Architectural Implementation of Physiological Signals Analysis

Final Year Project Report
Presented
By

Javeria Ehsan
CIIT/FA19-EEE-023/ISB

Muzammil Arif
CIIT/FA19-EEE-002/ISB

In Partial Fulfillment
of the Requirement for the Degree of
Bachelors of Science in Electrical (Electronics) Engineering
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COMSATS UNIVERSITY ISLAMABAD

JULY 2023

Architectural Implementation of Physiological Signals Analysis

Final Year Project Report
Presented
by

Javeria Ehsan
CIIT/FA19-EEE-023/ISB

Muzammil Arif
CIIT/FA19-EEE-002/ISB

In Partial Fulfillment
of the Requirement for the Degree of

Bachelors of Science in Electrical (Electronics) Engineering

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

COMSATS UNIVERSITY ISLAMABAD

JULY 2023

Declaration

We, hereby declare that this project is not copied from any source. It is further declared that we have developed this project and the report entirely on the basis of our personal efforts made under the sincere guidance of our supervisor. The work presented in this report has not been submitted before in any other University or Institute of learning, if found we shall stand responsible.

Signature: _____
Name: Javeria Ehsan

Signature: _____
Name: Muzammil Arif

COMSATS UNIVERSITY ISLAMABAD

JULY 2023

Architectural Implementation of Physiological Signals Analysis

An Undergraduate Final Year Project Report submitted to the
Department of
ELECTRICAL AND COMPUTER ENGINEERING

As a Partial Fulfillment for the award of Degree
Bachelors of Science in Electrical (Electronics) Engineering
By

Name	**Registration Number**
Javeria Ehsan	CIIT/FA19-EEE-023/ISB
Muzammil Arif	CIIT/FA19-EEE-002/ISB

Supervised by

Dr. M. Faisal Siddiqui
Tenured Associate Professor
Dr. Ghufran Shafiq
Assistant Professor

Department of Electrical and Computer Engineering
CU Islamabad

COMSATS UNIVERSITY ISLAMABAD
JULY 2023

Final Approval

This Project Titled

Architectural Implementation of Physiological Signals Analysis

Submitted for the Degree of

Bachelors of Science in Electrical (Electronics) Engineering

by

Name	**Registration Number**
Javeria Ehsan	CIIT/FA19-EEE-023/ISB
Muzammil Arif	CIIT/FA19-EEE-002/ISB

has been approved for

COMSATS UNIVERSITY ISLAMABAD

_____ _____

Supervisor
Dr. M. Faisal Siddiqui
Tenured Associate Professor

Co-Supervisor
Dr. Ghufran Shafiq
Assistant Professor

_____ _____

Internal Examiner-1
Dr. Dilshad Sabir
Assistant Professor

Internal Examiner-2
Dr. Fasih Uddin Butt
Professor

_____

External Examiner
Dr. M. Naeem Ali Bhatti,
Tenured Associate Professor,
QAU, Islamabad

_____

Head of Department of Electrical and Computer Engineering

Dedication

This study is wholeheartedly dedicated to heart patients for their everyday struggle. Your strength and resilience inspire us all to keep pushing for more work to bring better treatments, care, and awareness. Your unwavering spirit in the face of adversity is a testament to the power of the human heart. May this work serve as a small reminder that we stand with you every step of the way.

Acknowledgements

First and foremost, all our praise and thanks be to Allah. We would like to express our thankfulness for the wisdom and knowledge Allah has bestowed upon us. As we complete this phase of our academic journey, we acknowledge that all our achievements come from Allah. May we always be mindful of His infinite blessings.

We offer our deepest gratitude to our parents who taught us the philosophy of hard work, honesty, and unflinching faith in Allah Almighty.

We extend our gratitude to our respectful teachers and mentors Dr. Muhammad Faisal Siddiqui and Dr. Ghufran Shafiq for their guidance and support throughout this project. Your unwavering commitment to our success and your tireless efforts to ensure that we were able to navigate the challenges we faced are a testament to your dedication as an educator. Without your guidance, we would not have been able to achieve the level of success we had. Your valuable feedback on our work proved to be instrumental to our success.
Finally, we are thankful to everyone who happened to be of great help in one way or the other.

Javeria Ehsan
Muzammil Arif

Table of Contents

List of Acronyms

MIT BIH…….....Massachusetts Institute of Technology-Beth Israel Hospital
PTB…..……Physikalisch-Technische Bundesanstalt
CNN……..Convolutional Neural Network
ECG…….....Electrocardiogram
CVDs……..……Cardiovascular Diseases
DNN…….Deep Neural Network
SGDs…….Sustainable Development Goals
FPGA……Field Programmable Gate Array
DPU……Deep Learning Processing Unit
CPVT……Catecholaminergic Polymorphic Ventricular Tachycardia
ARVD/C……...Arrhythmogenic right ventricular dysplasia/cardiomyopathy
ROC…….....Receiver operating curve
VT…….....Ventricular tachycardia
VF……Ventricular fibrillation
NAS……..Neural Architecture Search
CCE……Categorical Cross Entropy
SCCE……..Sparse Categorical Cross Entropy
SE…….....Squeeze and Excite
GPU…….Graphics processing unit
TP……True positive
TN……..True negative
FP……...False positive
FN…….....False negative

List of Figures

s

List of Tables

Abstract

Introduction: ECG Arrhythmias can occur for various reasons, including heart disease, medications, or genetic factors. Some arrhythmias are harmless, while others can be life-threatening. It is important to detect and diagnose arrhythmias accurately to provide timely and appropriate treatment. It is essential to ensure that ECG arrhythmia detection is done accurately to provide appropriate treatment and prevent complications. Therefore, it is important to use reliable and validated methods for detecting arrhythmias and to have experienced healthcare professionals who can interpret ECG signals accurately. In this work, CNNs are used for the detection of ECG Arrhythmias which will classify the arrhythmia into its respective class. A hardware-friendly deep learning system is proposed for correct classification of Cardiac Arrhythmias. **Methodology:** The bio-medical signal processing world is evolving, to keep pace with advancement, scientists and researchers are compelled to achieve accuracy as it reflects the factuality and reliability of the research. To enhance the diagnostic procedure for automatic arrhythmia detection, our work presents proof of a concept for a lightweight, computationally inexpensive, and efficient Arrhythmia classifier that will reveal the potential clinical utility of ECG Arrhythmia signals for the detection and monitoring of certain cardiovascular conditions. In this work, analysis is performed using different End-to-End Machine learning approaches, well-known CNN architectures i.e., AlexNet, ResNet18, VGG1**6**, MobileNet, and EfficientNet with varying parameters are used to enhance the efficiency of the system and for an in-depth analysis. The proposed system is tested for classification on a well-known and publically available MIT BIH Arrhythmia and PTBDB image dataset. Furthermore, the analysis for all CNNs using Transfer Leaning as a feature extractor and fine-tuning the MobileNet architecture is performed. **Results and Discussion:** After training and evaluating, our proposed work highlighted architectures that possess fewer parameters and small model size, less average training, and testing time without compromising on accuracy. Our method of transfer learning showed improved accuracy by reducing overfitting in EfficientNet with maximum accuracy of 99.57%. While ResNet18 outperformed state-of-art models with 99.33% accuracy. The VGG16 model showed 98.70% accuracy. This work can further be utilized to implement on embedded devices.

Chapter – 1

Introduction

1.1 Background:

In the world of Science and Technology, AI has recently garnered widespread acclaim. With increase in cardiovascular diseases (CVDs) researchers have considered using AI which can aid in clinical utility. ECG arrhythmia is an important group of CVDS which refers to an abnormal heart rhythm that is detected using an electrocardiogram (ECG) signal. An ECG signal measures the irregularities in the heart's rhythm.

The accuracy of ECG arrhythmia detection depends on several factors**:**

- ✓ Type of arrhythmia,
- ✓ Quality of the ECG signal,
- ✓ Experience and skill of the healthcare provider analyzing the ECG signal.

A correctly trained healthcare professional with experience in interpreting ECG signals can detect arrhythmias with high accuracy. However, automated algorithms and machine learning techniques are also being developed to enhance accuracy and speed of Cardiac arrhythmia detection.

Deep Learning has been used to classify different types of Cardiac Arrhythmias. For an ECG, accurate detection and classification is still a challenge as some types of arrhythmias may be difficult to detect using ECG alone, such as those that occur infrequently or those that are not sustained long enough to be captured on the recording.

1.2 Rare and Deadly Arrhythmias:

Some arrhythmias are difficult to detect. What makes them difficult is the inconsistency of their occurrence. Some arrhythmias come and go, they are for short duration. Some are occasional that do not come to appear in routine ECG. For this purpose, prolonged ECG is required. The arrhythmias that fall under this category are paroxysmal arrhythmias such as Atrial Fibrillation.

The rarity of arrhythmias is directly linked with prevalence in population such as Brugada syndrome, catecholaminergic polymorphic ventricular tachycardia (CPVT), and arrhythmogenic right ventricular dysplasia/cardiomyopathy (ARVD/C). These arrhythmias are the arrhythmias that exist in a small percentage of the population. Detecting rare arrhythmias may be challenging due to their low occurrence, lack of awareness, and the need for specialized diagnosis.

The deadliest arrhythmia is ventricular arrhythmia that can cause sudden cardiac arrest where the heart suddenly stops beating. Ventricular fibrillation (VF) is a chaotic and dis-organized electrical activity of the ventricles, while ventricular tachycardia (VT) is a rapid heart rate originating in the ventricles. Both VF and VT can be fatal within minutes if not treated immediately. They are considered highly lethal arrhythmias and

require immediate medical attention, including defibrillation and advanced life support measures.

1.3 Research Aim

The objective of this report is to propose a hardware friendly architecture that can be deployed on tightly constraint environment. Expensive ECG systems make it less accessible to patients and healthcare providers in low-resource settings.

The aim of this study is to examine various aspects, with a particular focus on hardware favorability, affordability, and simplicity. Hence, to analyze different End to End Learning Architectures suitable for personalized ECG monitoring. ECG has attained recognition due to the recent advancement in the field of bio medical signal processing. It has achieved new heights in recent years which demonstrates the potential clinical utility of ECG signals for the detection and monitoring of certain cardiovascular conditions. This work is going to open doors for further research and real-time implementation which will offer variety of diagnostics.

1.4 Limitations:

One of the most undermined and overlooked area in the research for the detection of Arrhythmia is inability to detect certain Arrhythmias. ECG arrhythmia detection systems can sometimes detect abnormal rhythms that are not actually present, leading to unnecessary diagnostic tests and interventions. In some cases, it can also miss abnormal rhythms, which can delay diagnosis and treatment. Despite advances in technology, ECG arrhythmia detection systems are not always accurate, particularly in cases where the arrhythmia is not obvious or the ECG recording quality is poor.

1.5 Problem Statement:

There exists scope for improvements for attaining more accuracy and reliability of the techniques as most of the research is focused on Arrhythmias that are easy to detect or Cardiac features that are not very challenging. Additionally, there is a need to make ECG arrhythmia detection more accessible and affordable, particularly in low-resource settings.

This will reveal the potential clinical utility of ECG Arrhythmia signals for the detection, monitoring, and prevention of certain cardiovascular conditions.

1.6 Proposed Solution:

This work implements different Convolutional Neural Networks to analyze in terms of hardware favorability and reliability. Our lightweight hardware compatible Architecture will create a breakthrough in the field of ECG Arrhythmia detection which can be implemented on hardware devices saving costly machines, being a portable

device that can bring convenience with reliable accuracy. ECG arrhythmia detection is essential for advancing our understanding of the underlying mechanisms and risk factors for heart rhythm disorders and other cardiovascular diseases.

1.7 Broader Impact (UN Sustainable Development Goals):

There are 17 Sustainable Development Goals by United Nation that is its Agenda which provides a comprehensive framework to address social, economic, and environmental challenges faced world widely. Every person is encouraged to put efforts and take initiatives to contribute to this achievement. They reflect the urgent need to address poverty, inequality, climate change, environmental degradation, and social injustice to create a better and more sustainable world for present and future generations.

Figure 1.1: UN SDGs [1]

1.6.1 Targeted Sustainable Development Goals:

The targeted SDGs are given below**:**

Table 1.1: Targeted SDGs

#	Sustainable Development Goals	Addressed
1	No poverty	
2	Zero Hunger	
3	Good Health and Well-being	✓
4	Quality Education	✓
5	Gender Equality	
6	Clean Water and Sanitation	
7	Affordable and Clean Energy	

8	Decent Work and Economic Growth	
9	Industry, Innovation, and Infrastructure	✓
10	Reduced Inequalities	
11	Sustainable cities and communities	
12	Responsible consumption and production	
13	Climate Action	
14	Life below water	
15	Life on Land	
16	Peace, Justice, and strong institutions	
17	Partnerships for the Goals	✓

1.6.2 Potential Mapping:

Our work ECG Arrhythmia detection perfectly aligns with Good Health and Well-being goal of the UN's SDGs in a clear and impactful manner. Moreover, it also falls under the umbrella of Industry, Innovation and Infrastructure, and Quality Education.

Table 1.2: Address SDGs in this work

SDG Title	Aim	Addressed
Good Health and Well-being	Ensure Healthy lives and promote well-being for all ages.	• Improved Healthcare Access • Cost and Resource efficiency • Enhanced Diagnosis accuracy • Empowering Patients and Professionals • Disease Prevention and Management
Quality Education	Promoting life-long learning for all.	• leverage shared knowledge, and expertise. • Inculcating Inquisitiveness • Inclusive education • Access to up-to-date resources

Industry, Innovation, and Infrastructure		• Improving existing practices • Technology advancement
Partnerships for the Goals	Strengthen the implementation means.	• Collaboration and Teamwork

1.7 Organization of the Report:

The next chapter is a detailed literature review, where all the previous work and state-of-art-methods are discussed. It will reveal the potential of AI in classification of Cardiac Arrhythmias. The 3rd chapter is detailed theory and discussion of implemented techniques. The 4th chapter is an implementation of this work to achieve the objective. Furthermore, the 5th chapter deals with the results and complete analysis of methodology and outcomes in terms of hardware complexity, utility, and reliability. In the **6**th chapter conclusion is stated. Finally, the potential area for further research is revealed.

Chapter-2

Literature Review of Automatic Arrhythmia Detection

This chapter entails the literature review related to different aspects of research on Cardiac Arrhythmia Detection and a comparative study involving different Deep Neural Networks implementation and scope of being implemented on hardware devices.

2.1 Background

The year 1887 marks the inception of ECG. Since the birth of clinical electrocardiogram, many researchers continued to further analyze ways to get benefitted for clinical purposes [2]. From early beginning to present days, Artificial Intelligence has created massive breakthroughs in the field of health sciences. It has created room for improvements and opened new doors for medical applications. One of the purpose ECG is fulfilling is the detection of Cardiac Arrhythmias. It is an irregular heartbeat referred to as abnormal rhythm which is life threatening. Early detection and diagnosis can prevent the disease from worsening.

2.2 ECG Signal Processing and Feature Extraction

Like any signal, ECG signal can be represented in a time domain as well as frequency domain. Both representations have pros and cons. There exists different analysis based on both the representations. ECG signal is a very noisy signal which if interpreted needs to be denoised, so the artifacts are removed. This denoising can be done using different techniques. Some researchers used Empirical mode decomposition to break down noisy ECG signal into a finite set of small chunks [3].

Not all small IMFs are noisy, so Spectral Flatness is measured for detection of noise after which if it appears to be noisy, as bandpass Butterworth filter is used. One can also use moving average filter for the baseline correction [4]. Another approach used to remove baseline wandering so there is no undesired interference is to use symlet scaling filter from wavelet transform and detrend operation. If some noise is still left the author used Savitzky-Golay filter [5]. There exists two kind of noises, baseline wandering which is low frequency noise and high frequency motion artifacts. To keep a signal raw, baseline correction is performed and a high-frequency noise filtering in sequence. ECG signals were preprocessed utilizing a 200-ms width median filter to remove the P waves and QRS complexes followed by a median filter of **6**00-ms to remove the T waves [**6**]. In this paper the author used second-order integer low-pass filter for the removal of the high-frequency noise components [7]. For denoising, this paper implements Daubechies wavelet of order four and to correct baseline moving

average filter is used [8]. In another paper denoising is performed using relaxed median filtering as moving average may not be an optimal choice since ECG signal has Q peak which may get compromised in averaging filters [9].

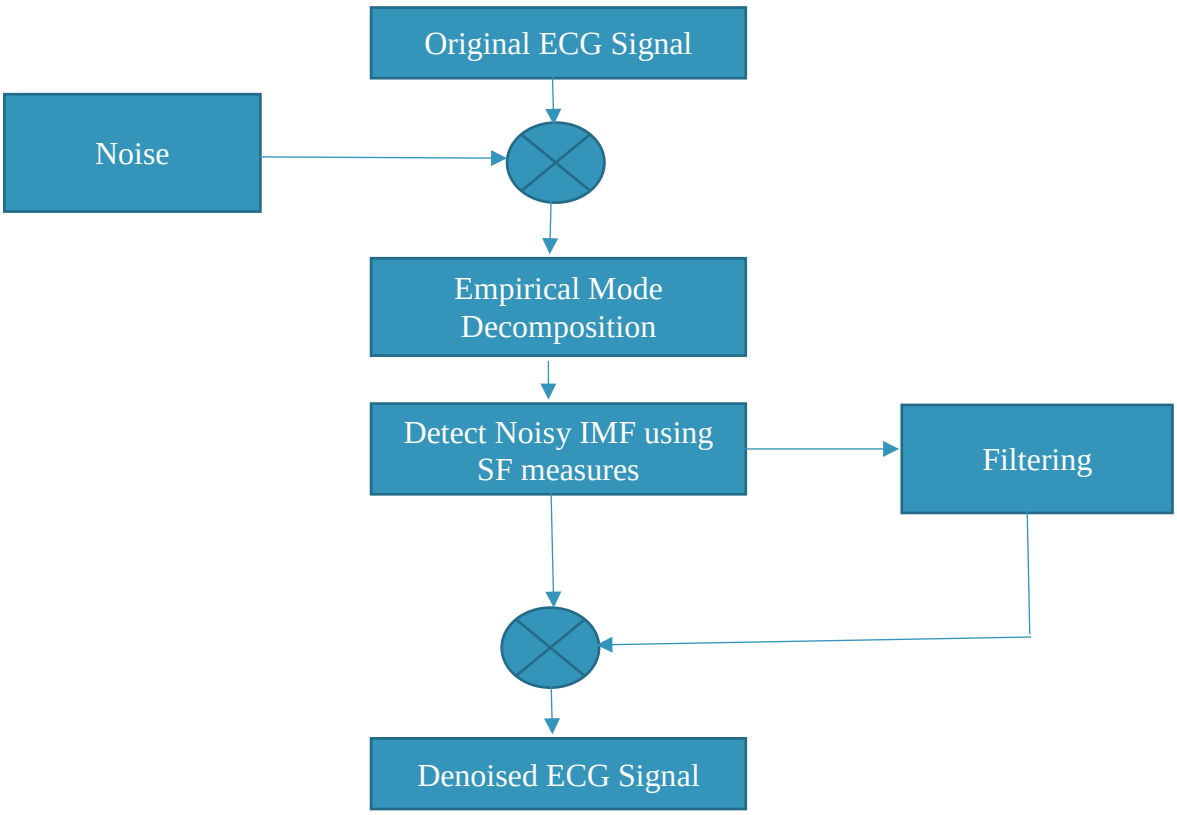


Figure 2.1: Denoising using EMD.

ECG signal has different features based on which analysis is performed. A comparative study of different feature extraction methods is performed in which wavelet transform, independent components analysis, Eigenvector method, auto-regression method, linear prediction and Fast Fourier Transform is performed [10]. The comparative results shown in the paper are given in **Figure 2.**2.

Another remarkable work is done using time domain morphology and gradient based algorithm for the feature extraction from PQRST complex [11]. The block diagram of implementation is given in the paper in **Figure 2.**3.

This paper utilizes both fiducial and non-fiducial features by using the consecutive change of ECG power spectral density as significant feature**.** ECG fiducial features have been shown to exist, though but are diverse, making them challenging to use for human identification. The paper examined the viability to address this problem of simulating the human ECG and using it for identification in time, amplitude, and distance variations in the ECG features. Getting a cross feature matrix helps us accomplish this goal that is used to simulate the dynamic change in the fiducial features of the QRS [12].

A detailed survey of different feature extraction techniques is discussed in this paper [13].The main reason to use wavelet transforms is that they are localized both in the

time and frequency domains. This paper presents optimal mother wavelet, based on the wavelet transform, for feature extraction [14].

Table 2.1: Comparison of different feature extraction methods

Feature Extraction method	Application Domain	Competence	SuiTable Classification Method	Accuracy (%)
AR	Time-Frequency	Classify Cardiac Arrhythmias	QDF	9**6.6**
WT	Time-Frequency	Local analysis of fast time varying and non-regular signals	ANN	92.20
Eigenvector	Frequency	Signals composed of sinusoids buried in noise	MME	98.0**6**
FFT	Time-Frequency	Short-term heart rate variability	ANN	92.47
LP	Time	No explicit assumptions for actual shape of the signal	LDA	93.2
ICA	Time-Frequency	Linear mixture of independent sources	Fast ICA	90.13

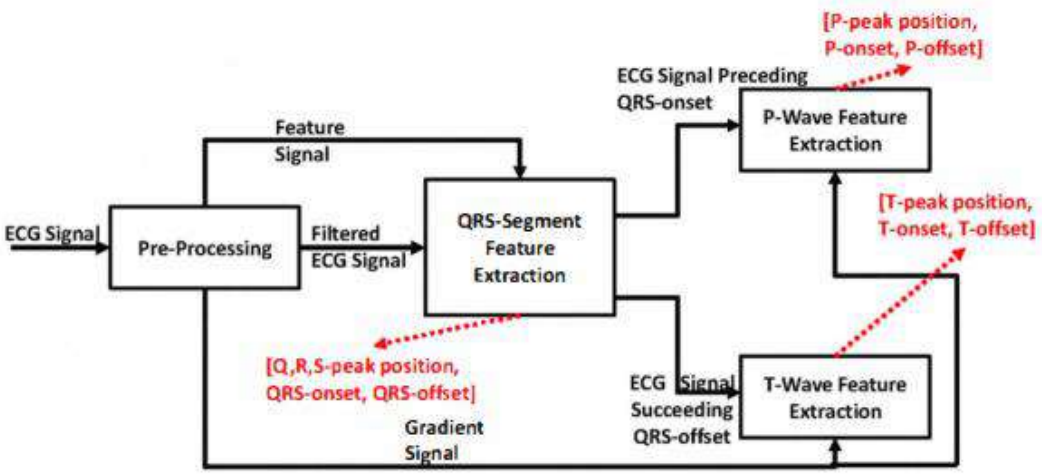


Figure 2.2: Gradient Based Algorithm

In this research optimization using feature selection is performed. Features like temporal, morphological and statistical are taken for observations [15]. Since fiducial features involve high complexities which is why a low complexity feature extraction method is performed. In this context, discrete wavelet transform (DWT) with the Haar function being the mother wavelet, as our principal analysis method is performed [16]. Multiresolution wavelet transformation is also the talk of the town. This paper used it for the QRS detector which achieved of sensitivity of 99.18% and a positive rate of

98.00% on validation data [17]. In this paper two different feature extraction techniques are applied simultaneously to obtain the feature vector. The wavelet transform is used to extract the coefficients of the transform as the features of each ECG segment while autoregressive modelling (AR) is also applied to obtain the temporal structures [18]. Pan Tompkins is very efficient way to detect QRS complexes. This paper performed Pan Tompkins algorithm for the detection of QRS complex [19].

2.3 Classification Algorithms

Different algorithms are used for the detection of ECG Arrhythmia like SVM, ANN, Random Forest, but CNNs with state of art accuracies are the choice of every Machine learning practitioner. This paper implements Support Vector Machines [18], whereas a review of different deep learning models is given in this paper. It discussed CNNs, RNN, LSTM, DBN and GRU where CNN has proven to be dominant and a better suited classification method [20]. To achieve accuracies higher and improve the speed as well, simpler architectures are used. The architectures like VGG1**6** and MobileNetV2 are implemented with 0.95% validation accuracy [21]. This paper used VGG and compared the results with many states of art classifiers. The results of comparison is given below [22]**:**

Table 2.2: Micro-architectures comparison

Network	Number of Layers	Parameters	Training Time	Accuracy (%)
GoogleNet	144	5.9 million	132 minutes	99.90
ResNet	71	4.8 million	48 minutes	100
EfficientNet	290	4.1 million	112 minutes	99.70
MobileNet	154	2.4 million	53 minutes	100
Proposed Classifier	29	34 thousand	15 minutes	99.90

1D CNNs are also used for ECG classification. Since the goal has always been accuracy and speed, this paper implements 1D CNN [23]. This work is used for real time patient specific heartbeat classification on VEB and SVEB classes. The paper claims that this method has achieved robustness, computationally is excellent and can be carried further for hardware implementation. The results proved that the proposed work out performs other state-of-art methods.

AlexNet is an architecture with only 8 layers which makes it a choice for implementation where computations are tightly constraint. In the proposed model, a new dataset is made by the combination of the Kaggle dataset and the other, which is made by taking the real-time healthy and unhealthy datasets. The AlexNet transfer

learning approach is applied [24] for classifying Q, N, F, V and S classes. In this research, the DVEEA-TL model diagnoses heart abnormality in respect of accuracy during the training and validation stages as 99.9% and 99.8%, respectively. The results are given below**:**

Table 2.3: Results of DVEEA-TL model

Performance matrices	Training (%)	Validation (%)
Accuracy	99.9	99.8
Classification miss rate	0.05	0.07
Sensitivity	99.8	99.7
Specificity	21.09	2**6**.5
Precision	90.9	99.80
F1 score	0.98	0.97
FPR	0.75	0.73
FNR	0.002	0.002
MCC	99.2	98.5
Kappa Score	0.98	0.97

This paper implements ResNet50, AlexNet, and SqueezeNet where it showed an accuracy of 98.8%, 90.08%, and 91% for AlexNet, SqueezeNet, and ResNet50, respectively [25]. Another paper implements AlexNet, Resnet18, and GoogleNet [2**6**]. The paper used 7 classes for the classification in which different optimizers such as SGDM, RMSprop, and Adam are used to observe the behavior of models. It is observed that fine-tuned AlexNet is a good choice with SGDM optimizer having accuracy 99.09%. A survey was conducted which addressed the issues involved in classification, feature extraction, pre-processing of an ECG signal [27].

ECG scalogram is used for classification purposes. In this paper different micro architectures are used. MobileNetV2, SqueezeNet, ShuffleNet, GoogleNet, EfficientNet, and ResNet-18 are used and compared [28]. SqueezeNet proved to be slightly advantageous. The paper classified ARR, CHF and NSR classes. Comparative results are displayed in paper as**:**

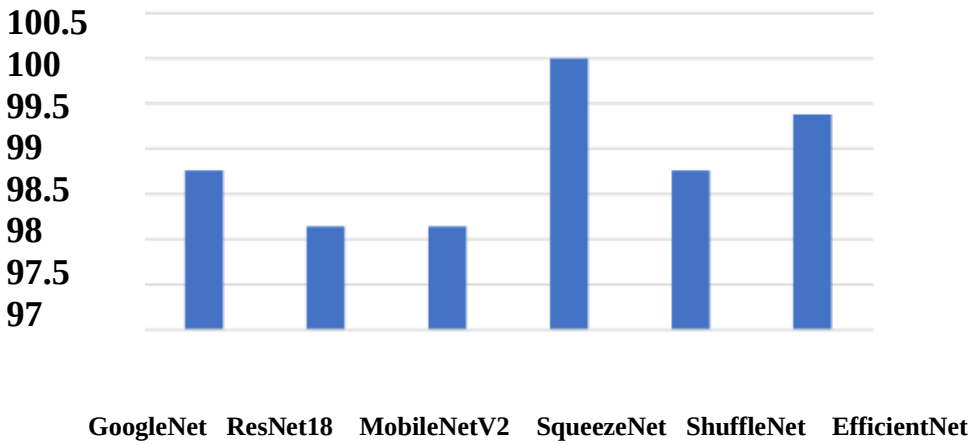


Figure 2.3: Classification results on ARR, CHF, NSR classes

This paper implements ResNet, Inception, and Xception alongside more recent EfficientNet and a spatiotemporal method involving convolutional LSTMs [29]. The classes used in this paper are N, S, V, F and Q which are the standards of ANSI/AAMI. Transfer learning is used to implement these models as a fine-tuning method. Input shape for the networks is taken as 18**6**x18**6**x3 and ConvLSTM with **6** x 18**6** x 31 x 3 input shape. ResNet50 showed 99.40% accuracy whereas VGG1**6** and VGG19 with 99.20%. The ResNet50 version 2 showed 97.**6**0% accuracy. EfficientNet B0 with 9**6**.20%, ConvLSTM with 9**6**.15%, Xception showed 94.40%, Inception V3 with 85.**6**0% and Inception ResNet V2 with lowest 48.**6**0% accuracy.

The goal of this paper is to convert one dimensional (1-D) ECG signals to two dimensional (2-D) scalogram images with the help of Continuous Wavelet (CWT) [30]. Four different MIT BIH Databases are used such as Arrhythmia database, Normal Sinus Rhythm database, Malignant Ventricular Ectopy database and BIDMC Congestive heart failure database. The transfer learning technique for AlexNet pepped up with an accuracy of 95.**6**7%.

The number of classes used for classification are not very much consistent. Since some arrhythmias are difficult to detect and some are rare in population as stated in chapter 1 makes it a difficult choice to come up with specific class as there is a lack of consistency regarding the usage of classes from MIT BIH Arrhythmia Database. Hence there is a lack of research. This paper used different number of classes from MIT BIH Arrhythmia database and compared the results [31]. It uses pre-trained EfficientNet B7 model. Multi-class classifications of arrhythmia such as 17-class, 15-class, 13-class, and 12- class classifications are used. The proposed approach achieved the highest average of 99.23% accuracy for 13-class classification.

2.4 Newly introduced Hardware Efficient Architectures

EfficientNet and MobileNet architectures are newly introduced architectures which are solely made to reduce computational power by reducing number of parameters used at the same time enhancing accuracy of the model.

2.4.1 EfficientNet

In recent time researchers have started to focus more on these as they are made for hardware implementations. In a recent work, detection of Myocardial Infarction from 12-Lead ECG using Eigen-domain representation is performed. This paper used EfficientNet V2 B2 as a transfer learning method which achieved 98.**6**8% accuracy [32]. A fine-tuned EfficientNet B0 is used for the detection of Atrial Fibrillation [33]. The paper uses normal and A-fib classes to be classified using EfficientNet. It addressed data imbalance problems as well. The model showed accuracy of 9**6**.79% and with data augmentation the accuracy is 95.8**6**%.

In a recent study a modified EfficientNet is used which has enhanced the accuracy and came with computational advantages [34]. In order to better assign weights of the features, an attention feature fusion module (AFF) was introduced into the network to replace the addition operation in the mobile inverted bottleneck convolution MBConv structure of the network. The model achieved 99.5**6**% accuracy for 8 different types of heartbeat in the famous MIT BIH Arrhythmia database. This paper uses pre-trained EfficientNet B7 model. Multi-class classifications of arrhythmia such as 17-class, 15-class, 13-class, and 12- class classifications are used. The proposed approach achieved the highest average of 99.23% accuracy for 13-class classification [31].

2.4.2 MobileNet

A latest work is done using MobileNet V1 architecture in which ensemble of Convolutional Auto encoders are used with Transfer learning [35]. It achieved 97.3% accuracy. It uses binary classification either normal or Arrhythmic. Another approach is used to classify Arrhythmia through an ensemble classifier which combines MobileNetV2 and BiLSTM. I gives an accuracy of 91.7% [3**6**]. The classes used in this work are NSR, AFIB, PVC and LBBB.

2.5 Transfer Learning

Developing a Model from scratch is time taking, so the concept of transfer learning is used. In transfer learning approach a model must take the previously trained or pretrained weights and apply that knowledge by passing the learned features in classification on a custom dataset. Its idea is to freeze all the learnable layers except the dense layer and only modify the number of classes used in custom dataset. More details can be found in chapter 3. Since we look to benefit from transfer learning as it saves computational power by allowing us to leverage the already trained network on millions of images and only pass it to a small dataset which in comparison is nothing. This paper implemented Transfer learning which increased accuracy from 3.**6**7 to 4.89%. [37]. In this paper generative-adversarial-network-based auxiliary domain with a domain-feature classifier negative-transfer-avoidance (GANAD-DFCNTA) algorithm was proposed to bridge the knowledge transfer from distant sources to target domains. Eight benchmark datasets were chosen, with four from ECG datasets and four from the distant domains**:** ImageNet, COCO, WordNet, and Sentiment140.

The proposed method in this paper used fine-tuned ResNet-18 with MIT-BIH arrhythmia dataset in accordance with the AAMI standards [38]. It achieved 90.8% accuracy using transfer learning. This paper pretrained CNN on Icentia11K for the classification of AFIB. CNN predicts heart rate, rhythm and abnormal beat in short frame [39]. It is fine-tuned on the PhysioNet/CinC Challenge 2017 dataset. It shows that pre-training helped with a **6**.57% accuracy rise.

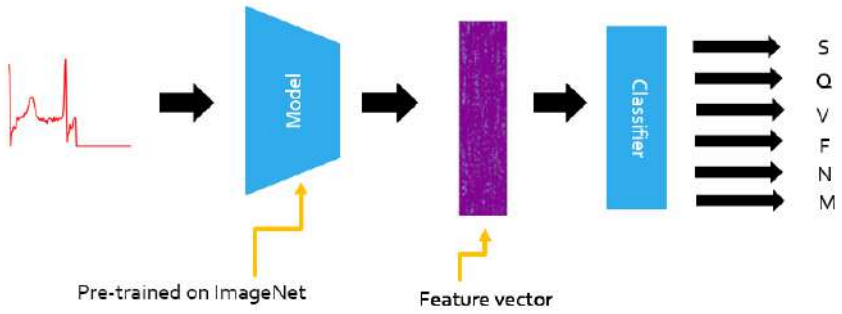


Figure 2.4: Transfer Learning Approach

2.6 Existing Hardware Solutions

Ever since ECG classification algorithms have existed, the aim is to go for hardware devices, but it is not an easy task because the algorithms are expensive, the computational power and resources are limited. This is why reliable and efficient architecture is a need that can better fit in a low resource setting which can be implemented on embedded devices.

A wearable heart rate anomaly detection chip is designed [40]. It uses 1**6**-bit floating pointer numbers for inference. The design of the chip was completed on the TSMC **6**5 nm process. It has an area of 0.191 mm^2, a core voltage of 1 V, an operating frequency of 20 MHz, a power of 1.1419 mW, and storage space of 5.12 kB. The architecture showed accuracy of 97.**6**9% and a classification time of 0.3 ms for a single heartbeat.

An efficient hardware architecture is presented by using 1D U-net. A two-stage pipeline Winograd structure is designed to increase computational power. It also addresses improving resource utilization and overall throughput. A Xilinx Zynq ZC70**6** board is used for the implementation [41]. The results show 1D U-net achieves an average accuracy of 95.55% for the pixel-level classification of five heartbeats. In this work the resource efficiency and computing efficiency reached 8.27 GOPS/kLUT and 123% respectively at clock frequency of 200 MHz A low power co-processor is used for the classification of Arrhythmia [42]. It consumes 8.75µW at 12 kHz, when implemented using 180nm Bulk CMOS technology. Architectural block is given as**:**

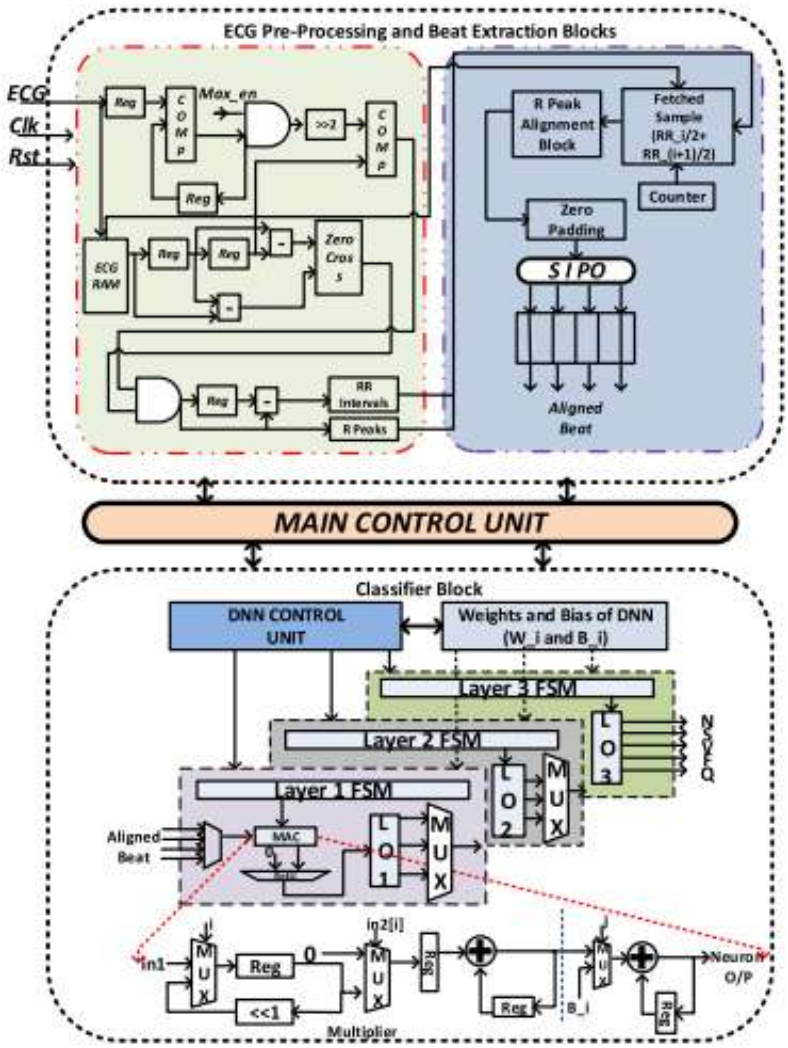


Figure 2.5: Architecture of Co-processor **[42]**

The five classes N, V, F, S and Q are used for the classification of this work. Class S and V are given more importance. It achieved 97.35% accuracy for class-oriented scheme. Resource utilization and ASIC implementation is shown as**:**

Table 2.4: Resource utilization

Resources	Total Available	Resources Utilized	Utilization (%)
Slice LUT	303**6**00	11125	3.**66**
Slice REG	**6**07200	4884	0.80
F7 MUX	151800	1080	0.71
F8 MUX	75900	255	0.33
IOB	**6**00	24	4
BUFGCTRL	32	1	3.125
Total Resources	1139132	173**6**9	1.52

Table 2.5: ASIC details

CMOS Process	SCL 180nm
Area	$1.32mm^2$
Voltage	1.98V
Frequency	12kHz
Dynamic Power	7.5403uW
Static Power	1.2097uW

An embedded system is designed for online and real time ECG classification [43]. The model is tuned to achieve an optimal result. This work has presented the hardware implementation with the predictive model embedded in an NVIDIA Jetson Nano processor. The waveforms for normal sinus, sudden death, arrhythmia, and supraventricular arrhythmia are used in this research. The proposed CNN is shown below**:**

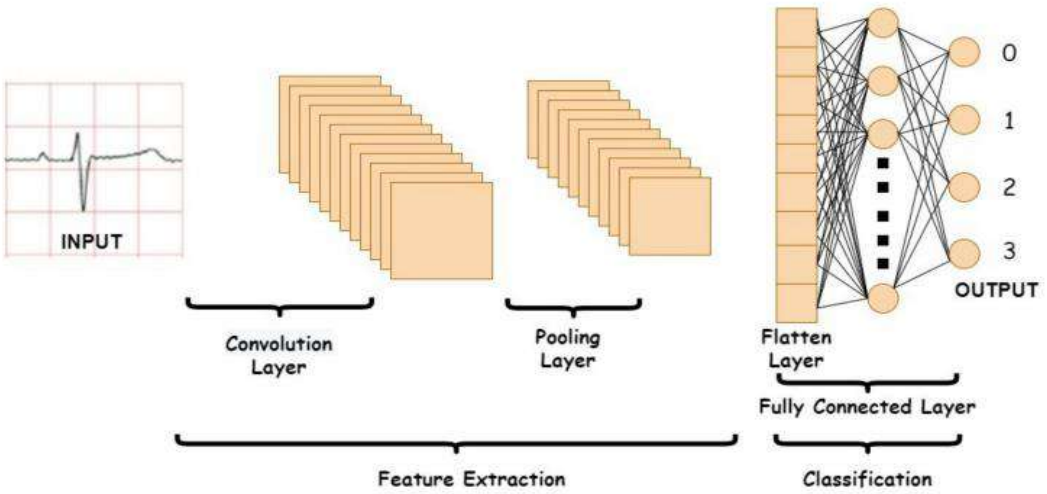


Figure 2.6: Model used in above implementation **[43]**

Instead of ReLU, LeakyReLU as activation function, as it may increase speed and break the zero slope. The parameters and results of this work are given below**:**

Table 2.6: Parameters Details

Training Parameters	Description/values
Optimizer	Adam
Loss	Cross Entropy
Mini-Batch Size	1**6**
Epochs	100
Training Dataset	80%
Validation Dataset	20%

Table 2.7: Classification Report

Class	Precision	Recall	F1-Score	Accuracy (%)	Loss
Arrhythmia	0.99	0.95	0.97	0.959**6**	0.0859
Normal Sinus	0.99	0.93	0.95		
Sudden Death	1.00	1.00	1.00		
Supraventricular Arrhythmia	0.91	0.9**6**	0.93		

The average accuracy is 95.9**6**%.

An ultra-high energy processor is developed [44]. In this work different techniques have been proposed.

- reconfigurable SNN/ANN inference architecture
- reconfigurable on-chip learning architecture
- dual-purpose binary encoding scheme of ECG heartbeats

Fabricated with a 28nm CMOS technology, the proposed design consumes energy of 0.3µJ while achieving accuracy of 97.3**6**%.

Home-care oriented classifier for Embedded Systems is proposed in this paper [45]. Parameters quantization strategy and Channel-level pruning are used to optimize the network.

A reconfigurable accelerator hardware architecture is designed to accelerate the convolution computation on FPGA. The model achieved a promising F1 score of 0.913% and 8**6**.7% exact match ratio, in which parameters and FLOPs are significantly penalized. Real-time analysis is performed. The average processing time is 2.895 s.

Recently ectopic beat classification is proposed on STM32 –based edge device [4**6**]. The classes used in this work are S, V, N, F, and Q. The research uses k-fold cross-validation to choose the best model for hardware implementation. It showed using a 5-layer CNN with pixel 5**6** could get better performance than an 8- layer CNN simplified AlexNet with accuracy of 99.89%. Moreover, the combination of SEmbedNet with an input image size of pixel 5**6** and STM32 can achieve the benefits of 1.3s and 1.1 W per heartbeat in the classification task, and it only takes about 4 seconds. A multiple-STM32 cross-validation platform is built to reduce the validation time. It can process over a hundred thousand heartbeats in just **6**.4 hours.

Classification results are shown below**:**

Table 2.8: Results of CNN model

Input Size (Pixels)	SEmbedNet			Simplified AlexNet			Simplified GoogleNet			Latency (s)	Power (W)
	Sen.	Pre.	Acc.	Sen.	Pre.	Acc.	Sen.	Pre.	Acc.		
5**6**	88.82	97.03	99.89	79.04	N/A	99.78	82.17	87.58	99.1**6**	1.33	1.10
112	91.29	95.**66**	99.84	80.42	98.90	99.73	79.93	87.11	98.71	5.34	1.30

This paper also compares the proposed methodology with other hardware implementations. The results of comparison are show below**:**

The architectures like VGG1**6** and MobileNetV2 are implemented on Raspberry Pi which showed 0.90 and 0.94 respectively [21].

Table 2.9: Comparative results with other hardware implementations

Author	Year	Method	Class Type	Hardware	Total Accuracy (%)
S.Raj [9]	2018	DOST+LSTM	(N, S, V, F, Q)	V	9**6**.08
Y.Zhao [11]	2019	ANN	(N, S, V, F)	V	98.00
Y.Xu [10]	2019	SVM	(N, S, V)	V	89.00
N.Wang [12]	2019	CNN	(N, S, V, F, Q)	N/A	99.00
Proposed	2022	CWT+CNN	(N, S, V, F, Q)	V	99.89

2.7 Critical Analysis:

Different techniques exist for the removal of noise and extracting features in an ECG signal. Empirically, the algorithms are expensive, the computational power and resources are limited. This is why reliable and efficient architecture is a need that can better fit in a low resource setting which can be implemented on embedded devices.
If the goal is to design a hardware or propose a hardware architecture, smaller and simpler Architectures are more suitable for that. Factually, different applications have different architecture more suited to them and the inconsistency of the classes in MIT BIH Arrhythmia used throughout the research needs to be studied further. Data imbalance needs to be taken care of.

\

Chapter – 3

Embarking into the Depths: Exploring Deep Learning, CNNs, and Multiple Architectures

This chapter provides all the details relevant to this research. It contains background knowledge of implementation. It entails knowledge of software and hardware techniques that lead to completion of this work. This chapter covers a Machine Learning technique known as End-to-End learning.
It has enough information related to Deep learning, Transfer Learning, Neural Networks, and CNN Architectures like EfficientNet, MobileNet, VGG1**6**, AlexNet and Resnet18.

3.1 End-To-End Learning

In End-To-End learning, model learns all the steps during initial and final output phase, hence reducing the effort. In this way models are trained to automatically extract features, learn, and work with the data.

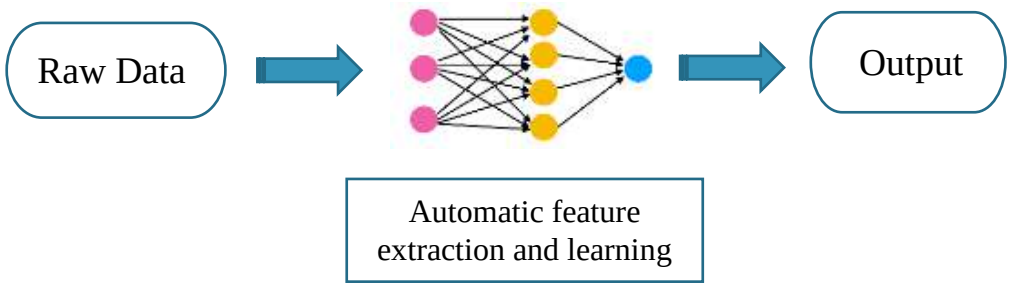


Figure 3.1: End-To-End Learning

3.2 Deep Learning

In today's fast paced world, Artificial Intelligence has been introduced to almost everything. From our very own homes to offices and in between, everything is under the spell of AI. Deep Learning is the heart of Machine Learning that comes from fact that we add more layers to learn from the data. As the name "Deep" suggests it is used for very large data, hence having extensive layers.

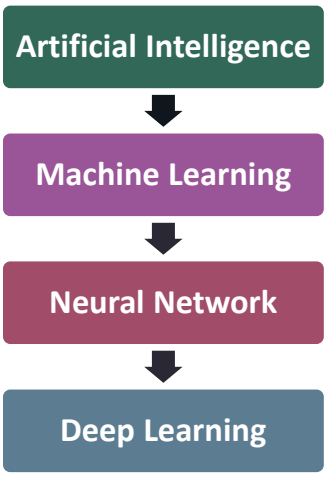


Figure 3.2: Hierarchy of Artificial Intelligence

In a traditional deep learning network unstructured and labelled data is fed to a model which acts as a black box. As a result of what the model learned, it predicts the output. But how do models learn? It learns by calculating the difference in the predicted output and actual true label, the model trains itself to minimize the difference. Unstructured data are qualitative in nature such as an image where we can experience a variety of features in different ways. Other examples are audio, video etc. Unlike structured data which are quantitative such as product databases, a list of housing prices against size or area, etc. In deep learning, the input data passes through the layers where features are extracted and predicts output through a classifier layer, respectively.

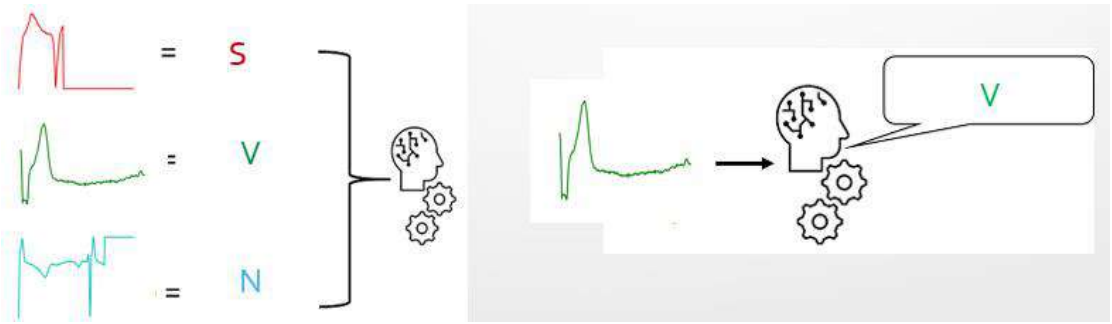


Figure 3.3: Model predictions

Deep learning is a vast concept, and it possesses different techniques that are application specified. Some of the techniques are CNNs, RNNs, LSTM, GANs, RBFNs, auto encoders. The focus of this work is on Convolutional Neural Networks for a multi-class classification problem. To fully understand CNNs, it is important to dive into Neural Networks first.

3.3 Neural Networks

A simple Neural Network comprises of artificial neurons stacked to form a single layer which learns and gets smarter by analyzing the data patterns. It contains three layers

i.e., input layer, hidden layer, and output layer. Neural Networks are inspired by the Human Brain, its structure and functionality. Neurons are connected in a stack, process the information, and generate output. The input layer accepts input features from the outside raw data. No computations are being performed here. From the input layer nodes pass information to the hidden layer or layers. In the hidden layer computation is performed which transfers the details to the output layer. The output layer brings up the learned information to predict.

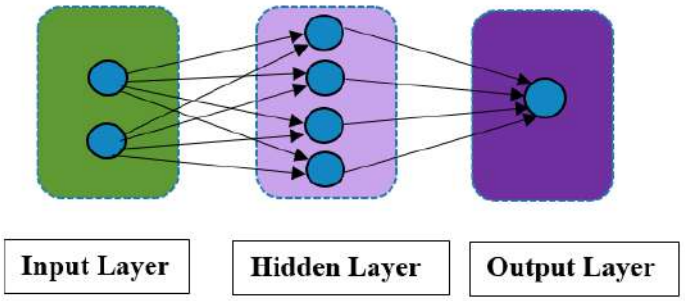


Figure 3.4: A traditional Neural Network

Each neuron in a former layer is connected to the neurons present in the next layer. These layers are also called fully connected or dense layers because of their dense connections. Hence deeper layers with more neurons make up a deep neural network. In the hidden layers a mathematical computation is performed which is represented as**:**

$$Y = w * X + \mathbf{b}$$

Where w is the weight containing vector of each connection
X is the input feature vector.
b is a scalar bias that is added to the product of weight and input
Y is the output feature vector or number of neurons in that layer.
In this way the weights are updated forming a forward propagation.

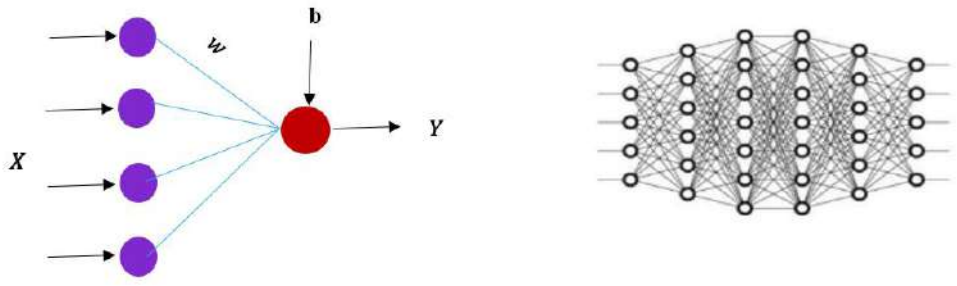


Figure 3.5: Hidden Layer in a Neural Network

For very deep neural networks deeper features can be extracted but with more layers comes more neurons. Having said that deeper networks are prone to overfitting because of redundancy as there is a possibility of overlapping of features.

3.3.1 Convolutional Neural Network (CNN)

A typical Convolutional Neural Network can be thought as a combination of two components**:**

- ✓ Feature Extraction
- ✓ Classification

A convolutional neural network is amalgamation of multiple layers namely convolution layer, pooling layer, fully connected layers used for classification of images. These layers work on a multi-dimensional input feature map and can perform different operations which can be unattainable using Artificial Neural Networks. In a simple Convolutional Neural Network, input is fed to the first convolutional layer where features are extracted after it goes through pooling layer. As we move through, the feature dimensions become smaller, and depth becomes larger at each convolution before being flattened into a vector for the fully connected layer. The fully connected layer at the end is used as a classifier.

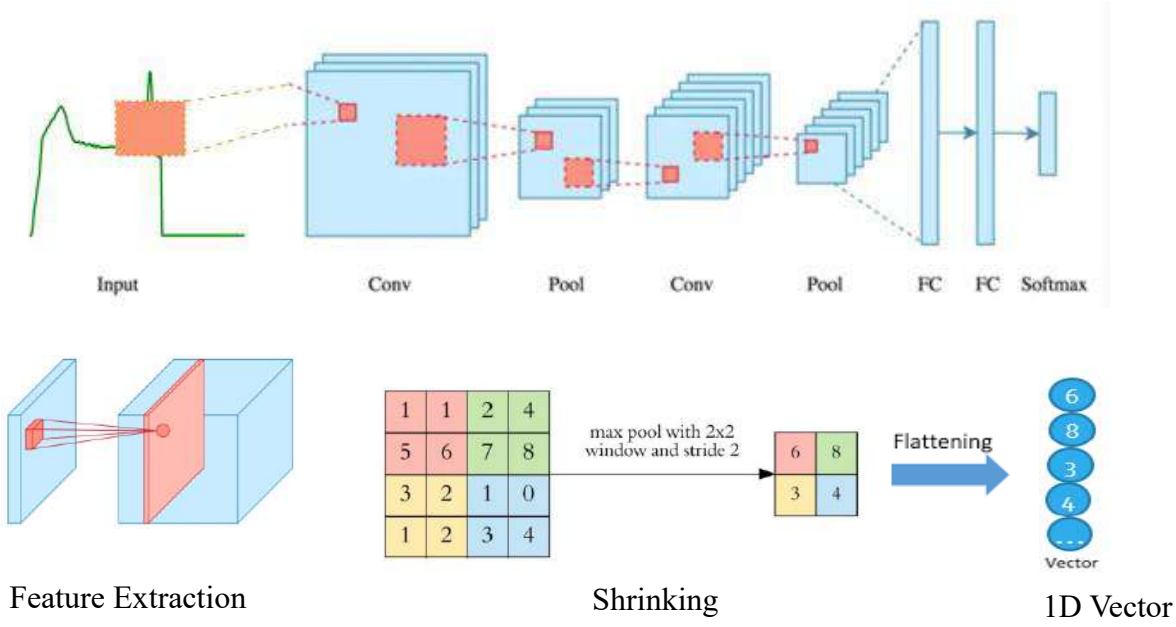


Figure 3.6: Representation of a Convolutional Neural Network

3.3.1.1 Dimensional Convolutional Layer

This layer employs a 2D kernel or filter which moves across the 2D input and generates a corresponding 2D output (also known as feature maps). These kernels contain weights that require training, such as a 3 × 3 kernel having 9 weights and a bias. Once these weights are trained, the kernels can extract significant information from the input feature map, which is a contrast to an ANN. A convolutional layer can extract meaningful features better than fully connected layers for an image. The reason being

that the coherency or correlation between a group of pixels gets lost in a fully connected layer where the image is flattened into a 1D vector. A kernel provides the following advantages**:**

- The kernel operates on a group of pixels that maintain their correlation; hence they result in better feature extraction than a fully connected layer where the image flattens into 1D vector.
- A kernel decreases the operations and parameters significantly.
- It extracts more information from the image.

Convolutional layer can have multiple input channels and kernels. Kernel operates a sliding protocol for each channel in multiple input channels and the outputs are summed into a single output channel by element-wise summation. This equation defines a single convolution operation**:**

$$\sum_{u=-k}^{k} \sum_{v=-k}^{k} I(u,v)K(i-u,j-v)$$

i, = output pixel at location (i,),
k = kernel size,
K = kernel,
I = input feature map.
Multiple kernels allow for each kernel to have its own output channel.
No. of output channels = No. of kernels
The output dimensions depend on the input size and kernel size. For example, if we apply a 3 × 3 convolution (kernel size of 3) on an input image with dimensions 25**6** × 25**6** × 3, and use 32 kernels, the resulting output dimensions will be 254 × 254 × 32.
This is generalized in the following equations**:**

$$input\ dimensions = l \times m \times n$$
$$kernel\ size = i \times j \times k$$
$$output\ dimensions = (l - i + 1) \times (m - j + 1) \times k$$

$l \times m$ = dimension for a single input channel
n = no. of input channels
$i \times j$ = dimension for a single kernel (normally $i = j$)
k = no. of kernels
The figure below represents blue matrix as a single channel 2D input feature map, the single 3x3 kernel displayed as light grey, convolves with the input, and produces a single channel output feature map. The figure demonstrates how padding can help conserve the dimensions i.e., the output dimensions are the same as the input without padding.

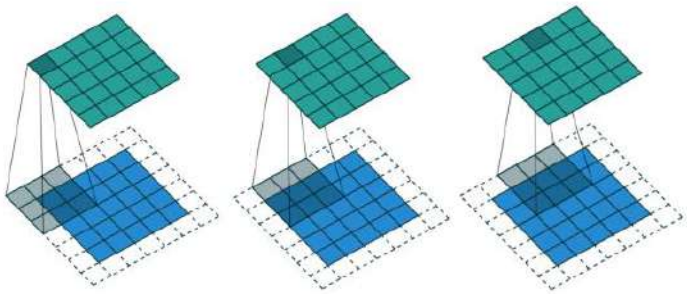


Figure 3.7: Visual representation of kernel (light grey) sliding across the padded input (light blue) producing the corresponding output (light green) [47]

From the figure above, we can visualize how padding can help us to conserve the dimensions i.e., the output dimensions are the same as the input without padding. Padding such as zero-padding or replicate is one of the useful techniques to preserve these dimensions.

3.3.1.2 Pooling Layer

Pooling layer is typically used after one or more Convolutional layers. The main objective of a pooling layer is to lessen the spatial dimensions (i.e., height and width) of the input feature map, while keeping the number of channels consistent. By reducing the spatial dimensions of the input feature map, the Pooling layer helps to reduce the computational cost and memory requirements of the subsequent layers in the CNN, while also preventing overfitting by introducing some degree of translation invariance to the learned features.

Mostly used Max Pooling layer, which operates by dividing the feature maps into non-overlapping known as pooling regions and generating the maximum value within each sub-region as the corresponding output pixel. This has the effect of down sampling the input feature map, while retaining the most salient features.

Pooling layers include Average Pooling, which computes the average value within each sub-region, and L2-norm Pooling, which computes the square root of the sum of squares within each sub-region. However, Max Pooling is the most widely used type of pooling layer due to its effectiveness and simplicity.

However, too much pooling can result in loss of information and spatial resolution, so the choice of pooling size and stride should be carefully considered depending on the specific application.

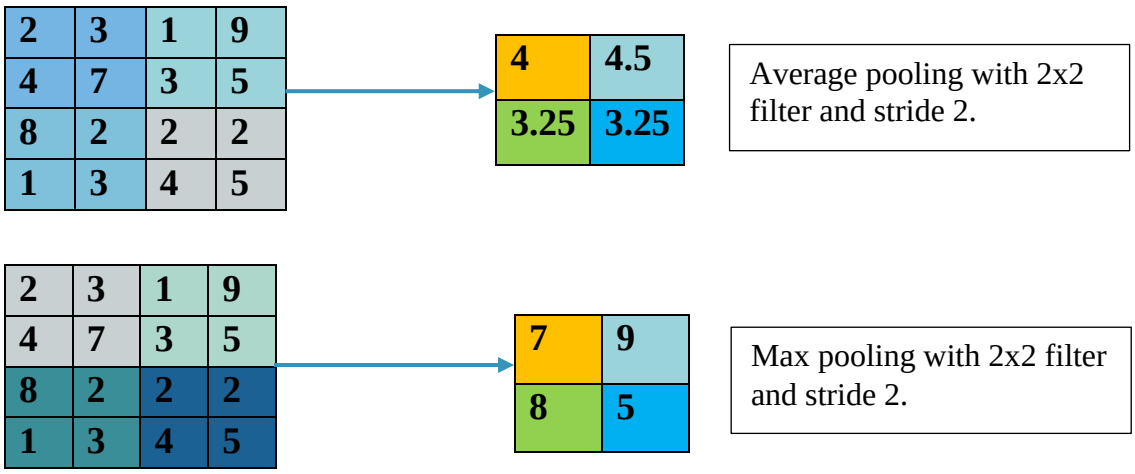


Figure 3.8: Max and Average Pooling of 2x2 filter

3.3.1.3 Activation Layer

Convolutional layers have a function called the activation function. This function allows the linear output to be transformed into non-linear so that real world scenarios can be performed. Hence, it introduces non-linearity in the network. This means, the network can learn non-linear and more complex relationships between input and the output data. Activation layer may also be used as a separate layer to give flexibility in dataflow manipulation. In addition to this, the activation layer normalizes the output of the previous layer to improve the efficiency of the network. It limits the weighted sum of input in a specific range. But all the layers in CNN do not require activation layer. For example, the pooling layer is a linear operation, so it does not require an activation function.

Activation function is represented as**:**

$$Y = activation\,(w * X + b)$$

Where activation can be a ReLU function, SoftMax, Sigmoid, tanh etc.

It can also be used as a separate layer which gives flexibility in manipulating the dataflow. The most used activation functions are given below.

3.3.1.3.1 Sigmoid

The sigmoid function is an activation function which is a mapping of input value to a value between 0 and 1, which translated as probability of the input belonging to a particular class. The sigmoid activation function is used in binary classification because it scales the values between 0 and 1. Then we can select a threshold value e.g., 0.5 above which will be classified as true and vice versa. Mathematically this can be represented as**:**

$$a\,(z) = 1 / (1 + e - z)$$

Where z is the input to the activation function and a (z) is the corresponding output. By plotting the graph, we can see that the output of the activation function is exactly 1 or 0 when the input z approaches +∞ or -∞ respectively i.e.**:**

$$\lim_{z \to +\infty} a(z) = 1, \qquad \lim_{z \to -\infty} a(z) = 0$$

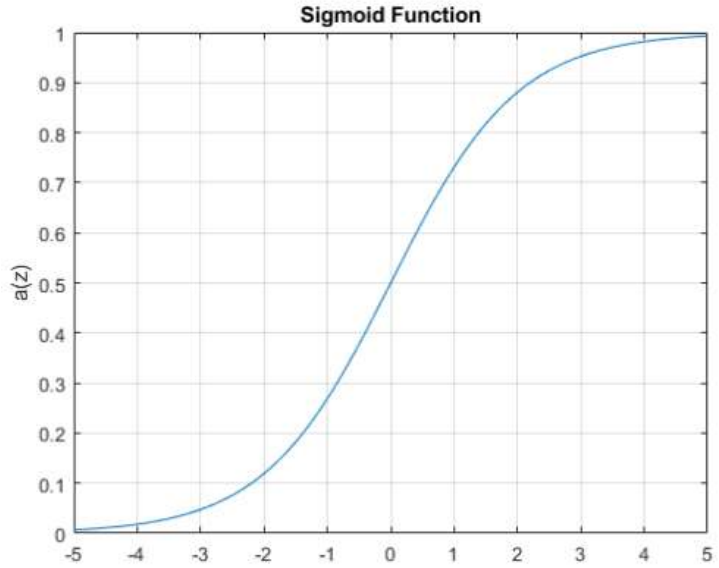


Figure 3.9: Sigmoid Activation Function

One of the drawbacks of the sigmoid function is that it can cause a problem called "vanishing gradients" when used in intermediate layers. This happens when the gradients of the loss function regarding the weights become very small, slowing down the convergence or fails to converge to global minima or maxima, which can make it difficult for the network to learn. Hence, this type of activation function is usually used at the last layer or the classifier as a result, other activation functions such as the ReLU function are often used instead of the sigmoid function in intermediate layers.

3.3.1.3.2 SoftMax

The softmax function takes a vector of real-valued inputs and applies the exponential function to each element of the vector. It then divides the sum of all exponentials to normalize the resulting values. Its output is a vector of probabilities that adds up to 1.
This function is used for a multi-class classification problem with a goal to predict the probability of each class, given the input. Hence, it is used at the last layer or the classification layer. The length of the vector is equal to the number of classes. The class with the highest probability among all the probabilities is considered and the data point will belong to that class.
Mathematically representation of Softmax function is as follows**:**

$$a(z) = \frac{e^z j}{\sum_{k=1}^{K} e^z k} \quad for\ j = 1, \ldots, K.$$

Where zj is the input value for jth class, K is the total number of classes, and a (z) is the softmax value for jth class.

One of the advantages of the softmax function is that it produces a smooth and differentiable output, which makes it suitable for use in gradient-based optimization algorithms, such as stochastic gradient descent, which are commonly used to train neural networks.

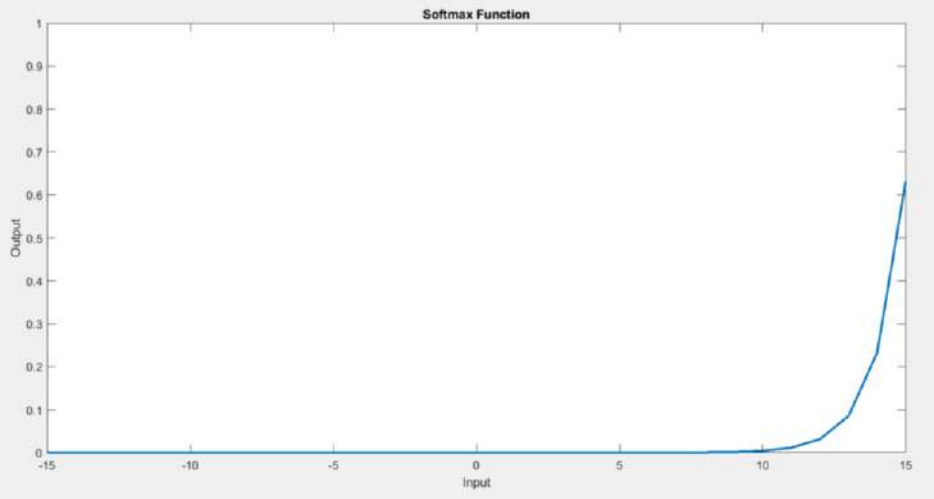


Figure 3.10: Softmax Activation Function Graph

3.3.1.3.3 ReLU

ReLU or rectified linear unit is biologically inspired, as it resembles the firing pattern of real neurons in the brain. Due to its simplicity, computational advantage, and efficiency, it is very popular. It is resilient to vanishing gradient problem which helps the model to in training and better performance hence they are mostly used in intermediate layers. ReLU function does not activate all the neurons at the same time as neurons will be deactivated when the output is 0 or less than 0.

It is a linear-piecewise activation function that allows positive values to accentuate, otherwise returns 0. Mathematically it is represented as**:**

$$a\ (z) = \max\ (0,\ z)$$

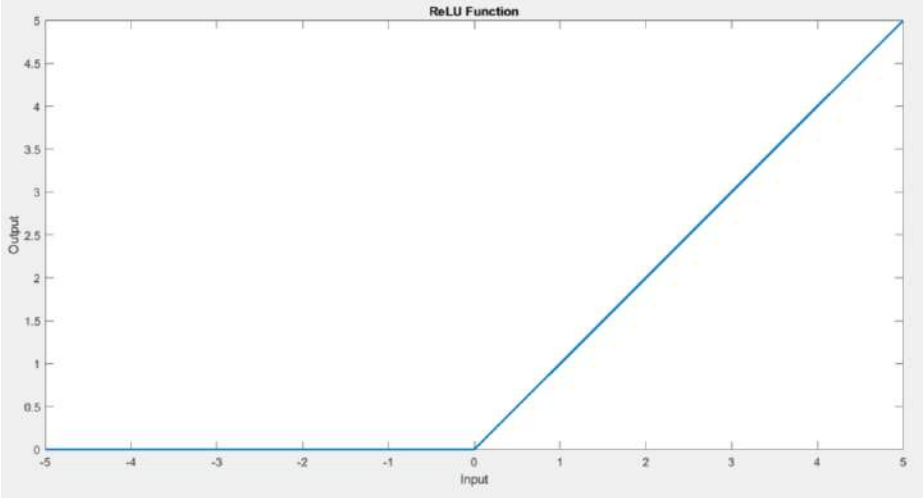


Figure 3.11: ReLU Activation Function Graph

3.3.1.4 Loss and Loss Function

Generally, loss is the error associated with the CNN model given the input. It is calculated using predicted value minus actual value/label. Let us denote the loss as Li for the ith training example.

$$Li = \hat{y}\, i - y\, i$$

Where $\hat{y}\, i$ is the predicted value and $y\, i$ is the actual value for ith training example.

A loss function evaluates the solution which will be used by the optimizer. Normally during training, it is calculated over the entire batch by averaging the individual losses in that batch. Similarly, this averaging can be extended over the entire dataset which gives out the overall average loss function of the CNN model. In machine learning different types of loss functions exist like MSE, Hinge Loss, and Cross Entropy etc. These loss functions are being used depending upon the nature of the problem you are working on. For a binary class problem binary cross entropy is usually used. For multi-class problem categorical cross entropy is used. There is sparse categorical entropy as well for multi-class problems.

- **Categorical Cross Entropy (CCE)** generates a one-hot array containing the probability for each category.
- **Sparse Categorical Cross Entropy (SCCE)** generates an index of the most likely to match category.

For Categorical Entropy Loss, the actual value which is a binary corresponds to a single class while the prediction is a probability. Since the last activation function in the CNN model is a SoftMax activation that returns the probability for classification. This loss function costs on the difference between the probability and the expected value. Hence larger cost/penalty for larger difference from the actual value. It is represented as**:**

$$CCE = \frac{1}{N} \sum_{1}^{N} t_i \log(p_i)$$

Where ti is the ground-truth which is one-hot encoded, pi is the probability of softmax, n is the number of classes and N is the total number of images over which the loss is calculated by taking average.

3.3.1.5 Epochs

Epochs represents a full iteration of the dataset during training. The entire dataset is analyzed by the model in one epoch, and parameters are adjusted as the model learns the data. The number of epochs to run depends on a variety of variables, including the size and the complexity of the model. Having a set number of epochs for which the validation loss stays constant is a smart idea. Less epochs can lead to underfitting, whereas many epochs can lead to overfitting. The number of epochs is determined using a variety of methods**:**

- **Manual choosing:** Begin with a small number of epochs and gradually expand it. When the performance on the validation set starts to decline, you can stop.
- **Early Stopping:** When a metric stop improving, the training process is automatically terminated.
- **Cross Validation:** The dataset is partitioned into several folds, and in each epoch, the model is trained and assessed on several folds.

3.3.1.6 Learning Rate

The parameters are updated in small steps while the model is trained. It is crucial since it determines how quickly or slowly a model reaches the best parameters for minimizing the loss.

With a high learning rate, the model updates parameters in greater increments, which could lead to a faster convergence. The model may overshoot the optimal parameter values and fail to converge, leading to subpar performance, if the learning rate is set too high. On the other hand, a low learning rate causes the model to take smaller steps and converge more slowly. The model may become stuck in local minima or plateaus if the learning rate is set too low, which will hinder convergence and extend training times.

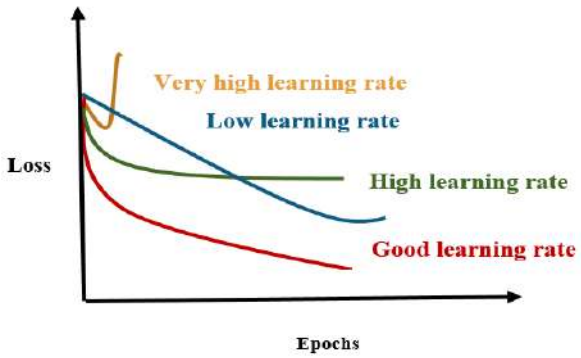


Figure 3.12: Epochs vs Loss

For an optimal learning rate manual selection or scheduler is used.

Manual Selection: You can start with a reasonable initial learning rate based on empirical guidelines or previous experience.

Scheduler: You can use predefined learning rate schedules, such as decreasing the learning rate by a fixed factor after a certain number of epochs or when a specific condition is met. Usually step decay is used.

Grid Search: You can try different learning rates and see which model behaves better.

3.3.1.7 Learning curves

Loss of updates after each batch results in fluctuations. Too much noise or jumping loss can never converge to local minima while larger batch sizes with no noise at all/fluctuation can stuck in local minima. Little or less fluctuation is a good solution. The gap between training and validation accuracy shows overfitting.

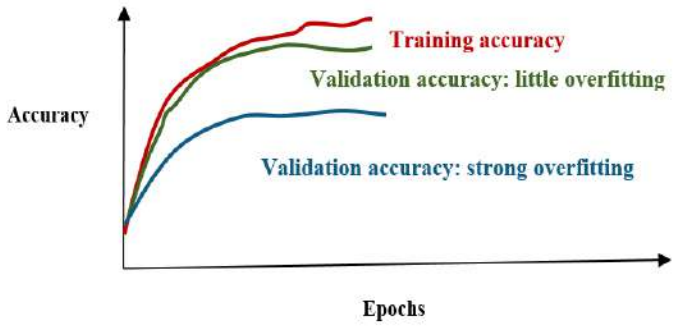


Figure 3.13: Epochs vs Accuracy

Batch Size

It is the number of images used for training in a single iteration.

Smaller batch sizes enable quick computations and reduce the number of training samples needed in a single iteration.

Memory:

Larger batch sizes call for more memory, which may not be enough to accommodate them, resulting in out-of-memory problems that impair performance.

On the other hand, lower batch sizes need less memory, which is advantageous in contexts with memory restrictions.

Convergence:

More randomness is introduced into the training process when smaller batch sizes are used. This stochasticity can function as a type of regularization, aiding the model's ability to generalize and possibly enhancing its performance on unobserved data.

Since the model updates its parameters more frequently with smaller batch sizes, the model may explore the solution space more extensively. The model may be able to identify better solutions and escape local minimum points as a result.

Smaller batch sizes could occasionally lead to a faster convergence of the model during training. This is so that the model can be tuned more carefully and prevent overshooting the ideal values by using smaller updates to the parameters.

Generalization Ability:

Small batches generalize well and allow us to start learning before seeing all the data and it is highly probable to converge on optimal solution. While larger batch generalizes poorly hence impacts on the performance and it is less likely to converge on optimal solution. Generally, it is taken as a power of 2. Batch size 1**6** or 32 is a usual choice. But it depends on the dataset and application.

3.3.1.8 Optimizer

Optimizers are the learning algorithm used to update the weights and biases which aid in reduction of cost function in back propagation. There are different optimizers used in machine learning like Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) optimizers. Difference in both most used optimizers is given below**:**

Updating Model Parameters:

SGD computes the loss function's gradient with respect to the parameters for each example in the training data and steps in the direction of the negative gradient at a fixed learning rate. On the other hand, Adam uses a more sophisticated update rule that includes both momentum and second-order adaptive learning rate information. This allows Adam to adaptively adjust the learning rate based on gradient history, which in some cases can result in faster convergence and better performance.

Momentum:

Adam uses momentum, a technique that speeds up the convergence process. Momentum adds a fraction of the previous update to the current update, which can help the optimizer bypass local minima and reach convergence faster. SGD, on the other hand, does not consider Momentum by default, although it can also be combined with Momentum in the form of variants such as SGD with Momentum.

Adaptive Learning Rate:

Adam adjusts the learning rate for each parameter based on the estimated second-order moments of the gradients, allowing the learning rate to be adaptively scaled for different parameters. This can be particularly useful in scenarios where the gradients of different parameters are of significantly different magnitudes. SGD, on the other hand, uses a fixed learning rate for all parameters throughout the training process, which may not be optimal for all scenarios.

Parameter Updates:

Adam updates the model's parameters using a combination of the gradient of the loss function and the accumulated momentum and adaptive learning rate information. SGD, on the other hand, updates the parameters using only the gradient of the loss function scaled by a fixed learning rate.

Memory Requirements:
Adam requires additional memory to store the accumulated momentum and adaptive learning rate information for each parameter, which can increase the memory requirements compared to SGD, which only requires storing the gradients.

3.3.1.9 Training /Test/Validation set.

Generally, a dataset is split into train set, test set and validation set. The training set is with large samples on which the model learns patterns and test set is used to make prediction which is the unseen data. While validation set is used to make predictions during training to understand and observe how well a model generalized on new data. Since weights update in training set while back propagation is turned off during validation set makes it possible to observe the behavior of model in predicting while training so quick changes can be made for a better generalization ability and optimized parameters.

3.3.1.10 Bias-Variance Trade-off

Bias and variance are two important concepts that describe the performance and generalization ability of a model.

Bias:
Bias is the error that results from using a simple model to approximate a complex real-world problem. A biased model is more likely to consistently commit systematic mistakes. It might oversimplify the underlying information or issue, leading to wrong forecasts. High bias can cause underfitting, where the model performs poorly on both the training data and new, unobserved data and fails to capture the underlying patterns in the data.

Variance:
Variance is the error caused by the model's sensitivity to the training set that was employed. A high variance model could be too complex and sensitive to the training data, which adds in overfitting. When a model performs well on training data but struggles to generalize to fresh, untried data, overfitting has taken place. Poor generalization performance might result from high variance since the model may be highly specialized to the training data and unable to accommodate new input data.

Trade-off:
A model's complexity and flexibility increase when its bias is minimized, which may lead to larger variance as the model becomes more sensitive to the particular training data. On the other hand, a model becomes more stable and less prone to overfitting when its variance is lowered, but it may also introduce more bias because it loses flexibility.

Balance:
Balancing bias and variance is an important goal in machine learning model development. A model with high bias may require more complex features or a larger dataset to capture underlying patterns, while a model with high variance may require

regularization techniques such as regularization or increasing the training data size to reduce overfitting.

3.3.1.11 Metrics

There exists different methods or metrics to evaluate the model's ability to predict correctly. Since the nature of this work being multi-classification problem, different metrics are used to observe model's behavior and accurate detection. One of which is Confusion Matrix. It is a representation of the true positive, true negative, false positive, and false-negative cases.

To evaluate model's performance, it gives out predicted labels and actual labels. Based on confusion matrix classification report is made which entails precision, recall, F1 score. In order to interpret a confusion matrix, you can observe the values in the diagonal (TP and TN) which is the accurate predictions made by the model.

The off-diagonal (FP and FN) values represent incorrect predictions made by the model.

A well-performing model possesses higher true positive and true negative, and a lower false positive and false negative shows correct predictions. The model with a high number of false positive and false negative, indicates inaccurate predictions.

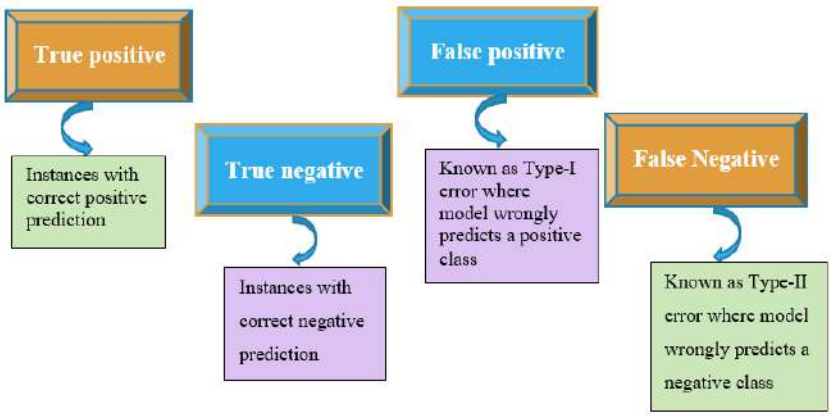


Figure 3.14: Errors in Machine Learning

Accuracy is a measure of correct predictions, calculated as

$$(TP + TN) / (TP + FP + TN + FN)$$

Precision is a measure of true positives among the positives predicted by the model, calculated as

$$TP / (TP + FP)$$

Recall is a measure of true positives among the actual positives, calculated as

$$TP / (TP + FN)$$

F1-score is a measure of harmonic mean of precision and recall, calculated as

$$2 * (Precision * Recall) / (Precision + Recall)$$

3.3.1.12 Receiver operating curve (ROC)

Area under the ROC is used to represent each class and its average. The ROC is plotted with the false positive rate (FPR) on the x-axis and the true positive rate (TPR) on the y-axis. The range of AUC is between 0 and 1. If the AUC is 1 then this means that the model is a good classifier. If AUC is below 0.5, it means random classification.

3.3.2 Transfer Learning

It is a technique where model is trained on pre-trained network taking benefits of previously trained weights. It is an art of reusing a model leveraging the knowledge of previous features to improve performance of new task by saving significant amount of time and resources. Transfer learning can be implemented as

- **Feature Extractor**
- **Fine-tuning Network**

In transfer learning as a fixed feature extractor, the model takes the pretrained knowledge and passes it on the new data by only adjusting the new number of classes in the classifier layer. While in fine tuning model is trained and modified to re-train to achieve specific goal.

3.3.2.1 AlexNet Architecture

AlexNet architecture is widely known as a winner of ImageNet ILSVRC challenge due to its revolutionary amendments in a typical CNN. For example, ReLU being introduced for the first time as a replacement to tanh and sigmoid functions which were slower to train.

Moreover, drop out was first introduced to overcome overfitting as well as Local Response Normalization (LRN) for better generalization with ReLU as the learned variable being unnecessarily high. The idea behind is to amplify excited neurons and dampen the neighboring pixels. This architecture has a total of 8 layers from which 5 are convolutional layers, 3 are fully connected layers with max pooling and dropout being used. It has **6**2.3 million learnable parameters.

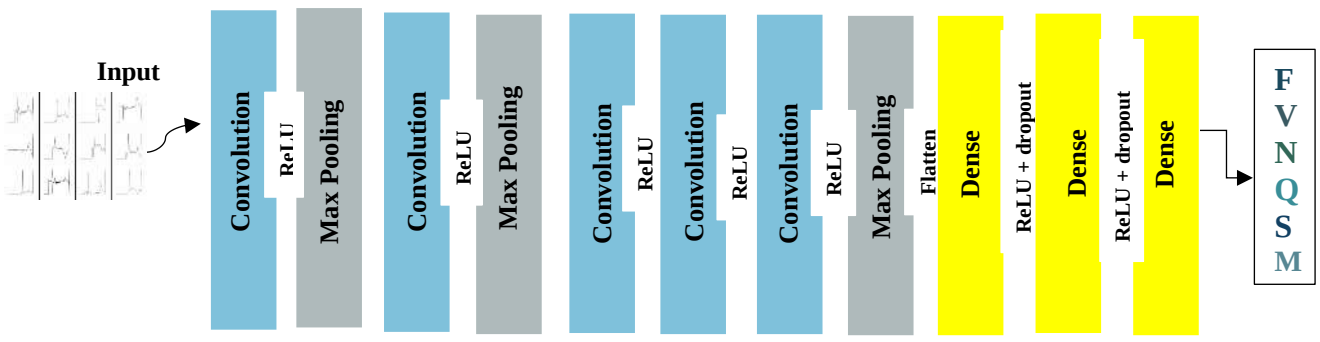


Figure 3.15: Architectural Diagram of AlexNet

Table 3.1: AlexNet Layers

Layer	# of Filters	Stride	Padding	Filter Size	Output Feature map
Input layer	-	-	-	-	227x227x3
Conv1	9**6**	4	-	11x11	55x55x9**6**
Conv2	25**6**	1	2	5x5	27x27x25**6**
Conv3	384	1	1	3x3	13x13x384
Conv4	384	1	1	3x3	13x13x384
Conv5	25**6**	1	1	3x3	13x13x25**6**
FC	-	-	-	-	409**6**
FC	-	-	-	-	409**6**
FC	-	-	-	-	1000

Note that in case of customized input, the output size of a convolution layer is calculated as:

$$Output = ((Input - filter\ size)/\ stride) + 1$$

3.3.2.2 VGG16 Architecture

VGG architecture came out of the need for reduced computational time and parameters. It has different variants which only differ in the number of layers. VGG1**6** has a total of 1**6** layers out of which 13 are convolutional layers, 3 dense layers and 5 max pooling layers which are not learnable. Since AlexNet has variable kernels size for different layers which increases the parameters while vgg1**6** has a fixed kernel size of 3x3 in all layers with stride 1 and max pool kernel size of 2x2 and a stride of 2. The idea behind is to reduce the number of parameters.

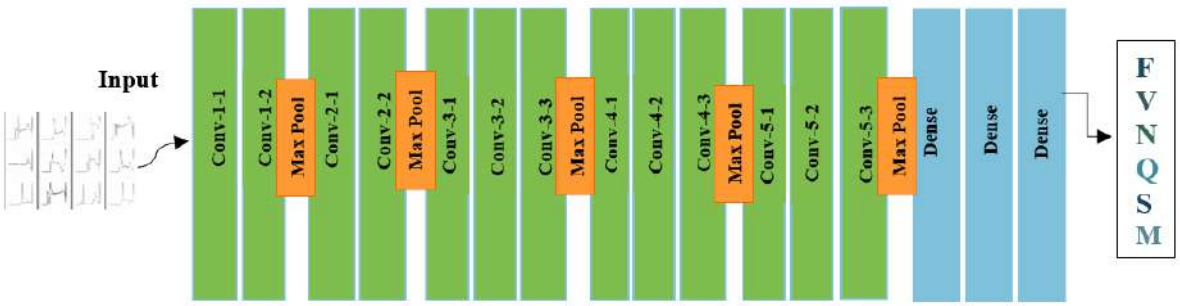


Figure 3.16: Architectural Diagram of Vgg1**6**

Table 3.2: Details of Vgg16 Layers

Layer	# of Filters	Stride	Padding	Filter Size	Output Feature map
Input layer	-	-	-	-	224x224x3
Conv1	**6**4	1	same	3x3	224x224x**6**4
Conv2	128	1	same	3x3	112x112x128
Conv3	25**6**	1	same	3x3	5**6**x5**6**x25**6**
Conv4	512	1	same	3x3	28x28x512
Conv5	512	1	same	3x3	14x14x512
FC	-	-	-	-	409**6**
FC	-	-	-	-	409**6**
FC	-	-	-	-	1000

Conv1 and Conv2 have 2 convolutional layers stacked while Conv3, Conv4 and Conv5 have 3 convolutional layers stacked in a block. Same padding means the output shape is same as of the input. It has 138 million parameters.

3.3.2.3 ResNet18 Architecture

ResNet18 is a small architecture with 18 trainable layers. It introduced skipping connections which are used solely to overcome the problem of vanishing gradient. Vanishing gradient happens when the gradients of loss function fail to update properly as they become so small leading to 0 where no more updates to weights occur. It leads to stagnant learning process. For this problem to be eradicated, skipping connections are introduced so if the gradient becomes too small it skips that and move to deeper layer providing improved learning instead of slowed process. It uses a residual block which is repeated throughout the model. Instead of learning the mapping from input to output it learns the mapping from input to output plus the identity function that is a short connection called identity connection. Due to which no vanishing gradient occurs. This architecture has 2 pooling layers 3x3 max pooling at the start and 7x7 average pooling at the end. It has a total of 11 million parameters.

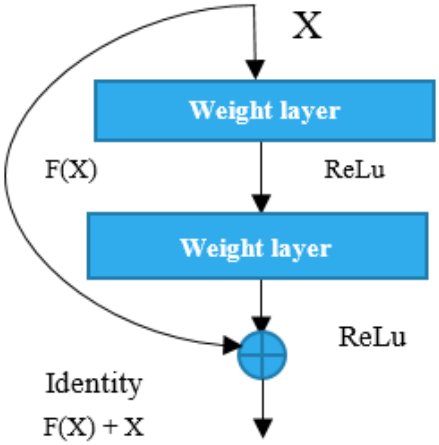


Figure 3.17: Residual Block

Table 3.3: Details of ResNet Layers

Layer	# of Filters	Stride	Padding	Filter Size	Output Feature map
Input layer	-	-	-	-	227x227x3
Conv1	64	2	1,1,1,1	7x7	112x112x64
Conv2	64	1	0.5,1,1,1	3x3	56x56x64
Conv3	128	1	0.5,1,1,1	3x3	28x28x128
Conv4	256	1	0.5,1,1,1	3x3	14x14x256
Conv5	512	1	0.5,1,1,1	3x3	7x7x512
FC	-	-	-	-	1000

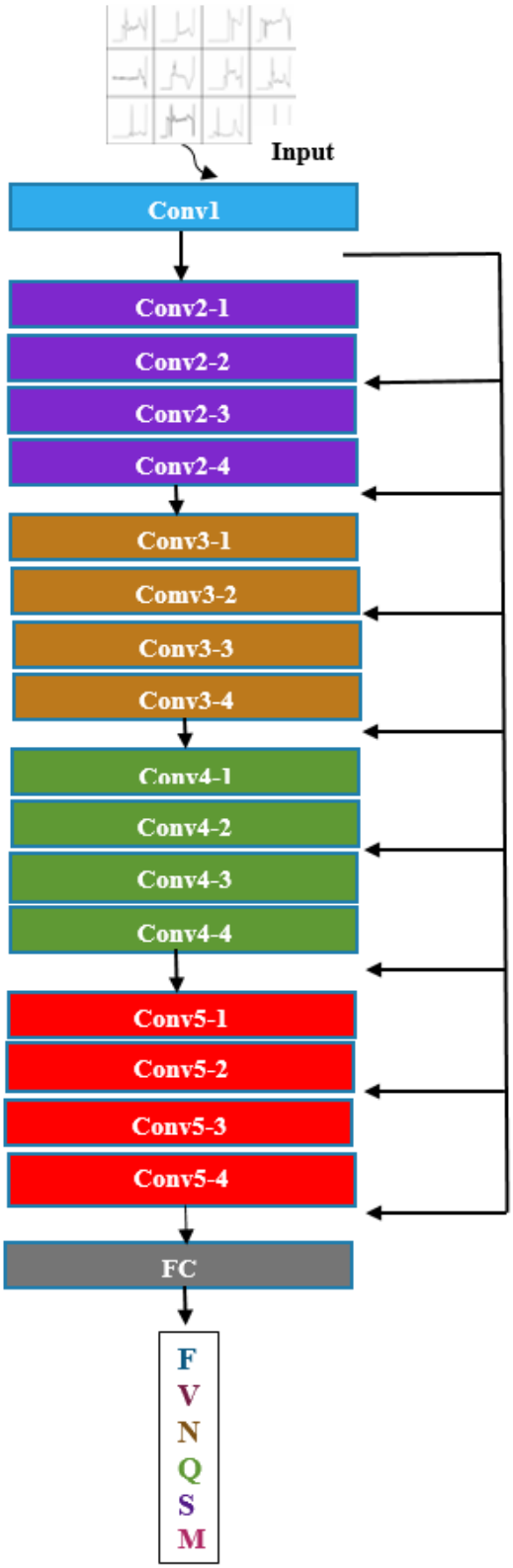


Figure 3.18: ResNet Architecture

3.3.2.4 EfficientNet Architecture

Since many state-of art ConvNets have hit memory limits researchers looked for resource efficient architecture without compromising on accuracy. The problem with deep networks is that lots of computation happens, with lots of layers comes lots of processing. It becomes time consuming. In a traditional convolutional neural network, we often scale depth only, but EfficientNet came up with the idea of compound scaling in which we not only scale depth, but width and resolution can also be scaled. By scaling depth means adding more layers, by scaling resolution means size of an image is increased, by scaling width means channels/features are increased. The paper performed experiment and proposed that by scaling only one of these can lead to saturation as a point reaches where no more scaling helps. Thus, the idea of compound scaling emerged. In compound scaling resolution, width and depth are scaled together.

How much depth, width, resolution scaling is required?

For compound scaling network scaling factor F is represented as

$$F = \alpha . \beta^{\theta} . \gamma^{\theta}$$

Where α is the depth,

β is the width,

γ is the resolution,

θ is a hyper-parameter.

Using grid search it is decided that depth = 1.2, width= 1.1, resolution =1.15 and θ= 1

3.3.2.5 EfficientNet series

EfficientNet was made to give better results in reasonable parameters. The EfficientNet model has a series starting from a baseline model B0 which is scaled to achieve up to B7. These models are not human designed, but they are made by NAS (neural architecture search). EfficientNet has come up with version 2 as well which proved to have better accuracy and less computational time. It also has B0-B7 series additionally consists of small, medium, and large models which are made by adding layers in stage 5 and **6** as shown below. This architecture proves to be superior to many state of art models. More details can be found in the paper [48].

3.3.2.5.1 EfficientNet B0 and B1

EfficientNet B0 and B1 are used in this research as the aim of the work is to propose hardware efficient architecture hence taking the benefits of lower series of the architecture.

The common part in whole EfficientNet series is the stem and final layer. The details of the EfficientNet Architecture are given below [49].

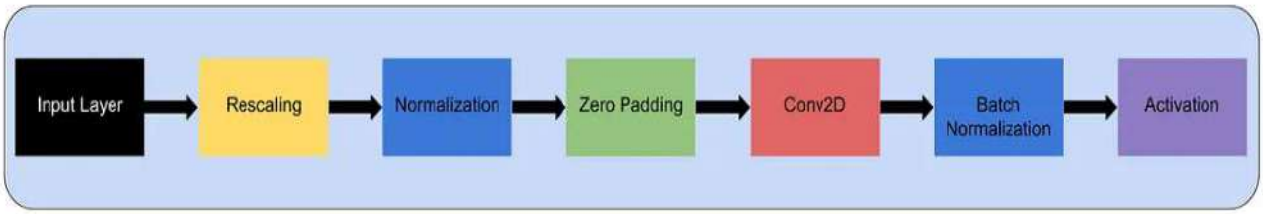


Figure 3.19: Stem Layer

Figure 3.20: Final Layer

Each model in the series has 7 blocks with varying sub-blocks as we move from B0 to B7. The total number of layers in B0 are 310 and B1 are 439 but they can only be made by reusing only 5 modules as given below**:**

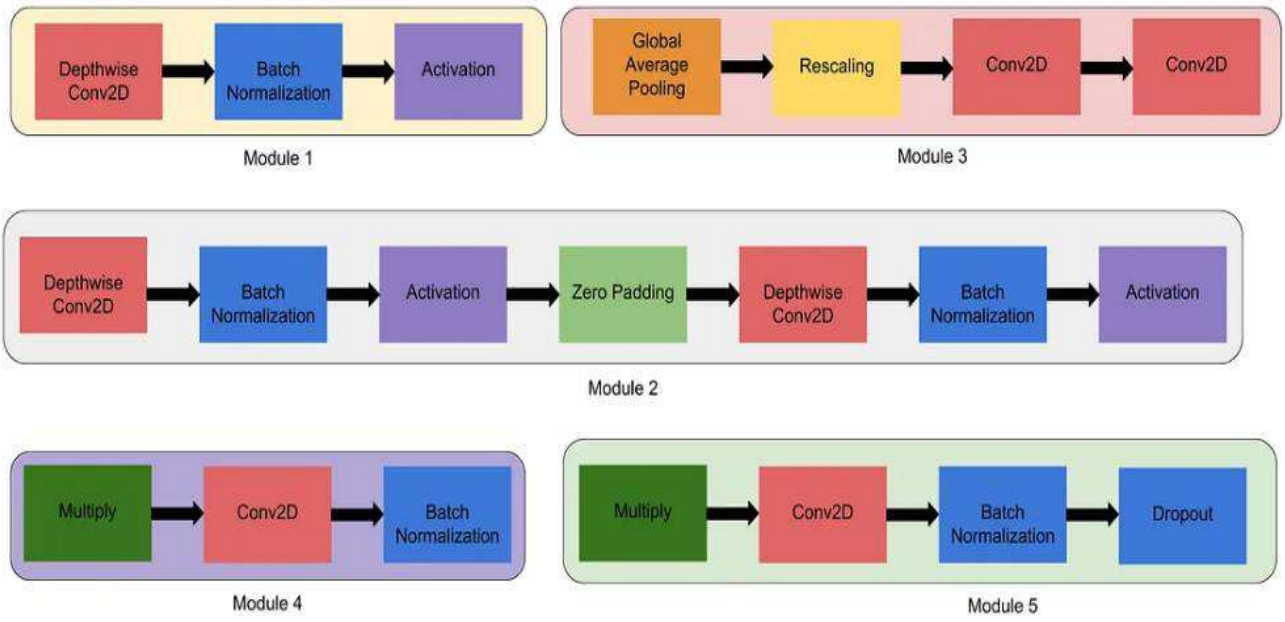


Figure 3.21: Modules in EfficientNet Architecture

- Module 1 is a starting point for the sub-blocks.
- Module 2 is a starting point for the first sub-block of all the 7 main blocks except the 1st one.
- Module 3 is connected as a skipping connection to all the sub-blocks.
- Module 4 is for combining the skipping connection in the first sub-blocks.
- Module 5 for each sub-block is connected to its previous sub-block in a skipping connection and they are combined using this module.

These modules together make sub blocks which are given as**:**

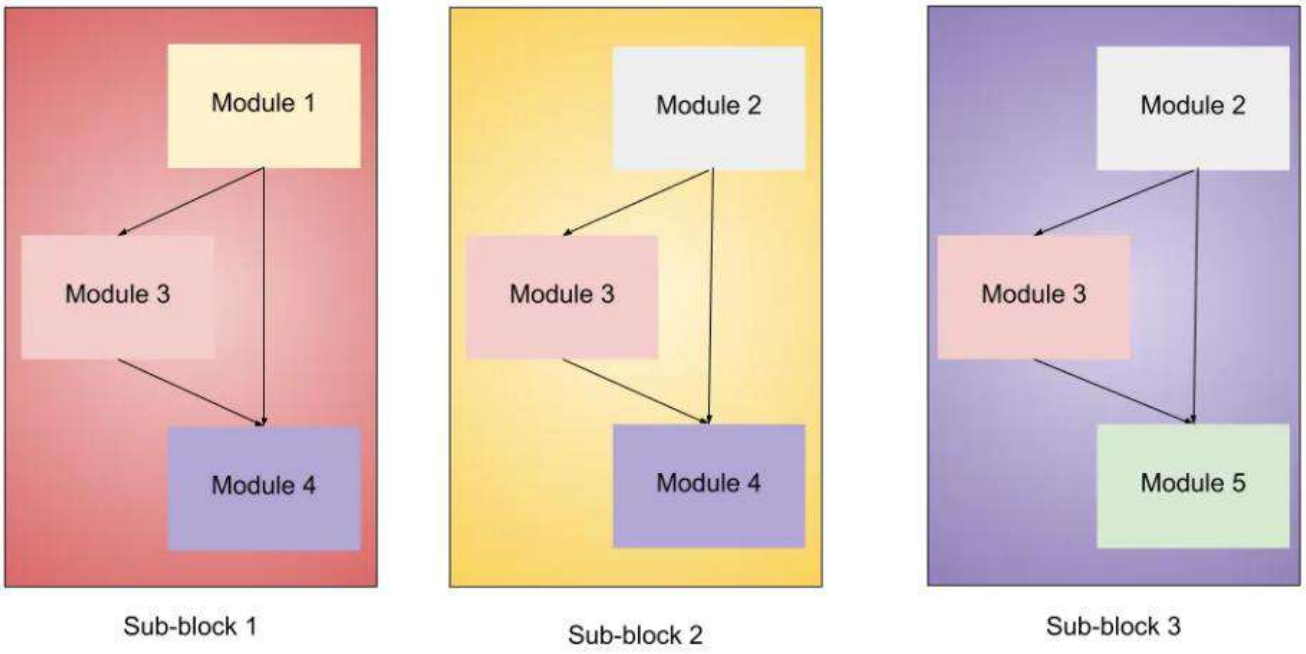


Figure 3.22: Sub-blocks in EfficientNet Architecture

The MBConv block and Squeeze and Excite (SE) block is given as**:**

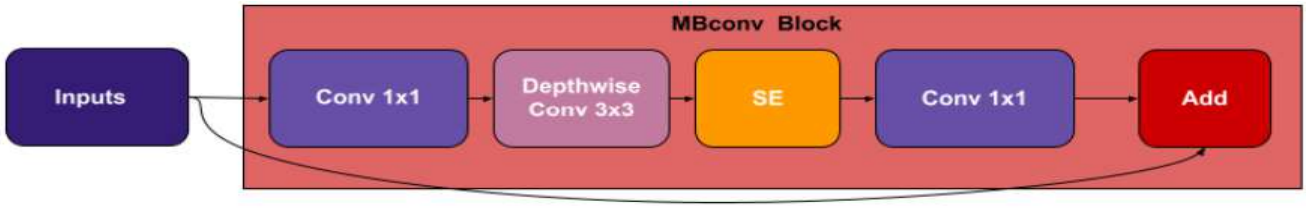


Figure 3.23: MBConv Architecture

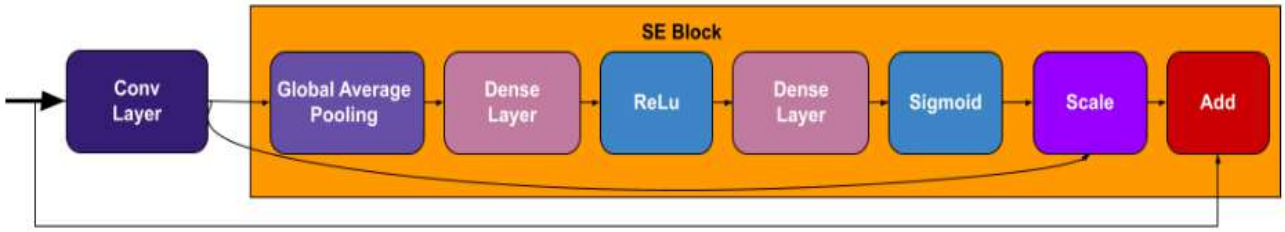


Figure 3.24: Squeeze and Excite Architecture

Below are the details for EfficientNet B0 and B1 architectures.

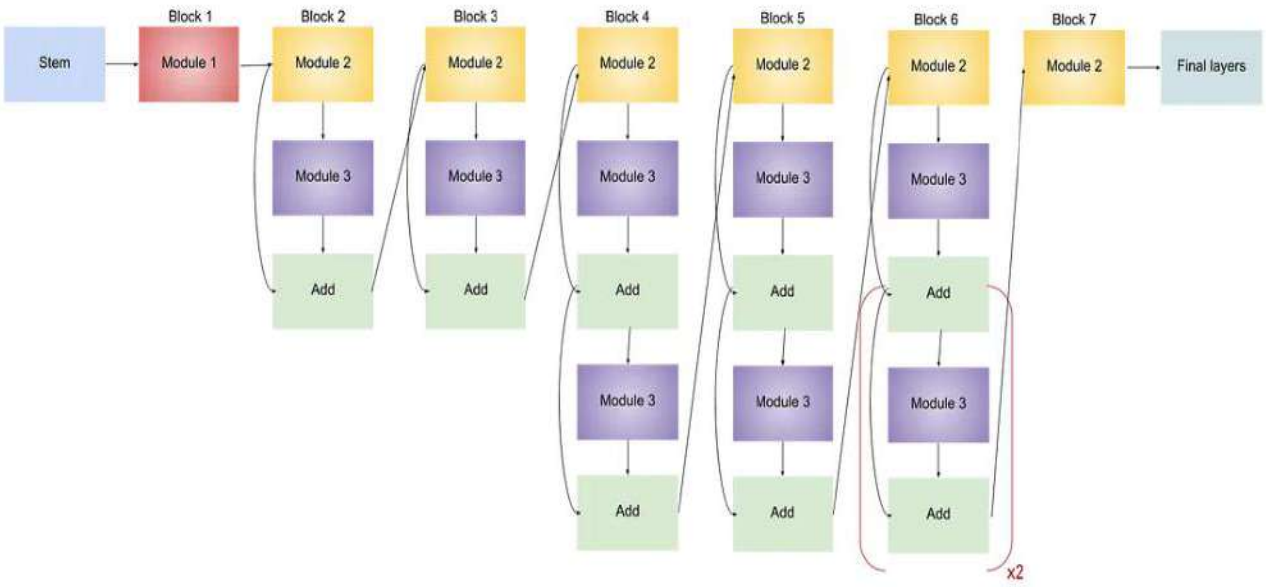


Figure 3.25: EfficientNet B0 Architecture

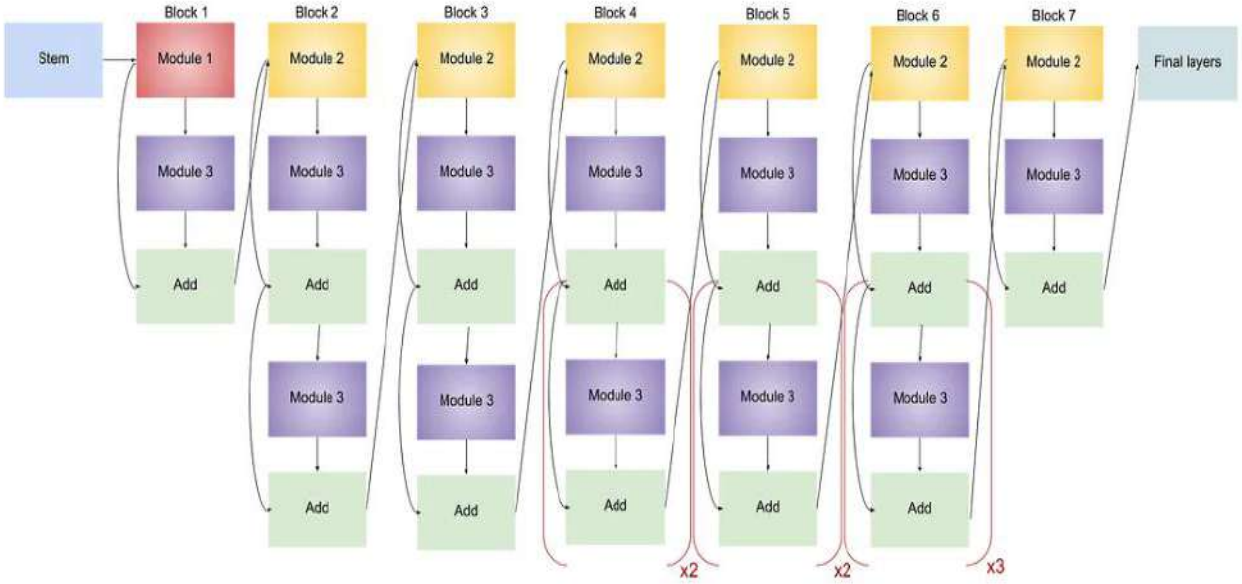


Figure 3.26: EfficientNet B1 Architecture

Drawback of EfficientNet-V1

- ❑ Training with large image size is slow.
- ❑ Depthwise convolution is expensive.
- ❑ Equally scaling up creates doubling in all stages.

Benefits of EfficientNet V2

- ❑ Less training time.
- ❑ More accuracy
- ❑ Depthwise Convolutions are slow in early stages but are effective later.
- ❑ To compensate for the loss receptive field, depthwise conv3×3 and expansion conv1×1 exchanged with single traditional conv3×3 known as FusedMBConv.

Table 3.4: EfficientNet baseline model version1

Stage *i*	Operator	Resolution	# Channels	# Layers
1	Conv3x3	224x224	32	1
2	MBConv1, k3x3	112x112	16	1
3	MBConv6, k3x3	112x112	24	2
4	MBConv6, k5x5	56x56	40	2
5	MBConv6, k3x3	28x28	80	3
6	MBConv6, k5x5	14x14	112	3
7	MBConv6, k5x5	14x14	192	4
8	MBConv6, k3x3	7x7	320	1
9	Conv1x1, Pooling, FC	7x7	1280	1

Table 3.5: EfficientNet Resolution

Model Series	Resolution
B0	224
B1	240
B2	260
B3	300
B4	380
B5	456
B6	528
B7	600

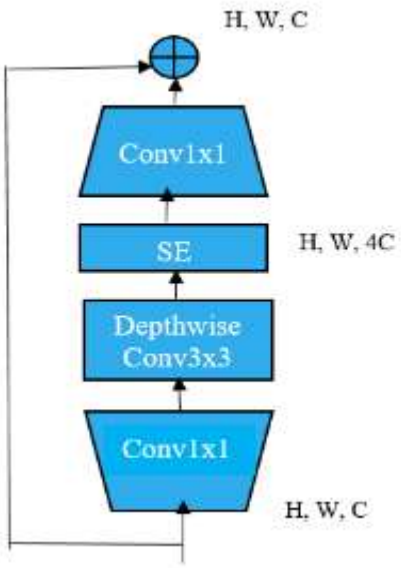


Figure 3.27: MBConv Architecture

Table 3.6: EfficientNet model version2-small

Stage *i*	Operator	Stride	# Channels	# Layers
0	Conv3x3	2	24	1
1	FusedMBConv1, k3x3	1	24	2
2	FusedMBConv4, k3x3	2	48	4
3	FusedMBConv4, k3x3	2	64	4
4	MBConv4, k3x3, SE 0.25	2	128	6
5	MBConv6, k3x3, SE 0.25	1	160	9
6	MBConv6, k3x3, SE 0.25	2	256	15
7	Conv1x1, Pooling, FC	-	1280	1

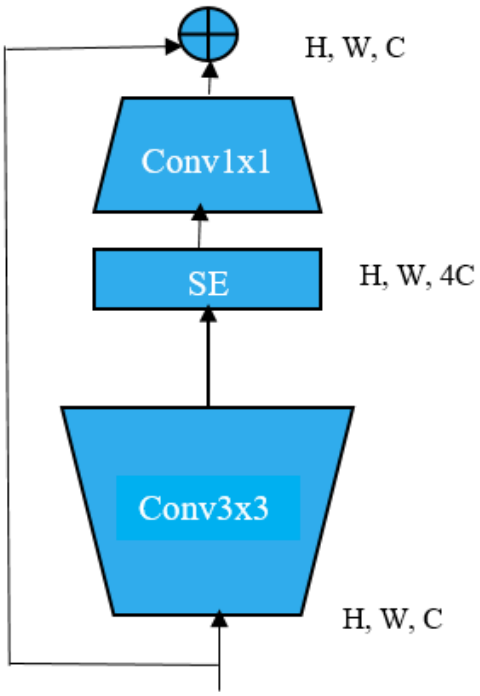


Figure 3.28: FusedMBConv Architecture

3.3.2.5.2 Details of the prominent layers

Sigmoid Linear Unit (SiLU)/Swish function

The network uses swish activation function which captures a wider range of values and gradients. It prevents vanishing gradient problems as ReLU nullifies the negative values and only allows positive values, but SiLU allows both positive and negative values providing wide range of values.

It is a product of linear and sigmoid function given as**:**

$$\boldsymbol{SiLU}(\boldsymbol{x}) = \boldsymbol{x} * \boldsymbol{sigmoid}(\boldsymbol{x})$$

Below is the MATLAB plot that shows how SiLU looks like

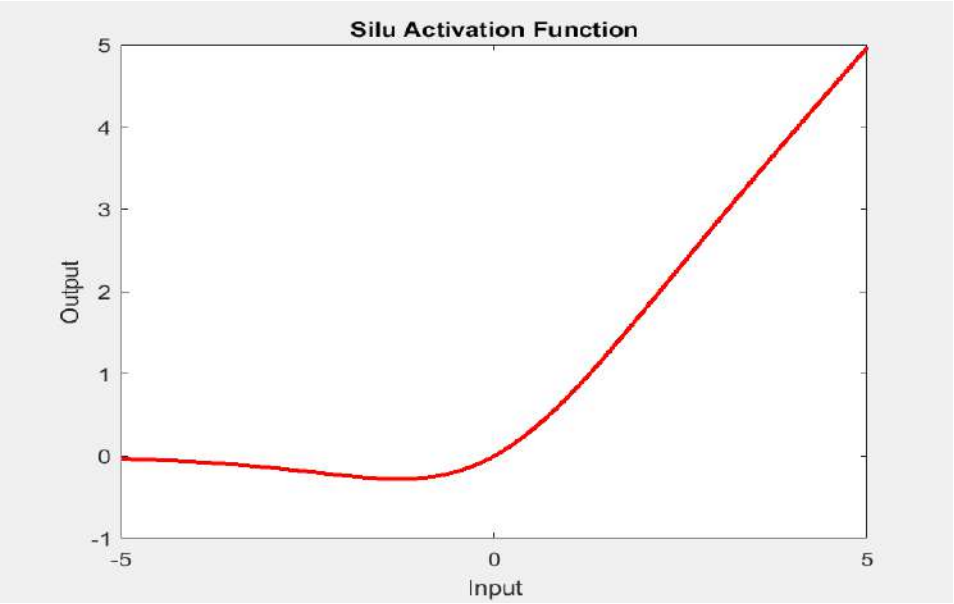


Figure 3.29: SiLU Activation Function

Identity

ResNet18 uses the skipping connections concept which takes some of the previous layer's output and passes it down to overcome vanishing gradient. Just like ResNet, EfficientNet uses identity to achieve it. It improves gradient flow and makes a direct flow from previous layers to the next layers.

Depthwise Separable Convolution

Instead of traditional convolution which perform channel wise and spatial wise convolution in one go. DSC is introduced which reduces the multiplications by incorporating depthwise and pointwise Convolution separately. It performs depthwise convolution first and then pointwise convolution which decreases the trainable parameters by a large number. It applies one filter per channel whereas pointwise convolution creates a linear combination of its output.

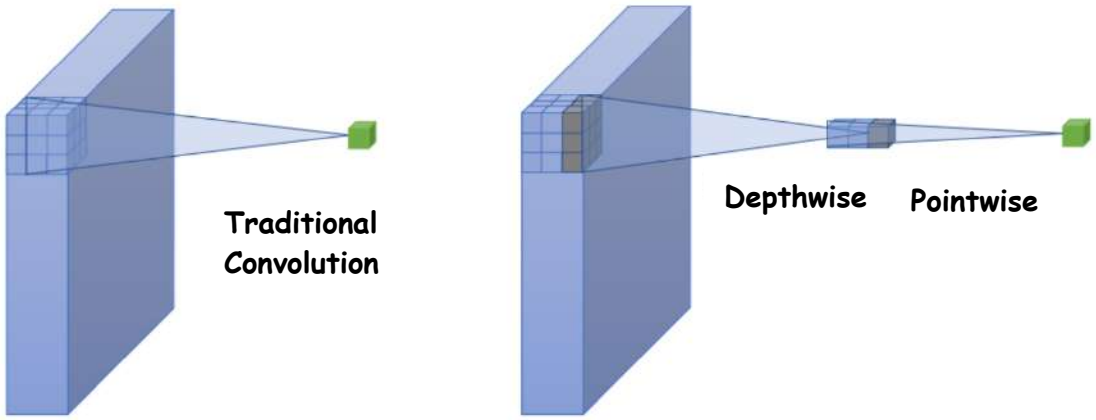


Figure 3.30: Regular Convolution Depthwise and Pointwise Convolution [50]

Squeeze and excite used by inverted residual block (MBConv)

It allows us to emphasize important features and suppress subordinate ones. Instead of assigning weights to the channels equally, it will dynamically assign the high weight for the foremost channels.

Zero Padding

To maintain spatial size of the feature map, padding is done by adding zeros to the border so original information is preserved and to extract fine details without losing information.

Batch Normalization

As the name says this helps to normalize the previous layer output so the training is stable.

Multiply

It is used to rescale channels in Squeeze and Excite block. Its role is to emphasize or de-emphasize certain features based on importance and relevance.

Edge residual

Edge residual is a type of inverted residual block that adds additional edge features to the input feature map before depthwise separable convolution. These edges features will capture the edge details which are useful for building efficient neural networks that can run on resource-constrained devices while still maintaining high accuracy.

To find the output size of layers one can use this formula**:**

$$\boldsymbol{Output\_size} = (\boldsymbol{input\_size} - \boldsymbol{kernel\_size} + \mathbf{2} * \boldsymbol{padding}) / \boldsymbol{stride} + \mathbf{1}$$

3.3.2.6 MobileNet Architectures

MobileNet is a convolutional neural network architecture that utilizes depth wise separable convolutions to build efficient and lightweight models for mobile and embedded vision applications. The key feature of MobileNet is its use of depth wise separable filters, as illustrated in **1. This approach enables the network to reduce the number of parameters and computations while maintaining high accuracy.**

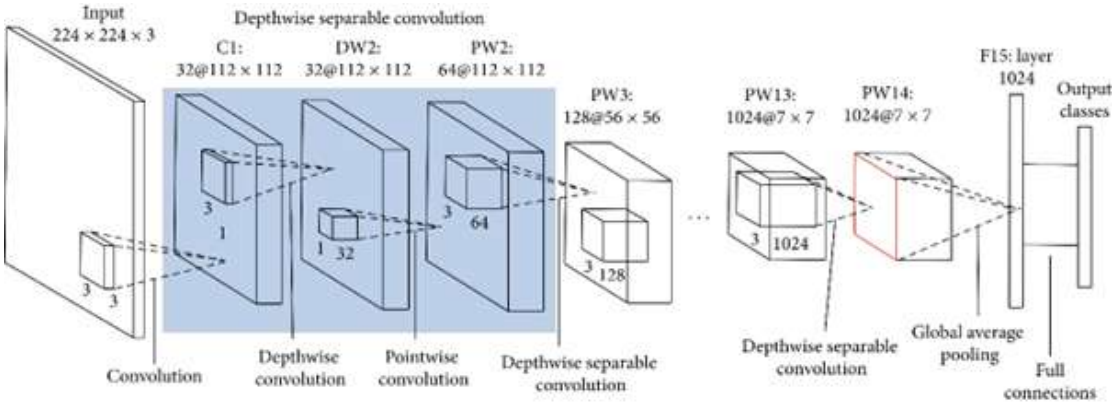


Figure 3.31: Architecture of MobileNet [51]

3.3.2.7 MobileNet Versions

MobileNet is a family of models that have been designed to achieve high accuracy with fewer parameters, making them suitable for mobile and embedded devices. The original MobileNet model, known as MobileNetV1, uses depth-wise separable convolutions to achieve a good trade-off between accuracy and efficiency. The MobileNet architecture has been further improved in subsequent versions, with MobileNetV2 adding linear bottlenecks and inverted residuals to the depth-wise separable convolutions which can accelerate convergence and prevent degradation then in MobileNetV3 introducing h-swish and h-sigmoid activation functions and improved architecture search techniques. The MobileNet models come in various sizes, ranging from small (e.g., MobileNet V1 0.25) to large (e.g., MobileNetV3 Large), allowing users to choose the right model for their specific application based on the trade-off between accuracy and efficiency.

3.3.2.7.1 MobileNetV1, V2 and V3

In the present research, MobileNetV1, V2 and V3 architectures were employed due to their potential to provide hardware-efficient solutions. As the primary goal was to develop a resource-efficient model, leveraging the advantages of these three versions was deemed appropriate. MobileNetV1, V2 and V3 are a series of convolution neural network architectures that are designed for efficient mobile and embedded vision applications. Due to their streamlined structure and depth-wise separable convolution filters they are used in these types of applications. Therefore, incorporating these architectures in the study is expected to yield optimal results while minimizing the computational burden.

3.3.2.8 MobileNetV1:

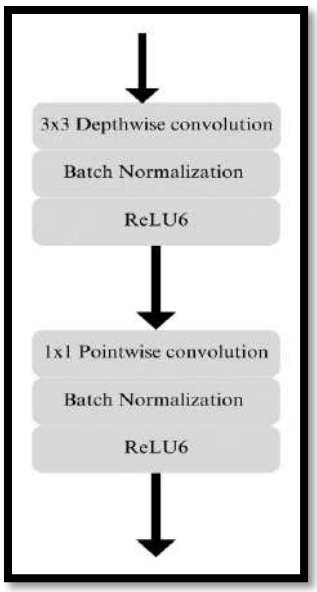


Figure 3.32: Depth wise Separable Convolution block

3.3.2.8.1 Details of the prominent layers

Depth-wise convolutional layer

This layer applies a separate 3x3 convolution filter to each of the channels of the input image, resulting in a set of output channels that is equal to the number of input channels. The same filter was used across all input channels, so that the network learns to extract the same type of features across all channels. This operation helps to capture spatial dependencies within the image that are specific to each channel, the depth-wise convolutional layer can learn channel-specific features that are optimized for the spatial location of that channel in the input image.

Point-wise convolutional layer

This layer applies a 1x1 convolutional filter to the output of the depth-wise convolutional layer, the point-wise convolution performs a linear combination of the input channels, allowing the network to learn a weighted sum of the feature maps produced by the previous depth-wise convolutional layer. The output we get has reduced the number of channels compared to the input, which helps to reduce the computational cost of the subsequent layer.

This point-wise convolutional layer helps to reduce the computational cost of the network but also increases the representation power of the network enables to learn non-linear interactions between the input channels. This is because the weights of the 1x1 convolutional filter are learned during the training process to allow to learn more complex features interactions that may not be captured by the depth-wise convolutional layer alone. MobileNetV1 uses both batch normalization and ReLU non-linearity for both layers.

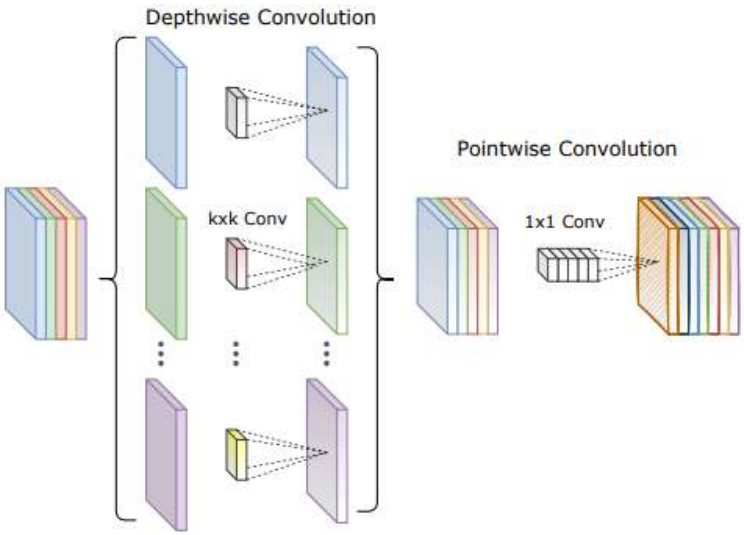


Figure 3.33: Depthwise Separable Convolution [52]

Down-sampling layer

A down-sampling layer is a type of layer in a convolutional neural network that reduces the spatial dimensions of feature maps. The down-sampling layers are implemented using a depth-wise separable convolution with a stride of 2 in the depth-wise convolutional operation this allows mobilenetv1 architecture to reduce the spatial

dimensions of the feature maps while maintaining a high level of accuracy on image classification tasks.

Fully Connected Layer

The purpose of the fully connected layers in a convolutional neural network (CNN) is to perform the final classification of the input image. These layers take the output of the last convolutional layer or the last pooling layer as input and produce a vector of class probabilities as output.

In MobileNetv1, the final layers of the network consist of a global average pooling layer followed by a fully connected layer with a SoftMax activation function, which produces the final classification output.

Activation Function

- ReLU**6** (activation Function)**:**
- Real world data is non-linear.
- Computationally fast
- Zero if it is negative and **6** if it is positive.
- ReLU**6** is used due to its robustness when used with low-precision computation based on MobileNetV1.

The ReLU**6** activation function can be represented mathematically as**:**

$$\mathbf{ReLU6(x) = min\ (max\ (x,\ 0),\ 6)}$$

.

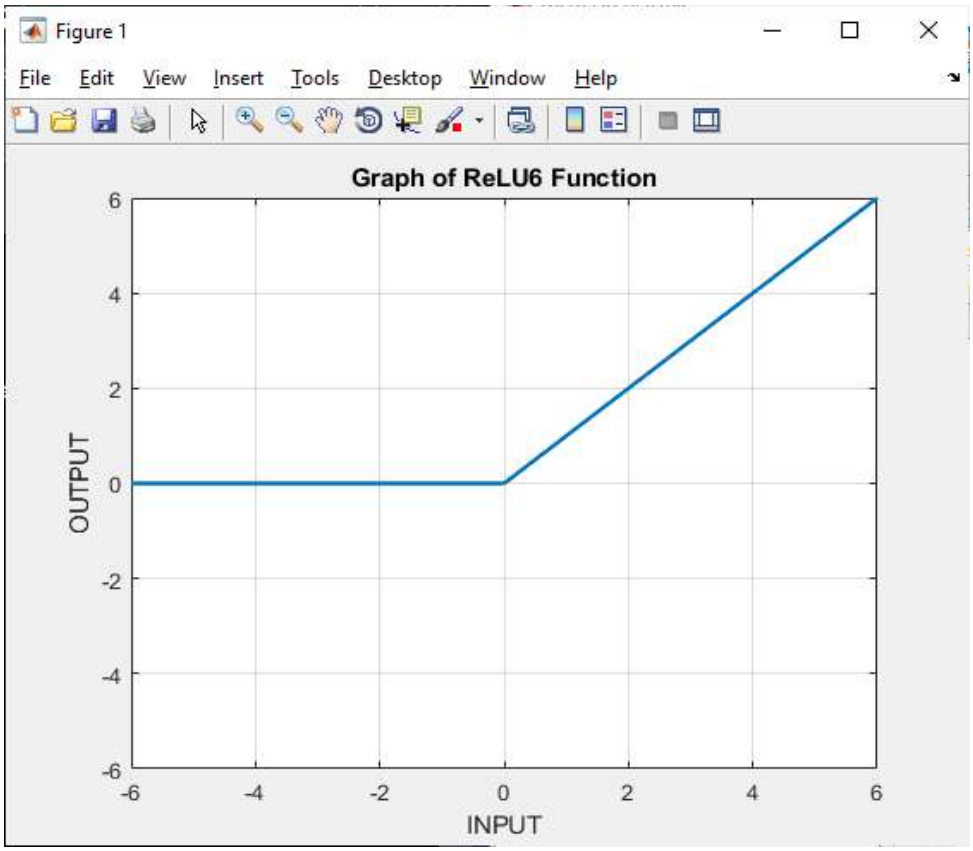


Figure 3.34: ReLU6 Activation Function

Batch Normalization

- Used as regularization technique.
- Performance and stability of the model
- Higher learning rates

1x1 convolution

- Pointwise convolution.
- Reduces the number of channels and the computational cost.
- Non-linearity in the network.
- Reduces the number of parameters.

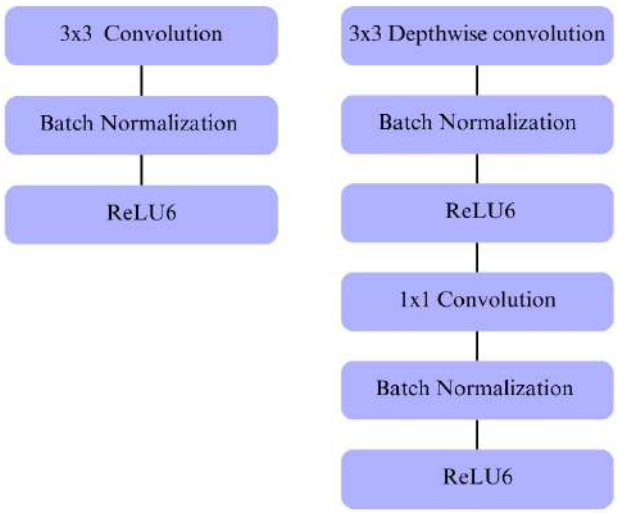


Figure 3.35: standard convolution followed by normalization and RELU (left). Depth-wise convolution layer and pointwise convolution layer, each followed by batch normalization and RELU (Right)

Table 3.7: MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw [1]/ s1	$3 \times 3 \times 32\ dw$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times \mathbf{6}4$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times \mathbf{6}4\ dw$	$112 \times 112 \times \mathbf{6}4$
Conv / s1	$1 \times 1 \times \mathbf{6}4 \times 128$	$\mathbf{5}6 \times \mathbf{5}6 \times \mathbf{6}4$
Conv dw / s1	$3 \times 3 \times 128\ dw$	$\mathbf{5}6 \times \mathbf{5}6 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$\mathbf{5}6 \times \mathbf{5}6 \times 128$
Conv dw / s2	$3 \times 3 \times 128\ dw$	$\mathbf{5}6 \times \mathbf{5}6 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 25\mathbf{6}$	$28 \times 28 \times 128$

[1] dw**:** Depth wise
[2] Avg Pool**:** Average pooling

Conv dw / s1	$3 \times 3 \times 256\ dw$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256\ dw$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5 × Conv dw / s1 conv / s1	$3 \times 3 \times 512\ dw$ $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512\ dw$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024\ dw$	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 102 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

3.3.2.9 Proposed Novelty

There are two new hyper-parameters introduced in the MobileNetV1**:**

1) Width Multiplier
2) Resolution Multiplier

3.3.2.9.1 Width multiplier

Width multiplier is introduced to control the number of channels or controls the overall width of the network which is determined by the number of filters in each layer. It is a global Hyperparameters that is used to construct smaller and less computationally expensive models. Its value lies between 0 and 1. A smaller width multiplier will reduce the number of filters in the network and make it more computationally efficient, while a larger width multiplier will increase the number of filters and improve the accuracy of the network.

$$\mathbf{D_k} \cdot \mathbf{D_K} \cdot \boldsymbol{\alpha}\mathbf{M} \cdot \mathbf{D_F} \cdot \mathbf{D_F} + \boldsymbol{\alpha}\mathbf{M} \cdot \boldsymbol{\alpha}\mathbf{N} \cdot \mathbf{D_F} \cdot \mathbf{D_F}$$

3.3.2.9.2 Resolution multiplier

This Second hyper-parameter is used to decrease the computational cost of a neural network is a resolution multiplier this hyper-parameter reduces the resolution of the input image and this subsequently reduces the input to every layer by the same factor if we have a smaller resolution multiplier it reduces the size of the input images and make the network more computationally efficient, while a larger resolution multiplier make the input images larger in size potentially improve the network's accuracy.

The resolution and width multipliers in MobileNetV1 allow for the creation of lightweight neural networks that can achieve high accuracy on tasks such as image classification while using fewer computational resources.

$$\mathbf{D_k} \cdot \mathbf{D_K} \cdot \boldsymbol{\alpha}\mathbf{M} \cdot \boldsymbol{\rho}\mathbf{D_F} \cdot \boldsymbol{\rho}\mathbf{D_F} + \boldsymbol{\alpha}\mathbf{M} \cdot \boldsymbol{\alpha}\mathbf{N} \cdot \boldsymbol{\rho}\mathbf{D_F} \cdot \boldsymbol{\rho}\mathbf{D_F}$$

3.3.2.10 MobileNetV2

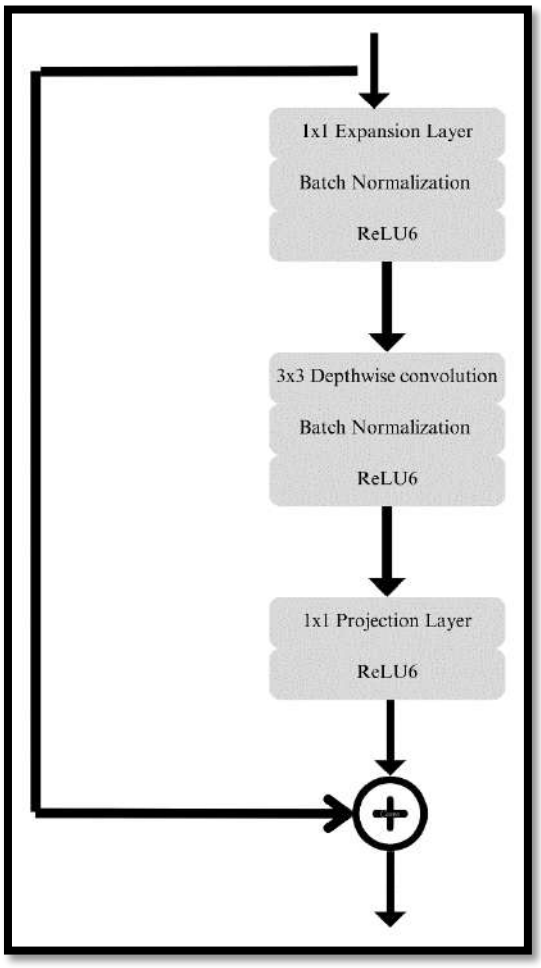


Figure 3.36: Linear Bottleneck and inverse Residual Block

- The bottleneck residual block has three convolution layers.
- The last two layers in MobileNetV1 are depth-wise convolution and 1 x 1 point-to-point convolution layers.
- In MobileNetV1, the pointwise convolution either keeps the number of channels the same or doubles them, while in the bottleneck residual block, the 1 x 1 convolution layer reduces the number of channels.

- The first layer in the bottleneck residual block is the 1 x 1 expansion layer, which expands the data by increasing the number of channels.
- The second layer in the bottleneck residual block is the depth-wise convolution layer, which we already know from MobileNetv1.
- The bottleneck residual block includes a residual connection, which works the same way as in ResNet.
- ReLU**6** is used as the activation function in each layer of the bottleneck residual block except the projection layer.
- The projection layer only has a batch normalization layer because introducing nonlinearity with ReLU**6** will decrease the performance as the output from the projection layer is of low dimension.

The motivation for inserting shortcuts in like that of the classical residual connections we want to improve the ability of a gradient to propagate across multiplier layers.

The basic building block is a bottleneck depth-separable convolution with residuals. The detailed structure of this block was shown in the table below.

Input	**Operator**	**Output**
$\boldsymbol{h \times w \times k}$	1×1 conv2d, ReLU**6**	$h \times w \times (tk)$
$\boldsymbol{h \times w \times k}$	3×3 depth-wise s=s	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\boldsymbol{\frac{h}{s} \times \frac{w}{s} \times tk}$	Linear 1×1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Figure 3.36: Bottleneck Architecture

The architecture of MobileNetV2 contains the initially fully convolution layer with 32 filters, which was then followed by 19 residual bottleneck layers described in the table below.
We have used the ReLU**6** as non-linearity because of its robustness when we used with low precision computation as a common practice in modern networks, we employ a kernel size of 3 × 3, which is a standard choice. Additionally, we incorporate dropout and batch normalization techniques during the training process.

Table 3.8: MobileNetV2 Body Architecture

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2D	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2D 1 × 1	-	1280	1	1
$7^2 \times 1280$	avgpool 7 × 7	-	-	1	-
$1 \times 1 \times 1280$	conv2D 1 × 1	-	k	-	-

Drawback of MobileNet-V1

- Limited accuracy compared to larger models.

Benefits of MobileNet-V2

- Improved performance with higher accuracy.
- Better generalization capabilities.
- Flexibility and customizability in architecture and Hyperparameters.
- Multi-scale feature extraction for tasks like object detection and segmentation.

3.3.2.11 MobileNetV3

MobileNetV3 is a family of lightweight neural network architectures designed for efficient and high-performance deep learning on resource-constrained devices. MobileNetV3 aims to strike a balance between model size, computational efficiency, and accuracy. The "Small" variant of MobileNetV3 is specifically designed to be even more lightweight, making it suitable for mobile and embedded applications. It achieves this by leveraging efficient depth-wise separable convolutions, squeeze-and-excitation modules, and improved architecture design. Despite its compact size, MobileNetV3-Small demonstrates impressive performance on various computer vision tasks such as image classification and object detection. It offers a practical solution for deploying deep learning models on devices with limited computational resources, enabling a wide range of applications in real-world scenarios.

While MobileNetV3-Large is a high-performance and efficient neural network architecture. It achieves a balance between accuracy and computational efficiency. It incorporates advanced features like inverted residual blocks and attention mechanisms. MobileNetV3 Large can achieve state-of-the-art performance on various computer vision tasks.
This architecture was implemented as a function of different resolutions and multipliers.so in given below figure we can see that the MobileNetV3-Small outperforms the MobileNetV3-Large with multiplier scaled to match the performance.

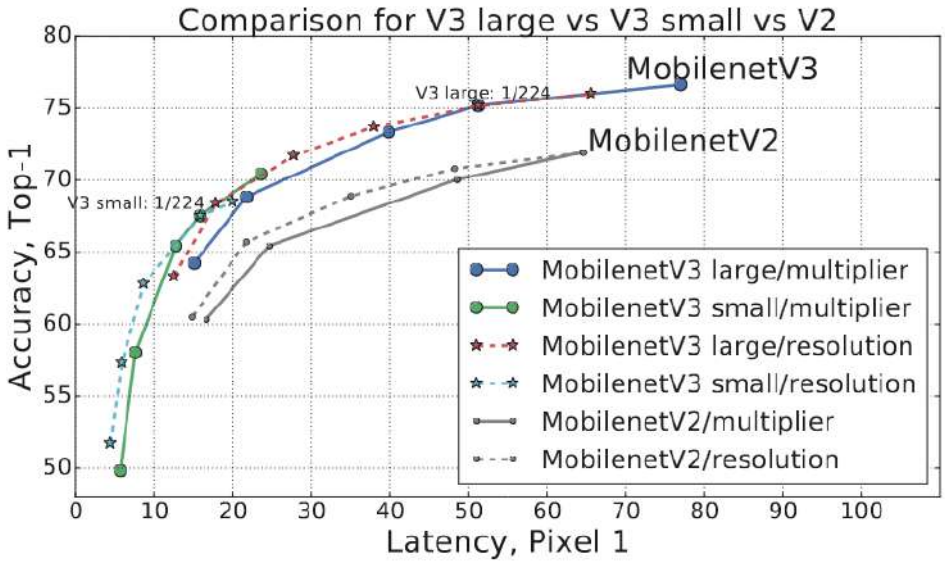


Figure 3.37: Comparison of V3 large vs V3 small vs V2 [53]

MobileNetV3 is defined as two models**:** MobileNetV3-Large and MobileNetV3-Small. These models are targeted at high and low resource use cases respectively for more details please refer [53] .

Drawback of MobileNet-V2

- Increased computational complexity compared to MobileNet-V1 due to the addition of new features and techniques.

Benefits of MobileNet-V3

- Further improved accuracy compared to MobileNet-V2.
- Enhanced efficiency and performance with advanced design choices.
- Introduces the concept of network architecture search (NAS) for optimizing model design.
- Offers both "Small" and "Large" variants for different resource constraints and application needs.
- Demonstrates state-of-the-art performance on various computer vision tasks.

3.3.3 ECG Databases

The famous Databases for ECG that are publically available are PhysioNet's MIT BIH Arrhythmia that is most commonly used [54]. PTBD Database is another ECG dataset widely available [55]. UC Irvine Machine Learning repository has Arrhythmia dataset [56]. All these are non-image databases. Since CNNs except images, these files are carefully being converted into images.

Chapter – 4

Methodology

This chapter entails the tools and software used in the study to perform classification, analysis, and other computational tasks.
The proposed methodology used Transfer Learning approach[2] (subsequently referred as method 1 in this report) to implement all the CNNs that are being selected for the work and later modified Transfer Learning[3] (subsequently referred as method 2 in this report) is implemented for EfficientNet. In the other part a comparison is made with previous methods literature. The details are discussed in the below section. Following is the agenda of this work**:**

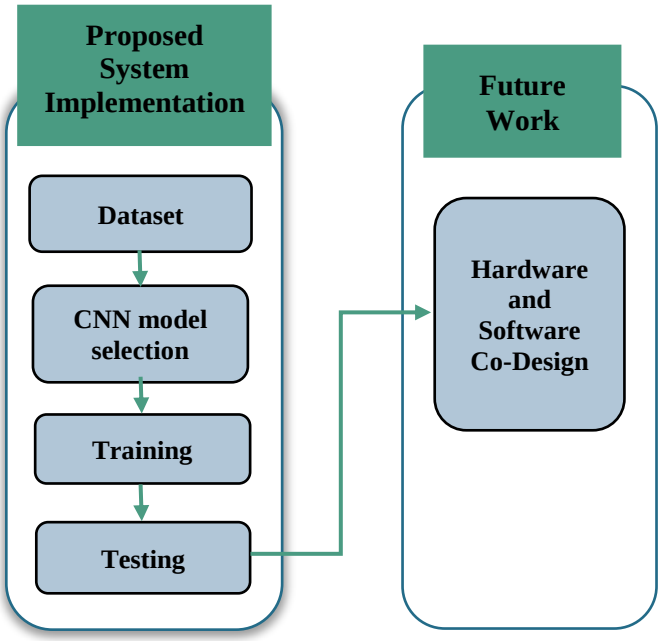


Figure 4.1: Framework of proposed system

4.1 Experimental Setup

Programming Languages: Python, MATLAB
Frameworks: TensorFlow / Keras, PyTorch
Tools/Software and Environment: Google Collaboratory Notebook, MATLAB

[2] Method 1
[3] Method 2

Dataset: MIT BIH Arrhythmia Database and PTBDB

4.2 Implementation

This chapter outlines our proposed method's procedure, which is divided into two independent parts. The first part, known as CNN Selection and Implementation, involves preparing the dataset, training, evaluating, and finalizing the CNN architecture. Later part is comparison with other state-of-art models.

4.2.1 ECG Dataset Preparation:

Various datasets are available for ECG, but the only famous Arrhythmia dataset is from MIT BIH Arrhythmia and PTB Database. For Deep Learning, large dataset is required. As CNN accepts Images so the image version of MIT BIH Arrhythmia Database and PTBDB available on Kaggle is used [57].

The dataset has been used for ECG Classification using Deep Learning Architectures and Transfer Learning. The signal has normal, and cases affected by different Arrhythmias and Myocardial Infarction. Kaggle allows direct Dataset access through its APIs. Since the dataset is from Kaggle, instead of downloading the dataset which takes up space locally, Kaggle API token is used to download dataset for a runtime virtually. You must run commands in the following manner.

Figure 4.2: Downloading dataset from Kaggle API

The dataset has train and test folders each with total 6 Classes 'F','M','N', 'Q', 'S','V'. The F, N, V, Q, S classes are recommended by ANSI/AAMI standards.

Table 4.1: Dataset Specifications

Dataset	
MIT BIT Arrhythmia	**PTB Database**
Samples: 109446	Samples: 14552
Categories: 5	Categories: 2
Classes: {N, V, Q, F, S}	Classes: {N, M}
Sampling Frequency: 125Hz	Sampling Frequency: 125Hz

Table 4.2: Class Labels

Class Labels and Names					
N	**S**	**Q**	**V**	**F**	**M**
Normal	Supraventricular Premature	Unclassifiable	PVC	Fusion of Ventricular and Normal	Myocardial Infarction

Since the training folder has 99199 images and the testing folder has 24799 images. Validation data was created of 90/10 split from the training folder. At this point, aim is not care about less data in validation as testing data is already available separately. The only motive right now is to observe the behavior during training for understanding purpose. As train test split has data leakage possibility, split folder library is used to create a separate folder to avoid data leakage.

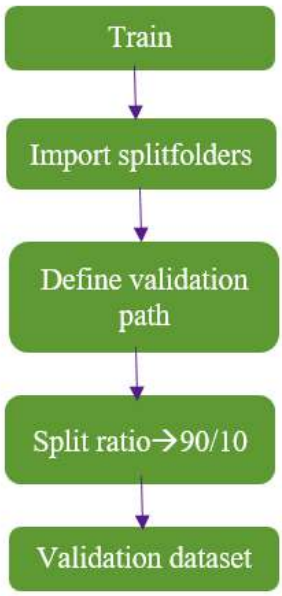


Figure 4.3: Dataset split

4.2.2 Model Selection:

Considering the hardware favorability, memory size, parameters, performance and computational ability, CNN architecture is being chosen. We aim for a lightweight, small, and less complex network that can be deployed on tightly constraint hardware. Based on simplicity and hardware friendliness we chose these networks**:**

- AlexNet
- VGG16
- ResNet18
- MobileNet (V1/V2)
- EfficientNet B0-B1 (V1/V2)

4.2.3 Designing of CNNs:

The design specifications of implemented convolutional neural networks is given below**:**

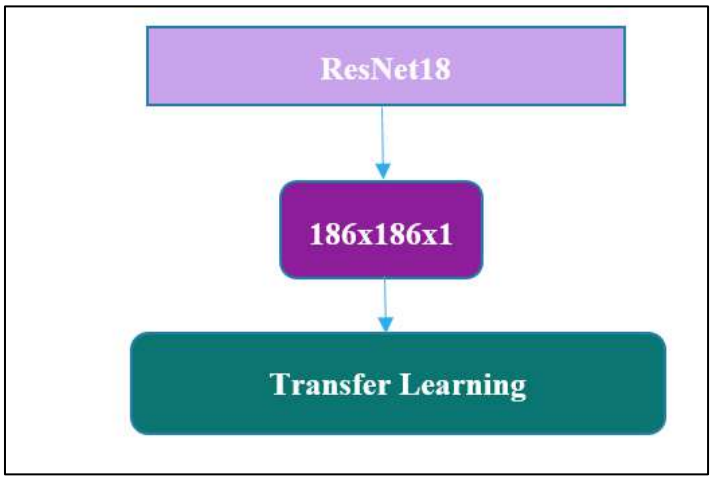


Figure 4.4: Proposed ResNet18 model

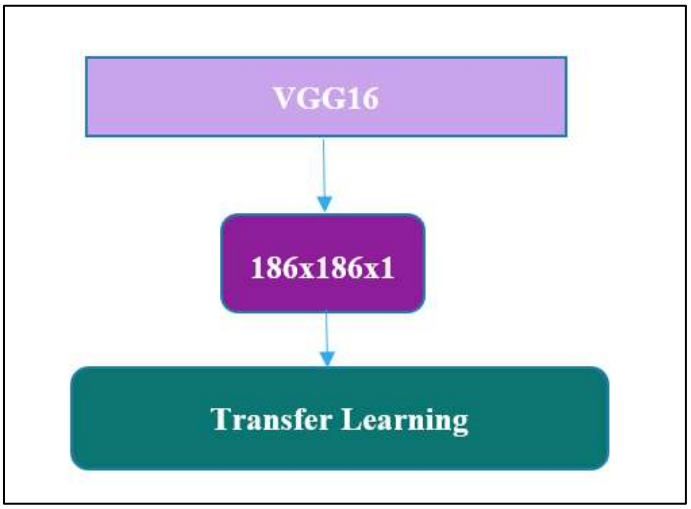


Figure 4.5: Proposed VGG16 model

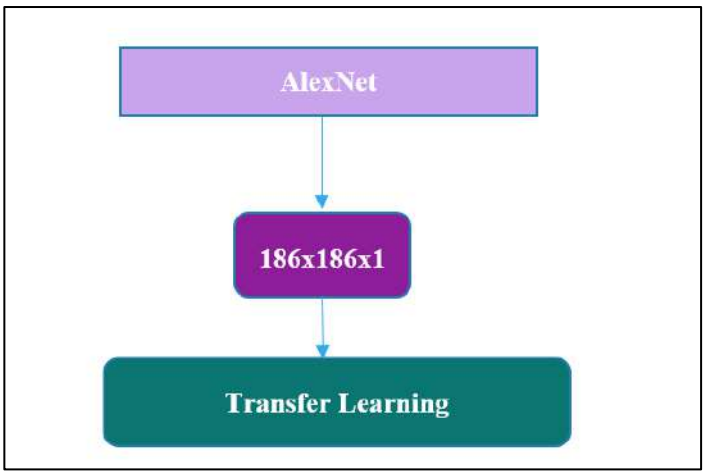


Figure 4.6: Proposed AlexNet model

EfficientNet

As we are already familiar with EfficientNet series B0-B7 discussed previously. B0 being a baseline model uses compound scaling in implementation of higher models. In this research we have implemented EfficientNet version 1 and version 2 for B0 and B1 models. Since we aim to have an efficient network in terms of hardware favorability which compelled us to take benefits of these lower series models. These models were computationally efficient, meaning they could run on hardware with limited resources without compromising the performance. Additionally, the simple architecture allowed for faster training and testing times while still achieving high accuracy.

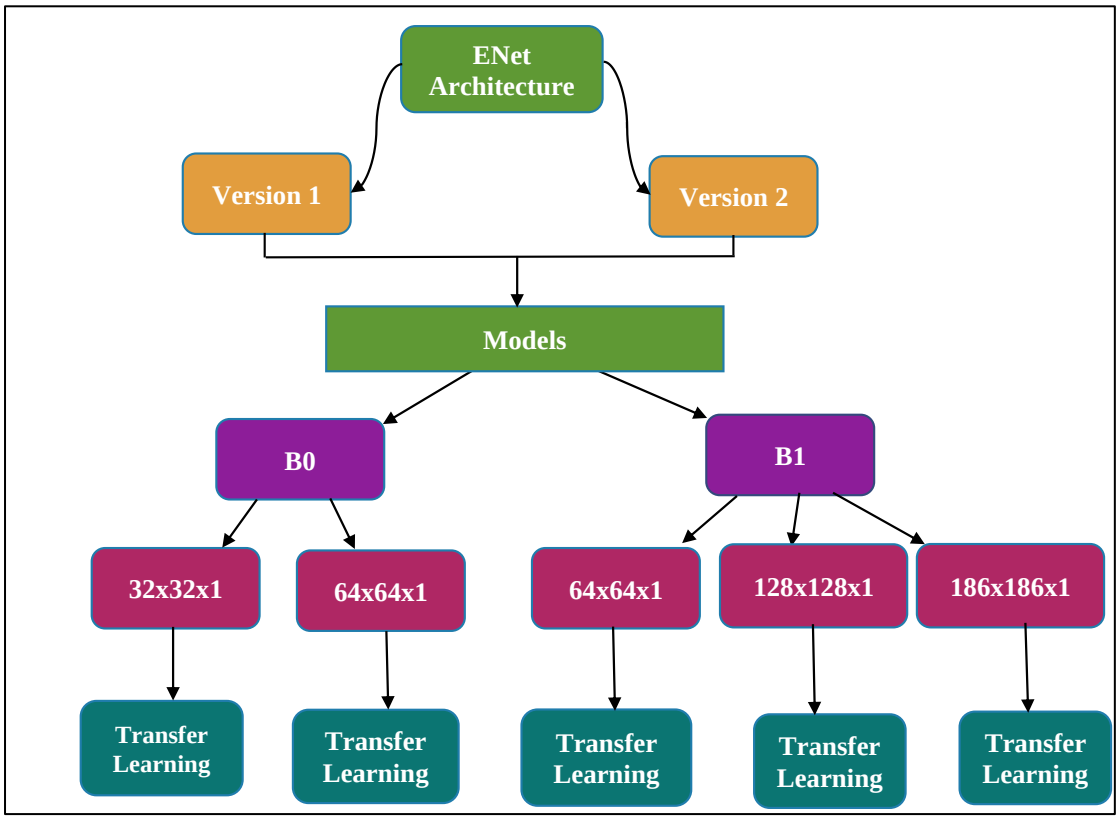


Figure 4.7: Proposed EfficientNet model

These models are implemented using Transfer Learning as a feature extractor approach, pretrained on imagenet weights. The minimum size EfficientNet accepts is 32 so keeping hardware compatibility in mind 32 is taken as a starting size. For B0 the size is 32x32 and we tested on a slightly bigger size like 64x64 also for comparing impact of sizes. As EfficientNet has compound scaling so we arbitrarily decided 64x64 size and we tested on a slightly bigger 128x128 size as well. This work implements EfficientNet with both the methods discussed below. Further a comparison is made which is discussed in the next chapter.

AlexNet, ResNet18, VGG16 are implemented using transfer learning method. Since the accuracy achieved is outstanding there is no need to significantly increase parameters

with only a slight increase in accuracy hence method 2 is not implemented for these models.

MobileNet

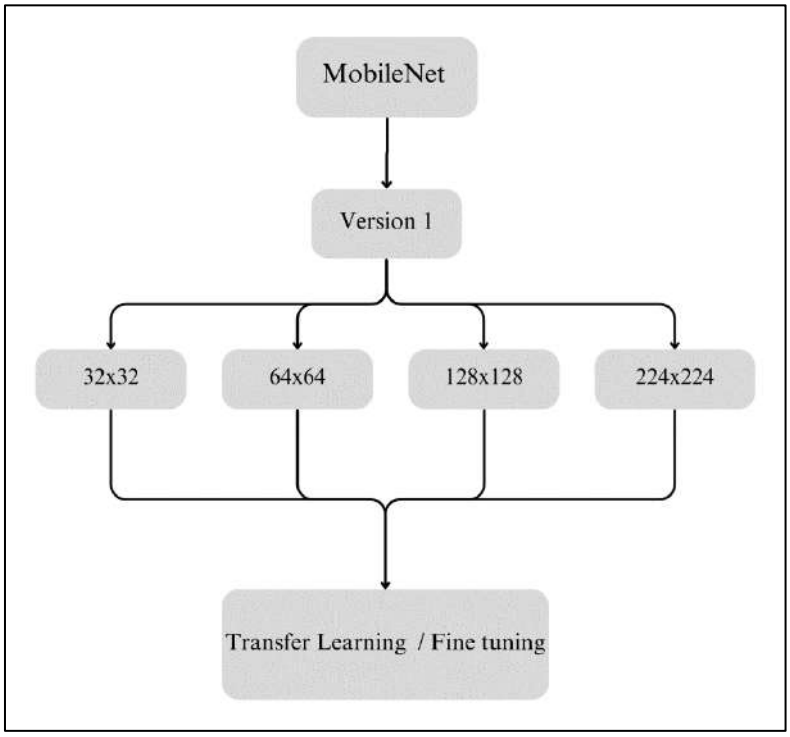


Figure 4.8: Proposed MobileNet Version 1 model

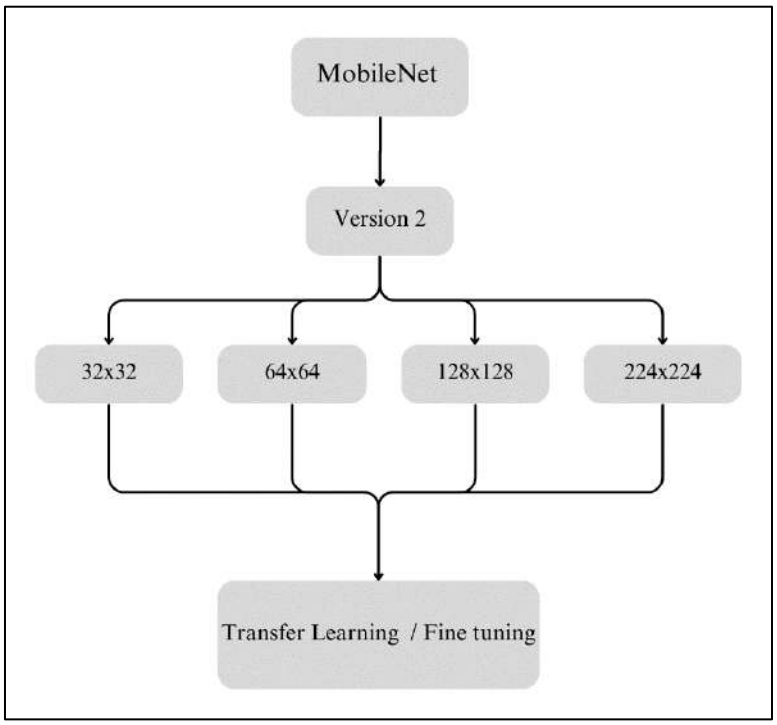


Figure 4.9: Proposed MobileNet Version 2 model

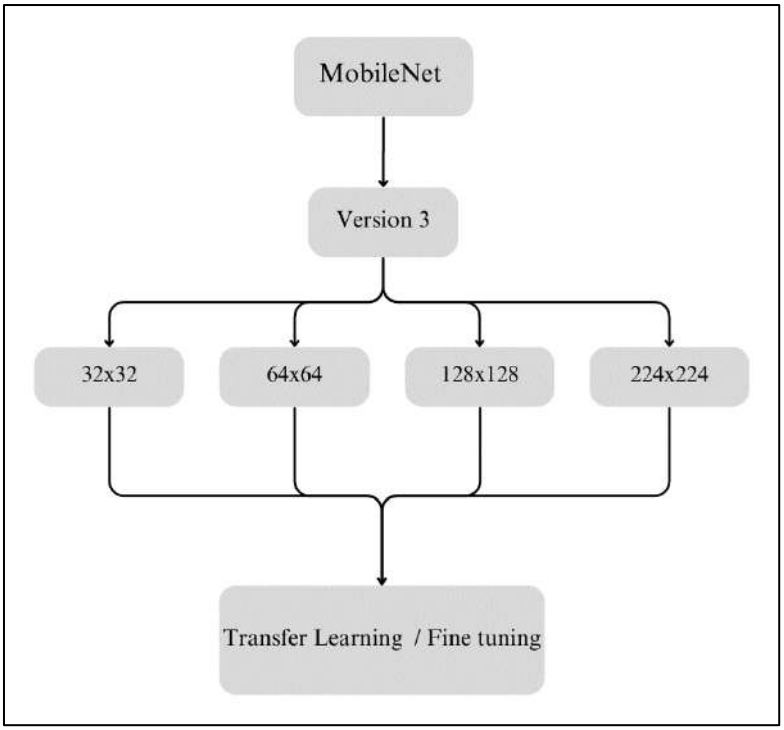


Figure 4.10: Proposed MobileNet Version 3 model

These models are implemented using Transfer Learning as well as fine tuning optimizing some of the Hyperparameters, pretrained on ImageNet weights. The minimum size MobileNet accepts is 32 so keeping hardware compatibility in mind 32 is taken as a starting size. For MobileNetV1 the size is 32x32 and we tested on a slightly bigger size like 64x64, 128x128, and 224x224 also for comparing impact of sizes. Further a comparison is made which is discussed in the next chapter.

4.2.4 Implementation of CNN Architectures

A very famous Pythonic styled PyTorch framework is used to implement EfficientNet, ResNet18, VGG16 and AlexNet.

Importing all the necessary libraries. Since we are running on Google Colab Notebook, we used “tqdm” package for displaying progress bar which will be useful during training and testing for timing analysis. The notebook has some functions created for displaying images, plotting, and calculating accuracy. Since PyTorch uses (channel, height, width) convention whereas matplotlib uses (height, width, channel) convention. So, the 3D tensor is arranged accordingly. One thing worth noting is that matplotlib uses “viridis” color map by default. Since we desire Grayscale images even after converting to grayscale matplotlib displays colored images. To tackle this problem, we pass a parameter cmap='gray' to ensure the display is in grayscale as well.

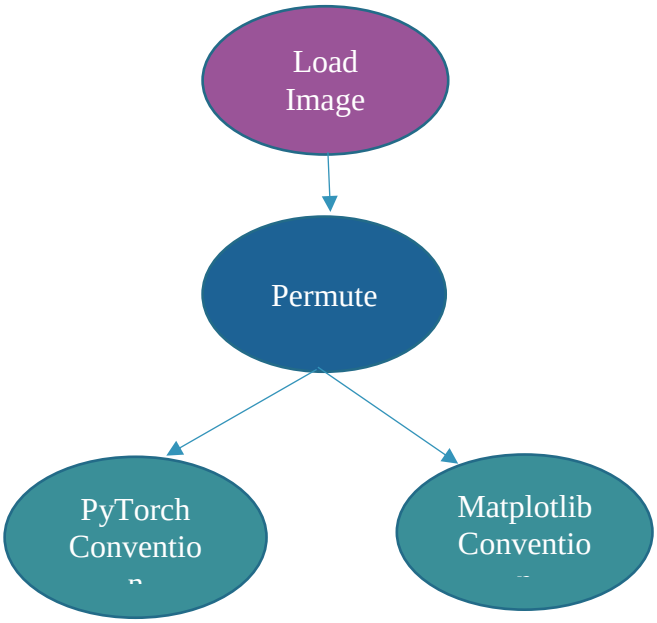


Figure 4.11: 3D-Tensor Rearrangement

4.2.5 Preprocessing

Next step is to apply transformation on the dataset. It is an important step of preprocessing before passing towards neural network. There are plenty of options available in transforms method. Our point of interest is conversion of RGB images to single channel grayscale, resizing the images to desired size and convert images in the range [0, 255] to a float tensor of shape (C, H, W) in the range [0.0, 1.0]. The reason for doing so is to normalize the pixel values to a common scale. Besides many activation functions like sigmoid and ReLU are in the range 0 to 1. Another reason is to reduce the memory requirements since many deep learning models have 32-bit floating point precision by default. All the transforms are being applied using compose method in which all the transforms work at a time whereas there is another way known as sequential method in which transforms are being applied sequentially. The transforms are applied on the train, test, and validation folders.

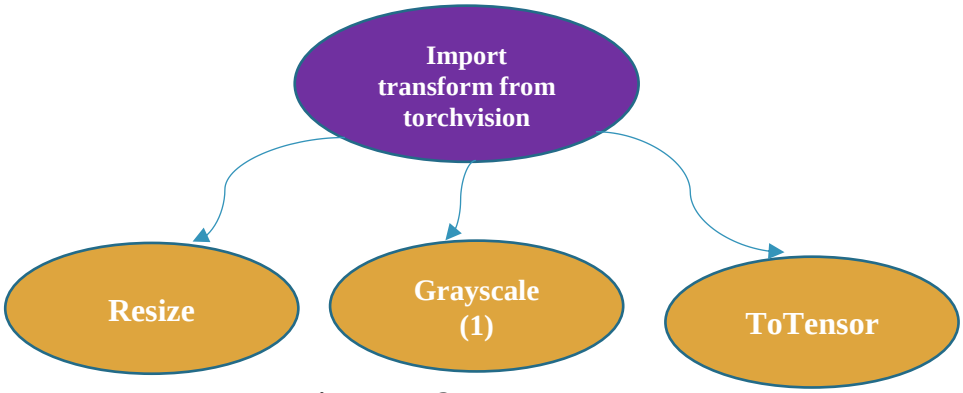


Figure 4.12: Pre-processing

4.2.5.1 Exploratory Analysis

The next step is to do some exploratory analysis to check images in each folder and display some images to ensure the color and size of image has been transformed.
Now we will ready the train, validate, and test loader for which we used DataLoader from torch.utils.data.
The DataLoader class is used to wrap a dataset and provides several useful features, such as**:**

Batch loading: It loads the data in batches of a specified size.
Shuffle: It shuffles the data before each epoch, helping in improving model's accuracy and generalization.
Parallel loading: It can load the data in parallel using multiple workers, speeding the data loading process.
Overall, the DataLoader class simplifies the process of loading and preprocessing data for deep learning applications in PyTorch.

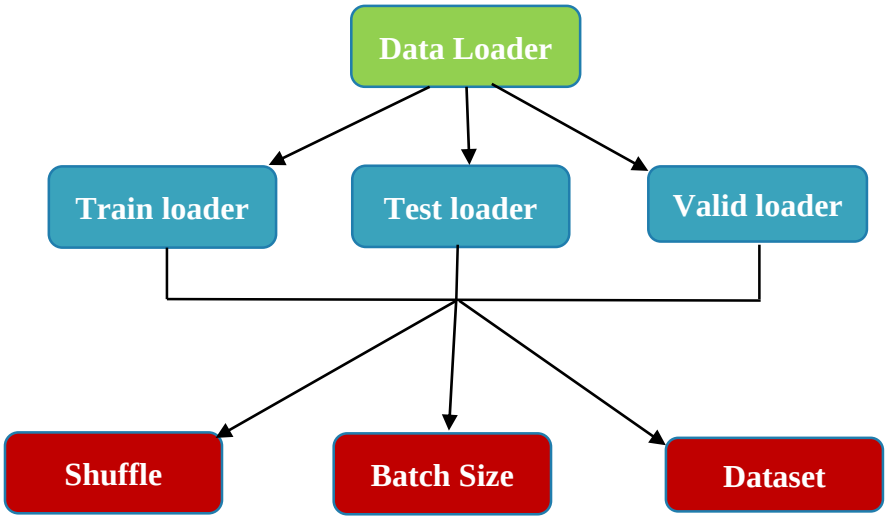


Figure 4.13: Data-loaders

Except AlexNet all the architectures are imported from timm library that provides a collection of state-of-the-art computer vision models and efficient training utilities.

4.2.5.2 Transfer Learning

We create our model with weights pretrained on imagenet for transfer learning.
Since we are accommodating to transfer learning, we need to set few things in a particular way such as**:**

- The number of classes returned by a pretrained imagenet is 1000. But our multiclass problem has only 6 classes. Hence, we need to modify the linear classifier layer of the network in such a way it has 6 out_features.

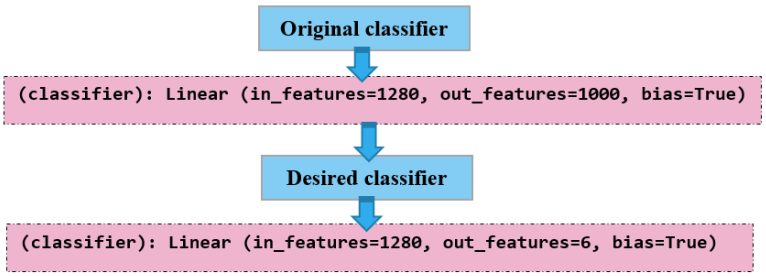


Figure 4.14: Modified Classifier

- Since the all the layers are frozen in transfer learning, we achieve it by setting requires_grad to false so parameters are not updated in those layers. Only the first and last layer is trainable so the pre learned weights are only transferred to custom classes.

Figure 4.15: Freezing layers

Notice that we updated the linear classifier to 6 classes but how?
There exist two methods to do so.

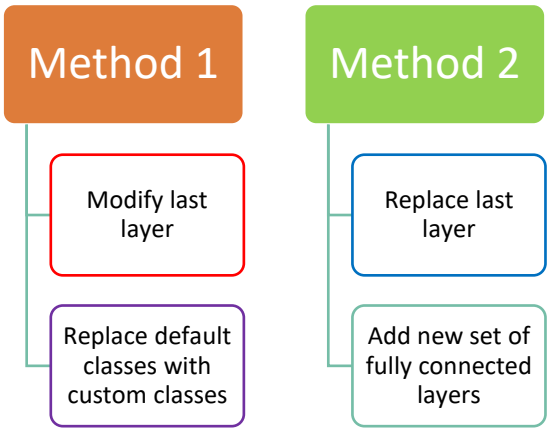


Figure 4.16: Transfer learning as a feature extractor method used in this work.

The first way, where you directly modify the last layer to have an out_features of 6, is a simple and straightforward approach. However, if the pre-trained model was trained on a significantly different dataset, this may not give the best results as the features learned by the pre-trained model may not be optimal for the new task.
The second way, where you replace the last layer with a new set of layers, allows you to customize the architecture of the classifier to better suit your new task and reduce overfitting. This is useful if the number of classes in the new task is significantly different from the classes in the pre-trained model.

Either way, you should ensure that the input size of the next layer(s) matches the output size of the pre-trained model's last layer (in your case, 1280), so that the output of the pre-trained layers can be fed into the new layer(s) correctly. Additionally, make sure that the activation functions and dropout rates are appropriate for your task.

Overall, both approaches can be effective, and the choice between them depends on the specific requirements of your task and the architecture of the pre-trained model. ResNet18 and VGG16 showed good results with method 1 while on EfficientNet method 2 showed better results.
The second method takes slightly more time, and it has more parameters but better accuracy. Since imagenet dataset is very much different from ECG Dataset, hence method 2 is preferable. It also reduces overfitting by varying dropout and out_feature. One can see which value fits better for custom dataset. The sequential block used in method 2 for the implementation of EfficientNet is given below.

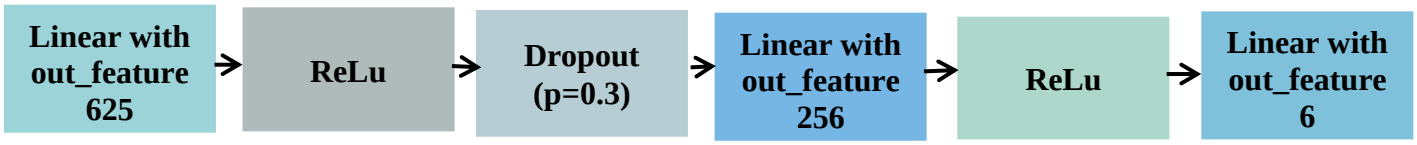


Figure 4.17: Modified layers in Method 2

While adding sequential block, you do not need to pass softmax activation function before the linear classifier since PyTorch comes with CrossEntropyLoss which already contains Softmax so it will be redundant to add another softmax.

4.2.5.3 Contribution to knowledge

For a network to be conductive to hardware needs to accept smaller size and single channel so the layers can be designed efficiently on hardware such as FPGAs.
To accomplish this the size taken for a B0 network is 32x32 and 64x64 with single grayscale channel. The comparative analysis is carried out for different sizes to understand how it affects the memory, parameters, timings, and other hardware constraints.
Since all the CNNs accept 3 channels RGB images, converting the RGB images to grayscale was performed. One can use the transforms method as done in this work or OpenCV to achieve it. But the primary obstacle is the conversion of 3 channels to 1 channel as the model accepts only 3 channels and throw errors if not realized.
The flexibility of PyTorch's timm library allows us to overcome this very easily.

4.2.5.3.1 Potential methods to attain single channel.

- Train grayscale version of imagenet from scratch, too expensive?

- Modify the first hidden layer input channels from three to one. But how the next layer will behave to this change that expects 3 channels output to be passed on it?

The model's architecture is fixed due to the training of weights for a specific configuration of the input. Altering the initial layer would render the remaining weights ineffective. Neural networks are designed to extract complex features from lower-level features as they move deeper into the network. Eliminating the initial layers of a convolutional neural network would break this feature hierarchy, as subsequent layers would not receive the expected input features. This is because the second layer has been trained to anticipate the features of the first layer and changing it would disrupt the flow of feature extraction through the model.

- One simple approach is to create a new dimension and repeat the image array three times within it. This effectively converts the grayscale image into a three-channel image, where each channel contains the same grayscale values. But this is not an efficient way if your bigger goal is hardware compatibility. Having 3 channels each of similar information would add up in nothing but computations.
- It is possible to modify the weights of a model's first convolutional layer and achieve the desired goal. While modifying the weights of the first layer can result in reduced accuracy, the model can still be fine-tuned for improved performance.

Modifying the weights of the first layer does not render the rest of the weights useless, contrary to what others may have suggested. To accomplish this, you will need to add code that modifies the pretrained weights when loading them into your 1-channel model. This can be done by summing the weight tensor over the dimension of the input channels.
The organization of the weight tensor varies depending on the framework being used. For instance, in PyTorch, the default weight tensor organization is {out channels, in channels, kernel height, kernel width}, while in TensorFlow, it is {kernel height, kernel width, in channels, out channels}. You will need to figure out how to grab the weights of the first convolutional layer in your network and modify them before assigning them to the 1-channel model.

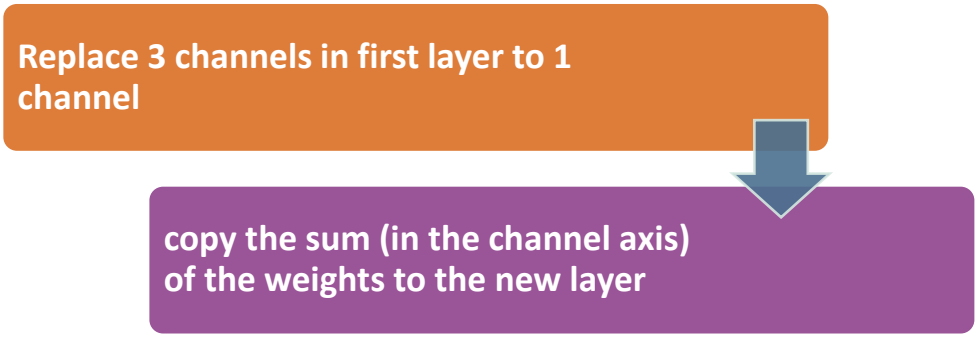


Figure 4.18: Approach 1 for updating weights.

The similar thing can be done using this approach**:**

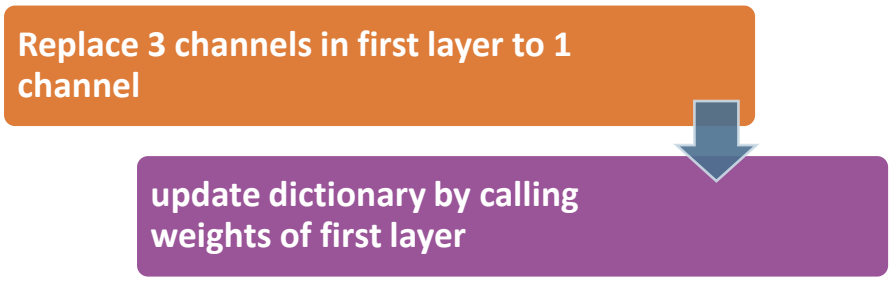


Figure 4.19**:** Approach 2 for updating weights.

The simplest way that timm offers is to set in_chans=1

```
model = timm.create_model (CFG.model_name, pretrained=True, in_chans=1)
```

Figure 4.20: Built-in parameter for updating weights.

As not all pre trained models are available in timm just like Alexnet. So, it is important to know other ways to achieve single channel.

For MobileNet Architecture, fine-tuning is performed to get the most out of transfer learning. After training a model traditionally, the model is trained again with a new learning rate of 0.0001 and for a batch size of 32 from 16. It is worth noting that the model's last four layers are removed and output from the Global Average Pooling layer is to be reshaped to pass it to the classifier layer. The final 22 layers of MobileNet's version 1 and version 3 are trained, for version 2 only the final 25 layers are trained. This proved to be promising as the results achieved are astonishing. This is not an optimal choice; one can vary and observe the behavior and choose whatever best suits one's goal.

4.2.6 Training and Evaluation

The next step after updating model to cater to our needs is to pass the model to the device which is GPU. For training and testing, Colab GPU and NVIDIA GPU cards with CUDA compute capability are used.

Summary of the model gives details about the size of model, trainable and non-trainable parameters, and parameters of each layer with its output shape which will be beneficial in deciding hardware utility.

Before the training of the model begins, we need to define which optimizer is used, which loss function it will follow, how training and testing accuracies and losses will be calculated. Since during training drop out is turned on as the weights gets updated

during backpropagation and gets off during validation, for that we only set gradient for training loop in ECGTrainer class.
For the training, we used Adam Optimizer, nn.CrossEntropyLoss. It combines the nn.LogSoftmax function and the nn.NLLLoss function into a single class. In other words, nn.CrossEntropyLoss takes in raw logits or scores from the last layer of the neural network and applies a softmax function to them. The softmax function converts the logits into a probability distribution over the classes. The class with the largest probability is then considered the predicted class. The Hyperparameters used for the implementation are stated below**:**

Table 4.3: Hyperparameters

Batch Size	Learning Rate	Optimizer	Epochs	Criterion
16	0.001	Adam	10	Cross Entropy Loss

One can experiment with removing ReLU from linear classifier and varying dropout or putting dropout before the ReLU function.

4.2.6.1 Weights

We saved the best weights so every time validation loss decreases from first time, it will save in a dictionary. The weights are saved in .pt format which contains weight and bias of the model that can be used for further inference. Netron is a free tool to visualize .pt files. The visualization is shown below. You can hover over the modules, and it shows the details. Or one can convert .pt file into HDF format to view hierarchy of the model using HDF viewer.

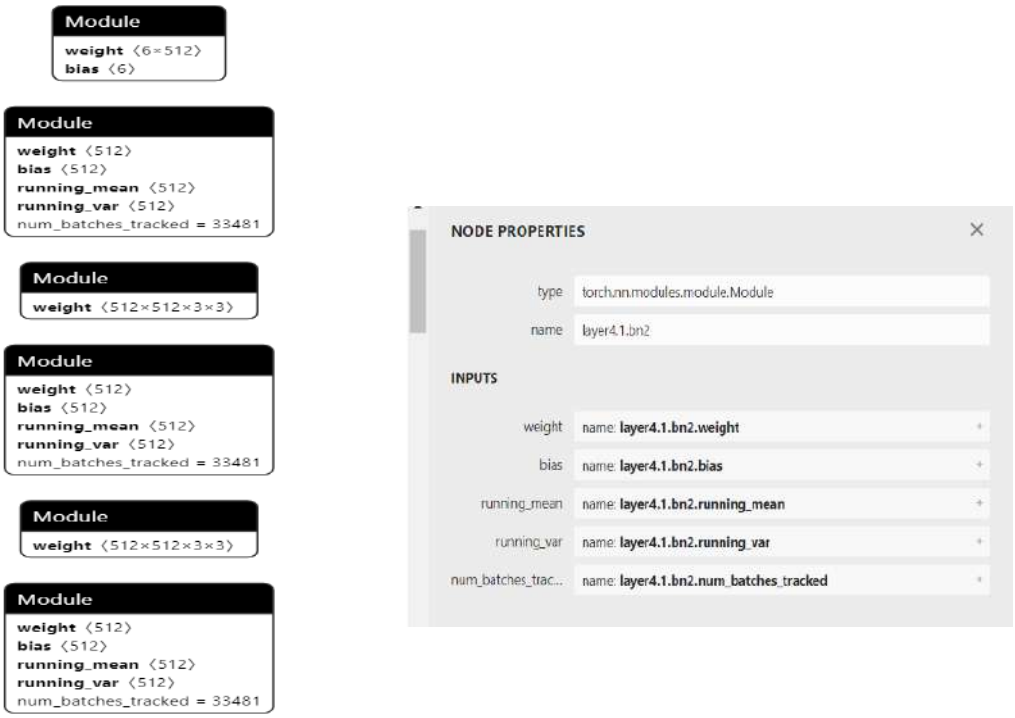


Figure 4.21: Weights Visualization

The results of training and testing will be discussed in the next chapter.

4.2.6.2 Evaluation/Testing

Once the training is over, it must be evaluated on the whole test data to check the overall performance. The metrics used to evaluate the model are as follows.

1. Confusion Matrix
2. Categorical Accuracy
3. Precision
4. Recall
5. F1 Score
6. ROC_AUC Curve

4.2.6.2.1 Timing Analysis

The major impediment here is the time taken by a model to train and test the input. Since our interest is hardware integration, for which timing will be crucial. To evaluate time taken by a model to train in one epoch and single test time we can use notebook's tqdm package which displays a progress bar with iterations per second.

4.2.6.2.2 TQDM Package

The time duration for training can be displayed using tqdm. It tells each epoch time and iterations/seconds. But no image takes equal time in training and testing, so we are interested in average time of each epoch and average single testing time.

4.2.6.2.3 Profiler Run

Besides we can run profiler or timer on a cell where we can predict a single image. The detailed analysis is discussed in the next chapter with a comparison with varied sizes.

4.2.6.3 Class Imbalance

Class imbalance occurs when one or more classes have significantly less or more samples as compared to the other classes in the dataset. Class imbalance is very common in machine learning where the distribution of classes in the training data is not equal. This can lead to a biased model that performs poorly on the underrepresented classes.

To eradicate class imbalance, there are multiple techniques to be used such as oversampling the minority class, under sampling the majority class, or using cost-sensitive learning algorithms. Another creative way is to use Generative Adversarial Networks (GANs) for data generation. Other approaches include data augmentation, ensemble learning, and synthetic data generation.

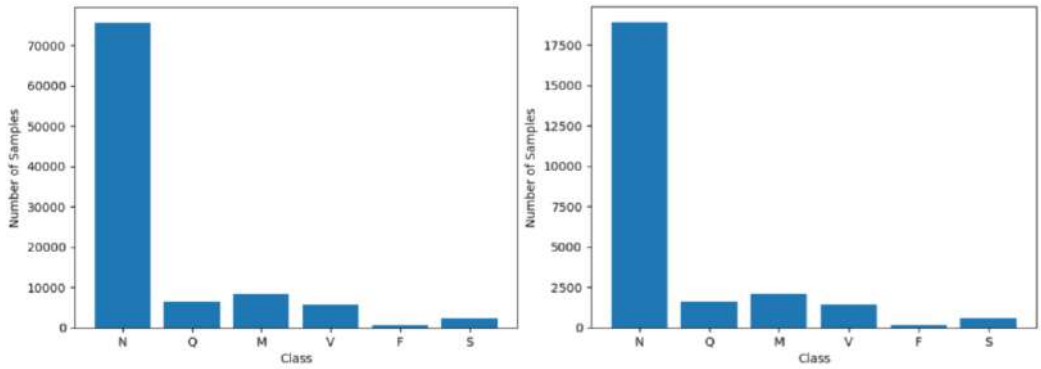


Figure 4.22: Distribution of all classes in train(on left) and test folder(on right)

4.2.6.4 Handling Imbalance

It's important to note that the appropriate technique to address class imbalance depends on the specific problem and the available data. Additionally, the evaluation of the model should be done using appropriate metrics, such as precision, recall, F1-score that consider the class imbalance.

Unfortunately, this is one of the factors causing overfitting that will be addressed in the next chapter.

Since ECG signal is a critical signal with lots of critical information. Performing data augmentation on certain datasets can be challenging, as cropping or clipping signals can result in information loss, while extending them or creating larger windows may introduce overlap with other classes.

Chapter 5

Results and Analysis/Comparison

The results of training and evaluating CNNs are given in this section. Analysis is performed in context of hardware compatibility for which different aspects are considered as stated below**:**

- **Computational Time**
- **Model Size**
- **Accuracy of the model**

5.1 EfficientNet (Method 1) Results and Comparison

5.1.1 Overall Timing Comparison

Table 5.1: Overall Timing details of EfficientNet -V1

Model-V1	Size (Height ×Width)	Training Time (s)	Testing Time (s)
B0	$32x32$	3527	103
	$64x64$	4139	112
B1	$64x64$	4076	103
	$128x128$	4777	127

Table 5.2: Overall Timing details EfficientNet-V2

Model-V2	Size (Height × Width)	Training Time (s)	Testing Time (s)
B0	$32x32$	3855	101
	$64x64$	4078	106
B1	$64x64$	4045	103
	$128x128$	4624	133

5.1.2 Average Time Comparison

Table 5.3: Average time comparison of EfficientNet-V1

Model-V1	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (Milli-seconds)
B0	$32x32$	5.878	4.15
	$64x64$	6.898	4.51
B1	$64x64$	6.828	4.15
	$128x128$	7.96	5.12

Table 5.4: Average time comparison of EfficientNet-V2

Model-V2	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (Milli-seconds)
B0	$32x32$	6.425	4.07
	$64x64$	6.796	4.27
B1	$64x64$	6.741	4.15
	$128x128$	7.706	5.36

5.1.3 Accuracy Comparison

Table 5.5: Accuracy comparison of EfficientNet-V1 sizes

Model-V1	Size (Height × Width)	Accuracy (%)
B0	$32x32$	86.13
	$64x64$	93.95
B1	$64x64$	93.14
	$128x128$	98.15

Table 5.6: Accuracy comparison of EfficientNet-V2 sizes

Model-V2	Size (Height × Width)	Accuracy (%)
B0	$32x32$	87.93
	$64x64$	95.44
B1	$64x64$	95.62
	$128x128$	99.16

5.1.4 Model Parameters and Size

Table 5.7: Trainable Parameters comparison

Model	B0	B1	B0	B1
Version	**V1**		**V2**	
Parameters Size (MB)	15.31	24.87	22.38	26.20
Trainable Parameters	7,686	7,686	7,686	7,686
Total Parameters	4,014,658	6,520,294	5,865,814	6,867,162

Table 5.8: Model Size comparison

Model	B0	B0	B1	B1	B0	B0	B1	B1
Version	**V1**				**V2**			
Input Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Model Size (MB)	19.99	33.85	51.10	129.95	26.02	36.35	44.81	100.11

5.1.5 Model size, computational cost, and accuracy analysis:

The results above showed that by increasing size comes more computational expenses but better accuracy. The benefits of transfer learning can be seen here as out of total parameters in table above only 7,686 are trainable thus saving us a lot of computational power. While comparing V1 with V2, we can observe that we got better accuracy with better computational cost. To understand the comparison let's take B0 size 32x32 of version 1 from above table which has 86.13% accuracy with 5.878 minutes of average training and 4.15ms single shot testing time and B1 with size 64x64 of version 1 with 93.14% accuracy with 6.828 minutes training and 4.15ms single shot testing time. But if you look closely at the B1 64x64 from version 2 you can see it has better accuracy of 95.62% with 6.741 minutes training and 4.15ms single shot testing time which is better

than going for 64x64 size of B1 in version 1. So, if the choice has to be made one can opt for V2 instead of moving to B1 of V1 from B0 of V1.

5.2 EfficientNet (Method 1) Version 1

5.2.1 Performance Metrics

Confusion Matrix

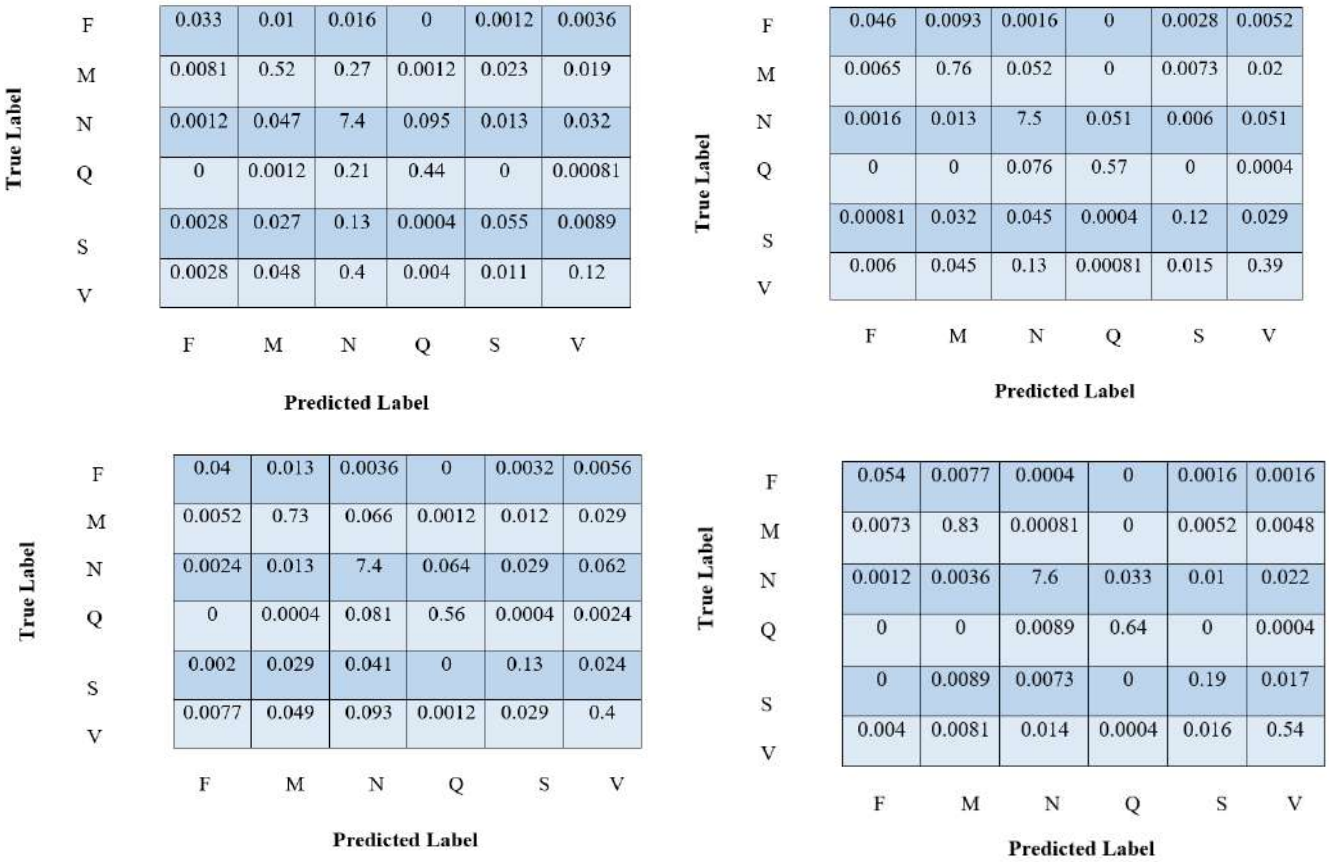
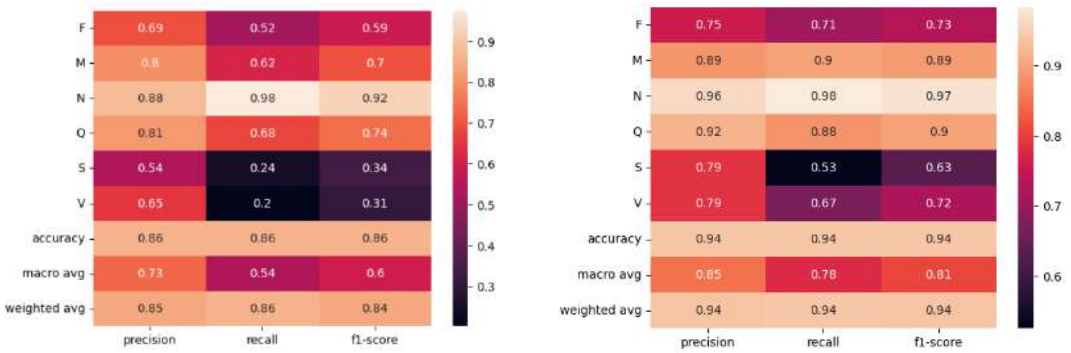


Figure 5.1 First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

Classification report



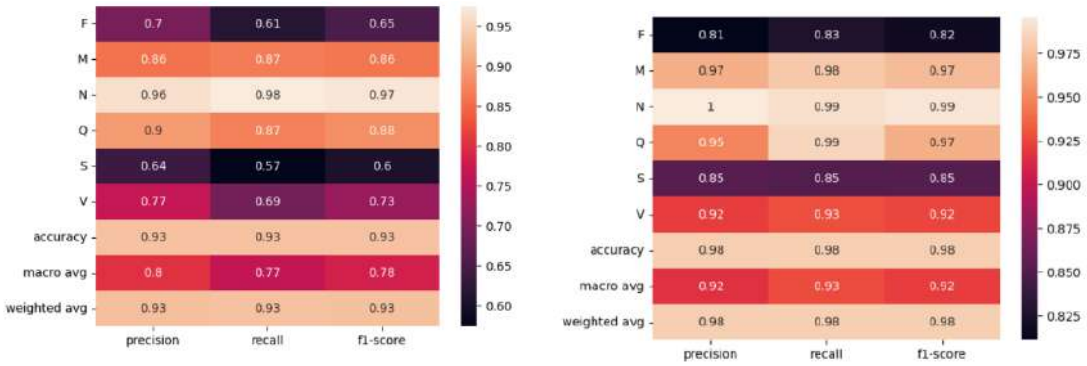


Figure 5.2: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

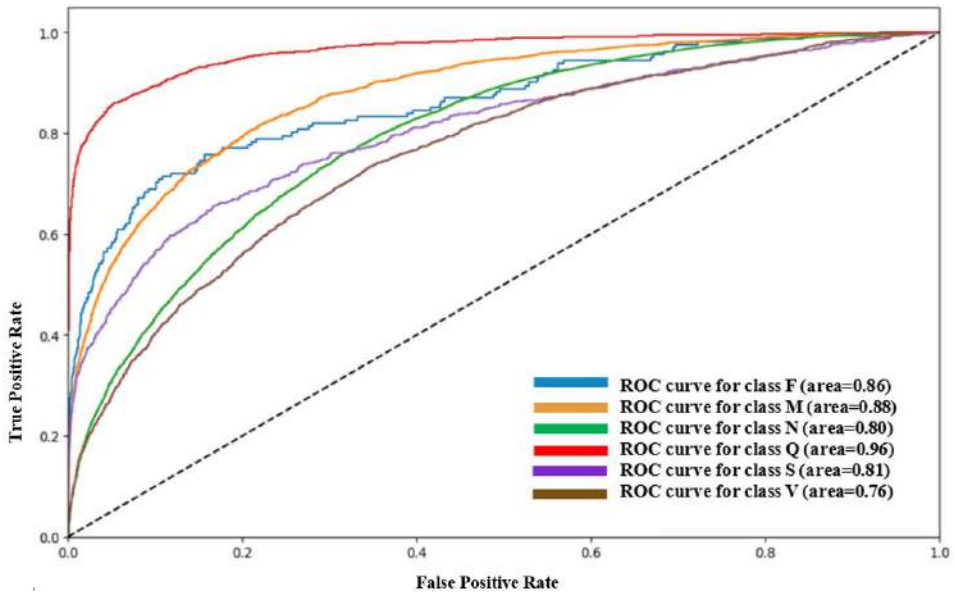


Figure 5.3: ROC curve of B0 of size 32x32

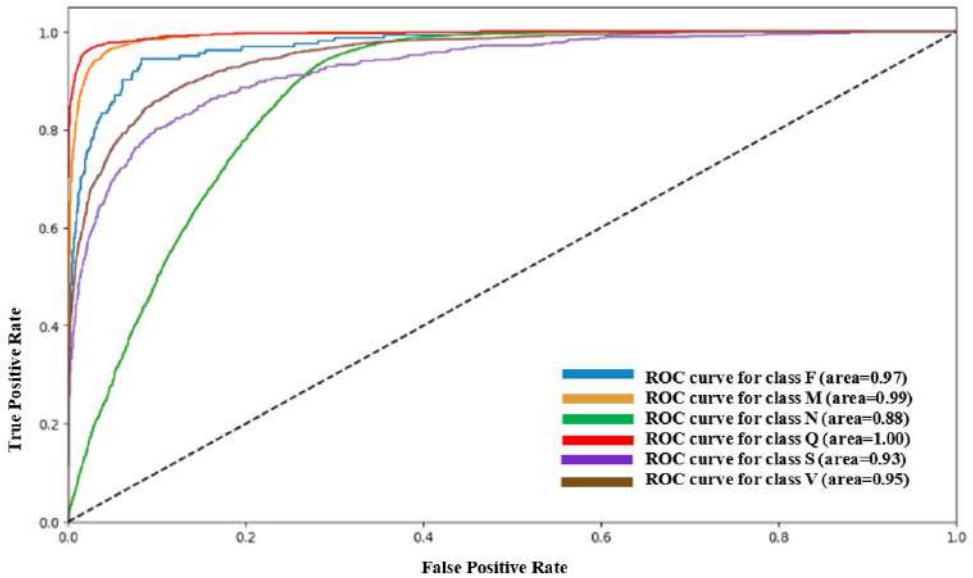


Figure 5.4: ROC curve of B0 of size 64x64

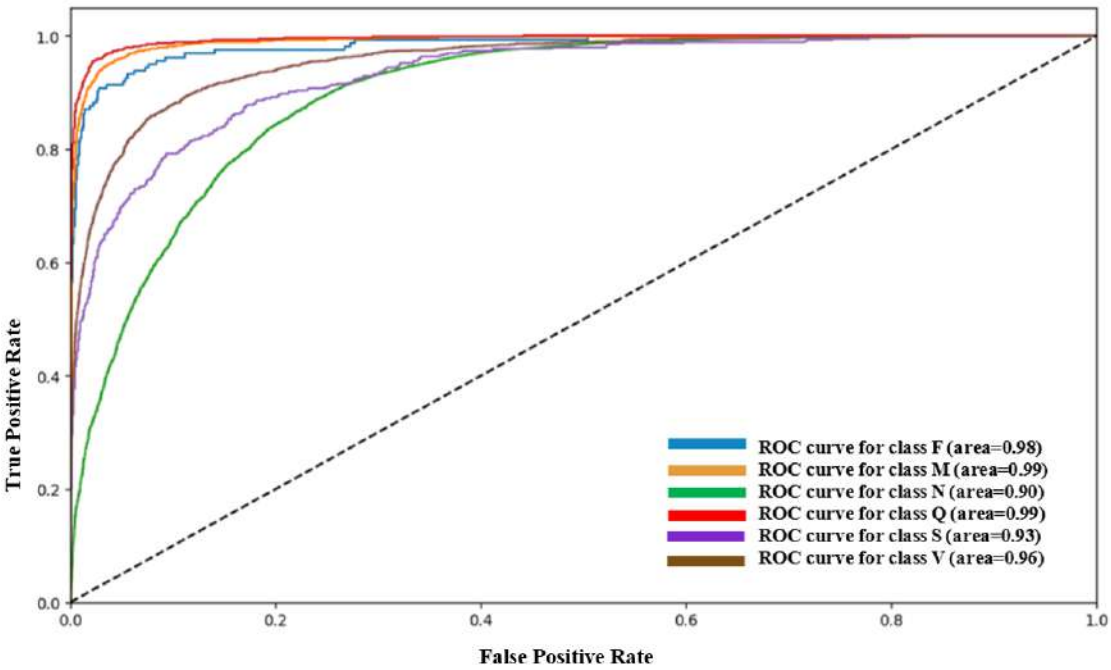


Figure 5.5: ROC curve of B1 of size 64x64

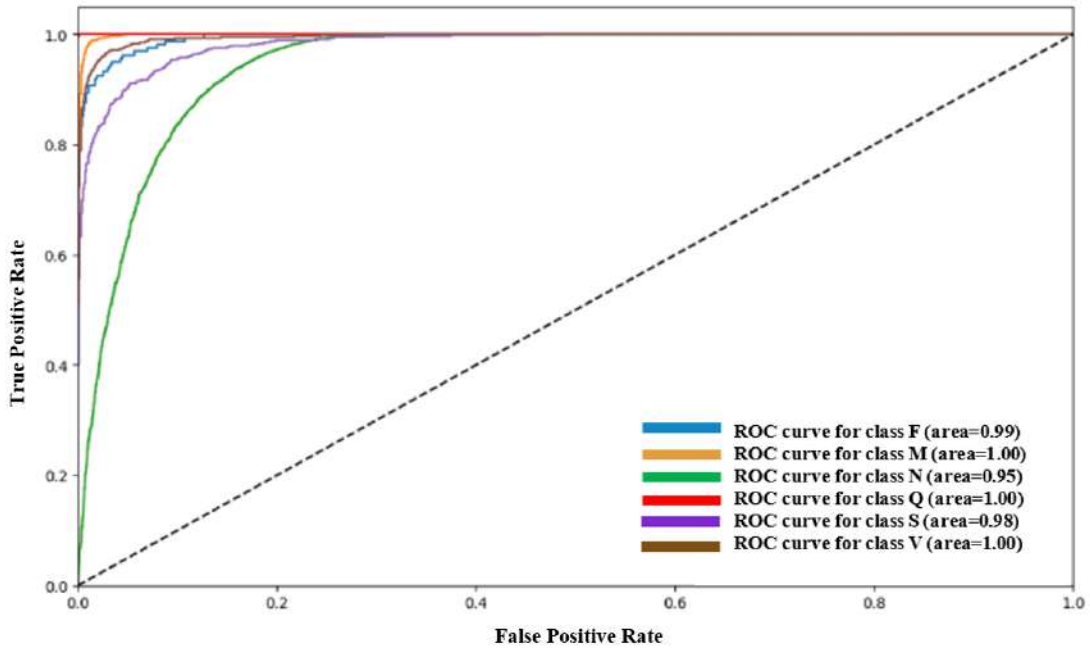


Figure 5.6: ROC curve of B1 of size 128x128

Learning Curves

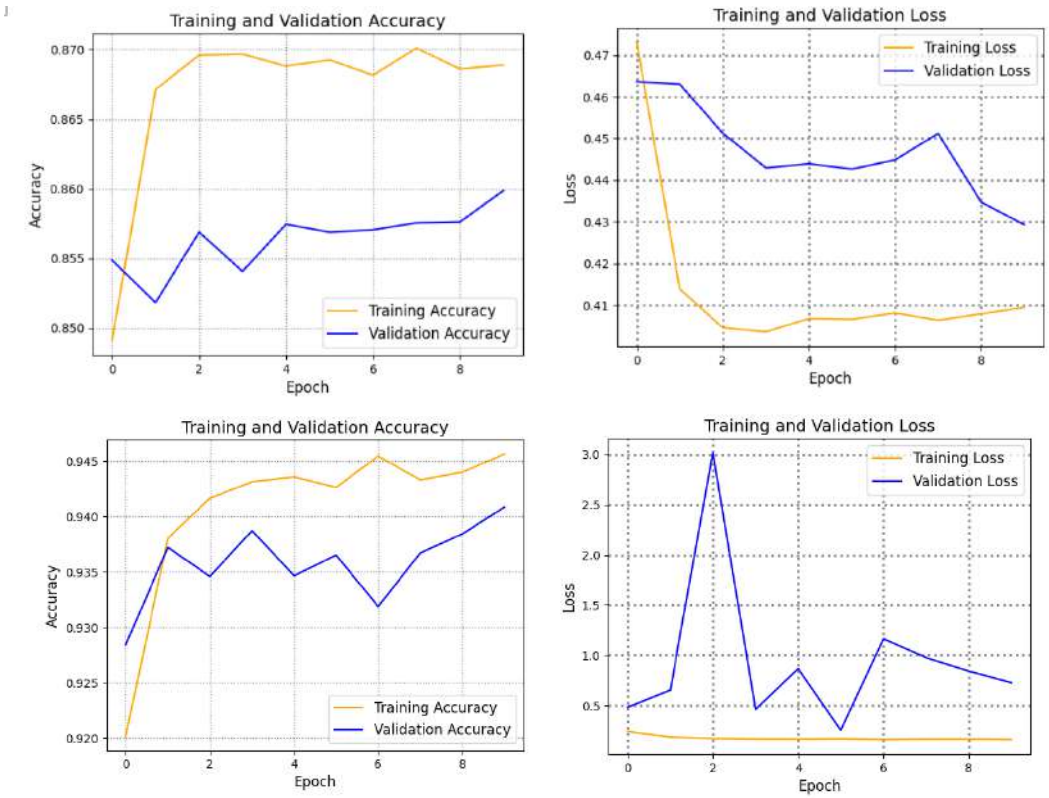


Figure 5.7: First row from left B0 size 32x32 and second row 64x64 respectively

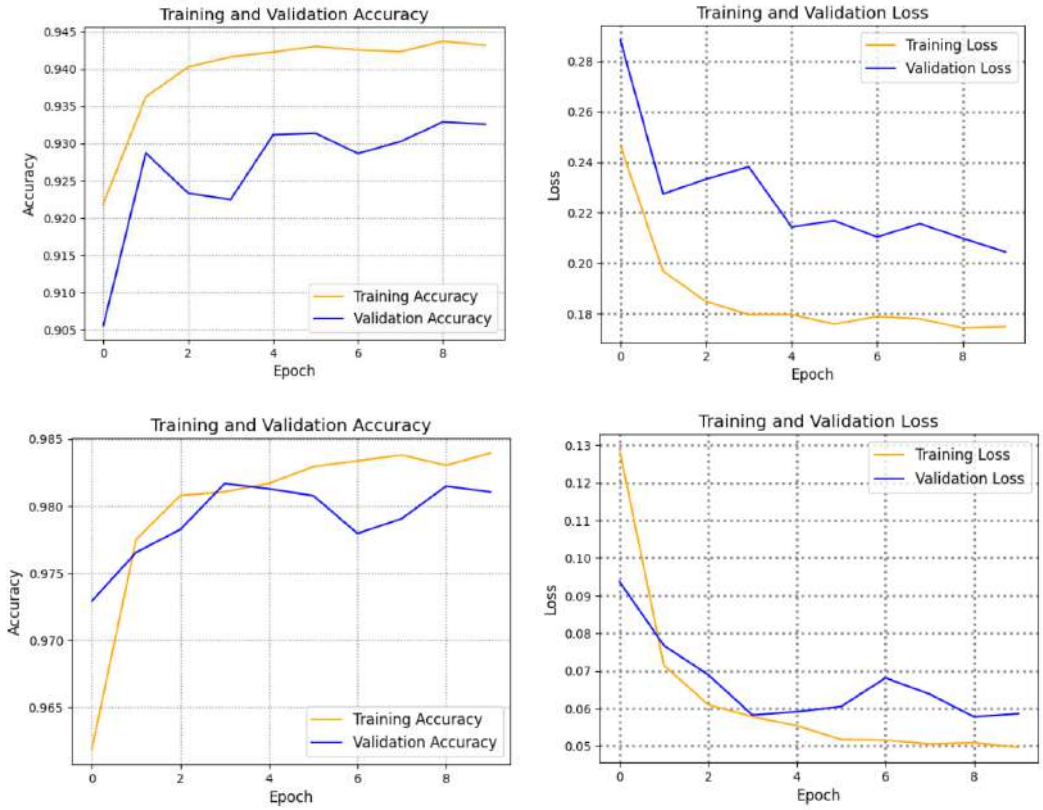


Figure 5.8: First row from left B1 size 64x64 and second row 128x128 respectively

5.3 EfficientNet (Method 1) Version 2

5.3.1 Performance Metrics

Confusion Matrix

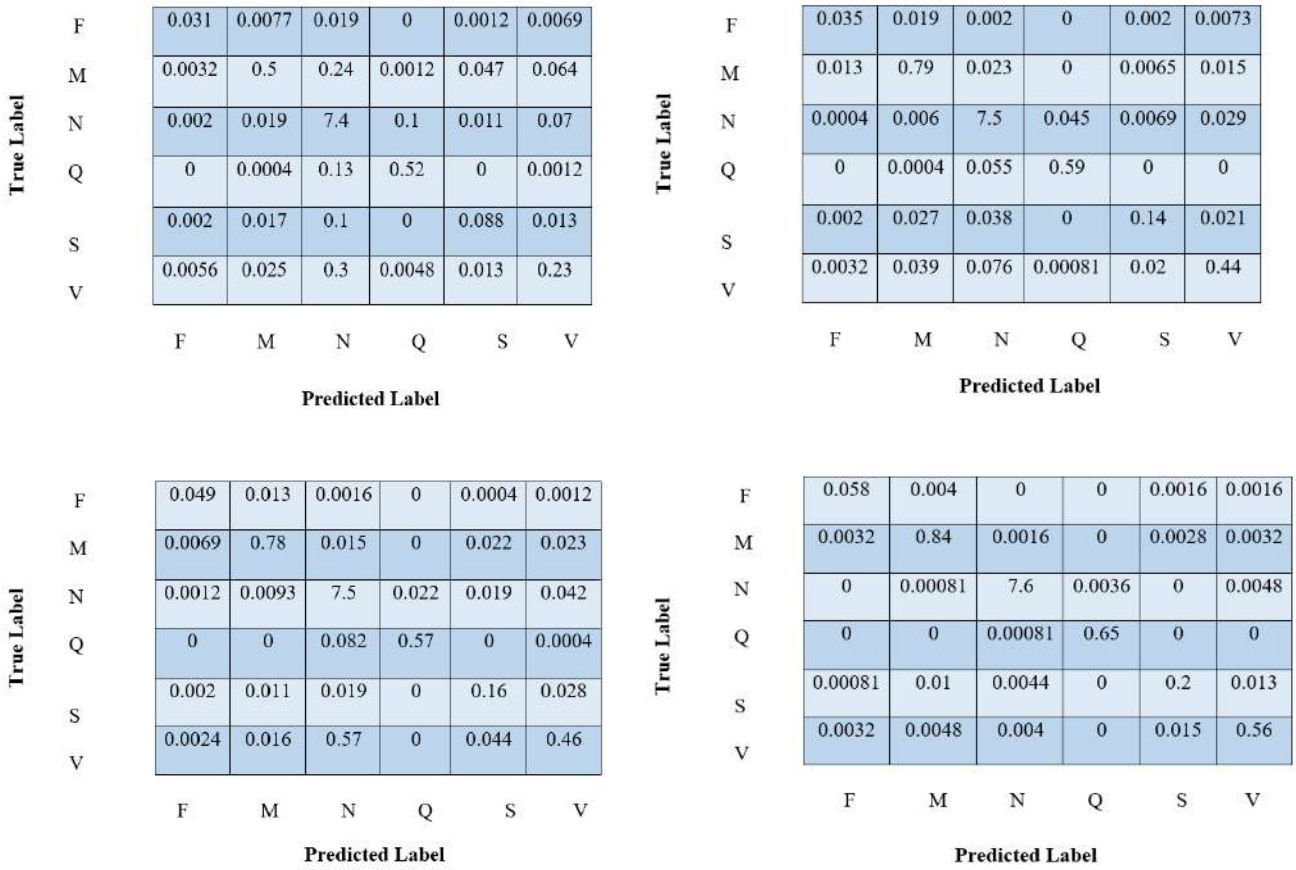
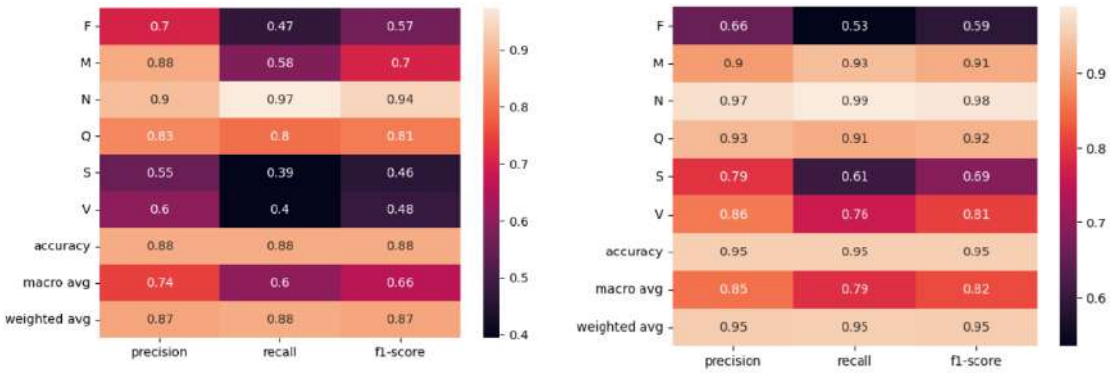


Figure 5.9: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively



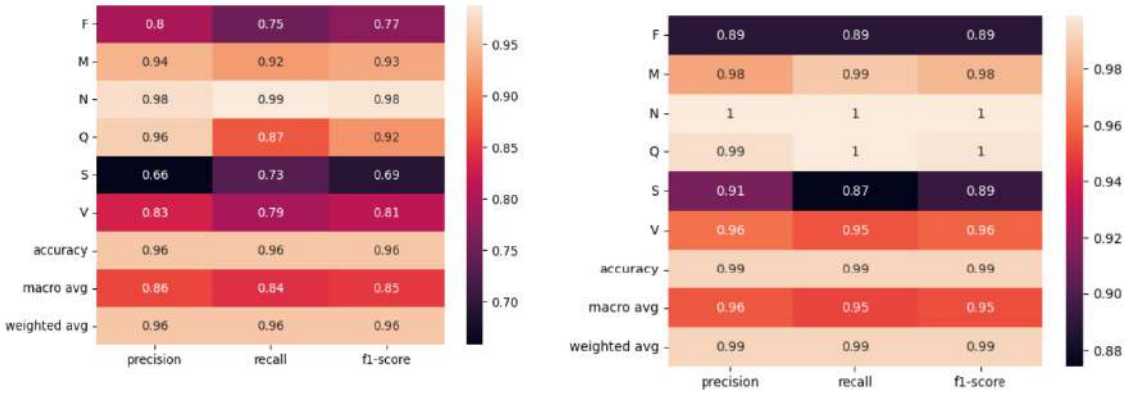


Figure 5.10: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

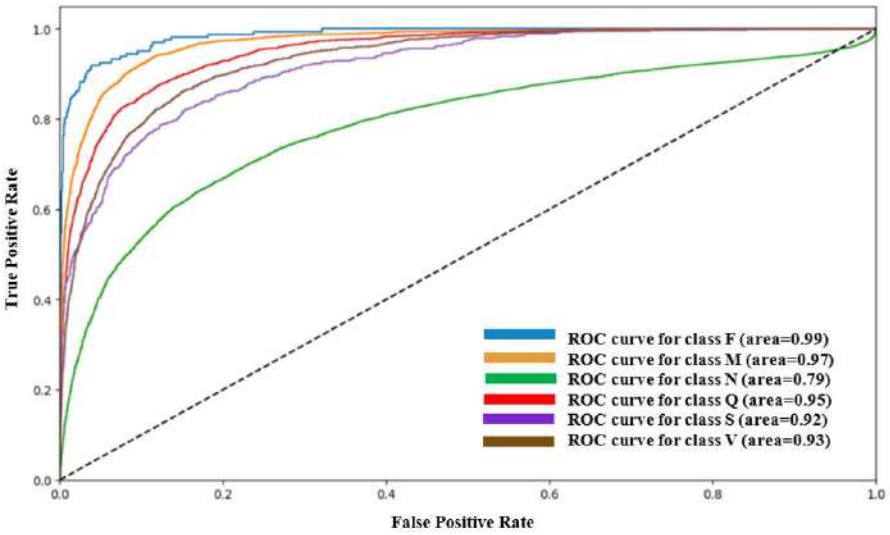


Figure 5.11: ROC curve of B0 of size 32x32

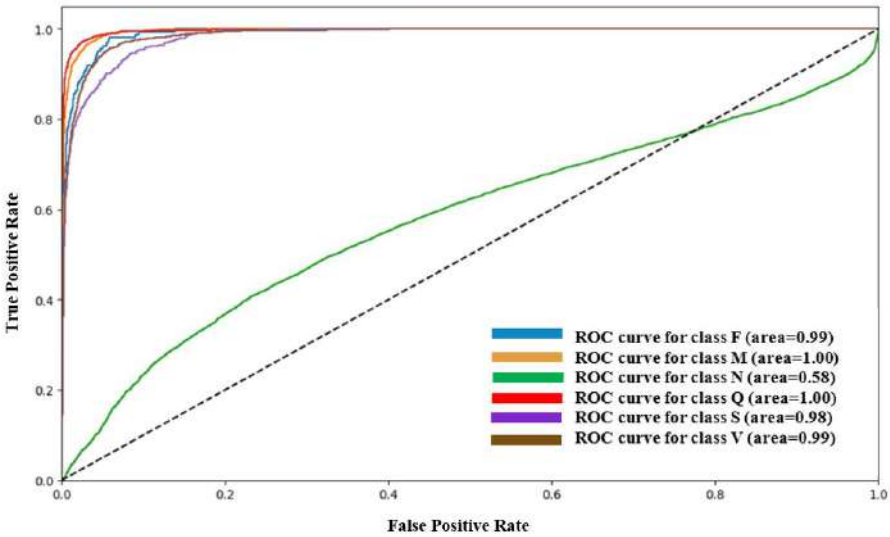


Figure 5.12: ROC curve of B0 of size 64x64

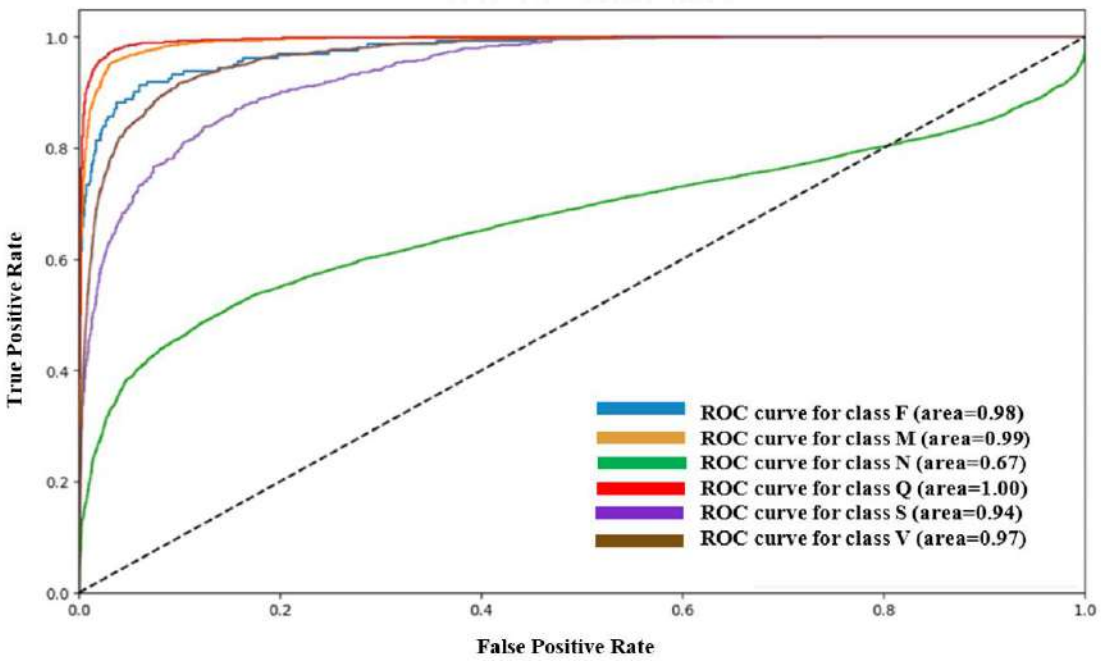


Figure 5.13: ROC curve of B1 of size 64x64

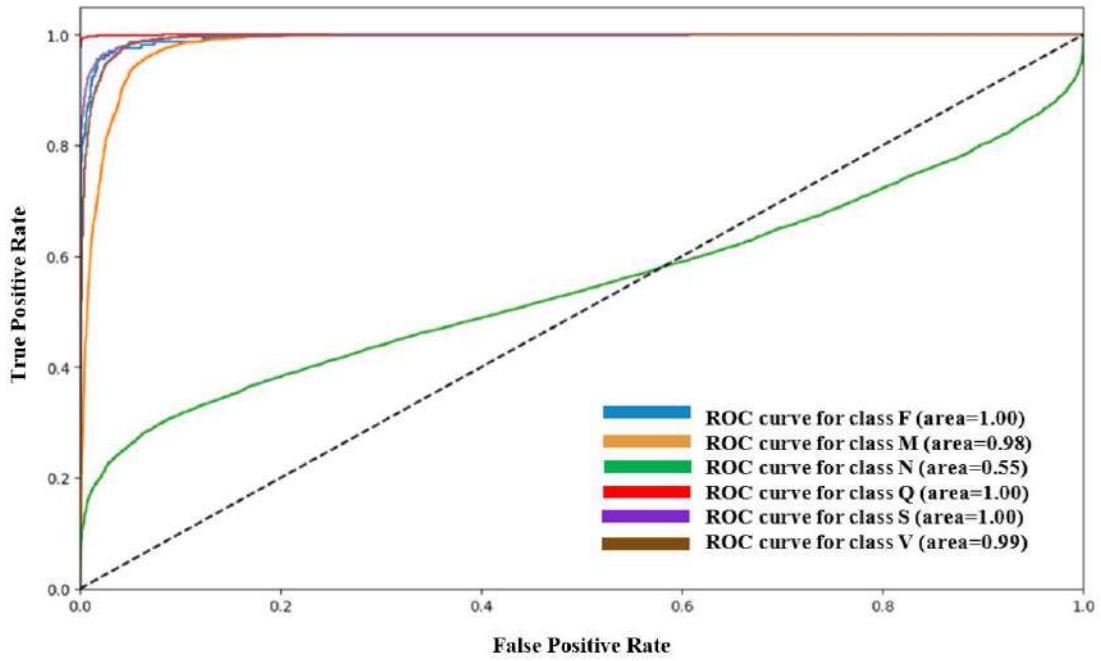


Figure 5.14: ROC curve of B1 of size 128x128

Learning Curves

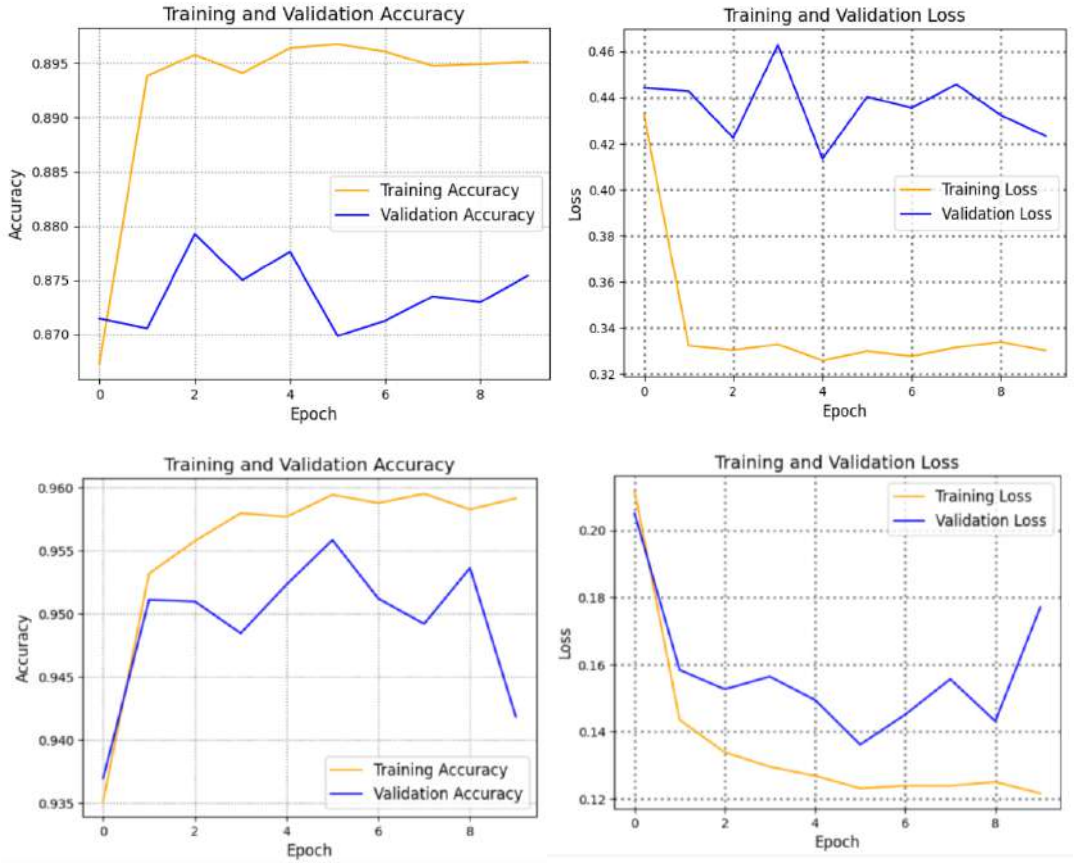


Figure 5.15: First row from left B0 size 32x32 and second row 64x64 respectively

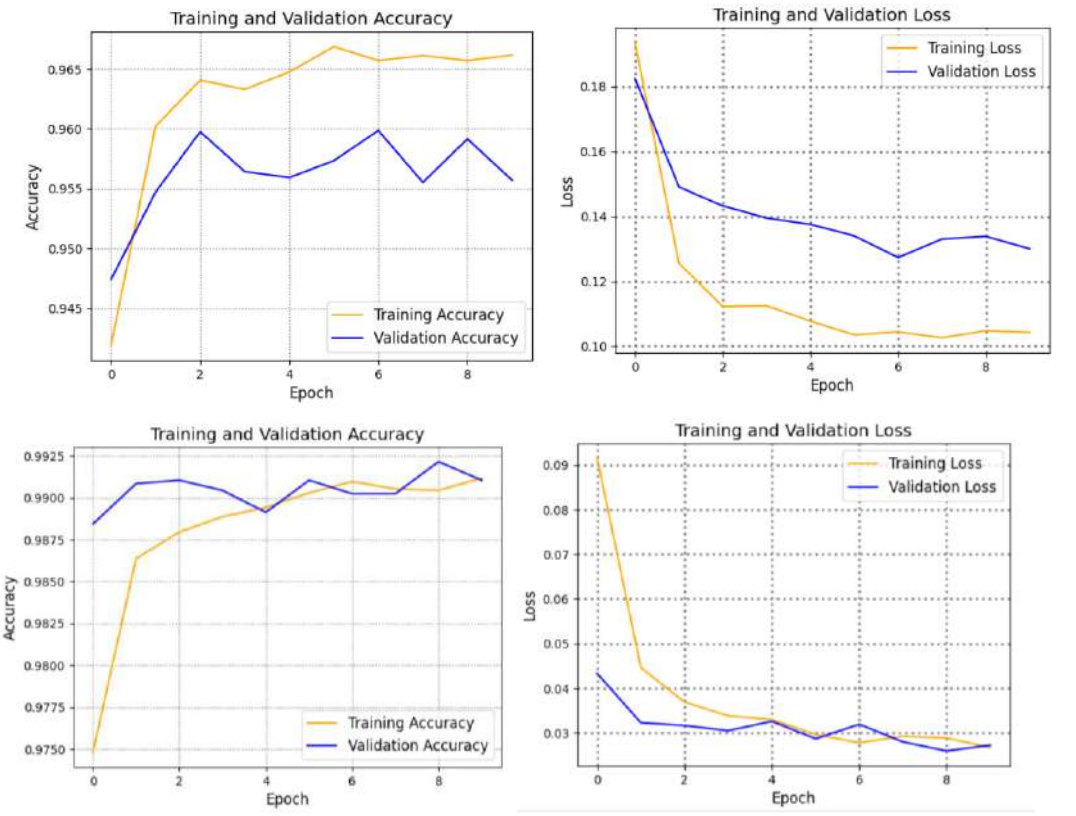


Figure 5.16: First row from left B1 size 64x64 and second row 128x128 respectively

5.3.2 Performance Analysis:

A class-wise distribution of predictions made by the model can be displayed using a confusion matrix. The predicted results are on y-axis while true on x-axis. The confusion matrix for a multiclass problem is a bit tricky. The diagonal elements are correctly predicted samples. The higher the diagonal the better it is. The off diagonal is the misclassification of classes. The classes with the highest off-diagonal values are the ones that are most frequently misclassified. If the diagonal values are high and the off-diagonal values are low, then the model is performing well. If the off-diagonal values are high, then the model is frequently misclassifying samples, and further investigation is required to improve the model's performance. Based on confusion matrix, a classification report is generated. Through precision, recall and F1 score we can interpret the impact of class imbalance as well as the overfitting caused by the model. The lower sizes seemed to be confused by the model as smaller size has blurring effect which can cause similarity in classes while with size being increased the model learned the complex features and patterns, hence making good predictions.

A similar thing is noticed in learning curves as well where there is overfitting as one can observe less convergence which gets better on moving to a bigger size where the model has a good fit. To observe the impact of imbalance on the learning of model AUC is plotted above. The plot has false positive rate (FPR) on the x-axis and a true positive rate (TPR) on the y-axis. The range of AUC is between 0 and 1. If the AUC is 1 or closer to 1 then this means that the model is a good classifier. If AUC is below 0.5, it means random classification occurred. To interpret the curve, the class closer to top left shows is a good class in terms of classification. As in the AUC plots above normal class is mostly poor this is since normal class have large imbalance compared to other classes. A detailed comparison of method 1 is given in a table below**:**

Comparison of V1 and V2 with Method 1:

Table 5.9: Comparison of Version 1 vs Version 2 of Method 1

Model	**B0**		**B1**		**B0**		**B1**	
Version	**V1**				**V2**			
Image Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Accuracy (%)	86.13	93.95	93.14	98.15	87.93	95.44	95.62	99.16
Testing Time (ms)	4.15	4.51	4.15	5.12	4.07	4.27	4.15	5.36
Training Time (minutes)	5.878	6.898	6.828	7.96	6.425	6.796	6.741	7.706
Trainable Parameters	7,686	7,686	7,686	7,686	7,686	7,686	7,686	7,686
Total Parameters	4,014,658	4,014,658	6,520294	6,520,294	5,865,814	5,865,814	6,867,162	6,867,162

5.4 Method 2 Comparison Analysis

In comparison with version 2, the EfficientNet model's accuracy has increased when moved from version 1 to version 2. While comparing V1 with V2, we can observe that we got better accuracy with better computational cost. To understand the comparison let's take B0 size 32x32 of version 1 from above table which has 86.13% accuracy with 5.878 minutes of average training and 4.15ms single shot testing time and B1 with size 64x64 of version 1 with 93.14% accuracy with 6.828 minutes training and 4.15ms single shot testing time. But if you look closely at the B1 64x64 from version 2 you can see it has better accuracy of 95.62% with 6.741 minutes training and 4.15ms single shot testing time which is better than going for 64x64 size of B1 in version 1. So, if the choice must be made one can opt for V2 instead of moving to B1 of V1 from B0 of V1.

5.5 Modified EfficientNet (Method 2) Results and Comparison

Overall Timing Comparison:

Table 5.10: EfficientNet -V1 with Method 2

Model-V1	Size (Height × Width)	Training Time (s)	Testing Time (s)
B0	$32x32$	3641	99
	$64x64$	3828	97
B1	$64x64$	4190	108
	$128x128$	4706	121

Table 5.11: EfficientNet –V2 with Method 2

Model-V2	Size (Height × Width)	Training Time (s)	Testing Time (s)
B0	$32x32$	3799	99
	$64x64$	4156	113
B1	$64x64$	4077	104
	$128x128$	4705	129

Average Time Comparison

Table 5.12: EfficientNet -V1 with Method 2

Model-V1	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (milli-seconds)
B0	$32x32$	6.06	3.99
	$64x64$	6.38	3.91
B1	$64x64$	6.98	4.35
	$128x128$	7.84	4.87

Table 5.13: EfficientNet-V2 with Method 2

Model-V2	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (milli-seconds)
B0	$32x32$	6.33	3.99
	$64x64$	6.92	4.55
B1	$64x64$	6.79	4.19
	$128x128$	7.84	5.20

Accuracy Comparison:

Table 5.14: EfficientNet-V1 with Method 2

Model-V1	Size (Height × Width)	Accuracy (%)
B0	$32x32$	89.05
	$64x64$	96.16
B1	$64x64$	94.78
	$128x128$	98.89

Table 5.15: EfficientNet-V2 with Method 2

Model-V2	Size (Height × Width)	Accuracy (%)
B0	$32x32$	90.60
	$64x64$	96.67
B1	$64x64$	96.32
	$128x128$	99.12

Model Parameters and Size

Table 5.16: Trainable parameters comparison in Method 2

Model	B0	B1	B0	B1
Version	V1		V2	
Parameters Size (MB)	18.96	28.51	26.02	29.84
Trainable Parameters	962,423	962,423	962,423	962,423
Total Parameters	4,969,395	7,475,031	6,820,551	7,821,899

Table 5.17: Model size comparison in Method 2

Model	B0	B0	B1	B1	B0	B0	B1	B1
Version	V1				V2			
Input Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Model Size (MB)	23.65	37.51	54.77	133.61	29.68	40.01	48.47	103.77

5.5.1 Analysis:

The results above showed that by increasing size comes more computational expenses but better accuracy. The benefits of transfer learning can be seen here as out of total parameters in table above only 962,423 are trainable thus saving us a lot of computational power. While comparing V1 with V2, we can observe that we got better accuracy with better computational cost. To understand the comparison let's take B0 size 32x32 of version 1 from above table which has 89.05% accuracy with 6.07 minutes of average training and 3.99ms single testing time and B1 with size 64x64 of version 1

with 94.78% accuracy with 6.98 minutes training and 4.35ms single sample testing time. But if you look closely to B1 64x64 from version 2 you can see it have better accuracy of 96.32% with 6.79 minutes training and 4.19ms single sample testing time which is better than going for 64x64 size of B1 in version 1. So if the choice has to be made one can opt for V2 instead of moving to B1 of V1 from B0 of V1.

5.5.2 Comparison with Method 1

But when compared to Method 1 stated above, the trainable parameters are increased due to additional layers being added. Whereas the accuracy achieved is better except for B1 size 128x128 in version 2 of both the method has a slight difference which is not uncommon. Overall, Method 2 is better with accuracy but at the cost of increased parameters for training.

5.6 EfficientNet (Method 2) Version 1

5.6.1 Performance Metrics

Confusion Matrix

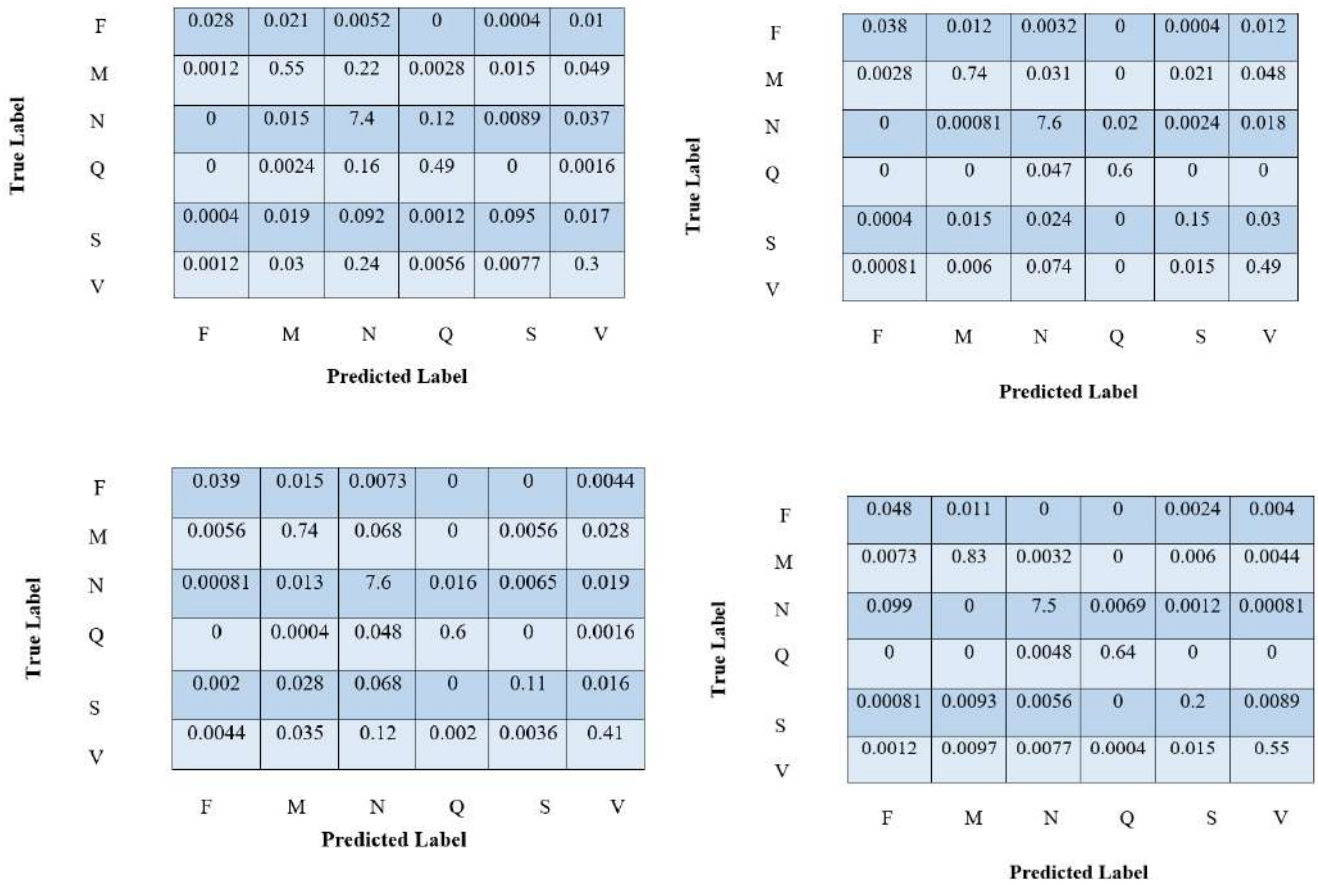


Figure 5.17: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

Classification Report

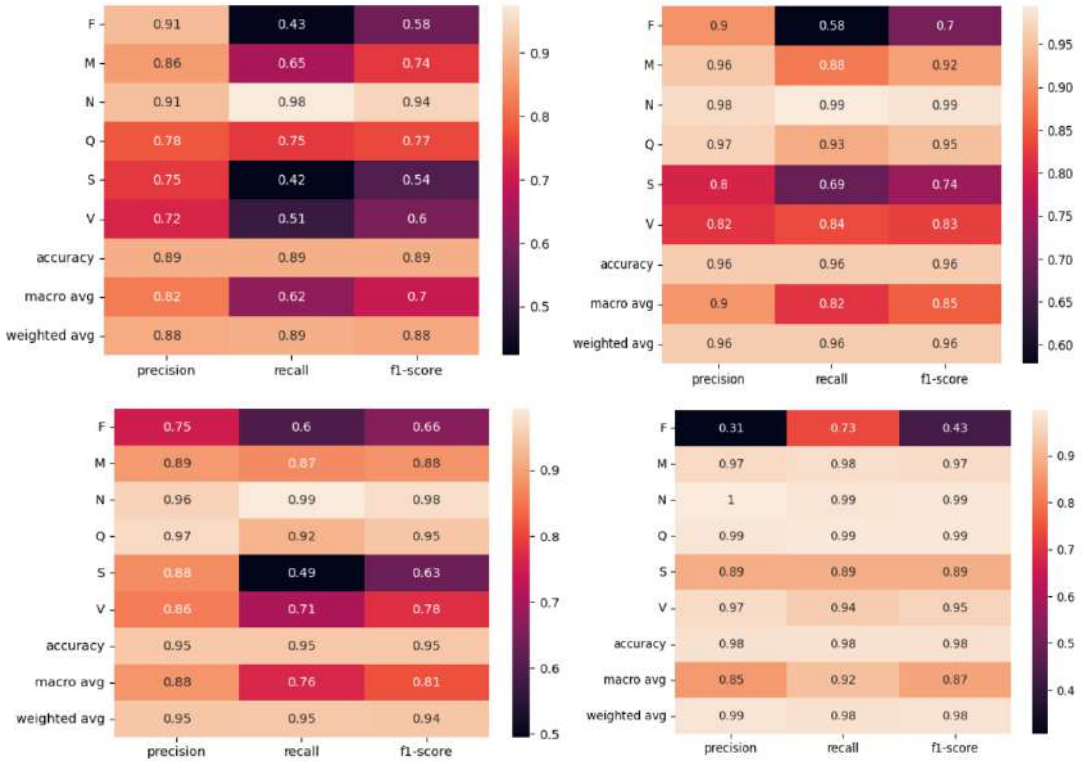


Figure 5.18: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

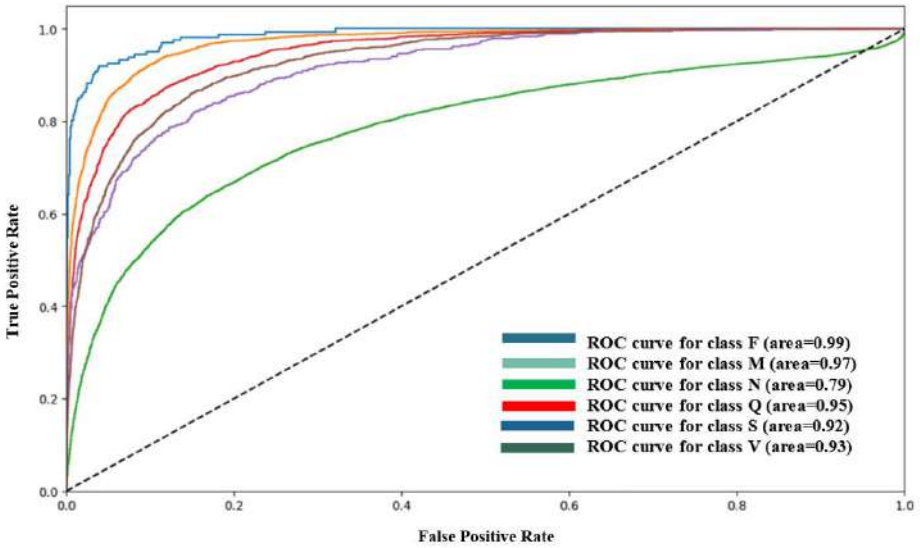


Figure 5.19: ROC curve of B0 of size 32x32

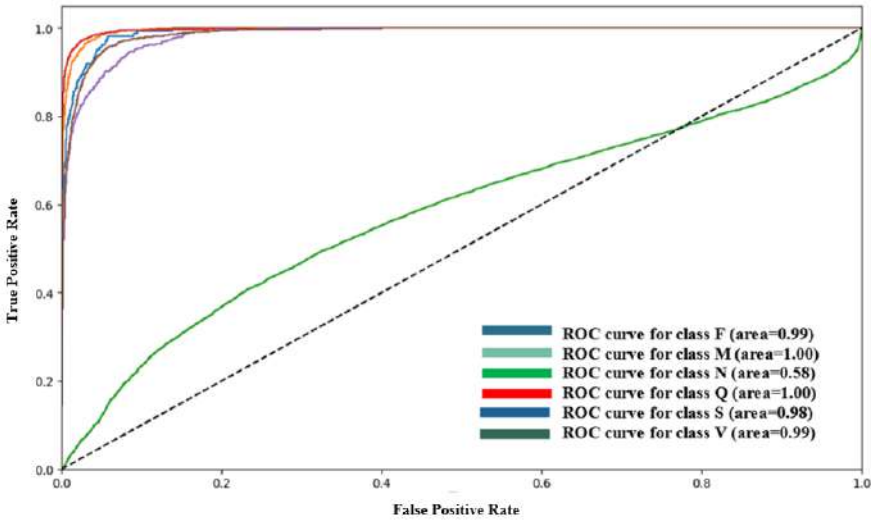


Figure 5.20: ROC curve of B0 of size 64x64

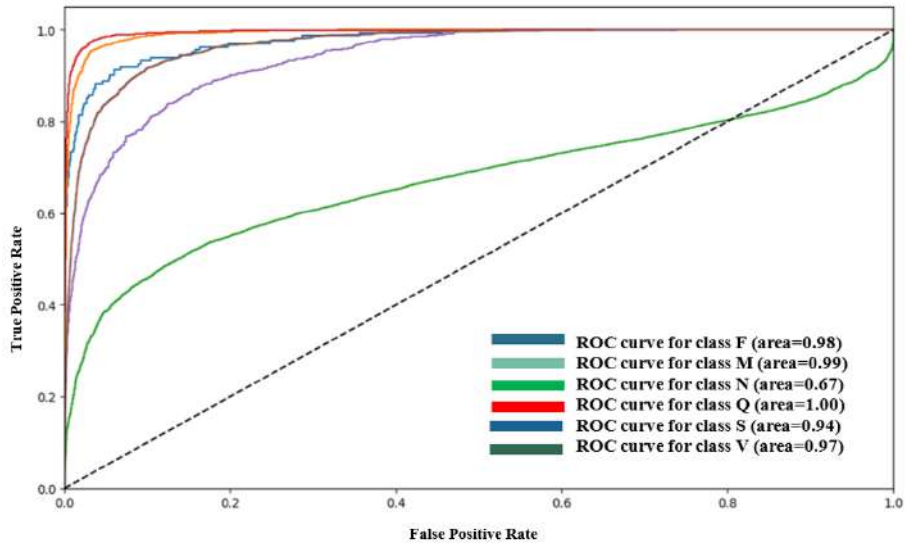


Figure 5.21: ROC curve of B1 of size 64x64

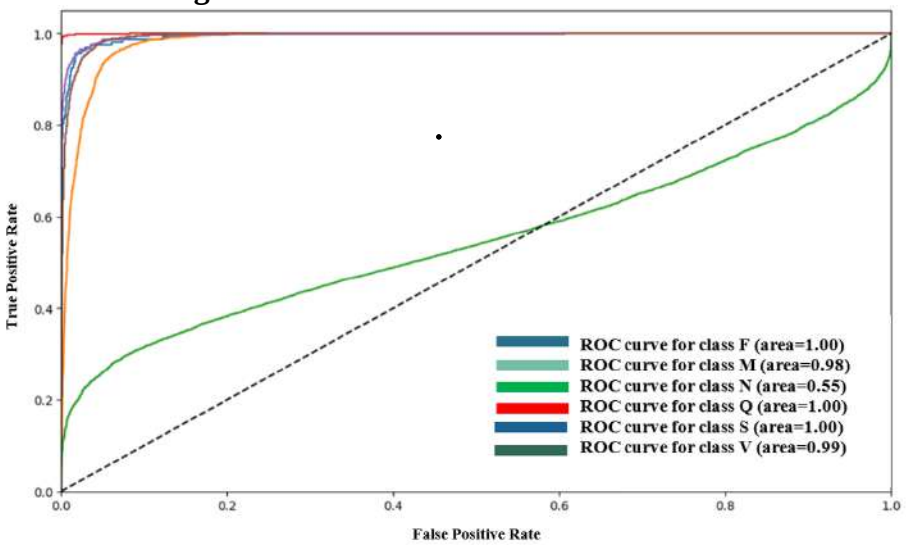


Figure 5.22: ROC curve of B1 of size 128x128

Learning Curves

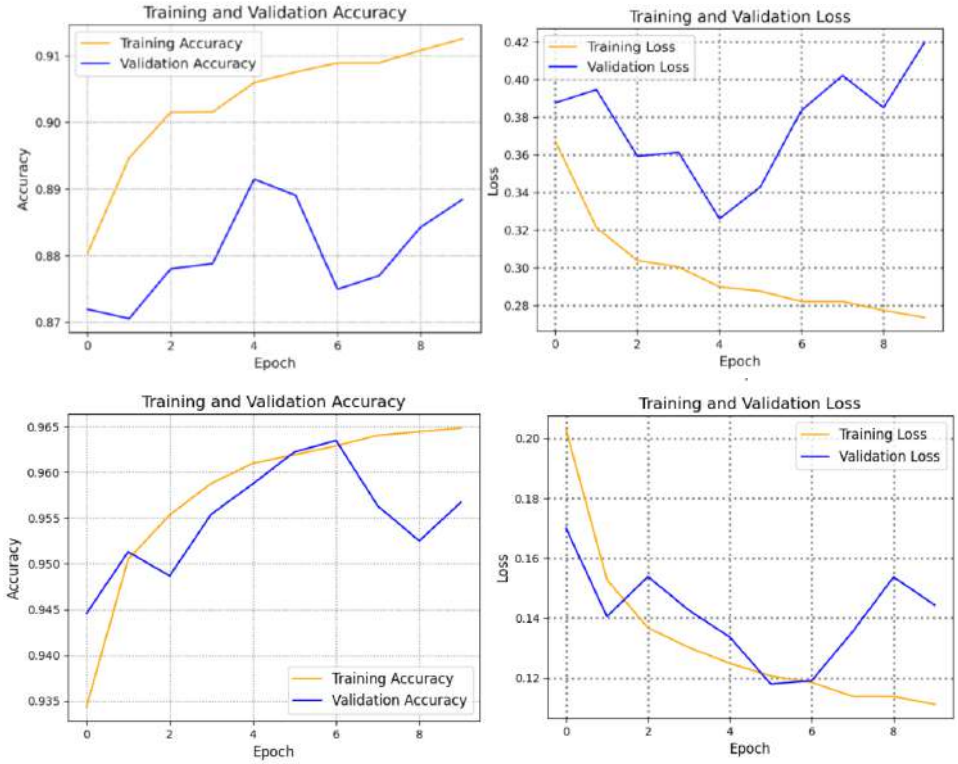


Figure 5.23: First row from left B0 size 32x32 and second row 64x64 respectively

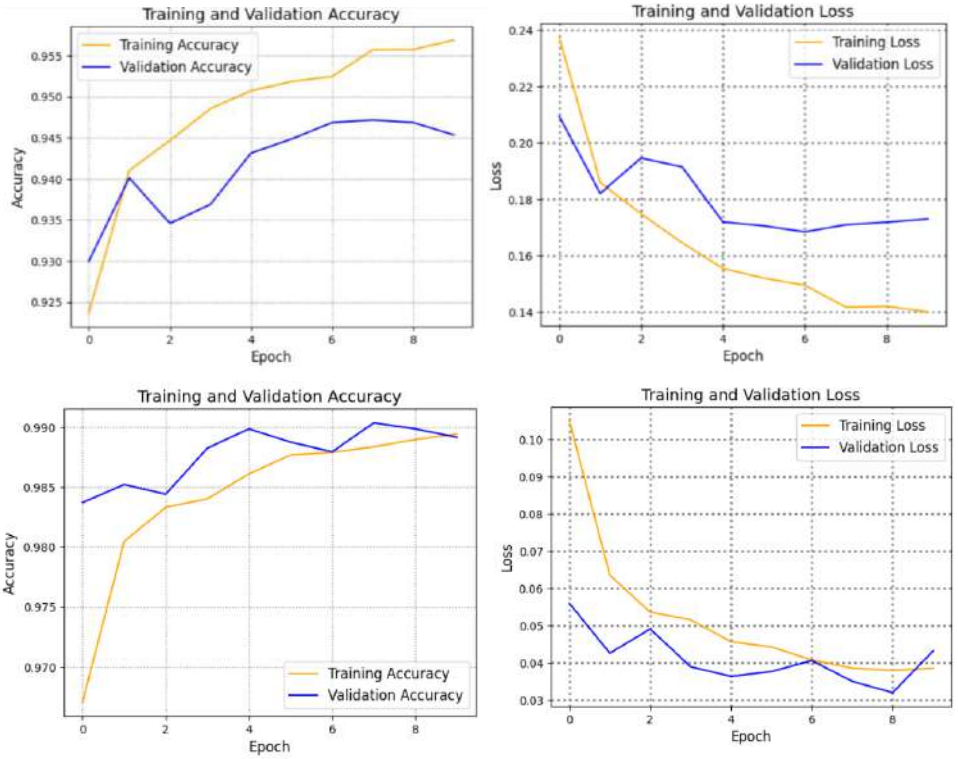


Figure 5.24: First row from left B1 size 64x64 and 128x128 respectively

5.7 EfficientNet (Method 2) Version 2

5.7.1 Performance Metrics

Confusion Matrix

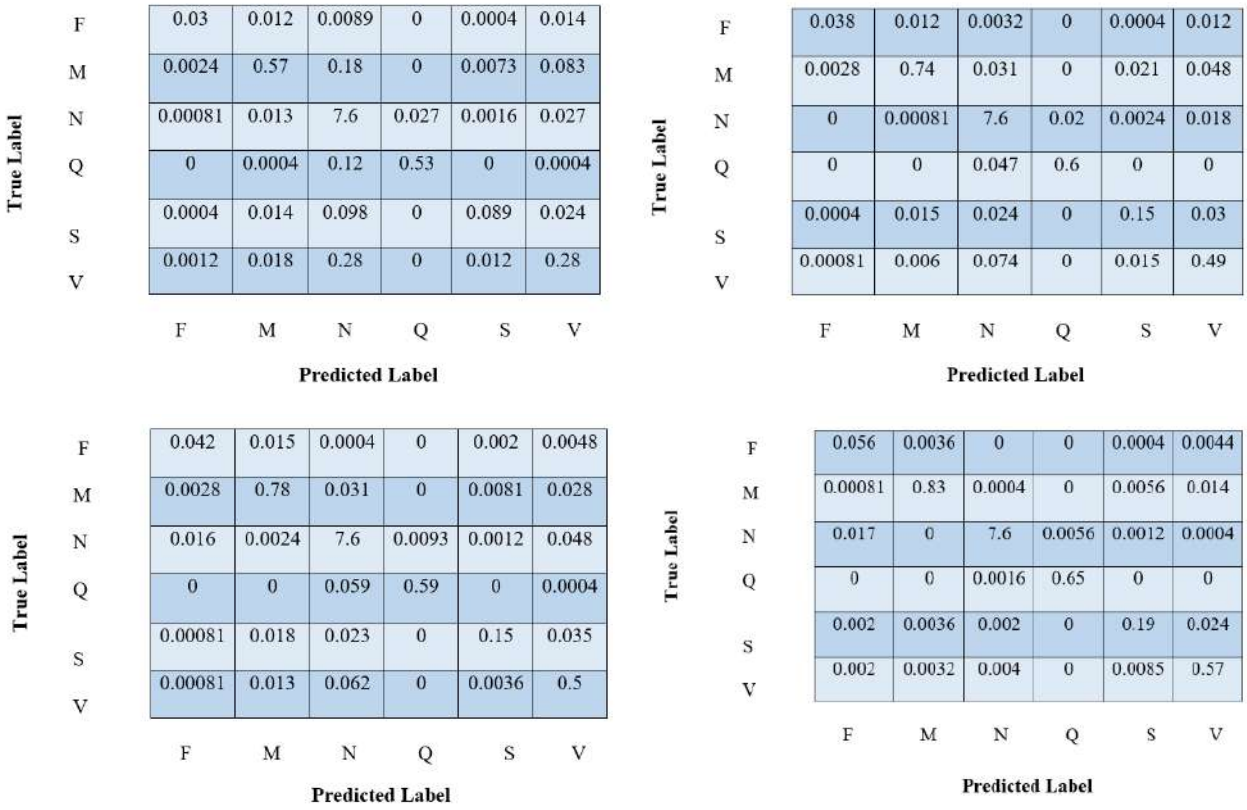
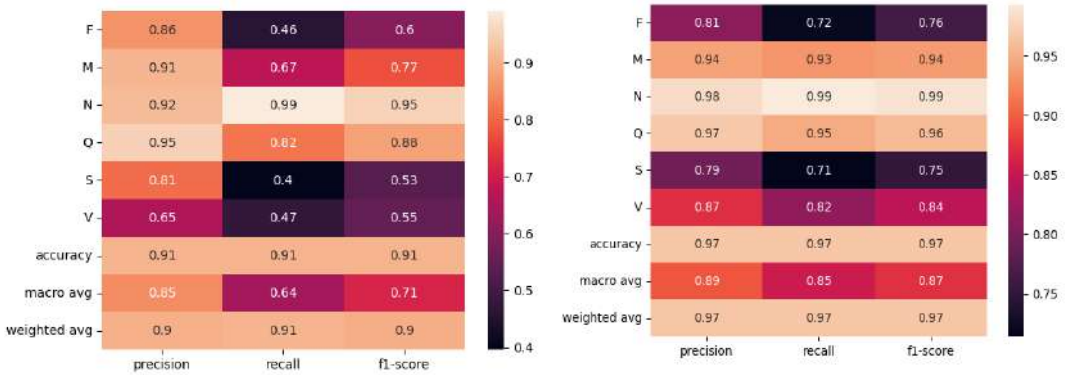


Figure 5.25: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

Classification Report



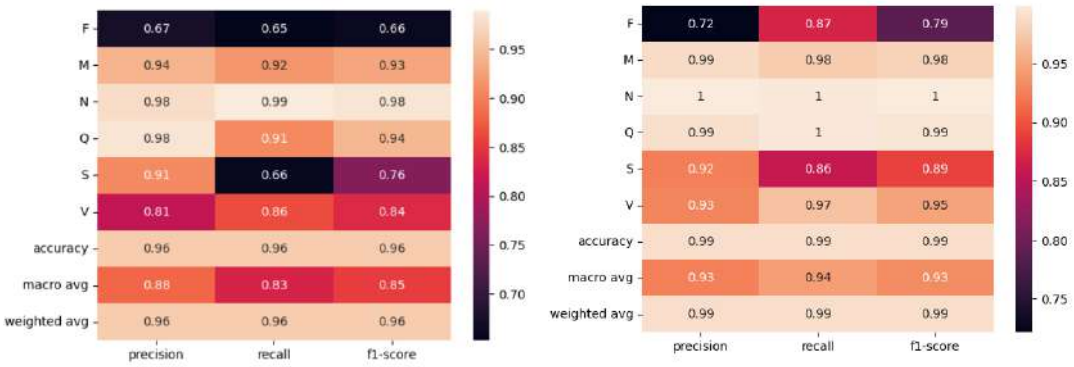


Figure 5.26: First row from left B0 size 32x32 and 64x64 and second row B1 size 64x64 and 128x128 respectively

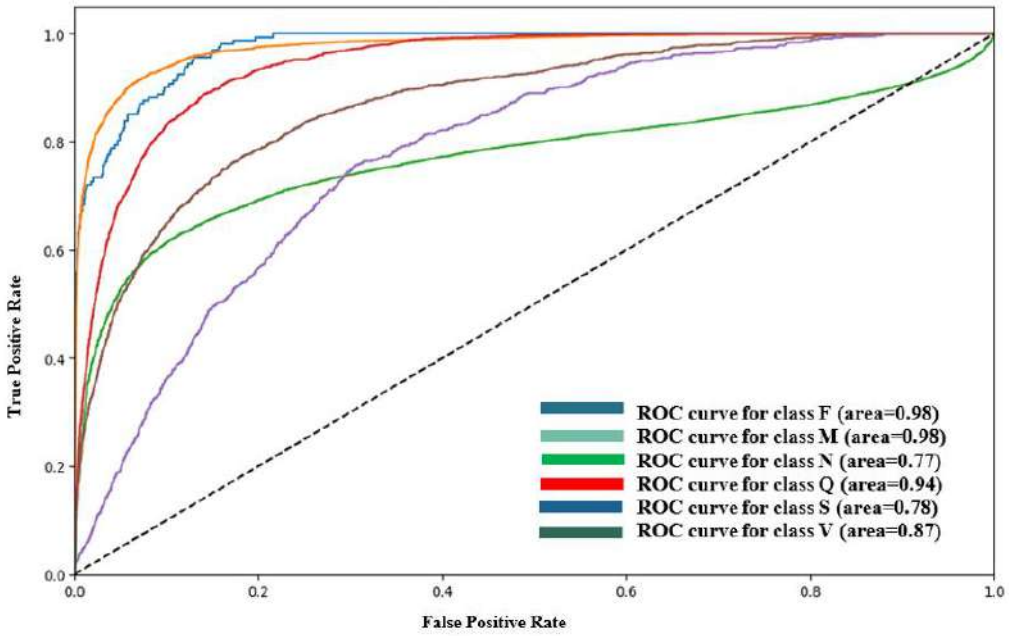


Figure 5.27: ROC curve of B0 of size 32x32

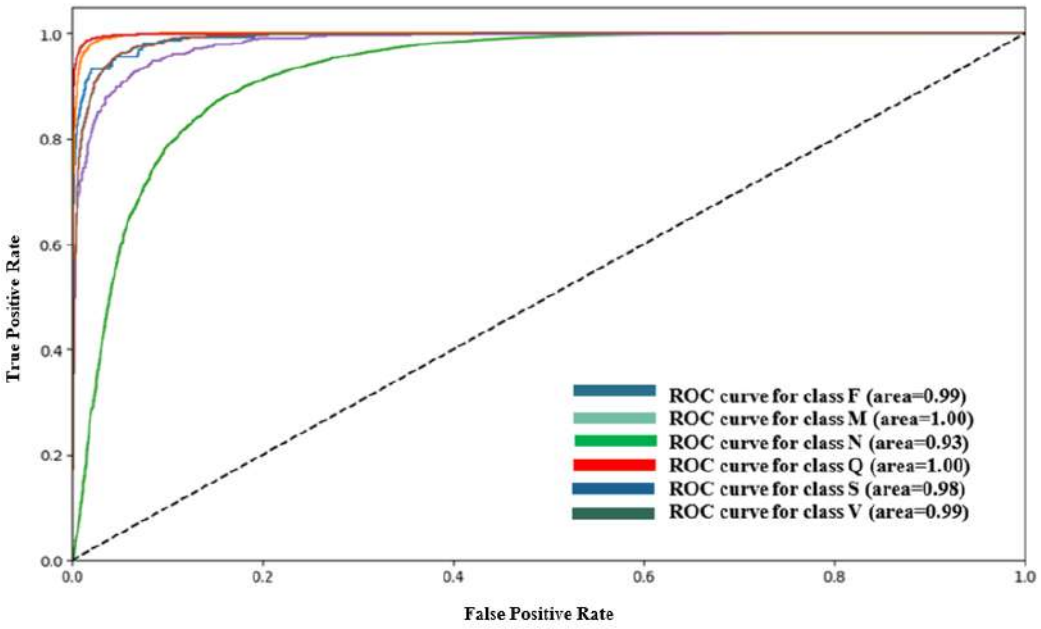


Figure 5.28: ROC curve of B0 of size 64x64

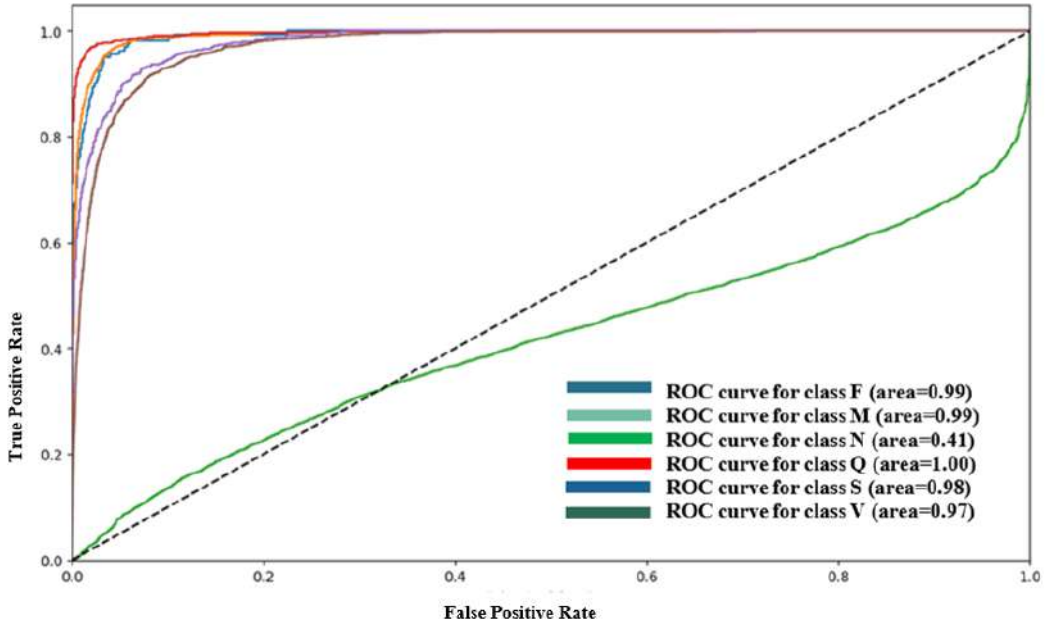


Figure 5.29: ROC curve of B1 of size 64x64

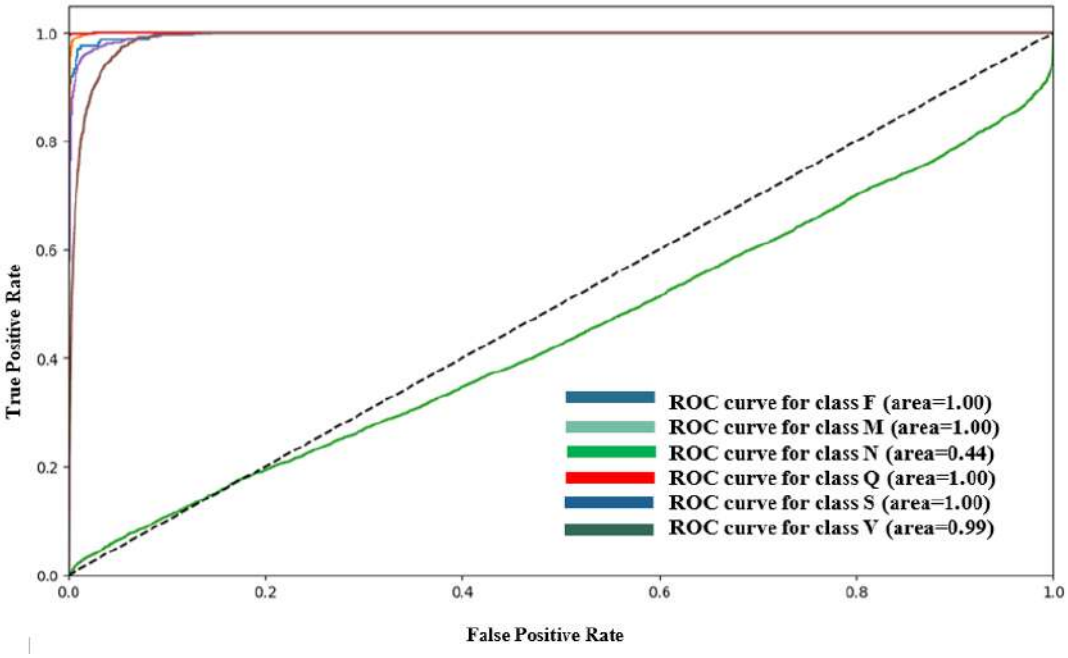


Figure 5.30: ROC curve of B1 of size 128x128

Learning Curves

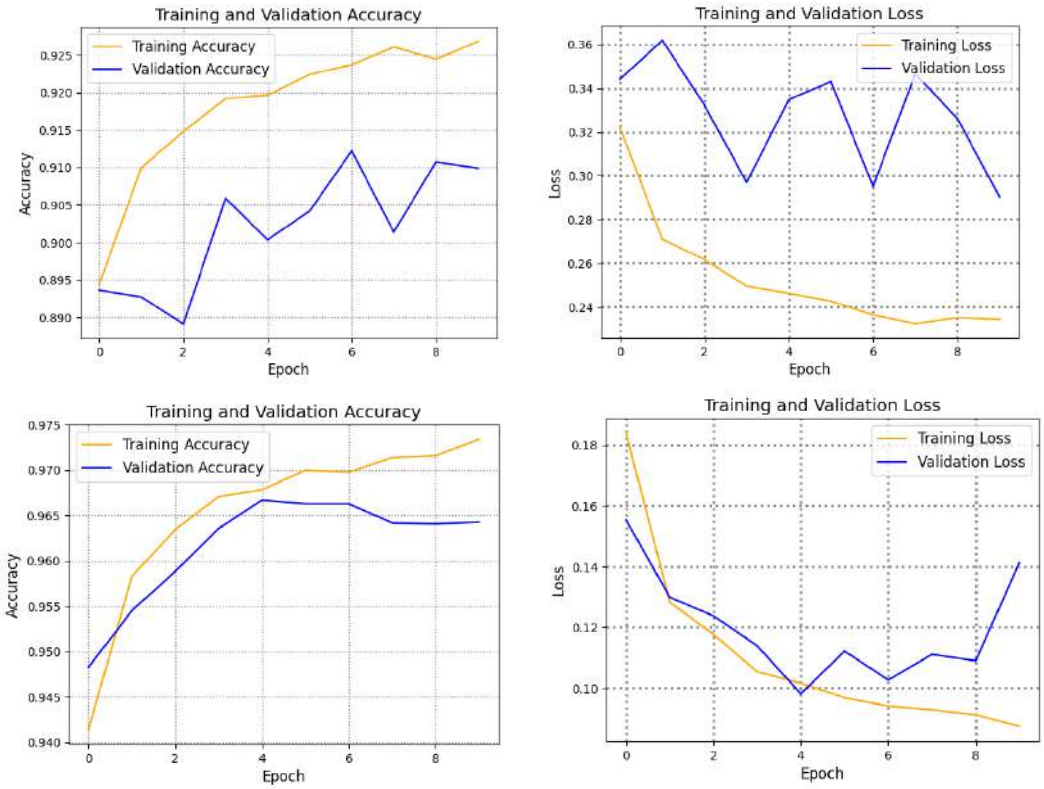


Figure 5.31: From left B0 size 32x32 and second row 64x64 respectively

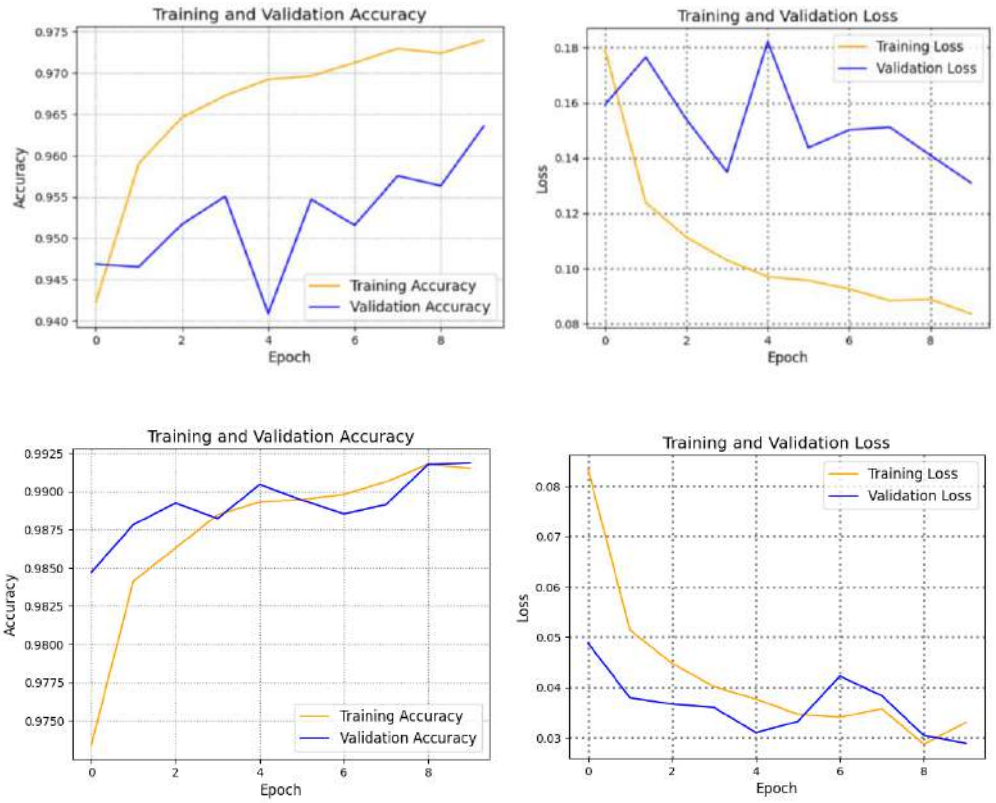


Figure 5.32: From left B1 size 64x64 and second row 128x128 respectively

5.7.2 Performance Analysis:

The confusion matrix is a valuable tool for evaluating the performance of a multiclass model by displaying the class-wise distribution of predictions. The diagonal elements represent correctly predicted samples, and the higher the values on the diagonal, the better the model's performance. The off-diagonal elements indicate misclassifications between classes, with higher values indicating more frequent misclassifications. A well-performing model will have high diagonal values and low off-diagonal values. If the off-diagonal values are high, further investigation is needed to improve the model's performance. The confusion matrix is used to generate a classification report that includes precision, recall, and F1 score. These metrics provide insights into the impact of class imbalance and overfitting on the model's performance. In the case of smaller sizes, the model may experience confusion due to blurring effects, resulting in similar classes being misclassified. However, as the size increases, the model learns complex features and patterns, leading to better predictions. This observation is supported by the learning curves, which show improved convergence and reduced overfitting with larger sizes, indicating a better fit of the model.

5.8 Method 2 Comparison Analysis:

The only difference here is the better accuracy and classification by the model. Compared to Method 1, this method has a good strategy for overfitting reduction as the

AUC got better. Learning curves are also comparatively better. Accuracy improved but as it is said earlier, the parameters are increased. A detailed comparison of method 2 is given below**:**

Comparison of V1 and V2 with Method 2:

Table 5.18: Comparison of EfficientNet V1 and V2 with Method 2

Model	**B0**		**B1**		**B0**		**B1**	
Version	**V1**				**V2**			
Image Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Accuracy (%)	89.0518	96.16	94.786	98.89	90.60	96.67	96.32	99.12
Testing Time (ms)	3.99	3.91	4.35	4.87	3.99	4.55	7.84	5.20
Training Time (minutes)	6.06	6.38	6.98	7.84	6.33	6.92	6.79	4.19
Trainable Parameters	962,423	962,423	962,423	962,423	962,423	962,423	962,423	962,423
Total Parameters	4,969,395	4,969,395	7,475,031	7,475,031	6,820,551	6,820,551	7,821,899	7,821,899

While comparing V1 with V2, we can observe that we got better accuracy with better computational cost. So, if the choice must be made one can opt for V2 instead of moving to B1 of V1 from B0 of V1.

Comparing Method 1 with Method 2 of EfficientNet

Method 1								
Model	**B0**		**B1**		**B0**		**B1**	
Version	**V1**				**V2**			
Image Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Accuracy (%)	86.13	93.95	93.14	98.15	87.93	95.44	95.62	99.16
Testing Time (ms)	4.15	4.51	4.15	5.12	4.07	4.27	4.15	5.36
Training Time (minutes)	5.878	6.898	6.828	7.96	6.425	6.796	6.741	7.706
Trainable Parameters	7,686	7,686	7,686	7,686	7,686	7,686	7,686	7,686
Total Parameters	4,014,658	4,014,658	6,520294	6,520,294	5,865,814	5,865,814	6,867,162	6,867,162

Method 2

Table 5.19: Method 1 and 2 comparisons for both versions of EfficientNet

Model	**B0**		**B1**		**B0**		**B1**	
Version	**V1**				**V2**			
Image Dimensions	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$	$32x32x1$	$64x64x1$	$64x64x1$	$128x128x1$
Accuracy (%)	89.0518	96.16	94.786	98.89	90.60	96.67	96.32	99.12
Testing Time (ms)	3.99	3.91	4.35	4.87	3.99	4.55	7.84	5.20
Training Time (minutes)	6.06	6.38	6.98	7.84	6.33	6.92	6.79	4.19
Trainable Parameters	962,423	962,423	962,423	962,423	962,423	962,423	962,423	962,423
Total Parameters	4,969,395	4,969,395	7,475,031	7,475,031	6,820,551	6,820,551	7,821,899	7,821,899

By adding a drop out, overfitting has reduced significantly. Method 2 has better Accuracy but, the parameters are now increased as new layers are being added. Whereas method 1 has poor ROC curves and by incorporating new layers it has been improved.

5.9 MobileNet Results and Comparison Analysis

5.9.1 Overall Timing Comparison:

Table 5.20: MobileNet -V1 Timing Results Comparison

Model-V1	**Size (Height × Width × channel)**	**Training Time (s)**	**Testing Time (s)**
V1	32×32×3	4195	39.5
	64 × 64×3	4304	41.5
	128 × 128×3	4500	43.2
	224 × 224×3	4895	48.5

Table 5.21: MobileNet-V2 Timing Results Comparison

Model-V2	Size (Height × Width × channel)	Training Time (s)	Testing Time (s)
V2	32×32×3	3650	37.4
	64 × 64×3	4120	37.8
	128 × 128×3	4460	39.2
	224 × 224×3	4790	53.5

Table 5.22: MobileNet-V3_Small Timing Results Comparison

Model-V2	Size (Height × Width × channel)	Training Time (s)	Testing Time (s)
V3	32×32×3	3855	37.3
	64 × 64×3	4001	39.5
	128 × 128×3	4293	41.6
	224 × 224×3	5201	50.8

Average Timing Analysis

Table 5.23: MobileNet-V1 Average Time Comparison

Model	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (Milli-seconds)
V1	32×32×3	6.99	1.59
	64 × 64×3	7.17	1.67
	128 × 128×3	7.52	1.74
	224 × 224×3	8.15	1.95

Table 5.24: MobileNet-V2 Average Time Comparison

Model	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (Milli-seconds)
V2	32×32×3	5.75	1.50
	64 × 64×3	6.86	1.52
	128 × 128×3	7.42	1.58
	224 × 224×3	7.94	2.15

Table 5.25: MobileNet-V3_Small Average Time Comparison

Model	Size (Height × Width)	Average Training Time per Epoch (minutes)	Average Testing Time per Image (Milli-seconds)
V3- Small	32×32×3	5.62	1.53
	64 × 64×3	6.66	1.61
	128 × 128×3	7.15	1.63
	224 × 224×3	8.66	2.01

5.9.2 Accuracy Comparison

Table 5.26: MobileNet -V1 Accuracy comparison

Model	Size (Height × Width)	Accuracy (%)
V1	32×32	99.65
	64 × 64	99.91
	128 × 128	99.90
	224 × 224	99.97

Table 5.27: MobileNet-V2 Accuracy comparison

Model	Size (Height × Width)	Accuracy (%)
V2	32×32	98.88
	64 × 64	99.86
	128 × 128	99.88
	224 × 224	99.92

Table 5.28: MobileNet-V3_Small Accuracy comparison

Model	Size (Height × Width)	Accuracy (%)
V3	32×32	92.85%
	64 × 64	99.81%
	128 × 128	99.90%
	224 × 224	99.97%

Table 5.29: Trainable Parameters comparison

Version	V1	V2	V3_Small
Parameters Size (MB)	16.23 MB	13.50 MB	9.57 MB
Trainable Parameters	4,231,976	3,504,872	2,542,856
Total Parameters	4,253,864	3,538,984	2,554,968

5.9.3 Model size, computational cost, and accuracy Analysis:

The combined analysis, considering both the comparison of computational parameters (computation time, expenses, model size) and the training/testing times**:**

The results presented indicate the need to strike a balance between various parameters when selecting a model. By increasing the model size, computational time and expenses tend to rise, but with the potential for improved accuracy. There is a trade-off between computational costs and accuracy, and it becomes crucial to carefully consider these factors.

Comparing MobileNet-V1, MobileNet-V2, and MobileNet-V3, MobileNet-V2 emerges as a balanced choice. It offers a compromise between computational cost, time, and model size. With a smaller size of 13.50 MB and a lower number of trainable parameters compared to MobileNet-V1, MobileNet-V2 provides efficient resource utilization. MobileNet-V2 achieves high accuracy across all image sizes, with the largest size (224x224) achieving an impressive accuracy of 99.92%.

MobileNet-V3_Small, with its lower computational cost and potentially faster processing time, presents an alternative option. It has a smaller model size of 9.57 MB and lower trainable parameters compared to MobileNet-V2. Although it sacrifices a bit of accuracy, particularly for smaller image sizes, it offers acceptable accuracy levels for most practical applications.

On the other hand, MobileNet-V1 boasts higher accuracy, especially for larger image sizes, but comes with a larger model size of 16.23 MB and higher computational requirements. MobileNet-V1 may be preferred when maximum accuracy is critical and computational constraints are not a significant concern.

Considering the training and testing times, MobileNet-V2 demonstrates faster training times compared to MobileNet-V1 and MobileNet-V3, across various image sizes. MobileNet-V2 consistently outperforms the other models in terms of training efficiency. When it comes to testing time, MobileNet-V1 and MobileNet-V2 have similar performance, with MobileNet-V2 slightly edging out in some cases. MobileNet-V3 generally exhibits slightly higher testing times.

Conclusion

Overall, MobileNet-V2 appears as the most balanced and computationally efficient choice among the three models. It offers a compromise between accuracy, computational costs, time, and model size. MobileNet-V3_Small can be a suitable alternative if a slight decrease in accuracy is acceptable in exchange for lower computational requirements. MobileNet-V1, with its larger model size and computational demands, is preferred when maximum accuracy is crucial, particularly for larger image sizes.

5.10MobileNetV1 (RGB)

5.10.1Performance Metrics Version 1

Confusion Matrix

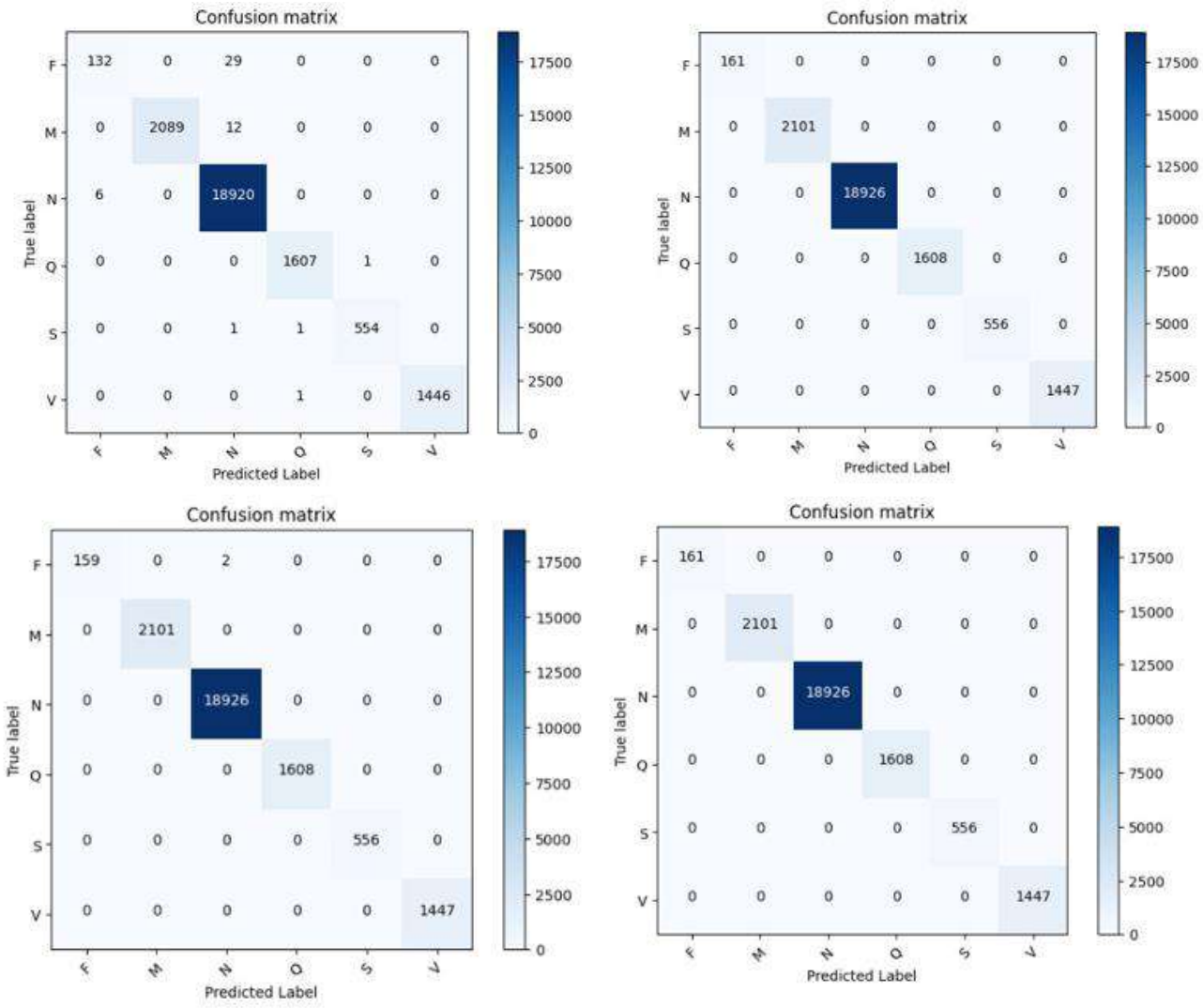


Figure 5.33: First column from left V1 size 32×32 and 64×64 and second column V1 size 128×128 and 224×224 respectively

Classification Report:

	precision	recall	f1-score	support
F	0.96	0.82	0.88	161
M	1.00	0.99	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	0.99	0.97	0.98	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.34: MobileNetV1 32x32

	precision	recall	f1-score	support
F	1.00	0.99	0.99	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.35: MobileNetV1 64x64

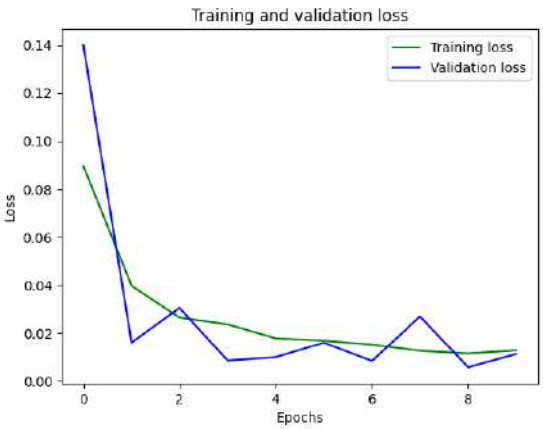
	precision	recall	f1-score	support
F	1.00	1.00	1.00	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

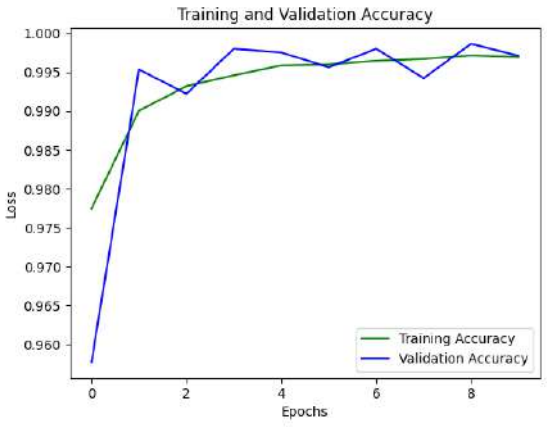
Figure 5.36: MobileNetV1 128x128

	precision	recall	f1-score	support
F	1.00	1.00	1.00	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.37: MobileNetV1 224x224

Learning Curves:





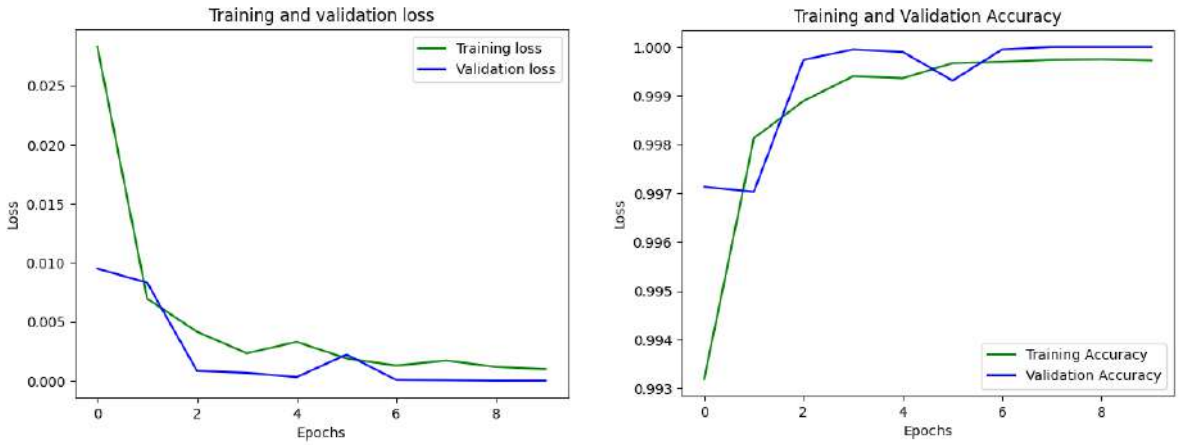


Figure 5.38: Top row size 32×32, Bottom row size 64×64

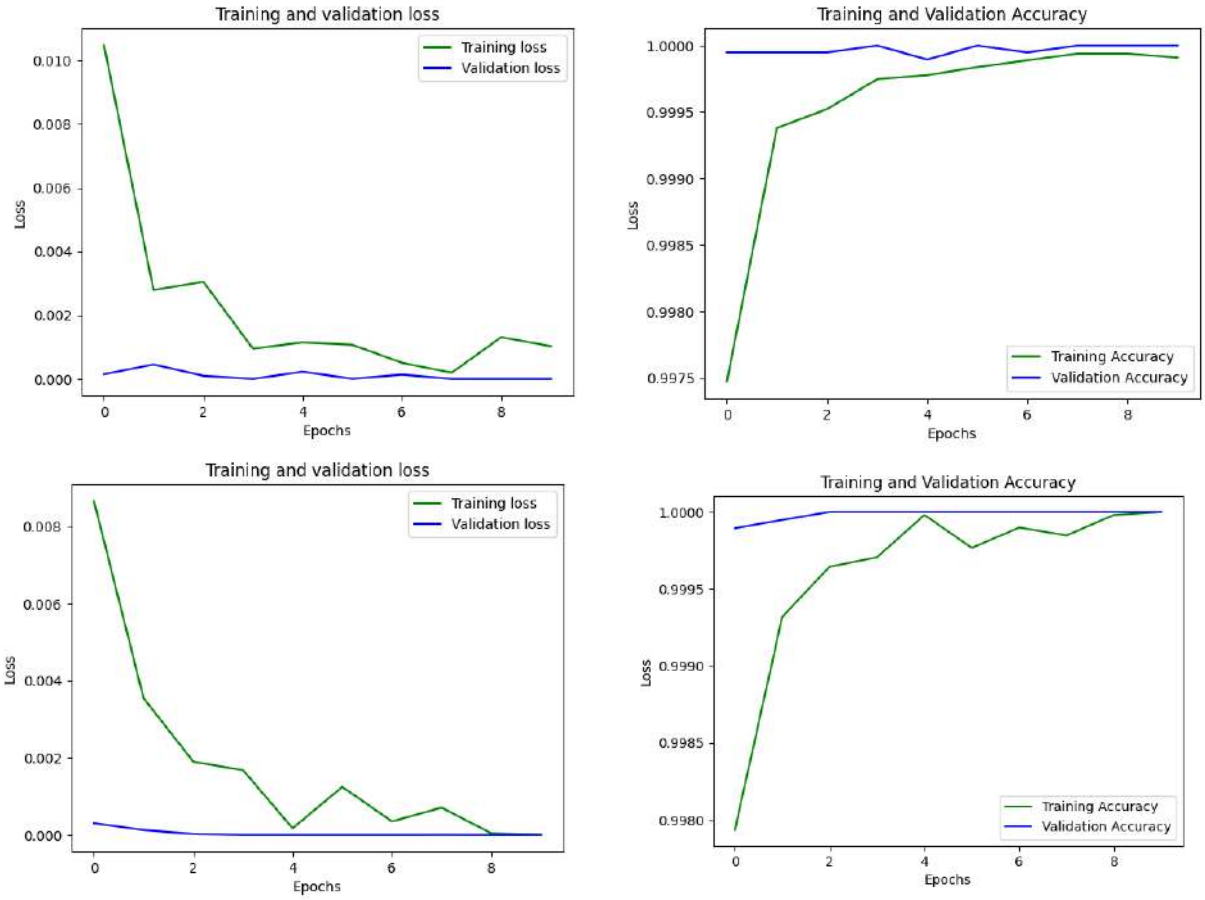


Figure 5.39: Top row size 128×128, Bottom row size 224×224

5.11 MobileNetV2 (RGB)

5.11.1 Performance Metrics Version: 2

Confusion matrix

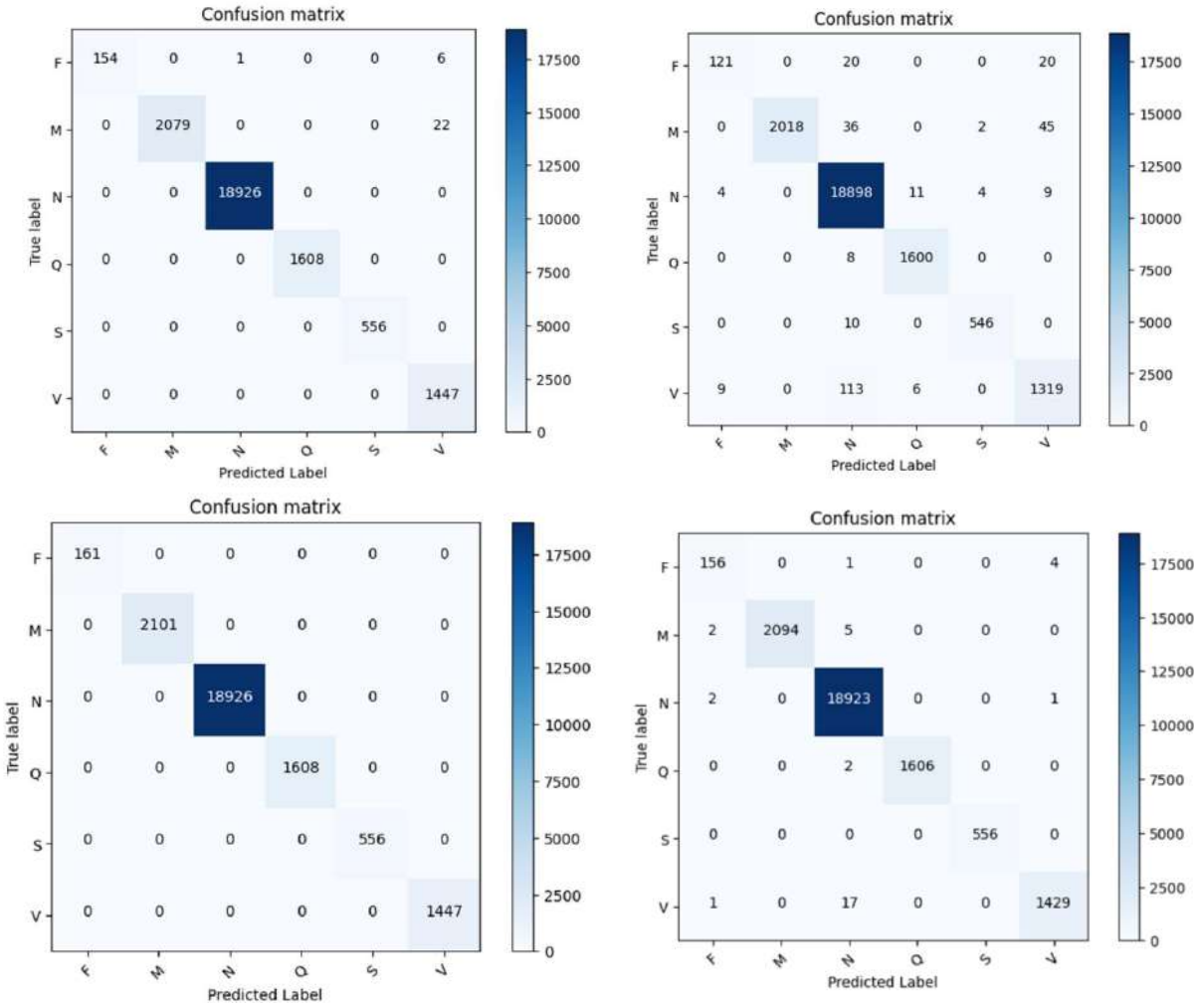


Figure 5.40: First column from left V2 size 32×32 and 64×64 and second column V2 size 128×128 and 224×224 respectively

Classification Report:

	precision	recall	f1-score	support
F	0.90	0.75	0.82	161
M	1.00	0.96	0.98	2101
N	0.99	1.00	0.99	18926
Q	0.99	1.00	0.99	1608
S	0.99	0.98	0.99	556
V	0.95	0.91	0.93	1447
accuracy			0.99	24799
macro avg	0.97	0.93	0.95	24799
weighted avg	0.99	0.99	0.99	24799

Figure 5.41: MobileNetV2 32x32

	precision	recall	f1-score	support
F	0.97	0.97	0.97	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	0.99	0.99	1447
accuracy			1.00	24799
macro avg	0.99	0.99	0.99	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.42: MobileNetV2 64x64

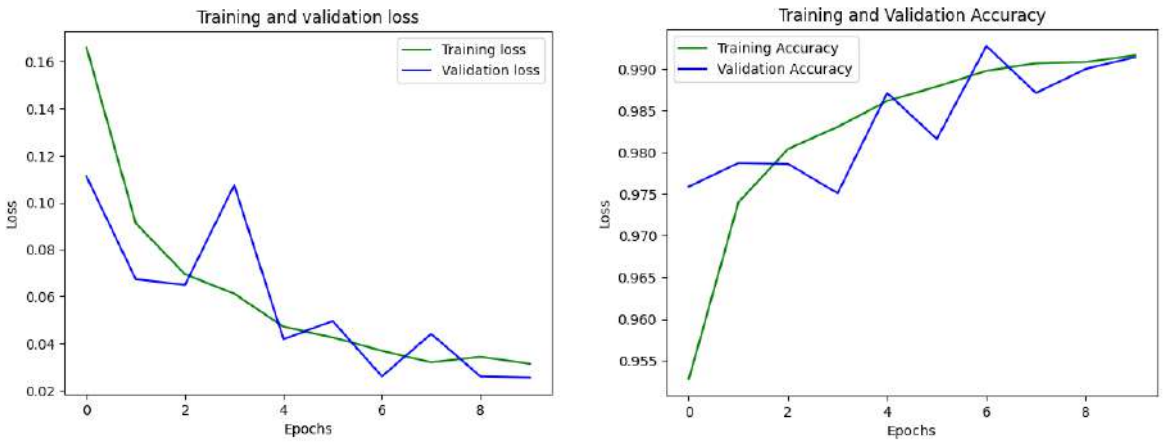
	precision	recall	f1-score	support
F	1.00	0.96	0.98	161
M	1.00	0.99	0.99	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	0.98	1.00	0.99	1447
accuracy			1.00	24799
macro avg	1.00	0.99	0.99	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.43: MobileNetV2 128x128

	precision	recall	f1-score	support
F	1.00	1.00	1.00	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.44: MobileNetV2 224x224

Learning Curves:



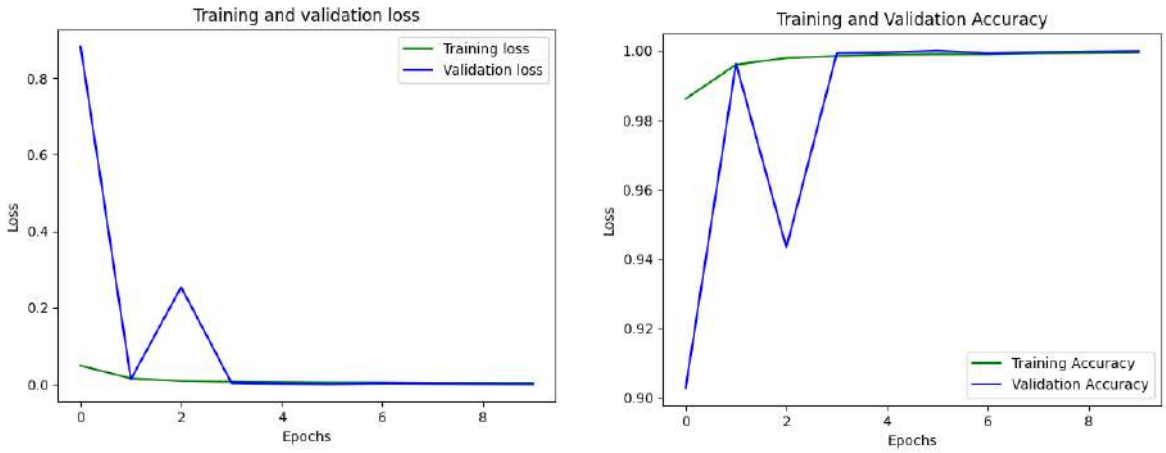


Figure 5.45: Top row size 32×32, Bottom row size 64×64

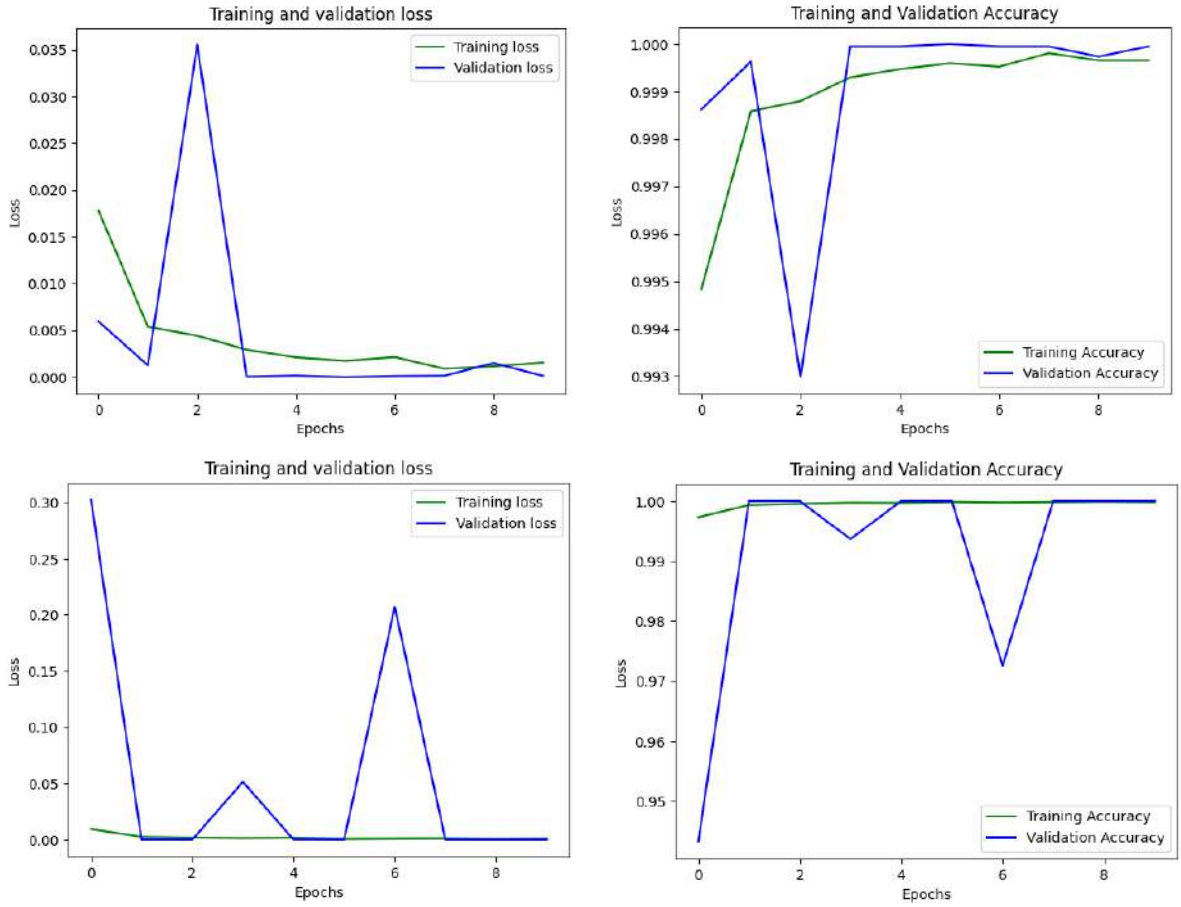


Figure 5.46: Top row size 128×128, Bottom row size 224×224

5.12 MobileNetV3_Small (RGB)

5.12.1 Performance Metrics Version: 3

Confusion matrix

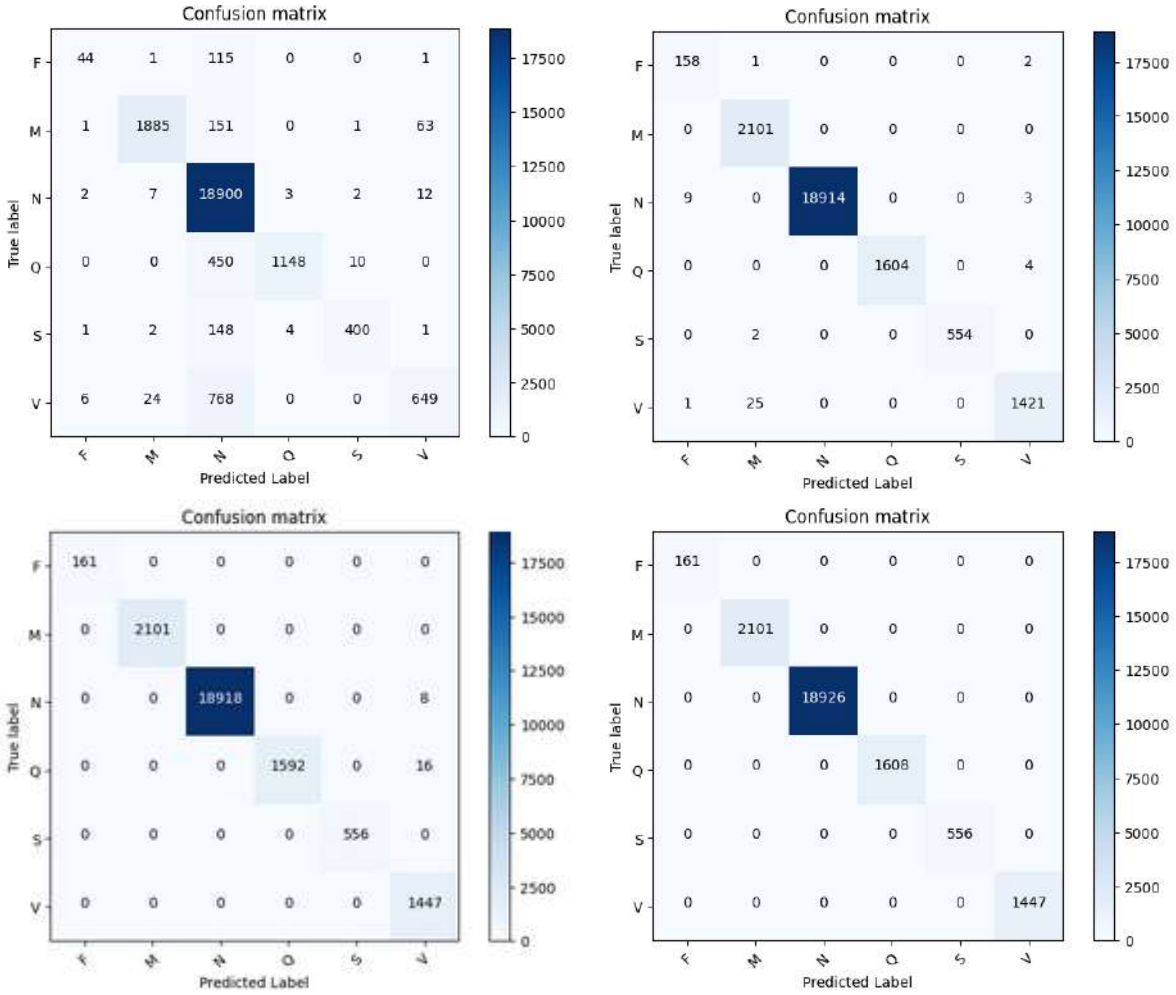


Figure 5.47: First column from left V2 size 32×32 and 64×64 and second column V2 size 128×128 and 224×224 respectively

Classification Report:

	precision	recall	f1-score	support
F	0.81	0.27	0.41	161
M	0.98	0.90	0.94	2101
N	0.92	1.00	0.96	18926
Q	0.99	0.71	0.83	1608
S	0.97	0.72	0.83	556
V	0.89	0.45	0.60	1447
accuracy			0.93	24799
macro avg	0.93	0.68	0.76	24799
weighted avg	0.93	0.93	0.92	24799

Figure 5.48: MobileNetV3_Small 32x32

	precision	recall	f1-score	support
F	0.94	0.98	0.96	161
M	0.99	1.00	0.99	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	0.99	0.98	0.99	1447
accuracy			1.00	24799
macro avg	0.99	0.99	0.99	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.49: MobileNetV3_Small 64x64

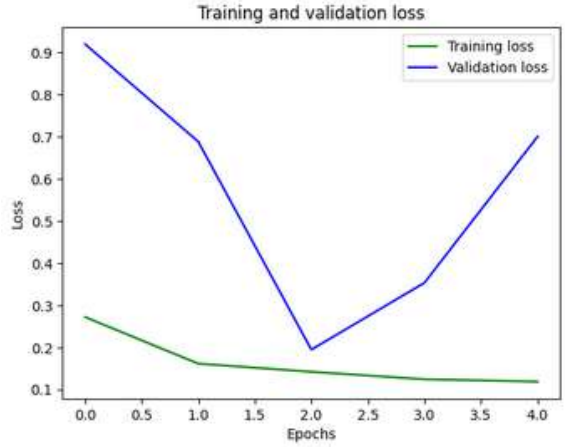
	precision	recall	f1-score	support
F	1.00	1.00	1.00	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	0.99	0.99	1608
S	1.00	1.00	1.00	556
V	0.98	1.00	0.99	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

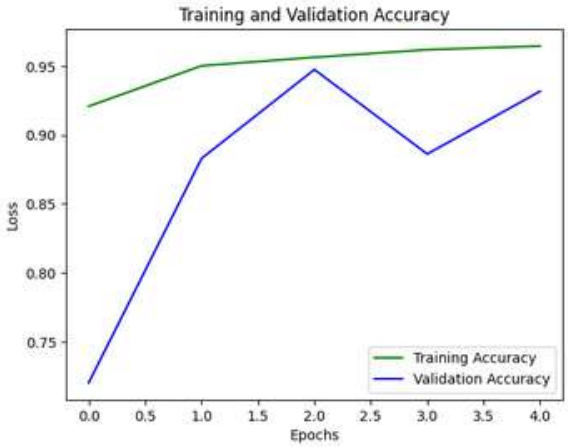
Figure 5.50: MobileNetV3_Small 128x128

	precision	recall	f1-score	support
F	1.00	1.00	1.00	161
M	1.00	1.00	1.00	2101
N	1.00	1.00	1.00	18926
Q	1.00	1.00	1.00	1608
S	1.00	1.00	1.00	556
V	1.00	1.00	1.00	1447
accuracy			1.00	24799
macro avg	1.00	1.00	1.00	24799
weighted avg	1.00	1.00	1.00	24799

Figure 5.51: MobileNetV3_Small 224x224

Learning Curves:





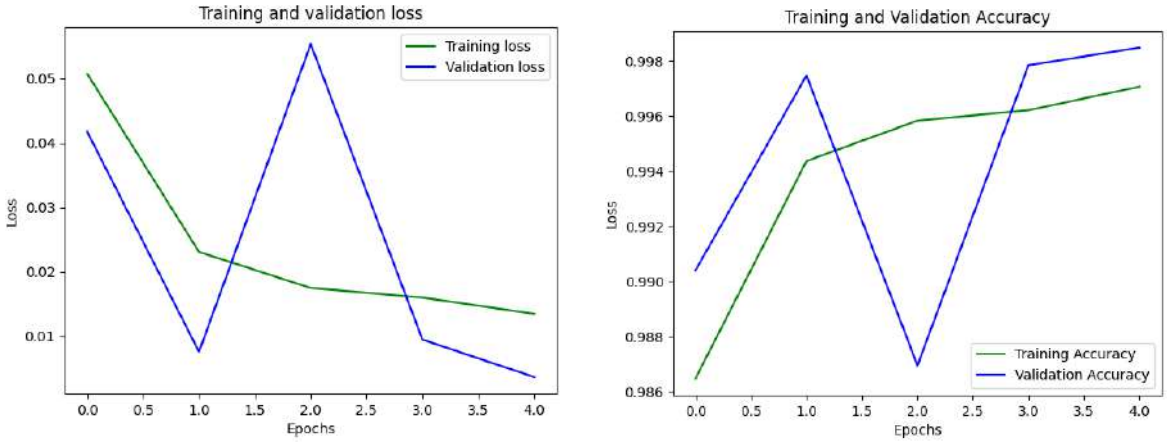


Figure 5.52: Top row size 32×32, Bottom row size 64×64

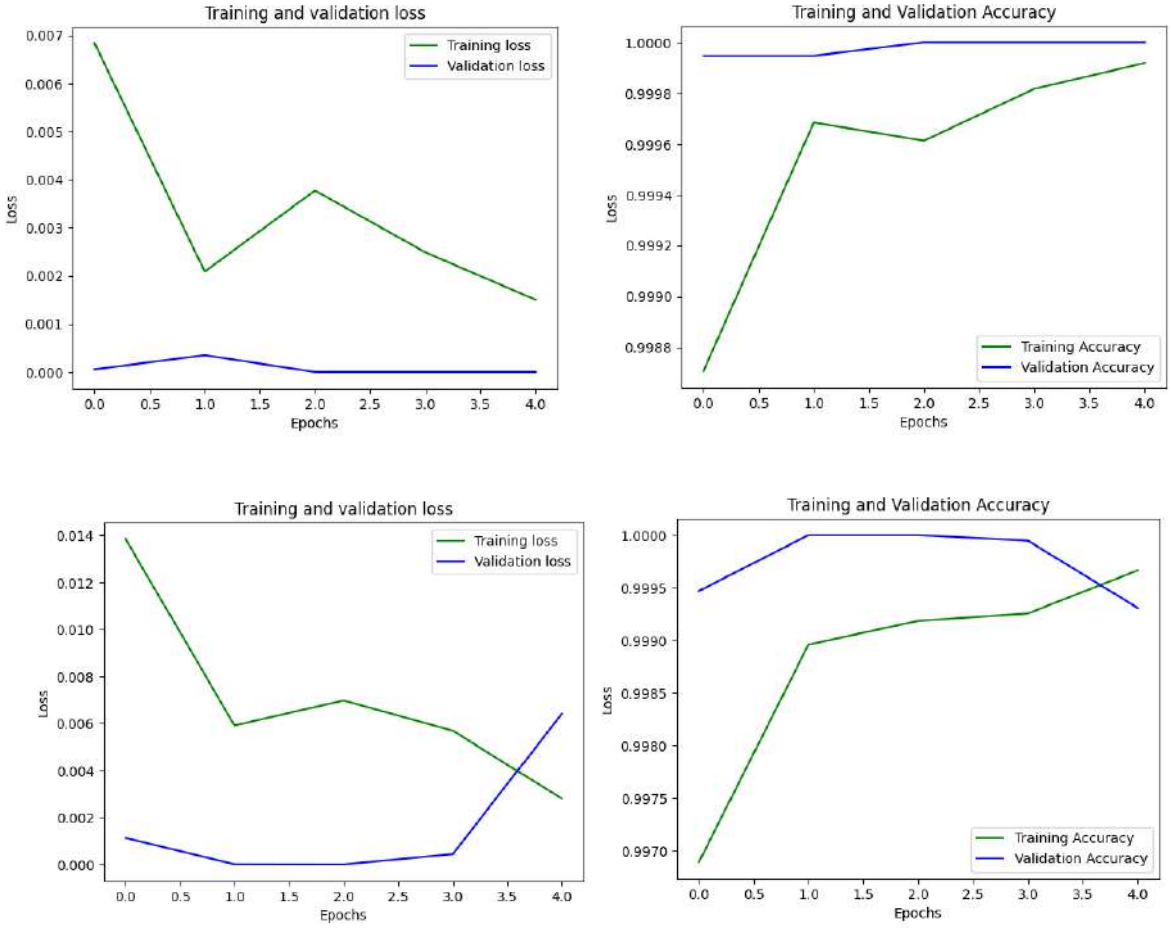


Figure 5.53: Top row size 128×128, Bottom row size 224×224

5.13 Proposed System Comparison with state-of-art Models

5.14 Proposed Model Results

AlexNet, ResNet18, VGG16, EfficientNet are implemented for comparison with state of art models.

Table 5.30: Hyperparameters

Batch Size	Learning Rate	Epochs	Optimizer	Criterion
16	0.001	10	Adam	Cross Entropy Loss

EfficientNet here is implemented using Method -2 discussed in this work as it got better results.

Table 5.31: EfficientNet B0 V1 Results

Model	Size (Height xWidth)	Average Training Time per epoch (minutes)	Average Testing Time per image (ms)	Accuracy (%)	Trainable Parameters	Parameters Size (MB)	Total Size (MB)
EfficientNet B0-V1	$186x186$	9.538	5.766	99.57	962,423	18.96	178.10

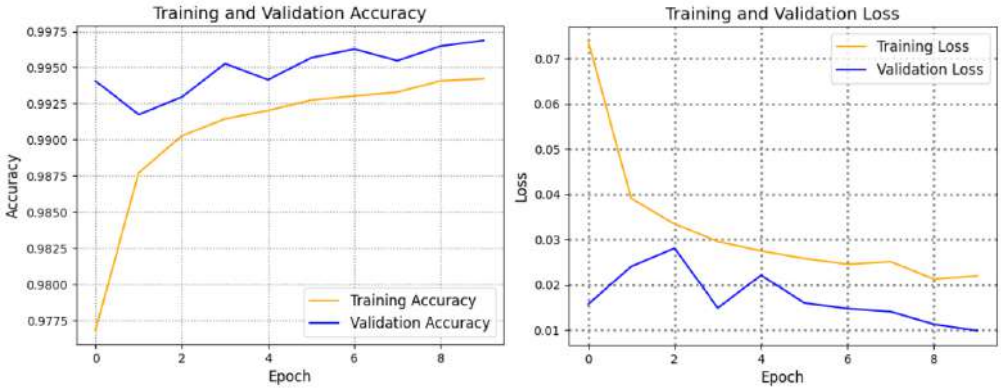


Figure 5.54: Learning curves of EfficientNet B0 Version 1

Performance Metrics

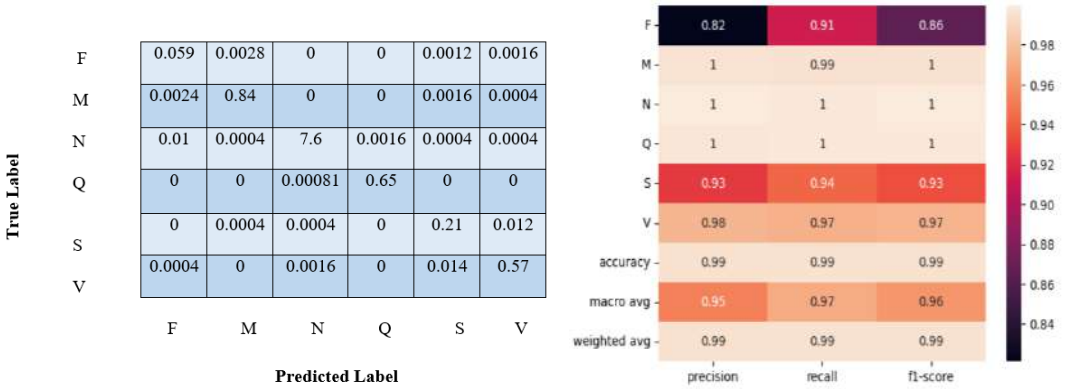


Figure 5.55: Classification Report (on right) and Confusion Matrix (on left)

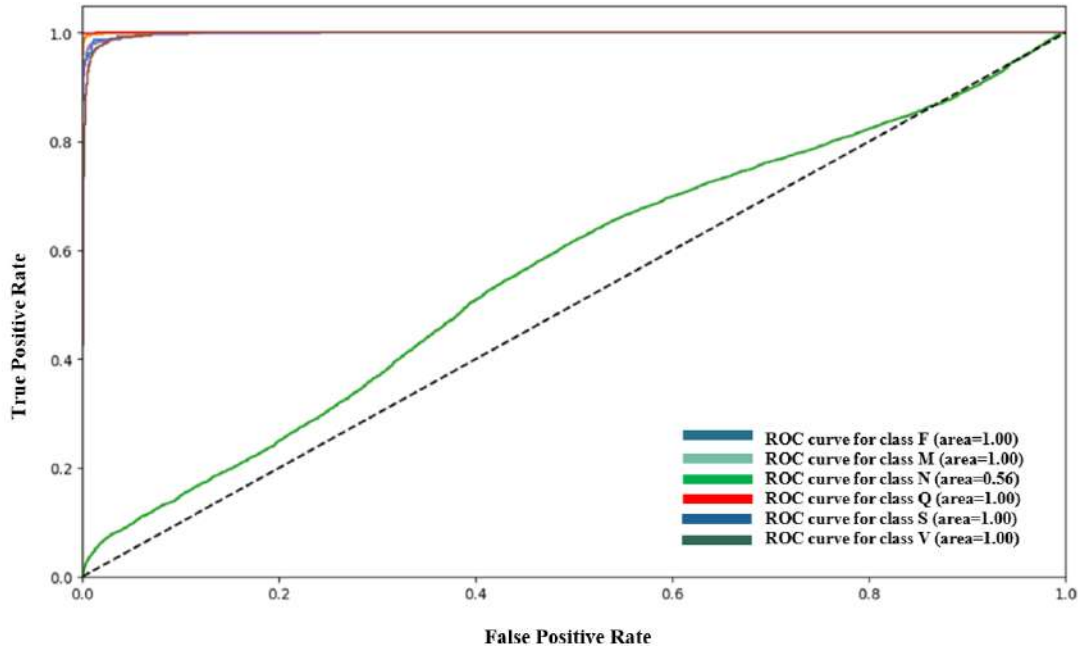
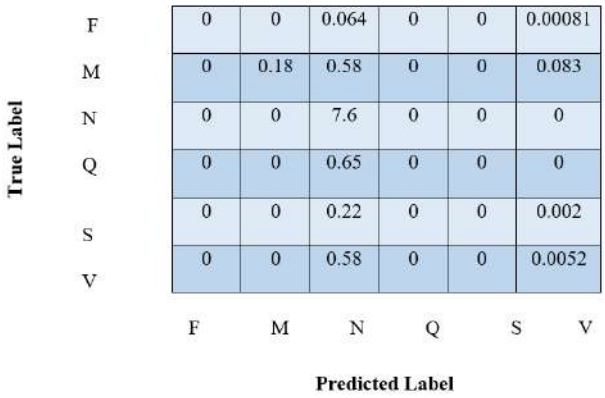


Figure 5.56: ROC Curve

Table 5.32: AlexNet Results

Model	Size (HeightxWidth)	Average Training Time per epoch (mins)	Average Testing Time per image (ms)	Accuracy (%)	Trainable Parameters	Parameters Size (MB)	Total Size (MB)
AlexNet	$186x186$	6.86	4.51	78.2	24,582	217.49	223.09



	precision	recall	f1-score	support
0	0.00	0.00	0.00	161
1	1.00	0.22	0.36	2101
2	0.78	1.00	0.88	18926
3	0.00	0.00	0.00	1608
4	0.00	0.00	0.00	556
5	0.06	0.01	0.02	1447
accuracy			0.78	24799
macro avg	0.31	0.20	0.21	24799
weighted avg	0.69	0.78	0.70	24799

Figure 5.57: Classification Report (on right) and Confusion Matrix (on left)The classification report of AlexNet has shown miss-classification for class F, class, Q, and class S. The model is unable to detect these classes**.**

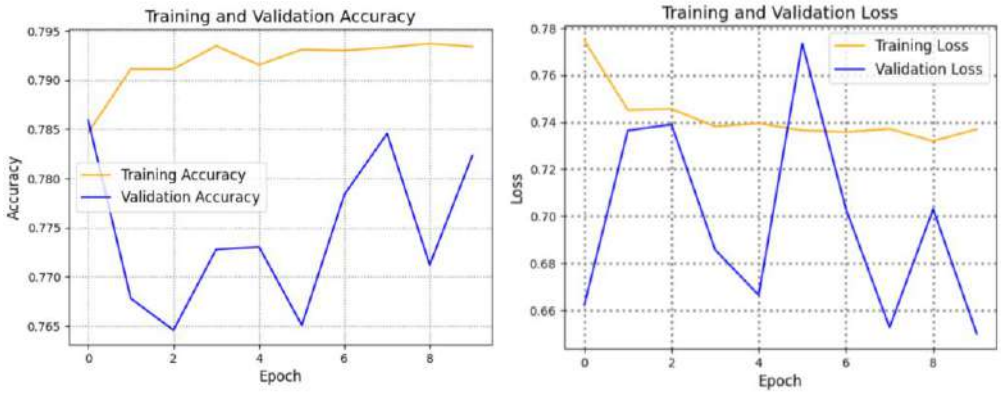


Figure 5.58: Learning Curves

5.14.1 Architecture differences

AlexNet has a different architecture compared to other CNNs like EfficientNet, ResNet18, and VGG16. It has fewer layers and may not have as much capacity to learn complex patterns from the dataset. It's possible that the other CNN architectures are better suited to the dataset in terms of their capacity to capture the patterns present in the ECG signal images in the dataset used for this work.

Table 5.33: ResNet18 Results

Model	Size (HeightxWidth)	Average Training Time per epoch (mins)	Average Testing Time per image (ms)	Accuracy (%)	Trainable Parameters	Parameters Size (MB)	Total Size (MB)
ResNet 18	$186x186$	7.97	5.20	99.33	3,078	42.62	95.62

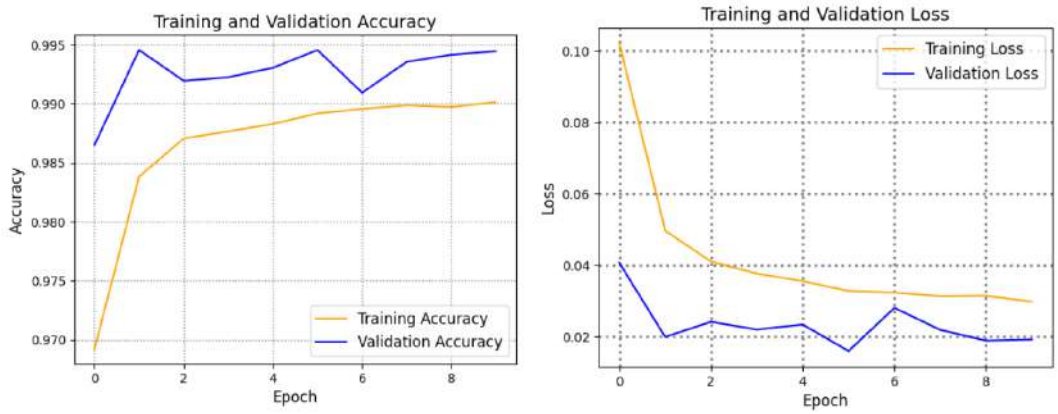


Figure 5.59: Learning Curves

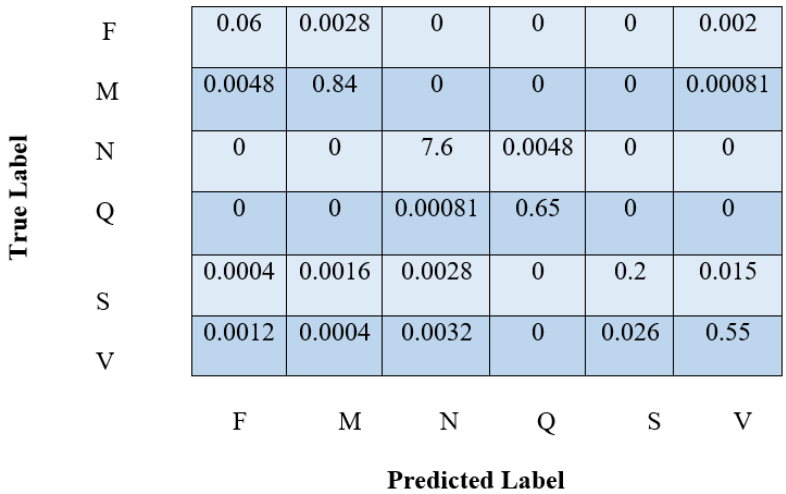


Figure 5.60: Confusion Matrix

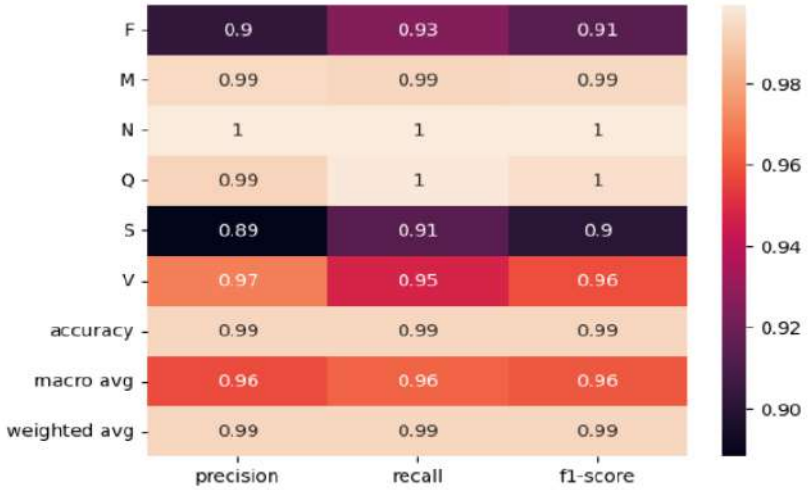


Figure 5.61: Classification Report

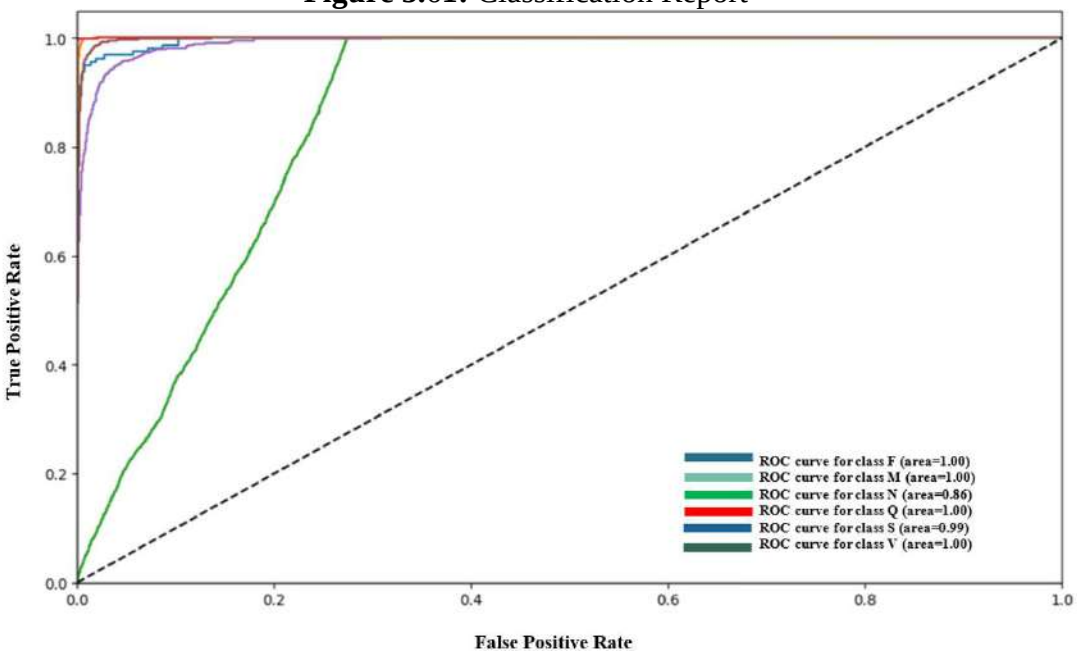


Figure 5.62: ROC Curve

Table 5.34: VGG16 Results

Model	Size (HeightxWidth)	Average Training Time per epoch (mins)	Average Testing Time per image (ms)	Accuracy (%)	Trainable Parameters	Parameters Size (MB)	Total Size (MB)
VGG16	186x186	12.81	8.024	98.70	24,582	512.25	661.95

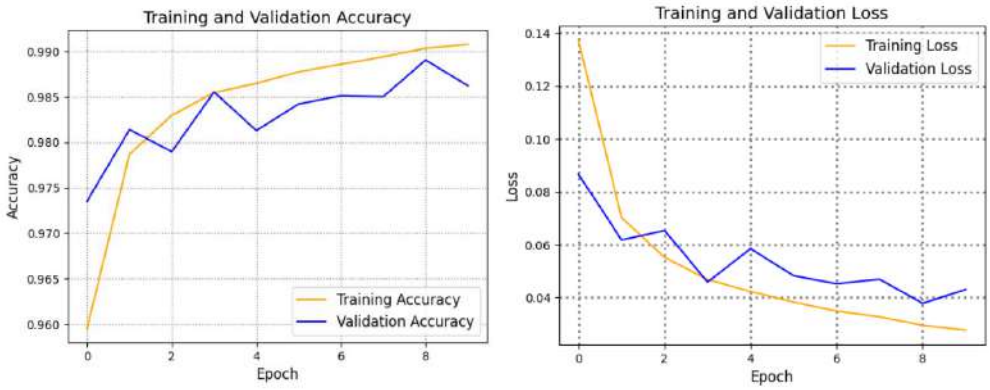


Figure 5.63: Learning Curves

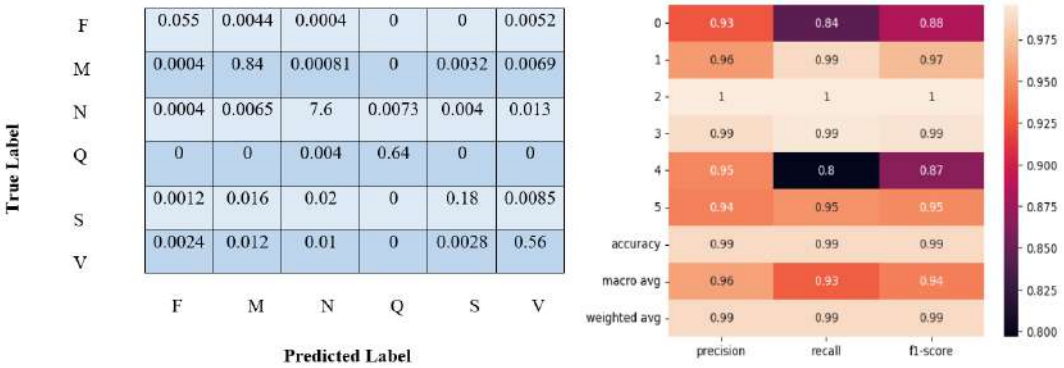


Figure 5.64: Confusion Matrix (on left) and Classification report (on right)

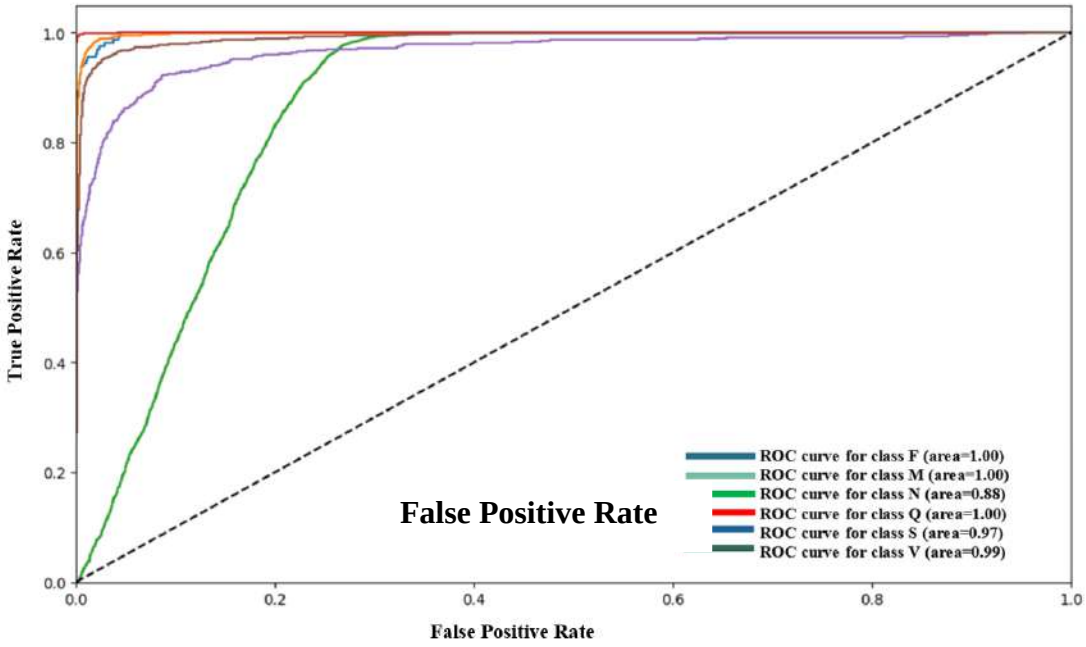


Figure 5.65: ROC Curve

5.15 Comparison Table:

Table 5.35: Comparison of proposed methodology with existing models

Specifications	Paper 1[4] [29]	Paper 2[5] [25]	Paper 3[6] [38]	Proposed
Goal	Arrhythmia Classification	Arrhythmia Detection	Arrhythmia Detection	Arrhythmia Classification
Approach	Transfer learning with fine tuning	Transfer learning as a feature extractor	Transfer learning with fine tuning	Transfer Learning as a feature extractor
Dataset	MIT BIH Arrhythmia Database	MIT BIH Arrhythmia Fused with Real Time Data	MIT BIH Arrhythmia Database	MIT BIH Arrhythmia Database and PTBDB
Classes	N S V F Q	N S V F Q	N S V F	F M N Q S V
Model	• vgg16 • vgg19 • resnet50 • resnet50 V2 • EfficientNet B0 • V1,ConvLSTM • Xception • Inception V3 • Inception ResNet V2	• ResNet50 • AlexNet • SqueezeNet	• ResNet18	• AlexNet • ResNet18 • Vgg16 • EfficientNet B0-V1
Channels	3	-	1	1
Accuracy (%)	99.20, 99.20, 99.40, 97.60, 96.20, 96.15, 94.40, 85.60, 48.60	91 , 98.8, 90.08	90.8	78.2 99.33 98.70 99.57 respectively
Scope	Misclassification in ConvLSTM	Computational Complexities	Lack of standard classes reporting	Imbalance

[4] ECG heartbeat classification using Wavelet transform and different Neural network Architectures.
[5] ECG Classification for Detecting ECG Arrhythmia Empowered with Deep Learning Approaches
[6] ECG heartbeat classification using deep transfer learning with convolutional neural network and STFT technique.

5.15.1 Analysis:

If we make a comparison of our proposed models (AlexNet, ResNet18, VGG16, and EfficientNet-B0 version 1) with implementations mentioned in the table above, we can conclude that our ResNet18 has out-performed paper 3 in the table. While EfficientNet B0 Version 1 out-performed paper 1. Moreover, for the classification of N, V, S, F, and Q classes the models implemented in this work are a better choice which can further be used to build hardware.

The hardware implementations given in the literature review chapter can be implemented using these proposed models which give better accuracy. Further analysis can be performed which can save computational cost, is more reliable and can be transformed into a lightweight, portable ECG Arrhythmia Classifier.

Chapter 6

Conclusion

6.1 Conclusion

In this work, multiple Convolutional Neural Networks are implemented in the search of finding a hardware friendly architecture. CNN Architecture is selected which showed very good accuracy without much compromise on the computational ability. Techniques have been mentioned in the prior section that how EfficientNet Method 2 showed better results than Method 1. While our proposed models outperformed other state-of-art models. The highest accuracy 99.57% is achieved by a model is of EfficientNet B0 V1 of size 186x186 whereas ResNet18 showed 99.33% accuracy having size 186x186. While EfficientNet V2 of size 128x128 showed 99.12% and 99.16% accuracy for Method-1 and Method-2 respectively. At this point accuracy is not the only concern but a bigger concern is the computational cost, parameters, and model size. ResNet18 has lowest trainable parameters which are only 3,078. But time it took is slightly more than most of other models implemented. The lowest time taken by a model to train, and test is EfficientNet B0 V1 of 32x32 size which only took training time of 5.876 minutes on average. While EfficientNet B0 V1 Method 1 with 64x64 has 3.91 average testing time. As the details of all the models are given in this report, one can decide to move forward with the most suitable model for their environment and carry this work further. Due to limited resources in real scenarios, these computationally inexpensive models can be implemented on hardware.

While MobileNet architecture that is being trained with fine tuning has proved to be fruitful. The accuracy achieved is remarkable. MobileNet V3_small has got better accuracy than other versions as well as it has provided less computational time. On the other hand, MobileNet V1 has larger model size and computational requirements.

6.2 Limitations

The main limitation of this work is data imbalance, it would be better to come up with a data augmentation technique or build ECG Images from scratch which must not compromise on information pruning and leads to better results and reduction of overfitting unlike in this work. More sizes can be tested with varying batch sizes and trained for higher number of epochs as the GPU resources were limited in this work, so we only trained for 10 epochs. Other techniques like scheduler can be incorporated whereas one can also modify layers or vary dropout probability to observe the behavior. In future, MobileNet can be implemented on grayscale images for analyzing performance.

6.3 Challenges and Future Direction

In future, a decision can be made on which classifier is best suited for the application. Since there is a trade-off between accuracy and size. If the goal is to design a hardware or propose a hardware architecture, EfficientNet and MobileNet are more suitable for that. With one cannot deny the fact that different applications have different architecture more suited to them. The inconsistency of the classes in MIT BIH Arrhythmia needs to

be studied further. Data imbalance needs to be addressed. Data augmentation can be performed in a more sophisticated way as ECG Signal is a crucial signal, and information can easily be pruned.

This work can be extended to Hardware and Software Co-Design which entails compressing and compiling the model into a set of executables. The Deep Learning Processing Unit (DPU) can be then implemented on the FPGA. Finally, the model can be deployed on the FPGA for inference.

Appendix A

Grayscale Resized images: The images are resized and converted to single channel grayscale. The blurred effect is due to small resolution of an image.

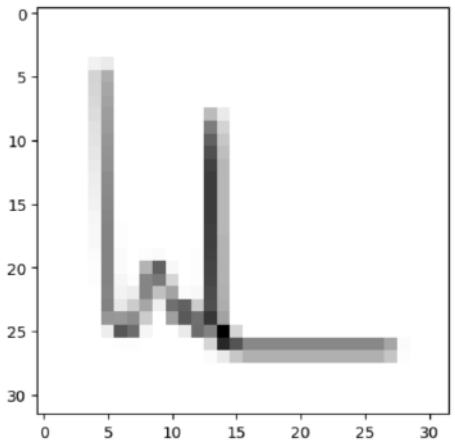


Figure A.7.1: Image resolution 32x32

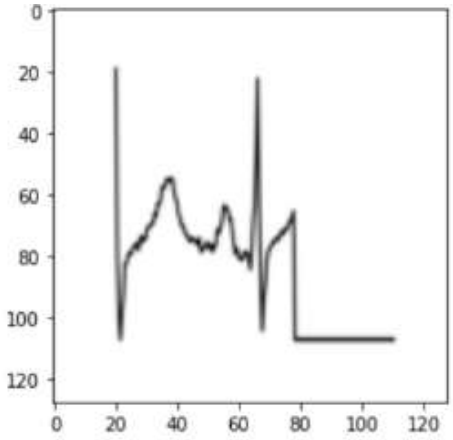


Figure A.7.2: Image resolution 128x128

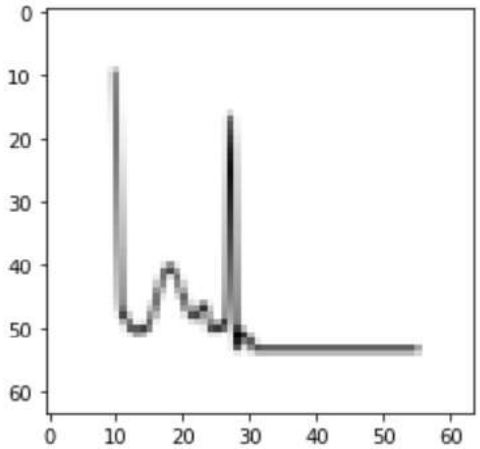


Figure A.7.3: Image resolution 64x64

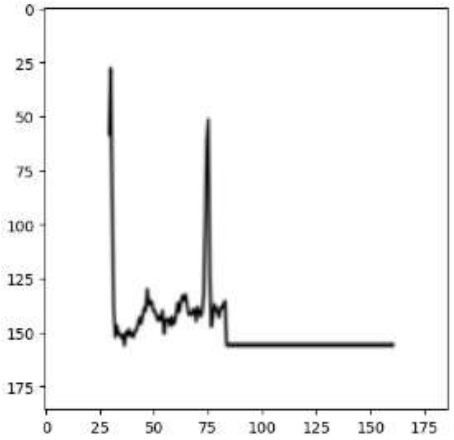


Figure A.7.4: Image resolution 186x186

Appendix B

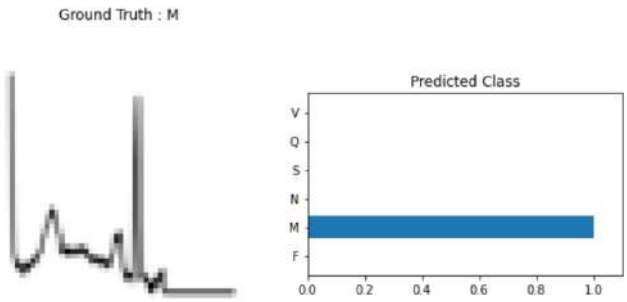


Figure B.7.1: testset [324]

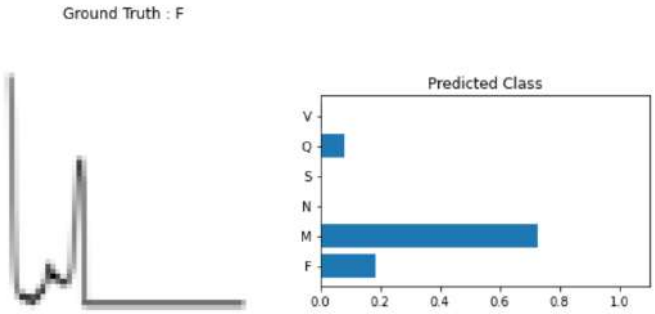


Figure B.7.2: testset[111]

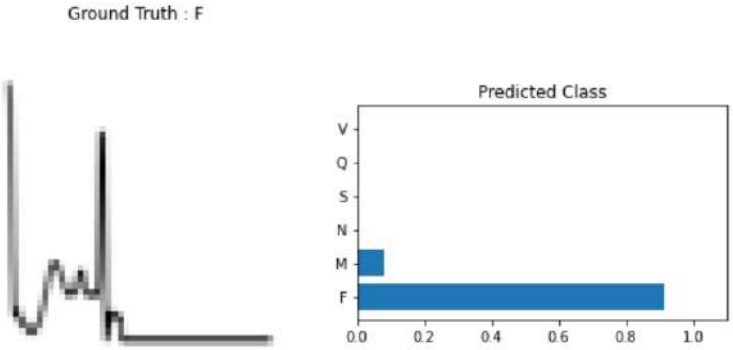


Figure B.7.3: testset[4]

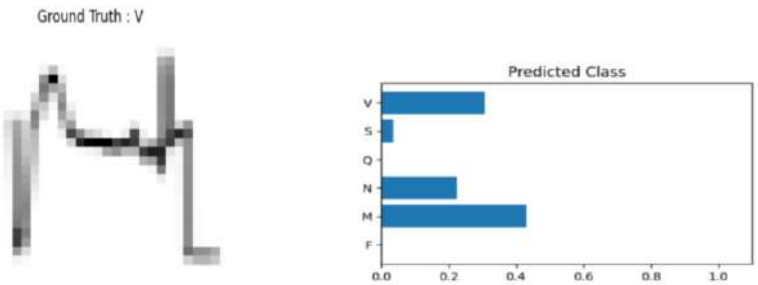


Figure B.7.4: testset[23855]

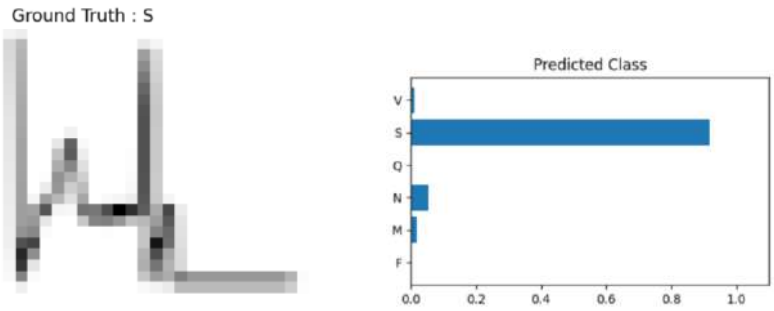


Figure B.7.5: testset[22946]

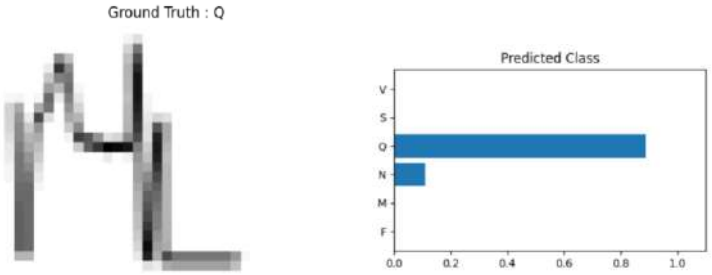


Figure B.7.6: testset[21948]

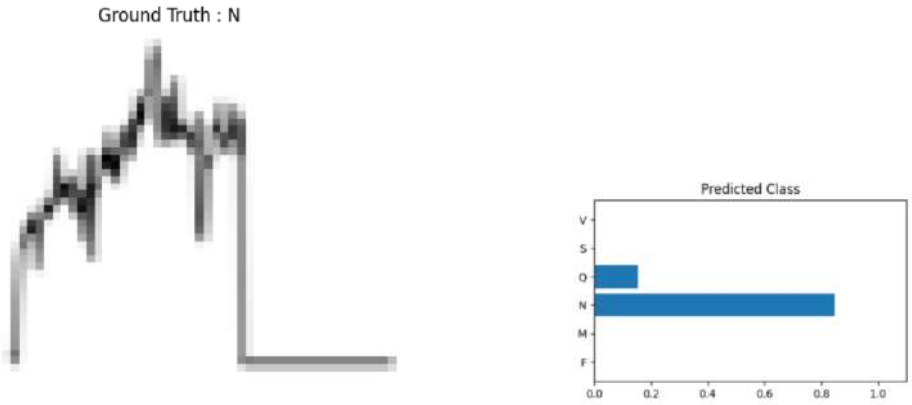


Figure B.7.7: testset[2346]

For example, replacing the first layer.

With the following layer:

```
(conv1): Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
```

And then copy the sum (in the channel axis) of the weights to the new layer, for example, the shape of the original weights was:

Modified:

```
torch.Size([64, 3, 7, 7])
```

```
model.conv1.weight.data = model.conv1.weight.data.sum(axis=1).reshape(64, 1, 7, 7)
```

The similar thing can be done using this approach:
First Layer:

```
print(model.conv_stem)

>>Conv2dSame(3, 32, kernel_size=(3, 3), stride=(2, 2), bias=False)
```

```
model.conv_stem = nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), bias=False)
model.state_dict()['conv_stem.weight'] =
model.state_dict()['conv_stem.weight'].sum(dim=1, keepdim=True)
```

Update state_dict:

Modified Layer:

```
print(model.conv_stem)
>>Conv2dSame(1, 32, kernel_size=(3, 3), stride=(2, 2), bias=False)
```

In this way the channels are modified from 3 to 1.
The simplest way that timm offers is:

```
model = timm.create_model(CFG.model_name,pretrained=True, in_chans=1)
```

Appendix D

AAMI class	MIT-BIH Class	MIT-BIH Heartbeat
Normal beats (N)	N	Normal beat
	L	Left bundle branch block beat
	R	Right bundle branch block beat
	e	Atrial escape beat
	j	Nodal escape
Supraventricular ectopic beats (SVEB)	A	Atrial premature beat
	a	Aberrated atrial premature beat
	J	Nodal (junctional) premature beat
	S	Supraventricular premature beat
Ventricular ectopic beat (VEB)	V	Premature ventricular contraction
	E	Ventricular escape beat
Fusion beats (F)	F	Fusion of ventricular and normal beat
Unknown beats (Q)	/	Paced beat
	f	Fusion of paced and normal beat
	Q	Unclassified beat

Figure D.7.1: ANSI/AAMI standards in ECG Class Interpretation [58]

Table D.7.1: Versions of environments used in this work

Versions				
Python	**PyTorch**	**TensorFlow**	**Keras**	**MATLAB**
3.9.16	2.0.0+cu118	2.11.0	2.11.0	2018a

Appendix E

Python Programming Language

Python is a versatile and popular programming language with a wide range of uses in numerous industries. It appeals to both beginners and experts due to its simplicity, readability, and usability. It is an interpretive, object-oriented, and dynamically typed high-level language.

MATLAB

The name of a programming environment, MATLAB, which is short for "matrix laboratory," refers to the program's primary use of matrices and arrays. It offers a comprehensive collection of built-in tools and routines that may be utilized to tackle a variety of computational issues.

Users can interactively explore and analyze their data using MATLAB's robust graphical user interface (GUI). It can interface with other languages like C, C++, and Python and supports a variety of data kinds like numerical, text, and image data.

TensorFlow and Keras Framework

TensorFlow is an open-source machine learning framework made by Google that is used for many deep learning methods to implement neural networks. On top of that lies Keras which is an API that contains all the models for classification, regression which allows easier implementation of the neural networks.

PyTorch Framework

Based on the Torch framework, PyTorch is an open-source library. This robust deep learning framework is a favorite among programmers and academics because it provides a wealth of capabilities, great GPU acceleration support, and an intuitive API. It is largely created by Facebook's AI Research lab (FAIR), and deep learning models are produced using it frequently. Dynamic computation graphs, as opposed to static computation graphs used in other deep learning frameworks, allow for faster experimentation, better flexibility, and simpler debugging. PyTorch enables developers to create and train neural networks utilizing these dynamic computation graphs. We are using PyTorch version 2.0.0+cu118, where "cu118" refers to CUDA version 11.1.8, a patch version of CUDA 11.1. Version numbers for PyTorch are often combined with the relevant CUDA version number to create the version names.

Google Collaboratory Notebook

For launching and executing Python code on the cloud, particularly for data science and machine learning activities, Google Colab offers a potent and simple-to-use platform. While it does have certain restrictions in that it only offers a small number of computing resources. When working with huge datasets or complicated models, this might be a bottleneck and force customers to either pay for more resources or migrate to a different

platform entirely. We trained our networks using the GPU resources offered by Google Colab. Depending on its availability, the GPU used on Google Colab varies from session to session.
The GPU available on Google Colab varies from session to session depending on its availability. These are the GPU available in Google Colab**:**

- NVIDIA Tesla K80
- NVIDIA Tesla T4
- NVIDIA P4
- NVIDIA P100
- NVIDIA V100

We have used standard package that comes with Tesla T4 GPUs usually.

Appendix F

1 Visualize Ground Truth and class prediction probabilities

```
def view_classify (image, ps, label):
class_name = ['F', 'M' ,'N', 'Q', 'S', 'V']
classes = np.array(class_name)
ps = ps.cpu().data.numpy().squeeze()
image = image.permute(1,2,0)
ax2.barh(classes, ps)
    Function call
view_classify(image, ps.squeeze(0), label)
```

Figure F.7.1: Function to visualize class prediction probabilities

2 Class CFG

```
class CFG:
epochs= number of epochs
lr= learning rate
batch = batch size for dataset
size= Input image size
model = model name that is imported
train_path= dataset directories for loading training data
test_path= dataset directories for loading testing data
```

Figure F.7.2: Configuration class

3 Class ECGTrainer

```
class ECGTrainer():
Initialize class instance and its parameters criterion, optimizer, scheduler

   def __init__(self, criterion=None, optimizer=None, schedular=None):
      self.criterion = criterion
      self.optimizer = optimizer
      self.schedular = schedular
      self.train_loss = []
      self.train_acc = []
      self.valid_loss = []
      self.valid_acc = []
```

Function train_batch_loop

```
def train_batch_loop(self, model, trainloader):
    train_loss = 0.0
    train_acc = 0.0

    for images, labels in tqdm(trainloader):
        images = images.to(device)
        labels = labels.to(device)

        logits = model(images)
        loss = self.criterion(logits, labels)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        train_loss += loss.item()
        train_acc += accuracy(logits, labels)

    return train_loss / len(trainloader), train_acc / len(trainloader)
```

Function valid_batch_loop

```
def valid_batch_loop(self, model, validloader):
    valid_loss = 0.0
    valid_acc = 0.0

    for images, labels in tqdm(validloader):
        images = images.to(device)
        labels = labels.to(device)

        logits = model(images)
        loss = self.criterion(logits, labels)

        valid_loss += loss.item()
        valid_acc += accuracy(logits, labels)

    return valid_loss / len(validloader), valid_acc / len(validloader)
```

Function to fit model to the dataset

```
def fit(self, model, trainloader, validloader, epochs):
    valid_min_loss = np.Inf

    for i in range(epochs):
        model.train()
        avg_train_loss, avg_train_acc = self.train_batch_loop(model, trainloader)
```

```
        model.eval()
        avg_valid_loss, avg_valid_acc = self.valid_batch_loop(model, validloader)

        if avg_valid_loss <= valid_min_loss:
            print("Valid_loss decreased {} --> {}".format(valid_min_loss, avg_valid_loss))
            torch.save(model.state_dict(),' modelWeights_path.pt')
            valid_min_loss = avg_valid_loss

        self.train_loss.append(avg_train_loss)
        self.train_acc.append(avg_train_acc)
        self.valid_loss.append(avg_valid_loss)
        self.valid_acc.append(avg_valid_acc)

        print("Epoch : {} Train Loss : {:.6f} Train Acc : {:.6f}".format(i+1, avg_train_loss,
avg_train_acc))
        print("Epoch : {} Valid Loss : {:.6f} Valid Acc : {:.6f}".format(i+1, avg_valid_loss,
avg_valid_acc))
```

Figure F.7.3: ECG trainer class with multiple functions to train and evaluate model

Bibliography

[1]U. Nations. *Sustainable Development Goals*. Available**:** https**:**//www.un.org/en/sustainable-development-goals

[2]S. S. Barold, "Willem Einthoven and the birth of clinical electrocardiography a hundred years ago," *Cardiac electrophysiology review*, vol. 7, p. 99, 2003.

[3]L. R. Kumari, Y. P. Sai, N. Balaji, and K. Viswada, "FPGA based arrhythmia detection," *Procedia Computer Science*, vol. 57, pp. 970-979, 2015.

[4]S. Dalal and V. P. Vishwakarma, "Classification of ECG signals using multi-cumulants based evolutionary hybrid classifier," *Scientific Reports*, vol. 11, p. 15092, 2021.

[5]A. Diker, D. Avci, E. Avci, and M. Gedikpinar, "A new technique for ECG signal classification genetic algorithm Wavelet Kernel extreme learning machine," *Optik*, vol. 180, pp. 46-55, 2019.

[6]P. Yang, D. Wang, W.-B. Zhao, L.-H. Fu, J.-L. Du, and H. Su, "Ensemble of kernel extreme learning machine based random forest classifiers for automatic heartbeat classification," *Biomedical Signal Processing and Control*, vol. 63, p. 102138, 2021.

[7]C.-C. Lin and C.-M. Yang, "Heartbeat classification using normalized RR intervals and morphological features," *Mathematical Problems in Engineering*, vol. 2014, pp. 1-11, 2014.

[8]H. M. Rai, A. Trivedi, and S. Shukla, "ECG signal processing for abnormalities detection using multi-resolution wavelet transform and Artificial Neural Network classifier," *Measurement*, vol. 46, pp. 3238-3246, 2013.

[9]A. Kumar M and A. Chakrapani, "Classification of ECG signal using FFT based improved Alexnet classifier," *Plos one*, vol. 17, p. e0274225, 2022.

[10]F. M. Vaneghi, M. Oladazimi, F. Shiman, A. Kordi, M. Safari, and F. Ibrahim, "A comparative approach to ECG feature extraction methods," in *2012 Third International Conference on Intelligent Systems Modelling and Simulation*, 2012, pp. 252-256.

[11]E. Mazomenos, T. Chen, A. Acharyya, A. Bhattacharya, J. Rosengarten, and K. Maharatna, "A time-domain morphology and gradient based algorithm for ECG feature extraction," in *2012 IEEE International conference on industrial technology*, 2012, pp. 117-122.

[12]A. Biran and A. Jeremic, "ECG bio-identification using Fréchet classifiers**:** A proposed methodology based on modeling the dynamic change of the ECG features," *Biomedical Signal Processing and Control,* vol. 82, p. 104575, 2023.

[13]S. Karpagachelvi, M. Arthanari, and M. Sivakumar, "ECG feature extraction techniques-a survey approach," *arXiv preprint arXiv**:**1005.0957,* 2010.

[14]B. Castro, D. Kogan, and A. Geva, "ECG feature extraction using optimal mother wavelet," in *21st IEEE Convention of the Electrical and Electronic Engineers in Israel. Proceedings (Cat. No. 00EX377)*, 2000, pp. 346-350.

[15]T. Mar, S. Zaunseder, J. P. Martínez, M. Llamedo, and R. Poll, "Optimization of ECG classification by means of feature selection," *IEEE transactions on Biomedical Engineering,* vol. 58, pp. 2168-2177, 2011.

[16]E. B. Mazomenos, D. Biswas, A. Acharyya, T. Chen, K. Maharatna, J. Rosengarten, *et al.*, "A low-complexity ECG feature extraction algorithm for mobile healthcare applications," *IEEE journal of biomedical and health informatics,* vol. 17, pp. 459-469, 2013.

[17]S. Mahmoodabadi, A. Ahmadian, M. Abolhasani, M. Eslami, and J. Bidgoli, "ECG feature extraction based on multiresolution wavelet transform," in *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference,* 2006, pp. 3902-3905.

[18]Q. Zhao and L. Zhang, "ECG feature extraction and classification using wavelet transform and support vector machines," in *2005 International Conference on Neural Networks and Brain*, 2005, pp. 1089-1092.

[19]A. M. Patel, P. K. Gakare, and A. Cheeran, "Real time ECG feature extraction and arrhythmia detection on a mobile platform," *Int. J. Comput. Appl*, vol. 44, pp. 40-45, 2012.

[20]Z. Ebrahimi, M. Loni, M. Daneshtalab, and A. Gharehbaghi, "A review on deep learning methods for ECG arrhythmia classification," *Expert Systems with Applications: X*, vol. 7, p. 100033, 2020.

[21]L. Mhamdi, O. Dammak, F. Cottin, and I. B. Dhaou, "Artificial Intelligence for Cardiac Diseases Diagnosis and Prediction Using ECG Images on Embedded Systems," *Biomedicines,* vol. 10, p. 2013, 2022.

[22]V. Narayana, A. K. Vobbilisetty, S. Mantripragada, V. Merugu, and K. Prakash, "ECG Based Biometric Authentication System using Deep Learning Methods," in *2022 3rd International Conference for Emerging Technology (INCET)*, 2022, pp. 1-4.

[23]S. Kiranyaz, T. Ince, and M. Gabbouj, "Real-time patient-specific ECG classification by 1-D convolutional neural networks," *IEEE Transactions on Biomedical Engineering,* vol. 63, pp. 664-675, 2015.

[24]R. N. Asif, S. Abbas, M. A. Khan, K. Sultan, M. Mahmud, and A. Mosavi, "Development and Validation of Embedded Device for Electrocardiogram Arrhythmia Empowered with Transfer Learning," *Computational Intelligence and Neuroscience,* vol. 2022, 2022.

[25]A.-u. Rahman, R. N. Asif, K. Sultan, S. A. Alsaif, S. Abbas, M. A. Khan, *et al.*, "ECG Classification for Detecting ECG Arrhythmia Empowered with Deep Learning Approaches," *Computational Intelligence and Neuroscience*, vol. 2022, p. 6852845, 2022/07/31 2022.

[26]L. R. Kumar and Y. P. Sai, "A new transfer learning approach to detect cardiac arrhythmia from ECG signals," *Signal, Image and Video Processing,* vol. 16, pp. 1945-1953, 2022.

[27]S. H. Jambukia, V. K. Dabhi, and H. B. Prajapati, "Classification of ECG signals using machine learning techniques**:** A survey," in *2015 International Conference on Advances in Computer Engineering and Applications*, 2015, pp. 714-721.

[28]E. Benmalek, J. Elmhamdi, and A. Jilbab, "ECG scalogram classification with CNN micro-architectures," *Research on Biomedical Engineering,* pp. 1-11, 2022.

[29]A. Datta, B. Kolwadkar, A. Rauta, S. Handal, and V. Ingale, "ECG heartbeat classification using Wavelet transform and different Neural network Architectures," in *2021 6th International Conference for Convergence in Technology (I2CT)*, 2021, pp. 1-7.

[30]P. G. Gaddam and R. Sreehari, "Automatic classification of cardiac arrhythmias based on ECG signals using transferred deep learning convolution neural network," in *Journal of Physics**:** Conference Series*, 2021, p. 012058.

[31]S. Aphale, A. Jha, and E. John, "High Accuracy Arrhythmia Classification using Transfer Learning with Fine-Tuning," in *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, 2022, pp. 0480-0487.

[32]S. Bhaskarpandit, A. Gade, S. Dash, D. K. Dash, R. K. Tripathy, and R. B. Pachori, "Detection of Myocardial Infarction From 12-Lead ECG Trace Images Using Eigendomain Deep Representation Learning," *IEEE Transactions on Instrumentation and Measurement,* vol. 72, pp. 1-12, 2023.

[33]T.-R. Wei, S. Lu, and Y. Yan, "Automated Atrial Fibrillation Detection with ECG," *Bioengineering,* vol. 9, p. 523, 2022.

[34]C.-f. Zhao, W.-y. Yao, M.-j. Yi, C. Wan, and Y.-l. Tian, "Arrhythmia Classification Algorithm Based on a Two-Dimensional Image and Modified EfficientNet," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.

[35]R. M. Obaidi, R. A. Sattar, M. Abd, I. A. Almani, T. Alghazali, S. G. Talib, *et al.*, "ECG Arrhythmia Classification based on Convolutional Autoencoders and Transfer Learning," *Majlesi Journal of Electrical Engineering*, vol. 16, pp. 41-46, 2022.

[36]S. Shin, M. Kang, G. Zhang, J. Jung, and Y. T. Kim, "Lightweight Ensemble Network for Detecting Heart Disease Using ECG Signals," *Applied Sciences*, vol. 12, p. 3291, 2022.

[37]K. T. Chui, B. B. Gupta, M. Zhao, A. Malibari, V. Arya, W. Alhalabi, *et al.*, "Enhancing Electrocardiogram Classification with Multiple Datasets and Distant Transfer Learning," *Bioengineering*, vol. 9, p. 683, 2022.

[38]M. Cao, T. Zhao, Y. Li, W. Zhang, P. Benharash, and R. Ramezani, "ECG heartbeat classification using deep transfer learning with convolutional neural network and STFT technique," *arXiv preprint arXiv:2206.14200*, 2022.

[39]K. Weimann and T. O. Conrad, "Transfer learning for ECG classification," *Scientific reports*, vol. 11, pp. 1-12, 2021.

[40]M. Gu, Y. Zhang, Y. Wen, G. Ai, H. Zhang, P. Wang, *et al.*, "A lightweight convolutional neural network hardware implementation for wearable heart rate anomaly detection," *Computers in Biology and Medicine*, p. 106623, 2023.

[41]X. Cheng, D. Liu, J. Lu, L. Wei, A. Hu, J. Lei, *et al.*, "Efficient hardware design of a deep U-net model for pixel-level ECG classification in healthcare device," *Microelectronics Journal*, vol. 126, p. 105492, 2022.

[42]M. Janveja, R. Parmar, M. Tantuway, and G. Trivedi, "A DNN-based low power ECG co-processor architecture to classify cardiac arrhythmia for wearable devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, pp. 2281-2285, 2022.

[43]W. Caesarendra, T. A. Hishamuddin, D. T. C. Lai, A. Husaini, L. Nurhasanah, A. Glowacz, *et al.*, "An embedded system using convolutional neural network model for online and real-time ECG signal classification and prediction," *Diagnostics*, vol. 12, p. 795, 2022.

[44]R. Mao, S. Li, Z. Zhang, Z. Xia, J. Xiao, Z. Zhu, *et al.*, "An Ultra-Energy-Efficient and High Accuracy ECG Classification Processor With SNN Inference Assisted by On-Chip ANN Learning," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 16, pp. 832-841, 2022.

[45]S. Ran, X. Yang, M. Liu, Y. Zhang, C. Cheng, H. Zhu, *et al.*, "Homecare-oriented ECG diagnosis with large-scale deep neural network for continuous monitoring on embedded devices," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1-13, 2022.

[46]Y.-L. Xie, X.-R. Lin, and C.-W. Lin, "SEmbedNet**:** Hardware-Friendly CNN for Ectopic Beat Classification on STM32-Based Edge Device," in *2022 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)*, 2022, pp. 1-6.

[47]L. Pettersson, "Convolutional neural networks on FPGA and GPU on the edge**:** A comparison," ed, 2020.

[48]M. Tan and Q. Le, "Efficientnet**:** Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, 2019, pp. 6105-6114.

[49]V. Agarwal. (2020). *Complete Architectural Details of all EfficientNet Models*. Available**:** https**:**//towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142

[50]Y. Guo, Y. Li, L. Wang, and T. Rosing, "Depthwise convolution is all you need for learning multiple visual domains," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 8368-8375.

[51]W. Wang, Y. Li, T. Zou, X. Wang, J. You, and Y. Luo, "Research Article A Novel Image Classification Approach via Dense-MobileNet Models," 2020.

[52]F. Sultonov, J.-H. Park, S. Yun, D.-W. Lim, and J.-M. Kang, "Mixer U-Net**:** An improved automatic road extraction from UAV imagery," *Applied Sciences*, vol. 12, p. 1953, 2022.

[53]A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314-1324.

[54]G. B. Moody and R. G. Mark, "MIT-BIH Arrhythmia Database," ed**:** physionet.org, 1992.

[55]R.-D. Bousseljot, D. Kreiseler, and A. Schnabel, "The PTB Diagnostic ECG Database," ed**:** physionet.org, 2004.

[56]. *UCI Machine Learning Repository**:** Arrhythmia Data Set*. Available**:** https**:**//archive.ics.uci.edu/ml/datasets/arrhythmia

[57]. *ecg_image_data*. Available**:** https**:**//www.kaggle.com/datasets/erhmrai/ecg-image-data

[58]S. Liu, J. Shao, T. Kong, and R. Malekian, "ECG Arrhythmia Classification using High Order Spectrum and 2D Graph Fourier Transform," *Applied Sciences*, vol. 10, p. 4741, 07/09 2020.