Undergraduate Final Project

A CNN model to Classify Traditional Ethnic Clothing Style Garments

Larissa Duarte Santana
lds@ic.ufal.br

Advisor:
Márcio de Medeiros Ribeiro

Maceió, July of 2024

Larissa Duarte Santana

A CNN model to Classify Traditional Ethnic Clothing Style Garments

Thesis presented as a partial requirement for obtaining the degree of Bachelor of Computer Engineering from the Institute of Computing at the Federal University of Alagoas.

Advisor:

Márcio de Medeiros Ribeiro

Maceió, July of 2024

Catalogação na Fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecário: Marcelino de Carvalho Freitas Neto – CRB-4 - 1767

S232c Santana, Larissa Duarte.
A CNN model to classify traditional ethnic clothing style garments / Larissa Duarte Santana. – 2024.
73 f. : il.

Orientador: Márcio de Medeiros Ribeiro.
Monografia (Trabalho de conclusão de curso em Engenharia de Computação) - Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2024.

Bibliografia: f. 61-62.
Apêndices: f. 63-73.

1. Inteligência artificial. 2. Aprendizagem profunda. 3. Redes neurais. 4. Redes neurais convolucionais. 5. Moda - Classificação. 6. Trajes étnicos. I. Título.

CDU: 004.81:159.953.5

Acknowledgements

I would first like to thank my parents for always encouraging my studies. Education has always been a primary focus in my household. Next, I would like to thank Professor Márcio Ribeiro for guiding and supporting me in this final stage of my graduation, especially for his willingness to accept a more unconventional topic like this. Furthermore, I would like to thank all my professors and colleagues who were part of my graduation process, whether it was attending classes together or providing support with materials and motivation. Finally, I would like to thank all the important people in my life, regardless of their connection with this institution, for the moments of relaxation and fun in everyday life. A happy person thrives not only through studies, academic, and professional life but also through the joy and balance brought by personal relationships.

Abstract

The classification of traditional garments is in the early stages of exploration within the academic study of fashion technology, with relatively few studies dedicated to this specific area. Despite the growing interest in applying computer vision techniques to the fashion industry, current research predominantly focuses on modern fashion styles, leaving a need for more attention to culturally significant traditional clothing. This research aims to address this need by utilizing Convolutional Neural Networks (CNNs) to classify feminine traditional ethnic clothing styles, specifically garments from four nationalities: German (dirndl), Indian (saree), Japanese (kimono), and Spanish (flamenco). By leveraging deep learning techniques, particularly CNNs, we seek to enhance the accuracy and efficiency of fashion classification systems for traditional garments. Our study involves the construction of a diverse dataset with four nationality labels with a total of 5,879 images and the developing of a novel classifier model, preceded by a comprehensive literature review that examines existing classification techniques and the contexts in which various ethnic clothing types have been studied academically. This review underscores the limited but growing work focusing on traditional garments. We empirically evaluate the performance of our model against established models, comparing its effectiveness in recognizing and categorizing traditional clothing styles from different cultures.As result, we achieved an accuracy rate of 94%, and loss 21% demonstrating the model's effectiveness in classifying traditional ethnic garments. Additionaly, observing from the confusion matrix, the model classified correctly from the test dataset german (95.88%), indian (97.54%), japanese (92.59%) and spanish (89.22%). These findings are expected to contribute significantly to computer vision and fashion technology, offering new insights and practical applications for the fashion industry. By focusing on traditional garments, our work not only enhances the technical capabilities of fashion classification systems but also promotes a deeper appreciation and understanding of diverse cultural heritages through technology.

Keywords: Artificial intelligence, Deep Learning, Neural Networks, Convolutional Neural Networks, Classification, Fashion, Traditional Garments.

Resumo

A classificação de vestimentas tradicionais está nos estágios iniciais de exploração dentro do estudo acadêmico da tecnologia da moda, com relativamente poucos estudos dedicados a essa área específica. Apesar do crescente interesse em aplicar técnicas de visão computacional na indústria da moda, a pesquisa atual foca predominantemente em estilos de moda moderna, deixando a necessidade de mais atenção para roupas tradicionais culturalmente significativas. Esta pesquisa visa atender a essa necessidade utilizando Redes Neurais Convolucionais (CNNs) para classificar estilos de vestimentas femininas tradicionais étnicas, especificamente roupas de quatro nacionalidades: alemã (dirndl), indiana (sari), japonesa (quimono) e espanhola (flamenco). Ao aproveitar técnicas de aprendizado profundo, particularmente CNNs, buscamos melhorar a precisão e a eficiência dos sistemas de classificação de moda para vestimentas tradicionais. Nosso estudo envolve a construção de um conjunto de dados diversificado com quatro etiquetas de nacionalidade, totalizando 5.879 imagens, e o desenvolvimento de um novo modelo de classificador, precedido por uma revisão abrangente da literatura que examina as técnicas de classificação existentes e os contextos nos quais vários tipos de vestimentas étnicas foram estudados academicamente. Esta revisão destaca o trabalho limitado, mas crescente, focado em vestimentas tradicionais. Avaliamos empiricamente o desempenho do nosso modelo em comparação com modelos estabelecidos, comparando sua eficácia em reconhecer e categorizar estilos de vestimentas tradicionais de diferentes culturas. Como resultado, alcançamos uma taxa de precisão de 94% e uma perda de 21%, demonstrando a eficácia do modelo na classificação de vestimentas tradicionais étnicas. Além disso, observando a matriz de confusão, o modelo classificou corretamente o conjunto de testes para as etiquetas alemã (95,88%), indiana (97,54%), japonesa (92,59%) e espanhola (89,22%). Esses achados são esperados para contribuir significativamente para a visão computacional e a tecnologia da moda, oferecendo novos insights e aplicações práticas para a indústria da moda. Ao focar em vestimentas tradicionais, nosso trabalho não só aprimora as capacidades técnicas dos sistemas de classificação de moda, mas também promove uma apreciação e compreensão mais profundas das diversas heranças culturais através da tecnologia.

Palavras-chave: Inteligência Artificial, Aprendizado Profundo, Redes Neurais, Redes Neurais Convolucionais, Classificação, Moda, Vestimentas Tradicionais.

Contents

1 Introduction **1**

2 Background **4**
2.1 Fashion . . . 4
2.2 Deep Learning . . . 6
2.3 Computer Vision . . . 8
2.4 Neural Networks . . . 10
2.4.1 Perceptron . . . 11
2.4.2 Activation Functions . . . 13
2.4.3 Backpropagation . . . 14
2.5 Training Neural Networks . . . 16
2.5.1 Training Dataset . . . 16
2.5.2 Pre-processing . . . 16
2.5.3 Epochs . . . 19
2.5.4 Cost Functions . . . 20
2.6 Evaluating Neural Networks . . . 21
2.6.1 Validation and Test Datasets . . . 21
2.6.2 Metrics . . . 21
2.6.3 Confusion Matrix . . . 25
2.6.4 Learning Curves . . . 26
2.7 Convolutional Neural Networks . . . 30
2.7.1 Convolutional Neural Networks Layers . . . 31

3 A CNN model to Classify Traditional Garments **35**
3.1 Tools and Libraries . . . 35
3.2 Knowledge Discovery in Databases Process . . . 37
3.3 Dataset . . . 38
3.4 Pre-processing . . . 40
3.4.1 Dataset path . . . 41
3.4.2 Train, Validation, and Test . . . 41
3.4.3 Redefinition and Normalization . . . 42
3.4.4 Data Augmentation . . . 43
3.5 Creating the Convolutional Neural Network Model . . . 44
3.5.1 Model Architecture . . . 45
3.5.2 Overview of the Layers . . . 47
3.5.3 Compiling and Training the Model . . . 48
3.6 Saving the Model . . . 49
3.7 Evaluating the Model . . . 50

3.7.1 Metrics Values . . . 50
3.7.2 Learning Curves . . . 50
3.7.3 Confusion Matrix . . . 51
3.8 Results and Discussions . . . 51
3.8.1 Metrics Values . . . 51
3.8.2 Learning Curves . . . 52
3.8.3 Confusion Matrix . . . 52

4 Related Work **54**

5 Concluding Remarks **57**
5.1 Analysis of Results . . . 57
5.2 Limitations . . . 58
5.3 Implications for Practice . . . 58
5.4 Future Work . . . 60

References **61**

A List of Websites used to create our dataset **63**

B Some examples for each label present in our Dataset **66**

C Important GitHub Links **71**

D Hugging Face Model **72**

1

Introduction

Fashion plays a crucial role in contemporary society, serving as a form of personal expression and cultural identity. It allows individuals to convey their personality, mood, and values through the clothes they choose to wear Entwistle (2000). Besides, fashion is a significant industry driving the global economy by creating jobs and fostering innovation in design, materials, and production technologies Kawamura (2005). Overall, by reflecting social, economic, and political trends, fashion acts as a mirror of our times, capturing changes and cultural movements Lipovetsky (1994).

Moreover, fashion can be classified in various ways, with one prominent category being Classic fashion. This type of fashion is enduring, with customers typically owning several such items that are suitable for various occasions. Often becoming the most promotable, it maintains its presence in the fashion scene regardless of current trends Bruzzi and Gibson (2000). Additionally, traditional costumes considered a classic style are a vital aspect of cultural heritage and essential elements for preserving a community's cultural and historical heritage Craik (2009).

Furthermore, with the advent of technology and the widespread availability of the internet, fashion has transcended geographical boundaries, becoming accessible to a global audience Jenkins (2006). Consequently, every aspect of fashion interpretation is indispensable for the industry, and understanding the longevity of fashion's relevance is crucial Agins (1999). Accurately classifying traditional ethnic garments presents several challenges and opportunities, especially in terms of cultural significance. Nonetheless, it is potentially essential for various applications, such as e-commerce and personalized recommendations.

The chosen garment types for this study include the dirndl, saree, kimono, and flamenco. These clothing styles, despite their traditional aspects, are well-known and recognizable, which aids in making an initial impact and facilitates the search for materials. Considering this initial work in the classification field, the dirndl is noted for its distinctive apron and bodice, the saree for its elegant drape and intricate patterns, the kimono for its flowing fabric and detailed designs, and the flamenco dress for its vibrant colors and adorned ruffles.

By selecting these iconic garments, this study not only leverages their cultural significance and popularity but also highlights the diversity and richness of traditional clothing across different regions. This approach ensures that the model developed in this research is trained on a variety of styles, each with unique features and historical importance. Furthermore, these garments' widespread recognition provides a robust foundation for future research, making it easier to obtain comprehensive datasets and garner interest from both the academic community and the fashion industry. Ultimately, this selection underscores the potential of technology to preserve and celebrate cultural heritage through advanced image classification techniques.

Despite significant recent advancements, the field of fashion in computer vision is still in its early stages. While there are numerous studies on general fashion recognition, few have delved into the nuances of traditional ethnic clothing, which often involves complex patterns and culturally specific details that modern fashion does not typically exhibit. Consequently, there is a notable gap in the availability of diverse datasets and AI models capable of performing this task. This research aims to address this gap by developing a robust model for the classification of traditional garments, thereby contributing valuable insights and tools to the field.

This gap is evident when examining the Hugging Face, a popular repository for datasets and machine learning models. Currently, there are around 180 thousand datasets on the website. When searching for image datasets this number goes to around 19 thousand. And filtering for fashion-related datasets this number goes to around a hundred. When searching for the specific garments used in this study the only one that gets some results is the "sarre", with only two which are not that big and considerably recent (see "Saree_dataset" and "Saree-NIFT-Style").

Additionally, currently, there are around 750 thousand models available on Hugging Face, but this number decreases to 13 thousand when filtering for Image Classification Models. Further narrowing the search to models related to "fashion" reduces the number to 46. Crucially, when searching for terms specific to this study's traditional garments, the number of relevant models is zero. Even the search for "ethnic" yields only one model, which is not specific to fashion but rather tests for ethnicity (see Ethnicity Test v003).

Other popular platforms like Kaggle, OpenML, and TensorFlow Model Garden offer a vast array of datasets and pre-trained models for modern fashion and general image classification tasks. However, finding models and datasets dedicated to traditional garments such as dirndls, sarees, kimonos, or flamenco dresses is rare. This analysis underscores the significant lack of specialized AI models and diverse datasets for identifying traditional clothing styles from different cultures, highlighting the need for research and development in this area.

Diverse datasets are crucial for training robust AI models that can accurately identify and classify garments from various cultures. Without access to diverse and representative datasets, the development of effective AI models for traditional clothing remains hindered. Moreover, the scarcity of pre-trained models that are specifically tailored for traditional garments means that researchers and developers must often start from scratch, significantly increasing the time and effort required to develop effective solutions.

By creating and sharing more specialized datasets and models, the academic and professional communities can better support the advancement of AI in this niche but culturally important area. Such efforts would not only enhance the technical capabilities of fashion classification systems but also promote a deeper appreciation and understanding of diverse cultural heritages through technology. As the field progresses, it is essential to bridge these gaps to ensure that the benefits of AI and deep learning are accessible and applicable to all areas of fashion, including the rich and varied world of traditional garments.

Given this context, our work aims to contribute to the AI community by addressing these gaps through the following initiatives:

- **Dataset:** Making a diverse dataset of traditional clothing with four labels representing nationality and their respective styles: German (dirndl), Indian (saree), Japanese (kimono), and Spanish (flamenco). This dataset will be curated to ensure it represents the intricacies and variations within each traditional style.
- **Model:** Developing and training a model capable of classifying these four distinct clothing styles from Germany, India, Japan, and Spain. Specifically a Keras Convolutional Neural Network Model.
- **Hugging Face Project:** Sharing the trained model on the Hugging Face website will enable other researchers and developers to build upon our work utilizing deep learning techniques. Overall, we hope to facilitate further advancements in the field and encourage more research focused on traditional garment classification.

As a result, a dataset of traditional clothing with a total of 5,879 images was compiled, and distributed across four labels and their respective styles. Including 1,388 images of dirndls, 1,744 images of sarees, 1,548 images of kimonos, and 1,199 images of flamenco dresses.

And with the trained model, we achieved an accuracy rate of 94% and a loss of 21%, demonstrating the model's effectiveness in classifying traditional ethnic garments. Additionally, the confusion matrix, a tool used to evaluate the performance of a classification model by providing a detailed breakdown of the model's predictions in comparison to the actual labels, offers valuable insight into the model's performance across different categories. It revealed that the model correctly classified 95.88% of the German garments, 97.54% of the Indian garments, 92.59% of the Japanese garments, and 89.22% of the Spanish garments.

These results underscore the potential of AI in advancing fashion technology and preserving cultural heritage through accurate and automated garment classification. Applications of such technology include virtual try-on systems, digital archiving, enhanced fashion recommendation engines, and educational tools that promote cultural awareness and appreciation. Overall, by leveraging AI, we can bridge the gap between modern fashion trends and traditional garments, ensuring that cultural heritage is preserved and celebrated in the digital age.

2

Background

This chapter delves into the foundational concepts necessary for understanding this undergraduate final project. It begins with an exploration of fashion, specifically focusing on traditional clothing styles. The chapter then transitions into the topic of deep learning, discussing its history and its correlation with Artificial Intelligence (AI) and Machine Learning (ML). Following this, it covers the topic of Computer Vision, emphasizing the study's focus on image classification within the fashion industry. The chapter further deepens the reader's understanding of Neural Networks by examining their architecture and various techniques for training and evaluating them. Finally, the chapter concludes with an introduction to Convolutional Neural Networks (CNNs), the model utilized in this research.

2.1 Fashion

Fashion itself can be classified into the following ways: Style, Fashion forecasting, Fad, and Basic or Classic. Style is the basic outline of any garment, and it is unique, while fashion is always changing. It can be defined as a manner of doing something, and when accepted by others, it is called fashion. Fashion forecasting is a category that predicts consumers' moods, behaviors, and buying habits. Therefore it focuses on upcoming trends, being an important part of the fashion scene and spread by many communicating media such as fashion shows, cinema, press, window displays, and nowadays more than ever on social media.

Moreover, Fad can be defined as a short-lived fashion, staying for a very brief period. It is commonly accepted by only a certain group of people, and very costly, with not everyone being able to afford it. Whereas, Basic or Classic are long-lasting fashion. It is comparatively constant and a customer has one or more in his wardrobe, to be worn to suit different occasions. In certain times, the basic becomes the most important promotable fashion, and regardless, they remain a part of the fashion scene.

That being said, understanding the longevity of fashion's relevance is paramount for fashion designers and manufacturers. By pinpointing this characteristic, they can effectively gauge its significance in retail inventory management. Moreover, with the advent of technology and the ubiquitous presence of the internet, fashion has transcended geographical boundaries, becoming accessible to a global audience. Thus, every aspect of fashion interpretation is indispensable for the industry, and many other sections can be optimized, such as product search, recommendation systems, marketing campaigns, product categorization, and online shopping experiences in general.

In recent years, fashion image analysis has emerged as a dynamic area of research, driven by its immense potential to revolutionize the industry. And this study seeks to enrich the analysis of classic fashion types. Therefore, we have chosen to concentrate on traditional clothing styles, focusing on four nationalities: German, Indian, Japanese, and Spanish. Table 2.1 provides an overview of the characteristic garment types associated with each nationality. It is essential to note that the classification process in the study, dictates each nationality the clothing represents addressing only feminine clothes.

Nationality	German	Indian	Japanese	Spanish
Garment Style	dirndl	sarres	kimono	flamenco

Table 2.1: Garment styles for each nationality approached

Briefly discussing the above clothing types, the Dirndl is a feminine garment with roots in German-speaking Alpine regions that hold deep cultural significance in Bavarian traditions, symbolizing heritage, tradition, and pride. This traditional dress once represented as workwear, is now trendier than ever becoming synonymous with Oktoberfest and other Bavarian festivities. It has always been known as a three-piece garment, consisting of the Dirndl dress, the apron, and the blouse.

The Saree is said to be the most representative traditional clothing of the Indian subcontinent. They represent the region's culture in a beautifully fashionable way, and even when fashion trends move towards, utilizing Western concepts, a combination with the beauty of Indian wear is maintained, creating what is called fusion wear. The garment consists of a drape that can be from 4.5 meters to 8 meters long, and about 1 meter wide, that can be worn in 80 different ways. But the usual way is wrapped around the body with the end left hanging or used over the head as a hood.

Likewise, Kimonos are a traditional Japanese garment that captures the Japanese culture and design very well. It is considered the national dress of Japan and for generations, has been defining this nation's style. They are made in a range of styles and patterns and are usually dictated by a range of specific criteria, including gender, marital status, and event. Finally, they are typically hand-sewn into a "T" shape from four single pieces of fabric called tans and tied with a belt called obi.

Moreover, few regional costumes are as recognizable as the flamenco dress. It is worn by women in the fairs and pilgrimages that are celebrated in Andalusia, a Spanish region, But also on special occasions or events, such as traditional dance performances. Considering the dress structure, it is usually long and has a neckline, either round, v-neck, or square, adjusting at the waist to open at the hip. It is typically adorned with ruffles on the sleeves and skirt and is usually in bright colors. And, we can find them with plain designs, but usually, they are printed. These four types of garments are exemplified in Figure 2.18.

Figure 2.1: Examples of traditional garment styles

2.2 Deep Learning

Creating machines that think has always been an obsession in the human mind[. This] is noticed in very early fiction. For instance, the novel “Do Androids Dream of Electric Sheep?” by Philip K. Dick (1968), which afterward inspired the film “Blade Runner”, explores the theme of artificial beings known as androids who possess human-like intelligence and emotions. Another notable example is the short story “The Sandman” by E.T.A. Hoffmann (1816), where the protagonist Nathanael becomes obsessed with a mechanical doll named Olympia, whom he believes possesses human-like intelligence and emotions.

Going even further, in ancient Greece, there are philosophical discussions about the creation of artificial beings and the concept of automata. Similarly, in ancient China, there are legends and stories about automated figures and mechanical devices that display human-like behavior. While these early pieces of evidence may not align directly with modern artificial intelligence research and technology, they demonstrate early attempts to conceptualize and explore the idea of artificial beings and automated systems.

Considering early scientific developments in the field of Artificial Intelligence. We had in 1956, a program called “Logic Theorist”, one of the first AI, it was designed by Allen Newell and Herbert A. Simonto to mimic the problem-solving skills of a human, particularly in the

domain of symbolic logic. Around the same year, John McCarthy coined the term "Artificial Intelligence" and organized the Dartmouth Conference, which is often considered the birth of AI as a field of study. It brought together researchers from various disciplines to discuss the possibility of creating intelligent machines and laid the groundwork for future research.

Ironically, while problems that can be described by a list of formal, mathematical rules are very easy for a computer, problems that we solve intuitively, and feel automatic, like recognizing spoken words or faces in images are the hardest for machines. Thus why in its early days of operation, artificial intelligence, rapidly addressed and solved problems that are intellectually difficult for human beings but relatively straightforward for computers such as the defeat of the best human chess player in 1997, by the algorithm Deep Blue Developed by IBM.

Still in the realm of AI, systems that depended on pre-programmed information, eventually, started to experience challenges. Thus, the necessity for systems to possess the capacity to gain knowledge autonomously was placed and explored even further, emerging the field we know as machine learning. Accordingly, its necessity surges for scenarios where it is complicated to use more traditional methods or more than one algorithm is needed to solve a problem. So, in dynamic environments that require constant adaptation and manual changes would be quite costly and time-consuming. Nevertheless, the effectiveness of these machine learning techniques relies significantly on how the data they receive is structured and presented since pattern identification happens from unprocessed data. These specific attributes or aspects are termed "features".

Some of the earliest examples of machine learning algorithms include Rosenblatt's "Perceptron", a significant milestone in the field developed in 1957 capable of binary classification tasks. And, decision tree algorithms such as the CART algorithm introduced by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in 1984, and the ID3 algorithm developed by J. Ross Quinlan in 1986. These are among the first to automate decision-making based on input data and paved the way for more complex models.

Furthermore, due to the increase in data availability, advances in computational capacity, and the need to address even more complex and large-scale problems the field of deep learning emerged, a particular type of machine learning. A deep learning model achieves great power and flexibility by learning to represent the world as a nested hierarchy of concepts, where each concept is defined with simpler ones. This way, it expands upon traditional machine-learning techniques by introducing more complex and hierarchical models of learning inspired by the structure and function of the human brain, known as artificial neuron networks.

Some key milestones in deep learning, include the Neocognitron, a hierarchical artificial neural network capable of pattern recognition and inspired by the organization of the visual cortex in the human brain, introduced by Kunihiko Fukushima in 1980. And, the backpropagation algorithm, developed independently by multiple researchers in 1986 including Geoffrey Hinton, David Rumelhart, and Ronald Williams, that allowed for efficient training of deep neural networks by adjusting the weights of connections between neurons. These concepts will be

further explained in future sections.

In summary, while Artificial Intelligence encompasses the broader goal of creating intelligent systems, Machine Learning, and Deep Learning are specific approaches within the AI field. Machine Learning focuses on training algorithms to learn from data and make predictions and often requires feature engineering and domain expertise, while Deep Learning extends this concept by leveraging complex neural network architectures to learn hierarchical representations of data automatically from raw input. Nonetheless, the research and development in these fields often happen simultaneously, as each approach deals with problems that can be better addressed by its methodologies. Below in Figure 2.2 is a representation of these fields' correlation.

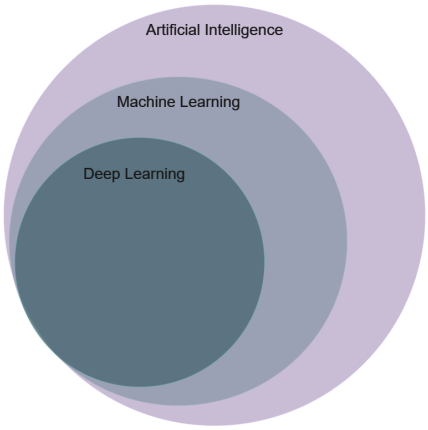


Figure 2.2: Diagram representing the correlation between the AI fields

2.3 Computer Vision

Computer vision is a field of artificial intelligence and computer science that focuses on enabling computers to interpret and understand visual information from the real world. It involves developing algorithms and techniques that allow computers to analyze and extract meaningful insights from digital images or videos. Its ultimate goal is to enable computers to "see" and understand the visual world in a similar manner to humans.

Therefore, there has been a surge of new research dedicated to exploiting deep learning in computer vision, particularly in enhancing classification results on large datasets such as "ImageNet" Deng et al. (2009) or "Places" Zhou et al. (2017). The first is a widely used dataset consisting of millions of labeled images across thousands of categories, serving as a benchmark for evaluating the performance of image classification algorithms. The second is a dataset containing millions of images categorized into scenes and locations, providing valuable training data for tasks such as scene recognition and localization.

For a comprehensive understanding of the algorithms and applications in computer vision, the book by Szeliski Szeliski (2010) is an invaluable resource. It covers a wide range of topics,

from fundamental principles to advanced techniques in the field, making it essential reading for anyone interested in computer vision.

In Figure 2.3 we see the Computer Vision Field correlation between the AI fields, as well as CNN, Convolutional Neural Networks, the specific type of deep learning model used in our study. This will be approached further in this chapter.

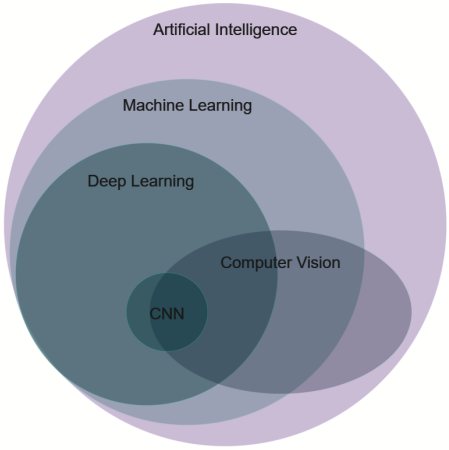


Figure 2.3: Diagram representing the correlation between the AI fields

As mentioned previously, over the past decades, the fashion industry has gone through significant changes, leading to its growth and development. With this, it is difficult for people to recognize the latest fashion variations because of many types and trends. Besides, the fashion industry has difficulty understanding customer tastes, and directing sales to be better is a way to increase profits. All this is also culminated by the rise of the buying and selling process through digital channels. Consequently, the interest in fashion has been growing in the computer vision community.

As a result, large-scale datasets such as "FashionMNIST" Xiao et al. (2017) and "DeepFashion" Liu et al. (2016) have become integral resources for training and evaluating fashion-related computer vision models. FashionMNIST, similar to ImageNet in the general domain, comprises tens of thousands of labeled images depicting various fashion items across multiple categories, including tops, dresses, shoes, and accessories. This dataset serves as a benchmark for assessing the performance of image classification algorithms tailored to fashion-specific tasks.

On the other hand, DeepFashion encompasses millions of images spanning diverse fashion scenes, styles, and contexts. Categorized meticulously, these images provide valuable training data for tasks such as fashion recognition, clothing attribute prediction, and fashion recommendation systems. Together, these datasets play a pivotal role in advancing the capabilities of computer vision in the domain of fashion, enabling researchers to develop more accurate and robust algorithms.

Nevertheless, computer vision has demonstrated remarkable success in many applications in this domain, including more complex ones like trend forecasting, the creation of capsule

wardrobes, interactive product retrieval, recommendation, and fashion design. For instance, Swain et al. Swain et al. (2023) developed an intelligent fashion object classification system using convolutional neural networks (CNNs) to enhance trend forecasting and product retrieval. Kayed et al. Kayed et al. (2020) applied the LeNet-5 architecture to classify garments from the Fashion MNIST dataset, demonstrating its effectiveness in semantic segmentation of garments. Nawaz et al. Nawaz et al. (2018) focused on the automatic categorization of traditional clothing using CNNs, showing significant improvements in image retrieval. And Aditya et al. Aditya et al. (2023) implemented a CNN method for identifying fashion images, which has been instrumental in advancing the classification of garments and styles, which is the focus of this research.

2.4 Neural Networks

For now let us concentrate on specifying our knowledge in deep learning. To better understand this field, we are about to detail its components and operation. Deep learning algorithms are typically characterized by the use of artificial neural networks with multiple layers. Its concepts are built on top of each other, in "deep" graph visualizations, hence the term in its terminology.

Also, these networks are comprised of different node layer types as seen in Figure 2.4. An input layer, one or more hidden layers, and an output layer. These hidden layers are defined this way because their values are not given in the data, instead, the model must determine which concepts are useful for explaining the relationships in the observed data.

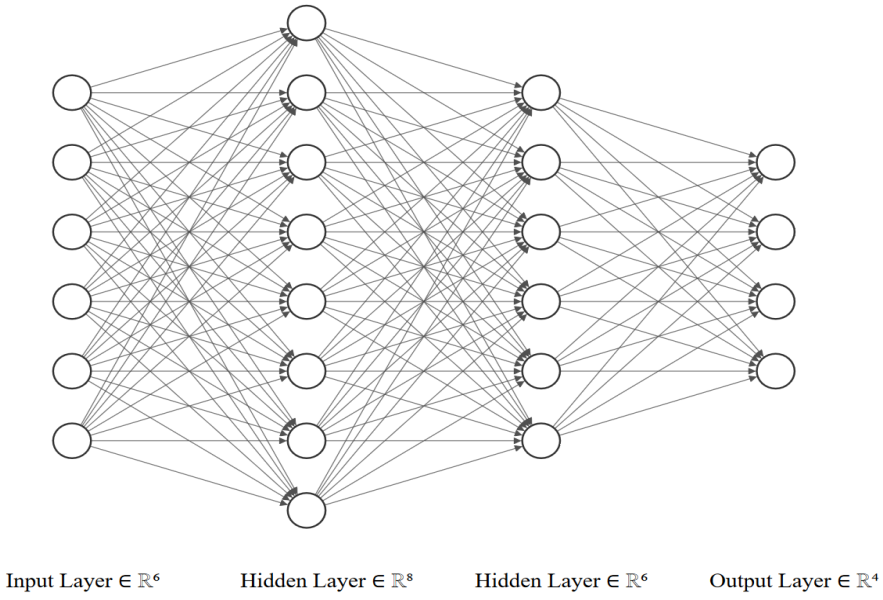


Figure 2.4: Representation of a neural network

Each node, also known as an artificial neuron, is interconnected with others and possesses an

associated weight and threshold. These weights and thresholds play crucial roles in determining how information flows through the network.

2.4.1 Perceptron

To understand these parameters and clarify how they work in a node, we can observe the internal process of a Perceptron. This algorithm, as mentioned is one of the earliest examples of machine learning, and can be considered a baseline for an artificial neuron.

A neuron implements a two-stage process to map inputs to an output. The first stage involves the calculation of a weighted sum of inputs. This sum is, then, passed through a second function, called the unit step function or activation function, that maps the results to the neuron's final output value. In the perceptron's case, a binary classification result.

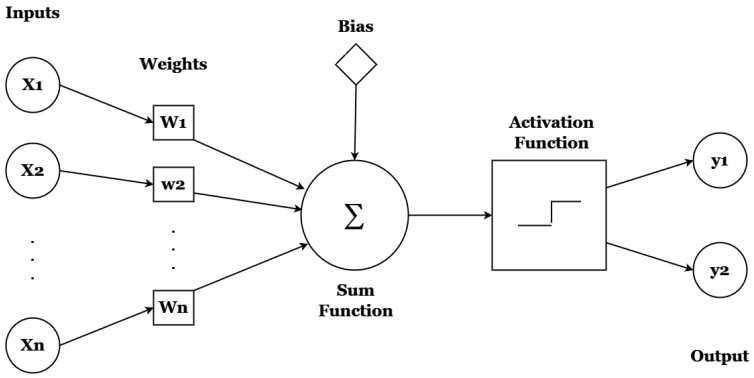


Figure 2.5: Perceptron Representation

As can be seen, the neuron in Figure 2.5 receives "n" inputs on different input connections, and each connection has an associated weight. Therefore, it has two vectors: X = [x1, x2, ..., xn] and W = [w1, w2, ..., wn]. The weighted sum calculation involves the multiplication of inputs by weights and the summation of a bias or threshold.

Mathematically this calculation is written as:

$$\begin{aligned} z &= X \cdot W + b \\ &= \sum_{i=1}^{n} w_i \cdot x_i + b \\ &= (w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n) + b \end{aligned}$$

Furthermore, this unit step function involves setting 0 or 1 as an output, depending on the result of the weighted sum, on 0 the node will be deactivated, correspondingly on 1 activated.

Mathematically this process is represented like this:

$$f(x) = \begin{cases} 1 & \text{if } W \cdot X + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

This mentioned bias term is added to the input layer to provide the perceptron with additional flexibility in modeling complex patterns in the input data. For simplicity, this bias or threshold can be brought to the left and represented as an input and weight pair, w0.x0.

Mathematically this calculation is now written as:

$$\begin{aligned} z &= X \cdot W \\ &= \sum_{i=1}^{n} w_i \cdot x_i \\ &= w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n \end{aligned}$$

In this representation, w0 = -θ and x0 = 1.

The value w0 is called the bias unit or threshold and is mathematically represented as:

$$\theta(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

Then, we have in Figure 2.6 the representation of the simplified diagram version. And, consequentially, the mathematical decision function becomes:

$$f(x) = \begin{cases} 1 & \text{if } W \cdot X > 0 \\ 0 & \text{otherwise} \end{cases}$$

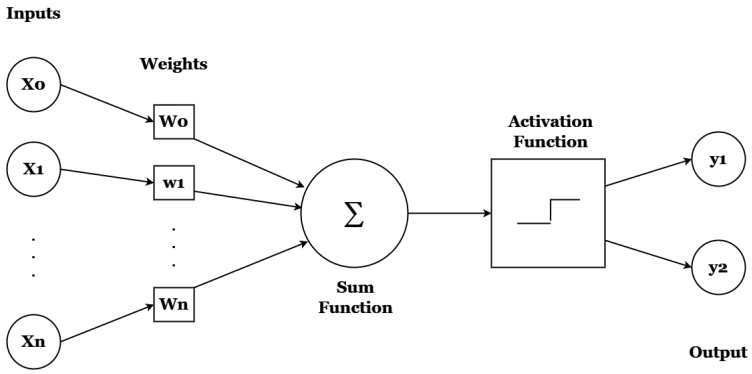


Figure 2.6: Simplified Perceptron Representation

2.4.2 Activation Functions

Essentially, activation functions serve to introduce non-linearity into the output of a neuron, allowing the model to have complex relationships within the data, thus enhancing its capacity to learn and model intricate patterns. The term stems from its role in activating or "firing" a neuron based on its input.

There are several types of activation functions. Each one possesses unique characteristics, making them suitable for different tasks and yielding varying performance results. Furthermore, the choice of an activation function profoundly influences the performance and behavior of a neural network. Hence, understanding their characteristics and suitability for different tasks is essential for designing effective neural architectures.

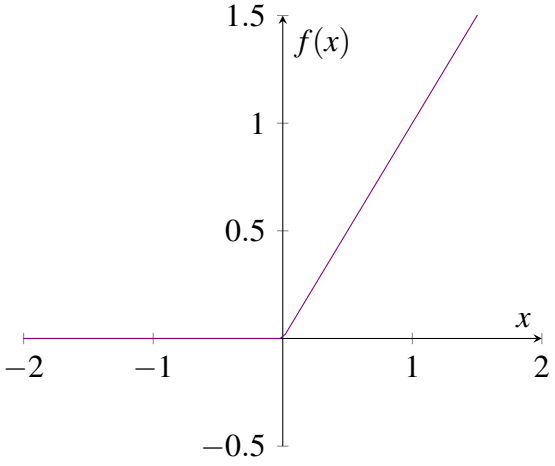
ReLU

The function Rectified Linear Unit (ReLU), is favored for its simplicity and effectiveness. It sets negative inputs to zero and passes positive inputs unchanged, offering efficient training. Since it lets positive inputs remain unchanged, big numbers pass through. This makes a few neurons stale and they do not fire, in other words, neurons do not fire all the time, increasing sparsity and making the train faster. Also, since the function is simple, it is computationally the least expensive.

The ReLU function introduces non-linearity by mapping all negative input values to zero and leaving positive values unchanged.

Choosing the activation function is very dependent on the application. Nevertheless, ReLU works well for a large range of problems.

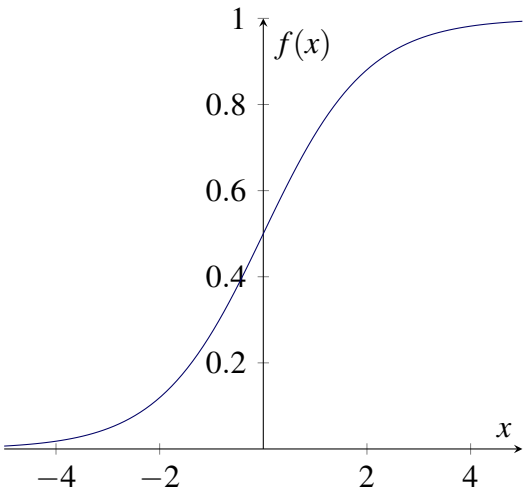
$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



Sigmoid

One fundamental activation function is the sigmoid function. It squashes its inputs into a range between 0 and 1, making it particularly useful in binary classification tasks where the output needs to be interpreted as probabilities. Sigmoid can be considered a smoothened step function and hence differentiable.

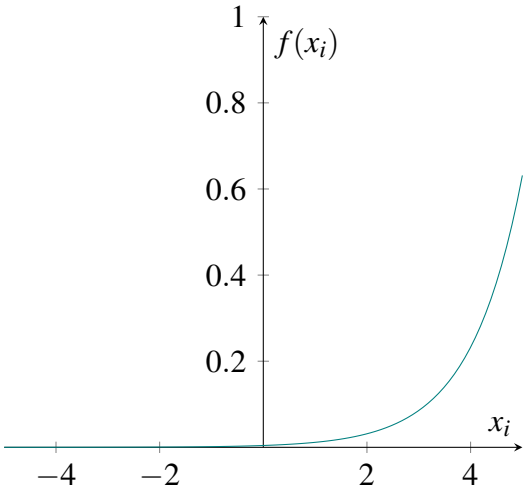
$$f(x) = \frac{1}{1+e^{-x}}$$



Softmax

Softmax is another notable activation function. It is a way of forcing the neural networks to output the sum of 1. Thereby, it transforms the output of a neural network into a probability distribution over multiple classes, facilitating the model's interpretation and enabling it to make confident class predictions. This way, it is primarily employed in the output layer of classification models.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$



2.4.3 Backpropagation

"The backpropagation algorithm is possibly the most important algorithm in deep learning" - A Brief History of Deep Learning (Li and Du (2023))

We learned that the weight of a connection affects how a neuron processes the information it receives along the connection. In fact, training an artificial neural network essentially involves searching for the best set of weights, and this is what is referred to when a model is learning.

That is why, as mentioned, the backpropagation algorithm allowed for efficient training of deep neural networks by adjusting the weights of connections between neurons.

During the weight update process, the neural network compares its predictions to the actual values, calculating the error or the difference between them. This error is then used to adjust the weights in such a way that the error decreases over time. The error E can be defined using a loss function $L(y, \hat{y})$, where y represents the actual values and $\hat{y}$ represents the predicted values.

$$E = L(y, \hat{y})$$

Gradient Descent and Learning Rate

The learning rate η plays a crucial role in this process, determining how big of a step the weights take in the direction of reducing the error. A higher learning rate can cause the weights to update too quickly, potentially overshooting the optimal solution, while a lower learning rate may slow down the learning process. Finding the right balance is essential for efficient training of neural networks.

$$w_{new} = w_{old} - \eta \frac{\partial E}{\partial w}$$

- w represents the weights and $\frac{\partial E}{\partial w}$ is the gradient of the error with respect to the weights.

Loss Function and Error Propagation

Additionally, the error values, typically computed using a loss function such as mean squared error (MSE) or cross-entropy, guide the adjustment of weights. The gradient of the error concerning each weight indicates the direction and magnitude of the adjustment needed to minimize the error. Through techniques like backpropagation, the network propagates these errors backward, updating the weights layer by layer.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

- o_j is the output of neuron j, and net_j is the weighted sum of inputs to neuron j.

Convergence and Optimization

This iterative process continues until the model converges to a satisfactory solution, balancing between speed and accuracy in learning. Convergence occurs when changes in the weights become sufficiently small, indicating that the network has found a local minimum in the error surface. Various optimization techniques such as momentum, RMSprop, and Adam can be employed to enhance the convergence process.

2.5 Training Neural Networks

Training neural networks is a fundamental process in the development of models capable of performing complex tasks in computer vision. These models rely on training data to learn and improve their accuracy over time. Once these learning algorithms are fine-tuned for accuracy, they become powerful tools that allow for the classification and clustering of data at high velocity, leveraging both labeled and unlabeled data.

2.5.1 Training Dataset

The dataset is divided into three subsets: training, validation, and test sets. A typical split might be 70% training, 15% validation, and 15% test. In the next section we will learn more about validation and test, for now, the training dataset is a critical component in the development of neural networks. It comprises the data used to train the model, allowing it to learn the underlying patterns and relationships within the data. The quality and quantity of the training data significantly influence the model's performance.

Characteristics of a good training dataset include:

- **Diversity**: A diverse training dataset contains a wide variety of examples, ensuring that the model can learn to generalize across different scenarios.
- **Volume**: A larger volume of data provides more opportunities for the model to learn, leading to better generalization and robustness.
- **Relevance**: The data should be relevant to the task at hand, containing features and labels that accurately represent the problem domain.
- **Quality**: High-quality data with minimal noise and errors is essential for effective training, as poor-quality data can lead to misleading patterns and a suboptimal performance.

2.5.2 Pre-processing

Pre-processing is a crucial step in preparing the training dataset for a neural network. Effective pre-processing ensures that the data is in a suitable format for training, enhancing the model's performance and generalization capabilities.

Normalization

Normalization is the process of scaling the pixel values in images to a specific range. This step ensures that the neural network can learn more efficiently and helps in speeding up the

convergence of the training process.

Normalization can be achieved using various techniques:

- **Min-Max Scaling**: This technique scales the pixel values to a specific range, usually [0, 1]. It is achieved by subtracting the minimum value of the feature and then dividing by the range (maximum value - minimum value). This technique ensures that all pixel values are within the specified range.
- **Z-Score Normalization**: Also known as standardization, this technique transforms the data to have a mean of zero and a standard deviation of one. It is achieved by subtracting the mean of the pixel values and then dividing by the standard deviation. This method is particularly useful when the data follows a Gaussian distribution.
- **BatchNormalization**: This technique normalizes the input to each layer within a neural network, not just the initial input. It standardizes the inputs to each layer to have zero mean and unit variance. Batch normalization helps in stabilizing the learning process, improve the convergence, this way it allows for higher learning rates, and reduces the dependency on the initialization.

Regularization

Regularization techniques are used to enhance the generalization ability of neural networks, preventing them from overfitting to the training data. Overfitting occurs when a model performs well on training data but poorly on unseen data. Regularization methods introduce additional constraints or penalties on the model parameters.

Common regularization techniques include:

- **Data Augmentation**: Is a crucial technique used to increase the diversity of the training dataset without actually collecting new data, helping the model to generalize better. It involves applying various transformations to the existing data, thereby generating new samples. This technique is particularly valuable in training neural networks as it helps improve the model's generalization ability and robustness.

  Common data augmentation techniques include:

  - **Rotation**: Rotating images by a certain angle to create different perspectives.
  - **Flipping**: Flipping images horizontally or vertically to add variation.
  - **Scaling**: Changing the size of the images while maintaining the aspect ratio.
  - **Translation**: Shifting images along the x or y axis.

- **Brightness and Contrast Adjustment**: Modifying the brightness and contrast of images to simulate different lighting conditions.
- **Noise Addition**: Adding noise to images to make the model used to variations.
- **Cropping and Padding**: Randomly cropping or padding images to different sizes.
- **Color Jittering**: Changing the colors of images to introduce variation.

These techniques help create a richer and more varied training dataset, enabling the neural network to learn more robust features. By augmenting the data, we improve the model's performance on unseen data. It is especially beneficial in scenarios where collecting additional data is impractical or costly. By effectively applying these techniques, we can enhance the training process and develop more reliable and accurate models.

- **Dropout**: During training, dropout involves randomly disabling a fraction of neurons in the network, which helps prevent the network from becoming overly reliant on any single pathway and encourages the learning of redundant representations.

  Figure 2.7 illustrates the concept of dropout. On the left side of the figure, we see a neural network without dropout, where all neurons and connections are active during training. On the right side, the network with dropout is shown, where several neurons are randomly disabled during each training iteration with a dropout rate of $p = 0.5$. This means that each neuron has a 50% chance of being disabled during training. This randomness helps to prevent any one neuron from becoming too important, forcing the network to develop redundant paths for the same output. As a result, dropout helps in reducing overfitting by making the model more robust and less sensitive to the specific neurons and their weights, leading to better generalization on unseen data.

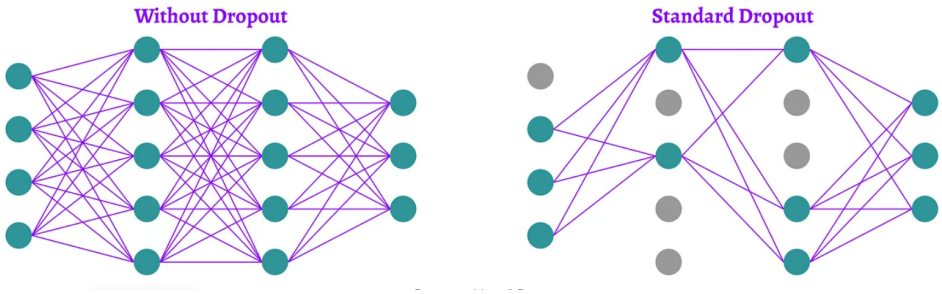


Figure 2.7: Comparison of neural networks without dropout and with dropout.

- **Early Stopping**: To prevent overfitting, early stopping can be used. This involves halting the training process when the validation loss stops improving for a specified number of epochs, known as the patience parameter. For example, if the patience parameter is set to 10, training will stop if the validation loss does not improve for 10 consecutive epochs. This ensures the model does not overfit by training too long on the training data.

- **Other callback methods**: Other methods like learning rate schedules, which adjust the learning rate during training to improve convergence, and model checkpointing, which involves saving the model at various points during training, typically when the model shows improvement on the validation set. This allows you to restore the best-performing model rather than the last trained model. For instance, if training is set to run for 100 epochs but the model performs best at epoch 75, checkpointing ensures that the model from epoch 75 can be restored and used.

By employing these regularization techniques, neural networks can be made more robust and capable of generalizing well to new, unseen data.

2.5.3 Epochs

In the context of training neural networks, an epoch refers to one complete pass through the entire training dataset. During each epoch, the model processes every sample in the training set once, allowing it to learn from the data and update its parameters accordingly.

Understanding and choosing the appropriate number of epochs is crucial for effective model training. Here are some key points to consider:

- **Training Duration**: The number of epochs determines how long the model will be trained. A higher number of epochs means the model will have more opportunities to learn from the data, but it also increases the risk of overfitting if the model starts to learn noise and irrelevant patterns in the training data.
- **Learning Dynamics**: Monitoring the training and validation loss over epochs helps in understanding the learning dynamics. If the training loss decreases consistently and the validation loss also decreases or stabilizes, it indicates that the model is learning effectively. If the validation loss starts to increase while the training loss continues to decrease, it suggests overfitting.
- **Model Evaluation**: Evaluating the model's performance at different epochs can help identify the optimal number of epochs. This ensures that the model is trained sufficiently to generalize well to unseen data without overfitting. We will learn more about evaluation in the next section.

Choosing the right number of epochs is a balance between training the model enough to capture the underlying patterns in the data and stopping before it starts to overfit. Monitoring the learning curves and using strategies like early stopping can determine the optimal training duration and enhance the model's performance.

2.5.4 Cost Functions

Cost functions, also known as loss functions, play a crucial role in training neural networks. They measure the difference between the predicted outputs of the network and the actual target values. By minimizing this difference, the network can improve its performance. Common cost functions include Mean Squared Error (MSE) for regression tasks and Cross-Entropy Loss for classification tasks. The choice of cost function depends on the type of problem being solved.

For regression tasks, where the goal is to predict a continuous value, Mean Squared Error (MSE) is commonly used. MSE calculates the average of the squares of the errors between predicted and actual values, penalizing larger errors more severely.

Mathematically, it is expressed as:

$$\text{Mean Squared Error} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- y_i represents the actual value and $\hat{y}_i$ represents the predicted value for the i-th sample.

For classification tasks, where the goal is to predict discrete class labels, Cross-Entropy Loss is often used. Cross-Entropy Loss measures the dissimilarity between the predicted probability distribution and the actual distribution. It is particularly effective for multi-class classification problems.

The formula for Cross-Entropy Loss for a single instance is:

$$\text{Cross-Entropy Loss} = -\sum_{c=1}^{C} y_{o,c}\log(p_{o,c})$$

- $y_{o,c}$ is a binary indicator (0 or 1) if class label c is the correct classification for observation o, and $p_{o,c}$ is the predicted probability that observation o belongs to class c.

Another important aspect of cost functions is their impact on the training process. It guides the optimization algorithm, typically gradient descent, in adjusting the model parameters to minimize the loss. A well-chosen cost function ensures that the optimization process is smooth and converges to a solution that generalizes well to new data. In some cases, custom cost functions can be defined to address specific needs of a given problem. For example, if a problem involves imbalanced classes, a weighted cross-entropy loss can be used to give different importance to different classes, helping the model to focus more on underrepresented classes.

Overall, the cost function is a vital component in the neural network training pipeline. It quantitatively defines the objective that the training process aims to achieve and directly influences the performance and effectiveness of the resulting model.

2.6 Evaluating Neural Networks

Evaluating the performance of neural networks is a critical aspect of developing robust and effective machine learning models. This section focuses on the various methods and tools used to assess how well a neural network performs on given tasks. By understanding and applying these evaluation techniques, we can ensure that our models are not only accurate but also generalize well to unseen data.

2.6.1 Validation and Test Datasets

The validation and test datasets play crucial roles in the evaluation of neural networks. They are used to measure how well the model generalizes to new, unseen data, and to identify potential issues such as overfitting or underfitting, these will be detailed ahead when understanding how to interpret the learning curves.

- **Validation Dataset**: The validation dataset is used during the training process to periodically evaluate the model's performance. By monitoring the validation loss and other metrics, we can tune hyperparameters and make adjustments to the model to improve its generalization ability. This helps in detecting overfitting early and allows for the implementation of techniques like early stopping.
- **Test Dataset**: The test dataset is used to evaluate the final performance of the model after the training process is complete. It provides an unbiased assessment of how well the model is likely to perform on real-world data. The results on the test dataset are crucial for understanding the model's true predictive power and for comparing different models.

2.6.2 Metrics

Metrics are quantitative measures used to assess the performance of a model or system. They provide insights based on different types of metrics such as accuracy, loss, precision, recall, F1-score, and more, depending on the specific task and goals.

Regarding model evaluation, it is essential to understand metric types and definitions. They serve as objective criteria for assessing the quality and reliability of predictions or classifications made by machine learning models. Let us see them in more detail.

Accuracy

Accuracy is a fundamental metric in evaluating the performance of classification models. It represents the ratio of correctly predicted instances to the total number of instances in the dataset. In essence, it indicates the model's ability to make correct predictions across all classes.

Mathematically, this is defined as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

For instance, if a model correctly identifies 90 out of 100 instances, its accuracy would be 90%. While accuracy provides a straightforward measure of overall model performance, it may not always be sufficient, especially in cases of imbalanced datasets where one class dominates the others. In such scenarios, accuracy alone might be misleading, and other metrics become crucial for a comprehensive evaluation of the model's effectiveness. Therefore, while it serves as a primary indicator of model success, it is essential to interpret it alongside other performance metrics to gain a comprehensive understanding of the model's behavior and capabilities.

Improving accuracy is a multifaceted process that involves various strategies and techniques aimed at enhancing the model's ability to make correct predictions. Ultimately, it requires a combination of careful preprocessing, model selection, hyperparameter tuning, and iterative refinement, guided by a thorough understanding of the dataset and problem domain.

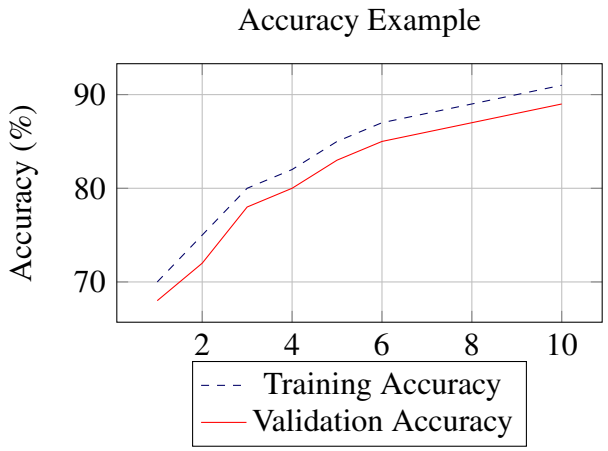


Figure 2.8: Accuracy curves showing training and validation accuracy improving over epochs.

Loss

Loss is a crucial measure indicating how well the model performs the task for which it was trained. Essentially, it quantifies how much the model's predictions deviate from the true labels or expected outcomes. During training, loss is calculated based on the model's predictions compared to the true labels.

Mathematically, for a set of predictions $\hat{y}$ and true labels y, the loss L can be expressed as:

$$L = \frac{1}{N}\sum_{i=1}^{N} \ell(y_i, \hat{y}_i)$$

- ℓ is the loss function and N is the number of instances.

A low loss indicates that the model's predictions are close to the true labels, while a high loss indicates a significant discrepancy between predictions and labels.

The definition of an acceptable loss varies depending on the specific problem being addressed. In some cases, a low loss may be critical, such as in medical problems, while in others, a higher loss may be tolerable. For instance, in medical diagnostics, a model with high accuracy and low loss is essential because the consequences of incorrect predictions can be severe, potentially leading to incorrect treatments or missed diagnoses. Conversely, in other fields such as movie recommendations, a slightly higher loss may be more acceptable because the stakes are lower, and users are more forgiving of occasional mismatches in suggestions.

It's important to interpret loss alongside other evaluation metrics, as a low loss doesn't automatically guarantee good performance across all metrics. For example, in a classification problem with imbalanced classes, a model could achieve a low loss by accurately predicting the majority class while failing to predict the minority class correctly. Therefore, metrics like precision, recall, and F1-score are also essential to provide a complete picture of model performance and to ensure that the model performs well across all relevant aspects.

Finally, to reduce loss during training, various strategies can be employed. This includes choosing an appropriate model architecture for the problem at hand, as selecting a convolutional neural network (CNN) for image-related tasks. Adjusting hyperparameters, like the learning rate and the number of layers, can also significantly impact model performance. Data augmentation techniques, such as rotating or flipping images in computer vision problems, help the model generalize better by providing a more diverse set of training examples. updateRegularization techniques, like dropout, prevent overfitting by randomly omitting neurons during training, thus forcing the model to learn more robust features. Additionally, the choice of activation function, such as ReLU or sigmoid, can impact the model's ability to learn complex patterns during training.

These strategies collectively contribute to improving the model's ability to minimize loss while maintaining or improving performance across various evaluation metrics, ultimately leading to more reliable and effective machine learning solutions.

Precision, Recall, and F1-Score

In addition to accuracy and loss, precision, recall, and F1-score are vital metrics, particularly in the context of classification tasks. Using concepts defined as follows:

True Positive (TP): The number of correctly predicted positive instances.

True Negative (TN): The number of correctly predicted negative instances.

False Positive (FP): The number of incorrectly predicted positive instances.

False Negative (FN): The number of incorrectly predicted negative instances.

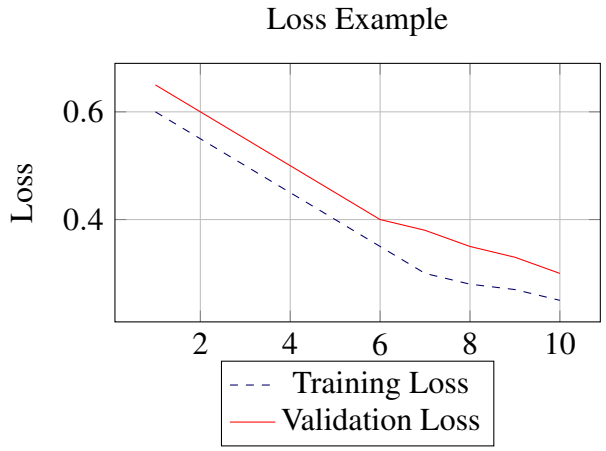


Figure 2.9: Loss curves showing training and validation loss decreasing over epochs.

- **Precision:** It is the ratio of true positive predictions to the total predicted positives.

  It is given by:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall:** It is the ratio of true positive predictions to the total actual positives.

  It is defined as:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

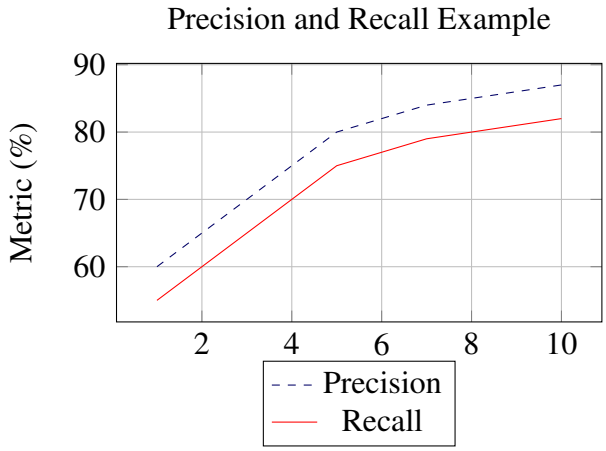


Figure 2.10: Precision and recall curves improving over epochs.

- **F1-Score:** It is the harmonic mean of precision and recall, providing a single metric that balances both. It is calculated as:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

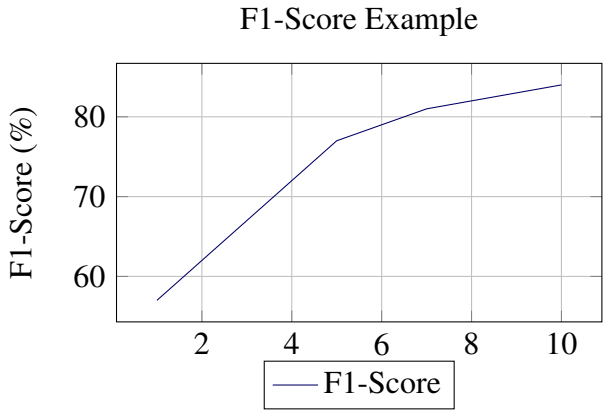


Figure 2.11: F1-Score curve showing improvement over epochs.

These metrics are particularly useful in scenarios with imbalanced classes, where accuracy might not fully reflect model performance. By understanding and utilizing these various metrics, one can thoroughly evaluate the effectiveness and reliability of neural networks in different contexts and ensure that the chosen models are well-suited to the tasks at hand.

2.6.3 Confusion Matrix

Considering classification problems, a Confusion Matrix is an essential tool in evaluating a model's performance. It provides a detailed breakdown of the model's predictions compared to the actual outcomes, allowing for a more nuanced understanding of its performance beyond simple accuracy metrics. The confusion matrix is beneficial for identifying how well a model distinguishes between different classes and where it might be making errors.

The matrix is structured as a square grid, representing the actual classes on one axis and the predicted classes on the other. Each cell in the matrix represents the number of instances where the actual class matches or does not match the predicted class (see Table 2.2).

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 2.2: Confusion Matrix

These elements provide the foundation for several important metrics mentioned. By providing a comprehensive view of model performance, the confusion matrix helps identify specific areas where the model excels or needs improvement. For instance, in cases of imbalanced datasets, where one class significantly outnumbers the others, accuracy alone may be misleading. The confusion matrix reveals how well the model performs across all classes, ensuring that minority classes are appropriately accounted for.

2.6.4 Learning Curves

When getting towards the step of the evaluation model it's also essential to understand learning curves, a plot of a model learning performance over time and a widely used diagnostic tool in machine learning for algorithms that incrementally learn from a training dataset.

During the training of a model, the current state of the model at each step of the training algorithm can be evaluated by both the train learning curve and the validation learning curve. Respectively, a learning curve calculated from the training dataset gives an idea of how well the model is learning, and a learning curve calculated from a hold-out validation dataset gives an idea of how well the model is generalizing.

In some cases, it is also common to create learning curves for multiple metrics, such as in classification predictive modeling problems, where the model may be optimized according to cross-entropy loss and model performance is evaluated using classification accuracy. In this case, two plots are created: one for the learning curves of each metric, and each plot can show two learning curves, one for each of the training and validation datasets.

Overall, the shape and dynamics of a learning curve can be used to diagnose the behavior of a machine learning model and, in turn, suggest the type of configuration changes that may be made to improve learning and/or performance.

Under-fitting

Under-fitting occurs when a model cannot learn the training dataset adequately, resulting in poor performance on both the training and validation datasets. It can be identified from the learning curve of the training loss. If the training loss continues to decrease until the end of training, it suggests that the model is still capable of learning and improving, indicating that the training process may have been stopped too early. This scenario implies that additional training epochs could enhance the model's performance.

Another indication of under-fitting is a learning curve where the training loss remains flat or shows high noise levels without significant reduction. This pattern indicates that the model struggles to capture the underlying patterns in the training data. Such flat or noisy training loss curves suggest that the model architecture might be too simple to represent the complexity of the data or that the learning algorithm is not well-suited to the task.

Figure 2.12 illustrates the learning curves for a model experiencing under-fitting. The dashed blue line represents the training loss, and the solid red line represents the validation loss. Both curves show high loss values and decrease slowly over epochs, indicating that the model is not learning the training data effectively. The training loss decreases gradually, suggesting that the model is still capable of learning but requires more epochs. However, the high initial loss and slow reduction in both training and validation losses highlight the need for a more complex model architecture or better-suited learning algorithm to capture the data's underlying patterns.

To address under-fitting, various strategies can be employed:

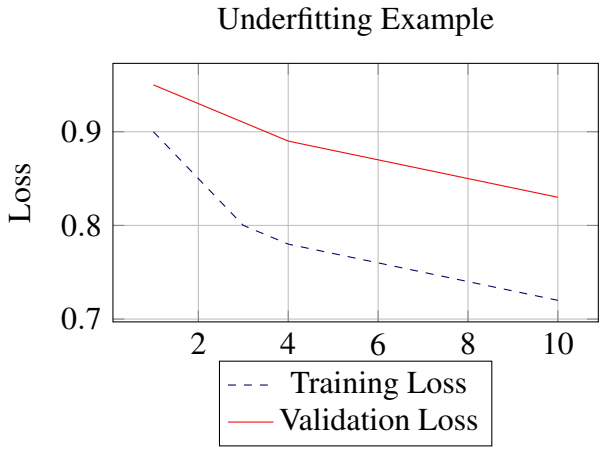


Figure 2.12: Learning curves indicating underfitting, where both training and validation loss are high and decreasing slowly.

- **Increasing Model Complexity**: Using a more complex model with additional layers or neurons can help capture more intricate patterns in the data.
- **Extending Training Duration**: Allowing the model to train for more epochs can provide additional opportunities for learning and improvement.
- **Improving Feature Engineering**: Enhancing the quality and quantity of features used for training can provide the model with more relevant information.
- **Reducing Regularization**: Decreasing regularization parameters such as dropout rates or weight penalties can allow the model to fit the training data more closely.

Over-fitting

Over-fitting refers to a model that has learned the training dataset too well, including the statistical noise or random fluctuations in the training dataset. The problem with overfitting is that the more specialized the model becomes to the training data, the less well it can generalize to new data, increasing generalization error. This increase in generalization error can be measured by the performance of the model on the validation dataset.

In the figure 2.13, we can see the training loss continuing to decrease with experience, or the validation loss decreasing to a point and then beginning to increase again. This inflection point in validation loss may indicate the point at which training should be halted, as further training beyond this point demonstrates the dynamics of overfitting.

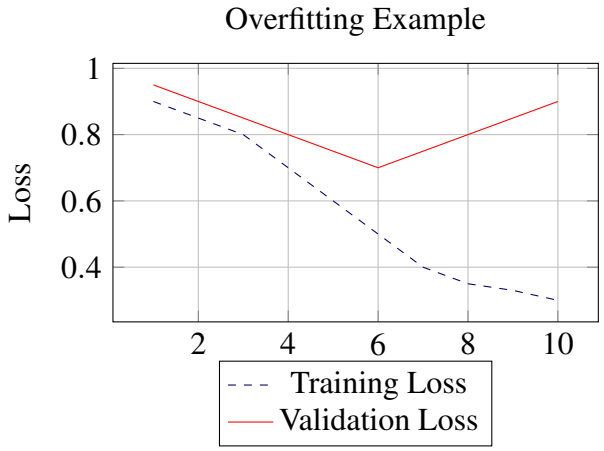


Figure 2.13: Learning curves indicating overfitting, where the training loss decreases continuously but the validation loss decreases to a point and then increases.

To address overfitting, various strategies can be employed:

- **Adding More Training Data**: Increasing the size of the training dataset can help the model generalize better by providing more examples to learn from.
- **Data Augmentation**: Techniques such as rotating, flipping, and scaling images can artificially increase the size of the training dataset and help the model generalize better.
- **Simplifying the Model**: Reducing the complexity of the model by decreasing the number of layers or neurons can prevent it from learning the noise in the training data.
- **Regularization Techniques**: Methods such as L1 or L2 regularization add a penalty to the loss function to discourage the model from fitting the noise in the training data.
- **Early Stopping**: Monitoring the model's performance on the validation set and stopping training when the performance starts to degrade can prevent overfitting.
- **Dropout**: Randomly omitting neurons during the training process can prevent the model from becoming too specialized to the training data.

Good fit

A good fit is identified by training and validation losses that decrease to a point of stability with a minimal gap between the two final loss values. In this case, the loss of the model will almost always be lower on the training dataset than on the validation dataset. This means that we should expect some gap between the training and validation loss learning curves, known as the "generalization gap." The plot of validation loss should decrease to a point of stability and have a small gap with the training loss, as illustrated in Figure 2.14.

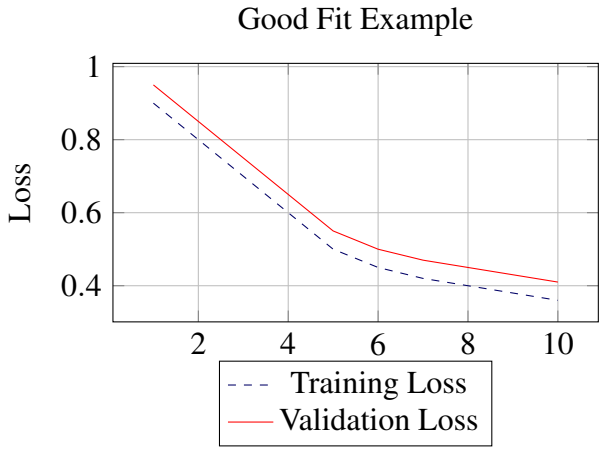


Figure 2.14: Learning curves indicating a good fit, where both training and validation loss decrease and converge to stability with a minimal gap.

Characteristics of a good fit include:

- **Stable Loss Curves**: Both training and validation loss curves decrease and stabilize, indicating that the model has learned the underlying patterns in the data without overfitting.
- **Minimal Generalization Gap**: There is a small gap between the final values of the training and validation losses, suggesting the model generalizes well to new, unseen data.
- **Consistent Performance**: The model shows consistent performance across training and validation sets, which implies that it is neither overfitting nor under-fitting.
- **Convergence**: The model's performance converges to a stable point, where additional training does not significantly change the loss values, indicating that the optimal learning state has been reached.

Diagnosing Unrepresentative Datasets

An unrepresentative dataset is one that may not capture the statistical characteristics relative to another dataset drawn from the same domain, such as between a training and a validation dataset. This issue can commonly occur if the number of samples in a dataset is too small relative to another dataset.

Figure 2.15 illustrates this scenario. The dashed blue line represents the training loss, and the solid red line represents the validation loss. The significant gap between the two curves suggests that the validation set does not adequately represent the training set. This discrepancy indicates that the model's performance on the validation set is not a reliable indicator of its generalization capability.

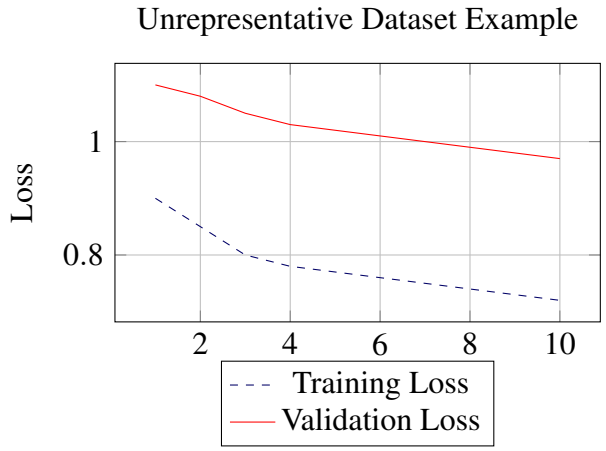


Figure 2.15: Learning curves indicating an unrepresentative dataset, where there is a significant gap between training and validation losses, suggesting that the validation set does not adequately represent the training set.

Characteristics of unrepresentative datasets include:

- **Small Sample Size**: A dataset with too few samples may fail to capture the diversity and variability of the domain, leading to biased training and validation results.
- **Non-random Sampling**: If the dataset samples are not randomly selected, they may not represent the overall population accurately, resulting in skewed model performance.
- **Domain Mismatch**: Differences in the statistical properties between training and validation datasets can indicate that one or both datasets are not representative of the domain.
- **High Variance in Performance**: Significant fluctuations in model performance across different datasets suggest that the datasets might not be representative of the true data distribution.

2.7 Convolutional Neural Networks

Now that we have a good knowledge of neural networks, from the functionality of a neuron to functions and techniques explored in the subject, and even how to analyze its results and quality. We can explore the specific type of network implemented in the model of this research, Convolutional Neural Networks - CNN's. They were first developed in the late 1980s to early 1990s, by Yann LeCun and others, and revolutionized deep learning by introducing shared weights and hierarchical feature learning, making them highly effective with image, speech, or audio signal inputs. Before, manual, time-consuming feature extraction methods were used to identify objects in images.

CNN's now provide a more scalable approach to image classification and object recognition tasks, leveraging principles from linear algebra, specifically matrix multiplication, to identify patterns within an image. Since an image contains pixel data that can be represented in a numerical form, this representation is what is passed as input. Also, We can note that in the realm of images, there are elementary features like corners and contours, which are further defined by the presence of edges, these are the patterns identified along the process.

Therefore, given the pixels that represent an image, the first layer can easily identify edges, by comparing the brightness of neighboring pixels. Given the first hidden layer's description of the edges, the second hidden layer can easily search for corners and extended contours, which are recognizable as collections of edges. Given the second hidden layer's description of the image in terms of corners and contours, the third hidden layer can then detect entire parts of specific objects, by finding specific collections of contours and corners.

2.7.1 Convolutional Neural Networks Layers

The CNNs have three main layers: Convolutional layer, Pooling layer, and Fully-connected layer. With each layer, the network increases in complexity, identifying greater portions of the image. Let us see these details.

Convolutional Layer

The convolutional layer is the fundamental building block of a CNN, and it is where most computation occurs. It earns its name from the mathematical operation it performs, an element-wise multiplication of a filter with the matrix representation of the input image, resulting in what is known as a feature map. The objective is to reduce the image size being passed to the CNN while maintaining the important features.

This filter, also referred to as a kernel, typically takes the form of a 3 by 3 matrix, although other sizes and types of matrices can also be utilized to capture different aspects of the input data. Also, it slides step by step through each of the elements in the input image. These steps are known as strides and can be defined when creating the CNN.

The mathematical operation is as follows:

$$O(x,y) = \sum\sum I(a,b) * K(x-a, y-b)$$

- I represent the image.
- K is the kernel.
- O is the output result, known as feature map.

- $\sum\sum$ makes sure that the operation is performed over all possible positions of the kernel within the image.
- a and b are the dimensions of the kernel.
- $(x-a, y-b)$ is the relative position of the kernel concerning the output pixel (x, y).

An example is shown in Figure 2.16. Let us dissect it, Matrix 1 Figure 2.16 (a) is a 12 x 12 array representing a square, created to be our input. This input is multiplied with a filter, Matrix 2 Figure 2.16 (b), called the Laplacian Filter, known for detecting edges and corners. This results in a feature map seen in Matrix 3 Figure 2.16 (c). We see that the the square representation content was kept, while not changing much, with other layers and more iterations the image continuous to be successfully filtered.

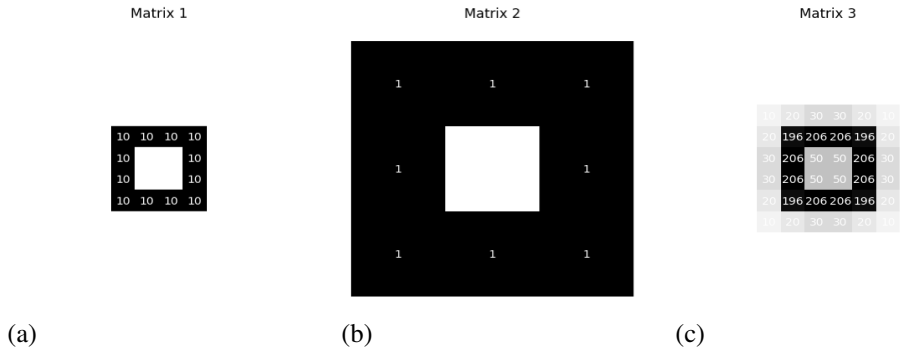


Figure 2.16: Convolution with the Laplacian filter

Once the feature map is obtained, the Rectified Linear unit (ReLU) is applied to prevent the operation from being linear. This is important because the relationships and patterns present in images are inherently non-linear, and the ReLU activation function helps capture these complex relationships, allowing the CNN to learn more expressive and distinctive features from the data.

Pooling Layer

In this layer happens another operation, called pooling that results in what is known as a pooled feature map. This process ensures that the neural network can detect features in an image regardless of their precise location, a property known as spatial invariance. Moreover, while downsampling the feature maps and reducing their spatial dimensions, this operation retains the most pertinent information for subsequent layers in the neural network.

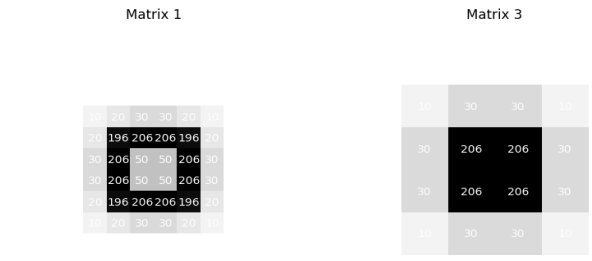
There are several types of pooling, for example, max-pooling, average pooling, and min pooling. In max-pooling, a 2 by 2 matrix is slid over the feature map while picking the largest value in a given box. Similarly, average pooling computes the average value within each box, and min-pooling operates by selecting the smallest value within each box.

The mathematical formula of max pooling is:

$$\text{MaxPooling}(X)_{i,j,k} = \max_{m,n} X_{i+s_m, j+s_n, k}$$

- X is the input matrix.
- (i, j) are the indices of the output matrix.
- k is the channel index.
- s_m and s_n are the stride values in the horizontal and vertical directions, respectively.
- The pooling window is defined by the filter size f_x and f_y, centered at the output index (i, j).

An example is shown in Figure 2.17. It receives the output obtained in Figure 2.16 (c) at the end of the convolution operation, here as Matrix 1 Figure 2.17 (a). This input is delimited to be multiplied by a 2x2 matrix. Resulting in the seen polled feature map in Matrix 3 Figure 2.17 (b). Notice how this pooled feature map represents a square, just like the input before the convolutional operation, Figure 2.16 (a). Illustrating how this layer reduces the image size being passed to the CNN while maintaining the important features.



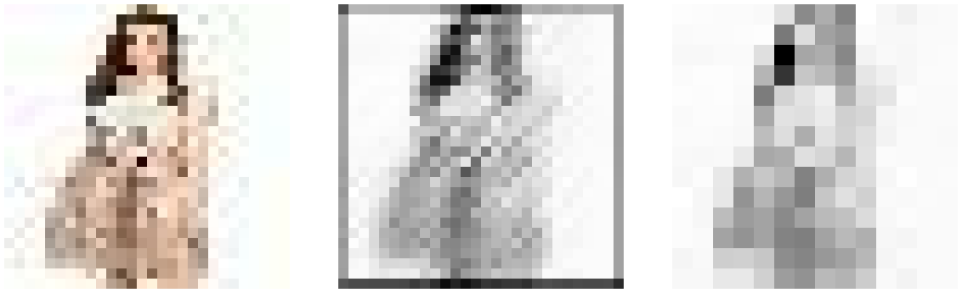
(a) (b)

Figure 2.17: Convolution with the Laplacian filter

In Figure 2.18 we have an example of a convolution and pooling result with a real input from our used data, an image of a German dirndl. The Figure 2.18 (a) is the input, the Figure 2.18 (b) is the convolution result and the Figure 2.18 (c) is the pooling result. These images look like this because the input image has been resized to 28 x 28 pixels. This resizes and all the details of the data will be further explained. For now, observe only how the convolutional and pooling processes work filtering edges and contours.

Fully Connected Layer

Right before entering the fully connected layer, the pooled feature map is flattened, this involves transforming the entire matrix into a single column. This is then passed to the input layer of the neural network and right after passed to a fully connected layer. By the end, the result of the entire process is emitted by the output layer.



(a) (b) (c)

Figure 2.18: Examples of the first layers of convolutional and max pooling

When performing a classification task based on the features extracted through the previous layers and their different filters. While the convolutional and pooling layers tend to use ReLU functions to introduce non-linearity and capture complex patterns in the data.

Fully connected (FC) layers usually leverage a sigmoid activation function to produce a single probability value between 0 and 1, indicating the likelihood of the input belonging to the positive class, when binary. And a softmax activation function for multi-class classification tasks, that outputs probabilities for each class, ensuring that the sum of probabilities across all classes equals 1. Allowing the model to assign a likelihood score to each class, aiding in decision-making.

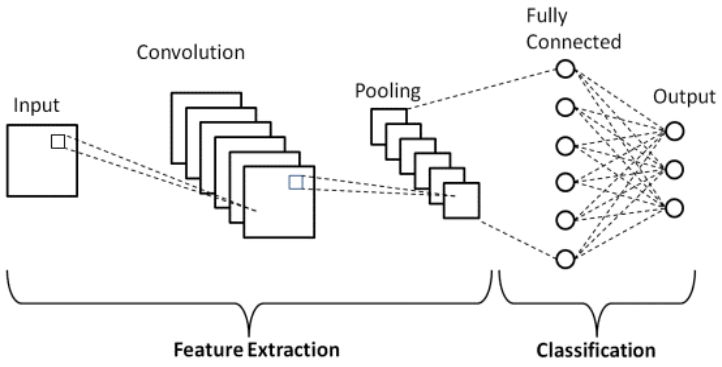


Figure 2.19: Convolutional Neural Networks Layers

We could say that this fully connected layer is close to what was learned above as a neural network, and the differential of a Convolutional Neural Network is its two first steps, which identify patterns applying mathematical calculations, in the process known as Feature Extraction and the fully connected layer does the classification job, as seen in Figure 2.19.

3

A CNN model to Classify Traditional Garments

This chapter details the steps undertaken in our study, starting with a description of the tools and libraries used in the project. Following this, we delve into the Knowledge Discovery in Databases (KDD) process, covering dataset collection and pre-processing steps comprehensively. The chapter then provides an overview of the dataset, including the dataset path, the process of splitting the data into training, validation, and test sets, and the techniques used for redefinition and normalization. Data augmentation methods employed to enhance the dataset's variability are also discussed.

Next, the creation of the Convolutional Neural Network (CNN) model is described in detail. This includes an overview of the model's architecture and layers, followed by the compilation and training process. The chapter culminates with the evaluation of the model, presenting the achieved results, analyzing the performance, and visualizing the model's accuracy and efficiency in classifying garments. Through these steps, we aim to build a robust and effective model capable of accurately classifying traditional garments from various cultures.

3.1 Tools and Libraries

In the implementation process, several libraries are utilized to accomplish various tasks efficiently. Firstly, the `os` library provides functionalities for interacting with the operating system, enabling tasks such as file manipulation and directory navigation. Next, the `shutil` module, specifically the `copyfile` function, is employed for copying files, allowing seamless file management operations. And, the `matplotlib.pyplot` module is utilized for generating visualizations and plots, aiding in the analysis and presentation of data.

Additionally, the `numpy` library, often abbreviated as `np`, is a fundamental package for scientific computing in Python, offering support for array operations and numerical computations.

Also, the `cv2` library, known as `OpenCV`, is a powerful computer vision library used for image processing, computer vision tasks, and machine learning applications. It offers a wide range of functionalities for image manipulation, feature detection, and object tracking, making it indispensable for image-related tasks.

Moving on, the project heavily relies on two powerful deep-learning libraries: `TensorFlow` and `Keras`. `TensorFlow` is an open-source machine learning framework developed by Google, widely known for its flexibility, scalability, and comprehensive support for building various machine learning models, including neural networks. `Keras`, on the other hand, is a high-level neural networks API that runs on top of TensorFlow. And provides a user-friendly interface for constructing neural networks with minimal boilerplate code, making it ideal for rapid prototyping and experimentation.

The `tensorflow.keras.preprocessing.image` module provides tools for loading, and preprocessing, so functions such as `'load_img'`, `'img_to_array'`, and `'array_to_img'`, facilitate loading images from disk, converting images to arrays, and converting arrays back to images, respectively. And although also providing augmenting image data it was not chosen for this specific task, for so the `imgaug` module, offers a wide range of augmentation techniques and allows for more fine-grained control over its parameters. It provides numerous methods, including geometric transformations, color manipulations, and noise addition. Users can compose multiple augmentation techniques into custom augmentation pipelines, providing flexibility in designing complex augmentation strategies.

Moreover, the `tensorflow.keras.layers` module combines the strengths of both `TensorFlow` and `Keras`, offering a wide range of essential layers specifically designed for constructing CNNs. These layers serve as the building blocks for defining its architecture, enabling researchers and developers to create complex models with ease. Some of the key layers included in this module are Conv2D, which performs convolutional operations to extract features from input data, MaxPooling2D, used for spatial downsampling to reduce the spatial dimensions of feature maps, BatchNormalization, which normalizes activations within each mini-batch to improve convergence and generalization, Flatten, responsible for converting multidimensional feature maps into a flattened vector format suitable for input to fully connected layers, Dense, for implementing fully connected layers that perform classification or regression tasks, and Dropout, a regularization technique that randomly drops a fraction of input units during training to prevent overfitting and improve model generalization.

Additionally, the project harnesses the power of the `tensorflow.keras.optimizers` module, which offers a variety of optimization algorithms to fine-tune model parameters during the training process. One such optimizer is Adam, a popular choice for its adaptive learning rate and efficient parameter updates, which help accelerate convergence and improve model performance. By intelligently adjusting the learning rate based on the gradient magnitudes of individual parameters, Adam ensures swift convergence while mitigating the risk of getting stuck in local minima.

Furthermore, the project leverages the capabilities of `tensorflow.keras.callbacks` to enhance the training process and monitor model performance in real time. Among the plethora of available callback functions, the project utilizes ModelCheckpoint to periodically save the best-performing model weights during training, enabling seamless recovery in case of interruptions and facilitating model evaluation on the validation set. Additionally, EarlyStopping is employed to prevent overfitting by monitoring the validation loss and halting training when no significant improvement is observed over a specified number of epochs. This proactive approach helps prevent the model from memorizing the training data and encourages generalization to unseen examples. Furthermore, the project incorporates custom callbacks tailored to specific requirements, such as learning rate scheduling, gradient clipping, or model visualization, allowing for fine-grained control over the training dynamics and optimization process.

Furthermore, the project leverages the `sklearn.metrics` module, specifically the `confusion_matrix` function, a vital tool for evaluating the performance of classification models by comparing predicted labels against true labels. This enables a deeper understanding of model performance across different classes and aids in identifying areas for improvement. Finally, the `seaborn` library, as `sns`, creates visually appealing and informative statistical graphics.

Overall, the combination of these libraries provides a comprehensive toolkit for efficient development and execution. As ensures robust evaluation and visualization of the classification model performance, contributing to informed decision-making and model refinement.

3.2 Knowledge Discovery in Databases Process

In the process of image classification using neural networks, several key stages are followed to ensure the effectiveness and accuracy of the model. These stages can be mapped to the steps of the Knowledge Discovery in Databases (KDD) process. First, during the Data Selection phase, relevant images are collected for the classification task. Next, the Data Preprocessing stage involves performing operations such as normalization, resizing, and data augmentation on the images to prepare them for analysis. Following this, the Data Transformation step converts the images into a format suitable for input into a neural network, such as tensors. The Data Mining phase involves training the neural network to learn patterns and classify the images accurately. Finally, in the Interpretation/Evaluation stage, the performance of the model is assessed, and the results are interpreted to refine and improve the process. Each of these stages will be discussed in detail in the following sections, providing a comprehensive understanding of the image classification workflow.

- Data Selection: Collect relevant images for the classification task.
- Data Preprocessing: Perform operations such as normalization, resizing, and data augmentation on the images.

- Data Mining: Train the neural network to learn patterns and classify the images.
- Interpretation/Evaluation: Assess the model's performance and interpret the results to improve the process.

3.3 Dataset

Selecting an appropriate dataset is a critical aspect of any deep learning project as it directly influences the model's performance. Firstly, understanding the problem domain and defining the specific task or goal of the project is essential. This helps in identifying the types of data required and the relevant features to be considered. This way, it involves several key considerations to ensure that the data aligns with the set goals and provides sufficient diversity and relevance for effective model training.

In some cases, leveraging publicly available datasets or benchmark datasets specific to the problem domain can provide valuable resources for model development and evaluation. However, in this context there were no public datasets for this specific case, with the four nationalities garment types for the classification aimed. Thus, we decided to assemble a Dataset that would best represent this domain.

At first, the idea was to use a web crawler to collect the data. And the chosen one was `icrawler-0.6.7`, a multi-thread crawler framework with many built-in image crawlers. However, with this tool, there was no success in obtaining enough data for each label, in this case, German, Indian, Japanese, and Spanish. Because of this, a manual scrolling and collecting of images was made yet not fulfilling a great number of data.

Here in Listing 3.1 is how this crawler was utilized. Apart from `GoogleImageCrawler`, `BingImageCrawler`, and `BaiduImageCrawler` were also used. And many keywords were tested, with none getting more than around a hundred images for each label.

Listing 3.1: Image Crawler

```
filters = dict(size='large')
google_Crawler = GoogleImageCrawler(storage={'root_dir':r'/content/drive/MyDrive/UFAL/TCC/
    TradicionalClothesDataset/icrawler/german'})
google_Crawler.crawl(keyword='traditional german dress', filters=filters, max_num=1000)
```

Additionally, evaluating the quality and quantity of data available is crucial to determine if the dataset adequately represents the underlying population and encompasses the variability present in real-world scenarios. Respectively, after a manual cleaning of the Dataset, the quantity for each label was even smaller. For this reason, the models run with this dataset were presenting overfitting and its graphs curves and metrics were not showing good results. The Table 3.1 shows the quantities on each label on the cleaned version of this Dataset.

Labels	German	Indian	Japanese	Spanish
Quantity	55	138	100	62

Table 3.1: Number of images for each label in the Dataset created using icrawler

Accordingly, when noticed that this problem with few data was determining. A more time-consuming and complex approach was decided. A new dataset should be created from scratch with self-implemented crawling scripts, aiming to classify traditional clothing styles with success. These scripts in Python utilized the Selenium library to scroll several online shopping websites for each nationality and garment type. Appendix A shows a table with all the specific links from where numerous images were downloaded, totaling 82 websites.

Selenium is a powerful and widely used open-source library for automating web browsers. One of its key features is WebDriver, which allows users to control web browsers programmatically and execute commands such as clicking buttons, filling forms, and navigating through web pages. Overall, Selenium is a versatile and essential tool for web developers and testers, facilitating the creation of robust and efficient automated tests and web scraping scripts.

In this context, the scripts enabled scrolling through all the websites listed and downloading to a local folder the garment images by filtering these elements by their paths. Also, there are different types of websites, ones with an infinite scroll down, ones with a scroll down that eventually shows a load more button, and websites that have pagination. So to scroll and download all the images in a starting point URL, three different types of implementation were considered. Overall, the page scraping was made with a recursive function as seen in Listing 3.2

Here in Listing 3.3 is a segment of the code implementation. Note that the XPATH parameter for the `find_elements` in the URL of this example was `//img[contains(@title, 'Dirndl')]`, but many variations existed. As well as for the `get_attribute` that varied from `'url'` to `'srcset'`, `'data-original'`, and more.

Listing 3.2: Function that crawls through websites

```
def scrape_pages_recursively(driver, current_page, URL, images_urls):
    scroll_to_end(driver)
    images_urls = process_images(driver, images_urls)

    if navigate_to_next_page(driver, current_page, URL):
        images_urls = scrape_pages_recursively(driver, current_page + 1, URL, images_urls)

    return images_urls
```

Listing 3.3: Function that define image paths using selenium webdriver

```
def process_images(driver, images_urls):
    images_tags = driver.find_elements(By.XPATH, "//img[contains(@title, 'Dirndl')]")
    new_image_urls = [img.get_attribute('src') for img in images_tags]
    images_urls.extend(new_image_urls)
    return images_urls
```

While running the many versions of scripts, a folder for each website was filled. After that, a manual selection was made gathering data in another folder that would be the path for the official Dataset. This folder was divided into four sub folders for each label, with each composed of all the images for all websites of your respective nationality.

In this manual selection, a few images were discarded if not considered clean. The rules considered for this choice are, images with more than one person and garment, images with backgrounds composed of many items and details making it a messy background, images with only garments in the image, without a person wearing it, and images not showing the entire garment wore by the person, being to zoomed in.

Apart from the manual selection, two small scripts were utilized for the refinement of the Dataset. A script to crop images where the person using the traditional clothing was to zoomed out, showing more of the background them the garment. And a shuffle script to distribute randomly the images from all the shopping websites. Since some websites had patterned scenarios, avoiding an order of similar images. The scripts using selenium that downloaded the dataset image as well as the scripts used to crop, shuffle, etc, can be viewed on GitHub, these links are present in Appendix C.

Consequently, with the development of these scripts, the total of images gathered was way higher. Even after the mentioned filtering, the number of data for each label was promising, as can be seen in Table 3.2.

Labels	**German**	**Indian**	**Japanese**	**Spanish**
Quantity	1,388 images	1,744 images	1,548 images	1,199 images
Percentage	23.61%	29.67%	26.34%	20.38%

Table 3.2: Distribution of Images for each label in the Dataset created with our own Scripts.

3.4 Pre-processing

The pre-processing step is a crucial component of the workflow when using Convolutional Neural Networks for classification tasks, as well as for any machine learning technique. This step involves preparing the raw input data before feeding it into the model for training or inference. Pre-processing plays a vital role in ensuring the effectiveness and efficiency of the model in learning meaningful patterns from the data and making accurate predictions.

Here are the steps performed in our pre-processing:

- Defining Dataset path
- Loading data in a train, validation, and test folder
- Saving images with redefinition and normalization

- Including augmented images on the train directory

Some versions of the model can also be viewed on GitHub, link present in Appendix C. This repository includes code for the pre-processing steps, as well as for future sections covering creating the convolutional model, saving the model, and evaluating the model. For now, let us focus on the pre-processing steps.

3.4.1 Dataset path

The dataset built process was already discussed, the original Dataset created has a total of 5,879 images. The path was set and prepared as seen in Listing 3.4

Listing 3.4: Defining Dataset Path

```
import os
dataset_directory = 'D:\\.Dataset05032024 - TCC\\labels'
os.makedirs(dataset_directory, exist_ok=True)
os.chdir(dataset_directory)
```

Considering that data augmentation would be applied and various aspects of the model, such as logs, model checkpoints, and training history, needed to be saved, the dataset was stored on an external hard drive. This setup ensured adequate storage capacity and facilitated the management and backup of large volumes of data, thereby supporting the extensive pre-processing and training processes.

3.4.2 Train, Validation, and Test

Loading data into train, validation, and test folders plays a critical role as it directly impacts the performance, generalization, and interpretability of the resulting models. Another thing is that, organizing data into these folders streamlines the model development process and enhances reproducibility. Since with clearly defined subsets, researchers and practitioners can easily track the origin and distribution of the data used for training and evaluation. In Listing 3.5 we can see how the paths were set.

Listing 3.5: Defining train, validate and test subsets path

```
from shutil import copyfile
train_directory = 'D:\\.Dataset05032024 - TCC\\result\\train'
validation_directory = 'D:\\.Dataset05032024 - TCC\\result\\validate'
test_directory = 'D:\\.Dataset05032024 - TCC\\result\\test'
```

The training subset is used to train the model's parameters and learn from the provided examples. The validation set, on the other hand, serves as a checkpoint during training, allowing practitioners to monitor the model's performance on unseen data and tune hyperparameters accordingly. Finally, the test set is reserved for evaluating the model's performance once training is complete. It provides an unbiased assessment of the model's generalization ability and helps detect any overfitting tendencies.

Moreover, defining the percentages for train, validation, and test sets is a crucial step, and the precise percentages allocated to each subset depend on various factors, including the size of the dataset, the complexity of the task, and computational constraints. Striking the right balance between the train, validation, and test percentages is essential for developing robust and reliable CNN models that can effectively generalize to unseen data. Nonetheless, the majority of the dataset is usually allocated to the training set, and in this case, the chosen percentages were 69%, 17%, and 14% respectively, mimicking the percentages of Aditya et al. (2023).

In Listing 3.6, we can see how the percentages were defined and the pseudo-code of how the folders are filled.

Listing 3.6: Preparing train, validate and test subsets

```
def load_data(dataset_dir, train_dir, validation_dir, test_dir):

    Set validation_percent to 0.17
    Set test_percent to 0.14

    For each class_folder in dataset_dir:
        Get images in class_folder
        Calculate the number of validation and test images
        Split images into train, validation, and test sets
        Create directories if they do not exist
        Copy images to respective directories
```

3.4.3 Redefinition and Normalization

CNN models often require input images to be of a fixed size. Therefore, preprocessing may involve resizing the images to a standard size before feeding them into the network. It reduces computational complexity and memory requirements during training and inference and ensures consistency in input dimensions across all samples, allowing the model to learn spatial features effectively, leading to faster training times and lower resource consumption.

The selected resizing was 28×28 pixels following the FashionMNIST dataset. Resizing to a smaller size can help mitigate overfitting by reducing the model's capacity and forcing it to focus on the most salient features of the images. Despite the reduction in resolution, downsampling the image input can preserve essential visual information, especially in tasks where fine-grained details are less critical, such as object classification or pattern recognition.

Preprocessing also often includes normalization, which involves scaling the input data to a standardized range. By scaling the pixel values, normalization ensures that the features have similar scales, preventing certain features from dominating others during training. This practice is crucial for maintaining the stability of the optimization process and accelerating training convergence. Additionally, normalizing the data to a standardized range facilitates comparisons between different features and improves the interpretability of the model's learned parameters. The process involved transforming the values of pixel intensities in images to a range between 0 and 1.

Overall, defining a resizing of 28×28 pixels in the preprocessing stage of a CNN model strikes a balance between computational efficiency, model complexity, and performance, making it a practical choice for various applications. Analogously, normalization plays a vital role in ensuring the robustness and effectiveness of the CNN model during both the training and inference phases. In Listing 3.7 we can see the preprocessing function utilized.

Listing 3.7: Pre processing function

```
target_size = (28, 28) # Size redefinition

def preprocess_image(image_path):

    img = load_img(image_path, target_size=target_size)
    img_array = img_to_array(img)
    img_array /= 255.0  # Normalization [0, 1]

    return img_array
```

Figure 3.1 illustrates the process of resizing input images as part of the data preprocessing steps for training neural networks. Subfigure (a) shows an original input image with its inherent dimensions and resolution. In contrast, subfigure (b) displays the resized version of the same image, adjusted to a standard dimension suitable for input into the neural network. This resizing step is crucial as it ensures uniformity in image dimensions across the dataset, facilitating efficient and consistent processing by the neural network. The resized images maintain the essential features of the original inputs while conforming to the required input size for the model, thereby optimizing the training process.

(a) (b)

Figure 3.1: Resized Image Example

3.4.4 Data Augmentation

Data augmentation is a critical technique in machine learning and computer vision that plays a pivotal role in enhancing the robustness and generalization ability of models. It involves artificially expanding the size of a dataset by applying a variety of transformations to the existing data

samples, such as rotation, scaling, cropping, flipping, or adding noise. By introducing diverse variations to the input data, data augmentation helps the model learn invariant features, thereby improving its ability to accurately classify or recognize objects under different conditions.

Additionally, data augmentation reduces the risk of overfitting by exposing the model to a broader range of data patterns, preventing it from memorizing the training set and thus enabling a better performance. Overall, data augmentation serves as a powerful tool for improving model performance and increasing dataset diversity. Also, by applying data augmentation techniques only to the training set, practitioners can prevent data leakage and ensure that the validation and test sets accurately reflect real-world scenarios. In Listing 3.8 are the augmentation parameters applied.

Listing 3.8: Augumentation Definitions

```
from imgaug import augmenters as iaa

rotation_range = 10
width_shift_range = 0.1
height_shift_range = 0.1
shear_range = 0.1
zoom_range = 0.1
horizontal_flip = True

generating_augmented_images = iaa.Sequential([
    iaa.Fliplr(p=0.5),
    iaa.Sometimes(0.4, iaa.Affine(rotate=(-rotation_range, rotation_range))),
    iaa.Sometimes(0.3, iaa.Affine(translate_percent={
        "x": (-width_shift_range, width_shift_range),
        "y": (-height_shift_range, height_shift_range)})),
    iaa.Sometimes(0.2, iaa.Affine(scale=(1 - zoom_range, 1 + zoom_range))),
    iaa.Affine(shear=(-shear_range, shear_range)),
], random_order=True)
```

Figure 3.2 demonstrates the application of data augmentation techniques to input images, which is essential for enhancing the diversity and robustness of the training dataset. Subfigures (a) and (c) display the original images, while subfigures (b) and (d) show the augmented versions of these images. The augmentation techniques applied include transformations such as rotation, flipping, and scaling. These transformations generate new, varied samples from the original images, helping the neural network to generalize better by exposing it to a wider range of data variations. This process reduces the likelihood of overfitting and improves the model's performance on unseen data.

3.5 Creating the Convolutional Neural Network Model

The architecture of the Convolutional Neural Network (CNN) model used in this project was inspired by the work of Aditya et al. (2023). This architecture is designed to process images of size $28 \times 28 \times 3$ (height, width, and channels respectively), which is typical for tasks involving

(a) (b) (c) (d)

Figure 3.2: Examples of augmented versions of images

small image datasets. In Listing 3.9 we can see the code part where the Keras sequential model is created and its layers definitions.

Listing 3.9: CNN Model Architecture

```
model = tf.keras.Sequential([

    BatchNormalization(input_shape=(28, 28, 3)),

    Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.1),

    Conv2D(filters=64, kernel_size=(3,3), padding='same', activation='relu'),
    MaxPooling2D(pool_size=(2, 2), strides=2),
    Dropout(0.2),

    Flatten(),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(64, activation='relu'),
    BatchNormalization(),

    Dense(4, activation="softmax")
])
```

3.5.1 Model Architecture

Figure 3.3 illustrates a diagram obtained from the upload of our trained and saved Keras model, "traditional_clothing_classifier.h5" in Netron, a viewer for neural network, deep learning, and machine learning models. Netron provides an intuitive interface to visualize and explore model structures, making it easier to understand and debug complex architectures.

The network begins with a batch normalization layer, which normalizes the inputs and helps in stabilizing and accelerating the training process. This is followed by a convolutional layer

with 64 filters and a kernel size of 3×3, using the ReLU activation function to introduce non-linearity. Subsequently, a max pooling layer with a pool size of 2×2 reduces the spatial dimensions, aiding in the extraction of dominant features while reducing computational complexity. Dropout layers are strategically placed after the pooling layers with dropout rates of 0.1 and 0.2, respectively, to prevent overfitting by randomly disabling a fraction of the neurons in training.

Continuing, the flattened layer converts the 2D matrix into a 1D vector, which is then fed into a dense (fully connected) layer with 256 units and ReLU activation. Another dropout layer with a dropout rate of 0.5 is applied here to further mitigate overfitting. After that, another dense layer with 64 units and ReLU activation is followed by a batch normalization layer to normalize the outputs and improve the model's robustness. Finally, the output layer is a dense layer with 4 units and a softmax activation function, which outputs the probabilities for each of the four classes. This comprehensive architecture leverages batch normalization, dropout, and convolutional operations to build a robust and effective model for image classification tasks.

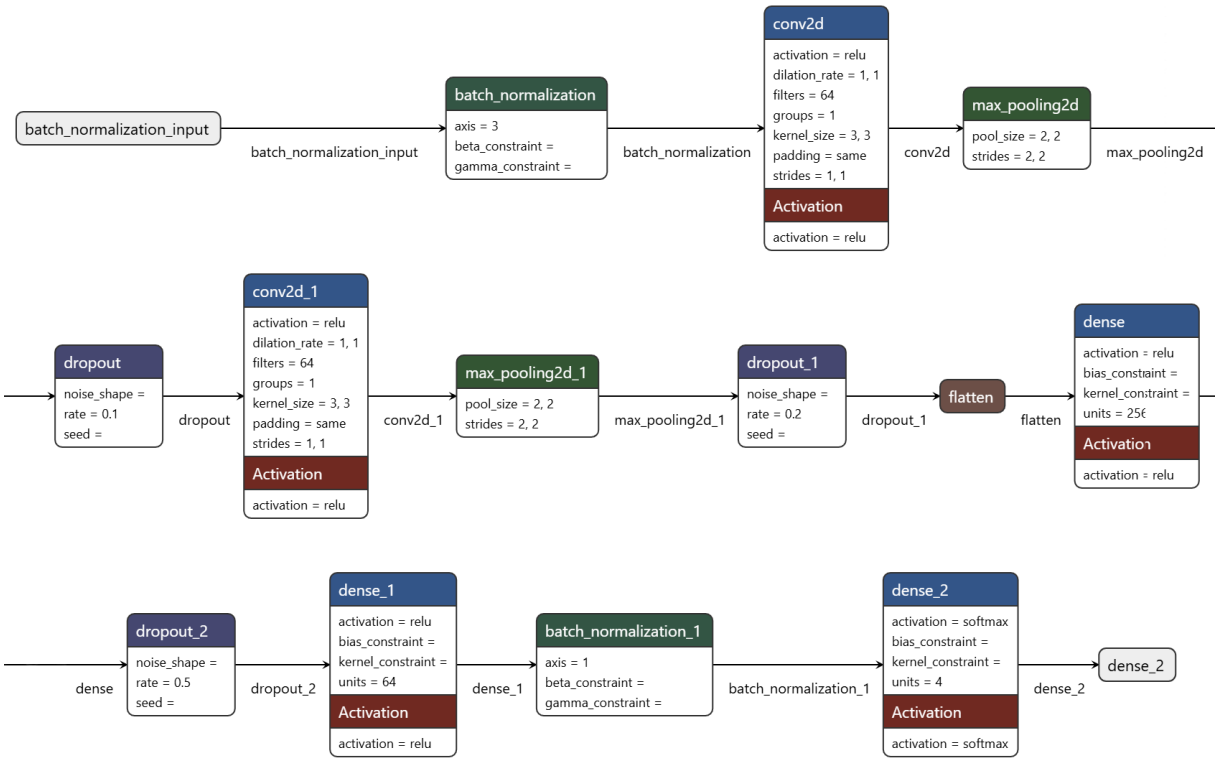


Figure 3.3: Architecture of the convolutional neural network (CNN) used in this study. The model includes layers for batch normalization, convolution, max pooling, dropout, and dense (fully connected) layers.

The model was run on a notebook with an Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz processor and 24.0 GB of installed RAM. This setup, along with a one T external drive, used to contain the dataset, provided adequate computational power and memory to handle the training and evaluation of the Convolutional Neural Network (CNN) model, ensuring efficient processing and storage capabilities.

3.5.2 Overview of the Layers

As seen, the CNN model is composed of the following layer types:

- **BatchNormalization:** Normalizes the input to the network to improve the convergence of the training process.
- **Conv2D (Convolutional Layer):** Applies 64 filters of size 3×3 with ReLU activation to capture spatial features from the input images. The padding is set to 'same' to ensure the output has the same spatial dimensions as the input.
- **MaxPooling2D (Pooling Layer):** Reduces the spatial dimensions of the feature maps by taking the maximum value over 2×2 regions with a stride of 2, effectively down-sampling the feature maps.
- **Dropout:** Randomly sets a fraction of input units to 0 at each update during training time to prevent overfitting. The dropout rate is set to 0.1 after the first convolutional and pooling layers, and 0.2 after the second layers.
- **Flatten:** Flattens the input from the second and last convolutional and pooling layers to a 1D array, preparing it for the fully connected layers.
- **Dense (Fully Connected Layer):** Consists of 256 neurons with ReLU activation.
- **Dropout:** Here the dropout rate is set to 0.5 after the fully connected layer.
- **Dense (Fully Connected Layer):** Consists of 64 neurons with ReLU activation.
- **BatchNormalization:** Here it normalizes the output of the previous dense layer.
- **Dense (Output Layer):** Consists of four neurons with softmax activation, providing probabilities for each of the four output classes.

Table 3.3 ilustrates a resume. This architecture is designed to effectively extract features from the input images and classify them into one of four classes. The use of BatchNormalization, Dropout, and a combination of convolutional and fully connected layers helps in building a robust model that generalizes well to new data.

Table 3.3: CNN Model Architecture

Layer Type	Filters	Kernel Size	Activation
BatchNormalization	-	-	-
Conv2D	64	3x3	ReLU
Continued on next page			

Table 3.3 – continued from previous page

Layer Type	Filters	Kernel Size	Activation
MaxPooling2D	-	2x2	-
Dropout	-	-	-
Conv2D	64	3x3	ReLU
MaxPooling2D	-	2x2	-
Dropout	-	-	-
Flatten	-	-	-
Dense	256	-	ReLU
Dropout	-	-	-
Dense	64	-	ReLU
BatchNormalization	-	-	-
Dense	4	-	Softmax

3.5.3 Compiling and Training the Model

After defining the architecture, the next step is to compile and train the model. The Adam optimizer with a learning rate of 0.001 is used for its efficient gradient-based optimization. The loss function chosen is 'sparse_categorical_crossentropy', suitable for multi-class classification problems. And the model is evaluated using the accuracy metric as can the seen in Listing 3.10.

Listing 3.10: Compiling and Training the CNN Model

```
adam = Adam(learning_rate=0.001)
model.compile(optimizer=adam, loss='sparse_categorical_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint(checkpoint_path, monitor='val_accuracy', save_best_only=True,
    verbose=1)
early_stopping = EarlyStopping(monitor='accuracy', patience=100)
csv_logger = CSVLogger(filename=history)

history = model.fit(
    train_dataset,
    epochs=1000,
    validation_data=validation_dataset,
    callbacks=[csv_logger, checkpoint, tensorboard_callback, early_stopping],
    verbose=1
)
```

Also in Listing 3.10 we see that, during training, several callback functions are employed:

- **ModelCheckpoint:** Saves the model with the highest validation accuracy during training, ensuring the best version of the model is retained.
- **EarlyStopping:** Monitors the training process and stops training if the accuracy does not improve for a specified number of epochs (patience).

- **CSVLogger:** Logs training and validation metrics to a CSV file, enabling detailed analysis and visualization of the training process.
- **TensorBoard Callback:** Provides visualizations for the training process, allowing monitoring of metrics in real-time.

The training process involves evaluating the model's performance on the validation dataset after each epoch, and adjusting the model weights to minimize the loss and maximize accuracy. The model starts training with a default of 1000 epochs, with early stopping set to potentially halt training earlier if the accuracy does not improve. The training log demonstrates the iterative improvement of the model, showing both the training and validation loss and accuracy. We see that after 369 epochs, the model achieved a training accuracy of 91.55% and a validation accuracy of 94.78%, with a training loss of 0.2929 and a validation loss of 0.1994. The training was halted because the validation accuracy did not improve beyond the best result reached (0.95988), as can be seen in the log excerpt below:

```
Epoch 369/1000
5412/5415 [============================>.] -
ETA: 0s - loss: 0.2929 - accuracy: 0.9156
Epoch 369: val_accuracy did not improve from 0.95988
5415/5415 [==============================] -
147s 27ms/step
    - loss: 0.2929
    - accuracy: 0.9155
    - val_loss: 0.1994
    - val_accuracy: 0.9478
```

3.6 Saving the Model

After the training, the Keras model was saved as "traditional_clothing_classifier.h5", as shown in Listing 3.11. The HDF5 format is used by Keras/TensorFlow and is a hierarchical data format widely used for storing large amounts of data. Typically, .h5 files store the entire model architecture, weights, and optimizer states, making it easy to save and load models for inference or continued training. The .h5 format facilitates the seamless saving and loading of models within the Keras/TensorFlow ecosystem, ensuring that the entire model, including architecture, weights, and training configuration (loss and optimizer), is preserved.

Listing 3.11: Saving CNN Model

```
1 trained_model_path = 'traditional_clothing_classifier.h5'
2 model.save(trained_model_path)
```

This trained model, along with saved details is available on Hugging Face (Appendix D), a popular repository for datasets and machine learning models. This will enable other researchers and developers to build upon our work utilizing deep learning techniques.

3.7 Evaluating the Model

The evaluation of the model was conducted using the test dataset to measure its generalization capability. This comprehensive evaluation approach combined metrics evaluation, learning curves analysis, and confusion matrix analysis to gain a holistic understanding of the model's performance.

Additionally, the Metrics Space in the Hugging Face repository mentioned in the previous section includes a TensorBoard, which allows us to observe not only scalar graphs of accuracy and loss but also graphical representations of the model's architecture and layer histograms. These visualizations provide deeper insights into the model's behavior and training dynamics.

In this work, let us focus on the following evaluations.

3.7.1 Metrics Values

The primary evaluation metrics used were accuracy and loss. Accuracy measures the overall correctness of the model by calculating the proportion of correctly predicted instances out of the total instances. Loss quantifies the error made by the model during prediction, with lower values indicating better performance. Listing 3.12 shows how the evaluation is performed.

Listing 3.12: Getting Model Evatuation

```
test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
print(f'Test -> Loss: {test_loss}, Accuracy: {test_accuracy}')
```

3.7.2 Learning Curves

The evaluation process also involved analyzing the learning curves, which plot the training and validation accuracy and loss over epochs. These curves help in understanding the model's learning behavior and identifying potential issues such as overfitting or underfitting. Listing 3.13 shows how they are generated by accessing the saved model's history.csv.

Listing 3.13: Getting Learning Curves

```
df_history = pd.read_csv(history.csv)

accuracy = df_history['accuracy'].to_numpy()
val_accuracy = df_history['val_accuracy'].to_numpy()
loss = df_history['loss'].to_numpy()
val_loss = df_history['val_loss'].to_numpy()

plot_curves(accuracy, val_accuracy, loss, val_loss)
```

3.7.3 Confusion Matrix

Lastly, the confusion matrix analysis was chosen to provide a detailed view of the model's performance for each class. It allows us to visualize the true positives, false positives, false negatives, and true negatives for each class, offering insights into specific areas where the model excels or may require improvement. In Listing 3.13 we see how the Confusion Matrix is obtained. Showing the process of predicting the labels for the test dataset, extracting the true labels, and then calculating and displaying the confusion matrix.

Listing 3.14: Getting the Confusion Matrix

```
predictions = model.predict(test_dataset)

predicted_labels = np.argmax(predictions, axis=1)  # Use argmax para obter a classe prevista
true_labels = np.array([label for _, label in test_dataset.unbatch()])
print("True Labels: ", true_labels)
print("Predicted Labels: ", predicted_labels)

conf_matrix = confusion_matrix(true_labels, predicted_labels)
print("Confusion Matrix:\n", conf_matrix)

class_labels = ["German", "Indian", "Japanese", "Spanish"]
cm = confusion_matrix(true_labels, predicted_labels)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Purples)
plt.show()
```

3.8 Results and Discussions

In this section, we present and analyze the performance of the Convolutional Neural Network (CNN) model on the test dataset. This analysis includes the mentioned topics in the evaluation section. A combination of metrics, learning curves analysis, and confusion matrix analysis.

3.8.1 Metrics Values

The evaluation results in Listing 3.12 indicate that the model has a high level of accuracy in classifying the images, with a reasonably low loss value.

The best results achieved were:

- **Loss:** 0.21722690761089325
- **Accuracy:** 0.941534698009491

This suggests that the model has effectively learned the features of the dataset and can generalize well to new data.

3.8.2 Learning Curves

To further understand the model's performance, we can visualize the training and validation curves for loss and accuracy obtained from Listing 3.13. These visualizations help in diagnosing potential issues such as overfitting or underfitting and provide a clear picture of how the model learns over time. These, shown in Figure 3.4, indicated a good fit, with the training and validation accuracy increasing and stabilizing over time, and the loss decreasing consistently.

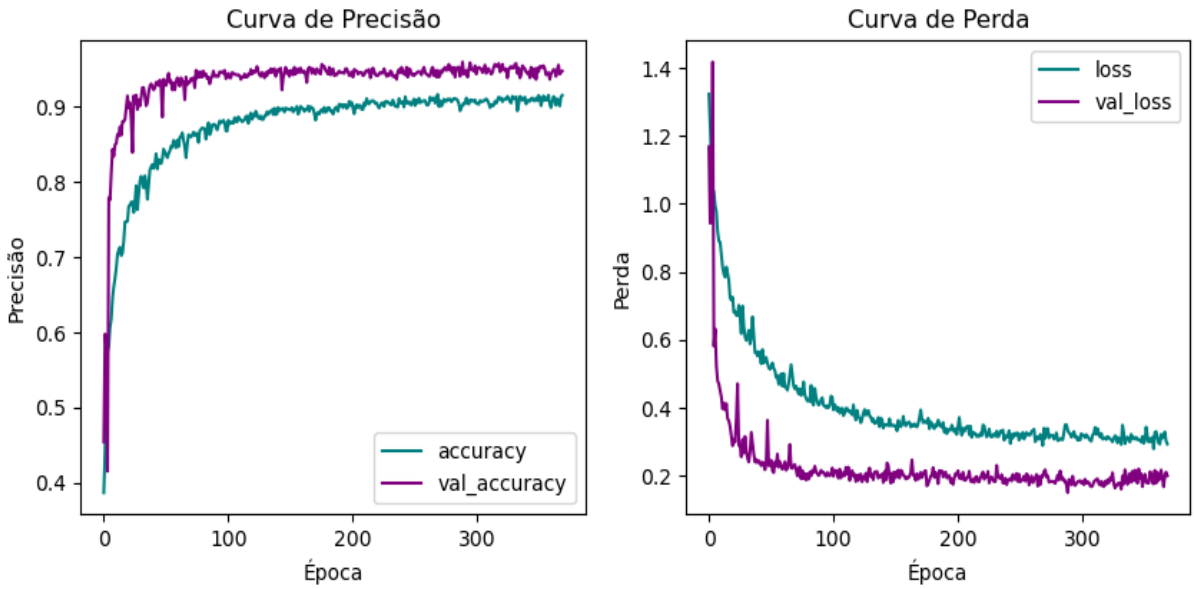


Figure 3.4: Result Plots

It is evident that the training and validation loss decrease consistently over epochs, and the accuracy improves correspondingly. This indicates a well-converged model with minimal signs of overfitting. Reinforced the conclusion that the model is well-tuned and capable of high performance in real-world applications.

3.8.3 Confusion Matrix

Lastly, from the confusion matrix obteined from Listing 3.13 and shown in Figure 3.5, we observe that, from 654 test images, 194 for the German label, 244 for the Indian label, 216 for the Japanese label, and 167 for the Spanish label. The model's performance across these different categories was:

- From The model correctly classified 186 instances of the German class, with only a few misclassifications (3 as Indian, 4 as Japanese, and 1 as Spanish).
- The Indian class has 238 correctly classified instances, with minimal misclassifications (2 as German, 1 as Japanese, and 3 as Spanish).

- For the Japanese class, 200 instances were correctly classified, but there were some misclassifications (5 as German, 7 as Indian, and 4 as Spanish).
- The Spanish class had 149 correctly classified instances, with misclassifications including 2 as German, 8 as Indian, and 8 as Japanese.

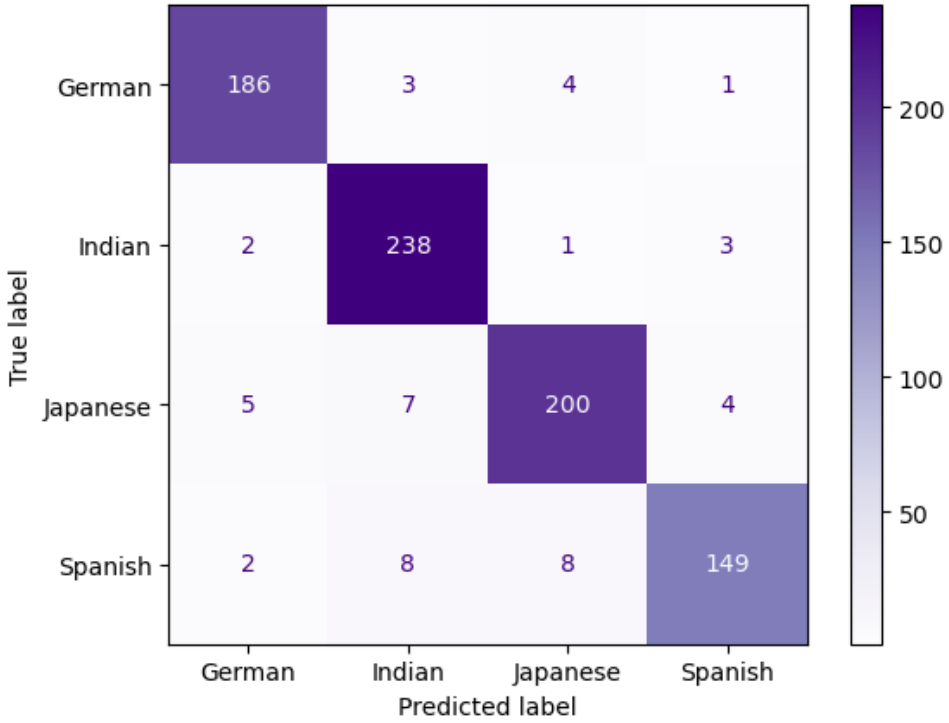


Figure 3.5: Confusion Matrix of the Model's Predictions

Label	**Accuracy (%)**
German	95.88%
Indian	97.54%
Japanese	92.59%
Spanish	89.22%

Table 3.4: Classification Accuracy for Each Label

These results indicate that the model performs exceptionally well for the German and Indian classes, with high true positive rates and low false positive rates. The Japanese and Spanish classes show slightly higher misclassification rates, suggesting areas where the model could be further refined.

Overall, the confusion matrix highlights the model's strong performance and its ability to distinguish between different classes effectively. The few misclassifications present can be attributed to the similarities between some classes, which could be addressed in future iterations of the model through techniques such as data augmentation or fine-tuning the model parameters.

4

Related Work

This chapter reviews existing literature and previous studies relevant to this dissertation. It compares and contrasts different approaches to garment classification and neural network applications in fashion. In doing so, it provides context for the current research within the broader field of study and identifies the gaps that this dissertation aims to fill. Additionally, it allows us to discover techniques that inspire future improvements in this work.

We have learned that in recent years, there has been significant interest in applying computer vision techniques to various aspects of fashion. Cheng et al. (2021) provides a comprehensive survey of the state of fashion research, discussing multiple directions in which fashion is being explored through computer vision. This survey highlights the increasing importance of automated garment classification and the potential of deep learning models to enhance the accuracy and efficiency of these tasks.

Another fact is that the FashionMNIST dataset, introduced by Xiao (2017), has become a benchmark for evaluating image classification models in the context of fashion. Several studies have utilized this dataset to experiment with Convolutional Neural Networks (CNNs) and other machine learning techniques. For instance, Teow (2020) and Leithardt et al. (2021) have conducted extensive experiments with CNNs on the FashionMNIST dataset, demonstrating the effectiveness of these models in classifying various fashion items. Similarly, Kayed et al. (2020) explored different classification techniques using FashionMNIST, further validating the utility of this dataset for fashion-related research.

The use of CNNs in garment classification has been widely studied, with various techniques being employed to enhance model performance. Lydia and Chandrasekar conducted a comparative analysis of different regularization techniques, emphasizing their importance in improving model generalization. Regularization methods such as dropout, discussed by Shen and Shafiq (2018), have been shown to prevent overfitting and enhance the robustness of CNNs. These studies underscore the critical role of regularization in developing effective deep learning models for fashion classification.

In terms of specific implementations, Aditya et al. (2023) presented a CNN architecture that balances time efficiency and accuracy, inspiring the model architecture used in this dissertation. Their work demonstrated the practicality of CNNs in real-world fashion classification tasks and provided a foundation upon which this research builds.

At this point, we know that the classification of traditional garments is a complex task that involves recognizing and categorizing clothing items from various cultures. And that not many studies have explored this field, demonstrating a gap in the academy to related research in this matter. Nevertheless, the few studies that dived into ethnic style classification have proven the rise of this field's investment in the last years and the potential of CNNs in classifying these traditional garments. Highlighting the importance and relevance of this line of research.

In 2018, Nawaz et al. (2018) developed an automatic categorization system for five different clothing items that define the traditional clothing of Bangladesh using CNNs. Their research demonstrated the early potential of deep learning in ethnic garment classification, setting a foundation for subsequent advancements in the field. Similar to our study, their work utilized CNNs for image classification, but it focused specifically on Bangladeshi traditional clothing items, whereas our study aims to classify traditional entire garments from a broader range of nationalities.

In 2019 Sandhu and Vaishnav worked on the multi-label geographic classification of apparel, including traditional garments like Korean wear (Hanbok), Scottish wear (Kilts), Japanese wear (Kimono), Indian wear men (Kurta), Indian wear women (Saree), Spanish wear (Mantilla), Men western wear, and Women western wear. Their approach leverages CNNs to classify garments based on geographic origin, showcasing an attempt to unite different types of traditional garments from various nationalities and backgrounds into a comprehensive classification system. This approach aligns closely with our study's objective to classify traditional garments from multiple cultures, emphasizing the versatility and robustness of CNNs.

In 2023, Li et al. (2023) focused on the classification of She ethnic clothing, a distinct minority in southeastern China, using a method that combines the Flower Pollination Algorithm (FPA) and CNN. This innovative combination of techniques highlights the evolving complexity and specificity of methods in this field. Our study, while not incorporating FPA, or any metaheuristic optimization algorithm, similarly seeks to innovate by applying CNNs to a diverse set of traditional garments from multiple cultures.

Also in 2023, Scharff et al. (2023) developed a system for the classification of Senegalese fashion clothing, employing CNNs to achieve high accuracy. Their research emphasizes the adaptability of CNNs to different cultural contexts, a principle that aligns with our study's goal of accurately classifying garments from various nationalities, including German, Indian, Japanese, and Spanish traditional clothing.

Moreover, Ai et al. (2023) aimed to enhance the design and classification of ethnic garments, with a specific emphasis on Miao women's apparel, another ethnic group in China known for their vibrant colors, intricate designs, and cultural significance. This study's focus on design

enhancement alongside classification parallels our research's broader implications for the fashion industry, where accurate classification aids in preserving and innovating traditional designs. In 2024, Baalamash et al. (2024) explored the classification of Saudi traditional costumes using deep learning techniques. Similarly, Nandi et al. (2024) focused on the classification of Bangladeshi sarees using deep learning techniques.

Overall, the successful application of CNNs in these studies underscores the versatility and robustness of deep learning models in handling diverse and culturally significant clothing styles. Furthermore, accurately classifying traditional garments using CNNs offers numerous practical benefits, such as preserving cultural heritage through digitization and providing a digital archive for future generations, significantly contributing to computer vision and cultural heritage preservation. In the fashion industry, designers and manufacturers can also use these classification models to understand trends and preferences, aiding in inventory management and targeted marketing strategies. In e-commerce, these models enhance search and recommendation features, improving the shopping experience for these items.

By comparing our study with these related works, we highlight the continuous advancement in using CNNs for traditional garment classification, demonstrating the unique contributions of each study to this evolving field. This comparisons underscores that traditional garment classification using CNNs is a relatively new and emerging line of study. The limited number of studies and their focused nature on specific ethnic styles illustrate the initial steps being taken in this area. Our research contributes to this nascent field by expanding the scope to include four nationalities, thereby addressing the identified research gaps and furthering the potential applications of CNNs in preserving cultural heritage and enhancing the fashion industry.

5

Concluding Remarks

This chapter summarizes the findings of this work and reflects on its contributions. It discusses the limitations of the study, addressing potential weaknesses and areas for improvement. Additionally, the practical implications of the research for the fashion industry and the academic community are highlighted. Finally, the chapter outlines directions for future work, suggesting how the research can be expanded or applied in different contexts.

5.1 Analysis of Results

By combining the analysis of metrics such as accuracy, loss, learning curves, and confusion matrix, we can conclude that the model is well-balanced, robust, and capable of high performance in real-world applications. These comprehensive evaluation metrics provide confidence in the model's ability to generalize effectively and make accurate predictions on new data.

The performance metrics achieved by the model highlight several important aspects:

- **High Accuracy:** The accuracy of 94.15% demonstrates the model's strong capability in correctly classifying the majority of the test samples. This level of accuracy is indicative of a well-trained model with effective feature extraction and classification mechanisms.
- **Low Loss:** The loss value of 0.2172 indicates that the discrepancies between the predicted and actual labels are minimal. This low loss value reinforces the notion that the model has learned to make precise predictions with minimal errors.
- **Consistent Learning Curves:** The learning curves for both training and validation loss and accuracy indicated a good fit, with the training and validation accuracy increasing and stabilizing over time, and the loss consistently decreasing. This consistency in the

learning curves is a strong indication that the model has been well-tuned and is capable of learning effectively from the data without significant overfitting or underfitting.

- **Confusion Matrix Insights:** The confusion matrix provided a detailed view of the model's performance across different classes. The high true positive rates for the German and Indian classes, and relatively low false positive rates, demonstrate the model's effectiveness in these categories. Although the Japanese and Spanish classes showed slightly higher misclassification rates, techniques can further refine these.

Overall, the results from this study demonstrate the significant potential of using Convolutional Neural Networks (CNNs) for the classification of traditional ethnic garments. The high accuracy, low loss, consistent learning curves, and detailed confusion matrix analysis collectively suggest that the model is robust and not significantly overfitting the training data. This balance is crucial for ensuring the model performs well on unseen data. And, this way confirms that the model is well-suited for practical deployment and capable of contributing valuable insights and improvements to the fashion technology and e-commerce industries.

5.2 Limitations

Despite the promising results, this study has some limitations that should be acknowledged:

- **Dataset Size:** The dataset size, while adequate for initial studies, may be insufficient to capture the full variability and complexity of traditional ethnic garments. A larger dataset could provide a more robust training process and improve the model's performance.
- **Limited Variety in Clothing:** The dataset primarily contains feminine clothing items. There is a notable absence of masculine clothing, which limits the model's applicability to a broader range of fashion products.
- **Limited number of Nationalities:** The dataset includes a limited number of nationalities, potentially affecting the model's ability to generalize across diverse demographic groups. This limitation might impact the model's performance in a global e-commerce context.

5.3 Implications for Practice

The findings from this study have several practical implications, particularly for the e-commerce industry. While the current version of the model may not yet bring significant improvements in these areas, the research and functional model available on Hugging Face (Appendix D) provides a solid foundation for ongoing refinement and development.

Continued efforts in this research can lead to substantial advancements over time, ultimately making a meaningful impact in the field. So, by further developing and fine-tuning the model,

we can expect to see real improvements in various applications, enhancing the overall effectiveness and efficiency of e-commerce platforms.

These areas in the e-commerce industry are:

- **Enhanced Product Recommendations:** The model's ability to accurately classify clothing items can be leveraged to improve product recommendation systems. By accurately identifying and categorizing products, e-commerce platforms can provide more personalized and relevant suggestions to customers.
- **Inventory Management:** Retailers can utilize the model to better manage their inventory by categorizing products accurately and understanding trends in customer preferences, leading to more efficient stock management and reduced overstocking or understocking issues.
- **Market Expansion:** The model can help expand the market reach of e-commerce platforms, making them more inclusive and appealing to a broader audience.
- **Automated Tagging and Categorization:** E-commerce platforms can automate the process of tagging and categorizing new products, reducing manual effort and increasing consistency and accuracy in product listings.
- **Visual Search Enhancement:** The model can be integrated into visual search tools, allowing customers to upload images and find similar products more easily, enhancing the shopping experience.
- **Quality Control and Fraud Detection:** The model can assist in quality control by identifying counterfeit or mislabeled products, ensuring that only genuine and accurately described items are listed on the platform.
- **Cultural Representation and Inclusivity:** By accurately classifying traditional garments from different cultures, the model can help e-commerce platforms offer a more diverse range of products, promoting cultural representation and inclusivity.
- **Customer Service Improvement:** Enhanced product classification can aid customer service representatives in quickly identifying products and resolving customer inquiries more efficiently.
- **Trend Analysis and Forecasting:** The model can help retailers analyze fashion trends by categorizing and tracking the popularity of different clothing styles, aiding in market research and strategic planning.

5.4 Future Work

While the current model achieves high accuracy and low loss, there are always areas for improvement and further research:

- **Hyperparameter Tuning:** Fine-tuning the hyperparameters, such as learning rate, batch size, and dropout rates, could improve the model's performance even further.
- **Data Augmentation:** Applying advanced data augmentation techniques could help in making the model more robust and improve its generalization capability. For example, in our study, the number of images generated in the labels was patterned, but to balance the labels completely this number can vary depending on the label percentage in the dataset.
- **Model Architecture:** Experimenting with different architectures, such as deeper networks or different types of layers (e.g., residual blocks), could lead to better performance. Incorporating attention mechanisms could also improve the model's focus on relevant parts of the images.
- **Transfer Learning:** Leveraging pre-trained models on large datasets and fine-tuning them for the specific task may significantly enhance performance and reduce training time.
- **Inclusion of Masculine Clothing:** Expanding the dataset to include a wide range of masculine clothing items will make the model more versatile and applicable to a broader audience, enhancing its utility in the fashion industry.
- **Diverse Nationalities:** Integrating a more diverse range of nationalities in the dataset will help the model to generalize better across different demographic groups, making it more applicable in global markets.

In conclusion, the CNN model demonstrates strong performance in terms of both accuracy and loss. The results validate the effectiveness of the architecture and the training process used in this study. Further improvements and experimentation could yield even better results, enhancing the model's robustness and generalization capabilities. The incorporation of diverse clothing types and broader demographic data will significantly expand the practical applications of this model in the e-commerce industry.

References

C. S. K. Aditya, V. R. S. Nastiti, Q. R. Damayanti, and G. B. Sadewa. Implementation of convolutional neural network method in identifying fashion image. *JUITA: Jurnal Informatika*, 11(2):195–202, 2023.

T. Agins. *The End of Fashion: The Mass Marketing of the Clothing Business*. William Morrow, 1999.

X. Ai et al. Ai-driven innovation in ethnic clothing design: an intersection of machine learning and cultural heritage. *Journal of Cultural Heritage*, 57:123–136, 2023. **DOI** 10.1016/j.culher.2023.123456. URL https://doi.org/10.1016/j.culher.2023.123456.

F. Baalamash et al. Classification of saudi traditional costumes using deep learning technique. *International Design Journal*, 14(2):128–140, 2024.

S. Bruzzi and P. C. Gibson. *Fashion Cultures: Theories, Explorations, and Analysis*. Routledge, 2000.

W.-H. Cheng, S. Song, C.-Y. Chen, S. C. Hidayati, and J. Liu. Fashion meets computer vision: A survey. *ACM Computing Surveys (CSUR)*, 54(4):1–41, 2021.

J. Craik. *The Face of Fashion: Cultural Studies in Fashion*. Routledge, 2009.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

J. Entwistle. *The Fashioned Body: Fashion, Dress and Modern Social Theory*. Polity Press, 2000.

H. Jenkins. *Convergence Culture: Where Old and New Media Collide*. NYU Press, 2006.

Y. Kawamura. *Fashion-ology: An Introduction to Fashion Studies*. Berg Publishers, 2005.

M. Kayed, A. Anter, and H. Mohamed. Classification of garments from fashion mnist dataset using cnn lenet-5 architecture. In *2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pages 238–243. IEEE, 2020. **DOI** 10.1109/ITCE48509.2020.9047736. URL https://doi.org/10.1109/ITCE48509.2020.9047736.

V. Leithardt et al. Classifying garments from fashion-mnist dataset through cnns. *Advances in Science, Technology and Engineering Systems Journal*, 6(1):989–994, 2021. **DOI** 10.25046/aj0601115. URL https://doi.org/10.25046/aj0601115.

F. Li and Y. Du. Introduction: A brief history of deep learning and its applications in power systems. In *Deep Learning for Power System Applications: Case Studies Linking Artificial Intelligence and Power Systems*, pages 1–13. Springer, 2023.

T. Li, J. Chen, L. Ma, and F. Zou. Research on the clothing classification of the she ethnic group in different regions based on fpa-cnn. *Applied Sciences*, 13(17):9676, 2023. **DOI** 10.3390/app13179676.

G. Lipovetsky. *The Empire of Fashion: Dressing Modern Democracy*. Princeton University Press, 1994.

Z. Liu, P. Luo, S. Qiu, X. Wang, and X. Tang. Deepfashion: Powering robust clothes recognition and retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

A. A. Lydia and S. Chandrasekar. A comparative study on regularization techniques in convolutional neural networks.

S. Nandi et al. Bangladeshi saree classification using image analysis. *Journal of Imaging*, 10(1):10, 2024. **DOI** 10.3390/jimaging10010010.

M. T. Nawaz et al. Automatic categorization of traditional clothing using convolutional neural network. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, pages 98–103. IEEE, 2018. **DOI** 10.1109/ICIS.2018.8466467.

J. S. Sandhu and D. Vaishnav. Multi-label geographic classification of apparel.

C. Scharff, A. C. Seck, Kaleemunnisa, and K. Bathula. Senegalese fashion clothing classification system by deep learning and convolutional neural networks. *Electronic Research Archive*, 31(9):5793–5814, 2023. **DOI** 10.3934/era.2023295. URL https://doi.org/10.3934/era.2023295.

J. Shen and M. O. Shafiq. Deep learning convolutional neural networks with dropout-a parallel approach. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 572–577. IEEE, 2018.

D. Swain, K. Pandya, J. Sanghvi, and Y. Manchala. An intelligent fashion object classification using cnn. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 10(4):1–10, 2023. **DOI** 10.4108/eetinis.v10i4.4315. URL https://doi.org/10.4108/eetinis.v10i4.4315.

R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

M. Y. W. Teow. Experimenting deep convolutional visual feature learning using compositional subspace representation and fashion-mnist. In *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pages 1–6. IEEE, 2020. **DOI** 10.1109/IICAIET49801.2020.9257819. URL https://doi.org/10.1109/IICAIET49801.2020.9257819.

H. Xiao. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. **DOI** 10.48550/arXiv.1708.07747. URL https://doi.org/10.48550/arXiv.1708.07747.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1452–1464, 2017.

Appendix A

List of Websites used to create our dataset

Below is a table with all the websites used to create our dataset. It provides a comprehensive list of sources categorized by nationality, ensuring transparency and allowing for verification of the data collection process. This detailed breakdown highlights the diversity of the dataset and the breadth of resources utilized in compiling the information.

Nationality	**Website**
German	https://www.lederhosenstore.com/product-category/women/dirndl/ https://trachtenhimmel.de/dirndl-mini/?p=1 https://trachtenhimmel.de/dirndl-midi/?p=1 https://trachtenhimmel.de/dirndl-lang/?p=1 https://ernstlicht.com/product-category/12-ladies-clothing/97-ladies-dirndls https://www.dirndl.com/en/dirndl/mini-dirndl/ https://www.dirndl.com/en/dirndl/knee-length-dirndl/ https://www.dirndl.com/en/dirndl/full-length-dirndl/ https://www.alpenclassics.co.uk/dirndl/ https://www.chiemseer-dirndl.de/kaufen/Damen/Dirndl/ https://www.trachtenland.de/damen/bekleidung/ https://www.finest-trachten.de/damen/dirndl/
Indian	https://www.panashindia.com/newarrivals?cat=3 https://www.houseofindya.com/women-clothing/pre-stitched-sarees/cat?depth=3&label=lehengas%20&%20sarees-pre-stitched%20sarees&sort=product https://www.fabricoz.com/collections/indian-dresses-online-all https://www.kalkifashion.com/ethnic/organza-sarees.html
Japanese	https://japan-avenue.com/collections/japanese-women-kimono https://shop.japanobjects.com/collections/womens-yukata https://furisode-firstcollection.com/furisode-collection/karankoe

Nationality	Website
	https://furisode-firstcollection.com/furisode-collection/tokyo-retro/ https://furisode-firstcollection.com/furisode-collection/kimono-princess/ https://japan-clothing.com/collections/japanese-women-kimono https://kimurakami.com/collections/japanese-kimono-dress https://kimono-hearts-store.com/collections/neojapanesque https://kimono-hearts-store.com/collections/gothic https://kimono-hearts-store.com/collections/mode-style https://kimono-hearts-store.com/collections/ikikoten https://kimono-hearts-store.com/collections/street-co https://kimono-hearts-store.com/collections/dentoukoten https://kimono-hearts-store.com/collections/maihimekoten https://kimono-hearts-store.com/collections/taisyouromanesuku https://kimono-hearts-store.com/collections/frontpage https://toiki.jp/fs/toiki/c/toiki218 https://toiki.jp/fs/toiki/c/gr509 https://toiki.jp/fs/toiki/GoodsSearchList.html_e_k=%82%60&keyword=1A000454 https://global.kimono-yamato.com/Form/Product/ProductList.aspxshop=0&cat=199001 https://global.kimono-yamato.com/Form/Product/ProductList.aspxshop=0&cat=199002 https://global.kimono-yamato.com/Form/Product/ProductList.aspxshop=0&cat=199002&pgi=&cicon=&dosp=&dpcnt=-1&img=2&max=&min=&sort=07&swrd=&udns=2&fpfl=0&col=&bids=&cols=&pno=2 https://komaya.info/gallery/s_cate=12&s_type=#links https://komaya.info/gallery/s_cate=11&s_type=#links https://www.the-kimonoshop.jp/collections/all https://www.kimonomachi.co.jp/c/kimono/araerukimono https://www.kimonomachi.co.jp/p/search?keyword=%E6%9C%A8%E7%B6%BF&sort=priority https://www.furisodeshop.com/collection/ https://www.furisodeshop.com/collection/page/2/ https://mgos.jp/shop/mimatsu/c/c100102/ https://mgos.jp/shop/mimatsu/c/c100105/ https://mgos.jp/shop/mgos/r/rgr18/

Nationality	Website
	https://mgos.jp/shop/mgos/r/rgr20/
Spanish	https://www.lunaresflamenco.com.br/en/16-flamenco-dresses https://www.vestidosdesevillanas.com/mujer/trajes-colecciones-seora/index.php https://micaelavilla.com/flamenca/ https://sibilinaflamenca.es/categoria-producto/shop/flamenca/ https://laboutiqueflamenca.com/trajes-de-flamenca/ https://saradebenitez.com/moda-exclusiva/moda-flamenca/trajes-flamenca/ https://yolandamodaflamenca.com/trajes-de-flamenca-mujer/ https://carolymodaflamenca.es/categoria-producto/vestidos-flamenca/ https://www.isabelhernandez.es/categoria-producto/colecciones/trajes-de-flamenca/ https://trajesdeflamencalolaylo.com/category/vestidos/ https://www.pilarveratienda.com/modaflamenca http://www.molinaflamenca.es/es/133/outlet-pasarela http://www.molinaflamenca.es/es/86/pasarela http://www.molinaflamenca.es/es/127/noche-en-el-real http://www.molinaflamenca.es/es/126/dia-de-farolillos http://www.molinaflamenca.es/es/125/canasteras-en-el-albero http://www.molinaflamenca.es/es/137/embrujo http://www.molinaflamenca.es/es/136/jaleo http://www.molinaflamenca.es/es/135/querencia https://www.elrocio.fr/24-robes-de-danse https://www.flamencoelrocio.com/41-flamenco-dresses-offers https://www.vivalaferia.es/en/catalog/category/view/go/42 https://www.tamaraflamenco.com/en/flamenco-dresses-in-stock-immediate-shipment–74 https://www.ytutanflamenca.com/50-trajes-de-flamenca https://www.flamenca.com/categoria-producto/trajes-de-flamenca https://mepongoflamenca.com/trajes-de-flamenca/ https://www.mitrajedeflamenca.es/ https://flamencaycomplementos.com/en/category-product/flamenco/flamenco-dresses/ https://www.lolaazahares.com/product-category/nueva-coleccion-minerva-mi-otra-yo/ https://www.sonibel.es/shop/

Appendix B

Some examples for each label present in our Dataset

Below are some examples of images present in our dataset for each label. As we learned, each label corresponds to a country, each with its unique traditional garment. The Dirndl, representing the German label, hails from the German-speaking Alpine regions and comprises a dress, apron, and blouse. The Saree, symbolizing the Indian label, is an iconic and versatile drape measuring between 4.5 to 8 meters in length. The Kimono, representing the Japanese label, is Japan's national dress, known for its "T" shape made from four fabric pieces called tans and secured with an obi belt. And, the Flamenco dress, associated with the Spanish label, is distinctive to Andalusia, Spain, and is recognized for its vibrant colors, ruffles, and fitted waist. These garments are a testament to their respective culture.

Figure B.1: Examples of German images in our dataset

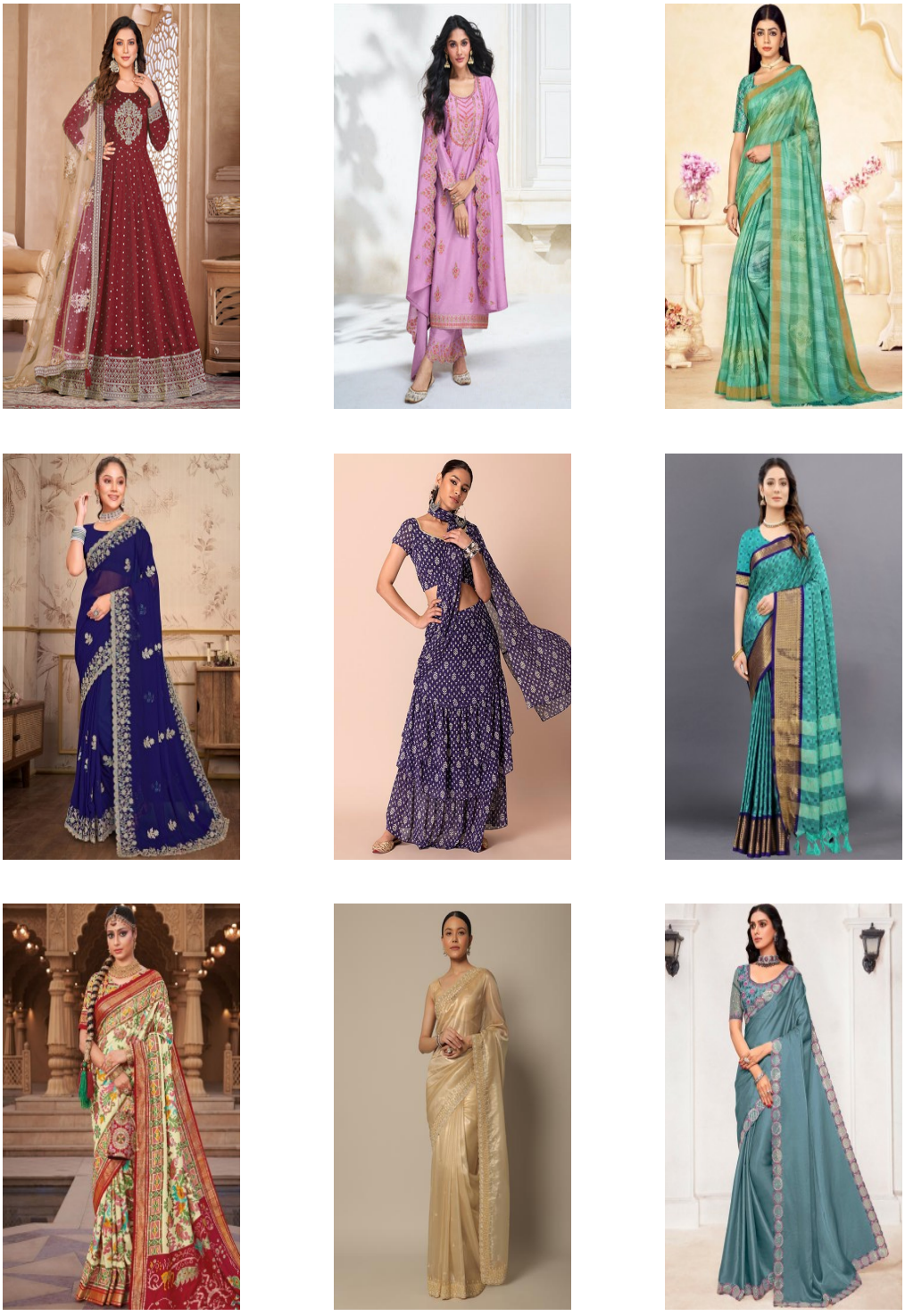


Figure B.2: Examples of Indian images in our dataset

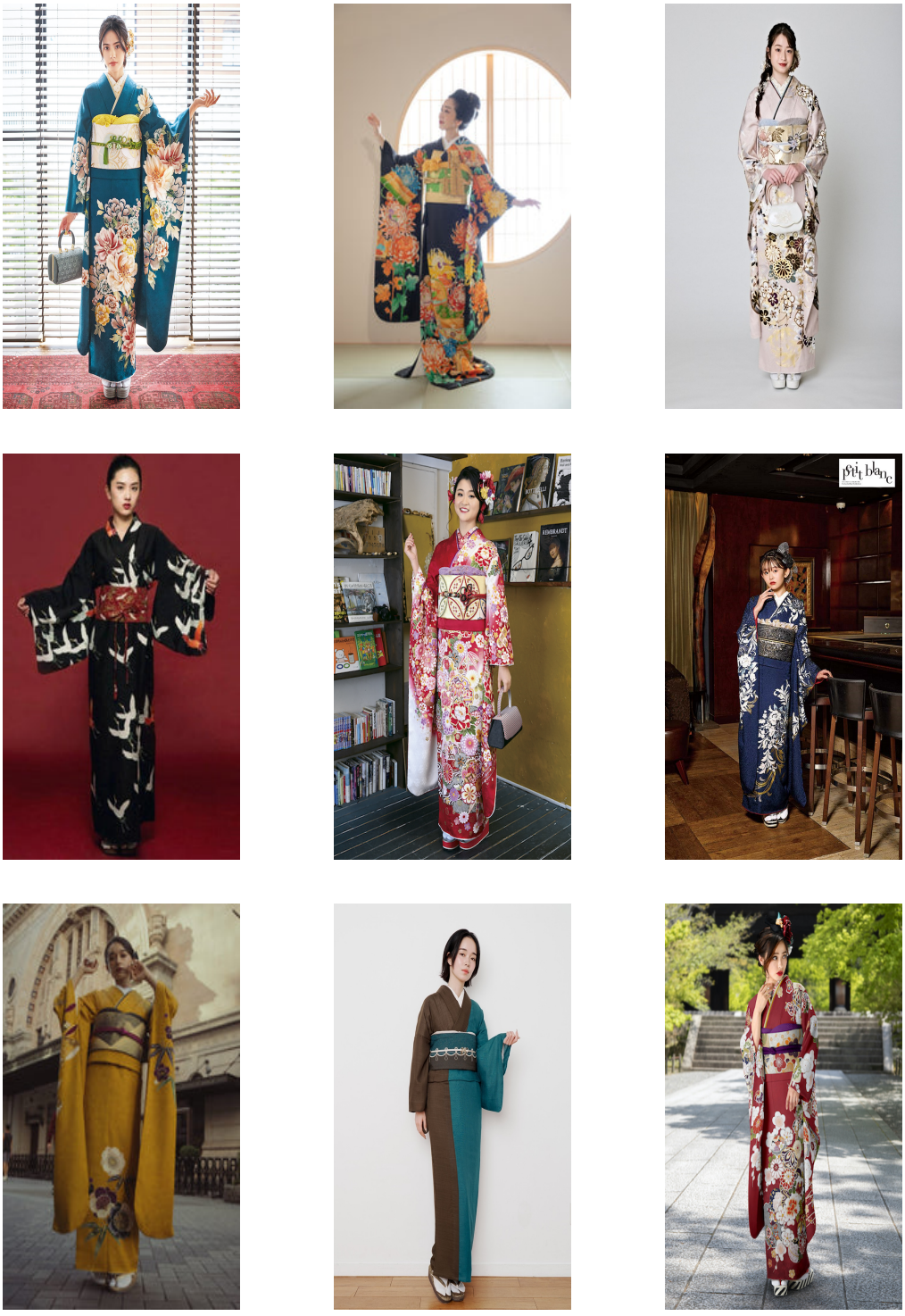


Figure B.3: Examples of Japanese images in our dataset

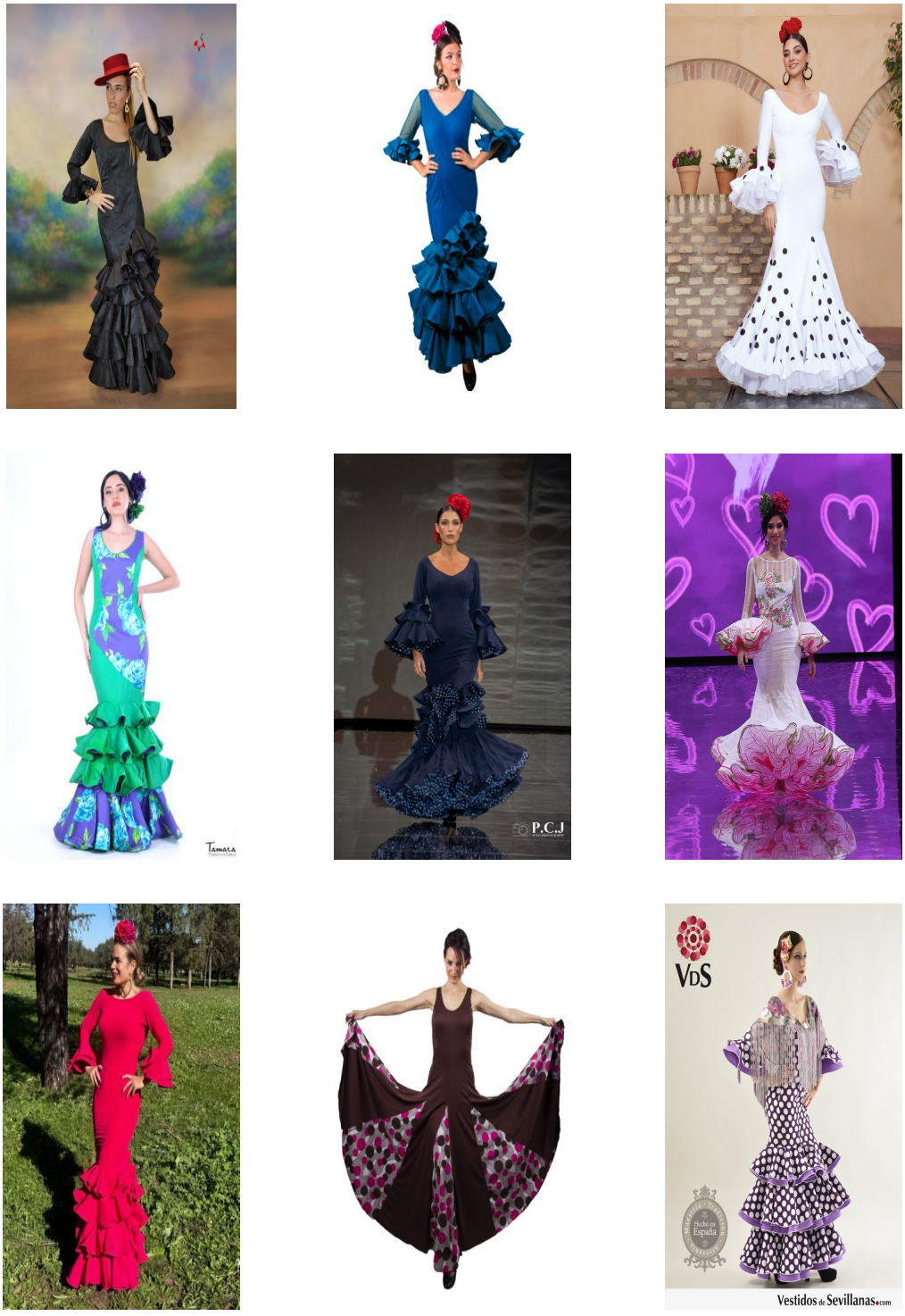


Figure B.4: Examples of Spanish images in our dataset

Appendix C

Important GitHub Links

Here we provide essential resources and repositories that were instrumental in the development and execution of this project. These links offer access to various scripts, datasets, and model versions that can aid in reproducing and extending the research presented in this document. Whether you are looking to download images using Selenium, access dataset manipulation scripts, or explore different versions of the models used, the following resources will be invaluable.

- Selenium images downloader Scripts on GitHub
- Dataset Images Scripts on GitHub
- Models Versions on GitHub

Appendix D

Hugging Face Model

Here we present the latest version of the traditional clothing classification model, which is available on Hugging Face for ease of deployment and experimentation. Hugging Face provides a robust platform for sharing and accessing machine learning models, making it convenient for researchers and developers to collaborate and improve upon existing models.

The model can be accessed through the following link:

- Latest Model version on Hugging Face

This model has been trained to classify traditional clothing from various cultures, and it is designed to be easily integrated into various applications, such as e-commerce platforms, cultural heritage projects, and educational tools. By providing this model on Hugging Face, we aim to facilitate further research and development in the field of traditional clothing classification, encouraging the community to build upon our work.

Additionally, we provide some screenshots showing how the model is organized and how it can be accessed on Hugging Face. These images offer a visual guide to the structure of the model files and the performance metrics available on the platform.

Figure D.1 shows the Hugging Face model files page. This page includes all the necessary files for the model, such as the model architecture, weights, and configuration files. Users can easily download these files and load the model into their own projects, allowing for seamless integration and experimentation.

Figure D.2 displays the model metrics page on Hugging Face. This page provides detailed information about the model's performance, including accuracy, loss, precision, recall, and F1-score. These metrics are crucial for understanding how well the model performs and for identifying areas where further improvements can be made. By sharing these metrics, we aim to promote transparency and enable other researchers to benchmark their models against ours.

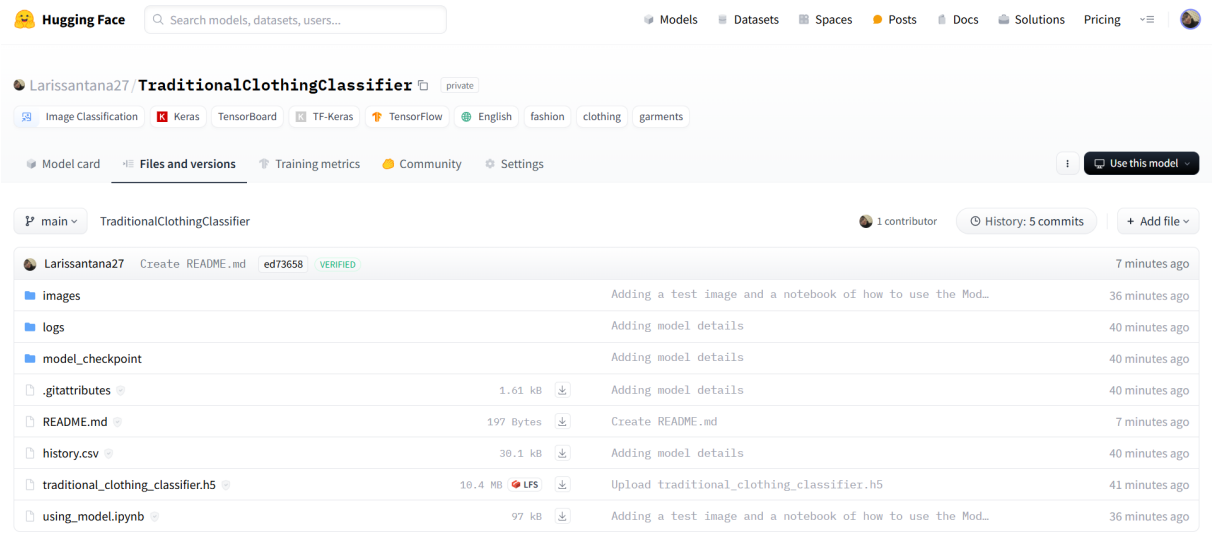


Figure D.1: Hugging Face Model Files Page

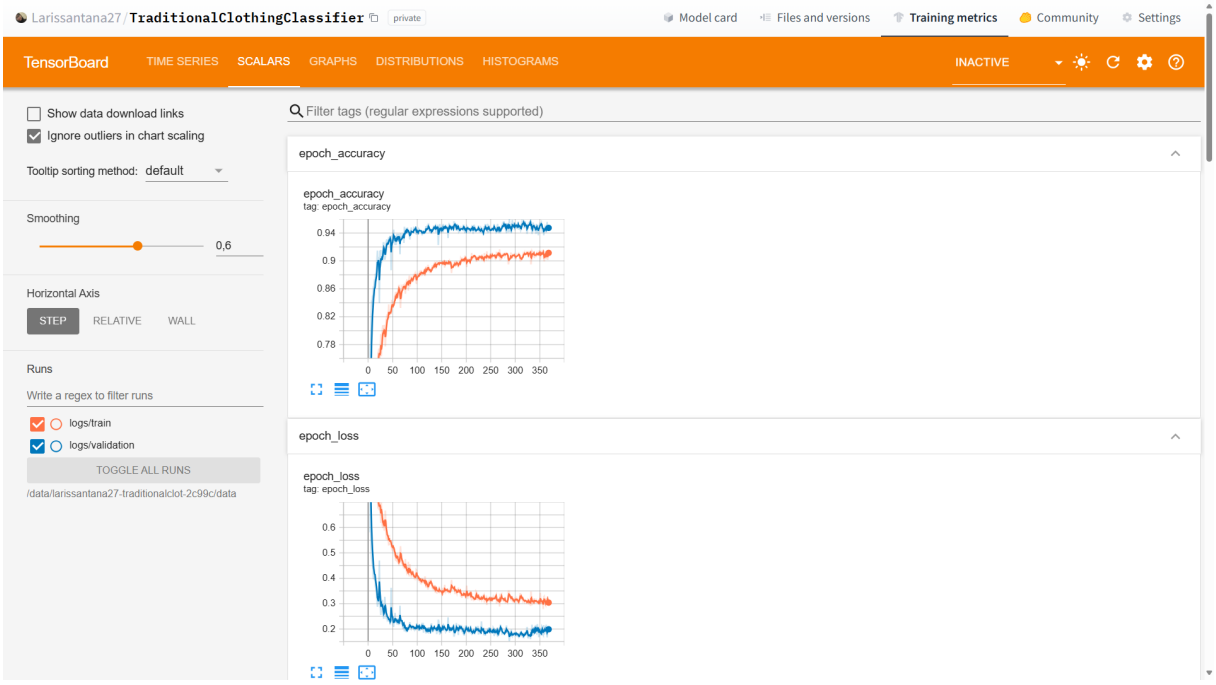


Figure D.2: Hugging Face Model Metrics Page

In conclusion, by making the traditional clothing classification model available on Hugging Face, we provide a valuable resource for the research community and industry practitioners. This model serves as a foundation for further advancements in the field, encouraging collaboration and innovation. We invite researchers, developers, and enthusiasts to explore the model, provide feedback, and contribute to its ongoing development.