ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

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

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

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

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

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 Dalı

Matematik Mühendisliği Programı

Tez Danışmanı: Assist. Prof. Dr. Gül İNAN

AĞUSTOS 2021

Ozan GÜLDALİ, 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 BİCİ 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

Page

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
CT : Computed Tomography
CV : Cross-Validation
DL : Deep Learning
DTL : Deep Transfer Learning
FN : False Negative
FP : False Positive
RBFP : Gaussian Radial Basis Function
KNN : K-Nearest Neighbor
LOO : Leave-One-Out
LDA : Linear Discriminant Analysis
LR : Logistic Regression
LSTM : Long Short-Term Memory
ML : Machine Learning
N : Negative
NN : Neural Network
PCR : Polymerase Chain Reaction
P : Positive
PPV : Positive Predictive Value
PA : Posterior-Anterior
ReLU : Rectified Linear Units
RGB : Red-Green-Blue
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
VGG : Visual Geometry Group
WHO : World Health Organization

LIST OF TABLES

Page

LIST OF FIGURES

Page

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 affects 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 beginning of July 2021, over 185 million people have been infected and more than 4 million people died because of COVID-19. For that reason, 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. Hence, it has been an important challenge and problem to find a procedure to classify COVID-19 with high success rate.

In this thesis, we suggested use of deep feature transformation from deep learning models to machine learning (ML) algorithms to classify patients COVID-19 infected via chest X-rays. In addition to image data, we also used the demographic information of patients during ML process to contribute to the information coming from deep features. Chapter 1 gives background information about our problem, the purpose of our study, the related literature survey and the structure of this thesis. The basic information about our image data, chest X-rays, are given in Chapter 2.

The problem we focused on is a binary classification between COVID-19 patients and other people. In order to solve this problem, we used data set 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 then augmented the train set with horizontal flip, vertical flip, 90 degrees of rotation, 180 degrees of rotation, and 270 degrees of rotation to increase the size of the train set. 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 was used on CNN experiments only. At the beginning of Chapter 5, the data set 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 model such that SGD with momentum, Adam and Adam with decoupled weight decay. The objective loss function was to minimize cross-entropy loss function which was common for each model. Each image sample was resized to 227 x 227, center cropped, converted to gray-scale, and then 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 ML algorithms. Pre-trained CNN models are the models that previously trained on ImageNet data set 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 ML 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 ML, 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 in 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 was mainly carried out in Python programming language, and a small part was done 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 for 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 0.5686, the specificity of 0.5837 the precision of 0.6863, the F1

score of 0.4955, and the AUC score of 0.5745. For the deep feature matrix obtained from ResNet-50 model weights, SVM with Ridge penalty, LR, LR with penalty, 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 penalty, LR, LR with Ridge penalty, 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. In general, 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 data set 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

Koronavirus 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. Temmuz 2021 başı itibariyle 185 milyondan fazla insan COVID-19 hastalığına yakalanmış ve 4 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 Ağı 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 Ağı (ESA) modelleridir. Her ESA modeli için özellikle üç farklı optimizasyon yöntemi uyguladık, ve bunlar SGD momentum, Adam ve ayrıştırılmış ağırlık düşüşüne sahip Adam'dır. Her model için ortak olarak kayıp fonksiyonununu çözebilmek adına çapraz-entropy kayıp fonksiyonunu kullandık. Her

bir görüntü 227 x 227 olarak yeniden boyutlandırıldı, merkezsel olarak kırpıldı, 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 aktarımlı 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 aktarımlı öğrenme ise, ESA modelinin özelliklerini tam bağlı model bloğundan çıkartarak makine öğrenmesi algoritmaları gibi başka bir yapay zeka tekniğinde öznitelik matrisi olarak kullanmaktır.

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 düzenlileştirme tekniklerini uyguladık. 4. Bölüm, makine öğrenmesine giriş, makine öğrenmesi algoritmaları ve düzenlileştirciler hakkında temel bilgiler ve çapraz doğrulama tekniğini içermektedir. Başlangıçtaki bölünmüş test kümemize özgü 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 kümesine, aktırılmış 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 yaş 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 düzenlileştirilmiş 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, içermiş 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, 0.5686 duyarlılık, 0.5837 özgüllük, 0.6863 kesinlik, 0.4955 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. Genel olarak, 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 ettiğmiz 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 July 10, 2021, COVID-19 has caused more than 185 million people to be infected and more than 4 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. Furthermore, the demographic information of a patient, i.e., the age and sex information, has a major role on how the patient is affected by COVID-19 [13].

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 [14] 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, COVID-19 is mostly confused with the diseases having pretty similar symptoms with pneumonia. 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, detecting and distinguishing COVID-19, pneumonia and similar diseases from chest X-rays may be a difficult task even for expert radiologists [15]. Therefore, computerized support systems are very much needed to help radiologists to analyze chest X-ray images.

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 [16]. On the other hand, machine learning has various strong classification algorithms on categorical data including either small or large information. 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, 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. This data set consists of samples with non-diseased, virus-induced pneumonia, bacterial-induced pneumonia, fungal-induced pneumonia, and lipoid-induced pneumonia labels, along with demographic information. Moreover, samples, whose label includes pneumonia, have detailed disease causes such as COVID-19, Influenza, Escherichia coli, Aspergillosis spp., etc. The study data set was constructed with the help of a data cleaning process consisting of two steps: 1) eliminating samples without demographic information, and 2) then separating samples into two classes as non-COVID-19 which includes samples not having COVID-19 and COVID-19 which includes samples having COVID-19.

1.2 Literature Review

With the increasing information and data on COVID-19, artificial intelligence researchers started to work on the diagnosis and classification of this disease. Since the success and effectiveness of machine learning and deep learning algorithms, which are sub-branches of artificial intelligence, are well-known, researchers were able to start working on COVID-19 classification problems without wasting time and reached satisfactory results immediately.

Ardakani et al. [17] worked on chest computed tomography (CT) images to detect whether a person has COVID-19 disease or not. They studied with ten different convolutional neural network (CNN) models, and compared them. The CNN models they compared were AlexNet, VGG-16, VGG-19, SqueezeNet, GoogleNet, MobileNet-V2, ResNet-18, ResNet-50, ResNet-101, and Xception. They achieved the

best results with ResNet-101 CNN model with 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.

Pathak et al. [18] used a deep transfer learning technique on CT images to classify non-COVID-19 and COVID-19 people. They used pre-trained ResNet-50 CNN architecture on ImageNet [19] data set to train their classification problem, and 10-fold cross-validation to prevent over-fitting. Then, they obtained the testing accuracy as 93.02%.

Ozturk et al. [20] 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 [21] and constructed theirs by 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.

Oh et al. [22] 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 base 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 [23] proposed four different deep learning models to solve a 3-class classification problem. The data set consist of chest X-ray images and the three classes were no finding, bacterial pneumonia, and COVID-19. A CNN model containing 4 convolutional layers and 3 fully-connected layers was 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 long short-term memory (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.

Al-falluji et al. [24] 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%.

Nour 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 were support vector machines, decision tree, and k-nearest neighbor, for final classification. The result achieved for features extracted from fc2 were 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 the studies mentioned in this section is summarized in the Table 1.1.

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

Authors	Technique	Data set	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 algorithms, cross-validation, the regularization approaches, and the machine learning (ML) algorithms experimented in this study such as support vector machines, logistic regression, k-nearest neighbor and linear discriminant analysis.

In Chapter 5, all experiments on convolutional neural networks and machine learning algorithms carried out in this study are given together with how the study data set 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, results of all the experiments are given together model performance metrics.

Finally, in Chapter 7, the conclusion of our study is presented with some 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. The first medical X-ray print can be found in Figure 2.1.

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.

The fewest photons are absorbed by the air, while the most photons are absorbed by the calcium in the bones. This is why bones appear white on X-ray images. Fat and other soft tissues moderately absorb X-rays, so they appear gray. Due to the aforementioned absorption properties of X-rays, the lungs appear blackish.

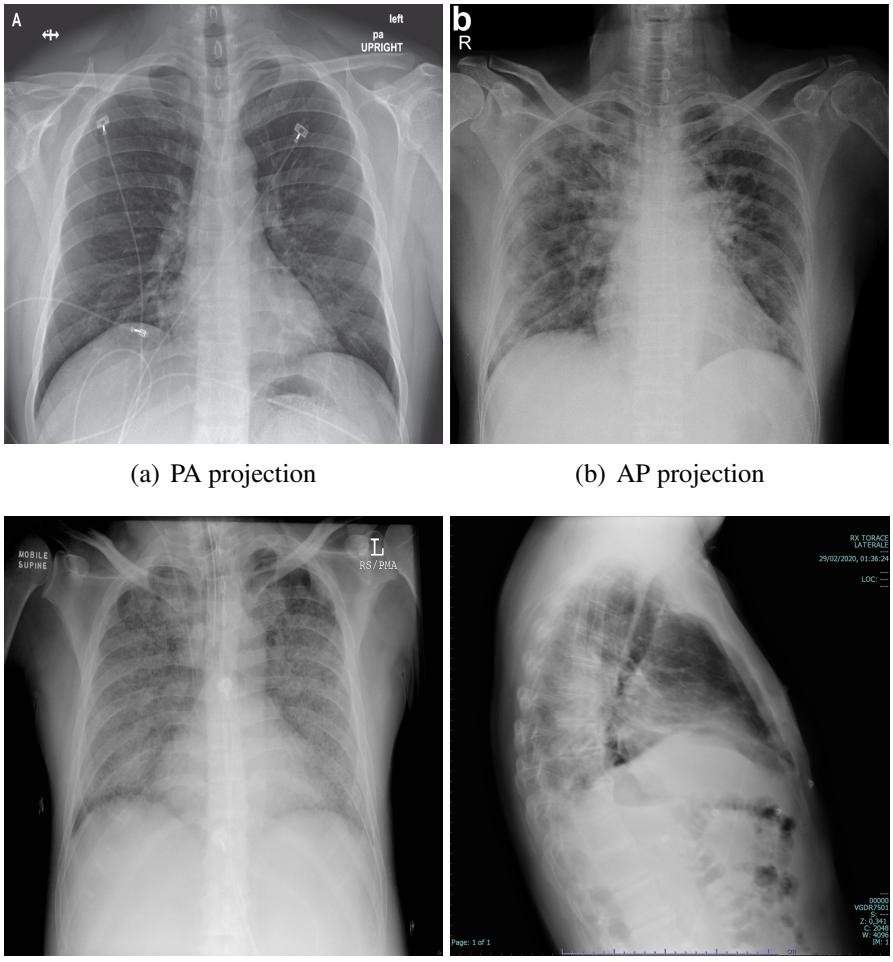
While the chest X-ray projection methods may vary from country to country or even from hospital to hospital, chest X-rays can be filmed through three main projection methods. The most preferred projection method is Posterior-Anterior (PA) view in which X-rays go through from patient's posterior to anterior. 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): the 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.

[1]Retrieved 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 function, and the optimization methods used to train deep learning algorithms. We conclude the section with the basics of convolutional neural networks (CNNs), CNN architectures used in this thesis, and the transfer learning concept.

3.1 The Basics of Deep Learning

Here we list basic terminologies associated with deep learning algorithms.

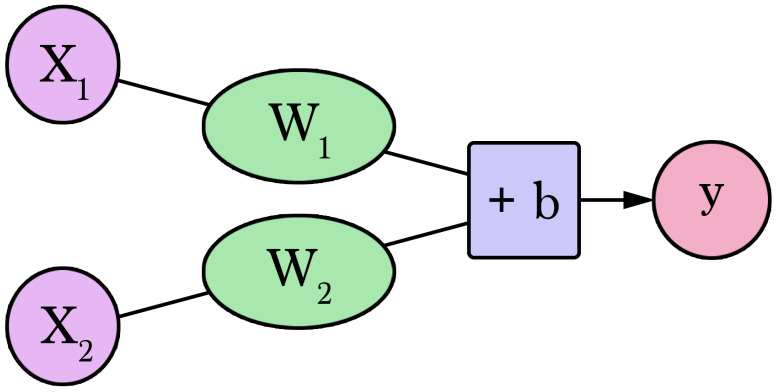


Figure 3.1 : A simple neural network with two neurons.[1]

- **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 into a desired output. For a given input data, after the input layer reads the data, the hidden layers extract and transform the information and feed the output layer with them. The output layer is the final layer and it is used to obtain the outputs of one iteration of network in desired format. The main inspiration of neural networks is the human brain structure. A simple example for a neural network consisting of two

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

neurons can be seen in Figure 3.1, and the formulation for the output in this neural network is:

$$y = \mathbf{W}^T \mathbf{x} + b\,, \tag{3.1}$$

where $\mathbf{x} \in \mathbb{R}^2$ is the input vector, $\mathbf{W} \in \mathbb{R}^2$ is the weight vector, $b \in \mathbb{R}$ is the bias term, and $y \in \mathbb{R}$ is the scalar output.

- **Nodes (Neurons) and Links**: Neural networks are made of nodes (neurons). The connections between the layer nodes are called as links. Each link has weight and bias parameters which are carried over neurons. Nodes receives the parameters of incident links, process them, and produces an output.
- **Weight and Bias**: Weight and bias terms are learnable parameters related to input data and they must be updated iteratively to reach the optimal cost value. At the initial iteration, they can be taken as zeros or a random initialization can be used. Both of them are transformed from input nodes to output nodes over the nodes of hidden layers via links in a neural network. Some sources include the bias term into the weight, and only use the weight value as well.
- **Feed-forward and Back-propagation**: Basically, a neural network which has no loops or cyclic links over nodes considered as a group of feed-forward networks. The direction of information flow is straight-forward. Input layer handles the input data, and the outputs of input layer goes through the hidden layer as input. There may be one or several number of hidden layers, and the output of each is the input of the next hidden layer. Finally, the output of last hidden layer feeds the output layer, and the output nodes forward their outputs to the loss function. During this process, the weights and bias values do not change. To update and find the most appropriate weight and bias values, back-propagation is used. To reach the optimal loss value, at each iteration of training, the negative gradient of the loss function is used to update the weight and bias terms. At the first iteration, this gradients are considered as zero or a random initialization is used. More information and examples can be found in Section 3.4.

- **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, data in the epoch can be dived into several smaller batches. Then, a batch is defined as a group of data samples selected from this data. The number of data samples 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 used on training process only. 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 used on testing process only. The training process should not see any of test data. Testing process is usually 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 may have under-fitting problem when it cannot capture or fit the data well. Under-fitting problem results in insufficient metrics during the training process since the learnable parameters are not optimized enough to reach at a satisfactory loss value. On the other hand, when the model fits the train data perfectly, it might have started to memorize the train data and may not learn something new from the data. Therefore, even though training metrics are nearly perfect, the testing metrics in the opposite direction of what is desired can be seen. The main reason for this is that the learnable parameters were directly adjusted for train data and not feasible for test data anymore. This situation is called as over-fitting. A validation set can be used to control the over-fitting by periodically checking the metrics of a data out of train set, while trying to rise above the under-fitting. By that way, a well-fitting or appropriate-fitting may be caught. The illustrations for three situations can be seen in the Figure 3.2.

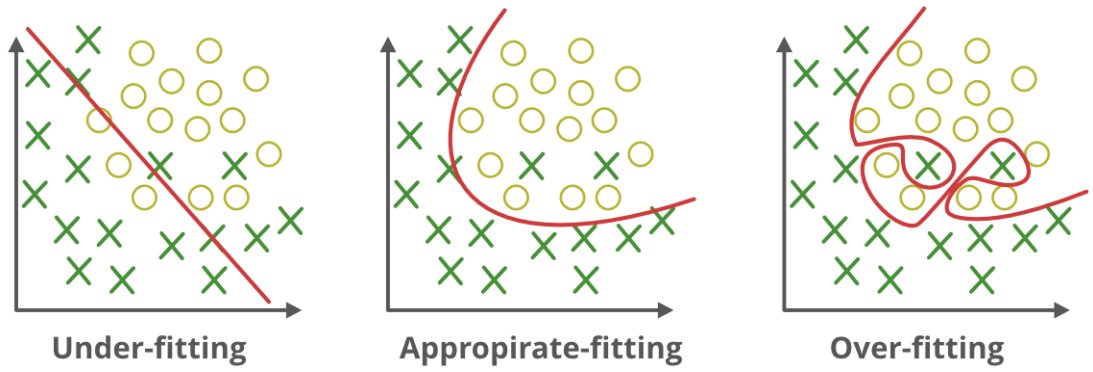


Figure 3.2 : Illustration of under-fitting, well-fitting, and over-fitting of models to the data.[2]

3.2 The Cross-Entropy Loss Function

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.

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

The cross-entropy loss function is one of the most commonly used loss functions to measure the performance of a classifier. The cross-entropy function calculates the cost through the negative log-likelihood function [25] which uses log class probability values computed by log-Softmax function [26]. Here the loss value increases together with the divergence between the grand-truth label and how likely this label is estimated.

The cross-entropy loss function for a sample, f, as a function of weights, is defined as given below:

$$f(\theta) = -\sum_{i=1}^{n} y_i \log(p_i)\,, \tag{3.2}$$

where n is the total number of classes, y_i is the observed class label, and p_i is the estimated class probability, which is a function of weight, θ, from the model of interest for the i^{th} class, respectively, with restriction $\sum_{i=1}^{n} p_i = 1$. Thus, for a binary classification problem, the cross-entropy loss function can be re-stated as:

$$f(\theta) = -\left(y \log(p) + (1-y) \log(1-p)\right)\,, \tag{3.3}$$

where y is the binary variable with two class labels 1 and 0, where 1, true class label, shows the presence of of the condition of interest, whereas 0 shows the opposite, and 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. However, it is always aimed to achieve the most satisfying results by using some optimization algorithms called as optimizers.

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 [27] 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 [28]. An example for a pseudo-code for SGD with momentum optimization is given in Figure 3.3. Note that the weight parameter θ is denoted by **x** in this figure.

Algorithm 1 SGD with momentum

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

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 [29]. An example for a pseudo-code for Adam optimization is given in Figure 3.4. Note that the weight parameter θ is denoted by **x** in this Figure.

3.3.3 Adam Decoupled Weight Decay Optimization Algorithm

Adam decoupled weight decay optimization algorithm (AdamW) does not use the regularization as the normalized by square root of second moment vector. Thus, it is only proportional to the weight itself [29]. An example for a pseudo-code for AdamW optimization is given in Figure 3.4. Again, note that the weight parameter θ is denoted by **x** in this figure.

The main difference of Adam and AdamW, as illustrated in Figure 3.4, is that, the weight decay is directly applied to the gradient of loss function on Adam, while it is decoupled on AdamW. Hence, the weight decay affects moment vectors and appears both in numerator and denominator during weight parameter update. Furthermore,

since it is in multiplicative form with learning rate parameter, the fine-tuning of learning rate and weight decay values become dependent to each other. On the other hand, weight decay is only applied at weight parameter update step, and it takes a separated charge during this update on AdamW.

Algorithm 2 Adam and AdamW

given $\alpha_t = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, w \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{x}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{x}_{t-1}) \leftarrow SelectBatch(\boldsymbol{x}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{x}_{t-1}) + w_t \boldsymbol{x}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ here, β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ here, β_2 is taken to the power of t
  $\eta_t \leftarrow SetScheduleMultiplier(t)$ ▷ can be fixed, decay, be used for warm restarts
  $\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \eta_t \left(\alpha_t \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + w_t \boldsymbol{x}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{x}_t$

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 analysis of visual problems. In this section, we give information on the fundamental layers and elements of CNN architectures.

A CNN model basically 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 layers, and one output layer.

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 the loss function f(θ). The weight matrix is defined as $\mathbf{W} = (\theta_1^T, \ldots, \theta_{n^{(i)}}^T) \in \mathbb{R}^{n^{(i)}} \times$ $m^{(i)}$ where $n^{(i)}$ and $m^{(i)}$ are the number of inputs for i^{th} layer and the size of each

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

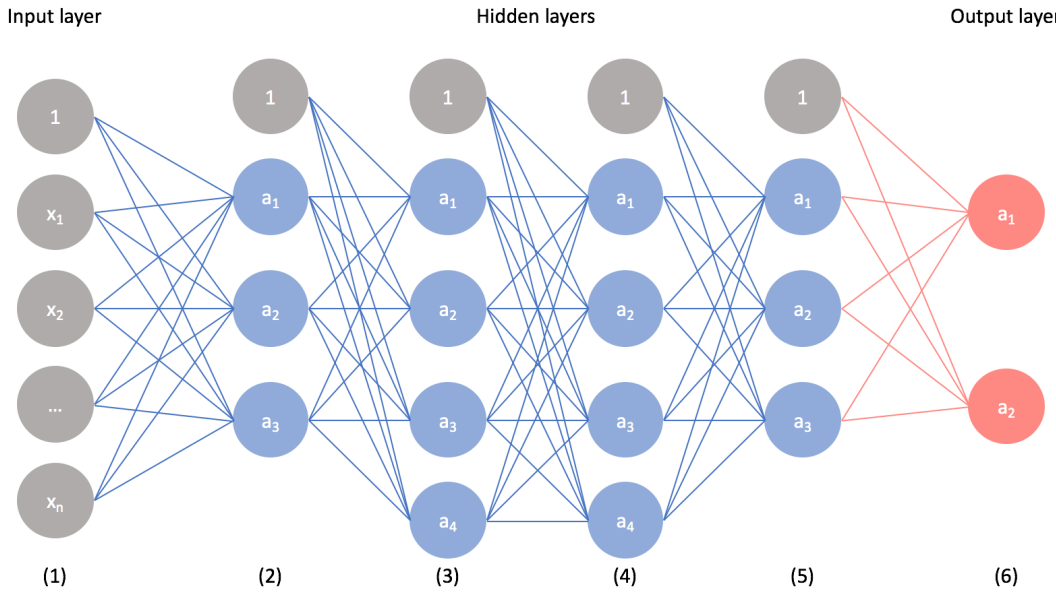


Figure 3.5 : An example for a simple CNN architecture.[3]

input in this layer, respectively. Then the weight space is updated with these gradients by the chosen optimizer such as SGD momentum, Adam, or AdamW. The gradient of objective loss function $f(\theta)$ at a fixed weight vector θ_t is computed via directional derivatives and chain rule and represented by $\nabla f(\theta_t)$.

Let's consider a simple CNN architecture illustrated in Figure 3.5 where each layer is denoted by $\mathbf{a}^{(i)}$ with $i = 1,2,\ldots,6$. Assume there exists n number of x_i initial input nodes, where $\mathbf{x} = \mathbf{a}^{(1)} = (x_1,\ldots,x_n)$ is an n-dimensional input layer vector. Furthermore, the layers $\mathbf{a}^{(i)}$, with $i = 2,\ldots,5$, are the hidden layers. For instance, $\mathbf{a}^{(3)} = (a_1^{(3)},a_2^{(3)},a_3^{(3)},a_4^{(3)})$ and $\mathbf{a}^{(5)} = (a_1^{(5)},a_2^{(5)},a_3^{(5)})$. Lastly, the output layer feeding the loss function is $\mathbf{a}^{(6)} = (a_1^{(6)},a_2^{(6)})$, and the output y value is determined by computing the class probabilities from $\mathbf{a}^{(6)}$.

Then the gradient of loss function f at the fixed weight vector θ_t is evaluated as:

$$\nabla f(\theta_t) = \begin{bmatrix} \frac{\partial f(\theta_t)}{\partial \theta_1} \\ \vdots \\ \frac{\partial f(\theta_t)}{\partial \theta_n} \end{bmatrix}. \tag{3.4}$$

The gray nodes in CNN architecture given in Figure 3.5 are bias terms. Assume that the objective loss function is the cross-entropy loss function f and our hypothesis function is to create a CNN architecture in the form $h_i : \mathbb{R}^{k^{(i)}} \to \mathbb{R}^{t^{(i)}}$, where $k^{(i)}$ and $t^{(i)}$ vary

according to the corresponding layer. Then define $\mathbf{a}^{(i)} = h_i(\mathbf{W}^{(i)}\mathbf{a}^{(i-1)} + b^{(i)})$ for each i^{th} layer, where $\mathbf{W}^{(i)}$ represents the weights of links from i^{th} to $(i+1)^{th}$ layers, and $b^{(i)}$ is the bias term accompanying to these weights. Moreover, let us denote the weight of the link between the j^{th} node of i^{th} layer and the z^{th} node of $(i+1)^{th}$ layer as $\mathbf{W}^{(i)}_{j,z}$. We obtain the backward pass of our loss function with respect to the first weight matrix and bias terms by chain rule, respectively, as follows:

$$\frac{\partial f}{\partial \mathbf{W}^{(1)}} = \frac{\partial f}{\partial \mathbf{a}^{(6)}} \frac{\partial \mathbf{a}^{(6)}}{\partial \mathbf{a}^{(5)}} \frac{\partial \mathbf{a}^{(5)}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{W}^{(1)}}, \text{ and} \tag{3.5}$$

$$\frac{\partial f}{\partial b^{(1)}} = \frac{\partial f}{\partial \mathbf{a}^{(6)}} \frac{\partial \mathbf{a}^{(6)}}{\partial \mathbf{a}^{(5)}} \frac{\partial \mathbf{a}^{(5)}}{\partial \mathbf{a}^{(4)}} \frac{\partial \mathbf{a}^{(4)}}{\partial \mathbf{a}^{(3)}} \frac{\partial \mathbf{a}^{(3)}}{\partial \mathbf{a}^{(2)}} \frac{\partial \mathbf{a}^{(2)}}{\partial b^{(1)}}. \tag{3.6}$$

For instance, if the forward pass connection between $\mathbf{a}^{(4)}_2$ and $\mathbf{a}^{(5)}_1$ is $W^{(4)}_{2,1}$, we use the directional derivative rule while computing $\frac{\partial f}{\partial \mathbf{a}^{(6)}}$ for the backward pass gradient to update $W^{(4)}_{2,1}$ value as:

$$\frac{\partial f}{\partial W^{(4)}_{2,1}} = \frac{\partial f}{\partial \mathbf{a}^{(6)}} \frac{\partial \mathbf{a}^{(6)}}{\partial \mathbf{a}^{(5)}_1} \frac{\partial \mathbf{a}^{(5)}_1}{\partial W^{(4)}_{2,1}} = \Big(\frac{\partial f}{\partial \mathbf{a}^{(6)}_1} \frac{\partial \mathbf{a}^{(6)}_1}{\partial \mathbf{a}^{(5)}_1} + \frac{\partial f}{\partial \mathbf{a}^{(6)}_2} \frac{\partial \mathbf{a}^{(6)}_2}{\partial \mathbf{a}^{(5)}_1} \Big) \frac{\partial \mathbf{a}^{(5)}_1}{\partial W^{(4)}_{2,1}}. \tag{3.7}$$

In the same way, the gradient for $b^{(5)}$ can be computed as:

$$\frac{\partial f}{\partial b^{(5)}} = \frac{\partial f}{\partial \mathbf{a}^{(6)}} \frac{\partial \mathbf{a}^{(6)}}{\partial b^{(5)}} = \frac{\partial f}{\partial \mathbf{a}^{(6)}_1} \frac{\partial \mathbf{a}^{(6)}_1}{\partial b^{(5)}} + \frac{\partial f}{\partial \mathbf{a}^{(6)}_2} \frac{\partial \mathbf{a}^{(6)}_2}{\partial b^{(5)}}. \tag{3.8}$$

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 one or multiple windows named as filter (or kernel) that can be of different sizes such as 2×2, 3×3, and so on. The coefficients, which are included in filters, change at 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.

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.

Sometimes, an inconsistency between filter, stride, and input layer sizes can be seen while trying to use desired convolutional layer. In this situation, 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. This method can be also used to pay attention to the edges and corners of original input matrix.

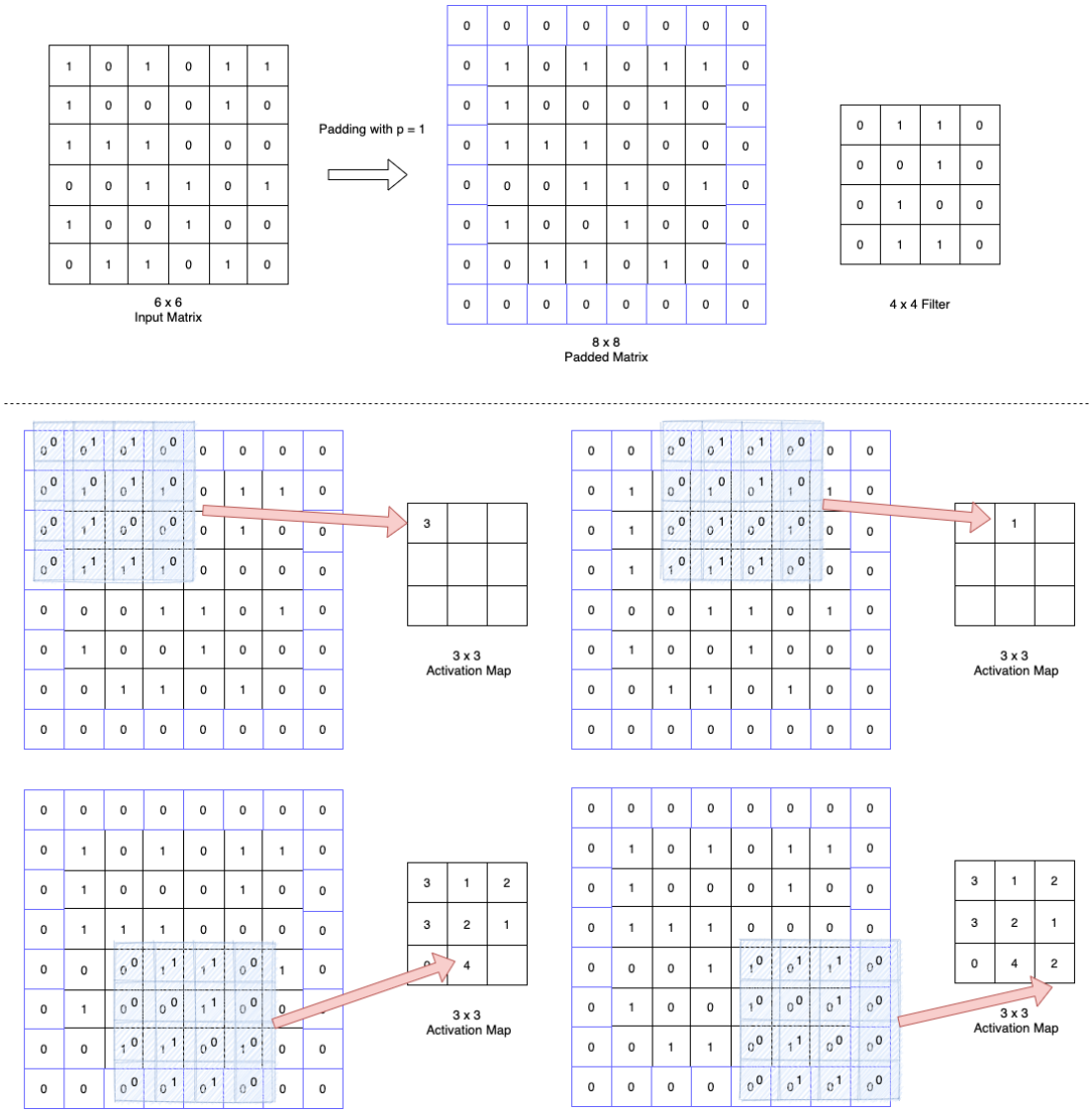


Figure 3.6 : Activation map construction with a filter window size of 4×4 and stride value of 2.

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 Figure 3.6 gives a summary about how activation map is constructed for a given input matrix and a filter.

Let us give another example to see the details about computations about it. Assume that the input data holds raw pixel values of an image of width 223, height 223, and three color channels red, green, and blue so that it has a size of $223 \times 223 \times 3$. Let's choose 2 filter windows in the size of 5×5, the stride (step size) as 2, and the padding size as 1. If we denote the input size by $W_{input} \times H_{input} \times D_{input}$, the padding by P, the number of filters by K, the size of a filter window by $F \times F$ and the stride by S, then we have:

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

The size of activation map is calculated as:

$$W_{output} = ((W_{input} - F + 2P)/S) + 1\,, \tag{3.9}$$

$$H_{output} = ((H_{input} - F + 2P)/S) + 1\,, \quad \text{and} \tag{3.10}$$

$$D_{output} = K\,. \tag{3.11}$$

Consequently, the activation map of size of $112 \times 112 \times 2$ is produced through the equations (3.9), (3.10), and (3.11) respectively.

3.4.2 Rectified Activation Function

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 (3.12), is called as Rectified Linear Units (ReLU) which is the most common used in neural networks. ReLU is defined as:

$$f(x) = \max(0, x)\,. \tag{3.12}$$

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. Moreover, a simple example on maximum pooling can be found in Figure 3.7.

- **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.

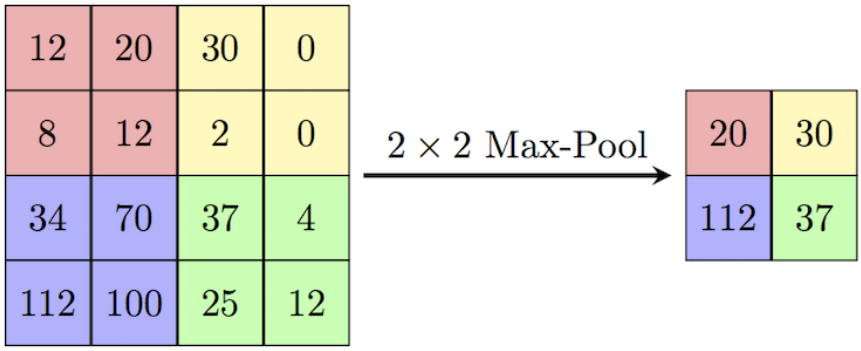


Figure 3.7 : Maximum pooling operation sample with 2×2 window and stride value of 2.[4]

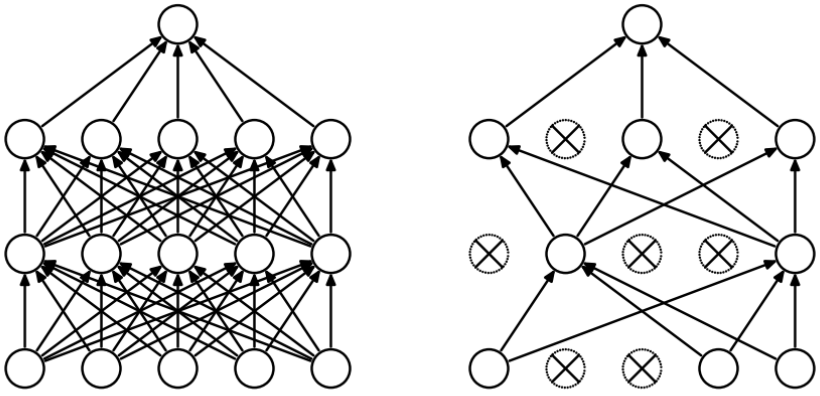
3.4.4 Batch Normalization

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

[4]Retrieved from: `https://computersciencewiki.org/index.php/Max-pooling_/_Pooling` on April 3, 2021.

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]. An illustration about drop-out can be seen in the Figure 3.8.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.8 : (a) A standard neural network, (b) A neural network after applying drop-out [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 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.A simple fully-connected layer is given in the figure 3.9.

3.5 Transfer Learning

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

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

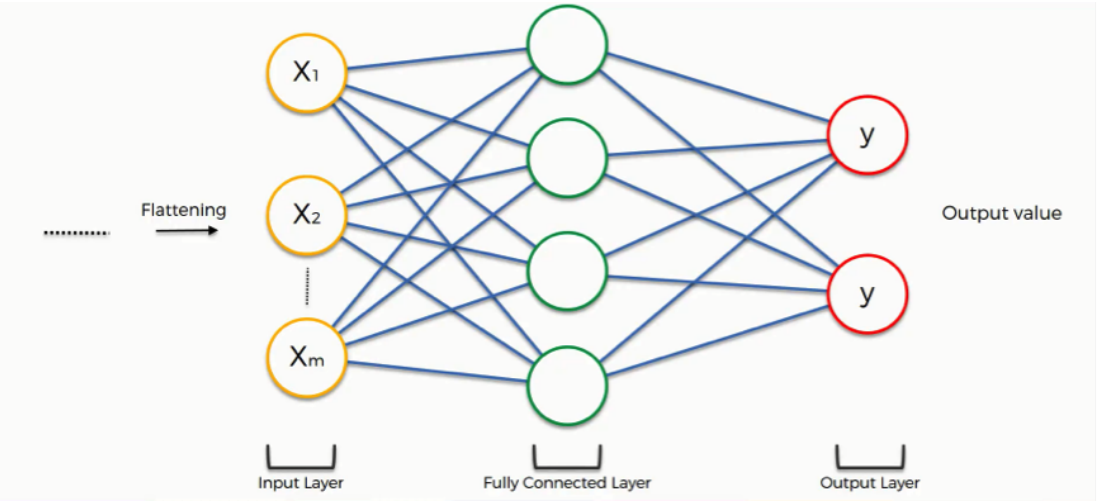


Figure 3.9 : The usage of a full-connected layer for a binary classifier.[5]

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 generally 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 data set may be considered as small.

To prepare a pre-trained CNN model for visual artificial intelligence problems, ILSVRC 2012 ImageNet [19] data set, 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. In 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 exists 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 others, and freezing all layers on the convolutional block. The choice of strategy, which is called as fine-tuning operation, depends on the size of data set and the similarity between the data set used in pre-training process and the data set 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. The mentioned three strategies can be seen in the Figure 3.10.

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

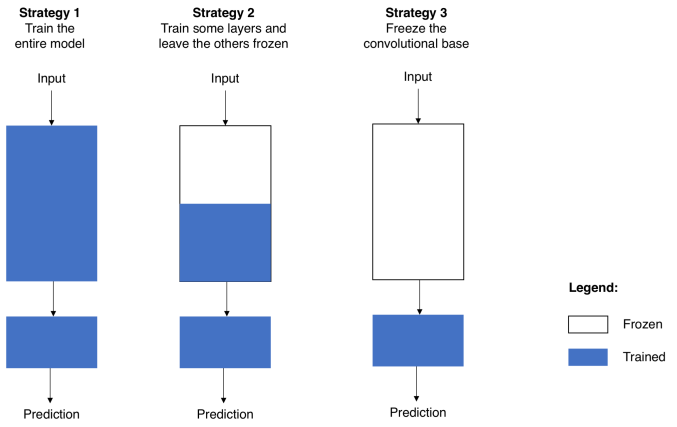


Figure 3.10 : Different fine-tuning strategies on a pre-trained model.[6]

On the other hand, 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 data set. A real application of deep feature transfer learning as a road-map, which also used in this thesis, can be found in Figure 3.11.

3.6 CNN Models

There are several 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.

In this section, the state-of-the-art CNN models used in this 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 [19] 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. 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 3.12. The remaining three layers are the fully-connected layers which

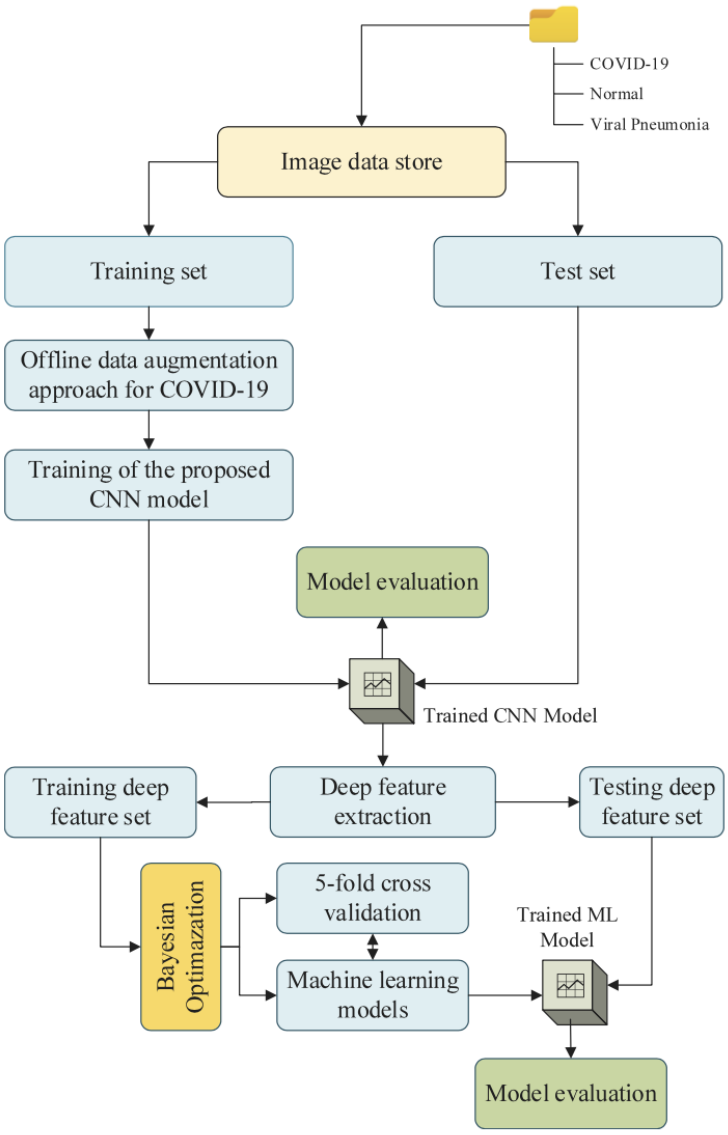


Figure 3.11 : Deep feature extraction from CNN models and using them in machine learning models [4].

have the sizes of 2048, 2048, and the number of classes, respectively. In Figure 3.12, the number of classes are considered as 1000; thus, the last fully-connected layer has the size of 1000. 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.

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

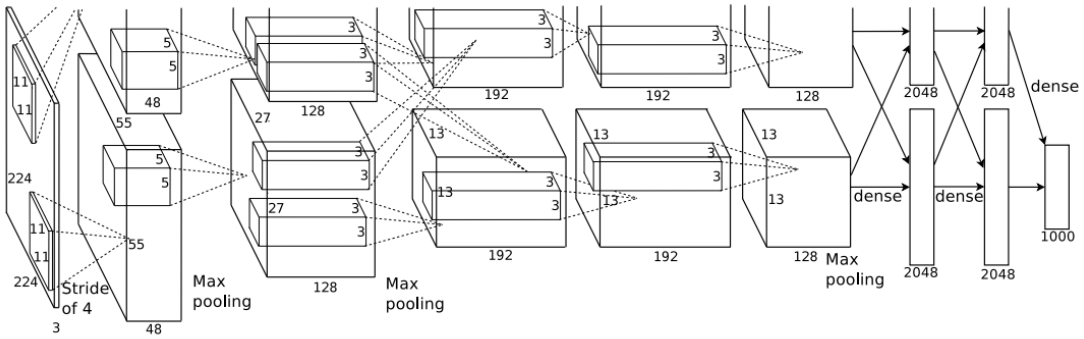


Figure 3.12 : Illustration of AlexNet architecture with a size of $224 \times 224 \times 3$ input image and 1000 class prediction support. [5].

layer in much deeper. This model resulted in high performance results on ILSVRC 2015 ImageNet [19] data set and won the 1st place on the corresponding task. In the Figure 3.13, it can be seen that the layer $a^{[l]}$ feeds two subsequent layers, namely, $a^{[l+1]}$ and $a^{[l+2]}$.

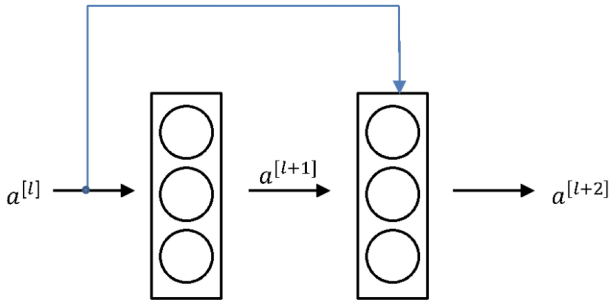


Figure 3.13 : An illustration for residual block. Here, $a^{[l]}$ is the starting activation layer and $a^{[l+2]}$ is the fed activation layer.[7]

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

ResNet-18

The first convolution layer includes a filter with a size of 7×7 and scans with a stride value of 2, then maximum pooling is applied with a 2×2 window along with a stride value of 2. Afterwards, 8 residual blocks, where each includes 2 convolutional layers with 3×3 filters, come. Hence, there exists 16 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following a

[7]Retrieved from: `http://datahacker.rs/deep-learning-residual-networks/` on April 4, 2021.

fully-connected layer with neurons in the size of the number of classes, which is 1000 in Figure 3.14.

ResNet-34

The first convolution layer includes a 7×7 filter and scans with a stride value of 2, then maximum pooling is applied with 2×2 window and stride value of 2. Afterwards, 16 residual blocks, such that each of them includes 2 convolutional layers with 3×3 filters, come. That is, there exists 32 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following a fully-connected layer with neurons in the size of the number of classes, which is 1000 in Figure 3.14.

ResNet-50

The first convolution layer includes a 7×7 filter and scans with a stride value of 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 and 1×1 filters in order, come. That is, there exists 48 convolutional layers inside of residual blocks. After the residual blocks, there is an average pooling following a fully-connected layer with neurons in the size of the number of classes, which is 1000 in Figure 3.14.

3.6.3 Visual Geometry Group

Simonyan and Zisserman [30] 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 [19] dataset and achieved high success results. This study has showed the importance of depth in image description models. The two types of Visual Geometry Group (VGG) architectures explained below can be seen in Figure 3.15. In the figure, the number of classes are considered as 1000.

VGG16

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

layer name	output size	18-layer	34-layer	50-layer
conv1	112×112	7×7, 64, stride 2		
conv2_x	56×56	3×3 max pool, stride 2		
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax		
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9

Figure 3.14 : The architectures of ResNet-18, ResNet-34, and ResNet-50 [6].

The model architecture starts with an input layer, and there are following 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 have 4096 neurons and the last one has neurons in the size of the number of classes, which is 1000 in Figure 3.15. The ReLU activation function was used after all the hidden layers.

VGG19

The model architecture starts with an input layer, and there are following 14 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 have 4096 neurons and the last one has neurons in the size of the number of classes, which is 1000 in Figure 3.15. The ReLU activation function was used after all the hidden layers.

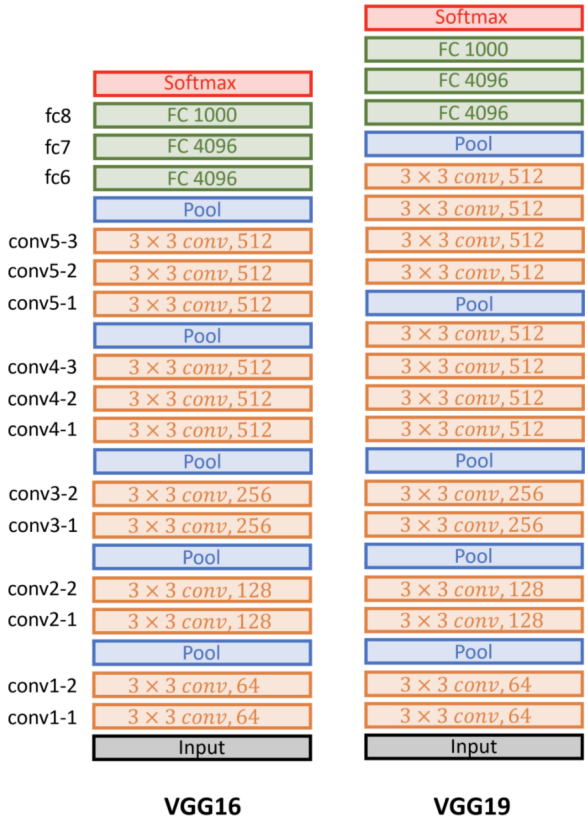


Figure 3.15 : An illustration of VGG architectures.[8]

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. The Figure 4.1 shows a simple road-map of a machine learning task.

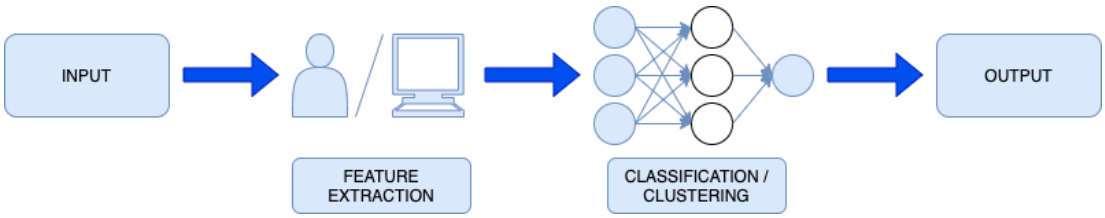


Figure 4.1 : Sample machine learning road map.

4.1 The Basics of Machine Learning

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

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

The majority of the problems in ML classified as either supervised or unsupervised learning approaches. Clustering as an example for unsupervised learning, and classification and regression as two examples for supervised learning are given in Figure 4.2.

4.1.1 Supervised Learning

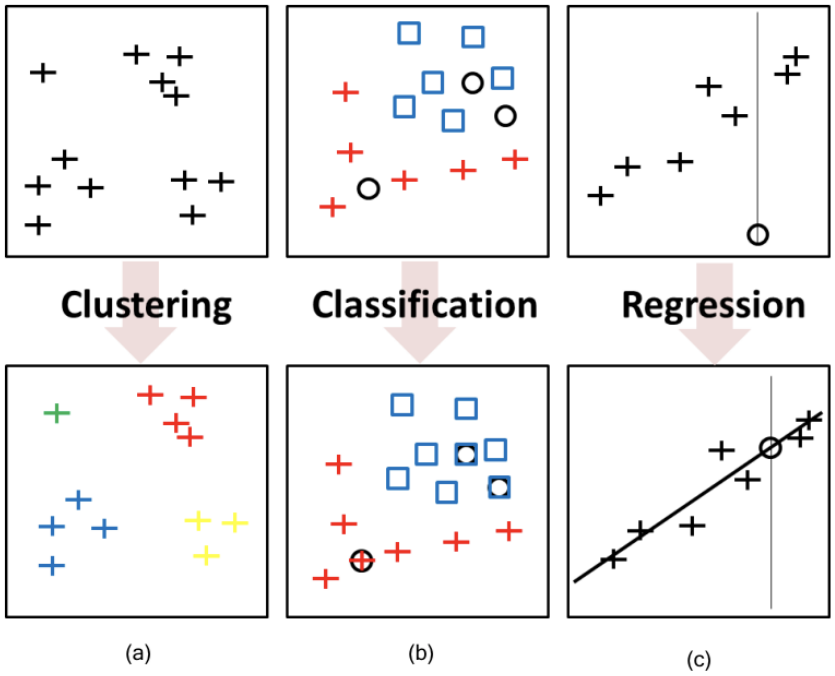


Figure 4.2 : (a) Clustering problem, (b) Classification problem, and (c) Regression problem [7].

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

The task in this thesis includes a data set where each datum has its true label value. Thus, all ML models used in this thesis are supervised learning algorithms.

4.1.2 Unsupervised Learning

The problems having data with unknown true values are solved by unsupervised learning. Some data sets can be deliberately left unlabeled since it is too hard to label the data and requires too much time, expert people, or special devices to label. On the other hand, not all problems needs to be classified, there are problems that only needs to be grouped. Clustering tasks are constructed on these kind of data sets and solved by unsupervised learning.

4.2 Cross-Validation

As in deep learning, over-fitting and under-fitting creates serious problems in machine leaning. Cross-validation (CV) methods can be used to overcome these problems. In the CV method, all the data is divided into train and test sets. Here, it should be noted that the train and test data sets must be strictly separated from each other, and any sample on test set should not be seen during training process.

4.2.1 K-Fold Cross-Validation

In 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 points. 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 the 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. A sample illustration of K-Fold cross-validation process can be seen in the Figure 4.3.

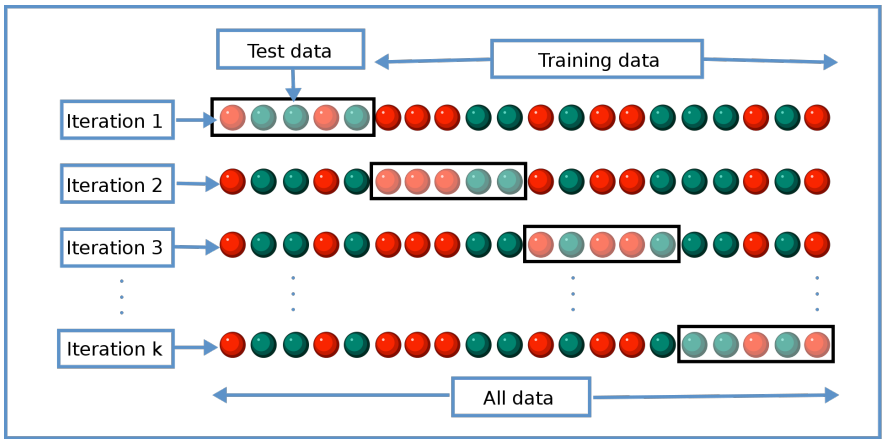


Figure 4.3 : K-fold cross-validation.[1]

4.2.2 Leave-One-Out

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

[1]Retrieved from: `https://towardsdatascience.com/cross-validation-c4fae714f1c5` on April 10, 2021.

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.

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:

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

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. This constraint enables to control the magnitude of regularization, and fine tune the hyper-parameter of regularizer.

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

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

$$\lambda \|\mathbf{w}\|_1 , \tag{4.2}$$

where $\mathbf{w}$ is the the weight coefficients to estimate and $\|.\|_1$ is the L1 norm such that:

$$\|\mathbf{w}\| = \sum_{i=1}^{p} |w_i|, \tag{4.3}$$

for a vector $\mathbf{w}$ with length of p. LASSO penalty may produce non-unique solutions [31].

4.3.2 L2 Regularization

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

$$\lambda \|\mathbf{w}\|_2^2 , \tag{4.4}$$

where $\mathbf{w}$ is the weight coefficients to estimate, and $\|.\|_2^2$ is the squared L2 norm such that:

$$\|\mathbf{w}\|_2^2 = \sum_{i=1}^{p} w_i^2, \tag{4.5}$$

for a vector $\mathbf{w}$ with length of p.

Since L1 regularizer takes the absolute value and L2 regularizer takes the square, the cost of outlier samples exponentially increase. For this reason, L2 regularization is less robust to outliers than L1 regularizer.

4.4 Machine Learning Algorithms

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

4.4.1 Support Vector Machines

The current support-vector machines algorithm is proposed by Corinna Cortes and Vladimir Vapnik at 1995 [32]. While it is mainly used for supervised learning

problems, the version for clustering, which is stated as support-vector clustering [33], can be used for unsupervised learning problems.

A support vector machine (SVM) constructs a set of hyper-planes in a high-dimensional or an infinite-dimensional space. It is usually 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 this thesis, we have a 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, in turn, margins differ depending on the kernel function used. The original hyper-plane algorithm proposed by Vapnik [32] 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 [34] suggest replacing the linear function with another nonlinear kernel functions. This technique is called as kernel trick. An illustration about how SVM creates the hyper-planes can be viewed in the Figure 4.4.

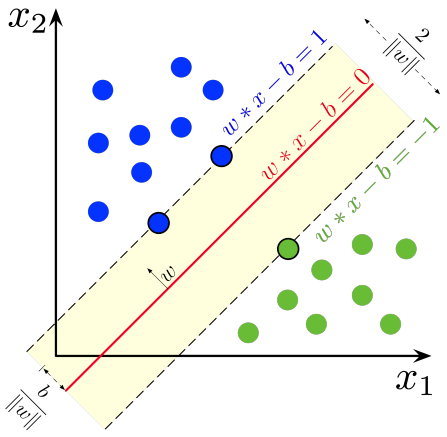


Figure 4.4 : A Hyper-plane and margins of an SVM for a binary classification problem.[2]

A linear SVM hyper-plane is defined as:

$$\mathbf{w}^T\mathbf{x} - b = 0, \tag{4.6}$$

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

where **x** is a p-dimensional real vector, **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, or inversely we want to minimize $\frac{\|\mathbf{w}\|_2}{2}$.

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

- **hyper-plane:**

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

- **margin for label +1:**

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

- **margin for label -1:**

$$H_{-1} = \mathbf{w}^T\mathbf{x} - b = -1\,. \tag{4.9}$$

Thus, we have the following condition and result relationships for a p-dimensional sample $\mathbf{x}^i$ vector with its observed label y^i where y^i takes 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 combine the conditions above, we obtain the following common condition:

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

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

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

Besides, the squared hinge loss is defined as:

$$loss_{squared\_hinge}(\mathbf{w}, b) = \sum_{i=1}^{n} (\max(0, 1 - (\mathbf{w}^T\mathbf{x}^i - b)y^i))^2. \tag{4.12}$$

The constrained optimization problem of SVM is derived as:

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

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:

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

where $\mathscr{L}$ is the lagrange function, $\mathbf{x}_i$ and y_i are a p-dimensional real valued feature vector and its label, that is -1 or +1, respectively, and n is the number of samples.

An optimization problem can be handled in two perspectives which are the primal problem as a minimization problem and the dual problem as a maximization problem. This kind of optimization problems includes a duality value called as duality gap which is the difference between the lower bound of primal problem and the upper bound of dual problem:

$$\text{duality gap} = \text{lower bound}_{\text{primal}} - \text{upper bound}_{\text{dual}}\,. \tag{4.15}$$

If duality gap $= 0$, it is called as strong duality. That is the solution found from either dual or primal problem is the optimal solution of original optimization problem. However, if duality gap > 0, it is called as weak duality and one can try to reduce the duality to zero by using constraints. In that case, the solution of original optimization problem is not the best; but, it can be still used depending on the cost value and tolerance. The illustration of dual and primal problems and duality gap can be viewed in Figure 4.5.

Further details on constraint optimization of SVM, primal and dual problems and their solutions can be found at [35, pg. 13-19].

Penalty Terms

[3]Retrieved from: `https://kseow.com/svm.html`.

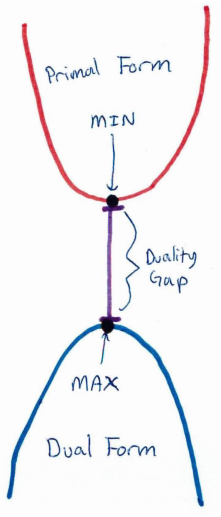


Figure 4.5 : The dual and primal problems, and the duality gap between their extreme points.[3]

To decrease the effects of original constraints of SVM problems and construct more feasible hyper-planes regarding to outlier samples, we use penalty terms adding to original SVM problems. Here λ always refers to the regularization parameter which is a new hyper-parameter to be tuned during this subsection.

- **L2 Regularizer:** According to Koshiba and Abe [36, pg. 2054-2059], let's first recall the primal form of SVM optimization problem in equation (4.13). We can modify the problem as:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} loss(\mathbf{w},b) + \lambda \frac{\|\mathbf{w}\|_2^2}{2}, \tag{4.16}$$

  where the loss function can be either hinge or squared hinge loss function.

  Then, the corresponding dual problem can be obtained as:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^i y^i \left(\mathbf{x}^{iT} \mathbf{x}^j + \lambda \delta_{ij} \right) \quad \text{such that} \quad \sum_{i=1}^{n} \alpha_i y^i = 0\,, \tag{4.17}$$

  where α_i 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 Koshiba and Abe [36, 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:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} loss(\mathbf{w},b) + \lambda \|\mathbf{w}\|_1 \,. \tag{4.18}$$

Then, the corresponding dual problem can be obtained as:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \, y^i y^i \, \mathbf{x}^{iT} \mathbf{x}^j \quad \text{such that } \sum_{i=1}^{n} \alpha_i y^i = 0 \text{ and } \alpha_i \leq \frac{1}{\lambda}, \quad (4.19)$$

where α_i are lagrange multipliers given in the equation (4.14), respectively.

Kernel Trick

Kernel trick uses a transformation function $\phi(.)$ for mapping samples into a new space so that samples become linearly separable. Then all appearances of samples are replaced with the transformation $\phi(.)$. Now, the kernel function for p-dimensional feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is defined as:

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

This replacement affects the whole structure of SVM optimization problem. The most commonly used kernel functions, which are also used in this thesis, are:

- **Linear function:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j. \quad (4.21)$$

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

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|_2^2), \quad (4.22)$$

  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:**

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i^T \mathbf{x}_j + c), \quad (4.23)$$

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

4.4.2 Logistic Regression

Logistic regression (LR) aims to solve binary classification problems. In the logistic regression 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 $Pr(\mathbf{x}) > 0.5$, then the prediction is true. By using the logistic sigmoid function, the model is obtained as:

$$Pr(\mathbf{x}) = \frac{1}{1+e^{-(\mathbf{w}^T\mathbf{x}+b)}}, \tag{4.24}$$

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. In the Figure 4.6, the behaviours of sigmoid function for different $\mathbf{w}$ values are given. Note that, in this figure, the dimensions of samples and weights are one, i.e. $d = 1$.

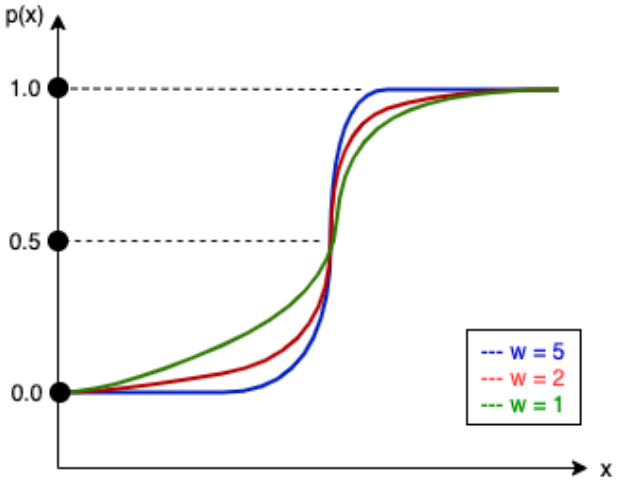


Figure 4.6 : Logistic sigmoid function with different scaling w values.

The sharpness of logistic curve increases with higher $\mathbf{w}$ values, and vice versa.

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

$$\ln\left(\frac{Pr(\mathbf{x})}{1-Pr(\mathbf{x})}\right) = \mathbf{w}^T\mathbf{x}+b\,. \tag{4.25}$$

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

For simplicity, let's define $z = \mathbf{w}^T\mathbf{x}+b$. Then, we have the following constraints:

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

Now, we can construct the natural logarithm of the loss function for a sample pair $(\mathbf{x}^i, y^i)$ for $i \in \{1,...,n\}$ as follows:

$$loss^i(\mathbf{w},b) = -y\ln(Pr(z^i)) + (1-y)\ln(1-Pr(z^i)). \tag{4.26}$$

Consequently, the optimization problem is yielded as:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} loss^{i}(\mathbf{w},b). \tag{4.27}$$

Penalty Terms

To take the loss function under control by a constraint while constructing the logistic curve separating classes, penalty terms are used. Here λ always refers to the regularization parameter during this subsection.

- **L2 Regularizer:**

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} loss^{i} + \lambda \left\|w\right\|_2^2. \tag{4.28}$$

- **L1 Regularizer:**

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} loss^{i} + \lambda \left\|w\right\|_1. \tag{4.29}$$

Optimizers

In this thesis, to solve the optimization problem on LR problem, we use the following solvers presented by scikit-learn package [37] in Python programming language:

- Newton's method (newton-cg) [38],
- Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (lbfgs) [39],
- A library for Large Linear Classification (liblinear) [40],
- Stochastic average gradient (sag) [41], and
- SAGA [42].

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 which are compatible with regularizers.

4.4.3 K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a classification algorithm developed by Fix and Hodges (1951) [43]. KNN can be used for regression problems as well; however, at this time, the output for an object is not a class label, but the property value which is the average

Table 4.1 : Compatibility of logistic regression optimization solvers with regularizers.

Solver	Regularizer		
	No Penalty	L1 Penalty	L2 Penalty
newton-cg	✓		✓
lbfgs	✓		✓
liblinear		✓	✓
sag	✓		✓
saga	✓	✓	✓

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

KNN 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 denoted 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 can be used. The metrics used in this thesis are:

   - **Euclidean Distance:**

   $$d(\mathbf{x}^*, \mathbf{x}^i) = \sqrt{\sum_{j=1}^{d} (\mathbf{x}^*{}_j - \mathbf{x}^i{}_j)^2}\,, \tag{4.30}$$

   - **Manhattan Distance:**

   $$d(\mathbf{x}^*, \mathbf{x}^i) = \sum_{j=1}^{d} |\mathbf{x}^*{}_j - \mathbf{x}^i{}_j|\,, \quad \text{and} \tag{4.31}$$

   - **Chebyshev Distance:**

   $$d(\mathbf{x}^*, \mathbf{x}^i) = \max(|\mathbf{x}^*{}_1 - \mathbf{x}^i{}_1|, |\mathbf{x}^*{}_2 - \mathbf{x}^i{}_2|, \ldots, |\mathbf{x}^*{}_d - \mathbf{x}^i{}_d|)\,. \tag{4.32}$$

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 at which class label has the majority in appearance around K nearest neighbors. K is preferred as an odd number. For example, if K is chosen as 3 and these three nearest neighbors have class labels as $\{+1,-1,+1\}$, then the majority voting

predicts the target label as +1 since it appears more frequent 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 approach 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 these three nearest neighbors have 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.

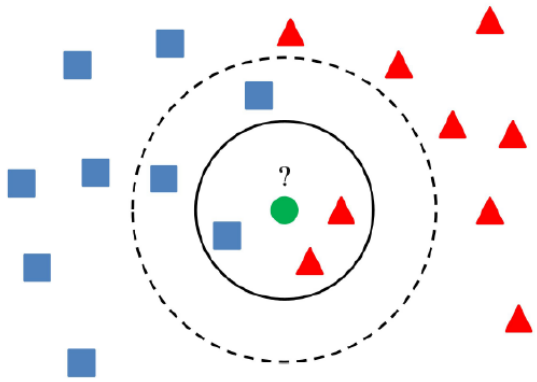


Figure 4.7 : A KNN example illustration for detecting the class of the green sample.[4]

Figure 4.7 illustrates a classification problem for the the sample colored in green. When KNN algorithm with K = 3 is used, the green sample is labeled as red both by majority voting and weighted majority voting. However, when K = 5, use of majority voting or weighted majority voting results in a different label for the green sample. While majority voting labels it as blue, weighted majority voting labels it as red.

Finding Neighborhood

In this thesis, three algorithms presented by scikit-learn package [37] in Python programming language are used to find neighborhoods of samples under KNN algorithm.

- **Brute Force:** Distances between all pairs of samples are computed. Brute force may be quite efficient for small sized dataset.

[4]Retrieved from: `https://ai.plainenglish.io/k-nearest-neighbors-from-scratch-633dfbeac740` on April 17, 2021.

- **K-dimensional Tree:** When Brute force becomes inefficient as the size of data set increases, 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 samples x_1 and x_2 are very close to each other, then we can directly conclude that the samples x_0 and x_2 are very distant to each other as well. 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 [44].
- **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 is more effective, especially in high dimensions. Further information can be found at [45].

4.4.4 Linear Discriminant Analysis

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 and $Pr(c_i|\mathbf{x})$ is the probability of $\mathbf{x}$ belonging to class c_i over k number of classes as given below:

$$Pr(c_i|\mathbf{x}) = \frac{Pr(\mathbf{x}|c_i)Pr(c_i)}{Pr(\mathbf{x})}, \tag{4.33}$$

where $Pr(\mathbf{x}|c_i)$ is the likelihood of observing $\mathbf{x}$ given class c_i, $Pr(c_i)$ is the prior probability of belonging to class c_i, $Pr(\mathbf{x})$ is the marginal likelihood of observing $\mathbf{x}$, and $i = 1,\ldots,k$. Here, we can discard the $Pr(\mathbf{x})$ since observing $\mathbf{x}$ is independent of class.

We can also take the logarithm of $Pr(c_i|\mathbf{x})$ in equation (4.33) since it is monotonically increasing. Then, a new function $g_i(\mathbf{x})$ can be defined as:

$$g_i(\mathbf{x}) = \ln(Pr(c_i|\mathbf{x})) = \ln(Pr(\mathbf{x}|c_i)) + \ln(Pr(c_i)) \,. \tag{4.34}$$

Here, we suppose that the class density $Pr(\mathbf{x}|c_i)$ follows a multivariate normal distribution with mean μ_i and variance Σ_i, where Σ_i describes class-specific covariance matrix which is generally considered as $\sigma^2 I$.

$$Pr(\mathbf{x}|c_i) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp\Big[-\frac{1}{2}(\mathbf{x}-\mu_i)^T {\Sigma_i}^{-1}(\mathbf{x}-\mu_i)\Big]. \tag{4.35}$$

By inserting equation (4.35) into equation (4.34), we yield:

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mu_i)^T {\Sigma_i}^{-1}(\mathbf{x}-\mu_i) - \frac{d}{2}\ln(\frac{\pi}{2}) - \frac{1}{2}\ln|\Sigma_i| + \ln(Pr(c_i)) \,. \tag{4.36}$$

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

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mu_i)^T {\Sigma_i}^{-1}(\mathbf{x}-\mu_i) - \frac{1}{2}\ln|\Sigma_i| + \ln(Pr(c_i)) \,. \tag{4.37}$$

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 which are given below:

- $\Sigma_i = \sigma^2 I$**:** All features have different means, but the same covariance matrix for mathematical convenience. By replacing Σ_i with $\sigma^2 I$, we yield the discriminant function as follows.

$$
\begin{aligned}
g_i(\mathbf{x}) &= -\frac{1}{2\sigma^2}(\mathbf{x}-\mu_i)^T(\mathbf{x}-\mu_i) - \frac{1}{2}\ln(\sigma^{2d}) + \ln(Pr(c_i))\,, \\
&= -\frac{1}{2\sigma^2}(\mathbf{x}-\mu_i)^T(\mathbf{x}-\mu_i) + \ln(Pr(c_i)) \quad \text{since } \frac{1}{2}\ln(\sigma^{2d}) \text{ is constant,} \\
&= -\frac{1}{2\sigma^2}\big(\mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mu_i - \mu_i^T\mathbf{x} - \mu_i^T\mu_i\big) + \ln(Pr(c_i))\,, \\
&= -\frac{1}{2\sigma^2}\big(-2\mathbf{x}^T\mu_i - \mu_i^T\mu_i\big) + \ln(Pr(c_i)) \quad \text{since } \mathbf{x}^T\mathbf{x} \text{ is constant and } \mathbf{x}^T\mu_i = \mu_i^T\mathbf{x}\,, \\
&= \frac{\mu_i^T}{\sigma^2}\mathbf{x} + \Big(\ln(Pr(c_i)) - \frac{\mu_i^T\mu_i}{2\sigma^2}\Big). \qquad (4.38)
\end{aligned}
$$

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

- $\Sigma_i = \Sigma$**:** Covariance matrices are all arbitrary, but equal across classes.

$$
\begin{aligned}
g_i(\mathbf{x}) &= -\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma^{-1}(\mathbf{x}-\mu_i) - \frac{1}{2}\ln|\Sigma| + \ln(Pr(c_i))\,, \\
&= -\frac{1}{2}(\mathbf{x}-\mu_i)^T\Sigma^{-1}(\mathbf{x}-\mu_i) + \ln(Pr(c_i)) \quad \text{since } \ln|\Sigma| \text{ is constant,} \\
&= \frac{-1}{2}\big(\mathbf{x}^T\Sigma^{-1}\mathbf{x} - \mu_i\Sigma^{-1}\mathbf{x}\mathbf{x}^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mu_i\big) + \ln(Pr(c_i))\,.
\end{aligned}
$$

Since $\mathbf{x}^T\Sigma^{-1}\mathbf{x}$ is constant and $\mathbf{x}^T\Sigma^{-1}\mathbf{x} = \mu_i\Sigma^{-1}\mathbf{x}$, it follows that:

$$
\begin{aligned}
g_i(\mathbf{x}) &= \frac{-1}{2}\big(-2\mathbf{x}^T\Sigma^{-1}\mu_i + \mu_i^T\Sigma^{-1}\mu_i\big) + \ln(Pr(c_i)) \\
&= \big(\mu_i^T\Sigma^{-1}\big)\mathbf{x} + \Big(\ln(Pr(c_i)) - \frac{1}{2}\mu_i^T\Sigma^{-1}\mu_i\Big)\,. \qquad (4.39)
\end{aligned}
$$

It can be seen that, the final discriminant function in equation (4.39) is a linear function. The illustrations for fixed and varying covariance having data can be found in Figure 4.8.

Penalty Terms

When the number of features is very high, LDA may experience computational difficulties while operating over high-dimensional covariance matrices and then may result in not optimal results. In that case, a simple modification on covariance matrix is made as [46]:

[5]Retrieved from: `https://scikit-learn.org/stable/modules/lda_qda.html` on April 18, 2021.

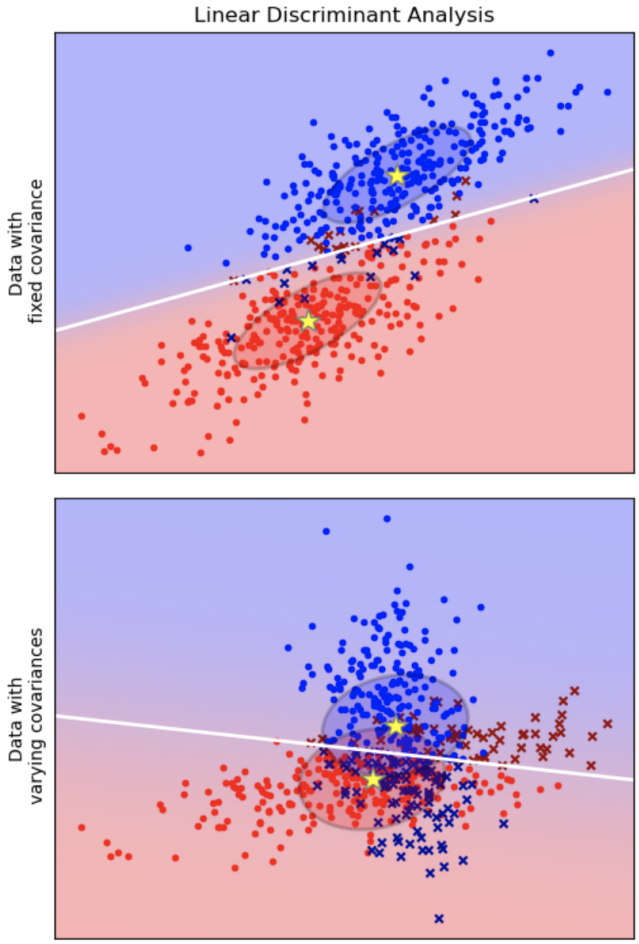


Figure 4.8 : LDA process for fixed and varying covariances.[5]

$$\widehat{\Sigma} = \alpha\Sigma + (1-\alpha)I, \tag{4.40}$$

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:

$$\widehat{\Sigma} = \lambda\Sigma + I \quad \text{, or} \tag{4.41}$$

$$\widehat{\Sigma} = \Sigma + \lambda I\,, \tag{4.42}$$

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

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\mu_i)^T\widehat{\Sigma}_i^{-1}(\mathbf{x}-\mu_i) - \frac{1}{2}\ln|\widehat{\Sigma}_i| + \ln(Pr(c_i))\,. \tag{4.43}$$

Solvers

In this thesis, to solve the LDA functions, we use the solvers presented by `scikit-learn` package [37] in `Python` programming language. Further information about estimators provided by `scikit-learn` package can be found at [47].

5. EXPERIMENTS

All the code developed for the experiments explained in this section can be found on master branch of the GitHub public repository: https://github.com/ozanguldali/modelsWithLASSO. h Python programming language (version of 3.7.6) along with Clang (version 4.0.1) was used to train, test, and classify data in 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 implementation of CNN and ML algorithms, respectively. All experiments were performed on the Google Colaboratory using its GPU with the number of workers as four.

The construction stages of the experiments we aim to carry out in the thesis can be listed as follows:

1. Determine the problem to study,
2. Explore the data source and choose the one containing appropriate samples for the determined problem, which is a chest X-ray image data source consisting of COVID-19 samples and non-COVID-19 samples and including the demographic information of samples for our study,
3. Know and understand the data source, and define classes,
4. Determine the rules to construct the data set for experiments by cleaning the samples from data source, then construct the data set and divide it into train and test sets containing each class,
5. Do CNN experiments on different pre-trained models with different transfer learning strategies and with different optimizers, then save the results and model weights to distinct files,
6. Choose the best results for each model, and use their model weight files to extract the deep features of images in dataset, say X_{cnn},

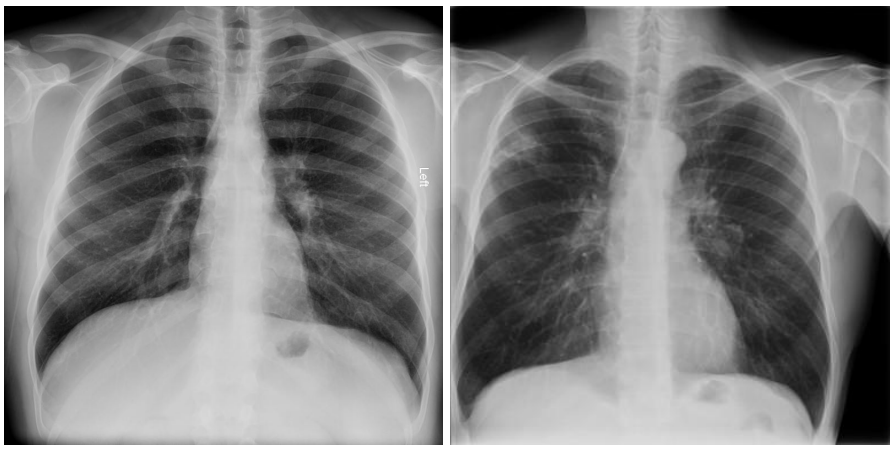
7. Do ML experiments using grid search to reach the optimal hyper-parameters on different models over X_{cnn} feature matrix with two methodologies: 1) apply cross-validation only on train set separated in step (4) to yield the generalized results, and 2) train on train set and test on test set separated in step (4) to yield the results for predetermined test set,
8. Extract the demographic information from dataset for each sample, and construct the demographic information feature matrix, say X_{info},
9. Do ML experiments using grid search to reach the optimal hyper-parameters on different models over X_{info} feature matrix with two methodologies: 1) apply cross-validation only on train set separated in step (4) to yield the generalized results, and 2) train on train set and test on test set separated in step (4) to yield the results for predetermined test set,
10. Combine feature matrices X_{cnn} and X_{info} column-wise to construct the feature matrix for each sample, say X_{all}, i.e. the feature matrix consisting of the deep features of image data and demographic information of each sample,
11. Do ML experiments using grid search to reach the optimal hyper-parameters on different models over X_{all} feature matrix with two methodologies: 1) apply cross-validation only on train set separated in step (4) to yield the generalized results, and 2) train on train set and test on test set separated in step (4) to yield the results for predetermined test set, and
12. Report all results of experiments.

5.1 Data Set

The data was obtained from a public `GitHub` repository which can be reached via the link https://github.com/ieee8023/covid-chestxray-dataset/. On the GitHub repository, the general information about data is in `metedata.csv` file and the image data is available under `images` folder.

To construct the data set, the `metedata.csv` file was parsed and the tabular data were filtered depending on whether it has age and sex information or not, its finding includes the word "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 information, 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". At the end, there were a total of 131 "COVID-19" and 123 "non-COVID-19" labeled data. An example for "COVID-19" and "non-COVID-19" labeled data can be seen in Figure 5.1. Afterwards, with the ratio of 80 to 20, the labeled data were divided into train and test data sets with approximate sizes, 203 and 51, respectively. While the train set now includes 105 "COVID-19" and 98 "non-COVID-19" labeled data, the test data set now includes 26 "COVID-19" and 25 "non-COVID-19".



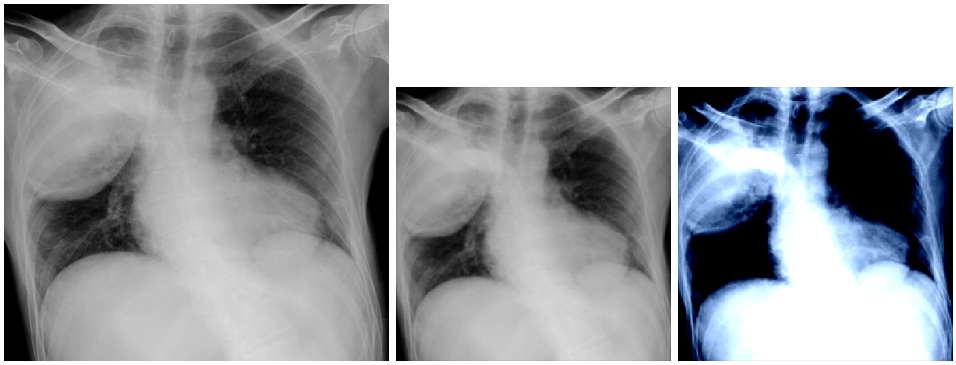
(a) Image labeled as COVID-19. (b) Image labeled as non-COVID-19.

Figure 5.1 : Data set samples.

Image data were appeared in different formats such that .png, .jpg, .jpeg, with 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. The described process can be seen with an example in the Figure 5.2.

5.2 Data Augmentation

Since class sizes are unbalanced, one or more data augmentation techniques are used on train data to balance the class sizes. Furthermore, even for a balanced data set, one can use data augmentation to increase the train data size.



(a) Original (826×768) (b) Resized, square cropped and gray-scaled (c) Normalized

Figure 5.2 : Image transformation steps - Test COVID-19 sample.

Data augmentation techniques on visual data can be classified in two 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.

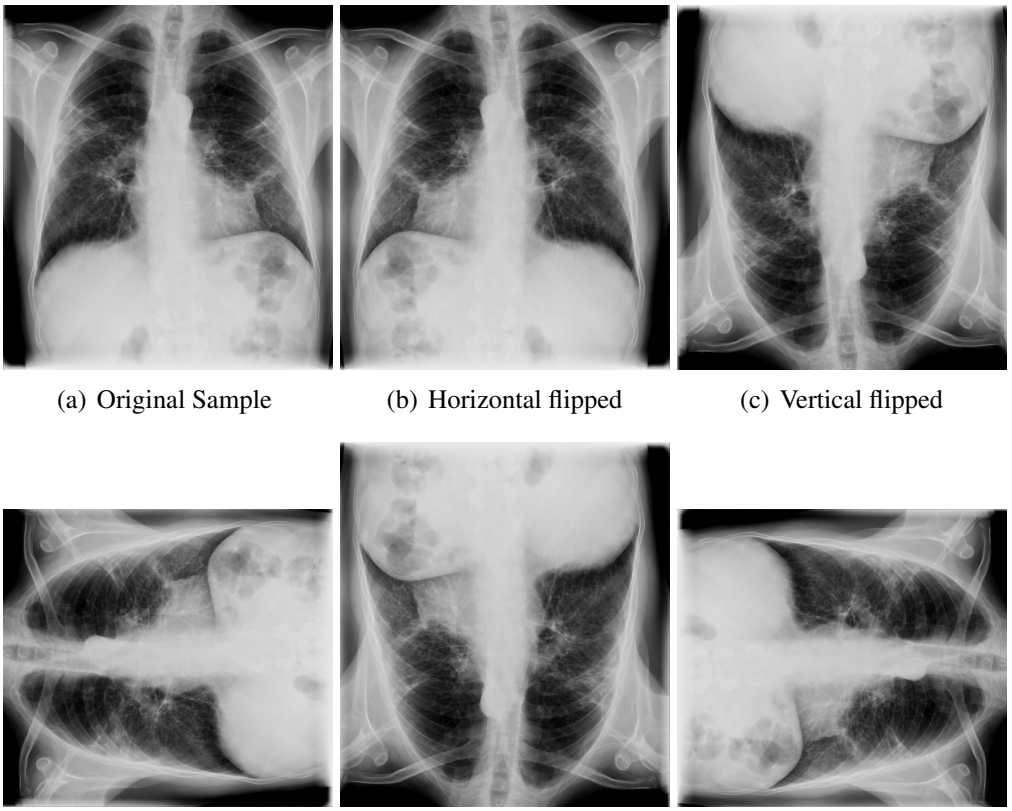
In the thesis, position augmentation, that can be seen via an example in Figure 5.3, including horizontal flip, vertical flip, 90 degrees of rotation, 180 degrees of rotation, and 270 degrees of rotation is used on the whole train data. These augmented data are not physically available, i.e. not saved as files, and only used throughout CNN training process. Data augmentation is not used on ML processes.

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

Table 5.1 : The distribution of class sizes across train (including augmented train set) and test sets.

Class	**Original Set**	**Train Set**		**Test Set**
		Chosen	**Augmented**	
COVID-19	131	105	630	26
Non-COVID-19	123	98	588	25
TOTAL	254	203	1218	51

5.3 Training and Testing with Convolutional Neural Network Models



(a) Original Sample (b) Horizontal flipped (c) Vertical flipped

(d) 90 degrees rotated (e) 180 degrees rotated (f) 270 degrees rotated

Figure 5.3 : An original non-COVID-19 and sample from train data set and position augmentations applied.

All CNN experiments used in this thesis was performed on pre-trained CNN models which are AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG16, and VGG19. In other words, the appropriate model weights were used with different strategies explained in Section 3.5 to feed the models.

Since our data set is small and the data content is different from ImageNet dataset [19], we experimented two different transfer learning strategies such that the first is to train the entire model, and the second is to freeze the first few layers and train others. The frozen layers on the second strategy for each experiment model are:

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

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

Throughout the study, our experience and earlier comparison results, which are not reported here, have shown that training the entire model instead of freezing the associated layers discussed above for the corresponding architectures gives more minimal losses and more accurate classification results. However, it must be remarked that, even the number of frozen layers are not high, since the number of parameters is smaller in freezing approach than the number of parameters in training the entire model, the computation time of each batch is observably shorter in freezing approach.

Train data were set to be shuffled at each epoch and the batch size was experimentally set from 2^4 to 2^6. The number of epochs was set to 50 and the validation period was set as $1/50$. In other words, 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 the 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, testing 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 to 50 is that the models were over-fitted after this number or even earlier than this number sometimes.

Because of 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, SGD momentum, Adam, and AdamW optimizers were

experimented with various learning rates, momentum values (if applicable), and weight decay parameter values (if applicable). Moreover, the learning rate was experimentally scheduled by the reducing rate as by 0.1 for each quarter of total epoch size. However, the scheduler approach drove the models to be over-fitted.

Final hyper-parameters that are accepted as the best for each model differ under each different optimizer. As an example, the best final hyper-parameters for ResNet-50 model under SGD momentum, Adam, and AdamW optimizers can be seen in Table 5.2. Furthermore, in terms of computational time and obtained losses, we encountered the optimal batch size as 2^4 in our experiments. Hence, the number of iterations at each epoch was 77 for augmented train data and 4 for test data which was used both on test and validation processes.

Table 5.2 : Final hyper-parameters for ResNet-50 model.

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 output as the class size, i.e. 2. Furthermore, for all architectures experienced in this thesis, a modification is applied onto the classification section by adding a block of ReLU, drop-out, and fully-connected layers right after the last fully-connected layer to adapt the models for our work.

Let us consider the 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 describes how the process starts with an input image with the size of 227 $\times$ 227, how the blocks of ResNet-50 are aligned along with the output visualizations for each block, and how to obtain a class prediction for this sample at the end. The higher resolution version of images in Figure 5.4, such as the visualizations of convolution blocks, is available at directory https://github.com/ozanguldali/modelsWithLASSO/blob/master/figures.

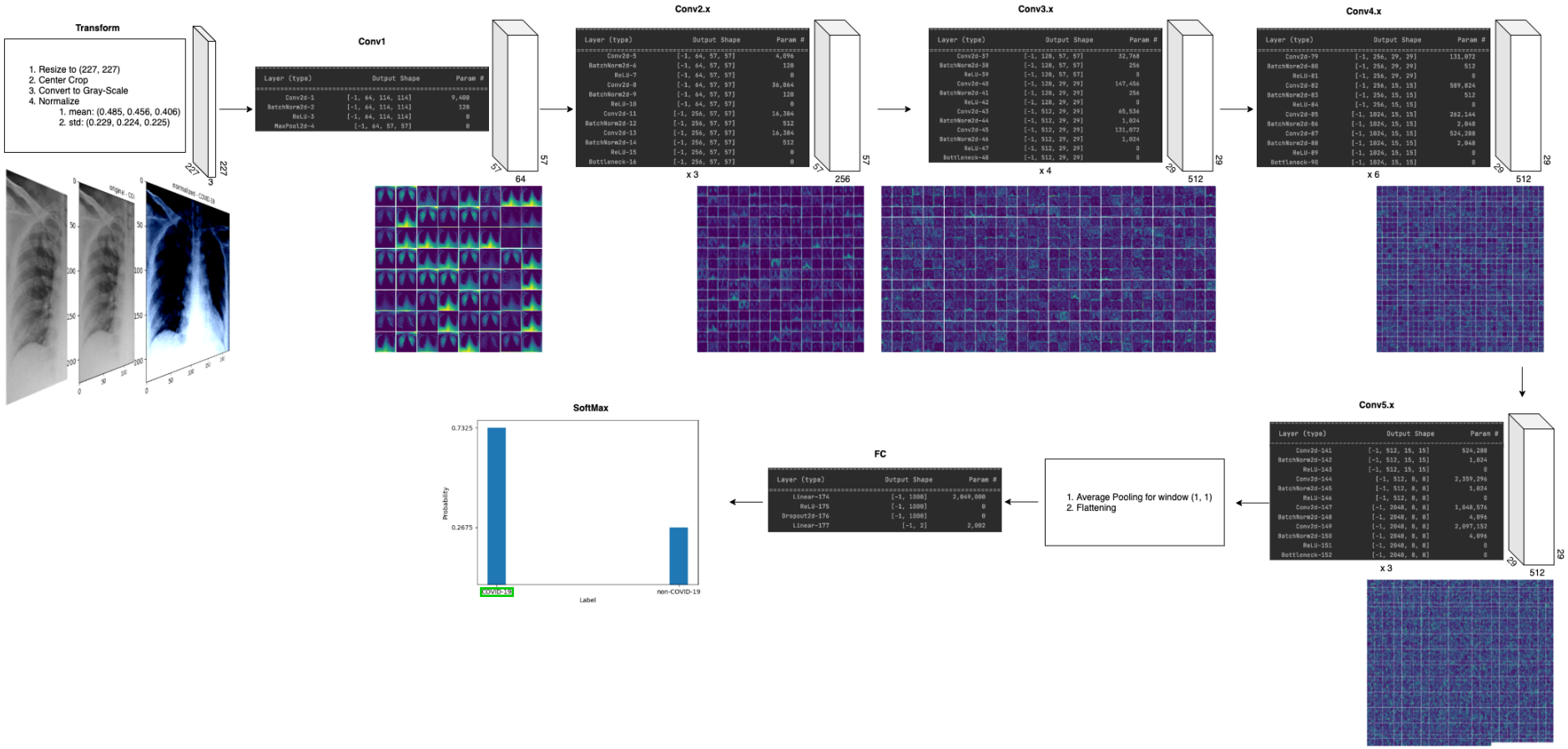


Figure 5.4 : The sample visualization representing one iteration on ResNet-50 architecture and class probability result for COVID-19 labeled data from our dataset. The higher resolution version of image is available at https://github.com/ozanguldali/modelsWithLASSO/blob/master/figures/resnet50_visual.png

5.4 Deep Feature Extraction

After training, validating, and testing the data set on pre-trained CNN models, the deep feature extraction progress could start. Since the model weight files for the best results according to accuracy were stored, these model weights files were used to extract deep features for the corresponding CNN architectures.

Here, the modifications on CNN models stated at previous subsection show its another purpose. The deep features are extracted from the latest fully-connected layer right before the modified classification block consisting of ReLu, drop-out, and the fully-connected layer. Figure 5.5 briefly gives the deep feature extraction layer and the modified classification block for simple AlexNet, ResNet and VGG architectures. As it can be seen from Figure 5.5, the deep features are extracted from:

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

which are all framed in red color. Afterwards, the deep features extracted from CNN models are used to feed machine learning algorithms to improve the classification success.

5.5 Forming Feature Matrices

In this study, we have initially two different types of feature matrices and then we will combine these two to obtain a unique feature matrix. 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 fully-connected layers used. We denote the matrix of deep features by X_{cnn} and the feature size is $d_{cnn} = 1000$.

On the other part, we have demographic information, that's age and gender information for each sample. We denote the matrix of demographic information by X_{info} and the

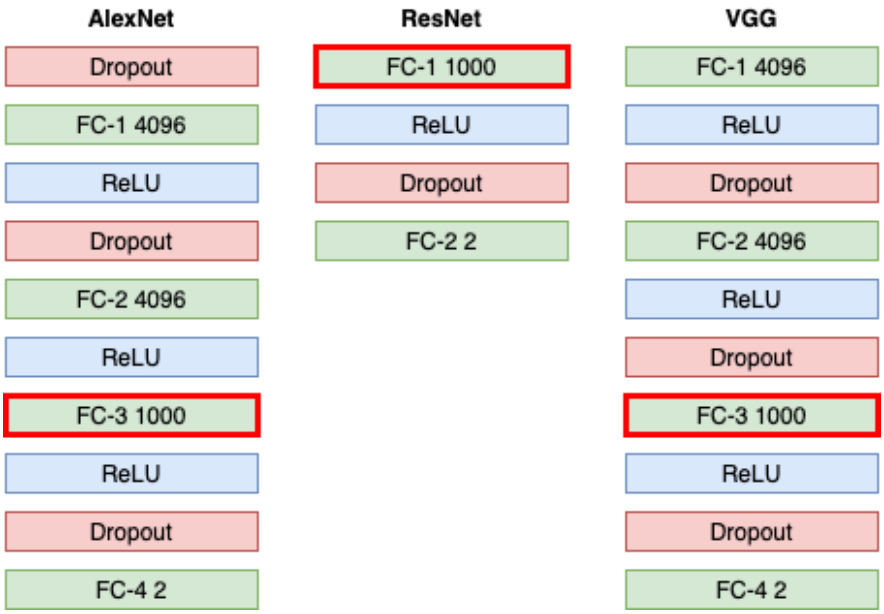


Figure 5.5 : Illustration of modified classification blocks for AlexNet, ResNet, and VGG.

feature size as $d_{info} = 2$. Age information was categorized to 5 groups such as under 18, between 18 and 37, between 38 and 59, between 60 and 79, and over 80 ages.

Hence, as can be seen in the Figure 5.6, the merged feature matrix denoted by X_{all} has the feature size as $d_{all} = 1002$, where the number of rows, N, is 254 in our study.

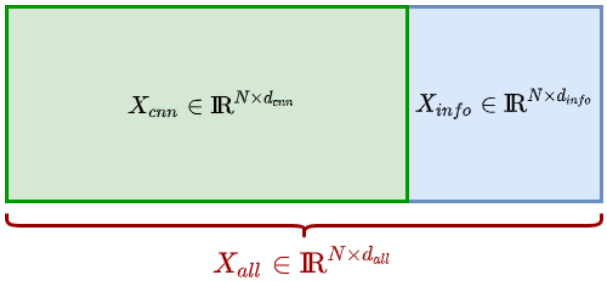


Figure 5.6 : Feature maps where $d_{cnn} = 1000$, $d_{info} = 2$, and $d_{all} = d_{cnn} + d_{info} = 1002$.

5.6 Data Pre-Processing

5.6.1 Data Standardization

Machine learning algorithms 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 mean and

standard deviation. In the experiments, we only centered the data through the formula given below:

$$Z = (X - \mu * I)/\sigma\,, \tag{5.1}$$

where μ is the mean of training samples and σ is set as 1.

5.6.2 Data Normalization

Data normalization scales each individual sample to have a real value in the unit interval [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 for a sample to 1. That is, it scales the S_{L1_k} to 1 in the equation (5.2):

$$S_{L1_k} = \sum_{i=1}^{d} \left| y_k - X_{k,i} \right|, \tag{5.2}$$

  where d is the number of features, X_k is the k^{th} sample ($k = 1, \ldots, N$) 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 $1/S_{L1_k}$; thus, X_k/S_{L1_k} gives the L1 normalized feature values for this sample.

- **L2 Norm:** Scales the sum of the squares of differences of each individual feature for a sample to 1. That is, it scales the S_{L2_k} to 1 in the equation (5.3):

$$S_{L2_k} = \sum_{i=1}^{d} \left(y_k - X_{k,i} \right)^2, \tag{5.3}$$

  where d is the number of features, X_k is the k^{th} sample ($k = 1, \ldots, N$) 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 $\sqrt{1/S_{L2_k}}$; thus, $X_k/\sqrt{S_{L2_k}}$ 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 and these hyper-parameter settings cannot be generalized since they differ for each problem and data set. For that reason, finding the optimal combination of hyper-parameters is a separate mission. When choosing the most optimal hyper-parameter combinations, which characteristics should be looked for totally depends on the problem and the researcher. In this thesis, we made decision according to accuracy values.

There are three common hyper-parameter optimization methods in machine learning such that grid search, random search, and Bayesian optimization.

- **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 $i \in \{1,2,\ldots,m\}$, 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:** Over all possible hyper-parameter combinations, randomly chosen ones are experienced. The number of combinations to be experienced are determined by the researcher. Here, the number of chosen combinations are less then the number of all 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 depends on the useful or irrelevant combination estimation.

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

5.8 Training and Testing with Machine Learning Models

SVM, LR, KNN and LDA algorithms were experienced as ML models.

The partitioning of data set into train and test sets are preserved on machine learning stage. Not that data augmentation is not applied onto the train set of machine learning models.

All three types of feature matrices mentioned above 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 out 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 with LDA, we used `TULIP` package [48] on version 1.0.1 in `R` programming language with the version of 4.0.3 to implement LDA model with lasso penalty. 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 penalty and ridge penalty, were not considered as hyper-parameters since they were experienced and compared separately. All hyper-parameter options tuned for SVM, LR, KNN, LDA can be found in Tables 5.3 -5.6, respectively.

When the optimal hyper-parameter combinations were determined with the aim of generalization, the model weights obtained by the training of a corresponding machine learning algorithm with these hyper-parameters are used on testing stage. The final results were 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.

Table 5.3 : The hyper-parameter tuning for SVM algorithm.

Hyper-Parameter	Lasso Penalty	Ridge Penalty
Kernel Function	linear	(linear, sigmoid, rbf)
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 : The hyper-parameter tuning for LR algorithm.

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 : The hyper-parameter tuning for KNN algorithm.

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 : The hyper-parameter tuning for LDA algorithm.

Hyper-Parameter	No Penalty	Lasso Penalty
Solver Function	(svd, lsqr, eigen)	-
Computing the Weighted Within-Class Covariance (for svd solver)	(Yes, No)	-
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)

6. RESULTS

6.1 Performance Measurement

6.1.1 Confusion Matrix

The confusion matrix is a table that constructed to visualize the performance of a classifier. It includes actual and predicted results for each class on the corresponding task. In our study, the rows specifies the predicted results whereas columns specifies actual 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). This notations can be seen on Table 6.1.

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

Table 6.1 : Confusion matrix.

		Actual	
		P (COVID-19)	N (non-COVID-19)
Predicted	P (COVID-19)	**TP**	FN
	N (non-COVID-19)	FP	**TN**

Various meaningful ratios can be extracted from the confusion matrix given in Table 6.1. In this section, we study the significant ones which were also used in the thesis to measure the performance of our tasks.

Sensitivity

Sensitivity, or in other words, true positive rate (TPR), measures the ratio of positive class samples predicted correctly over all samples predicted as positive as given below:

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}. \tag{6.1}$$

Specificity

Specificity, or in other words, true negative rate (TNR), measures the ratio of negative class samples predicted correctly over all samples predicted as negative as given below:

$$TNR = \frac{TN}{N} = \frac{TN}{FP+TN}. \tag{6.2}$$

Precision

Precision, or in other words, positive predictive value (PPV), measures the ratio of correctly predicted positive class samples over all samples having actual positive label as given below:

$$PPV = \frac{TP}{TP+FP}. \tag{6.3}$$

Accuracy

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

$$ACC = \frac{TP+TN}{P+N} = \frac{TP+TN}{TP+FP+FN+TN}. \tag{6.4}$$

F1 Score

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

$$F_1 = 2 \times \frac{PPV \times TPR}{PPV + TPR} = \frac{2 \times TP}{(2 \times TP) + FP + FN}. \tag{6.5}$$

Area Under the Curve

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:

$$\begin{aligned} &TPR(t) : t \to y(x), \\ &FPR(t) : t \to x, \text{ and} \\ &AUC = \int_0^1 TPR\left(FPR^{-1}(x)\right) dx = \int_{-\infty}^{\infty} TPR(t) \times FPR'(t)\, dt. \end{aligned} \tag{6.6}$$

6.1.2 Analysis of Confusion Matrix

We computed each sensitivity, specificity, precision, accuracy, and F1-score metric in a weighted way to achieve and report the metrics for the corresponding model, not for each class separately. In other words, each metric was computed for each class and then the weighted average was taken as given below:

$$\frac{\sum_{i=1}^{m} (\text{the metric value for class i}) \times (\text{the number of test samples in class i})}{(\text{the total number of test samples})}. \tag{6.7}$$

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:

$$M = \frac{M_{c_1} \times n_{c_1} + M_{c_2} \times n_{c_2}}{n_{c_1} + n_{c_2}}, \tag{6.8}$$

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.

6.2 Convolutional Neural Network Results

Here, we report the results of convolutional neural network experiments explained in Section 5.3. As stated earlier, AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG-16, and VGG-19 architectures, initialized with their pre-trained weights, were trained and three different optimizers such that SGD with momentum, Adam, and AdamW were used to optimize the cross-entropy loss function. The accuracy and loss curves for each architecture with three different optimizers on train and validation processes can be viewed on Figures 6.1-6.6 where the horizontal axis represents 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 including accuracy, sensitivity, specificity, precision and F1 score values, and loss values for the last train epoch and test run can be seen in Table 6.2. We obtained the overall best result from ResNet-50 model after 9^{th} 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 9^{th} train epoch and validation as 0.0223 and 1.3687. On the other hand, VGG19 was the model that achieved its best at third epoch with Adam optimizer. Moreover, AlexNet model stand out with its stable metric values, which were always around 0.90, for all experimented optimizers. In terms of last train loss and test loss, VGG16 with AdamW optimizer had the minimum last train loss as 0.0057 and the minimum test loss as 0.0831. ResNet-18 and ResNet-34 models achieved metric values around 0.90; however, they could not reach to ResNet-50 model whose architecture is more deeper than them. Lastly, it is clear that there can be no generalization about optimizers such that one optimizer is always better than other or vice versa.

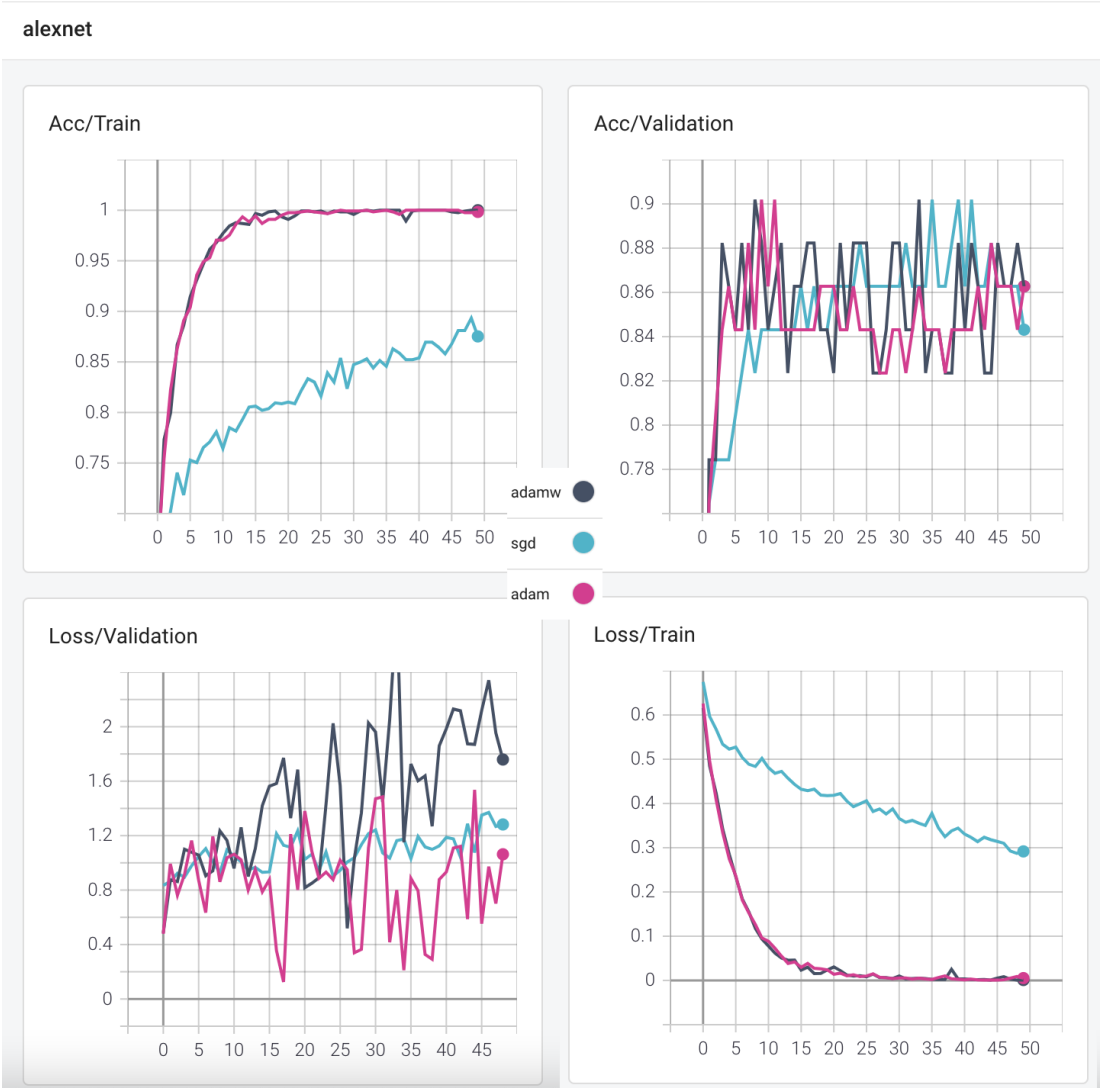


Figure 6.1 : AlexNet model accuracy and loss curves on train and validation processes with the SGD momentum, Adam, and AdamW optimizers.

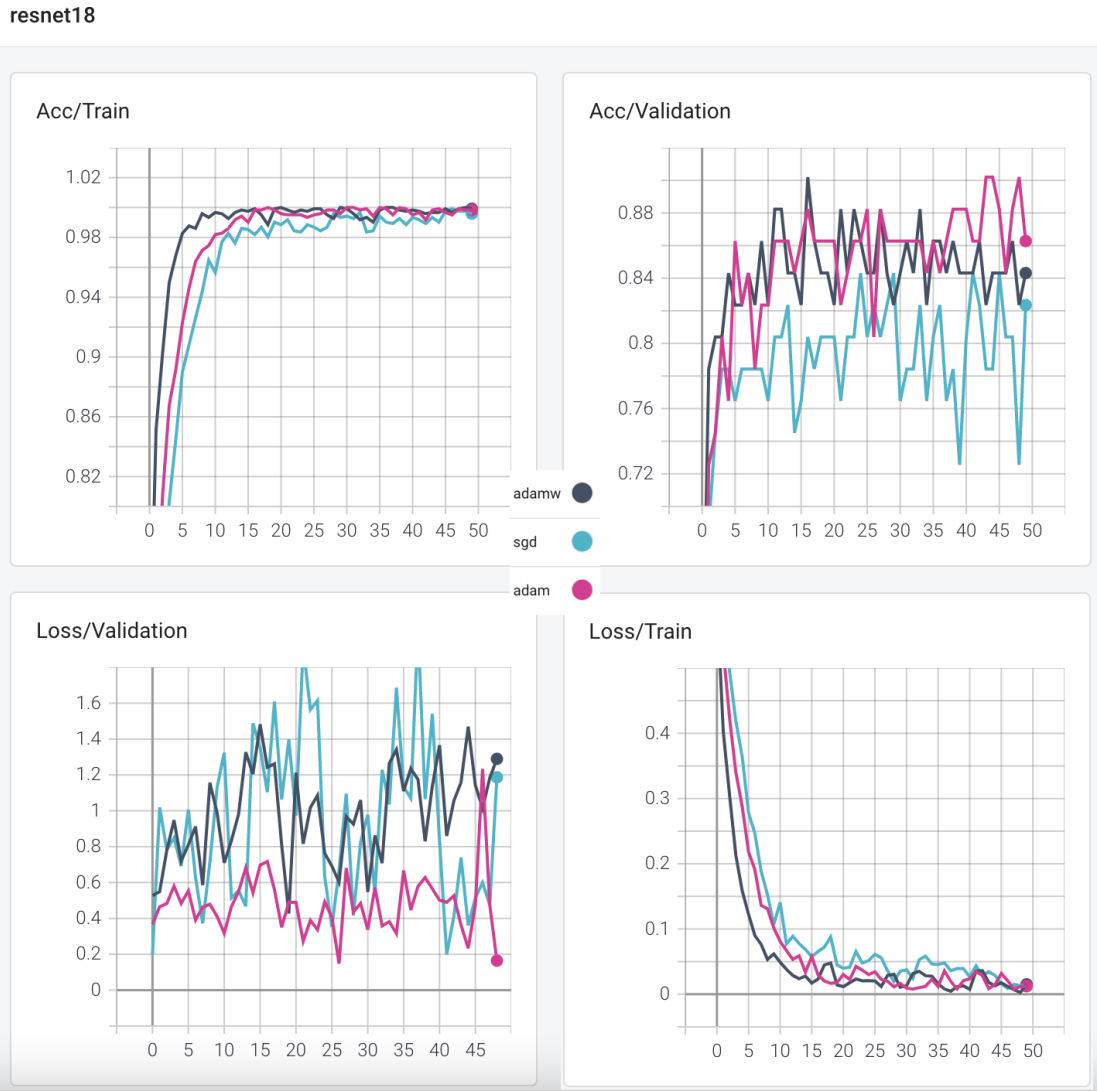


Figure 6.2 : ResNet-18 model accuracy and loss curves on train and validation processes with the SGD momentum, Adam, and AdamW optimizers.

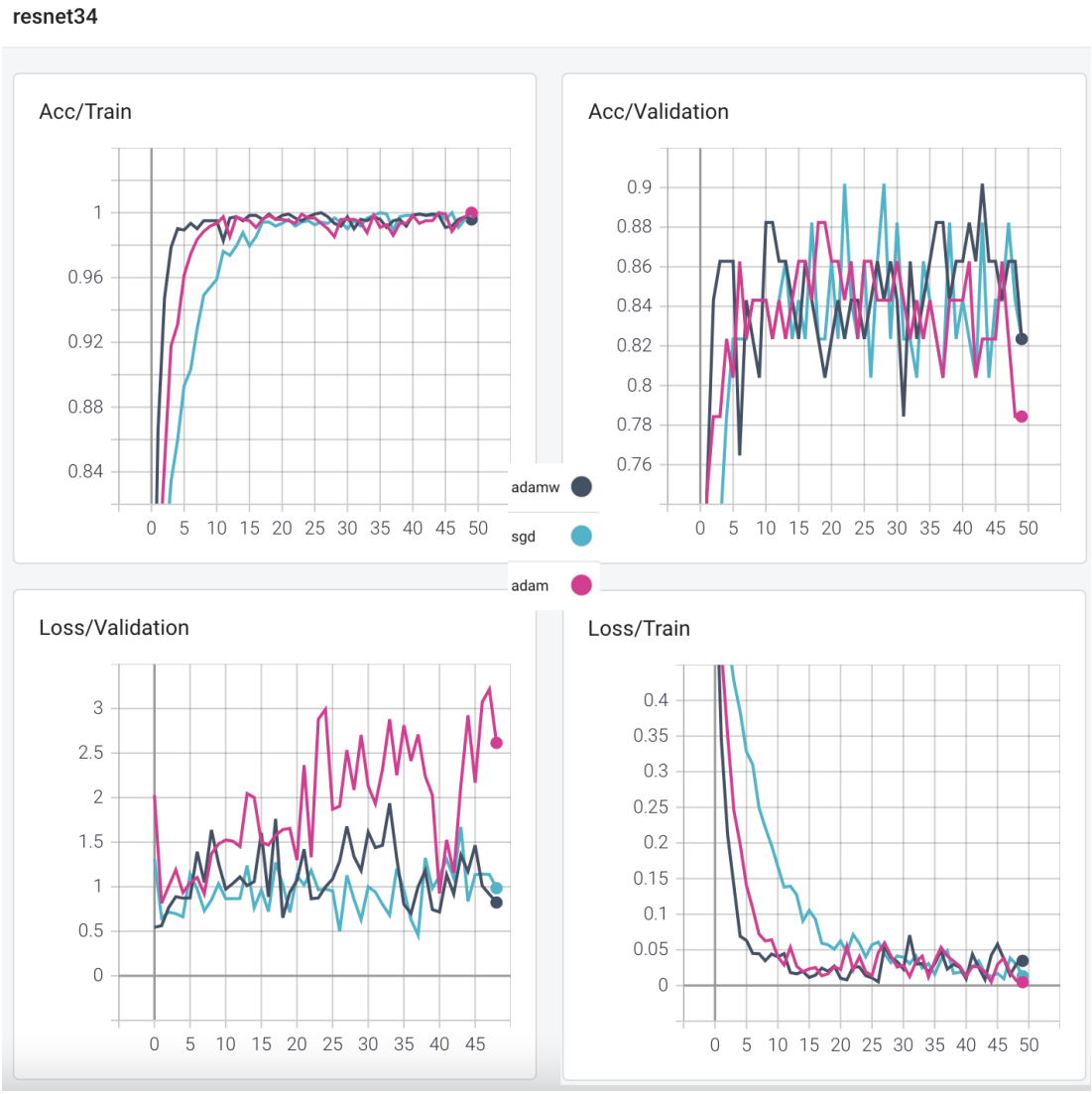


Figure 6.3 : ResNet-34 model accuracy and loss curves on train and validation processes with the SGD momentum, Adam, and AdamW optimizers.

Figure 6.4 : ResNet-50 model accuracy and loss curves on train and validation processes with the SGD momentum, Adam, and AdamW optimizers.

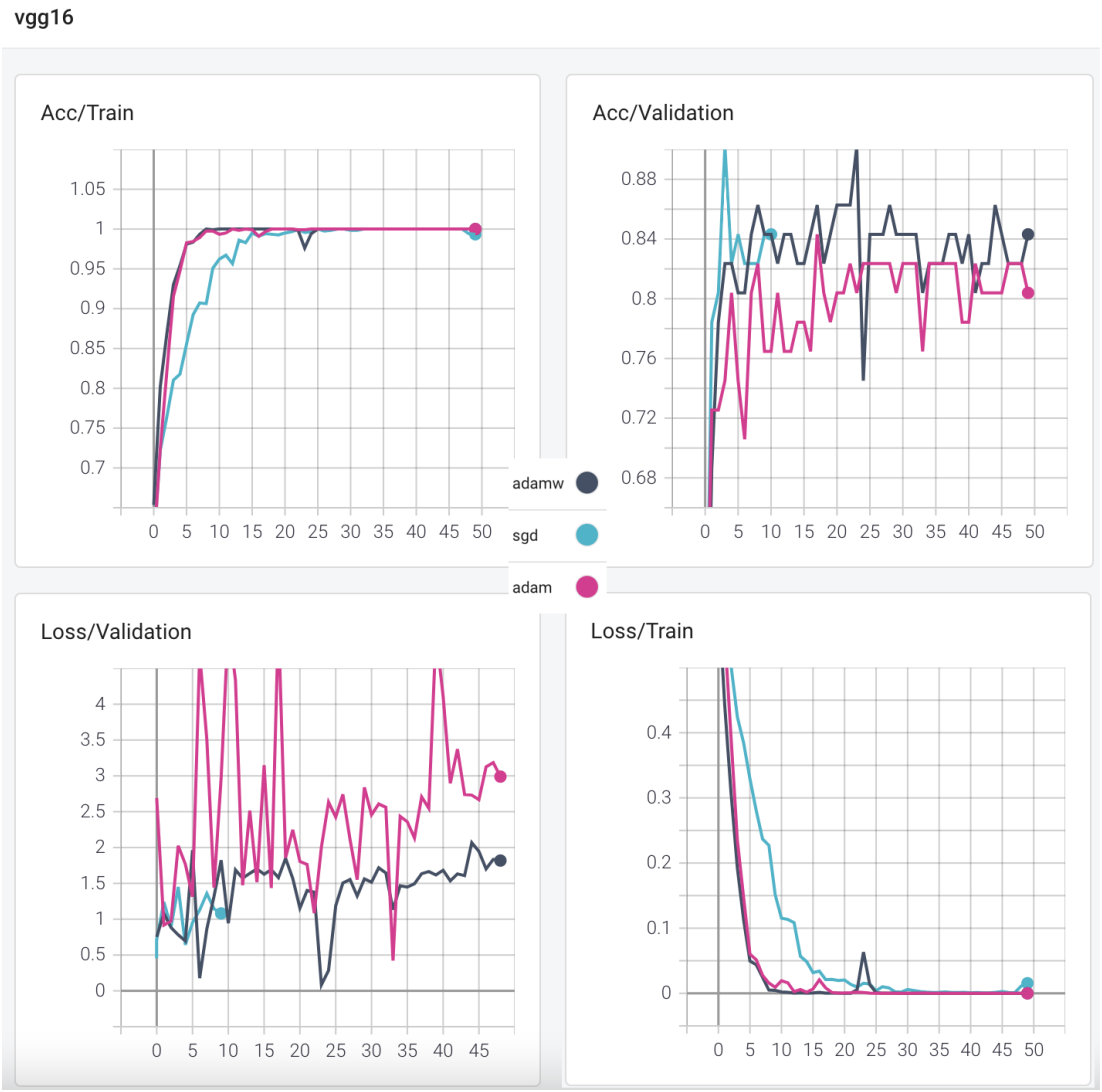


Figure 6.5 : VGG16 model accuracy and loss curves on train and validation processes for the SGD momentum, Adam, and AdamW optimizers.

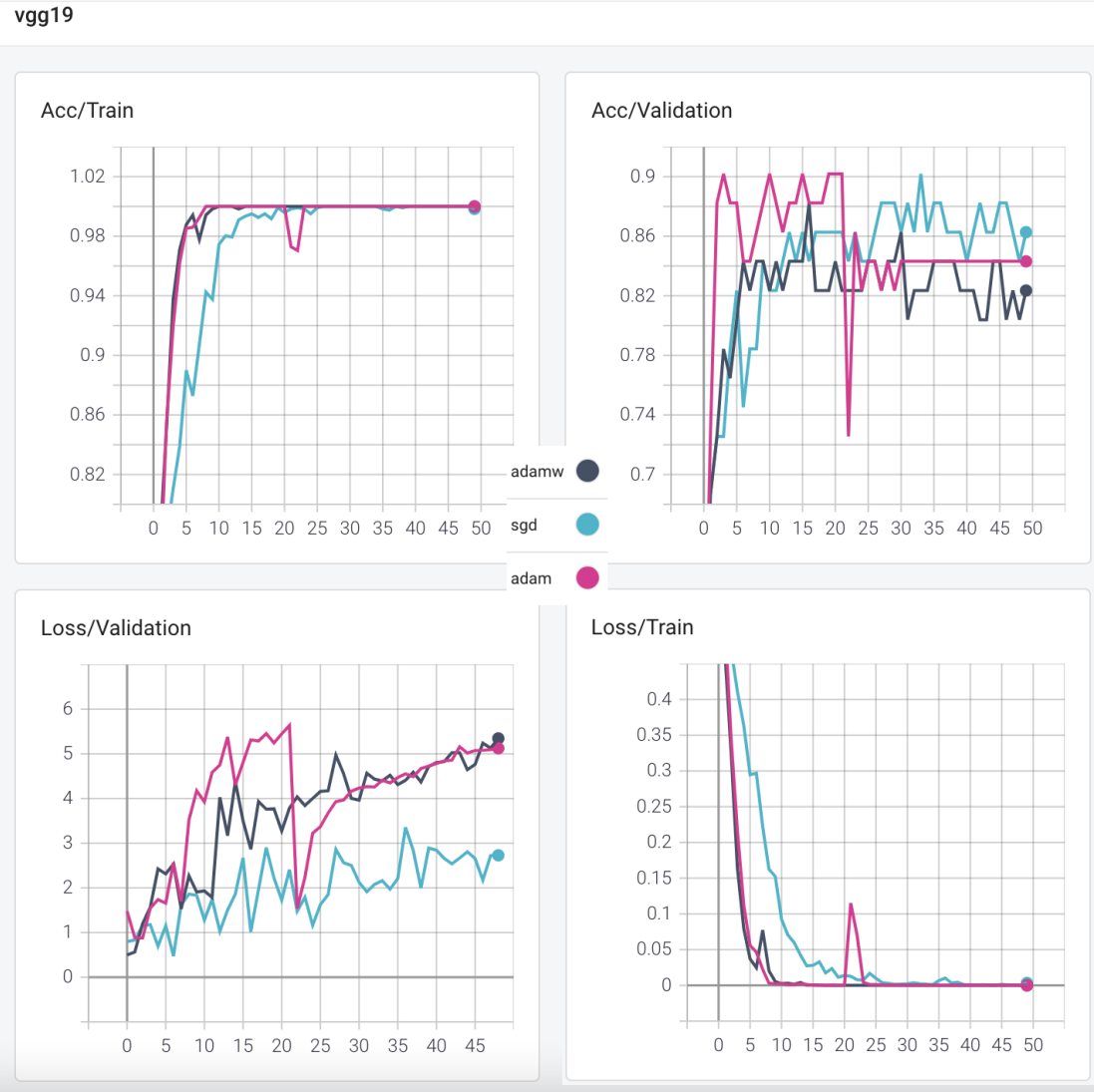


Figure 6.6 : VGG19 model accuracy and loss curves on train and validation processes for the SGD momentum, Adam, and AdamW optimizers.

Table 6.2 : Comparison of results of AlexNet, ResNet-18, ResNet-34, ResNet-50, VGG16, and VGG19.

Models and Optimizers		**Metrics**						**Losses**		**Last Train**	**Duration**
Model	**Optimizer**	**Accuracy (%)**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**AUC Score**	**Last Train**	**Test**	**Epoch**	**(minutes)**
AlexNet	SGD Momentum	90.20	0.9020	0.9026	0.9026	0.9020	0.9019	0.3503	1.0316	35	22
	Adam	90.20	0.9020	0.9011	0.9025	0.9019	0.9023	0.1265	1.0373	8	7
	AdamW	90.20	0.9020	0.9042	0.9078	0.9017	0.9018	0.1549	1.2347	9	13
ResNet-18	SGD Momentum	84.31	0.8431	0.8430	0.8431	0.8431	0.8434	0.0194	0.8252	29	20
	Adam	90.20	0.9020	0.9026	0.9026	0.9020	0.9022	0.0288	0.3631	43	33
	AdamW	90.20	0.9020	0.9042	0.9078	0.9020	0.9019	0.0171	1.2423	16	12
ResNet-34	SGD Momentum	90.20	0.9020	0.9026	0.9026	0.9020	0.9023	0.0460	1.1830	22	16
	Adam	88.24	0.8823	0.8838	0.8849	0.8823	0.8826	0.0136	1.6412	18	10
	AdamW	90.20	0.9020	0.9026	0.9026	0.9020	0.9019	0.0253	1.3553	43	25
ResNet-50	SGD Momentum	88.24	0.8823	0.8823	0.8823	0.8823	0.8821	0.0327	1.2134	29	14
	Adam	**92.16**	**0.9216**	**0.9215**	**0.9216**	**0.9216**	**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 Momentum	90.20	0.9020	0.9042	0.9078	0.9017	0.9021	0.0318	1.4500	15	11
	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 Momentum	90.20	0.9020	0.9026	0.9026	0.9020	0.9020	0.0024	2.1594	29	26
	Adam	90.20	0.9020	0.9026	0.9026	0.9025	0.9019	0.0344	1.5451	3	2
	AdamW	88.24	0.8823	0.8838	0.8849	0.8823	0.8825	0.0006	2.8652	16	15

For each CNN architecture, the best optimizer was detected according to its success on accuracy and loss values, and the confusion matrices for these CNN architecture and optimizer pairs were also reported 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

6.3 Machine Learning Results

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

		Actual	
		P	N
Predicted	P	23	2
	N	3	23

After performing the experiments of CNN models and obtaining the results, the deep feature sets were extracted from the same train and test data. On the other side, the demographic information feature matrices were extracted for the same separation. In this subsection, we report the results for machine learning experiments with SVM with Lasso penalty, SVM with Ridge penalty, LR with Lasso penalty, LR with Ridge penalty, KNN, LDA, and LDA with Lasso penalty on X_{info}, X_{cnn}, and X_{all} matrices, which were defined earlier in Section 5.5, respectively.

6.3.1 Results for X_{info}

In Table 6.9, the results for experimented machine learning algorithms, where the hyper-parameters were tuned with grid search, on X_{info} can be seen. The reported results can be reproduced with the Python system seed as 4. The best result was obtained with KNN algorithm, and the final hyper-parameters of KNN algorithm can be seen in Table 6.10. Furthermore, the associated confusion matrix is in Table 6.11.

Table 6.9 : The results of machine learning experiments on demographic information.

		Accuracy (%)	
Model	**Regularizer**	**Train Data**	**Test Data**
SVM	Lasso	52.94	52.94
	Ridge	52.94	52.94
LR	-	52.94	52.94
	Lasso	52.94	52.94
	Ridge	52.94	52.94
KNN	**-**	**56.86**	**56.86**
LDA	-	52.94	52.94
	Lasso	52.94	52.94

6.3.2 Results for X_{cnn}

In Table 6.12, the results for experimented machine learning algorithms, where the hyper-parameters were tuned with grid search, on X_{cnn} feature matrices obtained from

Table 6.10 : The tuned hyper-parameters of KNN algorithm on X_{info}.

Hyper-Parameter	Train Data	Test Data
Number of Neighbors	17	17
Type of Majority Voting	uniform	uniform
Neighborhood Finder	brute force	brute force
Metric Function	euclidean	euclidean

Table 6.11 : Confusion matrix for KNN experiment on demographic information.

		Actual			
		Train Data		Test Data	
		P	N	P	N
Predicted	P	5	1	5	1
	N	21	24	21	24

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 can be seen. The reported results can be reproduced with the Python system seed as 4.

Since all final results of models were yielded in both train data by using cross-validation, and test data in a directly testing method, all tables referenced here includes the results both for train data and test data. The reason why we experimented in both ways is to see the generalized performance of models, and the performance of models specific to our test data. Further information can be found in Section 5.8 and Section 7.

It can be seen in Table 6.12 that we obtained the highest metrics for machine learning algorithms on the feature matrix X_{cnn} extracted using the weights of ResNet-50 model with Adam optimizer. This makes sense as we obtained the higher result from ResNet-50 model with Adam optimizer in the CNN experiments.

Since the best performance was reached by ResNet-50 model together with Adam optimizer, here we only reported the outputs for this combination. The final hyper-parameters of machine learning algorithms on $X_{cnn\_ResNet50}$ feature matrix constructed by the help of ResNet-50 model Adam optimizer can be seen in Tables 6.13-6.20. These values can be used for reproducibility of our work. Furthermore, the confusion matrices and all other metrics derived from these confusion matrices are given in Tables 6.21-6.28. It can be seen from Tables 6.21-6.28 that the results

obtained on test data is clearly higher than the results obtained on train data. On the other hand, we cannot state the best one; however, the worst results were obtained by LDA algorithm for both test and train data performances.

Table 6.12 : The results of machine learning experiments on X_{cnn}.

Model	Regularizer	Accuracy (%)											
		Train Data						Test Data					
		AlexNet (Adam)	ResNet-18 (AdamW)	ResNet-34 (AdamW)	ResNet-50 (Adam)	VGG16 (AdamW)	VGG19 (SGD Momentum)	AlexNet (Adam)	ResNet-18 (AdamW)	ResNet-34 (AdamW)	ResNet-50 (Adam)	VGG16 (AdamW)	VGG19 (SGD Momentum)
SVM	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
	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
LR	-	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	84.31	86.27	84.31	86.27	86.27	84.31	88.24	86.27	90.20	92.16	86.27	86.27
	Ridge	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 : The tuned hyper-parameters of SVM algorithm with Lasso penalty on $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Regularization Parameter	5e-3	5e-2

Table 6.14 : The tuned hyper-parameters of SVM algorithm with Ridge penalty on $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Kernel Function	linear	linear
Regularization Parameter	5e-5	5e-5

Table 6.15 : The tuned hyper-parameters of LR algorithm on $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	newton-cg	newton-cg

Table 6.16 : The tuned hyper-parameters of LR algorithm wit Lasso penalty for $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	liblinear	liblinear
Regularization Parameter	2e-2	1.0

Table 6.17 : The tuned hyper-parameters of LR algorithm with Ridge penalty on $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	liblinear	liblinear
Regularization Parameter	5e-5	5e-5

Table 6.18 : The tuned hyper-parameters of KNN algorithm on $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	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 : The tuned hyper-parameters of LDA algorithm on $X_{cnn\_ResNet50}$.

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

Table 6.20 : The tuned hyper-parameters of LDA algorithm with Lasso penalty for $X_{cnn\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Regularization Parameter	5e-5	5e-3

Table 6.21 : The confusion matrices and results obtained by SVM algorithm with Lasso penalty for $X_{cnn\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	20	1							23	1						
	N	6	24							3	24						

Table 6.22 : The confusion matrices and results obtained by SVM algorithm with Ridge penalty on $X_{cnn\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.23 : The confusion matrices and results obtained by LR algorithm for $X_{cnn\_ResNet50}$.

		Train Data							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	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 with Lasso penalty on $X_{cnn\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	20	1							23	1						
	N	6	24							3	24						

Table 6.25 : The confusion matrices and results obtained by LR algorithm with Ridge penalty on $X_{cnn\_ResNet50}$.

		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	0.9215	0.9223	P	N	92.16	0.9216	0.9230	0.9243	0.9215	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.26 : The confusion matrices and results obtained by KNN algorithm on $X_{cnn\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.27 : The confusion matrices and results obtained by LDA algorithm on $X_{cnn\_ResNet50}$.

		Train Data							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	0.8454
Predicted	P	19	1							19	1						
	N	7	24							7	24						

Table 6.28 : The confusion matrices and results obtained by LDA algorithm with Lasso penalty on $X_{cnn\_ResNet50}$.

		Train Data							Test Data						
		Actual		**Accuracy (%)**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**AUC Score**	**Actual**		**Accuracy (%)**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**AUC Score**
		P	N	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 results for experimented machine learning algorithms, where the hyper-parameters were tuned with grid search, on different X_{all} feature matrices obtained from 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 can be seen. The reported results can be reproduced with Python system seed as 4.

Since all final results of models were yielded in both train data by using cross-validation, and test data in a directly testing method, all tables referenced here includes the results both for train data and test data. The reason why we experimented in both ways is to see the generalized performance of models, and the performance of models specific to our test data. Further information can be found in Section 5.8 and Section 7.

It can be seen in Table 6.29 that we obtained the highest metrics for machine learning algorithms on the feature matrix X_{all} extracted by using the weights of ResNet-50 model with Adam optimizer, as we obtained the higher result by ResNet-50 model with Adam optimizer in the CNN experiments.

Since the best performance was reached by ResNet-50 model together with Adam optimizer, here we only reported the outputs for this combination. The final hyper-parameters of machine learning algorithms on $X_{all\_ResNet50}$ feature matrix constructed by the help of ResNet-50 model Adam optimizer can be seen in Tables 6.30-6.37. These values can be used for reproducibility of our work. Furthermore, the confusion matrices and all other metrics derived from these confusion matrices are on Tables 6.38-6.45. It can be seen from these reports embedded in Tables 6.38-6.45 that the results obtained on test data is clearly higher than the results obtained on train data. On the other hand, we cannot state the best one; however, the worst results were obtained by LDA algorithm both for test and train data performances.

We can also briefly summarize the results of our experiments on our data set as follows:

- demographic information did not have a general improvement by combining with deep features; yet, only the performance of logistic regression algorithm with Lasso penalty was boosted by that way,
- penalty approach only increased the success of linear discriminant analysis algorithm, and
- generalized optimal hyper-parameters may not be the best combination for our initially determined test data.

Table 6.29 : The results of machine learning experiments on X_{all}.

Model	Regularizer	Accuracy (%)											
		Train Data						Test Data					
		AlexNet (Adam)	ResNet-18 (AdamW)	ResNet-34 (AdamW)	ResNet-50 (Adam)	VGG16 (AdamW)	VGG19 (SGD Momentum)	AlexNet (Adam)	ResNet-18 (AdamW)	ResNet-34 (AdamW)	ResNet-50 (Adam)	VGG16 (AdamW)	VGG19 (SGD Momentum)
SVM	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
	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
LR	-	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	84.31	86.27	84.31	88.24	86.27	86.27	88.24	86.27	90.20	92.16	86.27	88.24
	Ridge	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	-	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	-	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 6.30 : The tuned hyper-parameters of SVM algorithm with Lasso penalty on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Regularization Parameter	5e-3	5e-2

Table 6.31 : The tuned hyper-parameters of SVM algorithm with Ridge penalty on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Kernel Function	linear	linear
Regularization Parameter	5e-5	5e-5

Table 6.32 : The tuned hyper-parameters of LR algorithm on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	newton-cg	newton-cg

Table 6.33 : The tuned hyper-parameters of LR algorithm with Lasso on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	liblinear	liblinear
Regularization Parameter	2e-2	2.0

Table 6.34 : The tuned hyper-parameters of LR algorithm with Ridge penalty on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Solver Function	liblinear	liblinear
Regularization Parameter	5e-5	5e-5

Table 6.35 : The tuned hyper-parameters of KNN algorithm on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	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 : The tuned hyper-parameters of LDA algorithm on $X_{all\_ResNet50}$.

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

Table 6.37 : The tuned hyper-parameters of LDA algorithm with Lasso penalty on $X_{all\_ResNet50}$.

Hyper-Parameter	Train Data	Test Data
Regularization Parameter	5e-5	5e-3

Table 6.38 : The confusion matrices and results obtained by SVM algorithm with Lasso penalty on $X_{all\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	20	1							23	1						
	N	6	24							3	24						

Table 6.39 : The confusion matrices and results obtained by SVM algorithm with Ridge penalty on $X_{all\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.40 : The confusion matrices and results obtained by LR algorithm on $X_{all\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.41 : The confusion matrices and results obtained by LR algorithm with Lasso penalty on $X_{all\_ResNet50}$.

		Train Data							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	0.9215	0.9223
Predicted	P	21	1							23	1						
	N	5	24							3	24						

Table 6.42 : The confusion matrices and results obtained by LR algorithm with Ridge penalty on $X_{all\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.43 : The confusion matrices and results obtained by KNN algorithm on $X_{all\_ResNet50}$.

		Train Data							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	0.9223
Predicted	P	23	1							23	1						
	N	3	24							3	24						

Table 6.44 : The confusion matrices and results obtained by LDA algorithm on $X_{all\_ResNet50}$.

		Train Data							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	0.8454
Predicted	P	19	1							19	1						
	N	7	24							7	24						

Table 6.45 : The confusion matrices and results obtained by LDA algorithm with Lasso penalty for $X_{all\_ResNet50}$.

		Train Data							Test Data						
		Actual		**Accuracy (%)**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**AUC Score**	**Actual**		**Accuracy (%)**	**Sensitivity**	**Specificity**	**Precision**	**F1 Score**	**AUC Score**
		P	N	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						

7. CONCLUSION AND RECOMMENDATION

The contribution of this thesis was 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 for classification. To do research on this topic, a global health problem COVID-19 disease and chest X-Rays images data along with demographic information were used.

Although this study mainly focused on chest X-Rays and image data, one of the aims 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 could not be seen. In general, we did not have an improvement by combining the demographic information with deep features. However, the performance of logistic regression algorithm with Lasso penalty was boosted by that way, as the only one. 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.

During the machine learning experiments, the reason why we applied 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 these 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.

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 were 0.0223 and 1.3687, respectively.
- **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.

We recommend that the chest X-ray images to be used be reviewed and, if necessary, eliminated by experts. In studies similar to ours, getting support from expert radiologists ensures that better quality images are obtained. In this way, more successful models can be obtained.

After constructing our data set, which has two classes such that COVID-19 and non-COVID-19, including samples with demographic information, the deep learning experiments were done to obtain the features of images. Then, this features called as deep features are used to feed machine learning algorithms to complete the experiments of our study.

7.1 Future Work

Although this thesis focuses on chest X-Rays images, any other image classification problem can be studied as a future work. If the study data set consisting of images also includes non-image information, this thesis can be used as a guide for corresponding classification problem.

This study can also be applied to multi-class classification problems on various respiratory diseases diagnosed by chest computed tomography images of patients where in addition to the demographic information such as age, gender, and tobacco product use, clinical information such as doctors report, associated genetic diseases, and respiratory test information are available as well. We also recommend getting advise from radiologists and chest disease specialists on the data, for the given study example.

This study can be improved by using large data in size and can be extended by using chest computer tomography along with more various information obtained from the patients. On the other hand, different optimizers such as Adagrad [49] and Padam [50] for convolutional neural networks and elastic net regularization method [51] for machine learning algorithms can be used to extend this study. Moreover, ensemble learning [52] for convolutional neural network process and machine learning process can be separately 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, *New Brunswick, N.J. : Rutgers University Press*.

[2] **Loshchilov, I. and Hutter, F.** (2017). Fixing Weight Decay Regularization in Adam, *ArXiv*, *abs/1711.05101*.

[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 Pptimization, *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.** (2016). Deep Residual Learning for Image Recognition, *arXiv preprint arXiv:1512.03385*.

[7] **Nielsen, F.**, (2016). Parallel Linear Algebra, Springer International Publishing, 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] 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-mar

ch-2020, accessed: 2021-04.

[11] Coronavirus (COVID-19), https://news.google.com/covid19/map?hl=tr&gl=TR&ceid=TR%3Atr&state=1, accessed: 2021-07-10.

[12] Coronavirus disease (COVID-19), https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19, accessed: 2021-04.

[13] **Hu, D.**, **Lou, X.**, **Meng, N.**, **Li, Z.**, **Teng, Y.**, **Zou, Y. and Wang, F.** (2021). Influence of Age and Gender on the Epidemic of COVID-19, *Wiener klinische Wochenschrift*, *133*(7), 321–330, https://doi.org/10.1007/s00508-021-01816-z.

[14] **Clinic, C.**, (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, accessed: 2021-04.

[15] **Villines, Z. and Martinez, K.**, (2020), https://www.medicalnewstoday.com/articles/pneumonia-and-covid-19, accessed: 2021-04.

[16] **Wang, X.** (2016). Deep Learning in Object Recognition, Detection, and Segmentation, *Found. Trends Signal Process.*, *8*, 217–382.

[17] **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.

[18] **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 biomedicale : IRBM = Biomedical engineering and research*, 10.1016/j.irbm.2020.05.003, https://pubmed.ncbi.nlm.nih.gov/32837678, 32837678[pmid].

[19] **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.

[20] **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.

[21] **Redmon, J. and Farhadi, A.** (2017). YOLO9000: Better, Faster, Stronger, *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517–6525.

[22] **Oh, Y.**, **Park, S. and Ye, J.C.** (2020). Deep Learning COVID-19 Features on CXR Using Limited Training Data Sets, *IEEE Transactions on Medical*

Imaging, *39*(8), 2688–2700.

[23] **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.

[24] **Al-falluji, R.A.**, **Katheeth, Z.D. and Alathari, B.** (2021). Automatic Detection of COVID-19 Using Chest X-ray Images and Modified ResNet18-based Convolution Neural Networks, *Computers, Materials & Continua*, *66*, 1301–1313.

[25] **Zhu, D.**, **Yao, H.**, **Jiang, B. and Peng, Y.** (2018). Negative Log Likelihood Ratio Loss for Deep Neural Network Classification, *ArXiv*, *abs/1804.10690*.

[26] **Brébisson, A.D. and Vincent, P.** (2016). An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family, *Computing Research Repository*, *abs/1511.05042*.

[27] **Ruder, S.** (2016). An Overview of Gradient Descent Optimization Algorithms, *ArXiv*, *abs/1609.04747*.

[28] **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.

[29] **Graetz, F.M. and Matters, W.A.**, (2018), Why AdamW Matters, https://towardsdatascience.com/why-adamw-matters-736223f31b5d, accessed: 2021-04.

[30] **Simonyan, K. and Zisserman, A.** (2015). Very Deep Convolutional Networks for Large-scale Image Recognition, *ArXiv*, 1409.1556.

[31] **Chang, M.** (2020). *Artificial Intelligence for Drug Development, Precision Medicine, and Healthcare*, CRC Press.

[32] **Cortes, C. and Vapnik, V.** (1995). Support-vector Networks, *Machine Learning*, *20*(3), 273–297, https://doi.org/10.1023/A:1022627411411.

[33] **Ben-Hur, A.**, **Horn, D.**, **Siegelmann, H. and Vapnik, V.** (2001). Support Vector Clustering, *Journal of Machine Learning Research*, *2*, 125–137.

[34] **Boser, B.E.**, **Guyon, I.M. and Vapnik, V.N.** (1992). A Training Algorithm for Optimal Margin Classifiers, *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, Association for Computing Machinery, New York, NY, USA, p.144–152, https://doi.org/10.1145/130385.130401.

[35] **Christmann, A. and Steinwart, I.** (2008). *Support Vector Machines*, Support Vector Machines: Information Science and Statistics. ISBN 978-0-387-77241-7. Springer Science+Business Media, LLC, 2008.

[36] **Koshiba, Y. and Abe, S.** (2003). Comparison of L1 and L2 Support Vector Machines, *Proceedings of the International Joint Conference on Neural Networks, 2003.*, *3*, 2054–2059 vol.3.

[37] **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.

[38] **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.

[39] **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 Transactions on Mathematical Software*, *23*(4), 550–560, https://doi.org/10.1145/279232.279236.

[40] **Fan, R.E.**, **Chang, K.W.**, **Hsieh, C.J.**, **Wang, X.R. and Lin, C.J.** (2008). LIBLINEAR: A Library for Large Linear Classification, *Journal of Machine Learning Research*, *9*, 1871–1874.

[41] **Schmidt, M.W.**, **Roux, N.L. and Bach, F.** (2017). Minimizing Finite Sums with the Stochastic Average Gradient, *Mathematical Programming*, *162*, 83–112.

[42] **Defazio, A.**, **Bach, F. and Lacoste-Julien, S.** (2014). SAGA: A Fast Incremental Gradient Method With Support for Non-Strongly Convex Composite Objectives, *Conference on Neural Information Processing Systems (NIPS)*.

[43] **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.

[44] **Bentley, J.L.** (1975). Multidimensional Binary Search Trees Used for Associative Searching, *Communications of ACM*, *18*(9), 509–517, https://doi.org/10.1145/361002.361007.

[45] **Omohundro, S.M.** (1989). Five Balltree Construction Algorithms, **Technical Report**, International Computer Science Institute.

[46] **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.

[47] **scikit-learn developers**, Linear and Quadratic Discriminant Analysis, https://scikit-learn.org/stable/modules/lda_qda.html#estimation-algorithms, accessed: 2021-04.

[48] **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.

[49] **Duchi, J.**, **Hazan, E. and Singer, Y.** (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *Journal of Machine*

Learning Research, *12*(null), 2121–2159.

[50] **Chen, J. and Gu, Q.** (2018). Closing the Generalization Gap of Adaptive Gradient Methods in Training Deep Neural Networks, *Computing Research Repository*, *abs/1806.06763*, http://arxiv.org/abs/1806.06763, 1806.06763.

[51] **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.

[52] **Darıcı, M.B.**, (2020). Göğüs Kafesi Röntgen Görüntülerinde Derin Öğrenme Metoduyla Zatürre Hastalığının Tanısı, Master's thesis, Istanbul Technical University, Istanbul, Turkey.

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 GÜLDALİ

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

- data set may be a must for some run-configurations
  - Link to data set: 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 Leaning:

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.