

**DEEP FEATURE TRANSFER FROM
DEEP LEARNING MODELS INTO MACHINE LEARNING ALGORITHMS
TO CLASSIFY COVID-19 FROM CHEST X-RAY IMAGES**

M.Sc. THESIS

Ozan GÜLDALİ

Department of Mathematical Engineering

Mathematical Engineering Programme

AUGUST 2021

**DEEP FEATURE TRANSFER FROM
DEEP LEARNING MODELS INTO MACHINE LEARNING ALGORITHMS
TO CLASSIFY COVID-19 FROM CHEST X-RAY IMAGES**

M.Sc. THESIS

**Ozan GÜLDALİ
(509191238)**

Department of Mathematical Engineering

Mathematical Engineering Programme

Thesis Advisor: Assist. Prof. Dr. Gül İNAN

AUGUST 2021

**GÖĞÜS RÖNTGENİ GÖRÜNTÜLERİNDEN COVID-19 SINIFLANDIRMASI
YAPMAK AMACIYLA DERİN ÖĞRENME MODELLERİNDEN
MAKİNE ÖĞRENMESİ ALGORİTMALARINA DERİN ÖZNİTELİK AKTARIMI**

YÜKSEK LİSANS TEZİ

**Ozan GÜLDALİ
(509191238)**

Matematik Mühendisliği Anabilim Dah

Matematik Mühendisliği Programı

Tez Danışmanı: Assist. Prof. Dr. Gül İNAN

AĞUSTOS 2021

Ozan GÜLDALI, a M.Sc. student of ITU Graduate School student ID 509191238 successfully defended the thesis entitled “DEEP FEATURE TRANSFER FROM DEEP LEARNING MODELS INTO MACHINE LEARNING ALGORITHMS TO CLASSIFY COVID-19 FROM CHEST X-RAY IMAGES”, which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor : **Assist. Prof. Dr. Gül İNAN**
Istanbul Technical University

Jury Members : **Prof. Dr. Name SURNAME**
Yıldız Technical University

Prof. Dr. Name SURNAME
Boğaziçi University

Prof. Dr. Name SURNAME
Gebze Institute of Technology

Date of Submission : **22 September 2009**
Date of Defense : **21 December 2009**

To my family,

FOREWORD

I would like to express my deepest sincere gratitude to my thesis advisor Assist. Prof. Dr. Gül İNAN for supporting and encouraging me with her immense knowledge during the writing process of my thesis. From the day I enrolled in the master's program, regardless of time and place, she was with me at every step of the way and became more than an lecturer for me.

Last but not least, I would like to express my endless thanks to my beloved mother, father, Kübra Özcan, and my managers Mehmet Oğuz Bici and Kemal Uğur, who have never lost their support from me. They have never lost their faith in me and have always been full of love and understanding.

August 2021

Ozan GÜLDALİ

TABLE OF CONTENTS

| | <u>Page</u> |
|--|--------------|
| FOREWORD..... | ix |
| TABLE OF CONTENTS..... | xi |
| ABBREVIATIONS | xv |
| LIST OF TABLES | xvii |
| LIST OF FIGURES | xxi |
| SUMMARY | xxiii |
| ÖZET | xxvii |
| 1. INTRODUCTION | 1 |
| 1.1 Purpose of Thesis | 3 |
| 1.2 Literature Review | 4 |
| 1.3 Structure | 7 |
| 2. CHEST X-RAYS | 9 |
| 3. INTRODUCTION TO DEEP LEARNING | 13 |
| 3.1 The Basics of Deep Learning | 13 |
| 3.2 The Cross-Entropy Loss Function | 16 |
| 3.3 The Basics of Optimization Methods | 17 |
| 3.3.1 Stochastic Gradient Descent with Momentum Optimization Algorithm | 17 |
| 3.3.2 <i>Adaptive Moment Estimation</i> Optimization Algorithm | 17 |
| 3.3.3 Adam Weight Decay Optimization Algorithm | 18 |
| 3.4 The Basics of Convolutional Neural Networks | 18 |
| 3.4.1 Convolutional Layer | 21 |
| 3.4.2 Activation Function (Rectified Linear Units - ReLU) | 23 |
| 3.4.3 Pooling Layer | 23 |
| 3.4.4 Batch Normalization..... | 24 |
| 3.4.5 Drop-out | 24 |
| 3.4.6 Flattening | 25 |
| 3.4.7 Fully-Connected Layer..... | 25 |
| 3.5 Transfer Learning | 26 |
| 3.6 CNN Models..... | 27 |
| 3.6.1 AlexNet..... | 28 |
| 3.6.2 Residual Neural Networks..... | 28 |
| <i>ResNet-18</i> | 29 |
| <i>ResNet-34</i> | 29 |
| <i>ResNet-50</i> | 29 |
| 3.6.3 VGG | 30 |
| <i>VGG16</i> | 30 |
| <i>VGG19</i> | 31 |

| | |
|---|-----------|
| 4. INTRODUCTION TO MACHINE LEARNING..... | 33 |
| 4.1 The Basics of Machine Learning (ML) | 33 |
| 4.1.1 Supervised Learning..... | 33 |
| 4.1.2 Unsupervised Learning..... | 34 |
| 4.2 Cross-Validation (CV) | 34 |
| 4.2.1 K-Fold CV | 35 |
| 4.2.2 Leave-One-Out (LOO) | 35 |
| 4.3 Regularization..... | 36 |
| 4.3.1 L1 Regularization (LASSO)..... | 36 |
| 4.3.2 L2 Regularization (Ridge)..... | 37 |
| 4.4 ML Models | 37 |
| 4.4.1 Support-Vector Machines (SVM)..... | 37 |
| <i>Loss Functions</i> | 38 |
| <i>Penalty Terms</i> | 40 |
| <i>Kernel Trick</i> | 41 |
| 4.4.2 Logistic Regression (LR) | 41 |
| <i>Penalty Terms</i> | 42 |
| <i>Optimizers</i> | 43 |
| 4.4.3 K-Nearest Neighbor (KNN) | 43 |
| <i>Algorithm</i> | 44 |
| <i>Finding Neighborhood</i> | 45 |
| 4.4.4 Linear Discriminant Analysis (LDA) | 46 |
| <i>Derivation</i> | 46 |
| <i>Penalty Terms</i> | 48 |
| <i>Solvers</i> | 49 |
| 5. EXPERIMENTS | 51 |
| 5.1 Dataset | 51 |
| 5.2 Data Augmentation..... | 52 |
| 5.3 Training and Testing on Convolutional Neural Network Models | 53 |
| 5.4 Deep Feature Extraction | 58 |
| 5.5 Forming the Feature Maps..... | 59 |
| 5.6 Data Pre-Processing..... | 59 |
| 5.6.1 Data Standardization | 59 |
| 5.6.2 Data Normalization | 60 |
| 5.7 Hyper-Parameter Tuning | 60 |
| 5.8 Training and Testing on Machine Learning Models..... | 61 |
| 6. RESULTS | 65 |
| 6.1 Performance Measurement..... | 65 |
| 6.1.1 Confusion Matrix..... | 65 |
| 6.1.2 The Analysis of Confusion Matrix | 66 |
| 6.1.2.1 Sensitivity (True Positive Rate - TPR) | 66 |
| 6.1.2.2 Specificity (True Negative Rate - TNR) | 66 |
| 6.1.2.3 Precision (Positive Predictive Value - PPV) | 67 |
| 6.1.2.4 Accuracy (ACC) | 67 |
| 6.1.2.5 F1 Score | 67 |

| | |
|--|------------|
| 6.1.2.6 Area Under the Curve (AUC Score) | 67 |
| 6.2 Convolutional Neural Network Results..... | 67 |
| 6.3 Machine Learning Results | 77 |
| 6.3.1 Results for X_{info} | 77 |
| 6.3.2 Results for X_{cnn} | 77 |
| 6.3.3 Results for X_{all} | 85 |
| 7. CONCLUSIONS AND RECOMMENDATIONS..... | 93 |
| 7.1 Future Work..... | 95 |
| REFERENCES..... | 97 |
| APPENDICES | 101 |
| APPENDIX A.1 ReadMe File..... | 103 |
| APPENDIX A.2 Python Notebook File | 108 |
| CURRICULUM VITAE | 111 |

ABBREVIATIONS

| | |
|-------------------|---|
| ACC | : Accuracy |
| Adam | : Adaptive Moment Estimation |
| AdamW | : Adam Weight Decay |
| AP | : Anterior-Posterior |
| AUC | : Area Under the Curve |
| AI | : Artificial Intelligence |
| CNN | : Convolution Neural Network |
| CV | : Cross-Validation |
| DL | : Deep Learning |
| DTL | : Deep Transfer Learning |
| FN | : False Negative |
| FP | : False Positive |
| KNN | : K-Nearest Neighbor |
| LOO | : Leave-One-Out |
| LDA | : Linear Discriminant Analysis |
| LR | : Logistic Regression |
| ML | : Machine Learning |
| NN | : Neural Network |
| PCR | : Polymerase Chain Reaction |
| PPV | : Positive Prediction Value |
| PA | : Posterior-Anterior |
| ResNet | : Residual neural network |
| SARS-CoV-2 | : Severe Acute Respiratory Syndrome-related Coronavirus |
| SGD | : Stochastic Gradient Descent |
| SVM | : Support Vector Machines |
| TL | : Transfer Learning |
| TN | : True Negative |
| TPR | : True Negative Rate |
| TP | : True Positive |
| TNR | : True Positive Rate |
| WHO | : World Health Organization |

LIST OF TABLES

| | <u>Page</u> |
|--|-------------|
| Table 1.1 : Reviewed works in the literature and their stated results..... | 6 |
| Table 4.1 : Logistic Regression optimization problem solvers..... | 43 |
| Table 5.1 : The detailed number of data..... | 53 |
| Table 5.2 : CNN models final hyper-parameters..... | 55 |
| Table 5.3 : SVM algorithm hyper-parameters to tune..... | 62 |
| Table 5.4 : LR algorithm hyper-parameters to tune..... | 62 |
| Table 5.5 : KNN algorithm hyper-parameters to tune..... | 62 |
| Table 5.6 : LDA algorithm hyper-parameters to tune..... | 63 |
| Table 6.1 : Sample Confusion Matrix..... | 65 |
| Table 6.2 : CNN model comparison..... | 75 |
| Table 6.3 : The confusion matrix of AlexNet model result with Adam optimizer. | 76 |
| Table 6.4 : The confusion matrix of ResNet-18 model result with AdamW optimizer..... | 76 |
| Table 6.5 : The confusion matrix of ResNet-34 model result with AdamW optimizer..... | 76 |
| Table 6.6 : The confusion matrix of ResNet-50 model result with Adam optimizer..... | 76 |
| Table 6.7 : The confusion matrix of VGG16 model result with AdamW optimizer..... | 76 |
| Table 6.8 : The confusion matrix of VGG19 model result with SGD Momentum optimizer..... | 76 |
| Table 6.9 : The result of demographic information experiments with the Python system seed as 4..... | 77 |
| Table 6.10 : KNN algorithm hyper-parameters for X_{info} | 77 |
| Table 6.11 : Confusion matrices for KNN experiments on demographic information..... | 78 |
| Table 6.12 : The result of experiments on X_{cnn} with the Python system seed as 4. | 79 |
| Table 6.13 : SVM algorithm regularized by Ridge hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.14 : SVM algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.15 : LR algorithm hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.16 : LR algorithm regularized by Ridge hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.17 : LR algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.18 : KNN algorithm hyper-parameters for $X_{cnn_ResNet50}$ | 80 |
| Table 6.19 : LDA algorithm hyper-parameters for $X_{cnn_ResNet50}$ | 81 |

| | | |
|--------------------|---|----|
| Table 6.20: | LDA algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$ | 81 |
| Table 6.21: | The confusion matrices and results obtained by SVM algorithm regularized with Ridge for $X_{cnn_ResNet50}$ | 82 |
| Table 6.22: | The confusion matrices and results obtained by SVM algorithm regularized with Lasso for $X_{cnn_ResNet50}$ | 82 |
| Table 6.23: | The confusion matrices and results obtained by LR algorithm for $X_{cnn_ResNet50}$ | 82 |
| Table 6.24: | The confusion matrices and results obtained by LR algorithm regularized with Ridge for $X_{cnn_ResNet50}$ | 83 |
| Table 6.25: | The confusion matrices and results obtained by LR algorithm regularized with Lasso for $X_{cnn_ResNet50}$ | 83 |
| Table 6.26: | The confusion matrices and results obtained by KNN algorithm for $X_{cnn_ResNet50}$ | 83 |
| Table 6.27: | The confusion matrices and results obtained by LDA algorithm for $X_{cnn_ResNet50}$ | 83 |
| Table 6.28: | The confusion matrices and results obtained by LDA algorithm regularized with Lasso for $X_{cnn_ResNet50}$ | 84 |
| Table 6.29: | The result of experiments on X_{all} with the Python system seed as 4.. | 86 |
| Table 6.30: | SVM algorithm regularized by Ridge hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.31: | SVM algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.32: | LR algorithm hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.33: | LR algorithm regularized by Ridge hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.34: | LR algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.35: | KNN algorithm hyper-parameters for $X_{all_ResNet50}$ | 87 |
| Table 6.36: | LDA algorithm hyper-parameters for $X_{all_ResNet50}$ | 88 |
| Table 6.37: | LDA algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$ | 88 |
| Table 6.38: | The confusion matrices and results obtained by SVM algorithm regularized with Ridge for $X_{all_ResNet50}$ | 89 |
| Table 6.39: | The confusion matrices and results obtained by SVM algorithm regularized with Lasso for $X_{all_ResNet50}$ | 89 |
| Table 6.40: | The confusion matrices and results obtained by LR algorithm for $X_{all_ResNet50}$ | 89 |
| Table 6.41: | The confusion matrices and results obtained by LR algorithm regularized with Ridge for $X_{all_ResNet50}$ | 90 |
| Table 6.42: | The confusion matrices and results obtained by LR algorithm regularized with Lasso for $X_{all_ResNet50}$ | 90 |
| Table 6.43: | The confusion matrices and results obtained by KNN algorithm for $X_{all_ResNet50}$ | 90 |
| Table 6.44: | The confusion matrices and results obtained by LDA algorithm for $X_{all_ResNet50}$ | 90 |

Table 6.45: The confusion matrices and results obtained by LDA algorithm regularized with Lasso for $X_{all_ResNet50}$ 91

LIST OF FIGURES

| | <u>Page</u> |
|---|-------------|
| Figure 2.1 : Hand mit Ringen (Hand with Rings): print of Wilhelm Röntgen's first medical X-ray of his wife's hand [1]. .. | 10 |
| Figure 2.2 : Examples for COVID-19 chest X-rays filmed with PA, AP, AP-Supine, and Lateral projections, respectively. | 11 |
| Figure 3.1 : Sample Neuron. | 13 |
| Figure 3.2 : Illustration of under-fitting, well-fitting, and over-fitting of models to the data. | 16 |
| Figure 3.3 : A pseudo-code for SGD with momentum optimization [2]. .. | 17 |
| Figure 3.4 : Pseudo-codes for Adam and AdamW optimization algorithms [2]. . | 18 |
| Figure 3.5 : An example for a simple CNN architecture. | 19 |
| Figure 3.6 : An example for usage of directional derivatives and chain rule..... | 20 |
| Figure 3.7 : Activation map construction with a filter window size of 4×4 and stride value of 2..... | 23 |
| Figure 3.8 : Maximum pooling operation sample with 2×2 window and stride as 2. | 24 |
| Figure 3.9 : (a) A standard neural network, (b) A neural network after applying dropout [3]. | 25 |
| Figure 3.10 : The usage of full-connected layer for a binary classifier..... | 25 |
| Figure 3.11 : Different fine-tuning strategies on pre-trained model. | 26 |
| Figure 3.12 : Deep feature extraction from CNN models and using them in machine learning models [4]..... | 27 |
| Figure 3.13 : Illustration of AlexNet architecture with a size of $224 \times 224 \times 3$ input image [5]..... | 28 |
| Figure 3.14 : An illustration for residual block. Here, $a^{[l]}$ is the starting activation layer and $a^{[l+2]}$ is the fed activation layer. | 29 |
| Figure 3.15 : The ResNet architectures [6]. | 30 |
| Figure 3.16 : An illustration of VGG architectures..... | 31 |
| Figure 4.1 : Sample Machine Learning Road Map..... | 33 |
| Figure 4.2 : (a) Clustering Problem, (b) Classification Problem, and (c) Regression Problem [7]..... | 34 |
| Figure 4.3 : K-Fold Cross-Validation..... | 35 |
| Figure 4.4 : A Hyper-plane and its margins for an SVM to a binary classification problem. | 38 |
| Figure 4.5 : Logistic sigmoid function with different scaling w values. | 42 |
| Figure 4.6 : KNN example illustration to detect the class for the green sample in the middle. | 45 |
| Figure 4.7 : LDA process for fixed and varying covariances..... | 48 |
| Figure 5.1 : Dataset Samples..... | 52 |

| | |
|---|----|
| Figure 5.2 : Image Transformation Steps - Test COVID-19 Sample..... | 52 |
| Figure 5.3 : Original Sample and Augmentation..... | 54 |
| Figure 5.4 : The sample visualization representing one iteration on ResNet-50 architecture and class probability result for COVID-19 labeled data from our dataset. The higher resolution version of image is available at https://github.com/ozanguldali/modelWithLASSO/blob/master/figures/resnet50_visual.png | 57 |
| Figure 5.5 : Modified classification blocks where bold red windowed fully-connected layers denotes the deep feature extraction state..... | 58 |
| Figure 5.6 : Feature maps where $d_{cnn} = 1000$, $d_{info} = 2$, and $d_{all} = d_{cnn} + d_{info} = 1002$ | 59 |
| Figure 6.1 : AlexNet model accuracy and loss curves for three different optimizers on train and validation processes..... | 69 |
| Figure 6.2 : ResNet-18 model accuracy and loss curves for three different optimizers on train and validation processes..... | 70 |
| Figure 6.3 : ResNet-34 model accuracy and loss curves for three different optimizers on train and validation processes..... | 71 |
| Figure 6.4 : ResNet-50 model accuracy and loss curves for three different optimizers on train and validation processes..... | 72 |
| Figure 6.5 : VGG16 model accuracy and loss curves for three different optimizers on train and validation processes..... | 73 |
| Figure 6.6 : VGG19 model accuracy and loss curves for three different optimizers on train and validation processes..... | 74 |

**DEEP FEATURE TRANSFER FROM
DEEP LEARNING MODELS INTO MACHINE LEARNING ALGORITHMS
TO CLASSIFY COVID-19 FROM CHEST X-RAY IMAGES**

SUMMARY

Coronavirus disease 2019 (COVID-19) is a contagious disease caused by SARS-CoV-2. It was first reported on December 2019 in Wuhan, China, and declared as a pandemic on 11 March, 2020. Even though the disease is a severe acute respiratory illness, it effects various organs and causes several symptoms such as fever, dry cough, tiredness, the loss of taste or smell, diarrhoea, headache, aches and pains, sore throat, and conjunctivitis. As of the end of April 2021, over 145 million people has been infected and more than 3 million people died because of COVID-19. That is why, one of the most important issues is the diagnosis of COVID-19. Although the most basic method to diagnose COVID-19 is Polymerase Chain Reaction (PCR) test, different techniques have been being experimented and developed. Since COVID-19 has a huge effect on lungs, diagnosis methods based on lung characteristics and images are emphasized. However, there are various illnesses affecting lungs. Thus, it has been an important challenge and problem to find a procedure to classify COVID-19 having high success rate.

In this thesis, we suggest deep feature transform method from deep learning models to machine learning algorithms to classify patients COVID-19 infected via chest X-Rays. In addition to image data, we also use the demographic information of patients during machine learning process to contribute the information coming from deep features. Chapter 1 includes the introduction about our problem, the purpose of this study, the related literature search and the structure of this thesis. The basic information about our image data, chest X-Rays, are given in Chapter 2.

The problem we focus on is the binary classification between COVID-19 patients and other people. In order to solve this problem, we used dataset containing 131 COVID-19 and 123 non-COVID-19 labeled data. Then, we divided the data set into train and test sets so that 80% of the total data is in the train set, and augmented the train set with horizontal flip, vertical flip, 90 degrees of rotation, 180 degrees of rotation and 270 degrees of rotation. Thus, at the end, we yielded 630 COVID-19 and 588 non-COVID-19 labeled data in train set, and 26 COVID-19 and 25 non-COVID-19 labeled data in test set. The augmented data only used on CNN experiments. At the begin of Chapter 5, the dataset and augmentation technique was detailed.

The deep learning models we used are Convolutional Neural Network (CNN) models such that AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG16 and VGG19. We particularly experimented three different optimization methods for each CNN models such that SGD with momentum, Adam and Adam with decoupled weight decay. The objective loss function to minimize was common for each model as cross-entropy loss function. Each image sample was resized to 227 x 227, center cropped, converted to gray-scale and normalized. Chapter 3 consists of the introduction to deep learning, basic information about CNNs and how to perform transfer learning. Two types of

transfer learning were used in this study, which are transferring pre-trained model weights into CNN models and transferring deep features extracted from CNN models into machine learning algorithms. Pre-trained CNN models are the models that previously trained on ImageNet dataset on the record, and we performed re-train after initializing the models with these recorded weights. Deep feature transfer learning is extracting the features of CNN model from the fully-connected block of model, and using it as feature matrix in another artificial intelligence technique such as ML algorithms.

The machine learning algorithms we used are supervised learning algorithms such that Support Vector Machines (SVM), Logistic Regression (LR), K-Nearest Neighbor (KNN) and Linear Discriminant Analysis (LDA). We experimented different regularization techniques, which are Lasso known as L1 norm and Ridge known as L2 norm, on stated ML algorithms. Chapter 4 consists of the introduction to machine learning, basic information about algorithms and regularizers, and cross-validation technique. We performed 10-Fold cross-validation on train set to obtain the generalized hyper-parameter choices besides hyper-parameters specific to our initially split test set. The algorithms and experiments were applied to the feature set of demographic information, the deep transferred feature set, and the combination of transferred features and demographic information separately. The demographic information feature matrix clearly consists of two feature columns such that age and sex information. The length of transferred deep features for each sample is thousand. Hence, the combined feature matrix contains thousand two columns.

All experiments for CNN and ML are detailed under Chapter 5, including data pre-processing and hyper-parameter tuning techniques for ML specifically. Grid search was used to find optimal parameters for each feature matrix and algorithm. The source code for experiments is mainly in Python programming language, and a small part is in R programming language. The CNN models were applied using PyTorch library in Python, and the ML algorithms were applied using Sklearn library in Python. Only regularized LDA algorithm was coded in R programming language using TULIP library. We performed our CNN experiments on GPU to have faster and parallel processes. Since we did not have an opportunity to reach a physical computer including GPU that we can use during our experiments, we performed the experiments on the Google Colaboratory platform. It is a partially-free platform for Gmail users to implement CUDA to use its provided GPU. After collecting CNN results and *.pth files containing the best model weights, the ML experiments were performed locally on CPU.

We explained the performance measurement techniques in Chapter 6 together with experiment results for CNN and ML processes. The best result was achieved by using ResNet-50 model with Adam optimizer. The metrics on this result are 92.16%, 0.9216, 0.9215, 0.9216, 0.9216, 0.9215 for the accuracy, sensitivity, specificity, precision, F1 score and AUC score respectively. Since we experimented obtaining optimal hyper-parameters for both generalized and specific to our test data, the results for both were reported too. For the feature matrix of demographic information, the best results for both generalized and chosen test set hyper-parameters are the same, and achieved with KNN algorithm. The metrics on this result are the accuracy of 56.86%, the sensitivity of 0.56863, the specificity of 0.58368 the precision of 0.6863, the F1 score of 0.49545, and the AUC score of 0.5745. For the deep feature matrix obtained

from ResNet-50 model weights, SVM with Ridge regularizer, LR, LR with Ridge, and KNN algorithms had the same results according to generalized hyper-parameters. The metrics on this results are the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 respectively. Finally, for the combined feature matrix of demographic information and extracted deep features obtained from ResNet-50 model weights, SVM with Ridge regularizer, LR, LR with Ridge, and KNN algorithms had the same results as well according to generalized hyper-parameters. The metrics on this results are the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 respectively.

In conclusion, according to stated results, we yielded an improvement of using regularization with linear discriminant analysis and Lasso regularizer. We did not have an improvement by combining demographic information with deep features. However, we anticipate an improvement with this image and non-image data combining technique by using more data samples and more information about patients such as doctors report, tobacco product use, associated genetic diseases, respiratory test information, etc. Finally, even though we could not see an improvement from CNN testing results to ML testing results in terms of the accuracy, sensitivity and F1 score, the specificity and precision improved. As we discussed in Chapter 7, a dataset with more samples and these samples inspected by subject matter experts, such as specialist radiologists for our X-Rays, would allow the study to have better metric results and better comparison opportunities between experimental phases.

Keywords: COVID-19, Chest X-Ray, Data augmentation, Binary classification, Demographic information, Deep learning, Convolutional Neural Networks, Pre-trained CNN models, Transfer learning, Deep feature extraction, Deep feature transfer, Machine learning, Supervised learning, Regularization, Lasso, Ridge, Grid search.

GÖĞÜS RÖNTGENİ GÖRÜNTÜLERİNDEN COVID-19 SINIFLANDIRMASI YAPMAK AMACIYLA DERİN ÖĞRENME MODELLERİNDEN MAKİNE ÖĞRENMESİ ALGORİTMALARINA DERİN ÖZNİTELİK AKTARIMI

ÖZET

Koronavirüs hastalığı 2019 (COVID-19), SARS-CoV-2'nin neden olduğu bulaşıcı bir hastalıktır. İlk vakalar Aralık 2019'da Çin'in Wuhan kentinden bildirilmiş olup, 11 Mart 2020'de dünya genelini saran bir pandemi olarak ilan edilmiştir. Hastalık şiddetli bir akut solunum yolu hastalığı olmasına rağmen, çeşitli organları etkiler ve ateş, kuru öksürük, yorgunluk başta olmak üzere, tat veya koku kaybı, ishal, baş ağrısı, ağrı ve sızılar, boğaz ağrısı ve konjonktivit gibi çeşitli semptomlara neden olur. Nisan 2021 sonu itibarıyle 145 milyondan fazla insan COVID-19 hastalığına yakalanmış ve 3 milyondan fazla insan hayatını kaybetmiştir. Bu nedenle en önemli konulardan biri COVID-19'un hızlı ve erken tanısıdır. COVID-19'u teşhis etmenin en temel yöntemi Polimeraz Zincir Reaksiyonu (PCR) testi olsa da farklı teknikler denenmekte ve geliştirilmektedir. COVID-19'un akciğerler üzerinde bıraktığı etki çok büyük olduğu için, hastanın akciğerindeki duruma ve akciğer görüntülerine dayalı tanı yöntemleri üzerinde durulmaktadır. Ancak akciğerleri etkileyen çeşitli hastalıklar da vardır. Bu nedenle, COVID-19'u sınıflandırmak için yüksek başarı oranına sahip bir yöntem bulmak önemli bir zorluk ve problem haline gelmiştir.

Bu tezde, göğüs röntgen görüntüleri aracılığıyla COVID-19 hastalığına sahip hastaları sınıflandırmak için, derin öğrenme modellerinden makine öğrenmesi algoritmalarına derin öznitelik aktarımı yöntemini önermekteyiz. Görüntü verilerine ek olarak, derin özniteliklerden gelen bilgileri desteklemek için, makine öğrenimi sürecinde hastaların demografik bilgilerini de kullanıyoruz. Bölüm 1, problemimiz hakkında giriş, bu çalışmanın amacı, ilgili literatür taraması ve bu tezin yapısını içermektedir. Görüntü verilerimiz olan göğüs röntgenleri ile ilgili temel bilgiler Bölüm 2'de verilmiştir.

Odaklandığımız sorun, COVID-19 hastaları ve diğer insanlar arasındaki ikili sınıflandırmadır. Bu sorunu çözmek için 131 COVID-19 ve 123 COVID-19 olmayan etiketli veri içeren veri kümesi kullandık. Daha sonra veri kümesini, toplam verinin %80'i öğrenme kümesinde olacak şekilde, öğrenme ve test kümelerine böldük. Ayrıca öğrenme kümesi verilerine yatay çevirme, dikey çevirme, 90 derece döndürme, 180 derece döndürme ve 270 derece döndürme ile çoğaltma uyguladık. Böylece öğrenme kümesinde 630 COVID-19 ve 588 COVID-19 olmayan etiketli veri, test setinde 26 COVID-19 ve 25 COVID-19 olmayan etiketli veri elde ettik. Belirtilen çoğaltılma işlemi sadece Evrişimsel Sinir Ağrı deneyleri sırasında kullanılmıştır. Bölüm 5'in başlarında, veri seti ve çoğaltma işlemi tekniği detaylandırılmıştır.

Kullandığımız derin öğrenme modelleri AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG16 ve VGG19 Evrişimsel Sinir Ağrı (ESA) modelleridir. Her ESA modeli için özellikle üç farklı optimizasyon yöntemi uyguladık, ve bunlar SGD momentum, Adam ve ayırtırılmış ağırlık düşüşüne sahip Adam'dır. Her model için ortak olarak kayıp fonksiyonunu çözebilmek adına cross-entropy kayıp fonksiyonunu kullandık.

Her bir görüntü 227 x 227 olarak yeniden boyutlandırıldı, merkezsel olarak kırıldı, gri tonlamaya dönüştürüldü ve normalleştirildi. Bölüm 3, derin öğrenmeye giriş, ESA'lar hakkında temel bilgiler ve transfer öğreniminin nasıl gerçekleştirileceğinden oluşmaktadır. Bu çalışmada, önceden eğitilmiş modellerin ağırlıklarının mevcut ESA modellerine aktarılması, ve ESA modellerinden çıkarılan derin özniteliklerin makine öğrenimi algoritmalarına aktarılması olarak iki tür aktarımı öğrenme kullanılmıştır. Ön-eğitimli ESA modelleri, resmi olarak ImageNet veri kümesi üzerinde daha önceden eğitilmiş modeller olup, kaydedilen bu ağırlıklarla modeller başlatılmış ve sonrasında yeniden öğrenim gerçekleştirilmiştir. Derin öznitelik aktarımı öğrenme ise, ESA modelinin özelliklerini tam bağlı model bloğundan çıkartarak makine öğrenmesi algoritmaları gibi başka bir yapay zeka tekniğinde öznitelik matrisi olarak kullanmaktadır.

Kullandığımız makine öğrenmesi (MÖ) algoritmaları, Destek Vektör Makineleri (SVM), Lojistik Regresyon (LR), K-En Yakın Komşu (KNN) ve Doğrusal Ayırma Analizi (LDA) olan gözetimli öğrenme algoritmalarıdır. Belirtilen makine öğrenmesi algoritmaları üzerinde L1 normu olarak bilinen Lasso ve L2 normu olarak bilinen Ridge olarak farklı düzenleme tekniklerini uyguladık. 4. Bölüm, makine öğrenmesine giriş, makine öğrenmesi algoritmaları ve düzenleyiciler hakkında temel bilgiler ve çapraz doğrulama tekniğini içermektedir. Başlangıçtaki bölünmüş test kümemez özgür hiper parametrelerin yanı sıra, genelleştirilmiş hiper parametre seçeneklerini elde etmek için öğrenme kümesi üzerinde 10-Katlı çapraz doğrulama gerçekleştirdik. Algoritmalar ve tüm deneyler, demografik bilgilerden oluşan öznitelik kümесine, aktarılmış derin özniteliklerden oluşan öznitelik matrisine ve aktarılan derin öznitelikler ile demografik bilgilerin birleşiminden oluşan öznitelik matrisine ayrı ayrı uygulanmıştır. Demografik bilgi özellik matrisi, bilindiği üzere yaşı ve cinsiyet bilgisi olmak üzere iki özellik sütunundan oluşmaktadır. Her örnek için aktarılan derin özniteliklerin uzunluğu bindir. Dolayısıyla, birleşik öznitelik matrisi bin iki sütun içermektedir.

ESA ve MÖ için tüm deneyler, MÖ için veri ön işleme ve hiper parametre seçimi teknikleri dahil olmak üzere, Bölüm 5'te ayrıntılı olarak açıklanmıştır. Her bir öznitelik matrisi ve algoritma için en uygun parametreleri bulmak amacıyla Izgara araması yöntemi kullandık. Deneylerin kaynak kodu ağırlıklı olarak Python programlama dilinde, küçük bir kısmı ise R programlama dilinde yazılmıştır. ESA modelleri Python'da PyTorch kütüphanesi kullanılarak, MÖ algoritmaları ise Python'da Sklearn kütüphanesi kullanılarak uygulanmıştır. Yalnızca düzenlilikmiş LDA algoritması, TULIP kütüphanesi kullanılarak R programlama dilinde kodlanmıştır. Daha hızlı ve paralelleştirilmiş işlemler için ESA deneylerimizi GPU üzerinde gerçekleştirdik. Deneylerimiz sırasında kullanabileceğimiz GPU içeren fiziksel bir bilgisayara sahip olma fırsatımız olmadığından, deneyleri Google Colaboratory platformunda gerçekleştirdik. Google Colaboratory, Gmail kullanıcılarına, içeriği olduğu GPU'yu CUDA eklentisi teknolojisi ile kullanma imkanı sağlayan kısmen ücretsiz bir platformdur. ESA sonuçlarını ve en iyi model ağırlıklarını içeren *.pth dosyalarını toplandıktan sonra, MÖ deneyleri lokal olarak CPU üzerinde gerçekleştirilmiştir.

Bölüm 6'da performans ölçüm tekniklerini ESA ve MÖ süreçleri için deney sonuçlarıyla birlikte açıkladık. En iyi sonuca Adam optimizasyon yöntemi ile ResNet-50 modeli kullanılarak ulaşılmıştır. Bu sonuca ilişkin metrikler, doğruluk,

duyarlılık, özgüllük, kesinlik, F1 skoru ve AUC skoru için sırasıyla %92.16, 0.9216, 0.9215, 0.9216, 0.9216, 0.9215'tir. Hem genelleştirilmiş hem de test verilerimize özel optimum hiper parametreleri elde etmeyi amaçladığımızdan, her ikisi için de sonuçlar rapor edilmiştir. Demografik bilgilerin öznitelik matrisi için hem genelleştirilmiş hem de seçilmiş test kümesi hiper parametreleri için en iyi sonuçlar aynıdır, ve KNN algoritması ile elde edilir. Bu sonuca ilişkin metrikler, %56.86 doğruluk, 0.56863 duyarlılık, 0.58368 özgüllük, 0.6863 kesinlik, 0.49545 F1 skoru ve 0.5745 AUC skorudur. ResNet-50 model ağırlıklarından elde edilen derin öznitelik matrisi için, Ridge düzenleyicili SVM, Ridge düzenleyicili LR, LR ve KNN algoritmaları genelleştirilmiş hiper parametrelere göre aynı sonuçları vermiştir. Bu sonuçlara ilişkin metrikler sırasıyla %92.16, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 olmak üzere doğruluk, duyarlılık, özgüllük, kesinlik, F1 skoru ve AUC skorudur. Son olarak, demografik bilgiler ve ResNet-50 model ağırlıklarından elde edilerek çıkarılan derin özniteliklerin birleştirilmesiyle oluşan öznitelik matrisi için, Ridge düzenleyicili SVM, Ridge düzenleyicili LR, LR ve KNN algoritmaları da genelleştirilmiş hiper parametrelere göre aynı sonuçları vermiştir. Bu sonuçlara ilişkin metrikler sırasıyla %92.16, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 olmak üzere doğruluk, duyarlılık, özgüllük, kesinlik, F1 skoru ve AUC skorudur.

En nihayetinde, belirtilen sonuçlara bakılacak olursa, doğrusal ayırma analizi ve Lasso düzenleyicisini kullanarak, düzenleyicinin kullanımının bir gelişme sağlayabilmiş olduğunu gördük. Demografik bilgileri derin özniteliklerle birleştirerek bir iyileştirme elde edemedik. Ancak, sayı olarak daha fazla veri ve doktor raporu, tütün ürünü kullanımı, ilişkili genetik hastalık, solunum testi vb. gibi daha fazla hasta bilgileri kullanıldığı takdirde, görüntü ve görüntü olmayan bilgilerin birleştirilmesi tekniğinin gelişme sağlayacığını ön görüyoruz. ESA test sonuçlarından MÖ test sonuçlarına geçişte, doğruluk, duyarlılık ve F1 skoru açısından bir gelişme göremesek de, özgüllük ve kesinlik metriklerinde bir artış yakaladık. Bölüm 7'de söz ettigimiz gibi, daha fazla numune içeren bir veri kümesi kullanılır ve bu numuneler, bizim kullandığımız röntgenler için uzman radyologlardan yardım alınabilecek olması gibi, konu uzmanları tarafından incelenerek seçilirse, bu durum çalışmanın daha iyi metrik sonuçlarına ve deneysel aşamalar arasında daha iyi karşılaştırma fırsatlarına sahip olmasını sağlayacaktır.

Anahtar Kelimeler: COVID-19, Göğüs Röntgeni, Veri çoğaltma, İkili sınıflandırma, Demografik bilgi, Derin öğrenme, Evrişimsel Sinir Ağları, Ön-eğitimli ESA modelleri, Öğrenme aktarımı, Derin öznitelik çıkarma, Derin öznitelik aktarımı, Makine öğrenmesi, Gözetimli öğrenme, Düzenleme, Lasso, Ridge, Izgara araması.

1. INTRODUCTION

Coronaviruses are a family of RNA related viruses causing respiratory diseases in animals and humans. The first known disease caused by this family of viruses was reported in chickens in late 1920s and then coronaviruses in humans was detected in 1960s. This family of viruses was accepted as a new group of viruses and named as coronavirus in 1968 by scientists due to their the distinctive morphological appearance. The first two human coronaviruses discovered were named as coronavirus 229E and coronavirus OC43, respectively [8]. In the following years, different types of coronaviruses in humans have been discovered including SARS-CoV in 2003, HCoV NL63 in 2003, HCoV HKU1 in 2004, MERS-CoV in 2013. Most of these coronaviruses cause serious respiratory infections in humans which may also result in death.

In December 2019 a novel coronavirus was identified in Wuhan city of China and spread across the country in a short time. Transmission of the virus from human to human was confirmed by the Health Ministry of China in early January 2020 and then the first quarantine was applied by the Chinese Government in late January 2020. On February 2020, the World Health Organization (WHO) officially named this new coronavirus as severe acute respiratory syndrome-related coronavirus, or shortly, SARS-CoV-2. The disease which SARS-CoV-2 causes is called as Coronavirus disease 2019 (COVID-19) [9]. On 11 March, 2020, WHO declared a global pandemic due to the worldwide spread of the COVID-19 [10].

As of June 15, 2021, COVID-19 has caused more than 176 million people to be infected and more than 3.8 million deaths worldwide, and continues to spread globally [11]. The most common symptoms of COVID-19 is known as fever, dry cough and tiredness. The loss of taste or smell, diarrhoea, headache, aches and pains, sore throat, and conjunctivitis are also agreed to be strongest symptoms. The infected people may begin to show at least one of these symptoms within an average of 5-6 days; however, this period may take up to 14 days [12]. Since these symptoms are very common to

many other diseases as well, it is very important to find highly accurate distinctive methods to detect COVID-19.

Early and rapid diagnosis, as in all diseases, plays a very important role in the treatment of the COVID-19. For that reason, WHO published protocols on diagnostic detection of COVID-19 on 13 and 17 January, 2020. Polymerase Chain Reaction (PCR) test [13] is the most basic method of COVID-19 diagnosis; however, the sensitivity is not high enough for the low viral load. Moreover, laboratory errors may be present in test samples. Because of it, all symptoms, test results, and other available reports must be considered in conjunction in diagnostic process of COVID-19.

COVID-19, as a respiratory disease, also plays a major role in the increase of pneumonia. Therefore, the diseases most confused with COVID-19 are those that cause pneumonia zaturre bir hastalik, zaturreye sebep olan hastalıklar covid 19 ile karıştırılıyor diyor, burda bir mantık hatası var gibi. One way to distinguish them is to examine the most damaged organ, i.e., lungs. Chest X-rays are one of the fastest and the most accessible diagnostic tools to make observations on human lungs. However, the diagnosis of pneumonia from chest X-rays may be a difficult task even for expert radiologists. The most important part of detecting COVID-19 by chest X-rays is to distinguish the cause of pneumonia [14] (bu cümle kendinden bir önce gelen ve kendinden bir sonra gelen cümleyle anlam bütünlüğü taşımıyor, istersen konuyu bağla veya cümleyi çıkar, referansı var, cümleyi çıkarırsan referansları kontrol et). Therefore, computerized support systems are very much needed to help radiologists to diagnose pneumonia from chest X-ray images. (Zaturreden bahsедiyorsun, tekrar Covid-19'a donup, Covid-19 un da teshisi icin de boyle aralar gereklı demelisin burada)

Thanks to the publicly available high volumes of image data sets, significant progresses have been made in artificial intelligence-based computer vision problems. It has been observed that deep learning algorithms give better results compared to manual diagnosis in problems such as object recognition, perception, and segmentation (it has been observed diyince referans vermek gerekiyor diger turlu desteksiz bir cümle olarak kaliyor). On the other hand, machine learning has various strong classification algorithms on categorical data including either small or large information.(Bu cümle kendinden bir oncekine bir sey katmiyor, konuyu baglamiyor bir yere ama uygun bir sey demek lazim). Bir de konuyu teze hemen asagidaki paragrafa baglamak gerekiyor.

Soyle bir sey denilebilir. The recent advances in deep learning and machine learning algorithms also open a venue for proposing alternative approaches or developing new algorithms for detection of Covid-19 which can be served as a smart-assistance tool for medical doctors and radiologists whose workload has always been very heavy during the pandemic.

1.1 Purpose of Thesis

The scope of this study is two fold: 1) to show the capability of joint use of deep learning algorithms and machine learning algorithms for, and 2) to apply this knowledge onto an on-going global public health problem. In this sense, we first used deep neural networks with several architectures such as AlexNet, RestNet-18, ResNet-34, Resnet-50, VGG16, and VGG19 for extracting deep features from Covid-19 chest X-rays, and then we feeded the deep features coupled with demographic information of patients into machine learning algorithms such as Support Vector Machines, Logistic Regression, K-Nearest Neighborhoods, and Linear Discriminant Analysis along with their regularized versions (if exists) for classification Covid-19 status.

For this purpose, Covid-19 chest X-ray images publicly shared on GitHub platform were used. In this dataset, there are data with and without demographic information, non-diseased, virus-induced pneumonia, bacterial-induced pneumonia, fungal-induced pneumonia, and lipoinduced pneumonia. Moreover, samples with pneumonia finding have detailed causes such as COVID-19, Influenza, Escherichia coli, Aspergillosis spp., etc. Hence, data extraction on this dataset was applied to obtain two classes as non-COVID-19-diseased and COVID-19-diseased of data having demographic information.

By applying the stated scope to the mentioned dataset, the main aim of this thesis is to explain how to prepare data for study, how to perform deep learning algorithms, machine learning algorithms and transfer learning from deep learning to machine learning, how to use demographic information together with chest X-Ray image data, and to show how demographic information affects and what are the effects of regularization on machine learning algorithms by various performance measures.

1.2 Literature Review

With the increasing information on COVID-19 and the formation of data that can be used during this period, artificial intelligence researchers had started to work on this disease. Since both the effectiveness and the limitations of machine learning and deep learning, which are the sub-branches of artificial intelligence, had been already experienced and known by the previous numerous studies, researchers could start new studies without wasting time and achieved satisfying results rapidly.

Ali Abbasian Ardakani et al. [15] worked on chest computed tomography (CT) images to detect whether a person has COVID-19 disease or not. They studied on 10 different convolutional neural network (CNN) models, and compared them. The CNN models they compared are AlexNet, VGG-16, VGG-19, SqueezeNet, GoogleNet, MobileNet-V2, ResNet-18, ResNet-50, ResNet-101, and Xception. They achieved to the best results with ResNet-101 CNN model as the AUC score of 0.994, the sensitivity of 100%, the specificity of 99.02%, the accuracy of 99.51%, the precision of 99.03%, and the negative predictive value of 100%. Moreover, the results of other CNN architectures can be found on the original paper.

Y. Pathak et al. [16] used a deep transfer learning technique on chest computed tomography (CT) images to classify non-COVID-19 and COVID-19 people. They used pre-trained ResNet-50 CNN architecture on ImageNet [17] dataset to train their classification problem, and 10-Fold cross-validation to prevent overfitting. Then, they obtained the testing accuracy as 93.02%.

Tulin Ozturk et al. [18] developed a new CNN architecture, named as DarkCovidNet, to classify chest X-Ray images among COVID-19 and no-finding, and among COVID-19, no-findings and pneumonia not caused by COVID-19. They inspired by the DarkNet architecture [19] and constructed theirs as consisting of seventeen convolution layers and one fully-connected layer. The final test result were achieved as the accuracy of 87.02% for 3-class classification problem, while it was 98.08 for binary classification problem.

Yujin Oh et al. [20] solved COVID-19, normal and pneumonia not caused by COVID-19 classification problem with first segmenting the chest X-Ray images and yielding extracted lung areas. Then, segmented images were classified patch-by-patch

in ResNet-18 CNN architecture based model. For the final decision among patches, the majority voting method was used, and the results were obtained as the accuracy of 88.9%, the sensitivity of 85.9%, and the specificity of 96.4%.

Elshennawy and Ibrahim [21] proposed four different deep learning models to solve 3-class classification problem. The dataset consist of chest X-Ray images and the three classes are no finding, bacterial pneumonia and COVID-19. A CNN model containing 4 convolutional layers and 3 fully-connected layers is proposed and trained from scratch, and the validation loss and accuracy were obtained as 0.3020 and 92.19% respectively. A LSTM-CNN model containing one LSTM layer, 4 convolutional layers and 2 fully-connected layers were developed, and the final results were yielded as the loss of 0.5771 and the accuracy of 91.80%. Furthermore, two CNN models, ResNet152V2 and MobileNetV2, were used as pre-trained. The test results for ResNet152V2 and MobileNetV2, were achieved as the loss of 0.0523 and the accuracy of 99.22%, and as the loss of 0.1665 and the accuracy of 96.48% respectively.

Ruaa Adeeb Al-Falluji et al. [22] were used a modified ResNet-18 CNN model to classify X-Ray images among COVID-19, no-findings and pneumonia not caused by COVID-19. The original ResNet-18 architecture was modified as changing the kernel size of conv1 convolution layer from 7 to 3, and adding two new convolution layers after the global average pooling layer. The final test result obtained with this technique was the accuracy of 96.73%.

Majid Noue et al. [4] was developed a novel CNN architecture, and solved the 3-class, COVID-19, normal and viral pneumonia other than SARS-CoV-2, based on Deep Features and Bayesian Optimization. The developed CNN architecture includes 5 convolution layers, 3 fully-connected layers and Softmax activation layer at the end. Deep features extracted from fc1 and fc2 layers are used to feed machine learning algorithm, which are support vector machine, decision tree and k-nearest neighbor, for final classification. The result achieved for features extracted from fc2 are obtained by SVM classifier as the sensitivity of 89.39%, the specificity of 99.75%, the F-score of 96.72, and the accuracy of 98.97%.

The results of studies stated in this section can be viewed as merged in the Table 1.1.

Table 1.1 : Reviewed works in the literature and their stated results.

| Authors | Technique | Dataset | The Number of Classes | Accuracy (%) | Sensitivity (%) | Specificity (%) |
|-------------------------------------|--|--------------------|-----------------------|--------------|-----------------|-----------------|
| Ali Abbasian Ardakani et al. (2020) | ResNet-101 CNN model | Chest CT images | 3 | 99.51 | 100.00 | 99.02 |
| Y. Pathak et al. (2020) | Deep transfer learning on ResNet-50 CNN model | Chest CT images | 3 | 93.02 | 91.46 | 94.78 |
| Tulin Ozturk et al. (2020) | DarkCovidNet CNN model | Chest X-Ray images | 3 | 87.02 | 92.18 | 89.96 |
| Tulin Ozturk et al. (2020) | DarkCovidNet CNN model | Chest X-Ray images | 2 | 98.08 | 95.13 | 95.30 |
| Yujin Oh et al. (2020) | Patch-by-patch classification in ResNet-18 CNN based model after segmentation | Chest X-Ray images | 2 | 88.90 | 85.90 | 96.40 |
| Elshennawy and Ibrahim (2020) | Pre-trained ResNet152V2 CNN model | Chest X-Ray images | 2 | 99.22 | 99.44 | 99.45 |
| Ruaa Adeeb Al-Falluji et al. (2020) | Deep transfer learning on the modified ResNet-18 CNN model | Chest X-Ray images | 2 | 96.73 | 94.00 | 100.00 |
| Majid Noue et al. (2020) | SVM classifier fed with deep features obtained from the novel proposed CNN model | Chest X-Ray images | 2 | 98.97 | 89.39 | 99.75 |

1.3 Structure

In Chapter 1, we presented an overview of the study including the history of COVID-19 disease, the scope and aim of the thesis, and the literature survey on the related works.

This rest of thesis is organized in six chapters. In Chapter 2, the information about chest X-rays are given. Chapter 3 introduces the basics of deep learning networks, loss functions and optimization methods, the basics of convolutional neural networks, transfer learning, and the CNN architectures experimented in this study such as AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG16 and VGG19.

Chapter 4 introduces the basics of machine learning, cross-validation, the regularization approaches, and the ML algorithms experimented in this study such as SVM, LR, KNN and LDA.

In Chapter 5, all experiments on convolutional neural networks and machine learning algorithms carried out in this study take place together with how the dataset was constructed, data augmentation, deep feature extraction, how the feature matrices were formed, and the methods for hyper-parameter tuning on ML.

In Chapter 6, all results of experiments are given together with how we measure the performance by confusion matrices.

Finally, in Chapter 7, the conclusions of our study are presented with interpretations and suggestions for further researches.

2. CHEST X-RAYS

X-rays (or X-radiations) are electromagnetic waves which is actually a type of radiation with wavelengths shorter than ultraviolet light. German physicist Wilhelm Konrad Röntgen discovered this specific type of energy photons while investigating cathode rays in Crookes tubes in 1895. Thereafter, Röntgen named this rays as X-rays to identify an unknown type of radiation.

On the other hand, a chest X-ray, or in other words a chest radiograph, is a projection radiograph of the chest used to diagnose health problems associated with the chest and its surroundings. The first use of X-rays for chest imaging dates back to 1896. Nowadays, X-rays are mostly used for medical and radiological imaging in medical industry and for security reasons in public areas. **aşağıdaki figür metin içinde konuşulmamış kendi başına duruyor.**

The least (bu ne demek?) photons are absorbed by air while the most (bu ne demek?) photons are absorbed by the calcium in bones. For that reason, bones are visible as white. Fat and other soft tissues absorbs X-rays in middle level, so they are seen as gray. Because of the absorption properties of X-rays, stated above, the lungs are observed in blackish color. **bu paragrafi bir daha okur musun? pek anlayamadım**

While the chest X-ray projection methods may vary from country to country or even from hospital to hospital, chest X-rays can be filmed through three main projection methods. The most preferred projection method is Posterior-Anterior (PA) view in which X-rays go through from patient's posterior to anterior. The patient has to be stand up during the operation. The second most used method is Anterior-Posterior (AP) erect view. On the contrary of frontal view, beams traverse from anterior to posterior in AP projection. When the PA or AP view is not possible due to the health issue of patient such that the patient may not able to stay erect, supine position is applied as AP-Supine projection method. The other most used view type is Lateral projection. It is performed from the left lateral of patient while the patient is standing. An illustration of each projection method for chest X-rays for detecting COVID-19 is



Figure 2.1 : Hand mit Ringen (Hand with Rings): print of Wilhelm Röntgen's first medical X-ray of his wife's hand [1].

given in Figure 2.2. There are other various projection methods to film the chest of patient via X-rays.

Chest X-ray imaging has a very significant role on detecting serious diseases such as pneumonia and its causes, pneumothorax, heart failures, emphysema, lung cancer, broken ribs for years, and COVID-19 recently. Thanks to the accessibility and the ease of use of chest X-rays, various artificial intelligence problems can be formed and accomplished for the benefit of medical use, computer science, and mathematics.

¹Retrieved from GitHub: <https://github.com/ieee8023/covid-chestxray-dataset/tree/master/images> on March 25, 2021.



(a) PA projection

(b) AP projection



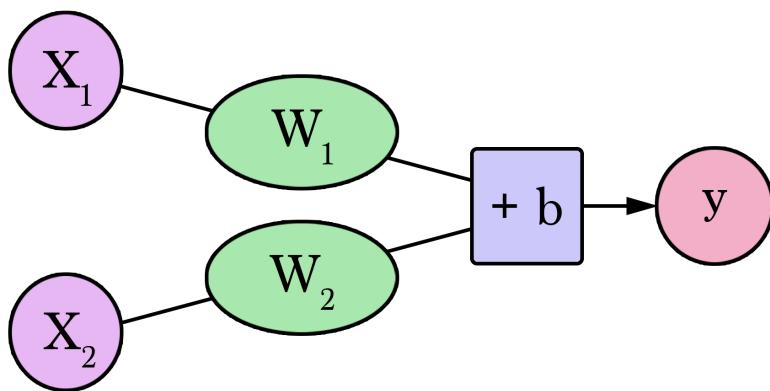
(c) AP-Supine projection

(d) L projection

Figure 2.2 : Examples for COVID-19 chest X-rays filmed with PA, AP, AP-Supine, and Lateral projections, respectively.¹

3. INTRODUCTION TO DEEP LEARNING

In this section, first of all, we provide information on the basics of deep learning algorithms, and then we mention the loss functions, and the optimization methods used to train deep learning algorithms. We conclude the section with the basics of convolutional neural networks, the CNN models used in the thesis, and the transfer learning concept.



(a) Neuron with two variables

$$y = W_1 x_1 + W_2 x_2 + b$$

(b) Formula for the output of neuron on (a)

Figure 3.1 : Sample Neuron.¹

3.1 The Basics of Deep Learning

Here we list basic terminologies associated with deep learning algorithms.

- **Neural Network:** A neural network is a learning framework for machines using a collection of functions over hidden layers to understand and translate an input data

¹Retrieved from GitHub: <https://jalammar.github.io/visual-interactive-guide-basics-neural-networks/> on March 28, 2021.

into a desired output. The main inspiration is the human brain structure. bi tık daha aciklayici yazmakta fayda var. katmanlardan olusuyor vs demek gerekiyor sanirim. Asagida birdenbire classifier ifadesi geciyor, bilmeyen ne oldugunu kafasinda canlandiramayabilir. burda soylemek gerekebilir.

- **Nodes (Neurons):** Neural networks are made of nodes (neurons). A node is a set of weights for that node and its associated nodes. Each node has an input data, a weight, and a bias value. bi tık daha aciklayici yazmakta fayda var

Aşağıda CNN kısmında geçiyor ama tanımı yer almıyor. Feed-forward, back-forward nedir buradan kısaca bahsetsen mi acaba?

- **Weight and Bias:** Weight transforms input data to the hidden layers of neural network. Some sources includes the bias into the weight. Both of them are learnable parameters from data, but bias is used and optimized to tune the distance between the predicted value and the intended value.(cumle anlasılmıyor, intended value ne demek?) Yukarıdaki figur tanimlarla ilişkilendirilmemis. Bir de, matematiksel formuller, metne figur olarak girmese daha iyi, görevi gerek, denklem olarak girebilir metne. Bir de hidden layer ifadesi daha once gecmemisti, simdi birden gecince anlasılmıyor. Input ile output ile arasindaki katmanlar arasında
- **Epoch:** An epoch is an iteration when the train data passes forward and backward through the whole neural network only once. Generally, the train data passes through the whole network multiple times to optimize the learning process.
- **Batch:** If an epoch is too large to enter the neural network at once, the train data in the epoch can be divided into several smaller batches. Then, a batch is defined as a group of data points selected from the train data. The number of data points in a batch are mostly chosen as a power of 2 such as 2^4 , 2^5 , and 2^6 etc. If the train data enters the neural network in batches, then it is recommended to shuffle data order in each epoch before creating new batches from train data.
- **Train Data and Training:** The data group reserved to be used in learning process is called as train data. The ratio of train data size to whole data is generally determined as 7:10 or 8:10, and the rest of data is placed into the test data group. The train data must be strictly separated from validation and test data, and should be only used on training process. Training process consists of multiple epochs. At each epoch, the

train data goes into the model as input, the neural network works for learning data, the losses and accuracy (if desired) are computed, and the losses are optimized. If batching is used, this process is applied onto each batch, and the average and final losses and accuracy (if desired) are calculated over all batches at the end of each epoch.

- **Validation Data and Validating:** If there exists enough train data for the problem and learning process, one may need validation data to validate the learning process periodically. **The validation data group is strictly separated from train data such that its size is generally 70% or 80% of train data.** During the validation process no optimization process is applied, for that reason, the weights and biases are stable. The losses and other desired metrics such as accuracy are computed only.
- **Test Data and Testing:** The data group reserved to be used in testing process is called as test data. **The ratio of the size of test data to the whole data is generally determined as 2:10 or 3:10.** The test data must be strictly separated from train data, and should be only used on testing process. The training process should not see any of test data. Testing process is usually only applied at the end of all epochs. However, when the train data is not feasible to be divided into validation data, at the end of each epoch testing process may be applied on test data. This method is also used to determine the over-fitting and stop point.
- **Under-fitting and Over-fitting:** A learning algorithm is appeared to have under-fitting when it cannot capture or fit the data well. On the other hand, when the model fits on the train data perfectly, it might have started to memorize the train data and may not learn something new from the data. Therefore, the testing metrics in the opposite direction of what is desired can be seen. This situation is called as over-fitting. **bu tanımı biraz daha açsan mı acaba? Bu tanımlara bakınca, under-fitting ve over-fittin arasındaki fark anlaşılmışıyor. Aşağıdaki figuru metne bağlamak gerek, bir de figur de too simple to gibi yazan kısmı kesmek gerek bence. Figur içinde ufak not, bilimsel yazıya uymuyor, yesil cerceve de gitse iyi olur).**

3.2 The Cross-Entropy Loss Function

²Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/> on March 28, 2021.

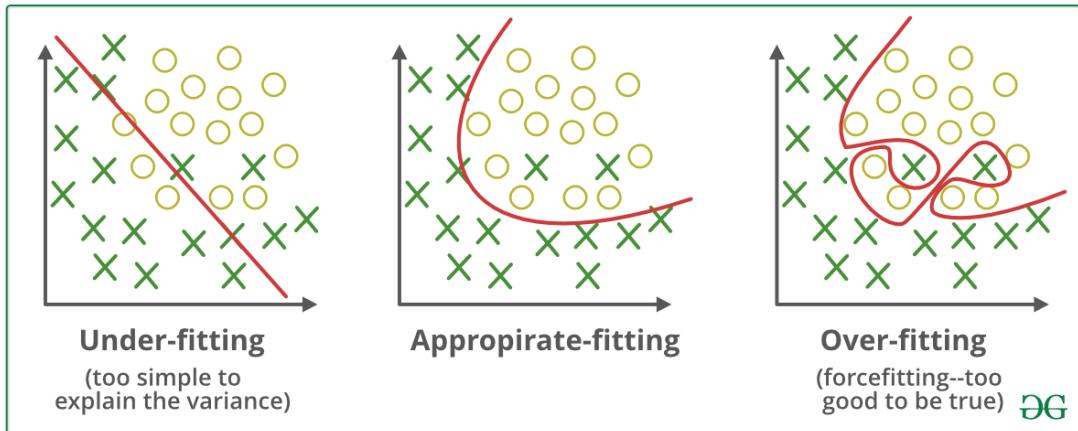


Figure 3.2 : Illustration of under-fitting, well-fitting, and over-fitting of models to the data.²

Loss function is the function that represents the error between true and predicted values which can be called as the cost of the problem to be optimized.

The cross-entropy loss function is one of the most commonly used loss functions to measure the performance of a classifier. **The function maps the probability values changing between 0 and 1 to the cost value.** Here the loss value increases together with the divergence between the true label and the predicted probability. burayı anlayamadım, tekrar ifade edebilir misin?

The cross-entropy loss function is defined as given below:

$$L_{\text{cross-entropy}} = - \sum_{i=1}^n y_i \log(p_i), \quad (3.1)$$

where n is the total number of classes, and y_i and p_i are the observed class label and the estimated probability from the model of interest for the i^{th} class, respectively. Thus, for a binary classification problem, the cross-entropy loss function can be re-stated as:

$$L_{\text{cross-entropy}} = -(y \log(p) + (1-y) \log(1-p)), \quad (3.2)$$

where y is the binary indicator variable with two class labels 1 and 0, where 1, true class label, shows the presence of the condition of interest, whereas 0 shows the opposite, and p is the estimated probability for the true class label.

3.3 The Basics of Optimization Methods

Various optimization methods can be used for reducing the losses and reaching more accurate predictions. The weights are initialized and updated at each training epoch. The initialization strategies may vary depending on the neural network architecture. It is aim to achieve the most satisfying results by using some optimization algorithms called as optimizers. burayı bir daha yazar mısın, cümleler sanki kopuk olmuş

3.3.1 Stochastic Gradient Descent with Momentum Optimization Algorithm

Stochastic gradient descent (SGD) can be considered as a stochastic approximation of gradient descent optimization daha once gradient descent tan bahsetmedigimiz icin okuyucuya bir referans versen burada iyi olur and is an iterative method for optimizing an objective function with suitable smoothness properties. The momentum remembers the weight update at each iteration and determines the next update as a linear combination of the gradient and the previous update [23]. An example for a pseudo-code for SGD with momentum optimization is given in Figure 3.3. bu figurdeki mor yerleri silmemiz mumkun mu, scientific durmuyor

Algorithm 1 SGD with momentum

```

1: given learning rate  $\alpha_t \in \mathbb{R}$ , momentum factor  $\beta_1 \in \mathbb{R}$ , weight decay factor  $w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\mathbf{x}_{t=0} \in \mathbb{R}^n$ , first moment vector  $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow SelectBatch(\mathbf{x}_{t-1})$      $\triangleright$  select batch and return the corresponding gradient
6:    $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1}) + w_t \mathbf{x}_{t-1}$ 
7:    $\eta_t \leftarrow SetScheduleMultiplier(t)$             $\triangleright$  can be fixed, decay, be used for warm restarts
8:    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + \eta_t \alpha_t \mathbf{g}_t$ 
9:    $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{m}_t$ 
10:  until stopping criterion is met
11:  return optimized parameters  $\mathbf{x}_t$ 

```

Figure 3.3 : A pseudo-code for SGD with momentum optimization [2].

3.3.2 Adaptive Moment Estimation Optimization Algorithm

Adaptive moment estimation (Adam) is an optimization algorithm that uses the running averages of both the gradients and the second moments of the gradients. Before each step, the gradient of loss function is calculated and a step in the opposite direction is taken with corresponding rate. This process is also called as Adam with L2 regularization [24]. An example for a pseudo-code for ADAM optimization is given in Figure 3.4. bu figurdeki mor ve yesil yerleri silmemiz mumkun mu, scientific

durmuyor adam ve adamw arasındaki farkı gösteriyor ama yine de bilemedim. Ayrıca, figürleri verince metin içinde tartışmak gerekiyor. Çıkarım yapma işini okuyucuya bırakırsan, vermek istenilen mesajı iletemedigimiz anlamına gelir. O yuzden en azından Adam ve AdamW yi kıyaslayarak anlatsan daha iyi olur.

3.3.3 Adam Weight Decay Optimization Algorithm

Adam weight (AdamW) decay optimization algorithm does not use the regularization as the normalized by square root of second moment vector. Thus, it is only proportional to the weight itself [24]. An example for a pseudo-code for AdamW optimization is given in Figure 3.4. yukarıda dedigim gibi bu figuru algoritmaları kıyaslayarak tartışmaka fayda var.

Algorithm 2 Adam and AdamW

```

1: given  $\alpha_t = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$ 
2: initialize time step  $t \leftarrow 0$ , parameter vector  $\mathbf{x}_{t=0} \in \mathbb{R}^n$ , first moment vector  $\mathbf{m}_{t=0} \leftarrow \mathbf{0}$ , second
   moment vector  $\mathbf{v}_{t=0} \leftarrow \mathbf{0}$ , schedule multiplier  $\eta_{t=0} \in \mathbb{R}$ 
3: repeat
4:    $t \leftarrow t + 1$ 
5:    $\nabla f_t(\mathbf{x}_{t-1}) \leftarrow SelectBatch(\mathbf{x}_{t-1})$      $\triangleright$  select batch and return the corresponding gradient
6:    $\mathbf{g}_t \leftarrow \nabla f_t(\mathbf{x}_{t-1}) + w_t \mathbf{x}_{t-1}$ 
7:    $\mathbf{m}_t \leftarrow \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$            $\triangleright$  here and below all operations are element-wise
8:    $\mathbf{v}_t \leftarrow \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$ 
9:    $\hat{\mathbf{m}}_t \leftarrow \mathbf{m}_t / (1 - \beta_1^t)$                    $\triangleright$  here,  $\beta_1$  is taken to the power of  $t$ 
10:   $\hat{\mathbf{v}}_t \leftarrow \mathbf{v}_t / (1 - \beta_2^t)$                    $\triangleright$  here,  $\beta_2$  is taken to the power of  $t$ 
11:   $\eta_t \leftarrow SetScheduleMultiplier(t)$        $\triangleright$  can be fixed, decay, be used for warm restarts
12:   $\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \eta_t \left( \alpha_t \hat{\mathbf{m}}_t / (\sqrt{\hat{\mathbf{v}}_t} + \epsilon) + w_t \mathbf{x}_{t-1} \right)$ 
13: until stopping criterion is met
14: return optimized parameters  $\mathbf{x}_t$ 

```

Figure 3.4 : Pseudo-codes for Adam and AdamW optimization algorithms [2].

3.4 The Basics of Convolutional Neural Networks

The convolutional neural networks (CNNs) are the special research area of deep neural networks which are mostly used in visual problem analyzing and solving. In this section and its subsections we study on the fundamental layers and elements of CNN architectures.

Basically, a CNN model consists of three main layers: 1) An input layer, 2) hidden layers, and 3) an output layer. For example, Figure 3.5 is an example for a CNN model with six layers consisting of one input layer, four hidden layer, one output layer.

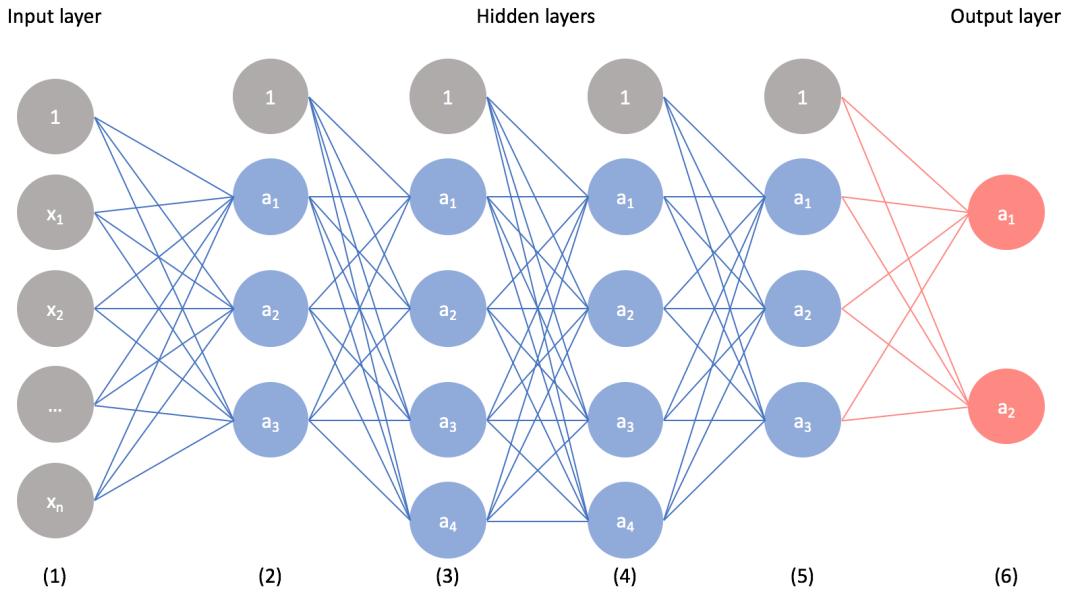


Figure 3.5 : An example for a simple CNN architecture.³

The structure of CNNs comports with the structure of feed-forward networks where the information goes through network in forward direction only. Back-propagation algorithm is commonly used to feed and train a feed-forward network. The algorithm orders to compute the gradients in weight space of a feed-forward network with respect to a loss function. Then the weight space is updated with these gradients by an appropriate optimizer. burası açık değil. The gradient of objective loss function $f(\theta)$ at a weight vector θ_t is computed via directional derivatives and chain rule, and represented by $\nabla f(\theta_t)$. Şimdiye kadar weightleri W ile göstermiştin, şimdi theta ya gecimişsin, iki gösterim tutarsız kalınca, kafa karışıyor, theta tam olarak ne?W olamaz mı gösterimi? Bir de bu ifade zorunlu mu bir daha bir yerde geçmiyor sanki..

Let's consider the simple CNN architecture given in Figure 3.5 with each layer is denoted by \mathbf{a}_i , where $i = 1, 2, \dots, 6$. Assume there exists n number of x_i nodes feeding \mathbf{y} , where $\mathbf{x} = \mathbf{a}_1 = (x_1, \dots, x_n)$ is a n-dimensional the input layer vector. Furthermore, the vector \mathbf{a}_i , with $i = 2, \dots, 5$, are the hidden layers. For instance, $\mathbf{a}_3 = (a_{3,1}, a_{3,2}, a_{3,3}, a_{3,4})$ and $\mathbf{a}_5 = (a_{5,1}, a_{5,2}, a_{5,3})$. Lastly, the output layer vector is $\mathbf{y} = \mathbf{a}_6 = (a_{6,1}, a_{6,2})$.

³Retrieved from: <https://www.jeremyjordan.me/convolutional-neural-networks/> on March 31, 2021.

Then the gradient of \mathbf{y} at \mathbf{x} is constructed as: \mathbf{y} of \mathbf{x} dememiz gerekir mi? \mathbf{y} neden bold degil vectörün içinde, gradient vektör mü, matrix mi burada çıkaramadım. size i verseydin iyi olabilirdi.

$$\nabla \mathbf{y}(\mathbf{x}) = \begin{bmatrix} \frac{\partial \mathbf{y}(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial \mathbf{y}(\mathbf{x})}{\partial x_n} \end{bmatrix}. \quad (3.3)$$

Then, using directional derivatives and chain rule can be illustrated in Figure 3.6. Bu figuru sen mi cizdin? a_1, a_6, a_2 de $a_6, 2$ olmalı sanırımlı yukarıdaki paragrafa bakarsak. bir de denklemi, denklem olarak ifade edersen, figur yerine iyi olur gibime geldi. $i=1,2,\dots,n$. n kullandım cunku CNN mimarsini gösteren figurde x katmanı $1,\dots,n$. bir de figurde y bold ama denklemde bold degil. hangisi daha doğru acaba?

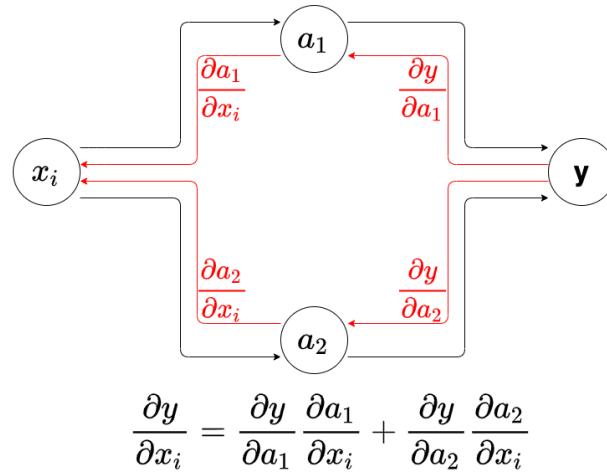


Figure 3.6 : An example for usage of directional derivatives and chain rule.

The gray nodes in some layers of CNN architecture given in Figure 3.5 are real valued bias terms. Assume that the objective loss function is the least squares loss, i.e. $l = l(\mathbf{y}, \mathbf{y}^{GT}) = \frac{(\mathbf{y}^{GT} - \mathbf{y})^2}{2}$ where \mathbf{y}^{GT} is the grand-truth labels of our data set. Assume that our hypothesis function is to create a CNN architecture in the form $f : \mathbb{R}^k \rightarrow \mathbb{R}^t$, where k and t vary according to the corresponding layer. Then define $\mathbf{a}_i = f_i(W_i \mathbf{a}_{i-1} + b_i)$ for each i^{th} layer, where $\mathbf{W} \in \mathbb{R}^{t \times k}$ is a weight matrix. We obtain the backward pass of our loss function with respect to the first weight matrix and bias term by chain rule as follows: Figure 3.1 de \mathbf{W}_1 indis asagida diye indisleri asagi cektim am sonra boldW ile i indisini kafamda bagdastiramadim. tartisalim. bu agirlik matrisinin boyutunu. Ayrıca deklem 3.3 u aslinda ihtiyacım yok sanırımlı, bir yerde kullanmıyoruz. burayı tartisalim,

duzeltelim. bir de backward pass, forward pass ifadesi de kafamı karistirdı, CNN ileri besleme dedik, backward pass, forward pass ve backpropagation iliskisini acikca ifade edelim. Bir de loss function olarak yukarida cross-entropy den bahsettik, simdi least squares diyoruz. Tutarlı olmakta fayda var. 3.4.1 e kadar olan yere bakmadım. tartisalim duzenlendikten sonra.

$$\frac{\partial l(\mathbf{y}, \mathbf{y}^{GT})}{\partial \mathbf{W}_1} = \frac{\partial l}{\partial \mathbf{a}_6} \times \frac{\partial \mathbf{a}_6}{\partial \mathbf{a}_5} \times \frac{\partial \mathbf{a}_5}{\partial \mathbf{a}_4} \times \frac{\partial \mathbf{a}_4}{\partial \mathbf{a}_3} \times \frac{\partial \mathbf{a}_3}{\partial \mathbf{a}_2} \times \frac{\partial \mathbf{a}_2}{\partial \mathbf{W}_1}, \text{ and} \quad (3.4)$$

$$\frac{\partial l(\mathbf{y}, \mathbf{y}^{GT})}{\partial b_1} = \frac{\partial l}{\partial \mathbf{a}_6} \times \frac{\partial \mathbf{a}_6}{\partial \mathbf{a}_5} \times \frac{\partial \mathbf{a}_5}{\partial \mathbf{a}_4} \times \frac{\partial \mathbf{a}_4}{\partial \mathbf{a}_3} \times \frac{\partial \mathbf{a}_3}{\partial \mathbf{a}_2} \times \frac{\partial \mathbf{a}_2}{\partial b_1}. \quad (3.5)$$

Here, each partial derivative is computed as shown in Figure 3.6. For instance, if the forward pass connection between $\mathbf{a}_{4,2}$ and $\mathbf{a}_{5,1}$ is $W_{2,1}^4$, we use the directional derivative rule while computing $\frac{\partial l}{\partial \mathbf{a}_6}$ for the backward pass gradient to update $W_{2,1}^4$ value as:

$$\frac{\partial l}{\partial W_{2,1}^4} = \frac{\partial l}{\partial \mathbf{a}_6} \times \frac{\partial \mathbf{a}_6}{\partial \mathbf{a}_{5,1}} \times \frac{\partial \mathbf{a}_{5,1}}{\partial W_{2,1}^4} = \left(\frac{\partial l}{\partial \mathbf{a}_{6,1}} \times \frac{\partial \mathbf{a}_{6,1}}{\partial \mathbf{a}_{5,1}} + \frac{\partial l}{\partial \mathbf{a}_{6,2}} \times \frac{\partial \mathbf{a}_{6,2}}{\partial \mathbf{a}_{5,1}} \right) \times \frac{\partial \mathbf{a}_{5,1}}{\partial W_{2,1}^4}. \quad (3.6)$$

In the same way, the gradient for b^5 can be computed as:

$$\frac{\partial l}{\partial b^5} = \frac{\partial l}{\partial \mathbf{a}_6} \times \frac{\partial \mathbf{a}_6}{\partial b^5} = \frac{\partial l}{\partial \mathbf{a}_{6,1}} \times \frac{\partial \mathbf{a}_{6,1}}{\partial b^5} + \frac{\partial l}{\partial \mathbf{a}_{6,2}} \times \frac{\partial \mathbf{a}_{6,2}}{\partial b^5}. \quad (3.7)$$

3.4.1 Convolutional Layer

Convolutional layer is the building block of CNN architectures and is used to reveal the distinctive features of input data. A convolution layer includes a window named as filter (or kernel) that can be of different sizes such as 2×2 , 3×3 etc. Filtering is started to be applied from the upper left corner of the image and keeps moving towards the right. When there are no cells left to be scanned on the right, it continues from the lower cells as before. At each step, the filter and the corresponding area of input matrix are multiplied by element-wise. The step length of scanning process is called as stride. The default stride value is 1; however, different stride values may be used depending on the input data and CNN architecture. If the stride is set as s , then the kernel window move by jumping s cells. In some situations such as a size inconsistency or paying

attention to the information burayı tekrar yaz on the edges and corners of the image, the edges of input matrix may be filled by zeros. These process is called as padding. If the padding value is set as p , the p cells with zero values are added to the edges of input matrix.

The various filters can be applied onto the input data to reveal low and high-level feature maps. After convolution, the size of the input data changes depending on the filter (or kernel), stride, and padding choices. The outputs of the convolution layers are called activation maps.

The coefficients that filters include change on each iteration during the training of a CNN. This coefficients and their updates are used to detect the significant and learnable parts of the data. burayı bir yere baglamadım. uygun yere koyarsan sevinirim.

The Figure 3.7 gives a summary about how activation map is constructed for a given input matrix and a filter. Let us give a detailed example to about it. Assume that the input data holds the raw pixel values of an image of width 223, height 223 with three color channels red, green, and blue, in short, it has a size of $223 \times 223 \times 3$. Let's choose two filter windows in the size of 5×5 , the stride (step size) as 2 and the padding size as 1. Let denote the input size with $W_{input} \times H_{input} \times D_{input}$, the padding with P , the number of filters with K , the size of a filter window with $F \times F$ and the stride with S .

Here, we have:

- $W_{input} = 223, H_{input} = 223, D_{input} = 3,$
- $K = 2, F = 5,$
- $P = 1, \text{ and } S = 2.$

The size of activation map is calculated as:

$$W_{output} = (W_{input} - F + 2P) / S + 1, \quad (3.8)$$

$$H_{output} = (H_{input} - F + 2P) / S + 1, \quad \text{and} \quad (3.9)$$

$$D_{output} = K. \quad (3.10)$$

Consequently, the activation map is yielded in the size of $112 \times 112 \times 2$ by the equations (3.8), (3.9) and (3.10) respectively.

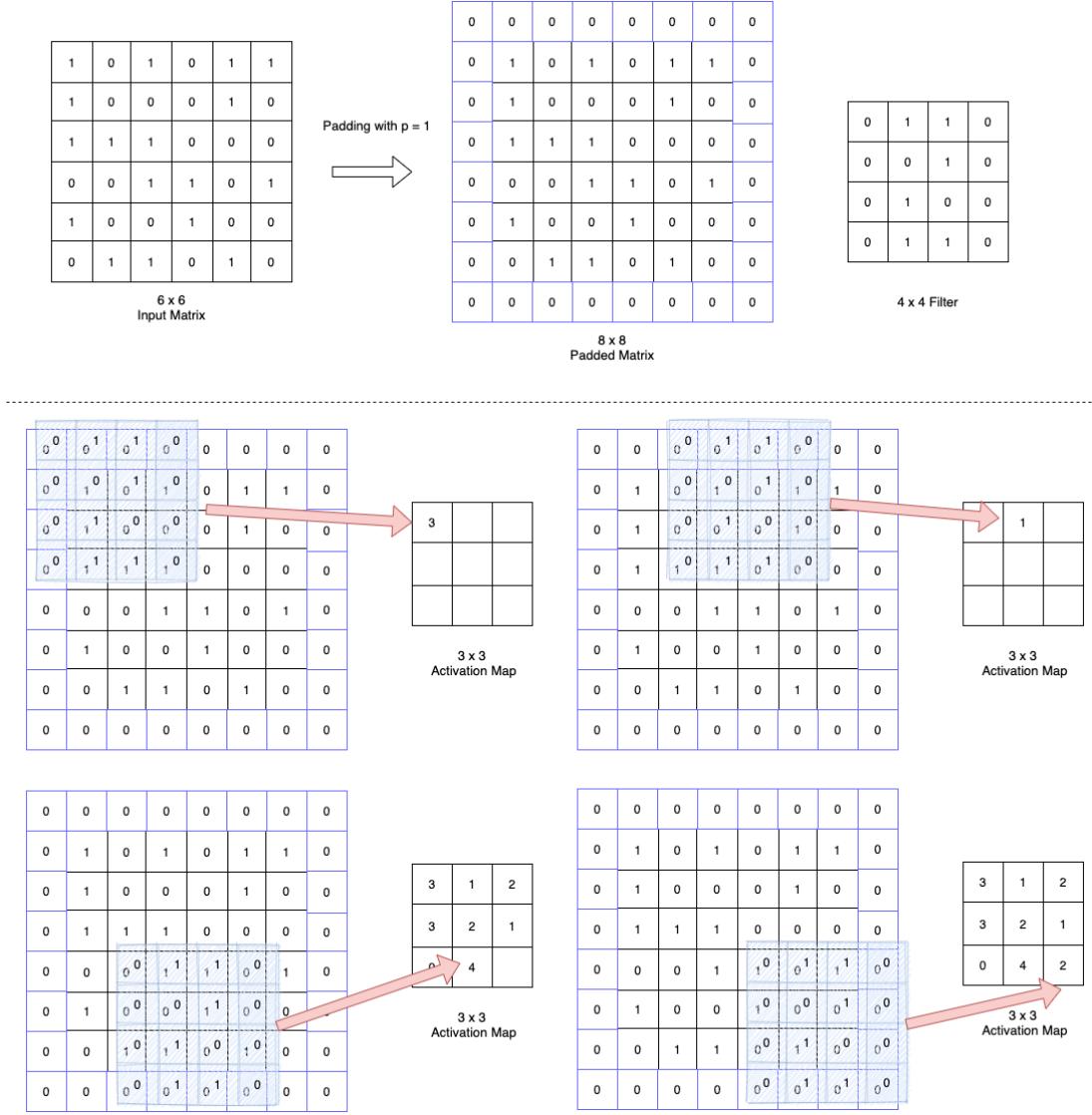


Figure 3.7 : Activation map construction with a filter window size of 4×4 and stride value of 2.

3.4.2 Activation Function (Rectified Linear Units - ReLU)

The activation function forms the output of the node for a given input or input set. Rectifier activation function maps the input to non-negative values. A unit form of the rectifier activation function, which is given in equation (3.11), is called as Rectified Linear Units (ReLU) which is the most common used in neural networks.

$$\text{ReLU is defined as: } f(x) = \max(0, x). \quad (3.11)$$

3.4.3 Pooling Layer

Pooling layers are commonly used in between the convolutional layers, especially after activation functions. A pooling layer outputs a new feature map with the lower size than the size of input feature map. The characteristics of output feature map depends on the type of pooling layer. The pooling layers have a stride parameter to control the step size of the movement of pooling window. The most common pooling functions are given below.

- **Average Pooling:** The average of corresponding window is computed and inserted into output feature map.
- **Maximum Pooling:** The maximum value of corresponding window is inserted into output feature map.

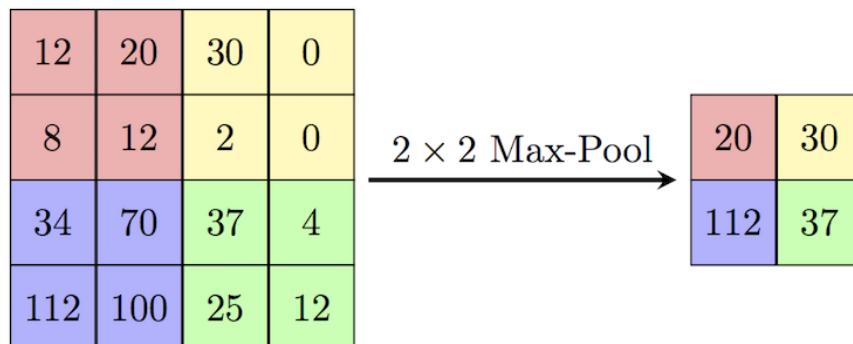


Figure 3.8 : Maximum pooling operation sample with 2×2 window and stride as 2.⁴

3.4.4 Batch Normalization

Batch normalization is a technique for training the deep neural networks that standardize the inputs to one layer for each mini batch. This has the effect of stabilizing the learning process and significantly reducing the number of training epochs required [4].

3.4.5 Drop-out

Drop-out is the removing operation of the nodes below certain threshold value in the network at each training iteration. In other words, it is aimed not to use weak information and memorize the network. Drop-out can be in any layer and its threshold value, which can differ in different layers, is between [0,1].

⁴Retrieved from: <https://computersciencewiki.org/index.php/Max-pooling> on April 3, 2021.

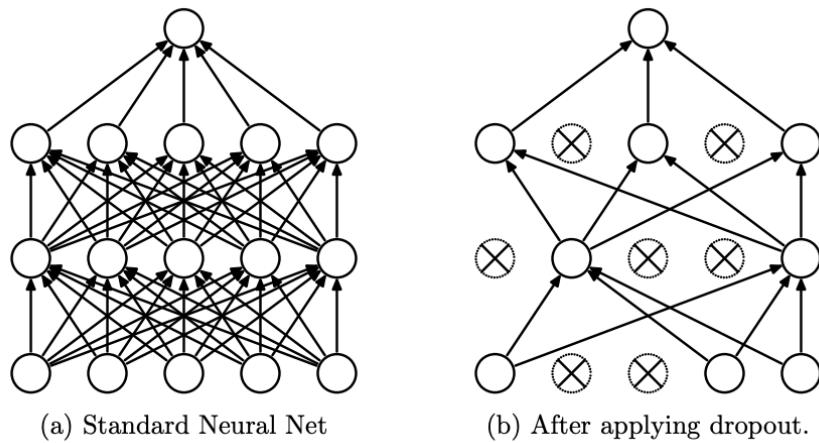


Figure 3.9 : (a) A standard neural network, (b) A neural network after applying dropout [3].

3.4.6 Flattening

Flatten operation reshapes the input into the one-dimension. It is used before passing to the fully-connected layers.

3.4.7 Fully-Connected Layer

This structure is used to combine the information in the previous layers. It pre requires the flatten operation before itself. The fully-connected layers are used to classify data in different categories, or construct feature maps to be used in another artificial network methods.

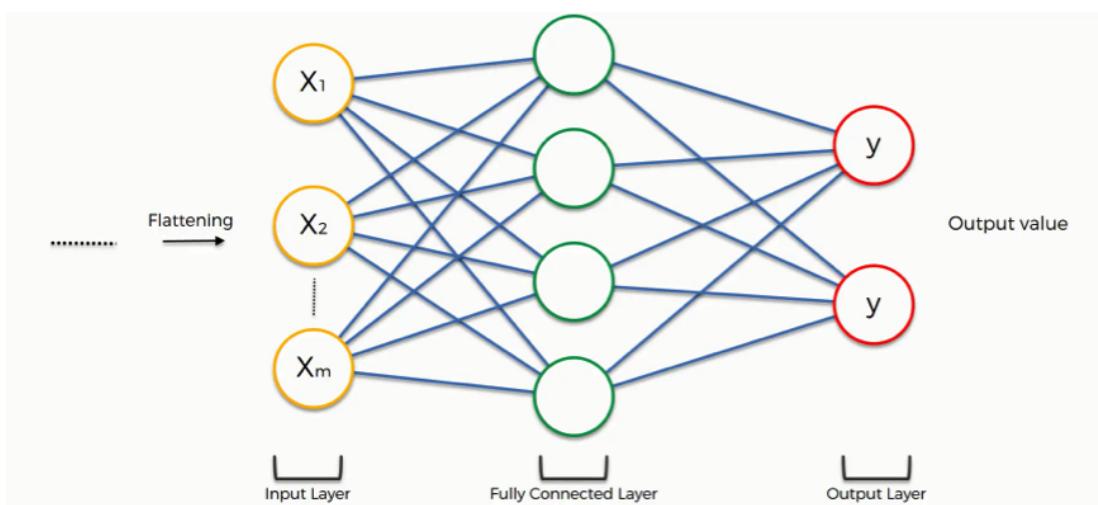


Figure 3.10 : The usage of full-connected layer for a binary classifier.⁵

⁵Retrieved from: <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-4-full-connection> on April 3, 2021.

3.5 Transfer Learning

Transfer learning is transferring the features, weights, etc. in pre-trained model to another artificial learning model. This can be between two deep learning models or across a machine learning and deep learning model. In this thesis, the weight transfer from pre-trained CNN models to CNN models and feature transfer from CNN models to machine learning models are used.

Transfer learning is used to shorten the training time if the data set is insufficient and to achieve high performance with less data. If the data on each class is less than 1000, the dataset may be considered as small.

To prepare a pre-trained CNN model for visual artificial intelligence problems, ILSVRC 2012 ImageNet [17] dataset, including 1000 class with large data, is commonly used. The weights after the training process is saved and shared to use in transfer learning afterwards. On transfer learning, the pre-trained model is called with its pre-computed weights, and the weights are transferred into the same empty neural network. There exist different approaches to train this new neural network; such that, training the all layers on the convolutional block, training a part of the layers on the convolutional block and freezing the other, or freezing all layers on the convolutional block. The choice of strategy, which is called as fine-tuning operation, depends on the size of dataset and the similarity between the dataset used in pre-training process and the dataset to be used in current task. Then the last fully-connected layer is adapted to the current task by updating itself or adding new fully-connected layers.

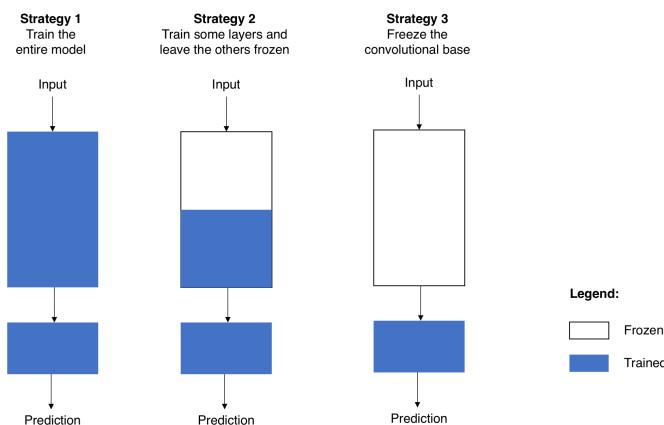


Figure 3.11 : Different fine-tuning strategies on pre-trained model.⁶

To prepare a deep feature set to be used in another artificial learning model, such as machine learning models, the features on a fully-connected layer, mostly the first or second, are extracted and saved. Then the saved deep features are used in the target artificial learning model as the feature map of dataset.

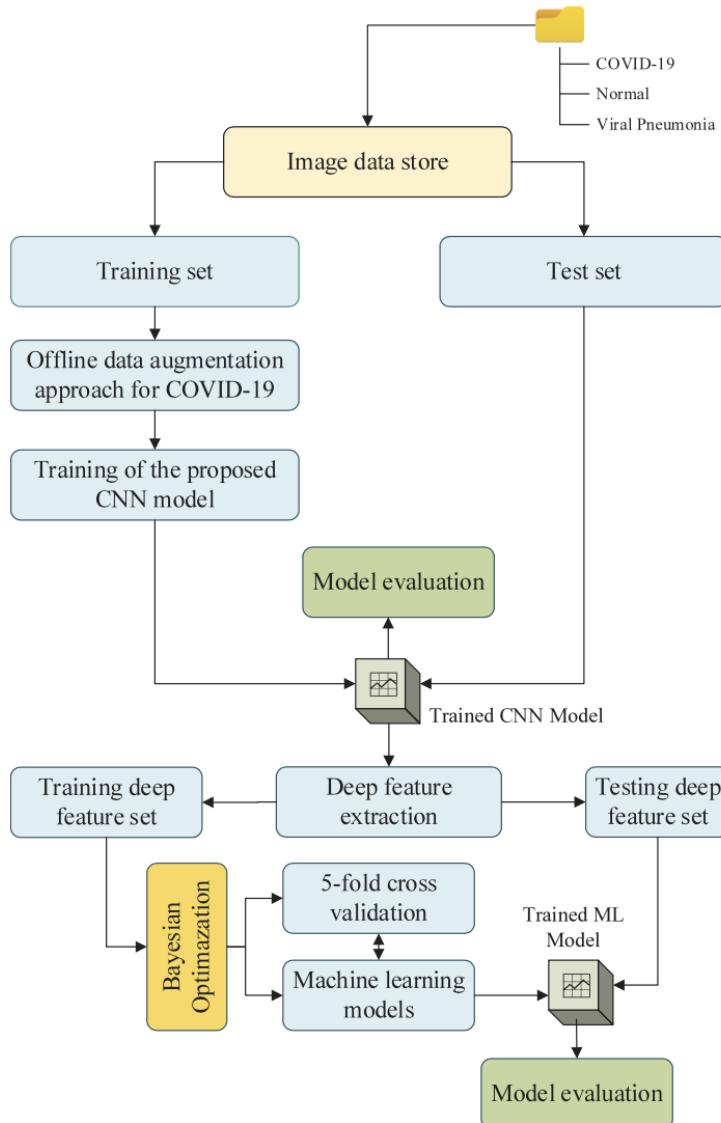


Figure 3.12 : Deep feature extraction from CNN models and using them in machine learning models [4].

3.6 CNN Models

There are too many similar and different CNN architectures which can be simply called as models. These models consist of various combinations of input layer, hidden layers and an output layer such that some basics of them are explained in Section 3.4.

⁶Retrieved from: <https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751> on April 3, 2021.

In this section, the state-of-the-art CNN models used in the thesis are introduced and detailed.

3.6.1 AlexNet

The AlexNet architecture was developed by Krizhevsky et al. [5] for the object recognition problem. This model was trained with the ILSVRC 2012 ImageNet [17] dataset, achieved high performance results, and won the first place in the ImageNet competition in 2012. There are 9 layers in total in the AlexNet architecture and the size of the input image is $224 \times 224 \times 3$. Following the input layer, there are five convolution layers which use filters of size of 11×11 with a stride value of 4, where some layers are supported by the maximum pooling layers as shown in Figure 3.13. The remaining three layers are the fully-connected layers. The ReLU activation function is used after each convolution and fully-connected layer. the number of filters used increases as going deeper into the model. size 11 demistin, simdi degisiyor diyorsun. Output 1000 class vs dememissin. figurde oyle gozukuyor.

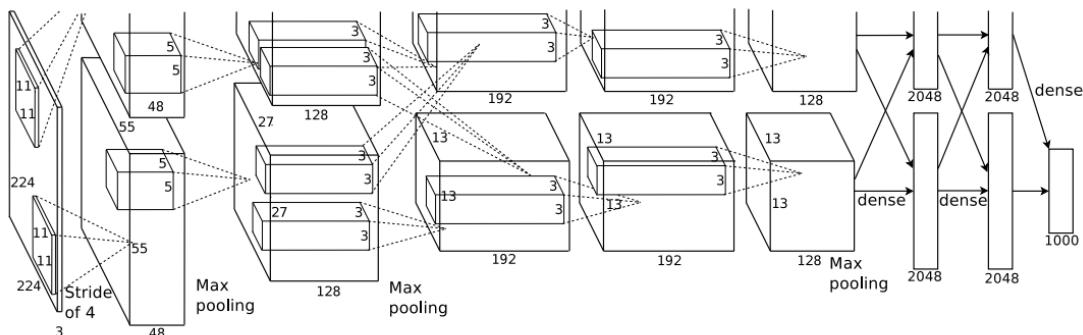


Figure 3.13 : Illustration of AlexNet architecture with a size of $224 \times 224 \times 3$ input image [5].

3.6.2 Residual Neural Networks

The residual neural network (ResNet) architectures was proposed by He et al. [6] to avoid vanishing and exploding gradient problems by using residual blocks which is a simple connection which takes the activation from one layer and feeds it to another layer in much deeper. This model resulted in high performance results on ILSVRC 2015 ImageNet [17] data set and won the 1st place on the corresponding task. **Figürü açıklar misin burada. REsnet, Alexnet gibi belirli bir image size istiyor mu oyleyse yaz lutfen..**

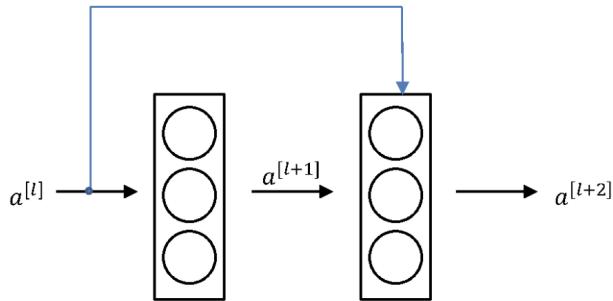


Figure 3.14 : An illustration for residual block. Here, $a^{[l]}$ is the starting activation layer and $a^{[l+2]}$ is the fed activation layer.⁷

Depending on the number of layers, there are a variety of special ResNets architectures which are briefly mentioned below.

ResNet-18

The first convolution layer includes a filter of size 7×7 and scans with a stride value of 2, then maximum pooling is applied with a 2×2 window and a stride value of 2. Afterwards, 8 residual blocks, such that each of them includes 2 convolutional layers with 3×3 filters, come. That is, there exist 16 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following with a fully-connected layer with 1000 neurons, which represents the class number.

ResNet-34

The first convolution layer includes a 7×7 filter and scans with stride as 2, then maximum pooling is applied with 2×2 window and stride as 2. Afterwards, 16 residual blocks, such that each of them includes 2 convolutional layers with 3×3 filters, come. That is, there exist 32 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following with a fully-connected layer with 1000 neurons, which represents the class number.

ResNet-50

The first convolution layer includes a 7×7 filter and scans with stride as 2, then maximum pooling is applied with 2×2 window and stride as 2. Afterwards, 16 residual blocks, such that each of them includes 3 convolutional layers with 1×1 , 3×3

⁷Retrieved from: <http://datahacker.rs/deep-learning-residual-networks/> on April 4, 2021.

and 1×1 filters in order, come. That is, there exist 48 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following with a fully-connected layer with 1000 neurons, which represents the class number.

| layer name | output size | 18-layer | 34-layer | 50-layer | 101-layer | 152-layer |
|------------|--------------|---|---|---|--|--|
| conv1 | 112×112 | | | $7 \times 7, 64, \text{stride } 2$ | | |
| | | | | $3 \times 3 \text{ max pool, stride } 2$ | | |
| conv2.x | 56×56 | $\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 2$ | $\left[\begin{array}{l} 3 \times 3, 64 \\ 3 \times 3, 64 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{array} \right] \times 3$ |
| conv3.x | 28×28 | $\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 2$ | $\left[\begin{array}{l} 3 \times 3, 128 \\ 3 \times 3, 128 \end{array} \right] \times 4$ | $\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$ | $\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 4$ | $\left[\begin{array}{l} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{array} \right] \times 8$ |
| conv4.x | 14×14 | $\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 2$ | $\left[\begin{array}{l} 3 \times 3, 256 \\ 3 \times 3, 256 \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 6$ | $\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 23$ | $\left[\begin{array}{l} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{array} \right] \times 36$ |
| conv5.x | 7×7 | $\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 2$ | $\left[\begin{array}{l} 3 \times 3, 512 \\ 3 \times 3, 512 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$ | $\left[\begin{array}{l} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{array} \right] \times 3$ |
| | 1×1 | | | average pool, 1000-d fc, softmax | | |
| FLOPs | | 1.8×10^9 | 3.6×10^9 | 3.8×10^9 | 7.6×10^9 | 11.3×10^9 |

Figure 3.15 : The ResNet architectures [6].

3.6.3 VGG

Simonyan and Zisserman [25] proposed a very profound architecture for the problem of image recognition. It is aimed to achieve higher classification performance by using a small size filter and increasing the depth of the model. This model was trained with the ILSVRC 2012 ImageNet [17] dataset and achieved high success results. This study has proven the importance of depth in image description models.

VGG16

The input layer of the model architecture starts with a 2424 color image, and there are 12 convolution layers using 3×3 filters. There are a total of 5 maximum pooling layers used after some convolution layers. In maximum pooling, 2×2 windows were used with stride as 2. After the convolution layers, there are three fully-connected layers such that the first two of which are 4096 neurons and the last one is 1000 neurons, which represents the class number. The ReLU activation function was used after all the hidden layers.

VGG19

⁸Retrieved from: <http://datahacker.rs/deep-learning-vgg-16-vs-vgg-19/> on April 4, 2021.

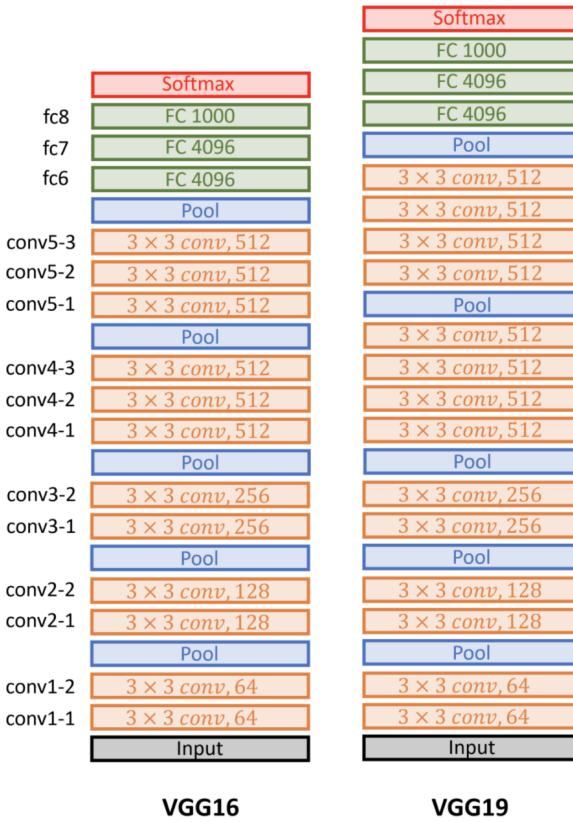


Figure 3.16 : An illustration of VGG architectures.⁸

The input layer of the model architecture starts with a 2424 color image, and there are 14 convolution layers using 33 filters. There are a total of 5 maximum pooling layers used after some convolution layers. In maximum pooling, 2x2 windows were used with stride as 2. After the convolution layers, there are three fully-connected layers such that the first two of which are 4096 neurons and the last one is 1000 neurons, which represents the class number. The ReLU activation function was used after all the hidden layers.

4. INTRODUCTION TO MACHINE LEARNING

In this chapter, we start with the basics of machine learning. Then we mention cross-validation approach, which is a model evaluation technique, and the regularization concept. Finally, we end the section with the ML algorithms used in the thesis.

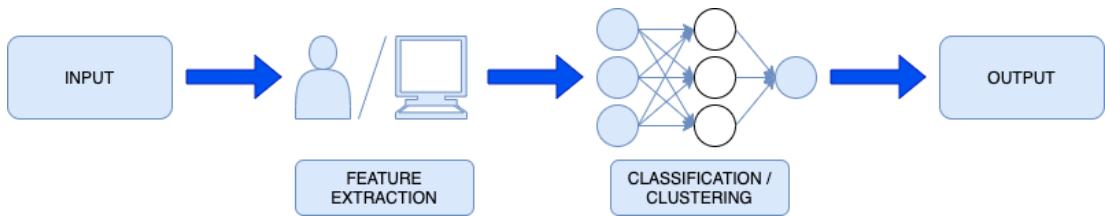


Figure 4.1 : Sample Machine Learning Road Map.

4.1 The Basics of Machine Learning (ML)

The goal of machine learning (ML) is to learn from available data or experiences to solve a given problem. The pipeline may be listed as:

- defining the problem to solve,
- collecting the data for training and testing,
- designing or extracting the features describing data,
- training the model to tune the parameters by experiments and optimization on the appropriate loss function, and
- testing the model to evaluate the performance of trained model on unseen test data.

The majority of the problems in ML are fall into supervised and unsupervised learning approaches.

4.1.1 Supervised Learning

Supervised learning is a ML method that establishes a relationship between the each attribute of data feature map and its target value. Estimating the output value for

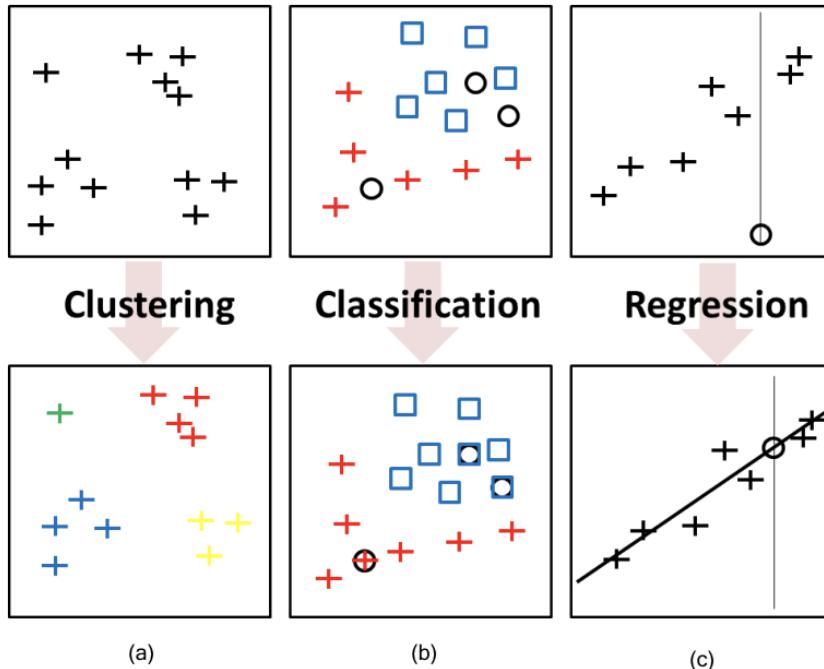


Figure 4.2 : (a) Clustering Problem, (b) Classification Problem, and (c) Regression Problem [7].

each new datum is the main goal. All true target values are known during training and testing. These known true values are used to construct the trained model, and to evaluate the testing performance. Regression and classification problems are solved by supervised learning.

The task in this thesis includes dataset in that each datum has its true label value. Thus, all ML models used in this thesis are from supervised learning algorithms.

4.1.2 Unsupervised Learning

The problems having data with unknown true values are solved by unsupervised learning. While there are hard or impossible to labeling data sets, some data sets can be deliberately left unlabeled since it requires too much time, expert people and special devices to label. Clustering tasks are constructed on these kind of data sets and solved by unsupervised learning.

4.2 Cross-Validation (CV)

As in deep learning, overfitting and underfitting are also big troubles in machine learning. Cross-validation (CV) methods can be used to overcome these problems. In the CV method, all data is divided into train and test sets. Here, it should not be

forgotten that the train and test sets must be strictly separated from each other, and any sample on test set should not be seen during training process.

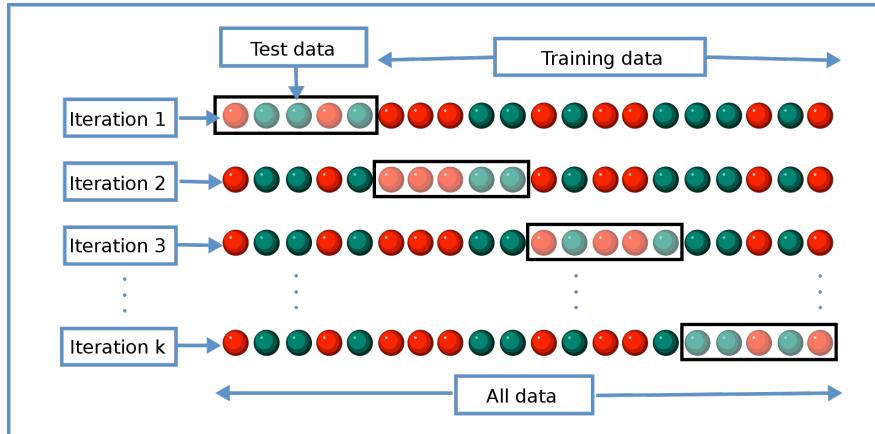


Figure 4.3 : K-Fold Cross-Validation.¹

4.2.1 K-Fold CV

On K-Fold CV, K is the hyper-parameter to tune that defines the number of folds during partitioning. The choice of K must be greater than 1 and less than the number of data. All data is divided into K folds of the same (if possible) or different sizes. The folds are enumerated from 1 to K. Here, the model is evaluated and tested ($K - 1$) times. On each iteration i , where i is an integer from 1 to K, the i^{th} fold is chosen as the test set and rest as the train set, and model forgets what it learned previously and starts from zero. Otherwise, the test data on the $(i + 1)^{th}$ iteration takes place in the train set on the i^{th} iteration and is learned by the model.

4.2.2 Leave-One-Out (LOO)

If the K of K-Fold CV is chosen as the number of data, say N, it is a special case of CV and called as leave-one-out (LOO) method. As a convenience, all data are enumerated from 1 to N. Thus, model is evaluated and tested for N times. On each iteration i , where i is an integer from 1 to N, the i^{th} datum is chosen as the test sample and rest as the train set, and model forgets what it learned previously and starts from zero. Otherwise, the test sample on the current iteration takes place in the train set on the previous iteration and is learned by the model.

¹Retrieved from: <https://towardsdatascience.com/cross-validation-c4fae714f1c5> on April 10, 2021.

Briefly, LOO is the K-Fold CV where the number of folds is chosen as the number of data.

4.3 Regularization

As discussed in Section 3.3, the training process includes the optimization of corresponding loss function in machine learning. To minimize the loss and obtain the more optimized results, the regularization term is applied to the objective loss function. In general, the regularized loss function for supervised learning can be motivated as:

$$\sum_{(x^i, y^i \in D)} \left(\frac{\text{Error}(f_w(x^i), y^i)}{n} \right) + \lambda R(f_w), \quad (4.1)$$

where n is the number of training samples, D is the training feature set with x^i as training input and y^i as the corresponding target output, f_w is the mapping function between feature map and target values, λ is a real number hyper-parameter greater than or equal to zero controlling the effect of regularization term, and R is the regularization term.

Notice that, if λ is chosen as too high, the regularizer drowns out the loss function; and, if λ is chosen as zero, there is no regularization. λ is usually chosen between 0 and 0.1; however, it must be tuned for the corresponding task.

In addition, the regularization term R has a constraint such that $R(f_w) \leq \mu$ for an appropriate real valued constant μ .

The regularization is also called as penalty to the objective function. In this section, we focus on LASSO penalty and Ridge regression concepts which are used in this thesis.

4.3.1 L1 Regularization (LASSO)

L1 regularization term, which is known as LASSO, only penalizes the high coefficients. The LASSO penalty is in the form of:

$$\lambda \|w\|_1, \quad (4.2)$$

where w is the weight coefficients to optimize, and $\|\cdot\|_1$ is the L1 norm such that:

$$\|x\| = \sum_{i=1}^p |x_i|, \quad (4.3)$$

for a vector x in the size of p .

LASSO penalty may produce non-unique solutions [26].

4.3.2 L2 Regularization (Ridge)

L2 regularization term is known as Ridge regression. The penalty term in Ridge regression is in the form of:

$$\lambda \|w\|_2^2, \quad (4.4)$$

where w is the weight coefficients to optimize, and $\|\cdot\|_2^2$ is the squared L2 norm such that:

$$\|x\|_2^2 = \sum_{i=1}^p x_i^2, \quad (4.5)$$

for a vector x in the size of p .

4.4 ML Models

In this section, the state-of-the-art ML algorithms used in the thesis are introduced and detailed.

4.4.1 Support-Vector Machines (SVM)

The current support-vector machines algorithm is published by Corinna Cortes and Vladimir Vapnik at 1995 [27]. While it is used for supervised learning problems, the form of it for clustering, which is stated as support-vector clustering [28], can be used for unsupervised learning problems.

A SVM constructs a set of hyper-planes in a high-dimensional or infinite-dimensional space. Usually, it is desired to have samples far away from hyper-planes to achieve a good separation result. The parallel hyper-plane to a SVM hyper-plane including the nearest samples of a class, which are the support-vectors, is called as the margin

of that class to the corresponding hyper-plane, and larger margins provides lower generalization errors for the classifier.

In the task the thesis has, we have binary classification problem; thus, our space is two-dimensional and we have one hyper-plane that has two margins in total. The hyper-planes and so margins differ depending on the used kernel function. The original hyper-plane algorithm proposed by Vapnik is the linear classifier whose kernel function is clearly called as linear kernel. However, to create nonlinear hyper-planes, Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggest to replace the linear function with another nonlinear kernel functions. This technique is called as kernel trick.

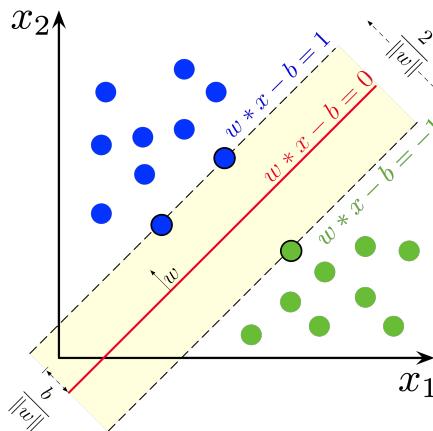


Figure 4.4 : A Hyper-plane and its margins for an SVM to a binary classification problem.²

The linear SVM hyper-plane is defined as:

$$\mathbf{w}^T \mathbf{x} - b = 0, \quad (4.6)$$

where \mathbf{x} is a p -dimensional real vector, \mathbf{w} is a weight vector normal to the hyper-plane, and b is a real valued bias parameter. Thus the distance between margins can be computed as $\frac{2}{\|\mathbf{w}\|_2}$, and we want to maximize it.

Loss Functions

First, let us name the hyper-plane and margins in Figure 4.4 as:

- **hyper-plane:**

$$H_0 = \mathbf{w}^T \mathbf{x} - b = 0, \quad (4.7)$$

²Retrieved from: https://en.wikipedia.org/wiki/Support-vector_machine on April 11, 2021.

- **margin for label +1:**

$$H_{+1} = \mathbf{w}^T \mathbf{x} - b = +1, \text{ and} \quad (4.8)$$

- **margin for label -1:**

$$H_{-1} = \mathbf{w}^T \mathbf{x} - b = -1. \quad (4.9)$$

Thus, we have the following condition and result relationships for a sample \mathbf{x}^i p-dimensional real vector with its estimated label y^i where y^i is either -1 or +1:

- if $\mathbf{w}^T \mathbf{x}^i - b \geq +1$ and $y^i = +1$, then the prediction is correct and the sample belongs to class +1;
- else, if $\mathbf{w}^T \mathbf{x}^i - b \leq -1$ and $y^i = -1$, then the prediction is correct and the sample belongs to class -1;
- else, the prediction is incorrect and the sample belongs to opposite of predicted label.

When we compound the above conditions, we yield the common condition as:

$$1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i \leq 0. \quad (4.10)$$

And finally, the margin perception loss function, called as hinge loss, is written as:

$$\text{loss}_{\text{hinge}}(\mathbf{w}, b) = \sum_{i=1}^n \max(0, 1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i). \quad (4.11)$$

Besides, the squared hinge loss is defined as:

$$\text{loss}_{\text{squaredHinge}}(\mathbf{w}, b) = \sum_{i=1}^n (\max(0, 1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i))^2. \quad (4.12)$$

The constrained optimization problem of SVM is derived as:

$$\min_w \ell(\mathbf{w}) = \min_w \frac{\|\mathbf{w}\|_2^2}{2} \quad \text{such that } (\mathbf{w}^T \mathbf{x}^i - b)y^i - 1 \geq 0, \forall i \in \{1, \dots, n\} \quad (4.13)$$

The equation (4.13) is called as the primal problem.

By using the Lagrange Function with real valued lagrange multipliers α_i greater than or equal to 0, the dual problem to solve is constructed as:

$$\max_{\alpha} \mathcal{L}(\alpha) = \max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^i y^j \mathbf{x}^{iT} \mathbf{x}^j \quad \text{such that } \sum_{i=1}^n \alpha_i y^i = 0, \quad (4.14)$$

where x_i and y_i are a p-dimensional real valued feature vector and its label, that is -1 or +1, respectively, w is a weight vector normal to the hyper-plane, b is a real valued bias parameter, and n is the number of samples.

Further details on constraint optimization of SVM, primal and dual problems, and their solutions can be found at [29, pg. 13-19].

Penalty Terms

Here λ always refers to the regularization parameter during this subsection.

- **L2 Regularizer:** According to [30, pg. 2054-2059], let first recall the primal form of SVM optimization problem in equation (4.13). We can modify the problem as:

$$\min_{\mathbf{w}, b} \left\{ \sum_{i=1}^n [1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i]_+ + \lambda \frac{\|\mathbf{w}\|_2^2}{2} \right\}, \quad (4.15)$$

where $[1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i]_+$ is the hinge loss term $\max(0, 1 - (\mathbf{w}^T \mathbf{x}^i - b)y^i)$. Yet, the squared hinge loss can be placed as well.

Then, the corresponding dual problem can be obtained as:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y^i y^j (\mathbf{x}^{iT} \mathbf{x}^j + \lambda \delta_{ij}) \quad \text{such that } \sum_{i=1}^n \alpha_i y^i = 0, \quad (4.16)$$

where α is the lagrange multiplier given in the equation (4.14) and δ_{ij} is the Kronecker's delta function such that it returns 1 for $i = j$, and 0 otherwise.

- **L1 Regularizer:** As stated in [30, pg. 2054-2059], L1 regularization is only applicable for the linear SVM problem. The L1 penalized SVM optimization problem is given by the following equation where the $loss(\mathbf{w}, b)$ is either hinge or squared hinge loss function:

$$\min_{\mathbf{w}, b} \sum_{i=1}^n loss(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_1. \quad (4.17)$$

Then, the corresponding dual problem can be obtained as:

$$\max_{\alpha} \mathcal{L}(\alpha) \quad \text{such that } \sum_{i=1}^n \alpha_i y^i = 0 \text{ and } 0 \leq \alpha_i \leq \frac{1}{\lambda}, \quad (4.18)$$

where \mathcal{L} and α_i are lagrange function and lagrange multipliers given in the equation (4.14).

Kernel Trick

Kernel trick is using a feature transformation ϕ to map samples into a new space where samples become linearly separable. Then all appearances of samples are replaced with the transformation ϕ . Now, the kernel function for p-dimensional feature vectors \mathbf{x}_i and \mathbf{x}_j is defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j). \quad (4.19)$$

This replacement affects to the whole structure of SVM optimization problem.

The most used kernel functions, that are also experienced in this thesis, are:

- **Linear function:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j. \quad (4.20)$$

- **Gaussian radial basis function (RBF):**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|_2^2), \quad (4.21)$$

for $\gamma > 0$. It is common to choose $\gamma = \frac{1}{2\sigma^2}$ where σ represents the standard deviation of original feature map.

- **Sigmoid function:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i^T \mathbf{x}_j + c), \quad (4.22)$$

for some $\alpha > 0$ and $c < 0$, and $\tanh(\cdot)$ refers the hyperbolic tangent function.

4.4.2 Logistic Regression (LR)

Logistic regression (LR) aims to solve binary classification problems. If the problem contains more than 2 classes, the model named as Multinomial LR can be used. In the logistic model, each sample $\mathbf{x} \in \mathbb{R}^d$ has a predicted value $0 \leq Pr(\mathbf{x}) \leq 1$ referring if the sample belongs to its actual label y or not. If $0.5 < Pr(\mathbf{x})$, then the prediction is true. By using the logistic sigmoid function, the model is obtained as:

$$Pr(\mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + b)}}, \quad (4.23)$$

where $\mathbf{w} \in \mathbb{R}^d$ is the weight parameter tuning the scaling, and $b \in \mathbb{R}$ is the bias term tuning the shifting in logistic sigmoid function.

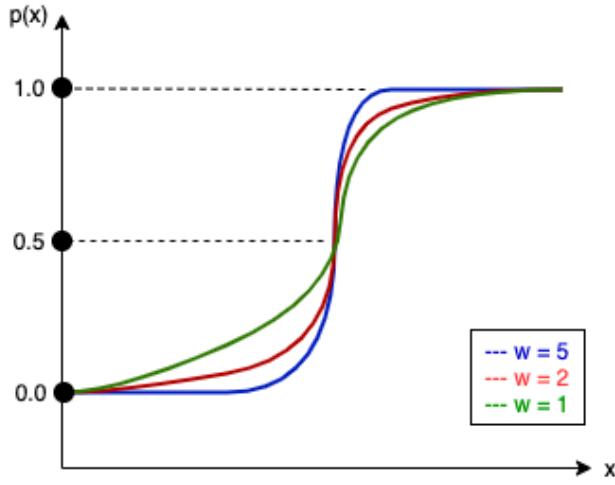


Figure 4.5 : Logistic sigmoid function with different scaling w values.

The sharpness of logistic curve increases with higher w values, and vice versa.

Simply, the hypothesis function can be obtained from sigmoid function as:

$$\ln\left(\frac{Pr(\mathbf{x})}{1-Pr(\mathbf{x})}\right) = \mathbf{w}^T \mathbf{x} + b. \quad (4.24)$$

Here, $\frac{Pr(\mathbf{x})}{1-Pr(\mathbf{x})}$ is called as odd, and $\log\left(\frac{Pr(\mathbf{x})}{1-Pr(\mathbf{x})}\right)$ as logit.

For simplicity, let define $z = \mathbf{w}^T \mathbf{x} + b$. Then, we have following constraints:

- if $y = 1$, we want $Pr(z) \approx 1$, i.e $z \gg 0$, and
- if $y = 0$, we want $Pr(z) \approx 0$, i.e $z \ll 0$.

Now, we can construct the log-loss function for a sample pair (\mathbf{x}^i, y^i) for $i \in \{1, \dots, n\}$ as:

$$loss^i(\mathbf{w}, b) = -y \ln(Pr(z^i)) + (1-y) \ln(1-Pr(z^i)). \quad (4.25)$$

Consequently, the optimization problem is yielded as:

$$\max_{\mathbf{w}, b} \ell(\mathbf{w}, b) = \max_{\mathbf{w}, b} \sum_{i=1}^n loss^i(\mathbf{w}, b). \quad (4.26)$$

Penalty Terms

Here λ always refers to the regularization parameter during this subsection.

- **L2 Regularizer:**

$$\max_{\mathbf{w}, b} \ell(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_2^2. \quad (4.27)$$

- **L1 Regularizer:**

$$\max_{\mathbf{w}, b} \ell(\mathbf{w}, b) + \lambda \|\mathbf{w}\|_1. \quad (4.28)$$

Optimizers

In this thesis, to solve the optimization problem on LR problem, we use the following the solvers presented by scikit-learn package [31] in Python programming language:

- Newton's Method (newton-cg) [32],
- Limited-Memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (lbfgs) [33],
- A Library for Large Linear Classification (liblinear) [34],
- Stochastic Average Gradient (sag) [35], and
- SAGA [36].

Different types of optimizers were used for a better approximation to the loss function, and to find the best minimization on this loss function.

Table 4.1 shows the solvers eligibility to regularizers.

Table 4.1 : Logistic Regression optimization problem solvers.

| Solver | Optimization Problem | | |
|-----------|----------------------|------------|------------|
| | No Penalty | L1 Penalty | L2 Penalty |
| newton-cg | X | | X |
| lbfgs | X | | X |
| liblinear | | X | X |
| sag | X | | X |
| saga | X | X | X |

4.4.3 K-Nearest Neighbor (KNN)

K-Nearest Neighbor (KNN) algorithm is a classification algorithm developed by Fix, E. and Hodges, L. in 1951 [37]. KNN can be used for regression as well; however, at this time the, the output for an object is not a class label, but the property value which

is the average of the values of its k nearest neighbors. For both usage cases, KNN is a supervised learning algorithm.

Algorithm

Below, the basic algorithm for KNN is explained for n samples $\mathbf{x}^i \in \mathbb{R}^d$ and their corresponding class labels $c^i \in \{-1, +1\}$. Let the target pair is referred by (\mathbf{x}^*, c^*) .

1. The K nearest neighbors of \mathbf{x}^* is found with respect to distance between target \mathbf{x}^* and all other samples. To calculate the distance, different metric functions may be used. The used metrics in this thesis are:

- **Euclidean Distance:**

$$d(\mathbf{x}^*, \mathbf{x}^i) = \sqrt{\sum_{j=1}^p (\mathbf{x}^*{}_j - \mathbf{x}^i{}_j)^2}, \quad (4.29)$$

- **Manhattan Distance:**

$$d(\mathbf{x}^*, \mathbf{x}^i) = \sum_{j=1}^p |\mathbf{x}^*{}_j - \mathbf{x}^i{}_j|, \text{ and} \quad (4.30)$$

- **Chebyshev Distance:**

$$d(\mathbf{x}^*, \mathbf{x}^i) = \max(|\mathbf{x}^*{}_1 - \mathbf{x}^i{}_1|, |\mathbf{x}^*{}_2 - \mathbf{x}^i{}_2|, \dots, |\mathbf{x}^*{}_p - \mathbf{x}^i{}_p|). \quad (4.31)$$

2. The class label for \mathbf{x}^* is estimated by majority voting around K nearest neighbors of target sample found in step 1. Majority voting makes its decision by looking for which class label has the majority in appearance around K nearest neighbors. Hence, K is preferred as an odd number. For example, if the K is chosen as 3 and 3 nearest neighbors has class labels as $\{+1, -1, +1\}$, then majority voting predicts the target label as +1 since it appears more than label -1. Besides, it can be considered that all of the voters may not have the same effect. In this scenario, one can use weighted majority voting which allows the closer neighbor in K nearest neighbors to have the higher effect. That is, the weight of a voter is the multiplicative inverse of its distance to the target sample. For example, assume that the K is chosen as 3 and 3 nearest neighbors has class labels as $\{+1, -1, +1\}$ with their distances to the target sample as $\{3, 1, 2\}$ respectively. Then the weights of the neighbors are considered as $\{\frac{1}{3}, 1, \frac{1}{2}\}$ or $\{2, 6, 3\}$ respectively. Here, the label -1 has the majority; thus, the target label is predicted as -1.

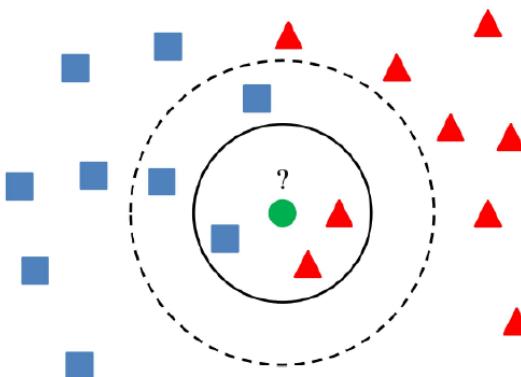


Figure 4.6 : KNN example illustration to detect the class for the green sample in the middle.³

At Figure 4.6, when $K = 3$, it is labeled as red both by majority voting and weighted majority voting. However, when $K = 5$, it may change whether to use majority voting or weighted majority voting. It is labeled as blue by majority voting; but, it may be labeled as red by the weighted majority voting.

Finding Neighborhood

In this thesis, three algorithms presented by scikit-learn package [31] in Python programming language are used.

- **Brute Force:** Distances between all pairs of samples are computed. Brute force may be quite efficient for small sized dataset.
- **K-Dimensional Tree:** Since brute force becomes inefficient by the increase of dataset size, K-Dimensional (K-D) tree may be used to reduce the required number of computations. If samples x_0 and x_1 are very distant, and x_1 and x_2 are very close, then we can conclude that x_0 and x_2 are very distant too. To create a K-D tree structure, the data space partitioned along the Cartesian axes. Because of the curse of dimensionality, the construction of K-D tree may be inefficient for large dimensions. Further information can be found at [38].
- **Ball Tree:** Ball tree algorithm was developed to solve the size and dimensionality problems. The data space are partitioned along the Spherical axes to construct the tree structure; hence it is more costly than K-D tree. However the structure itself

³Retrieved from: <https://ai.plainenglish.io/k-nearest-neighbors-from-scratch-633dfbeac740> on April 17, 2021.

is more effective, especially in high dimensions. Further information can be found at [39].

4.4.4 Linear Discriminant Analysis (LDA)

The linear discriminant analysis (LDA) is a supervised learning algorithm that is generalized from discriminant analysis developed by Sir Ronald Fisher, who was an statistician, geneticist, and academic, in 1936.

Derivation

A discriminant function $g(\mathbf{x})$ from \mathbb{R}^d to \mathbb{R} is, generally, a monotonically increasing function defining a boundary between two groups or classes. Here $\mathbf{x} \in \mathbb{R}^d$ is a sample with d features. $g_i(\mathbf{x})$ is the probability of \mathbf{x} belonging to class c_i as given below:

$$g_i(\mathbf{x}) = Pr(c_i|\mathbf{x}) = \frac{Pr(\mathbf{x}|c_i)Pr(c_i)}{Pr(\mathbf{x})}, \quad (4.32)$$

where $Pr(c_i|\mathbf{x})$ is the likelihood of observing \mathbf{x} given class c_i , $Pr(c_i)$ is the prior probability of belonging to class c_i , and $Pr(x)$ is the marginal likelihood of observing \mathbf{x} . Here, we discard the $Pr(x)$ since observing \mathbf{x} is independent of class.

For normal data distribution, we can take the logarithm of $g(\mathbf{x})$ since it is monotonically increasing:

$$g_i(\mathbf{x}) = \ln(Pr(c_i|\mathbf{x})) = \ln(Pr(\mathbf{x}|c_i)) + \ln(Pr(c_i)). \quad (4.33)$$

Here, we suppose that the class density $Pr(x|c_i)$ is normal distribution with mean μ_i and variance Σ_i where Σ describes the covariance matrix considered as $\sigma^2 I$.

$$Pr(\mathbf{x}|c_i) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp \left[-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right]. \quad (4.34)$$

By inserting equation (4.34) into equation (4.33), we yield:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) - \frac{d}{2} \ln(\frac{\pi}{2}) - \frac{1}{2} \ln |\Sigma_i| + \ln(Pr(c_i)). \quad (4.35)$$

Finally, since $\frac{d}{2} \ln(\frac{\pi}{2})$ is constant, we obtain the linear discriminant function for class i as:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \ln |\boldsymbol{\Sigma}_i| + \ln(Pr(c_i)). \quad (4.36)$$

This linear discriminant function outputs the likelihood of \mathbf{x} to be in the class i.

There are two presumptive cases on the linear discriminant function.

- $\boldsymbol{\Sigma}_i = \sigma^2 I$: All features have different means, but the same variants.

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \ln(\sigma^{2d}) + \ln(Pr(c_i)), \\ g_i(\mathbf{x}) &= -\frac{1}{2\sigma^2}(\mathbf{x} - \boldsymbol{\mu}_i)^T (\mathbf{x} - \boldsymbol{\mu}_i) + \ln(Pr(c_i)) \quad \text{since } \frac{1}{2} \ln(\sigma^{2d}) \text{ is constant,} \\ g_i(\mathbf{x}) &= -\frac{1}{2\sigma^2}(\mathbf{x}^T \mathbf{x} - \mathbf{x}^T \boldsymbol{\mu}_i - \boldsymbol{\mu}_i^T \mathbf{x} - \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i) + \ln(Pr(c_i)), \\ g_i(\mathbf{x}) &= -\frac{1}{2\sigma^2}(-2\mathbf{x}^T \boldsymbol{\mu}_i - \boldsymbol{\mu}_i^T \boldsymbol{\mu}_i) + \ln(Pr(c_i)) \quad \text{since } \mathbf{x}^T \mathbf{x} \text{ is constant and } \mathbf{x}^T \boldsymbol{\mu}_i = \boldsymbol{\mu}_i^T \mathbf{x}, \\ g_i(\mathbf{x}) &= \frac{\boldsymbol{\mu}_i^T}{\sigma^2} \mathbf{x} + \left(\ln(Pr(c_i)) - \frac{\boldsymbol{\mu}_i^T \boldsymbol{\mu}_i}{2\sigma^2} \right). \end{aligned} \quad (4.37)$$

It can be seen that, the final discriminant function in equation (4.37) is a linear function.

- $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}$: Covariance matrices are all arbitrary, but equal across classes.

$$\begin{aligned} g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \ln |\boldsymbol{\Sigma}| + \ln(Pr(c_i)), \\ g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) + \ln(Pr(c_i)) \quad \text{since } \ln |\boldsymbol{\Sigma}| \text{ is constant,} \\ g_i(\mathbf{x}) &= \frac{-1}{2}(\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x} - \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \ln(Pr(c_i)), \\ g_i(\mathbf{x}) &= \frac{-1}{2}(-2\mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i) + \ln(Pr(c_i)) \quad \text{since } \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \mathbf{x} \text{ is constant} \\ &\text{and } \mathbf{x}^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i = \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \mathbf{x}, \\ g_i(\mathbf{x}) &= (\boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1}) \mathbf{x} + \left(\ln(Pr(c_i)) - \frac{1}{2} \boldsymbol{\mu}_i^T \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_i \right). \end{aligned} \quad (4.38)$$

It can be seen that, the final discriminant function in equation (4.38) is a linear function.

Penalty Terms

⁴Retrieved from: https://scikit-learn.org/stable/modules/lda_qda.html on April 18, 2021.

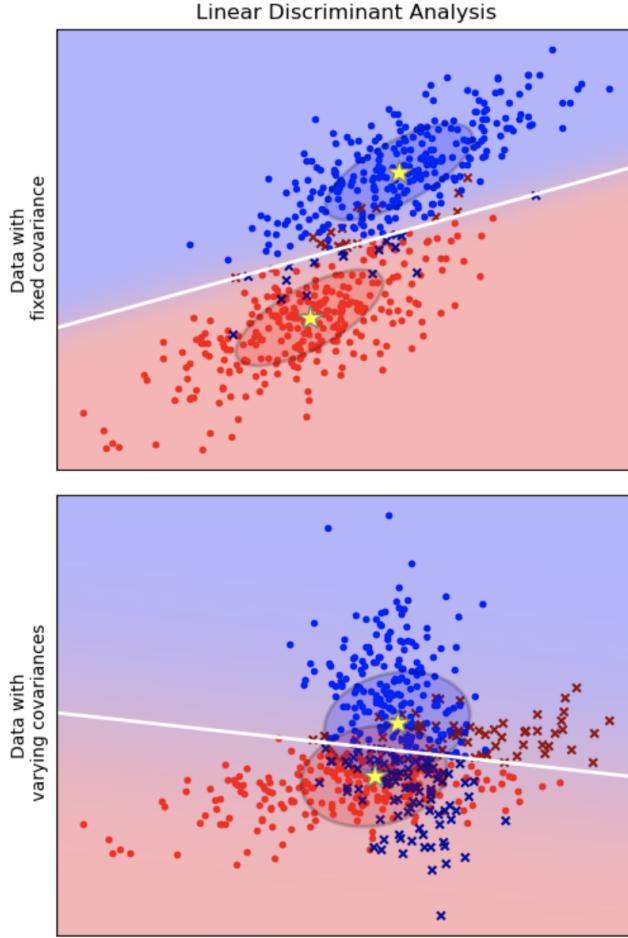


Figure 4.7 : LDA process for fixed and varying covariances.⁴

Since high dimensionality may cause LDA not to be optimal and hard to operate over matrices. That is why, a simple modification on covariance matrix is made as [40]:

$$\widehat{\Sigma} = \alpha \Sigma + (1 - \alpha) I, \quad (4.39)$$

for some real valued α in between the range of $[0, 1]$. We can consider this regularization as what we are used to use in Section 4.3:

$$\widehat{\Sigma} = \lambda \Sigma + I \quad , \text{or} \quad (4.40)$$

$$\widehat{\Sigma} = \Sigma + \lambda I, \quad (4.41)$$

for positive real valued regularization parameter λ . Consequently, the regularized LDA is formulated as:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \widehat{\boldsymbol{\Sigma}}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{1}{2} \ln |\widehat{\boldsymbol{\Sigma}}_i| + \ln(Pr(c_i)). \quad (4.42)$$

Solvers

In this thesis, to solve the LDA functions, we use the solvers presented by scikit-learn package [31] in Python programming language.

Further information about estimators provided by scikit-learn package can be found at [41].

5. EXPERIMENTS

All the code used for the experiments explained in this Chapter can be found on master branch of the GitHub public repository:

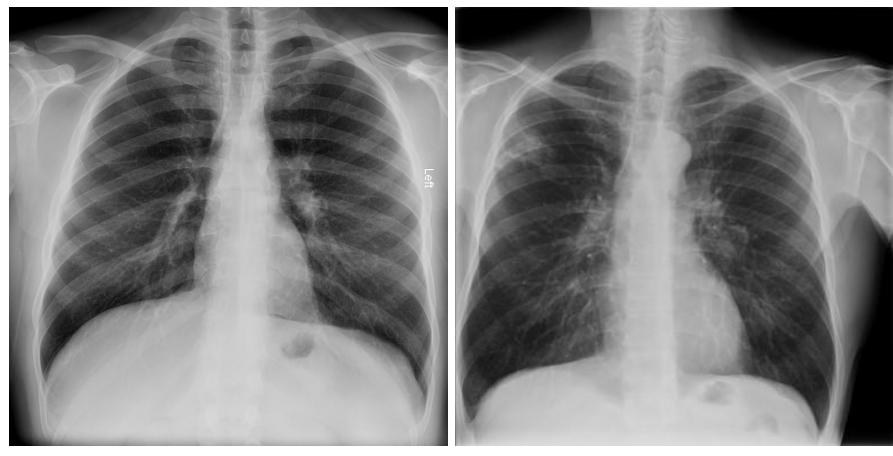
<https://github.com/ozanguldali/modelsWithLASSO>.

Python programming language in version of 3.7.6 combining with Clang 4.0.1 was used to train, test, and classify on our environment. PyTorch module with the torch version of 1.5.0, the torchsummary version of 1.5.1 and the torchvision version of 0.6.0, and Scikit-learn library with the version of 0.23.2 were used for CNN and ML coding respectively. All experiments were performed on the Google Colaboratory using its GPU with the number of workers as four.

5.1 Dataset

The basis data was obtained from GitHub platform repository which can be reached via link <https://github.com/ieee8023/covid-chestxray-dataset/>. On the GitHub repository, the data are placed in "images" folder and the information file about data is in 'metadata.csv' file.

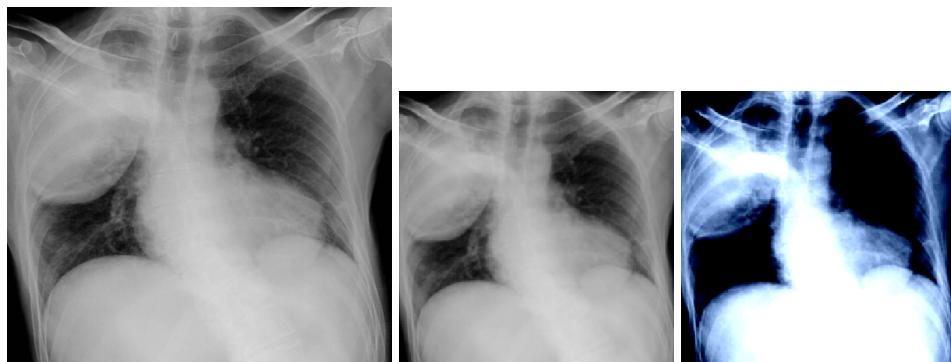
To construct the dataset, the metedata.csv file was parsed and data were filtered depending on whether it has age and sex info or not, the its finding includes "COVID-19" term or not, its modality is "X-ray" or not, and its view is "PA" or not. First, data having sex and age info, modality as "X-ray", and view as "PA" were chosen. Then two main classes were constructed by separating the chosen data whether its finding includes "COVID-19" or not. The data including "COVID-19" in its finding were labeled as "COVID-19" and the others were labeled as "non-COVID-19". Finally, in the ratio of 0.8, the labeled data were divided into train and test sets. At the end, 131 "COVID-19" and 123 "non-COVID-19" labeled data were obtained and splitted into train and test sets. Train set includes 105 "COVID-19" and 98 "non-COVID-19" labeled data, while test set includes 26 "COVID-19" and 25 "non-COVID-19".



(a) Train COVID-19 Data (b) Test non-COVID-19 Data

Figure 5.1 : Dataset Samples.

Data are appeared in different formats such that .png, .jpg, .jpeg, in different aspect ratios and dimensions such as 384×384 , 462×450 , etc, and in different sizes such as 606 KB, 69 KB, 1 KB, etc. Thus, before using the data, each datum was resized to 224×224 , center cropped, gray scaled and normalized by the mean of (0.485, 0.456, 0.406) and the standard deviation of (0.229, 0.224, 0.225) for red, blue and green channels respectively.



(a) Original (826×768)

(b) Resized, Square
Cropped and Gray-Scaled

(c) Normalized

Figure 5.2 : Image Transformation Steps - Test COVID-19 Sample.

5.2 Data Augmentation

While data sizes on classes are unbalanced, one or more data augmentation techniques are used on train data to balance them. Furthermore, for a balanced dataset, one can use data augmentation to increase the train data size.

Data augmentation techniques on visual data can be classified in 2 groups such that position augmentation and color augmentation. Position augmentation takes the

original data and applies the desired operations from followings: flipping, rotation, translation and scaling. On the other hand, during the color augmentation, the augmented datum is obtained by changing one or more of followings: hue, saturation, brightness and contrast. If n kind augmentation choice is used to m number of data, at the end $m*(n+1)$ data are obtained. That is the summation of n augmented datum for each sample of m data, and the original m samples.

In the thesis, position augmentation with horizontal flip, vertical flip, 90 degrees of rotation, 180 degrees of rotation and 270 degrees of rotation is used on the whole train data. These augmented data are not physical, i.e. not saved as files, and only used on CNN training process. Data augmentation is not used on ML processes.

The number of data on train and test sets for each class after augmentation can be seen on Table 5.1.

Table 5.1 : The detailed number of data.

| Class | Original Set | Train Set | | Test Set |
|--------------|--------------|-----------|--------------------|----------|
| | | Chosen | After Augmentation | |
| COVID-19 | 131 | 105 | 630 | 26 |
| non-COVID-19 | 123 | 98 | 588 | 25 |
| TOTAL | 254 | 203 | 1218 | 51 |

5.3 Training and Testing on Convolutional Neural Network Models

All CNN experiments was made on pre-trained CNN models. That is, the corresponding model weights was used with different strategies to feed the models.

Since our dataset is obviously small and the data content is different from ImageNet dataset [17], we experimented to train the entire model, and to freeze the first few layers and train others. The frozen layers for each experiment model are:

- the first convolution layer, and its following ReLU and maximum pooling layers for AlexNet architecture,
- the first convolution layer, and its following batch normalization, ReLU and maximum pooling layers for ResNet architectures, and

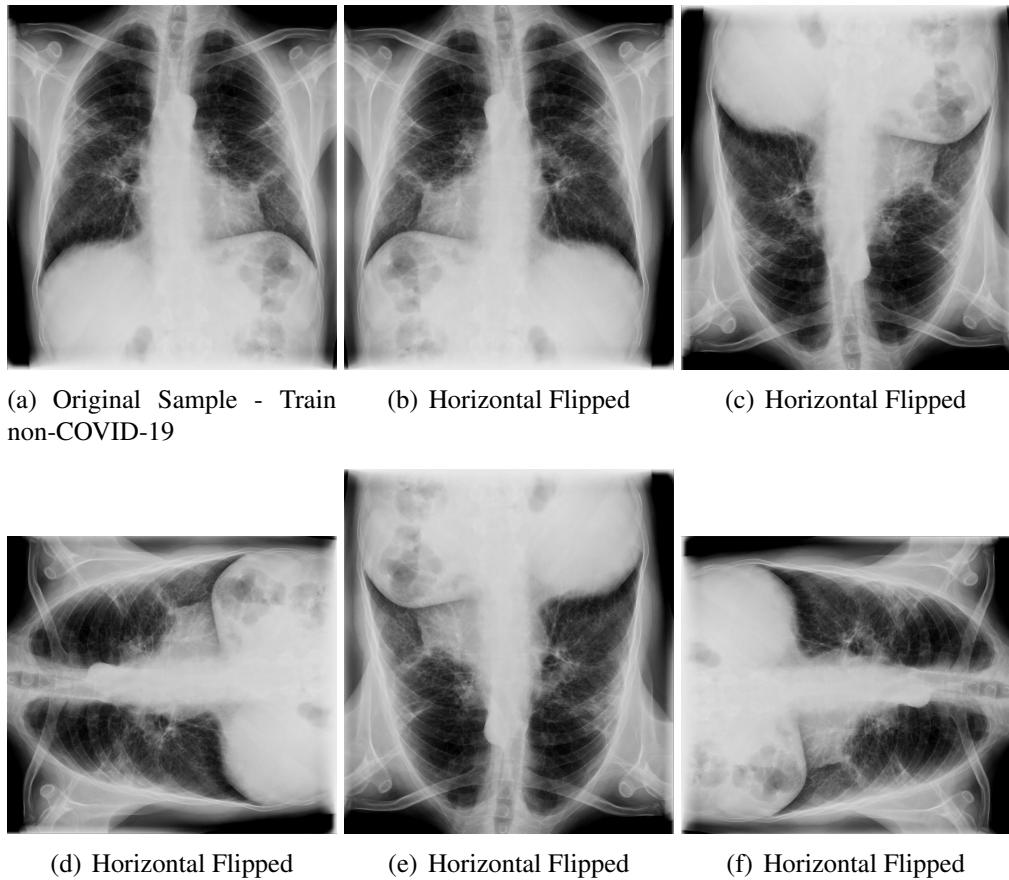


Figure 5.3 : Original Sample and Augmentation.

- the first six layers which are in order of convolution, batch normalization, ReLU, convolution, batch normalization, ReLU and maximum pooling layers for VGG architectures.

The comparison results have shown that, training the entire model instead of freezing the stated number of layers above gives more minimal losses and more accurate classification results. However, it must be remarked that, even the number of frozen layers are not much, since the number of parameters are less on this technique than entire model training, the computation time of each batch is observably shorter as well.

Train data were set to be shuffled on each epoch and the batch size was experimentally set from 2^4 to 2^6 . The number of epoch was set to 50, and the validation period was set as 1/50. That is, before the first training epoch and after each training epoch, validation process was applied on test set. On each validation, the saved model weights files for corresponding configurations were checked. If there exists no saved file having greater accuracy percentage than the current state, the current model weights are saved

as a new model weight *.pth file with the name including the validation result and training configurations. The saved CNN model weights files can be found under the */cnn/saved_models* directory of project repository at <https://github.com/ozanguldali/modelsWithLASSO>. After the last training epoch, test process is applied rather than validation, and the same accuracy comparison over saved *.pth files is executed. That is why, the number of validation accuracy results are 50 whereas the number of validation losses are 49. The reason why we limited the epoch size as 50 is that, the models were overfitting after this number, and even before sometimes.

By our weights file comparison rule, there may be multiple *.pth weights files for the same configuration and accuracy value. This time, the train loss values, before the validation or test result obtained, and the validation loss values, if available, are considered. Here, the less loss is better to choose.

The cross-entropy loss function, explained in Section 3.2, was used for all CNN models. To optimize it, Adam, SGD Momentum and Adam with decoupled weight decay optimizers were experimented with various learning rate, momentum and weight decay parameter values. Moreover, the learning rate was experimentally scheduled by the reducing rate as 0.1 for each quarter of total epoch size. However, the scheduler approach drove the models to overfitting.

Final hyper-parameters experienced as the best for each model and each optimizer differ. The best ones for ResNet-50 model can be seen in Table 5.2. Furthermore, on our experiments, we encountered the optimal batch size, in terms of computational time and obtained losses, as 2^4 . Hence, the number of iteration on each epoch was 77 for augmented train data, and 4 for test data which was used on both test and validation processes.

Table 5.2 : CNN models final hyper-parameters.

| Optimizer | Learning Rate | Momentum | Weight Decay |
|------------------|----------------------|-----------------|---------------------|
| SGD Momentum | 1e-4 | 0.9 | - |
| Adam | 1e-5 | - | 1e-3 |
| AdamW | 1e-5 | - | 1e-4 |

All pre-trained CNN models were experimented by setting the final layer out feature size as the class size, i.e. 2. Furthermore, for all architectures experienced in this

thesis, a modification is applied on the classification section as adding a block of ReLU, dropout and fully-connected layers in order after the last fully-connected layer.

For ResNet-50 model, the total parameters, so the total trainable parameters on full-training, is 25,559,034 for one sample. For an input image in the size of 0.59 MB, forward and backward pass size is 309.48 MB and parameters size is 97.50 MB. Hence, the estimated total size of one iteration on this image is 407.57 MB according to PyTorch.

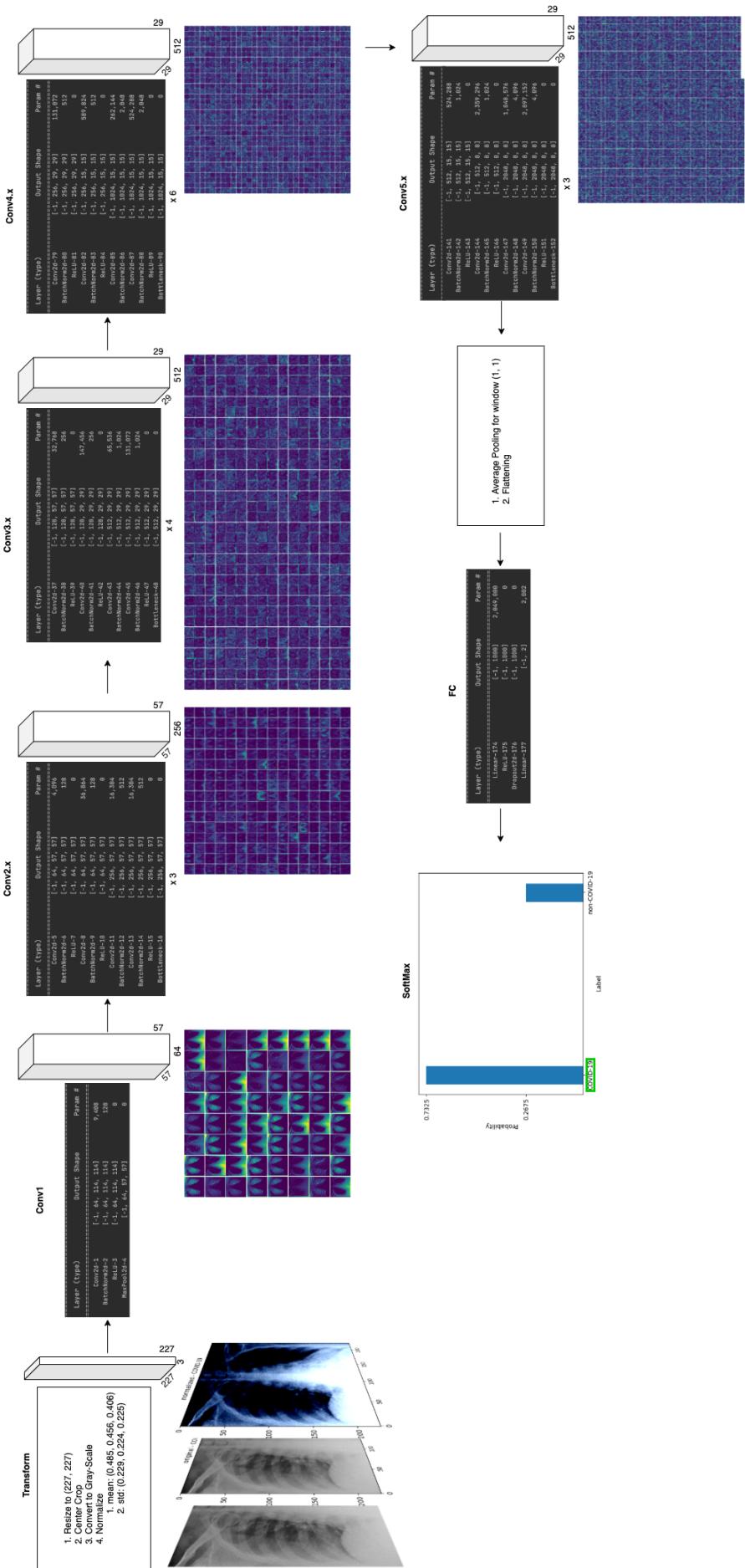


Figure 5.4 : The sample visualization representing one iteration on ResNet-50 architecture and class probability result for COVID-19 labeled data from our dataset. The higher resolution version of image is available at https://github.com/ozanguldali/modelsWithLASSO/blob/master/figures/ressnet50_visual.png

The higher resolution version of images in Figure 5.4, such as the visualizations of convolution blocks, is available at directory [https://github.com/ozangulda- li/modelsWithLASSO/blob/master/figures](https://github.com/ozangulda/li/modelsWithLASSO/blob/master/figures).

5.4 Deep Feature Extraction

After training, validating and testing the dataset on pre-trained CNN models, the deep feature extraction progress was started. Since the best results according to accuracy are stored, these model weights files are used to extract deep features from their corresponding CNN architectures.

Here, the modifications on CNN models stated at previous section show its another purpose. The deep feature are extracted from these added or replaced fully-connected layers as:

- using the third fully-connected layer for AlexNet architecture,
- using the first fully-connected layer for ResNet architectures, and
- using the third fully-connected layer for VGG architectures.

The deep features extracted from CNN models are used to feed machine learning algorithms to improve the classification success.

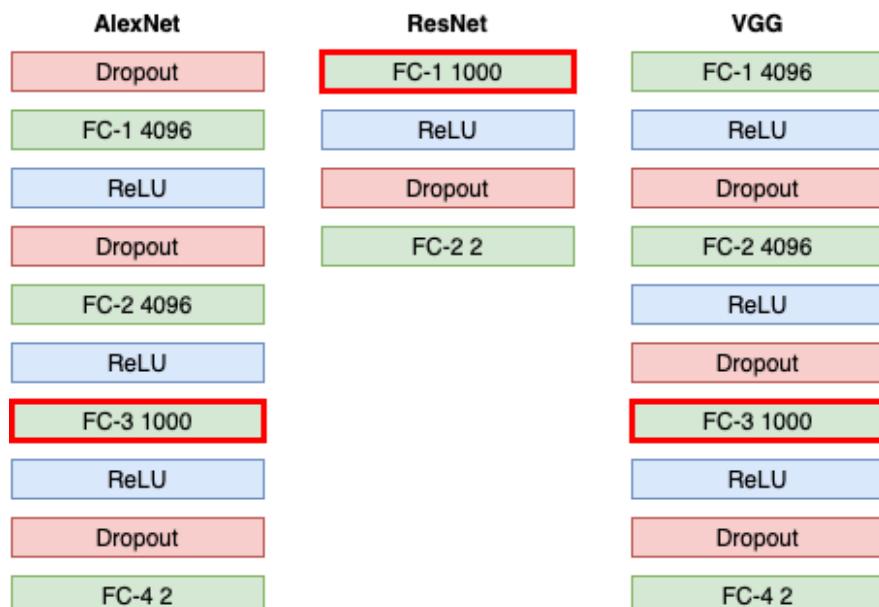


Figure 5.5 : Modified classification blocks where bold red windowed fully-connected layers denotes the deep feature extraction state.

5.5 Forming the Feature Maps

In this study, we have two types of feature matrices and one consisting of their conjunction. The first feature matrix includes the deep features extracted by the help of CNN models. Instead of image segmentation, Red-Green-Blue (RGB) color codes or manual image partition labeling, we use the features derived from deep learning process. The number of features are 1000 for each CNN architecture, because of the output size of used fully-connected layers. We denote the matrix of deep features as X_{cnn} and the feature size as $d_{cnn} = 1000$.

On the other part, we have demographic information for each sample, and that is the why our dataset is restricted. We denote the matrix of demographic information as X_{info} and the feature size as $d_{info} = 2$. Age information was categorized to 5 groups such as under 18, between 18 and 37, between 39 and 59, between 60 and 79, and over 80 ages.

Hence, the merged features matrix denoted by X_{all} has the feature size as $d_{all} = 1002$.

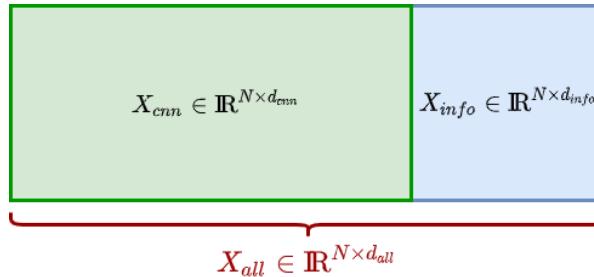


Figure 5.6 : Feature maps where $d_{cnn} = 1000$, $d_{info} = 2$, and $d_{all} = d_{cnn} + d_{info} = 1002$.

5.6 Data Pre-Processing

5.6.1 Data Standardization

Machine learning estimators commonly requires data standardization; because, the data may not be always in the form of standard normal distribution. There are three basic ways to standardize the data such as 1) centering data with the mean, 2) scaling data with the standard deviation, and 3) centering and scaling data with the main and standard deviation. In the experiments, we only centered the data. That is, we set μ as the mean of training samples and σ as 1 in the equation (5.1):

$$Z = (X - \mu * I) / \sigma. \quad (5.1)$$

5.6.2 Data Normalization

Data normalization scales each individual sample to have a unit norm in the rational numbers range [0.0, 1.0]. Two ways of normalization was experimented which are least absolute technique, known as L1 normalization, and least squares technique, known as L2 normalization.

- **L1 Norm:** Scales the sum of the absolute differences of each individual feature on a sample to 1. That is, scales the S_{L1} to 1 in the equation (5.2):

$$S_{L1} = \sum_{i=1}^d |y_k - X_{k,i}|, \quad (5.2)$$

where d is the number of features, X_k is the k^{th} sample and $X_{k,i}$ is the i^{th} feature value of corresponding sample, and y_k is the target value for corresponding sample. Here, the scaling ratio is $1/S_{L1}$; thus, X_k/S_{L1} gives the L1 normalized feature values for this sample.

- **L2 Norm:** Scales the sum of the squares of differences of each individual feature on a sample to 1. That is, scales the S_{L2} to 1 in the equation (5.3):

$$S_{L2} = \sum_{i=1}^d (y_k - X_{k,i})^2, \quad (5.3)$$

where d is the number of features, X_k is the k^{th} sample and $X_{k,i}$ is the i^{th} feature value of corresponding sample, and y_k is the target value for corresponding sample. Here, the scaling ratio is $\sqrt{1/S_{L2}}$; thus, $X_k/\sqrt{S_{L2}}$ gives the L2 normalized feature values for this sample.

We used the L2 normalization as the final settings of the data pre-processing pipeline.

5.7 Hyper-Parameter Tuning

For each machine learning algorithm, there exists several hyper-parameters. Since the hyper-parameter settings cannot be generalized, and so differ for each problem and dataset, finding the optimal combination of hyper-parameters is a separate mission.

Which characteristic to look for when choosing the most optimal hyper-parameter combination depends on the problem and the researcher. In this thesis, we decide according to accuracy values. There are three common optimization method such that bayesian optimization, grid search and random search.

- **Grid Search:** all combinations between hyper-parameter options are experimented. For instance, if there exists m number of hyper-parameters and each of them has a_i options where a_i 's are positive integers for $i \in \{1, 2, \dots, m\}$, then the total number of grid search iterations are $\prod_{i=0}^m a_i$. Moreover, if K-Fold cross validation is applied, the number of iterations are raised to $K \times \prod_{i=0}^m a_i$.
- **Random Search:** randomly chosen hyper-parameter option combinations over all are experienced. The number of combinations to be tried are determined by researcher as less than the total number of combinations.
- **Bayesian Optimization:** this methodology is based on Bayesian Principle. While the combinations of hyper-parameter options are experimented, the previously known results leads to create a new combination space by using bayesian rule. This time, the choice of combinations are not random, but depend on the useful or irrelevant combination estimation.

In thesis, we experimented grid search with 10-Fold cross validation.

5.8 Training and Testing on Machine Learning Models

The partitioning of dataset into train and test sets are preserved on machine learning stage. The only difference is, data augmentation is not applied on train set.

All three types of feature matrices were experienced separately. For each, 10-Fold cross validation was applied on train set only by using grid search to find the generalized optimal combination of hyper-parameter options. To decide the best combination, the first criterion was set as accuracy value, and if there exists multiple same values, the second criterion was determined as the computation time. If there exists combinations such that two of criteria are the same, the last experienced combination was chosen.

For the repeatability of our work, we initialized the Python random number generation with different seeds. In other words, the seed parameter of Python random library is

one of our hyper-parameters, and we chose the best seed among our experiments to achieve final results.

Here, a parenthesis must be open for linear discriminant analysis algorithm. Since there is no library in Python that supports regularization on LDA, we used TULIP package [42] on version 1.0.1 in R programming language with the version of 4.0.3 to implement L1 regularized LDA model. Furthermore, grid search tuning was not applied to this model. Hyper-parameter tuning process was applied only for seed and regularization parameters.

Penalty terms, which are no penalty, Lasso and Ridge, were not considered as hyper-parameters, because they were experienced and compared separately. All hyper-parameter options to be combined can be found in Tables 5.3 - 5.6.

Table 5.3 : SVM algorithm hyper-parameters to tune.

| Hyper-Parameter | Ridge Penalty | Lasso Penalty |
|--------------------------|---|---|
| Kernel Function | linear, sigmoid, rbf | linear |
| Regularization Parameter | 5e-5, 1e-4, 2e-4, 5e-4, 5e-3, 1e-2, 2e-2, 5e-2, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0 | 5e-5, 1e-4, 2e-4, 5e-4, 5e-3, 1e-2, 2e-2, 5e-2, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0 |

Table 5.4 : LR algorithm hyper-parameters to tune.

| Hyper-Parameter | No Penalty | Lasso Penalty | Ridge Penalty |
|--------------------------|--------------------------------|---|---|
| Solver Function | newton-cg, lbfgs, sag, saga | liblinear, saga | newton-cg, lbfgs, sag, saga, liblinear |
| Regularization Parameter | - | 5e-5, 1e-4, 2e-4, 5e-4, 5e-3, 1e-2, 2e-2, 5e-2, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0 | 5e-5, 1e-4, 2e-4, 5e-4, 5e-3, 1e-2, 2e-2, 5e-2, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0 |

Table 5.5 : KNN algorithm hyper-parameters to tune.

| Hyper-Parameter | No Penalty |
|-------------------------|---|
| Number of Neighbors | 3, 5, $\sqrt{\text{the number of samples}}$ |
| Type of Majority Voting | uniform, weighted |
| Neighborhood Finder | brute force, ball tree, k dimensional tree |
| Metric Function | euclidean, manhattan, chebyshev |

Table 5.6 : LDA algorithm hyper-parameters to tune.

| Hyper-Parameter | No Penalty | Lasso Penalty |
|---|------------------|---|
| Regularization Parameter | - | 5e-5, 1e-4, 2e-4, 5e-4, 5e-3, 1e-2, 2e-2, 5e-2, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0 |
| Solver Function | svd, lsqr, eigen | - |
| Computing the Weighted Within-Class Covariance (for svd solver) | Yes, No | - |

When the optimal hyper-parameter combination was determined with the aim of generalization, the model weights obtained by training on corresponding machine learning algorithm with these hyper-parameters are used on testing stage. The final results are achieved by testing on this fitted model. Furthermore, another grid search was applied on determined train and test data, and the results were compared by the generalized ones.

6. RESULTS

6.1 Performance Measurement

6.1.1 Confusion Matrix

The confusion matrix is a table that constructed to visualize the performance of classifier. It includes actual and predicted results for each class on the corresponding task. In our study, the rows specifies the actual results while columns specifies predicted ones.

During the performance measurement, the focused class is called as Positive (P), and the others are as Negatives (N). For binary classification problems, which is as in our study, one class can be directly specified as P and the other as N. We indicated the COVID-19 label as Positive class, and the non-COVID-19 label as Negative class.

If a sample in actual P class is predicted as in P class, then it is a True Positive (TP) sample, otherwise it is False Positive (FP). On the other hand, if a sample in actual N class is predicted as in P class, then it is a False Negative (FN) sample, otherwise it is True Negative (TN).

From the definitions, the following equalities can be easily derived: $P = TP + FN$ and $N = FP + TN$.

Table 6.1 : Sample Confusion Matrix.

| CONFUSION MATRIX | | Actual | |
|------------------|---------------------|-----------------|---------------------|
| | | P (COVID-19) | N (non-COVID-19) |
| Predicted | P (COVID-19) | TP | FN |
| | N (non-COVID-19) | FP | TN |

6.1.2 The Analysis of Confusion Matrix

Various meaningful ratios can be extracted from the confusion matrix. In this section, we study the significant ones which were also used in the thesis to measure the performance on our tasks.

We computed the metrics Sensitivity, Specificity, Precision, Accuracy and F1-Score in a weighted way. That is, these stated metrics were computed for each class and then taken the weighted average as in:

$$\frac{\sum_{i=1}^m (\text{the metric value for class } i) \times (\text{the number of test samples in class } i)}{(\text{the total number of test samples})}. \quad (6.1)$$

Recall that the task we worked on is a binary classification problem. That is, m was 2 in our computations. So that, the average value for metric M of an experiment, where M is one of Sensitivity, Specificity, Precision, Accuracy and F1-Score metrics, was computed as:

$$M = \frac{M_{c_1} \times n_{c_1} + M_{c_2} \times n_{c_2}}{n_{c_1} + n_{c_2}}, \quad (6.2)$$

where M_{c_1} , M_{c_2} , n_{c_1} and n_{c_2} refer the value of M for class 1, the value of M for class 2, the number of test samples in class 1 and the number of test samples in class 2 respectively.

6.1.2.1 Sensitivity (True Positive Rate - TPR)

Sensitivity (TPR) measures the ratio of positive class samples predicted correctly over all samples predicted as positive as given below:

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}. \quad (6.3)$$

6.1.2.2 Specificity (True Negative Rate - TNR)

Specificity (TNR) measures the ratio of negative class samples predicted correctly over all samples predicted as negative as given below:

$$TNR = \frac{TN}{N} = \frac{TN}{FP + TN}. \quad (6.4)$$

6.1.2.3 Precision (Positive Predictive Value - PPV)

Precision (PPV) measures the ratio of correctly predicted positive class samples over all samples having actual positive label as given below:

$$PPV = \frac{TP}{TP + FP}. \quad (6.5)$$

6.1.2.4 Accuracy (ACC)

Accuracy (ACC) measures the ratio of correctly predicted samples over all samples as given below:

$$ACC = \frac{TP + TN}{P + N} = \frac{TP + TN}{TP + FP + FN + TN}. \quad (6.6)$$

6.1.2.5 F1 Score

F1 Score is the harmonic mean of sensitivity and precision as given below:

$$F_1 = 2 \times \frac{PPV \times TPR}{PPV + TPR} = \frac{2 \times TP}{2 \times TP + FP + FN}. \quad (6.7)$$

6.1.2.6 Area Under the Curve (AUC Score)

Area Under the Curve (AUC Score) measures the probability of classifier to rank a randomly chosen positive sample higher than a random chosen negative sample (under the assumption that "positive" ranks are higher than "negative" ranks) for normalized samples. The computation of AUC Score value is as given below:

$$TPR(t) : t \rightarrow y(x),$$

$$FPR(t) : t \rightarrow x, \text{ and}$$

$$AUC = \int_0^1 TPR(FPR^{-1}(x)) dx = \int_{-\infty}^{\infty} TPR(t) \times FPR'(t) dt. \quad (6.8)$$

6.2 Convolutional Neural Network Results

Here, we report the results of convolutional neural network experiments explained in Section 5.3. As stated, AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG-16 and VGG-19 architectures were trained initialized by their pre-trained weights with three

different optimizers such that SGD with momentum, Adam and Adam with decoupled weight decay. The accuracy and loss curves for three different optimizers on train and validation processes can be viewed on Figures 6.1 - 6.6 whose x-axis represent the number of epochs. The plots were created by TensorBoard platform prepared with the Python TensorFlow package version 2.3.1. Additional required package versions are as follows: 1) TensorFlow Addons with the version of 0.11.2, 2) TensorFlow Estimator with the version of 2.3.0, 3) TensorFlow Hub with the version of 0.10.0, and 4) TensorFlow Probability with the version of 0.10.0.

The results can be also seen in Table 6.2 including accuracy, sensitivity, specificity, precision and F1 score values, and loss values for last train epoch and test run. We obtained the best result from ResNet-50 model after 9th train epoch in 7 minutes with the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9215, 0.9216, 0.9216, 0.9215 respectively, and the loss of 9th train epoch and validation as 0.0223 and 1.3687.

alexnet

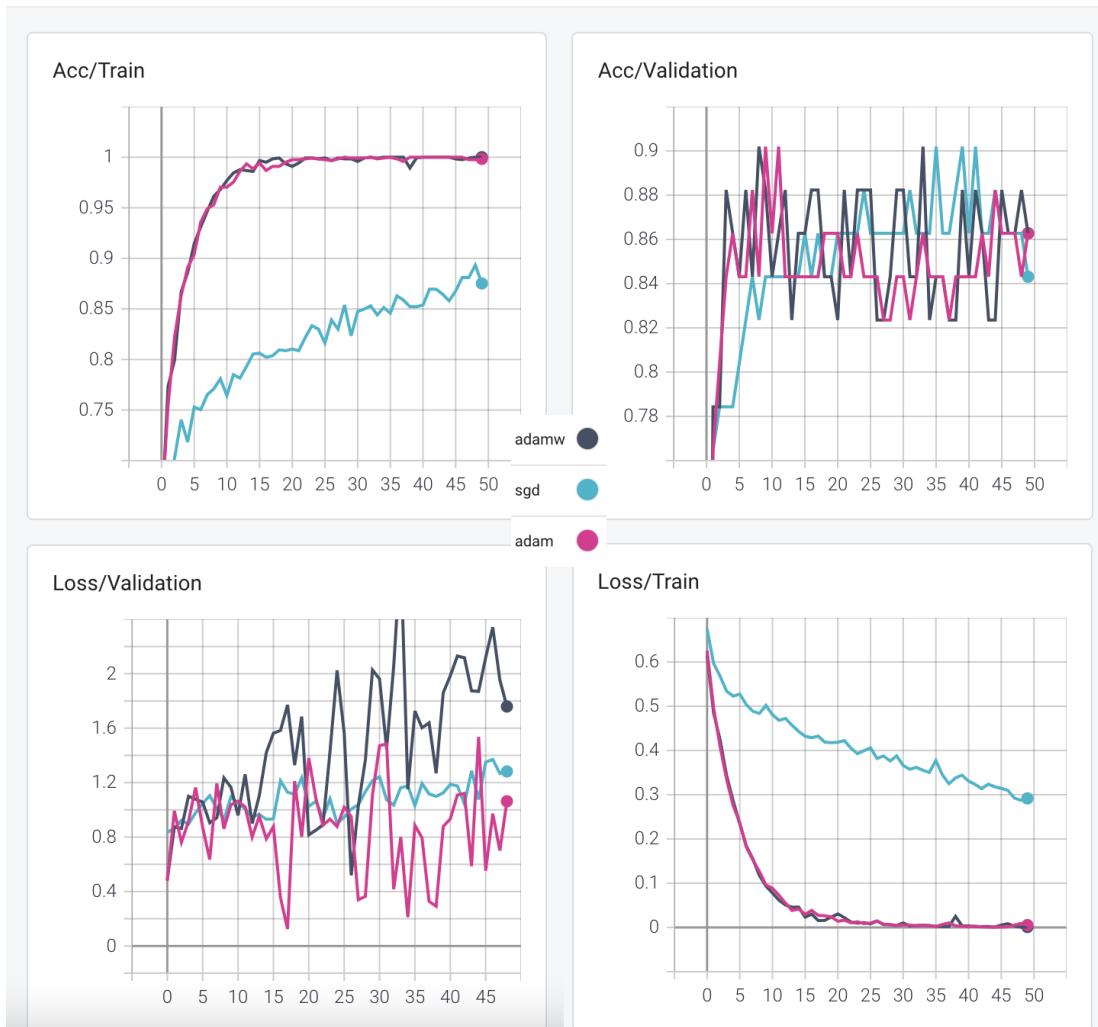


Figure 6.1 : AlexNet model accuracy and loss curves for three different optimizers on train and validation processes.

resnet18

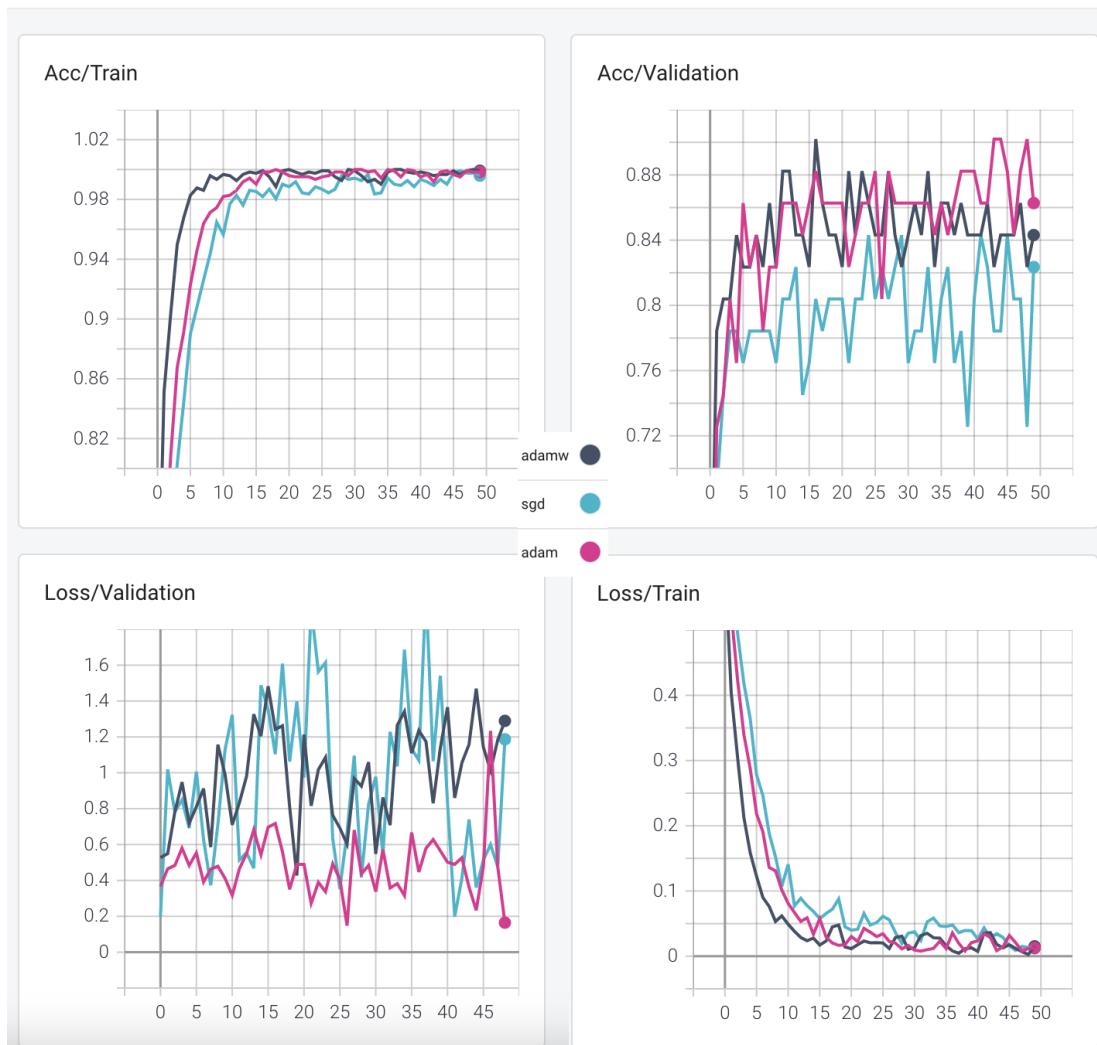


Figure 6.2 : ResNet-18 model accuracy and loss curves for three different optimizers on train and validation processes.

resnet34

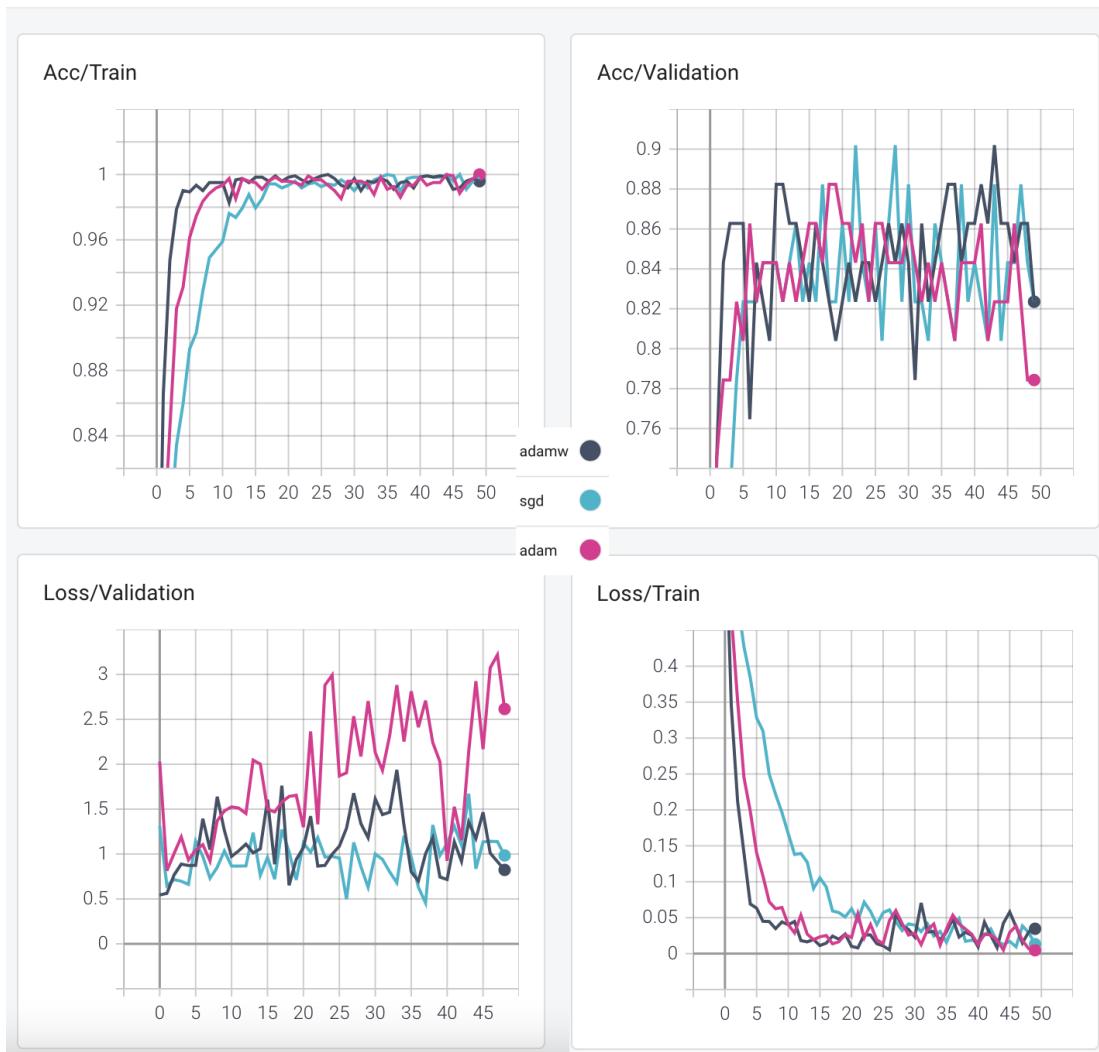


Figure 6.3 : ResNet-34 model accuracy and loss curves for three different optimizers on train and validation processes.

resnet50

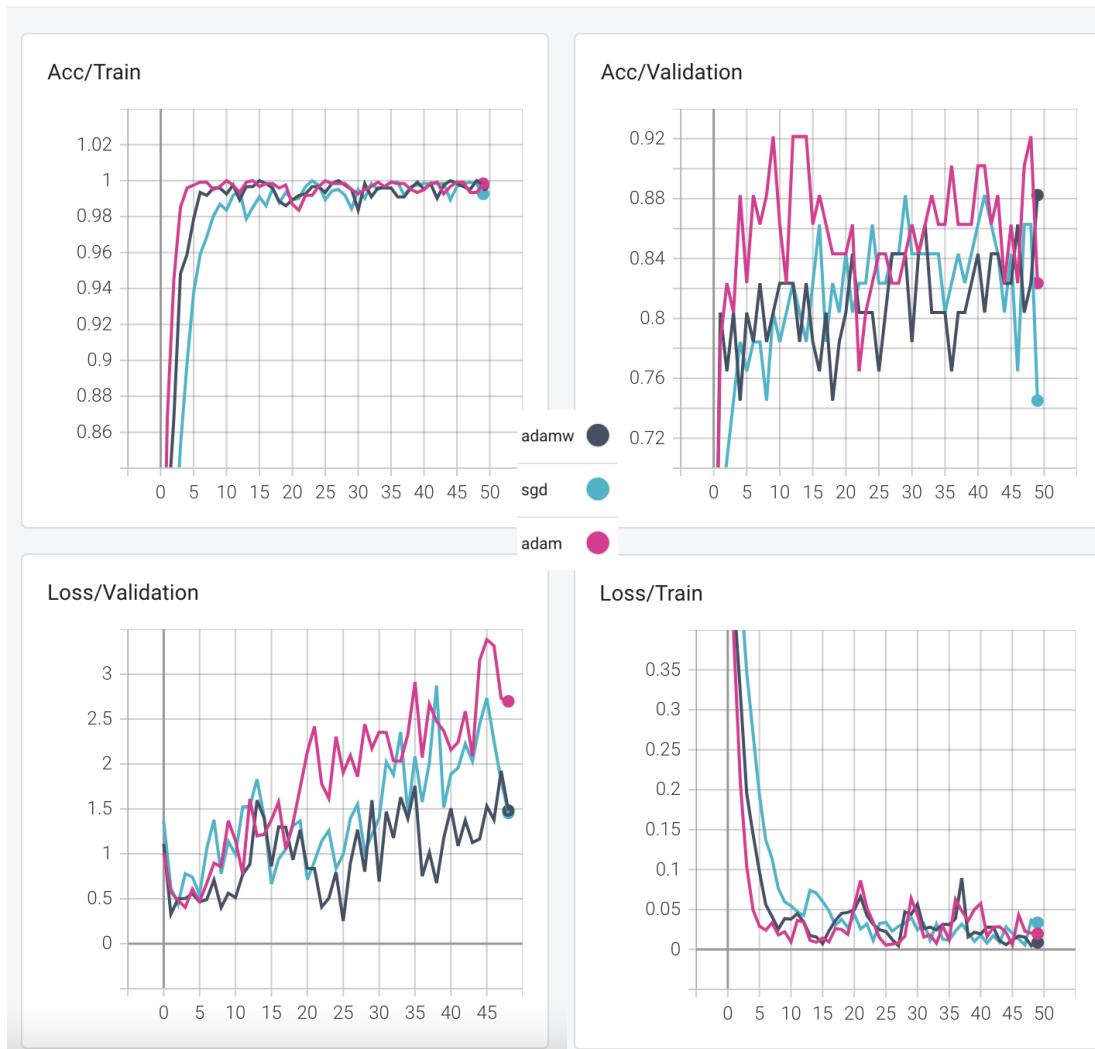


Figure 6.4 : ResNet-50 model accuracy and loss curves for three different optimizers on train and validation processes.

vgg16

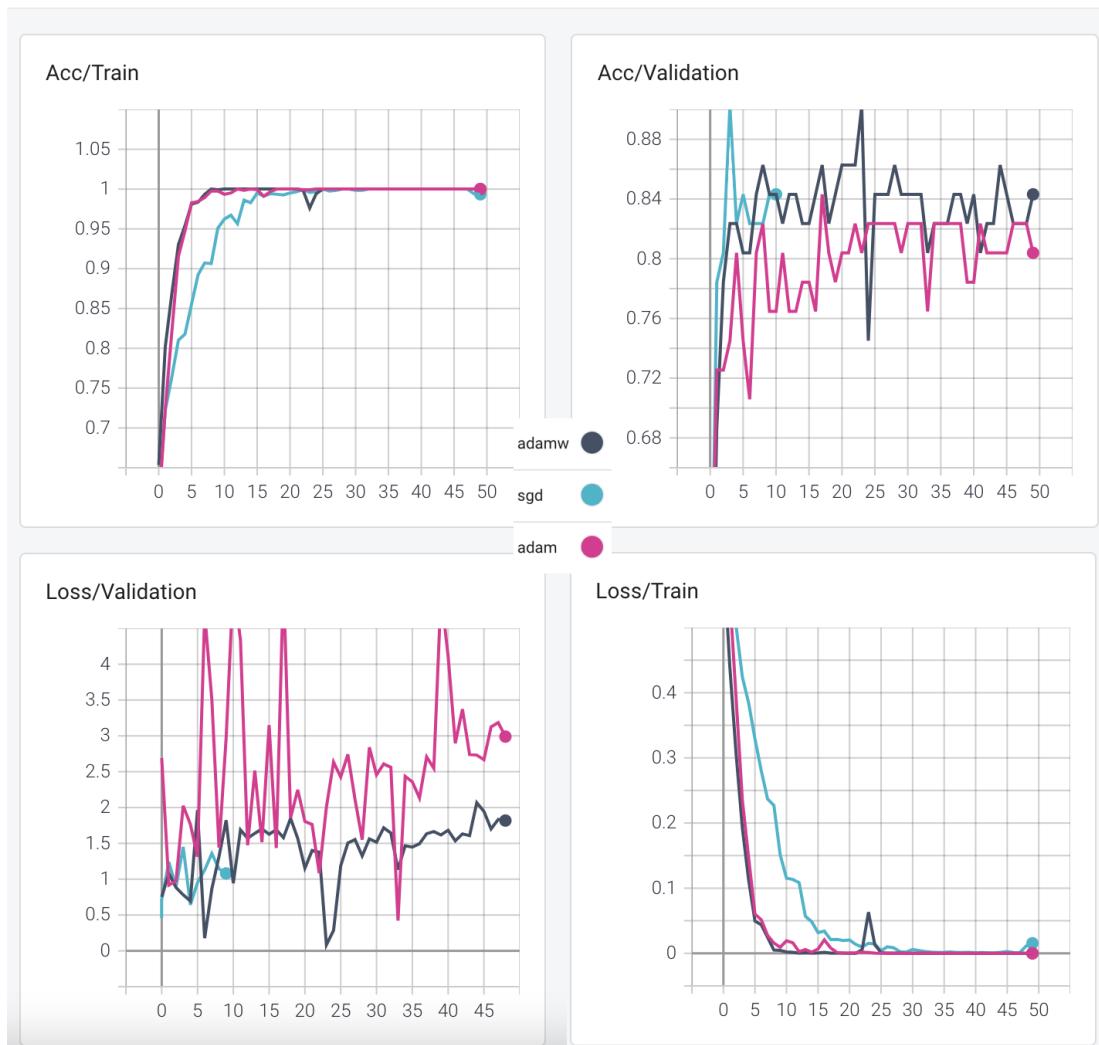


Figure 6.5 : VGG16 model accuracy and loss curves for three different optimizers on train and validation processes.

vgg19

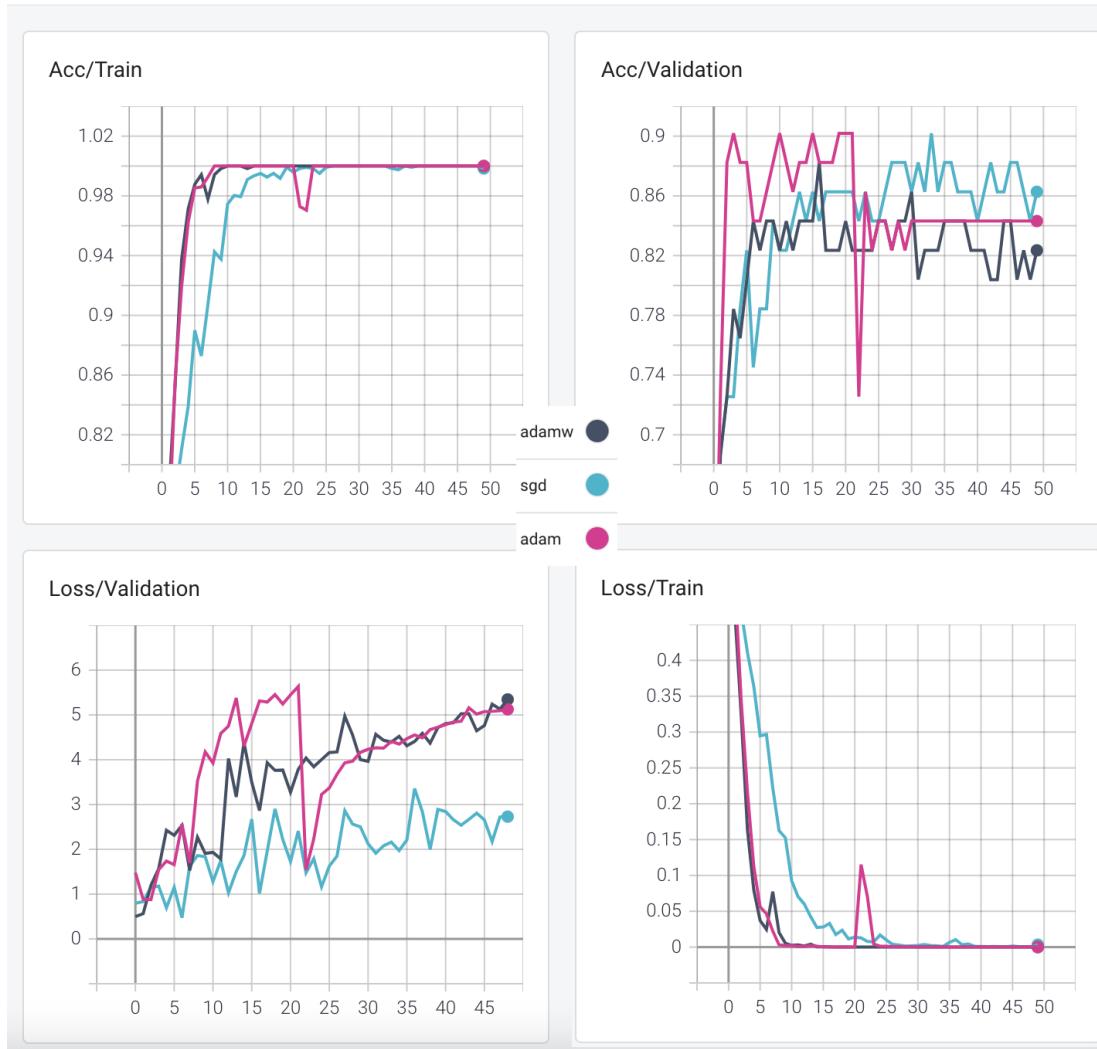


Figure 6.6 : VGG19 model accuracy and loss curves for three different optimizers on train and validation processes.

Table 6.2 : CNN model comparison.

| Models and Optimizers | | Metrics | | | | | | Losses | | Duration (minutes) | |
|-----------------------|------------|--------------|---------------|---------------|---------------|---------------|---------------|---------------|---------------|--------------------|----------|
| Model | Optimizer | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Last Train | Test | Last Train Epoch | |
| AlexNet | SGD | 90.20 | 0.9020 | 0.9026 | 0.9020 | 0.9019 | 0.3503 | 1.0316 | 35 | 22 | |
| | Momentum | | | | | | | | | | |
| | Adam | 90.20 | 0.9020 | 0.9011 | 0.9025 | 0.9019 | 0.9023 | 0.1265 | 1.0373 | 8 | 7 |
| | (no decay) | | | | | | | | | | |
| ResNet-18 | SGD | 90.20 | 0.9020 | 0.9042 | 0.9078 | 0.9017 | 0.9018 | 0.1549 | 1.2347 | 9 | 13 |
| | Momentum | | | | | | | | | | |
| | Adam | 90.20 | 0.9020 | 0.9026 | 0.9020 | 0.9020 | 0.9022 | 0.0288 | 0.3631 | 43 | 33 |
| | (no decay) | | | | | | | | | | |
| ResNet-34 | SGD | 90.20 | 0.9020 | 0.9026 | 0.9026 | 0.9020 | 0.9023 | 0.0460 | 1.1830 | 16 | 12 |
| | Momentum | | | | | | | | | | |
| | Adam | 88.24 | 0.8823 | 0.8838 | 0.8849 | 0.8823 | 0.8826 | 0.0136 | 1.6412 | 18 | 10 |
| | (no decay) | | | | | | | | | | |
| ResNet-50 | SGD | 88.24 | 0.8823 | 0.8823 | 0.8823 | 0.8823 | 0.8821 | 0.0327 | 1.2134 | 29 | 14 |
| | Momentum | | | | | | | | | | |
| | Adam | 92.16 | 0.9216 | 0.9215 | 0.9216 | 0.9219 | 0.9219 | 0.0223 | 1.3687 | 9 | 7 |
| | AdamW | 88.24 | 0.8823 | 0.8823 | 0.8823 | 0.8823 | 0.8827 | 0.0260 | 1.4826 | 50 | 34 |
| VGG16 | SGD | 90.20 | 0.9020 | 0.9042 | 0.9078 | 0.9017 | 0.9021 | 0.0318 | 1.4500 | 15 | 11 |
| | Momentum | | | | | | | | | | |
| | Adam | 84.31 | 0.8431 | 0.8415 | 0.8450 | 0.8428 | 0.8432 | 0.0206 | 5.1689 | 17 | 15 |
| | AdamW | 90.20 | 0.9020 | 0.8996 | 0.9074 | 0.9015 | 0.9023 | 0.0057 | 0.0831 | 23 | 19 |
| VGG19 | SGD | 90.20 | 0.9020 | 0.9026 | 0.9026 | 0.9020 | 0.9020 | 0.0024 | 2.1594 | 29 | 26 |
| | Momentum | | | | | | | | | | |
| | Adam | 90.20 | 0.9020 | 0.9026 | 0.9025 | 0.9019 | 0.9019 | 0.0344 | 1.5451 | 3 | 2 |
| | (no decay) | | | | | | | | | | |
| | AdamW | 88.24 | 0.8823 | 0.8838 | 0.8849 | 0.8823 | 0.8825 | 0.0006 | 2.8652 | 16 | 15 |

One chosen confusion matrix for each CNN architecture according to accuracy and loss values are given in Tables 6.3 - 6.8.

Table 6.3 : The confusion matrix of AlexNet model result with Adam optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 24 | 3 |
| | N | 2 | 22 |

Table 6.4 : The confusion matrix of ResNet-18 model result with AdamW optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 22 | 1 |
| | N | 4 | 24 |

Table 6.5 : The confusion matrix of ResNet-34 model result with AdamW optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 23 | 2 |
| | N | 3 | 23 |

Table 6.6 : The confusion matrix of ResNet-50 model result with Adam optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 23 | 3 |
| | N | 2 | 23 |

Table 6.7 : The confusion matrix of VGG16 model result with AdamW optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 25 | 4 |
| | N | 1 | 21 |

Table 6.8 : The confusion matrix of VGG19 model result with SGD Momentum optimizer.

| | | Actual | |
|-----------|---|--------|----|
| | | P | N |
| Predicted | P | 23 | 2 |
| | N | 3 | 23 |

6.3 Machine Learning Results

After the experiments of CNN models and obtaining the results, the deep feature sets were extracted from exactly the same train and test data. On the other side, the demographic information feature matrices were extracted for the same separation. In this section, we report the results for machine learning experiments on X_{info} , X_{cnn} and X_{all} defined in Section 5.5.

6.3.1 Results for X_{info}

In Table 6.9, the result for experimented machine learning algorithms on X_{info} can be viewed. The reported results can be reproduced with the Python system seed as 4. The best results were obtained with KNN algorithm, and the final hyper-parameters can be seen on Table 6.10. Furthermore, the confusion matrices are on Table 6.11.

Table 6.9 : The result of demographic information experiments with the Python system seed as 4.

| Models and Regularizers | | Accuracies (%) | |
|-------------------------|-------------|---|--|
| Model | Regularizer | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
| SVM | Ridge | 52.94 | 52.94 |
| | Lasso | 52.94 | 52.94 |
| LR | - | 52.94 | 52.94 |
| | Ridge | 52.94 | 52.94 |
| KNN | Lasso | 52.94 | 52.94 |
| | - | 56.86 | 56.86 |
| LDA | - | 52.94 | 52.94 |
| | Lasso | 52.94 | 52.94 |

Table 6.10 : KNN algorithm hyper-parameters for X_{info} .

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|-------------------------|---|--|
| Number of Neighbors | 17 | 17 |
| Type of Majority Voting | uniform | uniform |
| Neighborhood Finder | brute force | brute force |
| Metric Function | euclidean | euclidean |

6.3.2 Results for X_{cnn}

In Table 6.12, the machine learning experiments on different X_{cnn} feature matrices can be seen for AlexNet model with Adam optimizer, ResNet-18 model AdamW optimizer,

Table 6.11 : Confusion matrices for KNN experiments on demographic information.

| | | Actual | | | |
|-----------|---|---|----|--|----|
| | | Result for Grid Search on Train Data | | Result for Grid Search on Test Data | |
| | | P | N | P | N |
| Predicted | P | 5 | 1 | 5 | 1 |
| | N | 21 | 24 | 21 | 24 |

ResNet-34 model AdamW optimizer, ResNet-50 model with Adam optimizer, VGG16 model with AdamW optimizer, and VGG19 model SGD optimizer. The final hyper-parameters for $X_{cnn_ResNet50}$ feature matrix constructed by the help of ResNet-50 model Adam optimizer can be seen on Tables 6.13 - 6.20. Furthermore, the confusion matrices and all other metrics derived from these confusion matrices are on Tables 6.21 - 6.28.

Table 6.12 : The result of experiments on X_{cm} with the Python system seed as 4.

| Models and Regularizers | | Accuracies (%) | | | | | | | | | | | |
|-------------------------|-------------|--|-----------------------|-----------------------|-----------------------|-------------------|---------------------------------------|-------------------|-----------------------|-----------------------|---------------------|-------------------|-------------------------|
| | | Result After Grid Search on Train Data | | | | | Result After Grid Search on Test Data | | | | | | |
| Model | Regularizer | AlexNet (Adam) | ResNet-18 (Adam W) | ResNet-34 (Adam W) | ResNet-50 (Adam W) | VGG16 (Adam W) | VGG19 (SGD Momentum) | AlexNet (Adam) | ResNet-18 (Adam W) | ResNet-34 (Adam W) | ResNet-50 (Adam) | VGG16 (Adam W) | VGG19 (SGD Momentum) |
| SVM | Ridge | 88.24 | 88.24 | 90.20 | 92.16 | 88.24 | 88.24 | 90.20 | 88.24 | 90.20 | 92.16 | 88.24 | 90.20 |
| | Lasso | 90.20 | 90.20 | 84.31 | 86.27 | 88.24 | 86.27 | 90.20 | 90.20 | 90.20 | 92.16 | 88.24 | 88.24 |
| LR | Ridge | 90.20 | 88.24 | 90.20 | 92.16 | 90.20 | 90.20 | 90.20 | 88.24 | 90.20 | 92.16 | 90.20 | 90.20 |
| | Lasso | 90.20 | 88.24 | 90.20 | 92.16 | 88.24 | 88.24 | 90.20 | 88.24 | 90.20 | 92.16 | 88.24 | 88.24 |
| KNN | - | 88.24 | 88.24 | 90.20 | 92.16 | 86.27 | 90.20 | 90.20 | 88.24 | 90.20 | 92.16 | 86.27 | 90.20 |
| | LDA | 82.35 | 82.35 | 82.35 | 84.31 | 82.35 | 84.31 | 84.31 | 84.31 | 82.35 | 84.31 | 84.31 | 84.31 |
| Lasso | | 88.24 | 88.24 | 88.24 | 90.20 | 88.24 | 88.24 | 90.20 | 90.20 | 90.20 | 92.16 | 88.24 | 90.20 |

Table 6.13 : SVM algorithm regularized by Ridge hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Kernel Function | linear | linear |
| Regularization Parameter | 5e-5 | 5e-5 |

Table 6.14 : SVM algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Regularization Parameter | 5e-3 | 5e-2 |

Table 6.15 : LR algorithm hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|-----------------|---|--|
| Solver Function | newton-cg | newton-cg |

Table 6.16 : LR algorithm regularized by Ridge hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Solver Function | liblinear | liblinear |
| Regularization Parameter | 5e-5 | 5e-5 |

Table 6.17 : LR algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Solver Function | liblinear | liblinear |
| Regularization Parameter | 2e-2 | 1.0 |

Table 6.18 : KNN algorithm hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|-------------------------|---|--|
| Number of Neighbors | 3 | 3 |
| Type of Majority Voting | uniform | uniform |
| Neighborhood Finder | ball tree | ball tree |
| Metric Function | euclidean | euclidean |

Table 6.19 : LDA algorithm hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|---|---|--|
| Solver Function | svd | svd |
| Computing the Weighted Within-Class Covariance (for svd solver) | True | True |

Table 6.20 : LDA algorithm regularized by Lasso hyper-parameters for $X_{cnn_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Regularization Parameter | 5e-5 | 5e-3 |

Table 6.21 : The confusion matrices and results obtained by SVM algorithm regularized with Ridge for $X_{cnn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 23 | 1 | | | | | | 23 | 1 | | | |
| | N | 3 | 24 | | | | | | 3 | 24 | | | |

Table 6.22 : The confusion matrices and results obtained by SVM algorithm regularized with Lasso for $X_{cnn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 86.27 | 0.8627 | 0.8665 | 0.8777 | 0.8617 | 0.8646 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 20 | 1 | | | | | | 23 | 1 | | | | |
| | N | 6 | 24 | | | | | | 3 | 24 | | | | |

Table 6.23 : The confusion matrices and results obtained by LR algorithm for $X_{cnn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 23 | 1 | | | | | | 23 | 1 | | | |
| | N | 3 | 24 | | | | | | 3 | 24 | | | |

Table 6.24 : The confusion matrices and results obtained by LR algorithm regularized with Ridge for $X_{cmn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 |
| Predicted | P | 23 | 1 | | | | | P | N | 23 | 1 | | | 0.9223 |
| | N | 3 | 24 | | | | | | | 3 | 24 | | | |

Table 6.25 : The confusion matrices and results obtained by LR algorithm regularized with Lasso for $X_{cmn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 86.27 | 0.8627 | 0.8665 | 0.8777 | 0.8617 | 0.8646 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 |
| Predicted | P | 20 | 1 | | | | | P | N | 23 | 1 | | | 0.9223 |
| | N | 6 | 24 | | | | | | | 3 | 24 | | | |

Table 6.26 : The confusion matrices and results obtained by KNN algorithm for $X_{cmn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 |
| Predicted | P | 23 | 1 | | | | | P | N | 23 | 1 | | | 0.9223 |
| | N | 3 | 24 | | | | | | | 3 | 24 | | | |

Table 6.27 : The confusion matrices and results obtained by LDA algorithm for $X_{cmn_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 84.31 | 0.8431 | 0.8476 | 0.8638 | 0.8413 | 0.8454 | P | N | 84.31 | 0.8431 | 0.8476 | 0.8638 | 0.8413 |
| Predicted | P | 19 | 1 | | | | | P | N | 19 | 1 | | | 0.8454 |
| | N | 7 | 24 | | | | | | | 7 | 24 | | | |

Table 6.28 : The confusion matrices and results obtained by LDA algorithm regularized with Lasso for $X_{cm_ResNet50}$.

| | | | Result for Grid Search on Train Data | | | | | Result for Grid Search on Test Data | | | | | | | | | |
|-----------|---|----|--------------------------------------|--------------|-------------|-------------|-----------|-------------------------------------|-----------|--------|--------------|-----------------|-------------|-----------|----------|-----------|--------|
| | | | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity (%) | Specificity | Precision | F1 Score | AUC Score | |
| | | | P | 90.20 | 0.9020 | 0.9042 | 0.9078 | 0.9017 | 0.9045 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 | 0.9223 |
| Predicted | P | 22 | 1 | | | | | | | 23 | 1 | | | | | | |
| | N | 4 | 24 | | | | | | | 3 | 24 | | | | | | |

6.3.3 Results for X_{all}

In Table 6.29, the machine learning experiments on different X_{all} feature matrices can be seen for AlexNet model with Adam optimizer, ResNet-18 model AdamW optimizer, ResNet-34 model AdamW optimizer, ResNet-50 model with Adam optimizer, VGG16 model with AdamW optimizer, and VGG19 model SGD optimizer. The final hyper-parameters for $X_{all_ResNet50}$ feature matrix constructed by the help of ResNet-50 model Adam optimizer can be seen on Tables 6.30 - 6.37. Furthermore, the confusion matrices and all other metrics derived from these confusion matrices are on Tables 6.38 - 6.45.

Table 6.29 : The result of experiments on X_{ull} with the Python system seed as 4.

| Models and Regularizers | | Accuracies (%) | | | | | | | | | | | |
|-------------------------|-------------|--------------------------------------|----------------------|----------------------|-----------------|------------------|-------------------------------------|-------------------|----------------------|----------------------|---------------------|------------------|-------------------------|
| | | Result for Grid Search on Train Data | | | | | Result for Grid Search on Test Data | | | | | | |
| Model | Regularizer | AlexNet (Adam) | ResNet-18 (AdamW) | ResNet-34 (AdamW) | VGG16 (Adam) | VGG19 (AdamW) | VGG16 (SGD Momentum) | AlexNet (Adam) | ResNet-18 (AdamW) | ResNet-34 (AdamW) | ResNet-50 (Adam) | VGG16 (AdamW) | VGG19 (SGD Momentum) |
| SVM | Ridge | 88.24 | 90.20 | 90.20 | 92.16 | 88.24 | 88.24 | 90.20 | 88.24 | 90.20 | 92.16 | 88.24 | 90.20 |
| | Lasso | 90.20 | 90.20 | 84.31 | 86.27 | 88.24 | 86.27 | 90.20 | 90.20 | 90.20 | 92.16 | 88.24 | 88.24 |
| LR | Ridge | 90.20 | 88.24 | 90.20 | 92.16 | 90.20 | 90.20 | 90.20 | 88.24 | 90.20 | 92.16 | 88.24 | 88.24 |
| | Lasso | 84.31 | 86.27 | 90.20 | 92.16 | 88.24 | 86.27 | 86.27 | 88.24 | 86.27 | 90.20 | 92.16 | 88.24 |
| KNN | - | 90.20 | 88.24 | 90.20 | 92.16 | 86.27 | 90.20 | 90.20 | 88.24 | 90.20 | 92.16 | 86.27 | 90.20 |
| LDA | Ridge | 82.35 | 82.35 | 82.35 | 84.31 | 82.35 | 84.31 | 84.31 | 84.31 | 82.35 | 84.31 | 84.31 | 84.31 |
| | Lasso | 88.24 | 88.24 | 88.24 | 90.20 | 88.24 | 90.20 | 90.20 | 90.20 | 90.20 | 92.16 | 88.24 | 90.20 |

Table 6.30 : SVM algorithm regularized by Ridge hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Kernel Function | linear | linear |
| Regularization Parameter | 5e-5 | 5e-5 |

Table 6.31 : SVM algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Regularization Parameter | 5e-3 | 5e-2 |

Table 6.32 : LR algorithm hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|-----------------|---|--|
| Solver Function | newton-cg | newton-cg |

Table 6.33 : LR algorithm regularized by Ridge hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Solver Function | liblinear | liblinear |
| Regularization Parameter | 5e-5 | 5e-5 |

Table 6.34 : LR algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Solver Function | liblinear | liblinear |
| Regularization Parameter | 2e-2 | 2.0 |

Table 6.35 : KNN algorithm hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|-------------------------|---|--|
| Number of Neighbors | 3 | 3 |
| Type of Majority Voting | uniform | uniform |
| Neighborhood Finder | ball tree | ball tree |
| Metric Function | euclidean | euclidean |

Table 6.36 : LDA algorithm hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|---|---|--|
| Solver Function | svd | svd |
| Computing the Weighted Within-Class Covariance (for svd solver) | True | True |

Table 6.37 : LDA algorithm regularized by Lasso hyper-parameters for $X_{all_ResNet50}$.

| Hyper-Parameter | Result for Grid Search on Train Data | Result for Grid Search on Test Data |
|--------------------------|---|--|
| Regularization Parameter | 5e-5 | 5e-3 |

Table 6.38 : The confusion matrices and results obtained by SVM algorithm regularized with Ridge for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | Result for Grid Search on Test Data | | | | | | | |
|--------------------------------------|--------------|-----------------|-----------|-------------------------------------|-----------|--------|--------------|-----------------|-----------|----------|-----------|
| Actual | Accuracy (%) | Sensitivity (%) | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity (%) | Precision | F1 Score | AUC Score |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| Predicted | P | 23 | 1 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| | N | 3 | 24 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9223 |

Table 6.39 : The confusion matrices and results obtained by SVM algorithm regularized with Lasso for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | Result for Grid Search on Test Data | | | | | | | |
|--------------------------------------|--------------|-------------|-----------|-------------------------------------|-----------|--------|--------------|-------------|-----------|----------|-----------|
| Actual | Accuracy (%) | Sensitivity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Precision | F1 Score | AUC Score |
| P | N | 86.27 | 0.8627 | 0.8665 | 0.8777 | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| Predicted | P | 20 | 1 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| | N | 6 | 24 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9223 |

Table 6.40 : The confusion matrices and results obtained by LR algorithm for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | Result for Grid Search on Test Data | | | | | | | |
|--------------------------------------|--------------|-------------|-----------|-------------------------------------|-----------|--------|--------------|-------------|-----------|----------|-----------|
| Actual | Accuracy (%) | Sensitivity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Precision | F1 Score | AUC Score |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| Predicted | P | 23 | 1 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9215 |
| | N | 3 | 24 | | | P | N | 92.16 | 0.9216 | 0.9243 | 0.9223 |

Table 6.41 : The confusion matrices and results obtained by LR algorithm regularized with Ridge for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 23 | 1 | | | | | P | N | 23 | 1 | | | |
| | N | 3 | 24 | | | | | | | 3 | 24 | | | |

Table 6.42 : The confusion matrices and results obtained by LR algorithm regularized with Lasso for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 88.23 | 0.8824 | 0.8853 | 0.8923 | 0.8818 | 0.8838 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 21 | 1 | | | | | P | N | 23 | 1 | | | |
| | N | 5 | 24 | | | | | | | 3 | 24 | | | |

Table 6.43 : The confusion matrices and results obtained by KNN algorithm for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9215 | 0.9223 | P | N | 92.16 | 0.9216 | 0.9230 | 0.9243 | 0.9223 |
| Predicted | P | 23 | 1 | | | | | P | N | 23 | 1 | | | |
| | N | 3 | 24 | | | | | | | 3 | 24 | | | |

90

Table 6.44 : The confusion matrices and results obtained by LDA algorithm for $X_{all_ResNet50}$.

| Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | | |
|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|--------|
| Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | |
| P | N | 84.31 | 0.8431 | 0.8476 | 0.8638 | 0.8413 | 0.8454 | P | N | 84.31 | 0.8431 | 0.8476 | 0.8638 | 0.8454 |
| Predicted | P | 19 | 1 | | | | | P | N | 19 | 1 | | | |
| | N | 7 | 24 | | | | | | | 7 | 24 | | | |

Table 6.45 : The confusion matrices and results obtained by LDA algorithm regularized with Lasso for $X_{all_ResNet50}$.

| | | Result for Grid Search on Train Data | | | | | | Result for Grid Search on Test Data | | | | | | | |
|-----------|---|--------------------------------------|--------------|-------------|-------------|-----------|----------|-------------------------------------|--------|--------------|-------------|-------------|-----------|----------|-----------|
| | | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score | Actual | Accuracy (%) | Sensitivity | Specificity | Precision | F1 Score | AUC Score |
| | | P | 90.20 | 0.9020 | 0.9042 | 0.9078 | 0.9017 | 0.9045 | P | N | 92.16 | 0.9230 | 0.9243 | 0.9215 | 0.9223 |
| Predicted | P | 22 | 1 | | | | | | 23 | 1 | | | | | |
| | N | 4 | 24 | | | | | | 3 | 24 | | | | | |

We can brief all these results about our experiments on our dataset such as:

- generalized optimal hyper-parameters may not be best combination for our initially determined test data,
- demographic information support only boosts the performance on logistic regression algorithm regularized by Lasso, and
- penalty approach only increases the success on linear discriminant analysis algorithm.

7. CONCLUSIONS AND RECOMMENDATIONS

The main scope of this thesis was to show a method to use image data without segmenting or manually extracting features from them on machine learning algorithms, and how to use it on a continuing global public health problem. One recommendation here can be given as to use data eliminated by professionals. In the studies like ours, getting support from specialist radiologists provides to have more qualified chest X-Ray images, and through this better classifiers may be yielded.

The reason why applying cross validation on train data and yielding optimized hyper-parameters from these experiments is that to find the most generalized hyper-parameter settings. If the grid search was applied to the experiment including training on train data and testing on test data, the yielding parameters indeed would be only optimized for our test data. This would be enough for this work, if we only cared about the mathematical results and out success. However, we aimed to find more generalized optimal parameters. Thus, we used 10-Fold cross validation which means we experimented on 10 different validation sets. Then we tested these hyper-parameters on test data and reported the final results. In addition to that, we also experimented grid search only on the initially determined test set and reported the results. In this way, the comparison on metrics and optimized parameters can be performed between generalized and specific to our test data.

After the experiments, we encountered the benefits of obtaining parameters by using grid search on train set. We have seen that the parameters in this way are not always the best parameters for our test set. Thanks to this way, we provided the method for how to obtain results not specific to a group of data but more suitable for random grouped data.

Although this study mainly focused on chest X-Rays and image data, the one of aim was to show non-image data can be combined with image data. Our non-image data is the demographic information of patients for corresponding chest X-Rays. Since the data size was not enough for a good classification and the non-image data types

was restricted to two features, desired result may not be seen. To see the results of the singular use of demographic information, we also experimented only on these two features with machine learning algorithms. We expect a visible effect of non-image data, if more information from patient such as doctors report, tobacco product use, associated genetic diseases, etc. is used. Moreover, the singular use results of non-image information would be better with more effective and numerically superior features.

Another main comparison was about the effect of regularization on machine learning algorithms. We wanted to see the performance of Lasso and Ridge, and obtained an improvement on linear discriminant analysis with Lasso.

The final best results are as follows:

- **CNN:** the best result was obtained from ResNet-50 model after 9th train epoch in 7 minutes with the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9215, 0.9216, 0.9216, 0.9215 respectively, and the loss of 9th train epoch and validation as 0.0223 and 1.3687, and
- **ML:** since the best model on CNN experiments was ResNet-50 model, the feature matrices of deep features providing the best results were also the ones transferred from ResNet-50 model. The results for three feature matrices X_{info} , $X_{cnn_ResNet50}$ and $X_{all_ResNet50}$ are as follows:
 - the best result obtained from demographic information features was on KNN algorithm as the accuracy of 56.86%, the sensitivity of 0.56863, the specificity of 0.58368 the precision of 0.6863, the F1 score of 0.49545, and the AUC score of 0.5745. The hyper-parameters were the number of neighbors as 17, the type of majority voting as uniform, the neighbor finder as brute force, and the metric function as euclidean,
 - the best result obtained from deep transferred features was not singular. According the generalized hyper-parameters, SVM with Ridge regularizer, LR, LR with Ridge, and KNN algorithms had the same results with the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 respectively, and

- the best result obtained from the combined features of demographic information and deep features was not singular. According the generalized hyper-parameters, SVM with Ridge regularizer, LR, LR with Ridge, and KNN algorithms had the same results with the accuracy, sensitivity, specificity, precision, F1 score and AUC score of 92.16%, 0.9216, 0.9230, 0.9243, 0.9215, 0.9223 respectively.

As it can be seen on confusion matrices of convolutional neural network results and machine learning results, the performance of classification for non-COVID labeled data was improved by machine learning algorithms. Moreover, the sensitivity, specificity, precision and F1 score was 0.9216, 0.9215, 0.9216 and 0.9216 respectively for ResNet-50 model, the result for machine learning algorithms was 0.9216, 0.9230, 0.9242, and 0.9216 for the sensitivity, specificity, precision and F1 score respectively. Briefly, even though the accuracy, sensitivity and F1 score did not improve, the specificity and precision improved. This can be interpreted as the answers of two questions could be given better: 1) how many of data estimated as infected were actually infected, and 2) how many of non-COVID-19 data over all patients correctly predicted.

7.1 Future Work

The contribution of this thesis is to understand how to use deep features extracted from convolutional neural network models on machine learning algorithms, and how to combine the image data features with non-image data to use in machine learning algorithms. To research on this topic, a global health problem COVID-19 disease and chest X-Rays images data including demographic information were used. However, any other image classification problem can be studied for future work. If a dataset consisting of images also includes non-image information, this thesis can be used as a guide for corresponding classification problem.

For instance, this study can be applied on multi-class classification problem on various respiratory diseases by chest computed tomography images of patients together with the demographic, doctors report, tobacco product use, associated genetic diseases, and respiratory test information. We also recommend to get help from one or more

specialist radiologists and chest disease specialists on eliminating the data, for the given study example.

This study can be improved by using more qualified dataset with more data in number, and extended by using chest computer tomography and more information of the patients in terms of variety. On the other hand, optimizers on convolutional neural networks such as Adagrad [43] and Padam [44], and elastic net regularization method [45] on machine learning algorithms can be used to extend this study. Moreover, ensemble learning [46] on convolutional neural network process and machine learning process separately can be used to extend the scope and results of this study.

REFERENCES

- [1] **Kevles, B.** (1997). Naked to the bone: Medical imaging in the twentieth century.
- [2] **Loshchilov, I. and Hutter, F.** (2017). Fixing weight decay regularization in Adam, *CoRR*.
- [3] **Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.** (2014). Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, 15(56), 1929–1958, <http://jmlr.org/papers/v15/srivastava14a.html>.
- [4] **Nour, M., Cömert, Z. and Polat, K.** (2020). A novel medical diagnosis model for COVID-19 infection detection based on deep features and Bayesian optimization, *Applied Soft Computing*, 97, 106580, <https://www.sciencedirect.com/science/article/pii/S1568494620305184>.
- [5] **Krizhevsky, A., Sutskever, I. and Hinton, G.E.** (2012). ImageNet classification with deep convolutional neural networks, *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Curran Associates Inc., Red Hook, NY, USA.
- [6] **He, K., Zhang, X., Ren, S. and Sun, J.** (2015), Deep residual learning for image recognition, 1512.03385.
- [7] **Nielsen, F.**, (2016). Parallel linear algebra, p.140.
- [8] **Kahn, J.S. and McIntosh, K.** (2005). History and recent advances in coronavirus discovery, *The Pediatric Infectious Disease Journal*, 24(11), https://journals.lww.com/pidj/Fulltext/2005/11001/History_and_Recent_Advances_in_Coronavirus.12.aspx.
- [9] **Gorbalenya, A.E., Baker, S.C., Baric, R.S., de Groot, R.J., Drosten, C., Gulyaeva, A.A., Haagmans, B.L., Lauber, C., Leontovich, A.M., Neuman, B.W., Penzar, D., Perlman, S., Poon, L.L.M., Samborskiy, D.V., Sidorov, I.A., Sola, I., Ziebuhr, J. and of Viruses, C.S.G.o.t.I.C.o.T.** (2020). The species severe acute respiratory syndrome-related coronavirus: Classifying 2019-nCoV and naming it SARS-CoV-2, *Nature Microbiology*, 5(4), 536–544, <https://doi.org/10.1038/s41564-020-0695-z>.
- [10] (2020), WHO Director-General's opening remarks at the media briefing on COVID-19 - 11 March 2020, <https://www.who.int/director-general/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19---11-march-2020>.

- [11] (2021), Coronavirus (COVID-19), <https://news.google.com/covid19/map?hl=tr&gl=TR&cid=TR%3Atr&state=1>.
- [12] (2020), Coronavirus disease (COVID-19), <https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19>.
- [13] (2020), PCR test for COVID-19: What it is, How its done, What the results mean, <https://my.clevelandclinic.org/health/diagnostics/21462-covid-19-and-pcr-testing>.
- [14] **Villines, Z. and Martinez, K.** (2020). *Medical News Today*, <https://www.medicalnewstoday.com/articles/pneumonia-and-covid-19>.
- [15] **Ardakani, A.A., Kanafi, A.R., Acharya, U.R., Khadem, N. and Mohammadi, A.** (2020). Application of deep learning technique to manage COVID-19 in routine clinical practice using CT images: Results of 10 convolutional neural networks, *Computers in Biology and Medicine*, 121, 103795, <https://www.sciencedirect.com/science/article/pii/S0010482520301645>.
- [16] **Pathak, Y., Shukla, P.K., Tiwari, A., Stalin, S. and Singh, S.** (2020). Deep transfer learning based classification model for COVID-19 disease, *Ingenierie et recherche biomédicale : IRBM = Biomedical engineering and research*, 10.1016/j.irbm.2020.05.003, <https://pubmed.ncbi.nlm.nih.gov/32837678>, 32837678[pmid].
- [17] **Deng, J., Dong, W., Socher, R., Li, L., Kai Li and Li Fei-Fei** (2009). ImageNet: A large-scale hierarchical image database, *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp.248–255.
- [18] **Ozturk, T., Talo, M., Yildirim, E.A., Baloglu, U.B., Yildirim, O. and Rajendra Acharya, U.** (2020). Automated detection of COVID-19 cases using deep neural networks with X-ray images, *Computers in Biology and Medicine*, 121, 103792, <https://www.sciencedirect.com/science/article/pii/S0010482520301621>.
- [19] **Redmon, J. and Farhadi, A.**, (2016), YOLO9000: Better, Faster, Stronger, 1612 .08242.
- [20] **Oh, Y., Park, S. and Ye, J.C.**, (2020), Deep learning COVID-19 features on CXR using limited training data sets, 2004.05758.
- [21] **Elshennawy, N.M. and Ibrahim, D.M.** (2020). Deep-pneumonia framework using deep learning models based on chest X-ray images, *Diagnostics*, 10(9), <https://www.mdpi.com/2075-4418/10/9/649>.
- [22] **Ruaa A. Al-Falluji, Zainab Dalaf Katheeth, B.A.** (2021). Automatic detection of COVID-19 using chest X-ray images and modified ResNet18-based convolution neural networks, *Computers, Materials & Continua*, 66(2), 1301–1313, <http://www.techscience.com/cmc/v66n2/40667>.
- [23] **Sutskever, I., Martens, J., Dahl, G. and Hinton, G.E.** (2013). On the importance

- of initialization and momentum in deep learning, *ICML*, 28, 1139—1147.
- [24] **Graetz, F.M. and Matters, W.A.**, (2018), Why AdamW matters, <https://towardsdatascience.com/why-adamw-matters-736223f31b5d>.
 - [25] **Simonyan, K. and Zisserman, A.**, (2015), Very deep convolutional networks for large-scale image recognition, 1409.1556.
 - [26] **Chang, M.**, (2020). Artificial Intelligence for Drug Development, Precision Medicine, and Healthcare, p. 38.
 - [27] **Cortes, C. and Vapnik, V.** (1995). Support-vector networks, *Mach. Learn.*, 20(3), 273–297, <https://doi.org/10.1023/A:1022627411411>.
 - [28] **Ben-Hur, A., Horn, D., Siegelmann, H. and Vapnik, V.** (2001). Support vector clustering, *Journal of Machine Learning Research*, 2, 125–137.
 - [29] **Christmann, A. and Steinwart, I.** (2008). *Support Vector Machines*.
 - [30] **Koshiba, Y. and Abe, S.** (2003). Comparison of L1 and L2 Support vector machines, volume 3, pp.2054 – 2059 vol.3.
 - [31] **Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E.** (2011). Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, 12, 2825–2830.
 - [32] **Yu, H.F., Huang, F.L. and Lin, C.J.** (2011). Dual coordinate descent methods for logistic regression and maximum entropy models, *Machine Learning*, 85(1), 41–75, <https://doi.org/10.1007/s10994-010-5221-8>.
 - [33] **Zhu, C., Byrd, R.H., Lu, P. and Nocedal, J.** (1997). Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization, *ACM Trans. Math. Softw.*, 23(4), 550–560, <https://doi.org/10.1145/279232.279236>.
 - [34] **Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R. and Lin, C.J.** (2008). LIBLINEAR: A library for large linear classification, *J. Mach. Learn. Res.*, 9, 1871–1874.
 - [35] **Schmidt, M., Roux, N.L. and Bach, F.**, (2016), Minimizing finite sums with the stochastic average gradient, 1309.2388.
 - [36] **Defazio, A., Bach, F. and Lacoste-Julien, S.**, (2014), SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives, 1407.0202.
 - [37] **Fix, E. and Hodges, J.L.** (1989). Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties, *International Statistical Review / Revue Internationale de Statistique*, 57(3), 238–247, <http://www.jstor.org/stable/1403797>.
 - [38] **Bentley, J.L.** (1975). Multidimensional binary search trees used for associative searching, *Commun. ACM*, 18(9), 509–517, <https://doi.org/10.1145/361002.361007>.

- [39] **Omohundro, S.M.** (1989). Five Balltree Construction Algorithms, **Technical Report**.
- [40] **Guo, Y., Hastie, T. and Tibshirani, R.** (2006). Regularized linear discriminant analysis and its application in microarrays, *Biostatistics*, 8(1), 86–100, <https://doi.org/10.1093/biostatistics/kxj035>, <https://academic.oup.com/biostatistics/article-pdf/8/1/86/698312/kxj035.pdf>.
- [41] **scikit-learn developers**, Linear and quadratic discriminant analysis, https://scikit-learn.org/stable/modules/lda_qda.html#estimation-algorithms.
- [42] **Pan, Y., Mai, Q. and Zhang, X.** (2021). TULIP: A toolbox for linear discriminant analysis with penalties, *The R Journal*, 12(2), 61–81, <https://doi.org/10.32614/RJ-2021-025>.
- [43] **Duchi, J., Hazan, E. and Singer, Y.** (2011). Adaptive subgradient methods for online learning and stochastic optimization, *J. Mach. Learn. Res.*, 12(null), 2121–2159.
- [44] **Chen, J. and Gu, Q.** (2018). Closing the generalization gap of adaptive gradient methods in training Deep Neural Networks, *CoRR*, [abs/1806.06763](https://arxiv.org/abs/1806.06763), [tp://arxiv.org/abs/1806.06763](https://arxiv.org/abs/1806.06763), 1806.06763.
- [45] **Zou, H. and Hastie, T.** (2005). Regularization and variable selection via the Elastic net, *Journal of the Royal Statistical Society, Series B*, 67, 301–320.
- [46] **Darıcı, M.B.**, (2020), Göğüs kafesi röntgen görüntülerinde derin öğrenme metoduyla zatürre hastalığının tanısı, <https://tez.yok.gov.tr/UlusalTezMerkezi/tezDetay.jsp?id=YSoGULG5Nf-WfjvJZXgxQQ&no=VhZc08s4-L5HmRYXK4bwIA>.

APPENDICES

APPENDIX A.1 : ReadMe File

APPENDIX A.2 : Python Notebook File

APPENDIX A.1 ReadMe File

The ReadMe.md file is available at <https://github.com/ozanguldali/modelsWithLASSO/blob/master/ReadMe.md>.

ISTANBUL TECHNICAL UNIVERSITY - Institute of Science and Technology

Ozan GÜLDALI

Department of Mathematical Engineering - Mathematical Engineering Program

M.Sc. THESIS

**DEEP FEATURE TRANSFER FROM DEEP LEARNING MODELS INTO
MACHINE LEARNING ALGORITHMS TO CLASSIFY COVID-19 FROM
CHEST X-RAY IMAGES**

- dataset may be a must for some run-configurations
 - Link to dataset: <https://github.com/ozanguldali/modelsWithLASSO/blob/master/dataset>
- dataset_constructor.py was used to create the dataset from <https://github.com/ieee8023/covid-chestxray-dataset> source
- image_operations.py was used to investigate the augmentations of image data
- visualize_layers was used to visualize the layers of cnn models
- To run only ML, only CNN or both as transfer learning, app.py file can be run with corresponding function parameters.

How to Run

- **transfer_learning:** True if wanted to transfer deep features from CNN model
- **save_numpy:** True if wanted to save computed features. Default is False.
- **load_numpy:** True if wanted to use previously computed features. Default is False.
- **numpy_prefix:** Prefix for numpy feature files. Default is empty string.
- **method:** “ML” or “CNN”. Required if transfer_learning is False.
- **ml_model_name:** Model name for ML. “svm”, “lr”, “knn”, “lda”, or “all”. Default is empty string. Required if method is not CNN.

- **ml_features:** Type of features. “info”, “cnn”, or “all”. Default is empty “all”.
- **validate_cv:** True if wanted to apply cross-validation on train set. Default is False.
- **save_ml:** True if wanted to save ML weights. Default is False.
- **save_cnn:** True if wanted to save CNN weights. Default is False.
- **cv:** Type of cross-validation. Any positive integer or “LOO”. Default is 10.
- **lasso:** Type of regularization. True, False, “l2”, or None. None is used for all choices. Default is False.
- **dataset_folder:** Folder name of dataset. Default is “dataset”.
- **pretrain_file:** CNN pretrained pth file name without “pth” extension. Default is None.
- **batch_size:** Size of each batch. Default is 16.
- **img_size:** Size of image dimension. Default is 227.
- **num_workers:** Number of parallel workers. Default is 2.
- **augmented:** True if wanted to augment data. Default is False.
- **cnn_model_name:** Name of CNN model. “alexnet”, “resnet18”, “resnet34”, “resnet50”, “vgg16”, or “vgg19”. Default is empty string. Required if method is not ML.
- **optimizer_name:** Name of optimizer on CNN process. “Adam”, “AdamW” or “SGD”. Default is “Adam”.
- **validation_freq:** Validation frequency ratio on CNN process. Any positive rational number. Default is 0.02.
- **lr:** Learning rate for CNN optimizers. Any positive rational number. Default is 0.00001.
- **momentum:** Momentum ratio for SGD optimizer. Any positive rational number. Default is 0.9.
- **weight_decay:** Weight decay ratio for Adam and AdamW. Any positive rational number. Default is 0.0001.

- **update_lr:** True if wanted to periodically decrease the learning rate on CNN process. Default is False.
- **is_pre_trained:** True if the CNN model wanted to use is pretrained. Default is True.
- **fine_tune:** True if wanted to freeze first convolution block on CNN models. Default is False.
- **num_epochs:** Number of epochs for CNN process. Any positive integer. Default is 50.
- **normalize:** True if wanted to normalize the data. Default is True.
- **lambdas:** Lambda values for regularization on ML process. Single positive real number, or a list of positive real numbers. Default is None. List of [0.00005, 0.0001, 0.0002, 0.0005, 0.005, 0.01, 0.02, 0.05, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0, 15.0] is used, if it is None.
- **seed:** System seed value for ML process. Any positive integer. Default is 4.

Example of Transfer Learning:

1. (Dataset folder is required: <https://github.com/ozanguldali/modelsWithLASSO/blob/master/dataset>)
 - Unless exists, 92.16_resnet50_Adam_out.pth file must be downloaded and inserted into “cnn/saved_models” directory.
 - Link to file:
 - https://github.com/ozanguldali/modelsWithLASSO/blob/master/cnn/saved_models/92.16_resnet50_Adam_out.pth

```
app.main(transfer_learning=True, ml_model_name="all",
         ml_features="all", cnn_model_name="resnet50",
         is_pre_trained=True, cv=10, dataset_folder="dataset",
         pretrain_file="92.16_resnet50_Adam_out", seed=4)
```
2. (Dataset folder is not needed)
 - Unless exists, 92.16_resnet50_Adam_final_X_cnn_train.npy, 92.16_resnet50_Adam_final_X_cnn_test.npy, X_info_train.npy, X_info_test.npy, y_train.npy, and y_test.npy files must be downloaded and inserted into project root directory.
 - Link to files:
 - https://github.com/ozanguldali/modelsWithLASSO/blob/master/92.16_resnet50_Adam_final_X_cnn_t

```

rain.npy

- https://github.com/ozanguldali/modelsWithLASSO
/blob/master/92.16_resnet50_Adam_final_X_cnn_t
est.npy

- https://github.com/ozanguldali/modelsWithLASSO
/blob/master/X_info_train.npy

- https://github.com/ozanguldali/modelsWithLASSO
/blob/master/X_info_test.npy

- https://github.com/ozanguldali/modelsWithLASSO
/blob/master/y_train.npy

- https://github.com/ozanguldali/modelsWithLASSO
/blob/master/y_test.npy
app.main(transfer_learning=True, ml_model_name="all",
ml_features="all", load_numpy=True, validate_cv=True,
cv=10, numpy_prefix="92.16_resnet50_Adam_final",
seed=4)

```

Package Versions

- Python Language: 3.7.6
- Clang: 4.0.1
- pip: 20.1.1
- PyTorch: 1.5.0
- TorchSummary: 1.5.1
- TorchVision 0.6.0
- Scikit-Learn: 0.23.2
- R Language: 4.0.3
- TULIP: 1.0.1
- TensorFlow: 2.3.1
- TensorFlow-Addons: 0.11.2
- TensorFlow-Estimator: 2.3.0
- TensorFlow-Hub: 0.10.0
- TensorFlow-Probability: 0.10.0
- log4p: 2019.7.13.3

- log4python: 0.2.31

APPENDIX A.2 Python Notebook File

The Python Notebook app_run.ipynb file is available at https://github.com/ozanguldali/modelsWithLASSO/blob/master/app_run.ipynb.

```
[ ]: # To run on Google Colaboratory
%cd
%cd ..
%cd content
%ls
# User must see sample_data/ folder
```

```
[ ]: import os.path
```

```
[ ]: project_exists = False
if os.path.exists("modelsWithLASSO"):
    print("project directory is already exist, pulling_
→last changes")
    %cd modelsWithLASSO
    ! git fetch --all
    ! git reset --hard origin/method-in-paper
    ! git pull origin method-in-paper
else:
    print("project directory is NOT exist, checkouting_
→the project")
    ! git clone https://github.com/ozanguldali/
→modelsWithLASSO.git
    %cd modelsWithLASSO
    ! git pull origin method-in-paper
```

```
[ ]: %ls
# User must see project inner folder and files
```

```
[ ]: import os.path
```

```
[ ]: ! pip install log4p
```

```
[ ]: import run_CNN
from importlib import reload
run_CNN = reload(run_CNN)
```

```
[ ]: run_CNN.main(save=True, model_name="resnet50",_
→optimizer_name="Adam", is_pre_trained=True,_
→batch_size=16, lr=0.00001, num_epochs=50,_
→validation_freq=1/50, augmented=True, num_workers=2)
```

```
[ ]:
```

```
# To run CNN process
run_CNN.main(test_without_train=True,
             model_name="resnet50", is_pre_trained=True,
             pretrain_file="92.16_resnet50_Adam_out")
```

```
[ ]: import app
from importlib import reload
app = reload(app)
```

```
[ ]: # To run deep feature transfer from saved CNN model_
      weights to ML algorithms
app.main(transfer_learning=True, ml_model_name="all",
         ml_features="all", cnn_model_name="resnet50",
         is_pre_trained=True, cv=10, dataset_folder="dataset",
         pretrain_file="92.16_resnet50_Adam_out", seed=4)
```


CURRICULUM VITAE

Name Surname: Ozan GÜLDALİ

Place and Date of Birth: İstanbul, 16 August 1994

E-Mail: ozan.guldali@hotmail.com

EDUCATION:

- **B.Sc.:** 2018, Istanbul Technical University, Faculty of Science and Letters, Mathematics Engineering
- **M.Sc.:** Present, Istanbul Technical University, Graduate School, Mathematics Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2018 Completed Undergraduate Education at Istanbul Technical University.
- 2017-2019 Software Test and Automation Engineer at Finartz Information Technologies Inc.
- 2019-Current Software Test and Automation Engineer at AVCR Information Technologies Inc.