BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
MASTER OF SCIENCE IN ELECTRICAL AND ELECTRONICS ENGINEERING

HAND STRUCTURE DETECTION AND SUITABLE NAIL TYPE RECOMMENDATION SYSTEM

BY

YAĞMUR YARIMBIYIK

MASTER OF SCIENCE THESIS

ANKARA - 2024

BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING
DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
MASTER OF SCIENCE IN ELECTRICAL AND ELECTRONICS ENGINEERING

HAND STRUCTURE DETECTION AND SUITABLE NAIL TYPE RECOMMENDATION SYSTEM

BY

YAĞMUR YARIMBIYIK

MASTER OF SCIENCE THESIS

ADVISOR

PROF. DR. HAMİT ERDEM

ANKARA - 2024

BAŞKENT UNIVERSITY
INSTITUTE OF SCIENCE AND ENGINEERING

This study, which was prepared by Yağmur YARIMBIYIK, for the program of Electrical and Electronics Engineering Master's with Thesis, has been approved in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in Electrical and Electronics Engineering by the following committee.

Date of Thesis Defense: 12 / 08 / 2024

Thesis Title: Hand Structure Detection and Suitable Nail Type Recommendation System

Examining Committee Members	**Signature**
Assoc. Prof. Dr. Derya YILMAZ, Gazi University	……
Prof. Dr. Hamit ERDEM, Başkent University	……
Assoc. Prof. Dr. Selda GÜNEY, Başkent University	……

APPROVAL

Prof. Dr. Dilek ÇÖKELİLER SERDAROĞLU
Director, Institute of Science and Engineering
Date: … / … / ……..

…….

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU

Tarih: … / … / 20…

Öğrencinin Adı, Soyadı	:	Yağmur YARIMBIYIK
Öğrencinin Numarası	:	22110047
Anabilim Dalı	:	Elektrik-Elektronik Mühendisliği Ana Bilim Dalı
Programı	:	Elektrik-Elektronik Mühendisliği Tezli Yüksek Lisans Programı
Danışmanın Adı, Soyadı	:	Prof. Dr. Hamit ERDEM
Tez Başlığı	:	Hand Structure Detection and Suitable Nail Type Recommendation System

Yukarıda başlığı belirtilen Yüksek Lisans/Doktora tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 113 sayfalık kısmına ilişkin, 21/08/2024 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %12'dir. Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası:…….

ONAY
Öğrenci Danışmanı:
Prof. Dr. Hamit ERDEM

……..
Tarih: … / … / 20…

To all those who feel lost in life. Be patient and don't give in. You will eventually find your way.

Yağmur YARIMBIYIK

Ankara - 2024

ACKNOWLEDGMENTS

I would like to express my respect and gratitude to my former graduation project and current thesis advisor Prof. Dr. Hamit ERDEM, who always spreads joy around with his energy and always has a positive outlook on life. During my thesis studies, he always supported me, lifted my morale when I was down and gave me the self-confidence to continue, which I am very lucky to have.

I would like to thank my family who has been with me since the day I was born; my mother Artemiz YARIMBIYIK, my father Bülent YARIMBIYIK and my brother Fırat YARIMBIYIK. Thank you for protecting and taking care of me, never withholding your love, helping me to become an engineer and sharing all your professional knowledge with me to help me progress. I am so happy to have you.

I would like to thank my cat Karides, who has been with me since I was 11 years old, for his love and friendship. He is the one who always lifts my spirits when I am sad and sometimes the only one I can tell my troubles to. We are so lucky to have you as a part of our family.

I would like to thank all my friends who has been there for me during the process, providing me emotional support, whom I could not talk to or meet during my studies, for their understanding and patience. I am so glad that life brought us together.

I would also like to thank my fellow research assistants who helped me in every way during my studies and created an environment that allowed me to work comfortably.

ABSTRACT

Yağmur YARIMBIYIK

HAND STRUCTURE DETECTION AND SUITABLE NAIL TYPE RECOMMENDATION SYSTEM

Başkent University Institute of Science and Engineering Department of Electrical and Electronics Engineering 2024

Hands play a crucial role in human interaction and functionality, essential in activities ranging from basic tasks to complex operations. They are essential to fields like biometrics, ergonomics, healthcare, robotics, and the cosmetics industry, particularly in nail care and aesthetics. Understanding hand types can significantly enhance product development and personalization. This article proposes a novel approach for classifying hands based on their dimensions using deep learning methods to recommend nail types. Traditional methods rely on manual measurements or complex feature engineering, which are labor-intensive and prone to errors. In this study, deep learning techniques have been leveraged to automatically classify hands into distinct categories based on palm length, palm width, and middle finger length, and suggest nail types for each hand shape accordingly. A dataset of 2050 images was collected and annotated for classification. Various Convolutional Neural Network (CNN) architectures, including VGG16, LeNet-5, AlexNet, GoogLeNet, Residual Network (ResNet), Dense Convolutional Network (DenseNet), and MobileNet, were tested and compared for accuracy. VGG16 emerged as the most successful model, achieving high accuracy in classifying hands into predefined categories. Based on these classifications, the suggested model recommends two nail types for each hand type, from a total of seven different nail types. The outcome of the applied model was assessed using standard metrics, advancing hand classification techniques to offer a robust, automated solution for personalized nail recommendations.

KEYWORDS: Hand Type Classification, Deep Learning, Convolutional Neural Networks, Nail Type Recommendation, Hand Dimensions, Nail Shape, Transfer Learning.

ÖZET

Yağmur YARIMBIYIK

EL YAPISI TESPİTİ VE UYGUN TIRNAK TİPİ ÖNERİ SİSTEMİ

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

2024

Eller, insan etkileşimi ve işlevselliğinde kritik bir rol oynar; temel görevlerden karmaşık operasyonlara kadar geniş bir yelpazede kullanılır. Biyometrik, ergonomi, sağlık, robotik ve kozmetik endüstrisi gibi alanlarda önemlidir, özellikle tırnak bakımı ve estetiği alanında. El tiplerinin anlaşılması ürün geliştirme ve kişiselleştirme açısından önemlidir. Bu makale, el boyutlarına dayalı olarak ellerin sınıflandırılmasını ve her el şekli için tırnak tipleri önerilmesini sağlayan derin öğrenme yöntemlerini önermektedir. Geleneksel yöntemler manuel ölçümlere veya karmaşık özellik mühendisliğine dayanır, bu da işgücü yoğunluğuna ve hata riskine neden olabilir. Bu çalışmada, avuç içi uzunluğu, avuç içi genişliği ve orta parmak uzunluğuna dayalı olarak ellerin otomatik olarak farklı kategorilere sınıflandırılması ve her el şekli için tırnak tipleri önerilmesi için derin öğrenme teknikleri kullanılmıştır. Sınıflandırma için 2050 görüntüden oluşan bir veri seti toplanmış ve etiketlenmiştir. VGG16, LeNet-5, AlexNet, GoogLeNet, Residual Network (ResNet), Dense Convolutional Network (DenseNet) ve MobileNet gibi çeşitli Evrişimli Sinir Ağı (CNN) mimarileri doğruluk açısından test edilmiş ve karşılaştırılmıştır. VGG16, önceden tanımlanmış kategorilere eli başarıyla sınıflandırma konusunda yüksek doğruluk elde ederek en başarılı model olarak ortaya çıkmıştır. Bu sınıflandırmalar temelinde önerilen model, her el tipi için yedi farklı tırnak tipinden ikisini önermektedir. Uygulanan modelin sonucu standart metrikler kullanılarak değerlendirilmiş ve kişiselleştirilmiş tırnak önerileri için sağlam, otomatik bir çözüm sunulmuştur.

ANAHTAR KELİMELER: El Tipi Sınıflandırması, Derin Öğrenme, Evrişimsel Sinir Ağları, Tırnak Tipi Önerisi, El Boyutları, Tırnak Şekli, Transfer Öğrenmesi.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF SYMBOLS AND ABBREVIATIONS

Adam	Adaptive moment estimation
AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Networks
CT	Computed Tomography
DBN	Deep Belief Networks
DBSCAN	Density Based Spatial Clustering of Applications with Noise
DenseNet	Densely Connected Convolutional Networks
e.g.	exempli gratia
et al.	et alia
FN	False Negative
Fortran	the FORmula TRANslation
FP	False Positive
GAN	Generative Adversarial Network
GB	Gigabyte
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
ID3	Iterative Dichotomiser 3
i.e.	id est
iOS	iPhone Operating System
KNN	K-Nearest Neighbor
Linux	Lovable Intellect Not Using XP
LSTM	Long Short-Term Memory Network
MacOS	Macintosh Operating System
MAE	Mean Absolute Error
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron

MRI	Magnetic Resonance Imaging
MS COCO	Microsoft Common Objects in Context
MSE	Mean Squared Error
NumPy	Numerical Python
OpenCV	Open Source Computer Vision Library
OS	Operating System
PCA	Principal Component Analysis
PDF	Portable Document Format
PNG	Portable Network Graphic
PSF	Python Software Foundation
RAM	Random-Access Memory
ReLU	Rectified Linear Unit
ResNet	Residual Network
RGB	Red, Green and Blue
RL	Reinforcement Learning
RMSprop	Root Mean Square propagation
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
tanh	hyperbolic tangent
TN	True Negative
TP	True Positive
t-SNE	t-distributed Stochastic Neighbor Embedding
Unix	Uniplexed Information Computing System
VGG	Visual Geometry Group
VGGNet	Visual Geometry Group Network
VOC	Visual Object Classes
Windows	Wide Interactive Network Development for Office Work Solution
XAI	eXplainable AI
YOLO	You Only Look Once
C	components in the numerator
D	components in the denominator
e	Euler's number

i	class index
M	value of metric
N	number of classes
w	number of instances
π	the number Pi

1. INTRODUCTION

Throughout history, perceptions of beauty have undergone a fascinating evolution, reflecting the dynamic interplay of cultural, societal, and individual influences. As a continuation of this, in the contemporary world, the importance of beauty has become increasingly pronounced, influencing various aspects of our lives from personal confidence to professional opportunities. We want to look good for acceptance in a certain place, sometimes we just want to be healthy, maybe we get influenced by media, or we just want to feel good for ourselves. Beauty can have lots of meanings and it can differ from person to person. However, regardless of how much other people may not value it, it still exists and can be achieved in different ways. There are many ways to do this. It could be a visit to a hair salon where we customize our hair color. Otherwise, trying to feel bold for once and changing our outfits and making up differently would help us accomplish our goals too. These are the things that make us happy. And what makes us happy, makes us beautiful.

Although people feel beautiful in different ways, it all comes down to aesthetics. Within this superficial stress on attractiveness, one often forgets that the beauty of hands is not just about their nails' shape. For instance, fingernails come in various shapes and styles that help enhance overall looks. By considering how society's ideas of beauty and each person's unique style interact, this work tries to get to the bottom of how hand aesthetics are related to nail shape so as to promote a more holistic or individual approach towards looking great in today's ever-changing world.

The need to establish the right nail shape for each individual becomes important in light of this emphasis on hand and nail aesthetics. This person has some choices that will help them to know the best nail type that suits them. Given the multitude of options available, social media is now a popular platform for various types of content, including personalized beauty advice. The person interested in the subject has to consult a professional or look through different websites with different suggestions, which takes time and causes confusion. But, as mentioned in the previous section, one of the best ways to find out which nail shape is suitable for the person is to look at the hands of the individual, since they complete each other by looks, and it is also an approved method by most nail artists.

Hands exhibit a wide variety of shapes, influenced by factors such as genetics, gender, age, and ethnicity. Their specific dimensions are not as useful to focus on as their overall shape if hands are to be effectively classified. By measuring some hand dimensions and

examining shape ratios, we can put hands into four clear types. This is helpful in giving appropriate advice like the best nail shapes for each type of hand which improve beauty and increase use.

The approach to classifying and recommending fingers can be further developed with advanced technology. For example, retinal recognition, ear shape analysis, and gait analysis have all witnessed rapid advances in pattern recognition over the past decade [1,2,3]. These areas have shown how image-based methods particularly deep learnings are capable of handling complex visual inputs.

Likewise, hand recognition has become-to-date one of the most established applications of this technological development, as a common form of biometric identification of persons based on the distinct characteristics of hands. This application is highly backed up by deep learning most importantly through Convolutional Neural Networks (CNNs) which are able to learn features from the input images in a hierarchical fashion [4]. They excel at recognizing patterns and objects in visual data and are useful to solve such problems as hand recognition. However, tuning the parameters of CNN can be a rather difficult task because of the architecture of this type of network and the computational complexity of the task. In order to resolve these challenges, transfer learning has become a feasible approach [5]. Instead of training from scratch and overcoming all the disadvantages of such an approach, transfer learning significantly alleviates the computational burden and shortens training times by utilizing the valuable image features learned by pre-trained CNN models during their initial training phases. In this respect, it improves not only the speed but also accuracy and optimization of detecting specific features on a hand. Therefore, this method helps in identifying a person and is expanded to the relevant fields such as nail disease identification and categorization. The application of CNN approaches in these fields demonstrates the potential of deep learning in enhancing the practicality and accuracy of hand and nail analysis and achieving the transition between practice and classification.

Based on this possibility, this study employs a CNN-based method for promoting the practical application. The developed model applies CNN techniques to examine images of human hands to identify the most appropriate nail types based on the shape of the hands. In the era of social networking sites where people are able to get customized beauty tips every now and then, people get confused by the variety of options they are offered or the fact that they have to seek professional help. This traditional approach is quite time consuming and at the same time confusing to most users. Instead, users can take a picture of their hand and, as a result, get nail type recommendations from the software. This new idea not only does

not require contacting outside experts like beauty specialists or nail artists but also saves money and time. So, it can be said that the process is aimed at minimizing the amount of time needed to determine which type of nail is the best for the client by offering recommendations based on the shape of users' hands.

The process includes recommending two different nail types for each of the four hand types based on analysis from photographs of hands using VGG16-based deep learning techniques. It allows users to observe what type of nail is recommended with just a photograph of the hand. This eliminates the need for visiting a beauty center or making an online search, it facilitates saving in both time and effort. In this paper, the task of hand shape recognition is done with VGG16, and the success rate of the process has been checked with a standard success criteria.

The goal is to determine the efficiency and effectiveness of recommending the most appropriate type of nail depending on the hand types of people. Done as a deep learning based research, different deep learning models would be examined and results would be compared. The findings of the study will demonstrate how effective and efficient deep learning methods are for recommending nail types based on hand types. It is anticipated that this study will contribute significantly to the automation of hand type determination and hand-based recommendation systems.

Contribution of this study is as follows:

- Constructing a well-defined and annotated dataset, identifying distinct categories of handheld devices through extensive literature and online research.
- Developing a new hand classification system based on hand shape, including palm sizes and finger length, has been proposed for use in cosmetic applications.
- Applying transfer learning to assess various CNN architectures, concluding that VGG16 demonstrates the highest performance.
- Resulting a study applicable to the cosmetic industry.

2. LITERATURE REVIEW

2.1. Overview of Previous Research

Research on hand types and nails has rapidly been changing and has taken two very different directions with respect to nail classification methodology: classical statistics, as the use of extensive and explicit detail in such hand measurements as lengths of fingers and palms as a means to characterize hand types, has been employed for a long time. Among those, factor and cluster analysis on large data sets have been very useful in finding human hand morphology to design for ergonomic, biometric, and robotic systems. Deep learning techniques, such as CNN, have recently been developed, furthering the ability of hand localization and classification with refined algorithms and specific datasets more accurately.

Nail-related research, on the other hand, is more health-related in applications. Although several traditional studies reported on different aspects of nail aesthetics and nail dermatology, recent works have continued to use CNNs and other contemporary Machine Learning (ML) methods in analyzing the condition of the nail, with the aim of improving the segmentation and classification of nail images for the purpose of early detection of diseases and accurate clinical assessment. This rise is related to the automated, reliable diagnostic tools, which have improved the efficiency and accuracy of medical assessments.

2.2. Studies on Hand Structure and Function

Hands are used in a lot of human interaction and functionality, central to tasks from daily activities to complex operations. These are pivotal across different disciplines of research, having applications in biometric recognition, ergonomic design, and healthcare, while also being applied in robotic applications and sign language research [6]. For example, in biometrics, examination of hands and fingers enhances the security system since it adds more layers of verification of one's identity [7,8,9]. In ergonomics, knowledge about the different kinds of hands informs the design of tools, workspaces, and consumer products for more comfort and efficiency of users [10,11]. Healthcare benefits from insights into hand morphology and nails for early disease diagnosis [12]. Hand structure is also beneficial in robotics, as hand classification is useful in robotics for creating sophisticated prosthetic limbs and robotic hands that mimic human dexterity [13,14].

The classification of human hands has been explored in many studies due to their use in various areas. Statistical analysis has been a common method, as seen in a survey by Jee

and Yun [15] on Korean hand shapes. Hand images from the "Korean Hand Measurement Project" were classified via factor and clustering analysis, considering 3D detailed hand measurements, including finger, hand, and palm lengths, joint and wrist breadth, circumferences, and hand depth. Another study used hand length and finger lengths for determining gender from hand dimensions via statistical analysis, differing by using ratios to determine shape [16]. An anthropometric hand survey was done with similar statistical methods and measurements for dentistry students and another for the Czech population [17,18].

A study by Ermolenko and Khayrullin [19] employed regression tree analysis to predict morphological hand types using radiography data. This method provided clearer and more accurate measurements by focusing on the fingers and distances between bone joints. Regression analysis has also been used to classify hands for smart wearables, focusing on fingers to ensure various hand types fit smart wearables like gloves. This classification was done by analyzing finger lengths, creating four finger types: Uphill, Downhill, Mountain, and Horizon [20].

Aside from statistical methods, various deep learning methods have also been used in studies about hand identification and biometry, such as detecting hands in the wild or biometric identification by analyzing hand veins [21,22]. These studies have employed CNN-based methods and achieved high accuracy. Popular datasets like Microsoft Common Objects in Context (MS COCO) and and Pascal Visual Object Classes (Pascal VOC) have been used to identify hands in real-world environments, and the "Badawi hand vein dataset" has been used for biometric hand identification.

2.3. Studies on Nail Morphology

Nails, in the context of hand functioning, form an important aspect not only from an aesthetic but also a functional perspective. Nails have been studied extensively in the disciplines of dermatology, cosmetics, histology, and identification [23,24]. They have also played a main role in the cosmetics sector, as they offer customization potential that is worthy of investigation for the understanding of shapes and aesthetics of natural nails [25]. More recently, CNNs have also been used toward further analyzing and classifying nail shapes, underpinning our understanding about nails in the context of hand structure and implications.

In a paper by Galmés et al., [26] a method has been proposed to segment nails in images accurately for clinical purposes. There are majorly three steps: preprocessing to enhance the quality of the image, then applying geometric constraints like Hough transforms in finding

the nail boundaries, and using ML algorithms such as Random Forest and Support Vector Machine to further refine the segmentation process. All of these techniques are applied to make the segmentation automated and the nail measurement clinical for more accuracy, hence rendering quickness and reliability of measurement in medical applications.

In the paper by Alghamdi et al., [27] a system for identifying a person with images of fingernails and knuckles is proposed. It uses features extracted from these images for deep learning-based identification, which applies the Bray-Curtis similarity measure. Deep learning techniques comprise CNNs in the work for feature extraction, while Random Forest and SVM algorithms are employed to improve precision and resilience of the identification process. This approach would provide a reliable, automated model for biometric identification using the unique pattern in fingernail and knuckle images.

In another study by Indi and Patil, [28] the same authors used ANNs for classifying the extracted features under different disease conditions with ML methods, excluding KNN. A system such as this will improve early detection and diagnosis of health issues by using visual changes in nails as diagnostic markers. Similarly, the study by Yamaç et al. [29] used CNNs to evaluate a database of 234 nail images, classifying them according to five various diseases using Python programming. These studies demonstrate highly efficient advanced ML techniques in nail analysis, which offers huge potential to increase the rate of early illness detection and diagnostic accuracy.

2.4. Gaps in the Literature

The approaches researched for this study were normally done using statistical methods reliant on such manual measurements or complex feature engineering. By convention, hand classification is done by methodologies based on manual measurements or complex feature engineering. All of these methods are mostly time-consuming and not accurate. Deep learning methods, very common in most fields of hand analysis, have not been applied explicitly in hand type classification. It has, therefore, developed a sophisticated system in this regard that can divide hands into different groups according to their ratios with regard to palm length, width, and middle finger length by using deep learning techniques in order to bridge the gap and automate the process.

While there had been progress in nail research and cosmetic applications, very little work was done within the cosmetic industry pertaining to automation. Therefore, a huge lacuna still remains regarding correct and personalized nail type recommendations, which holds a central place in the cosmetic nail industry.

3. THEORETICAL FRAMEWORK

3.1. Hand Structures and Nail Types: Their Relationship

This section provides a theoretical background to the thesis, explaining the types of hands, how they are classified, and the types of nails, what shapes they come with, and the relationship between the two.

3.1.2. Classification of hand structures

Hand type can tell us a lot about a person. Sometimes it offers valuable insights into an individual's nature, encompassing their personality traits, preferences, intellectual inclinations, career paths, and beyond. A hand can be broadly categorized based on its shape, meaning overall structure, proportion, and finger length. We can divide the hands into many classes by looking at different features. But in general, hand types are divided into four, based on the ratio of palm lengths and finger lengths. These types are:

- **Grounded Hands:** Also known as "earth hands", grounded hands are characterized by a square (wide) palm and short fingers, indicating stability and a focus on tangible results. Such hands usually have thick skin and a firm grip, a sign of resiliency. They are normally strong and muscular, denoting the strength needed to do physical tasks and to overcome problems by application of physical strength.
- **Piano Hands:** Known as "fire hands," the piano hand is recognized by its rectangular (long) palm and short fingers, signifying a balance between creativity and practicality. The grip will likely be firm and strong in these hands, showing tenacity. Most of the fingers are flexible, well-coordinated, and suited for intricate tasks and artistic works.
- **Hi-Five Hands:** Also referred to as "air hands," hi-five hands present a square (short) palm with long fingers, symbolic of articulate expression and clear communication. They often have prominent knuckles, reflecting dexterity and agility in conveying ideas. These hands are usually less muscular and more bony, emphasizing finesse and precision.
- **Generous Hands:** Commonly known as 'water hands', generous hands are characterized by a rectangular (long) palm and long fingers, indicating flexibility and adaptability. The soft, smooth skin suggests a focus on tactile sensitivity rather than physical strength. Additionally, the overall elegant appearance of the hand

supports precise and adaptable movement.

The illustration of four hand types and their characteristics are presented below in Figure 3.1.

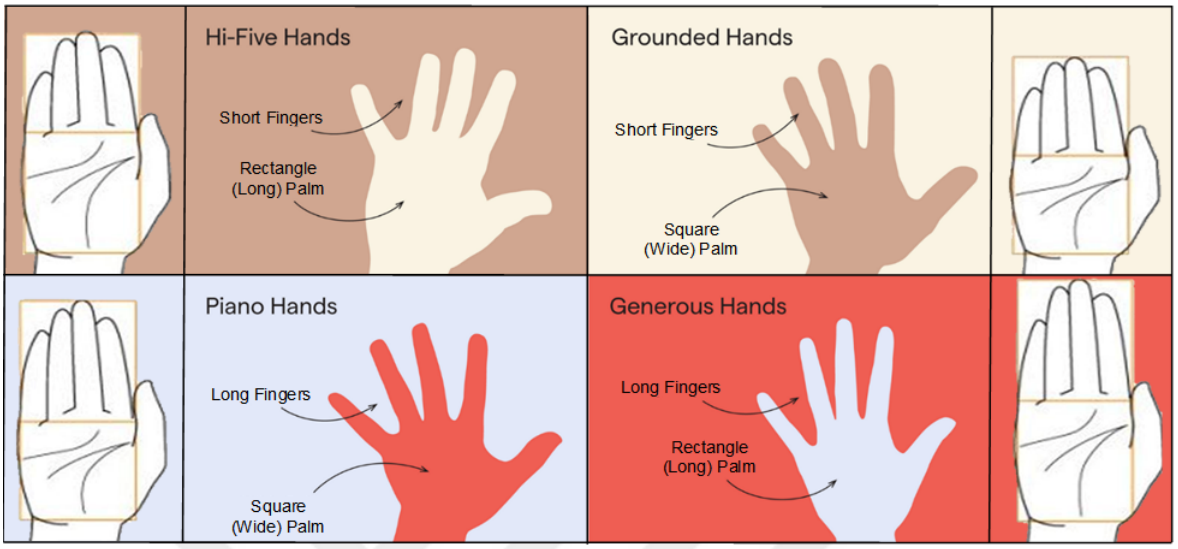


Figure 3.1. Types of hand shapes and their characteristics (Adapted from [30; 31])

3.1.3. Common Nail Shapes

Nail shapes can be chosen to complement and enhance the natural appearance of the hands. Generally, nail shapes are named after a geometric shape or a similarly shaped object. Some popular nail shapes include:

- **Natural Nails:** This nail type refers to the nails in their unaltered state, without any enhancements like acrylics or gels. They are trimmed and shaped to maintain a neat and clean appearance. Maintenance involves regular trimming, filing, and moisturizing to keep them healthy. Although being versatile and suitable to be used every day, they can be susceptible to breakage, hence requiring protection and care.
- **Oval Nails:** They have tapered sides with a rounded tip and are hence egg-shaped. They bring about a sleek, classy overview that stretches your fingers while it provides a soft alternative to the square nail. They need regular filing to maintain the shape. The oval nails are versatile and suitable for various activities, but they can be prone to snagging if not properly maintained.
- **Square Nails:** They have straight sides with a flat, square tip. They have sharp corners with straight edges and, therefore, give out a very strong and bold appearance. The routine filing is necessary to maintain them smooth on the edges

so they do not snag. Square nails are robust and suitable for daily wear but often require additional care to avoid chipping.

- **Almond Nails:** The sides of the nail are tapered, ending in a rounded point, giving it the shape of an almond. They make fingers appear longer, achieving an image of elegance and refinement. Maintenance involves daily filing to keep the tip at the almond shape it's meant to mimic. Almond nails are in style and perfect for special events, but they may require attention to prevent breakage.
- **Squoval Nails:** These nails are made strong by joining the strength of the square nails and the softness of oval nails. These nails have straight sides with a slightly rounded-off tip. This squoval shape makes one look balanced and very versatile, fitting well with different hand shapes. They are maintained by regular filing to retain the shape of squoval. Squoval nails are practical for everyday use and, therefore, serve as a compromise between square and oval shapes.
- **Round Nails:** These types are rounded at the tip, following the natural contour of the fingertip. They have a soft and unobtrusive look, offering a natural and versatile look for short nails. Maintenance is shaping on a regular basis in order to keep the roundness of the tip. It works very well in real life and doesn't snag on clothes like other shapes do.
- **Coffin/Ballerina Nails:** Their sides are tapered with a flat, squared-off tip, and they resemble either a coffin or a ballerina slipper. They create a sleek, elongated appearance with a modern, dramatic look. They require filing regularly to maintain the shape and prevent chipping. Coffin nails are quite fashionable for special events but not so practical for everyday since they are long.
- **Stiletto Nails:** These are long and pointy nails that resemble the stiletto heels of a shoe. They present length, which gives a bold, stylish look. But due to their shape, they need to be carefully maintained. With their sharp point, there is a need for regular shaping of the edge in order to be able to maintain the profile, so they don't become weak points. These nails are fashionable for special events only because they might not be practical for wearing every day and are fairly sharp and pretty long.

The illustration of four nail types is shown below in Figure 3.2.

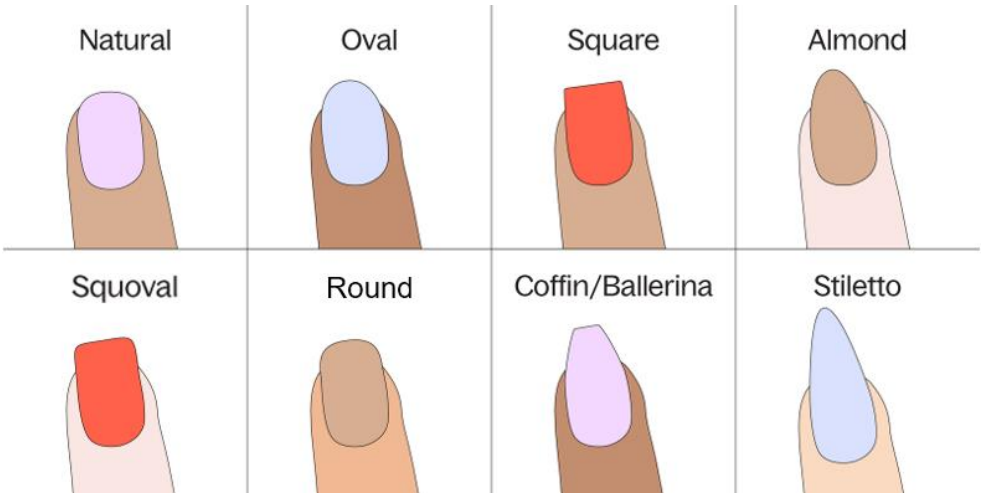


Figure 3.2. Types of nail shapes [31]

3.1.4. Impact of Hand Structures on Nail Types

The relationship between hand shapes and nail shapes is crucial for achieving a balanced and aesthetically pleasing appearance. Selecting the appropriate nail shape can bring out the hands' inherent beauty, create a harmonious appearance, and even cover up perceived flaws. The right nail shape for the hand types and their reasons are as following:

- **Grounded Hands - Almond Nails & Oval Nails :** Grounded hands, known for their short fingers and wide palm, can benefit from nail shapes that balance and elongate their appearance. Almond nails, with their tapered shape ending in a rounded point, make fingers look longer and add elegance. This helps counteract the short finger appearance, creating a more balanced look. Similarly, oval nail, gently tapered with a rounded tip, elongate fingers and soften the overall hand appearance. Choosing almond or oval nail shapes can enhance the natural features of grounded hands, creating a more balanced and refined look while maintaining proportion.
- **Piano Hands - Coffin/Ballerina Nails, Stiletto Nails:** Piano hands have long fingers with a wide palm, which is a little balancing act. Coffin or ballerina nails are long and tapering; they help in elongating the fingers and bring in sophistication. These nails draw attention away from the wide palm, making the hands look more balanced. Stiletto nails, being pointed, also elongate the fingers and add has an elegant effect. Both the nail styles go well with the elegance of the piano hands. giving a sophisticated look.

- **Hi-Five Hands - Oval Nails & Round Nails:** Hi-five hands have short fingers and a long palm, which gives unique proportion to it. Oval nails are recommended, that have a gently tapered shape with a rounded tip. They make fingers look longer and soften the appearance of the hand, thus balancing well with the long palm. Round would also work well. Their natural and rounded shape creates an illusion of longer fingers. In both nails, the result is a proportionate and balanced hi-five hands look.
- **Generous Hands - Square Nails & Squoval Nails:** Square nails and squoval nails are the best choice for generous hands. Generous hands are those that possess the characteristics of having long fingers and a long palm. The square nail has straight edges with sharp corners, which provides structure. It goes well on the hands and makes the hands look clean and modern. Squoval nails take the structure of the square nail with softened corners. This thus provides it with a well-balanced, stylish look that is not too sharp in nature. Both nail shapes do their part in complementing full hands with their natural features and creating a sophisticated look in harmony.

As explained above, it can be said that the recommended nail shapes for different hand types are selected to balance proportions, enhance natural features, and create a harmonious appearance. Whether it is making short fingers look longer, adding structure to wide palms, or keeping long hands sleek, the right nail shape can make a big difference. The appropriate nail type for each hand type and the reasons for each are briefly explained below in Figure 3.3.

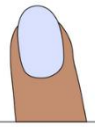
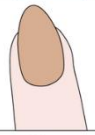
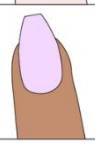
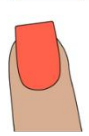
HAND TYPE	HAND FEATURES	BEST SHAPE	WHY?
Hi-Five Hands	Short fingers, rectangle palm		This shape will help make your fingers appear longer and more slender to balance out the length of your palm.
Grounded Hands	Short fingers, square palm		Tapered shapes like almond elongate shorter or wider fingers.
Piano Hands	Long fingers, square palm		The edgy coffin or ballerina shape is enhanced by long fingers and balanced by a square palm.
Generous Hands	Long fingers, rectangle palm		Short but stylish shapes ensure your fingers don't look too long and spider-like.

Figure 3.3. Optimal nail shapes for different hand types [31]

3.1.5. Practical and Aesthetic Considerations

When recommending nail shapes based on hand shapes, both practical and aesthetic factors should be considered. Practical factors include the individual's lifestyle, nail maintenance preferences, and the natural strength of their nails. Aesthetically, the goal is to create a balanced and harmonious look that enhances the natural beauty of the hands.

- **Lifestyle:** Active individuals or those who frequently use their hands may prefer round or square nails for their durability.
- **Maintenance:** Some nail shapes are pretty tricky and a bit time-consuming to be maintained; for example, stilettos or coffins are not for everyone to wear.
- **Nail Strength:** If the person has weak nails, they would prefer shapes that reduce breakage, such as a round or oval shape.

By knowing how to relate the shape of the hands with that of the nails, one can then make personalized recommendations on this aspect to satisfy both the overall appearance and needs and wishes of the subject. This holistic approach will ensure that the nail shape chosen will not only complement the hand shape but also have practicality with which it has a satisfying and pleasing outcome.

3.2. Artificial Intelligence (AI) and Machine Learning (ML)

Of the emerging technologies, AI and ML have been leading the revolution into a new technological age. These two fields, while quite intertwined, drive innovation in different industries with varying methodologies and competencies. The focus of this section is to consider the basic ideas of AI and ML, therefore giving a fair overview of their principles, applications, and effects on modern technology.

3.2.1. Artificial intelligence (AI)

AI refers to developing computer systems that perform concrete tasks that are very much like human beings. This involves natural language understanding, data pattern recognition, decision-making based on the conclusions drawn, and learning through prior experiences. It aims at machines that can emulate human abilities, including activities such as problem-solving, reasoning, perception, and understanding language.

The AI systems are developed in such a way that they can process an extensive amount of data, where inferences can be drawn and decisions or predictions can be made. They rely on techniques like ML and deep learning for modeling and interpreting complex patterns

while sweeping disciplines such as natural language processing, computer vision, and robotics in enhancing these capabilities over various kinds of tasks.

History of AI dates back to the 1950s, with pioneers like Alan Turing and John McCarthy, who laid early foundations for the field. Turing's idea of a "universal machine," including his almost folkloric "Turing Test," set the thinking about machine intelligence. In 1956, at the Dartmouth Conference, McCarthy coined the term "artificial intelligence," marking the birth of AI as a research field. The early research in AI focused on symbolic reasoning and problem-solving, but it slowed down in the 1970s because of limited computational power and overstated ambition. It then reemerged in the 1980s with the introduction of expert systems to again regain prominence in the 21st century with the developments in ML, deep learning, and the availability of omens of data and powerful computing resources. AI is evolving today, continuously undergoing innovations in most fields.

The ultimate goals of AI are to offer systems that can learn over time, self-improve, and become more capable of executing complicated tasks independently. However, in recent years, due to the advancement of AI systems, it has become very important to understand the reasons behind their decisions. This need for clarity gave birth to Explainable AI (XAI), which is one of the currently trending research areas focused on improving AI models' interpretability [32]. While colossal steps have been taken in AI and sub-domains like XAI increasingly gain prominence, it is in continuous evolution because research and development have to be continued and carried forward for further advancements and mitigation of capability and transparency-related challenges.

AI is an umbrella term representing a large ensemble of techniques and methodologies for the construction of intelligent systems, whereas ML is a subdomain of AI dealing with algorithms that empower computers to generate predictions and learn from the data without being explicitly programmed. Deep learning is a subcategory of ML that involves training several-layered deep neural networks to learn complex patterns in the data [33]. AI, ML, and deep learning thus form a hierarchy of techniques where the latter is a subset of the second and the second is a subset of the first (Figure 3.4.). These very interconnected fields drive innovation and advancement in industries with powerful tools to solve complex problems.

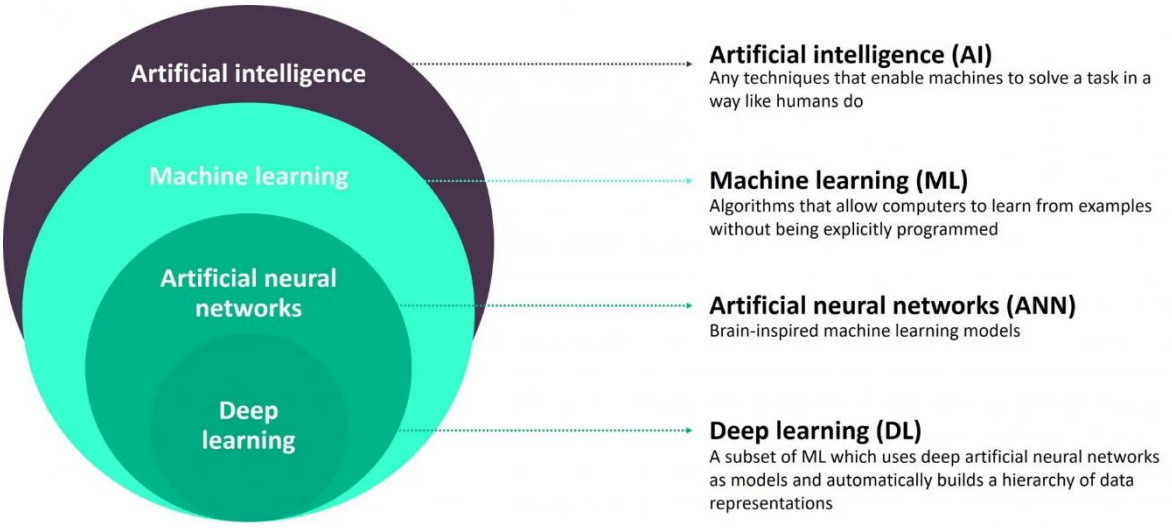


Figure 3.4. Venn diagram for AI subsets [34]

3.2.2. Machine learning (ML)

ML is a sub-branch of AI dealing with designing algorithms and statistical models that let computers perform specific tasks without explicit programming [35]. There is one core around the real topic of ML: learning patterns and basing judgments or predictions on them. As illustrated by Figure 3.5., it starts with the collection of relevant data from different sources. After that, it is pre-processed to ensure the dataset is appropriate. Then, model training and testing proceed with the input provided, all in the attempt to achieve an accuracy level as high as possible. If the validation goes well, the model is deployed, and the performance is tracked continuously to ensure the best possible results.

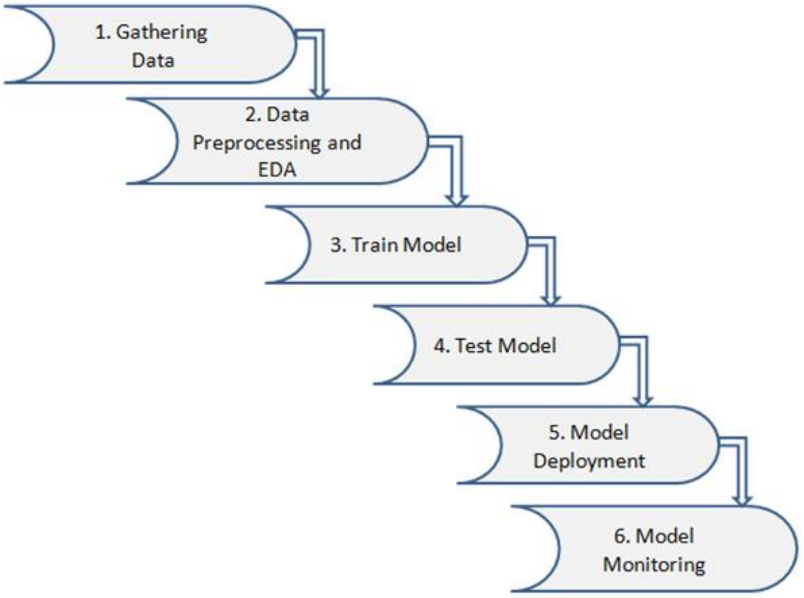


Figure 3.5. ML life-cycle [36]

There is a variety of ML models available, like supervised, unsupervised, and Reinforcement Learning (RL) by their nature, that can apply to a number of tasks and can make activities work by their nature without human interference [37]. It is a very potent procedure for making predictions, automating decision-making processes, and gaining insights from data in a range from finance and healthcare to marketing.

ML history goes back to the 50s and 60s, when some of the primitive models were developed along with their computational algorithms. Among the many, there was the perceptron, a simple kind of artificial neural network developed by Frank Rosenblatt in 1957. To this very period, other fundamental pieces of work belong: among others, decision tree algorithms developed by researchers like Arthur Samuel and John Ross Quinlan.

During the course of the 1970s and 1980s, ML developed and branched off into two main venues: symbolic AI and connectionism. Symbolic AI is concerned with using logical rules and symbols to represent knowledge, while connectionism places its emphasis on using ANNs as a possible model structure of the human brain. Important algorithms, like the ID3 (Iterative Dichotomiser 3) decision tree algorithm by Quinlan [38] and the backpropagation algorithm for neural network training, were developed during this period.

Some decade into the 1990s, the ML community gradually got inclined toward approaches based on statistical learning. This was the period when algorithms, including SVMs, Bayesian networks, and some others based on the principles of probability theory and statistics, were introduced. A further inclination in research included the working out of ensemble learning techniques, which combine several models to enhance the accuracy of predictions.

More interest and investment in ML came in the 2000s, partly because of the huge datasets called "big data" and improvements in computing power. Deep learning is just a subfield of machine learning, which deals with deep neural network training using a number of layers. Basically, it became very useful for image and speech recognition and natural language processing. Major turning points during this time include several breakthroughs in topics of CNN, Recurrent Neural Network (RNN), and domains like autonomous driving and computer vision.

From its very inception, ML has developed with the help of the development of algorithms, an increase in statistical data, and advancements in computing infrastructure. In turn, it gave birth to modern deep learning as a subfield of AI. Currently, ML is used for solving many different problems in such areas as medicine, finance, leisure, and

transportation.

Nowadays, with the abundance of existing datasets, there has been an increased interest and demand for ML. The primary objective of ML is to extract information and learn from data. Research on how machines can learn on their own involves examining various approaches to understand how computers can learn without being explicitly programmed. Many mathematicians and programmers have applied different methods to solve this problem. As a result, different algorithms have emerged to solve data problems, and there is no single solution method or algorithm. The type of algorithm used varies depending on elements like the problem's nature to be solved, the number of variables, and the type of model that is most suitable. This relationship is depicted in Figure 3.6. which categorizes algorithms into supervised, unsupervised, and RL.

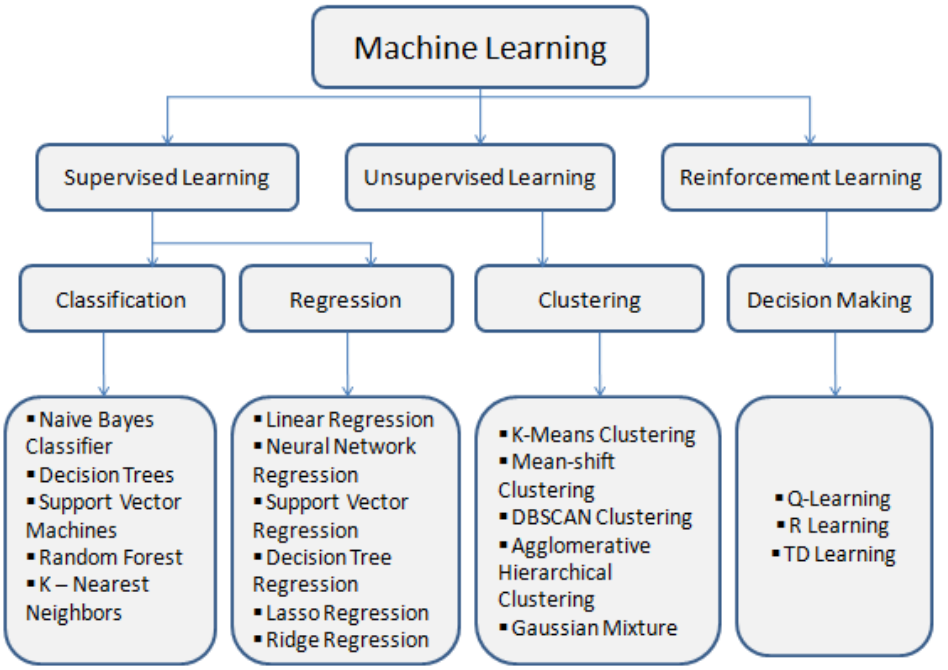


Figure 3.6. Types of ML algorithms [36]

3.2.2.1. Supervised learning

One of the most important approaches to ML is supervised learning, whereby a model is trained according to its given input-output pairs. The training datasets have input-output pairs wherein the input holds features and the output has respective labels against them. It should be ensured that a mapping from inputs to outputs is learned with respect to the model through the minimization of quantifiable differences between the predicted and ground-truth labels, and the parameters of a model are adjusted via an iterative optimization algorithm,

usually gradient descent, for reducing error on these predictions to improve model accuracy.

The two major types into which the process of supervised learning can be divided are classification and regression. In classification, the outputs are discrete labels, and the task is to assign the correct label to each input. For example, in image recognition, a model would sort animal images into 'cat', 'dog', or 'horse'. In regression, the outputs are real-valued, and it is a task that predicts a quantitative response. For instance, house prices should be directly estimated with respect to features like location, size, and the number of rooms. Each of these two types of tasks requires an in-depth understanding of the underlying distribution of data and the connections between the features of the input and the output in order to build robust models [39].

The supervised learning models' performance is assessed by different metrics, all of which vary based on the task's nature. For example, in classification, accuracy, precision, recall, and the F1 score are some of the most common, while in regression, some of the common ones are Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared. The serious problem in supervised learning is overfitting, where it does fine on the training data and poorly on unseen test data. Cross-validation, regularization, and pruning prevent these techniques to avoid overfitting and improve the generalization capability of the model [40]. An overall structure of supervised ML algorithms is shown in Figure 3.7.

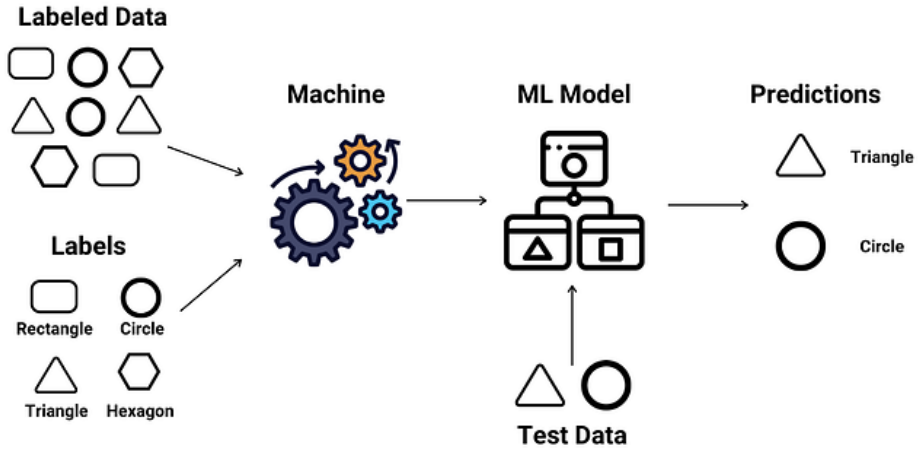


Figure 3.7. Structure of supervised learning algorithms [41]

3.2.2.2. Unsupervised learning

Another very important branch of ML that deals with unlabeled response data is unsupervised learning. Unlike supervised learning, where the aim is to learn a mapping using labeled data to move from inputs to outputs, unsupervised learning looks to find underlying structure and hidden patterns within the input data with a minimum of human supervision [42]. These are algorithms that try to identify groupings, trends, and connections inside the dataset without having prior knowledge of what the output should be. The goal is to simulate the data's underlying structure or distribution in order to gain a better understanding about it.

Some of the important techniques in unsupervised learning include clustering and dimensionality reduction. Algorithms for clustering, like k-means, hierarchical clustering, and Density Based Spatial Clustering of Applications with Noise (DBSCAN), are based on some measure of similarity and partition the data into disjoint sets or clusters [43]. For example, customer segmentation in clustering aids in identifying various customer groups with similar purchasing patterns. The dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), decrease the number of features in a dataset while keeping its principal structure. Among other applications, the techniques allow the visualization of high-dimensional data and preprocessing in order to make other ML algorithms more efficient.

Analyzing unsupervised learning models' effectiveness is inherently more difficult compared to supervised learning since ground truth labels are not available. The silhouette score, Davies-Bouldin index, and inertia quantify the quality of clustering. Reconstruction error and explained variance are common metrics to evaluate dimensionality reduction. Lastly, domain knowledge is often used to verify that discovered patterns make sense. It finds a lot of applications in other domains, such as anomaly detection, market basket analysis, and bioinformatics, among others, in contributing towards the discovery of hidden structures and insights in data that does not come with explicit labels. The overall structure of unsupervised ML algorithms is given in Figure 3.8.

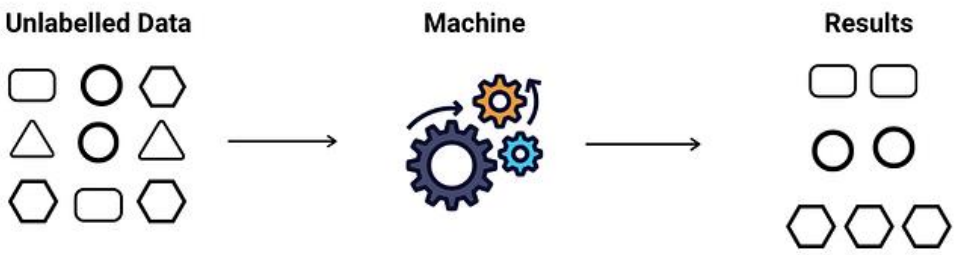


Figure 3.8. Structure of unsupervised learning algorithms [41]

3.2.2.3. Reinforcement learning (RL)

RL is a distinctive paradigm within ML where an agent learns to make decisions by interacting with an environment to achieve a specific goal. Unlike supervised and unsupervised learning, which work with labeled data and use the mapping between input and output, RL focuses on learning a policy through trial and error [44]. The agent takes actions in an environment and receives feedback in the form of rewards or penalties. The primary objective of the agent is to maximize the cumulative reward over time, which is achieved by learning the optimal policy - a mapping from states of the environment to the best possible actions.

The RL framework is typically formalized using concepts from Markov decision processes (MDPs). A MDP is characterized by states, actions, rewards, and transition probabilities. At each time step, the agent observes the current state, selects an action based on its policy, and transitions to a new state while receiving a reward. The agent's goal is to learn a policy that maximizes the expected cumulative reward, known as the return [45]. Key algorithms in RL include Q-learning, which updates the action-value function to estimate the expected return of taking an action in a given state, and policy gradient methods, which optimize the policy directly by adjusting the parameters in the direction of higher expected rewards.

Evaluating the performance of RL algorithms involves measuring how well the learned policy performs in terms of the cumulative reward. Common evaluation methods include episodic returns, which measure the total reward accumulated over an episode, and average reward, which averages the rewards over multiple episodes. Challenges in RL include exploration-exploitation trade-offs, where an agent has to balance between the exploration of new actions because it doesn't know their effects and exploit those known actions that yield high rewards. Techniques such as ε-greedy policies and entropy regularization help address this trade-off. RL has demonstrated impressive performance across a range of

applications, from gaming - as seen in AlphaGo - to robotic control, and autonomous driving, where it enables systems to learn complex behaviors through interaction with dynamic environments. The overall structure of reinforcement ML algorithms is given In Figure 3.9.

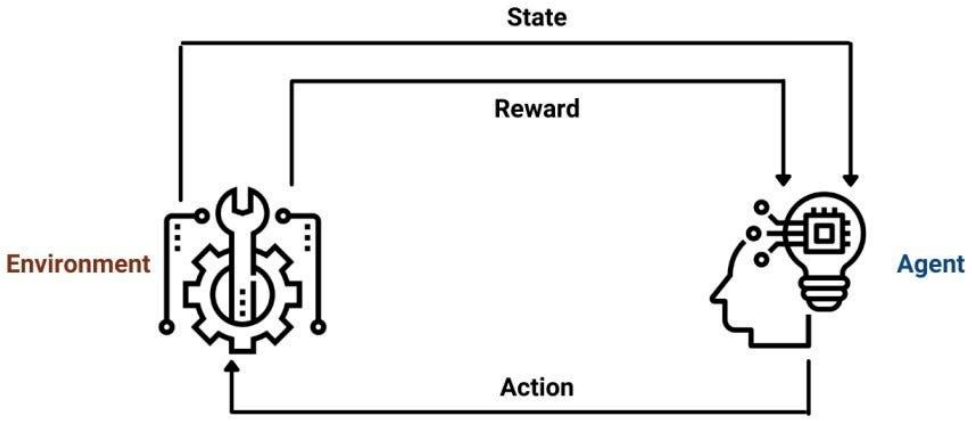


Figure 3.9. Structure of RL algorithms [41]

3.3. Deep Learning

Deep learning is a sub-branch of ML concentrated on algorithms inspired by the structure and function of neural networks in the human brain. It involves the use of ANNs with many layers and is, therefore, termed "deep." These deep neural networks are able to automatically learn representations from data through multiple levels of abstraction. This capability of handling high-dimensional data and extracting intricate patterns from them makes deep learning so powerful, especially in image and speech recognition, natural language processing, and playing complex games [46]. It means the process is one in which the network itself is trained, and through vast amounts of data, it learns. Each layer transforms the input data into higher and higher abstract and complex representations.

The training process of deep learning models includes a concept called backpropagation, where the discrepancy between what was predicted at an output and what truly is the output is propagated through the network to update the weights of the connections. Deep layers in a network are capable of capturing hierarchical features of data. This feature has been the key factor in deep learning, which has led to a good understanding and the prediction of the phenomena. Activation functions, one example being Rectified Linear Unit (ReLU), provide the network with nonlinearity, letting it learn nonlinear mappings [47]. Moreover, there exist techniques such as dropout for preventing overfitting and batch normalization for stabilizing the training process.

Deep learning can be traced all the way back to the middle of the 20th century, with

Warren McCulloch and Walter Pitts pioneering neural network early work in the 1940s. They came up with the very first mathematical model for a neuron. The second milestone happened in the 1950s and 1960s when Frank Rosenblatt invented the perceptron—a simple neural network model able to do binary classification. Development came to a halt in the 1970s due to lower computational power and the fact that single-layer networks couldn't resolve very complex problems, as Marvin Minsky and Seymour Papert pointed out in their famous book "Perceptrons" in 1969.

Interest in neural networks resurged with backpropagation in the 1980s and the 1990s. Major contributions came from Geoffrey Hinton, Yann LeCun, and Yoshua Bengio in these two decades. Deep learning really took off in the mid-2000s, based on improvements in computational power with Graphic Processing Units (GPUs), large data sets, and innovations in network structures. Landmark successes, most notably the win of AlexNet in the 2012 ImageNet competition, proved that deep learning really worked in practice and engendered full-throttle adoption and research. Deep learning is still evolving today, breaking AI boundaries and transforming industries.

3.3.1. Artificial neural networks (ANN)

ANNs are fundamental architectures in ML, motivated by the biological neural networks present in the brain of a human being. They are made out of linked nodes called neurons - the basic processing units in the human brain – which are arranged in layers [48]. These layers usually include an input layer, one or more hidden layers, and an output layer (Figure 3.10). Neurons in each layer process input data and transmit the results to the following layers through weighted connections. The essence of ANNs is their capacity to learn complex patterns and relationships within data by modifying the weights of these connections based on the error of their predictions [49]. Notable examples of ANN architectures include Multi-Layer Perceptrons (MLPs), RNNs, and CNNs, each tailored to specific tasks and data types, ranging from structured tabular data to sequential data like time series and images.

Historically, ANNs have their roots in the innovative research conducted in the 1940s by Walter Pitts and Warren McCulloch, who proposed the first mathematical model of a neuron. Subsequent decades saw advancements in neural network research, including the development of the perceptron by Frank Rosenblatt in the 1950s, which is able to perform binary classification tasks. In spite of early excitement, progress stagnated in the 1970s due

to computational limitations and the incapacity of the perceptron to handle complex problems. However, a resurgence occurred in the 1980s and 1990s with the introduction of backpropagation, a method for training multi-layer networks.

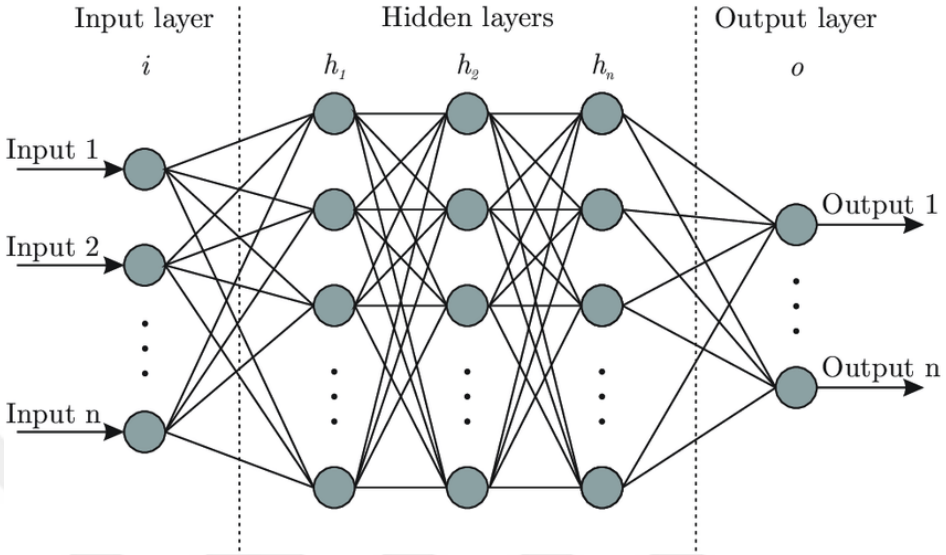


Figure 3.10. ANN architecture [50]

3.3.2. Deep learning algorithms

Deep learning algorithms have a different variety of architectures, and each one of them is specifically crafted to execute types of tasks or carry out certain aspects with types of data (Figure 3.11). Some of the famous examples include CNNs, which do image and video processing very well due to their spatial hierarchy feature detection ability. Thus, RNNs and their derivatives, like Gated Recurrent Units (GRUs) and Long Short-Term Memory Networks (LSTMs), are tailored for sequence data and hence very effective in applications like time series prediction and language modeling. Autoencoders are used for learning dimensionality reduction and anomaly detection by compressing and reconstructing data in an unsupervised manner. Generative Adversarial Networks (GANs) are made up of two neural networks in competition with each other; applications range from generating images to style transfer. Deep Belief Networks (DBNs) give a framework to pre-train deep networks unsupervised. The Transformer Neural Networks (Transformers), supplied with the attention mechanism, significantly change natural language processing tasks on machine translation and text summarization. All these different algorithms give an idea of the strength and versatility that deep learning has in handling complex problems in a wide array of domains.

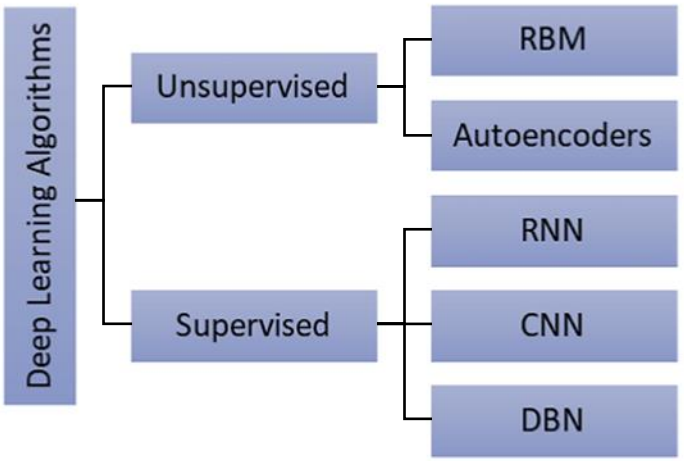


Figure 3.11. Deep learning algorithms [51]

3.3.2.1. Convolutional neural network (CNN)

CNN is an architecture in the subset domain of deep neural networks that are primarily built to process structured grid data, mostly images and videos. CNNs leverage a number of convolutional layers that apply a series of filters to the input data. This makes them efficient in highlighting hierarchical features of the data at different levels of abstraction. This has given them an inherent ability to automatically tell intricate spatial patterns, thus making CNN one of the key image-related tasks, like object detection, image recognition and medical image analysis, in the literature [52]. Some of the famous CNN architectures include LeNet-5, pioneered by Yann LeCun et al., after which innovation in handwritten digit recognition took center stage in the 1990s, followed by new models of AlexNet, Visual Geometry Group Network (VGGNet), GoogLeNet (Inception), Residual Network (ResNet), and MobileNet, each bringing some new novelties to specifically target certain bottlenecks in image comprehension and processing.

In the history of deep learning, the inception of CNNs was catalyzed by the seminal work of Yann LeCun and colleagues in the late 80s and early 90s. One of his seminal creations among them is LeNet, which proved really instrumental in proving the efficacy of CNNs on handwritten digit recognition. More than just vindicating the potential of CNNs, the success of LeNet opened the doors for many modern applications.

3.3.2.2. Recurrent neural network (RNN)

RNNs are specially designed architectures of ANNs that are particularly engineered to process sequential data, either time series or natural language. This is different from feedforward neural networks in that the recurrent connections of RNNs give them properties of memory, which can pick out the temporal dependencies underlying sequential data. This intrinsic capability of memory makes RNNs very suitable for tasks such as speech recognition, language modeling, and sequential prediction [53]. While the idea of RNNs can be traced back to the 1980s, their general use was first curtailed by challenges such as the vanishing gradient problem, which limits how far back in time information can flow and thus how well they could capture long-range dependencies.

It was not until the late 1990s and early 2000s, with the invention of LSTMs and GRUs, more sophisticated architectures that had overcome the problems traditional RNNs were plagued with, and in fact moved the study and application of RNNs much further ahead.

3.3.2.3. Generative adversarial network (GAN)

GANs are a very dynamic framework of deep learning that entails the communication between two neural networks called discriminator and a generator. The generator is designed to synthesize data samples, while the discriminator has the prime task of distinguishing between artificially produced and real samples. In particular, the adversarial setup generates competitive dynamics where the generator tries to produce samples that are ever more realistic in order to fool the discriminator, while the discriminator tries to refine its algorithm to tell the difference between this real-fake data. In particular, GANs have emerged as very helpful tools to generate high-quality synthetic data across a wide array of domain applications, going from images and videos up to audio. Furthermore, they find their application in tasks such as data augmentation and conversion from image to image, proving their versatility and potential usefulness in very different applications [54].

Invented in 2014 by Ian Goodfellow and colleagues, GANs opened up a new era in the development of generative models [55]. They have since been widely investigated and developed, and there have been a large number of different variants and improvements. This continuous development is actually one of the reasons why GANs have really surged in recent years, both at the academic level and in industry applications, driving innovation throughout the most diverse areas, from computer vision to natural language processing and healthcare.

3.3.2.4. YOLO (You Only Look Once)

You Only Look Once (YOLO) is a unique object detection algorithm that made its landmark entry in the field of computer vision with its accuracy and performance in real time. Proposed in 2016 by Joseph Redmon et al., YOLO proposed a revolutionizing approach toward object detection by presenting it as a regression issue. Contrary to conventional techniques that used region-specific proposal networks and processed the image multiple times, YOLO predicts the class probabilities of objects and bounding boxes in one pass. This cleaned-up architecture lets YOLO achieve impressive speed, so it is quite suitable for applications that need fast yet accurate detection, like self-driving and video surveillance [56].

The development of YOLO was a quantum leap in object detection methodology, motivating further research and innovations in the deep learning domain. YOLO wedged the efficacy of CNNs with a holistic approach of localization and classification, revealing its real-time visual comprehension ability [57]. Its influence is beyond the academic circle; its implementations, based on YOLO, are part of a number of commercial products and applications..

3.4. Image Processing

Image processing is defined as a process for checking, improving, and manipulating digital images to understand different meanings or improve the quality of visualization by using many techniques or methodologies. Image processing dates back to the middle of the 20^{th} century and can be treated as a sub-branch of signal processing. Its development was stimulated first by the growth of computing technology, followed by the diversity of digital imaging devices. Contributors at a very early stage in this field, such as Claude Shannon and Norbert Wiener, created the theoretical platform of image processing through their questioning of information theory and stochastic processes with regard to visual data. Indeed, image processing has benefited immensely from very different areas of expertise, especially mathematics, computer science, and electrical engineering, during its development [58]. Fundamental methods like segmentation, edge detection, and filtering are actually at the core of image processing algorithms for much more sophisticated tasks like noise reduction, feature extraction, and object recognition. Further developments, in particular deep learning-based image processing, have taken place over the past few decades. CNNs revolutionized tasks such as image classification, object detection, and semantic segmentation.

The practical applications of the image processing technique are very broad, covering a wide range of applications in medical imaging, remote sensing, security and surveillance, and multimedia content analysis [59]. Techniques in image processing facilitate diagnostic imaging procedures in medicine, like X-ray, Magnetic Resonance Imaging (MRI), and Computed Tomography (CT) scans, which help the clinicians in disease detection and treatment planning [60]. In the fields of satellite imagery and aerial photography, image processing supports land cover classification, environmental observation, and urban planning [61]. Security and surveillance systems use image processing in face and license plate detection and anomaly detection to afford improved safety and security among citizens [62].

Image processing is furthermore the process that includes several steps, which realize certain objectives:

1) **Pre-processing:** Preparing an image for further analysis or enhancement is called pre-processing. This may involve the elimination of noise, resizing, and conversions of color space. This level of preprocessing contributes to the aspect of providing an image in good quality, ensuring the consistency of images to be conserved in follow-up processes.
2) **Enhancement:** The process of an image that enhances better visual quality by bringing out essential features or increasing definite characteristics. Common enhancement techniques include contrast adjustment, brightness correction, and histogram equalization. These techniques are also used for making details visible and enhancing the overall clarity of an image.
3) **Filtering:** In this procedure, mathematical operations like filters are applied to the image for the extraction of unknown noise or the enhancement of certain features. Typical filtering operations include smoothing filters (e.g., Gaussian blur) to perform low-pass filtering for the removal of noise and edge enhancement filters (e.g., Sobel filter) applied to show edge details. Filtering improves the general quality of the image and can be customized in application-specific ways.
4) **Segmentation:** Segmentation is the process of partitioning an image into sections or meaningful areas depending on certain attributes, like color, texture, or intensity. As a result, segmentation automatically causes a partitioning of the image to meaningful regions that further allows its analysis with the extraction of more proper information. Accordingly, segmentation is used in object detection, image labeling, and feature extraction.

5) **Feature Extraction:** It is the process of finding and extracting salient features or patterns from an image. These would include edges, corners, textures, or shapes. Such extracted features in image or signal processing can be very useful for a job like, for example, object recognition, pattern recognition, and classification of an image. It helps to portray the image in a more compact and informative manner, which in turn is conducive for further analysis and processing.
6) **Object Detection and Recognition:** There are object detection and recognition algorithms designed based on the identification and location of certain objects within the image variety containing certain patterns found in the image. In many cases, the techniques involve ML algorithms such as the CNN for tasks such as face detection, object tracking, and understanding the scene. Basically, these are techniques used for tasks ranging from automated image analysis to interpretations of images, used in realizing systems like automated surveillance, autonomous driving, and medical diagnosis.
7) **Post-processing:** In post-processing, final adjustments or refinements are given to the processed image. It may involve image stitching, image blending, and even some amount of image composition. In a general sense, post-processing makes the final output congruent with the requirements for future use.

Digitalization is the process of converting analog signals or phenomena into a digital form. Basically, the digitalization of an image clicked by a camera is the representation of visual data in discrete numerical values, called pixels. Understanding pixel values is therefore very crucial because it provides a basis for various operations such as enhancement, filtering, and analysis in image processing.

Every digital image is composed of pixels, where every pixel corresponds to a small area of an image. A pixel contains color and intensity information for that small region. They are represented mostly by numeric values, usually ranging from 0 to 255 for grayscale images or with varying values for each color channel (i.e., Red, Green, Blue (RGB)) in the case of color images. These encode the brightness or color of each pixel in an image. For example, in a grayscale image, 0 may stand for black and 255 for white, while different classes of gray fill in the middle.

Pixel values can also be manipulated in order to give techniques in image processing the power to change the appearance of an image, extract useful information from it, or even detect objects and patterns within. Understanding these manipulations often relies on understanding the structure and distribution of pixels. Accordingly, Figure 3.12. is the digital

representation of a grayscale image. Plotting of pixel values and their distribution shows how images are digitally encoded for storage and processing. Every pixel gets a value that determines the color and intensity of the pixel; those pixels then become the building blocks of a digital image. It visualizes how the digital images are constructed, processed, and look in the actual world of image processing.

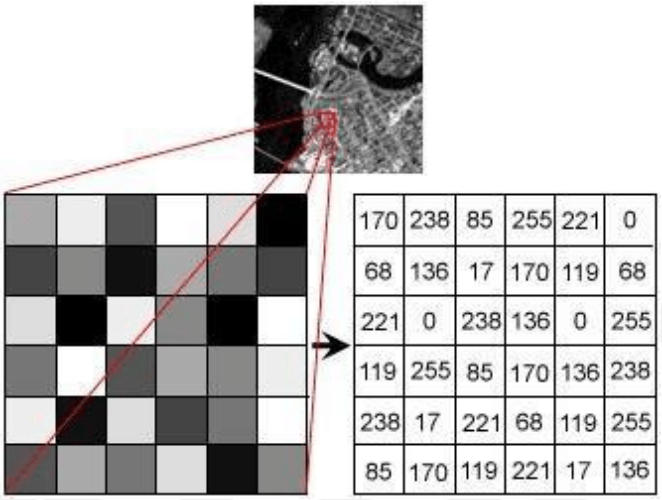


Figure 3.12. Intensity matrix for a certain area of a grayscale image [63]

One of the common steps in image processing as a part of preprocessing is rescaling. In particular, intensity normalization ensures that pixel values are brought to a standard range, usually [0, 1]. This is done simply by dividing each pixel's value by the highest value that it can have, usually 255. Such normalization processes are essential for maintaining consistency across image data and improving model performance [64]. In practical applications, this is implemented through various libraries. For instance, in Python's Keras library, setting the parameter `rescale=1./255` in the `ImageDataGenerator` class normalizes the pixel values from the range [0, 255] to [0, 1]. This normalization stabilizes and speeds up the training of ML models, as it ensures consistency in the input data and enhances the performance of the model. By normalizing the intensity of the pixels, the models can learn more effectively, leading to better accuracy and generalization in tasks like object detection and image classification. An example of intensity rescaling application is provided in Figure 3.13.

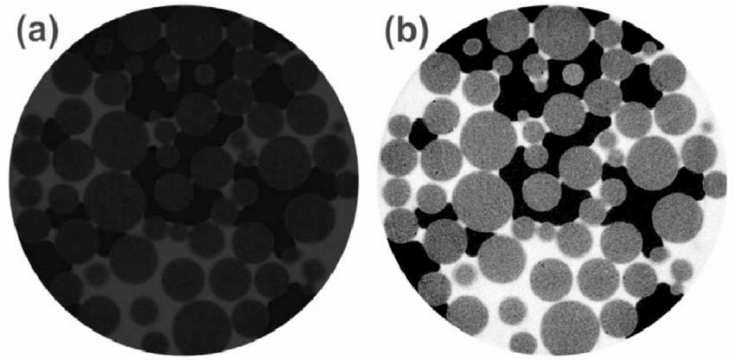


Figure 3.13. (a) Intensity rescaling applied to an 8-bit grayscale image obtained with synchrotron microtomography (b) the same image after intensity rescaling [65]

4. MATERIALS AND METHODS

In this study, CNN transfer learning methods were employed. Additionally, data augmentation was conducted to enhance the model's performance. Furthermore, hand classification was performed. The hands were classified into four categories, and the seven nail types were similarly divided into four categories, with two nail shapes in each. The requisite libraries were installed to facilitate hand feature measurement and the automatic construction of the dataset.

4.1. Applied Deep Learning Techniques and Model Optimization

In the previous section, deep learning techniques were discussed in depth. Although we employed CNNs in this study, the primary method utilized was transfer learning. While transfer learning is built upon CNNs, it introduces variations that distinguish it from traditional CNN methods.

4.1.1. Key features and structure of CNNs

CNN is an algorithm for deep learning that has been employed in numerous applications, such as machine translation, image processing and natural language processing. They employ filters to capture local features of images. They are the most popular deep learning model for processing images and videos. This algorithm has the ability to automatically learn features within images, and is employed in various tasks, like object classification, detection and face recognition. They represent one of the most successful deep learning algorithms in the field of AI [66].

CNNs are highly effective deep learning algorithms that are employed within the specific context of image processing. They are designed to identify features within an image. The aforementioned features are identified through the utilization of filters or matrices, designated as "kernels," at each layer of the network. The filters are equipped with pre-trained weights and are designed to detect specific features, colors, or objects. The sliding of these filters over the image (convolution operation) generates activation maps.

CNNs are capable of detecting features of varying sizes by utilizing filters of differing sizes. The initial layers of this network are in charge of identifying basic features like lines and edges. Subsequent layers are tasked with identifying more complex features, including circles, corners, and objects. The layers mentioned above can further include larger filters,

allowing for the extraction of the broader context of each feature. The most popular application of CNNs is in image classification, but they have been applied to a host of other problems, from object detection and face recognition to image segmentation, video processing, and even natural language processing.

It can be said that CNN is a type of artificial neural network which has been inspired by the human visual cortex. It is capable of achieving effective results, particularly when working with image data. The primary objective of them is to analyze an image and extract its features. In contrast to traditional neural networks, CNNs consider the two-dimensional structure of images and the overlapping properties between them. Consequently, it employs particular layers - like convolution and pooling - to extract image features.

The functioning of CNNs is based on a series of stages, as outlined below [67]:

1) **Introduction:** A CNN is provided with an image to process.
2) **Convolution:** In the first layer, a number of convolutional filters have been applied to the image. These convolution filters are designed to extract specific features from the image.
3) **Pooling:** The second layer does a pooling operation, which has the effect of performing downsampling on the outputs of the convolution. Pooling allows the model to learn more general features.
4) **Full connection:** The final layer transforms the outputs of pooling into an appropriate form for a classification problem.

The core components of CNNs are as follows:

- **Convolutional Layers:** Convolutional layers apply convolution operations to extract feature maps from an image. Convolution involves sliding a filter matrix over the image and performing a dot product to create a new feature map. These layers are designed to capture local image features and identify the existence of different features.
- **Activation Functions:** After convolutional layers, the feature maps that are produced pass through an activation function. Commonly used activation functions include Sigmoid, ReLU and hyperbolic tangent (tanh). These functions introduce non-linearity into the output, providing a measure of the neuron's activation level.
- **Pooling Layers:** Pooling layers are used to reduce the spatial dimensions and computational complexity of feature maps. They typically select the maximum or average values of features using methods like max pooling or average pooling.

- **Fully Connected Layers:** Fully connected layers allow the flattening of feature maps into a single output. These layers are traditional artificial neural network layers used for tasks like regression, classification, or other particular tasks.

CNNs are trained using pre-labeled data. Throughout the training process, the network compares the target outputs with the input images and utilizes the backpropagation algorithm to optimize the weights and learning parameters. Backpropagation enables the weights to be updated by propagating errors in the network in reverse. This and all other algorithms have transformed the domain of deep learning and are employed in a multitude of application domains [68].

Some advantages of CNNs are as follows.

- **Feature-based learning:** CNNs allow the learning of the features from data automatically. Convolutional and pooling layers hierarchically extract significant features from images. This requires less manual tuning and allows for the automatic discovery of more universally applicable features.
- **Scalability:** CNNs can work with different sized images. The convolution operation is resilient to scaling issues due to its application depending on the size of the images. This feature provides an advantage in processing images of varying resolutions and in real-time applications.
- **High Performance:** Training with voluminous data and equipped with abundant resources, CNNs perform very well. The algorithm is very suitable for visual data analysis with complex structures of individual objects and variability.

However, CNNs also have some drawbacks:

- **Data Dependency:** CNNs require both large and diverse datasets. If the amount and/or diversity of the training data are small, it may substantially decrease the performance and can also lead to such issues as bias or overfitting [69].
- **Computational Power:** Since CNNs have deep network structures, they are relatively computationally heavy. Handling large data and complex models requires high memory and processing power. This is time-consuming, sometimes hard for applications based on hardware requirements. Computation times can be reduced using accelerators like GPUs.
- **Data Labeling:** CNNs require typically labeled data, which is time consuming and labor expensive. Accurate labeling often may be cumbersome, particularly for certain complex problems concerning the classification of objects.

- **Position and Scale Invariance:** A certain amount of position and scale invariance is inherent to CNNs because of convolutions and poolings. They may fail when there is a need for the representation of objects in different sizes and perspectives.

In other words, it can be described that CNNs are one of the robust deep models in deep learning, which have been very efficient in problems regarding image processing and pattern recognition [70]. They represent a deep learning algorithm that is widely used in the area of image processing. They are employed to extract features from image data and perform tasks. Nevertheless, it is important to acknowledge the existence of certain limitations, including the necessity for accurate data sets, sufficient computational power, and the availability of appropriate labeling requirements.

In this study, we preferred to use CNNs because they excel in visual data processing through their specialized convolutional and pooling layers, which effectively learn complex features and spatial hierarchies in images.

4.1.2. Transfer learning

Transfer learning is a way in ML that involves the re-use of the model trained for one task or as the starting point of another model in order to be used for some other different task. The phrase transfer learning refers to the use of a pre-trained model on another model. Transfer learning applies knowledge acquired through the solution of one problem to the solution of related but different problems more efficiently. In the traditional ML approaches, models are generally trained from scratch using task-specific labeled datasets. Such an approach, however, requires substantial amounts of data and computational resources. In contrast, transfer learning uses a pre-trained model that already has been trained on a large-scale dataset and then applies it to another model, as shown in Figure 4.1. These models are typically pre-trained on tasks like image classification on huge datasets such as ImageNet, after which they have learned generic features useful for most related tasks [71].

The process of transfer learning involves several key steps. First, a pre-trained model is selected according to its performance on a similar task or domain. Then, the model is adapted or fine-tuned using a smaller dataset designed for the target task. For fine-tuning, some of the layers from a pre-trained model could be frozen to retain the learned features, while others are fine-tuned with respect to the new data. That is what makes this method fast and of good performance with less data than training from scratch. Transfer learning is of great benefit when labeled data is limited or relatively costly to obtain and build models with

high accuracy and efficiency for new applications [72,73].

Its concept took form back in the 1980s and 1990s, specifically from very early neural network research, and gained further traction with the introduction of pre-trained models in the 2000s. Deep learning research in the 2010s accelerated it further by showing how models trained on large datasets could use learned features for transfer to new tasks.

Transfer learning is very useful in some fields. In computer vision, for instance, transfer learning has been effectively used for tasks like object detection, semantic segmentation, and medical image analysis. Similarly, in natural language processing, pre-trained language models like Generative Pre-trained Transformer (GPT) and Bidirectional Encoder Representations from Transformers (BERT) have been fine-tuned for tasks such as text generation, question answering, and sentiment analysis [74,75]. Overall, this technique democratizes access to powerful models and accelerates progress in AI research and application development by facilitating rapid prototyping and deployment of models in diverse real-world scenarios.

Some pre-trained CNN architectures are given as below:

- **VGG (Visual Geometry Group):** A Visual Geometry Group, mainly designed for image classification tasks. There are many convolution layers, followed by fully connected layers in the architectures of VGG16 and VGG19. Those models, pre-trained on large datasets like ImageNet, are taken as a starting point for transfer learning, notably in tasks of object recognition and localization.
- **ResNet (Residual Network):** ResNet architecture introduced residual learning and bridged the vanishing gradient problem in deep networks. Models like ResNet50 and ResNet101 have been pre-trained on the ImageNet dataset and are now used as a benchmark in many applications for transfer learning where tasks involve deeper architectures and better feature extraction and classification ability.
- **GoogLeNet (Inception):** Parallel convolution layers in an inception architecture use different kernel sizes to realize features at different scales. Models are pre-trained on large datasets and thus often fine-tuned for transfer learning in applications like image recognition and medical imaging.
- **MobileNet:** It is specifically designed for mobile and embedded vision applications. It uses depthwise separable convolutions to strike a harmony between accuracy and computational efficiency. Pre-trained MobileNet models are used for many transfer learning processes that are light-weight, such as real-time object

detection and image classification on mobile devices.

- **DenseNet (Densely Connected Convolutional Networks):** DenseNet architecture is connected to every other layer in a feed-forward manner. Models like DenseNet121 and DenseNet169, pre-trained on large datasets, participate in transfer learning for tasks requiring high feature propagation and extraction—tasks related to medical image analysis and pathology detection [76].

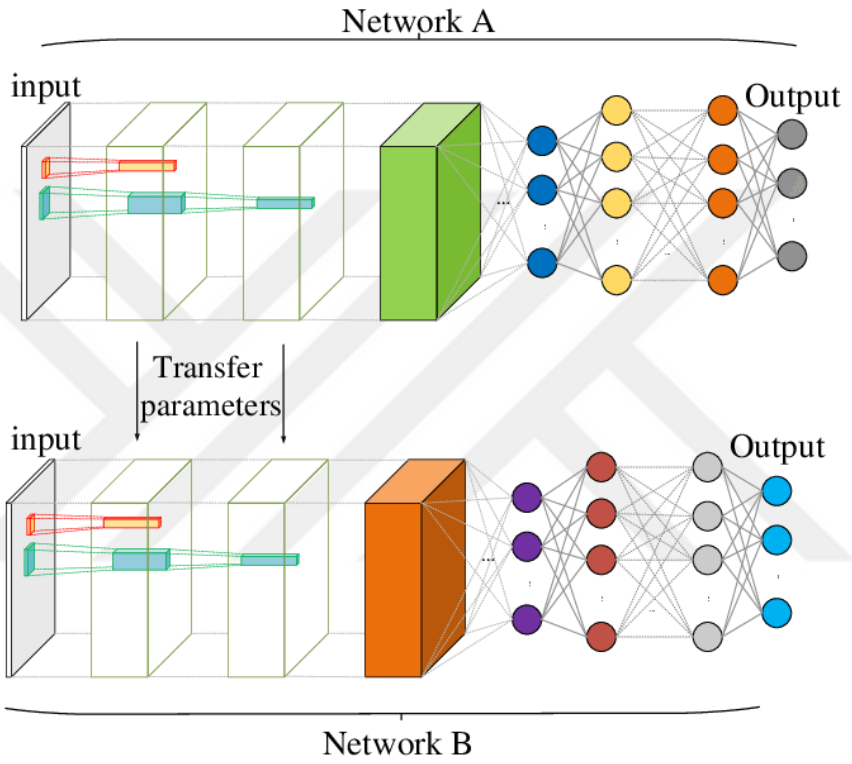


Figure 4.1. Illustration of transfer learning concept [77]

4.1.3. Comparative analysis of custom and pre-trained models

In the realm of modern ML and AI, a model can be either trained from scratch (custom) or fine-tuned using pre-trained models (transfer learning). Although they are both effective in certain ways, there are some differences. While training a model from scratch allows for complete customization and optimization for specific tasks, transfer learning has further enhanced these capabilities by allowing pre-trained models to be adapted for specific tasks, thereby decreasing the requirement for large amounts of labeled data and computing power. [78]. This comparative analysis will discuss concepts, strengths, limitations, and areas of application of the training models from scratch and transfer learning to help gain an all-rounded understanding of what they do in modern AI research and applications.

A comparison between pre-trained and custom models reveals a number of differences and complimentary attributes:

- **Data Requirements:** In order to learn robust feature representations, custom models require massive amounts of annotated data. This is usually a limiting factor in most research domains where data is usually scarce [79]. Transfer learning overcomes this challenge through eligible pre-trained models, hence needing only a small proportion of task-specific data for fine-tuning. This makes it more pragmatic in scenarios constrained by data.
- **Training Time and Computational Resources**: Typically, learning a model from scratch is very computationally heavy and also resource-intensive in terms of time for deep architectures. As a drastically effective way to reduce training time and computational requirements, transfer learning is most suitable for those who do not have higher computational resources.
- **Performance and Generalization:** Custom models on a large dataset often perform well in the exact task that they are trained for but often fail to generalize so well to the other tasks. Very often, the ability of pre-trained models to generalize to new tasks is enhanced by the reuse of learned features from a related task. This normally leads to improved performance in real-world applications (Figure 4.2).
- **Flexibility and Adaptability:** Transfer learning brings forth a flexible framework that can be adapted to many tasks with minimal changing. This is particularly useful in a very fast-evolving field where new tasks and challenges keep appearing. In contrast, the custom approach does not have this kind of flexibility and needs re-engineering efforts to be made when adapting to new tasks.

Custom and transfer learning have been applied to a wide range of domains. For instance, it is well established that custom-designed CNNs are used for various computer vision tasks Transfer learning boosts these capabilities by fine-tuning models on large and pre-trained datasets for particular tasks. Traditionally, CNNs have been used in speech recognition for feature extraction from audio signals; this is where the concept of transfer learning can be applied by fine-tuning models that are pre-trained on big speech datasets to specific tasks like identifying a speaker or emotion recognition [80].

Given all this information, it can be said that using custom models and pre-trained models are distinct approaches that have advanced ML and AI. Custom models that are trained from scratch excel at learning hierarchical features from data, while transfer learning adapts pre-trained models to new tasks, reducing data needs and computational costs while enhancing performance. Understanding strengths and limitations of each approach is essential for effective application. As research progresses, the integration of custom training and transfer learning will keep influencing the course of intelligent systems.

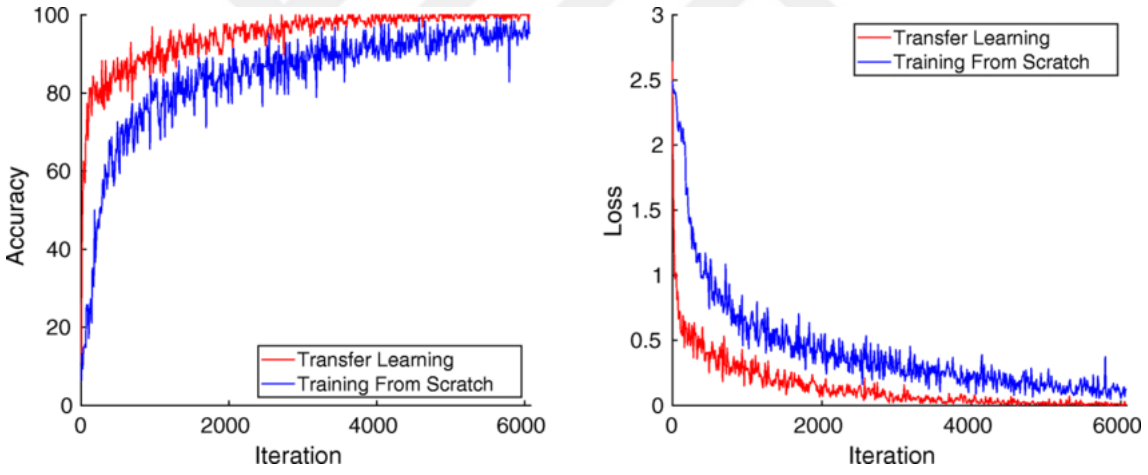


Figure 4.2. Comparison of training accuracy and loss: transfer learning vs. training a network from scratch [81]

In this study, transfer learning methods have been preferred because pre-trained CNN models offer a rich set of general features that enhance performance and reduce training time on new tasks. They provide high accuracy in tasks requiring detailed feature extraction and classification, adapt quickly to new datasets, and have a large number of applications, including object recognition, medical imaging, and more. Their design makes them ideal for transfer learning, allowing efficient adaptation to new tasks while leveraging existing knowledge.

4.1.4. Hyperparameters in optimization

Such techniques that model optimization does to improve the performance of a deep learning model include, but are not limited to, hyperparameter tuning.

Basically, the parameters of the training process for an ML model are referred to as hyperparameters. As opposed to other model parameters learned from the data itself, they are defined before the start of training and remain fixed during the whole process. They have an extremely large impact on the quality of the fit and the generalization capacity of the model.

Some of the important hyperparameters in deep learning are as followed:

- **Learning Rate:** It is the rate at which the weights in the model are adjusted or assigned new values while training. A high learning rate speeds up training with a risk of causing instability. On the other hand, a low learning rate may be stable in adjusting the model but poses higher risks of slower convergence.
- **Batch Size:** It defines the number of training samples after which model weights would be updated. Smaller-sized batches can make the model generalize at the cost of noisier gradient estimates, while larger batch sizes will give more stable estimates but at higher computational costs.
- **Number of Epochs:** It denotes how much time the whole dataset is used to train the models. The model will underfit with too few epochs, whereas there is a possibility of overfitting with too many epochs.
- **Dropout Rate:** The technique of regularization, whereby a given fraction of neuron units are randomly dropped from the training cycle to avoid getting fit. Dropout rate should be tuned to trade-off between the error of overfitting and sacrificing overall model performance.
- **Optimizer:** It is the algorithm that determines the method to update model weights based on gradients. Common optimizers include Adaptive Moment Estimation (Adam), Stochastic Gradient Descent (SGD), and Root Mean Square Propagation (RMSprop). all of which come with hyperparameters that affect training dynamics.
- **Activation Functions:** Non-linearity is introduced to the model by activation functions such as tanh, Sigmoid, and ReLU. The choice of activation function can impact the learning capability of the model.

Hyperparameter tuning is really important in ML, as it significantly improves the performance, as shown in Figure 4.3., how hyperparameter tuning affects various model

performances. This can be done automatically using grid search, random search, Bayesian optimization, or gradient-based optimization, or it can be done manually by experimenting with different sets of hyperparameters and analyzing the results, i.e., using the trial-and-error method [82].

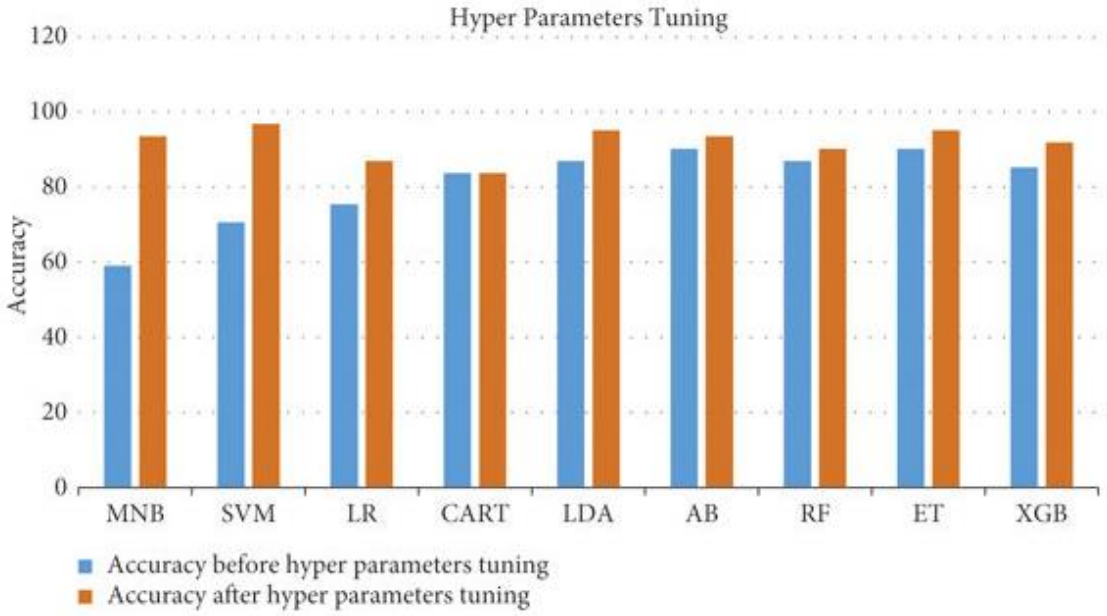


Figure 4.3. Accuracy of classifiers before and after hyperparameter tuning ("MNB": Multinomial Naive Bayes, "SVM": Support Vector Machine, "LR": Logistic Regression, "CART": Classification and Regression Trees, "LDA": Linear Discriminant Analysis, "AB": AdaBoost, "RF": Random Forest. "ET": Extra Trees, "XGB": XGBoost) [83]

In this study, the dataset was self-constructed and the features didn't vary much from class to class, making it difficult to adapt to automated processes. For this reason, manual hyperparameter tuning was used to find the optimal model settings. This process involved adjusting hyperparameters based on empirical results and domain knowledge, requiring careful experimentation. Manual tuning allows for iterative refinement to enhance performance. However, it is important to perform effective tuning, since it is crucial for maximizing model accuracy and robustness, guided by both practical and theoretical insights.

4.2. Overview of Models

The classification logic behind each of the two types of models has been thoroughly explained, including detailed discussions of their respective architectures.

4.2.1. Hand structure classification model

In this model, the hands were classified into four classes based on their structures, using a transfer learning-based model. These classes are: Grounded Hands (H1), Piano Hands (H2), Hi-Five Hands (H3), Generous Hands (H4). The hand images were taken as input and with the proposed model, which is based on a deep learning method, each hand was classified into one of the classes. The input-output diagram of the system is shown in Figure 4.4.

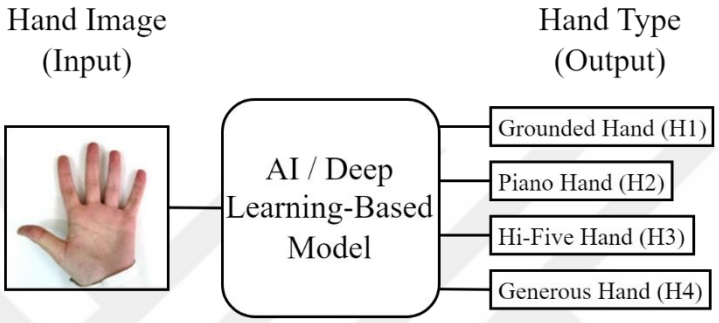


Figure 4.4. Model architecture diagram

Hands are classified based on palm and finger types, with each hand having one palm type and one finger type. First, these types were determined for each hand, and by combining these types, the overall hand type was determined.

There are two types of palms and two types of fingers. The palm types can be classified as either Rectangle Palm (P1), which is characterized by a long palm, or Square Palm (P2), which is defined by a shorter palm. The finger types are: Short Finger (F1) and Long Finger (F2). The criteria for classification are as follows: If a hand's palm type is square and its finger type is short, it is classified as a Grounded Hand. Similarly, if the hand has a square palm and long fingers, it is classified as a Piano Hand. A hand with a rectangle palm and short fingers is classified as a Hi-Five Hand, and one with a rectangle palm and long fingers is classified as a Generous Hand. This classification system is summarized in Figure 4.5. below, along with sample photos for each hand type.

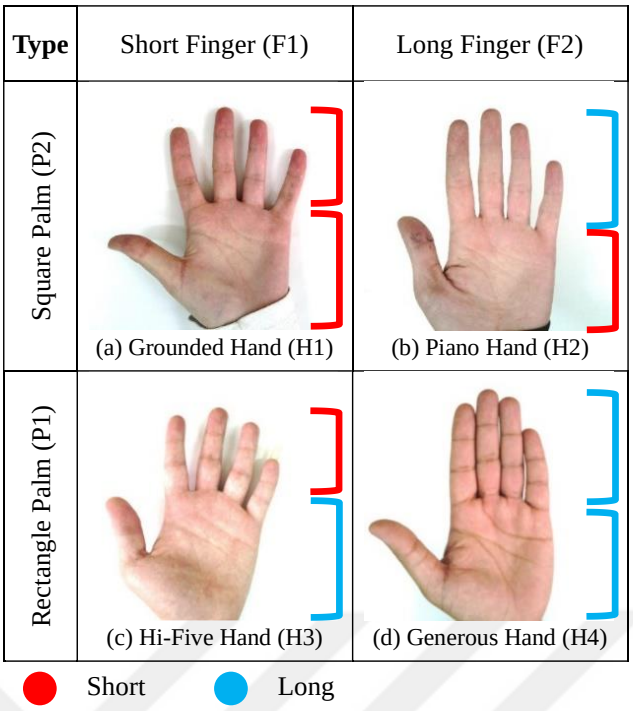


Figure 4.5. Sample images of the four hand types and their classification logic

The terms "short" and "long" used in Figure 7.5. are for convenience only, as relying solely on the vertical length of the palms or the length of the fingers is not sufficient for accurate classification. This is due to the inherent variability in the dimensions of the human hand, which can be affected by various factors, especially gender and ethnicity. Therefore, a comprehensive approach that considers palm and finger dimensions as a whole and uses ratios for more accurate categorization is essential. This is the reason why we proceeded to measure hand features.

4.2.2. Nail type recommendation model

This model is built upon the hand classification model. After classifying hands, the next step was taken: nail recommendation. Through extensive web research and detailed analysis, a total of seven commonly used nail shapes were identified. Subsequently, the determination was made as to which nail shape is suitable for each hand type. Based on comprehensive research, relevant information was gathered, and appropriate nail types were identified for each hand type, recommending two nail types for each. Given that two types of nails are recommended for each hand type, they were categorized into four groups as follows: Oval Nail & Round Nail (N1), Almond Nail & Oval Nail (N2) Coffin/Ballerina Nail & Stiletto Nail (N3), Square Nail & Squoval Nail (N4). The model architecture diagram in

Figure 4.6. shows the recommended nail types for each hand type, with each hand and its recommended nail type represented by the same color.

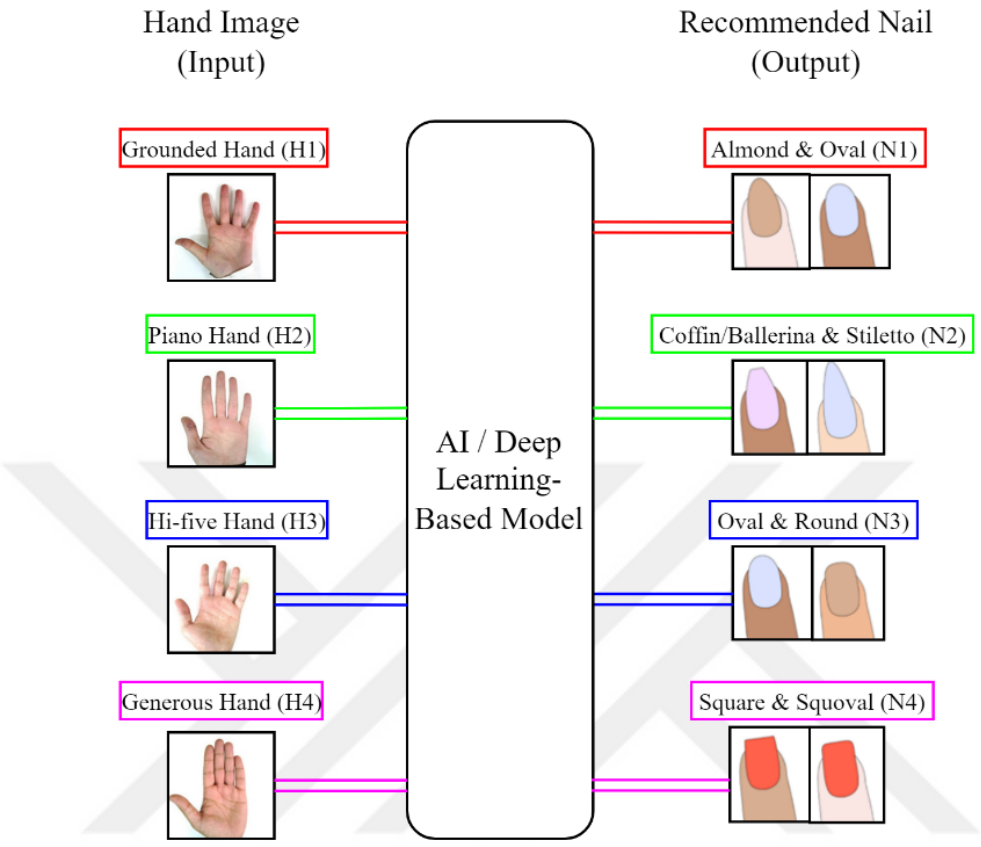


Figure 4.6. Mapping from hand type to nail type

4.3. Dataset

In this study, due to the absence of a suitable dataset for hand structure classification, a dataset was constructed using an automated method. Also, the images in the dataset were appropriately data-augmented to increase the accuracy of the dataset.

4.3.1. Model architecture and data augmentation

A dataset comprising 2050 hand images was used for classification. The dataset was split into train, validation, and test sets with a ratio of 3:1:1. Data augmentation included cropping to a 1:1 ratio, horizontal and vertical flipping, and rotations of 90°, 180°, or 270° clockwise to enhance model robustness and generalization [84]. The images were cropped manually to prevent the hand part of the images being cropped, ensuring optimal results. Furthermore, the images are manually flipped and rotated in order to ensure an even

distribution of augmentation. Images were resized to 224x224 using ImageDataGenerator for compatibility with the pretrained VGG16 model. After augmentation, feature extraction was performed with VGG16, followed by training and validation for hand classification. Finally, performance was measured on the test set to conclude the process. The flowchart detailing the overall process and model is provided in Figure 4.7.

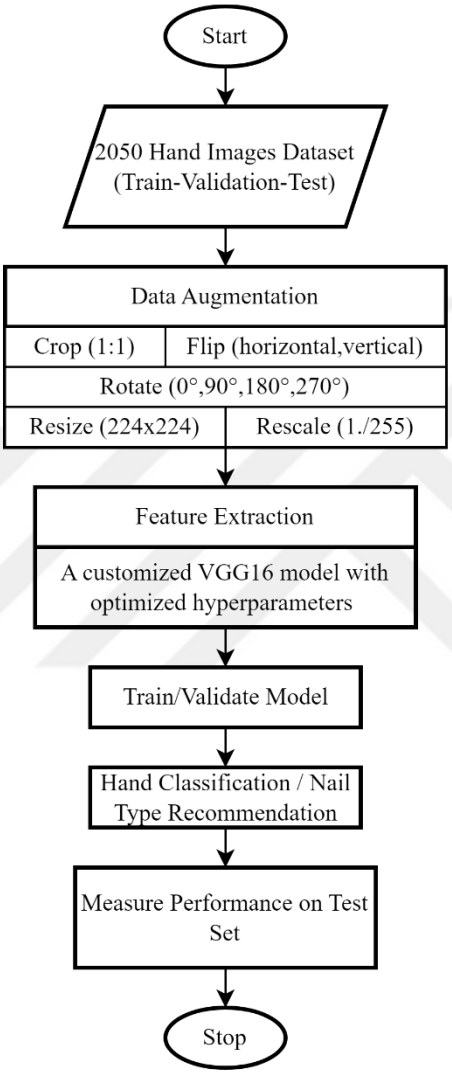


Figure 4.7. Flowchart of the proposed model

Overall, a comprehensive dataset of hand images was constructed. A custom script was developed to automatically split the random images into four class folders. Then, deep learning techniques were used to analyze the dataset to classify hand types and recommend suitable nail shapes. This method streamlined and accelerated the data preparation process and improved the accuracy of the classification model.

4.3.2. Used libraries

Various libraries were downloaded for the purpose of implementing the hand feature measurement method and for the automatic creation of the dataset. These tasks, along with the classification process, were executed using the Python programming language within the Google Colab environment.

- **MediaPipe:** MediaPipe is a cross-platform framework that was developed by Google and built for the multimodal ML pipeline and, specifically, for computer vision and perception. Proposed in 2019, MediaPipe allows real-time processing on visual and audio data. In particular, it will be optimal for applications such as hand tracking, face detection, pose estimation, and object detection [85,86]. It provides speed-optimized, very accurate pre-trained models fit for mobile and web use cases. MediaPipe has a modular architecture, so it is easy to customize by bringing together different building blocks, such as detectors and trackers. It is also language-agnostic, supporting languages like Python, C++, and JavaScript, which run on Android, iOS, and desktop environments. Specifically, one of MediaPipe's modules is its Hand Tracking Module, which makes real-time hand detection and gesture recognition possible.
  - **MediaPipe Hand Tracking Module:** This module can detect in real time and track hands, letting applications recognize gestures and movements of hands. It identities hands with landmark estimation and then recognizes key points on the hands like the fingertips and joints through a lightweight neural network. Strong in real-time performance, it processes video streams from webcams or mobile cameras and can thus be used on a wide range of platforms. It is most commonly applied in gesture controls in games, sign language recognition, and virtual reality and interactive installations, providing advanced hand tracking for developers.
- **OpenCV (Open Source Computer Vision Library):** OpenCV is an open-source library developed by Intel in 2000 for computer vision and ML. The solution evolved into one of the main tools in the area of image and video processing, proposing all types of imagery features related to capture, manipulation, and processing. OpenCV confirms the different formats and integrates with ML frameworks like TensorFlow and PyTorch to make intelligent visual data analysis easy. This library is compatible with Python, C++, and Java programming

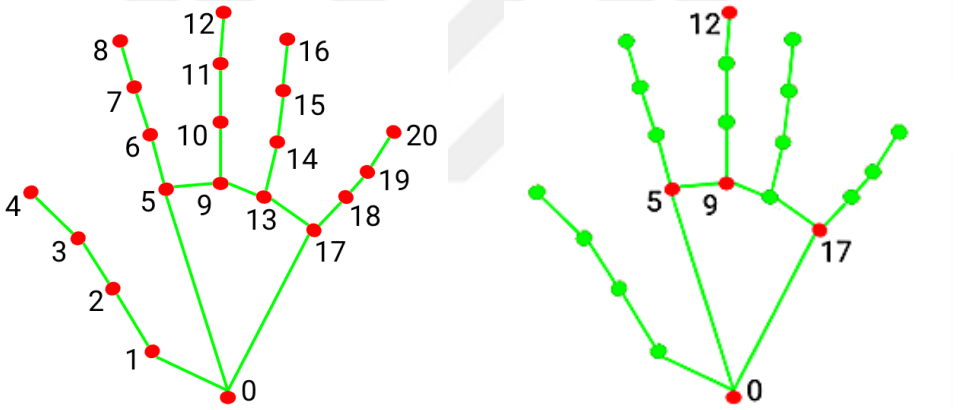
languages; also, it works on different operating systems and mobile platforms [87]. Some of the major functions it supports are reading and displaying images (cv2.imread()), displaying (cv2.imshow()), video capturing (cv2.VideoCapture()), and resizing (cv2.resize()), making OpenCV a great all-rounder for developers and researchers in the field of computer vision.

This study also made extensive use of Python's standard libraries. These libraries support a great part of suites of modules and packages that provide standard solutions for the many common tasks of programming, such as I/O on file, interacting with the system, data manipulation, and mathematical operations, therefore assisting the developer to write an effective and powerful application with minimal effort. Three Python libraries were used in our code:

- **Random:** The "random" module supports essential operations of generating random numbers and executing random actions. It contains various handy functions to generate random integers, floating-point numbers, and random sequences. It allows shuffling and choice of data, too. These considerations are many times applicable in many contexts like simulation and random sampling, algorithm testing, and so on.
- **Math:** This module supports numerous mathematical constants and functions — from elementary mathematics learned in high school to trigonometry, some logarithmic functions, and a wide range of exponents. It harbors some of the key tools required for complex calculations and transformations, including basic constants like π (pi) or e. For geometric calculations, statistical analysis, or accurate mathematical operations, the "math" module is bound to deliver meticulous and effective results.
- **Os:** The "os" module permits interaction with the operating system, furnishing functions to perform manipulations on files and directories, work with environment variables, and deal with process management. It provides tools for file system navigation and handling environment settings: It prepares one for the execution of commands in the system, an integral part of the arsenal of how to manage system-level operations within Python scripts.

4.3.3. Hands features measurement method

In this study, a new dataset was constructed due to the lack of existing datasets suitable for hand type classification. The aim was to create the dataset and distinguish the images from each other, with the aim of separating the images into class folders. However, determining the hand class without the use of measurements is susceptible to error, as some hands were challenging to classify without specific measurements. In order to achieve this, the MediaPipe Hand Tracking module that was developed by Google was employed. This employs ML to achieve real-time hand and finger tracking. As depicted in Figure 4.8. and Figure 4.9., this module detects and tracks 21 landmarks across both the palmar and dorsal sides of the hand, enabling precise hand gesture recognition and analysis [88]. Notably, by displaying the hand joints, the module can be utilized to measure hand dimensions and consequently detect hand shapes. This functionality forms the basis of our study's use of the module.



(a) (b)

Figure 4.8. (a) MediaPipe hand landmarks (b) Used hand landmarks (Adapted from Alyami et al., [89])

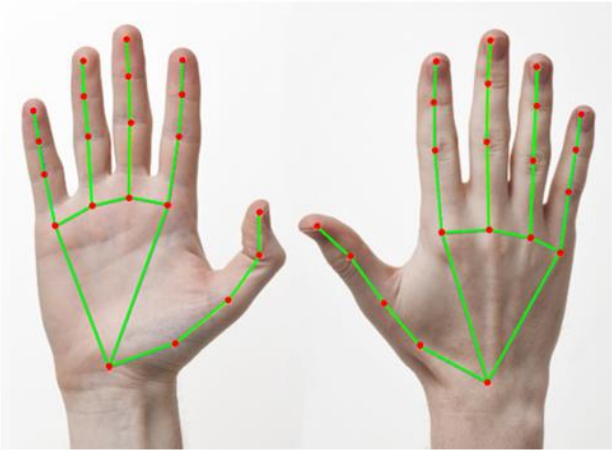


Figure 4.9. Palmar and dorsal sides of hand as seen in the module (Adapted from [90])

A total of 3 hand dimensions were measured with the help of the landmarks—only four landmarks have been used (Figure 4.8.)—that were placed on the images by the module. The dimensions are measured for both dorsal and palmar sided images. D1 - length of palm (vertical distance from center of wrist crease to starting joint of the middle finger, distance between landmarks 0 - 9), D2 - width of palm (horizontal distance from center of far right and left points of the palm, distance between landmarks 5 - 17), D3 - length of fingers (vertical distance from starting joint of middle finger to tip of the middle finger, distance between landmarks 9 - 12). With these measurements, we get 2 palm and 2 finger types. These are classified as P1 (Rectangle Palm), P2 (Square Palm), F1 (Short Finger), F2 (Long Finger). To determine the palm type, we use D1/D2 ratio. If $D1/D2 \geq 1.400$, the palm type is P1, else it is P2. To determine the finger type, we use D1/D3 ratio. If $D1/D3 \geq 0.980$, the finger type is F1, else it is F2. After determining the palm and finger type of the hand - there is 1 palm and 1 finger type of each hand -, we use these factors to determine the overall hand type. To determine the hand type, we use AND logic. If palm and finger types are determined as P2 & F1, the hand type is H1, if types are P2 & F2, the hand type is H2, if types are P1 & F1, the hand type is H3, and finally, if types are P1 & F2, the hand type is H4. The measured and used palmar side hand dimensions on module can be seen in Figure 4.10.

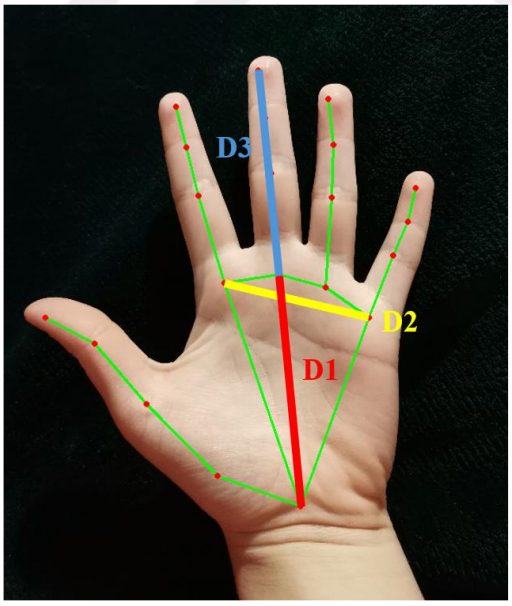


Figure 4.10. Measured hand dimensions by using MediaPipe hand tracking module (Adapted from [91])

Prior to the construction of the dataset, the script created all four class folders. Once the images' hand types have been determined, they are copied into their corresponding folders. Subsequently, the same procedure was repeated for the remaining hand images.

Consequently, all images are automatically sorted into four classes. Finally, the train, validation and test folders were created manually, with all four class folders under them. The classified images were then distributed into the folders in a ratio of 3:1:1. The overall process is illustrated in Figure 4.11., which also provides a pseudocode for the procedure.

```
For each hand:
    Get landmark coordinates
    Calculate hand dimensions
    If D1/D2 ratio ≥ 1.400:
        Palm type is Rectangle Palm
    Else:
        Palm type is Square Palm
    If D1/D3 ratio ≥ 0.980:
        Finger type is Short Finger
    Else:
        Finger type is Long Finger
    If palm type is Square Palm && finger type is Short Finger:
        Hand type is Grounded Hand
    Else if palm type is Square Palm && finger type is Long Finger:
        Hand type is Piano Hand
    Else if palm type is Rectangle Palm && finger type is Short Finger:
        Hand type is Hi-five Hand
    Else if palm type is Rectangle Palm && finger type is Long Finger:
        Hand type is Generous Hand
Copy image to hand type folder
    If hand type is Grounded Hand:
        Copy image to grounded folder
    Else if hand type is Piano Hand:
        Copy image to piano folder
    Else if hand type is Hi-five Hand:
        Copy image to hifive folder
    Else if hand type is Generous Hand:
        Copy image to generous folder
End
```

Figure 4.11. Pseudocode for the dataset construction script

4.3.4. Organization of the dataset

This study uses a dataset that is a combination of ”11k Hands” [92], ”Hand Class Image Dataset” datasets and photos that are collected from Google. Dataset consists of a total of 2050 annotated hand images (1200x1200) of approximately 60-80 subjects, including both male and female participants, varying between the ages of 18-75 years old. For each subject, photographs were taken of the right and left hands with fingers closed and open. Each hand was photographed from both the dorsal and palmar sides with a uniform white background

and placed approximately in the same distance from the camera.

Each photo has been annotated using folder-based labeling and each folder has been named as hand structure / nail type class names. The images in the folders have been shuffled by changing the file names with random numbers, making it possible for the dataset to be used without the need of shuffling, which eliminates extra steps and reduces variability in results. Then images were distributed in folders in certain proportions according to 3:1:1 ratio (training folder:validation folder:test folder), making it consisting from approximately 60% training, 20% validation and 20% test set. Number of photos in each dataset folder and their distribution, along with the directory architecture has been given in Table 4.1. and Figure 4.12.

Table 4.1. Class distribution across dataset folders

Class	**Train**	**Validation**	**Test**	**Total**
H1 / N1	306	111	107	524
H2 / N2	311	111	111	533
H3 / N3	300	99	98	497
H4 / N4	298	100	98	496
# of Total Images	**1215**	**421**	**414**	**2050**
Percentage	**59.27%**	**20.54%**	**20.19%**	**100%**

H1: Grounded, H2: Piano, H3: Hi-Five, H4: Generous
N1: Almond&Oval, N2: Coffin/Ballerina&Stiletto, N3: Oval&Round, N4: Square&Squoval

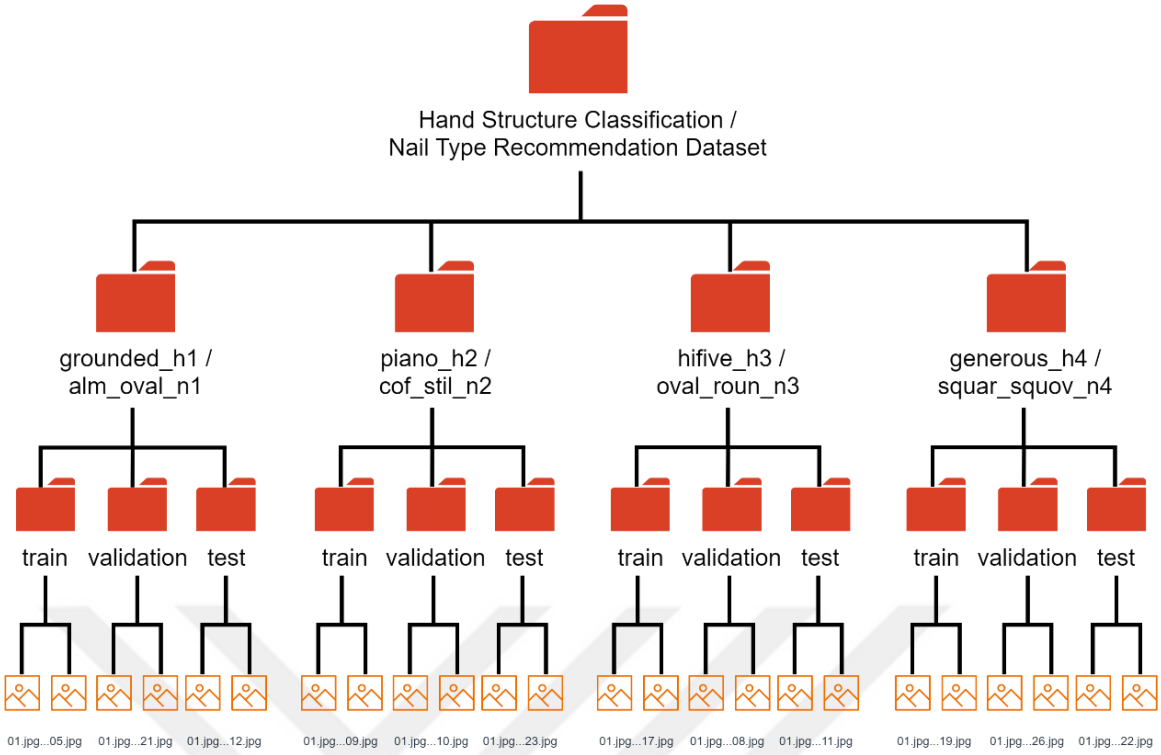


Figure 4.12. The directory architecture of the dataset

To clarify the file naming conventions used in the dataset, it is important to understand that the files are categorized into two distinct names due to the dual classification purposes within the same dataset. The reason for this naming convention is that both classifications use the same dataset, but with different folder names. Specifically, folders labelled “h” were used for hand structure classification, while folders labelled “n” were used for nail type recommendation.

It is also important to note that the majority of the dataset is derived from the “11k Hands” dataset, as the reason was that the images were the most suitable for the classification task at hand. In order to introduce variability, the “Hand Class Dataset” was also used, and images from Google were included in order to achieve a balanced number of images across different classes.

4.4. Performance Evaluation Methods

Performance measurement functions drive the performance evaluation of a CNN on different aspects of the computer vision tasks, such as image classification, object detection, and segmentation. The metrics provide quantitative assessments in regard to model performance, giving insights into how accurate and effective CNN-based solutions are. The major performance measures to evaluate a CNN involve learning curves and confusion matrices.

4.4.1. Confusion matrix

A confusion matrix is a summary of prediction results against actual classifications for a classification problem. It represents the performance of classification model components with the consideration of each class independently, through specific values in the matrix. The logic of the matrix is based on a certain rule. In this matrix, one class is chosen as a "positive" class, making other classes "negative." Accordingly, values are obtained which are: the number of True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN). The confusion matrix is the tool that presents all these values in a tabular format, as shown in Figure 4.13. These values are defined below:

- **True Positive (TP):** The number of instances belonging to a particular category that the model correctly predicts (e.g., correctly identifying a particular class).
- **True Negative (TN):** The number of other categories that the model predicts correctly not to be the specific category (e.g., correctly identifying other classes as not being the specific class).
- **False Positive (FP):** The number of times the model predicts another category as that particular category (e.g., predicting another class as the specific class).
- **False Negative (FN):** The number of times the model predicts a special category as another one incorrectly (e.g., predicting the specific class as a different class).

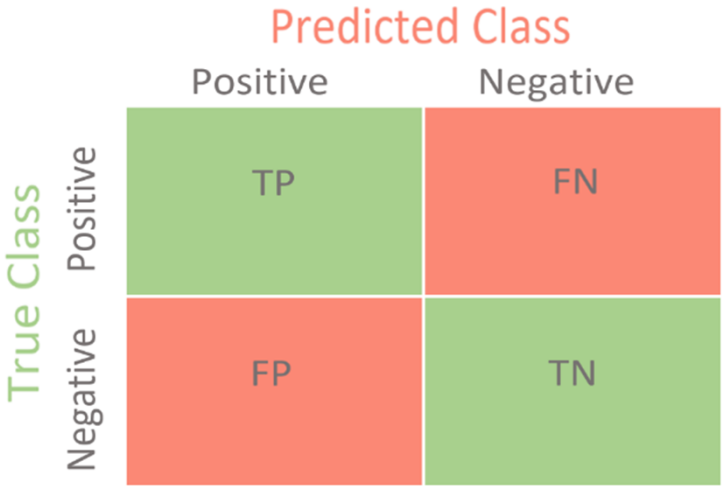


Figure 4.13. Confusion matrix for binary classification (Adapted from Bilgin and Altınışık, [93])

In general, the explanation of a confusion matrix is as explained above. However, it gets a little bit complicated since confusion matrices change as the label / class number change. So, classification type must be taken into account. There are two types of classification: binary and multi-class. Binary classifications consist from two classes only, whereas multi-class classification consists from more than two classes. This makes binary classification matrix 2x2, and multi-class confusion matrix 3x3 or 4x4, etc.

In literature, multi-class classification is sometimes confusing, since individuals generally use another calculation methods for multi-class studies, and there are even methods that developed just for this topic [94]. So, for this, the values on the matrix are showed a bit differently, as in Figure 4.14.

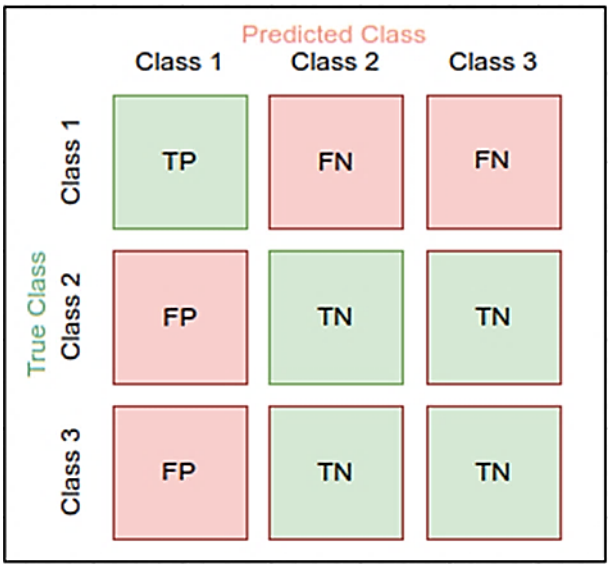


Figure 4.14. Confusion matrix for multi-class classification (Adapted from Pagano et al., [95])

Differently from the binary confusion matrix, multi-class matrices consist from more than four values. And for this, a slightly different approach is needed to calculate the values from the matrix, which will be discussed in the following paragraph.

The situation in which the values are computed for Class 1 is depicted in Figure 4.14. above. Here, TP represents the correct predictions matching the target, whereas TN corresponds to the total number (sum) of classes that do not include Class 1 in any way, neither in the prediction nor in the target. FP is the total number of all data that are incorrectly predicted as Class 1, while FN represents the total number of all data that were misclassified as other classes, which ought to have been predicted as Class 1 [95]. After this, same steps should be carried out for all other classes, since these values change for each class, as the positive value changes.

There are also metrics that are obtained from the confusion matrix, such as specificity (TN rate), sensitivity (TP rate), and overall accuracy, which offer insights into the nuances of model performance. They are calculated individually for each class, evaluating classes' performance one by one. So, we can say that these metrics aid in understanding the accuracy of the model's predictions and its ability to recognize different classes. Some key metrics and their corresponding formulas are listed below:

- **Accuracy:** Measures the overall correctness of the model. It can be obtained by the ratio of all that is correct to the total number of predictions, as shown in equation (8.1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

- **Precision (Positive Predictive Value):** Indicates the positivity of the positive prediction. It tells how many of the positive predictions are actually positive. The formula in (8.2) can be used to compute it.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

- **Recall (Sensitivity, True Positive Rate):** The measure that determines how well the model identifies all the positive instances, meaning how many of the actual positives the model predicts are asserted by the model. It can be calculated via (8.3).

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

- **Specificity (True Negative Rate):** Indicates how well the model identifies all negative instances, meaning how many of the actual negatives the model predicts are asserted by the model. It can be calculated via (8.4).

$$Specificity = \frac{TN}{TN + FP} \tag{4.4}$$

- **F1 Score:** The single value that provides the balancing of both precision and recall. It attempts to provide an overall performance of a model by considering these two aspects. It can be computed using the formula (8.5).

$$F1\ Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.5}$$

Even though these are the main metrics that are used, for a better and clearer understanding of each class, we will use only three of them, which are precision, recall and F1 score.

These metrics can also be averaged with different methods, like micro-averaging, macro-averaging, and weighted-averaging, which are particularly useful in the multi-class classification problems, especially in imbalanced datasets, to aggregate performance across multiple classes. They can be described as follows:

- **Micro-averaging:** Aggregates metrics by treating each prediction equally, making it ideal for evaluating overall performance across all instances without considering class distribution. It can be calculated by the formula (8.6) as followed:

$$M_{micro} = \frac{\sum_{i=1}^{N} C_i}{\sum_{i=1}^{N} D_i} \tag{4.6}$$

where N is the total number of classes, i is the class index, M is the value of the metric to which this method is to be applied, C is the components in the numerator,

D is the components in the denominator of the relevant metric's formula.

- **Macro-averaging:** Calculates metrics for each class separately and averages them, ensuring each class is equally considered regardless of its size. It can be calculated by the formula (8.7) as followed:

$$M_{macro} = \frac{1}{N}\sum_{i=1}^{N} M_i \tag{4.7}$$

- **Weighted-averaging:** adjusts the average metric based on the number of instances per class, balancing the influence of each class according to its frequency, which is useful in imbalanced datasets. It can be calculated by the formula (8.8) as followed:

$$M_{weighted} = \sum_{i=1}^{N} w_i * M_i \tag{4.8}$$

where w is the number of instances, meaning the number of total data points (e.g. images) for that specific class.

Since our dataset is evenly distributed across classes, using micro-averaging is particularly useful for evaluating performance, as it treats each prediction equally and provides an overall view of the model's effectiveness. Even though macro and weighted-averaging are less critical in this context, we will include them to have more understanding of how well the model does for all classes individually.

4.4.2. Learning curves

Learning curves, the collective name for accuracy and loss curves, are some of the key model performance assessment tools. Accuracy curves show how good models are on training and validation data over time, whereas loss curves show measures of the model with respect to its error on the same datasets. In this paper, both training and validation accuracy and loss curves were employed to ascertain whether a model was learning or generalizing properly.

- **Accuracy Curve:** This graph typically has two important elements: training accuracy and validation accuracy. Training accuracy is how well the model performs on the training data at every epoch. It is indicative of how well the model is actually learning from the data being seen. On the contrary, validation accuracy

returns how good the model is at each epoch against new, unseen validation data; it generally informs about the model's ability to generalize with unknown instances. Typically, the number of epochs indicating the iterations in training is on the x-axis, and the model's performance over time is revealed on the y-axis.

- **Loss Curve:** It is essentially like the accuracy curve with its two components: one being training loss and the other being validation loss. In other words, training loss measures the model error in training data at each epoch, telling how well the model is fitting the seen data. On the other hand, validation loss is model error to the validation data at every epoch; hence, it describes how well the model generalizes to new, unseen data. The loss curve is usually plotted against the epochs on the x-axis, which indicate training progression—a measure of how far the model is in the training process against the error on the y-axis.

Even though relevant curves provide additional insights into model performance and generalization, interpreting the results from validation and training curves can sometimes be challenging. Therefore, it is advisable to keep the following points in mind when examining these curves:

- It is optimal for both curves to exhibit a high degree of similarity.
- If both scores are low, it means the model is most likely underfitting. This means either the model is too simple, or it is propped up by an insufficient number of features. It could also mean that the model is being too over-regularized.
- If the training curve obtains a high score rather rapidly and the validation curve lags behind, the model is probably overfitting. This indicates that the model is very complex, yet there is far less data than there should be, leading the model to overly learn with the little data it has. Other than that, it could just mean there is insufficient data.
- The goal is to determine the parameter value that produces the closest alignment between the training and validation curves [96].

The validation and training loss curves can be employed to identify an optimal bias/variance tradeoff. As illustrated in Figure 4.15., the training process should be terminated when the validation error trend transitions from a descending to an ascending trajectory. Terminating the process prior to this point will result in the model being underfitted, whereas terminating it after this point will result in the model exhibiting overfitting [97].

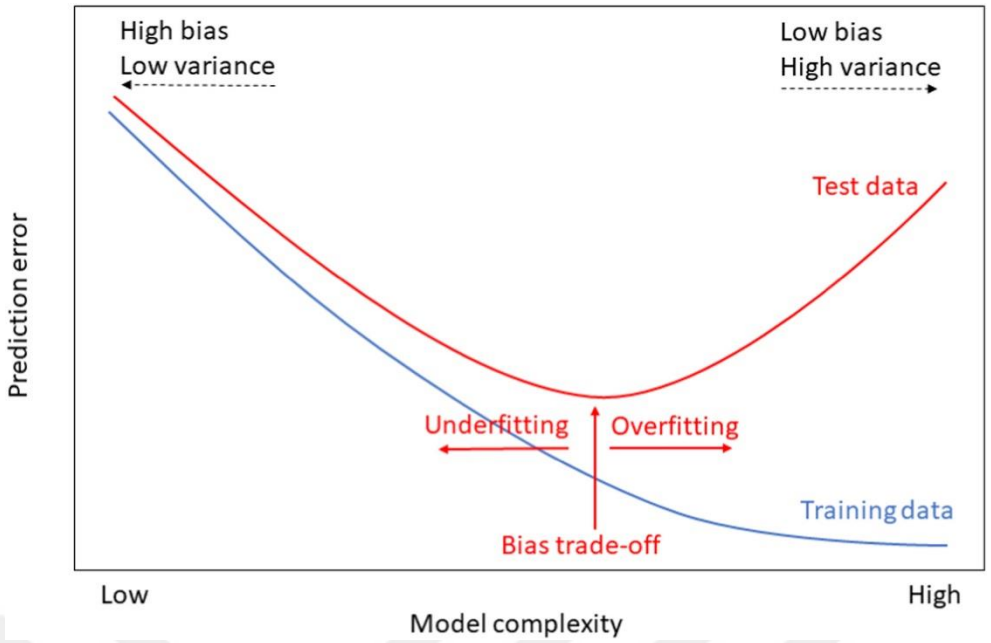


Figure 4.15. Learning curves for model fit analysis [98]

Similarly, the training process should be monitored using the accuracy curve. The training should be terminated when the validation accuracy trend ceases to improve and begins to decline. If the process is terminated before this point, the model is likely to be underfitted, as it has not learned enough from the data. On the other hand, if the training is allowed to continue after this point, the model will start to overfit, memorizing the training data instead of generalizing from it. This careful monitoring ensures the model is neither prematurely stopped nor excessively trained, maintaining a balance between underfitting and overfitting.

4.5. Development Tools and Environment

In the development of this study, the main environment used was Google Colab due to its support for collaborative work and better integration with various ML tools. Python was used as the programming language since it is multi-purpose with extensive libraries that are well-fitted for working with data analysis and ML tasks.

4.5.1. Google Colab (Colaboratory)

Colaboratory, commonly referred to as Google Colab, is a free cloud-based interactive computing platform using Python, offered by Google. This tool gives a Jupyter notebook environment that makes it possible for users to directly write and execute Python code within their web browsers. This makes it a cloud-based nature that does not require any software

installation on the user's end, since everything is run and processed on Google's servers [99].

ne of the striking features of Google Colab is that it is highly integrated with Google Drive. This allows a user to save and load their notebooks and data from their Google Drive account. Further, Colab graciously exposes hardware accelerators like GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units), which significantly speed up the training of ML models and other computationally intensive tasks [100].

Colab has many of the most popular Python libraries pre-installed, such as TensorFlow, Keras, and Scikit-learn, which are currently widely used in deep learning applications [101]. This basically means that a user does not need to install these libraries. Again, this eases the development process and allows it to be done at a relatively faster rate. In addition to the prebuilt libraries, one can install any other libraries to use in this environment. It further extends the flexibility offered by Colab.

One of the salient features is real-time collaboration, which allows several users to share the same notebook and work on it at the same time. This feature is very much like Google Docs and facilitates collaboration with others on data science and ML projects. One has control over the sharing permissions, thereby managing who has permission to view or edit the notebooks.

It is owing to all these features that Colab becomes a powerful and versatile tool for data science and ML. Particularly, its cloud environment with access to high-performance hardware and very nicely integrated libraries makes it an ideal choice for individual researchers and collaborative teams to use for their projects.

4.5.2. Python programming language

Python is a high-level, multi-purpose programming language known for its simplicity, readability, and overall applicability to many fields. Developed by Guido van Rossum and released in 1991, and because of its simplicity and syntax, its learning curve is so smooth that it grew to be one of the most popular languages. It is designed as a general-purpose language, applicable for a very wide scope of tasks—web development, scientific computing, data analysis, AI, and automation [102].

One of the interesting features of Python as a language is that it is interpreted. Contrary to languages directly compiled into machine code, Python code is interpreted one line at a time, thus allowing an interactive and iterative development cycle. This is especially useful in rapid prototyping and testing of ideas [103]. Syntax is chosen to emphasize readability by

using indentation to define code blocks instead of depending on curly braces or keywords. This is an important aspect of this language: the design decision to avoid clutter in code but also to make Python programs easier to understand and maintain.

Python is dynamically typed, and hence variables need not declare the explicit type of data. This flexibility frees the developer from more 'problem-solving' rather than 'datatype management,' increasing productivity and clarity of code. The language standard library is rich, with modules and packages for common tasks such as file I/O, networking, and web services. Furthermore, it is the case that Python hosts an extended third-party library base and frameworks that are continuously growing due to active communities, further creating their power in specialized applications.

It supports procedural, object-oriented, and functional programming styles. Such capabilities give a developer options to select the best technique during the development of the project for code reuse and scalability. Its portability to other operating systems—like Wide Interactive Network Development for Office Work Solution (Windows), Macintosh Operating System (MacOS), Lovable Intellect Not Using XP (Linux), and Uniplexed Information Computing System (Unix)—means that, without modification, Python applications can be widely deployed.

Being an open-source language, Python is developed under the supervision of the Python Software Foundation (PSF) with developers and enthusiasts from all around the globe. Such a collaborative model engenders constant improvement, strong documentation, and community support—something very suitable for both beginners learning to program and senior developers who address complex issues within different industries.

4.5.3. Implemented libraries

Many libraries were used in this study, some of which are already available in the Python ecosystem and some of which were provided by Google.

- **NumPy (Numerical Python):** The package for scientific computing with Python. It supports large, multi-dimensional arrays and matrices, besides providing a wide and extensive collection of high-level mathematical functions to operate on these arrays. Notably among the features provided in this library is an N-dimensional array object for which mathematical operations may be performed easily. It also has elemental-wise operations and linear algebra functions, hence forming a basis for data manipulation and analysis. In addition, it has different tools that enable its

integration with low-level languages such as C and the Formula Translation (Fortran) to improve performance and flexibility [104]. Besides, NumPy has facilities for generating random numbers and performing Fourier transformations, thus making this library really one of the cornerstones of most science computation packages.

- **TensorFlow:** TensorFlow is a popular, open-source ML framework developed at Google for the development and deployment of ML models, more so in deep learning [105]. Deep learning and neural network models are some of its main features, thanks to a flexible architecture that enables any developer to construct complicated systems easily. Tighter integration with Keras, a high-level neural network API, empowers ease of model building and experimentation for both beginners and advanced users, thus making it more useful for people at all experience levels [106]. TensorFlow includes pre-trained models and tools to perform transfer learning. It contains essential tools for deploying models into production, guaranteeing their scalability and reliability. It also provides good integration with many other tools and services provided by Google, thus expanding the functionality. Some of the main submodules used within TensorFlow are:
  - **tensorflow.keras:** Offers a high-level Application Programming Interface (API) for neural networks.
  - **tensorflow.keras.layers:** Defines neural network layers.
  - **tensorflow.keras.models:** Includes functions to create and manage models.
  - **tensorflow.keras.optimizers:** Implements optimization algorithms.
  - **tensorflow.keras.applications.VGG16:** Utilizes a pre-trained VGG16 model for transfer learning.
  - **tensorflow.keras.preprocessing.image.ImageDataGenerator:** Performs image preprocessing and augmentation.
- **Scikit-Learn:** Scikit-learn is one of the popular ML libraries in Python aimed at providing simple and efficient tools for data mining and data analysis. It is based on NumPy, SciPy, and Matplotlib; it has an extensive collection of algorithms for tasks such as classification, regression, clustering, and dimensionality reduction with basic utilities to perform model selection and model performance assessment [107]. Scikit-learn is also equipped with relevant utilities in model selection and model performance assessment. Moreover, the library provides the functionality for preprocessing and transforming data that is necessary for the preparation of

datasets for ML tasks. One of the useful modules applied in this domain is "sklearn.metrics.confusion_matrix"—it calculates the confusion matrix, a very useful tool while one is assessing the quality of classification models.

- **Matplotlib:** Matplotlib is among the widest libraries in Python for generating static, animated, and interactive visualizations. It's broadly applied in generating most kinds of graphical representations of data, such as plots, histograms, and bar charts. Among its major points is that there are a lot of plot kinds and options that can be tuned to fit visualizations into explicit needs. It integrates well with NumPy for array data plotting. In addition, it has different output formats, like Portable Network Graphic (PNG), Portable Document Format (PDF), and interactive plots, which can be versatile for a wide variety of applications. The "matplotlib.pyplot" module provides a state-based interface to simplify plot and visualization creation, improving the user experience for developers working in data visualization.
- **Seaborn:** Seaborn is a statistical data visualization library based on Matplotlib that offers a high-level interface for the creation of attractive and informative statistical graphics [108]. One of the core features of this library is simplifying the process of creating complex visualization in a way that it is available even for a beginner. Seaborn integrates well with Pandas data structures, enhancing its functionality toward data analysis tasks. The library also incorporated some built-in themes that improved the look and feel of the graphics, thus making it easy for any user to create appealing graphics. Besides the above, Seaborn also has methods for visualizing univariate and bivariate data, linear regression models, and statistical time series—a wide range of analytical needs. Another commonly used module is "seaborn.heatmap," which generates heatmaps. Such plots are very useful in visualizing confusion matrices or any other form of matrix data.

5. THE RESEARCH FINDINGS AND DISCUSSION

In this study, the performances of CNN-based architectures such as VGG16, VGG19, Le-Net5, AlexNet, GoogLeNet, ResNet, DenseNet, and MobileNet were compared for hand type classification from hand images. The study was conducted using the Google Colab platform and the Python programming language.

5.1. Structure of the Proposed Model

Training a model in CNN, like other tasks, also requires some key steps to be followed with full understanding of the architecture and the data. CNNs are one class among various deep-learning algorithms developed specially to be effective for grid-like data, such as images. The architecture of a CNN is designed to automatically learn spatial hierarchies of features, making it quite suitable for image recognition and classification tasks. In this section, the structure of the proposed model that is based on the VGG16 architecture, will be examined.

The first stage is to prepare data, which means collecting, cleaning, and preprocessing the dataset. This usually involves normalization of input numerical data to be on the same scale in pixel values, which are normally between 0 and 1, and possibly data augmentation, wherein the data is changed to prepare a model that is more robust to changes in input data.

After the data is ready, the CNN architecture is defined. A typical CNN will include three types of layers: convolutional layers, pooling layers, and fully connected layers. Since transfer learning was applied, the base model was pre-trained and it already contained predefined layers. Other than that, extra fully connected layers were added to make the performance of the model more robust. For instance, the overall structure of the model is shown in Figure 5.1.

The first two layers, convolution and pooling, were used for feature extraction, and the third layer, fully connected, mapped those features onto the final output, which means classification in this case [109]. To get an idea of what they really do, one can take a closer look: Convolutional layers convolve the input image with a set of filters to produce a feature map, which captures a different aspect of the image. Any single filter in a convolutional layer can detect edges, textures, and shapes—that is, one type of feature. Pooling layers usually serve the purpose of reducing the dimensionality of these feature maps and preserving important information while keeping computational complexity and the risk of overfitting as

low as possible. At the end of a network, fully connected layers such as `Flatten`, `Dense`, and `Dropout` are used to make final classification decisions based on features extracted by the convolutional layers.

After defining the architecture, it was followed by training the model on a labeled dataset. Basically, training involves feeding the images into the CNN and then making a forward pass to compute the predicted outputs, comparing these against the actual labels, and lastly computing the loss. The loss was then backpropagated through the network, updating the weights of the filters and the connections between neurons using the Adam, which is one of the most popular optimization algorithms [110]. This was repeated for 9 and 15 epochs, where an epoch is defined as a complete pass through the entire training dataset—9 for hand classification and 15 for nail type suggestion.

During training, several methods were employed to ensure good generalization on yet unobserved data. One major tool to ensure generalization is through the use of a validation set, typically denoting a subset of the data not used during training but for which performance is monitored. If it performs significantly better on the training set compared to the validation set, that may be due to overfitting. The model would have memorized the training data instead of generalizing from those. Another counter to prevent overfitting is a regularization technique of dropout—randomly setting some activations to zero during training—and it has been proven to be an efficient way [111].

Finally, the model was tested on unseen examples in the test set. This step checks if the model performs well after being trained and validated. In addition to this, some metrics were used to measure the performance in more detail. The typical ones used in measuring a model's effectiveness are accuracy, precision, recall, and the F1-score.

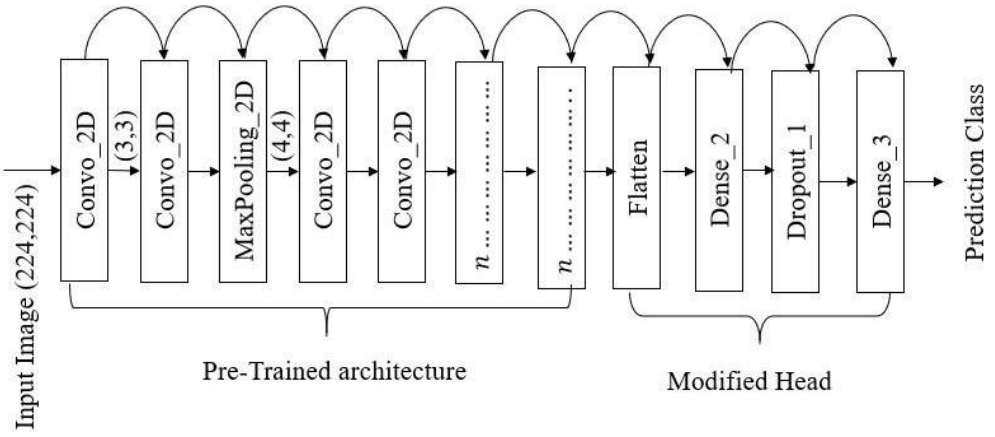


Figure 5.1. Example structure of a VGG16-based model (The "Pre-Trained architecture" refers to the original VGG16 model, while the "Modified Head" represents the fully connected layers that were added subsequently.) [112]

5.2. Model and Architecture Studies

In this study, the performance of many CNN-based architectures such as VGG16, VGG19, LeNet-5, AlexNet, GoogLeNet, ResNet, DenseNet, and MobileNet, was compared for hand type classification and nail type recommendation models, where hand images were used as input for the processes. VGG16 emerged as the most successful architecture for both models and was selected as the basis for the proposed model. Its performance has been evaluated further by confusion matrices and learning curves. To further improve the success rate of the model, various methods were explored, with a focus on hyperparameter tuning. Different hyperparameter values were tested and compared to optimize the performance.

5.2.1. Hyperparameter tuning

In this study, a CNN using the VGG16 architecture provided by TensorFlow's Keras Application Programming Interface (Keras API) is employed for image recognition tasks. Images were resized to 224x224 pixels and kept colored (RGB) to align with VGG16's input requirements. Several adjustments were made to enhance accuracy, including setting a random seed, preprocessing images with TensorFlow's ImageDataGenerator, and tuning key hyperparameters. Reducing the batch size and using a learning rate of 0.0001 significantly improved accuracy, while other learning rates did not yield better results. Pretrained layers were unfrozen, and custom layers such as Flatten, Dense (ReLU for hidden layer, Softmax for output), and Dropout were added. The model was compiled with the Adam optimizer, categorical crossentropy loss function, and accuracy metrics.

Various methods were attempted to improve overall accuracy, including replacing the flatten layer with global average pooling, adjusting epochs, altering dropout rate, adding more dropout layers, applying different data augmentation techniques (such as zoom and rotation), and using the epoch with the lowest validation loss. These methods either had no effect or decreased test accuracy.

The adjusted and used values of the hyperparameters and parameters after tuning and their values' effect on model performance (test accuracy) are given in Table 5.1. below.

Table 5.1. Used VGG16 hyperparameter values in models after tuning and their effect on model performances (“hand”: used in hand structure classification model, “nail”: used in nail type recommendation model, “both”: used in both models)

Hyperparameter	Set Value	Effect on Model Performance (Test Accuracy)
seed value	42 (both)	Fixed to ensure consistent results.
# of epochs	9 (hand), 15 (nail)	`< 9` (hand), `< 15` (nail): Performance decreases. `> 9` (hand), `> 15` (nail): Performance decreases.
batch size	4 (both)	`< 4`: Performance decreases. `> 4`: Performance decreases.
learning rate	0.0001 (both)	`< 0.0001`: Performance decreases. `> 0.0001`: Performance decreases.
dropout rate	0.1 (both)	`< 0.1`: Performance decreases. `> 0.1`: Performance decreases.
dense layer neurons	128 (both)	`< 128`: Performance decreases. `> 128`: Performance decreases.
Input image size	224x224 (both)	`≠ 224x224`: Performance decreases.
Optimizer	Adam (both)	`≠ Adam`: Performance decreases.
# of flatten layers	1 (both)	`< 1`: Performance doesn’t change. `> 1`: Performance doesn’t change.
# of dense layers	2 (both)	`< 2`: Performance doesn’t change. `> 2`: Performance doesn’t change.
# of dropout layers	1 (both)	`< 1`: Performance doesn’t change. `> 1`: Performance doesn’t change.

The model was trained using the fit method on the training set with a separate validation set for performance monitoring. Evaluation was conducted on a test set to examine performance on unseen data. Implementation was carried out on Google Colab, utilizing an NVIDIA L4 Tensor Core GPU with 24 Gigabyte (GB) Random-Access Memory (RAM) and 60 GB storage. The Sci-kit learn library facilitated image manipulation, while the cv2 package (OpenCV) assisted in accessing, loading, and manipulating image data.

5.2.2. Hand structure classification model analysis

Hand structure classification has been performed for different CNN networks and the results have been compared. The most successful model was studied in detail with confusion matrices and learning curves. A detailed analysis was also done by changing the hyperparameter values of the model.

The performance of various neural network architectures was evaluated over 9 epochs,

with results summarized in Table 8.2. VGG16 demonstrated the highest training accuracy, achieving 94.24%, and performed well with validation and test accuracies of 78.86% and 83.09%, respectively. This highlights VGG16's robust adaptability to our dataset and its capability to generalize effectively.

VGG19, chosen due to its architectural similarity to VGG16, closely followed with a training accuracy of 93.58%. It maintained strong performance with highest validation accuracy of 80.05% and test accuracy of and 78.74%, respectively, indicating its ability to learn and generalize similarly well across the dataset. This aligns with our decision to explore VGG19 based on its relationship to VGG16 and its demonstrated effectiveness.

LeNet-5 showed consistently modest performance with accuracies around 55% for training and slightly higher for validation, dropping to 53.38% on the test set. AlexNet performed poorly across all metrics, with accuracies around 26-28%, indicating significant challenges with the task.

GoogLeNet achieved a training accuracy of 76.95% but demonstrated overfitting with lower validation and test accuracies of 61.28% and 59.66%, respectively. ResNet and DenseNet also showed signs of overfitting, with ResNet’s accuracies dropping sharply from 51.85% training to around 26% for validation and test, and DenseNet showing a similar trend with training accuracy of 81.32% but much lower validation and test accuracies of 35.39% and 40.58%.

MobileNet achieved balanced performance with a training accuracy of 75.88% and higher validation and test accuracies of 70.31% and 70.53%, suggesting better generalization capabilities. Loss values corroborate these observations, where lower losses generally correspond to higher accuracies. VGG16 exhibited the lowest training loss, followed closely by MobileNet, which maintained balanced losses across training and validation/test sets, indicating consistent performance. In contrast, ResNet and DenseNet showed higher validation and test losses compared to their training losses, indicating overfitting issues.

Given the strong performance of VGG16 and the architectural similarities, we also trained VGG19, which demonstrated competitive results, reaffirming the effectiveness of this architecture family in learning and applying knowledge to our dataset.

These results illustrate the variability in how different architectures handle the training and generalization process for our dataset, with The VGG16 and VGG19 models, followed by the MobileNet, were found to be more robust in the conditions tested, as evidenced by the results in Table 5.2.

Table 5.2. Performance of architectures for hand structure classification model (a) Accuracies (b) Losses

Architecture	Accuracy (%)		
	Training	Validation	Test
VGG16	**94.24**	78.86	**83.09**
VGG19	93.58	**80.05**	78.74
LeNet-5	54.98	56.53	53.38
AlexNet	25.93	27.79	26.57
GoogLeNet	76.95	61.28	59.66
ResNet	51.85	26.37	25.85
DenseNet	81.32	35.39	40.58
MobileNet	75.88	70.31	70.53

(a)

Architecture	Loss		
	Training	Validation	Test
VGG16	**0.1813**	**0.5543**	**0.5242**
VGG19	0.2092	0.5584	0.6149
LeNet-5	1.0472	1.0682	1.1386
AlexNet	1.3712	1.3695	1.3700
GoogLeNet	0.5858	1.1129	1.1614
ResNet	1.0504	4.4165	4.5131
DenseNet	0.5371	1.9520	1.7798
MobileNet	0.6600	0.7302	0.7447

(b)

Given the optimal performance metrics of VGG16, a detailed analysis of the architecture was performed. This is evidenced by the confusion matrix shown in Figure 5.2., which shows that the classification model performs well overall. The model shows strong prediction accuracy for classes H1 (grounded) and H2 (piano), with 90 and 92 correct predictions respectively. However, it encounters some confusion between H4 (generous) and H3 (hifive), as evidenced by notable misclassifications. Specifically, H4 is misclassified as H3 10 times, and H3 is misclassified as H1 11 times. Misclassifications also occur between H4 and H1 and between H2 and the other classes, but these are less frequent. Although the model shows a high degree of accuracy, its performance could be further improved by focusing on reducing confusion between similar classes, particularly H4 and H3.

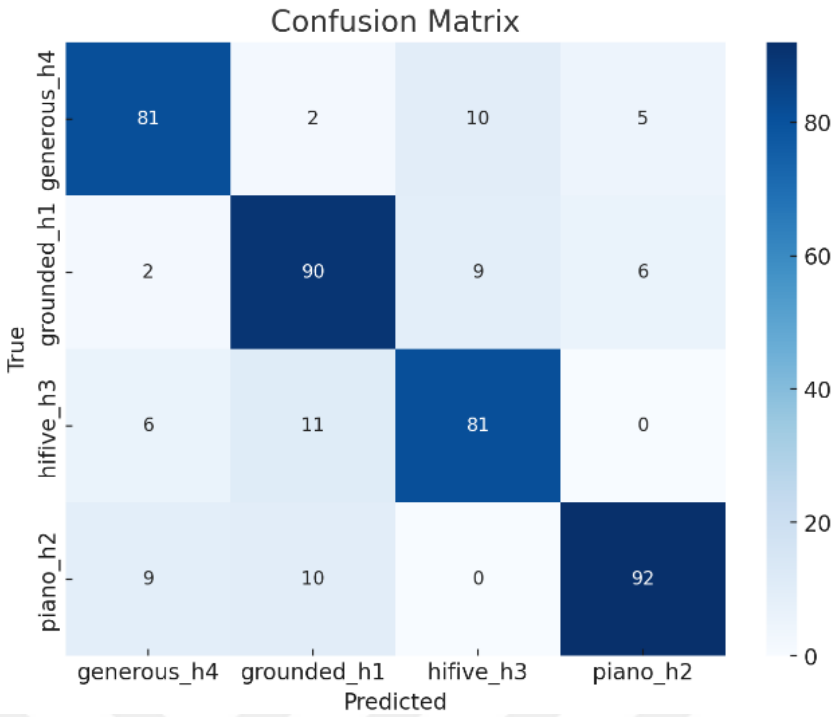


Figure 5.2. VGG16 confusion matrix for hand structure classification model

If confusion matrix is examined further, one can calculate the key metrics for each class and evaluate the performance in more detail. Table 5.3. below presents these performance metrics for each class. As shown, class H2 stands out with the highest precision of 89.32% and the highest F1-score of 85.98%, indicating the model's strongest ability to accurately predict and balance both precision and recall for this class. Then there is H1, which excels in recall, achieving the highest rate of 84.11%, demonstrating the model’s effectiveness in identifying H1 instances. Meanwhile, Classes H3 and H4 show balanced but less remarkable metrics, with H3 reflecting good overall effectiveness and H4 displaying consistent performance across precision, recall, and F1-score. Overall, the results were close to each other as there is no imbalance in the dataset. However, there were still some slight differences to point out. As stated, class H2 leads in precision and F1-score and class H1 excels in recall, but has the lowest rate in precision, whereas H3 and H4 remain balanced, indicating further refinement for these classes such as adding more diverse images to the classes’ dataset.

Table 5.3. Class performance metrics for hand structure classification model

Class	Precision (%)	Recall (%)	F1-Score (%)
H1	79.65	**84.11**	81.82
H2	**89.32**	82.88	**85.98**
H3	81.00	82.65	81.82
H4	82.65	82.65	82.65

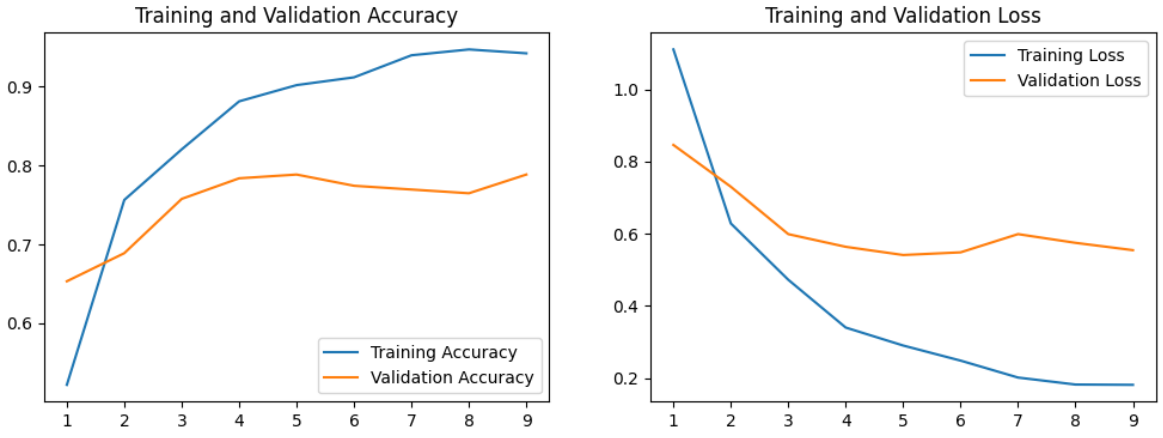
The averaging methods were also applied to the metrics, in the need of a further information about the dataset. Based on the Table 5.4. below, we can say that the model exhibits consistent performance across different averaging methods, reflecting stable and reliable classification across various classes. The higher precision and F1-score observed with macro averaging suggest that the model performs well on average, effectively balancing precision and recall, though it might miss some instances. Weighted averaging aligns closely with micro averaging, indicating that class distribution similarly impacts the overall performance, highlighting that the model's effectiveness is uniform across the dataset. So, it can be said that the minor variations between metrics show that the model is well-tuned and maintains a good balance between precision, recall, and F1-score.

Table 5.4. Averaged performance metrics for hand structure classification model

Averaging / Metric	Precision (%)	Recall (%)	F1-Score (%)
Micro-averaging	83.10	83.10	83.10
Macro-averaging	83.66	83.57	83.61
Weighted-averaging	83.29	83.10	83.17

In addition to the confusion matrix, the system's performance was evaluated using learning graphs. Looking at the training and validation accuracy graph in Figure 5.3., we can see that the training in the proposed model performs well and constantly increases to 94%, while the validation accuracy increases to 79% with dips in between, indicating that it is slightly lower than the training accuracy. The same goes for the training and validation loss graph, where the loss is 0.18 for training and 0.55 for validation. It can also be said that the accuracy changes gradually until the 3rd epoch and starts to stabilize at the 7th epoch. Meanwhile, the highest training accuracy is observed in the 8th epoch and the highest validation accuracy in the 9th epoch. It doesn't change much after the 9th epoch; it even decreases a little. That's why the number of epochs was reduced to 9, as nothing changes after this step and there is no need to increase the number of epochs accordingly. The detailed

training and validation loss of accuracy values are also given below in Table 5.5.



(a) (b)

Figure 5.3. VGG16 Training and Validation Graphs for hand structure classification model (a) Accuracy Graph (b) Loss Graph

Table 5.5. VGG16 The loss and accuracy of all epochs for hand structure classification model

Epoch	Training		Validation	
	Loss	Accuracy (%)	Loss	Accuracy (%)
1/9	1.1104	52.18	0.8459	65.32
2/9	0.6283	75.64	0.7300	68.88
3/9	0.4723	82.06	0.5987	75.77
4/9	0.3399	88.15	0.5636	78.38
5/9	0.2902	90.21	0.5409	78.86
6/9	0.2483	91.19	0.5483	77.43
7/9	0.2014	93.99	0.5988	76.96
8/9	0.1821	**94.73**	0.5746	76.48
9/9	**0.1813**	94.24	**0.5543**	**78.86**

The test accuracy for this system is 83.09%, meaning that the model performs well. However, it should be noted that the model uses a fixed seed, which means that the result is approximately the same for each training. For other seed values, the rates may change. In addition, the images of the hands are collected from different locations and the differences between the hands are large - the system may take other things like nails, skin color, patterns as differences - it may be difficult for the system to make classifications according to one feature. Taking all this into account, we can say that these factors may be the reason for the low test percentage.

Several hyperparameters of the proposed VGG16-based model were varied to determine their impact on system performance, focusing on test accuracy. First, examining

the number of epochs, the model achieved its highest test accuracy of 83.09% at 9 epochs. Beyond this point, the accuracy did not consistently improve and even decreased, as seen with 15 epochs (79.71%), 85 epochs (81.88%), and 165 epochs (80.68%). This indicates that training for more than 9 epochs may result in overfitting, where the model performs well on the training data but not on the test data.

When adjusting the learning rate, the model achieved its peak test accuracy of 83.09% with a learning rate of 0.0001. Lower learning rates like 0.00001 resulted in slightly lower accuracy (79.47%), while higher learning rates, such as 0.01 and 0.1, led to significantly lower accuracies of 25.85% and 27.29%, respectively. This suggests that higher learning rates cause the model to diverge, failing to converge properly during training.

Regarding batch size, a batch size of 4 yielded the best test accuracy of 83.09%. Other batch sizes, such as 2 (77.78%) and 16 (76.57%), resulted in lower accuracies. Even though a batch size of 64 provided a relatively high accuracy of 82.13%, it still did not surpass the performance with a batch size of 4. This indicates that a batch size of 4 offers the best balance for gradient estimation and computational efficiency.

Finally, the dropout rate was optimized at 0.1, where the model achieved its highest accuracy of 83.09%. Higher dropout rates like 0.3 and 0.5 led to reduced accuracies of 78.74%. A lower dropout rate of 0.2 resulted in an accuracy of 81.16%, which is still lower than the 0.1 dropout rate. Therefore, a dropout rate of 0.1 is optimal for preventing overfitting while maintaining high performance.

In summary, the optimal hyperparameters for the VGG16-based model are 9 epochs, a learning rate of 0.0001, a batch size of 4, and a dropout rate of 0.1. These settings achieved the highest test accuracy of 83.09%, balancing training duration, learning convergence, and regularization. The hyperparameters, along with their varied values and the corresponding test accuracies with the addition of training and validation accuracies are presented in Table 5.6.

Table 5.6. A comparison of the accuracies* of the hand structure classification model with altered hyperparameters (a) Epoch (b) Learning rate (c) Batch size (d) Dropout rate (Default values: epoch=9, learning rate=0.0001, batch size=4, dropout rate=0.1)

Epoch	Accuracy (%)		
	Training	Validation	Test
=5	89.79	77.43	77.54
=9	94.24	78.86	**83.09**
=15	97.28	79.10	79.71
=85	99.42	78.15	81.88
=165	**100.00**	**79.81**	80.68

(a)

Learning Rate	Accuracy (%)		
	Training	Validation	Test
=0.00001	89.88	75.06	79.47
=0.0001	**94.24**	**78.86**	**83.09**
=0.001	89.79	75.77	78.99
=0.01	23.54	26.37	25.85
=0.1	23.21	26.37	27.29

(b)

Batch Size	Accuracy (%)		
	Training	Validation	Test
=2	94.16	80.29	77.78
=4	94.24	78.86	**83.09**
=8	**95.23**	78.38	81.40
=16	94.98	76.72	76.57
=32	93.33	**80.76**	80.19
=64	94.90	77.91	82.13

(c)

Dropout Rate	Accuracy (%)		
	Training	Validation	Test
=0.1	**94.24**	78.86	**83.09**
=0.2	92.18	**79.33**	81.16
=0.3	90.95	78.38	78.74
=0.4	89.38	78.86	80.19
=0.5	84.77	75.53	78.74

(d)

*It should be noted that the accuracies were measured once, and they are prone to change %1-2 at each training, so results may vary.

Examining the final results alone does not fully capture the system's performance. To address this, the impact of hyperparameter variations on performance was studied in detail. It was observed that while increasing the number of epochs slightly reduces test accuracy, it leads to an increase in training accuracy. This is illustrated in Figure 5.4., which displays the training and validation accuracy and loss over 165 epochs. The training accuracy rapidly approaches nearly 100% - even reaching 100% in the last 3 epochs, as shown in the Figure 5.5. below - and remains stable in contrast, the validation accuracy fluctuates around 80%, almost the same as the 9-epoch model. his disparity suggests that while the model performs very well on training data, it has difficulty generalizing to validation data. The training loss decreases quickly and stabilizes at a low level, whereas the validation loss decreases first and then keeps increasing until the last epoch while exhibiting significant fluctuations, suggesting potential overfitting. This results in test accuracy not improving because validation accuracy fails to increase, leading to no improvement in overall system

performance.

A comparison of this model with the proposed 9-epoch model, reveals a more balanced learning process. Both training and validation accuracies increase steadily, with training accuracy reaching approximately 90% and validation accuracy plateauing at approximately 80%. Training loss consistently decreases, and validation loss levels off after an initial decrease, indicating effective learning without severe overfitting.

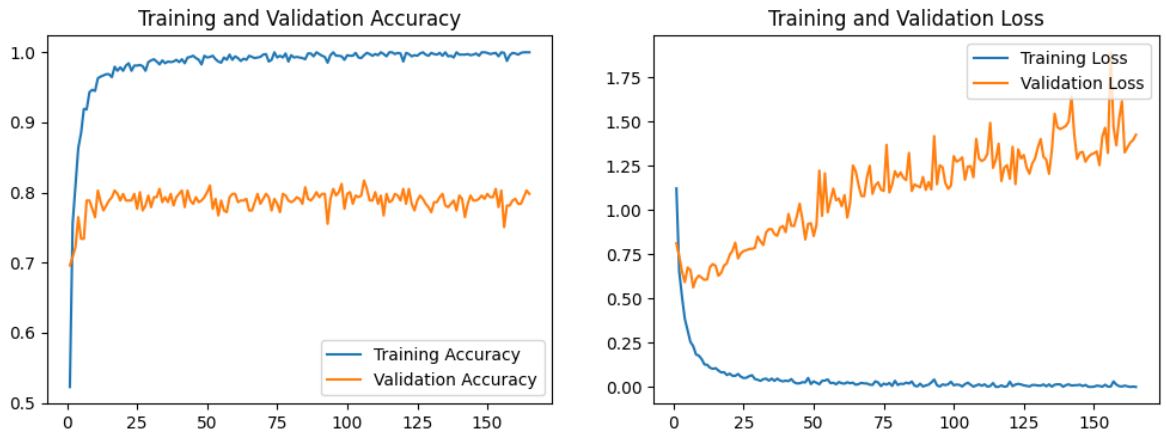


Figure 5.4. Training and validation graphs over 165 epochs for hand structure classification model (a) Accuracy graph (b) Loss graph

```
Epoch 163/165
304/304 [==============================] - 13s 43ms/step - loss: 6.0114e-04 - accuracy: 1.0000 -
Epoch 164/165
304/304 [==============================] - 14s 45ms/step - loss: 0.0026 - accuracy: 1.0000 - val_
Epoch 165/165
304/304 [==============================] - 14s 45ms/step - loss: 5.6610e-04 - accuracy: 1.0000 -
104/104 [==============================] - 3s 33ms/step - loss: 1.4199 - accuracy: 0.8068
```

Figure 5.5. Accuracies of the last 3 epochs (“accuracy” is training accuracy, reaching 1.0000 (100%))

It was also observed that even though the testing accuracy was best at a batch size of 4, the training and validation accuracies were not the highest. Instead, the system with a batch size of 32 had the highest validation accuracy. For that, a detailed analysis has been done. By looking at Figure 5.6., It is determined that the training accuracy increases steadily, nearing 95% by the 9th epoch, while the validation accuracy peaks around the 6th epoch and then changes. The training loss consistently decreases, indicating effective learning, but the validation loss starts to fluctuate after initially decreasing.

Although the batch size of 32 shows high validation accuracy, it is noted that the model with a batch size of 4 achieves the highest test accuracy. This indicates that while larger batch sizes may perform well during validation, smaller batch sizes like 4 may generalize

better to unseen test data.

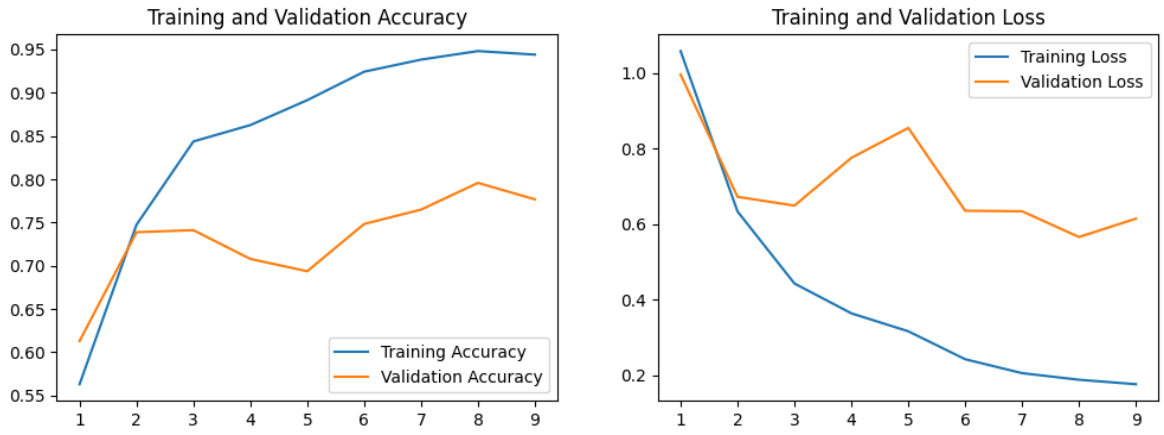


Figure 5.6. Training and validation graphs with a batch size of 32 for hand structure classification model (a) Accuracy graph (b) Loss graph

5.2.3. Nail type recommendation model analysis

Suitable nail type recommendations for hand structures were evaluated using different CNN networks and their results were compared. The most effective model was studied in detail using confusion matrices and learning curves. In addition, a in-depth analysis was performed by adjusting the hyperparameters of the model.

The nail recommendation system models were tested, and the evaluation of various CNN architectures over 15 epochs is presented in Table 5.7. VGG16 stood out with the highest training accuracy of 96.87%, along with validation and test accuracies of 80.29% and 82.37%, that can be attributed to the robust and deep architecture of the VGG16 which allows for effective feature extraction and generalization. VGG19 also performed well, achieving a training accuracy of 95.64% and validation and test accuracies of 79.10% and 79.71%, having the second-best performance that can be attributed to its similar architecture to VGG16, which allows it to achieve high accuracy while maintaining good generalization on validation and test datasets.

In contrast, LeNet-5's performance was relatively low, with a training accuracy of 56.63% and a test accuracy of 50.24%, indicating that it struggled with the complexity of the data. ResNet showed the poorest performance among the evaluated architectures, with a test accuracy of just 27.29%, possibly due to underfitting or difficulties in training.

GoogLeNet achieved a high training accuracy of 89.55%, but its performance significantly dropped during validation and testing, with accuracies of 54.63% and 54.59%, suggesting potential overfitting. DenseNet also showed signs of overfitting, achieving a

training accuracy of 91.44% but much lower validation and test accuracies of 65.56% and 65.70%, respectively.

MobileNet demonstrated balanced performance, with a training accuracy of 82.14% and validation and test accuracies of 73.87% and 74.40%, indicating good generalization capabilities. AlexNet had the lowest training accuracy at 67.90%, yet its performance on unseen data was moderate, with validation and test accuracies of 61.28% and 61.35%. These results indicate varying degrees of overfitting and generalization across different architectures.

Table 5.7. Performance of architectures for nail type recommendation model (a) Accuracies (b) Losses

Architecture	Accuracy (%)		
	Training	Validation	Test
VGG16	**96.87**	**80.29**	**82.37**
VGG19	95.64	79.10	79.71
LeNet-5	56.63	52.49	50.24
AlexNet	67.90	61.28	61.35
GoogLeNet	89.55	54.63	54.59
ResNet	58.85	26.37	27.29
DenseNet	91.44	65.56	65.70
MobileNet	82.14	73.87	74.40

(a)

Architecture	Loss		
	Training	Validation	Test
VGG16	**0.0939**	**0.6245**	**0.5769**
VGG19	0.1336	0.6288	0.6515
LeNet-5	1.0354	1.0879	1.1302
AlexNet	0.8310	0.9761	0.9823
GoogLeNet	0.2988	1.5293	1.5660
ResNet	0.9406	6.8886	6.7675
DenseNet	0.2890	1.3125	1.2490
MobileNet	0.4986	0.6397	0.6237

(b)

Overall, VGG16 and VGG19, followed by MobileNet emerged as the most robust models, demonstrating strong adaptance and generalization across datasets. Consequently, VGG16 will be analyzed in greater detail.

The confusion matrix in Figure 5.7. shows that the model generally performs well for N1 and N4 classes, with 94 and 90 correctly classified instances, respectively. However, there are notable misclassifications, such as 13 instances of N2 being misclassified as N1 and 10 instances of N1 as N4. The matrix shows that N2 is frequently confused with other classes, indicating a need for further tuning or more training data to enhance the model's ability to differentiate these classes. Despite these misclassifications, the model shows strong overall performance, particularly in accurately identifying N1 and N4.

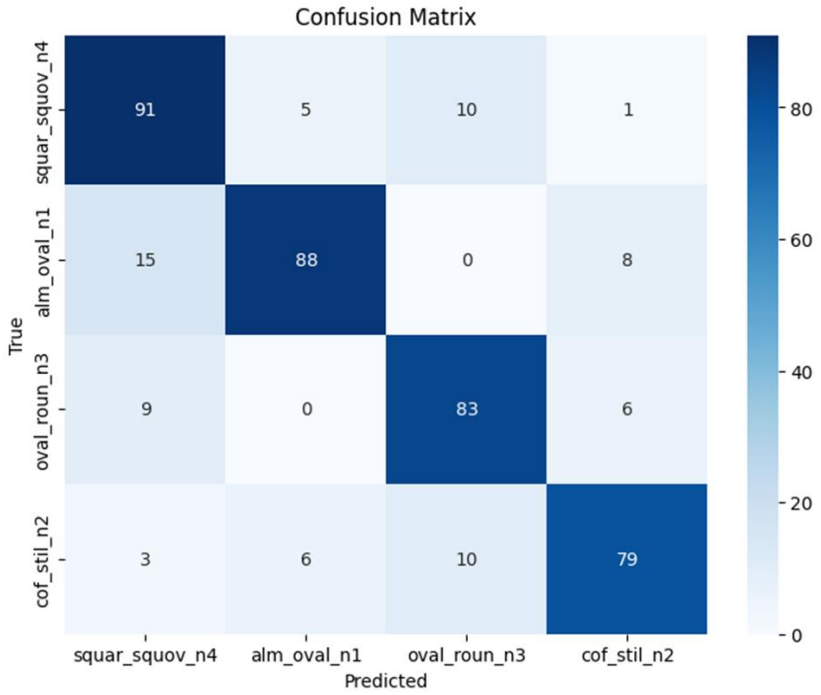


Figure 5.7. VGG16 confusion matrix for nail type recommendation model

By examining the confusion matrix in detail, one can calculate key performance metrics for each class, that offer a in-depth evaluation of the model. These metrics, obtained from the confusion matrix, are presented in Table 5.8. below. As shown, class N4 excels in recall with 85.98%, demonstrating the model's strong capability in identifying instances of this class. Meanwhile N1 stands out for its precision and F1-Score of 88.89% and 83.85%, reflecting exceptional accuracy and balance between precision and recall. On the other hand, N3 exhibits lower precision, indicating a higher rate of FPs, while class N4 has the lowest F1-Score. So as stated, even though some classed are superior to others in some metrics, none of them can be counted as the most successful class, as values are too different. Here, N1 excels at two metrics but its recall is too low for N4. This suggests an overall refinement for all classes.

Table 5.8. Class performance metrics for nail type recommendation model

Class	Precision (%)	Recall (%)	F1-Score (%)
N1	**88.89**	79.28	**83.85**
N2	84.04	80.61	82.27
N3	80.58	84.69	82.57
N4	77.12	**85.98**	81.27

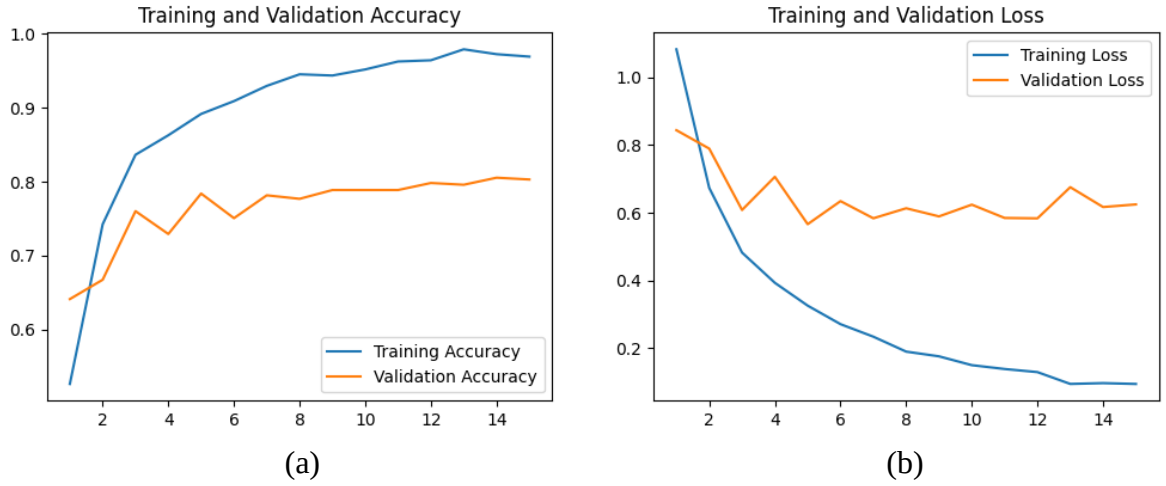
Averaging methods have also been applied for further information, below in Table 5.9.

The application of different averaging methods show that the model performs consistently across metrics. Micro-averaging indicates uniform effectiveness, while macro-averaging highlights balanced performance with a slight trade-off in precision. Weighted-averaging reflects the impact of class distribution but still aligns closely with overall performance. These findings suggest the model is well-tuned and maintains a good balance between precision, recall, and F1-Score with minor variations.

Table 5.9. Averaged performance metrics for nail type recommendation model

Averaging / Metric	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
Micro-averaging	82.40	82.40	82.40
Macro-averaging	80.91	82.14	82.24
Weighted-averaging	75.63	80.21	82.51

As in the previous section, the evaluation of the system's performance was extended to include accuracy and loss graphs in addition to the confusion matrix. The training and validation accuracy graph in Figure 5.8. shows that training accuracy reaches 97%, while validation accuracy peaks at 80% with fluctuations, significantly lower than training accuracy. Similarly, the training and validation loss graph indicates a training loss of 0.09 and validation loss of 0.62. Validation accuracy shows gradual improvement until the 7th epoch, stabilizing by the 9th epoch with minimal change thereafter. Given this stability, training was capped at 15 epochs. The detailed training and validation loss of accuracy values are also given below in Table 5.10.



(a) (b)

Figure 5.8. VGG16 Training and Validation Graphs for nail type suggestion model (a) Accuracy Graph (b) Loss Graph

Table 5.10. VGG16 The loss and accuracy of all epochs for nail type recommendation model

Epoch	Training		Validation	
	Loss	**Accuracy (%)**	**Loss**	**Accuracy (%)**
1/15	1.0828	52.67	0.8438	64.13
2/15	0.6733	74.24	0.7894	66.75
3/15	0.4821	83.62	0.6081	76.01
4/15	0.3928	86.26	0.7061	72.92
5/15	0.3253	89.14	**0.5660**	78.38
6/15	0.2706	90.86	0.6342	75.06
7/15	0.2336	92.92	0.5835	78.15
8/15	0.1895	94.49	0.6131	77.67
9/15	0.1755	94.32	0.5891	78.86
10/15	0.1494	95.14	0.6238	78.86
11/15	0.1381	96.21	0.5845	78.86
12/15	0.1289	96.38	0.5833	79.81
13/15	0.0940	**97.86**	0.6754	79.57
14/15	0.0964	97.20	0.6167	**80.52**
15/15	**0.0939**	96.87	0.6245	80.29

The test accuracy is 82.37%, indicating strong performance of the model. However, like the hand structure classification model, it uses a fixed seed, meaning results are consistent but may vary with different seeds. It also uses the same hand images collected from different sources, which have significantly different features, posing a challenge for single-feature classification. These factors likely contribute to the observed variability and lower test accuracy.

Several hyperparameters of the proposed VGG16-based model were varied to determine their impact on system performance, with a particular focus on test accuracy. First, when examining the number of epochs, the model achieved its highest test accuracy of 82.37% at 15 epochs. Training for more or fewer epochs resulted in lower accuracies, such as 81.88% at 9 epochs and 80.43% at 5 epochs. This suggests that training for 15 epochs provides a balanced approach where the model avoids both underfitting and overfitting, maintaining good generalization to unseen data.

In terms of learning rate adjustments, the model reached its peak test accuracy of 82.37% with a learning rate of 0.0001. Both lower and higher learning rates resulted in decreased performance, such as 78.74% at a rate of 0.00001, 75.60% at a rate of 0.001 and very poor performance, dropping as low as 20-25% at higher learning rates. This indicates that while extremely low learning rates may cause the model to converge too slowly,

excessively high rates may prevent proper convergence, leading to suboptimal performance.

When exploring the impact of batch size, a batch size of 4 yielded the highest test accuracy of 82.37%. Smaller batch sizes, such as 2, achieved a slightly lower accuracy of 80.43%, while higher sizes like 64 resulted in even lower accuracies, such as 79.95%. This suggests that a batch size of 4 provides the best balance between stability in gradient estimation and computational efficiency.

Lastly, examining the dropout rate, the model performed best with a dropout rate of 0.1, achieving a test accuracy of 82.37%. Increasing the dropout rate generally led to reduced performance, such as 79.23% at a dropout rate of 0.3. This suggests that while dropout is necessary for preventing overfitting, too high of a dropout rate can result in underfitting, where the model fails to obtain important patterns in the data.

It can be said that the optimal hyperparameters for the VGG16-based model are 15 epochs, a learning rate of 0.0001, a batch size of 4, and a dropout rate of 0.1. These settings led to the highest test accuracy of 82.37%, effectively balancing training duration, learning convergence, and regularization. Other hyperparameter values have also been tested and detailed alongside the optimal hyperparameter results in Table 5.11. below, alongside the accuracy results.

Table 5.11. A comparison of the accuracies* of the nail type recommendation model with altered hyperparameters (a) Epoch (b) Learning rate (c) Batch size (d) Dropout rate (Default values: epoch=15, learning rate=0.0001, batch size=4, dropout rate=0.1)

Epoch	Accuracy (%)		
	Training	Validation	Test
= 5	88.48	78.38	80.43
= 9	94.40	78.62	81.88
= 15	96.87	**80.29**	**82.37**
= 85	**99.01**	78.15	79.47
= 165	**99.01**	80.05	79.47

(a)

Learning Rate	Accuracy (%)		
	Training	Validation	Test
= 0.00001	96.02	77.20	78.74
= 0.0001	**96.87**	**80.29**	**82.37**
= 0.001	92.80	75.30	75.60
= 0.01	21.38	26.37	25.85
= 0.1	25.76	23.52	23.67

(b)

Batch Size	Accuracy (%)		
	Training	Validation	Test
= 2	97.74	78.38	80.43
= 4	96.87	80.29	**82.37**
= 8	97.46	81.24	81.16
= 16	96.97	80.29	81.64
= 32	**98.05**	78.15	80.19
= 64	96.87	**80.76**	79.95

(c)

Dropout Rate	Accuracy (%)		
	Training	Validation	Test
= 0.1	96.87	**80.29**	**82.37**
= 0.2	**97.71**	78.15	78.99
= 0.3	96.10	79.10	79.23
= 0.4	93.74	77.91	80.43
= 0.5	90.89	79.10	79.46

(d)

*It should be noted that the accuracies were measured once, and they are prone to change %1-2 at each training, so results may vary.

As in the previous section, the effects of the hyperparameters on the system performance were studied, starting with the 165-epoch model, whose graph can be seen in Figure 5.9. Similar to the hand structure classification model, increasing the number of epochs to 165 in the nail type recommendation model led to high training accuracy, approaching 1.0000 (100%) by the final epochs, as shown in Figure 5.10. However, the validation accuracy fluctuates around 80%, indicating the model's difficulty in generalizing to new data. The training loss decreases and stabilizes, while the validation loss increases, even more than hand classification model.

Comparatively, the 15-epoch model shows more balanced performance, with both training and validation accuracies reaching around 90% and 80%, respectively. The steady decrease in training loss and leveling off of validation loss further indicate effective learning without severe overfitting.

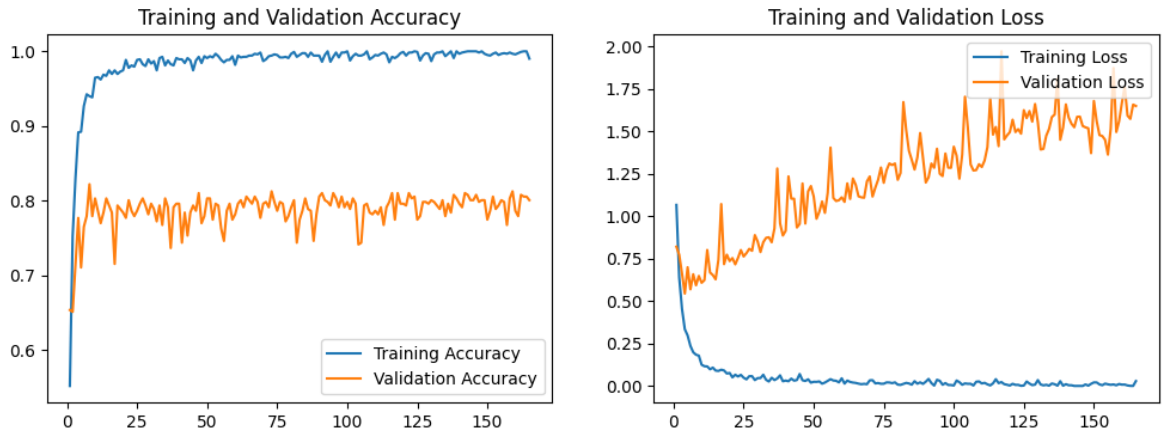


Figure 5.9. Training and validation graphs over 165 epochs for nail type recommendation model (a) Accuracy graph (b) Loss graph

```
Epoch 163/165
304/304 [==============================] - 13s 44ms/step - loss: 6.5937e-04 - accuracy: 1.0000 -
Epoch 164/165
304/304 [==============================] - 14s 45ms/step - loss: 3.6504e-04 - accuracy: 1.0000 -
Epoch 165/165
304/304 [==============================] - 14s 46ms/step - loss: 0.0290 - accuracy: 0.9901 - val_
104/104 [==============================] - 3s 33ms/step - loss: 1.5617 - accuracy: 0.7947
```

Figure 5.10. Training accuracies of the last 3 epochs

Observations also showed that although the performance does not vary significantly, the system with a 0.2 dropout rate achieved the highest training accuracy. A detailed analysis, supported by the accuracy and loss graphs in Figure 5.11., reveals that the training accuracy consistently increases, reaching approximately 90% by the 10th epoch, while the validation accuracy stabilizes around the 5th epoch with minor fluctuations. This implies that the model is effectively learning during training, as evidenced by the continuous decline in training loss.

However, it is worth noting that the system with a 0.1 dropout rate ultimately achieved the highest test accuracy. This indicates that slightly less regularization may allow for better model generalization, offering an optimal balance between preventing overfitting and maintaining high accuracy.

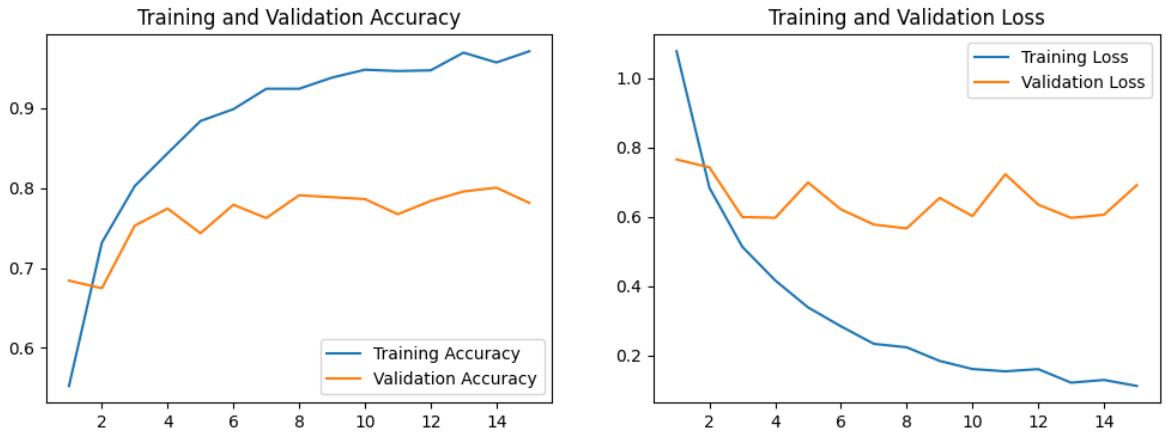


Figure 5.11. Training and validation graphs with a dropout rate of 0.2 for nail type recommendation model (a) Accuracy graph (b) Loss graph

5.2.4. Overall evaluation of the models

The hand structure classification (M1) and nail type recommendation (M2) models were evaluated separately, using the same dataset but with different folder names. As a result, the performances were similar, with some slight differences, as shown in Table 5.12. below.

Table 5.12. Comparison of hand structure classification (M1) and nail type recommendation (M2) models ("Prec.":Precision, "Rec.":Recall, "F1": F1 Score)

Model	Hand Structure Classification (M1)			Nail Type Recommendation (M2)		
Accuracy (%)	Training	Validation	Test	Training	Validation	Test
	94.24	78.86	83.09	96.87	80.29	82.37
Performance Metrics (%)	Prec., rec., $F1_{micro}$		$Prec._{weighted}$	Prec., rec., $F1_{micro}$		$Prec._{weighted}$
	83.10		83.29	82.40		75.63
Based Architecture	VGG16			VGG16		
Hyperparameters	epoch = 9, batch size = 4, learning rate = 0.0001, dropout rate = 0.1			epoch = 15, batch size = 4, learning rate = 0.0001, dropout rate = 0.1		
Dataset	Hand Structure Classification / Nail Type Recommendation			Hand Structure Classification / Nail Type Recommendation		

Both models had the highest success with the VGG16 model, making it the base architecture. If we compare the accuracies, the training and validation accuracy of M2 turned out to be slightly higher than M1 with 96.87% and 80.29%, respectively. However, the overall (test) accuracy of M1 resulted to be higher than M2 with 83.09%. The hyperparameter values of both models were adjusted to be the same, except for the number of epochs, which is higher for M2, explaining the higher training accuracy.

Performance metrics differed, especially the weighted-averaged precision. Though, the other were not overly different, as the models were practically the same with minor modifications. The only notable difference is that the micro-averaged metrics was approximately 0.7% higher in M1, which may indicate the model has a slight edge in overall performance across all classes.

The hyperparameter values were nearly identical, with the exception of the number of epochs. The increased number of epochs in M2 led to higher training accuracy; however, it did not significantly impact the test accuracy, as observed in M1, which was trained for only 9 epochs.

It is important to note that while M2 achieved higher training accuracy, its validation accuracy was not significantly higher than that of M1. This implies that M2 might have started to overfit the training data as the number of epochs increased. On the other hand, M1, with fewer epochs, demonstrated better generalization to unseen data, as reflected in its slightly higher test accuracy.

Furthermore, the dropout rate was set at 0.1 for both models. This value may have contributed to controlling overfitting in both cases, but given the results, it appears that M1's configuration was more effective in balancing training and generalization performance. Another aspect to consider is the batch size of 4, which was kept constant across models. A smaller batch size can result in a high number of noisy gradient estimations, but it may also help models to converge better by exploring a more diverse set of parameter updates.

To provide a clear overview of the implementation details and the model development steps, the pseudocode of the deep learning model is given in Figure 5.12. below:

```
1. Import Libraries
   • Import necessary libraries: TensorFlow, NumPy, Matplotlib,
     Seaborn, etc.
2. Set Random Seed
   • Set random seed for reproducibility.
3. Define Image Loading Function
   • Create function load_images:
        - Initialize ImageDataGenerator.
        - Load and preprocess training images.
        - Load and preprocess validation images.
        - Load and preprocess test images.
        - Return data generators and class names.
4. Specify Class Names
   • Define class_names list.
5. Load Datasets
   • Call load_images function with dataset paths to get training,
     validation, and test data generators.
6. Define Model Hyperparameters
   • Set epochs, batch_size, and learning_rate.
7. Create Deep Learning Model
   • Define function create_vgg_model:
        ◦ Load VGG16 base model (pre-trained).
        ◦ Freeze base model layers.
        ◦ Add new top layers: Flatten, Dense (128 units, ReLU),
          Dropout, Dense (softmax).
8. Compile Model
   • Compile model with Adam optimizer, categorical crossentropy loss,
     and accuracy metric.
9. Train Model
   • Fit model on training data with validation.
10.Evaluate Model
   • Evaluate model on test data.
   • Print test accuracy.
11.Generate Predictions
   • Predict test set classes.
12.Compute and Plot Confusion Matrix
   • Compute confusion matrix from true and predicted labels.
   • Plot confusion matrix using Seaborn.
13.Plot Training History
   • Extract training and validation accuracy and loss.
   • Plot accuracy and loss curves.
```

Figure 5.12. Pseudocode for the deep learning model

6. CONCLUSION

As a result of this study, a deep learning model using a CNN with VGG16 architecture and optimized hyperparameters was developed to classify hand images based on their structure and to recommend nail types.

Due to the lack of a suitable dataset for hand structure recognition, a dataset was created using various other resources. 2050 dorsal and palmar hand images were collected and automatically classified into dataset folders using feature measurement methods. The measurements were performed using the MediaPipe hand tracking module.

The study employs two models: The hand structure classification model and the nail type recommendation model.

Hand structure classification model, capable of automatically categorizing hands into four distinct structure types – Grounded Hands (H1), Piano Hands (H2), Hi-Five Hands (H3) and Generous Hands (H4) – as it achieved an accuracy rate of 83.09%.

In the training phase, the CNN model was fine-tuned to achieve optimal performance. The adjustments included resizing images to 224x224 pixels and rescaling them, setting the epoch to 9, using a dropout rate of 0.1, using a batch size of 4, and setting the learning rate to 0.0001, as other hyperparameter values success rate has been compared and these were the highest ones. These modifications contributed to a training accuracy of 94.24% and a validation accuracy of 78.86%, indicating strong learning and generalization capabilities.

Comparative performance evaluations with other neural network architectures highlighted the superiority of the VGG16 model. While VGG19, LeNet-5, AlexNet, GoogLeNet, ResNet, and DenseNet were also tested, VGG16 consistently outperformed these models in terms of training and validation accuracies. VGG19 and MobileNet demonstrated balanced performance but did not surpass VGG16 in overall accuracy.

The confusion matrix and performance metrics for the hand structure classification model highlight overall effectiveness with an accuracy of 83.09%, a precision of 68.07%, and a recall of 71.68%. Despite high specificity at 87.38% and a balanced F1 Score of 69.83%, there are notable misclassifications between classes H4, H3, and H2. The VGG16 model shows strong prediction accuracy for H1 and H2 but struggles with confusion between H4 and H3. Class H2 excels with the highest precision and F1-score, while H1 leads in recall but has lower precision. Classes H3 and H4 are balanced but less remarkable. Averaging methods reveal that the model performs consistently across different metrics, with macro

averaging indicating strong overall performance and weighted averaging aligning closely with micro averaging. Minor variations in metrics suggest that while the model is well-tuned, further refinement could enhance precision and recall.

The training accuracy of the proposed model steadily increases to 94%, while validation accuracy reaches 79% with some fluctuations, indicating it is consistently lower than training accuracy. The training loss is 0.18 and the validation loss is 0.55, reflecting similar trends. Accuracy improves gradually until the 3rd epoch, stabilizes by the 7th epoch, with the highest training accuracy observed at the 8th epoch and the highest validation accuracy at the 9th epoch. Surprisingly, however, both metrics change minimally after the 9th epoch. This leads to the decision to limit the training to 9 epochs, even though this is a low number for models in general.

The nail type recommendation model, which categorizes hand images into four nail type groups (N1: Oval & Round, N2: Almond & Oval, N3: Coffin/Ballerina & Stiletto, N4: Square & Squoval), achieved an accuracy of 82.37%. During training, images were rescaled and resized to 224x224 pixels, with a batch size of 4, epoch of 15, dropout rate of 0.1 and a learning rate of 0.0001, as other hyperparameter values success rate has been compared and these were the highest ones, resulting in a training accuracy of 96.87%, validation accuracy of 80.29% and a test accuracy of 82.37%.

Comparative evaluations with other architectures, including VGG19, LeNet-5, AlexNet, GoogLeNet, ResNet, DenseNet, and MobileNet, showed that VGG16 consistently outperformed them all in training and validation accuracies, confirming it as the best performer.

The confusion matrix indicates that the model performs effectively for classes N1 and N4, correctly classifying 94 and 90 instances, respectively. However, there are significant misclassifications, such as N2 being frequently mistaken for N1 and N1 for N4. This suggests that N2 is often confused with other classes, highlighting the need for additional tuning or more training data. Key metrics reveal that N4 excels in recall, showcasing strong identification capability, while N1 leads in precision and F1-Score, demonstrating a good balance between precision and recall. Conversely, N3 has lower precision and N4 has the lowest F1-Score. The averaging methods applied show consistent model performance, with micro-averaging reflecting uniform effectiveness, macro-averaging indicating balanced performance with slight trade-offs, and weighted-averaging aligning closely with overall results. These findings suggest that the model is well-tuned but could benefit from refinements to enhance performance across all classes.

The training and validation accuracy graph shows that training accuracy reaches 97%, while validation accuracy peaks at 80% with some fluctuations. The training loss is 0.09 and the validation loss is 0.62. Validation accuracy improves until the 7^{th} epoch, then stabilizes by the 9^{th} epoch, leading to the decision to cap training at 15 epochs, which is also surprisingly a low value for epoch.

The accuracies achieved by the proposed VGG16 model indicates strong performance; however, the use of a fixed seed may impact the consistency of the results. Variability in hand images—such as differences in nail types, skin color, patterns and the position of the hand—could affect the model's classification ability, given that dataset was not constructed specifically for hand classification, it was adapted to it. Another important element, and probably the main reason for affecting the results, would be that the images of the different classes not being so distinct from each other, as the hand types differ only by minimal dimensions, resulting in a limited number of features that could contribute to the observed lower test accuracy.

Overall, a specialized dataset was created from various sources, with images being automatically classified using a script. A CNN model based on the VGG16 architecture was developed with two main models for hand structure classification and nail type recommendation. Performance evaluations showed that the VGG16-based model outperformed other neural network architectures.

Building on these advancements, this study offers a comprehensive foundation for future research on hand structure and nail types, setting the stage for the development of specialized models for various applications. By thoroughly examining the intricate details of hand anatomy and nail characteristics, the research aims to inform and improve a range of fields, including medical assessment, prosthetic design, ergonomic tool development, and aesthetic applications. The knowledge gained from this study will be instrumental in creating tailored solutions that respond to each individual’s needs and improve practical outcomes in these diverse areas. Consequently, the results have the potential to make significant contributions to both scientific knowledge and practical applications.

REFERENCES

[1] F. Sadikoglu and S. Uzelaltinbulat, “Biometric retina identification based on neural network,” *Procedia Computer Science*, vol. 102, pp. 26–33, Jan. 2016, doi: 10.1016/j.procs.2016.09.365.

[2] R. A. Priyadharshini, S. Arivazhagan, and M. Arun, “A deep learning approach for person identification using ear biometrics,” *Applied Intelligence*, vol. 51, no. 4, pp. 2161–2172, Oct. 2020, doi: 10.1007/s10489-020-01995-8.

[3] C. Yücelbaş, “MLP tabanlı DNN modeli kullanılarak akıllı alanlar için yürüyüş analizinden kişi tanıma,” *Düzce Üniversitesi Bilim Ve Teknoloji Dergisi*, vol. 11, no. 2, pp. 1025–1036, Apr. 2023, doi: 10.29130/dubited.1187065.

[4] Z. Huang, Z. Pan, and B. Lei, “Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data,” *Remote Sensing*, vol. 9, no. 9, p. 907, Aug. 2017, doi: 10.3390/rs9090907.

[5] K. DiMiLiLer and B. Sekeroglu, “Skin lesion classification using CNN-based transfer learning model,” *GAZI UNIVERSITY JOURNAL OF SCIENCE*, vol. 36, no. 2, pp. 660–673, Jun. 2023, doi: 10.35378/gujs.1063289.

[6] Z. B. Kın, “Türk işaret dili alfabesinin derin öğrenme yöntemi ile sınıflandırılması,” M.S. thesis, Dept. Elec. Electron. Eng., Başkent Univ., Ankara, Türkiye, 2019.

[7] A. O. George, “Finger nail plate shape and size for personal identification – a possible low technology method for the developing world - Preliminary report,” *African Journal of Health Sciences*, vol. 12, no. 1, Nov. 2005, doi: 10.4314/ajhs.v12i1.30795.

[8] A. Kumar and D. Zhang, “Personal recognition using hand shape and texture,” *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2454–2461, Aug. 2006, doi: 10.1109/tip.2006.875214.

[9] S. Sharma, S. R. Dubey, S. K. Singh, R. Saxena, and R. K. Singh, “Identity verification using shape and geometry of human hands,” *Expert Systems With Applications*, vol. 42, no. 2, pp. 821–832, Feb. 2015, doi: 10.1016/j.eswa.2014.08.052.

[10] L. F. M. Kuijt-Evers, L. Groenesteijn, M. P. De Looze, and P. Vink, “Identifying factors of comfort in using hand tools,” *Applied Ergonomics*, vol. 35, no. 5, pp. 453–458, Sep. 2004, doi: 10.1016/j.apergo.2004.04.001.

[11] C. Pacchierotti, S. Sinclair, M. Solazzi, V. Hayward, and D. Prattichizzo, “Wearable haptic systems for the fingertip and the hand: taxonomy, review and perspectives,” *Institute of Philosophy*, Oct. 2017, [Online]. Available: sas-space.sas.ac.uk/6564/

[12] R. Regin, G. R. G, S. K. Ch, and J. CVN, “Nail disease detection and classification using deep learning,” *Central Asian Journal of Medical and Natural Science*, pp. 55–73, May 2022.

[13] L. Zollo, S. Roccella, E. Guglielmelli, M. C. Carrozza, and P. Dario, “Biomechatronic Design and control of an anthropomorphic artificial hand for prosthetic and robotic applications,” *IEEE/ASME Transactions on Mechatronics*, vol. 12, no. 4, pp. 418–429, Aug. 2007, doi: 10.1109/tmech.2007.901936.

[14] H. In, B. B. Kang, M. Sin, and K.-J. Cho, “Exo-Glove: A Wearable Robot for the Hand with a Soft Tendon Routing System,” *IEEE Robotics & Automation Magazine*, vol. 22, no. 1, pp. 97–105, Mar. 2015, doi: 10.1109/mra.2014.2362863.

[15] S.-C. Jee and M. H. Yun, “An anthropometric survey of Korean hand and hand shape types,” *International Journal of Industrial Ergonomics*, vol. 53, pp. 10–18, May 2016, doi: 10.1016/j.ergon.2015.10.004.

[16] K. E. Aboul-Hagag, S. A. Mohamed, M. A. Hilal, and E. A. Mohamed, “Determination of sex from hand dimensions and index/ring finger length ratio in Upper Egyptians,” *Egyptian Journal of Forensic Sciences*, vol. 1, no. 2, pp. 80–86, Jun. 2011, doi: 10.1016/j.ejfs.2011.03.001.

[17] E. Cakit, B. Durgun, O. Cetik, and O. Yoldas, “A survey of hand anthropometry and biomechanical measurements of dentistry students in Turkey,” *Human Factors and Ergonomics in Manufacturing & Service Industries*, vol. 24, no. 6, pp. 739–753, Jun. 2012, doi: 10.1002/hfm.20401.

[18] M. Bures, T. Gorner, and B. Sediva, “Hand anthropometry of Czech population,” in *IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, Dec. 01, 2015, vol. pp. 1077-1082. doi: 10.1109/ieem.2015.7385814.

[19] A. S. Ermolenko and R. M. Khayrullin, “Classification and regression tree analysis for predicting morphological hand types based on radiography data,” *International Journal of Morphology*, vol. 39, no. 6, pp. 1727–1730, Dec. 2021, doi: 10.4067/s0717-95022021000601727.

[20] Y. Hong, H. S. Kim, and H. E. Choi, “Development of a hand classification system for smart hand wearables,” *Journal of Industrial Textiles*, vol. 53, Jan. 2023, doi: 10.1177/15280837231188529.

[21] S. Narasimhaswamy, T. Nguyen, and M. H. Nguyen, “Detecting hands and recognizing physical contact in the wild,” *arXiv (Cornell University)*, vol. 33, pp. 7841–7851, Jan. 2020, [Online]. Available: arxiv.org/pdf/2010.09676

[22] M. Z. Nayebi and Z. Turgut, “Dorsal hand veins based biometric identification system using deep learning,” *Erzincan Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, vol. 14, no. 1, pp. 1–15, Mar. 2021, doi: 10.18185/erzifbed.848004.

[23] I. Fraser, W. Meier-Augenstein, and R. M. Kalin, “The role of stable isotopes in human identification: a longitudinal study into the variability of isotopic signals in human hair and nails,” *Rapid Communications in Mass Spectrometry*, vol. 20, no. 7, pp. 1109–1116, Mar. 2006, doi: 10.1002/rcm.2424.

[24] K. A. Zimmer, T. Clay, C. I. Vidal, S. Chaudhry, and M. Y. Hurley, “Microscopy of common nail cosmetics,” *American Journal of Dermatopathology*, vol. 39, no. 11, pp. 819–823, Nov. 2017, doi: 10.1097/dad.0000000000000809.

[25] P. Rich, “Nail Cosmetics and Esthetics,” *Skin Pharmacology and Physiology*, vol. 12, no. 3, pp. 144–145, Jan. 1999, doi: 10.1159/000029869.

[26] B. Galmés, G. Moyà-Alcover, P. Bibiloni, J. Varona, and A. Jaume-I-Capó, “Geometric-based nail segmentation for clinical measurements,” *Multimedia Tools and Applications*, vol. 81, no. 12, pp. 16117–16132, Mar. 2022, doi: 10.1007/s11042-022-12234-2.

[27] M. Alghamdi, P. Angelov, and L. P. Alvaro, “Person identification from fingernails and knuckles images using deep learning features and the Bray-Curtis similarity measure,” *Neurocomputing*, vol. 513, pp. 83–93, Nov. 2022, doi: 10.1016/j.neucom.2022.09.123.

[28] T. S. Indi and D. D. Patil, “Nail feature analysis and classification techniques for disease detection,” *International Journal of Computer Sciences and Engineering*, vol. 7, no. 5, pp. 1376–1383, May 2019, doi: 10.26438/ijcse/v7i5.13761383.

[29] S. A. Yamac, O. Kuyucuoglu, S. B. Koseoglu, and S. Ulukaya, “Deep learning based classification of human nail diseases using color nail images,” in *45th International Conference on Telecommunications and Signal Processing (TSP)*, Prague, Czechia, Jul. 13, 2022. doi: 10.1109/tsp55681.2022.9851300.

[30] R. Sharma, “Hand Shape Personality Test: Your hands reveal your true personality traits,” *jagranjosh.com*, Sep. 15, 2022. jagranjosh.com/general-knowledge/hand-shape-personality-test-your-hands-reveal-your-true-personality-traits-1663239231-1 (Accessed: Aug. 14, 2024).

[31] “How to find the best nail shape for your hands in 2024,” *kesterblack.co.nz.*, Jan. 01, 2024. kesterblack.co.nz/blogs/articles/how-to-find-the-most-flattering-nail-shape (Accessed: May 10, 2024).

[32] S. Ali *et al.*, “Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence,” *Information Fusion*, vol. 99, p. 101805, Nov. 2023, doi: 10.1016/j.inffus.2023.101805.

[33] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[34] A. Abdoullaev, “Artificial intelligence vs machine learning vs artificial neural networks vs deep learning,” *bbntimes.com*, Jul. 06, 2021. bbntimes.com/science/artificial-intelligence-vs-machine-learning-vs-artificial-neural-networks-vs-deep-learning (Accessed: Aug. 6, 2024).

[35] S. R. Pandey, J. Ma, C.-H. Lai, and P. R. Regmi, “A supervised machine learning approach to generate the auto rule for clinical decision support system,” *Trends in Medicine*, vol. 20, no. 3, Jan. 2020, doi: 10.15761/tim.1000232.

[36] S. Rajbanshi, “Everything you need to know about machine learning technology,” *medium.com*, Mar. 25, 2021. medium.com/machine-learning-community/everything-you-need-to-know-about-machine-learning-technology-f7e9b6ce07ce (Accessed: May 17, 2024).

[37] S. Kotsiantis, “Supervised Machine Learning: A review of classification techniques,” *Informatica*, vol. 31, pp. 249–268, Jun. 2007.

[38] J. R. Quinlan, “Induction of decision trees,” *Machine Learning*, vol. 1, no. 1, pp. 81–106, Mar. 1986, doi: 10.1007/bf00116251.

[39] S. R. Pandey, J. Ma, C.-H. Lai, and P. R. Regmi, “A supervised machine learning approach to generate the auto rule for clinical decision support system,” *Trends in Medicine*, vol. 20, no. 3, Jan. 2020, doi: 10.15761/tim.1000232.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, [Online]. Available: jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf

[41] R. Raj, “Supervised, unsupervised and semi-supervised learning with Real-Life Usecase,” *enjoyalgorithms.com*. enjoyalgorithms.com/blogs/supervised-unsupervised-and semisupervised-learning (Accessed: Jul. 31, 2024).

[42] Y. Zhou, “Advances of machine learning in multi-energy district communities–mechanisms, applications and perspectives,” *Energy and AI*, vol. 10, p. 100187, Nov. 2022, doi: 10.1016/j.egyai.2022.100187.

[43] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering,” *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, Sep. 1999, doi: 10.1145/331499.331504.

[44] Z. M. Nayeri, T. Ghafarian, and B. Javadi, “Application placement in Fog computing with AI approach: Taxonomy and a state of the art survey,” *Journal of Network and Computer Applications*, vol. 185, p. 103078, Jul. 2021, doi: 10.1016/j.jnca.2021.103078.

[45] Q. Zhang, Y. Liu, Y. Xiang, and T. Xiahou, “Reinforcement Learning in Reliability and Maintenance Optimization: A Tutorial,” *Reliability Engineering & System Safety*, vol. 251, p. 110401, Jul. 2024, doi: 10.1016/j.ress.2024.110401.

[46] N. Majumder, S. Poria, A. Gelbukh, and E. Cambria, “Deep Learning-Based Document Modeling for Personality Detection from Text,” *IEEE Intelligent Systems*, vol. 32, no. 2, pp. 74–79, Mar. 2017, doi: 10.1109/mis.2017.23.

[47] S. Kiliçarslan and M. Celik, “RSigELU: A nonlinear activation function for deep neural networks,” *Expert Systems With Applications*, vol. 174, p. 114805, Jul. 2021, doi: 10.1016/j.eswa.2021.114805.

[48] P. M. R. Bento, J. A. N. Pombo, M. R. A. Calado, and S. J. P. S. Mariano, “A bat optimized neural network and wavelet transform approach for short-term price forecasting,” *Applied Energy*, vol. 210, pp. 88–97, Jan. 2018, doi: 10.1016/j.apenergy.2017.10.058.

[49] P. Singh, P. Dwivedi, and V. Kant, “A hybrid method based on neural network and improved environmental adaptation method using Controlled Gaussian Mutation with real parameter for short-term load forecasting,” *Energy*, vol. 174, pp. 460–477, May 2019, doi: 10.1016/j.energy.2019.02.141.

[50] F. Bre, J. M. Gimenez, and V. D. Fachinotti, “Prediction of wind pressure coefficients on building surfaces using artificial neural networks,” *Energy and Buildings*, vol. 158, pp. 1429–1441, Jan. 2018, doi: 10.1016/j.enbuild.2017.11.045.

[51] S. K. Shetty and A. Siddiqa, “Deep learning algorithms and applications in computer vision,” *International Journal of Computer Sciences and Engineering*, vol. 7, no. 7, pp. 195–201, Jul. 2019, doi: 10.26438/ijcse/v7i7.195201.

[52] R. Yamashita, M. Nishio, R. K. Gian DO, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Into Imaging*, vol. 9, no. 4, pp. 611–629, Jun. 2018, doi: 10.1007/s13244-018-0639-9.

[53] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A Critical Review of Recurrent Neural Networks for sequence Learning,” *arXiv (Cornell University)*, Jan. 2015, doi: 10.48550/arxiv.1506.00019.

[54] A. Jabbar *et al.*, “Image-to-image translation based face photo de-meshing using GANs,” *Computer Vision and Image Understanding*, vol. 247, p. 104080, Jul. 2024, doi: 10.1016/j.cviu.2024.104080.

[55] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” *arXiv (Cornell University)*, Jan. 2014, doi: 10.48550/arxiv.1406.2661.

[56] R. Zhao, S. Tang, J. Shen, E. E. B. Supeni, and S. A. Rahim, “Enhancing autonomous driving safety: A robust traffic sign detection and recognition model TSD-YOLO,” *Signal Processing*, vol. 225, p. 109619, Jul. 2024, doi: 10.1016/j.sigpro.2024.109619.

[57] J. Du, “Understanding of object detection based on CNN Family and YOLO,” *Journal of Physics Conference Series*, vol. 1004, p. 012029, Apr. 2018, doi: 10.1088/1742-6596/1004/1/012029.

[58] S. Rasappan and P. Rajan, “An overview of linear algebra in image processing,” *Southeast Europe Journal of Soft Computing*, vol. 12, no. 2, pp. 72–77, Sep. 2023.

[59] A. Md. E. Arefin, “Proposal of a marine protected area surveillance system against illegal vessels using image sensing and image processing,” *Acta Ecologica Sinica*, vol. 38, no. 2, pp. 111–116, Apr. 2018, doi: 10.1016/j.chnaes.2017.06.015.

[60] E. Ferdian *et al.*, “Cerebrovascular super-resolution 4D Flow MRI – Sequential combination of resolution enhancement by deep learning and physics-informed image processing to non-invasively quantify intracranial velocity, flow, and relative pressure,” *Medical Image Analysis*, vol. 88, p. 102831, Aug. 2023, doi: 10.1016/j.media.2023.102831.

[61] R. Elstohy and E. M. Ali, “A flash flood detected area using classification-based image processing for sentinel-2 satellites data: A case study of Zafaraana Road at Red Sea,” *The Egyptian Journal of Remote Sensing and Space Science*, vol. 26, no. 3, pp. 807–814, Dec. 2023, doi: 10.1016/j.ejrs.2023.08.004.

[62] S. S. Sarikan and A. M. Ozbayoglu, “Anomaly Detection in Vehicle Traffic with Image Processing and Machine Learning,” *Procedia Computer Science*, vol. 140, pp. 64–69, Jan. 2018, doi: 10.1016/j.procs.2018.10.293.

[63] A. J. R. Neves, B. Barbosa, S. C. Soares, and I. D. Dimas, *Analysis of emotions from body postures based on digital imaging*. Third International Conference on Advances in Signal, Image and Video Processing, At: Nice: France, 2018.

[64] S. Albert *et al.*, “Comparison of image normalization methods for Multi-Site Deep Learning,” *Applied Sciences*, vol. 13, no. 15, p. 8923, Aug. 2023, doi: 10.3390/app13158923.

[65] M. Tuller, R. Kulkarni, and W. Fink, “Segmentation of X-Ray CT data of porous materials: A review of global and locally adaptive algorithms,” *SSSA Special Publication Series*, pp. 157–182, Oct. 2015, doi: 10.2136/sssaspecpub61.c8.

[66] T. J. Sejnowski, *The deep learning revolution*. 2018. doi: 10.7551/mitpress/11474.001.0001.

[67] Y. Qian, “Performance comparison among VGG16, InceptionV3, and resnet on galaxy morphology classification,” *Journal of Physics Conference Series*, vol. 2580, no. 1, p. 012009, Sep. 2023, doi: 10.1088/1742-6596/2580/1/012009.

[68] P. R. Patnaik, “Applications of neural networks to recovery of biological products,” *Biotechnology Advances*, vol. 17, no. 6, pp. 477–488, Nov. 1999, doi: 10.1016/s0734-9750(99)00013-0.

[69] M. Zolnouri, D. Lakhmiri, C. Tribes, E. Sari, and S. L. Digabel, “Efficient training under limited resources,” *arXiv (Cornell University)*, Jan. 2023, doi: 10.48550/arxiv.2301.09264.

[70] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. 2016. [Online]. Available: dl.acm.org/citation.cfm?id=3086952

[71] R. Fabricius and O. Šuch, “Detection of vowel segments in noise with ImageNet neural network architectures,” *Transportation Research Procedia*, vol. 55, pp. 1289–1295, Jan. 2021, doi: 10.1016/j.trpro.2021.07.112.

[72] L. Alzubaidi *et al.*, “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of Big Data*, vol. 8, no. 1, Mar. 2021, doi: 10.1186/s40537-021-00444-8.

[73] T. T. Khoei, H. O. Slimane, and N. Kaabouch, “Deep learning: systematic review, models, challenges, and research directions,” *Neural Computing and Applications*, vol. 35, no. 31, pp. 23103–23124, Sep. 2023, doi: 10.1007/s00521-023-08957-4.

[74] P. Dhiman, A. Kaur, D. Gupta, S. Juneja, A. Nauman, and G. Muhammad, “GBERT: a hybrid deep learning model based on GPT-BERT for fake news detection,” *Heliyon*, vol. 10, no. 16, p. e35865, Aug. 2024, doi: 10.1016/j.heliyon.2024.e35865.

[75] C. Leiter *et al.*, “ChatGPT: A meta-analysis after 2.5 months,” *Machine Learning With Applications*, vol. 16, p. 100541, Jun. 2024, doi: 10.1016/j.mlwa.2024.100541.

[76] A. Riasatian *et al.*, “Fine-Tuning and training of densenet for histopathology image representation using TCGA diagnostic slides,” *Medical Image Analysis*, vol. 70, p. 102032, May 2021, doi: 10.1016/j.media.2021.102032.

[77] J. Lemley, S. Bazrafkan, and P. Corcoran, “Transfer learning of temporal information for driver action classification.,” *MAICS*, pp. 123–128, Jan. 2017, [Online]. Available: ceur-ws.org/Vol-1964/NN1.pdf

[78] S. Ahuja, B. K. Panigrahi, N. Dey, V. Rajinikanth, and T. K. Gandhi, “Deep transfer learning-based automated detection of COVID-19 from lung CT scan slices,” *Applied Intelligence*, vol. 51, no. 1, pp. 571–585, Aug. 2020, doi: 10.1007/s10489-020-01826-w.

[79] N. Thompson, K. Greenewald, K. Lee, and G. F. Manso, *The Computational Limits of Deep Learning*. Ninth Computing within Limits, 2020. doi: 10.21428/bf6fb269.1f033948.

[80] A. Kumar, M. A. Shaun, and B. K. Chaurasia, “Identification of psychological stress from speech signal using deep learning algorithm,” *e-Prime - Advances in Electrical Engineering Electronics and Energy*, vol. 9, p. 100707, Sep. 2024, doi: 10.1016/j.prime.2024.100707.

[81] E. Baykal, H. Dogan, M. E. Ercin, S. Ersoz, and M. Ekinci, “Transfer learning with pre-trained deep convolutional neural networks for serous cell classification,” *Multimedia Tools and Applications*, vol. 79, no. 21–22, pp. 15593–15611, May 2019, doi: 10.1007/s11042-019-07821-9.

[82] S. J. Quan, “Comparing hyperparameter tuning methods in machine learning based urban building energy modeling: A study in Chicago,” *Energy and Buildings*, vol. 317, p. 114353, May 2024, doi: 10.1016/j.enbuild.2024.114353.

[83] A. Saboor, M. Usman, S. Ali, A. Samad, M. F. Abrar, and N. Ullah, “A method for improving prediction of human heart disease using machine learning algorithms,” *Mobile Information Systems*, vol. 2022, pp. 1–9, Mar. 2022, doi: 10.1155/2022/1410169.

[84] Z. Yang, R. O. Sinnott, J. Bailey, and Q. Ke, “A survey of automated data augmentation algorithms for deep learning-based image classification tasks,” *Knowledge and Information Systems*, vol. 65, no. 7, pp. 2805–2861, Mar. 2023, doi: 10.1007/s10115-023-01853-2.

[85] A. Latreche, R. Kelaiaia, A. Chemori, and A. Kerboua, “Reliability and validity analysis of MediaPipe-based measurement system for some human rehabilitation motions,” *Measurement*, vol. 214, p. 112826, Jun. 2023, doi: 10.1016/j.measurement.2023.112826.

[86] D. Ciraolo, M. Fazio, R. S. Calabrò, M. Villari, and A. Celesti, “Facial expression recognition based on emotional artificial intelligence for tele-rehabilitation,” *Biomedical Signal Processing and Control*, vol. 92, p. 106096, Jun. 2024, doi: 10.1016/j.bspc.2024.106096.

[87] A. Sharma, J. Pathak, M. Prakash, and J. N. Singh, “Object Detection using OpenCV and Python,” *2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, Dec. 2021, doi: 10.1109/icac3n53548.2021.9725638.

[88] A. Alvin, N. H. Shabrina, A. Ryo, and E. Christian, “Hand Gesture Detection for Sign Language using Neural Network with Mediapipe,” *Ultima Computing Jurnal Sistem Komputer*, vol. 13, no. 2, pp. 57–62, Dec. 2021, doi: 10.31937/sk.v13i2.2109.

[89] S. Alyami, H. Luqman, and M. Hammoudeh, “Isolated Arabic sign language recognition using a transformer-based model and landmark keypoints,” *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 23, no. 1, pp. 1–19, Jan. 2024, doi: 10.1145/3584984.

[90] “Hand tracking,” *apmonitor.com*, Nov. 15, 2023. apmonitor.com/pds/index.php/Main/HandTracking (Accessed: Aug. 14, 2024).

[91] “Python: Hand Landmark estimation with MediaPipe,” *techtutorialsx.com*, Apr. 10, 2021. techtutorialsx.com/2021/04/10/python-hand-landmark-estimation/ (Accessed: Aug. 14, 2024).

[92] M. Afifi, “11K Hands: Gender recognition and biometric identification using a large dataset of hand images,” *Multimedia Tools and Applications*, vol. 78, no. 15, pp. 20835–20854, Mar. 2019, doi: 10.1007/s11042-019-7424-8.

[93] T. T. Bilgin, S. B. Altınışık, and N. A. Adıgüzel, "A comparative study of classification and clustering methods for data analysis in digital transformation and IoT systems," *Orclever Proceedings of Research and Development*, vol. 3, no. 1, pp. 1–18, Dec. 2023, doi: 10.56038/oprd.v3i1.280.

[94] M. Heydarian, T. E. Doyle, and R. Samavi, "MLCM: Multi-Label Confusion Matrix," *IEEE Access*, vol. 10, pp. 19083–19095, Jan. 2022, doi: 10.1109/access.2022.3151048.

[95] T. P. Pagano *et al.*, "Bias and Unfairness in Machine Learning Models: A Systematic review on datasets, tools, fairness metrics, and identification and mitigation methods," *Big Data and Cognitive Computing*, vol. 7, no. 1, p. 15, Jan. 2023, doi: 10.3390/bdcc7010015.

[96] "Validation curve," *geeksforgeeks.org*, May 29, 2023. geeksforgeeks.org/validation-curve/ (Accessed: Aug. 3, 2024).

[97] "What is a learning curve in machine learning?," *baeldung.com*, Mar. 18, 2024. baeldung.com/cs/learning-curve-ml (Accessed: Aug. 3, 2024).

[98] T. Burzykowski, M. Geubbelmans, A.-J. Rousseau, and D. Valkenborg, "Validation of machine learning algorithms," *American Journal of Orthodontics and Dentofacial Orthopedics*, vol. 164, no. 2, pp. 295–297, Aug. 2023, doi: 10.1016/j.ajodo.2023.05.007.

[99] M. Kuroki, "Using Python and Google Colab to teach undergraduate microeconomic theory," *International Review of Economics Education*, vol. 38, p. 100225, Nov. 2021, doi: 10.1016/j.iree.2021.100225.

[100] J. P. Gujjar, H. R. P. Kumar, and N. N. Chiplunkar, "Image classification and prediction using transfer learning in colab notebook," *Global Transitions Proceedings*, vol. 2, no. 2, pp. 382–385, Nov. 2021, doi: 10.1016/j.gltp.2021.08.068.

[101] İ. Tarimer and B. C. Karadağ, "Genres Classification of popular songs listening by using keras," *Gazi University Journal of Science Part a Engineering and Innovation*, vol. 11, no. 1, pp. 123–136, Mar. 2024, doi: 10.54287/gujsa.1374878.

[102] P. H. Borcherds, “Python: a language for computational physics,” *Computer Physics Communications*, vol. 177, no. 1–2, pp. 199–201, Jul. 2007, doi: 10.1016/j.cpc.2007.02.019.

[103] M. Levy and F. Naiser, “Machine learning at the edge,” in *Elsevier eBooks*, 2019, pp. 549–601. doi: 10.1016/b978-0-12-809448-8.00015-1.

[104] “Fortran Programming with NumPy Arrays,” in *Texts in computational science and engineering*, 2007, pp. 451–482. doi: 10.1007/978-3-540-73916-6_9.

[105] P. Janardhanan, “Project repositories for machine learning with TensorFlow,” *Procedia Computer Science*, vol. 171, pp. 188–196, Jan. 2020, doi: 10.1016/j.procs.2020.04.020.

[106] A. L. S. Saabith, T. Vinothraj, and MMM. Fareez, “Popular Python libraries and their application domains,” *International Journal of Advance Engineering and Research Development*, vol. 7, no. 11, Nov. 2020.

[107] F. Pedregosa *et al.*, “SciKit-Learn: Machine Learning in Python,” *Journal of Machine Learning Research*, Nov. 2011, doi: 10.5555/1953048.2078195.

[108] A. H. Sial, S. Y. S. Rashdi, and A. H. Khan, “Comparative analysis of data visualization libraries MatPlotlib and Seaborn in Python,” *International Journal of Advanced Trends in Computer Science and Engineering*, vol. 10, no. 1, pp. 277–281, Feb. 2021, doi: 10.30534/ijatcse/2021/391012021.

[109] R. Yamashita, M. Nishio, R. K. Gian DO, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Into Imaging*, vol. 9, no. 4, pp. 611–629, Jun. 2018, doi: 10.1007/s13244-018-0639-9.

[110] S. Yang, J. Hu, H. Zhang, and G. Liu, “Simultaneous earthquake detection on multiple stations via a convolutional neural network,” *Seismological Research Letters*, vol. 92, no. 1, pp. 246–260, Oct. 2020, doi: 10.1785/0220200137.

[111] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014, [Online]. Available: jmlr.csail.mit.edu/papers/volume15/srivastava14a/srivastava14a.pdf

[112] K. Simonyan and A. Zisserman, “Very deep convolutional networks for Large-Scale image recognition,” *Computer Vision and Pattern Recognition*, Sep. 2014, [Online]. Available: export.arxiv.org/pdf/1409.1556