

**AUTOMATED DETECTION AND CLASSIFICATION  
OF LEUKEMIA USING DEEP LEARNING**

**LEE KYE FUNG**

**A project report submitted in partial fulfilment of the  
requirements for the award of the degree of  
Bachelor of Engineering (Hons) Electronic Engineering**

**Faculty of Engineering and Green Technology  
Universiti Tunku Abdul Rahman**

**May 2022**

**DECLARATION**

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :           *Aaron*          

Name :           Lee Kye Fung          

ID No. :           1703320          

Date :           20/4/2022

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled “**AUTOMATED DETECTION AND CLASSIFICATION OF LEUKEMIA USING DEEP LEARNING**” was prepared by **LEE KYE FUNG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons) Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : Humaira

Supervisor : Prof. Ts. Dr. Humaira Nisar

Date : 26.4.2022

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, LEE KYE FUNG. All right reserved.

## **ACKNOWLEDGEMENTS**

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Prof. Ts. Dr. Humaira Nisar and my moderator, Dr. Lee Yu Jen, for their invaluable advice, guidance and their enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement. I would also like to thank everyone who supported me along the way through the completion of this project.

## **AUTOMATED DETECTION AND CLASSIFICATION OF LEUKEMIA USING DEEP LEARNING**

### **ABSTRACT**

Leukemia is a type of blood cancer that has been affecting the lives of many. The main procedure to diagnose and classify leukemia is through microscopic examination of blood smears, which can be costly, time-consuming, and labour-intensive. Hence, this project aims to produce an efficient way to detect and classify leukemia by using deep learning. In this project, transfer learning is implemented on three pre-trained deep learning models, namely Inception-V3, ResNeXt, and SENet models. They were trained to tackle two main tasks: binary classification between ALL and healthy cells, and 5-class classification between ALL, AML, CLL, CML, and healthy cells. The microscopic image samples of these classes are retrieved from two sources, including the Acute Lymphoblastic Leukemia Image Database 1 (ALL-IDB1) and American Society of Hematology (ASH) ImageBank. It is observed that the SENet model performed the best out of the three, hence it is selected to undergo further fine-tuning to improve its performance. With a slow converging feature selection process added with the dropout regularization technique, the SENet model can achieve an average testing accuracy of 99.84% and 84.48% in binary and 5-class classification problems.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>ii</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>	<b>v</b>
<b>ABSTRACT</b>	<b>vi</b>
<b>TABLE OF CONTENTS</b>	<b>vii</b>
<b>LIST OF TABLES</b>	<b>xi</b>
<b>LIST OF FIGURES</b>	<b>xv</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>	<b>xxiii</b>
<b>LIST OF APPENDICES</b>	<b>xxv</b>

### CHAPTER

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
	1.1 Background	1
	1.2 Problem Statements	3
	1.3 Aims and Objectives	3
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>4</b>
	2.1 Leukemia	4
	2.1.1 Acute Lymphocytic Leukemia	6
	2.1.2 Acute Myelogenous Leukemia	6
	2.1.3 Chronic Lymphocytic Leukemia	7
	2.1.4 Chronic Myelogenous Leukemia	7
	2.2 Detection and Classification	8

2.3	Morphology of Leukemia and Peripheral Blood Smear Findings	9
2.4	Deep Learning	14
2.4.1	Deep Supervised Learning	17
2.4.2	Deep Unsupervised Learning	17
2.4.3	Deep Reinforcement Learning	18
2.4.4	Deep Semi-supervised Learning	19
2.4.5	Deep Transfer Learning	19
2.5	Types of Artificial Neural Networks	21
2.5.1	Recursive Neural Network	21
2.5.2	Recurrent Neural Network	23
2.5.3	Convolutional Neural Network	24
2.6	CNN Architecture	25
2.6.1	Convolution Layer	26
2.6.2	Pooling Layer	29
2.6.3	Activation Function	30
2.6.4	Batch Normalization	33
2.6.5	Dropout	34
2.6.6	Fully Connected Layer	35
2.7	Types of CNN Architectures	35
2.7.1	AlexNet	36
2.7.2	ZFNet	37
2.7.3	GoogLeNet	38
2.7.4	ResNet	42
2.7.5	ResNeXt	42
2.7.6	SENet	43
2.7.7	DenseNet	44
2.8	Performance Metrics	45
2.8.1	Accuracy	46
2.8.2	Precision	47
2.8.3	Recall	48
2.8.4	F1-score	48
2.8.5	Confusion Matrix	49
2.8.6	ROC Curve	50



2.9	Related Works	52
<b>3</b>	<b>METHODOLOGY</b>	<b>55</b>
3.1	Project Flow	55
3.2	Project Requirements	56
3.2.1	Hardware Requirements	57
3.2.2	Software Requirements	57
3.2.3	Programming Language Used	57
3.2.4	Open-Source Libraries	58
3.3	Dataset Acquisition	59
3.4	Dataset Splitting	59
3.5	Dataset Augmentation	61
3.6	Model Training	65
3.7	Model Evaluation	67
3.8	Model Improvement	67
3.9	Project Costs	69
3.10	Project Management	70
<b>4</b>	<b>RESULTS AND DISCUSSIONS</b>	<b>72</b>
4.1	Binary Classification Problem for Inception-V3, ResNeXt. And SENet	72
4.1.1	Fold 1	73
4.1.2	Fold 2	76
4.1.3	Fold 3	79
4.1.4	Fold 4	82
4.1.5	Fold 5	85
4.2	5-class Classification Problem for Inception-V3, ResNeXt. And SENet	88
4.2.1	Fold 1	89
4.2.2	Fold 2	93
4.2.3	Fold 3	97
4.2.4	Fold 4	101
4.2.5	Fold 5	105

4.3	Fine-tuned SENet Models	109
4.3.1	Fold 1	110
4.3.2	Fold 2	114
4.3.3	Fold 3	118
4.3.4	Fold 4	122
4.3.5	Fold 5	126
4.4	Binary Classification Problem for SENet with 3 Hidden Layers Plus Dropout Layers	130
4.4.1	Accuracy and Loss Against Number of Epochs	130
4.4.2	Precision, Recall, and F1-score for Leukemia Subtypes Classification	131
4.4.3	ROC Curve	132
4.4.4	Confusion Matrix	133
4.5	Discussion	134
<b>5</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	<b>150</b>
5.1	Project Review	150
5.2	Project Findings	151
5.3	Recommendations for Future Improvements	153
5.4	Conclusion	154
	<b>REFERENCES</b>	<b>155</b>
	<b>APPENDICES</b>	<b>164</b>

## LIST OF TABLES

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
2.1	Morphology of ALL Subtypes According to FAB Classification	11
2.2	Morphology of AML Subtypes According to FAB Classification	12
2.3	DL Models and Their References	16
3.1	Commands to Install Open-Source Libraries	58
3.2	Dataset Count for Each Class and Its Source	59
3.3	Training Set Distribution for Each Fold	60
3.4	Testing Set Distribution for Each Fold	61
3.5	Training Set Distribution for Each Fold After Augmentation	62
3.6	Testing Set Distribution for Each Fold After Augmentation	63
3.7	Binary Classification Problem Training Set Distribution	66
3.8	5-class Classification Problem Training Set Distribution for Each Fold	66
3.9	Equipment and Materials Cost	69
3.10	FYP 1 Gantt Chart	70
3.11	FYP 2 Gantt Chart	71
4.1	Accuracy and Loss Result for Binary Classification Problem of Each Fold for Inception-V3	72

4.2	Accuracy and Loss Result for Binary Classification Problem of Each Fold for ResNeXt	72
4.3	Accuracy and Loss Result for Binary Classification Problem of Each Fold for SENet	73
4.4	Precision, Recall, and F1-score for Binary Classification Problem in The First Fold Training for Each Model	74
4.5	Precision, Recall, and F1-score for Binary Classification Problem in The Second Fold Training for Each Model	76
4.6	Precision, Recall, and F1-score for Binary Classification Problem in The Third Fold Training for Each Model	79
4.7	Precision, Recall, and F1-score for Binary Classification Problem in The Fourth Fold Training for Each Model	82
4.8	Precision, Recall, and F1-score for Binary Classification Problem in The Fifth Fold Training for Each Model	85
4.9	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for Inception-V3	88
4.10	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for ResNeXt	88
4.11	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet	88
4.12	Precision, Recall, and F1-score for 5-class Classification Problem in The First Fold Training for Each Model	89
4.13	Precision, Recall, and F1-score for 5-class Classification Problem in The Second Fold Training for Each Model	93
4.14	Precision, Recall, and F1-score for 5-class Classification Problem in The Third Fold Training for Each Model	97
4.15	Precision, Recall, and F1-score for 5-class Classification Problem in The Fourth Fold Training for Inception-V3	101

4.16	Precision, Recall, and F1-score for 5-class Classification Problem in The Fifth Fold Training for Inception-V3	105
4.17	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet + SVM	109
4.18	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet with 3 Hidden Layers	109
4.19	Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet with 3 Hidden Layers Plus Dropout Layers	109
4.20	Precision, Recall, and F1-score for 5-class Classification Problem in The First Fold Training for Each Fine-tuned SENet Models	110
4.21	Precision, Recall, and F1-score for 5-class Classification Problem in The Second Fold Training for Each Fine-tuned SENet Models	114
4.22	Precision, Recall, and F1-score for 5-class Classification Problem in The Third Fold Training for Each Fine-tuned SENet Models	118
4.23	Precision, Recall, and F1-score for 5-class Classification Problem in The Fourth Fold Training for Each Fine-tuned SENet Models	122
4.24	Precision, Recall, and F1-score for 5-class Classification Problem in The Fifth Fold Training for Each Fine-tuned SENet Models	126
4.25	Accuracy and Loss Result for Binary Classification Problem of Each Fold	130
4.26	Precision, Recall, and F1-score for Binary Classification Problem	131
4.27	Average Precision, Recall, and F1-score for Binary Classification Problem for Each Model	136
4.28	Average Precision, Recall, and F1-score for 5-class Classification Problem for Each Model	139
4.29	Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and B	141

4.30	Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and C	143
4.31	Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and D	147

## LIST OF FIGURES

<b>FIGURE</b>	<b>TITLE</b>	<b>PAGE</b>
2.1	A Bar Chart Showing the Percentage of Deaths Due to Leukemia	5
2.2	(a) Blood Smears of Healthy Blood Cells (Scotti, Labati and Piuri, 2011). (b) ALL (Scotti, Labati and Piuri, 2011). (c) AML (American Society of Hematology, n.d.). (d) CLL (American Society of Hematology, n.d.). (e) CML (American Society of Hematology, n.d.)	10
2.3	Architecture of ANN (Adapted from Mishra and Gupta, 2017)	14
2.4	Illustration of ML In Comparison with DL (Adapted from Alzubaidi, et al., 2021)	15
2.5	ImageNet Top-5 Error Rate of DL Models Compared to Human Errors (Adapted from Alzubaidi, 2021)	16
2.6	Deep Supervised Learning (Qian, et al., 2020)	17
2.7	Deep Unsupervised Learning (Qian, et al., 2020)	18
2.8	Deep Reinforcement Learning (Amiri, et al., 2018)	18
2.9	Transfer Learning Process (Tan, et al., 2018)	21
2.10	Illustration of How RvNN Parses Scene Images (Socher, et al., 2011)	22
2.11	(a) Typical RNN Structure (b) One-To-Many Temporal Structure of RNN (c) Many-To-One Temporal Structure of RNN (d) Many-To-Many Temporal Structure of RNN (Adapted from Rezk, et al., 2020; Su and Li, 2019)	23

2.12	CNN Architecture (Adapted from Mishra and Gupta, 2017)	25
2.13	Image Classification Using CNN Architecture (Adapted from Alzubaidi, et al., 2021)	26
2.14	Convolution Between Two Functions (Pihlajamäki, 2009)	26
2.15	Convolution Operation Between a Kernel and An Input Tensor (Adapted from Reynolds, 2019; Yamashita, et al., 2018)	28
2.16	Zero Padding Before Performing Convolution Operation (Adapted from Reynolds, 2019; Yamashita, et al., 2018)	28
2.17	Illustration of Max Pooling and Global Average Pooling (Adapted from Alzubaidi, et al., 2021; Yamashita, et al., 2018)	30
2.18	Sigmoid Function and Its Derivative (Omkar, 2019)	31
2.19	Tanh Function and Its Derivative (Omkar, 2019)	31
2.20	ReLU Function and Its Derivative (Szandała, 2020)	32
2.21	Leaky ReLU and PReLU Functions (Omkar, 2019)	33
2.22	Illustration of Dropout (Adapted from Srivastava, et al., 2014)	35
2.23	AlexNet Architecture (Tsang, 2018)	36
2.24	ZFNet Architecture (Zeiler and Fergus, 2014)	37
2.25	GoogLeNet Architecture (Szegedy, et al., 2015)	39
2.26	(a) Unfactorized Inception Module (b) Factorized Inception Module Where Filter Of Size $5 \times 5$ Is Replaced with Two Filters of Size $3 \times 3$ (Szegedy, et al., 2015)	40
2.27	Inception Module with Asymmetric Convolutional Filters (Szegedy, et al., 2015)	41
2.28	Expanded Inception Module (Szegedy, et al., 2015)	41



2.29	The Building Block of ResNet (He, et al., 2015)	42
2.30	Building Block of ResNeXt with Cardinality of 32 (Xie, et al., 2017)	43
2.31	SE-block (Hu, et al., 2017)	44
2.32	DenseNet Architecture (Huang, et al., 2017)	45
2.33	Visualizing Accuracy (Maleki, et al., 2020)	47
2.34	Visualizing Precision (Maleki, et al., 2020)	47
2.35	Visualizing Recall (Maleki, et al., 2020)	48
2.36	Illustration of Confusion Matrix	49
2.37	Illustration of ROC Curve with (a) AUC-ROC of 1 (b) AUC-ROC of 0.8 (c) AUC-ROC of 0 (Narkhede, 2018)	51
2.38	Bar Chart of The Accuracy Comparison Between Related Works	53
2.39	Bar Chart of The Accuracy Comparison Between Related Works	53
3.1	Flowchart of The Project	56
3.2	(a) Original (b) 20° Shearing (c) 30% Zoom (d) 40° Rotation (e) 40% Height Shift (f) 40% Width Shift (g) Vertical Flip (h) Horizontal Flip	63
3.3	Bar Chart of Dataset Allocation for Each Fold	64
3.4	Visualizing the Pre-Processed Dataset	64
3.5	Transfer Learning with Pretrained Models as Feature Extractors	65
4.1	Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	73
4.2	ROC Curve in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	74
4.3	Confusion Matrix in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	75

4.4	Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	76
4.5	ROC Curve in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	77
4.6	Confusion Matrix in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	78
4.7	Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	79
4.8	ROC Curve in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	80
4.9	Confusion Matrix in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	81
4.10	Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	82
4.11	ROC Curve in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	83
4.12	Confusion Matrix in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	84
4.13	Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	85
4.14	ROC Curve in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	86
4.15	Confusion Matrix in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	87
4.16	Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	89
4.17	ROC Curve in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	91
4.18	Confusion Matrix in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	92

4.19	Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	93
4.20	ROC Curve in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	95
4.21	Confusion Matrix in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	96
4.22	Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	97
4.23	ROC Curve in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	99
4.24	Confusion Matrix in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	100
4.25	Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	101
4.26	ROC Curve in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	103
4.27	Confusion Matrix in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	104
4.28	Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	105
4.29	ROC Curve in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	107
4.30	Confusion Matrix in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet	108
4.31	Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers	110
4.32	ROC Curve in The First Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	112

4.33	Confusion Matrix in The First Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	113
4.34	Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers	114
4.35	ROC Curve in The Second Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	116
4.36	Confusion Matrix in The Second Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	117
4.37	Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers	118
4.38	ROC Curve in The Third Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	120
4.39	Confusion Matrix in The Third Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	121
4.40	Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers	122
4.41	ROC Curve in The Fourth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	124
4.42	Confusion Matrix in The Fourth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	125
4.43	Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers	126

4.44	ROC Curve in The Fifth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	128
4.45	Confusion Matrix in The Fifth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers	129
4.46	Accuracy and Loss Against Number of Epochs for Each Fold	131
4.47	ROC Curve for Each Fold	132
4.48	Confusion Matrix in The Fourth Fold Training for Each Fold	133
4.49	Overall Architecture of Each Model for (a) Binary Classification Problem (b) 5-class Classification Problem	134
4.50	Bar Chart of The Testing Accuracy Comparison of The Studies on Binary Classification Problem	135
4.51	Bar Chart of The Testing Accuracy Comparison of The Studies on 5-class Classification Problem	138
4.52	Overall Architecture of Each Model for SENet model B	141
4.53	Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and B	142
4.54	Overall Architecture of Each Model for SENet model C	143
4.55	Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and C	144
4.56	Overall Architecture of Each Model for SENet model D	145
4.57	Bar Chart of The Testing Accuracy Comparison of The Studies on 5-class Classification Problem Including Fine-tuned Models	146
4.58	Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and D	147

4.59	Bar Chart of The Testing Accuracy Comparison of The Studies on Binary Classification Problem Including Fine-tuned Models	148
------	--	-----

## LIST OF SYMBOLS / ABBREVIATIONS

$\mathcal{D}$	Domain
$\mathcal{F}$	Feature space
$f(\cdot)$	Predictive function
$P(X)$	Edge probability distribution
$\mathcal{T}$	Task
$\mathcal{Y}$	Label space
L1	Lymphoblastic Leukemia with Homogenous Structure
L2	Lymphoblastic Leukemia with Varied Structure
L3	Burkitt's leukemia
M0	Acute Myeloblastic Leukemia with Minimal Differentiation
M1	Acute Myeloblastic Leukemia without Maturation
M2	Acute Myeloblastic Leukemia with Maturation
M3	Promyelocytic Leukemia
M4	Acute Myelomonocytic Leukemia
M5a	Acute Monoblastic Leukemia
M5b	Acute Monocytic Leukemia
M6	Acute Erythroid Leukemia
M7	Acute Megakaryocytic Leukemia
ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
ALL	Acute Lymphocytic Leukemia
ALL-IDB1	Acute Lymphoblastic Leukemia Image Database 1
AML	Acute Myelogenous Leukemia
ANN	Artificial Neural Network
ASH	American Society of Hematology

AUC	Area Under the Curve
CLL	Chronic lymphocytic leukemia
CML	Chronic Myelogenous Leukemia
CNN	Convolutional Neural Network
CSR	Cancer Statistics Review
DL	Deep Learning
DNN	Deep Neural Network
FAB	French-American-British
FFNN	Feedforward Neural Network
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IR	Interventional Radiology
LDI-PCR	Long Distance Inverse Polymerase Chain Reaction
LGB	Lateral Geniculate Body
ML	Machine Learning
NC	Nucleocytoplasmic
NIN	Network in Network
RBC	Red Blood Cells
ROC	Receiver Operating Characteristics
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RvNN	Recursive Neural Network
WBC	White Blood Cells
PReLU	Parametric Rectified Linear Unit
RMSProp	Root Mean Square Propagation
SGD	Stochastic Gradient Descent
SEER	Surveillance, Epidemiology, and End Results Program
aCGH	Array-based Comparative Genomic Hybridization



**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	Code Listings	164

## CHAPTER 1

### INTRODUCTION

#### 1.1 Background

Leukemia is a type of blood cancer that involves white blood cells (WBC), whereby immature WBC produced in the body will affect the bone marrow and the blood. They will then spread to other parts of the body, resulting in other deadly cancers. Leukemia can be classified based on its development speed: acute or chronic, and based on the types of cells involved: lymphoid and myeloid cells. Hence, there can be four subtypes of leukemia: acute lymphocytic leukemia (ALL), acute myelogenous leukemia (AML), chronic lymphocytic leukemia (CLL), and chronic myelogenous leukemia (CML).

According to Ahmed, et al. (2019), the detection and classification of leukemia and its subtypes are usually done through a few techniques such as interventional radiology (IR), Array-based Comparative Genomic Hybridization (aCGH), Molecular Cytogenetics, and Long-Distance Inverse Polymerase Chain Reaction (LDI-PCR), but they may be limited by the image resolution and may take a lot of time, cost and effort to perform. Ergo, the main procedure to diagnose and classify leukemia is still through microscopic examination of blood smears.

In a pursuit to efficiently detect and classify leukemia, deep learning is utilized in this project. Deep learning is beneficial in medical image analysis because it does not have human limitations such as fatigue and slow speed. They are also easy to deploy compared to employing professional radiologists or oncologists to perform examinations on the medical images (Ker, et al., 2018). The earliest talks of deploying

deep learning in the medical field date back to 1995 where Lo, et al. proposed a CNN for lung nodule detection from chest X-rays. In 2016, Rajkomar, et al. used a pre-trained GoogLeNet model to classify whether the chest x-ray image orientation is a lateral or frontal side of view and achieved an accuracy of almost 100%. Though it may be a simple task, it demonstrated the effectiveness of deep learning applications on medical images. From then onwards, more and more successful implementations of deep learning emerge, which includes the classification of 14 diseases from chest x-rays done by Rajpukar, et al. in 2017, Alzheimer's Disease detection done by Hosseini-Asl, E., Gimel'farb, G. and El-Baz, A. in 2016, and diabetic retinopathy (DR) detection done by Pratt, et al. in 2016, just to name a few. All in all, detection and classification using deep learning is becoming more common in the medical field.

The main driving force that helped spread the significance of deep learning is through annual challenges such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), which started in 2010 until the present. It prompts researchers worldwide to best the previous model in a few challenges such as object localization, image classification, and object detection. Some state-of-the-art CNN image classification models that emerged from this challenge include AlexNet, DenseNet, Inception, ResNet, ResNeXt, and SENet, but it is just getting started. As technology advances, deep learning algorithms are also enabled to be deeper and smarter. The growing community in deep learning also emerges new innovative algorithms, improving the performance and feasibility of deep learning models for real-world applications. Furthermore, competitions like the one hosted by Kaggle can drive interest in many new or existing researchers to continuously push the boundaries of deep learning. In addition, the emerging term known as big data will also propel the evolution of deep learning algorithms due to the more extensive training set available.

In a nutshell, this project will demonstrate how deep learning can accurately and efficiently detect and classify leukemia from microscopic images. The methods for performing and tackling common problems in deep learning are discussed in the following chapters.

## **1.2 Problem Statements**

As aforementioned, microscopic examination of blood smears is still the standard technique to diagnose and classify leukemia. However, according to Bain (2015), The microscopic examination of blood smears requires a medically qualified pathologist or haematologist to perform, which implies that it can be costly, time-consuming, and labour-intensive.

Besides that, based on the statistics obtain from the National Cancer Institute (2021), all walks of life are susceptible to leukemia, with an estimation of 61090 new cases of leukemia where 23660 people will succumb to the disease. That makes up about 1 out of 3 people among the cases who will die of leukemia. Thus, a faster solution to detect and classify leukemia is vital as each subtype requires different types of medical treatment.

## **1.3 Aims and Objectives**

This study aims to produce an efficient way of detecting and classifying leukemia by using deep learning. The objectives of this thesis are shown below:

- i) To select and train a suitable deep learning model which can detect leukemia cells from healthy cells, and classify them into their subtypes.
- ii) To fine-tune the selected model for a better performance rate in the detecting and classifying leukemia.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Leukemia

Leukemia, also spelt as leukaemia, is a cancer of the blood cells. It is a disease that involves the leukocytes, also known as white blood cells (WBC). WBC are part of the human body's immune system, responsible for fending off infections and other diseases. In leukemia, malignant WBC will be produced in the human body, affecting the bone marrow and the blood. This will cause the human immune system to become vulnerable, deteriorating the health of the body. Bone marrow failure caused by the abnormal WBC may also affect all three major cell lineages, resulting in haemorrhage, infections, and anaemia (Pejovic and Schwartz, 2002). In addition, the malignant WBC can also spread to other parts of the body, such as the brain, kidney, liver, lymph nodes, spleen, and nervous system, arising to other deadly forms of cancers (Shafique and Tehsin, 2018).

According to National Cancer Institute (2021), leukemia is a relatively common type of cancer that occupies 3.2% of new cancer cases in the United States (US). It is also estimated that in 2021, there will be an additional 61090 cases of leukemia where 23660 people will succumb to this disease. The percentage of deaths due to leukemia by age group in the United States based from the year 2014 to the year 2018 is illustrated on a bar chart as shown in Figure 2.1. It is observed that the older age group has the highest mortality rate with the highest percentage of death between the age group of 75 to 84, the second-highest percentage of death between the age group of 65 to 74, and the third-highest percentage of death above the age of 84. This

is due mainly to their old age, whereby they have weaker immune systems to fend cancer.

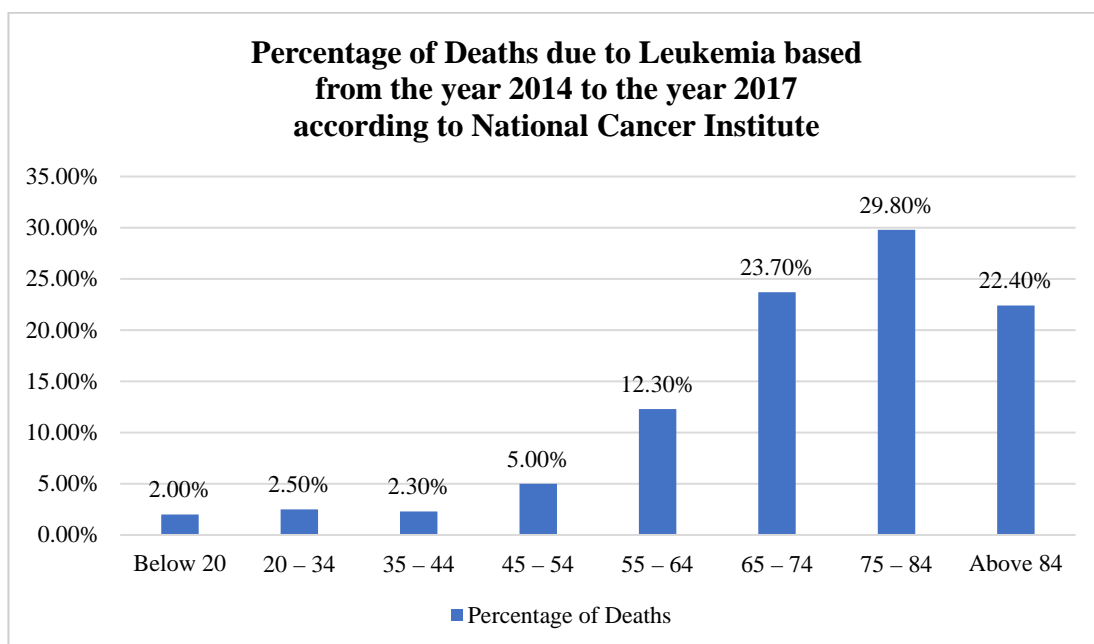


Figure 2.1: A Bar Chart Showing the Percentage of Deaths Due to Leukemia

Leukemia can be classified into two different types based on its development speed: acute and chronic. In acute leukemia, the WBC cannot perform their normal function and will multiply very fast. On the other hand, in chronic leukemia, the WBC can still perform their normal function for a short period and multiply very slowly. However, chronic leukemia may pose health threats as it may go undiagnosed since they are indistinguishable from healthy WBC and will not produce any early symptoms. There are also two additional subtypes from each type of leukemia based on the types of cells involved: lymphoid cells and myeloid cells (Ahmed, et al., 2019). Lymphoid cells are responsible for producing lymphoid or lymphatic tissues that help build our immune system, whereas myeloid cells are cells that will soon be developed into red blood cells (RBC), WBC, or platelets (Mayo Clinic, 2021). Therefore, there are a total of four leukemia subtypes, namely acute lymphocytic leukemia (ALL), acute myelogenous leukemia (AML), chronic lymphocytic leukemia (CLL), and chronic myelogenous leukemia (CML).

### **2.1.1 Acute Lymphocytic Leukemia**

Acute Lymphocytic Leukemia (ALL) is a type of leukemia that is mostly observed among young children, but it may occur in adults as well (Shaheen, et al., 2021). In the US from 2013 to 2017, ALL may occur for 3 to 4 per 100,000 in population from the age of 0 to 14, while approximately 1 per 100,000 only in population with age older than 14, according to the Surveillance, Epidemiology, and End Results Program (SEER) Cancer Statistics Review (CSR) in 2020 by National Cancer Institute. ALL can be further subdivided into three subtypes according to the French-American British (FAB) classification system: lymphoblastic leukemia with homogeneous structure (L1), lymphoblastic leukemia with varied structure (L2), and Burkitt's leukemia (L3) (Ladines-Castro, et al., 2016).

ALL primarily affects the bone marrow and the blood. It multiplies and reproduces immature cells rapidly in the bone marrow, crowding out the healthy cells. The leukemia cells then spread through the bloodstream to other parts of the body, including the brain, nervous system, spleen, liver, and lymph nodes (Rehman, et al., 2018). The symptoms of ALL are greatly similar to the flu, which includes signs of weakness, exhaustion, as well as bone and joint pain, which implies that diagnosing ALL is not an easy task (Bibi, et al., 2020). If timely treatment is not executed, ALL can progress very fast and take a life in a short span of a few months.

### **2.1.2 Acute Myelogenous Leukemia**

Acute Myelogenous Leukemia (AML) is the most common type of leukemia that occurs primarily among adults. However, it may happen to children as well. AML is seen for approximately 1 per 100,000 in population from the age 0 to 19, while 9 to 10 per 100,000 in population with age older than 19, in the US from 2013 to 2017 according to the SEER CSR in 2020 by National Cancer Institute. There is a further subdivision of AML into eight subtypes according to the FAB classification system: acute myeloblastic leukemia with minimal differentiation (M0), acute myeloblastic leukemia without maturation (M1), acute myeloblastic leukemia with maturation (M2),

promyelocytic leukemia (M3), acute myelomonocytic leukemia (M4), acute monoblastic leukemia (M5a) or acute monocytic leukemia (M5b), acute erythroid leukemia (M6), and acute megakaryocytic leukemia (M7) (Ladines-Castro, et al., 2016).

AML is mainly caused by the production of immature WBC and blasts by the bone marrow, which may also produce abnormal RBC and platelets. The early symptoms of AML have many similarities like influenza, which include signs of fever, fatigue and tiredness, easy bruising or bleeding, shortness of breath, and pale skin (Bibi, et al., 2020).

### **2.1.3 Chronic Lymphocytic Leukemia**

Chronic Lymphocytic Leukemia (CLL) is the most common type of leukemia that occurs among adults but is uncommon among children. There is no infection rate reported for ages below 25, but it is observed that CLL occurs for 13 to 14 per 100,000 in population from the age of 25 and above, in the US from 2013 to 2017 according to the SEER CSR in 2020 by National Cancer Institute. A person who suffers from CLL may be asymptomatic for years and live without the need for treatment. However, CLL symptoms may take time to develop, including fever, weight loss, recurring infections, and night sweats (Bibi, et al., 2020).

### **2.1.4 Chronic Myelogenous Leukemia**

Chronic Myelogenous Leukemia (CML) is also a type of leukemia that primarily occurs among adults. According to the SEER CSR in 2020 by National Cancer Institute, it is observed that CML occurs for approximately 1 per 100,000 in population from the age 0 to 19, while 4 to 5 per 100,000 in population with age older than 19, in the US from 2013 to 2017. Although CML is categorized as a chronic or slow progression type of leukemia, it may develop into an acute or rapid progression type



of leukemia, spreading quickly throughout the body. This development can be observed in 3 different phases: the chronic phase, accelerated phase, and blast phase (Bibi, et al., 2020). In the chronic phase, the leukemia is still steadily developing, and there may be no symptoms or have mild symptoms such as tiredness and a slight loss of weight. For the second phase, which is the accelerated phase, symptoms may be more obvious, such as increased tiredness, loss in weight, and a swollen stomach caused by an enlarged spleen. As leukemia worsens, it will then fall into the third phase, which is the blast phase. This phase is also known as the acute phase, whereby CML develops into AML. In this phase, the blood contains more than 30% of blast cells or immature WBC that fills the bone marrow and blood. The leukemia cells might also have spread to other parts of the body (Cancer Research UK, 2019).

## **2.2 Detection and Classification**

The detection and classification of leukemia are undeniably crucial as each subtype requires different types of medical treatment. In this project, the main focus is to classify the subtypes of leukemia, mainly ALL, AML, CLL, and CML, using deep learning. However, the morphology of the subtypes and their further classifications will also be explained in the later parts.

There are various advanced techniques to diagnose leukemia. One such technique is interventional radiology (IR), whereby minimally invasive procedures are performed by utilizing image-guided methods (Arnold, Keung and McCarragher, 2019). However, the limitations of this technique are the resolution of the radio images as well as imaging modality sensitivity. On the other hand, there are different techniques such as Array-based Comparative Genomic Hybridization (aCGH), Molecular Cytogenetics, and Long-Distance Inverse Polymerase Chain Reaction (LDI-PCR), but they require a great deal of hard work and time, and maybe costly (Ahmed, et al., 2019).

On that account, the standard technique in diagnosing leukemia is still through microscopic examination of blood smears. The subtypes are then classified based on

their morphological characteristics. The downside is that this procedure should be performed by highly trained and experienced persons such as a medically qualified pathologist or haematologist so that the most information from the blood smears can be obtained (Bain, 2005). Hence, it can still be time-consuming, labour-intensive, thus bears an exorbitant cost (Dwivedi, 2016).

### **2.3 Morphology of Leukemia and Peripheral Blood Smear Findings**

Each subtype has its morphological features. However, before looking into the subtypes, it is important to recognize the blood with leukemia cells from healthy cells. Figure 2.2 shows the peripheral blood smears of healthy blood cells and the four subtypes of leukemia. It is observed that a normal and healthy blood cells contains RBC, a considerate amount of WBC, as well as platelets. On the contrary, in leukemia, normal cells are often outnumbered by the leukemia cells.

Generally, the findings on the peripheral blood smear of ALL is the presence of leukemic blast cells. They may include early erythroblast, megakaryoblasts, monoblasts, myeloblasts, promyelocytes, or a mixture of the population. The morphology of ALL can be described as having a high nucleocytoplasmic (NC) ratio, which is the ratio of the volume of the nucleus to the volume of the cytoplasm. It has a discoid or ovoid-shaped nucleus with little cytoplasm that is agranular and moderately basophilic (Löffler and Gassmann, 1994). Besides that, it also has inapparent nucleoli and a coarse or clumped chromatin. The morphology characteristics of its three subtypes are illustrated and tabulated in Table 2.1. On the other hand, in AML, one may observe multiple myeloblasts on its peripheral blood smears. The morphology features of AML typically are having large cells with scant granular cytoplasm, fine chromatin, and a discoid or ovoid-shaped nucleus with 2 to 4 nucleoli. The cytoplasm may also have needle-shaped structures, called Auer rods. They are crystalline cytoplasmic inclusion that is produced from the abnormal fusion of azurophilic granules. The morphology characteristics of its eight subtypes are illustrated and tabulated in Table 2.2.

Furthermore, in CLL, smudge cells may be observed in the peripheral blood smears as neoplastic cells are fragile and might be smudged during the preparation of blood slides. The morphology features of CLL are that it has small neoplastic lymphocytes with scant cytoplasm. Its nuclei are also round or irregular in shape with clumped chromatin, as well as small nucleoli. Lastly, the findings on the peripheral blood smears of CML is that it has leucocytosis with left shift. This indicates that there is presence of immature granulocytes such as bands, metamyelocytes, myelocytes, and promyelocytes, as well as the presence of eosinophils and basophils (S, 2020).

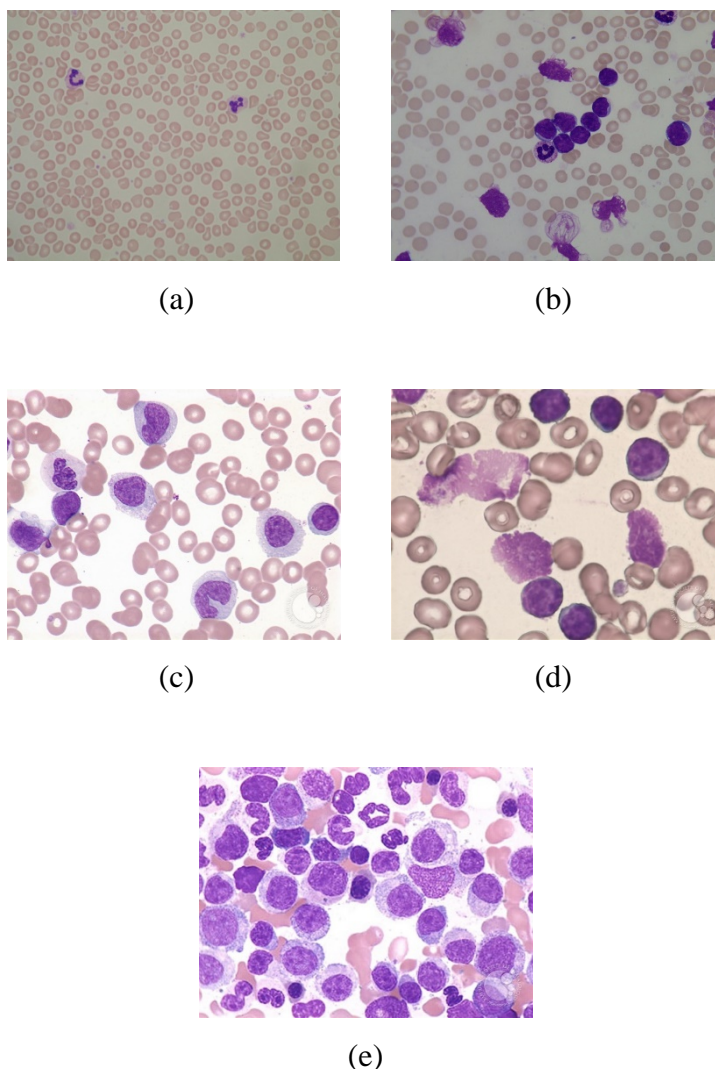












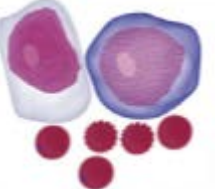
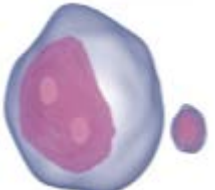
Figure 2.2: (a) Blood Smears of Healthy Blood Cells (Scotti, Labati and Piuri, 2011). (b) ALL (Scotti, Labati and Piuri, 2011). (c) AML (American Society of Hematology, n.d.). (d) CLL (American Society of Hematology, n.d.). (e) CML (American Society of Hematology, n.d.)

**Table 2.1: Morphology of ALL Subtypes According to FAB Classification**  
(S, 2020; Bain, 2015; Ladines-Castro, et al., 2016)

Subtypes	Illustration	Description
L1		<ul style="list-style-type: none"> <li>• Has a homogeneous blast cell population</li> <li>• Has a regular-shaped nucleus</li> <li>• Diffused or condensed chromatin</li> <li>• Minimal or no nucleoli</li> <li>• Cytoplasm is scanty and mild to moderately basophilic</li> </ul>
L2		<ul style="list-style-type: none"> <li>• Large blasts</li> <li>• Nuclei is shaped irregularly</li> <li>• Chromatin is structured heterogeneously and is weakly to strongly basophilic</li> <li>• Large nucleoli</li> <li>• Has more cytoplasm</li> </ul>
L3		<ul style="list-style-type: none"> <li>• Large-sized blasts</li> <li>• Nucleus is surrounded by a copious amount of chromatic vacuole</li> <li>• Chromatin is structured homogeneously and granular</li> <li>• Has prominent nucleoli</li> <li>• Cytoplasm is moderately basophilic</li> </ul>

**Table 2.2: Morphology of AML Subtypes According to FAB Classification**  
(Bain, 2015; Ladines-Castro, et al., 2016)

Subtypes	Illustration	Morphology description
M0		<ul style="list-style-type: none"> <li>• Medium-sized blasts</li> <li>• Circular-shaped nucleus</li> <li>• Has fine chromatin</li> <li>• Has prominent nucleoli</li> <li>• Has agranular and basophilic cytoplasm</li> </ul>
M1		<ul style="list-style-type: none"> <li>• High NC ratio</li> <li>• Medium-sized blasts</li> <li>• Has immature nuclei that is round in shape</li> <li>• Dispersed chromatin</li> <li>• May have one or more nucleoli</li> <li>• Cytoplasm may contain fine azurophilic granulation or Auer rods that are isolated</li> </ul>
M2		<ul style="list-style-type: none"> <li>• High NC ratio</li> <li>• Small to medium-sized blasts</li> <li>• Circular-shaped nucleus</li> <li>• Dispersed chromatin that is immature</li> <li>• May have one or more nucleoli</li> <li>• Basophilic cytoplasm and may contain primary azurophilic granulation or Auer rods</li> </ul>
M3		<ul style="list-style-type: none"> <li>• Nucleus is bean-shaped or bilobed with a deep cleft</li> <li>• Abundant azurophilic granulation</li> <li>• Cytoplasm is weakly basophilic</li> </ul>
M4		<ul style="list-style-type: none"> <li>• Moderate NC ratio</li> <li>• Large blasts</li> <li>• Circular or kidney-shaped nucleus</li> <li>• Has prominent nucleoli</li> </ul>

M5a		<ul style="list-style-type: none"> <li>• Circular-shaped nucleus</li> <li>• Dispersed chromatin that is immature</li> <li>• May have one to three nucleoli</li> <li>• Cytoplasm is abundant and strongly basophilic</li> <li>• Auer rods may be present</li> </ul>
M5b		<ul style="list-style-type: none"> <li>• Circular or kidney-shaped nucleus</li> <li>• Cytoplasm is weakly basophilic, highly granulated, and may contain vacuoles</li> </ul>
M6		<ul style="list-style-type: none"> <li>• Mushroom-shaped cells</li> <li>• Present as a circulating nucleated red blood cells (NRBC)</li> </ul>
M7		<ul style="list-style-type: none"> <li>• Has similar appearance to platelets</li> <li>• Eccentric nucleus</li> <li>• Dispersed chromatin</li> <li>• May have one to three nucleoli</li> <li>• Has agranular and basophilic cytoplasm</li> </ul>

## 2.4 Deep Learning

Deep learning (DL) is a sub-type of machine learning (ML) that is based on algorithms to learn in multiple layers of representations in a hierarchical structure in order to obtain a complex function that can extract the high-level features from the raw data (Deng and Yu, 2014; Mishra and Gupta, 2017). It is mainly inspired by how the nerve cells in the human brain work. Instead of nerve cells, DL models have artificial neurons that interconnect to build multiple layers of artificial neural networks (ANN). Each layer will provide its own interpretation of the input data to map them to their specific labels, without predefined rules designed by humans (Alzubaidi, et al., 2021). ANN mainly has three layers, including the input layer, hidden layer, and output layer.

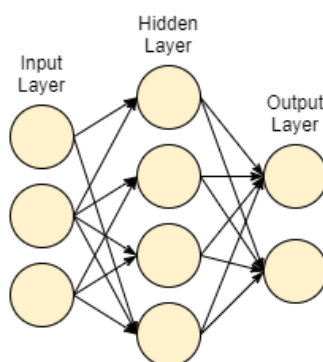


Figure 2.3: Architecture of ANN (Adapted from Mishra and Gupta, 2017)

The implementation of DL can be found in most industries, and it is used in several types of applications from day-to-day tasks such as object detection, image recognition, speech recognition, and language translation, to improving human lives with cancer detection, natural disaster prediction, brain circuitry reconstruction, and predicting the aftermath of mutation in diseases (Alzubaidi, et al., 2021; LeCun, Bengio and Hinton, 2015). Performing classification tasks using DL is also a walk in the park as it is capable of learning feature sets on its own. Compared with conventional machine learning (ML), several stages are required before completing the classification tasks. These stages include data pre-processing, feature extraction, feature selection, learning, and lastly, classification. Inaccurate classification among labels may occur in ML due to the discriminatory feature selection (Alzubaidi, et al., 2021). Figure 2.4 illustrates the difference between ML and DL.

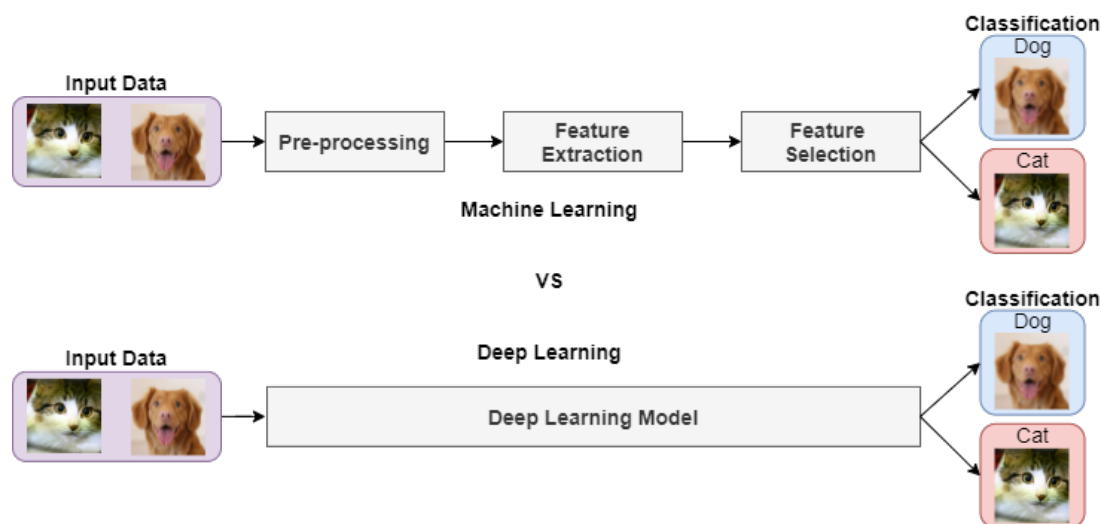


Figure 2.4: Illustration of ML In Comparison with DL (Adapted from Alzubaidi, et al., 2021)

Recent studies also prove that DL models had already surpassed humans in classifying images. This is observed in Figure 2.5, where the top-5 error rate of state-of-the-art DL models is compared with human errors estimated to be 5.1% according to Russakovsky, et al. (2015). The top-5 error rate is the percentage of classification made by the model on a given image where the correct label is not on its top 5 predictions. It is one of the methods used to evaluate machine learning models in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). The annual challenge started from 2010 until present uses a subset of ImageNet with approximately 1000 images for each of the 1000 classes. The images are split into 1.2 million for training, 50 thousand for validation, and 150 thousand for testing (Krizhevsky, Sutskever and Hinton, 2017). The goal for each year is to reduce the error rate of the previous models. However, ImageNet had announced that starting from 2018, the classification of 3D objects will be involved (Wikipedia, 2021). Hence, the DL models that participated in ILSVRC, used for comparison in Figure 2.5, are dated until 2017 only.

The three main techniques for DL include deep supervised learning, deep unsupervised learning, and deep reinforcement learning. There are also hybrid techniques such as semi-supervised learning, while another common method that is widely used is known as deep transfer learning. Deep transfer learning will be discussed more in-depth than other techniques as it will be utilized in this project.



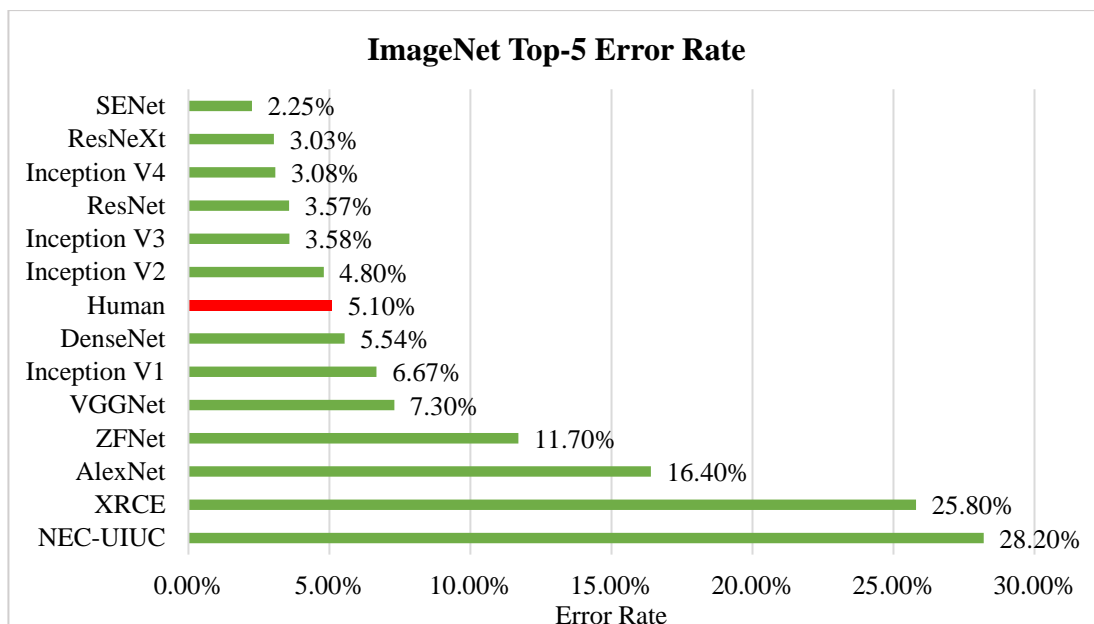


Figure 2.5: ImageNet Top-5 Error Rate of DL Models Compared to Human Errors  
(Adapted from Alzubaidi, 2021)

**Table 2.3: DL Models and Their References**

DL models	ILSVRC Results	Reference
SENet	2017 Winner	Hu, et al., 2019
ResNeXt	2016 1 <sup>st</sup> Runner-up	Xie, et al., 2017
Inception-V4	-	Szegedy, et al., 2017
ResNet	2015 Winner	He, et al., 2015
Inception-V3	2015 1 <sup>st</sup> Runner Up	Szegedy, et al., 2016
Inception-V2	-	Ioffe and Szegedy, 2015
DenseNet	-	Huang, et al., 2017
Inception V1 (GoogLeNet)	2014 Winner	Szegedy, et al., 2015
VGGNet	2014 1 <sup>st</sup> Runner up	Simonyan and Zisserman, 2015
ZFNet	2013 Winner	Zeiler and Fergus, 2014
AlexNet	2012 winner	Krizhevsky, Sutskever and Hinton, 2017
XRCE	2011 winner	Sanchez, et al., 2013
NEC-UIUC	2010 winner	Lin, et al., 2011

### 2.4.1 Deep Supervised Learning

Deep supervised learning is the most common and simplest approach for DL. It is used to train datasets that are labelled. During training, the input data are applied together with the resultant output. The agent then predicts the input to the desired output, optimizing the network's internal parameters using the loss function computed from previous predictions to minimize the error until the desired output is sufficiently met. A few algorithms used for deep supervised learning may include recurrent neural networks (RNN), convolutional neural networks (CNN), and deep neural networks (DNN). This approach benefits in retrieving data or producing a data output from the previous knowledge. However, it suffers from overstraining the decision boundary when the training set lacks samples from an existing class (Alzubaidi, et al., 2021). An illustration of deep supervised learning is shown below.

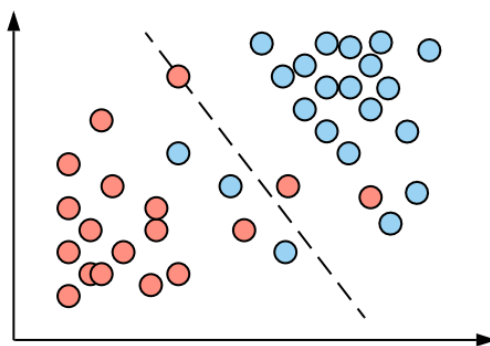


Figure 2.6: Deep Supervised Learning (Qian, et al., 2020)

### 2.4.2 Deep Unsupervised Learning

Deep unsupervised learning is a type of DL approach that involves unlabelled data. This means that only input data is applied in the training process. Internal representation or notable features are learnt instead to uncover the underlying links in the input data. Several algorithms used for deep unsupervised learning include dimensionality reduction, generative adversarial networks, and, most popularly, clustering. The disadvantage of this approach is that it is complex, and the data output

generated may be inaccurate for tasks related to sorting data (Alzubaidi, et al., 2021). An illustration of deep unsupervised learning is shown below.

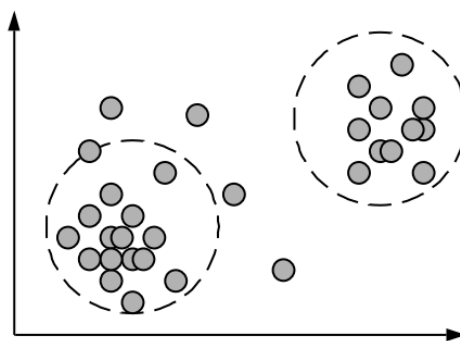


Figure 2.7: Deep Unsupervised Learning (Qian, et al., 2020)

### 2.4.3 Deep Reinforcement Learning

Deep reinforcement learning is another approach for DL that involves the agent interacting with its environment. The agent will make a series of actions whereby, for each iteration, a reward function will be produced from the environment in which the agent uses it to optimize its state for the next iteration (Amiri, et al., 2018). This approach will perform an infinite amount of iterations with an objective to reduce loss while maximizing reward (Neftci and Averbach, 2019). Deep reinforcement learning can be seen used to plan corporate strategy as well as industrial robots. This is because this approach can help identify the preferred action to obtain the maximum reward. However, it can take more time and computing power to achieve it in a larger workspace (Alzubaidi, et al., 2021).

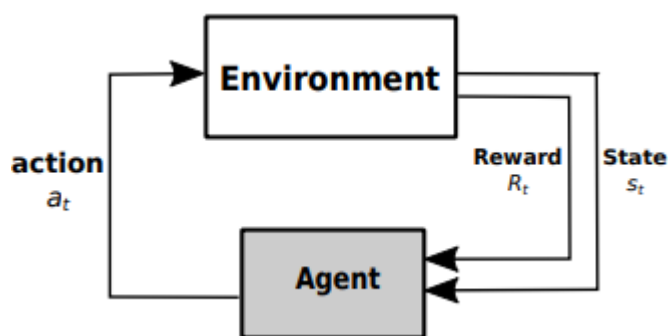


Figure 2.8: Deep Reinforcement Learning (Amiri, et al., 2018)

#### 2.4.4 Deep Semi-supervised Learning

Deep semi-supervised learning is a hybrid approach for DL that involves both labelled and unlabelled data. It contains the pros and cons of both supervised and unsupervised learning. This approach is favourable when labelled data are difficult to collect compared to the widely available unlabelled data. Hence, it can benefit DL where there is a lack of labelled data. The main drawback is that assumptions must be made so that this approach can work effectively. These assumptions include smoothness, cluster, and manifold assumptions (Ouali, Hudelot and Tami, 2020). One application of deep semi-supervised learning can be seen in classification tasks concerning text documents (Alzubaidi, et al., 2021).

#### 2.4.5 Deep Transfer Learning

Following the widespread use of deep architectures, most notably the convolutional neural network (CNN), deep transfer learning becomes increasingly popular for DL. This is because it is an effective approach for DL on undersized annotated datasets where overfitting is a major issue (Tan, et al., 2018). In contrast, traditional approaches often demand vast amount of datasets, which also requires a lot of time and computing resources, to train and build a DL model from scratch (Alzubaidi, et al., 2021). The main idea behind transfer learning is to repurpose existing DL models trained previously for one task to another novel task (Best, Ott and Linstead, 2020). These DL models are also known as pre-trained models.

The two common terminologies used in transfer learning are source and target, whereby each is further described by domain and task. The domain where knowledge will be learned is known as the source domain,  $\mathcal{D}_S$ , while the domain where the knowledge will be transferred to is known as the target domain,  $\mathcal{D}_T$ . The following notations and definitions will match closely on the survey paper done by Pan and Yang (2010).

Domain, denoted as  $\mathcal{D}$ , can be expressed mathematically as  $\mathcal{D} = \{\mathcal{F}, P(X)\}$ . It is observed that there are two components in the expression: a feature space,  $\mathcal{F}$ , and an edge probability distribution,  $P(X)$ , where  $X = \{x_1, \dots, x_m\} \in \mathcal{F}$  (Tan, et al., 2018). For a task concerning binary classification problem,  $\mathcal{F}$  is the space with a collection of all feature vectors,  $x_i$  is the  $i$ th feature vector corresponding to the  $i$ th term, while  $X$  is the particular samples used for training. Generally, if two domains are different, this could mean that either their feature spaces or the marginal probability distributions are different. Then, for a given  $\mathcal{D}$ , a task, denoted as  $\mathcal{T}$ , can be expressed mathematically as  $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ . It also consists of two parts, where the first part is a label space,  $\mathcal{Y}$ . The second part is a predictive function,  $f(\cdot)$ , which is learned from the instances and label training pairs  $\{x_i, y_i\}$ , where  $x_i \in X$  and  $y_i \in \mathcal{Y}$ , and it can also be alternatively viewed as a conditional probability distribution  $P(y/x)$ . The source task is denoted as  $\mathcal{T}_S$  and the source predictive function is denoted as  $f_S(\cdot)$ , whereas the target task is denoted as  $\mathcal{T}_T$  and the target predictive function can be denoted as  $f_T(\cdot)$ . Recalling back to the binary classification problem,  $\mathcal{Y}$  is the collection of all labels that contains true and false, and  $y_i$  can have a value of either true or false (Weiss, Khoshgoftaar and Wang, 2016).

With the notation defined above,  $\mathcal{D}_S$  can be formally expressed as  $\mathcal{D}_S = \{(x_{S1}, y_{S1}), \dots, (x_{Sm}, y_{Sm})\}$ , where  $y_{Si} \in \mathcal{Y}$  is the class label corresponding to the data instance,  $x_{Si} \in \mathcal{F}_S$ . Likewise,  $\mathcal{D}_T$  can be formally expressed as  $\mathcal{D}_T = \{(x_{T1}, y_{T1}), \dots, (x_{Tm}, y_{Tm})\}$ , where  $y_{Ti} \in \mathcal{Y}$  is the class label corresponding to the data instance,  $x_{Ti} \in \mathcal{F}_T$ . Finally, the definition of transfer learning can be formally described. For a given  $\mathcal{D}_S$  and its corresponding learning tasks  $\mathcal{T}_S$ , and a  $\mathcal{D}_T$  and its corresponding learning tasks  $\mathcal{T}_T$  the goal of transfer learning is to enhance the performance of  $f_T(\cdot)$  by leveraging the knowledge in  $\mathcal{D}_S$  and  $\mathcal{T}_S$ , where  $\mathcal{D}_S \neq \mathcal{D}_T$  or  $\mathcal{T}_S \neq \mathcal{T}_T$  (Tan, et al., 2018). The performance of transfer learning can be categorized into positive transfer and negative transfer. For a given predictive learner  $f_{T1}(\cdot)$  that is trained with  $\mathcal{D}_T$  only, and another predictive learner  $f_{T2}(\cdot)$  that is trained with  $\mathcal{D}_S$  and  $\mathcal{D}_T$  combined, the transfer is said to be negative if  $f_{T1}(\cdot)$  performs better than  $f_{T2}(\cdot)$ , whereas the transfer is said to be positive if  $f_{T2}(\cdot)$  performs better than  $f_{T1}(\cdot)$  (Weiss, Khoshgoftaar and Wang, 2016). Figure 2.9 illustrates the transfer learning process.

There are two types of approaches to perform transfer learning using pre-trained models: feature extraction, and fine-tuning. The former extracts the feature maps from the pre-trained model to be built on top of a shallow model, while the latter makes fine adjustments on the pre-trained model to increase its accuracy and performance whilst retaining the initial weights learned by the model for the new task (Mustafid, Pamuji and Helmiyah, 2020). All in all, transfer learning benefits in requiring lesser dataset and time for training while improving performance and network generalization (Alzubaidi, et al., 2021).

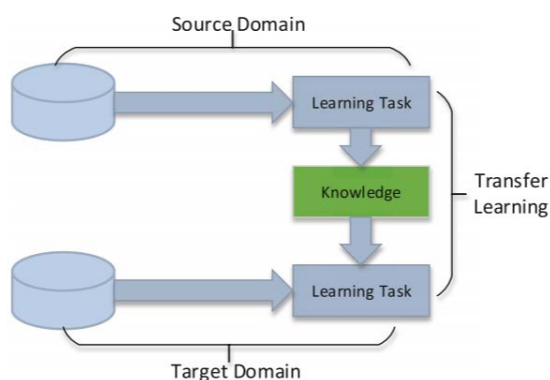


Figure 2.9: Transfer Learning Process (Tan, et al., 2018)

## 2.5 Types of Artificial Neural Networks

Few of the commonly known artificial neural networks (ANN) include recursive neural network (RvNN), recurrent neural network (RNN), and convolutional neural network (CNN). CNN will be discussed in-depth compared to others as it is the most widely used type of artificial neural network for DL.

### 2.5.1 Recursive Neural Network

Recursive neural network (RvNN) is a type of artificial neural network that can predict outputs from data that are structured hierarchically. It can process information of

different sizes with various topologies such as trees and graphs compared to conventional techniques that are based on features, which use fixed-size vectors to encode the information relevant to the problem (Chinea, 2009). Socher, et al. (2011) provided some examples of the application of RvNN such as parsing scene images, which can be helpful for computer vision. Figure 2.10 illustrates how RvNN parses scene images.

The approach of RvNN is to over-segment the image into smaller regions of interest, then the features of the image are extracted and mapped into a semantic space. The semantic representations of each region are then fed into the RvNN where it will compute a score. The ones with the highest score will be merged to the neighbouring units, producing a larger unit. A new feature and the class labels that represent the unit are generated for every large unit produced. The merging process happens recursively on the same neural network. As a result, an RvNN tree structure is implicitly created for each merging decision, whereby the final output is the complete scene image, which is said to be the root of the structure (Alzubaidi, et al., 2021). RvNN is still uncommon among the research community due to its intricately complex characteristics, requiring a steep learning curve (Chinea, 2009).

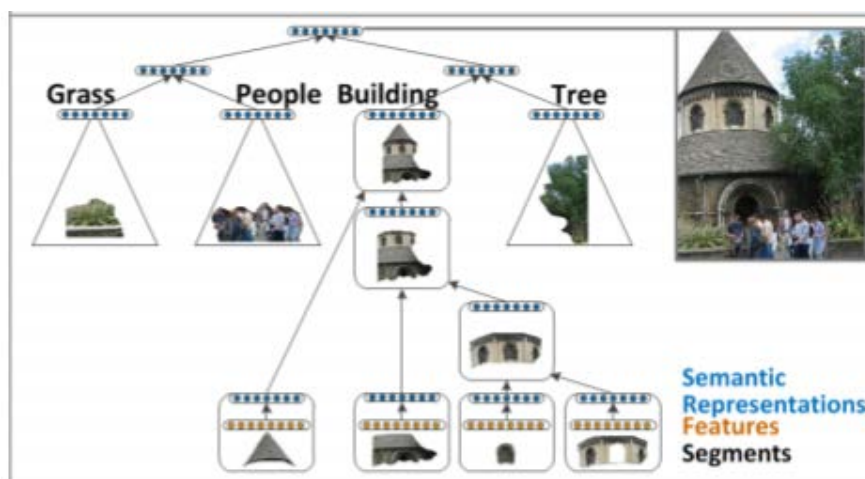


Figure 2.10: Illustration of How RvNN Parses Scene Images (Socher, et al., 2011)

## 2.5.2 Recurrent Neural Network

Recurrent neural network (RNN) is a type of artificial neural network that deals with information that is time-continuous by implementing feedback to feedforward neural networks (FFNN). The purpose of feedback neural networks is to possess the idea similar to the short-term and long-term memory demonstrated by humans. In the case of RNN, it uses past outputs to process the present input. Hence, RNN is mainly used in speech recognition, human activity recognition, and language translation (Rezk, et al., 2020). There are three collections of layers in RNN: input layers denoted as  $x$ , recurrent or hidden layers denoted as  $h$ , and output layers denoted as  $y$ .

Though RNN may seem like it has a deep network, whereby the input at time  $m < t$  propagates through multiple nonlinear layers before producing the output at time  $t$ . However, upon unfolding the network through steps of time, it has a temporal structure with shallow functions. These functions include input-to-hidden ( $x_t \rightarrow h_t$ ), hidden-to-output ( $h_t \rightarrow y_t$ ), and hidden-to-hidden ( $h_{t-1} \rightarrow h_t$ ) (Pascanu, et al., 2014). RNN can be unfolded into different types of structure as shown in Figure 2.11: one-to-many, many-to-one, and many-to-many. An RNN is called a deep transition RNN if additional nonlinear layers are stacked within the hidden layer; it is called a deep output RNN if additional nonlinear layers are stacked between the output and the hidden layer (Rezk, et al., 2020).

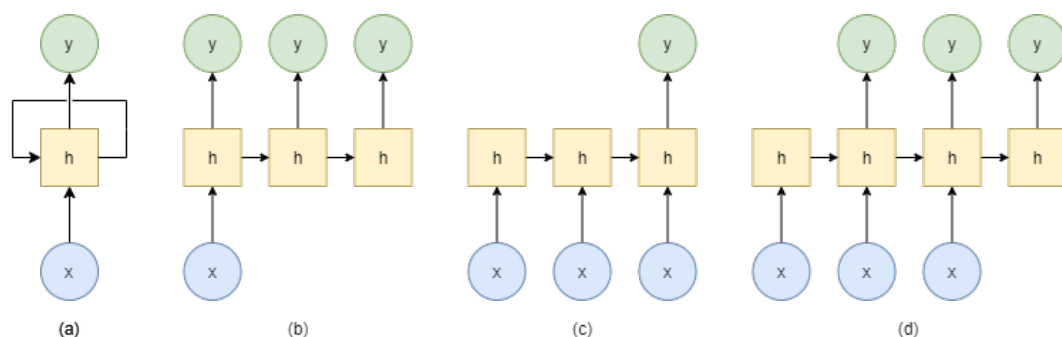


Figure 2.11: (a) Typical RNN Structure (b) One-To-Many Temporal Structure of RNN (c) Many-To-One Temporal Structure of RNN (d) Many-To-Many Temporal Structure of RNN (Adapted from Rezk, et al., 2020; Su and Li, 2019)



### 2.5.3 Convolutional Neural Network

Convolutional neural network (CNN, or ConvNet) is a type of neural network designed specifically to process two-dimensional inputs, which includes images and videos. It is the first artificial neural network that can truly accomplish DL where it successfully trained hierarchically structured layers in a robust way (Mishra and Gupta, 2017). An illustration of the CNN architecture is shown in Figure 2.12. It is an architecture inspired by the structure of the visual system of humans and animals, discovered in 1962 by Hubel and Wiesel, and digitalized in 1980 by Fukushima.

Through the discovery from the receptive fields of the cells in the primary visual cortex of a cat, Hubel and Wiesel proposed a hierarchy model of the visual neural network. The structure starts from the lateral geniculate body (LGB) to simple cells, followed by complex cells, lower-order hypercomplex cells, and finally, to the higher-order hypercomplex cells (Fukushima, 1980). Then, Fukushima presented an artificial neural network model called Neocognitron that followed the works of Hubel and Wiesel. Neocognitron was one of the first models that can be simulated on a computer. It is also considered the earliest version of CNNs since it was based on the hierarchical, multi-layered structure of neurons for image processing (Shamsaldin, et al., 2019).

CNNs are still obscured from the public until 1990. LeCun, et al. (1990) brought the idea to the limelight by using a multi-layered artificial neural network, known as LeNet, to recognize and classify handwritten digits. LeNet was the first CNN architecture that is able to perform image classification using deep learning. It utilizes an algorithm known as back-propagation to train the model, allowing patterns to be recognized from raw pixels. Though LeNet is incapable of solving complex classification problems, it instilled interest among the research community, paving the way for upcoming CNNs (Shamsaldin, et al., 2019).

One of the main interests in employing CNNs is the concept of shared weights, which reduced the number of parameters that had to be learned, enabling a better generalization and avoiding overfitting problems. The utilization of temporal and spatial relationships in the CNN architecture is also an effort in reducing the number

of parameters (Mishra and Gupta, 2017). Besides that, the classification stage is also combined with the feature extraction stage, which expedites the training process and ensuring the optimum output. Furthermore, CNNs also allows large-scale networks to be implemented easier compared to other types of neural networks (Alzubaidi, et al., 2021). Hence, due to the exceptional performance that CNNs is able to provide, it is currently widely applied in multiple applications such as face detection, object detection, image classification, facial expression recognition, speech recognition, and so on (Indolia, et al., 2019).

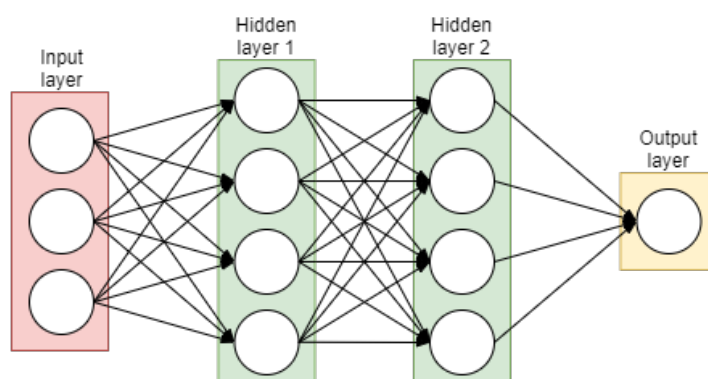


Figure 2.12: CNN Architecture (Adapted from Mishra and Gupta, 2017)

## 2.6 CNN Architecture

The typical architecture of CNN consists of several alternative layers of convolution layers and pooling layers, followed by activation function or non-linearity layer, and lastly, a fully connected layer (Indolia, et al., 2018). Other layers such as batch normalization and dropout are also added as regulatory units to improve the performance of CNN (Khan, et al., 2020).

The input format of CNN is a three-dimensional vector. The dimensions include height, width, and depth. The height and width will always have an equal length such that the three-dimensional input can be expressed as  $a \times a \times m$ , where  $m$  is the channel number. Figure 2.13 illustrates an example of image classification using CNN architecture (Alzubaidi, et al., 2021).

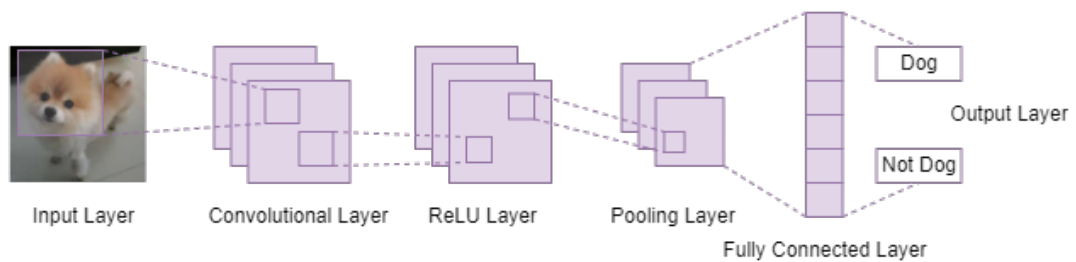


Figure 2.13: Image Classification Using CNN Architecture (Adapted from Alzubaidi, et al., 2021)

### 2.6.1 Convolution Layer

The convolutional layer is the core layer of the CNN architecture (Alzubaidi, et al., 2021). As the name suggests, it utilizes convolution, an operation that is widely used in applications such as image and signal processing, digital data processing, computer vision, and other mathematical problems. The convolution operation is a mathematical process that generates the third function from two primary functions  $f$  and  $g$ . This third function is the expression that describes how one of the functions modifies the other, providing the overlapping area between the two functions (Behl, Bhatia and Puri, 2014). An illustration of the convolution between two functions is shown in the figure below.

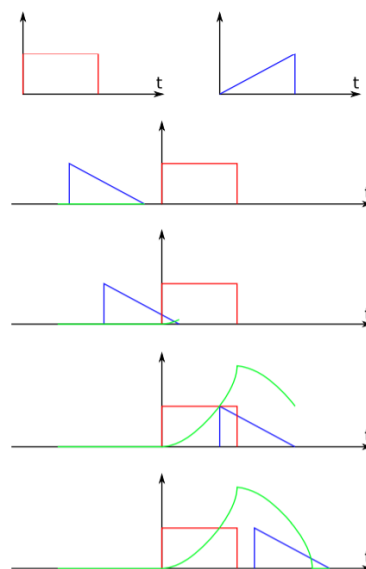


Figure 2.14: Convolution Between Two Functions (Pihlajamäki, 2009)

According to Behl, Bhatia and Puri (2014), the convolution operation between the functions  $f$  and  $g$ , written with an asterisk (\*) as the operator, can be mathematically defined as shown below.

$$(f * g)(t) := \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (2.1)$$

$$:= \int_{-\infty}^{\infty} f(t - \tau)g(\tau)d\tau \quad (2.2)$$

As shown in the expression, convolution can be defined as integrating the outcome after one function multiplied by another that is reversed and shifted (Behl, Bhatia and Puri, 2014).

In CNNs, convolution operations are used to perform feature extraction. The convolutional layers contain convolutional filters, also known as kernels, to perform the convolution operations. The number of kernels used is typically 32 to 512 so that features can be learned and in parallel using 32 to 512 ways to see the input data (Brownlee, 2019). The kernel is a grid of numbers, where each value in the grid is known as weights, and it is used to convolve the array of numbers at the input, also known as tensors. The initial weights of the kernel are selected arbitrarily but will be adjusted after each batch of training (Alzubaidi, et al., 2021). The convolution operation is determined by a few hyperparameters of the kernel such as size, number, and stride. The common choice for the size of the kernel is typically  $3 \times 3$ , but it can also be  $5 \times 5$ , or  $7 \times 7$ . On the other hand, the number of the kernel is decided depending on the complexity of the datasets. Besides, the stride is typically set as 1 pixel, meaning that the kernel will shift 1 pixel for each iteration of the convolution operation. Still, a larger stride can also be used to achieve sub-sampling of the feature map (Yamashita, et al., 2018).

To expand the knowledge on convolution operation in CNNs, the following example will be used. Given an  $5 \times 5$  grey-scaled image convolving a kernel of arbitrary weights with a size of  $3 \times 3$  pixel and stride of 1 pixel as shown in Figure 2.15. For each iteration of convolution operation, the kernel will perform horizontal and vertical shifts over the input tensor, dissecting the input image into smaller regions

called the receptive fields while evaluating the dot product between the receptive field and the kernel weights. The feature map generated is the output of the dot products.

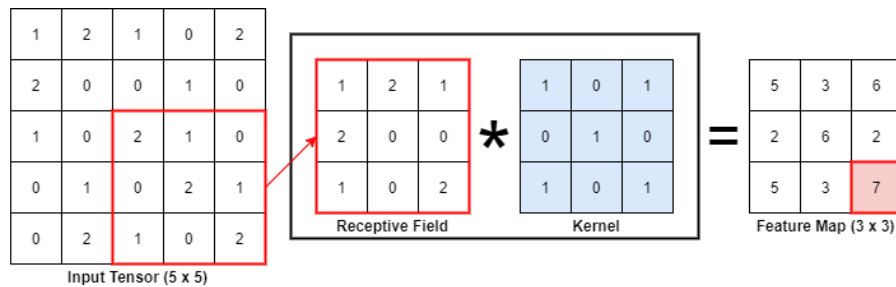


Figure 2.15: Convolution Operation Between a Kernel and An Input Tensor  
(Adapted from Reynolds, 2019; Yamashita, et al., 2018)

Since the convolution operation is done without padding, the feature map generated appears to have a reduced height and width compared to the input tensor. This will significantly reduce the performance of CNN if the size of feature maps keeps getting smaller for each complete convolution operation. However, this problem can be easily solved with the help of the padding technique. By zero-padding the input tensor, the outermost input tensor can then convolve with the centre of the kernel, allowing the feature map to retain the size of the input tensor.

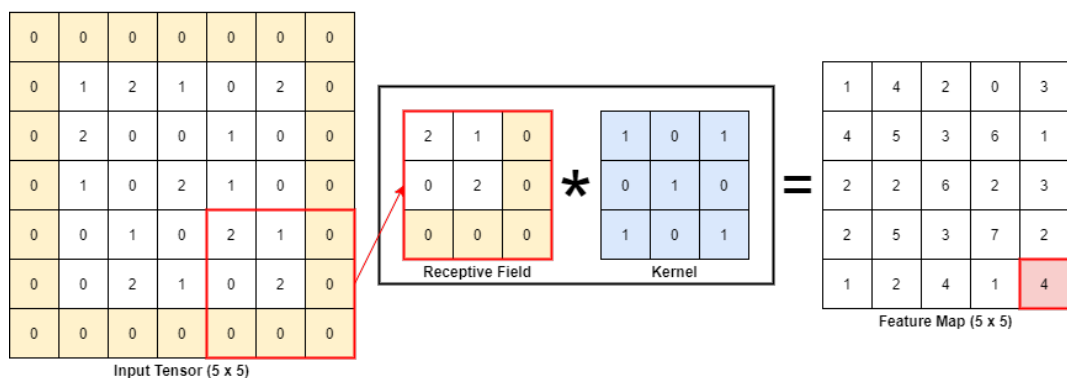


Figure 2.16: Zero Padding Before Performing Convolution Operation (Adapted from Reynolds, 2019; Yamashita, et al., 2018)

All in all, the convolutional layer is the reason why CNNs are preferred over other neural network architectures because they manifest the ability of shared weights, which avoids the need to learn weights for each available neuron in the layer. Hence, the time required for training is reduced. In addition, since the number of weights

between two neighbouring layers is typically less, the weights can be stored in a sufficiently small amount of memory, which reduces the computational cost for training compared to other neural network models.

### 2.6.2 Pooling Layer

The pooling layer is responsible for down-sampling the feature map that is generated from the convolutional layer (Yamashita, et al., 2018). The pooling operation works by gathering the dominant response within a region of interest and include them in the output vector, thereby shrinking the resolution of the feature map to generate a smaller feature map (Albelwi and Mahmood, 2017; Alzubaidi, et al., 2021). This resolution reduction helps decrease the number of learnable parameters and introduces translational invariances to small shifts and distortions (Indolia, et al., 2018; Yamashita, et al., 2018). However, the main drawback of the pooling layer is that it may reduce the performance of the CNN because it approximates the location of the features (Alzubaidi, et al., 2021; Khan, et al., 2020). The hyperparameters of the pooling layer are also somewhat similar to that of the convolutional layer. These hyperparameters include the size of the filter, padding, and stride (Yamashita, et al., 2018).

A few pooling techniques can be employed in the pooling layer, but those commonly used are max pooling and global average pooling (Alzubaidi, et al., 2021). Max pooling works by extracting the maximum value of the region of interest and generates an output vector based on those values. The typical filter size of max pooling used is  $2 \times 2$ . The striding of the filter is also commonly set to 2, which will reduce the resolution of the feature map by twice the original size. On the other hand, global average pooling works by averaging all the elements in the feature map, thus generating a feature map of size  $1 \times 1$  pixel. It is usually used prior to the fully connected layers. By utilizing global average pooling, trainable parameters will be reduced and input of various sizes can be fed into the CNN (Yamashita, et al., 2018). Both pooling operations only affect the height and width but not the depth of the feature map. An illustration of both pooling operations is shown in Figure 2.17.

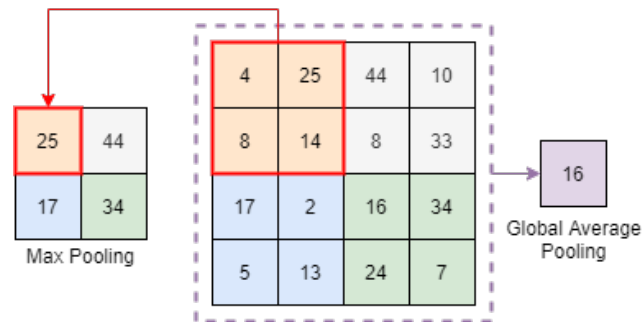


Figure 2.17: Illustration of Max Pooling and Global Average Pooling (Adapted from Alzubaidi, et al., 2021; Yamashita, et al., 2018)

### 2.6.3 Activation Function

Activation functions are used to map the input to the output, and it also decides whether to activate a neuron or not by computing the input neuron's weighted sum and its bias (Alzubaidi, et al., 2021; Szandała, 2020). There are two types of activation function: linear activation function that provides a constant output, and non-linear activation function, which is widely used in building neural networks due to the variations they create. A non-linear activation function is differentiable, which makes it useful for backpropagation algorithms. It also enables network generalization, allowing the network to adapt to different types of data (Feng and Lu, 2019).

The most commonly used non-linear activation functions are sigmoid, tanh, and the popularly used rectified linear unit (ReLU). The sigmoid is a function that takes in real numbers as its input and generates an output that ranges between 0 and 1 (Alzubaidi, et al., 2021). According to Feng and Lu (2019), it can be mathematically expressed as shown below.

$$f(x)_{sigmoid} = \frac{1}{1 + e^{-x}} \quad (2.4)$$

The sigmoid function curve, as shown in Figure 2.18 has an S-shape, revealing that the slight shifts in the input can significantly affect the output when the input is near 0. However, when the input is diverging away from 0, the output response obtained is

less pronounced. This will pose a problem known as “vanishing gradients” where the gradient becomes smaller to the point that it reaches its asymptotic ends, and the neural network is said to be saturated and will stop learning further (Rakitienskaia and Engelbrecht, 2015). Hence, it is important to carefully initialize the sigmoid function’s weights so that saturation will not happen (Feng and Lu, 2019).

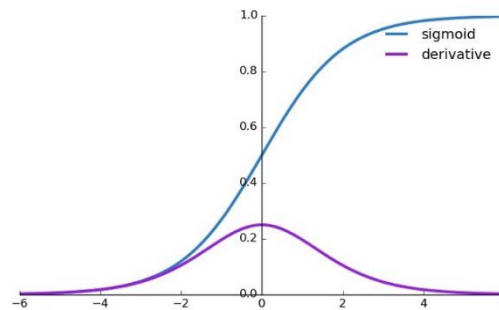


Figure 2.18: Sigmoid Function and Its Derivative (Omkar, 2019)

Next, the tanh function or Tangent Hyperbolic Function is a function that is similarly like the sigmoid function where it takes in real numbers as an input, but it generates an output that ranges between -1 and 1 (Alzubaidi, et al., 2021). According to Feng and Lu (2019), it can be mathematically expressed as shown below.

$$f(x)_{\tanh} = \frac{e^x + e^{-x}}{e^x - e^{-x}} \quad (2.3)$$

The curve of the tanh function and its derivative is illustrated in the figure below.

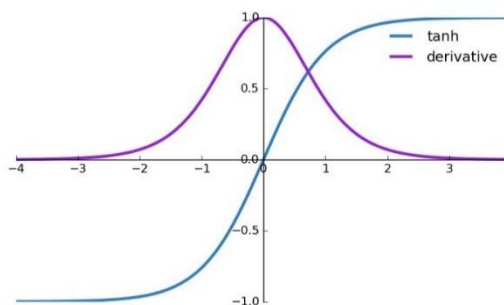


Figure 2.19: Tanh Function and Its Derivative (Omkar, 2019)



The “vanishing gradient” in the sigmoid function can also be observed in the tanh function. However, tanh function is able to concentrate the data and allows an easier training process because the output's mean is near 0 (Feng and Lu, 2019).

Lastly, ReLU is a function that outputs the input data if it is positive, and generates a zero if the input data is negative. According to Feng and Lu (2019), it can be mathematically expressed as shown below.

$$f(x)_{ReLU} = \max(0, x) = \begin{cases} x, & x > 0 \\ 0, & x < 0 \end{cases} \quad (2.4)$$

The curve of the ReLU function and its derivative is illustrated in the figure below.

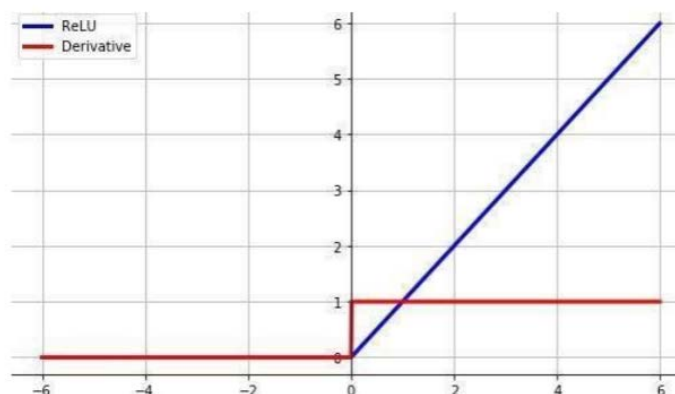


Figure 2.20: ReLU Function and Its Derivative (Szandała, 2020)

The benefits of utilizing the ReLU function are that the “vanishing gradient” problem observed in the activations discussed earlier is omitted. It also reduces the computational load that significantly speeds up the training process. However, when a larger gradient passes through the network using the backpropagation algorithm, the weights of the neurons for negative inputs will not be adjusted, or in another sense, the neurons become inactive and “die” (Feng and Lu, 2019; Szandała, 2020). This is known as the “Dying ReLU” problem (Alzubaidi, et al., 2021). The solution to this problem is to use one of the multiple variants of the ReLU functions, such as the Leaky ReLU and the Parametric ReLU (PReLU). In the Leaky ReLU, a small constant  $\alpha$ , typically with a value of 0.01, is added to ensure that the negative inputs are not

ignored by creating a small incline at the negative side of the function (Szandała, 2020). According to Feng and Lu (2019), it can be mathematically expressed as shown below.

$$f(x)_{LeakyReLU} = \begin{cases} x, & x > 0 \\ \alpha x, & x < 0 \end{cases} \quad (2.5)$$

On the other hand, PReLU is somewhat similar to the Leaky ReLU but the value of  $\alpha$  is learned through the training process (Feng and Lu, 2019). According to Feng and Lu (2019), it can be mathematically expressed as shown below.

$$f(x)_{PReLU} = \begin{cases} x, & x > 0 \\ \alpha x, & x < 0 \end{cases} \quad (2.6)$$

The figure below illustrates the curve of the Leaky ReLU and the PReLU functions.

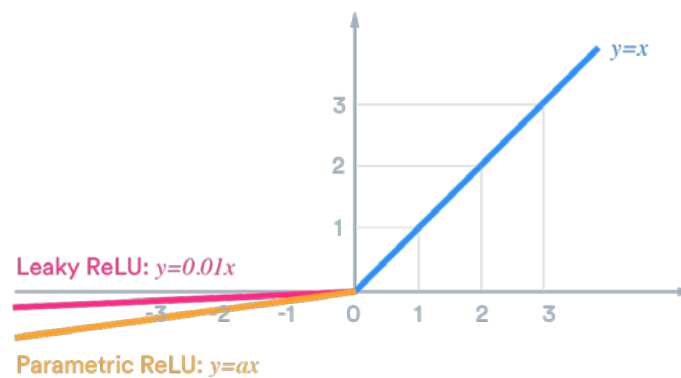


Figure 2.21: Leaky ReLU and PReLU Functions (Omkar, 2019)

#### 2.6.4 Batch Normalization

Batch normalization is used to reduce the shift in network activation distribution caused by the change in network parameters in the course of training. This shift can also be called the internal covariate shift (Ioffe and Szegedy, 2015). When the shift is very high, the model will take more time to converge the inputs from multiple neurons to their target neurons (Mpitsos and Burton, 1992). Hence, batch normalization addresses the issue by normalizing the output at each layer such that the mean is subtracted and the standard deviation is divided from it (Alzubaidi, et al., 2021).

According to Ioffe and Szegedy (2015), batch normalization can be mathematically expressed as shown in Equation 2.7. The normalized feature-map is denoted as  $\hat{x}_i$  while the input feature-map is denoted as  $x_i$ . The mini-batch mean and variance for the feature maps are denoted as  $\mu_B$  and  $\sigma_B^2$  respectively. Furthermore,  $\epsilon$  is added to prevent being divided by zero, allowing for numerical stability.

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2.7)$$

Batch normalization benefits in accelerating the speed to train the network, improving network generalization, and allows for a faster training rate (Bjorck, et al., 2018).

### 2.6.5 Dropout

Dropout is a method that addresses overfitting issues and improves network generalization by introducing regularization into the network (Khan, et al., 2020). Overfitting is when a network matches the training data too well and predicts poorly on the testing data. In other words, the training data, including any noise in it, are memorized instead of learning their abstract features (Ying, 2019). Hence, dropout is utilized to randomly select nodes and remove them, along with their connections, from the network. The hyperparameter to control how high the intensity of the dropout should be is known as the dropout rate  $p$ . There will be no dropout when  $p = 1$ , while a lower  $p$  will have a higher intensity of dropout. The typical value of  $p$  ranges between 0.5 and 0.8. An illustration of how dropout works is shown in Figure 2.22. After dropout, various thinned networks will be generated, but only one network with smaller weights will be used. The selected network is therefore regarded as a close approximation to all of the network that is proposed. The main drawback of applying dropout is that it elongates the training time of 2 to 3 times compared to training a standard network (Srivastava, et al., 2014).

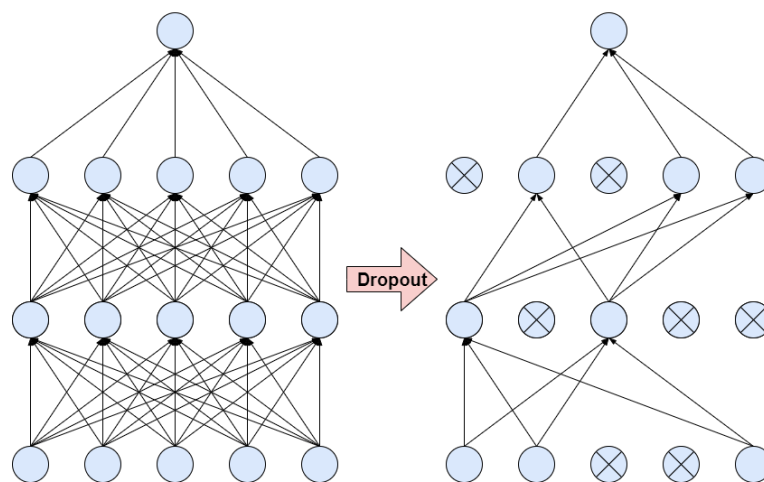


Figure 2.22: Illustration of Dropout (Adapted from Srivastava, et al., 2014)

### 2.6.6 Fully Connected Layer

The fully connected layer is commonly found at the end of a CNN architecture (Alzubaidi, et al., 2021). This layer consists of one or more dense layers whereby each node of the layers is interconnected with one another, with learnable weights. The input of this layer is typically the flattened output from the previous layer, which is an array of numbers that is one-dimensional (1D). Whereas the output of this layer is the probability to the class labels given that the task at hand is a classification problem. The last layer of the fully connected layer usually has the number of nodes that is the same as the number of classes (Yamashita, et al., 2018).

## 2.7 Types of CNN Architectures

Following the rise of the LeNet, more and more deep CNN architectures started to surface to best the benchmark set by their predecessors. ILSVRC is one of the main driving forces in this advancement, which enabled the birth of a few state-of-the-art CNN architectures such as AlexNet, ZFNet, GoogLeNet, ResNet, ResNeXt, SeNet, and DenseNet.

### 2.7.1 AlexNet

AlexNet was proposed by Krizhevsky, Sutskever and Hinton in 2012 and won first place in ILSVRC 2012 with a minimum 15.3% top-5 error rate, averaging around 16.4%. Although LeNet is the pioneer of deep CNN architecture, AlexNet was able to perform better than LeNet in tasks related to recognizing and classifying images. AlexNet was able to tackle the tasks even with images with diverse classes, while LeNet is limited to handwritten digits (Khan, et al., 2020). Thus, AlexNet surpassed LeNet, and it is regarded as the first deep CNN architecture.

AlexNet has a deeper network than LeNet, with three additional layers from LeNet that only has five layers. This enables AlexNet to be applicable in images with various types of categories. The increase in depth can improve network generalization, but there will be a greater chance of overfitting issues. This issue was able to be overcome by implementing dropout, local response normalization, as well as overlapping pooling (Khan, et al., 2020; Krizhevsky, Sutskever and Hinton, 2017). Furthermore, non-saturating activation functions such as ReLU function is also used to avoid the “vanishing gradient” problem, and enhancing the convergence rate of the network (Alzubaidi, et al., 2021). In addition, AlexNet was also trained on two Graphical Processing Units (GPU), both NVIDIA GTX 580 with 3GB of memory, to accelerate the training process of the network. Moreover, to further improve its performance, larger filter sizes such as  $5 \times 5$  and  $11 \times 11$  are utilized at the first few layers (Khan et al., 2020). The figure below illustrates the architecture of AlexNet.

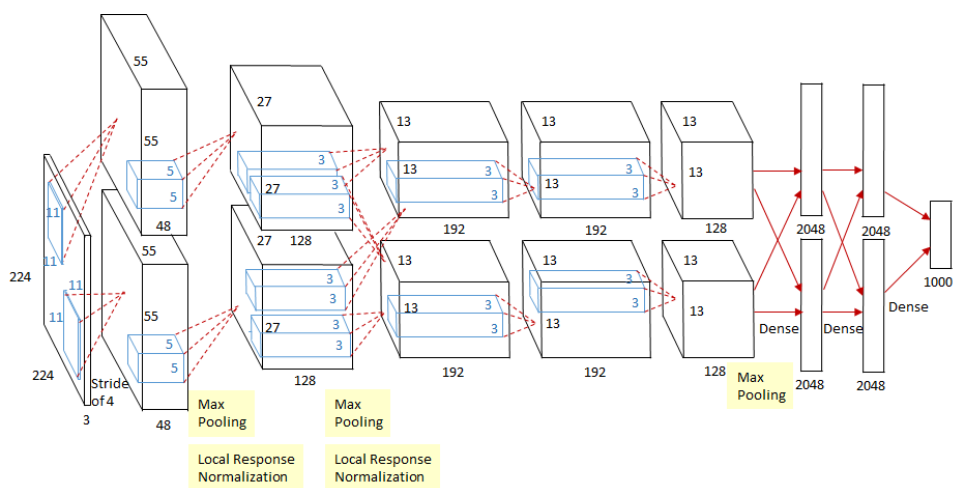


Figure 2.23: AlexNet Architecture (Tsang, 2018)

## 2.7.2 ZFNet

ZFNet was proposed by Zeiler and Fergus in 2013 and emerges as the winner of ILSVRC 2013 with a top-5 error rate of 11.7%. Before Zeiler and Fergus introduced the multilayer Deconvolutional Neural Network (DeconvNet), the insights on how the CNN can achieve its performance are unknown, prompting to train CNNs in a heuristic approach. With DeconvNet, the internal activity of the network can be visualized, allowing the evolution of features to be observed so that potential problems can be diagnosed and debugged easier (Zeiler and Fergus, 2014).

DeconvNet still operates like a standard forward pass CNN, but the locations of convolutional and pooling layers are swapped. This approach allows image patterns to be visually observable and reveals the feature representations learned at each layer in a neuron-level interpretation. Zeiler and Fergus demonstrated the idea by experimenting with it on AlexNet and came out with the architecture as shown in Figure 2.24. They discovered that most of the neurons are inactive with a few active neurons in the network. They then tweak the model and are able to achieve a top-5 error rate of 14.8% by visualizing and adjusting the network parameters (Khan, et al., 2020). All in all, the concept behind ZFNet is much like supervised learning whereby visualizations are used to monitor how the model is learning, and then debugging or tweaking the network parameters to achieve the desired performance (Alzubaidi, et al., 2021).

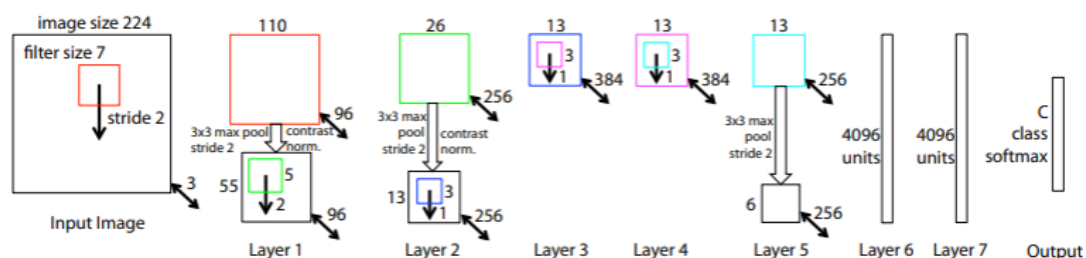


Figure 2.24: ZFNet Architecture (Zeiler and Fergus, 2014)

### 2.7.3 GoogLeNet

GoogLeNet, also known as Inception-V1, was proposed in 2014 by researchers at Google in collaboration with a few other universities. It was the winner of ILSVRC 2014 with a top-5 error rate of 6.67%. With an ambition to achieve a lower power consumption, reducing memory usage, and decrease the number of trainable parameters while maintaining a minimum budget, Szegedy, et al. proposed the idea of an inception module that performs feature extraction at multiple scales using split, transform, and merge functions. This idea is similar to the Network in Network (NIN) architecture proposed by Lin, et al. in 2013, in which micro neural networks are used to replace the conventional convolutional layers. Instead, GoogLeNet used smaller blocks of filters consisting of several sizes such as  $1 \times 1$ ,  $3 \times 3$ , and  $5 \times 5$  to replace the convolutional layers. Spatial information from the roughest to the finest detail at various scales are then learned by the blocks. However, using a larger filter such as the  $5 \times 5$  cause the network to suffer from representational bottleneck whereby useful information may be lost due to the significant reduce in feature space (Alzubaidi, et al., 2021).

On the other hand, the convolutional filters of size  $1 \times 1$  pixel is heavily used in the GoogLeNet architecture, and they are placed before kernels with a larger size to remove bottlenecks during computation. This approach allows the depth and width of the network to be increased without incurring substantial performance penalties. (Szegedy, et al., 2015). Furthermore, GoogLeNet utilizes an architecture that is sparsely connected instead of a fully connected one. This is because the fully connected architecture has limitations in hardware implementations due to their higher complexity, which requires a larger footprint, thus resulting in higher power consumption (Ardakani, Condo and Gross, 2017). With a sparsely connected architecture, the computational cost can be reduced by neglecting channels that are irrelevant, as well as rectifying the problem of having redundant information. The downside of this topology is that any required modifications need to be done from module to module. In addition, contrary to the use of a fully connected layer in the last layer as observed in AlexNet and ZFNet, a global average pooling layer is used instead to reduce the density of the connections (Khan, et al., 2020). These fine-tunings on the parameters are able to significantly reduce a total of 12 million trainable parameters

compared to AlexNet (Indolia, et al., 2018). On top of that, auxiliary classifiers are also implemented to improve the convergence rate of the network (Khan, et al., 2020). With that said, the overall architecture of GoogLeNet contains a total of 22 layers with almost 100 independent building blocks to build the network, and it is shown in the figure below.

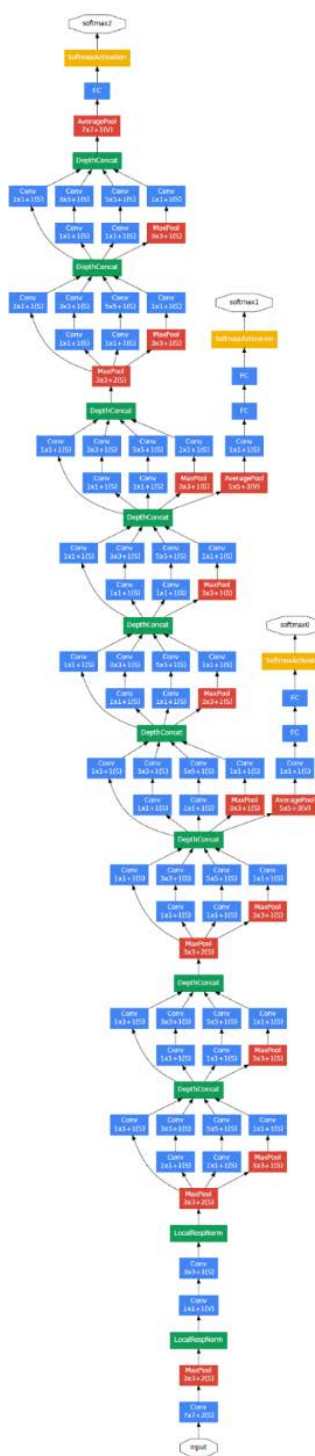


Figure 2.25: GoogLeNet Architecture (Szegedy, et al., 2015)



In hopes to tackle the problems encountered in GoogLeNet, Szegedy, et al. had given the architecture a few upgrades, namely Inception-V2 in 2015, Inception-V3 in 2015, and Inception-V4 in 2016. Inception-V2 is used to overcome representational bottlenecks by factorizing the larger convolutional filters to smaller in size. An example would be to reduce the filter of size  $5 \times 5$  into two filters of size  $3 \times 3$  as shown in the figure below.

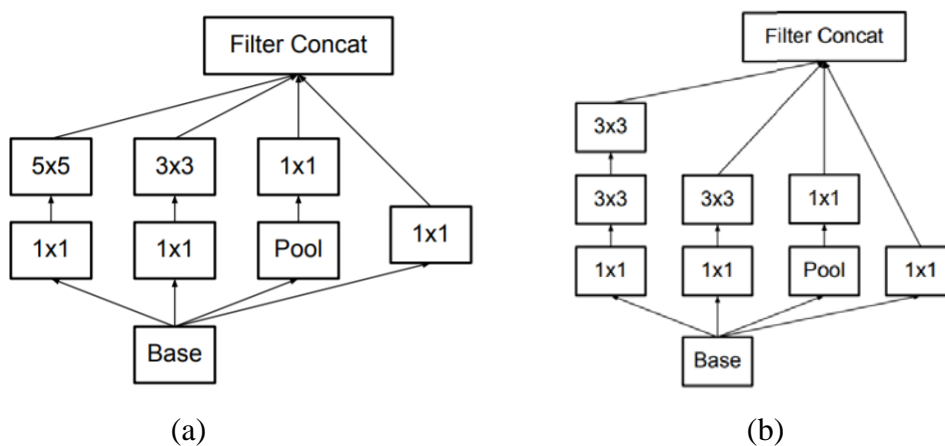


Figure 2.26: (a) Unfactorized Inception Module (b) Factorized Inception Module  
Where Filter Of Size  $5 \times 5$  Is Replaced with Two Filters of Size  $3 \times 3$   
(Szegedy, et al., 2015)

This factorization also helped to reduce the number of trainable parameters by weight sharing, and is able to effectively reduce the computational cost by 28%. In addition, the filter of size  $n \times n$  can be replaced with asymmetric convolutional filter such that a filter of size  $1 \times n$  followed by a filter of size  $n \times 1$  is used. Figure 2.27 shows a filter of size  $3 \times 3$  replaced with a filter of size  $1 \times 3$  followed by a filter of size  $3 \times 1$ . This approach can further reduce the computational cost by 33%. For a coarser grid, the asymmetric convolutional filters are expanded instead of stacking on top of each other to produce sparse representations of multiple dimensions. The expanded Inception module is shown in Figure 2.28. All in all, Inception-V2 is able to obtain a top-5 error rate of 4.8% (Szegedy, et al., 2015).

Inception-V3 is, in a sense, Inception-V2 but with a few more upgrades. It uses the Root Mean Square Propagation (RMSProp) optimizer, and performs label

smoothing to improve model regularization. It also utilizes factorized convolutional filters of size  $7 \times 7$ , and batch normalized auxiliary classifiers to accelerate the rate of convergence. All these newer upgrades enabled Inception-V3 to obtain a top-5 error rate of 3.58% (Szegedy, et al., 2015).

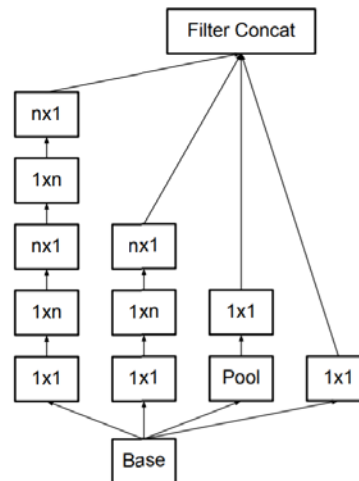


Figure 2.27: Inception Module with Asymmetric Convolutional Filters  
(Szegedy, et al., 2015)

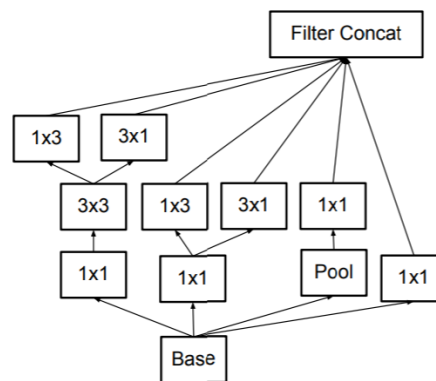


Figure 2.28: Expanded Inception Module (Szegedy, et al., 2015)

The evolution of Inception-V4 came about when Szegedy, et al. found out that the previous networks looked unnecessarily complicated. In this version, the Inception modules are more uniform, and a few reduction modules are added for grid reduction. This improvement allows Inception-V4 to obtain a top-5 error rate of 3.08% (Szegedy, et al., 2016).

### 2.7.4 ResNet

Residual Network (ResNet) was proposed by He, et al. in 2015 and won ILSVRC 2015 with a top-5 error rate of 3.57%. There are multiple types of ResNet according to the number of layers available in the network, typically from 34 to 1202 layers, and the network that won the challenge consists of 152 layers. The main objective of this network is to address the vanishing gradient issue that is commonly encountered when training deep neural networks (Khan, et al., 2020). To achieve this objective, skip connections that connect a residual block's input to its output are implemented. The residual block, consisting of a feedforward network and a skip connection, is the building block of ResNet and is shown in Figure 2.29. With the implementation of skip connections, the lower-level features can be preserved and the performance can be prevented from deteriorating, as more layers are added (Wu, et al., 2020).

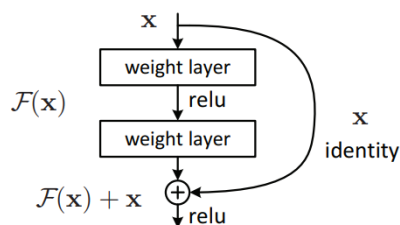


Figure 2.29: The Building Block of ResNet (He, et al., 2015)

### 2.7.5 ResNeXt

Aggregate Residual Transform Network, or popularly known as ResNeXt was proposed by Xie, et al. in 2016, which won ILSVRC 2016 with a top-5 error rate of 3.03%. It utilizes the idea of the split, transform, and merge functions similar to that in the architecture of Inception networks. However, in addition to the existing width and depth dimensions, Xie, et al. introduced a new dimension called cardinality, which corresponds to the size of the set of transformations. This is an improvement from the Inception networks whereby the need to modify the transformation branches from module to module is omitted (Khan, et al., 2020). Moreover, an experiment was conducted by Xie, et al., which shows that the increase in cardinality is able to improve

the accuracy in image classification. Besides Inception, ResNeXt also utilized residual learning from ResNet to have a better convergence and adopted VGGNet's deep homogenous topology with the simplified architecture of GoogLeNet by configuring spatial resolution of the Inception modules to  $3 \times 3$  filters (Alzubaidi, et al., 2021). The building block of ResNeXt is shown in the figure below.

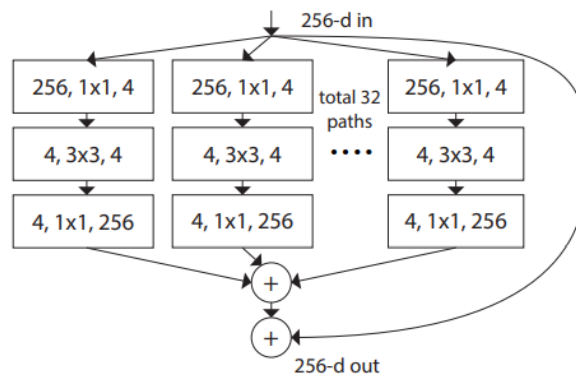


Figure 2.30: Building Block of ResNeXt with Cardinality of 32 (Xie, et al., 2017)

### 2.7.6 SENet

ILSVRC 2017 was won by Hu, et al., who proposed Squeeze-and-Excitation Network (SENet) in 2017, and obtained a top-5 error rate of 2.25%. In contrast to previous models that focus on spatial representations, Hu, et al. took a different route to investigate the relationship between channels. They introduced a new block called the SE-block in which the interdependencies across channels are modelled unequivocally to recalibrate channel-wise feature responses dynamically. This approach highlights the important class specifying features while suppressing those that are least informative. The block is designed in such a way that it can be easily stacked to create a network (Hu, et al., 2019). An illustration of the SE-block is shown in Figure 2.31.

As the name suggests, the block uses the squeeze and excitation operations to perform its functions. The squeeze operation uses global average pooling to aggregate global spatial information into channel-wise statistics. The excitation operation then uses the information by utilizing a sigmoid activation as the gating mechanism to

produce a collection of channel weights that describe the channel-wise dependencies (Hu, et al., 2019; Khan, et al., 2020).

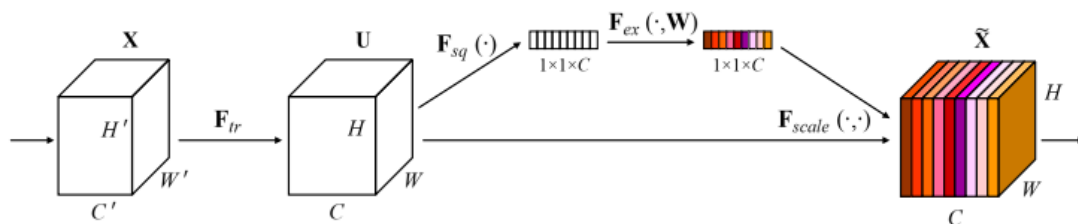


Figure 2.31: SE-block (Hu, et al., 2017)

### 2.7.7 DenseNet

Dense Convolutional Network (DenseNet) was proposed by Huang, et al. in 2017 and gotten the Best Paper Award in the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), with more than 2000 citations. When implemented on the ImageNet dataset, DenseNet was able to obtain a top-5 error rate of 5.54%. The direction of DenseNet is typically the same as ResNet, which is to overcome the problem of vanishing gradient. However, ResNet has its shortcoming when in comparison with traditional feedforward networks. In traditional feedforward networks, each layer changes the preceding layer's state and preserves the information that needs to be passed to the following layer. Whereas ResNet preserves the information explicitly by summing the identity mapping, but as a matter of fact, most layers can only provide a small amount of information or even none. DenseNet overcomes this problem by concatenating the identity mapping instead, which explicitly enables differentiation between preserved information and newly added information.

The architecture of DenseNet directly connects all the layers in the network together with one another, whereby feature maps of each layer will also be inputted to every subsequent layer in addition to the feedforward network. This architecture is illustrated in Figure 2.32. DenseNet has very thin layers, which is parametrically more efficient. It is also easier to train due to the increase in information flow and gradient

across the network. In addition, overfitting issues can also be overcome as dense connections exhibits regularization (Huang, et al., 2017).

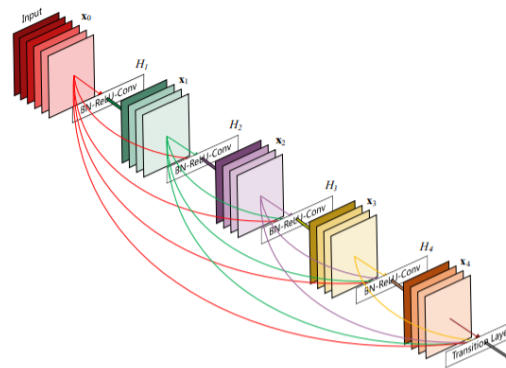


Figure 2.32: DenseNet Architecture (Huang, et al., 2017)

## 2.8 Performance Metrics

Prediction is one of the main focus in the topic of machine learning and deep learning. The task to predict an outcome represented in numeric measurements is known as a “regression problem”, while the task to predict an outcome that reflects distinct classes is known as a “classification problem”. The typical setting in “classification problems” mostly consists of two classes only, but there are also cases where there are three or more number of classes. Hence, the latter is dubbed as a “multi-class classification problem”. In the typical classification problem, the data,  $X$  will be used to make predictions of the outcome variable,  $Y$ , where the predicted outcome is denoted as  $\hat{Y}$ . As for multi-class classification problem,  $Y$  and  $\hat{Y}$  are seen as two discrete random variables, in which they are written as  $\{1, \dots, N\}$ , where  $N$  is the number of classes. Each element of the variables are numbers that represents a particular class. A classification model will generate  $\hat{Y}$ , where the elements are the probability that a given data is of a certain class. The rule of thumb to perform the classification is that the class with the highest probability will be assigned to the given data (Grandini, Bagli and Visani, 2020).

Metrics are a good performance indicator when evaluating and comparing multiple models of different algorithms and techniques. Some of the widely used performance metrics include accuracy, precision, recall, F1-score, ROC curve, and confusion matrix. Before diving into them, it is important to first understand a few key words, namely: true positive ( $tp$ ), true negative ( $tn$ ), false positive ( $fp$ ), and false negative ( $fn$ ). True positives are positive predictions made by the model, and the actual label is also positive, while true negatives are negative predictions made by the model, and the actual label is also negative. On the other hand, false positives are positive predictions made by the model, but the actual label is negative, whereas false negatives are negative predictions made by the model, but the actual label is positive. Henceforth,  $tp$  and  $tn$  are correct predictions, while  $fp$  and  $fn$  are incorrect predictions.

### 2.8.1 Accuracy

Accuracy is the most popular performance metric used for multi-class classification problems. As shown in the Equation 2.7, accuracy measures the ratio of the number of correct predictions made by the model to the number of data given. However, it is not the best performance metric to use on a problem with an imbalanced dataset because the class with the larger dataset will have more significance than the class with the smaller dataset (Grandini, Bagli and Visani, 2020). Consider a dataset with class A occupying 96% of the total while class B occupying 4% of the total, the model can simply predict that all samples belong to class A and easily achieve a training accuracy of 96%. Figure 2.33 gives a better illustration on how accuracy is determined.

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \quad (2.7)$$

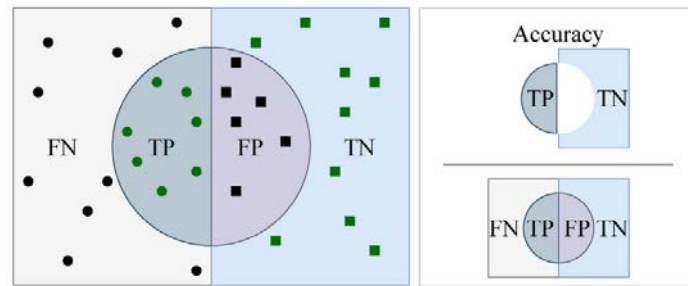


Figure 2.33: Visualizing Accuracy (Maleki, et al., 2020)

## 2.8.2 Precision

Precision tells about how much a model can be trusted when it labels a data as positive. As shown in Equation 2.8, precision measures the amount of true positives to all of the data where data that the model labels as positive. A model with high precision shows that the prediction of the model has a higher quality in labelling data with positive labels, whereas a model with low precision shows that the prediction model has a lower quality when labelling data with positive labels (Grandini, Bagli and Visani, 2020). Figure 2.34 gives a better illustration on how precision is determined.

$$Precision = \frac{tp}{tp + fp} \quad (2.8)$$

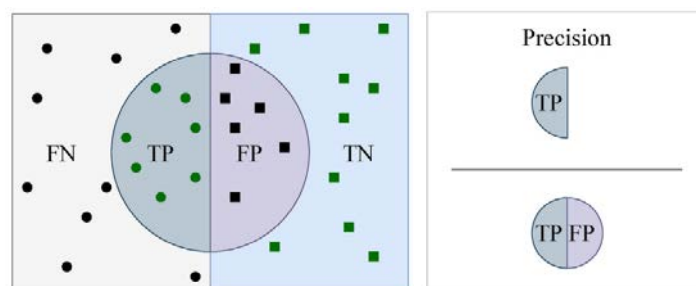


Figure 2.34: Visualizing Precision (Maleki, et al., 2020)



### 2.8.3 Recall

Recall, also referred to as sensitivity, tells about how well the model predicts data of positive labels. As shown in Equation 2.9, recall measures the amount of true positives to all of the data where its actual labels are positive. A model with a high recall shows that the model can find out a high amount of data with actual positive labels, whereas a model with a low recall shows that the model has a hard time trying to identify the data with actual positive labels (Grandini, Bagli and Visani, 2020). Figure 2.35 gives a better illustration on how recall is determined.

$$Recall = \frac{tp}{tp + fn} \quad (2.9)$$

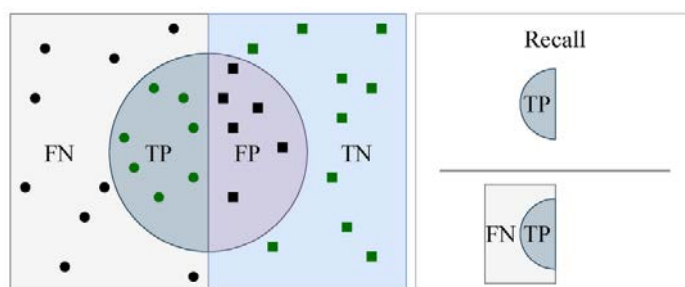


Figure 2.35: Visualizing Recall (Maleki, et al., 2020)

### 2.8.4 F1-score

Although with the precision and recall, it can still be quite confusing to determine whether the model is performing well or not. Henceforth, F1-score, which combines both of the metrics by determining their harmonic mean, is helpful in finding the best trade-off between the two metrics. Consider a Model X with a precision of 50% and a recall of 90%, the F1-score calculated as 0.3214 using the equation shown below.

$$F1 - score = 2 \left( \frac{precision \times recall}{precision + recall} \right) \quad (2.10)$$

Next, consider Model Y with the same precision and recall of 70%, the F1-score is calculated as 0.7. The mean of the precision and recall for both models are the same,

but the harmonic mean computed says otherwise. With that said, The F1-score rewards the models if the precision and recall does not differ too much (Grandini, Bagli and Visani, 2020). Hence, the higher the F1-score, the better the performance of the model.

### 2.8.5 Confusion Matrix

The confusion matrix is a  $N \times N$  cross table, where  $N$  corresponds to the number of classes. All of the predictions done by the model are plotted onto the confusion matrix, which allows better visualization of its performance. The columns of the confusion matrix are the predicted labels, while the rows of the confusion matrix are the true labels. The diagonal elements of the confusion matrix are known as the true positives or true negatives. With that said, the confusion matrix can immediately present how many correct predictions the model makes just by looking at the diagonal elements. The other elements of the confusion matrix are known as false positives or false negatives. The performance metrics mentioned earlier are mostly based on the confusion matrix since it encompasses all of the information regarding the classification rule performance and algorithm (Grandini, Bagli and Visani, 2020). Figure 2.36 shows an illustration of the confusion matrix. Note that the values in the elements are usually complemented with background colours for a better visualization, whereby a higher value will be given a darker tone, while a lower value will be given a lighter tone.

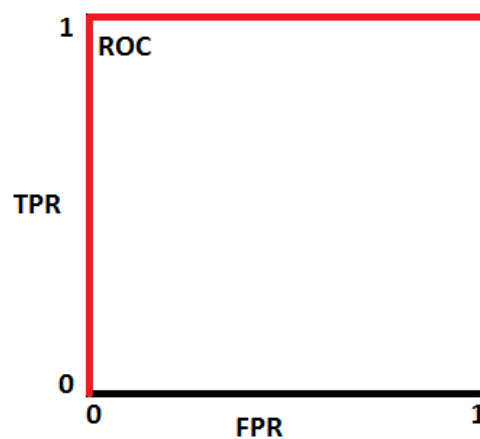
True Labels	C	2	1	0
	B	2	4	0
	A	0	0	1
		A	B	C
		Predicted labels		

Figure 2.36: Illustration of Confusion Matrix

### 2.8.6 ROC Curve

The Receiver Operating Characteristics (ROC) curve is another useful metric for determining a model's performance, primarily in classification problems. It is used to measure the class separability of a model. The ROC curve is plotted using the true positive rate (TPR), also known as the recall, as the y-axis, while the false positive rate (FPR) is the x-axis. The false positive rate is calculated using Equation 2.11. Multiple curves can be plotted on the same graph for a multi-class classification problem to visualize how the model is performing in each class. The area under the ROC curve (AUC-ROC) then measures the degree of the model's class separability for each class. The values of AUC-ROC can range between 0 to 1, and the closer the value is to 1, the better the model is at distinguishing the class against the rest (Narkhede, 2018). An illustration of the ROC curves with different AUC-ROC values is shown in Figure 2.37.

$$\text{False Positive Rate} = \frac{fn}{tn + fp} \quad (2.11)$$



(a)

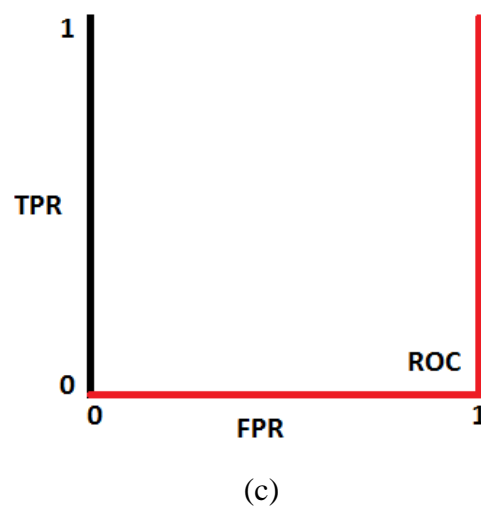
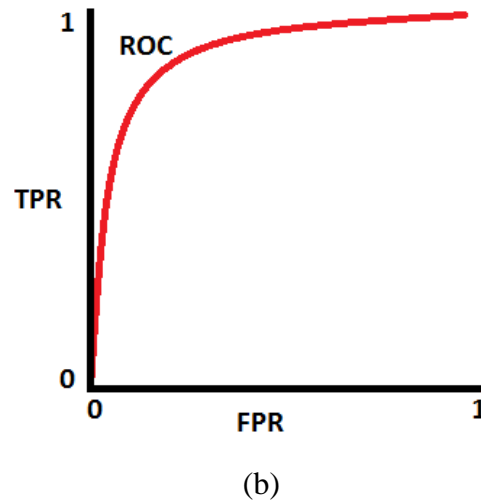


Figure 2.37: Illustration of ROC Curve with (a) AUC-ROC of 1  
(b) AUC-ROC of 0.8 (c) AUC-ROC of 0 (Narkhede, 2018)

Notice that a well-performing model with an ideal class separability will exhibit a sharp bent on the ROC curve with an AUC-ROC of 1. It tells that the model has no problem distinguishing between the positive and negative classes. As the model performs poorer, the sharp bent will slowly curve downwards, and for an AUC-ROC value of 0.8, the model is said to have an 80% chance of distinguishing between positive and negative classes. If a sharp bent is formed at the bottom of the graph with an AUC-ROC of 0, the model is performing very poorly, and the positive class is predicted as the negative class and vice versa.

## 2.9 Related Works

There is extensive research to provide solutions in detecting and classifying leukemia through deep learning approaches, but they are mostly done on binary classification between ALL and healthy cells and lesser on 5-class classification between ALL, AML, CLL, CML, and healthy cells. In the works by Ahmed, et al. (2019), a CNN model with 2 convolution layers is proposed to tackle both the binary and 5-class classification problem. The model successfully detects and classifies leukemia with 88.25% and 81.74% accuracy for binary and 5-class classification problems. On the other hand, Thanh, et al. (2018) proposed a deeper CNN model which consists of 5 convolution layers to tackle the binary classification problem. The proposed model can also successfully detect leukemia with an accuracy of 96.60%.

Instead of training CNN models from scratch, Shafique, et al. (2018) proposed implementing the transfer learning method on a pre-trained model called the AlexNet model for an efficient and faster model training process. The last 3 layers of the AlexNet model were removed, and the fully connected network was tuned to facilitate the binary classification problem. Multiple input feature vectors are also implemented in the fully connected network, and it is noticed that the input feature vector with the smallest dimension achieved the highest accuracy of 99.50%. Bibi, et al. (2020) also tackled the 5-class classification problem using the transfer learning method. Two pre-trained models, ResNet-34 and DenseNet-121, are utilized to detect and classify leukemia, and the highest accuracy achieved was 99.91% by the DenseNet-121 model.

Vogado, et al. (2019) also tackled the binary classification problem by implementing the transfer learning technique on the pre-trained models, including AlexNet, CaffeNet, and Vgg-f models. However, unlike Shafique et al. (2018), who used the softmax function as the classifier, Vogado, et al. (2019) proposed using a machine learning algorithm, the Support Vector Machine (SVM), as the classifier instead. Besides, they also proposed using attribute selection techniques called the gain ratio algorithm for feature selection. To take advantage of all three models in their feature extraction capability, a hybrid approach, which consists of all three models concatenated together, was proposed, and an accuracy of 99.20% was achieved. Another literature that focuses on hybrid approaches to detect leukemia is Das and

Meher (2021), in which the MobileNetV2 and ResNet18 models are hybridized into one model, and the transfer learning technique is also utilized in the training process. As a result, the proposed hybrid model is able to achieve an accuracy of 99.39% in the binary classification problem. All in all, the deep learning approaches utilized in the detection and classification of leukemia and their accuracies are accumulated and illustrated in the bar charts plotted below.

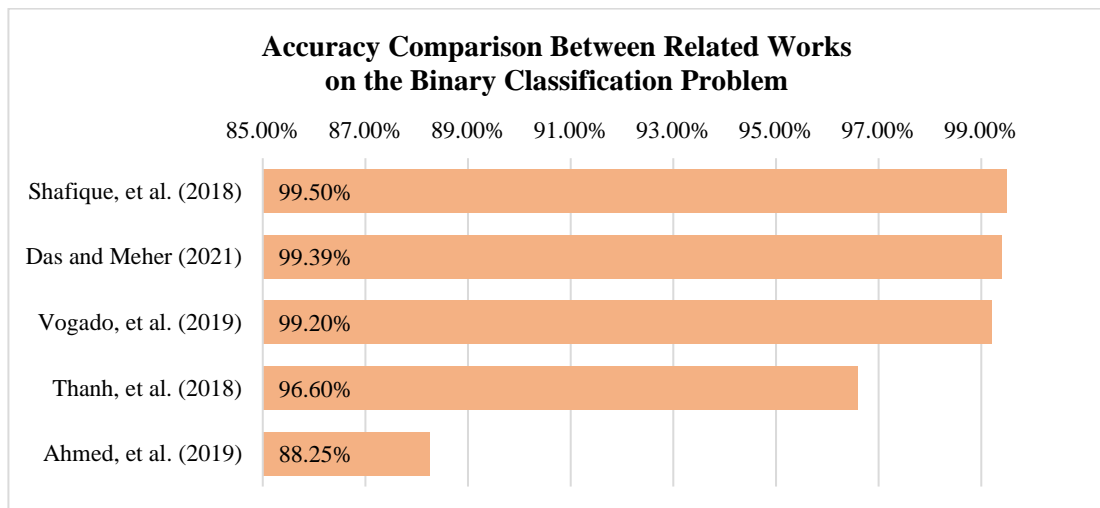


Figure 2.38: Bar Chart of The Accuracy Comparison Between Related Works on the Binary Classification Problem

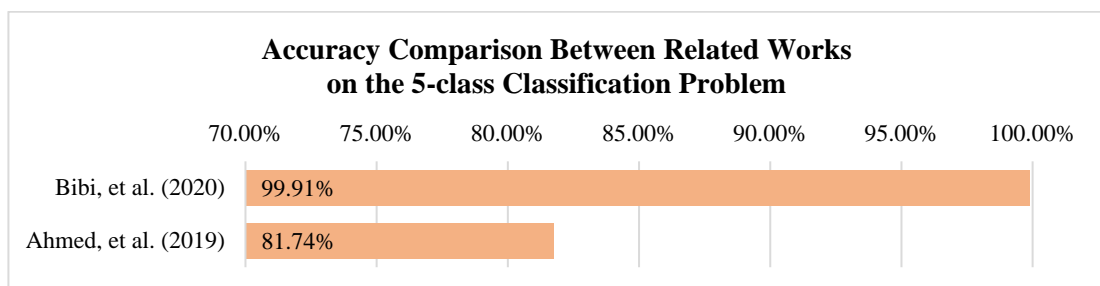


Figure 2.39: Bar Chart of The Accuracy Comparison Between Related Works on the 5-class Classification Problem

In the studies discussed earlier, most of the models are trained using microscopic samples primarily from the Acute Lymphoblastic Leukemia Image Database (ALL-IDB) (Scotti, Labati, and Piuri, 2011). The samples it provides come with the locations for the ALL blast cells annotated by qualified oncologists; therefore, it is much more reliable to use. Another widely used source to gather leukemia images is the American Society of Hematology (ASH) ImageBank (American Society of

Hematology, n.d.), an online library containing all kinds of images involving haematology. On top of that, most literature is also faced with the scarcity of samples available for training and testing. Henceforth, several data augmentation techniques commonly implemented to increase the dataset size include histogram equalization, translation, reflection, rotation, shearing, shifting, zooming, and blurring.

## CHAPTER 3

### METHODOLOGY

#### 3.1 Project Flow

In this project, five classes of data will undergo detection and classification using deep learning: healthy cell, ALL, AML, CLL, and CML. Based on the literature review, it is observed the detection and classification of leukemia are highly accurate when pre-trained models are used instead of training from scratch. Therefore, transfer learning is also utilized in this project for the detection and classification of leukemia. Three state-of-the-art pre-trained models, namely Inception-V3, SENet, ResNeXt, are selected because they have one of the lowest top-5 error rate compared to other models and have not been used before on this kind of project.

The project flow is illustrated in the flowchart shown in Figure 3.1. The first step is to acquire the dataset of leukemia subtypes and healthy cells from online sources. Then, the samples will be pre-processed through data augmentation techniques to increase the dataset count. In the meantime, the pre-trained models will be downloaded from the web to be ready for transfer learning. When the dataset is sufficiently enough, the training of the models will start, and they will be evaluated based on a few performance metrics. After achieving the desired accuracy, the best model among the pre-trained models will be selected to be utilized on any suitable applications.



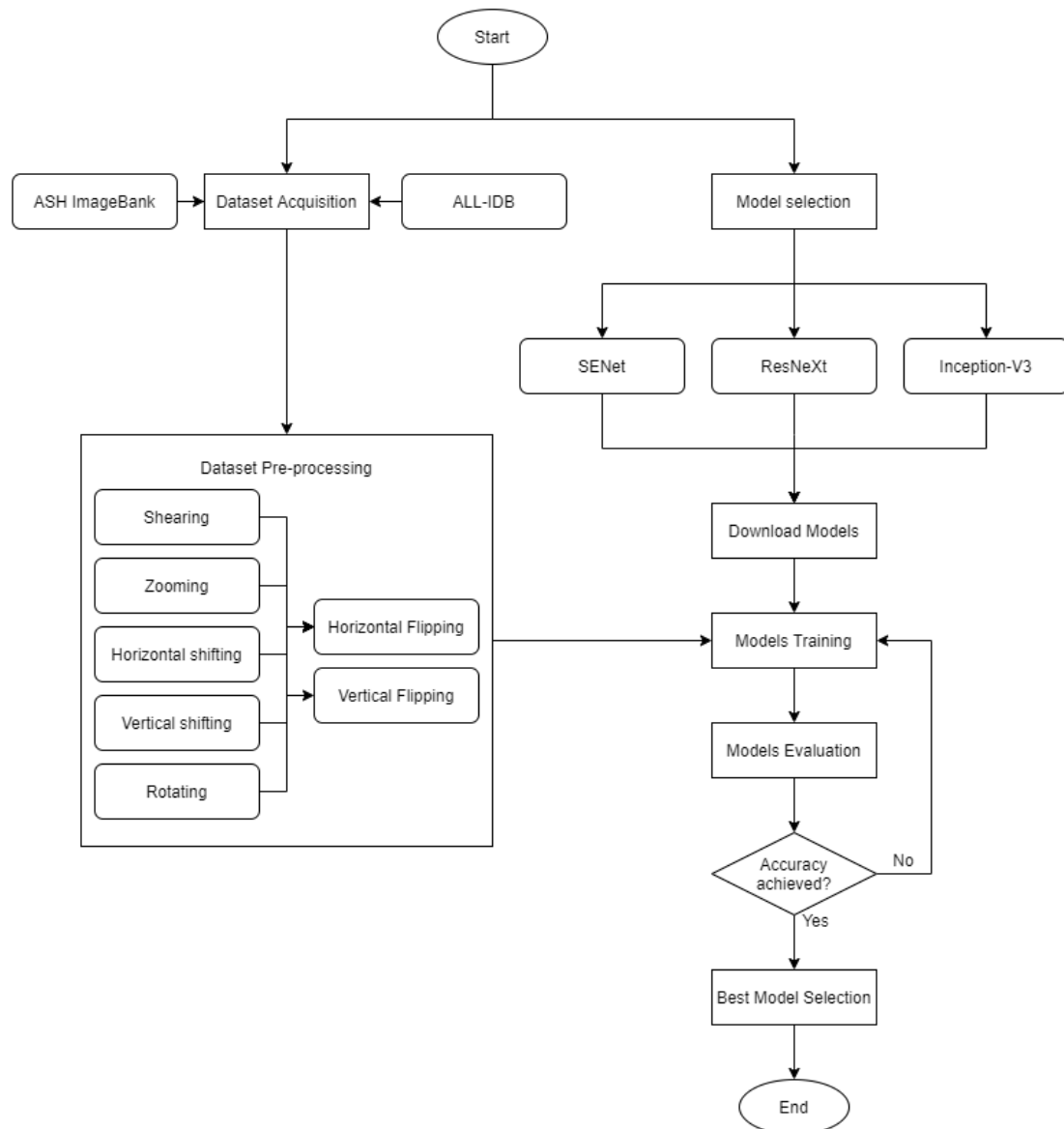


Figure 3.1: Flowchart of The Project

### 3.2 Project Requirements

The required hardware and software, as well as the programming language and open-source libraries used to perform the detection and classification of leukemia are discussed in the following sub-sections.

### **3.2.1 Hardware Requirements**

Besides a computer, there is no additional hardware required. The computer used in this project is an Asus ROG GL552VX packed with NVIDIA GeForce GTX 950M, Intel Core i7-6700HQ, and two 4GB RAM sticks.

### **3.2.2 Software Requirements**

The software used is an Internet browser that allows the use of two websites called Google Colab and Google Drive for the purpose of training DL models. There is a selection of Internet browsers on the market, such as Google Chrome, Mozilla Firefox, and Opera, where all of them are not limited to browse Google Colab and Google Drive.

Google Colab is an online environment that allows executing code on the browser, or more specifically on the cloud. It is targeted to AI researchers and data scientists allowing them to perform machine learning or deep learning by leveraging Google's cloud servers and GPUs. They offer a wide range of NVIDIA Tensor Core GPUs, including Tesla® K80, Tesla® P100, Tesla® P4, Tesla® T4, Tesla® V100, and A100, that can accelerate the speed for DL training, offering an efficient way compared to training on the local computer. On the other hand, Google Drive is an online service that provides cloud storage for all kinds of files and folders. The use of Google Drive allows the dataset to be available on the line and ready for use on any device by just mounting it onto Google Colab.

### **3.2.3 Programming Language Used**

The primary programming language that will be used for this project is Python 3.7.11 version. It is the most well-known high-level programming language, and according to Srivastava (2020), it is dubbed the programming language most preferred for

Artificial Intelligence (AI) projects in 2020. Though Python is often limited by its development speed, it still does not stop people from using it. Python is very similar to the English language, which allows better readability and understanding of the code. Besides, there is also an active and large community behind Python, which allows a problem to be easily solved. It also offers all kinds of open-source libraries made by other developers for all sorts of applications (Basel, 2018).

### 3.2.4 Open-Source Libraries

A few Python open-source libraries will be required, and they are tabulated in the Table 3.1. The table contains the version used, as well as the commands to install and import the libraries.

**Table 3.1: Commands to Install Open-Source Libraries**

<b>Python libraries</b>	<b>Version used</b>	<b>Install command</b>	<b>Import command</b>
Google-colab	0.0.1a2	<i>pip install google-colab</i>	import google.colab
Itertools	-	-	import itertools
Keras	2.8.0	<i>pip install keras</i>	import keras
Matplotlib	3.2.2	<i>pip install matplotlib</i>	import matplotlib
NumPy	1.21.5	<i>pip install numpy</i>	import numpy
OpenCV	4.1.2	<i>pip install opencv-python</i>	import cv2
OS	-	-	import os
Pandas	1.3.5	<i>pip install pandas</i>	import pandas
Pretrainedmodels	0.7.4	<i>pip install pretrainedmodels</i>	import pretrainedmodels
PyTorch	1.10.0+cu111	<i>pip install pytorch</i>	import torch
Random	-	-	import random
Scikit-learn	1.0.2	<i>pip install scikit-learn</i>	import sklearn
Torchvision	0.11.1+cu111	<i>pip install torchvision</i>	import torchvision
TensorFlow	2.8.0	<i>pip install tensorflow</i>	import tensorflow

### 3.3 Dataset Acquisition

The microscopic image samples for each class are retrieved from various sources, including ALL-IDB1 and ASH ImageBank. The datasets for healthy cells and ALL are obtained from ALL-IDB1, while the AML, CLL, and CML datasets are obtained from ASH ImageBank. The total dataset count acquired is 266 samples, whereby there are 59 images of healthy cells, and 49 images of ALL, 58 images of AML, 46 images of CLL, and 54 images of CML. The dataset is then uploaded to Google Drive. The number of samples obtained for each class and its source is tabulated as shown below.

**Table 3.2: Dataset Count for Each Class and Its Source**

<b>Sources</b> <b>Classes</b>	<b>ALL-IDB1</b>	<b>ASH ImageBank</b>	<b>Total</b>
<b>ALL</b>	-	49	49
<b>AML</b>	58	-	58
<b>CLL</b>	46	-	46
<b>CML</b>	54	-	54
<b>Healthy</b>	-	59	59
<b>Total number of samples</b>			<b>266</b>

### 3.4 Dataset Splitting

To start with, Google Drive is first mounted onto Google Colab using the code listed in Code Listing 1 in Appendix A to access the dataset. Then, the modules of the open-source libraries are imported as listed in Code Listing 2 in Appendix A. After that, the images are imported into the Google Colab notebook using the OS module available in the Python standard library. The images are then read using OpenCV, which automatically convert the images into array of numbers. The images are also resized to  $224 \times 224$  pixels and normalized. The collection of images turned into array are stored in a list. The labels of the images are also stored in the list in the form of whole numbers such that 0 represents ALL, 1 represents AML, 2 represents CLL, 3 represents

CML, and 4 represents healthy cells or HLT in short. The code for this segment is listed in Code Listing 3 in Appendix A.

Next, in order to ensure that the models will not be tested with images that are used for training, the training and testing set will be separated among the dataset. Furthermore, to properly estimate the performance and generalization of the models, a data resampling method known as k-fold cross-validation is implemented. This is done by using the ‘KFold’ class from scikit-learn library that splits the dataset into k folds of train and test sets, in which this project, a k of 5 is implemented. In order to preserve a balanced class distribution in the datasets, the stratified k-fold cross-validator, which is the ‘StratifiedKFold’ class will be used instead of the ‘KFold’ class. The code for this segment is listed in Code Listing 4 in Appendix A.

With that said, the dataset for binary and 5-class classification problem in each fold is distributed as shown in the Tables 3.3 and 3.4. It is observed that the dataset for each fold is distributed based on the 80/20 rule in which the training set consists of 80% of the dataset, whereas the testing set consists of 20% of the dataset. However, some imbalanced class distribution is still present, and the number of data is barely enough to train the models. Thus, the methods to tackle these problems is explained in the following sub chapters.

**Table 3.3: Training Set Distribution for Each Fold**

<b>Fold</b> <b>Classes</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	39	39	39	39	40
<b>AML</b>	46	46	47	47	46
<b>CLL</b>	37	37	36	37	37
<b>CML</b>	43	43	44	43	43
<b>Healthy</b>	47	48	47	47	47
<b>Total</b>	212	213	213	213	213

**Table 3.4: Testing Set Distribution for Each Fold**

<b>Classes \ Fold</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	10	10	10	10	9
<b>AML</b>	12	12	11	11	12
<b>CLL</b>	9	9	10	9	9
<b>CML</b>	11	11	10	11	11
<b>Healthy</b>	12	11	12	12	12
<b>Total</b>	54	53	53	53	53

### 3.5 Dataset Augmentation

It is evident that the current dataset available are too limited and unfit for training. Hence, following the footsteps of Ahmed, et al. (2020), 7 data augmentation techniques are implemented to avoid overfitting by increasing the size of the existing dataset. The methods include vertical and horizontal flipping, rotation, height and width shifting, shearing, and zooming.

With the dataset in place, data augmentation is then done using the ‘ImageDataGenerator’ class available in the Keras library and the flipping functions available in the NumPy library. The augmentation parameters to be used are translated as the shear range for shearing, zoom range for zooming, rotation range for rotating, height shift range and width shift range for shifting, as well as ‘flipud’ and ‘fliplr’ for flipping. They are fed in as arguments and parameters for the ‘ImageDataGenerator’ class and the flipping functions. The shear range is set to 20, which means that a 20° shearing in the counter-clockwise direction will be applied onto the images. Next, for a 30% zoom, the zoom range is set to 0.3, which means that a random zoom between 1.3 to 0.7 will be applied onto the images. On the other hand, the rotation is set to 40°, which implies that a random rotation between 0° to 40° or a range of 0° to -320° will be applied onto the images. Furthermore, for a 40% shifting, the height shift range is set to 0.4 so that the images will be shifted up or down for 0.4 pixels of

the total height, whereas the width shift range is set to 0.4 so that the images will be shifted left or right for 0.4 pixels of the total width. Lastly, by calling the ‘flipud’ function will flip the arrays vertically while the ‘fliplr’ function will flip the arrays horizontally.

Figure 3.2 illustrates the resultant images after one of the image samples is augmented. Figure 3.2a is the original image while Figures 3.2b to 3.2h are the augmented images. Figure 3.2b is the resultant image after a 20° shear whereas Figure 3.2c is the resultant image after a 30% zoom. Besides that, Figure 3.2d is the resultant image after a 40° rotation. On the other hand, Figure 3.2e is the resultant image after a 40% height shift while Figure 3.2f is the resultant image after a 40% width shift. Furthermore, Figure 3.2g is the resultant image after flipping vertically while Figure 3.2h is the resultant image after flipping horizontally.

The images are augmented in such a way that the images are sheared, zoomed, rotated, and shifted for 2 times each, which increases the dataset by 11-fold. Then, the original and augmented images, are flipped horizontally and vertically, which again increases the size of the dataset by another 3-fold. Hence, after the image transformations, each image sample are increased by a factor of 33-fold, effectively increasing the overall dataset for each class. The code for this segment is listed in Code Listing 5 in Appendix A and the new number of samples for each class and its source is tabulated as shown in Tables 3.5 and 3.6.

**Table 3.5: Training Set Distribution for Each Fold After Augmentation**

<b>Classes \ Fold</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	1287	1287	1287	1287	1320
<b>AML</b>	1518	1518	1551	1551	1518
<b>CLL</b>	1221	1221	1188	1221	1221
<b>CML</b>	1419	1419	1452	1419	1419
<b>Healthy</b>	1551	1584	1551	1551	1551
<b>Total</b>	6996	7029	7029	7029	7029

**Table 3.6: Testing Set Distribution for Each Fold After Augmentation**

<b>Classes \ Fold</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	330	330	330	330	297
<b>AML</b>	396	396	363	363	396
<b>CLL</b>	297	297	330	297	297
<b>CML</b>	363	363	330	363	363
<b>Healthy</b>	396	363	396	396	396
<b>Total</b>	1782	1749	1749	1749	1749

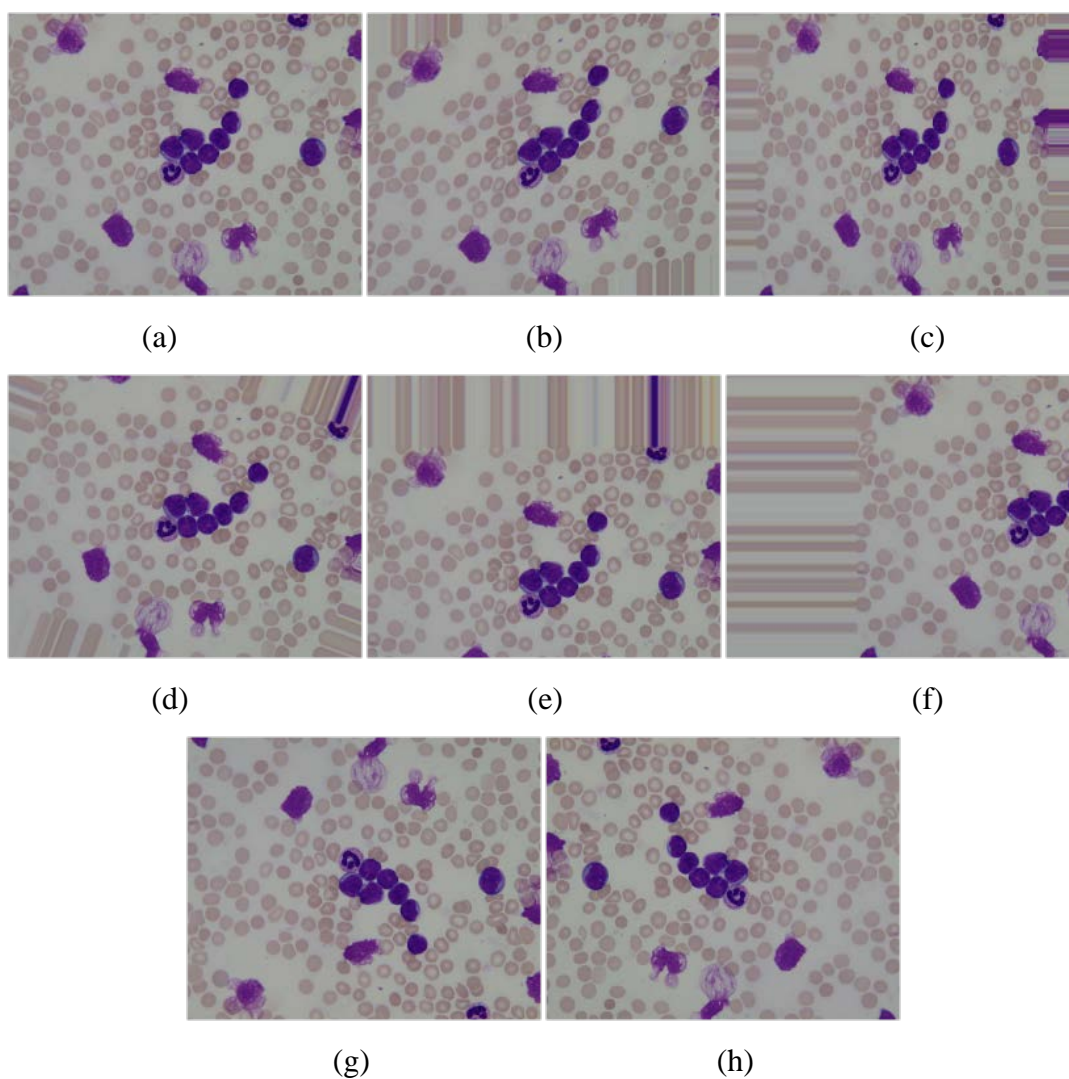


Figure 3.2: (a) Original (b) 20° Shearing (c) 30% Zoom (d) 40° Rotation  
(e) 40% Height Shift (f) 40% Width Shift (g) Vertical Flip (h) Horizontal Flip



With the overfitting issue out of the way, there is still one problem to address: the imbalance class distribution. It is observed that the number of CLL cells is always the least in each fold. Henceforth, the dataset will be equalized to have the same count as the number of CLL cells of each fold. The code for this segment is listed in Code Listing 6 in Appendix A. The bar chart shown in Figure 3.3 illustrates how the dataset for each fold is allocated. The dataset after pre-processing is visualized using Matplotlib as shown in Figure 3.4.

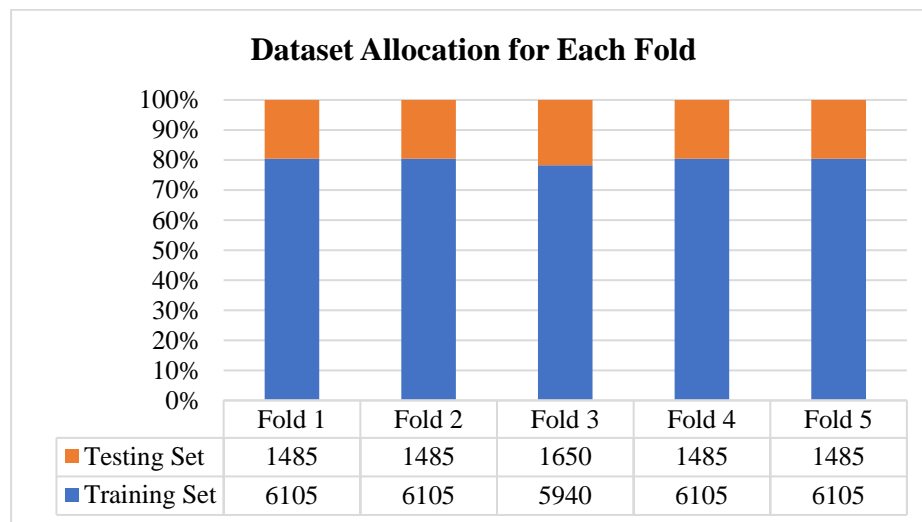


Figure 3.3: Bar Chart of Dataset Allocation for Each Fold

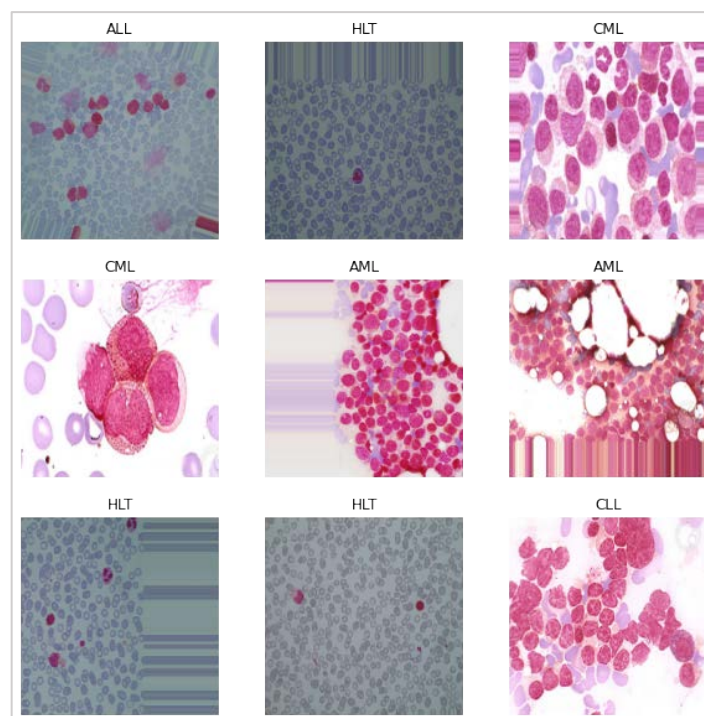


Figure 3.4: Visualizing the Pre-Processed Dataset

### 3.6 Model Training

The training process will be done through Google Colab since the hardware used is low performance and would take a long time to train the models. With that said, the models and their pretrained weights are accessed and downloaded through multiple application programming interfaces (API) available online. The Inception-V3 model is downloaded from the Keras's applications API, ResNeXT model from PyTorch's models API, and SENet model from the pretrainedmodels library that provides a unique API to access pre-trained ConvNets models for PyTorch. Transfer learning is performed by freezing the layers of the models' base network, then their fully connected networks are fine-tuned by replacing them with a hidden layer and an output layer that corresponds to the number of classes for binary and 5-class classification problem. The frozen network is known as the feature extractor, while the fully connected network is known as the classifier network. The image below illustrates the process of transfer learning. The code for this segment is listed in Code Listing 8 to 13 in Appendix A.

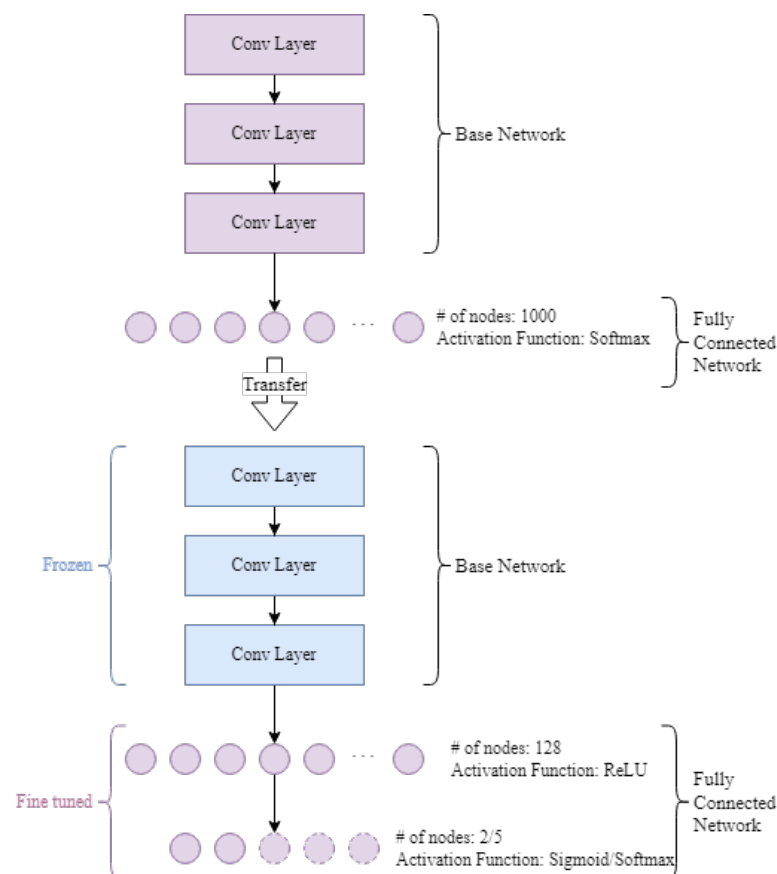


Figure 3.5: Transfer Learning with Pretrained Models as Feature Extractors

After the models are set up, they will be trained with the pre-processed dataset distributed for each fold. Starting with the binary classification problem, the final layer will have 2 nodes, and the sigmoid activation function will be utilized as the classifier. Besides, the binary cross entropy will be used as the loss function, ADAM as the optimizer, and accuracy as the evaluation metric. Then, the models will be fed with ALL and healthy cells images for training. On the other hand, the final layer will have 5 nodes instead for the 5-class classification problem, and the softmax activation function will be utilized as the classifier. Furthermore, categorical cross entropy will be used instead of binary cross entropy, while the loss function and evaluation metrics remain the same. Table 3.7 and 3.8 shows the dataset count of each classes used for binary and 5-class classification problem training respectively. The code for this segment is listed in Code Listing to 14 to 19 in Appendix A.

**Table 3.7: Binary Classification Problem Training Set Distribution for Each Fold**

<b>Classes \ Fold</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	1221	1221	1188	1221	1221
<b>Healthy</b>	1221	1221	1188	1221	1221
<b>Total</b>	2442	2442	2376	2442	2442

**Table 3.8: 5-class Classification Problem Training Set Distribution for Each Fold**

<b>Classes \ Fold</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>ALL</b>	1221	1221	1188	1221	1221
<b>AML</b>	1221	1221	1188	1221	1221
<b>CLL</b>	1221	1221	1188	1221	1221
<b>CML</b>	1221	1221	1188	1221	1221
<b>Healthy</b>	1221	1221	1188	1221	1221
<b>Total</b>	6105	6105	5940	6105	6105

The training dataset is split again during training based on the 80/20 rule, whereby 80% of the training set will be used for training while the remaining 20% will be used for model validation purposes. Therefore, as the models are being trained, the

metrics outputted include training accuracy, validation accuracy, training loss, and validation loss. The number of epochs for the binary classification problem training will be set to 10, whereas, during the 5-class classification problem training, it will be set to 25. In each epoch, the models with the highest validation accuracy will be kept as the final models, while the ones lower than the previous highest values will be neglected.

### **3.7 Model Evaluation**

The final models will be tested for binary and 5-class classification with unseen testing data using the code listed in Code Listing 20 to 24 in Appendix A, then evaluated based on a few performance metrics: accuracy, loss, precision, recall, F1-score, receiver operating characteristic (ROC) curve, area under the ROC curve (AUC-ROC), and the confusion matrix. For this project, accuracy will be used as the main evaluation metrics as it is used as the benchmark for improvement from other related works. The other metrics will be used to justify the findings from the binary and 5-class classification problem, as well as to make comparisons between the different type of models used in this project. The performance metrics will be implemented using the functions available in the 'metrics' class from scikit-learn library. In addition, the pandas library will be used to plot the graph of accuracy and loss against number of epochs, while the matplotlib library is used to visualize the ROC curve and the confusion matrix. The code for this segment is listed in Code Listing 25 to 30 in Appendix A.

### **3.8 Model Improvement**

After models are evaluated with the testing data, the model that got the highest testing accuracy out of the chosen models will be further fine-tuned to improve its overall performance. The fine-tuning methods will be implemented based on several factors such as the ideas adapted from related works, the feature selection process by the model, and the bias and variance of the model towards the data provided. Bias is the

difference between the predicted value by the model and the actual value of the data. A model with a high bias is said to be very simple, and is making basic assumptions about the given data (Singh, 2018). This phenomenon happens when the model is underfitting the data, resulting in the poor performance on the training data, which also leads to a high error on the unseen testing data as well. One of the ways to lower the bias of a model is to train a model with a larger and more complex network, or to train the model long enough so that the important features of the data can be learnt.

On the other hand, variance is the variability of the model in making predictions, such that it measures how much adjustments the model can make based on the data given. A model with a high variance is said to be very complex, and is unable to make generalized predictions on data that it had not seen before. This phenomenon happens when the model is overfitting the data, where the model shows a good performance on the training data, but a high error is observed on the unseen testing data. In order to lower the variance of a model, more data should be used to train the model, or implement regularization in the model (Wickramasinghe, 2021).

### 3.9 Project Costs

The equipment and material costs required to carry out this project is listed as shown in the table below.

**Table 3.9: Equipment and Materials Cost**

No.	Equipment and materials	Price (RM)	Quantity	Cost (RM)	Descriptions
1	Computer	0.00	1	0.00	Any computer in-use can run the project.
2	Google Colab	0.00	-	0.00	Google Colab's cloud service is free for all to be used.
3	Google Drive	8.49/month	4 months	33.96	Google Drive's cloud service is free for all to be used.
4	Opera	0.00	-	0.00	Opera is free for download.
<b>Total cost</b>				<b>33.96</b>	







## CHAPTER 4

## RESULTS AND DISCUSSIONS

## 4.1 Binary Classification Problem for Inception-V3, ResNeXt. And SENet

**Table 4.1: Accuracy and Loss Result for Binary Classification Problem of Each Fold for Inception-V3**

<b>Metrics</b>	<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>	<b>Average</b>
<b>test_acc</b>	99.33%	98.99%	98.79%	92.26%	95.79%	97.03%
<b>val_acc</b>	99.39%	99.39%	100.00%	99.39%	99.80%	99.59%
<b>trn_acc</b>	99.85%	99.90%	99.21%	100.00%	99.13%	99.62%
<b>test_loss</b>	0.0219	0.0261	0.0323	0.3694	0.1565	0.1212
<b>val_loss</b>	0.0225	0.0324	0.0086	0.0211	0.0089	0.0187
<b>trn_loss</b>	0.0084	0.0064	0.0244	0.0025	0.0210	0.0125

**Table 4.2: Accuracy and Loss Result for Binary Classification Problem of Each Fold for ResNeXt**

<b>Metrics</b>	<b>Fold 1</b>	<b>Fold 2</b>	<b>Fold 3</b>	<b>Fold 4</b>	<b>Fold 5</b>	<b>Average</b>
<b>test_acc</b>	99.66%	100.00%	99.09%	93.43%	96.13%	97.66%
<b>val_acc</b>	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
<b>trn_acc</b>	98.77%	99.23%	99.79%	99.44%	99.08%	99.26%
<b>test_loss</b>	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
<b>val_loss</b>	0.0132	0.0015	0.0090	0.0083	0.0093	0.0083
<b>trn_loss</b>	0.0412	0.0245	0.0170	0.0235	0.0352	0.0283

**Table 4.3: Accuracy and Loss Result for Binary Classification Problem of Each Fold for SENet**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	100.00%	99.83%	99.70%	100.00%	99.83%	99.87%
val_acc	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
trn_acc	98.82%	99.39%	100.00%	100.00%	97.08%	99.26%
test_loss	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
val_loss	0.0006	0.0006	0.0011	0.0003	0.0028	0.0011
trn_loss	0.0296	0.0183	0.0002	0.0005	0.0831	0.0263

#### 4.1.1 Fold 1

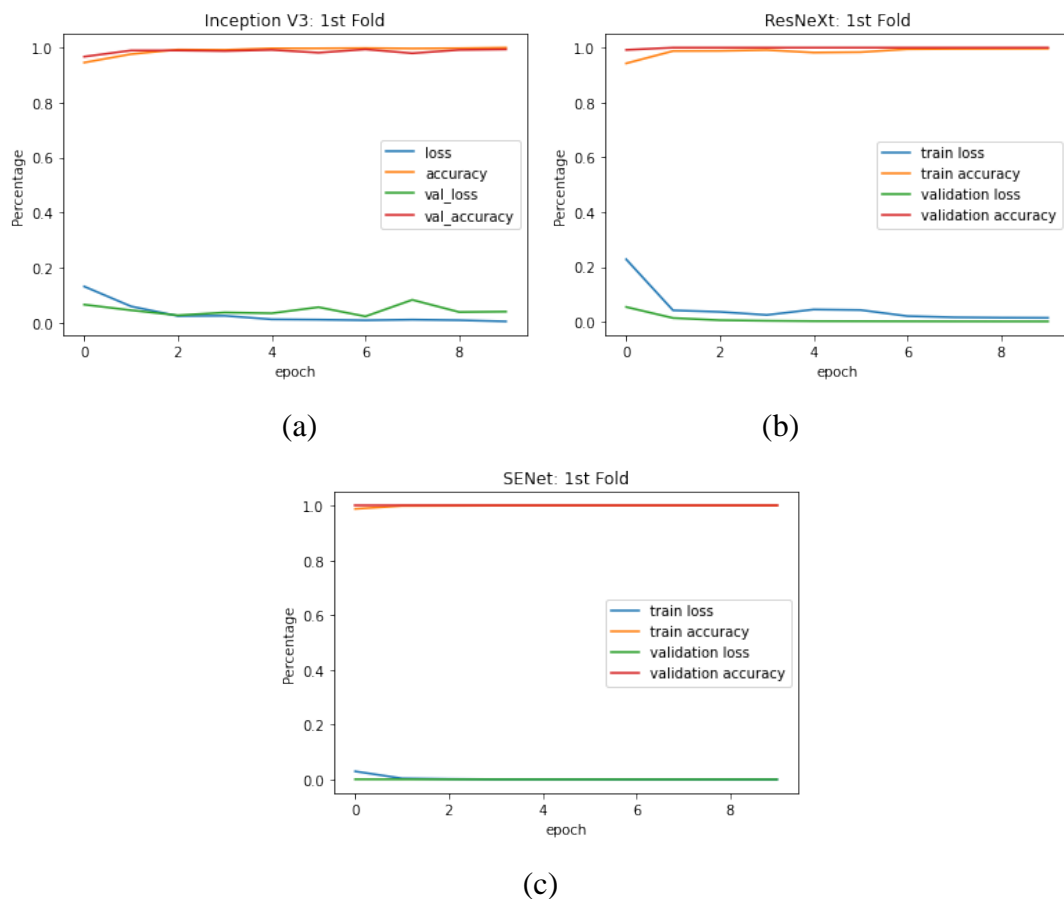


Figure 4.1: Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

**Table 4.4: Precision, Recall, and F1-score for Binary Classification Problem in The First Fold Training for Each Model**

Models	Precision	Recall	F1-score
<b>Inception-V3</b>	99.66%	98.99%	99.32%
<b>ResNeXt</b>	99.66%	99.66%	99.66%
<b>SENet</b>	100.00%	100.00%	100.00%

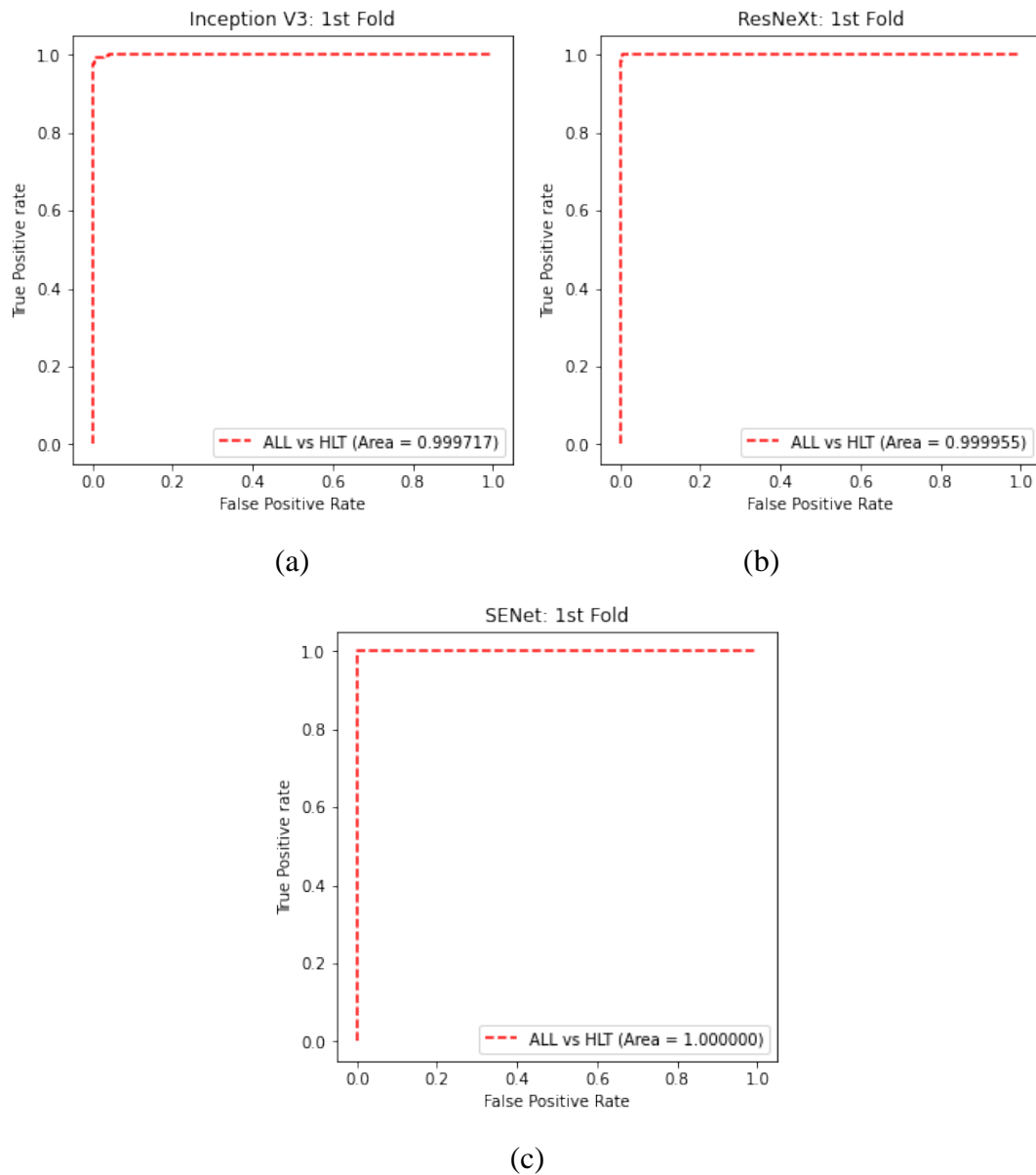


Figure 4.2: ROC Curve in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

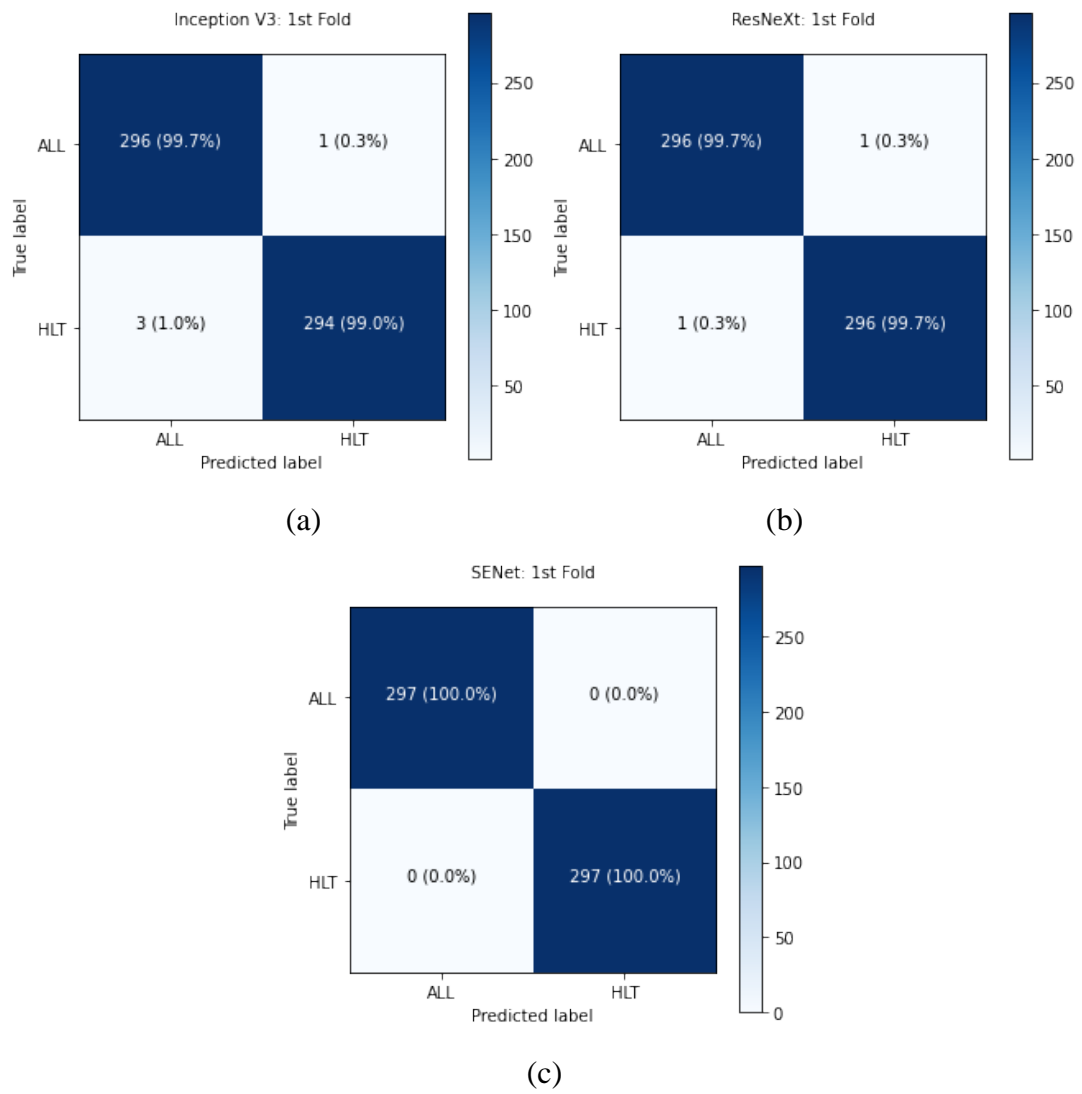


Figure 4.3: Confusion Matrix in The First Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

### 4.1.2 Fold 2

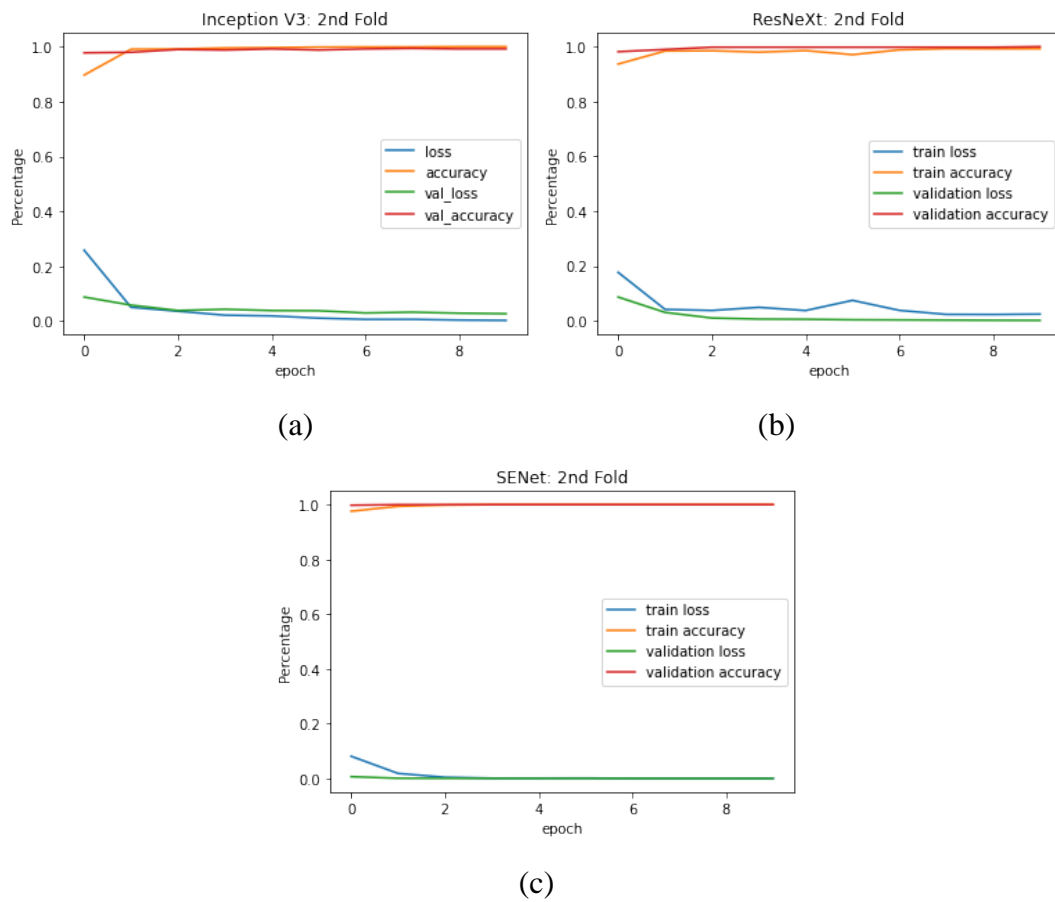


Figure 4.4: Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

**Table 4.5: Precision, Recall, and F1-score for Binary Classification Problem in The Second Fold Training for Each Model**

Models	Precision	Recall	F1-score
<b>Inception-V3</b>	98.02%	100.00%	99.00%
<b>ResNeXt</b>	100.00%	100.00%	100.00%
<b>SENet</b>	99.66%	100.00%	99.83%

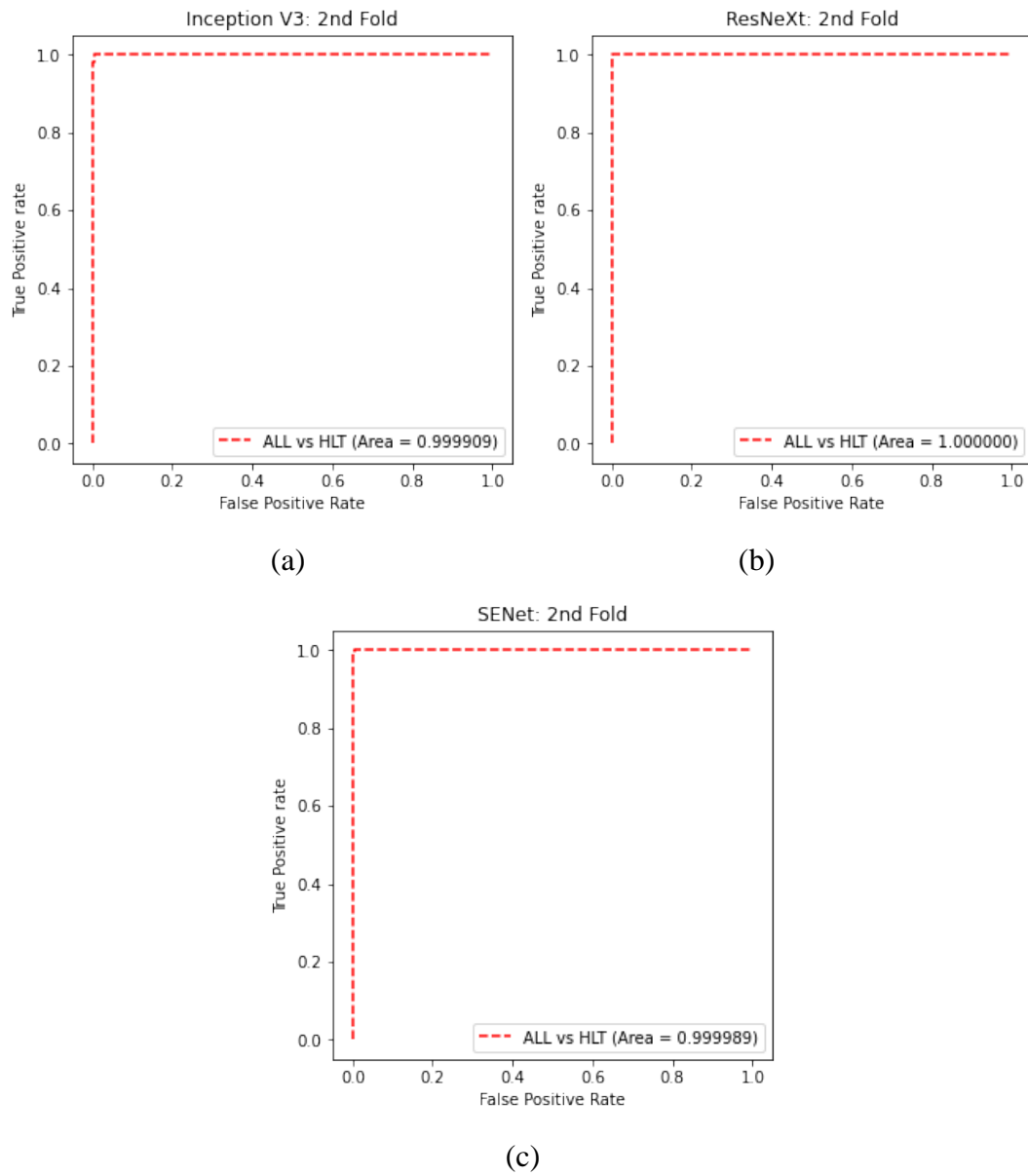


Figure 4.5: ROC Curve in The Second Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

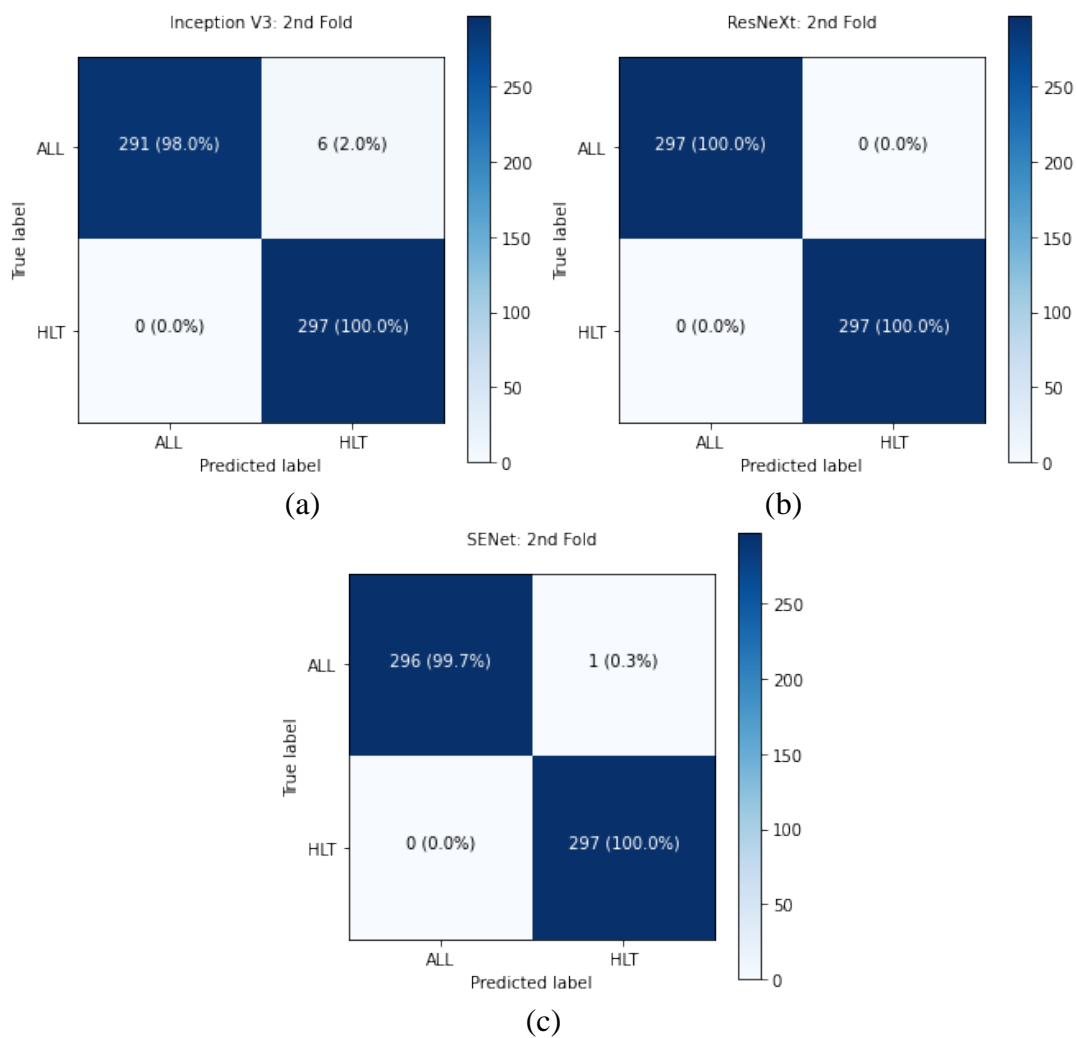


Figure 4.6: Confusion Matrix in The Second Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

### 4.1.3 Fold 3

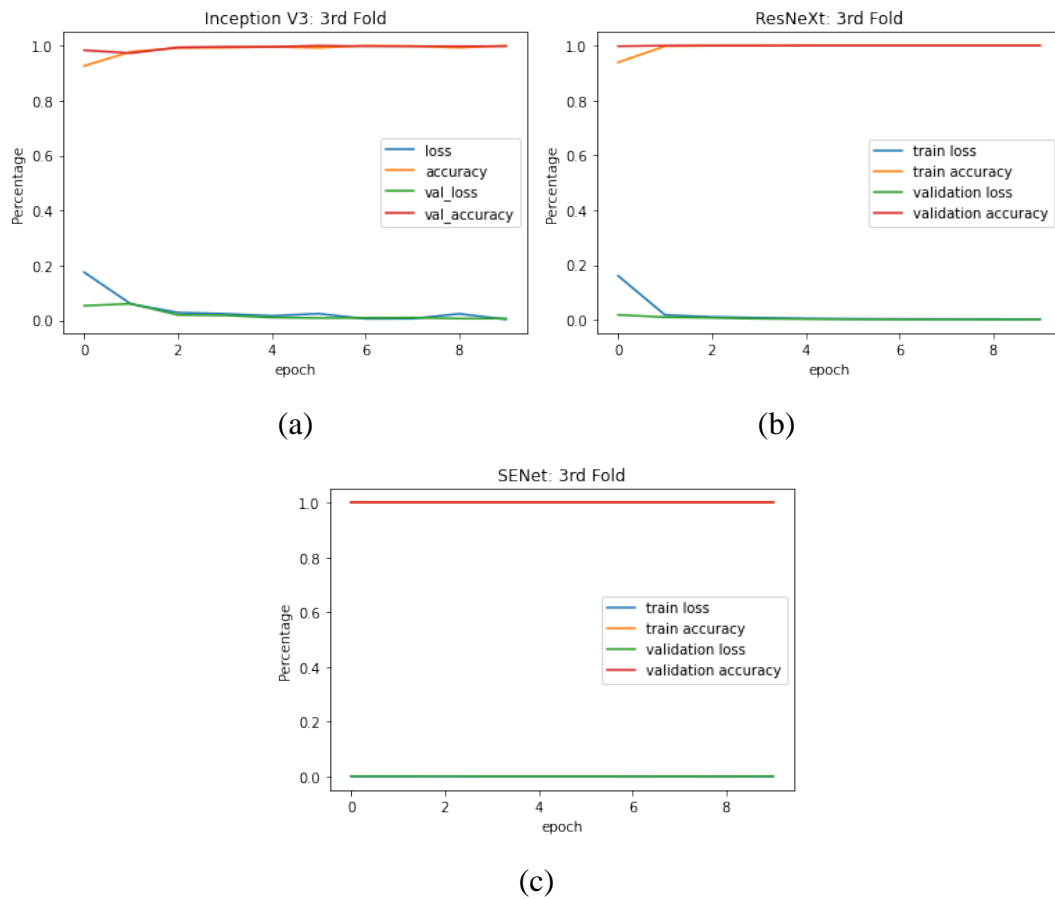


Figure 4.7: Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

**Table 4.6: Precision, Recall, and F1-score for Binary Classification Problem in The Third Fold Training for Each Model**

Models	Precision	Recall	F1-score
<b>Inception-V3</b>	98.79%	98.79%	98.79%
<b>ResNeXt</b>	100.00%	98.18%	99.08%
<b>SENet</b>	99.40%	100.00%	99.70%



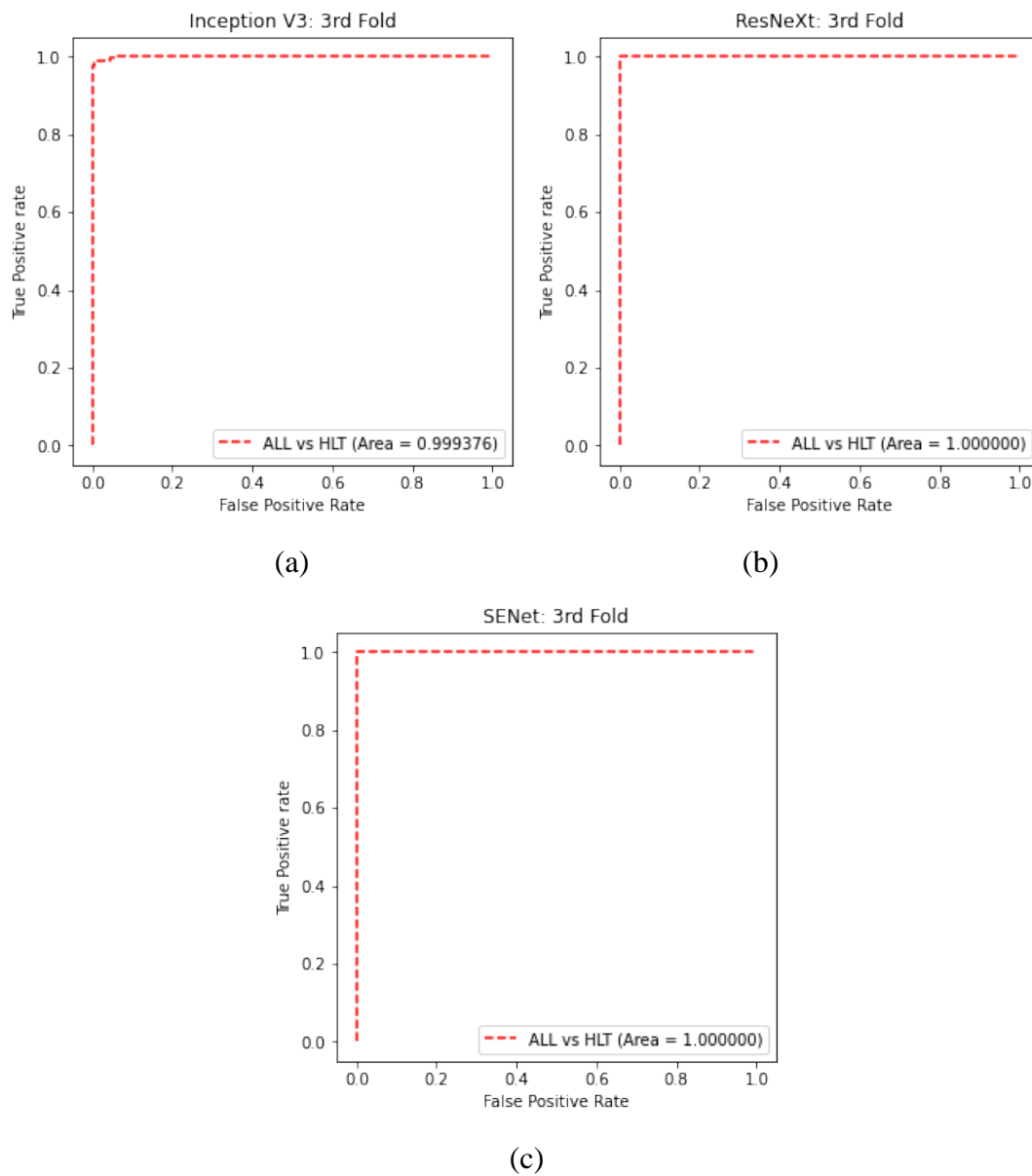


Figure 4.8: ROC Curve in The Third Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

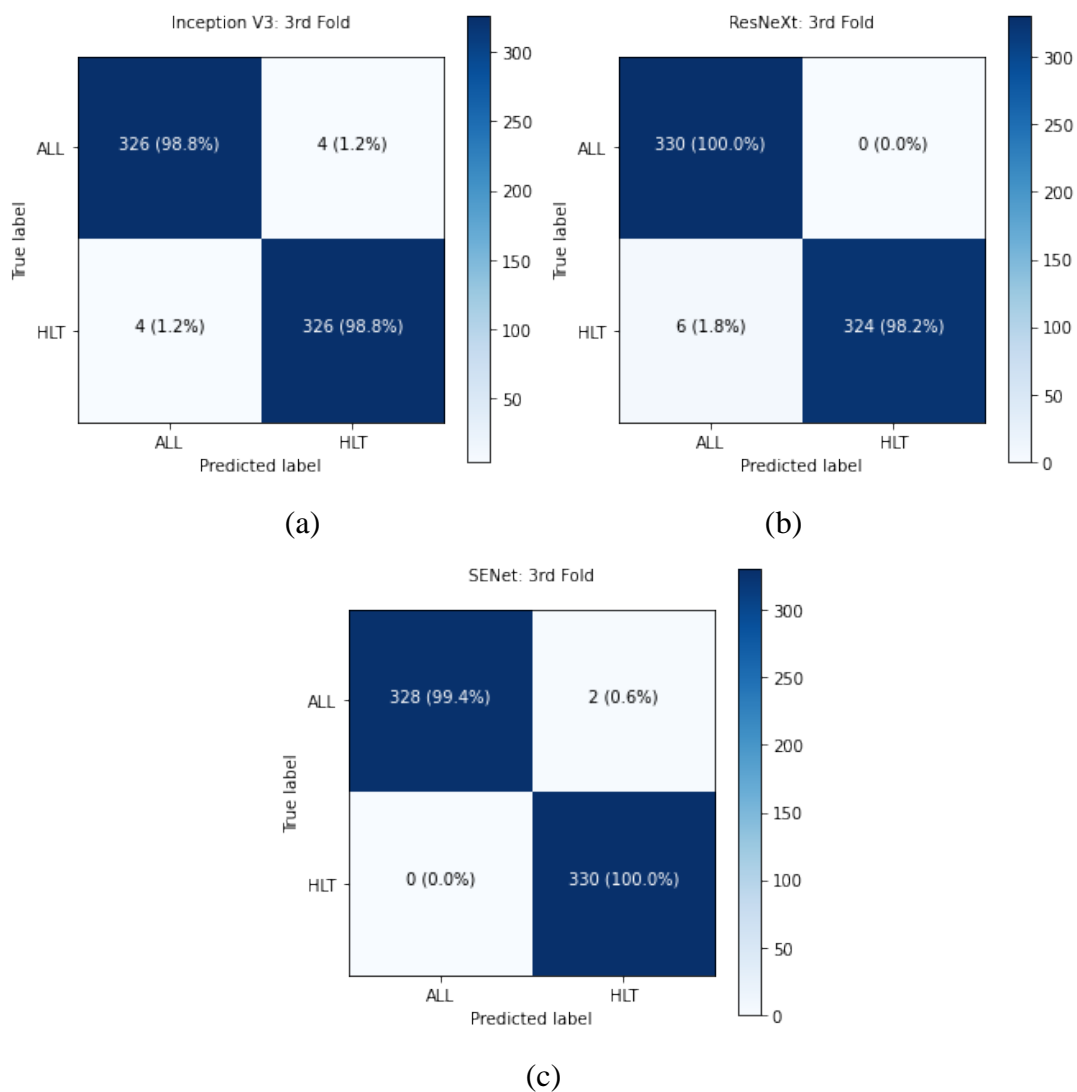


Figure 4.9: Confusion Matrix in The Third Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

#### 4.1.4 Fold 4

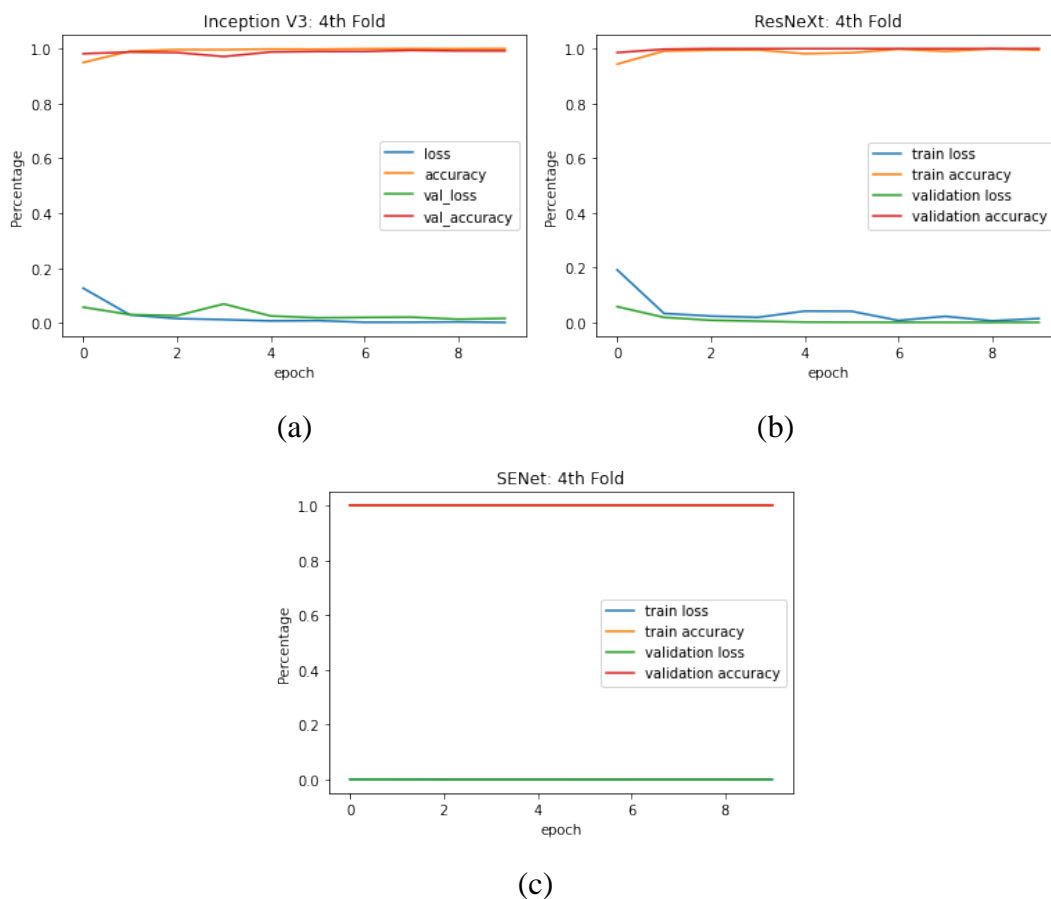


Figure 4.10: Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

**Table 4.7: Precision, Recall, and F1-score for Binary Classification Problem in The Fourth Fold Training for Each Model**

<b>Models</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>Inception</b>	97.36%	86.87%	91.81%
<b>ResNeXt</b>	100.00%	86.87%	92.97%
<b>SENet</b>	100.00%	100.00%	100.00%

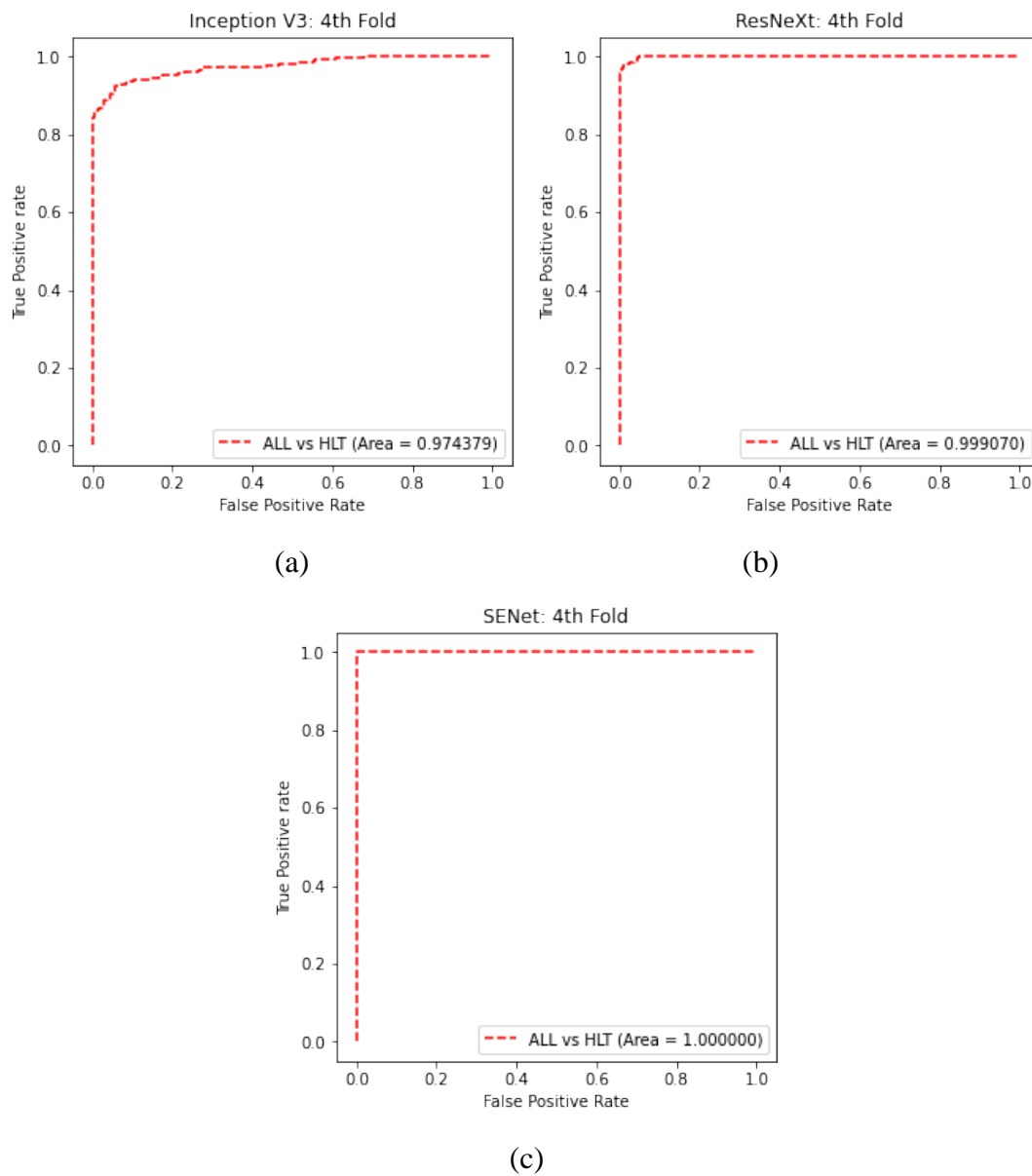


Figure 4.11: ROC Curve in The Fourth Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

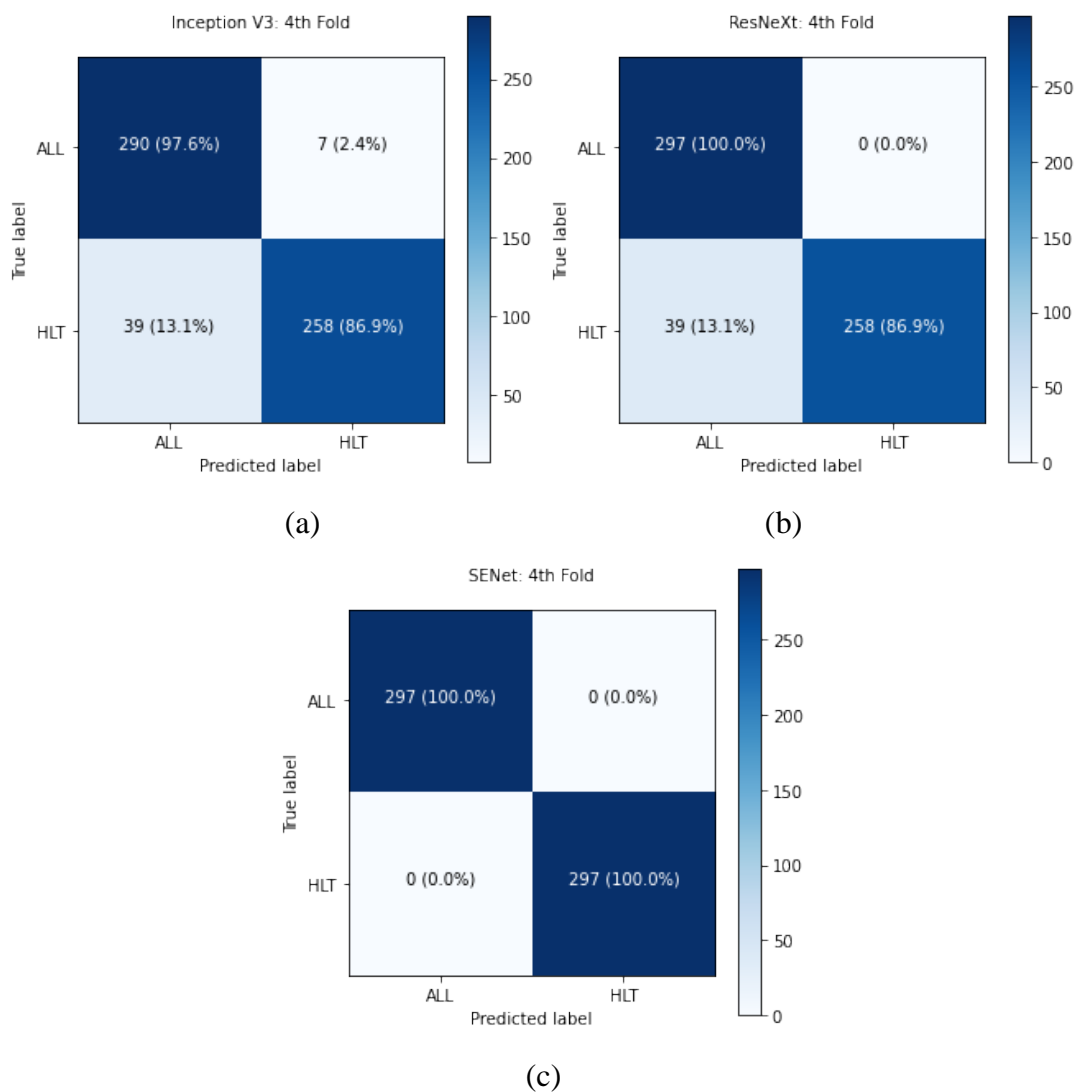


Figure 4.12: Confusion Matrix in The Fourth Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

### 4.1.5 Fold 5

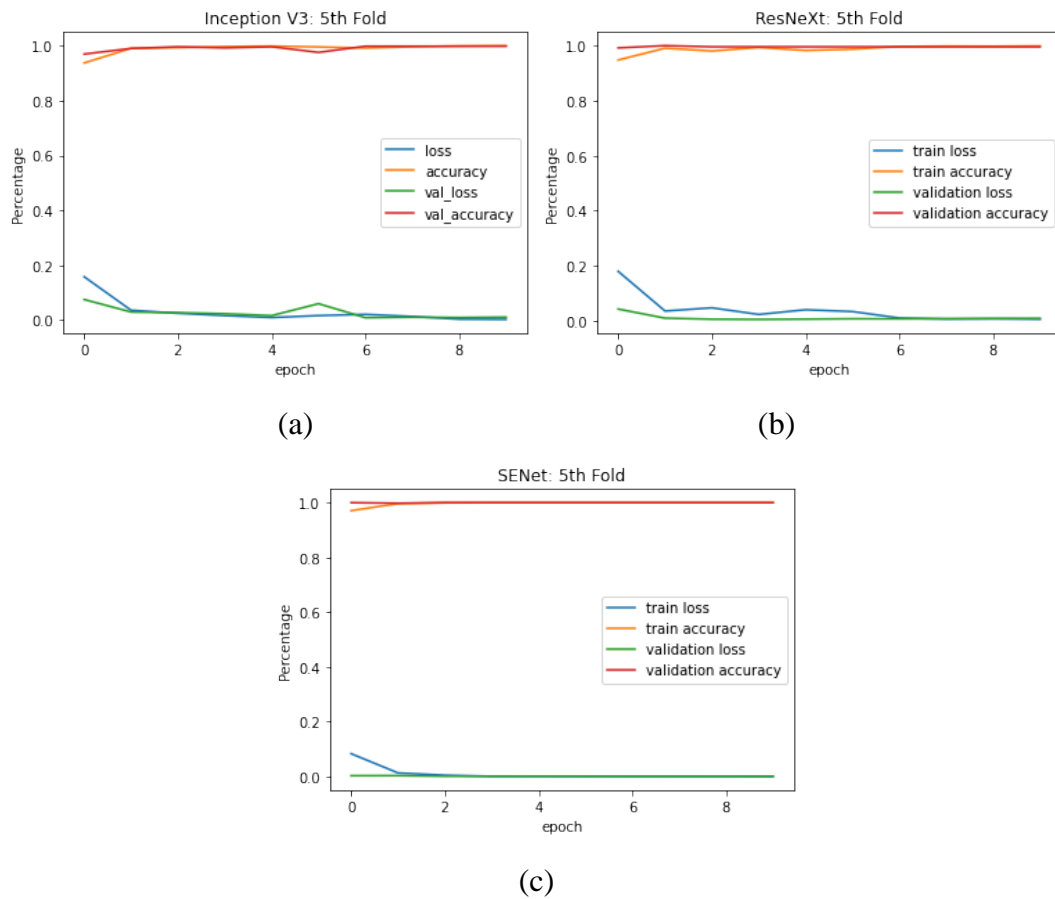


Figure 4.13: Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

**Table 4.8: Precision, Recall, and F1-score for Binary Classification Problem in The Fifth Fold Training for Each Model**

Models	Precision	Recall	F1-score
<b>Inception-V3</b>	99.64%	91.92%	95.62%
<b>ResNeXt</b>	100.00%	92.26%	95.97%
<b>SENet</b>	100.00%	99.66%	99.83%

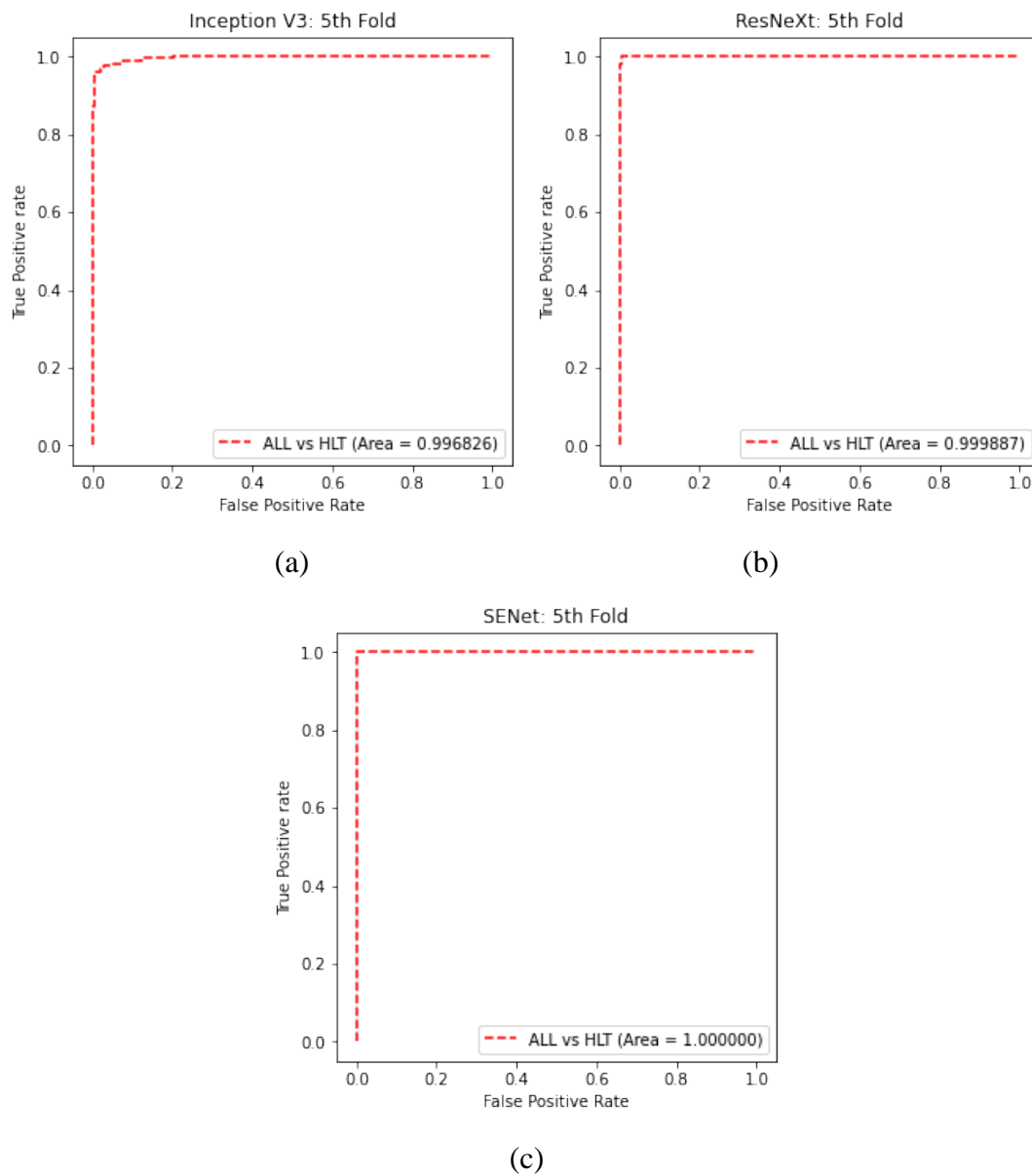


Figure 4.14: ROC Curve in The Fifth Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

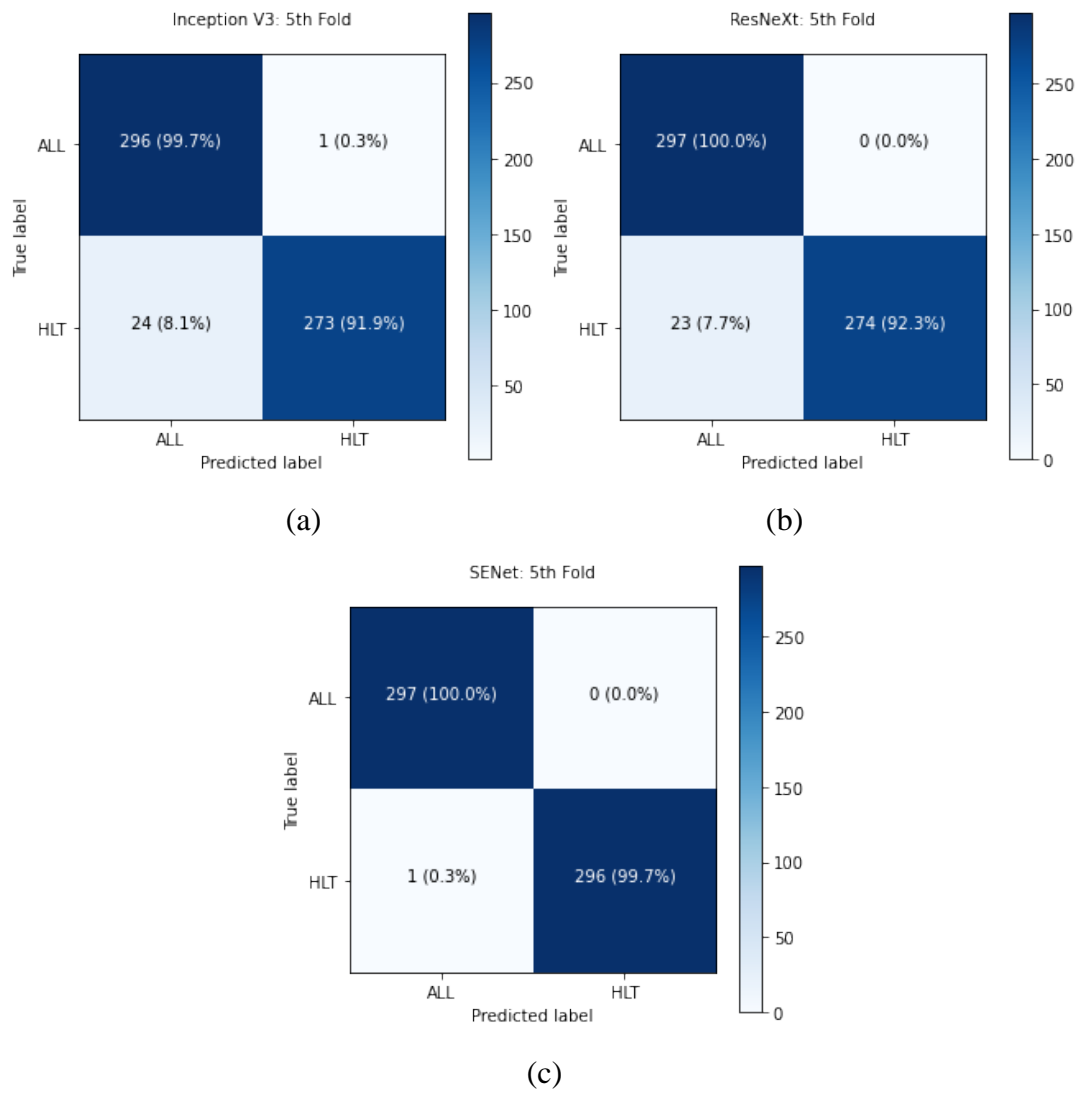


Figure 4.15: Confusion Matrix in The Fifth Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet



#### 4.2 5-class Classification Problem for Inception-V3, ResNeXt. And SENet

**Table 4.9: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for Inception-V3**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	80.07%	73.60%	67.58%	69.43%	76.43%	73.42%
val_acc	98.44%	98.03%	97.64%	98.03%	97.71%	97.97%
trn_acc	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
test_loss	0.9687	1.6426	1.5015	1.5968	1.1512	1.3722
val_loss	0.0537	0.0837	0.0608	0.0568	0.0768	0.0664
trn_loss	0.0041	0.0034	0.0011	0.0008	0.0027	0.0024

**Table 4.10: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for ResNeXt**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	83.37%	72.93%	80.55%	83.03%	84.04%	80.78%
val_acc	99.18%	99.59%	99.83%	99.51%	99.59%	99.54%
trn_acc	100.00%	98.81%	100.00%	99.92%	99.98%	99.74%
test_loss	0.9151	1.6140	1.1333	0.7043	0.8641	1.0462
val_loss	0.0280	0.0193	0.0072	0.0235	0.0191	0.0194
trn_loss	0.0004	0.0302	0.0006	0.0028	0.0022	0.0072

**Table 4.11: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	85.99%	78.18%	84.30%	79.26%	87.41%	83.03%
val_acc	99.84%	99.92%	99.75%	100.00%	99.75%	99.85%
trn_acc	99.92%	99.92%	99.98%	99.98%	99.63%	99.89%
test_loss	1.0089	1.4780	0.7474	1.1206	0.0066	0.8723
val_loss	0.0023	0.0032	0.0088	0.0027	0.0098	0.0054
trn_loss	0.0039	0.0021	0.0016	0.0012	0.0113	0.0040

## 4.2.1 Fold 1

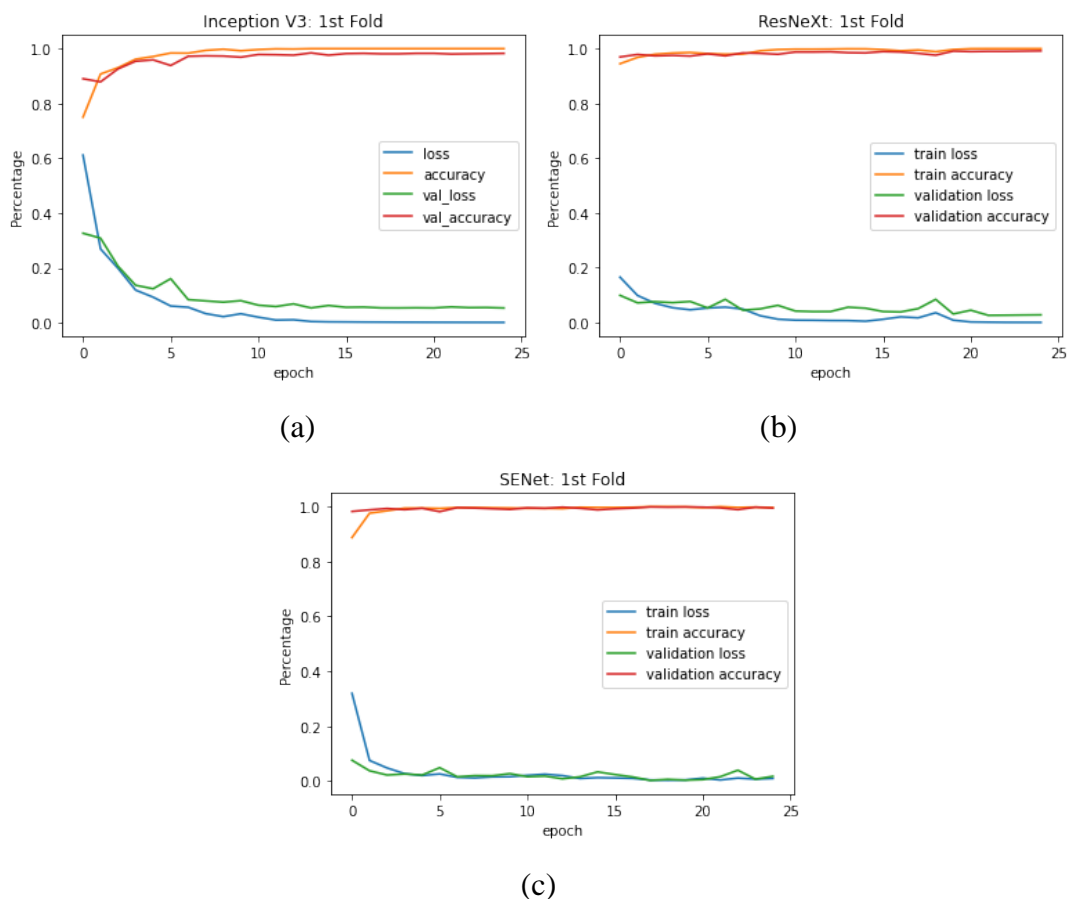
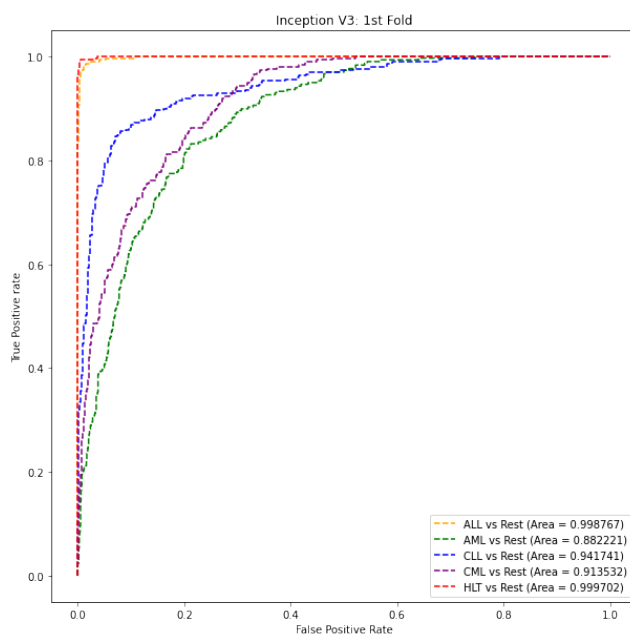


Figure 4.16: Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

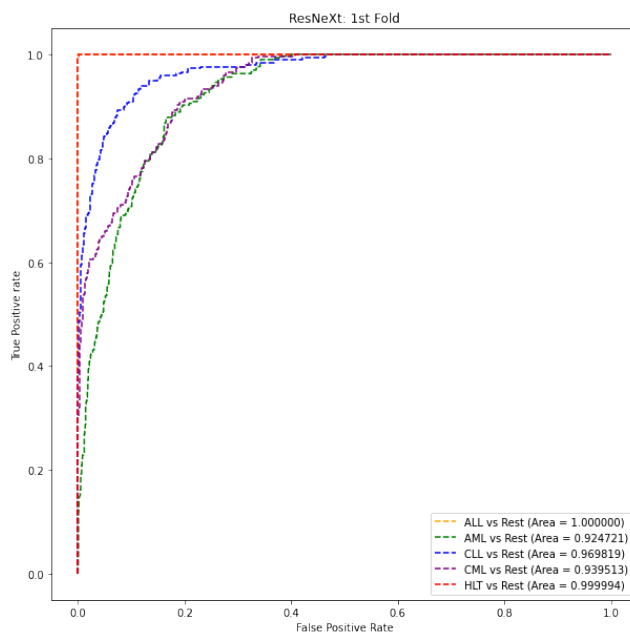
**Table 4.12: Precision, Recall, and F1-score for 5-class Classification Problem in The First Fold Training for Each Model**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	95.70%	97.31%	96.49%
	<b>AML</b>	61.18%	62.63%	61.90%
	<b>CLL</b>	85.25%	70.03%	76.89%
	<b>CML</b>	63.47%	71.38%	67.19%
	<b>Healthy</b>	97.67%	98.99%	98.33%
<b>ResNeXt</b>	<b>ALL</b>	100.00%	99.66%	99.83%
	<b>AML</b>	99.33%	73.40%	67.81%
	<b>CLL</b>	100.00%	74.41%	80.36%

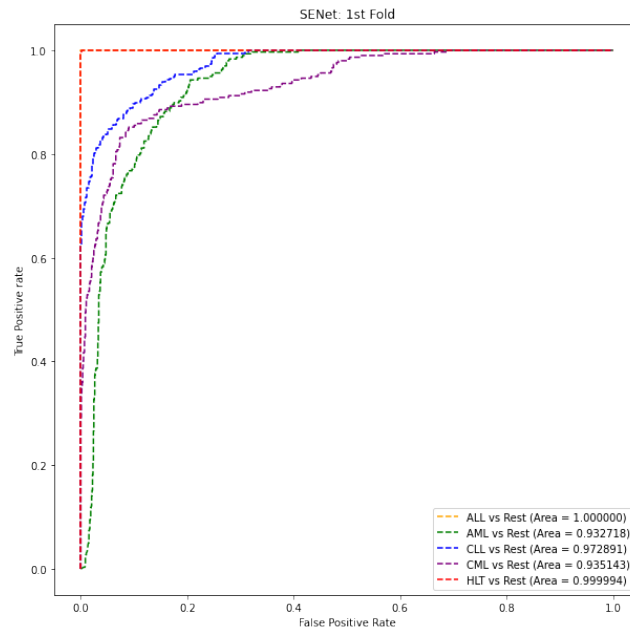
	<b>CML</b>	96.78%	69.36%	70.43%
	<b>Healthy</b>	100.00%	100.00%	99.17%
<b>SENet</b>	<b>ALL</b>	100.00%	100.00%	100.00%
	<b>AML</b>	73.61%	71.38%	72.48%
	<b>CLL</b>	91.06%	75.42%	82.50%
	<b>CML</b>	71.59%	83.16%	76.95%
	<b>Healthy</b>	96.12%	100.00%	98.02%



(a)

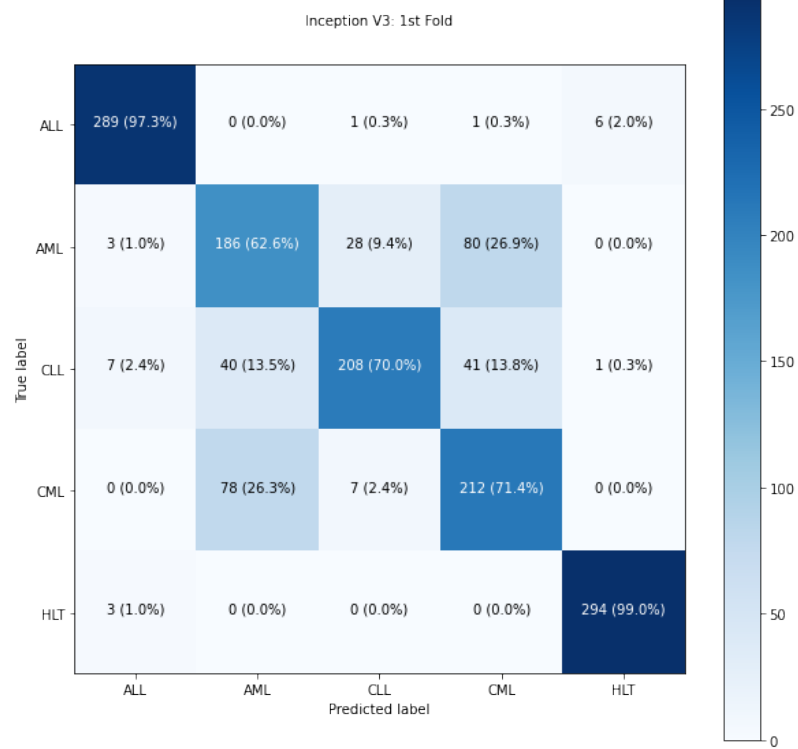


(b)

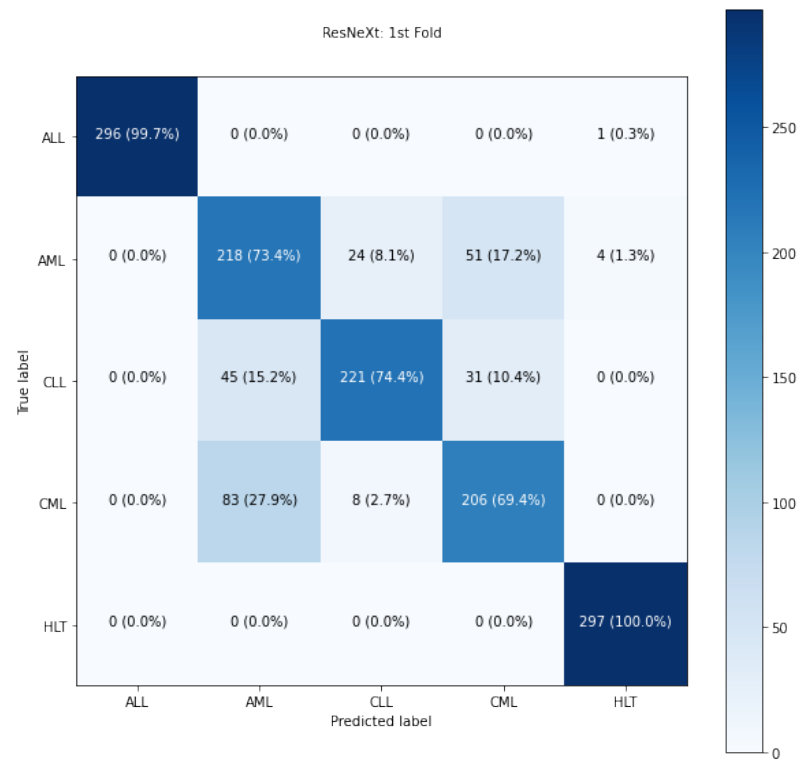


(c)

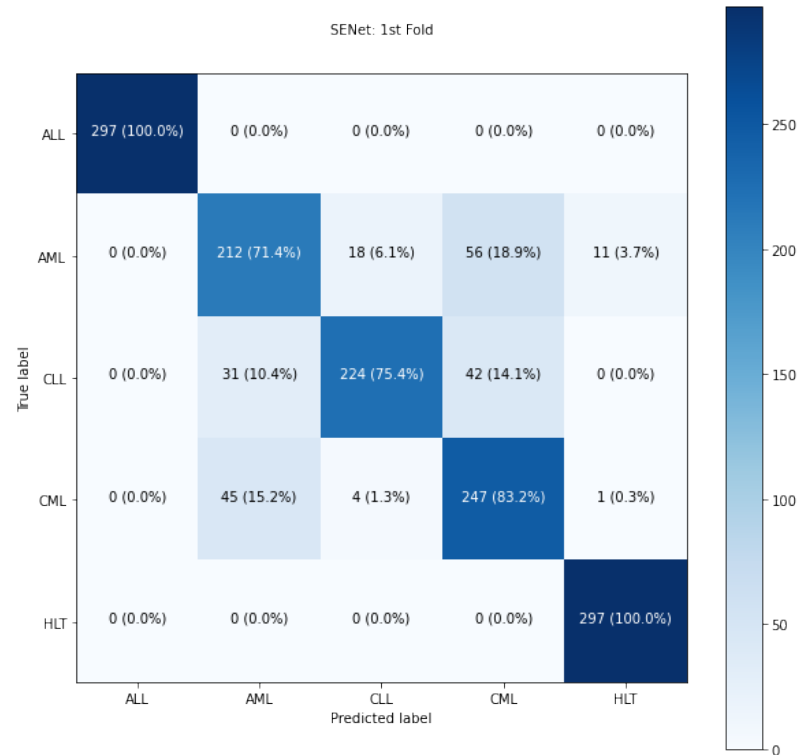
Figure 4.17: ROC Curve in The First Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet



(a)



(b)



(c)

Figure 4.18: Confusion Matrix in The First Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

## 4.2.2 Fold 2

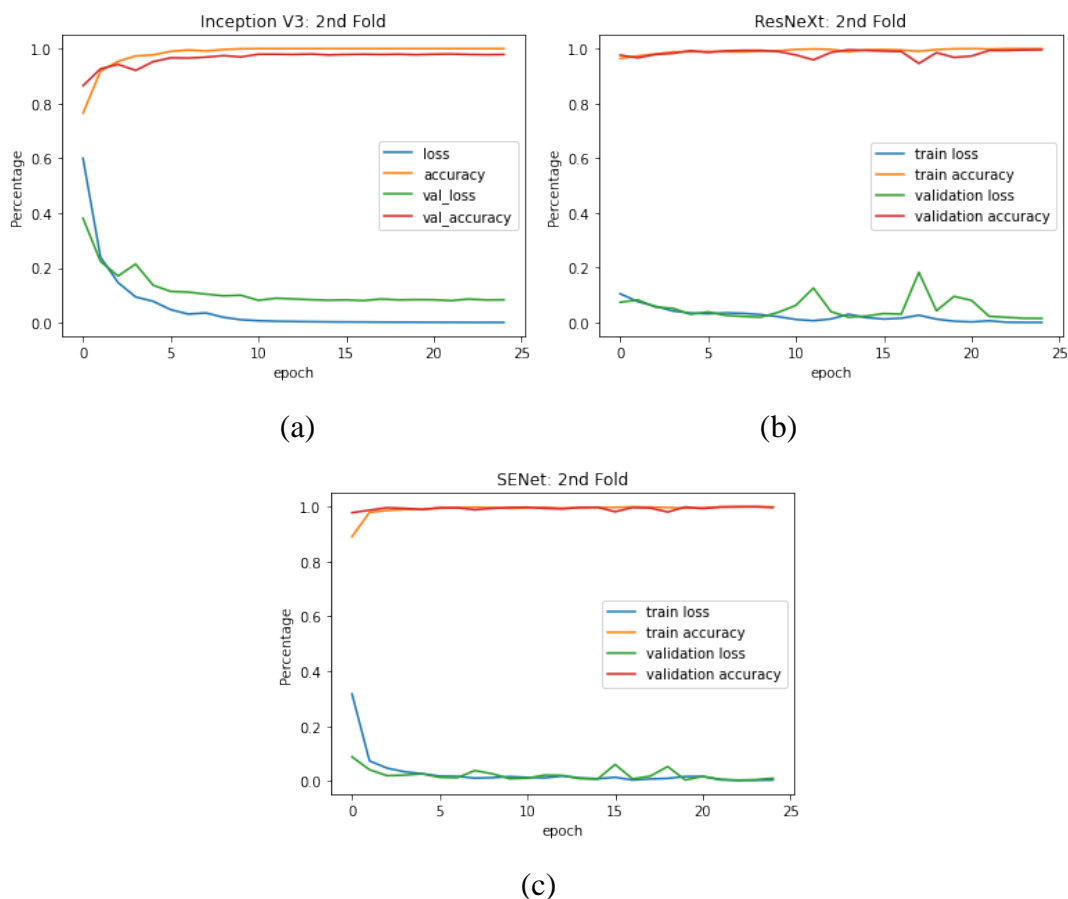
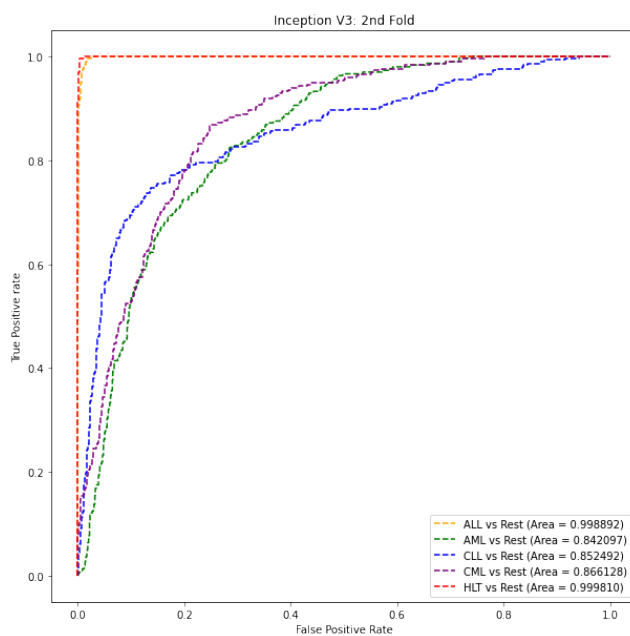


Figure 4.19: Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

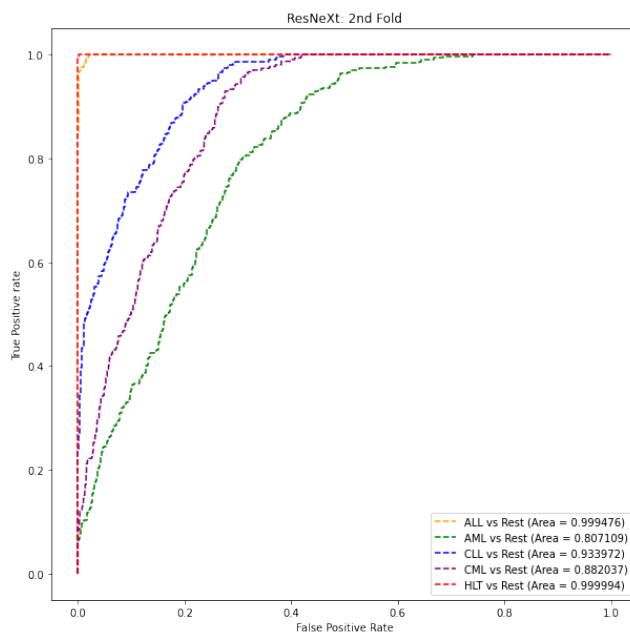
**Table 4.13: Precision, Recall, and F1-score for 5-class Classification Problem in The Second Fold Training for Each Model**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	95.10%	97.98%	96.52%
	<b>AML</b>	58.56%	35.69%	44.35%
	<b>CLL</b>	64.06%	69.02%	66.45%
	<b>CML</b>	53.44%	65.32%	58.79%
	<b>Healthy</b>	94.29%	100.00%	97.06%
<b>ResNeXt</b>	<b>ALL</b>	92.21%	99.66%	95.79%
	<b>AML</b>	56.20%	22.90%	32.54%
	<b>CLL</b>	68.71%	68.01%	68.36%

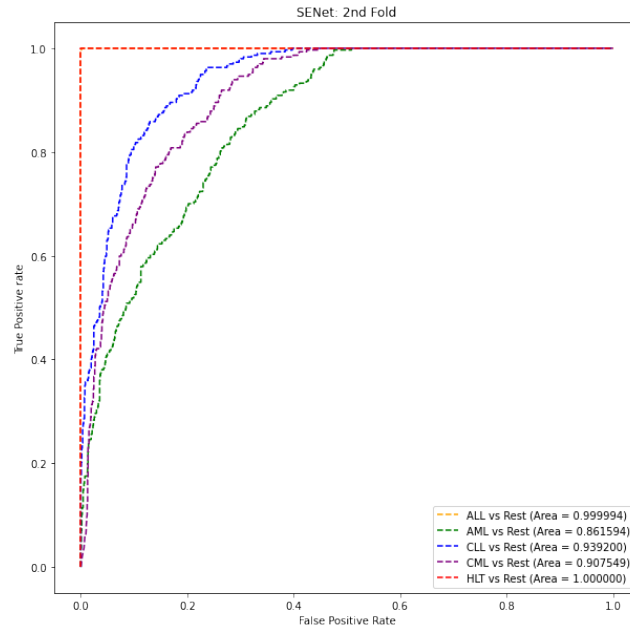
	<b>CML</b>	48.89%	74.41%	59.01%
	<b>Healthy</b>	99.66%	99.66%	99.66%
<b>SENet</b>	<b>ALL</b>	99.00%	100.00%	99.50%
	<b>AML</b>	60.25%	48.48%	53.73%
	<b>CLL</b>	69.65%	73.40%	71.48%
	<b>CML</b>	61.01%	69.02%	64.77%
	<b>Healthy</b>	100.00%	100.00%	100.00%



(a)

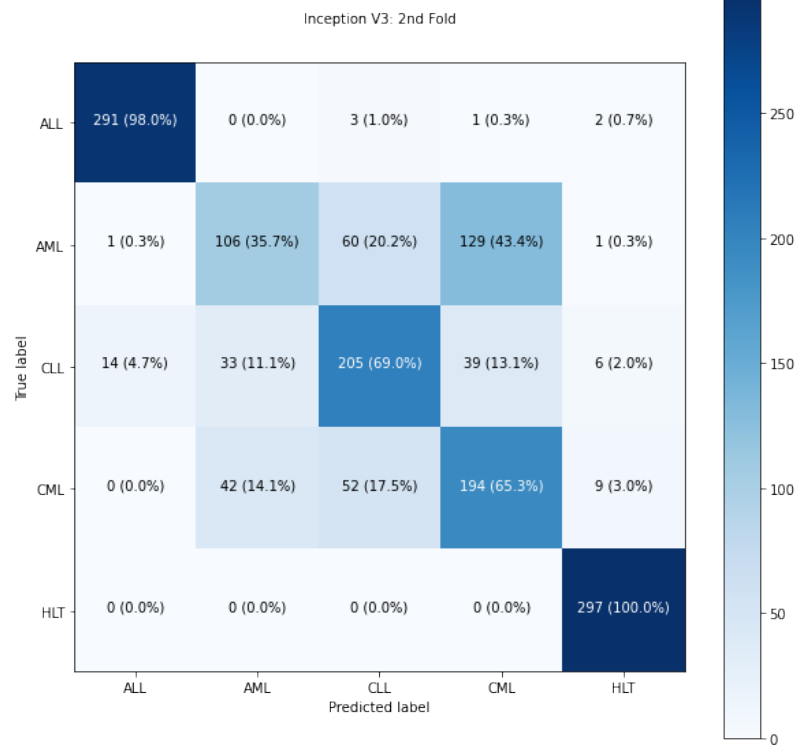


(b)



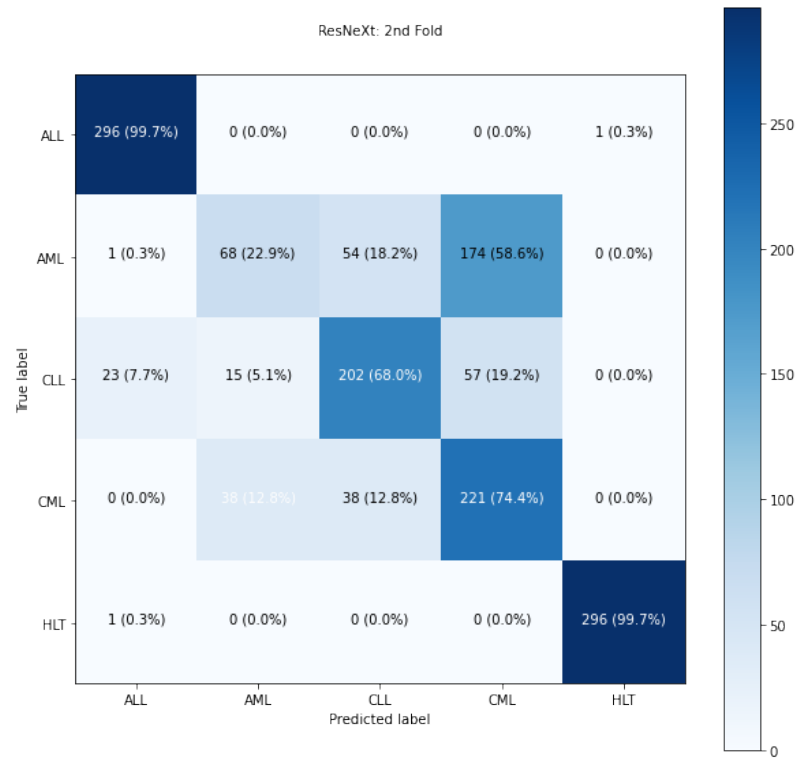
(c)

Figure 4.20: ROC Curve in The Second Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

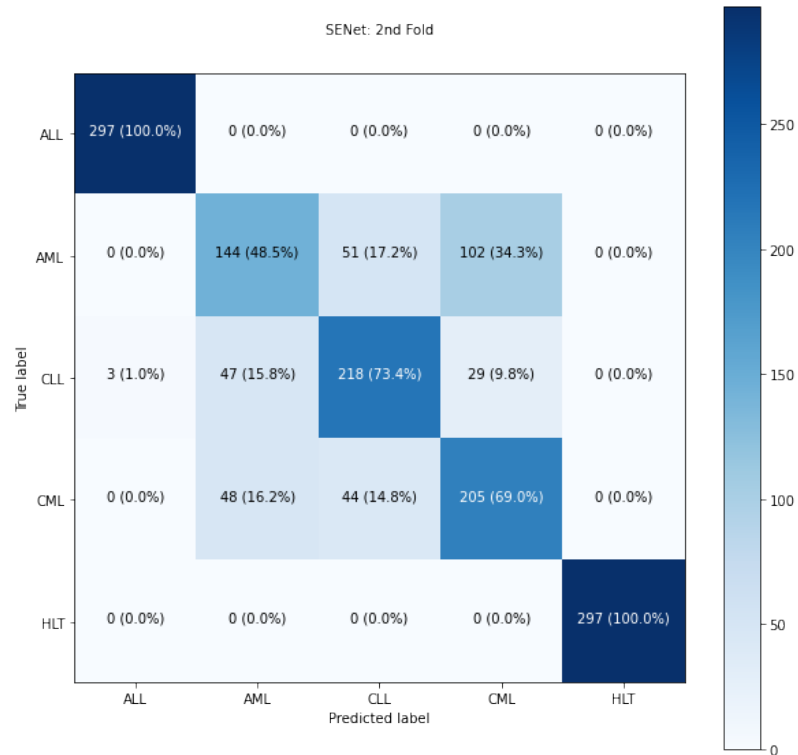


(a)





(b)



(c)

Figure 4.21: Confusion Matrix in The Second Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

## 4.2.3 Fold 3

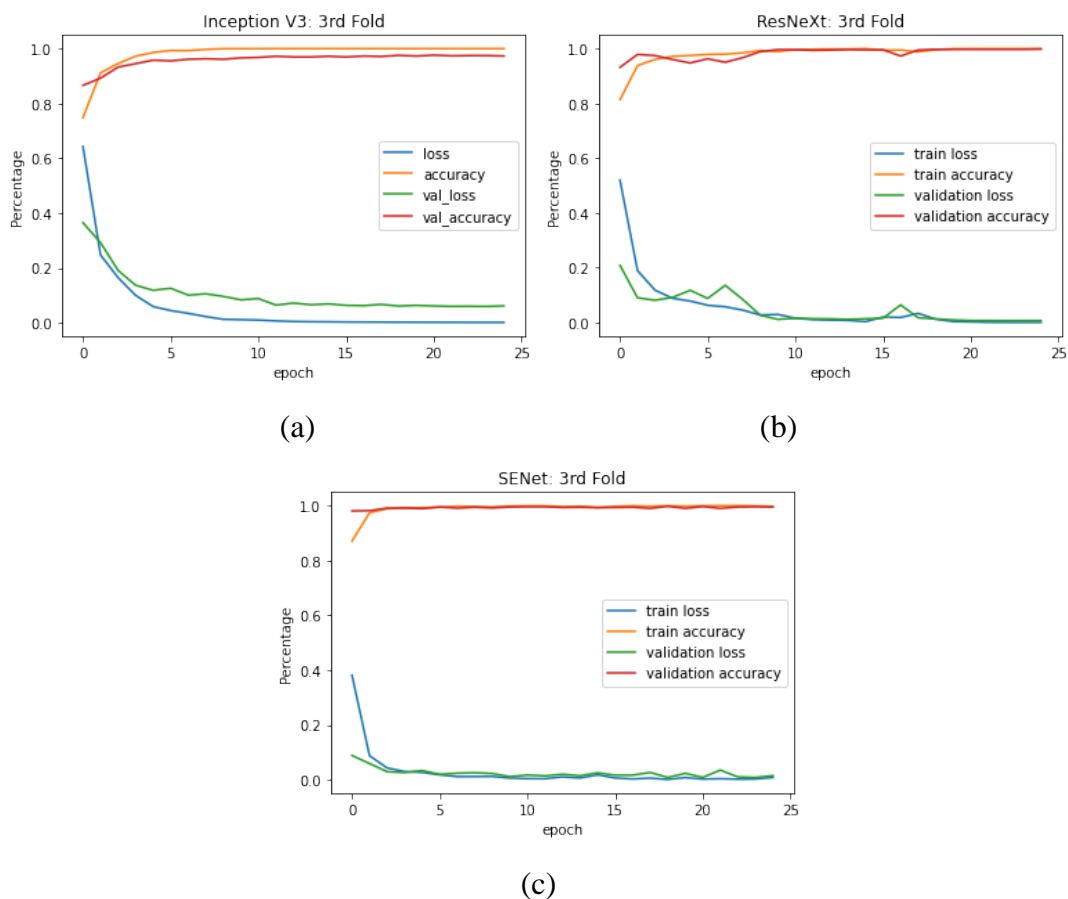
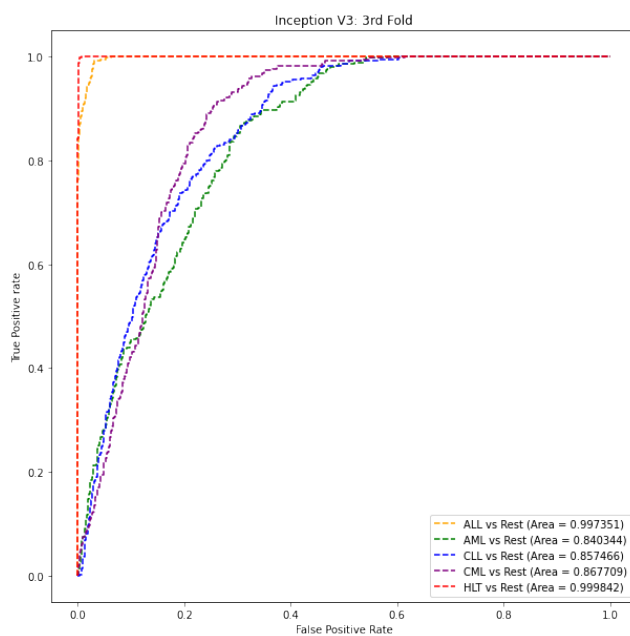


Figure 4.22: Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

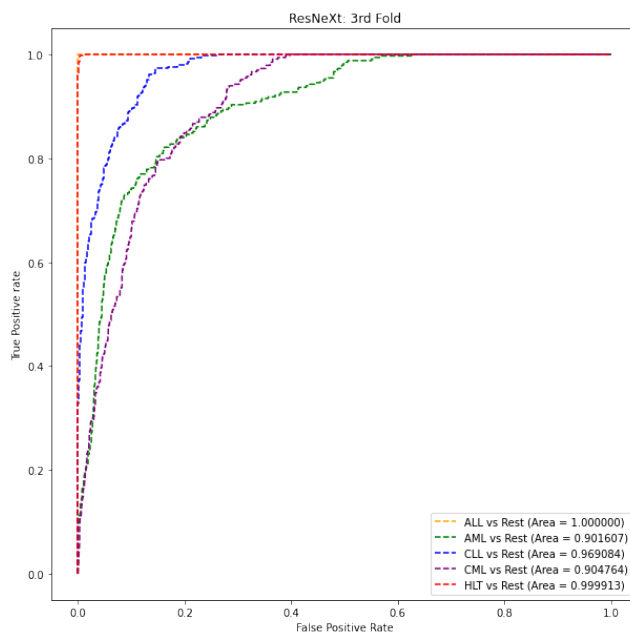
**Table 4.14: Precision, Recall, and F1-score for 5-class Classification Problem in The Third Fold Training for Each Model**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	83.46%	99.39%	90.73%
	<b>AML</b>	46.19%	59.79%	51.73%
	<b>CLL</b>	58.43%	31.52%	40.94%
	<b>CML</b>	50.32%	48.18%	49.23%
	<b>Healthy</b>	96.21%	100.00%	98.07%
<b>ResNeXt</b>	<b>ALL</b>	99.70%	100.00%	99.85%
	<b>AML</b>	61.20%	76.97%	68.19%
	<b>CLL</b>	89.79%	63.94%	74.69%

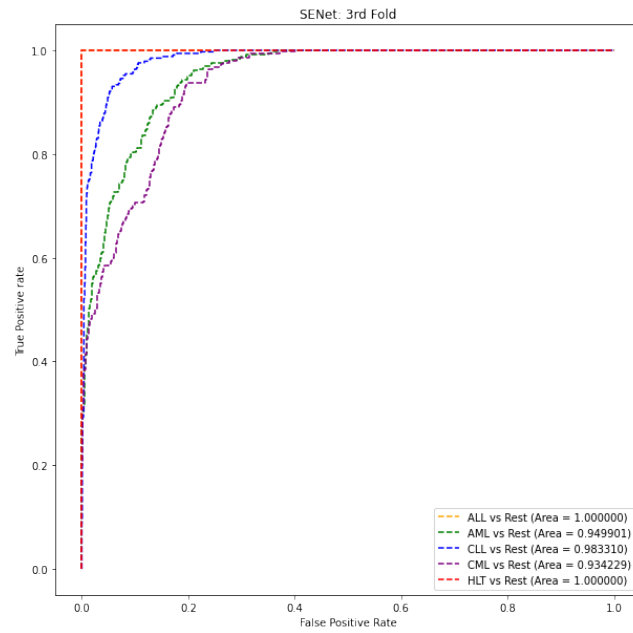
	<b>CML</b>	62.50%	62.12%	62.31%
	<b>Healthy</b>	96.48%	99.70%	98.06%
<b>SENet</b>	<b>ALL</b>	99.70%	100.00%	99.85%
	<b>AML</b>	63.41%	84.55%	72.47%
	<b>CLL</b>	90.88%	78.48%	84.23%
	<b>CML</b>	73.95%	58.48%	65.31%
	<b>Healthy</b>	99.10%	100.00%	99.55%



(a)

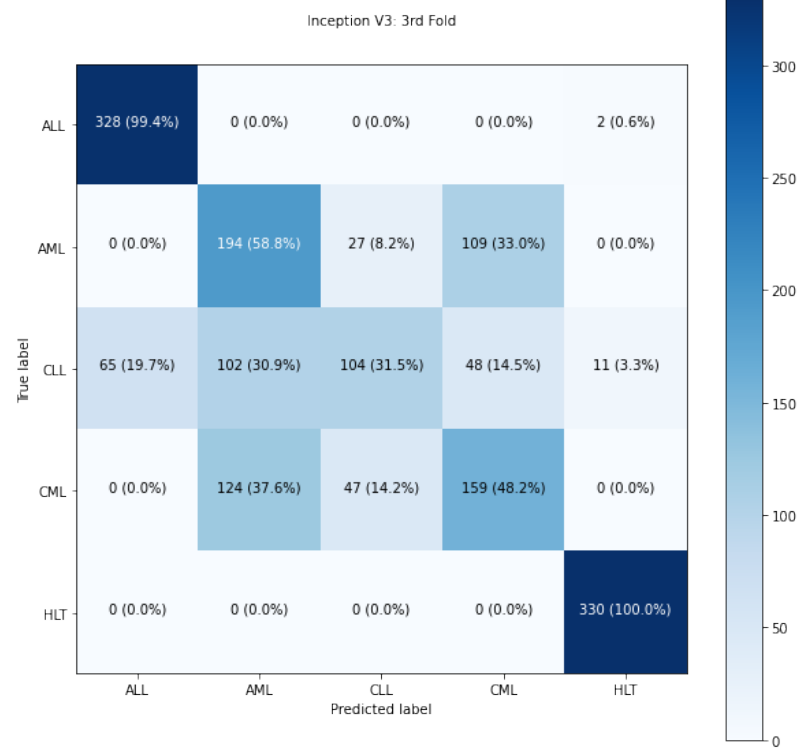


(b)

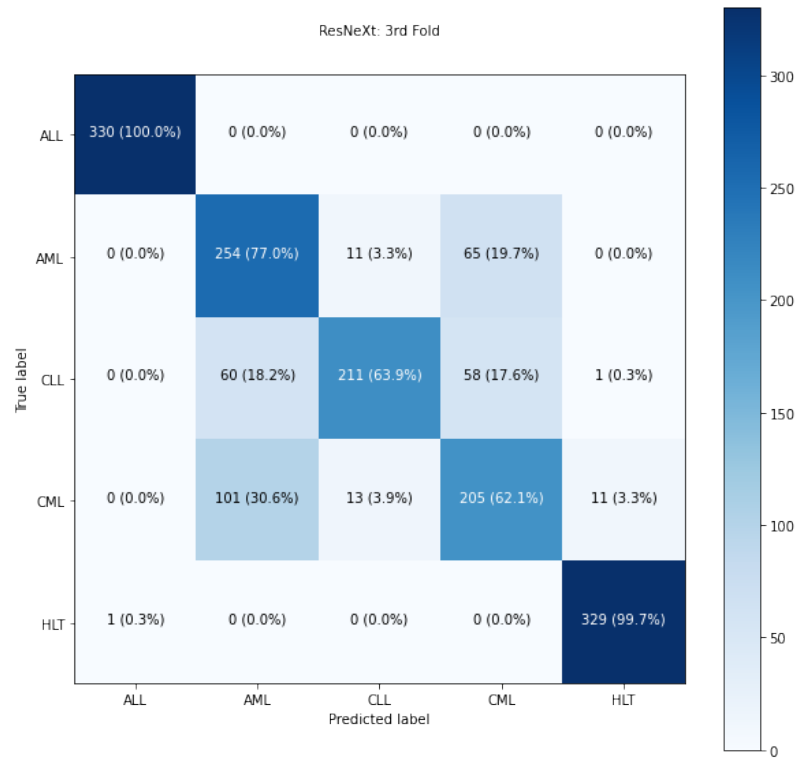


(c)

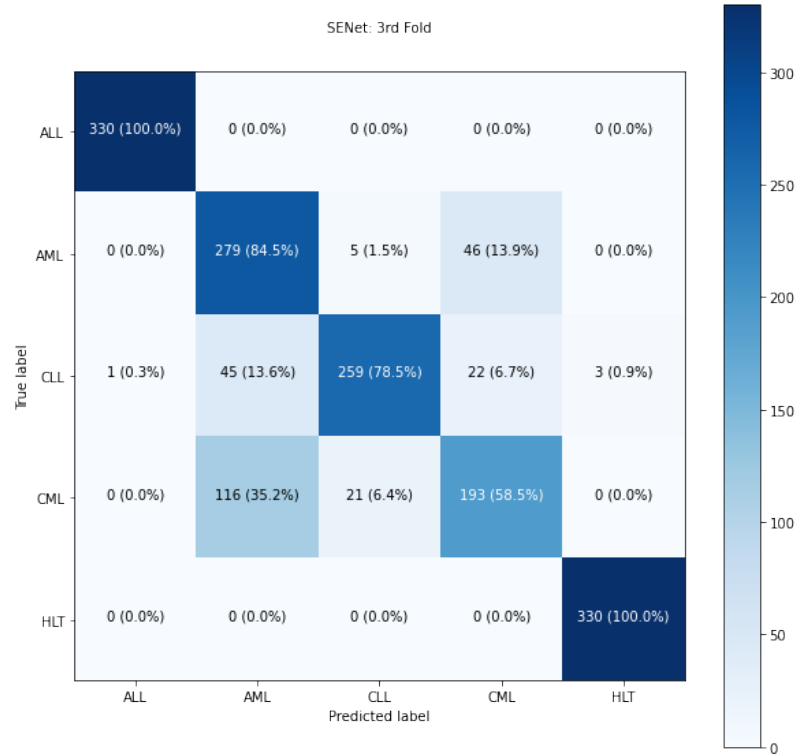
Figure 4.23: ROC Curve in The Third Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet



(a)



(b)



(c)

Figure 4.24: Confusion Matrix in The Third Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

#### 4.2.4 Fold 4

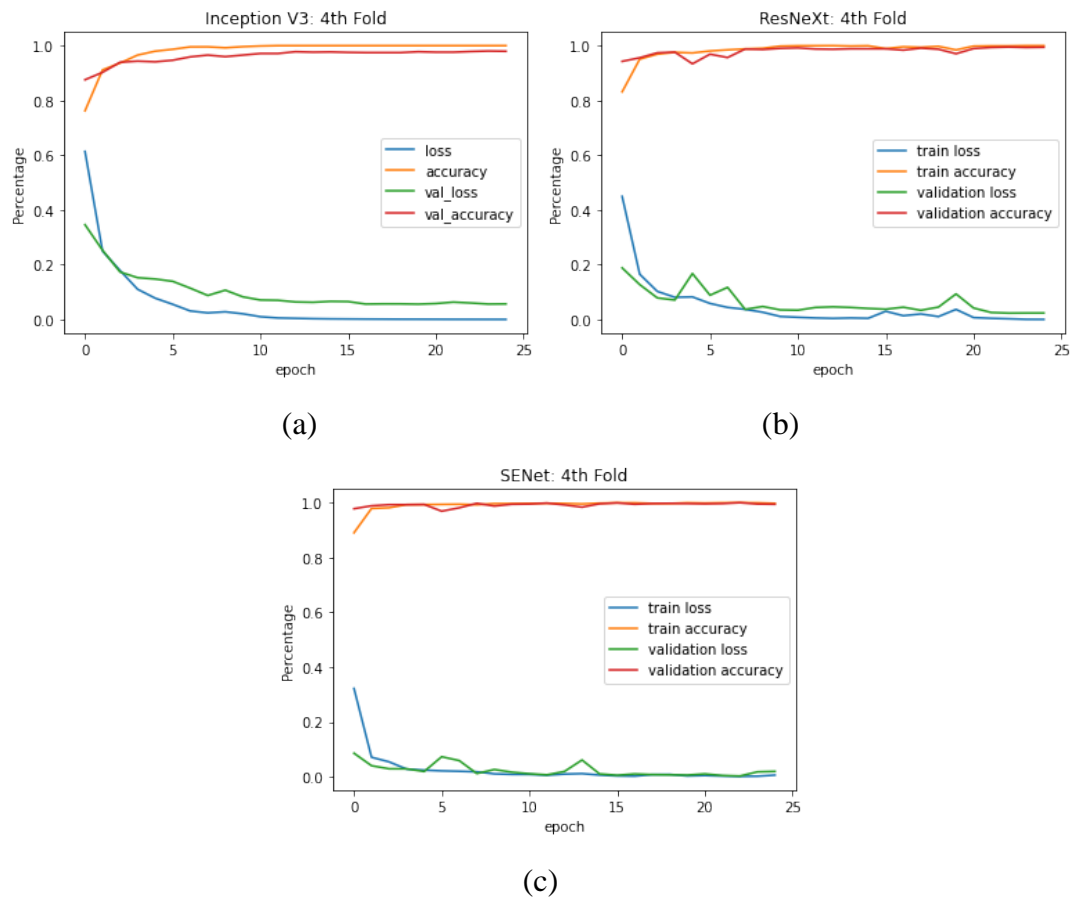
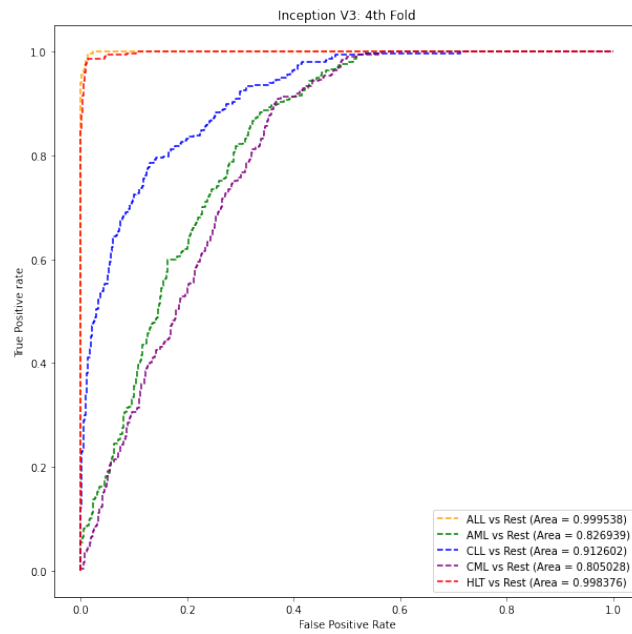


Figure 4.25: Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

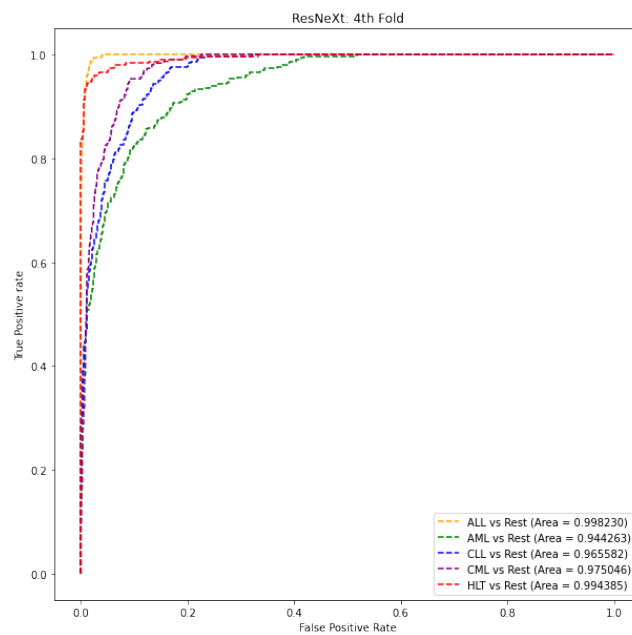
**Table 4.15: Precision, Recall, and F1-score for 5-class Classification Problem in The Fourth Fold Training for Inception-V3**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	95.75%	98.65%	97.18%
	<b>AML</b>	45.91%	49.16%	47.48%
	<b>CLL</b>	71.86%	63.64%	67.50%
	<b>CML</b>	40.92%	44.78%	42.77%
	<b>Healthy</b>	99.90%	90.91%	94.74%
<b>ResNeXt</b>	<b>ALL</b>	86.76%	99.33%	92.62%
	<b>AML</b>	72.20%	76.09%	74.10%
	<b>CLL</b>	84.62%	66.67%	74.58%

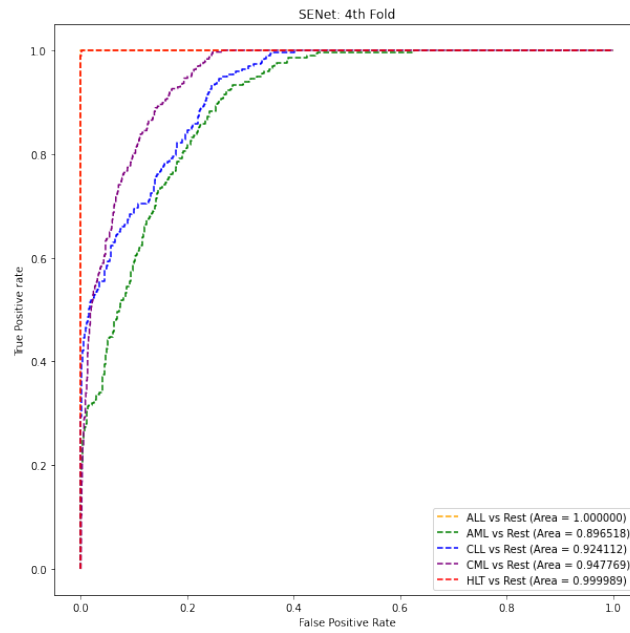
	<b>CML</b>	76.59%	89.23%	82.43%
	<b>Healthy</b>	98.81%	83.84%	90.71%
<b>SENet</b>	<b>ALL</b>	100.00%	100.00%	100.00%
	<b>AML</b>	62.04%	51.18%	56.09%
	<b>CLL</b>	73.95%	59.26%	65.79%
	<b>CML</b>	62.90%	86.20%	72.73%
	<b>Healthy</b>	99.33%	99.66%	99.50%



(a)

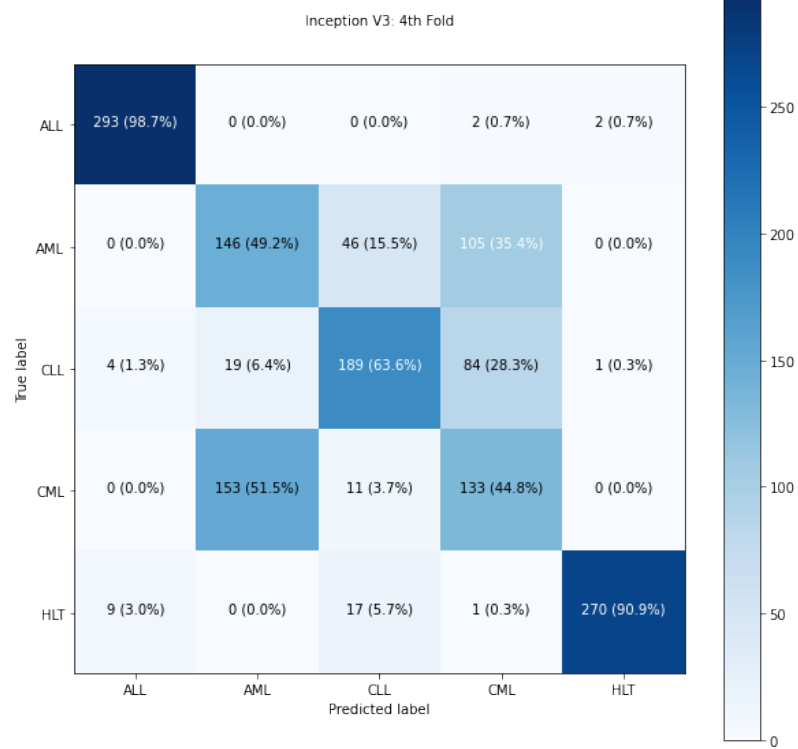


(b)



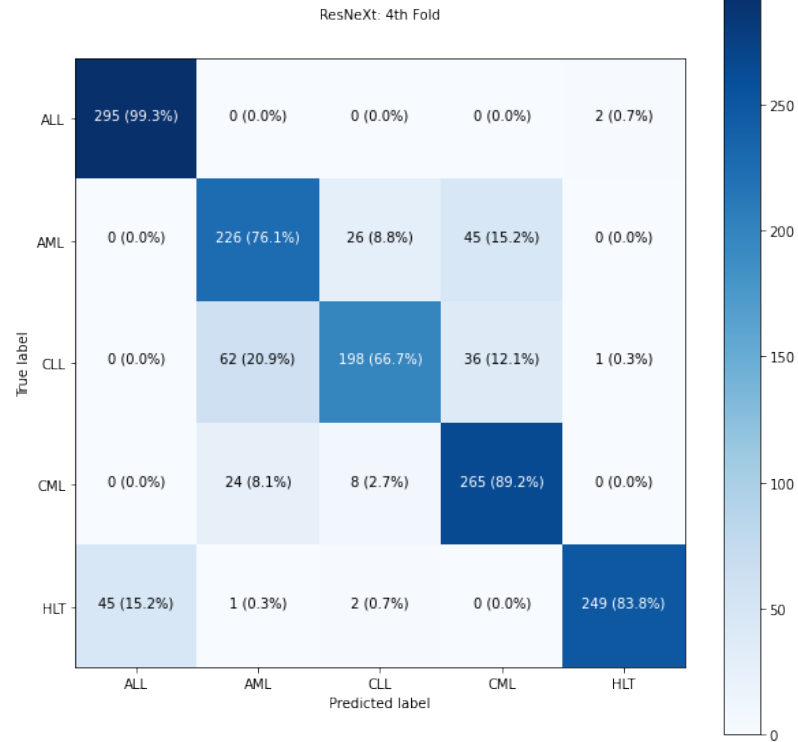
(c)

Figure 4.26: ROC Curve in The Fourth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

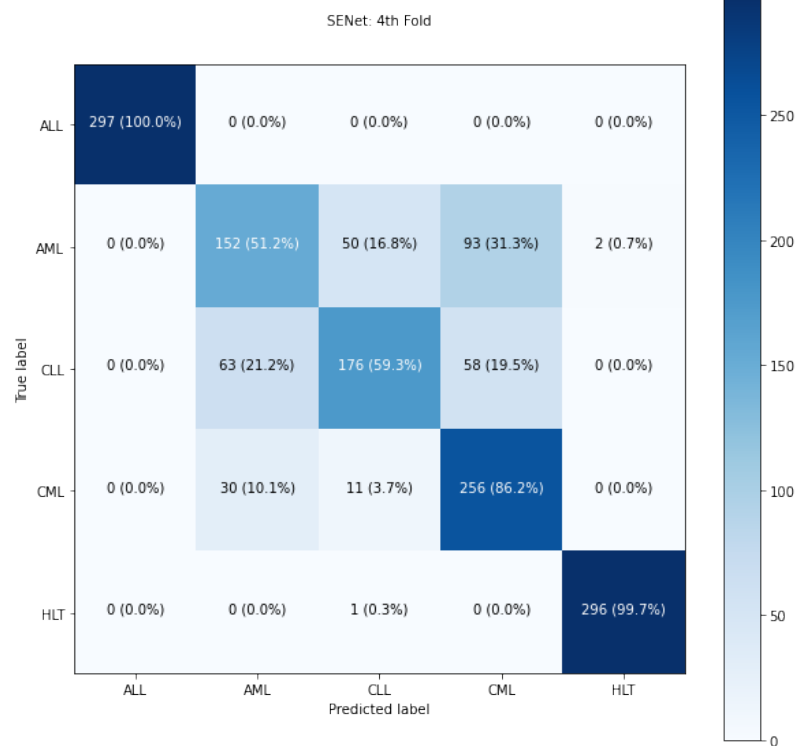


(a)





(b)



(c)

Figure 4.27: Confusion Matrix in The Fourth Fold Training for (a) Inception-V3  
(b) ResNeXt (c) SENet

## 4.2.5 Fold 5

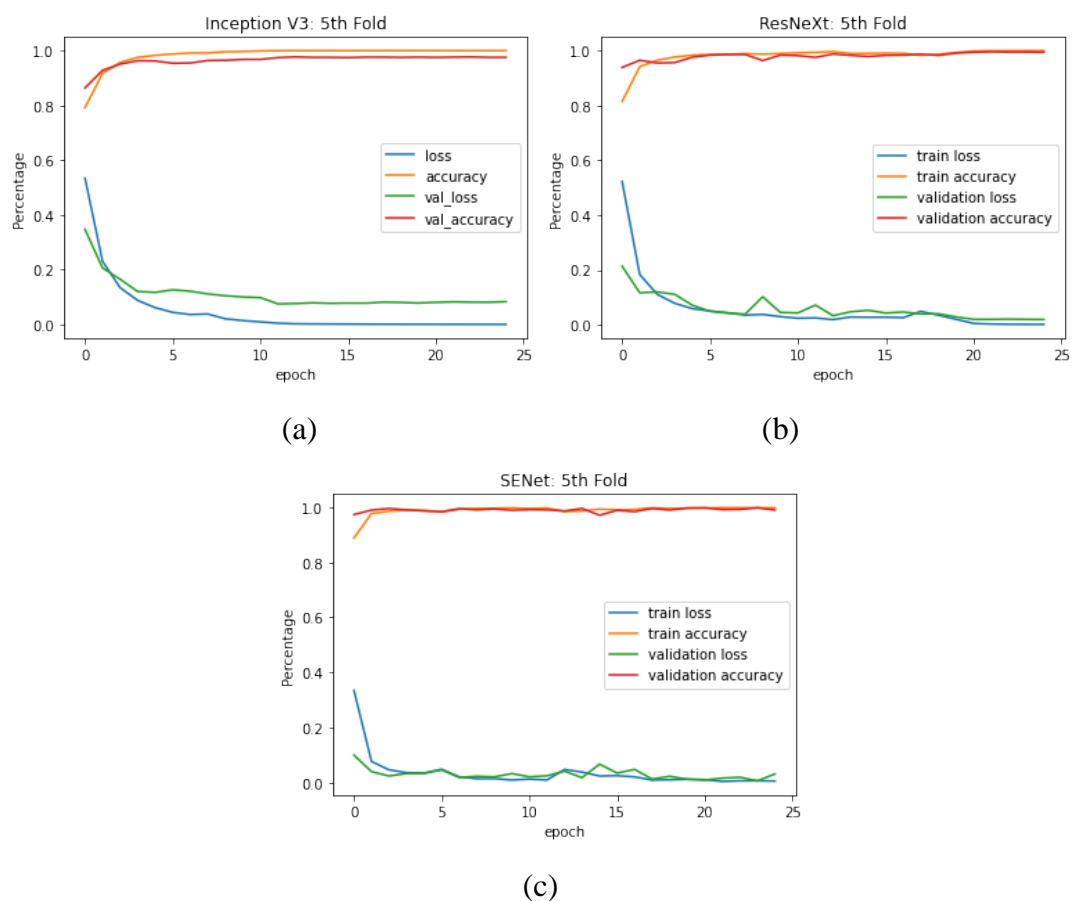
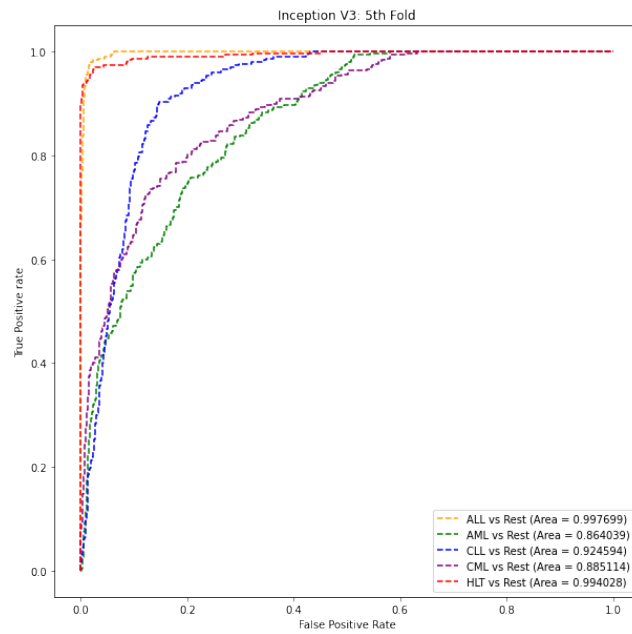


Figure 4.28: Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet

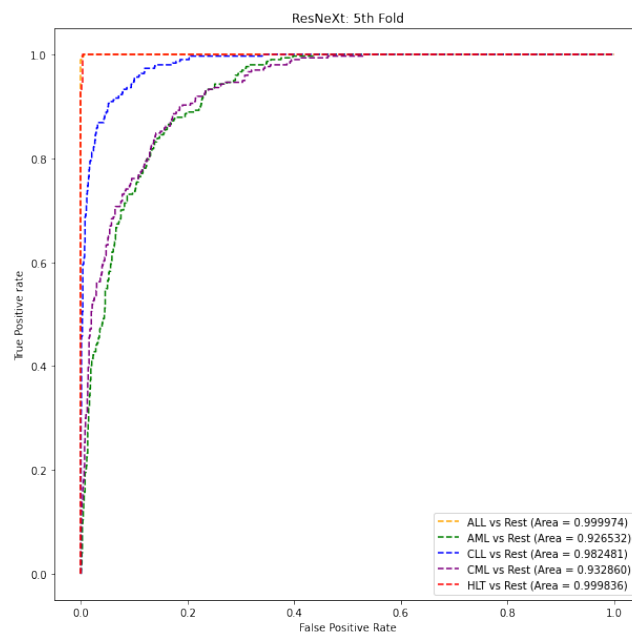
**Table 4.16: Precision, Recall, and F1-score for 5-class Classification Problem in The Fifth Fold Training for Inception-V3**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	91.51%	97.98%	94.63%
	<b>AML</b>	59.14%	55.56%	57.29%
	<b>CLL</b>	65.62%	77.10%	70.90%
	<b>CML</b>	67.45%	57.91%	62.32%
	<b>Healthy</b>	97.89%	93.60%	95.70%
<b>ResNeXt</b>	<b>ALL</b>	96.12%	100.00%	98.02%
	<b>AML</b>	72.62%	61.62%	66.67%
	<b>CLL</b>	81.93%	88.55%	85.11%

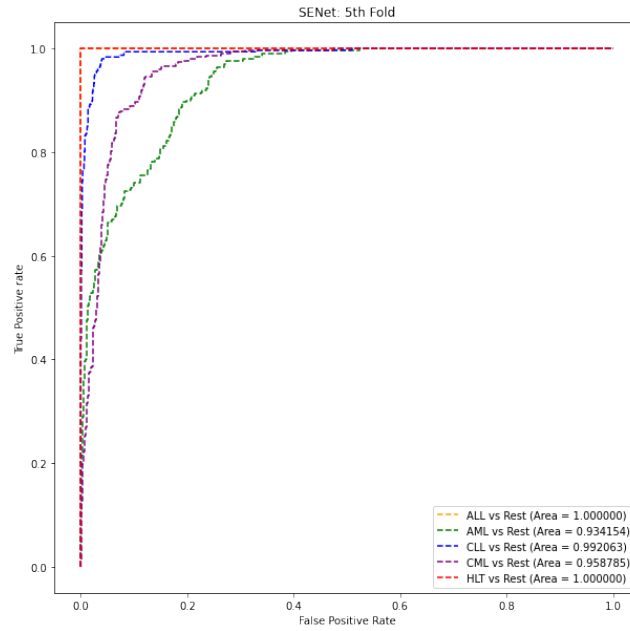
	<b>CML</b>	69.65%	73.40%	71.48%
	<b>Healthy</b>	98.97%	96.63%	97.79%
<b>SENet</b>	<b>ALL</b>	97.70%	100.00%	98.84%
	<b>AML</b>	83.58%	56.57%	67.47%
	<b>CLL</b>	79.78%	98.32%	88.08%
	<b>CML</b>	76.97%	82.15%	79.48%
	<b>Healthy</b>	100.00%	100.00%	100.00%



(a)

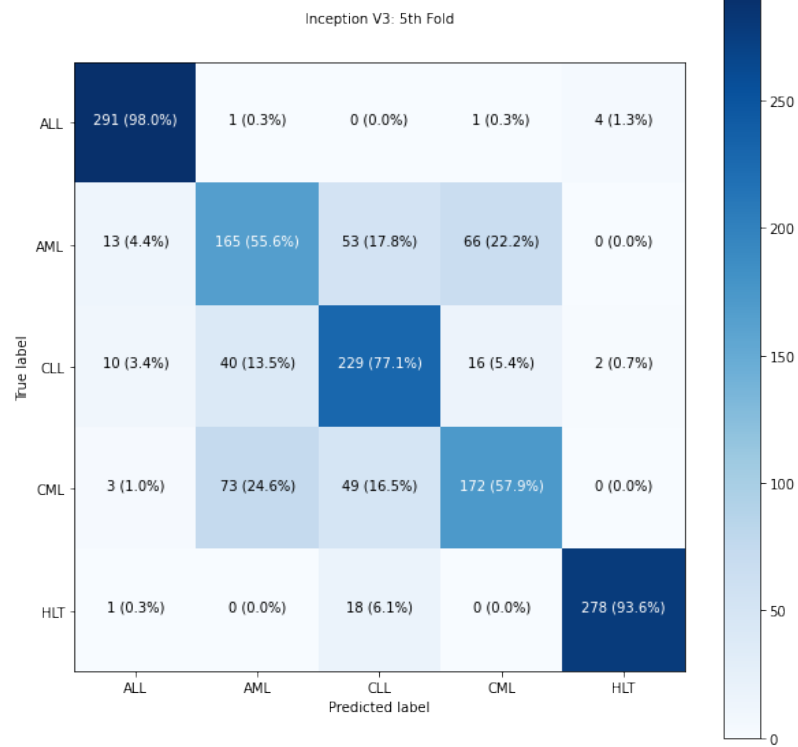


(b)

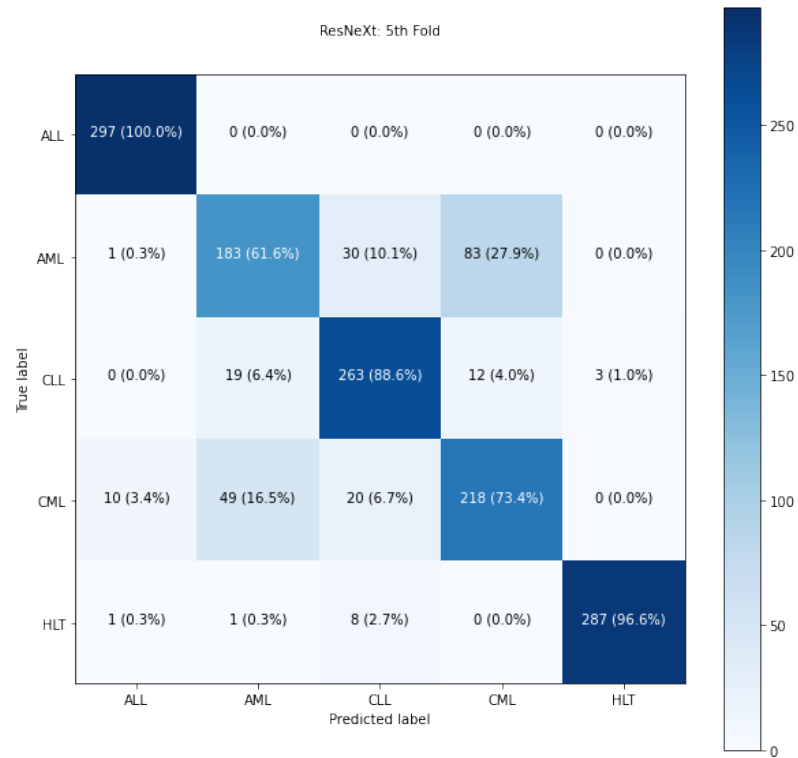


(c)

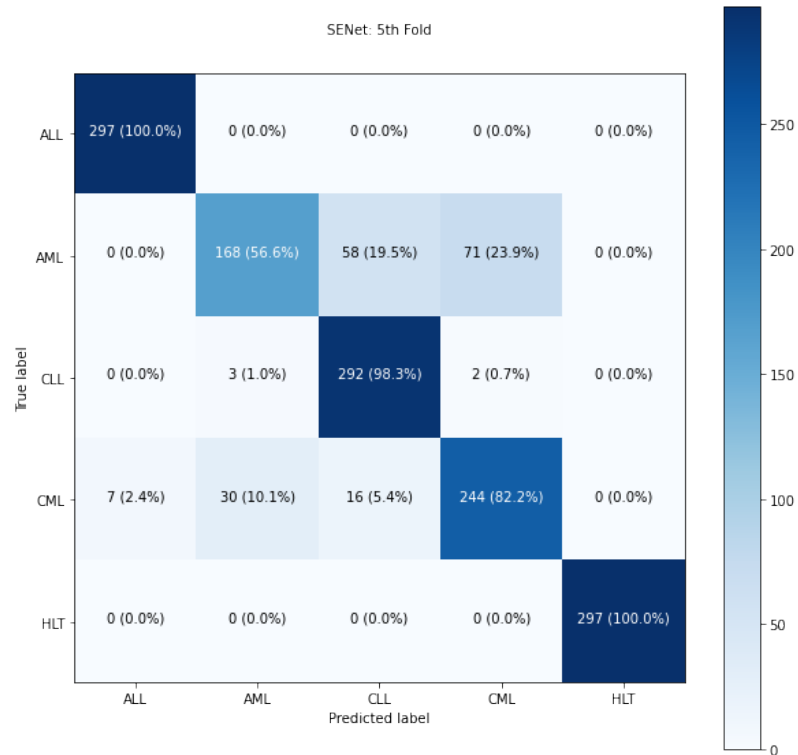
Figure 4.29: ROC Curve in The Fifth Fold Training for (a) Inception-V3 (b) ResNeXt (c) SENet



(a)



(b)



(c)

Figure 4.30: Confusion Matrix in The Fifth Fold Training for (a) Inception-V3

(b) ResNeXt (c) SENet

### 4.3 Fine-tuned SENet Models

**Table 4.17: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet + SVM**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	84.04%	75.22%	85.52%	75.22%	83.83%	80.77%

**Table 4.18: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet with 3 Hidden Layers**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	85.19%	78.59%	87.15%	79.60%	87.21%	83.55%
val_acc	99.92%	99.92%	99.66%	99.92%	100.00%	99.88%
trn_acc	99.63%	99.65%	99.66%	99.96%	99.65%	99.71%
test_loss	0.8970	1.3803	0.4585	1.5448	0.6734	0.9908
val_loss	0.0067	0.0020	0.0109	0.0026	0.0008	0.0046
trn_loss	0.0126	0.0128	0.0151	0.0005	0.0134	0.0109

**Table 4.19: Accuracy and Loss Result for 5-class Classification Problem of Each Fold for SENet with 3 Hidden Layers Plus Dropout Layers**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	86.33%	79.19%	86.55%	82.49%	87.68%	84.48%
val_acc	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
trn_acc	99.65%	99.22%	99.22%	99.22%	99.22%	99.31%
test_loss	0.9953	1.4517	0.3992	0.8036	0.6182	0.8536
val_loss	0.0008	0.0017	0.0017	0.0017	0.0017	0.0015
trn_loss	0.0134	0.0307	0.0307	0.0307	0.0307	0.0272

## 4.3.1 Fold 1

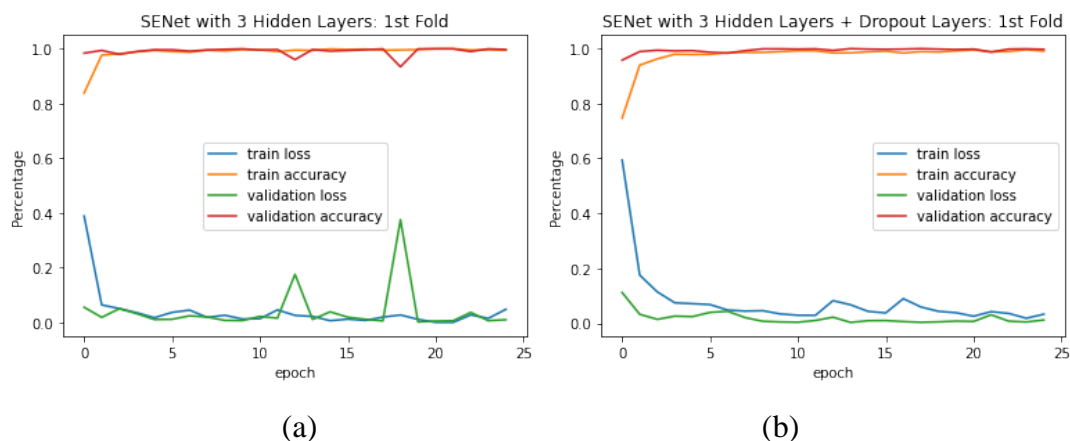
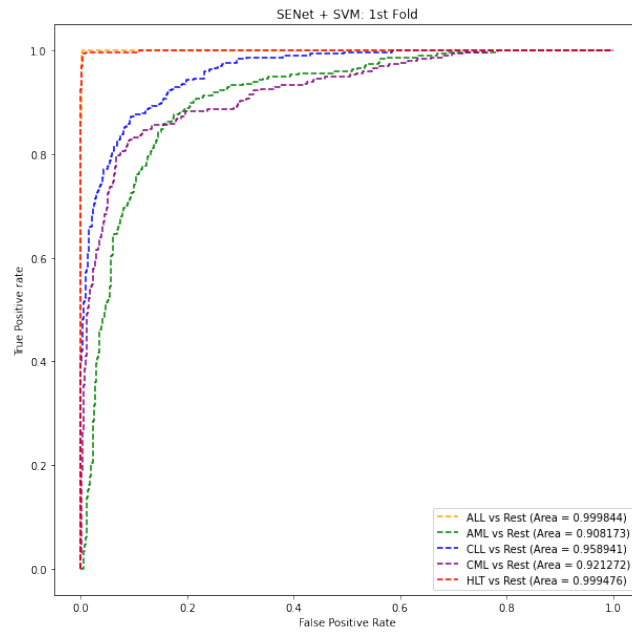


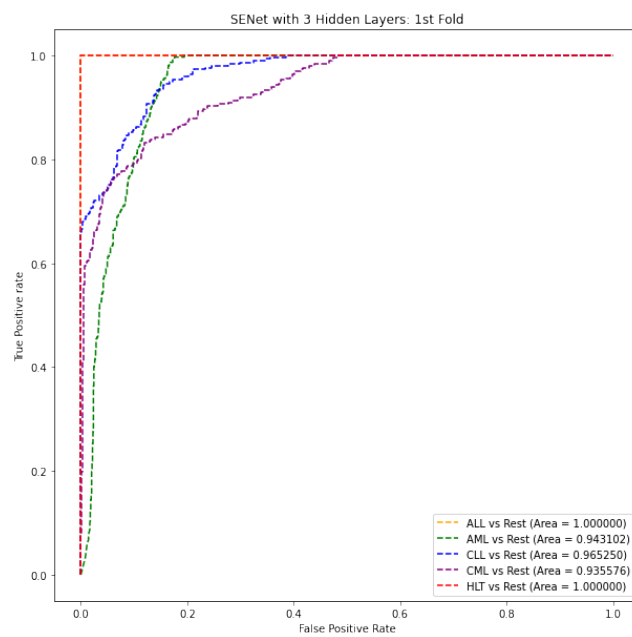
Figure 4.31: Accuracy and Loss Against Number of Epochs in The First Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.20: Precision, Recall, and F1-score for 5-class Classification Problem in The First Fold Training for Each Fine-tuned SENet Models**

Model	Classes	Precision	Recall	F1-score
SENet + SVM	ALL	98.02%	100.00%	99.00%
	AML	68.46%	68.69%	68.57%
	CLL	85.66%	72.39%	78.47%
	CML	71.82%	79.80%	75.60%
	Healthy	97.36%	99.33%	98.33%
SENet with 3 Hidden Layers	ALL	100.00%	100.00%	100.00%
	AML	66.76 %	80.47%	72.98%
	CLL	97.12%	68.01%	80.00%
	CML	73.02%	77.44%	75.16%
	Healthy	96.74%	100.00%	98.34%
SENet with 3 Hidden Layers Plus Dropout Layers	ALL	100.00%	100.00%	100.00%
	AML	70.93%	74.75%	72.79%
	CLL	93.00%	76.09%	83.70%
	CML	72.29%	80.81%	76.31%
	Healthy	99.00%	100.00%	99.50%

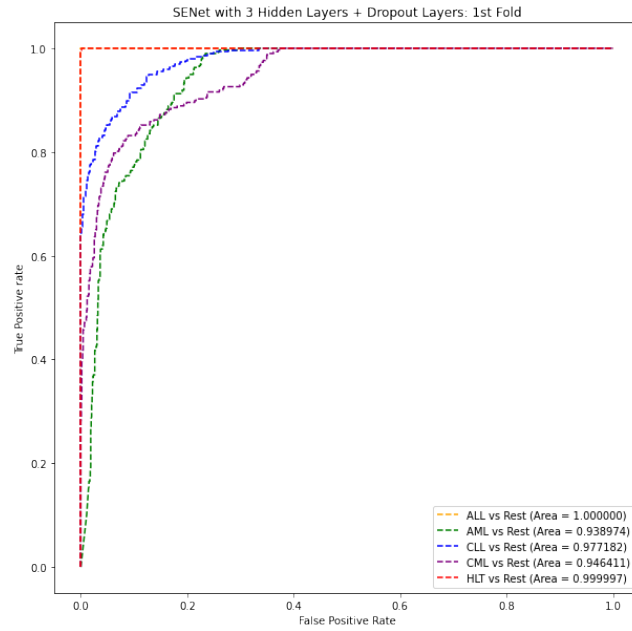


(a)



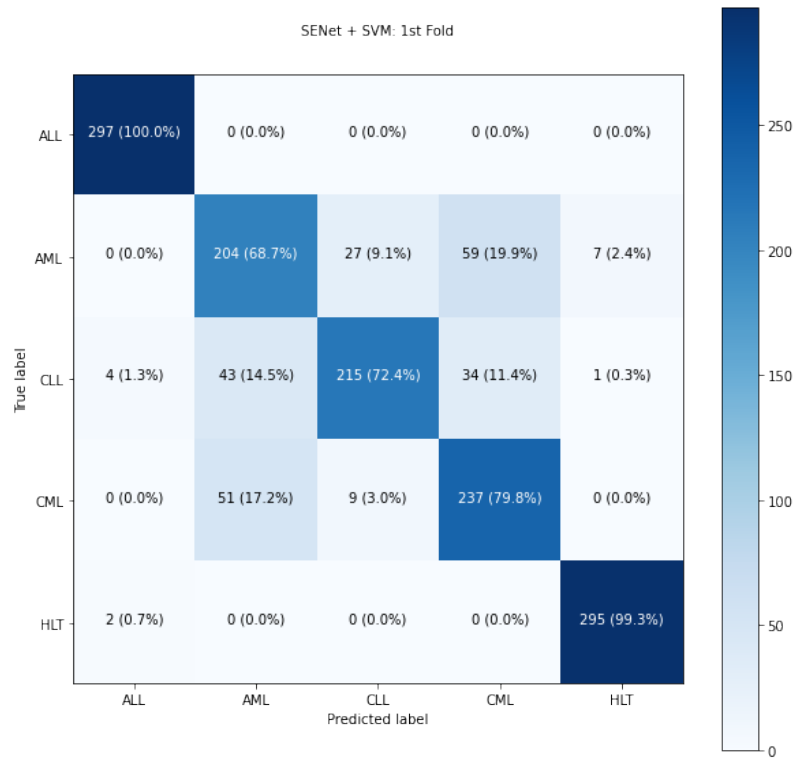
(b)



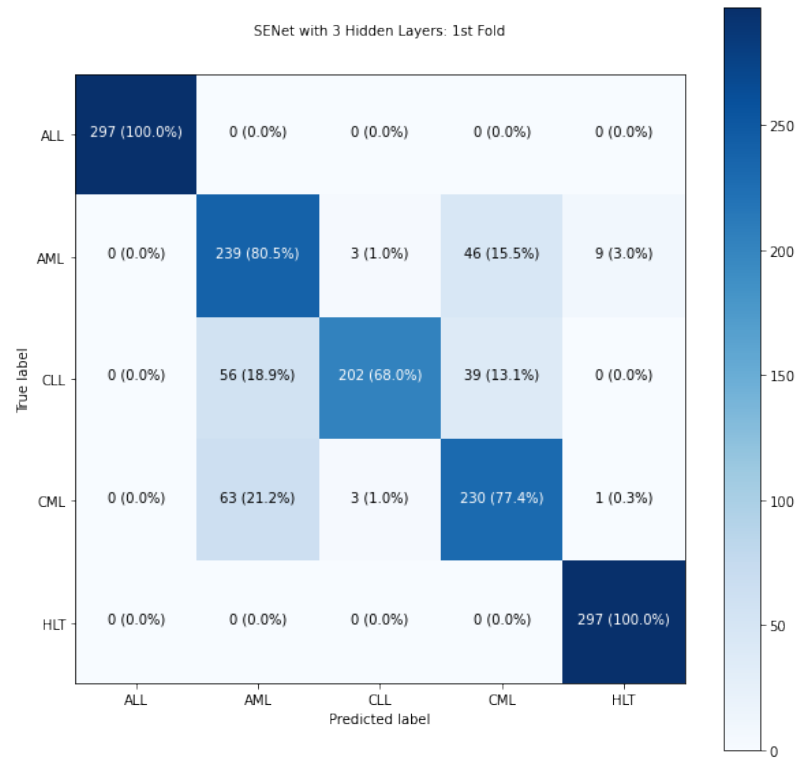


(c)

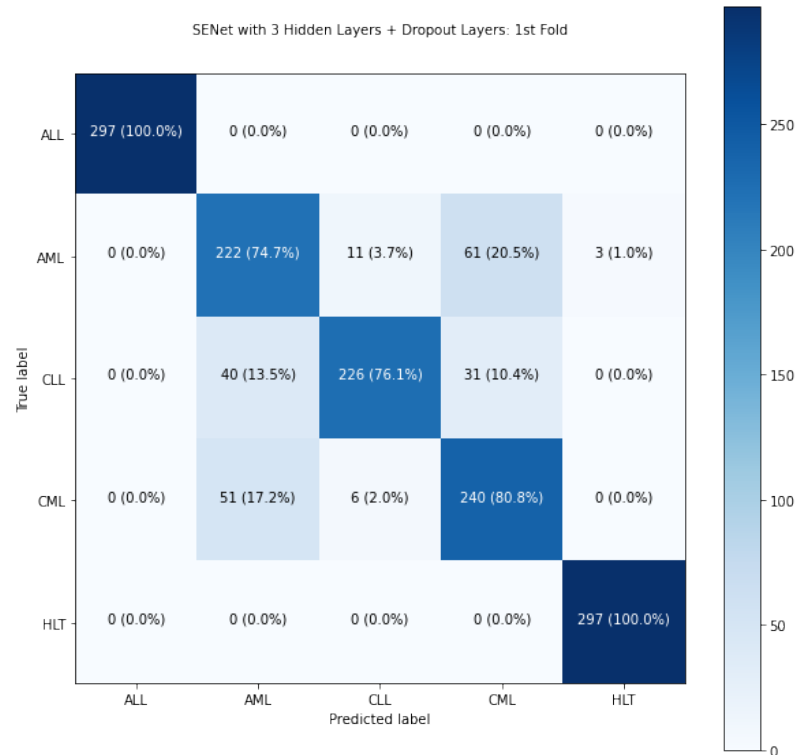
Figure 4.32: ROC Curve in The First Fold Training for (a) SENet + SVM  
 (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout  
 Layers



(a)



(b)



(c)

Figure 4.33: Confusion Matrix in The First Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers

### 4.3.2 Fold 2

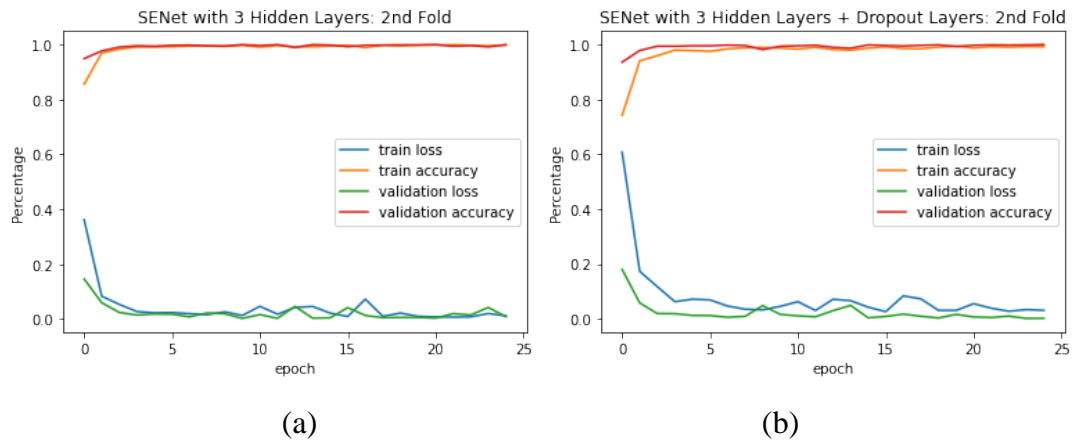
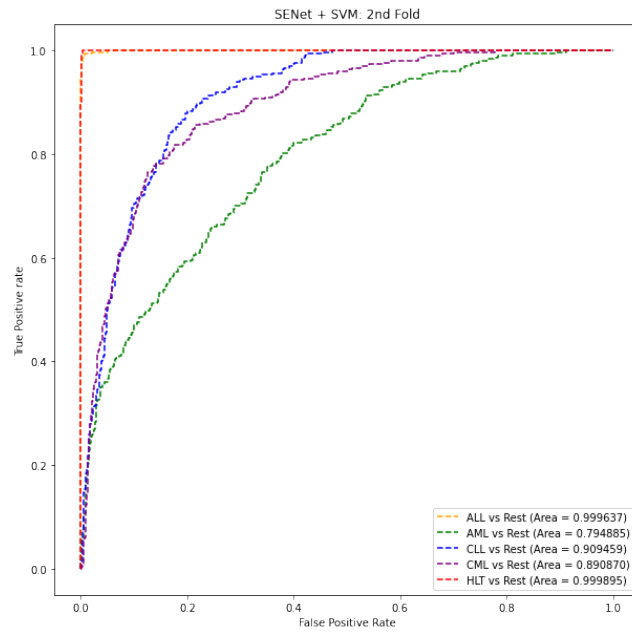


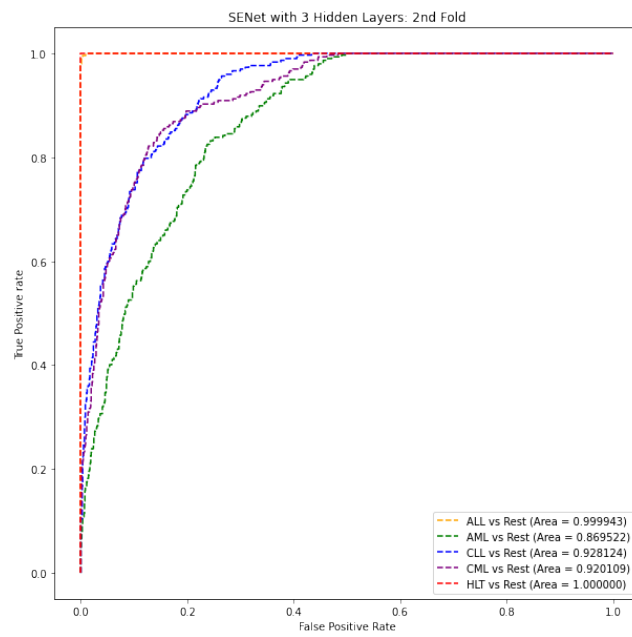
Figure 4.34: Accuracy and Loss Against Number of Epochs in The Second Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.21: Precision, Recall, and F1-score for 5-class Classification Problem in The Second Fold Training for Each Fine-tuned SENet Models**

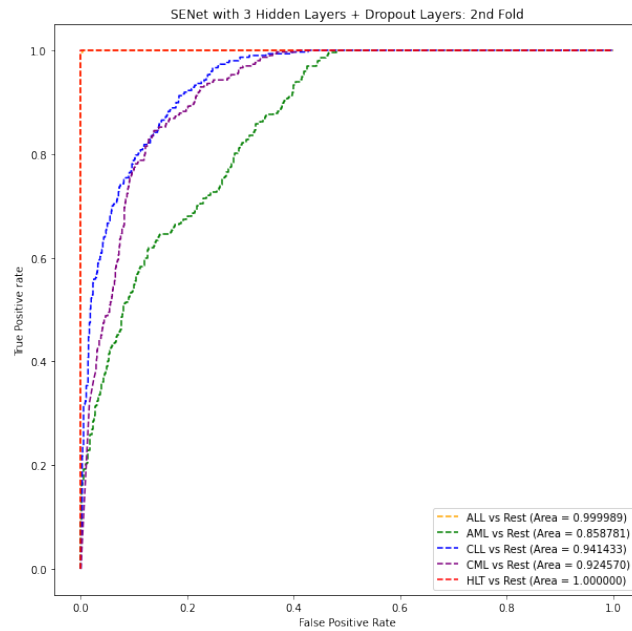
Model	Classes	Precision	Recall	F1-score
SENet + SVM	ALL	98.64%	97.98%	98.31%
	AML	49.50%	50.51%	50.00%
	CLL	64.55%	64.98%	64.77%
	CML	64.60%	63.30%	63.95%
	Healthy	99.33%	99.33%	99.33%
SENet with 3 Hidden Layers	ALL	98.99%	99.33%	99.16%
	AML	57.04%	55.89%	56.46%
	CLL	73.33%	62.96%	67.75%
	CML	64.91%	74.75%	69.48%
	Healthy	99.33%	100.00%	99.66%
SENet with 3 Hidden Layers Plus Dropout Layers	ALL	99.00%	99.66%	99.33%
	AML	61.16%	46.13%	52.59%
	CLL	73.84%	69.36%	71.53%
	CML	62.34%	80.81%	70.38%
	Healthy	99.66%	100.00%	99.83%



(a)

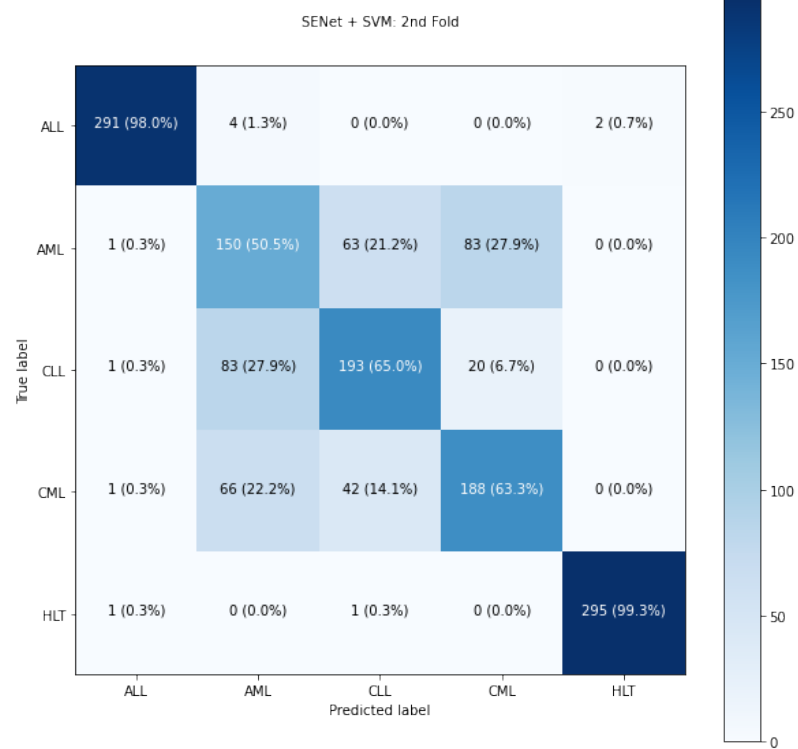


(b)

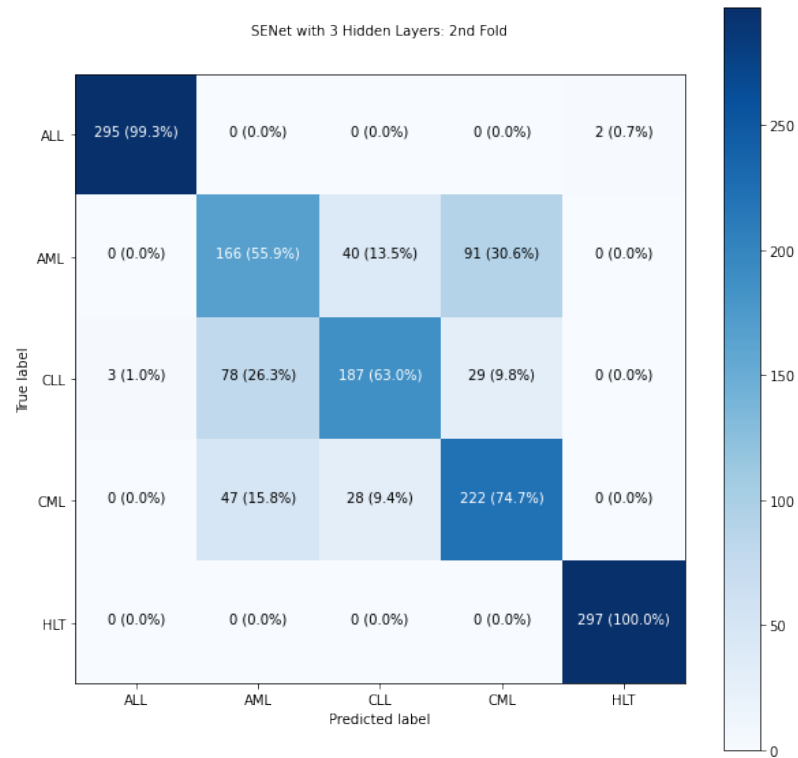


(c)

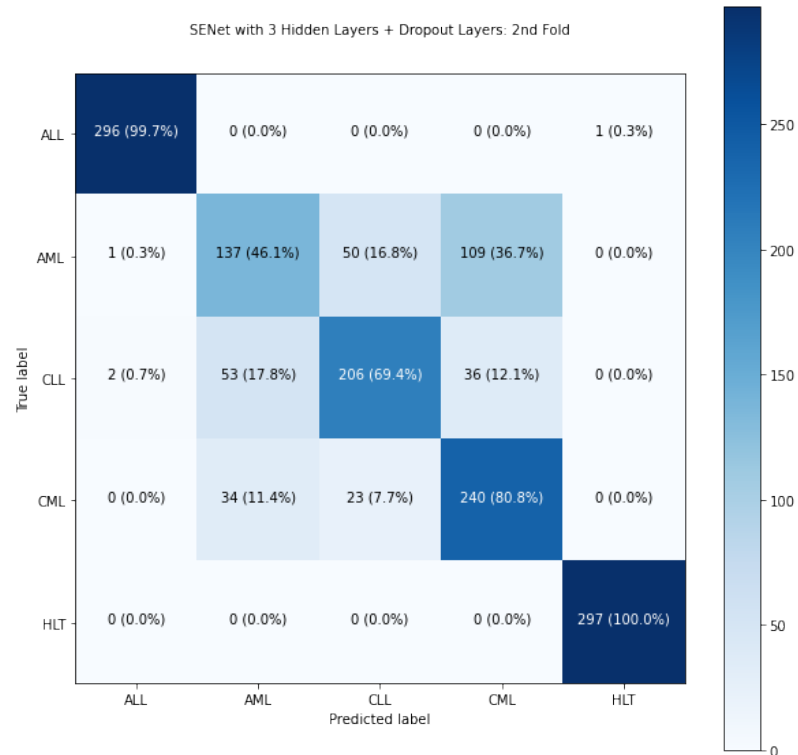
Figure 4.35: ROC Curve in The Second Fold Training for (a) SENet + SVM  
 (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout  
 Layers



(a)



(b)



(c)

Figure 4.36: Confusion Matrix in The Second Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers

## 4.3.3 Fold 3

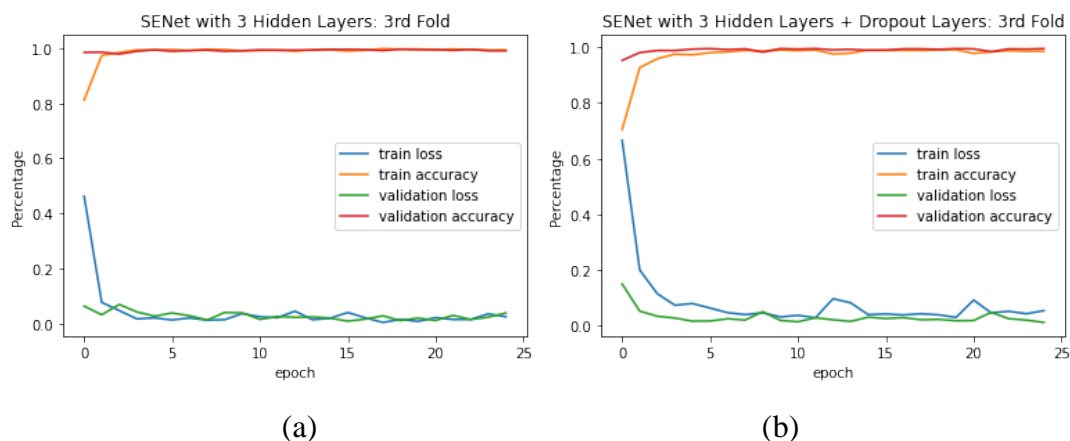
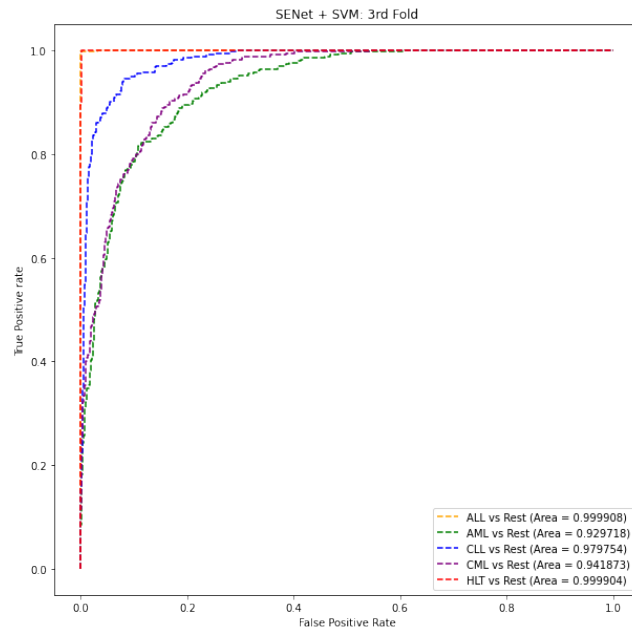


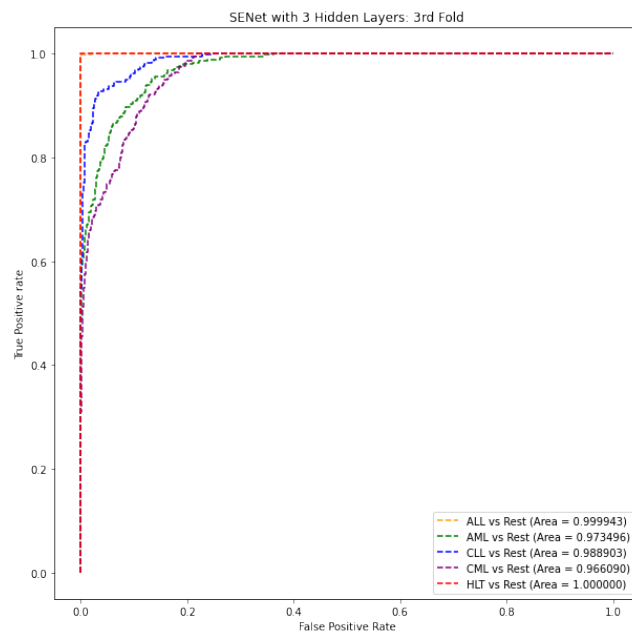
Figure 4.37: Accuracy and Loss Against Number of Epochs in The Third Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.22: Precision, Recall, and F1-score for 5-class Classification Problem in The Third Fold Training for Each Fine-tuned SENet Models**

Model	Classes	Precision	Recall	F1-score
<b>SENet + SVM</b>	<b>ALL</b>	100.00%	99.39%	99.70%
	<b>AML</b>	66.84%	78.18%	72.07%
	<b>CLL</b>	90.82%	80.91%	85.58%
	<b>CML</b>	74.03%	69.09%	71.47%
	<b>Healthy</b>	98.80%	100.00%	99.40%
<b>SENet with 3 Hidden Layers</b>	<b>ALL</b>	99.70%	99.39%	99.54%
	<b>AML</b>	65.52%	92.73%	76.79%
	<b>CLL</b>	96.86%	74.85%	84.44%
	<b>CML</b>	85.98%	68.79%	76.43%
	<b>Healthy</b>	98.51%	100.00%	99.25%
<b>SENet with 3 Hidden Layers Plus Dropout Layers</b>	<b>ALL</b>	100.00%	100.00%	100.00%
	<b>AML</b>	70.13%	81.82%	75.52%
	<b>CLL</b>	85.82%	73.33%	79.08%
	<b>CML</b>	79.75%	77.58%	78.65%
	<b>Healthy</b>	99.40%	100.00%	99.70%

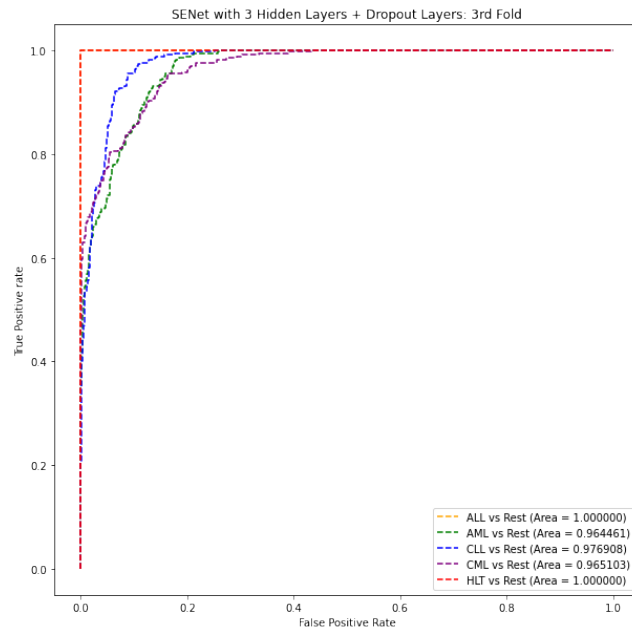


(a)



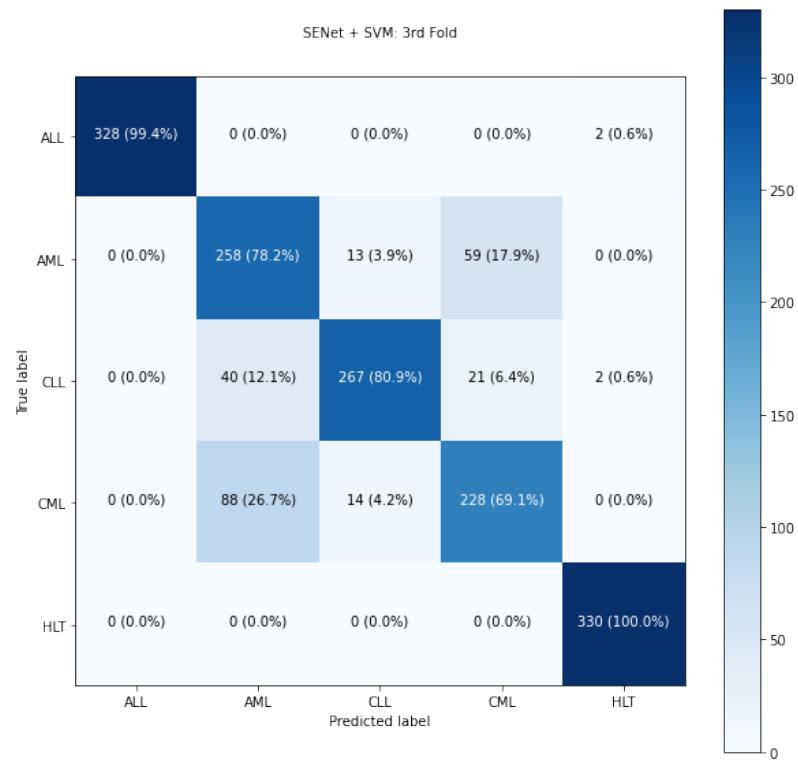
(b)



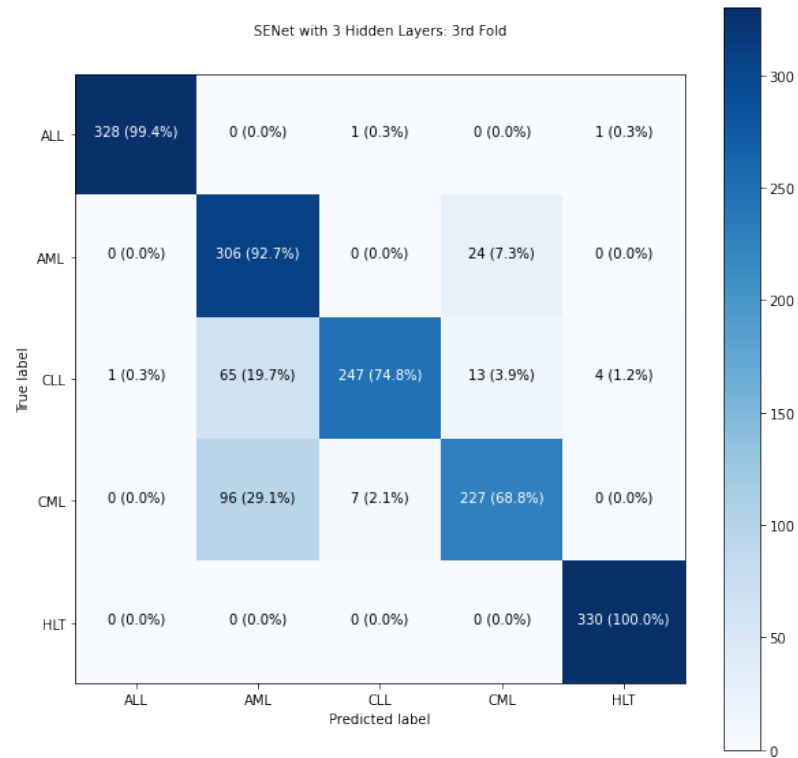


(c)

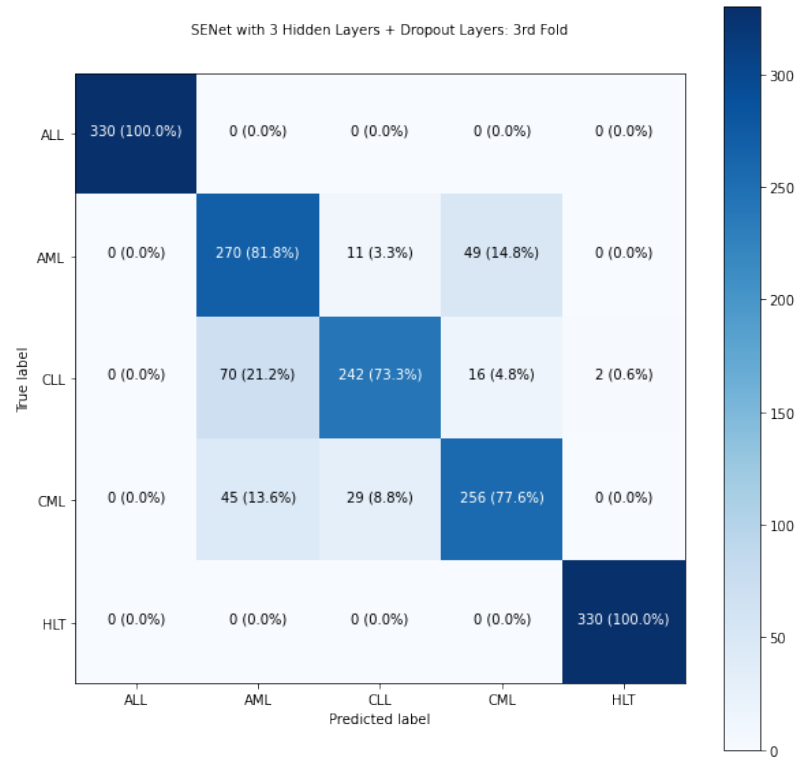
Figure 4.38: ROC Curve in The Third Fold Training for (a) SENet + SVM  
 (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout  
 Layers



(a)



(b)



(c)

Figure 4.39: Confusion Matrix in The Third Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers

## 4.3.4 Fold 4

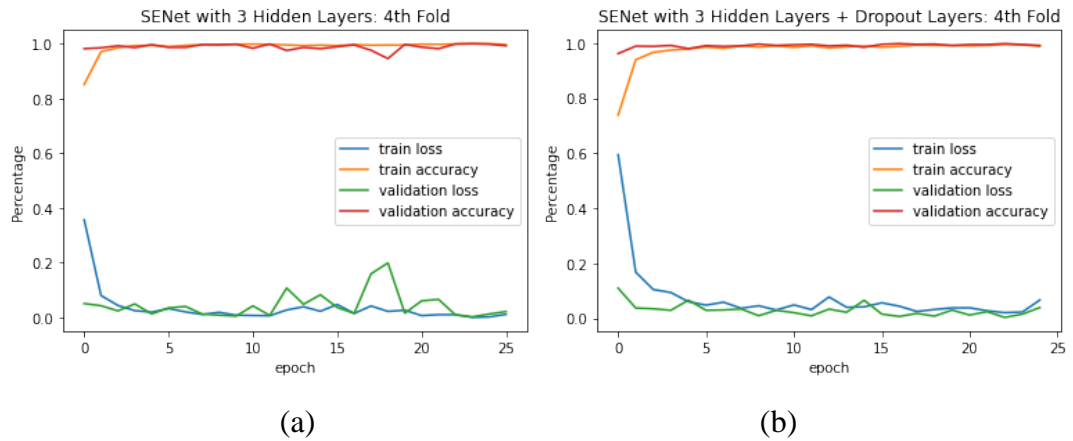
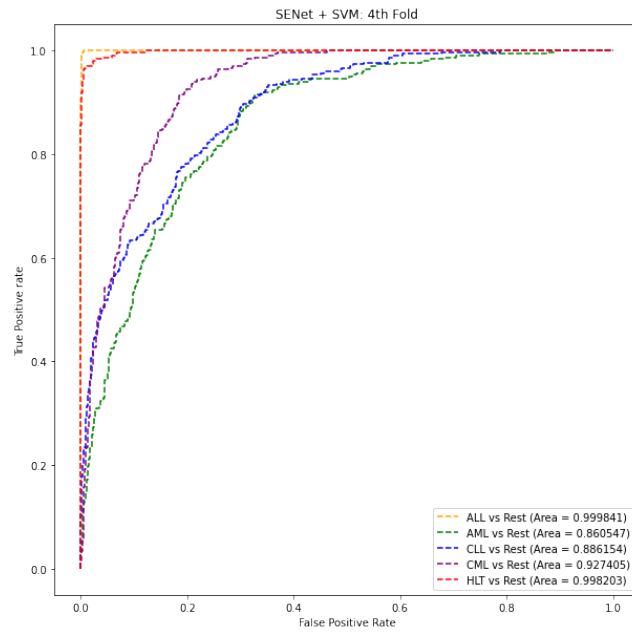


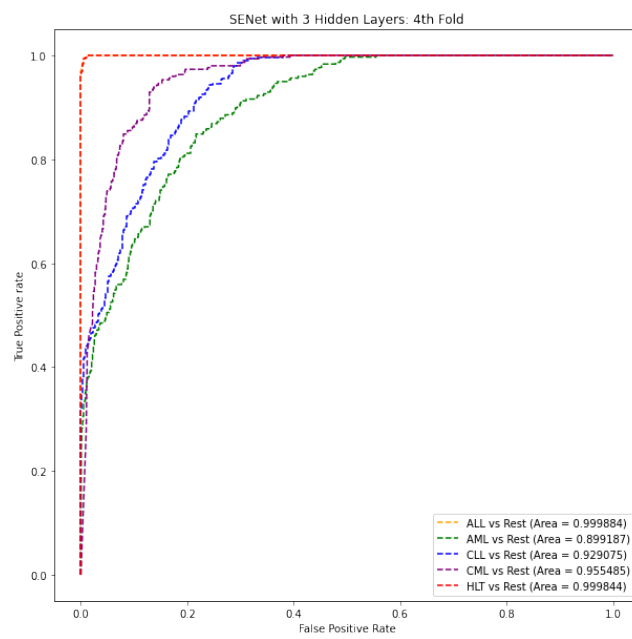
Figure 4.40: Accuracy and Loss Against Number of Epochs in The Fourth Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.23: Precision, Recall, and F1-score for 5-class Classification Problem in The Fourth Fold Training for Each Fine-tuned SENet Models**

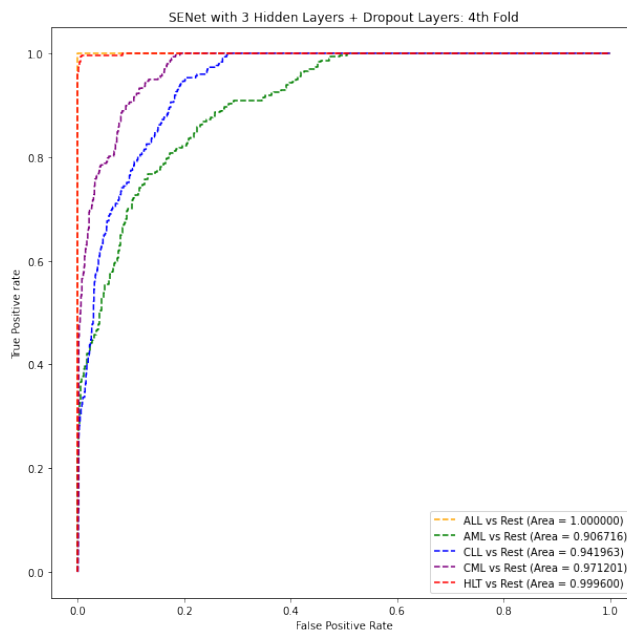
Model	Classes	Precision	Recall	F1-score
SENet + SVM	ALL	94.89%	100.00%	97.38%
	AML	58.09%	47.14%	52.04%
	CLL	67.05%	58.25%	62.34%
	CML	59.35%	80.13%	68.19%
	Healthy	98.90%	90.57%	94.55%
SENet with 3 Hidden Layers	ALL	98.01%	99.33%	98.66%
	AML	68.70%	53.20%	59.96%
	CLL	73.04%	56.57%	63.76%
	CML	63.04%	93.60%	75.34%
	Healthy	100.00%	95.29%	97.59%
SENet with 3 Hidden Layers Plus Dropout Layers	ALL	100.00%	100.00%	100.00%
	AML	69.83%	56.90%	62.71%
	CLL	70.30%	71.72%	71.00%
	CML	73.35%	86.20%	79.26%
	Healthy	98.64%	97.64%	98.14%



(a)

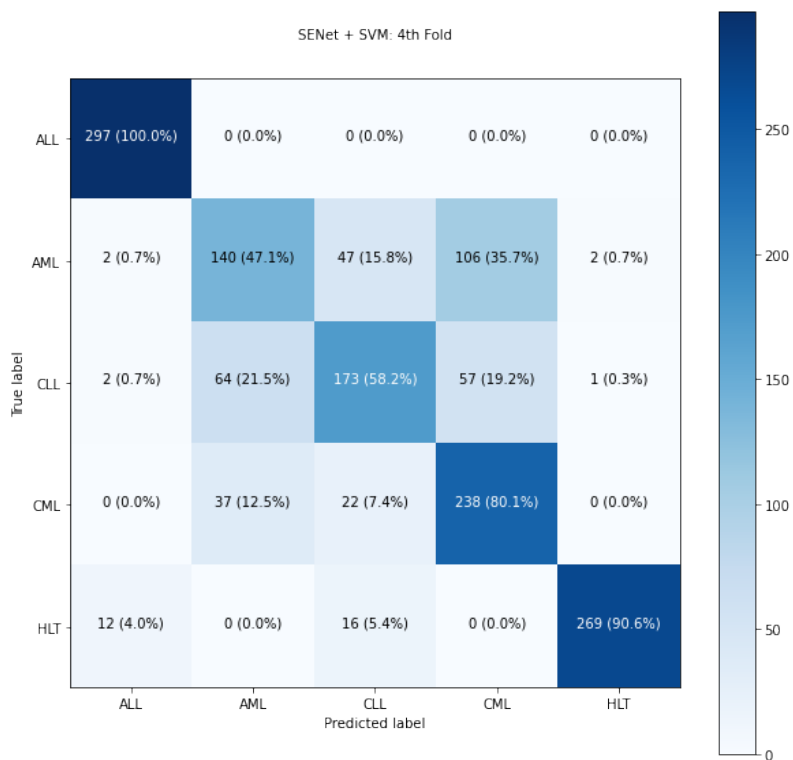


(b)

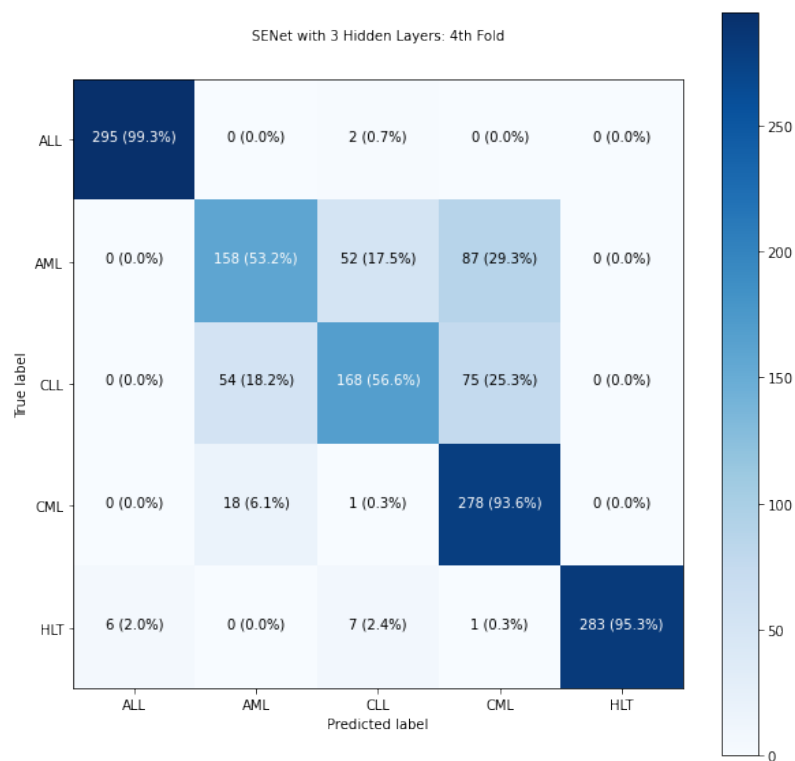


(c)

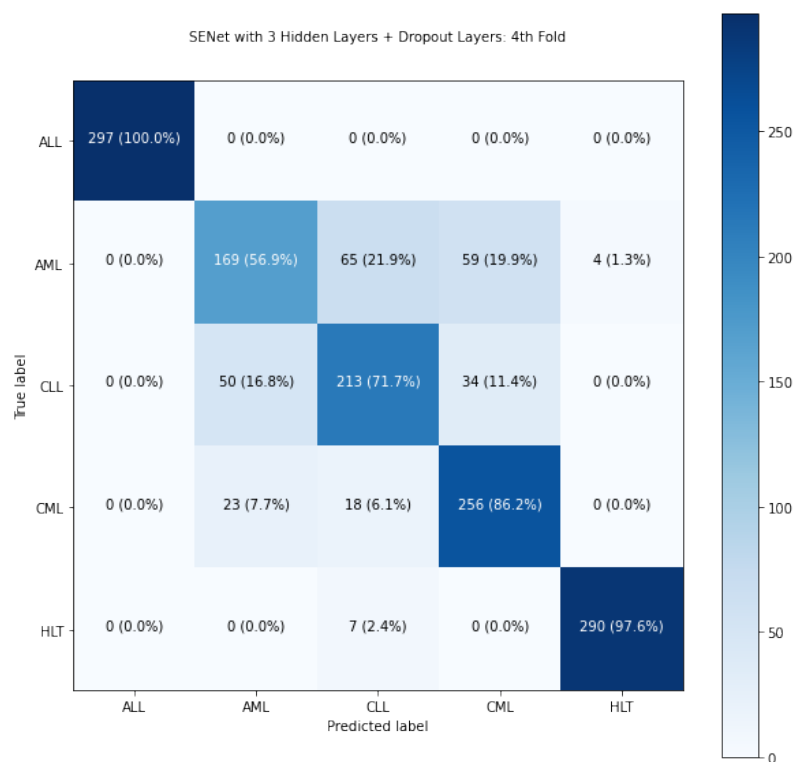
Figure 4.41: ROC Curve in The Fourth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers



(a)



(b)



(c)

Figure 4.42: Confusion Matrix in The Fourth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers

## 4.3.5 Fold 5

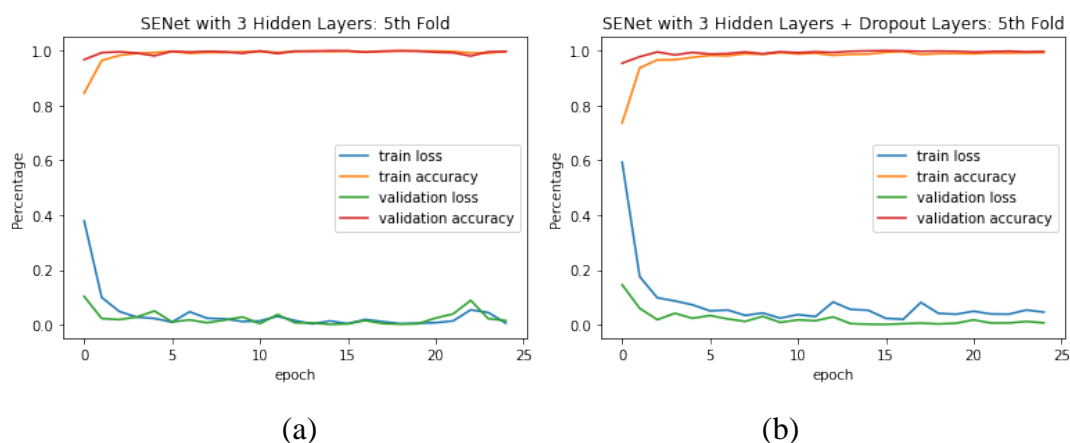
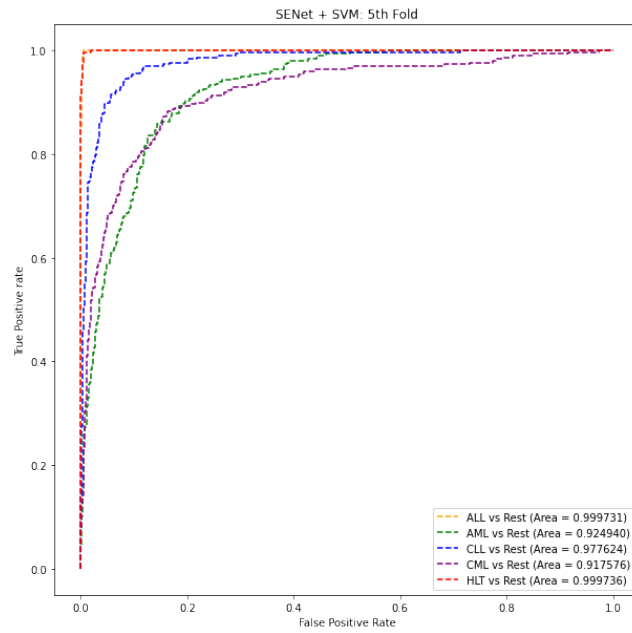


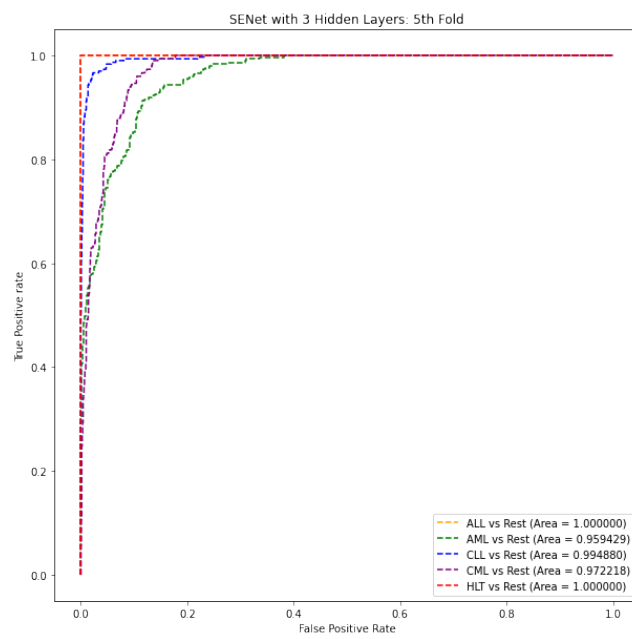
Figure 4.43: Accuracy and Loss Against Number of Epochs in The Fifth Fold Training for (a) SENet with 3 Hidden Layers (b) SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.24: Precision, Recall, and F1-score for 5-class Classification Problem in The Fifth Fold Training for Each Fine-tuned SENet Models**

Model	Classes	Precision	Recall	F1-score
SENet + SVM	ALL	97.06%	100.00%	98.51%
	AML	76.67%	54.21%	63.51%
	CLL	80.91%	89.90%	85.17%
	CML	66.67%	78.11%	71.94%
	Healthy	98.97%	96.97%	97.96%
SENet with 3 Hidden Layers	ALL	99.33%	100.00%	99.66%
	AML	96.99%	43.43%	60.00%
	CLL	80.05%	98.65%	88.39%
	CML	71.54%	93.94%	81.22%
	Healthy	100.00%	100.00%	100.00%
SENet with 3 Hidden Layers Plus Dropout Layers	ALL	98.67%	100.00%	99.33%
	AML	90.17%	52.53%	66.38%
	CLL	83.00%	96.97%	89.44%
	CML	72.53%	88.89%	79.88%
	Healthy	99.00%	100.00%	99.50%

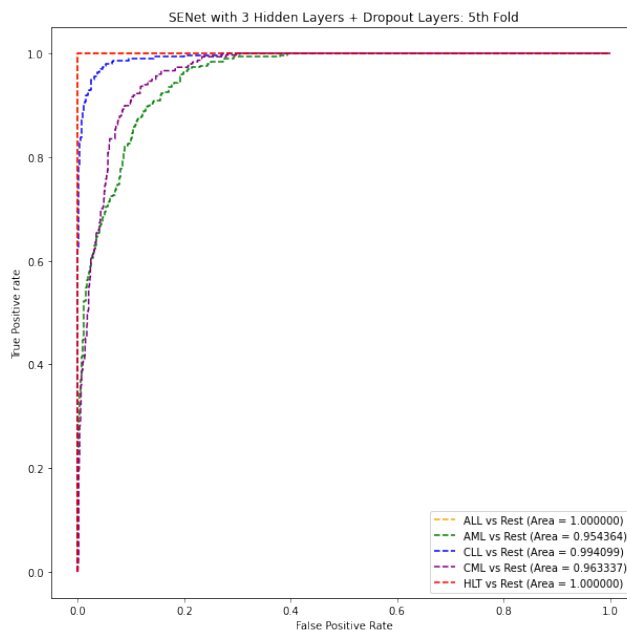


(a)



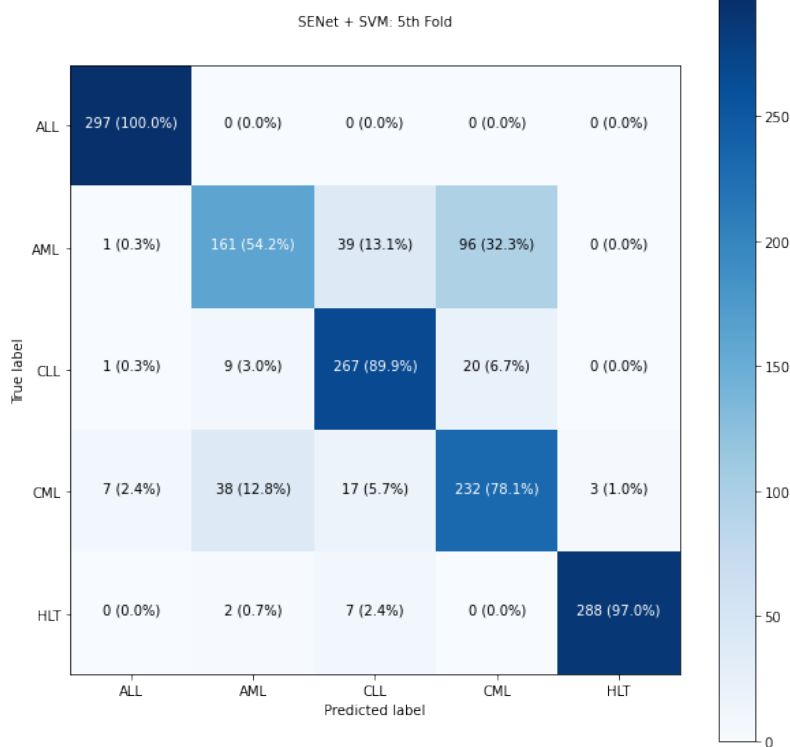
(b)



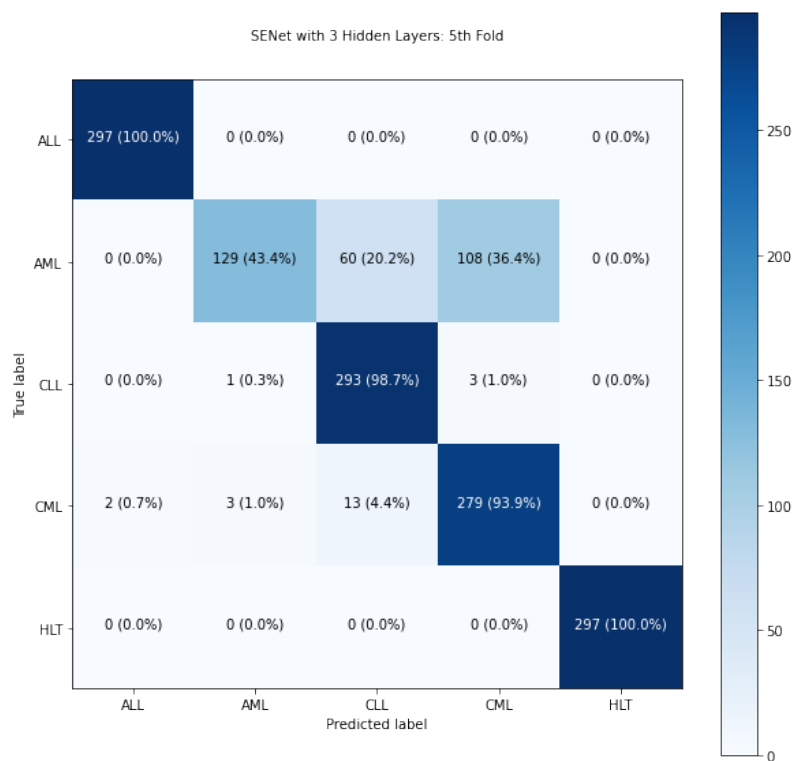


(c)

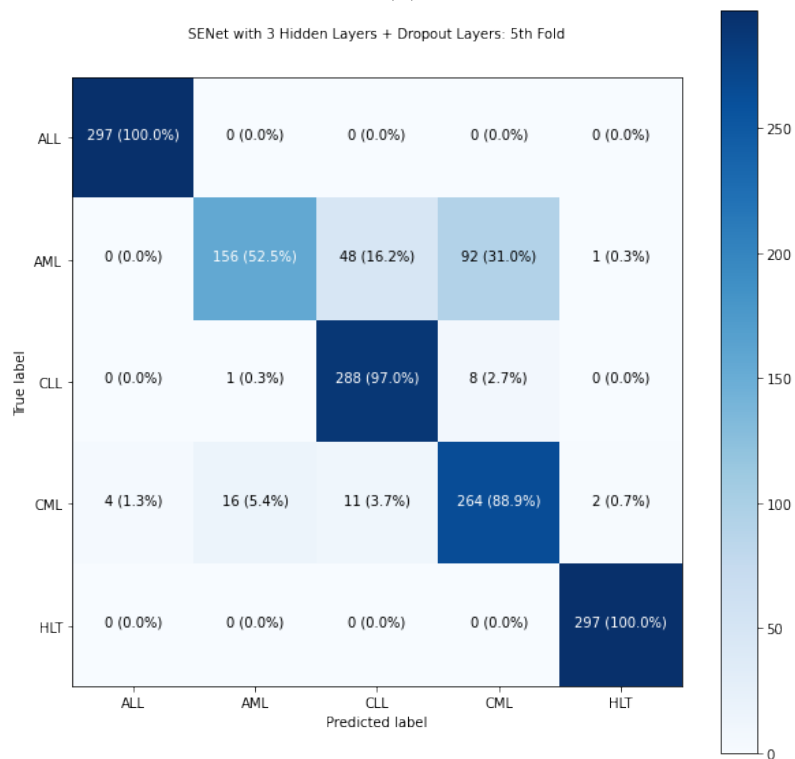
Figure 4.44: ROC Curve in The Fifth Fold Training for (a) SENet + SVM  
 (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout  
 Layers



(a)



(b)



(c)

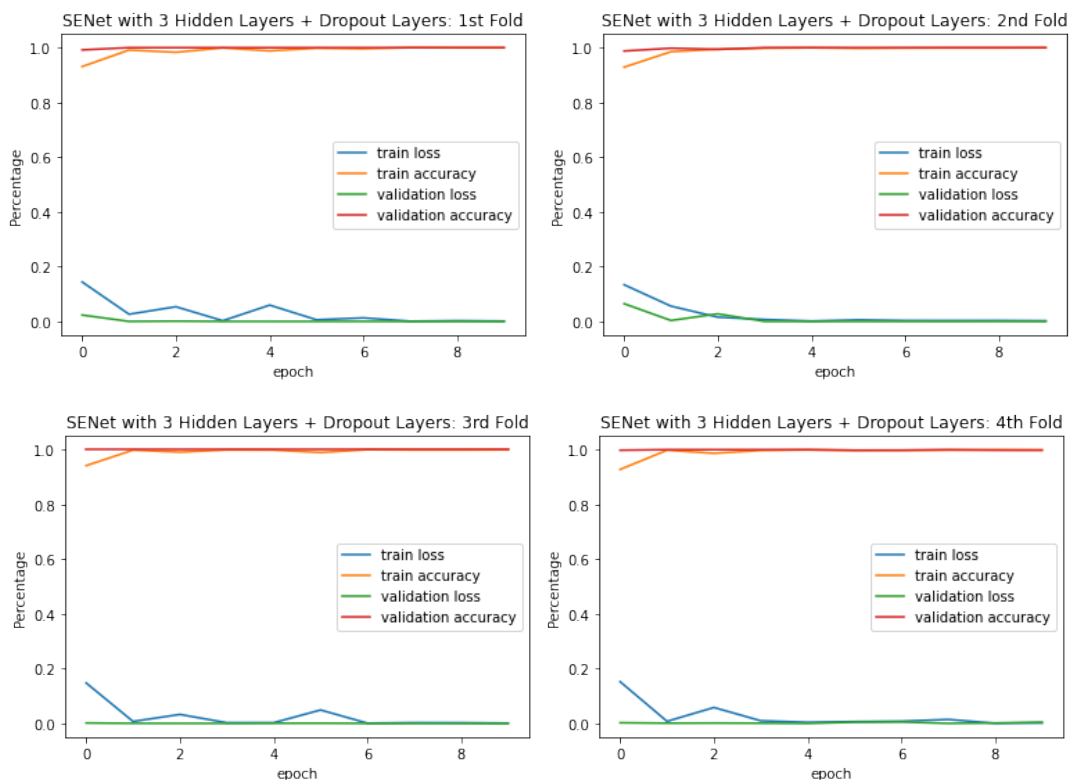
Figure 4.45: Confusion Matrix in The Fifth Fold Training for (a) SENet + SVM (b) SENet with 3 Hidden Layers (c) SENet with 3 Hidden Layers Plus Dropout Layers

#### 4.4 Binary Classification Problem for SENet with 3 Hidden Layers Plus Dropout Layers

**Table 4.25: Accuracy and Loss Result for Binary Classification Problem of Each Fold**

Metrics	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
test_acc	100.00%	100.00%	99.55%	100.00%	99.66%	99.84%
val_acc	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
trn_acc	99.13%	99.80%	94.16%	99.80%	99.80%	98.54%
test_loss	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
val_loss	0.0000	0.0000	0.0016	0.0009	0.0022	0.0009
trn_loss	0.0262	0.0070	0.1477	0.0076	0.0091	0.0395

##### 4.4.1 Accuracy and Loss Against Number of Epochs



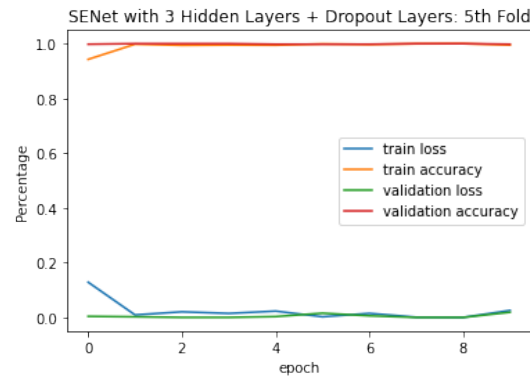


Figure 4.46: Accuracy and Loss Against Number of Epochs for Each Fold

#### 4.4.2 Precision, Recall, and F1-score for Leukemia Subtypes Classification

**Table 4.26: Precision, Recall, and F1-score for Binary Classification Problem for Each Fold**

<b>Fold</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>1</b>	100.00%	100.00%	100.00%
<b>2</b>	100.00%	100.00%	100.00%
<b>3</b>	99.40%	99.70%	99.55%
<b>4</b>	100.00%	100.00%	100.00%
<b>5</b>	100.00%	99.33%	99.66%

#### Average Precision, Recall, and F1-score for Binary Classification Problem

<b>ALL Against Healthy Class</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
	99.88%	99.81%	99.84%

### 4.4.3 ROC Curve

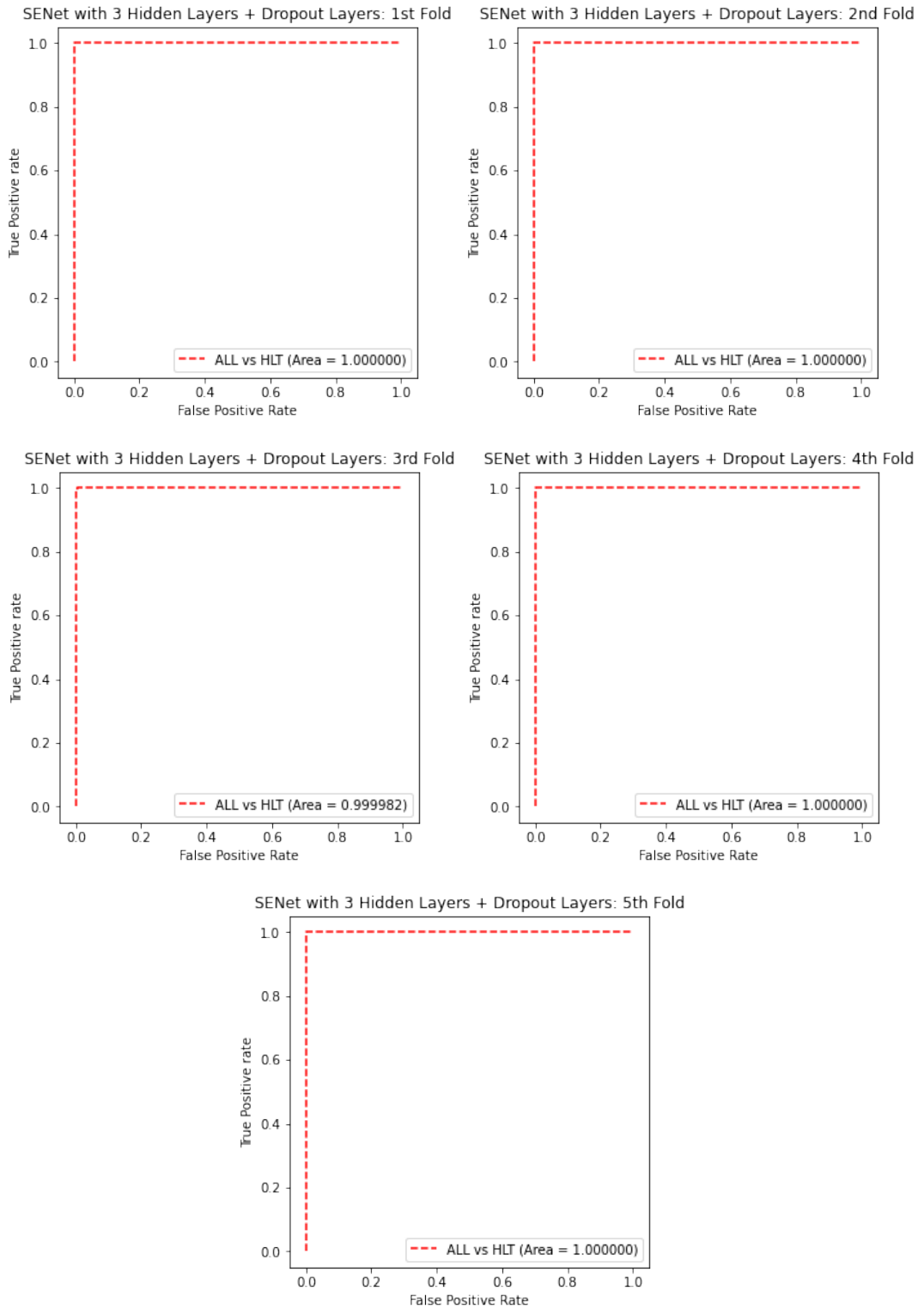


Figure 4.47: ROC Curve for Each Fold

#### 4.4.4 Confusion Matrix

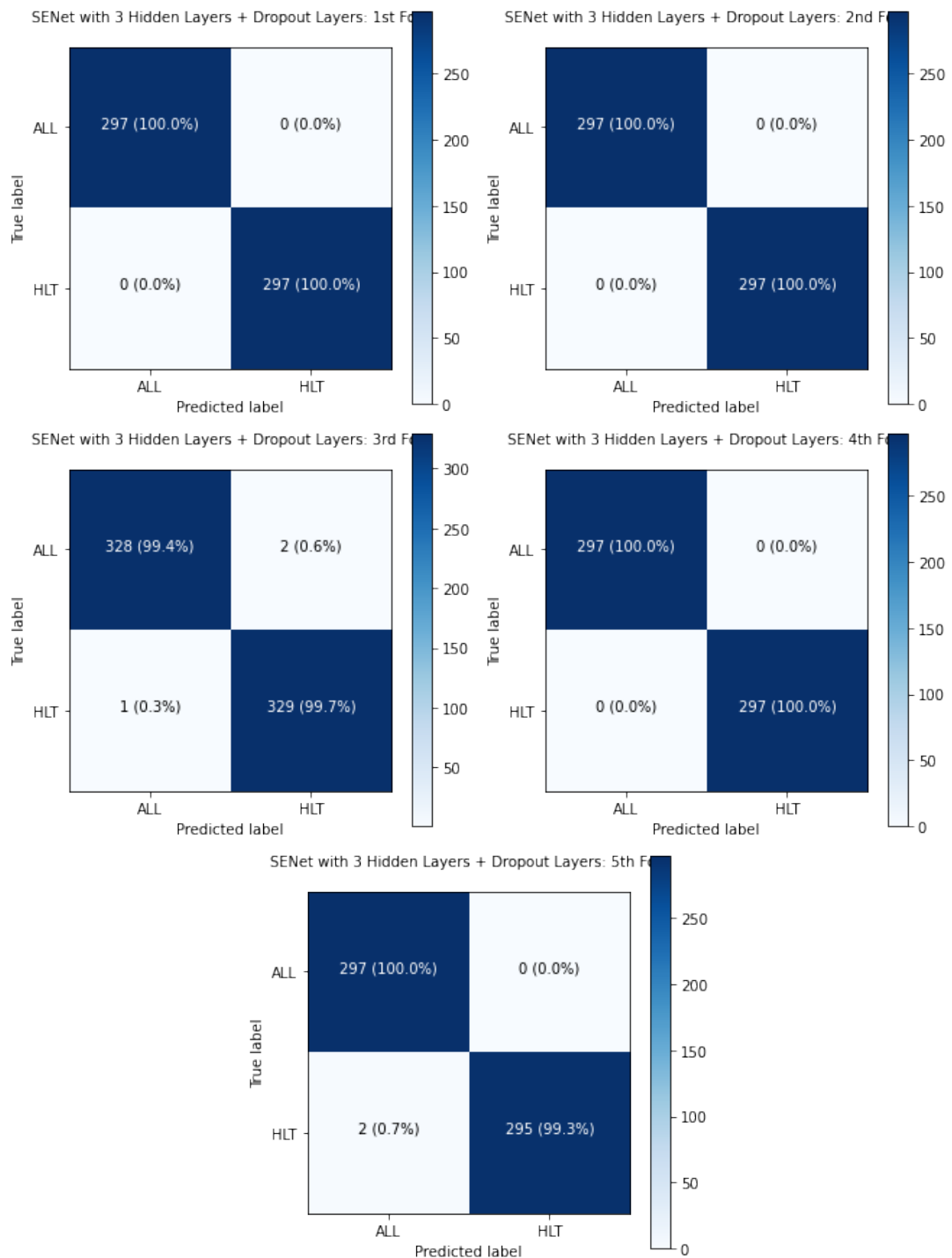


Figure 4.48: Confusion Matrix in The Fourth Fold Training for Each Fold

## 4.5 Discussion

For this project, three models are selected based on their performance in the ILSVRC competition (ImageNet, n.d.), namely Inception-V3, ResNeXt, and SENet. The method of using deep learning to detect and classify leukemia is through transfer learning. With that said, the pretrained models will first be downloaded from the Internet. Then their fully connected layer is replaced with a shallow network that consists of one hidden layer and an output prediction layer. After that, the models will be trained with the images of leukemia. The models will first be trained for the binary classification problem between ALL and healthy classes, followed by the 5-class classification problem between ALL, AML, CLL, CML, and healthy classes. The overall architecture and hyperparameters of the models used to tackle the binary and 5-class classification problems are illustrated as shown below.

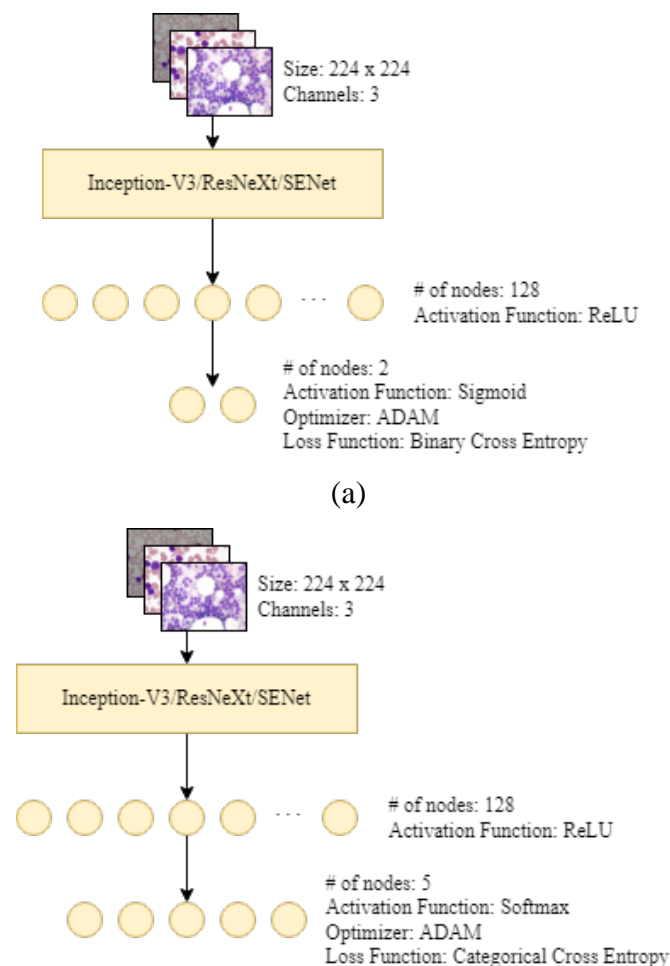


Figure 4.49: Overall Architecture of Each Model for (a) Binary Classification Problem (b) 5-class Classification Problem

During the models' training and testing, a method known as k-fold cross-validation is implemented, in which k of 5 is used for this project. This means that the dataset will be split into data groups consisting of 80% training set and 20% testing set.

For the binary classification problem, the accuracies and losses of the models after training and testing are tabulated as shown in Tables 4.1, 4.2, and 4.3. As observed, all of the models were able to obtain a high average training and validation accuracy at approximately 99.38% and 99.86%, respectively. However, during testing, the SENet model generated the highest average accuracy, which is 99.87%, followed by ResNeXt, which is 97.66%, and Inception-V3, which is 97.03%. Among the folds trained, the SENet model achieved a 100.00% testing accuracy in the first and the fourth fold. The SENet model also holds the best score compared to the other literature studies. It is followed closely by the model proposed by Shafique, et al. (2018), in which they achieved an average testing accuracy of 99.50%. The bar chart in Figure 4.50 compares the proposed models in this project to the other literature studies on the binary classification problem.

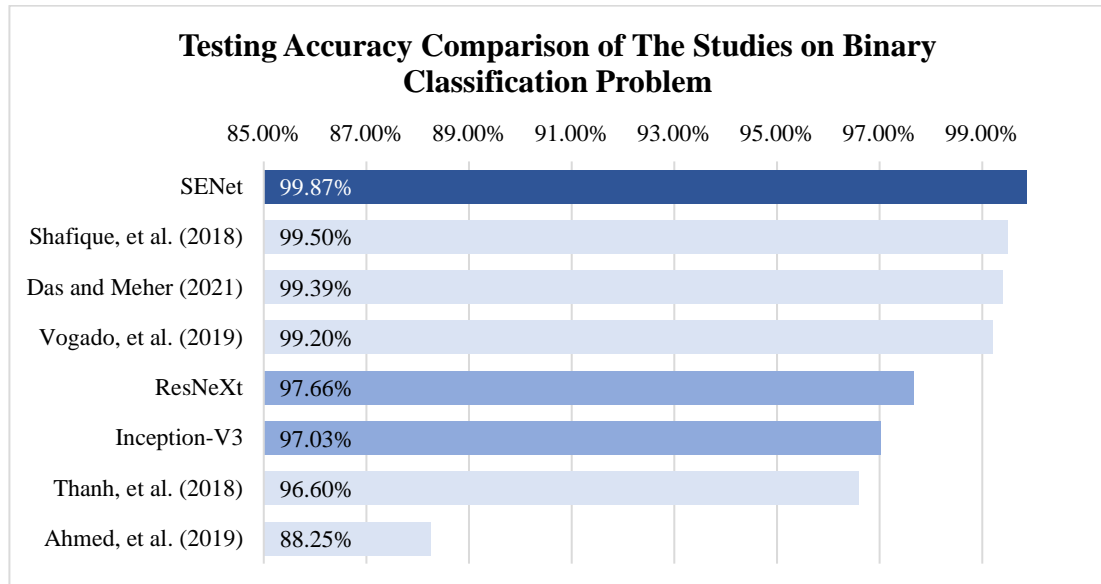


Figure 4.50: Bar Chart of The Testing Accuracy Comparison of The Studies on Binary Classification Problem

In terms of the precision, recall, and F1-score, the SENet model exhibited the best score with 100.00% on all three metrics in the first fold compared to the other models. However, in the second fold, it is observed that the ResNeXt model exhibited



the best score of 100.00% on all three metrics compared to the other models. In the third fold, the ResNext model has the highest precision, while the SENet model has the highest recall. Nevertheless, the SENet model exhibited a higher F1-score than the ResNeXt model by 0.62%. In the fourth fold, the SENet model once again exhibited the best score of 100.00% on all three metrics compared to the other models. Finally, the SENet model obtained the highest score on all three metrics in the fifth fold with a precision of 100.00%, a recall of 99.66%, and an F1-score of 99.83%. The average precision, average recall, and average F1-score is calculated and tabulated as shown in the table below. The table also includes studies that provided precision, recall, and F1-score of their proposed models on the binary classification problem.

**Table 4.27: Average Precision, Recall, and F1-score for Binary Classification Problem for Each Model**

<b>Models</b>	<b>Precision</b>	<b>Recall</b>	<b>F1-score</b>
<b>Inception-V3</b>	98.69%	95.31%	96.91%
<b>ResNeXt</b>	99.93%	95.39%	97.54%
<b>SENet</b>	99.81%	99.93%	99.87%
<b>Vogado, et al. (2018)</b>	99.20%	99.20%	99.20%
<b>Das and Meher (2021)</b>	99.33%	99.55%	99.44%

Without a doubt, the higher the recall and precision of the model, the better it is. However, it is crucial to avoid models with low recall in the medical field. This is because a model with a low recall may diagnose a patient who actually has leukemia as not having it at all. It will cause the patient to miss its window of opportunity for treatment, leading to undesirable consequences. Nevertheless, the precision of a model should still be taken into account so that patients who do not have leukemia will not be treated for it. All in all, it is better to look at the F1-score, since it provides a better measurement of the performance of the model in terms of precision and recall. The closer the F1-score is to 100.00%, the better the model's performance.

It is observed in Table 4.27 that the SENet has the highest average F1-score compared to other models. This shows that it can correctly classify most of the images into their correct classes, such that images with ALL cells are correctly classified as

ALL class, whereas images with no ALL cells are correctly classified as the healthy class. This can also be justified in the ROC curves, where sharp bents are observed in each fold, and the AUC-ROC values are determined to be at an average value of 1. It shows that the SENet model has an excellent class separability. It is also observed in the confusion matrixes, where the number in the diagonal elements is much higher and has a darker colour than the neighbouring elements. Thus, in the binary classification problem, it is without a doubt that the SENet model provides the best performance rate in detecting leukemia cells from healthy cells.

The 5-class classification problem concerns the detection of leukemia cells and the classification of leukemia subtypes between ALL, AML, CLL, and CML classes. The accuracies and losses of the models after training and testing are tabulated as shown in Tables 4.9, 4.10, and 4.11. It is observed that all of the models were able to obtain a high average training and validation accuracy but were unable to achieve an average testing accuracy close to them. The highest recorded average testing accuracy is 83.03%, achieved by the SENet model. On the other hand, the ResNeXt model achieved an average testing accuracy of 80.78%, while the Inception-V3 model achieved 73.42%. It is important to consider that the SENet model is much more complex than the other two models. Therefore, without a doubt, more important features can be captured and learned from the images, which results in a high prediction score.

Among the folds trained, the SENet model achieved a reasonably high testing accuracy of 87.41% in the fifth fold. Nevertheless, the SENet model still holds the best score compared with other literature studies. It is followed closely by the model proposed by Ahmed, et al. (2019), which achieved an average accuracy of 81.74%. The bar chart in Figure 4.51 compares the proposed models in this project to the other literature studies on the 5-class classification problem. The accuracy obtained by Bibi, et al. (2020) is not included in the bar chart because there is no indication on how the accuracy was obtained. Hence, it is assumed that the accuracy of 99.91% obtained from their studies was the training accuracy, in which all three of the models used in this project were approximately close to it. The highest average training accuracy of 100.00% is obtained by the Inception-V3 model, followed by 99.89% obtained by the SENet model and 99.74% obtained by the ResNeXt model.

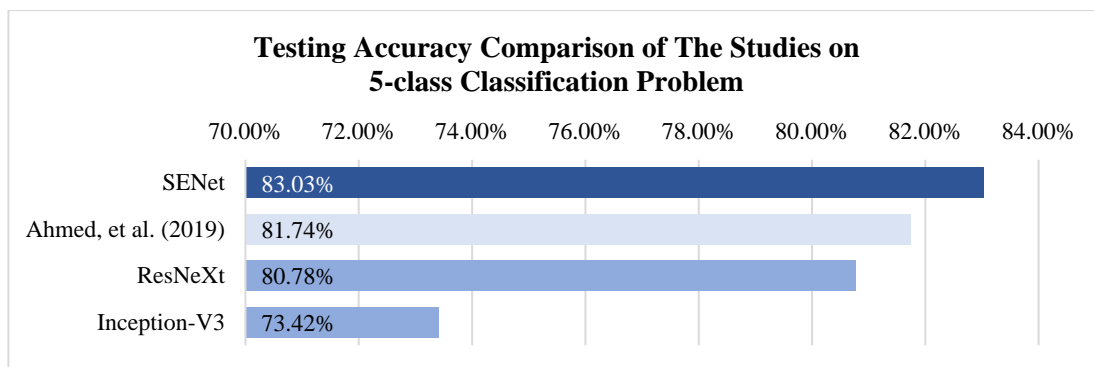


Figure 4.51: Bar Chart of The Testing Accuracy Comparison of The Studies on 5-class Classification Problem

In terms of the precision, recall, and F1-score, the SENet model obtained a higher average score in each class compared to the other models. Note that no other studies had provided the precision, recall, and F1-score of their proposed model on the 5-class classification problem. Nevertheless, the average precision, average recall, and average F1-scores for all models proposed in this project are calculated and tabulated in Table 4.28. The numbers obtained show that most models have a hard time classifying images of AML and CML cells while working moderately well in classifying images of CLL cells, and do not have much problems classifying images of ALL and healthy cells. For ALL class, the SENet model achieved the highest average score for all metrics, whereby precision is 99.28%, recall is 100.00%, and F1-score is 99.64%. As for AML class, the highest average precision of 72.31% is achieved by the ResNeXt model, while the highest average recall and F1-score of 62.43% and 64.45%, respectively, are achieved by the SENet model. In the case of CLL class, the highest average prediction score of 85.01% is achieved by the ResNeXt model, while the highest average recall and F1-score of 76.98% and 78.42%, respectively, are achieved by the SENet model. For CML class, the highest average precision of 70.88% is achieved by the ResNeXt model, while the highest average recall and F1-score of 75.80% and 71.85%, respectively, are achieved by the SENet model. Lastly, for healthy class, the SENet model achieved the highest average score for all metrics, whereby precision is 98.91%, recall is 99.93%, and F1-score is 99.41%. Although the ResNeXt model exhibited a good performance as observed in its precision, the SENet model still has a better performance as observed in its recall and F1-score for all classes. This finding will be justified by looking at the ROC curves, AUC-ROC values, and confusion matrixes generated by the models.

**Table 4.28: Average Precision, Recall, and F1-score for 5-class Classification Problem for Each Model**

Model	Classes	Precision	Recall	F1-score
<b>Inception-V3</b>	<b>ALL</b>	92.30%	98.26%	95.11%
	<b>AML</b>	54.20%	52.57%	52.55%
	<b>CLL</b>	69.04%	62.26%	64.54%
	<b>CML</b>	55.12%	57.51%	56.06%
	<b>Healthy</b>	97.19%	96.70%	96.78%
<b>ResNeXt</b>	<b>ALL</b>	94.96%	99.73%	97.22%
	<b>AML</b>	72.31%	62.20%	61.86%
	<b>CLL</b>	85.01%	72.32%	76.62%
	<b>CML</b>	70.88%	73.70%	69.13%
	<b>Healthy</b>	98.78%	95.97%	97.08%
<b>SENet</b>	<b>ALL</b>	99.28%	100.00%	99.64%
	<b>AML</b>	68.58%	62.43%	64.45%
	<b>CLL</b>	81.06%	76.98%	78.42%
	<b>CML</b>	69.28%	75.80%	71.85%
	<b>Healthy</b>	98.91%	99.93%	99.41%

By only observing the ROC curve in the fold where most of the models were able to achieve the highest testing accuracy, which is the fifth fold, it is evident that the SENet model has sharper bents on most of the classes compared to the ResNeXt model and the Inception-V3 model. On top of that, the AUC-ROC values of the SENet model for ALL class against the rest is 1, AML class against the rest is 0.9342, CLL class against the rest is 0.9921, CML class against the rest is 0.9588, and healthy class against the rest is 1. This shows that the SENet model has a stronger class separability for ALL and healthy classes while the weakest for AML class. Besides, the class separability can be observed in the confusion matrixes, whereby the higher the number and the darker the colour of the diagonal elements is, the better the model is in classifying the images into their correct classes. The confusion matrixes of the Inception-V3 model shows that the model exhibited good performance in classifying images of ALL and healthy cells but becomes weaker when it comes to classifying images of AML, CLL, and CML cells. As for the ResNeXt model, it is observed that

there is an improvement in classifying images of CLL class compared to the Inception-V3 model. Lastly, the SENet model is seen to be able to classify images of AML, CLL, CML, and healthy cells better, but it lacks behind when classifying images of AML cells compared to the ResNeXt model. Thus, in the 5-class classification problem, it is concluded that the SENet model provides the best performance rate in detecting leukemia cells from healthy cells and classifying the leukemia cells into their subtypes.

After evaluating the models, it is without a doubt that the SENet model is the clear winner among the three models. Henceforth, the SENet model will be fine-tuned to improve its accuracy and overall performance further. The 5-class classification problem will be tackled first, and when a reasonable improvement in the results is observed, the model will be implemented on the binary classification problem. The goal of fine-tuning is to increase the testing accuracy for the 5-classification task as much as possible by making two different adjustments to the model, which include replacing the model's classifier with a machine learning algorithm or increasing the layers in the fully connected network with different dimensional feature vectors.

The first idea of using machine learning algorithms as a classifier comes from the literature studies done by Vogado, et al. (2018). Their studies focus on using CNN models as the feature extractor and machine learning algorithms, such as the SVM, as the classifier. Their proposed model was used to tackle the same binary classification problem in this project which is leukemia detection. It is observed that they can achieve reasonably high accuracy, but the method was not applied to 5-class classification problems. Henceforth, an experiment is conducted to test the technique on the 5-class classification problem by replacing the softmax classifier of the SENet model with an SVM classifier. The overall architecture of the model is shown in Figure 4.52, and its code is listed in Code Listing 31 in Appendix A. Note that there are no training and validation data since only the classifier is being replaced while the previous weights and biases of the SENet model are preserved. The previous SENet model will be denoted as SENet model A, whereas the SENet model with an SVM classifier will be denoted as SENet model B to allow easier referencing in the subsequent discussion.

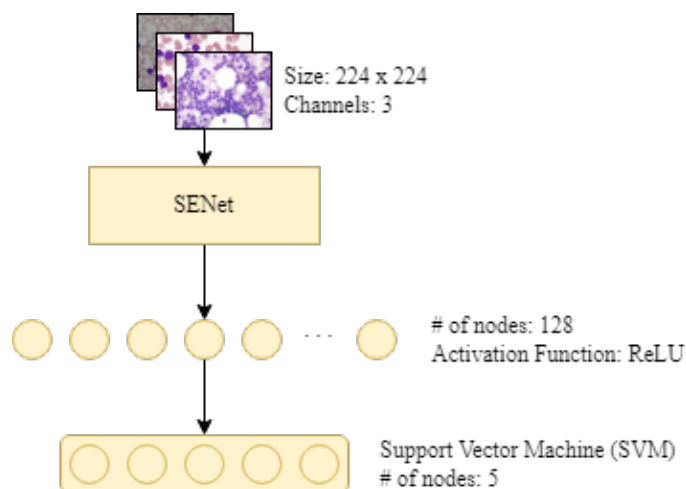


Figure 4.52: Overall Architecture of Each Model for SENet model B

The testing accuracies of each fold and their average value for the SENet model B are tabulated as shown in Table 4.17. Besides, the average precision, recall, and F1-score is calculated and tabulated together with the values for SENet model A as shown in Table 4.29. The bar chart shown in Figure 4.53 compares the average precision, recall, and F1-score between SENet model A and B.

**Table 4.29: Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and B**

Model	Classes	Precision	Recall	F1-score
SENet Model A	ALL	99.28%	100.00%	99.64%
	AML	68.58%	62.43%	64.45%
	CLL	81.06%	76.98%	78.42%
	CML	69.28%	75.80%	71.85%
	Healthy	98.91%	99.93%	99.41%
SENet Model B	ALL	97.72%	99.47%	98.58%
	AML	63.91%	59.75%	61.24%
	CLL	77.80%	73.29%	75.27%
	CML	67.29%	74.09%	70.23%
	Healthy	98.67%	97.24%	97.91%

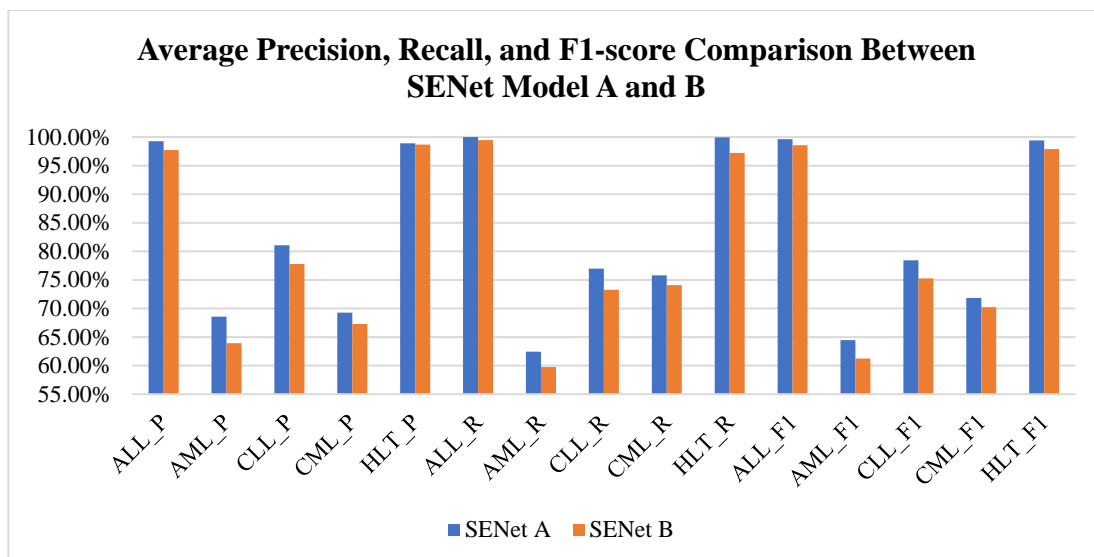


Figure 4.53: Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and B

It is observed that the SENet model B has a reduction in accuracy by 2.26% compared to the SENet model A. In addition, the model is also unable to surpass the average precision, recall, and F1-score for all classes of the SENet model A. This can also be observed in the ROC curves, AUC-ROC values, and the confusion matrixes, whereby the model exhibits a weaker class separability than the SENet model A. All in all, although the use of machine learning algorithm as the classifier can tackle the binary classification problem, it lacks behind when implemented on 5-class classification problems.

Moving on to the second fine-tuning method, more layers consisting of different dimensional feature vectors are added to the fully connected network of the SENet model A to observe whether a slow converging feature selection process can improve the training accuracy. The fully connected network, which initially has 1 hidden layer, is further increased to 3 hidden layers with different dimensional feature vectors. This model will be denoted as SENet model C. The overall architecture of the model is shown in Figure 4.54, and its code is listed in Code Listing 32 in Appendix A.

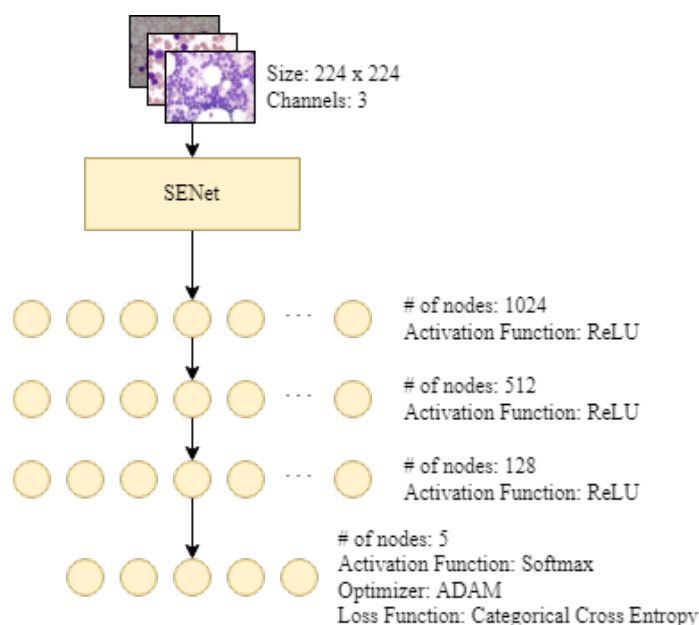


Figure 4.54: Overall Architecture of Each Model for SENet model C

Its accuracies and losses are tabulated as shown in Table 4.18. Furthermore, the average precision, recall, and F1-score is calculated and tabulated together with the values for SENet model A as shown in Table 4.30. The bar chart shown in Figure 4.55 compares the average precision, recall, and F1-score between SENet model A and C.

**Table 4.30: Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and C**

Model	Classes	Precision	Recall	F1-score
SENet Model A	ALL	99.28%	100.00%	99.64%
	AML	68.58%	62.43%	64.45%
	CLL	81.06%	76.98%	78.42%
	CML	69.28%	75.80%	71.85%
	Healthy	98.91%	99.93%	99.41%
SENet Model C	ALL	99.21%	99.61%	99.40%
	AML	71.00%	65.14%	65.24%
	CLL	84.08%	72.21%	76.87%
	CML	71.70%	81.70%	75.53%
	Healthy	98.92%	99.06%	98.97%



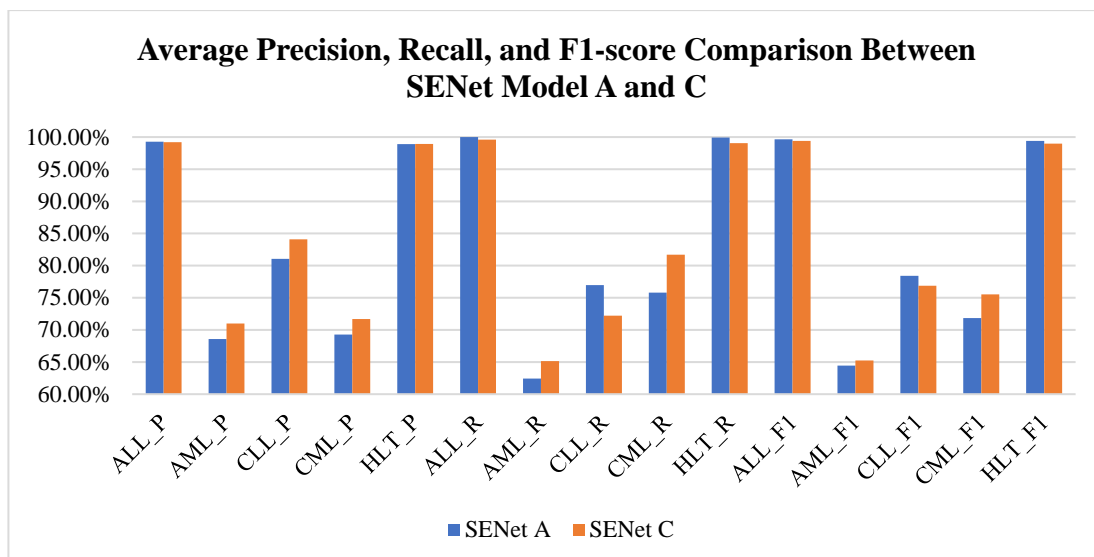


Figure 4.55: Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and C

It is observed that there is an increase of 0.52% in the average testing accuracy for the SENet model C compared to the SENet model A. In terms of the average precision of the SENet model C, a slight decrease of 0.07% is observed for ALL class compared to the SENet model A. In contrast, the other 4 classes observed an improvement in the average precision, such that there is an increase of 2.42% for AML class, 3.02% for CLL class, 2.42% for CML class, and 0.01% for healthy class. On the other hand, the average recall of the SENet model C saw a decrease for ALL, CLL, and healthy classes of 0.39%, 4.77%, and 0.87%, respectively, while an increase for AML and CML classes of 2.71%, and 5.9%, respectively, compared to the SENet model A. For the average F1-score, compared to the SENet model A, there is a slight decrease for ALL, CLL, and healthy classes of 0.24%, 1.55%, and 0.44%, respectively. However, this decrease is compensated with an increase in the average F1-score for AML and CML classes of 0.79% and 3.68%, respectively.

Following on the fifth fold comparison, in which both SENet models have the highest testing accuracy, it is observed that the ROC curves of the SENet model C has a sharper bent on the AML class compared to the SENet model A, while other classes remain approximately the same. Furthermore, looking into the AUC-ROC values, an improvement is observed in the SENet model C compared to the SENet model A, whereby there is an increase for the AUC-ROC value of AML class, CLL class, and CML class against the rest of 0.025275, 0.002817, and 0.013433, respectively. This

shows that SENet model C has a stronger class separability for all classes than SENet model A. Furthermore, by looking into the confusion matrixes, it is observed that the SENet model C has improved in correctly classifying images of CLL and CML cells but lacks behind when classifying images of AML cells compared to the SENet model A. All in all, minor improvements are observed in the SENet model C as it is able to surpass the testing accuracy and exhibits a slightly stronger class separability compared to the SENet Model A.

Additionally, the SENet model C is further fine-tuned by attempting to reduce the variance problem exhibited, such that the testing accuracy is much lower compared to the training accuracy. This shows that the model is somewhat overfitting the training dataset, causing it to perform poorly in the unseen testing dataset. Henceforth, a regularization technique known as dropout is implemented into the fully connected network of the SENet model C. The dropout layers are added in between the ReLU layers of the model, and the dropout rate,  $p$ , is set to 0.5. This model will be denoted as SENet model D. The overall architecture of the model is shown in Figure 4.56, and its code is listed in Code Listing 33 in Appendix A.

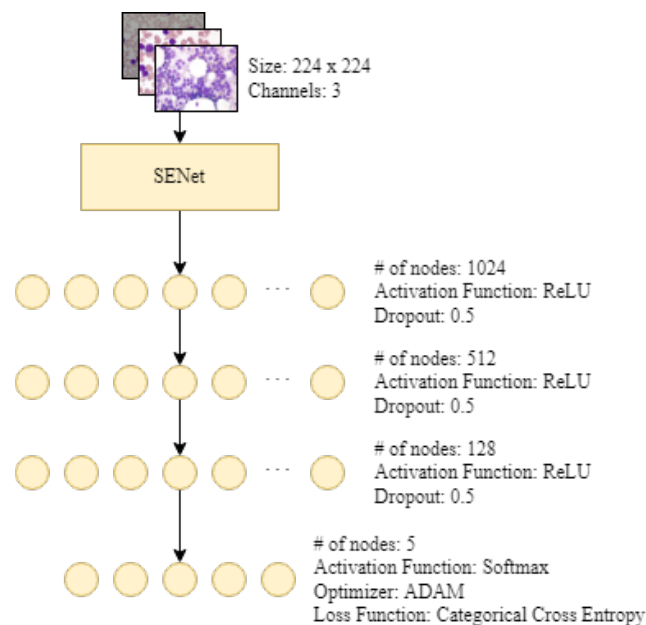


Figure 4.56: Overall Architecture of Each Model for SENet model D

Its accuracies and losses are tabulated as shown in Table 4.19. The bar chart in Figure 4.58 compares all proposed models, including the fine-tuned SENet models, to

the other literature studies on the 5-class classification problem. It is observed that there is an increase of 0.93% in the average testing accuracy for the SENet model D compared to the SENet model C, whereas an increase of 1.45% is observed when compared to the SENet model A. The SENet model D also surpassed both the Inception-V3 and ResNeXt models, as well as the model proposed by Ahmed, et al.

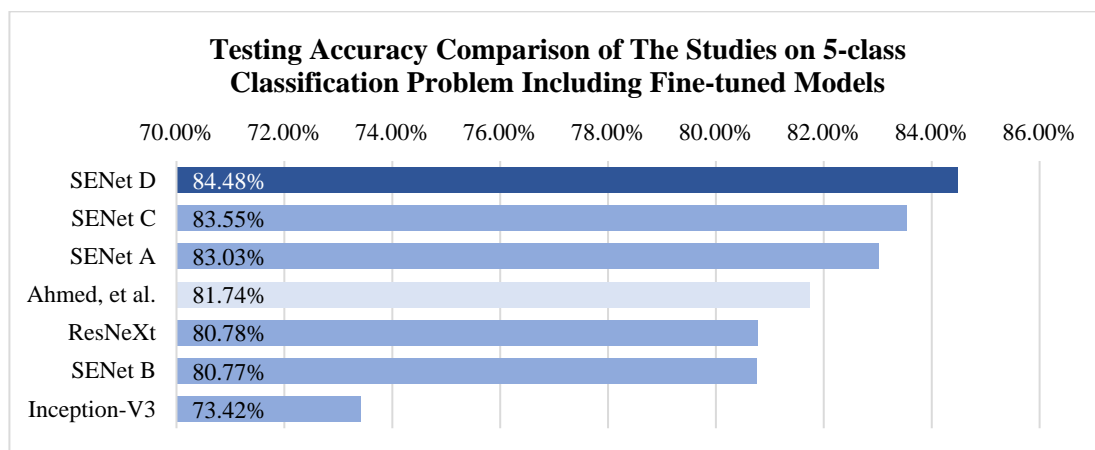
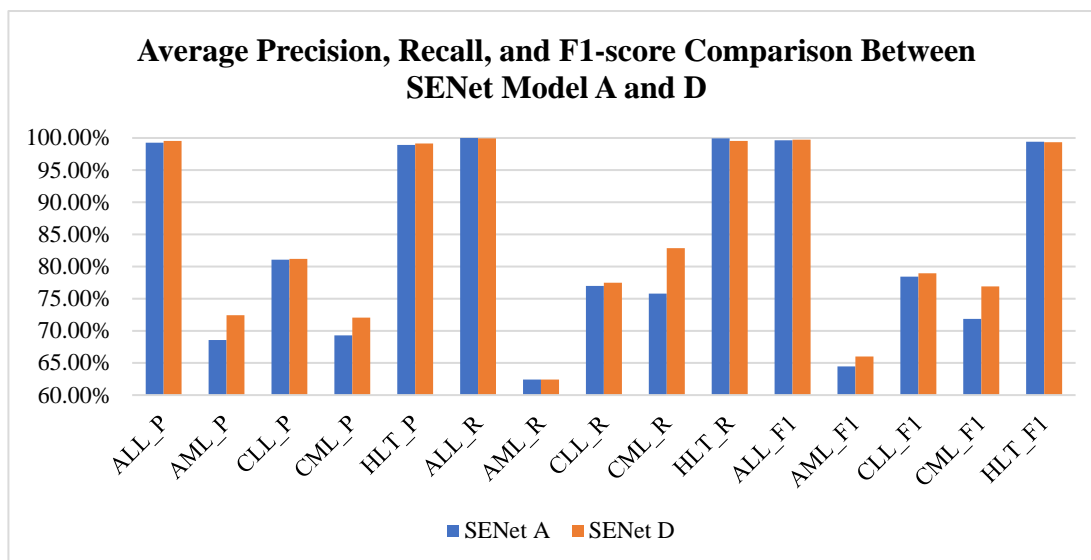


Figure 4.57: Bar Chart of The Testing Accuracy Comparison of The Studies on 5-class Classification Problem Including Fine-tuned Models

Furthermore, the average precision, recall, and F1-score is calculated and tabulated as shown in Table 4.31. The bar chart shown in Figure 4.57 compares the average precision, recall, and F1-score between SENet model A and D. In terms of the average precision, when compared with the SENet model A, most of the classes observed an improvement, such that an increase of 0.25% for ALL class, 3.86% for AML class, 0.13% for CLL class, 2.77% for CML class, and 0.23% for healthy class. On the other hand, the average recall of the SENet model D saw a minor decrease for ALL and healthy classes of 0.07% and 0.40%, respectively, while an increase for CLL and CML classes of 0.51% and 7.06%, respectively, compared to the SENet model A. For the average F1-score, compared to the SENet model A, there is a slight decrease for healthy cells of 0.08% only, but the other classes observed an improvement such that there is an increase of 0.09% for ALL class, 1.55% for AML class, 0.53% for CLL class, and 5.05% for CML class.

**Table 4.31: Average Precision, Recall, and F1-score for 5-class Classification Problem for SENet model A and D**

Model	Classes	Precision	Recall	F1-score
<b>SENet Model A</b>	<b>ALL</b>	99.28%	100.00%	99.64%
	<b>AML</b>	68.58%	62.43%	64.45%
	<b>CLL</b>	81.06%	76.98%	78.42%
	<b>CML</b>	69.28%	75.80%	71.85%
	<b>Healthy</b>	98.91%	99.93%	99.41%
<b>SENet Model D</b>	<b>ALL</b>	99.53%	99.93%	99.73%
	<b>AML</b>	72.44%	62.43%	66.00%
	<b>CLL</b>	81.19%	77.49%	78.95%
	<b>CML</b>	72.05%	82.86%	76.90%
	<b>Healthy</b>	99.14%	99.53%	99.33%



**Figure 4.58: Bar Chart of The Average Precision, Recall, and F1-score Comparison Between SENet Model A and D**

Next, taking into account only the fold where both SENet models have the highest testing accuracy, which is the fifth fold, it is also observed that the ROC curves of the SENet model D have a sharper bent on the AML class compared to the SENet model A, while other classes remain approximately the same. Looking into the AUC-ROC values, an improvement is observed in the SENet model D compared to the SENet model A, whereby there is an increase for the AUC-ROC values of AML

class, CLL class, and CML class against the rest by 0.02021, 0.002036, and 0.004552 respectively. This shows that the SENet model D has a stronger class separability for all classes than the SENet model A but slightly lacks behind compared to the SENet model C. Furthermore, by looking into the confusion matrixes, it is observed that the SENet model D also has improved in correctly classifying images of CLL and CML cells compared to the SENet model A but lacks behind when classifying images of AML cells. All in all, improvements are observed in the SENet model D as it is able to surpass the testing accuracy of both the SENet model A and C. It also has a higher average F1-score, as well as a stronger class separability. With that said, the SENet model D will be implemented on the binary classification problem.

The accuracies and losses for the binary classification problem are tabulated as shown in Table 4.25. The bar chart shown in Figure 4.59 compares all the proposed models in this project, including fine-tuned SENet models, to the other literature studies on the binary classification problem. It is observed that there is a minor decrease of 0.03% in the average testing accuracy compared to the SENet model A, while still surpassing both the Inception-V3 and ResNeXt model, as well as all other models proposed by related works.

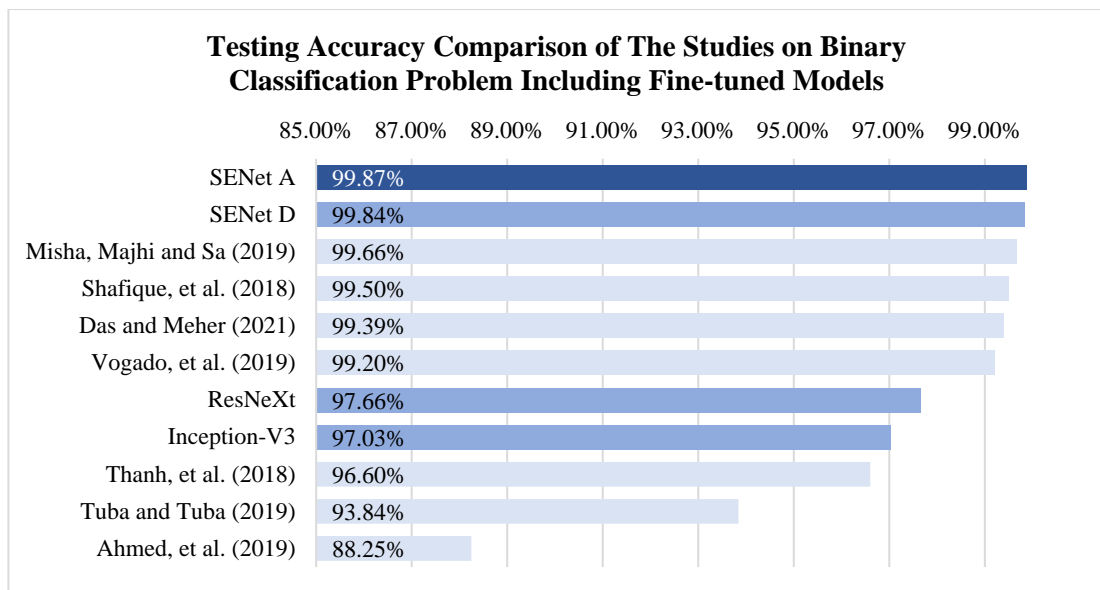


Figure 4.59: Bar Chart of The Testing Accuracy Comparison of The Studies on Binary Classification Problem Including Fine-tuned Models

Besides, the average precision, recall, and F1-score are calculated as 99.88%, 99.81%, and 99.84%, respectively. It is observed that the SENet model D has a minor improvement in terms of precision but has a slightly lower recall and F1-score compared to the SENet model A. Whereas, in terms of the ROC curves, AUC-ROC values, and confusion matrixes, both models have similar results. Overall, the SENet model D can still tackle the binary classification problem as good as the SENet model A, and it is also capable of tackling the 5-class classification problem very well. Therefore, it is without a doubt that the SENet model D provides the best performance rate in detecting leukemia cells from healthy cells and classifying the leukemia cells into their subtypes.

## CHAPTER 5

### CONCLUSION AND RECOMMENDATIONS

#### 5.1 Project Review

This project aims to produce an efficient way of detecting and classifying leukemia using deep learning models. The first objective is to select and train a suitable deep learning model to detect leukemia cells from healthy cells and classify them into their subtypes. Therefore, three models are selected for this project, namely the Inception-V3, ResNeXt, and SENet models, and are made to achieve this objective. The models are downloaded as pretrained models through multiple APIs available online, leukemia images are then fed to train the models, and unseen testing data are used to evaluate their capability in detecting and classifying leukemia.

The second objective of this project is to fine-tune the selected model for a better performance rate in detecting and classifying leukemia. Throughout the project, it is determined that the SENet model provides the best performance rate among the three selected models. Henceforth, the SENet model was fine-tuned to improve its performance rate further. The modifications include replacing the softmax classifier with an SVM classifier, increasing the layers in the fully connected network with different dimensional feature vectors, and implementing dropout layers in the fully connected network.

## 5.2 Project Findings

Among the models, before fine tunings are made, the SENet model produces the best performance rate in both binary and 5-class classification problems. The average testing accuracy achieved by the SENet model in the binary classification problem is 99.87%, whereby the highest it can obtain is 100.00% in the first and fifth fold. It holds the best score compared to the other literature studies, with the model proposed by Shafique, et al., following closely at an average testing accuracy of 99.50%. Besides, the SENet model also exhibited the highest average recall and F1-score of 99.93% and 99.87%, respectively, compared to the other models. Furthermore, by observing the ROC curves generated by the model, sharp bents are exhibited in each fold, and the AUC-ROC values are determined to be at an average value of 1. In addition, the diagonal elements of its confusion matrixes have a higher value and have a darker colour than the neighbouring elements. These performance metrics prove that the SENet model has an excellent class separability for the binary classification problem.

On the other hand, the average testing accuracy achieved by the SENet model in the 5-class classification problem is 83.03%, whereby the highest it can obtain is 87.41% in the fifth fold. It holds the best score compared to the other literature studies, with the model proposed by Ahmed, et al., closely following an average testing accuracy of 81.74%. In terms of precision, recall, and F1-score, it is found that the ResNeXt model exhibited a good performance as observed in its average precision for all classes, while the SENet model has a better performance as observed in its average recall and F1-score for all classes. It has been mentioned before that the F1-score provides a better measurement of the model's performance in terms of precision and recall. Besides, it is also better to avoid models with low recall in the medical field because it may result in a patient who actually has leukemia being diagnosed as not having it. It will cause the patient to miss its window of opportunity for treatment, leading to undesirable consequences. With that, it can be said that the SENet model provides the best performance compared to the other models. This finding is justified further by the ROC curves, AUC-ROC values, and confusion matrixes generated by the SENet model, whereby a strong class separability for most of the classes compared to the other models. Thus, it is concluded that the SENet model also provides the best performance rate in the 5-class classification problem.



It is obvious that the SENet model aced both binary and 5-class classification problem compared to the other models. Therefore, it will be fine-tuned to improve its accuracy and overall performance. Two modifications are experimented on the SENet model, or SENet model A, which includes replacing the model's classifier with a machine learning algorithm, and increasing the layers in the fully connected network with different dimensional feature vectors. It is observed that the latter model, SENet model C, saw a minor increase in the model's accuracy by 0.52% in tackling the 5-class classification problem, while the former, SENet model B, degrades it by 2.26%. The SENet model C also had a minor improvement on the class separability compared to the SENet model A. This shows that the slow converging feature selection process enabled by the different dimensional feature vectors allowed more important features to be captured by the model. Hence, the SENet model C is further fine-tuned by adding a regularization technique known as dropout into its fully connected network, producing the SENet model D. Dropouts are added to reduce the variance problem exhibited, such that the testing accuracy is much lower compared to the training accuracy.

From the results obtained, an increase of 0.93% in the average testing accuracy is observed compared to the SENet model C, whereas an increase of 1.45% is observed compared to the SENet model A. This shows that regularization techniques such as dropouts had effectively reduced the variance problem, allowing the model to perform better predictions on the unseen testing images. Furthermore, the SENet model D saw an improvement in the precision, recall, and F1-score for most of the classes compared to the SENet model A. The ROC curves, AUC-ROC values, and confusion matrixes also show that the SENet model D has a stronger class separability for all classes. With that said, the SENet model D will be implemented on the binary classification problem. Minor improvements are observed in terms of precision, but the model has a slightly lower recall and F1-score than the SENet model A. It is also found that both models have similar results in the ROC curves, AUC-ROC values, and confusion matrixes. Overall, the SENet model D is capable of tackling the binary classification problem as good as the SENet model A and tackling the 5-class classification problem very well.

### 5.3 Recommendations for Future Improvements

Though this project produced a model, the SENet model D, that can generate excellent results in detecting and classifying leukemia compared to other literature studies, there are still limitations that can be overcome to create a better model with higher performance. The first limitation is the scarcity of the cell images used to train the model. It is without a doubt that the larger the dataset, the better. However, in this project, although the dataset is large, the data are mostly augmented, affecting how the models learn the features from the images. Data augmentation may be a good technique to increase dataset size but should still be used considerably to avoid having the model capture random noise instead of the important features. Hence, it is recommended that more cell images of higher quality be used to train the models for this project, which could effectively improve the model's prediction.

Furthermore, the following limitation is the software and hardware used to train the models. The models chosen for this project have a complex architecture that requires a high-performance unit to train and test them. Although the current method of using Google Colab works well, the RAM size allowed to be used is 25.46GB only if the program is running on Intel Xeon CPU provided by the website. On the other hand, if GPU is selected, the only RAM size allowed to be used is only 12.69GB, which is too little for such a heavy computation required by the models. Needless to say, the hardware used in this project, which has Intel Core i7-6700HQ and RAM size of 8GB, has the worst computing performance but still allows websites such as Google Colab to be accessed smoothly. Henceforth, seeing no other options available, CPUs provided by the Google Colab are used to train the models, which took a long time, with the longest being one day of non-stop training. In addition, if the SENet model D were to detect and classify leukemia, it would also require a powerful processing unit or a cloud server to host the model. Thus, it is recommended that a much light-weight model be used for this project. Besides needing lesser time to train, the models may also be implemented on mobile devices or programmable circuit boards such as the Raspberry Pi or the Jetson Nano for a cheaper and more convenient usage.

On top of that, this project only focuses on detecting and classifying leukemia into the main types: ALL class, AML class, CLL class, and CML class. However, it is

studied that these leukemia classes can be further differentiated into more subtypes. For example, the ALL class can be further categorized into 3 subtypes: ALL-L1, ALL-L2, and ALL-L3. On the other hand, the AML class can be further categorized into 8 more subtypes: AML-M0, AML-M1, AML-M2, AML-M3, AML-M4, AML-M5a, AML-M5b, AML-M6, and AML-M7. Therefore, it is recommended that these subtypes are to be included in future projects so that a more specific prediction can be made, allowing efficient and accurate treatment of the types of leukemia detected and classified.

#### **5.4 Conclusion**

The objectives of this project are achieved. The models trained are able to detect leukemia cells from healthy cells and classify them into their subtypes. Among the three models selected for this project, the SENet model exhibited the best performance with the highest average testing accuracy of 99.87% and 83.03% for binary and 5-class classification, respectively. The SENet model is then fine-tuned by replacing the model's classifier with a machine learning algorithm, and increasing the layers in the fully connected network with different dimensional feature vectors. The latter modification saw a minor increase in the model's accuracy by 0.52% when tackling the 5-class classification problem, while the former degrades it by 2.26%. The latter model is further fine-tuned by adding a regularization technique known as dropout into its fully connected network, producing the SENet model D. The average testing accuracy obtained is 99.84% and 84.48% for binary and 5-class classification, respectively. All in all, the SENet model D is capable of tackling both binary and 5-class classification problem very well. It also achieved the highest testing accuracy on binary and 5-class classification problems compared to other works of literature.

## REFERENCES

- Ahmed, N., Yigit, A., Isik, Z. and Alpkocak, A., 2019. Identification of Leukemia Subtypes from Microscopic Images Using Convolutional Neural Network. *Diagnostics*, 9(3), pp. 104. <https://dx.doi.org/10.3390/diagnostics9030104>.
- Abhishek, A., Jha, R. K., Sinha, R. and Jha, K., 2021. Automated classification of acute leukemia on a heterogeneous dataset using machine learning and deep learning techniques. *Biomedical Signal Processing and Control*, 72(B). <https://doi.org/10.1016/j.bspc.2021.103341>.
- Ahmed, N., Yigit, A., Isik, Z. and Alpkocak, A., 2019. Identification of Leukemia Subtypes from Microscopic Images Using Convolutional Neural Network. *Diagnostics*, 9(3), pp. 104. <https://dx.doi.org/10.3390/diagnostics9030104>.
- Albelwi, S. and Mahmood, A., 2017. A Framework for Designing the Architectures of Deep Convolutional Neural Networks. *Entropy*, 19(6). <https://doi.org/10.3390/e19060242>.
- Alzubaidi, L., Zhang, J., Humaidi, A.J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaria, J., Fadhel, M.A., Al-Amidie, M. and Farhan, L., 2021. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data*, 8(53). <https://dx.doi.org/10.1186/s40537-021-00444-8>.
- American Society of Hematology, n.d.. *Image Bank*. [online] Available at: <<http://imagebank.hematology.org>> [Accessed 24 June 2021].
- Amiri, R., Mehrpouyan, H., Fridman, L., Mallik, R.K., Nallanathan, A. and Matolak, D., 2018. A Machine Learning Approach for Power Allocation in HetNets Considering QoS. *2018 IEEE International Conference on Communications (ICC)*. <https://dx.doi.org/10.1109/icc.2018.8422864>.
- Ardakani, A., Condo, C. and Gross, W.J., 2017. Sparsely-Connected Neural Networks: Towards Efficient VLSI Implementation of Deep Neural Networks. <https://arxiv.org/abs/1611.01427v3>.
- Arnold, M.J., Keung, J.J. and McCarragher, B., 2019. Interventional radiology: Indications and Best Practices. *American Family Physician*, 99(9), pp. 547-556. <https://www.aafp.org/afp/2019/0501/p547.html>.

- Bain, B.J. 2005. Diagnosis from the Blood Smear. *New England Journal of Medicine*, 353(5), pp. 498-507. <https://dx.doi.org/10.1056/nejmra043442>.
- Bain, B.J., 2015. *Blood cells*. 5th ed. Chichester: John Wiley & Sons Ltd.
- Basel, K., 2018. *Python Pros and Cons*. [online] Available at: <<https://www.netguru.com/blog/python-pros-and-cons>> [Accessed 17 August 2021].
- Behl, A., Bhatiam, A. and Puri, A., 2014. Convolution and Applications of Convolution. *International Journal of Innovative Research in Technology*, 1(6), pp. 2122-2126. <http://ijirt.org/Article?manuscript=101029>.
- Best, N., Ott, J. and Linstead E.J., 2020. Exploring the efficacy of transfer learning in mining image-based software artifacts. *Journal of Big Data*, 7(59). <https://dx.doi.org/10.1186/s40537-020-00335-4>.
- Bibi, N., Sikandar, M., Ud Din, I., Almogren, A. and Ali, S., 2020. IoMT-Based Automated Detection and Classification of Leukemia Using Deep Learning. *Journal of Healthcare Engineering*, 2020, pp. 1-12. <https://dx.doi.org/10.1155/2020/6648574>.
- Bjorck, J., Gomes, C., Selman, B. and Weinberger, K.Q., 2018. Understanding Batch Normalization. <https://arxiv.org/abs/1806.02375v4>.
- Brownlee, J., 2019. *How Do Convolutional Layers Work in Deep Learning Neural Networks?*. [online] Available at: <<https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/>> [Accessed 16 August 2021].
- Cancer Research UK, 2019. *Stages*. [online] Available at: <<https://www.cancerresearchuk.org/about-cancer/chronic-myeloid-leukaemia-cml/stages>> [Accessed 9 July 2021].
- China, A., 2009. Understanding the Principles of Recursive Neural Networks: A Generative Approach to Tackle Model Complexity. *Lecture Notes in Computer Science*, 5768, pp. 952-963. [https://dx.doi.org/10.1007/978-3-642-04274-4\\_98](https://dx.doi.org/10.1007/978-3-642-04274-4_98).
- Das, P. K. and Meher, S, 2021. An efficient deep Convolutional Neural Network based detection and classification of Acute Lymphoblastic Leukemia. *Expert Systems with Applications*. <https://doi.org/10.1016/j.eswa.2021.115311>.
- Deng, L. and Yu, D., 2014. Deep learning: Methods and Applications. *Foundations and Trends® in Signal Processing*, 7(3-4), pp. 197–387. <https://dx.doi.org/10.1561/20000000039>.

- Dwivedi, A.K., 2018. Artificial neural network model for effective cancer classification using microarray gene expression data. *Neural Computing & Applications*, 29, pp. 1545–1554. <https://dx.doi.org/10.1007/s00521-016-2701-1>.
- Feng, J. and Lu, S., 2019. Performance Analysis of Various Activation Functions in Artificial Neural Networks. <http://dx.doi.org/10.1088/1742-6596/1237/2/022030>.
- Fukushima, K., 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), pp. 193–202. <https://dx.doi.org/10.1007/bf00344251>.
- Grandini, M., Bagli, E. and Visani, G., 2020. Metrics for Multi-Class Classification: An Overview. <https://doi.org/10.48550/arXiv.2008.05756>.
- He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. <https://arxiv.org/abs/1512.03385v1>.
- Hosseini-Asl, E., Gimel'farb, G. and El-Baz, A., 2016. Alzheimer's Disease Diagnostics by a Deeply Supervised Adaptable 3D Convolutional Network. <https://arxiv.org/abs/1607.00556v1>.
- Hu, J., Shen, L., Albanie, S., Sun, G. and Wu, E., 2019. Squeeze-and-Excitation Networks. <https://arxiv.org/abs/1709.01507v4>.
- Huang, G., Liu, Z., Maaten, L. van der and Weinberger, K. Q., 2017. Densely Connected Convolutional Networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://dx.doi.org/10.1109/cvpr.2017.243>.
- ImageNet, n.d.. *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*. [online] Available at: <<https://www.image-net.org/challenges/LSVRC/>> [Accessed 14 April 2022].
- Indolia, S., Goswami, A. K., Mishra, S. P. and Asopa, P., 2018. Conceptual Understanding of Convolutional Neural Network- A Deep Learning Approach. *Procedia Computer Science*, 132, pp. 679–688. <https://dx.doi.org/10.1016/j.procs.2018.05.069>.
- Ioffe, S. and Szegedy, C., 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML'15 Proceedings of the 32<sup>nd</sup> International Conference on International Conference on Machine Learning*, 37, pp. 448-456. <https://dl.acm.org/doi/10.5555/3045118.3045167>.
- Ker, J., Wang, L., Rao, J. and Lim, T., 2018. Deep Learning Applications in Medical Image Analysis. *IEEE Access*, 6, pp. 9375–9389. <https://dx.doi.org/10.1109/access.2017.2788044>.

- Khan, A., Sohail, A., Zahoora, U. and Qureshi, A.S., 2020. A survey of recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, 53, pp. 5455-5516. <https://doi.org/10.1007/s10462-020-09825-6>.
- Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), pp. 84–90. <https://dx.doi.org/10.1145/3065386>.
- Ladines-Castro, W., Barragán-Ibañez, G., Luna-Pérez, M.A., Santoyo-Sánchez, A., Collazo-Jaloma, J., Mendoza-Garcia, E. and Ramos-Peñafiel, C.O., 2016. Morphology of leukaemias. *Revista Médica Del Hospital General de México*, 79(2), pp. 107-113. <https://dx.doi.org/10.1016/j.hgmx.2015.06.007>.
- LeCun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *Nature*, 521(7553), pp. 436–444. <https://dx.doi.org/10.1038/nature14539>.
- LeCun, Y., Jackel, L.D., Boser, B., Denker, J.S., Graf, H.P., Guyon, I., Henderson, D., Howard, R.E. and Hubbard, W., 1990. Handwritten Digit Recognition: Applications of Neural Net Chips and Automatic Learning. *Neurocomputing*, pp. 303–318. [https://dx.doi.org/10.1007/978-3-642-76153-9\\_35](https://dx.doi.org/10.1007/978-3-642-76153-9_35).
- Lin, M., Chen, Q. and Yan, S., 2014. Network In Network. <https://arxiv.org/abs/1312.4400v3>.
- Lin, Y., Lv, F., Zhu, S., Yang, M., Cour, T. and Yu, K., 2011. Large-scale Image Classification: Fast Feature Extraction and SVM Training. *CVPR 2011*. <https://dx.doi.org/10.1109/CVPR.2011.5995477>.
- Lo, S.-C. B., Lou, S.-L. A., Lin, J.-S., Freedman, M.T., Chien, M.V. and Mun, S.K., 1995. Artificial convolution neural network techniques and applications for lung nodule detection. *IEEE Transactions on Medical Imaging*, 14(4), pp. 711–718. <https://dx.doi.org/10.1109/42.476112>.
- Löffler, H. and Gassmann, W., 1994. Morphology and cytochemistry of acute lymphoblastic leukaemia. *Bailliere's Clinical Haematology*, 7(2), pp. 263-272. [https://dx.doi.org/10.1016/s0950-3536\(05\)80202-1](https://dx.doi.org/10.1016/s0950-3536(05)80202-1).
- Ma, H., Liu, Y., Ren, Y. and Yu, J., 2019. Detection of Collapsed Buildings in Post-Earthquake Remote Sensing Images Based on the Improved YOLOv3. *Remote Sensing*, 12(1), 44. <https://dx.doi.org/10.3390/rs12010044>.
- Maleki, F., Ovens, K., Najafian, K., Forghani, B., Reinhold, C. and Forghani, R., 2020. Overview of Machine Learning Part 1. *Neuroimaging Clinics of North America*, 30(4), pp. e17–e32. <https://doi.org/10.1016/j.nic.2020.08.007>.

- Mayo Clinic, 2021. *Leukemia*. [online] Available at: <<https://www.mayoclinic.org/diseases-conditions/leukemia/symptoms-causes/syc-20374373>> [Accessed 9 July 2021].
- Mishra, C. and Gupta, D.L., 2017. Deep Machine Learning and Neural Networks: An Overview. *IAES International Journal of Artificial Intelligence*, 6(2), pp. 66-73. <https://dx.doi.org/10.11591/ijai.v6.i2.pp66-73>.
- Mpitsos, G.J. and Burton, R.M., Jr., 1992. Convergence and divergence in neural networks: Processing of chaos and biological analogy. *Neural Networks*, 5(4), pp. 605–625. [https://doi.org/10.1016/S0893-6080\(05\)80039-5](https://doi.org/10.1016/S0893-6080(05)80039-5).
- Mustafid, A., Pamuji M.M. and Helmiyah, S., 2020. A Comparative Study of Transfer Learning and Fine-Tuning Method on deep Learning Models for Wayang Dataset Classification. *International Journal on Informatics for Development*, 9(2), pp. 100-110. <https://dx.doi.org/10.14421/ijid.2020.09207>.
- National Cancer Institute, 2020. *SEER Cancer Statistics Review (CSR) 1975-2017*. [online] Available at: <[https://seer.cancer.gov/archive/csr/1975\\_2017/](https://seer.cancer.gov/archive/csr/1975_2017/)> [Accessed 9 July 2021].
- Narkhede, S., 2018. *Understanding AUC-ROC Curve*. [online] Available at: <<https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>> [Accessed 15 April 2022].
- National Cancer Institute, 2021. *Cancer Stat Facts: Leukemia*. [online] Available at: <<https://seer.cancer.gov/statfacts/html/leuks.html>> [Accessed 9 July 2021].
- Neftci, E.O. and Averback, B.B., 2019. Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, 1, pp. 133-143. <https://dx.doi.org/10.1038/s42256-019-0025-4>.
- Omkar, N., 2019. *Activation Functions with Derivative and Python code: Sigmoid vs Tanh Vs ReLU*. [online] Available at: <<https://medium.com/@omkar.nallagoni/activation-functions-with-derivative-and-python-code-sigmoid-vs-tanh-vs-relu-44d23915c1f4>> [Accessed 14 August 2021].
- Onciu, M., 2009. Acute Lymphoblastic Leukemia. *Hematology/Oncology Clinics of North America*, 23(4), pp. 655–674. <https://dx.doi.org/10.1016/j.hoc.2009.04.009>.
- Ouali, Y., Hudelot C. and Tami, M., 2020. An Overview of Deep Semi-Supervised Learning. <https://arxiv.org/abs/2006.05278v2>.
- Pan, S.J. and Yang, Q., 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10), pp. 1345-1359. <https://dx.doi.org/10.1109/TKDE.2009.191>.



- Pascanu, R., Gulcehre, C., Cho, K. and Bengio, Y., 2014. How to Construct Deep Recurrent Neural Networks. <https://arxiv.org/abs/1312.6026v5>.
- Pejovic, T. and Schwartz, P.E., 2002. Leukemias. *Clinical Obstetrics and Gynecology*, 45(3), pp. 866–878. <https://dx.doi.org/10.1097/00003081-200209000-00033>.
- Pihlajamäki, T., 2009. Multi-resolution Short-time Fourier Transform Implementation of Directional Audio Coding. [https://www.researchgate.net/publication/267239829\\_Multi-resolution\\_Short-time\\_Fourier\\_Transform\\_Implementation\\_of\\_Directional\\_Audio\\_Coding](https://www.researchgate.net/publication/267239829_Multi-resolution_Short-time_Fourier_Transform_Implementation_of_Directional_Audio_Coding).
- Pratt, H., Coenen, F., Broadbent, D.M., Harding, S.P. and Zheng, Y., 2016. Convolutional Neural Networks for Diabetic Retinopathy. *Procedia Computer Science*, 90, pp. 200–205. <https://doi.org/10.1016/j.procs.2016.07.014>.
- Qian, B., Su, J., Wen, Z., Jha, D.N., Li, Y., Guan, Y., Puthal, D., James, P., Yang, R., Zomaya, A.Y., Rana, O., Wang, L., Koutny, M. and Ranjan, R., 2020. Orchestrating the Development Lifecycle of Machine Learning-based IoT Applications: A Taxonomy and Survey. *ACM Computing Surveys*, 53(4), pp. 1-47. <https://dx.doi.org/10.1145/3398020>.
- Rajkomar, A., Lingam, S., Taylor, A.G., Blum, M. and Mongan, J., 2016. High-Throughput Classification of Radiographs Using Deep Convolutional Neural Networks. *Journal of Digital Imaging*, 30(1), pp. 95–101. <https://dx.doi.org/10.1007/s10278-016-9914-9>.
- Rajpurkar, P., Irvin, J., Zhu, K., Yang, B., Mehta, H., Duan, T., Ding, D., Bagul, A., Ball, R.L., Langlotz, C., Shpanskaya, K., Lungren, M.P. and Ng, A.Y., 2017. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. <https://arxiv.org/abs/1711.05225v3>.
- Rakitienskaia, A. and Engelbrecht, A., 2015. Measuring Saturation in Neural Networks. *2015 IEEE Symposium Series on Computational Intelligence*. doi: <https://dx.doi.org/10.1109/ssci.2015.202>.
- Rehman, A., Abbas, N., Saba, T., Rahman, S.I. ur, Mehmood, Z. and Kolivand, H., 2018. Classification of acute lymphoblastic leukemia using deep learning. *Microscopy Research and Technique*, 81(11), pp. 1310-1317. <https://dx.doi.org/10.1002/jemt.23139>.
- Reynolds, A.H., 2019. *Convolutional Neural Networks (CNNs)*. [online] Available at: <https://anhreynolds.com/blogs/cnn.html> [Accessed 12 August 2021].
- Rezk, N.M., Purnaprajna, M., Nordström, T. and Ul-Abdin, Z., 2020. Recurrent Neural Networks: An Embedded Computing Perspective. *IEEE Access*, 8, pp. 57967-57996. <https://dx.doi.org/10.1109/ACCESS.2020.2982416>.

- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3), pp. 211-252. <https://dx.doi.org/10.1007/s11263-015-0816-y>.
- S, V.S., 2020. *Peripheral Smear Findings in Leukemia – Illustrated*. [online] Available at: <<http://ilovepathology.com/microscopy-of-leukemia-illustrated/>> [Accessed 9 July 2021].
- Sanchez, J., Perronnin, F., Mensink, T. and Verbeek, J., 2013. Compressed Fisher Vectors for Large-Scale Image Classification. RR-8209, 2013. <https://hal.archives-ouvertes.fr/hal-00779493v1>.
- Scotti, F., Labati, R.D. and Piuri, V., 2011. ALL-IDB: The acute lymphoblastic leukemia image database for image processing. *2011 18th IEEE International Conference on Image Processing (ICIP 2011)*, Brussels, Belgium, pp. 2045-2048, September 11-14, 2011. <https://dx.doi.org/10.1109/icip.2011.6115881>.
- Shafique, S. and Tehsin, S., 2018. Acute Lymphoblastic Leukemia Detection and Classification of Its Subtypes Using Pretrained Deep Convolutional Neural Networks. *Technology in Cancer Research & Treatment*. <https://dx.doi.org/10.1177/1533033818802789>.
- Shaheen, M., Khan, R., Biswal, R.R., Ullah, M., Khan, A., Uddin, M.I., Zareei, M. and Waheed, A., 2021. Acute Myeloid Leukemia (AML) Detection Using AlexNet Model. *Complexity*, 2021. <https://dx.doi.org/10.1155/2021/6658192>.
- Shamsaldin, A.S., Fattah, P., Rashid, T.A. and Al-Salihi, N.K., 2019. A Study of the Applications of Convolutional Neural Networks. *UKH Journal of Science and Engineering*, 3(2), pp. 31-40. <https://dx.doi.org/10.25079/ukhjse.v3n2y2019.pp31-40>.
- Simonyan, K. and Zisserman, A., 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. <https://arxiv.org/abs/1409.1556v6>.
- Singh, S. *Understanding the Bias-Variance Tradeoff*. [online] Available at: <<https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229>> [Accessed 30 March 2022].
- Socher, R., Lin C.C., Ng, A.Y. and Manning, C.D., 2011. Parsing natural scenes and natural language with recursive neural networks. *ICML'11: Proceedings of the 28<sup>th</sup> International Conference on International Conference on Machine Learning*, pp. 129-136. <https://dl.acm.org/doi/10.5555/3104482.3104499>.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of*

- Machine Learning Research*, 15(1), pp. 1929-1958.  
<https://dl.acm.org/doi/10.5555/2627435.2670313>.
- Srivastava, S., 2020. *Top Programming Languages in Trend for AI Projects in 2020*. [online] Available at: <<https://www.analyticsinsight.net/top-programming-languages-in-trend-for-ai-projects-in-2020/>> [Accessed 17 August 2021].
- Su, C.-J. and Li, Y., 2019. Recurrent neural network based real-time failure detection of storage devices. *Microsystem Technologies*. <https://dx.doi.org/10.1007/s00542-019-04454-8>.
- Szandała, T., 2020. Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks. <https://arxiv.org/abs/2010.09458>.
- Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A., 2017.-v4, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *AAAI'17: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 2017, pp. 4278-4284. <https://dl.acm.org/doi/10.5555/3298023.3298188>.
- Szegedy, C., Liu W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1-9. <https://dx.doi.org/10.1109/CVPR.2015.7298594>.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z., 2016. Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://dx.doi.org/10.1109/cvpr.2016.308>.
- Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C. and Liu, C., 2018. A Survey on Deep Transfer Learning. *Lecture Notes in Computer Science*, pp. 270–279. [https://dx.doi.org/10.1007/978-3-030-01424-7\\_27](https://dx.doi.org/10.1007/978-3-030-01424-7_27).
- Thanh, T.T.P., Vununu, C., Atoev, S., Lee S.-H. and Kwon, K.-R., 2018. Leukemia Blood Cell Image Classification Using Convolutional Neural Network. *International Journal of Computer Theory and Engineering*, 10(2), pp. 54-58. <https://doi.org/10.7763/IJCTE.2018.V10.1198>.
- Tsang, S.-H., 2018. *Review: AlexNet, CaffeNet – Winner of ILSVRC 2012 (Image Classification)*. [online] Available at: <<https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160>> [Accessed 17 August 2021].
- Vogado, L.H.S., Veras, R.M.S., Araujo, F.H.D., Silva, R.R.V. and Aires, K.R.T., 2018. Leukemia diagnosis in blood slides using transfer learning in CNNs and SVM for classification. *Engineering Applications of Artificial Intelligence*, 72, pp. 415–422. <https://dx.doi.org/10.1016/j.engappai.2018.04.024>.

- Weiss, K., Khoshgoftaar, T.M. and Wang, D., 2016. A survey of transfer learning. *Journal of Big Data*, 3(1). <https://dx.doi.org/10.1186/s40537-016-0043-6>.
- Wickramasinghe, S. *Bias & Variance in Machine Learning: Concepts & Tutorials*. [online] Available at: <<https://www.bmc.com/blogs/bias-variance-machine-learning/>> [Accessed 30 March 2022].
- Wikipedia, 2021. *ImageNet*. [online] Available at: <<https://en.wikipedia.org/wiki/ImageNet>> [Accessed 6 August 2021].
- Wu, D., Wang, Y., Xia, S.-T., Bailey, J. and Ma, X., 2020. Skip Connections Matter: On the Transferability of Adversarial Examples Generated with ResNets. <https://arxiv.org/abs/2002.05990v1>.
- Xie, S., Girshick, R., Dollár, P., Tu, Z. and He, K., 2017. Aggregated Residual Transformations for Deep Neural Networks. <https://arxiv.org/abs/1611.05431>.
- Yamashita, R., Nishio, M., Do, R.K.G. and Tagoshi, K., 2018. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, pp. 611-629. <https://doi.org/10.1007/s13244-018-0639-9>.
- Ying, X., 2019. An Overview of Overfitting and its Solutions. *Journal of Physics: Conference Series*, 1168. <https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022/pdf>.
- Zeiler, M.D. and Fergus, R., 2014. Visualizing and Understanding Convolutional Networks. *Lecture Notes in Computer Science*, pp. 818-833. [https://dx.doi.org/10.1007/978-3-319-10590-1\\_53](https://dx.doi.org/10.1007/978-3-319-10590-1_53).

## APPENDICES

### APPENDIX A: Code Listings

```
from google.colab import drive
drive.mount('/content/drive')
```

Listing A.1: Mount Google Drive onto Google Colab

```
# Import open-source libraries
import os
import cv2
import torch
import pickle
import random
import itertools
import torchvision
import numpy as np
import pandas as pd
import torch.nn as nn
import tensorflow as tf
import torch.optim as optim
from torchvision import models
import matplotlib.pyplot as plt
from sklearn.svm import LinearSVC
from keras import optimizers, losses, models
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential, Model
from sklearn.calibration import CalibratedClassifierCV
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import roc_curve, roc_auc_score, auc
from keras.layers import Dense, Dropout, GlobalAveragePooling2D
from sklearn.metrics import recall, f1_score, precision,
accuracy_score, confusion_matrix
```

Listing A.2: Import Open-Source Libraries

```

# Dataset directory in Google Drive
DIR = '/content/drive/MyDrive/Dataset/'
# Class labels in a list
classes = ['ALL', 'AML', 'CLL', 'CML', 'HLT']

# New list to store images
dataset = []
class_num = 0

for class_names in classes:
    path = os.path.join(DIR, class_names)
    for img in os.listdir(path):
        # Read images into array of numbers
        img_arr = cv2.imread(os.path.join(path, img))
        # Resize images
        new_arr = cv2.resize(img_arr, (224, 224))
        # Normalize images
        norm_arr = new_arr/255.0
        # Store images
        dataset.append([norm_arr, class_num])
    class_num += 1

```

Listing A.3: Import Dataset from Google Drive, Resize And Normalize It, Then Store into A List

```

X = []
Y = []

for data, label in dataset:
    X.append(data)
    Y.append(label)

skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

fold_index = []

for train_index, test_index in skf.split(X, Y):
    fold_index.append([train_index, test_index])

fold_var = int(input("Fold (0-4): "))

train_set = [dataset[train] for train in fold_index[fold_var][0]]
test_set = [dataset[test] for test in fold_index[fold_var][1]]
all_set = [train_set, test_set]

```

Listing A.4: Split Dataset into Stratified K Folds of Train and Test Sets

```

all_aug = []

#data augmentation
zooming = ImageDataGenerator(zoom_range=0.3)
shearing = ImageDataGenerator(shear_range=20)
rotating = ImageDataGenerator(rotation_range=40)
hshfiting = ImageDataGenerator(height_shift_range=0.4)
wshfiting = ImageDataGenerator(width_shift_range=0.4)

for get_set in all_set:
    aug_dataset = []
    for ori_img, class_num in get_set:
        aug_dataset.append([ori_img, class_num])
        aug_dataset.append([np.flipud(ori_img), class_num])
        aug_dataset.append([np.fliplr(ori_img), class_num])
        for _ in range(2):
            it = zooming.flow(np.expand_dims(ori_img*255.0, axis=0),
batch_size=1)
            batch = it.next()
            aug_img = (batch[0].astype('uint8'))/255.0
            aug_dataset.append([aug_img, class_num])
            aug_dataset.append([np.flipud(aug_img), class_num])
            aug_dataset.append([np.fliplr(aug_img), class_num])
        for _ in range(2):
            it = shearing.flow(np.expand_dims(ori_img*255.0, axis=0),
batch_size=1)
            batch = it.next()
            aug_img = (batch[0].astype('uint8'))/255.0
            aug_dataset.append([aug_img, class_num])
            aug_dataset.append([np.flipud(aug_img), class_num])
            aug_dataset.append([np.fliplr(aug_img), class_num])
        for _ in range(2):
            it = rotating.flow(np.expand_dims(ori_img*255.0, axis=0),
batch_size=1)
            batch = it.next()
            aug_img = (batch[0].astype('uint8'))/255.0
            aug_dataset.append([aug_img, class_num])
            aug_dataset.append([np.flipud(aug_img), class_num])
            aug_dataset.append([np.fliplr(aug_img), class_num])
        for _ in range(2):
            it = hshfiting.flow(np.expand_dims(ori_img*255.0, axis=0)
, batch_size=1)
            batch = it.next()
            aug_img = (batch[0].astype('uint8'))/255.0
            aug_dataset.append([aug_img, class_num])
            aug_dataset.append([np.flipud(aug_img), class_num])
            aug_dataset.append([np.fliplr(aug_img), class_num])
        for _ in range(2):

```

```

        it = wshfiting.flow(np.expand_dims(ori_img*255.0, axis=0)
, batch_size=1)
        batch = it.next()
        aug_img = (batch[0].astype('uint8'))/255.0
        aug_dataset.append([aug_img, class_num])
        aug_dataset.append([np.flipud(aug_img), class_num])
        aug_dataset.append([np.fliplr(aug_img), class_num])
    all_aug.append(aug_dataset)

```

Listing A.5: Augment the Datasets and Store into A List Called 'aug\_dataset'

```

# Training set
split_train = [[], [], [], [], []]

for new_img, class_num in all_aug[0]:
    if class_num == 0:
        split_train[class_num].append([new_img, class_num])
    if class_num == 1:
        split_train[class_num].append([new_img, class_num])
    if class_num == 2:
        split_train[class_num].append([new_img, class_num])
    if class_num == 3:
        split_train[class_num].append([new_img, class_num])
    if class_num == 4:
        split_train[class_num].append([new_img, class_num])

# Testing set
split_test = [[], [], [], [], []]

for new_img, class_num in all_aug[1]:
    if class_num == 0:
        split_test[class_num].append([new_img, class_num])
    if class_num == 1:
        split_test[class_num].append([new_img, class_num])
    if class_num == 2:
        split_test[class_num].append([new_img, class_num])
    if class_num == 3:
        split_test[class_num].append([new_img, class_num])
    if class_num == 4:
        split_test[class_num].append([new_img, class_num])

# Shuffles the images within the classes
random.seed(42)
for num in range(5):
    random.shuffle(split_train[num])
    random.shuffle(split_test[num])

# Training set
smallest_train = 10000

```



```

print("Training set:")
for num in range(5):
    print(f'{classes[num]}: {len(split_train[num])}')
    if smallest_train > len(split_train[num]):
        smallest_train = len(split_train[num])

print(f'\nSmallest number of samples in training set: {smallest_train}')

# Testing set
smallest_test = 10000

print("\nTesting set:")
for num in range(5):
    print(f'{classes[num]}: {len(split_test[num])}')
    if smallest_test > len(split_test[num]):
        smallest_test = len(split_test[num])

print(f'\nSmallest number of samples in testing set: {smallest_test}')

fold_train = []
for num in range(5):
    fold_train += split_train[num][:smallest_train]
print(f'Total training set count: {len(fold_train)} samples\nEach class: {int(len(fold_train)/5)} samples')

fold_test = []
for num in range(5):
    fold_test += split_test[num][:smallest_test]
print(f'\nTotal testing set count: {len(fold_test)} samples\nEach class: {int(len(fold_test)/5)} samples')

random.seed(42)
random.shuffle(fold_train)
random.shuffle(fold_test)

```

Listing A.6: Equalize and Split the Dataset into Train and Test Sets

```

def train_model(model, criterion, optimizer, num_epochs=3):
    prev_acc = 0

    print("Current K-
Fold Cross Validation: {}".format(fold_var + 1))
    print("Previous validation accuracy: {:.5f}\n".format(prev_acc))

    for epoch in range(num_epochs):

```

```

trn_acc = 0
val_acc = 0
trn_loss = 0
val_loss = 0

print('Epoch {}/{}'.format(epoch+1, num_epochs))
print('-' * 10)

for phase in ['train', 'validation']:
    if phase == 'train':
        model.train()
    else:
        model.eval()

    running_loss = 0.0
    running_corrects = 0

    for inputs, labels in dataloaders[phase]:
        inputs = inputs.to(device)
        labels = labels.unsqueeze(1)
        labels = labels.to(device)
        outputs = model(inputs.float())
        loss = criterion(outputs, labels.float())

        if phase == "train":
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        preds = torch.sigmoid(outputs) >= 0.5
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == (labels.
data == 1))

    if phase == "train":
        epoch_loss = running_loss / len(train)
        epoch_acc = running_corrects.double() / len(tra
in)

        trn_loss = float(epoch_loss)
        trn_acc = float(epoch_acc)
    else:
        epoch_loss = running_loss / len(valid)
        epoch_acc = running_corrects.double() / len(val
id)

        val_loss = float(epoch_loss)
        val_acc = float(epoch_acc)

print("{} loss: {:.5f}, acc: {:.5f}".format(phase,

```

```

epoch_l
oss,
epoch_a
cc))

    if phase == "validation":
        if epoch_acc > prev_acc:
            save_path = '/content/drive/MyDrive/SNsaved
_model/SN_' + str(fold_var) + '.h5'
            torch.save(model.state_dict(), save_path)
            print("Epoch {}: val_accuracy improved from
 {:.5f} to {:.5f}, saving model to {}".format(str(epoch+1).zfil
l(5), prev_acc, epoch_acc, save_path))
            prev_acc = epoch_acc
        else:
            print("Epoch {}: val_accuracy did not impro
ve from {:.5f}".format(str(epoch+1).zfill(5),

                prev_acc))

    try:
        str_history = pickle.load(open('/content/drive/MyD
rive/SNsaved_model/historySN_' + str(fold_var)), "rb"))
        str_trn_loss = str_history['train loss']
        str_trn_acc = str_history['train accuracy']
        str_val_loss = str_history['validation loss']
        str_val_acc = str_history['validation accuracy']
    except Exception as e:
        str_trn_loss = []
        str_trn_acc = []
        str_val_loss = []
        str_val_acc = []
    str_trn_loss.append(trn_loss)
    str_trn_acc.append(trn_acc)
    str_val_loss.append(val_loss)
    str_val_acc.append(val_acc)

    history = {"train loss": str_trn_loss, "train accuracy"
: str_trn_acc, "validation loss": str_val_loss, "validation acc
uracy": str_val_acc}

    with open('/content/drive/MyDrive/SNsaved_model/histor
ySN_' + str(fold_var)), 'wb') as wfile:
        pickle.dump(history, wfile)

    data = pickle.load(open('/content/drive/MyDrive/SNsaved_mo
del/historySN_' + str(fold_var)), "rb"))
    return data

```

Listing A.7: Function to Train ResNeXt and SENet Models

```

inception = InceptionV3(weights = 'imagenet',
                        include_top = False,
                        input_shape = (224, 224, 3))
inception.trainable=False

add_model = Sequential()
add_model.add(inception)
add_model.add(GlobalAveragePooling2D())
add_model.add(Dropout(0.2))
add_model.add(Flatten())
add_model.add(Dense(128, activation='relu'))
add_model.add(Dense(1, activation='softmax'))

model = add_model

```

Listing A.8: Transfer Learning of Inception-V3 Model for  
Binary Classification Problem

```

inception = InceptionV3(weights = 'imagenet',
                        include_top = False,
                        input_shape = (224, 224, 3))
inception.trainable=False

add_model = Sequential()
add_model.add(inception)
add_model.add(GlobalAveragePooling2D())
add_model.add(Dropout(0.2))
add_model.add(Flatten())
add_model.add(Dense(128, activation='relu'))
add_model.add(Dense(5, activation='sigmoid'))

model = add_model

```

Listing A.9: Transfer Learning of Inception-V3 Model for  
5-class Classification Problem

```

resnext = models.resnext101_32x8d(pretrained=True).to(device)

for param in resnext.parameters():
    param.requires_grad = False

resnext.fc = nn.Sequential(
    nn.Linear(2048, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 1)).to(device)

```

Listing A.10: Transfer Learning of ResNeXt Model for  
Binary Classification Problem

```

resnext = models.resnext101_32x8d(pretrained=True).to(device)

for param in resnext.parameters():
    param.requires_grad = False

resnext.fc = nn.Sequential(
    nn.Linear(2048, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 5)).to(device)

```

Listing A.11: Transfer Learning of ResNeXt Model for  
5-class Classification Problem

```

model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')

for param in senet.parameters():
    param.requires_grad = False

senet.last_linear = nn.Sequential(
    nn.Linear(2048, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 1)).to(device)

```

Listing A.12: Transfer Learning of SENet Model for  
Binary Classification Problem

```

model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')

for param in senet.parameters():
    param.requires_grad = False

senet.last_linear = nn.Sequential(
    nn.Linear(2048, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 5)).to(device)

```

Listing A.13: Transfer Learning of SENet Model for  
5-class Classification Problem

```

save_dir = '/content/drive/MyDrive/IV3saved_model/'
filepath = save_dir + 'BIN_IV3_' + str(fold_var) + '.h5'
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monit
or='val_accuracy', verbose=1, save_best_only=True, mode='max')

```

```

model.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

history = model.fit(x_bin, y_bin, validation_split=0.2, epochs=
10, callbacks=[checkpoint])

with open('/content/drive/MyDrive/IV3saved_model/BIN_historyIV
3_' + str(fold_var)), 'wb') as wfile:
    pickle.dump(history.history, wfile)

tf.keras.backend.clear_session()

```

Listing A.14: Training the Inception-V3 Model for Binary Classification Problem

```

save_dir = '/content/drive/MyDrive/IV3saved_model/'
filepath = save_dir + 'newIV3_' + str(fold_var) + '.h5'
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath, monit
or='val_accuracy', verbose=1, save_best_only=True, mode='max')

model.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=["accuracy"])

history = model.fit(x_train, tf.one_hot(y_train, depth=5), vali
dation_data=(x_valid, tf.one_hot(y_valid, depth=5)), epochs=25,
callbacks=[checkpoint])

with open('/content/drive/MyDrive/IV3saved_model/historyIV3_'
+ str(fold_var)), 'wb') as wfile:
    pickle.dump(history.history, wfile)

tf.keras.backend.clear_session()

```

Listing A.15: Training the Inception-V3 Model for 5-class Classification Problem

```

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(resnext.fc.parameters())
history = train_model(resnext, criterion, optimizer, num_epochs
=10)

```

Listing A.16: Training the ResNeXt Model for Binary Classification Problem

```

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(resnext.fc.parameters())
history = train_model(resnext, criterion, optimizer, num_epochs
=25)

```

Listing A.17: Training the ResNeXt Model for 5-class Classification Problem

```

criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(senet.last_linear.parameters())
history = train_model(senet, criterion, optimizer, num_epochs=1
0)

```

Listing A.18: Training the SENet model for Binary Classification Problem

```

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(senet.last_linear.parameters())
history = train_model(senet, criterion, optimizer, num_epochs=2
5)

```

Listing A.19: Training the SENet model for 5-class Classification Problem

```

y_preds = model.predict(x_test)

```

Listing A.20: Testing the Inception-V3 Model for Binary and 5-class Classification

### Problems with Unseen Testing Data

```

criterion = nn.BCEWithLogitsLoss()
resnext.eval()

y_preds = []
tup_preds = []
running_loss = 0.0
running_corrects = 0

for inputs, labels in dataloaders['test']:
    inputs = inputs.to(device)
    labels = labels.unsqueeze(1)
    labels = labels.to(device)
    outputs = resnext(inputs.float())
    loss = criterion(outputs, labels.float())

    tup_preds.append(outputs)
    preds = torch.sigmoid(outputs) >= 0.5
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == (labels.data == 1))

for p in tup_preds:
    for q in p:
        pred_list = []
        for s in torch.sigmoid(q):
            pred_list.append(float(s))
        y_preds.append(pred_list)

y_preds = np.array(y_preds, dtype="float32")

```

```

loss = running_loss / len(test)
acc = running_corrects.double() / len(test)
print("Loss: {:.4f}\nAccuracy: {:.2f}%".format(loss, acc*100))

```

Listing A.21: Testing the ResNeXt Model for Binary Classification Problem

```

criterion = nn.CrossEntropyLoss()
resnext.eval()

y_preds = []
tup_preds = []
running_loss = 0.0
running_corrects = 0

for inputs, labels in dataloaders['test']:
    inputs = inputs.to(device)
    labels = labels.to(device)
    outputs = resnext(inputs.float())
    loss = criterion(outputs, labels)

    _, preds = torch.max(outputs, 1)
    tup_preds.append(outputs)
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == labels.data)

for p in tup_preds:
    for q in p:
        pred_list = []
        for s in torch.nn.functional.softmax(q, dim=-1):
            pred_list.append(float(s))
        y_preds.append(pred_list)

y_preds = np.array(y_preds, dtype="float32")

loss = running_loss / len(test)
acc = running_corrects.double() / len(test)
print("Loss: {:.2f}\nAccuracy: {:.2f}%".format(loss, acc*100))

```

Listing A.22: Testing the ResNeXt Model for 5-class Classification Problem

```

criterion = nn.BCEWithLogitsLoss()
senet.eval()

y_preds = []
tup_preds = []
running_loss = 0.0
running_corrects = 0

```



```

for inputs, labels in dataloaders['test']:
    inputs = inputs.to(device)
    labels = labels.unsqueeze(1)
    labels = labels.to(device)
    outputs = senet(inputs.float())
    loss = criterion(outputs, labels.float())

    tup_preds.append(outputs)
    preds = torch.sigmoid(outputs) >= 0.5
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == (labels.data == 1))

for p in tup_preds:
    for q in p:
        pred_list = []
        for s in torch.sigmoid(q):
            pred_list.append(float(s))
        y_preds.append(pred_list)

y_preds = np.array(y_preds, dtype="float32")

loss = running_loss / len(test)
acc = running_corrects.double() / len(test)
print("Loss: {:.4f}\nAccuracy: {:.2f}%".format(loss, acc*100))

```

Listing A.23: Testing the SENet Model for Binary Classification Problem

```

criterion = nn.CrossEntropyLoss()
senet.eval()

y_preds = []
tup_preds = []
running_loss = 0.0
running_corrects = 0

for inputs, labels in dataloaders['test']:
    inputs = inputs.to(device)
    labels = labels.to(device)
    outputs = senet(inputs.float())
    loss = criterion(outputs, labels)

    _, preds = torch.max(outputs, 1)
    tup_preds.append(outputs)
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == labels.data)

for p in tup_preds:
    for q in p:
        pred_list = []

```

```

    for s in torch.nn.functional.softmax(q, dim=-1):
        pred_list.append(float(s))
    y_preds.append(pred_list)

y_preds = np.array(y_preds, dtype="float32")

loss = running_loss / len(test)
acc = running_corrects.double() / len(test)
print("Loss: {:.4f}\nAccuracy: {:.2f}%".format(loss, acc*100))

```

Listing A.24: Testing the SENet Model for 5-class Classification Problem

```

pd.DataFrame(history).plot(title="SENet with 3 Hidden Layers: {
} Fold".format(str_fold), xlabel="epoch", ylabel="Percentage")

```

Listing A.25: Plot the Graph of Accuracy and Loss Against Number of Epochs

```

fpr = {}
tpr = {}
roc_auc = {}
thresh = {}

n_class = 5

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test, y_preds[:,i],
    pos_label=i)
    roc_auc[i] = auc(fpr[i], tpr[i])

plt.figure(figsize=(10,10))
plt.plot(fpr[0], tpr[0], linestyle='--
',color='orange', label='ALL vs Rest (Area = {1:0.6f})'.format(
i, roc_auc[0]))
plt.plot(fpr[1], tpr[1], linestyle='--
',color='green', label='AML vs Rest (Area = {1:0.6f})'.format(i
, roc_auc[1]))
plt.plot(fpr[2], tpr[2], linestyle='--
',color='blue', label='CLL vs Rest (Area = {1:0.6f})'.format(i,
roc_auc[2]))
plt.plot(fpr[3], tpr[3], linestyle='--
',color='purple', label='CML vs Rest (Area = {1:0.6f})'.format(
i, roc_auc[3]))
plt.plot(fpr[4], tpr[4], linestyle='--
',color='red', label='HLT vs Rest (Area = {1:0.6f})'.format(i,
roc_auc[4]))
plt.title('SENet with 3 Hidden Layers: {} Fold'.format(str_fold
))
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive rate')

```

```
plt.legend(loc='best')
```

Listing A.26: Plot ROC curve

```
# Create the confusion matrix
def plot_confusion_matrix(y_true, y_pred, classes=None, figsize
=(10, 10), text_size=15):
    cm = confusion_matrix(y_true, y_pred)
    cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]
    n_classes=cm.shape[0]

    fig, ax = plt.subplots(figsize=figsize)

    # Create a matrix plot
    cax = ax.matshow(cm, cmap=plt.cm.Blues)
    fig.colorbar(cax)

    # labels to be classes
    if classes:
        labels = classes
    else:
        labels = np.arange(cm.shape[0])

    # Label the axes
    ax.set(title="SENet with 3 Hidden Layers: {} Fold".format(str
_fold),
           xlabel="Predicted label",
           ylabel="True label",
           xticks=np.arange(n_classes),
           yticks=np.arange(n_classes),
           xticklabels=labels,
           yticklabels=labels)

    # Set x-axis label to bottom
    ax.xaxis.set_label_position("bottom")
    ax.xaxis.tick_bottom()

    # Adjust label size
    ax.yaxis.label.set_size(text_size)
    ax.xaxis.label.set_size(text_size)
    ax.title.set_size(text_size)

    # Set threshold for different colors
    threshold = (cm.max() + cm.min()) / 2

    # Plot the text on each cell
    for i, j in itertools.product(range(cm.shape[0]), range(cm.sh
ape[1])):
        plt.text(i, j, f"{cm[j, i]} ({cm_norm[j, i]*100:.1f}%)",
```

```

        horizontalalignment="center",
        color="white" if cm[i, j] > threshold else "black",
        size=text_size)

plot_confusion_matrix(y_test, y_preds.argmax(axis=1), classes=c
lasses, text_size=10)

```

Listing A.27: Plot the Confusion Matrix

```

totalPrecision= 0

prec = precision_score(y_test, y_preds.argmax(axis=1), average=
None)

for i in range(n_class):
    totalPrecision += prec[i]
    print("For {} Precision: {:.2f}%".format(classes[i], prec[i]*
100))
print("Macro Precision: {:.2f}%".format(totalPrecision/n_class*
100))

```

Listing A.28: Calculate Precision of Each Class

```

totalRecall = 0

rec = recall_score(y_test, y_preds.argmax(axis=1), average=None
)

for i in range(n_class):
    totalRecall += rec[i]
    print("For {} Recall: {:.2f}%".format(classes[i], rec[i]*100)
)
print("Macro Recall: {:.2f}%".format(totalRecall/n_class*100))

```

Listing A.29: Calculate Recall of Each Class

```

totalF1 = 0

f1 = f1_score(y_test, y_preds.argmax(axis=1), average=None)

for i in range(n_class):
    totalF1 += f1[i]
    print("For {} F1-score: {:.2f}%".format(classes[i], f1[i] *
100))
print("Macro F1-score: {:.2f}%".format(totalF1/n_class*100))

```

Listing A.30: Calculate F1-score of Each Class

```

model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')

for param in senet.parameters():
    param.requires_grad = False

senet.last_linear = nn.Sequential(
    nn.Linear(2048, 128),
    nn.ReLU(inplace=True),
    nn.Linear(128, 5)).to(device)

#Load model
senet.load_state_dict(torch.load('/content/drive/MyDrive/SNsave
d_model/SN_' + str(fold_var) + '.h5'))

senet.last_linear = nn.Sequential(*[senet.last_linear[x] for x
in range(len(senet.last_linear) - 1)])

for param in senet.parameters():
    param.requires_grad = False

#Extract features
cnn_features = []
cnn_labels = []

for inputs, labels in trainloaders['train']:
    inputs = inputs.to(device)
    labels = labels.to(device)
    outputs = senet(inputs.float())

    for feature in outputs:
        cnn_features.append(np.array(feature))

    for label in labels:
        cnn_labels.append(label)

#SVM classifier
svm = LinearSVC()
clf = CalibratedClassifierCV(svm)
clf.fit(cnn_features, cnn_labels)

svm_preds = []
svm_labels = []

for inputs, labels in testloaders['test']:
    inputs = inputs.to(device)

```

```

labels = labels.to(device)
cnn_output = senet(inputs.float())

predicted = clf.predict_proba(cnn_output)
for p in predicted:
    svm_preds.append(p)

```

Listing A.31: SENet + SVM Model

```

model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')

for param in senet.parameters():
    param.requires_grad = False

senet.last_linear = nn.Sequential(
    nn.Linear(2048, 1024),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(1024, 512),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(512, 128),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(128, 5)).to(device)

```

Listing A.32: SENet Model with 3 Hidden Layers

```

model_name = 'senet154'
senet = pretrainedmodels.__dict__[model_name](num_classes=1000,
pretrained='imagenet')

for param in senet.parameters():
    param.requires_grad = False

senet.last_linear = nn.Sequential(
    nn.Linear(2048, 1024),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(1024, 512),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(512, 128),
    nn.ReLU(inplace=True),
    nn.Dropout(0.5),
    nn.Linear(128, 5)).to(device)

```

Listing A.33: SENet Model with 3 Hidden Layers Plus Dropout Layers