet Model

As shown in Fig 5.5 the training accuracy levels off at around 55.6%, while the validation accuracy stabilizes early at about 50.7%, which is nearly the same as random guessing for a binary classification task. This indicates that the model had a tough time learning the patterns in the data. There could be a few reasons for this lackluster performance: Underfitting, where the model is too simplistic or the training setup isn't allowing it to grasp the complexity of the data. Ineffective transfer learning, particularly if EfficientNet wasn't fine-tuned properly for the specific dataset.

* 1. **Comparison of models based on computational cost**

In this section, various deep learning models are compared not only based on their accuracy but also on their computational efficiency. This is a crucial aspect, particularly when deploying models on environments such as mobile or such devices. The comparison includes key parameters such as Total No. of parameters, FLOPs, Model size, Running Time and Accuracy.

Table 5.1 Comparison of models based on computational cost and accuracy

| Terms | ANN | CNN | DNN | MobileNet | EfficientNet |
| --- | --- | --- | --- | --- | --- |
| Total No. of parameters | **48769** | 158337 | 186305 | 2340033 | 4131620 |
| FLOPs (Floating Point Operations) | **97057** | 967745 | 371601 | 612890241 | 800949208 |
| Model Size (in MB) | **0.63** | 1.87 | 2.23 | 10.06 | 17.08 |
| Running Time (in sec) | **0.14** | 0.43 | 0.40 | 8.73 | 20.91 |
| Accuracy | **84.78%** | 77.17% | 83.15% | 80.43% | 59.23% |

As shown in Table 5.1 the highest accuracy (84.78%) and the lowest computational cost in terms of parameters, FLOPs, model size, and running time, the ANN model is clearly the most effective and well-balanced choice. Conversely, EfficientNet exhibits the highest computational cost and the lowest accuracy (59.23%) in this context, indicating poor suitability for the dataset, even though it is known for its high performance in image-based tasks. There are moderate trade-offs between CNN and DNN. DNN is marginally more accurate than CNN, but both require more resources than ANN. Despite being designed for edge efficiency, MobileNet only achieves moderate accuracy (80.43%) and exhibits relatively high FLOPs and runtime. Overall, the finding indicates that for some structured problems, simpler models such as ANN can perform better than deeper architectures.

1. **Conclusion**

This project designed a system to predict chronic diseases by utilizing a variety of deep learning models, such as ANN, DNN, CNN, MobileNet and EfficientNet. The system is adept at managing medical datasets, preprocessing complex and incomplete information and converting structured data into formats that are compatible with deep learning models. Among the different architectures examined, ANN and DNN struck a great balance between accuracy and computational efficiency.

The experimental findings revealed that deep learning models can greatly improve the early detection of chronic diseases, with models like ANN reaching accuracy levels of about 84.78%. On the other hand, lightweight models like MobileNet also showed encouraging results while minimizing computational requirements, making them ideal for real-time cloud applications. Although EfficientNet is quite advanced, it encountered some hurdles in adapting to transfer learning due to the nature of the datasets.

This project underscores the vast potential of merging deep learning with cloud computing technologies to facilitate proactive, personalized healthcare interventions and enhance patient outcomes.

1. **References**

[1] Wu, C. T., Wang, S. M., Su, Y. E., Hsieh, T. T., Chen, P. C., Cheng, Y. C., ... & Lai, F. (2022). A precision health service for chronic diseases: development and cohort study using wearable device, machine learning, and deep learning. IEEE Journal of Translational Engineering in Health and Medicine, 10, 1-14.

[2] Al Reshan, M. S., Amin, S., Zeb, M. A., Sulaiman, A., Alshahrani, H., Shaikh, A., & Elmagzoub, M. A. (2024). An Innovative Ensemble Deep Learning Clinical Decision Support System for Diabetes Prediction. IEEE Access.

[3] Bhaskar, N., Bairagi, V., Boonchieng, E., & Munot, M. V. (2023). Automated detection of diabetes from exhaled human breath using deep hybrid architecture. IEEE Access, 11, 51712-51722.

[4] Ali, M. M., Islam, M. S., Uddin, M. N., & Uddin, M. A. (2024). A conceptual IoT framework based on Anova-F feature selection for chronic kidney disease detection using deep learning approach. Intelligence-Based Medicine, 10, 100170.

[5] Chen, G., Ding, C., Li, Y., Hu, X., Li, X., Ren, L., ... & Xue, W. (2020). Prediction of chronic kidney disease using adaptive hybridized deep convolutional neural network on the internet of medical things platform. IEEE Access, 8, 100497-100508.

[6] Venkatrao, K., & Kareemulla, S. (2023). HDLNET: a hybrid deep learning network model with intelligent IoT for detection and classification of chronic kidney disease. IEEE Access.

[7] Elkholy, S. M. M., Rezk, A., & Saleh, A. A. E. F. (2021). Early prediction of chronic kidney disease using deep belief network. IEEE Access, 9, 135542-135549.

[8] Chaki, J., & Ucar, A. (2024). An efficient and robust approach using inductive transfer-based ensemble deep neural networks for kidney stone detection. IEEE Access.

[9] Golec, M., Gill, S. S., Parlikad, A. K., & Uhlig, S. (2023). HealthFaaS: Ai-based smart healthcare system for heart patients using serverless computing. IEEE Internet of Things Journal, 10(21), 18469-18476.

[10] Sarmah, S. S. (2020). An efficient IoT-based patient monitoring and heart disease prediction system using deep learning modified neural network. Ieee access, 8, 135784-135797.

[11] Almazroi, A. A., Aldhahri, E. A., Bashir, S., & Ashfaq, S. (2023). A clinical decision support system for heart disease prediction using deep learning. IEEE Access, 11, 61646-61659.

[12] Pan, Y., Fu, M., Cheng, B., Tao, X., & Guo, J. (2020). Enhanced deep learning assisted convolutional neural network for heart disease prediction on the internet of medical things platform. Ieee Access, 8, 189503-189512.

[13] Ali, S. N., Shuvo, S. B., Hasan, T., Al-Manzo, M. I. S., & Hasan, M. A. (2023). An End-to-end Deep Learning Framework for Real-Time Denoising of Heart Sounds for Effective Computer-Aided Auscultation in Unseen Noise. Authorea Preprints.

[14] Chakraborty, C., & Kishor, A. (2022). Real-time cloud-based patient-centric monitoring using computational health systems. IEEE transactions on computational social systems, 9(6), 1613-1623.

[15] Khan, M. A., & Algarni, F. (2020). A healthcare monitoring system for the diagnosis of heart disease in the IoMT cloud environment using MSSO-ANFIS. IEEE access, 8, 122259-122269.

[16] Sundas, A., Badotra, S., Shahi, G. S., Verma, A., Bharany, S., Ibrahim, A. O., ... & Binzagr, F. (2024). Smart Patient Monitoring and Recommendation (SPMR) Using Cloud Analytics and Deep Learning. IEEE Access.

[17] Halder, R. K., Uddin, M. N., Uddin, M. A., Aryal, S., Saha, S., Hossen, R., ... & Akter, M. F. (2024). ML-CKDP: Machine learning-based chronic kidney disease prediction with smart web application. Journal of Pathology Informatics, 15, 100371.

[18] Tabtaba, A. A., & Ata, O. (2024). Diabetic retinopathy detection using developed hybrid cascaded multi-scale DCNN with hybrid heuristic strategy. Biomedical Signal Processing and Control, 89, 105718.

[19] Ochoa-Ornelas, R., Gudiño-Ochoa, A., García-Rodríguez, J. A., & Uribe-Toscano, S. (2025). Enhancing early lung cancer detection with MobileNet: A comprehensive transfer learning approach. Franklin Open, 100222.