AUTOMATIC DEFECT DETECTION SYSTEM FOR FINISHED LEATHERS USING DEEP LEARNING

A MAJOR PROJECT REPORT

Submitted by

K.S.MURALI

V.AKSHAY

M.NAVEEN

Under the Guidance of

Dr. P. Anandan

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS & COMMUNICATION ENGINEERING

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

JUNE 2022

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this major project report entitled **" AUTOMATIC DEFECT DETECTION SYSTEM FOR FINISHED LEATHERS USING DEEP LEARNING "** is the bonafide work of **" K.S. MURALI (18UEEC0173), V AKSHAY (18UEEC0488) and M NAVEEN (18UEEC0265)"**, who carried out the major project work under my supervision.

SUPERVISIOR
Dr.P.ANANDAN
Associate Professor
Department of ECE

HEAD OF THE DEPARTMENT
Dr.P.ESTHER RANI
Professor
Department of ECE

Submitted for Evaluation of Major Project held on:_____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our deepest gratitude to our respected Founder President and Chancellor **Col.Prof.Dr.R. RANGARAJAN**, Foundress President **Dr.SAGUNTHALA RANGARAJAN**, Chairperson and Managing Trustee and Vice President for their immense support in all our endeavors.

We would like to express our gratitude to our Vice Chancellor **Dr.S.SALIVAHANAN**, for his kind cooperation and encouragement..

We are obligated to our beloved registrar **Dr.E.KANNAN**, for providing immense support in all our endeavors.

We are thankful to our esteemed Dean Academics **Dr.M.J.CARMEL MARY BELINDA (I/C)**, for providing a wonderful environment to complete our Project work successfully

We are extremely thankful and pay our gratitude to our Dean SoEC **Dr.V.JAYASANKAR**, for his valuable guidance and support on completion of this major Project.

It is a great pleasure for us to acknowledge the assistance and contribution of our Head of the Department **Dr.P.ESTHER RANI**, Professor for her useful suggestions, which helped us in completing the work in time and we thank her being instrumental in the completion of third year with her encouragement and unwavering support during the entire course.

We extremely thankful and pay our gratitude to our supervisor **Dr.P.ANANDAN**,Associate Professor for his valuable guidance and support on completion of this major project.

We are thankful to our department faculty, supporting staffs for their help and guidance to complete this major project.

K.S.MURALI

M.NAVEEN

V.AKSHAY

TABLE OF CONTENTS

ABSTRACT

Leather is a very important raw material in many manufacturing industries. India is the second largest exporter of leather garments, 4th largest exporter of Leather Goods in the world. India is the world's subsequent major exporter of leather goods and the fourth largest exporter of leather goods in the world. Leather is a natural and robust substantial material made by tanning animal skins and skins. The price of leather is subjective as it is very sensitive to the quality of the leather and the condition of surface imperfections. Before the mass manufacture of certain goods, there was little work to detect leather defects. But it's annoying because it's labor-intensive, time-consuming, eye strain, and often prone to human error. A manual error checking process is essential in the leather manufacturing industry as a quality control step. This paper proposes a fully automatic defect detection system that uses a convolutional neural network on the finished leather to provide defect-free leather. The camera then accumulates a picture of the leather, trains and tests the image using a deep learning architecture. The histogram gradient preprocessing has been introduced to improve the quality of the leather image to maintain the display of important features from a perceptibility viewpoint. After that convolutional neural network classifiers are used to distinguish Between a defective leather area and a non- defective leather area. The proposed method can generate a significant accuracy rate from the 1100 leather patch sample collection.

KEY WORDS: Convolutional Neural Network (CNN), Defect detection, Histogram Gradient, Leather patch

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
SVM	Support vector machine
PIA	Pixel Intensity Analyzer
EKM	Extreme Learning Machine
$SIFT$	Scale Invariant and have Transformation
$RLeU$	Rectified Linear Unit
HOG	Histogram Gradient
ROC	Receiver Operating Characteristic
$GLCM$	Grayscale Co-occurrence. Matrix

CHAPTER 1

INTRODUCTION

Indian leather and leather products have a standing place in India's economy. India's revolutionary community occupies about 13worldwide leather production and handles about 3 billion square leather production. It is important to ensure high quality of leather to improve customer satisfaction due to the importance of leather in the industrial business. Raw substantial excellence is usually negotiated by ante-mortem defects (scratches, rubs, horn rakes, yoke marks, scabies, smallpox, burn marks, etc.) or post-mortem defects (flame cuts, meat cuts, grain cracks). Leather quality plays an important role in the amount of the leather manufactured, accounting for about 55-75raw materials are the most valuable and important factor of production.

In short, the basic steps for transforming the leather skins are: (1) Immersion: Dirt and pickle salt from soaking rawhide in water, for example Hours to days; (2) Emerald: Cuticle, hair and subcutaneous substances; (3) Browning: Creates protein crosslinks in collagen by penetrating Chemicals in the skin; (4) Drying: To get rid of excess Water; (5) Coloring: Create the desired convention color. Few of the defects in nature were formerly inconspicuous. The tanning process and they seem to be gradually becoming visible Under leather finishing. On the additional hand, fault areas with slight damage are repaired, roughened with a filler to create a smooth and unvarying surface. Lastly, the completed portion of leather is pre-classified before Shipping to customers. The grading process is one of the most important and difficult steps as it involves a manual evaluation to visually inspect defective parts of the leather.

This decade has seen astounding progress in the application of intelligent systems to real-world problems in areas including but not limited to medicine, telecommunications, finance, medical diagnosis, transportation, information retrieval, energy and many more. The urge for automation has revolutionized the industry sector with expert and intelligent systems finding applications in almost all kinds of industrial processing, ranging from resource optimization to industrial inspection. Intelligent machine vision systems have been at the heart of industrial inspection and surveillance for the past two decades. Image analysis based methods proposed for industrial inspection include both heuristic and machine learning methods. Despite being an important subject in industrial inspection, leather

defect inspection has not received much attention yet. The majority of methods that exist for visual defect inspection of leather are heuristic with only limited studies that explore machine learning options for robust performance. Leather quality grading based on defect inspection is an important area of research and rapid advancements in intelligent systems for automatic leather grading are expected in the near future. Advancements in expert deep learning based systems and their ability to surpass human performance has increased their application in numerous fields including healthcare, automotive industry, telecommunications, industrial visual inspection and many more during recent years. Most convolutional neural networks (CNN) based methods proposed for various computer vision applications can be categorized by one of the following tasks: image classification, detection, semantic segmentation and multiclass classification. Owing to their success in the aforementioned tasks and their ability to match or surpass human level performance, we present a complete deep networks based machine vision pipeline for visual defect grading, which can be utilized by future researchers as a guideline. Figure 1 shows the recommended pipeline for leather defect inspection. In the first stage defect detection and classification is performed in parallel with defect segmentation. Based on the information from the detection, classification and segmentation; important features for grading including shape, texture, location and context are computed using an image analysis module. This in-depth analysis of the leather defect's characteristics is then fed to a multiclass classifier as a feature matrix to obtain the final leather quality grade. We expand on this pipeline in Section V, discuss and recommend CNN architectures that might be worth investigating for leather defect detection, classification, segmentation and multiclass classification (grading). These guidelines would help researchers in this field to investigate machine vision systems for leather defect detection.

Leather craft materials usually come from cow, crocodile, lizard, buffalo, goat, sheep, and stingray skin. The animal skin will go through tanning process before it can be used as crafting materials. In every level of tanning process, tanning agent will alter the physical properties and chemical compositions of the skin. The skin will become durable, pliant, and may have different color and texture from the original. Therefore, after tanning process, animal skins will have a large variety of color and texture within the same skin category and have high similarity with other skin categories. This make them difficult to be distinguished. Furthermore, skin from different part of animal body may have different color and texture. The result of classification determines the grade of tanned leather. The grade of leather will affect the price of leather. Therefore, classification procedure is the most important in automation system of tanning leather production because it is directly affects the price of final tanning leather products. Furthermore, a high return rate and disputes between customer and manufacturing industry which caused by failure in classification of leather usually cause additional costs [4]. From leather craftsmen point of view, the correct result of leather grading is important because it will be used as consideration to determine the type of leather craft product that will be made. Mistakes in determining the type of leather for leather craft products can make the resulting leather craft products become unfavorable and impact on loss in sales. Leather craftsmen in some area do not yet have the knowledge about feasible standard of leather crafting. The type and quality of leather that will be used as leather craft materials are known based on experience and tradition which inherited from generation to generation.

In specific, defect type, defect size, and severity are the most important aspects of quality control. Examiners are required to accomplish numerous detailed manual assessments of the same piece of leather from different angles, distances and lighting conditions to ensure accuracy and integrity. However, keep in mind that each decision is subjective, as it differs significantly from individual to individual. Therefore, human testing is costly, time consuming, inefficient and unreliable. Due to this boring and tiresome task, or when the operator feels stressed and completes the job in a rush, human error can be more expected to occur. Therefore, it is important to develop an automatic leather defect inspection system to improve the grading and inspection process while saving needless costs. The final goal of this paper is to detect the defects in finished leather samples.

The ultimate goal of this paper is to classify leather Samples into either a defective class or a non-defective class. The four main contributions to this research work In summary: (1) Preparing the image dataset by image acquisition. (2) Then proposed preprocessing step and Convolutional Neural Network Classifier for classifying defective leather patches (3) Wide-ranging experimental evaluation and relative analysis have been conducted on more than 1100 leather samples. (4) Demonstration of capable classification outcomes By reporting all performance metrics.

CHAPTER 2

LITERATURE SURVEY

2.1 PRELIMINARY WORK

Zhengyu He One of the most effective models for deep learning is the Convolutional Neural Network (CNN).The CNN comprises two unique types of layers, pooling layers and convolution. Layers of CNN contain well-designed filters to handle with input data. They convolve the range of input values, and finally get smaller range of them. And then, CNN can detect essential or specific features within the range we acquired before. The CNN generally consists of the input layer, convolution layer and Rectified Linear Unit. Rectified Linear Unit (ReLU) is mathematically expressed as Max(0, x), Max pooling and the final output layer. Convolution layer can produce a matrix of smaller dimension than input matrix, and max pooling can transmit the maximum value from amongst a rather small batch of data of the input matrix to the output. The output layer is fully connected, and based on the activation function.

Peng Gong* and Philip f. Howarth A contextual method for land-use classification has been developed and evaluated using the SPOT HRV XS data obtained over the rural-urban fringe of northeastern Metropolitan Toronto. It involves two steps: gray-level vector reduction and frequency-based classification. A new algorithm for gray-level vector reduction was developed and illustrated. The technical basis of the frequency-based contextual classifier and its advantages over commonly used contextual classification methods, such as those based on spatial feature extraction, have been introduced and discussed. A new criterion, average separability, for pixel window size selection was developed. The Kappa coefficient, the estimated variance of a Kappa value, and the conditional Kappa coefficient were used in the accuracy assessment of land-use classification results.

Kontoes et al. This paper describes a knowledge-based system which has been developed for integrating easily-available geographical context information from a GIS in remotely-sensed image analysis. An experiment is described in which soil maps and buffered road networks have been used as additional data layers for classifying single date SPOT images for estimates of crop acreages. The map datasets have been digitised, co-registered to the satellite imagery, and manipulated using ARC/INFO. The knowledge base consists of both image context rules and geographical context rules. Probabilistic

information from the image classifier and from the rule base is combined using the Dempster-Shafer model of evidential reasoning.

Stuckens et al. A hybrid segmentation procedure to integrate contextual information with per-pixel classification in a metropolitan area land cover classification project is described and evaluated. It is presented as a flexible tool within a commercially available image processing environment, allowing components to be adapted or replaced according to the user's needs, the image type, and the availability of state-of-the-art algorithms. In the case of the Twin Cities metropolitan area of Minnesota, the combination of the Shen and Castan edge detection operator with iterative centroid linkage region growing/merging based on Student's t-tests proved optimal when compared to other more common contextual approaches, such as majority filtering and the Extraction and Classification of Homogeneous Objects classifier.

Single image dehazing with a generic model-Agnostic Convolutional neural network the author used the technique called Divisions based on haze density, local atmospheric light density and transmission map estimation using iterate algorithm. The advantages in this paper are It over comes some of the common pitfalls of state-of-the-art methods and It has the property of color darkening and excessive edge sharpening where as the limitations are It only represents a progressive move towards developing a universal image restoration method.

For defect locations and segmentation tasks One of the frontier studies is carried out by **Lovergine et al. [6]**. They recognize and identify defective areas With a black and white CCD camera. The morphological segmentation process [16, 17] is then applied to the collected images to extract texture orientation features. leather. A sample defective leather image is shown in Figure 1. Paper, but quantitative methods and numbers There is not enough data to evaluate the proposed algorithm. The purpose of this study is to introduce an automatic defect detection system for segmenting defect areas of leather. Certain types of defects, namely tick bites. Such defects appear as small surface damage on animal skin. Instance segmentation Using a deep learning model, a convolutional neural network (CNN), to develop a robust architecture for evaluating test datasets.

To date, the literature that carried out the automatic classification or segmentation tasks on the leather pieces is yet limited [1–3]. Besides, the experimental data are varied and hence it is difficult to make a fair test of performance to verify the effectiveness of the proposed methods. For instance, reference [7] collects the leather patch dataset using a robot arm such that each image is captured under consistent lighting source, same viewing angle, and distance. In total, the dataset contains 584 images. .en, a series of procedures are introduced to localize the tick-bite defects on leather patches. Succinctly, a segmentation algorithm, namely, Mask Region-based Convolutional Neural Network (Mask R-CNN) is adopted to learn the local features from 84 defective images. As a result, a classification accuracy of 70 is obtained when evaluated on 500 testing images.

Later, reference [8] employs the same data elicitation process to collect a different piece of calf leather. In brief, 27 images are collected and each piece is partitioned into 24 small patches. .us, in total, 648 images are used in the experiment. Different from reference [4, 5] conducts both the classification and segmentation processes to predict two types of defects, namely, black lines and wrinkle. A transfer learning technique is adopted to fine-tune the parameters in AlexNet architecture for the classification task, whereas UNet architecture is employed for the segmentation task. As a result, the classification performance attained is 95the segmentation task obtained an Intersection over Union rate of close to 100. However, it should be noted that the black line and wrinkle defects are relatively obvious and occupy a larger region. .us, a reasonably higher classification result can be achieved. Reference [6] designs a statistical approach based on the image intensity to tackle the classification task for both the datasets released by references . Briefly, this work adopts simple statistical features operations such as mean, variance, variance, skewness, kurtosis, lower, and upper quartile values. .en a feature selection method of the 2-sample Kolmogorov–Smirnov test is exploited to determine meaningful features. .en, three methods are applied to eliminate redundant features: percentile thresholding, Gaussian mixture model (GMM), and K-means clustering. Finally, seven types of classifiers are adopted to differentiate between the defective and nondefective leather patches. .e best classification accuracy generated are 99 and 77 on two different datasets (i.e., [4, 5]), respectively. In short, this paper successfully outperforms reference [4] by 7 while obtaining a comparable performance with reference [5].

On the other hand, conventional methods such as feature extraction and reduction are adopted for leather defect detection task in which deep learning methods are not applied. For example, reference [7] utilizes the FisherFace feature reduction technique to project the local features of the leather images from high-dimensional image space to a lower-dimensional feature space in order to effectively distinguish the targeted classes. Concisely, the feature size of each image sample has been reduced from 4202 to 160. .e extracted features include the attributes of color details, histograms of the color, co-occurrence matrix, Gabor filters, and the original pixels. To validate the effectiveness of the proposed method, the experiment was tested on 2000 samples that are composed of seven defective classes. .en, three types of classifiers are employed to predict the defective type. .e best classification accuracies obtained are 88for wet blue and 92 for rawhide images. On the other hand, a leather type classification task was performed by reference [8] that evaluated 1000 leather sample images to differentiate among monitor lizard, crocodile, sheep, goat, and cow. Despite each leather type may contain samples with different colors, the proposed method is capable to distinguish the texture and characteristics of each leather type. .us, a 99.9 classification accuracy was achieved by adopting the pretrained AlexNet architecture. However, no defect inspection or defect classification task is involved in the experiment.

Based on the aforementioned discussion, the research works conducted thus far are manageably finite. Inspired by 2 Mathematical Problems in Engineering reference [4, 6], this paper aims to enhance the classification performance by introducing a simple yet effective solution. Particularly, the type of defect class in this classification task is strictly limited to only the tick bite. In brief, six preprocessing steps are applied to improve the images and to extract the local information of the

leather patches. Next, the feature sets are fed into several two-class classifier models independently by exposing the relationships between the encoded features, in order to generate corresponding predicted labels. .e classifiers involved in the experiment herein include decision tree, SVM, k-NN, Artificial Neural Network (ANN), XBoost ANN, and others which are employed to categorize the testing data. Kwon et al. propose a framework to identify several types of defects (i.e. pin hole, hole, wrinkle and scratch) based on the pixel intensity value histogram [20]. They found that the composition of the image pixels of non-defective leather was intended to represent standard normal distributions. In the case of hole defects, their Gaussian distribution for image pixels is usually concentrated at the brighter part (i.e. close to the pixel value of 255). On the other hand, the pin holes have much darker pixels (i.e., close to the pixel value of 0). Defects such as scratches and wrinkles usually have distinct patterns compared to the normal distribution. The grade of the leather (i.e., A, B or C) would then be determined on the basis of the result of the analysis, which refers to the density and number of defects extracted.

Villar et.al. also introduced an automated computer vision system to detect a few types of defect (e.g., closed cut, fly bite and open cut) [21]. Seven popular feature descriptors (i.e. Gabor features, contrast characteristics, local binary patterns, Hu moments with intensity information, Fourier and Cosine transform, Haralick descriptors and first order statistics) and a selection method are used to dynamically reduce the size of the feature. A multilayer perceptron neural classifier is then adopted to categorize the type of defect. Note that the training and testing datasets consisted of a total of approximately 1,800 sample images with a 40 40 spatial resolution. Similar defect categorization work is carried out by Pistori et al. [22]. In particular, they distinguish four types of defects: cuts, tick marks, brand marks made of hot iron and scabies, both raw hide and wet blue leather. The former has a more complex exterior with different surface types (i.e. textures, colors, shapes, thicknesses and even some serious defects), while the latter is a common type of leather that has undergone a tanning process that appears to be noticeable for both human and machine visual inspection. The features of the images are extracted using a popular technique of texture analysis, namely the Gray-scale Coocurrence Matrix (GLCM) [23], [24]. The proposed method is validated on a pre-built data set consisting of images of 258 pieces of raw hide and wet blue leather with 17 different types of defect.

VOLUME 8, 2020 198601 **M. Aslam et al**.: Ensemble Convolutional Neural Networks With Knowledge Transfer Recent research by Winiarti et al. classifies five types of leather by using both types of feature extractors: handcrafted feature descriptors and deep learning architecture [25]. Leather types include lizard monitors, crocodiles, sheep, goats and cows. For handcrafted representation, a combination of statistical color features (i.e. mean, standard deviation, skewness and curtosis) and statistical texture features (i.e. contrast, energy, correlation, homogeneity and entropy) is used. Pre-trained AlexNet is being exploited as a deep learning framework. Classification performance indicates that the deep learning method can better capture the leather characteristics. One important to note that all data images are non-defective images as there is no defect classification. Lovergine et al. has one of the pioneering research projects for fault location and segmentation tasks [1]. They detect and identify defective areas using a black and white CCD camera. A morphological segmen-

tation process is then applied to the collected images to extract the texture orientation features of the leather. Some of the qualitative results are shown in the paper, but there are no quantitative and numerical methods for evaluating the proposed algorithm.

Sobral et al. proposed a wavelet-based method using optimized filter banks, filters are adjusted according to defects shape [4]. Both filters and wavelet ranges are determined by optimising leather fault attributes. On the other hand, a sliding patch window is employed to obtain more sample images in our proposed framework. The sliding window is the window cropping the collected ultra-high definition leather sample image with a specific pixel size according to certain moving rules. The defects of the leather are inside the small regions on the leather sample images. A too small sliding patch window cannot fully contain the defect area, which will make the corresponding defect feature extraction inaccurate in the network training. On the contrary, a too large sliding patch window may weaken the effectiveness of the extracted defect features. Thus, how to determine the appropriate size of the sliding patch window is a challenge issue. In order to solve this problem, the comparative computer numerical simulations of different sizes of the sliding patch windows are conducted. Based on the computer numerical simulation results, the final size of the sliding patch window is determined by the least squares method.This method can detect defects, even if the attributes are slightly modified [5]. Furthermore, this technique has been shown to be fast enough to be detected in real time. He et al. developed an automatic band selection scheme based that is based on wavelet energy coefficients and wavelet transformation, distributed across different frequency channels

Jawahar et al. has provided a feature extraction method for the identification of leather defects using a wavelet feature extraction [7]. Wavelet Co-occurrence Feature (WCF) and Wavelet Statistical Feature (WSF) were used as a feature extractor. SVM was used as a classifier. They claimed good results when both the WCF and the WSF were combined. As SVM only works well with the binary classification, their system could only discriminate between defective and non-defective leather.

On the other hand, **Branca et al**. proposed a leatherdefect detection method using a Gaussian filter and its oriented texture. A Gaussian convolution is first applied on the leather image which serves as the smoothing operator to reduce the image noise. Then, the orientation flow field details are computed to derive the local gradient and its corresponding length for each neighborhood pixel. A neural network is trained by treating the orientation vector field as the input and representing them as sets of projection coefficients. Finally, the defects will be classified depending on these coefficients. The paper illustrates the results of nine samples which highlight the segmented areas as the defective regions. However, there are no performance metrics involved to quantify and verify the correctness of the proposed algorithm.

Amorim et al. [3] demonstrated several attributes reduction methods to perform leather defect classification. They utilize several FisherFace feature reduction techniques to represent the details of the leather images by projecting the attributes vectors onto a subspace. The initial length of the features is 4202 per sample, which include the attributes of color details (hue, saturation, brightness, red, green and blue), histograms of the color, co-occurrence matrix, Gabor filters and the

original pixels. After applying the discriminant analysis techniques, the feature length is reduced to 160 per sample. They tested on 2000 samples that contain eight classes, namely background, no-defect, hot-iron marks, ticks, open cuts, closed cuts, scabies and botfly larvae. The classifier types include C4.5, k-Nearest Neighbors (kNN), Naive Bayes and Support Vector Machine (SVM). The highest classification accuracy obtained is 88percent for wet blue images and 92percent for raw hide images.

A defect classification process on goat leather samples was carried out in [4]. A combination of features from Gray level Co-Occurrence Matrix (GLCM), Local Binary Pattern (LBP) and Pixel Intensity Analyzer (PIA) are formed. Then, classifiers such as kNN, MLM (Minimal Learning Machine), ELM (Extreme Learning Machine), and SVM are adopted independently and tested on the features extracted. The dataset collected comprises of 1874 samples with 11 classes (normal, wire risk, poor conservation, sign, bladder, scabies, mosquito bite, scar, rufa, vegetable fat and hole). The ground-truth defects are annotated by two leather classifier specialists who have undergone one month's training. As a result, an accuracy of 89 percent is exhibited when using the LBP feature descriptor on the SVM classifier. There are very few works in the literature exploiting deep learning techniques in analyzing the leather types.

One of the recent works that applied a pre-trained Convolutional Neural Network (CNN) was conducted by Winiarti et al. [5]. Using approximately 3000 leather sample images from the five types of leather (i.e., monitor lizard, crocodile, sheep, goat, and cow skin), they achieved a 99.9percent classification accuracy. Succinctly, they performed transfer learning process using AlexNet on 1000 leather images, with 200 leather images for each category. The paper also illustrates that there might be differences in the texture within the same category. Thus, it shows that deep neural networks perform well in leather classification tasks.

In a recent work on automated defect segmentation, Liong et al. [6] introduced a series of algorithms to predict tick-bite defects on leather samples . Different from the conventional methods that capture the leather sample manually, [6] elicits all the image data using a robot arm and draws the defect region with a chalk using the same robot arm. One of the state-of-the-art models for instance segmentation, namely, Mask Region-based Convolutional Neural Network (Mask R-CNN) is employed and fine tuned with 84 defective images to learn the local features of the leather samples. The proposed method exhibits an accuracy of 70 percent on 500 testing images. It should be noted that this segmentation task differs from the classification task, where the former localizes the defect region and the latter distinguishes the type of the defect on a leather sample.

2.2 COMPARISION

Here is the discussion of some of the techniques and about there description by some authors as shown in Table 2.1.

Table 2.1: Comparision Table of Literature Surveys

S. No.	Title of the paper	Authors	Year and Volume	Journal Name	Observation with performance metrics
1	Automated Classification System for Tick-Bite Defect on Leather	Y.S.Gan,Wei-Chuen Yau,Sze-Teng Liong,and Chih-Cheng Chen	2022 and volume 22	Mathematical Problems in Engineering	Classification accuracy obtained is 94 by using ANN classifiers this experiment strictly limited to the defect type of tick-bite defects.
2	Leather defect classification and segmentation using deep learning architecture	Sze-Teng Liong,Danna Zheng,Yen-Chang Huang, Y.S.Gan	2020 and volume 33	International Journal of Computer Integrated Manufacturing	There are three types of experimental configurations for the train/ test data distribution in classifying the defective images. By employing AlexNet, the accuracy and F1- score results obtained
3	On the Application of Automated Machine Vision for Leather Defect Inspection and Grading: A Survey	MASOOD ASLAM,TARIQ M.KHAN,SYED SAUD NAQV,GEOFF HOLMES,AND RAFEA NAFFa	2019 and volume 7	IEEE ACCESS	A detailed review of the role of deep learning methods in general visual inspection applications was presented, where recent CNN architectures are classified and compared.
4	A Novel Framework for Classifying Leather Surface Defects Based on a Parameter Optimized Residual Network	JIEHANG DENG, JIAXIN LIU,CHANGZHENG WU,TAO ZHONG, GUOSHENG GU, AND BINGO WING-KUEN LING	2020 and volume 8	IEEE ACCESS	The proposed framework first obtains the UHD leather images through the UHD leather imaging system.

Table 2.2: Comparision Table of Literature Surveys

S. No.	Title of the paper	Authors	Year and Volume	Journal Name	Observation with performance metrics
5	Ensemble Convolutional Neural Networks With Knowledge Transfer for Leather Defect Classification in Industrial Settings	MASOOD ASLAM, TARIQ M.KHAN,SYED SAUD NAQVI,GEOFF HOLMES,AND RAFEA NAFFA	2020 and volume 8	IEEE Access	EfficientNet-B3 and ResNext-101 stand out as compared with other architectures in terms of their performance on leather defect classification. Therefore, when performing our ensembling experiments, we used pre-trained architectures on ImageNet
6	Image-Based Surface Defect Detection Using Deep Learning	Prahar M. Bhatt, Rishi K. Malhan,Pradeep Rajendran,Shantanu Thakar	2021 and volume 21	Journal of Computing and Information Science in Engineering	It helps us to understand the different types of defect detection problems. We divide this section into four sections, (1)anomaly detection,(2)targeted defect detection, (3)concurrent identification of multiple defects, and (4)defect type clustering
7	Integrated Neural Network and Machine Vision Approach For Leather Defect Classification	Sze-Teng Liong, Y.S.Gan,Yen-Chang Huangb,Kun-Hong Liuc, , Wei-Chuen Yaud	2019 and volum 6	ARXIV	Proposal of ANN-learned and handcrafted features extractors independently on each leather sample patch.Demonstration of the scalability of the proposed algorithm by evaluating them on a variety of machine learning classifiers

Table 2.3: Comparision Table of Literature Surveys

S. No.	Title of the paper	Authors	Year and Volume	Journal Name	Observation with performance metrics
8	Automatic Defect Segmentation on Leather with Deep Learning	Sze-Teng Liong , Y.S. Gan , Yen-Chang Huang , Chang-Ann Yuan , Hsiu-Chi Changa	2019 and volume 18	ARXIV	automatic approach to inspect the defect instances of a leather, yielding promising segmentation accuracies of 91.50% and 70.35% on the train and test datasets, respectively.
9	Skin Lesion Classification Using Ensembles of Multi-Resolution EfficientNets with Meta Data	Nils Gessert, Maximilian Nielsen, Mohsin Shaikh, Rene Werner, and Alexander Schlaefer	2019 and volume 19	ARXIV	We overcome the typical problem of severe class imbalance for skin lesion classification with a loss balancing approach. To deal with multiple image resolutions, we employ various EfficientNets.
10	Deep Residual Learning for Image Recognition	Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun	2016 and volume 9	international journal of computer integrated manufacturing	Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

Table 2.4: Comparision Table of Literature Surveys

S. No.	Title of the paper	Authors	Year and Volume	Journal Name	Observation with performance metrics
11	Object Detection and Classification of Metal Polishing Shaft Surface Defects Based on Convolutional Neural Network Deep Learning	Qingsheng Jiang , Dapeng Tan , Yanbiao Li, Shiming Ji , Chaopeng Cai and Qiming Zheng	2019	MDPI	The Faster-R-CNN object detection framework with the ResNet101 CNN feature extraction algorithm can be applied to the industrial practical environments
12	Vision based inspection system for leather surface defect detection using fast convergence particle swarm optimization ensemble classifier approach	Malathy Jawahar, N. K. Chandra Babu, K. Vani L. Jani Anbarasi, S. Geeth.	2020 & volume 80	SPRINGER	Most prominent features for automatic leather defects were selected using correlation coefficient and test. Extracted features were given as input to different supervised classifiers namely Neural Network (NN), Decision Tree (DT), SVM, Naïve Bayes, KNN and Random Forest (RF) were employed to classify defective and normal regions of the leather images.

CHAPTER 3

METHODOLOGY

3.1 EXISTING METHOD

One of the early works is proposed by Georgieva et al.(2003) that employed criteria to analyze the leather surfaces. The author compares defective areas on the leather surface to an averaged histogram of non-defect leather samples. The application of the x criteria is then used for distance computing in relation to the averaged histogram.However, the author did not provide any concrete evidence of the efectiveness of this approach.He et al. (2006) outline a method to detect leather defects using wavelet band selection procedure. The authors aim to eliminate all regular and repetitive texture patterns by decreasing the resolution levels of the leather images until the texture patterns are undetectable. The selection of decomposition levels can be realized by adopting the wavelet band selection procedure. This is conducted by using the energy function and lowering the resolution until the energy ratio (ratio of detail between resolution levels) is minimum. Besides, this paper claims that this method can achieve the same detection percentage as a trained professional and is capable to implement in real-time inspection. However, no quantitative results were given and reported in the paper.

3.2 PROPOSED METHOD

The proposed automatic defect detection system included four steps.

3.1) Image acquisition.

3.2) Preprocessing procedure.

3.3) Classification using Convolutional Neural Network Architecture.

3.4) Calculating Performance metrics.

The figure 4 shows an overview of the process. In short, the image is first sent to a preprocessing step which is Histogram gradient. The classification task, on the other hand, uses state-of-the-art supervised classifiers such as Convolutional Neural Network (CNN). The details of the preprocessing method and the mathematical derivation of the classifier above are described in detail in Section 3.2 and Section 3.3, respectively.

3.2.1 Image acquisition:

We have collected 1100 defective and non-defective leather samples and prepared a dataset.

3.2.2 Preprocessing procedure:

Figure 3 shows the preprocessing technique used in the project. Each step is described as follows. An image is shown in Figure 3 to illustrate the effect of the Preprocessing step.
Histogram gradient:
Histograms of oriented gradients, also referred to as HOGs, are feature descriptors like Canny Edge Detector, SIFT (Scale Invariant and have Transformation). it's employed in computer vision and image processing for beholding purposes. This system counts the occurrence of gradient directions within the localized part of the image. This method is extremely similar to the sting Histogram and Scale Invariant Feature Transformation (SIFT).

Figure 3.1: Preprocessing example

The HOG signifier emphases on the construction or outline of the object and it is superior to any edge descriptor because it uses both gradient magnitude and angle to calculate features. For areas of the image, use the magnitude and direction of the gradient to get a histogram Procedure for calculating HOG features. 1.Gets the input image that calculates the HOG feature.Resize the image to an image of 4096*256 pixels.

$$G_x(r,c) = I(r,c+1) - I(r,c-1)$$

$$G_y(r,c) = I(r-1,c) - I(r+1,c)$$

2. The gradient of the image is calculated. Gradients are obtained by combining sizes and angles from the image. Considering a block of 3x3 pixels, Gx and Gy are first calculated for each pixel. First, calculate Gx and Gy for each pixel value using the following formula.

$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} Angle(\theta) = \left|tan^{-1}\right| \left(\frac{Gy}{Gx}\right)$$

After calculating with Gx, the size and angle of each pixel is calculated using the following formula.

3.2.3 CNN Architecture

This section describes the proprietary CNN architecture utilized in this defect detection system. Because this system must be able to swiftly scan images collected from High quality camera in order to inspect the roll as quickly as possible, a simpler architecture was selected instead of other more complex state-of-the-art deep learning models. To boost system speed even more, only flaw detection is conducted at this step, leaving defect classification at a subsequent stage. Figure 2 depicts a high-level overview of the CNN layers and their size. This architecture is made up of four convolutional and max-pool layers.

graphics

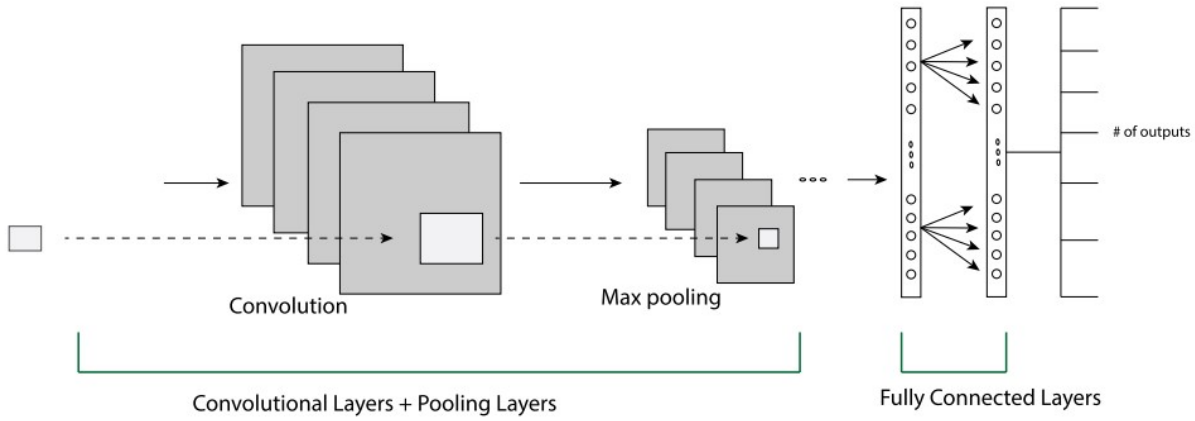


Figure 3.2: cnn architecture

which are followed by two fully-connected layers. All of the activations made use of ReLU. Table 1 lists all of the layers and their hyperparameters, where F denotes the number of feature mappings, K denotes the kernel size, and S is the stride parameter. As previously stated, this CNN was trained and validated using the Leather-Dataset. The network's activations at each layer were also displayed to ensure that it could recognize the leather defect features.

Table 3.1: The proposed custom CNN architecture hyperparameters

Layer	**Output**	**Hyperparameters**
INPUT	150*150*1	
CONV_1	150*150*64	F=64; K=5*5; S=1;
MAX-POOL_1	75*75*64	Pool=2*2; S=2;
CONV_2	75*75*64	F=64; K=5*5; S=1;
MAX_POOL_2	38*38*64	Pool=2*2; s=2;
CONV_3	38*38*124	F=128; k=3*3; S=1;
MAX_POOL_3	13*13*128	Pool=2*2; s=3;
CONV_4	13*13*128	F=128; K=3*3; S=1;
MAX_POOL_4	5*5*128	Pool=2*2; s=3;
Flatten	3200	
FC-2	256	Neurons=256
FC-2	128	Neurons=128
CNN OUTPUT	1	Optm=RMSPROP Loss=binary_crossentropy

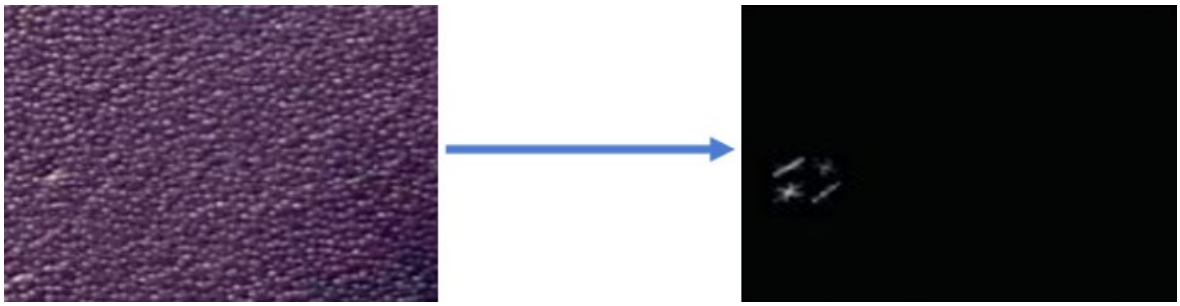


Figure 3.3: The example of after applying the step of preprocessing method: Histogram gradient

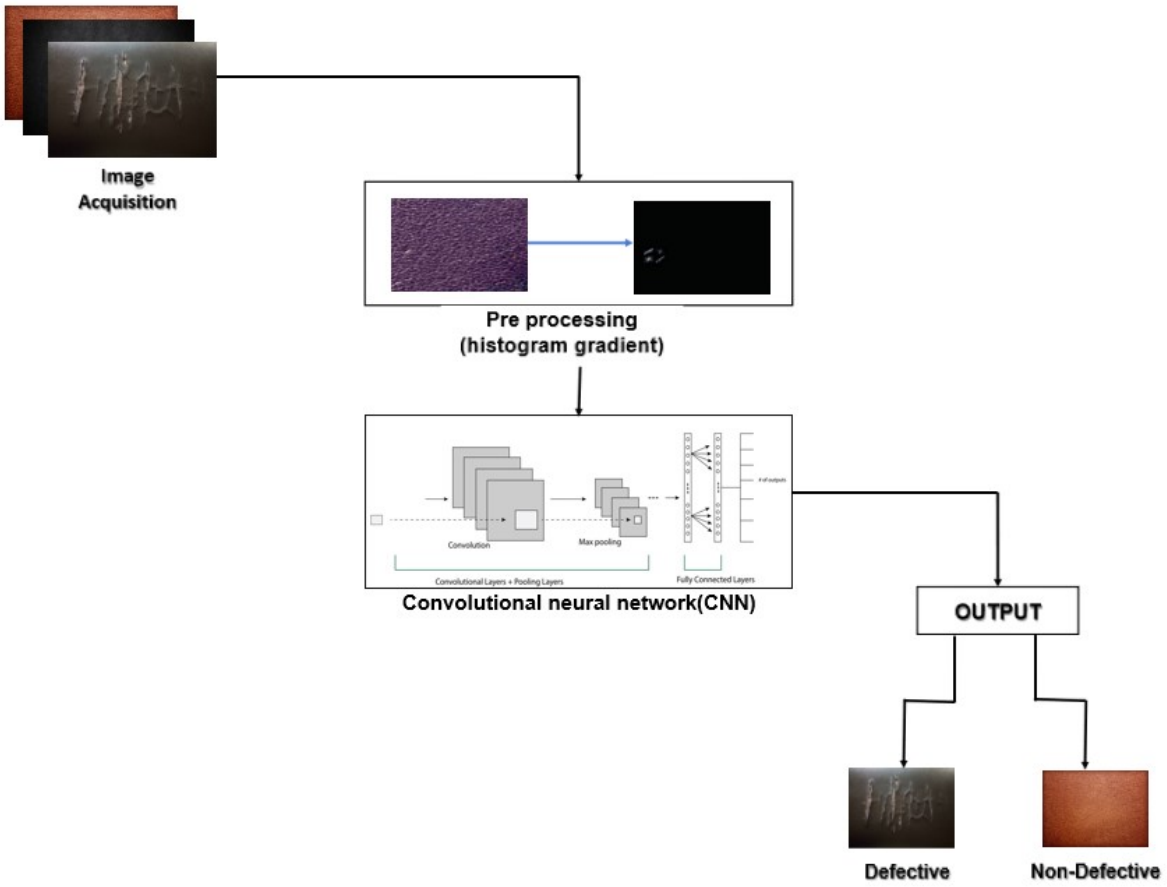


Figure 3.4: Block diagram of the Proposed automatic defect detection system

Figure 3.5: Sample leather images that contain (A) defective and (B) non defective.

3.4 Performance metrics

A) Confusion Matrix

Before going into the specifics of each statistic, let's first review the basic terminology utilized in categorization issues. The confusion matrix, which is a tabular depiction of the model predictions vs the ground-truth labels, is a crucial concept in classification performance. Each row of the confusion matrix represents occurrences from a predicted class, while each column represents instances from an actual class. Let's have a look at an example. Assume we're developing a binary classifier to distinguish between faulty and non-defective photos. Assume our data set contains 1100 leather sample pictures (1000 non-defective images and 100 defective images) and the confusion matrix shown below.

Table 3.2: Confusion matrix

		Actual class	
		Defective	Non-defective
Predicted class	Defective	90	60
	Non defective	10	940

The program accurately predicted 90 of the 100 Defective pictures and misclassified ten of them. If we consider the "defective" class to be positive and the non-defective class to be negative, then 90 samples projected as cat are genuine positives and 10 samples predicted as non-cat are false negatives. The model successfully categorized 940 of 1000 non-defective photos and incorrectly classified 60. The 940 correctly classified samples are referred to as true-negative, while the 60 incorrectly classified samples are referred to as false-positive. As we can see, the diagonal parts of this matrix represent the right prediction for distinct classes, but the off-diagonal elements represent the samples that were misclassified. We have a better knowledge of the now that we have a better comprehension of the confusion matrix Let's move on from the confusion matrix and into the real metrics.

B) Accuracy

This is a binary classification issue, with the result labelled "defective" or "non-defective." As a result of the 2*2 confusion matrix, the four metrics listed below may be produced.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{3.1}$$

Accuracy = (90+940)/ (1000+100) = 1030/1100= 93.6where TP is the predicted pixel that accurately recognized the defective pixel; TN is the correctly predicted non-defective pixel; FP is the pixel that is wrongly forecasted as a defective pixel; and FN is the undiscovered defective pixels.

C)Precision

In many circumstances, classification accuracy is not a reliable predictor of model performance. When your class distribution is skewed, this is one of these circumstances (one class is more frequent than others). In this situation, even forecasting all samples as the most common class would result in a

high accuracy rate, which makes no sense (because your model is not learning anything, and is just predicting everything as the top class). For example, in our defective vs non-defective classification above, if the model forecasts all samples as non-defective, the model would score 1000/1100= 90.9 percent correct. As a result, we must consider class-specific performance indicators as well. Precision is one of these measures, and it is defined as follows:

$$Precision = (TruePositive + FalsePositive)/(TruePositive + FalsePositive) \tag{3.2}$$

In the preceding example, the accuracy of the Defective and Non-Defective classes may be computed as follows: Precision Defective = samples properly predicted as Defective samples predicted as Defective = 90/ (90+60) = 60 Non-Defective= 940/950= 98.9 As we can see, the model predicts non-defective data significantly more accurately than defective. This is hardly surprising given that the model has seen more non-defective photos during training, making it stronger at categorizing that class.

D)Recall

Another significant metric is recall, which is defined as the proportion of samples from a class that the model correctly predicts. In more formal terms:

$$Recall = TruePositive/(TruePositive + FalseNegative) \tag{3.3}$$

As a result of our preceding example, the recall rate of defective and non-defective classes may be calculated as follows: Recall defective = 90/100 = 90Recall Non-defective= 940/1000 = 94

E) F1-Score Depending on the application, you may wish to prioritize recall or accuracy. However, there are numerous situations where both recall and accuracy are critical. As a result, it is logical to consider how to integrate these two into a single measure. The F1-score, which is the harmonic mean of accuracy and recall, is a common statistic that combines precision and recall.

$$F1 - score = 2 * Precision * Recall/(Precision + Recall) \tag{3.4}$$

So, for our classification example with the confusion matrix in Figure 1, the F1-score is F1 Defective= 2*0.6*0.9 / (0.6+0.9) = 72 The generalized definition of F-score is as follows. As we can see, F1-score is a subset of $F_B when B = 1$.

CHAPTER 4

PROJECT STUDY AND REQUIREMENTS

4.1 Feasibility Study

A feasibility study is carried out in order to select the optimal framework for meeting the execution requirements. I reduce the chances of extreme shame at a later stage of the framework undertaking just by putting out the effort to examine the risk.

4.1.1 Economic Feasibility

The cost-benefit analysis is one of the most important pieces of information in a plausibility study.That is, an assessment of the financial case for a PC-based system. In the framework, a cost benefit analysis depicts the costs of improvement and weighs them against significant and intangible advantages.

4.1.2 Technical Feasibility

specialized investigation evaluates the framework's specialised benefits while also gathering additional data on execution, reliability, viability, and profitability. Occasionally, this framework investigation step will also include a small amount of study and structure.

4.1.3 Social Feasibility

Operational practicality measures how well the arrangement will work in the organisation and how the executives feel about the framework as a whole. The proposed structure is beneficial to all of the association's clients.

4.2 System Specification

Profound learning is a CPU-intensive task for serious programmers, therefore be prepared to spend a lot of money on a good framework. Here are some prerequisites for the framework.

4.2.1 Hardware Specification

tel Core i7 Skylake or higher quad-core processor. M.2 PCIe or standard PCIe SSD with at least 256GB of capacity, while 512GB is recommended for performance. The better the framework performs, the faster you can stack and save your applications. High-end graphics cards, such as GTX 980 or 980Ms for a PC and 1080s or 1070s for a workstation. arrangement.

4.2.2 Software Specification

All you need is a Python IDE with built-in Python. GOOGLE COLAB is the one that is User friendly and simple to use. A section of the Modules/Package should have been introduced.

4.3 Standards and Policies

The project's standards ensure that quality is maintained throughout the project and that all paper- work produced is effectively managed. The project is kept up to date and completed in accordance with the system's guidelines. Within the scope, the project's policies are effectively controlled. quality, resources (including time and risk constraints). Appropriate governance and oversight are in place. Communication, quality, and risk management plans are designed and implemented throughout the life of a project. During the life of a project, appropriate authorisation and acceptance must be established.

CHAPTER 5

IMPLEMENTATION METHODOLOGY

5.1 SOFTWARE DESCRIPTION

GOOGLE COLAB

To be precise, Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

Figure 5.1: colab logo

What Colab Offers You? As a programmer, you can perform the following using Google Colab.

1)Write and execute code in Python

2)Document your code that supports mathematical equations

3)Create/Upload/Share notebooks

4)Import/Save notebooks from/to Google Drive

5)Import/Publish notebooks from GitHub

6)Import external datasets e.g. from Kaggle

7)Integrate PyTorch, TensorFlow, Keras, OpenCV

8)Free Cloud service with free GPU

5.2 MODULE DESCRIPTION

5.2.1 SCIKIT-LIBRARY

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy. Scikit-learn is a NumFOCUS fiscally sponsored project.

Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. Furthermore, some core algorithms are written in Cython to improve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR. In such cases, extending these methods with Python may not be possible.Scikit-learn integrates well with many other Python libraries, such as Matplotlib and plotly for plotting, NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.It was originally called scikits.learn and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation),took this project at another level .Prerequisites Installation If you already installed NumPy and Scipy,following are the two easiest ways to install scikit-learn Using pip Following command can be used to install scikit-learn via pip pip install -U scikit-learn. Using conda Following command can be used to install scikit-learn via conda conda install scikit-learn. On the other hand, if NumPy and Scipy is not yet installed on your Python workstation then, you can install them by using either pip or conda.Another option to use scikit-learn is to use Python distributions like Canopy and Anaconda because they both ship the latest version of scikit-learn. Features: Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data.

Some of the most popular groups of models provided by Sklearn are as follows Supervised Learning algorithms Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn. Unsupervised Learning algorithms On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks. Clustering This model is used for grouping unlabeled data. Cross Validation It is used to check the accuracy of supervised models on unseen data. Dimensionality Reduction It is used for reducing the

number of attributes in data which can be further used for summarisation, visualisation and feature selection. Ensemble methods As name suggest, it is used for combining the predictions of multiple supervised models. Feature extraction It is used to extract the features from data to define the attributes in image and text data. Feature selection It is used to identify useful attributes to create supervised models. Open Source It is open source library and also commercially usable under BSD license.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by French data scientist David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.[5] The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010.[6] Of the various scikits, scikit-learn as well as scikit-image were described as "well-maintained and popular" in November 2012.[7] Scikit-learn is one of the most popular machine learning libraries on GitHub.

Figure 5.2: scikit learn

5.2.2 NUMPY

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences:NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.

The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more

efficiently and with less code than is possible using Python's built-in sequences.

A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays; though these typically support Python-sequence input, they convert such input to NumPy arrays prior to processing, and they often output NumPy arrays. In other words, in order to efficiently use much (perhaps even most) of today's scientific/mathematical Python-based software, just knowing how to use Python's built-in sequence types is insufficient - one also needs to know how to use NumPy arrays.

Numpy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project. The Python programming language was not originally designed for numerical computing, but attracted the attention of the scientific and engineering community early on. In 1995 the special interest group (SIG) matrix-sig was founded with the aim of defining an array computing package; among its members was Python designer and maintainer Guido van Rossum, who extended Python's syntax (in particular the indexing syntax) to make array computing easier.An implementation of a matrix package was completed by Jim Fulton, then generalized[further explanation needed] by Jim Hugunin and called Numeric (also variously known as the "Numerical Python extensions" or "NumPy").Hugunin, a graduate student at the Massachusetts Institute of Technology (MIT), joined the Corporation for National Research Initiatives (CNRI) in 1997 to work on JPython,leaving Paul Dubois of Lawrence Livermore National Laboratory (LLNL) to take over as maintainer.Other early contributors include David Ascher, Konrad Hinsen and Travis Oliphant.

A new package called Numarray was written as a more flexible replacement for Numeric. Like Numeric, it too is now deprecated.Numarray had faster operations for large arrays, but was slower than Numeric on small ones,so for a time both packages were used in parallel for different use cases. The last version of Numeric (v24.2) was released on 11 November 2005, while the last version of numarray (v1.5.2) was released on 24 August 2006. There was a desire to get Numeric into the Python standard library, but Guido van Rossum decided that the code was not maintainable in its state then. In early 2005, NumPy developer Travis Oliphant wanted to unify the community around a single array package and ported Numarray's features to Numeric, releasing the result as NumPy 1.0 in 2006.This new project was part of SciPy.

To avoid installing the large SciPy package just to get an array object, this new package was separated and called NumPy. Support for Python 3 was added in 2011 with NumPy version 1.5.0.In 2011, PyPy started development on an implementation of the NumPy API for PyPy. It is not yet fully compatible with NumPy NumPy targets the CPython reference implementation of Python, which is a non-optimizing bytecode interpreter. Mathematical algorithms written for this version of Python often run much slower than compiled equivalents due to the absence of compiler optimization. NumPy addresses the slowness problem partly by providing multidimensional arrays and functions

and operators that operate efficiently on arrays; using these requires rewriting some code, mostly inner loops, using NumPy.

Using NumPy in Python gives functionality comparable to MATLAB since they are both interpreted,[21] and they both allow the user to write fast programs as long as most operations work on arrays or matrices instead of scalars. In comparison, MATLAB boasts a large number of additional toolboxes, notably Simulink, whereas NumPy is intrinsically integrated with Python, a more modern and complete programming language. Moreover, complementary Python packages are available; SciPy is a library that adds more MATLAB-like functionality and Matplotlib is a plotting package that provides MATLAB-like plotting functionality. Internally, both MATLAB and NumPy rely on BLAS and LAPACK for efficient linear algebra computations.

Python bindings of the widely used computer vision library OpenCV utilize NumPy arrays to store and operate on data. Since images with multiple channels are simply represented as three-dimensional arrays, indexing, slicing or masking with other arrays are very efficient ways to access specific pixels of an image. The NumPy array as universal data structure in OpenCV for images, extracted feature points, filter kernels and many more vastly simplifies the programming workflow and debugging. The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory.[12] In contrast to Python's built-in list data structure, these arrays are homogeneously typed: all elements of a single array must be of the same type.

Such arrays can also be views into memory buffers allocated by C/C++, Cython, and Fortran extensions to the CPython interpreter without the need to copy data around, giving a degree of compatibility with existing numerical libraries. This functionality is exploited by the SciPy package, which wraps a number of such libraries (notably BLAS and LAPACK). NumPy has built-in support for memory-mapped ndarrays. Inserting or appending entries to an array is not as trivially possible as it is with Python's lists. The np.pad(...) routine to extend arrays actually creates new arrays of the desired shape and padding values, copies the given array into the new one and returns it. NumPy's np.concatenate([a1,a2]) operation does not actually link the two arrays but returns a new one, filled with the entries from both given arrays in sequence. Reshaping the dimensionality of an array with np.reshape(...) is only possible as long as the number of elements in the array does not change. These circumstances originate from the fact that NumPy's arrays must be views on contiguous memory buffers. A replacement package called Blaze attempts to overcome this limitation.

Algorithms that are not expressible as a vectorized operation will typically run slowly because they must be implemented in "pure Python", while vectorization may increase memory complexity of some operations from constant to linear, because temporary arrays must be created that are as large as the inputs. Runtime compilation of numerical code has been implemented by several groups to avoid these problems; open source solutions that interoperate with NumPy include scipy.weave, numexpr and Numba.Cython and Pythran are static-compiling alternatives to these. Many modern large-scale scientific computing applications have requirements that exceed the capabilities of the NumPy arrays. For example, NumPy arrays are usually loaded into a computer's memory, which might have insufficient capacity for the analysis of large datasets.

Further, NumPy operations are executed on a single CPU. However, many linear algebra operations can be accelerated by executing them on clusters of CPUs or of specialized hardware, such as GPUs and TPUs, which many deep learning applications rely on. As a result, several alternative array implementations have arisen in the scientific python ecosystem over the recent years, such as Dask for distributed arrays and TensorFlow or JAX for computations on GPUs. Because of its popularity, these often implement a subset of Numpy's API or mimic it, so that users can change their array implementation with minimal changes to their code required.A recently introduced library named CUPy, accelerated by Nvidia's CUDA framework, has also shown potential for faster computing, being a 'drop-in replacement' of NumPy

Numpy is one of the most commonly used packages for scientific computing in Python. It provides a multidimensional array object, as well as variations such as masks and matrices, which can be used for various math operations. Numpy is compatible with, and used by many other popular Python packages, including pandas and matplotlib. Another reason is that numpy arrays and operations are vectorized, which means they lack explicit looping or indexing in the code. This makes the code not only more readable, but also more similar to standard mathematical notation.

Figure 5.3: Numpy

5.2.3 PANDAS:

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license.[2] The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals.

1.Data Frame object for data manipulation with integrated indexing.

2.Tools for reading and writing data between in-memory data structures and different file formats.

3.Data alignment and integrated handling of missing data.

4Reshaping and pivoting of data sets.

5.Label-based slicing, fancy indexing, and subsetting of large data sets.

6.Data structure column insertion and deletion.

7.Group by engine allowing split-apply-combine operations on data sets.

8.Data set merging and joining.

8.Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.

9.Time series-functionality: Date range generation[6] and frequency conversions, moving window statistics, moving window linear regressions, date shifting and lagging.

10.Provides data filtration.

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis/manipulation tool available in any language. It is already well on its way toward this goal. pandas is well suited for many different kinds of data:

• Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet.

• Ordered and unordered (not necessarily fixed-frequency) time series data.

• Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels.

• Any other form of observational / statistical data sets. The data need not be labeled at all to be placed into a pandas data structure.

The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, DataFrame provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Here are just a few of the things that pandas does well:

• Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data.

• Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects.

• Automatic and explicit data alignment: objects can be explicitly aligned to a set of labels, or the user can simply ignore the labels and let Series, DataFrame, etc. automatically align the data for you in computations.

• Powerful, flexible group by functionality to perform split-apply-combine operations on data sets, for both aggregating and transforming data.

• Make it easy to convert ragged, differently-indexed data in other Python and NumPy data structures into DataFrame objects.

• Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.

• Intuitive merging and joining data sets.

• Flexible reshaping and pivoting of data sets.

• Hierarchical labeling of axes (possible to have multiple labels per tick).

• Robust IO tools for loading data from flat files (CSV and delimited), Excel files, databases, and saving / loading data from the ultrafast HDF5 format.

• Time series-specific functionality: date range generation and frequency conversion, moving window statistics, date shifting, and lagging.

Many of these principles are here to address the shortcomings frequently experienced using other languages / scientific research environments. For data scientists, working with data is typically divided into multiple stages: munging and cleaning data, analyzing / modeling it, then organizing the results of the analysis into a form suitable for plotting or tabular display. pandas is the ideal tool for all of these tasks. Some other notes.

• pandas is fast. Many of the low-level algorithmic bits have been extensively tweaked in Cython code. However,as with anything else generalization usually sacrifices performance. So if you focus on one feature for your application you may be able to create a faster specialized tool.

• pandas is a dependency of statsmodels, making it an important part of the statistical computing ecosystem in Python.

• pandas has been used extensively in production in financial applications.

The axis labeling information in pandas objects serves many purposes:

• Identifies data (i.e. provides metadata) using known indicators, important for analysis, visualization, and interactive console display.

• Enables automatic and explicit data alignment.

• Allows intuitive getting and setting of subsets of the data set.

Figure 5.4: Pandas

5.2.4 MATPLOTLIB

Matplotlib is a library for making 2D plots of arrays in Python. Although it has its origins in emulating the MATLAB®1 graphics commands, it is independent of MATLAB, and can be used in a Pythonic, object oriented way. Although Matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays. Matplotlib is designed with the philosophy that you should be able to create simple plots with just a few commands, or just one! If you want to see a histogram of your data, you shouldn't need to instantiate objects, call methods, set properties, and so on; it should just work. For years, I used to use MATLAB exclusively for data analysis and visualization. MATLAB excels at making nice looking plots easy. When I began working with EEG data, I found that I needed to write applications to interact with my data, and developed an EEG analysis application in MATLAB. As the application grew in complexity, interacting with databases, http servers, manipulating complex data structures, I began to strain against the limitations of MATLAB as a programming language, and decided to start over in Python. Python more than makes up for all of MATLAB's deficiencies as a programming language, but I was having difficulty finding a 2D plotting package (for 3D VTK more than exceeds all of my needs).

When I went searching for a Python plotting package, I had several requirements:

• Plots should look great - publication quality. One important requirement for me is that the text looks good (antialiased, etc.)

• Postscript output for inclusion with TeX documents

• Embeddable in a graphical user interface for application development

• Code should be easy enough that I can understand it and extend it

• Making plots should be easy

Finding no package that suited me just right, I did what any self-respecting Python programmer would do: rolled up my sleeves and dived in. Not having any real experience with computer graphics, I decided to emulate MATLAB's plotting capabilities because that is something MATLAB does very well. This had the added advantage that many people have a lot of MATLAB experience, and thus they can quickly get up to steam plotting in python. From a developer's perspective, having a fixed user interface (the pylab interface) has been very useful, because the guts of the code base can be redesigned without affecting user code.

The Matplotlib code is conceptually divided into three parts: the pylab interface is the set of functions provided by matplotlib.pylab which allow the user to create plots with code quite similar to MATLAB figure generating code (Pyplot tutorial). The Matplotlib frontend or Matplotlib API is the set of classes that do the heavy lifting, creating and managing figures, text, lines, plots and so on (Artist tutorial). This is an abstract interface that knows nothing about output. The backends are device-dependent drawing devices, aka renderers, that transform the frontend representation to hardcopy or a display device (What is a backend?). Example backends: PS creates PostScript® hardcopy, SVG creates Scalable Vector Graphics hardcopy, Agg creates PNG output using the high quality Anti-Grain Geometry library that ships with Matplotlib, GTK embeds Matplotlib in a Gtk+

application, GTKAgg uses the Anti-Grain renderer to create a figure and embed it in a Gtk+ application, and so on for PDF, WxWidgets, Tkinter, etc. Matplotlib is used by many people in many different contexts. Some people want to automatically generate PostScript files to send to a printer or publishers. Others deploy Matplotlib on a web application server to generate PNG output for inclusion in dynamically-generated web pages. Some use Matplotlib interactively from the Python shell in Tkinter on Windows^{TM}. My primary use is to embed Matplotlib in a Gtk+ EEG application that runs on Windows, Linux and Macintosh OS X.

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

1.Create publication quality plots.
2.Make interactive figures that can zoom, pan, update.
3.Customize visual style and layout.
4.Export to many file formats .
5.Embed in JupyterLab and Graphical User Interfaces.
6.Use a rich array of third-party packages built on Matplotlib.

The Python shipped from https://www.python.org is compiled with Visual Studio 2008 for versions before 3.3, Visual Studio 2010 for 3.3 and 3.4, and Visual Studio 2015 for 3.5 and 3.6. Python extensions are recommended to be compiled with the same compiler. Since there is no canonical Windows package manager, the methods for building freetype, zlib, and libpng from source code are documented as a build script at matplotlib-winbuild.

Figure 5.5: Matplotlib

5.2.5 TENSORFLOW

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow provides all of this for the programmer by way of the Python language. Python is easy to learn and work with, and provides convenient ways to express how high-level abstractions

can be coupled together. Nodes and tensors in TensorFlow are Python objects, and TensorFlow applications are themselves Python applications.

The actual math operations, however, are not performed in Python. The libraries of transformations that are available through TensorFlow are written as high-performance C++ binaries. Python just directs traffic between the pieces, and provides high-level programming abstractions to hook them together.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. The resulting models created by TensorFlow, though, can be deployed on most any device where they will be used to serve predictions.

In this section we provide an in-depth discussion of the abstract computational principles underlying the TensorFlow software library. We begin with a thorough examination of the basic structural and architectural decisions made by the TensorFlow development team and explain how machine learning algorithms may be expressed in its dataflow graph language. Subsequently, we study TensorFlow's execution model and provide insight into the way TensorFlow graphs are assigned to available hardware units in a local as well as distributed environment. Then, we investigate the various optimizations incorporated into TensorFlow, targeted at improving both software and hardware efficiency. Lastly, we list extensions to the basic programming model that aid the user in both computational as well as logistical aspects of training a machine learning model with TensorFlow.

A. Computational Graph Architecture In TensorFlow, machine learning algorithms are represented as computational graphs. A computational or dataflow graph is a form of directed graph where vertices or nodes describe operations, while edges represent data flowing between these operations. If an output variable z is the result of applying a binary operation to two inputs x and y, then we draw directed edges from x and y to an output node representing z and annotate the vertex with a label describing the performed computation. Examples for computational graphs are given in Figure 2. The following paragraphs discuss the principle elements of such a dataflow graph, namely operations, tensors, variables and sessions.

1) Operations: The major benefit of representing an algorithm in form of a graph is not only the intuitive (visual) expression of dependencies between units of a computational model, but also the fact that the definition of a node within the graph can be kept very general. In TensorFlow, nodes represent operations, which in turn express the combination or transformation of data flowing through the graph [8]. An operation can have zero or more inputs and produce zero or more outputs. As such, an operation may represent a mathematical Fig. 3.11: Examples of computational graphs. The left graph displays a very simple computation, consisting of just an addition of the two input variables x and y. In this case, z is the result of the operation +, as the annotation suggests. The right graph gives a more complex example of computing a logistic regression variable yˆ in for some example vector x, weight vector w as well as a scalar bias b. As shown in the graph, yˆ is the result of the sigmoid or logistic function . equation, a variable or constant, a control flow directive, a file I/O operation or even a network communication port. It may seem unintuitive that an operation, which the

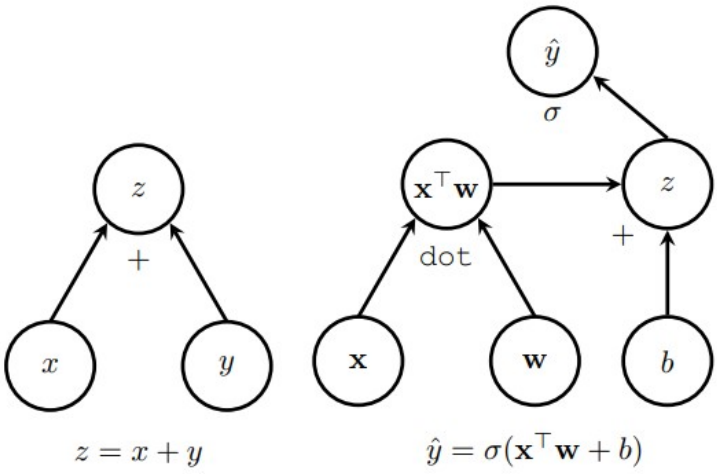


Figure 5.6: Examples of computational graphs

reader may associate with a function in the mathematical sense, can represent a constant or variable. However, a constant may be thought of as an operation that takes no inputs and always produces the same output corresponding to the constant it represents. Analogously, a variable is really just an operation taking no input and producing the current state or value of that variable. Table ?? gives an overview of different kinds of operations that may be declared in a TensorFlow graph. Any operation must be backed by an associated implementation. In [8] such an implementation is referred to as the operation's kernel. A particular kernel is always specifically built for execution on a certain kind of device, such as a CPU, GPU or other hardware unit.

2) Tensors: In TensorFlow, edges represent data flowing from one operation to another and are referred to as tensors. A tensor is a multi-dimensional collection of homogeneous values with a fixed, static type. The number of dimensions of a tensor is termed its rank. A tensor's shape is the tuple describing its size, i.e. the number of components, in each dimension. In the mathematical sense, a tensor is the generalization of two-dimensional matrices, one-dimensional vectors and also scalars, which are simply tensors of rank zero. In terms of the computational graph, a tensor can be seen as a symbolic handle to one of the outputs of an operation. A tensor itself does not hold or store values in memory, but provides only an interface for retrieving the value referenced by the tensor. When creating an operation in the TensorFlow programming environment, such as for the expression x + y, a tensor object is returned. This tensor may then be supplied as input to other computations, thereby connecting the source and destination operations with an edge. By these means, data flows through a TensorFlow graph. Next to regular tensors, TensorFlow also provides a SparseTensor data structure, allowing for a more spaceefficient dictionary-like representation of sparse tensors with only few non-zeros entries.

3) Variables: In a typical situation, such as when performing stochastic gradient descent (SGD), the graph of a machine learning model is executed from start to end multiple times for a single experiment. Between two such invocations, the majority of tensors in the graph are destroyed and do not persist.

However, it is often necessary to maintain state across evaluations of the graph, such as for the weights and parameters of a neural network. For this purpose, there exist variables in TensorFlow, which are simply special operations that can be added to the computational graph. In detail, variables can be described as persistent, mutable handles to in-memory buffers storing tensors. As such, variables are characterized by a certain shape and a fixed type. To manipulate and update variables, TensorFlow provides the assign family of graph operations. When creating a variable node for a TensorFlow graph, it is necessary to supply a tensor with which the variable is initialized upon graph execution. The shape and data type of the variable is then deduced from this initializer. Interestingly, the variable itself does not store this initial tensor. Rather, constructing a variable results in the addition of three distinct nodes to the graph:

1) The actual variable node, holding the mutable state.

2) An operation producing the initial value, often a constant.

3) An initializer operation, that assigns the initial value
to the variable tensor upon evaluation of the graph.

4) Sessions: In TensorFlow, the execution of operations and evaluation of tensors may only be performed in a special environment referred to as session. One of the responsibilities of a session is to encapsulate the allocation and management of resources such as variable buffers. Moreover, the Session interface of the TensorFlow library provides a run routine, which is the primary entry point for executing parts or the entirety of a computational graph. This method takes as input the nodes in the graph whose tensors should be computed and returned. Moreover, an optional mapping from arbitrary nodes in the graph to respective replacement values — referred to as feed nodes — may be supplied to run as well The three nodes that are added to the computational graph for every variable definition. The first, v, is the variable operation that holds a mutable in-memory buffer containing the value tensor of the variable. The second, i, is the node producing the initial value for the variable, which can be any tensor. Lastly, the assign node will set the variable to the initializer's value when executed. The assign node also produces a tensor referencing the initialized value v 0 of the variable, such that it may be connected to other nodes as necessary (e.g. when using a variable as the initializer for another variable).

Figure 5.7: Tensorflow

5.2.6 KERAS

Keras is an open source machine-learning library written in python and developed by a Googleengineer named François Chollet for quickly building deep learning models by only writing fewlines of code. It is a high-level API that runs on top of Tensorflow, Theano, and CNTK andwraps up extensive complex numerical computation to perform activities like objectsidentification, image labeling, sentiment analysis, automated car driving etc. keras provides aconvenient solution to deep learning problems by replacing hundreds of lines of conventionalcode with only few lines of codes. It removes the effort of building a complex network that couldbe challenging to build through conventional approaches that might take hours or days to handcode in Python or other languages. Both industry and research community has a strongeradoption of keras API as compared to other deep learning libraries. tf.keras module is the officialfrontend of TensorFlow, which is the most popular API among other deep learning libraries.Keras is used in Netflix, Yelp, Uber, Square, Instacart, Zocdoc, and many others. In addition, ittakes 2nd place in terms of research article published by preprint arXiv.org and is adopted byresearchers at large scientific organizations such as NASA and CERN.

Keras runs on top of open source machine libraries like TensorFlow, Theano or Cognitive Toolkit (CNTK). Theano is a python library used for fast numerical computation tasks. TensorFlow is the most famous symbolic math library used for creating neural networks and deep learning models. TensorFlow is very flexible and the primary benefit is distributed computing. CNTK is deep learning framework developed by Microsoft. It uses libraries such as Python, C, C++ or standalone machine learning toolkits. Theano and TensorFlow are very powerful libraries but difficult to understand for creating neural networks. Keras is based on minimal structure that provides a clean and easy way to create deep learning models based on TensorFlow or Theano. Keras is designed to quickly define deep learning models. Well, Keras is an optimal choice for deep learning applications.

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly

1.Keras is an API that was made to be easy to learn for people. Keras was made to be simple. It offers consistent simple APIs, reduces the actions required to implement common code, and explains user error clearly.

2.Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs.

3.Languages with a high level of abstraction and inbuilt features are slow and building custom features in then can be hard. But Keras runs on top of TensorFlow and is relatively fast. Keras is also deeply integrated with TensorFlow, so you can create customized workflows with ease.

4.The research community for Keras is vast and highly developed. The documentation and help available are far more extensive than other deep learning frameworks.

5.Keras is used commercially by many companies like Netflix, Uber, Square, Yelp, etc which have deployed products in the public domain which are built using Keras.

6.It runs smoothly on both CPU and GPU.

7.It supports almost all neural network models.

8.It is modular in nature, which makes it expressive, flexible, and apt for innovative research.

Keras leverages various optimization techniques to make high level neural network API easier and more performant. It supports the following features:

1. Consistent, simple and extensible API.
2. Minimal structure - easy to achieve the result without any frills.
3. It supports multiple platforms and backends.
4. It is user friendly framework which runs on both CPU and GPU.
5. Highly scalability of computation.

Keras is highly powerful and dynamic framework and comes up with the following advantages:

A. Larger community support.

B. Easy to test.

C. Keras neural networks are written in Python which makes things simpler.

D. Keras supports both convolution and recurrent networks.

Figure 5.8: Keras

5.2.7 IMAGEIO

Imageio is a Python library that provides an easy interface to read and write a wide range of image data, including animated images, volumetric data, and scientific formats.ImageIO reads and writes images by deligating your request to one of many backends. Example backends are pillow, ffmpeg, tifffile among others. Each backend supports its own set of formats, which is how ImageIO manages to support so many of them. To help you navigate this format jungle, ImageIO provides various curated lists of formats depending on your use-case.

Below you can find a list of each plugin that exists in ImageIO together with the formats that this

plugin supports. This can be useful, for example, if you have to decide which plugins to install and/or depend on in your project.

FreeImage tif, tiff, jpeg, jpg, bmp, png, bw, dds, gif, ico, j2c, j2k, jp2,pbm, pcd, PCX, pgm, ppm, psd, ras, rgb, rgba, sgi, tga, xbm, xpm, pic, raw, 3fr, arw, bay, bmq, cap, cine, cr2, crw, cs1, cut, dc2, dcr, dng, drf, dsc, erf, exr, fff, g3, hdp, hdr, ia, iff, iiq, jif, jng, jpe, jxr, k25, kc2, kdc, koa, lbm, mdc, mef, mos, mrw, nef, nrw, orf, pct, pef, pfm, pict, ptx, pxn, qtk, raf, rdc, rw2, rwl, rwz, sr2, srf, srw, sti, targa, wap, wbm, wbmp, wdp, webp

Pillow tif, tiff, jpeg, jpg, bmp, png, bw, dds, gif, ico, j2c, j2k, jp2, pbm, pcd, PCX, pgm, ppm, psd, ras, rgb, rgba, sgi, tga, xbm, xpm, fit, fits, bufr, CLIPBOARDGRAB, cur, dcx, DIB, emf, eps, flc, fli, fpx, ftc, ftu, gbr, grib, h5, hdf, icns, iim, im, IMT, jfif, jpc, jpf, jpx, MCIDAS, mic, mpo, msp, ps, pxr, SCREENGRAB, SPIDER, wmf, XVTHUMB

ITK tif, tiff, jpeg, jpg, bmp, png, pic, img, lsm, dcm, dicom, gdcm, gipl, hdf5, hdr, img.gz, ipl, mgh, mha, mhd, mnc, mnc2, nhdr, nia, nii, nii.gz, nrrd, vtk

Figure 5.9: ImageIo

5.2.8 CONVOLUTIONAL NEURAL NETWORKS(CNN)

A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data. It contains a series of pixels arranged in a grid-like fashion that contains pixel values to denote how bright and what color each pixel should be.The human brain processes a huge amount of information the second we see an image. Each neuron works in its own receptive field and is connected to other neurons in a way that they cover the entire visual field. Just as each neuron responds to stimuli only in the restricted region of the visual field called the receptive field in the biological vision system, each neuron in a CNN processes data only in its

receptive field as well. The layers are arranged in such a way so that they detect simpler patterns first (lines, curves, etc.) and more complex patterns (faces, objects, etc.) further along. By using a CNN, one can enable sight to computers.

In this section, we describe each step using CNN for feature extraction and/or classification.

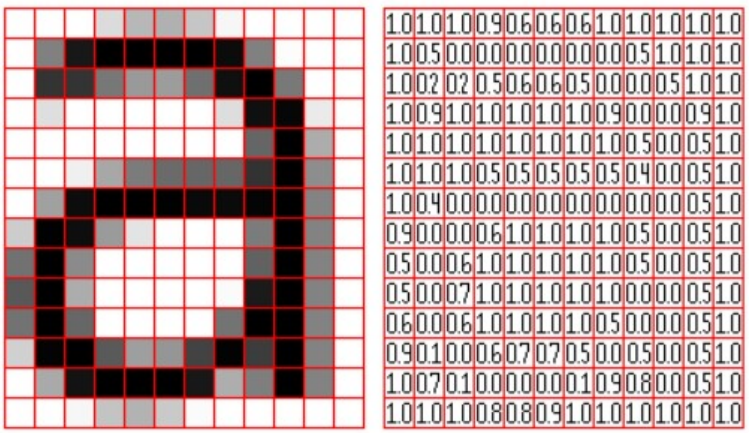


Figure 5.10: Representation of image as a grid of pixels

CNNs are deep feed-forward neural networks (NNs) composed of interconnected neurons that have inputs with learnable weights, biases, and activation functions. CNNs are built by repeatedly concatenating five classes of layers: convolutional (CONV), activation (ACT), and pooling (POOL), which are followed by a last stage that typically contains fully connected (FC) layers and a classification (CLASS) layer. The CONV layer performs feature extraction by convolving input to filters. After each CONV layer, a non-linear ACT layer is applied, such as the non-saturating ReLU (rectified linear unit) function f(x)=max(0,x) or the saturating hyperbolic tangent f(x)=tanh(x), f(x)=—tanh(x)—)— or the sig- moid function f(x)=(1+e-x)-1. Non-linearity activation is useful to improve classification and the learning capabilities of the network.

POOL layers perform non-linear downsampling operations aimed at reducing the spatial size of the representation while simultaneously decreasing (1) the number of parameters, (2) the possibility of overfitting, and (3) the computational complexity of the network. It is a common practice to insert a POOL layer between CONV layers. Typical pooling functions are max and average. FC layers have neurons that are fully connected to all the activations in the previous layer and are applied after CONV and POOL layers. In the higher layers, multiple FC layers and one CLASS layer perform the final classification. A widely used activation function in the CLASS layer is SoftMax. For audio classification, the audio images are downsized in order to speed up CNN classification performance [11]. Downsizing images reduces the number of neurons in the convolutional layers as well as the number of trainable parameters of the network. Downsizing is accomplished by taking only the first pixel of every four pixels in 22 subwindows of the image. As a result, both image height and width are cut by half. The CNN used in this work has two 2D convolutional layers with 64 filters followed by a max-pool layer. The 5th layer is a fully connected layer with 500 neurons. The activation function is the rectified linear units (ReLUs), except for the neurons of the last layer, which use Softmax, as mentioned above. It is important that the number of neurons in the last layer equals the number of classes for each problem.

Training is performed using backpropagation with 50 epochs. Once trained, the output of the 5th layer is used for feature extraction. This produces a 500- dimensional vector image representation. A convolutional neural network, or CNN, is a deep learning neural network sketched for

processing structured arrays of data such as portrayals. CNN are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces. This characteristic that makes convolutional neural network so robust for computer vision. CNN can run directly on a underdone image and do not need any preprocessing. A convolutional neural network is a feed forward neural network, seldom with up to 20. The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.

CNN contains many convolutional layers assembled on top of each other, each one competent of recognizing more so- phisticated shapes. With three or four convolutional layers it is viable to recognize handwritten digits and with 25 layers it is possible to differentiate human faces.
The agenda for this sphere is to activate machines to view the world as humans do, perceive it in a alike fashion and even use the knowledge for a multitude of duty such as image and video recognition, image inspection and classification, media recreation, recommendation systems, natural language processing, etc. Convolutional Neural Network Design includes, The construction of a convolutional neural network is a multi-layered feed-forward neural network, made by assembling many unseen layers on top of each other in a particular order. Itis the sequential design that give permission to CNN to learn hierarchical attributes. In CNN, some of them followed by grouping layers and hidden layers are typically convolutional layers followed by activation layers. The pre-processing needed in a ConvNet is kindred to that of the related pattern of neurons in the human brain and was motivated by the organization of the Visual Cortex.
Basic Design Principle for CNN is to develop an architecture and learning algorithm in such a way that it reduces the number of parameters without compromising the compression power of learning algorithms. In general, linear math operation of convolution followed by nonlinear activators, pooling layers, and deep neural net classification is one way to interpret the architecture. The convolution processes act as appropriate feature detectors that deal with a large amount of information (low level). Complete convolution layer has different feature detectors so multiple features can be extracted from same image. A single feature detector is smaller in size compared to input image and is slid over the image. Hence all units in feature detectors share same weight and bias, thereby helps in detecting the same feature in all points of image, and ,in addition, gives properties of invariance to transformation and shift of the images. Local connection between pixels are used many times in architecture. The idea of introducing local receptive field allows extraction of element of any features (architecture of edges, corners and end points)(See [Yann2] for more details). Higher degree of complex features are detected in hidden layers (when combined with hidden layer). Functions of sparse connectivity between subsequent layers, parameter sharing of weights between adjacent pixels and equivalent representation enable CNN to be more effective then by image recognition and image classification.

CNN ARCHITECTURE

Convolutional Neural Network Architecture A CNN typically has three layers: a convolutional layer, a pooling layer, and a fully connected layer.

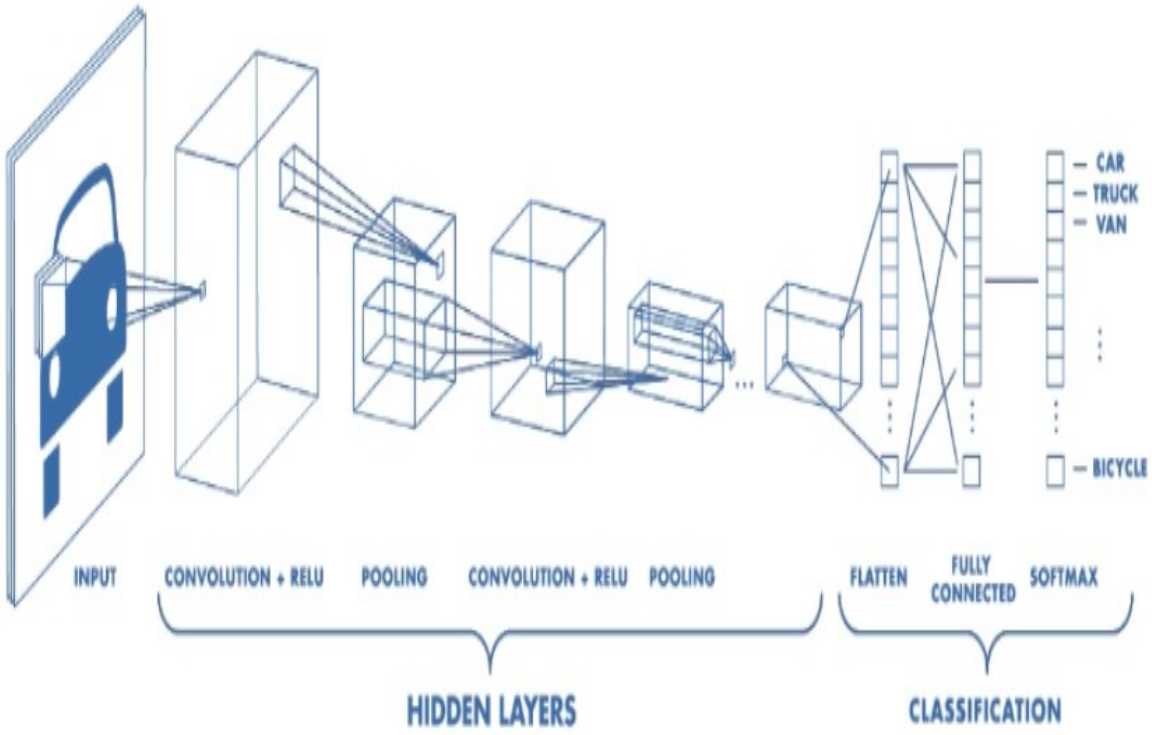


Figure 5.11: Architecture of a CNN

Convolution Layer

The convolution layer is the core building block of the CNN. It carries the main portion of the network's computational load.This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a kernel, and the other matrix is the restricted portion of the receptive field. The kernel is spatially smaller than an image but is more in-depth. This means that, if the image is composed of three (RGB) channels, the kernel height and width will be spatially small, but the depth extends up to all three channels.

During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a stride. If we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{OUT} \frac{W - F + 2P}{S} + 1 \tag{5.1}$$

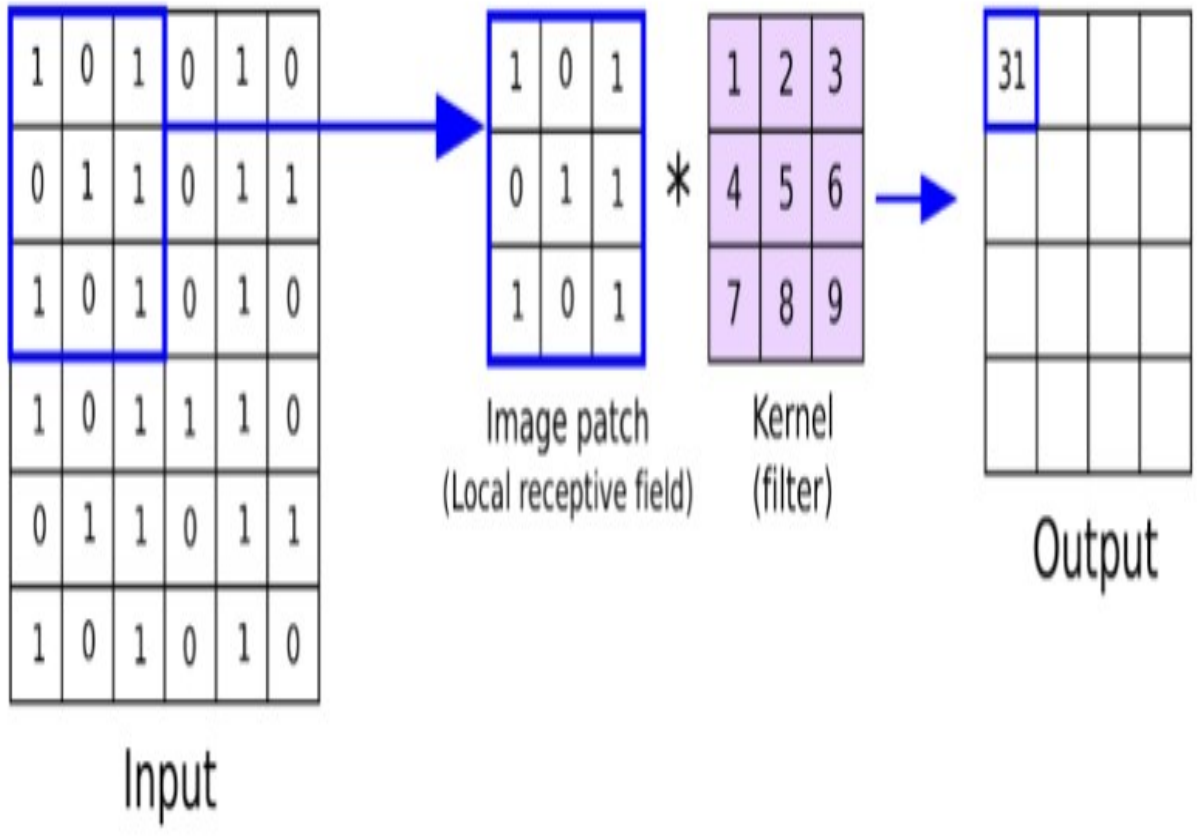


Figure 5.12: Illustration of Convolution layer Operation

This will yield an output volume of size Wout x Wout x Dout.

Motivation behind convolution

Convolution leverages three important ideas that motivated computer vision researchers: sparse interaction, parameter sharing, and equivariant representation. Let's describe each one of them in detail.

Trivial neural network layers use matrix multiplication by a matrix of parameters describing the interaction between the input and output unit. This means that every output unit interacts with every input unit. However, convolution neural networks have sparse interaction. This is achieved by making kernel smaller than the input e.g., an image can have millions or thousands of pixels, but while processing it using kernel we can detect meaningful information that is of tens or hundreds of pixels. This means that we need to store fewer parameters that not only reduces the memory requirement of the model but also improves the statistical efficiency of the model.

If computing one feature at a spatial point (x1, y1) is useful then it should also be useful at some other spatial point say (x2, y2). It means that for a single two-dimensional slice i.e., for creating one activation map, neurons are constrained to use the same set of weights. In a traditional neural network, each element of the weight matrix is used once and then never revisited, while convolution

network has shared parameters i.e., for getting output, weights applied to one input are the same as the weight applied elsewhere.

Due to parameter sharing, the layers of convolution neural network will have a property of equivariance to translation. It says that if we changed the input in a way, the output will also get changed in the same way.

Pooling Layer

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights. The pooling operation is processed on every slice of the representation individually. There are several pooling functions such as the average of the rectangular neighborhood, L2 norm of the rectangular neighborhood, and a weighted average based on the distance from the central pixel. However, the most popular process is max pooling, which reports the maximum output from the neighborhood.

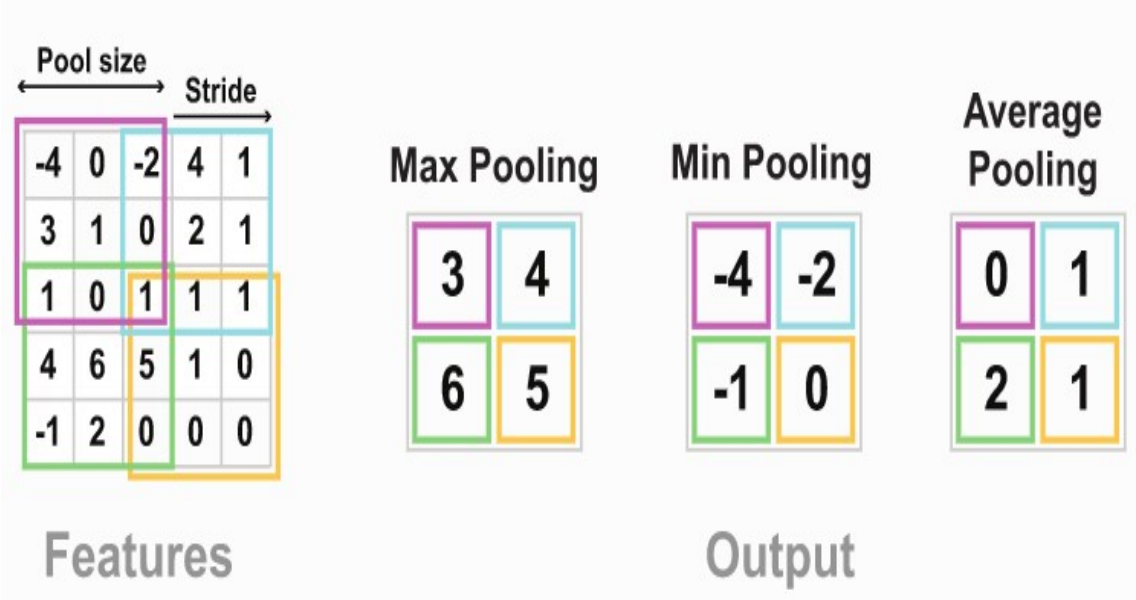


Figure 5.13: Illustration of pooling layer Operation

If we have an activation map of size W x W x D, a pooling kernel of spatial size F, and stride S, then the size of output volume can be determined by the following formula:

$$W_{out} \frac{W - F}{S} + 1 \tag{5.2}$$

Formula for Padding Layer. This will yield an output volume of size Wout x Wout x D. In all cases, pooling provides some translation invariance which means that an object would be recognizable regardless of where it appears on the frame.

Machine learning is the base of intelligence for computers and other electronic devices. It uses predictive models that can learn from existing data and forecast future behaviors, outcomes, and trends. Deep learning is a sub-field of machine learning, where models inspired by the human brain are expressed mathematically. In Deep Neural Networks, DNN, the parameters defining the mathematical models, which can be in the order of a few thousand to 100+ million, are learned automatically from the data. DNNs can model complex non-linear relationships between inputs and outputs. Their architectures generate compositional models where the object is expressed as a layered composition of primitives. Deep architectures include many variants of a few basic approaches.

DNN attempts to learn high-level abstractions in data by utilizing hierarchical architectures. It is an emerging approach and has been widely applied in traditional artificial intelligence domains, such as semantic parsing [1], transfer learning [2, 3], natural language processing [4], computer vision [5, 6] and many more. There are mainly three important reasons for the booming of deep learning today: the dramatically increased chip processing abilities, the significantly lowered cost of computing hardware, and the considerable advances in the machine learning algorithms [7].

In recent years, DNNs are widely considered and several models for different applications are proposed. These models can be divided into five categories [8, 9]: Convolution Neural Networks (CNN), Restricted Boltzmann Machines (RBM), Autoencoders, Sparse Coders, and Recurrent Neural Networks. CNN is one of the most important and useful types of DNNs, typically used in classification and object segmentation. A CNN consists of three main layers: convolution layer, pooling layer, and fully connected layer. Each of these layers does certain spatial operations. In convolution layers, CNN uses different kernels for convolving the input image for creating the feature maps. The pooling layer is usually inserted after a convolution layer. The application of this layer is reducing the size of feature maps and network parameters. After the pooling layer, there is a flatten layer followed by some fully connected layers. In the flatten layer, 2D feature maps produced in the previous layer are converted into 1D feature maps to be suitable for the following fully connected layers. The flattened vector can be used later for the classification of the images.

Pooling is a key-step in convolutional based systems that reduces the dimensionality of the feature maps. It combines a set of values into a smaller number of values, i.e., the reduction in the dimensionality of the feature map. It transforms the joint feature representation into valuable information by keeping useful information and eliminating irrelevant information. Pooling operators provide a form of spatial transformation invariance as well as reducing the computational complexity for upper layers by eliminating some connections between convolutional layers. This layer executes the down-sampling on the feature maps coming from the previous layer and produces the new feature

maps with a condensed resolution. This layer serves two main purposes: the first is to reduce the number of parameters or weights, thus lessening the computational cost and the second is to control overfitting. An ideal pooling method is expected to extract only useful information and discard irrelevant details.

Popular Pooling Methods

A.Average Pooling

The idea of average or mean for pooling and extracting the features, firstly introduced in [10] and used in [11] that is the first convolution-based deep neural network. As shown in Fig. 1, an average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region.

B.Max-Pooling

A max-pooling operator [12] can be applied to down-sample the convolutional output bands, thus reducing variability. The max-pooling operator passes forward the maximum value within a group of activations.

C.Mixed Pooling

Max pooling extracts only the maximum activation whereas average pooling down-weighs the activation by combining the non-maximal activations. To overcome this problem, Yu et al. [13] proposed a hybrid approach by combining the average pooling and max pooling. This approach is highly inspired by dropout [14] and Drop connect.

Fully connected layer

in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The FC layer helps to map the representation between the input and the output.

Non-Linearity Layers

Since convolution is a linear operation and images are far from linear, non-linearity layers are often placed directly after the convolutional layer to introduce non-linearity to the activation map.There are several types of non-linear operations, the popular ones being: At last we wanted to remove the data about the quantum portion again from the quantum circuit to take care of it into the traditional SVM calculation. This is genuine a non-paltry errand, since we need to quantify the cross-over of two states K(x ,z)=—(x)—(z)—2 yet there is a brilliant way of assessing this cross-over, in 1 they utilized this circuit for that assessment.

1.Sigmoid

The sigmoid non-linearity has the mathematical form () = 1/(1+e). It takes a real-valued number and "squashes" it into a range between 0 and 1.However, a very undesirable property of sigmoid is that when the activation is at either tail, the gradient becomes almost zero. If the local gradient becomes very small, then in backpropagation it will effectively "kill" the gradient. Also, if the data

coming into the neuron is always positive, then the output of sigmoid will be either all positives or all negatives, resulting in a zig-zag dynamic of gradient updates for weight.

2.Tanh

Tanh squashes a real-valued number to the range [-1, 1]. Like sigmoid, the activation saturates, but — unlike the sigmoid neurons — its output is zero centered.

3.ReLU

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function ()=max (0,). In other words, the activation is simply threshold at zero.In comparison to sigmoid and tanh, ReLU is more reliable and accelerates the convergence by six times.Unfortunately, a con is that ReLU can be fragile during training. A large gradient flowing through it can update it in such a way that the neuron will never get further updated. However, we can work with this by setting a proper learning rate.

$$\text{print}(\text{classification}_r eport(y_t est, labels))target_n ames = categories))$$

CHAPTER 6

RESULTS ANALYSIS

All tests were run on Python 3.6 with an Intel Core i5-9300H 2.40 GHz processor, 8.0 GB RAM, and an NVIDIA GeForce GTX 1080 Ti GPU. Table 3 displays the performance of the fault instance segmentation. To forecast the output, both the train and test datasets use the same learned CNN architecture. The train and test datasets include 100 and 1000 sample pictures, respectively. It is discovered that the accuracy of the train dataset is greater than that of the test dataset. The model, in particular, has accuracies of 95 percent for training data and 75.35 percent for testing data. A receiver operating characteristic (ROC) may be used to demonstrate the success of the statistical model of the classifier in further analyzing the influence of the HoG in the preprocessing stage. The ROC curve is given in Figure 8 when HoG is not used as one of the preprocessing processes. The accuracy of the micro-average ROC is found to be 69 percent, whereas the accuracy of the macro-average ROC is 50 percent. However, when HoG is added in the suggested technique, the ROC accuracies improve by up to 95 percent. In model, the test accuracy is lower than the train accuracy. This is because the trained model does not see the test data, and the train data is exactly the data used to train the model. Table 2 shows that the specificity and F1-score for the train dataset are both zero. This is due to the fact that there will be no TN case in analyzing the training data since the training pictures are limited to those that include a faulty area. In other words, non-defective photos cannot be used as training data. Furthermore, the specificity in the test dataset in Table 2 is 75.35 percent, indicating that the majority of the testing pictures are non-defective and that the model is capable of properly predicting them. To further assess the performance, confusion matrices for the train and test datasets are presented in Tables 4 and 5, respectively. In summary, a confusion matrix is a common statistic used to depict the categorization rate for each faulty or non-defective example. Table 4 shows that 97 of the faulty areas were appropriately identified. Although there are 100 photos in total, it should be noted that each image may have more than one flaws. There are 104 occasions where the model fails to recognize the problematic regions appropriately. However, because of the high TN value the total testing accuracy is reasonable.

```
[ ]  image_shape = (256, 4096, 1)
     model=build_model(image_shape)
     model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256, 4096, 1)]    0

 zero_padding2d_2 (ZeroPaddi  (None, 262, 4102, 1)     0
 ng2D)

 conv2d_4 (Conv2D)           (None, 256, 4096, 32)     1600

 bn0 (BatchNormalization)    (None, 256, 4096, 32)     128

 activation_4 (Activation)   (None, 256, 4096, 32)     0

 dropout_4 (Dropout)         (None, 256, 4096, 32)     0

 max_pooling2d_8 (MaxPooling  (None, 64, 1024, 32)     0
 2D)

 max_pooling2d_9 (MaxPooling  (None, 16, 256, 32)      0
 2D)

 conv2d_5 (Conv2D)           (None, 14, 254, 16)       4624

 bn1 (BatchNormalization)    (None, 14, 254, 16)       64

 activation_5 (Activation)   (None, 14, 254, 16)       0

 dropout_5 (Dropout)         (None, 14, 254, 16)       0

 max_pooling2d_10 (MaxPoolin  (None, 7, 127, 16)       0
 g2D)

 max_pooling2d_11 (MaxPoolin  (None, 3, 63, 16)        0
 g2D)

 flatten_2 (Flatten)         (None, 3024)              0

 dense_2 (Dense)             (None, 1)                 3025

=================================================================
Total params: 9,441
Trainable params: 9,345
Non-trainable params: 96
_________________________________________________________________
```

Figure 6.1: Performance of our proposed model

Performance metrics	Obtained result	
	Train dataset(%)	Test dataset(%)
precision	0.57	1.00
recall	1.00	0.62
F1-score	0.73	0.77
Accuracy	**0.75**	

Figure 6.2: Performance parameters for our proposed method

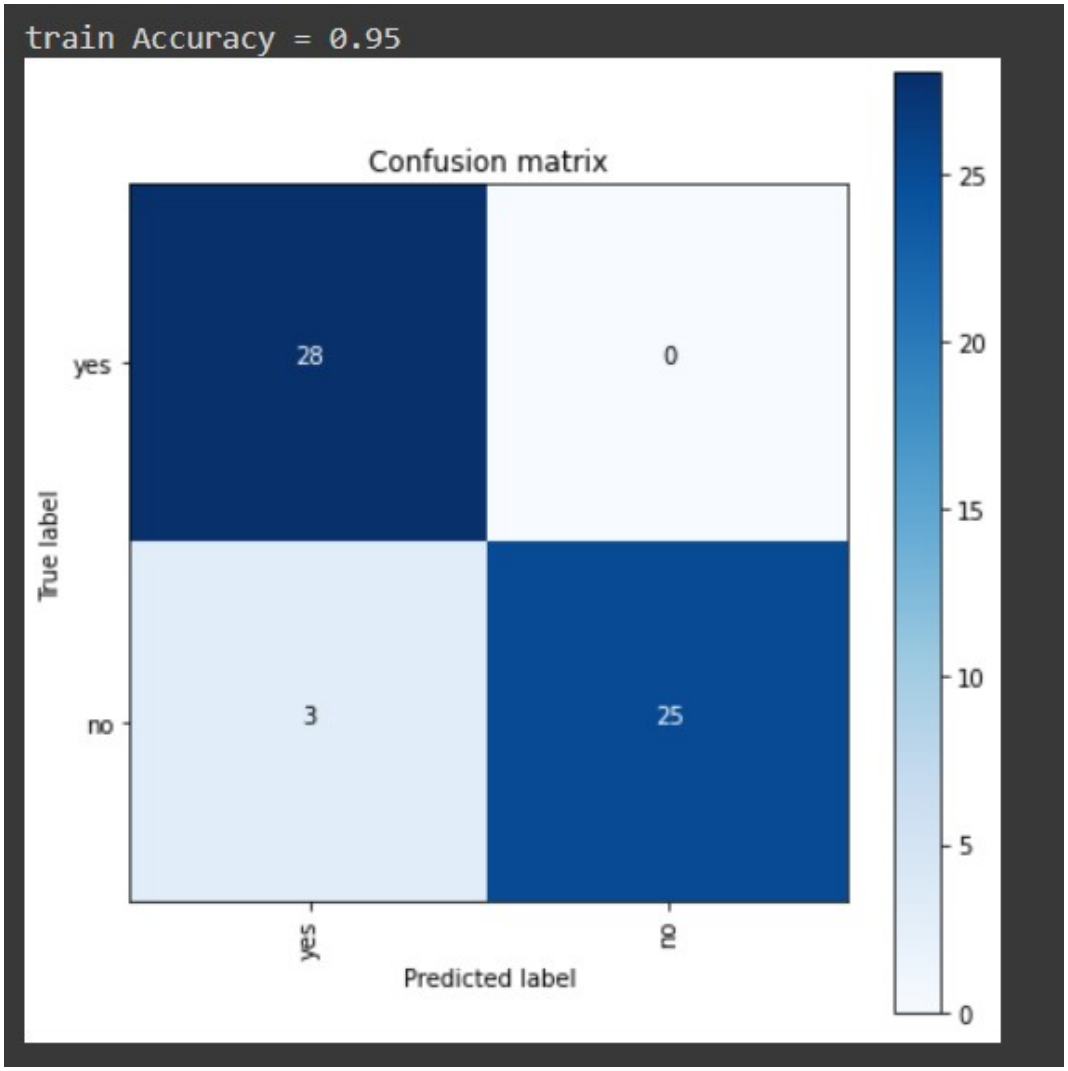


Figure 6.3: Confusion matrix for train accuracy

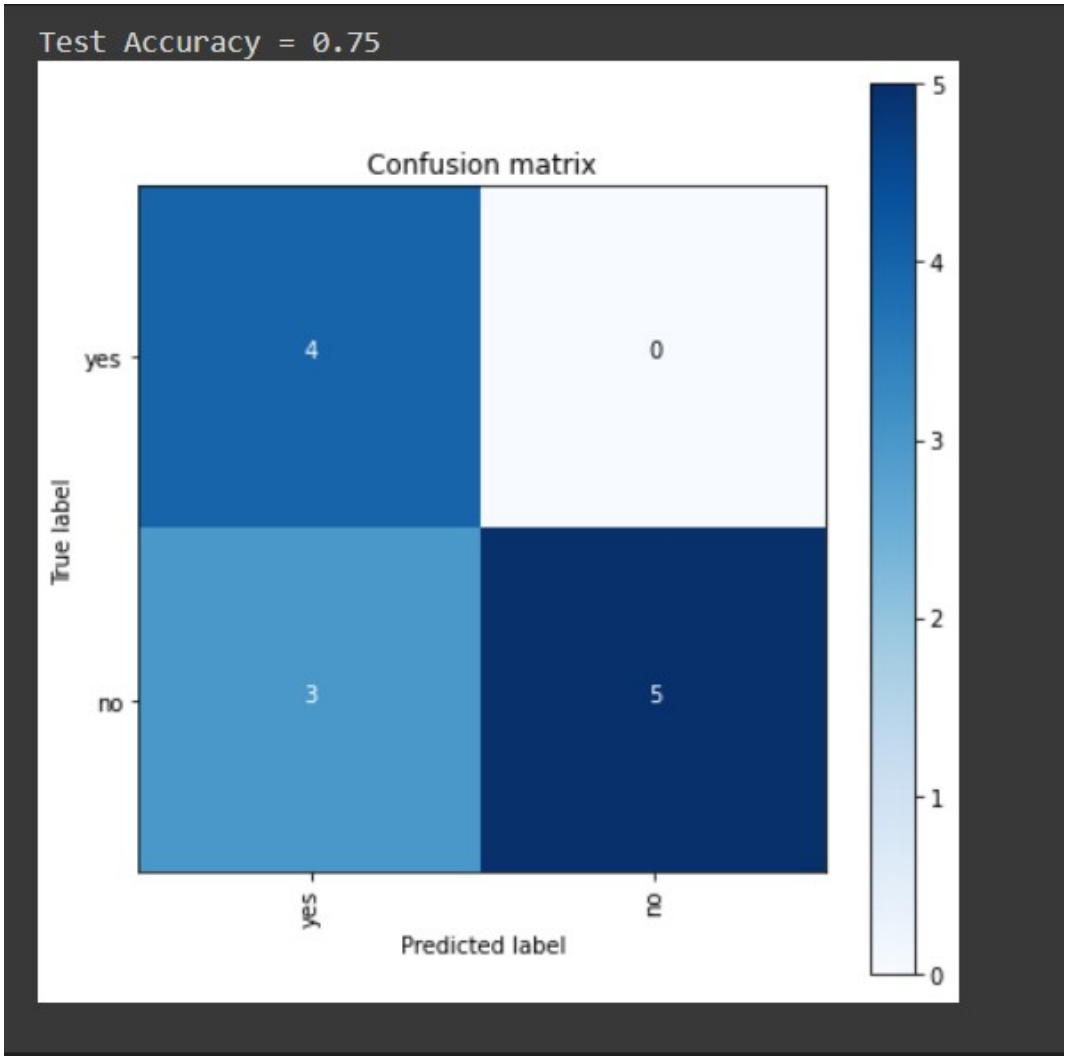


Figure 6.4: Confusion matrix for test accuracy

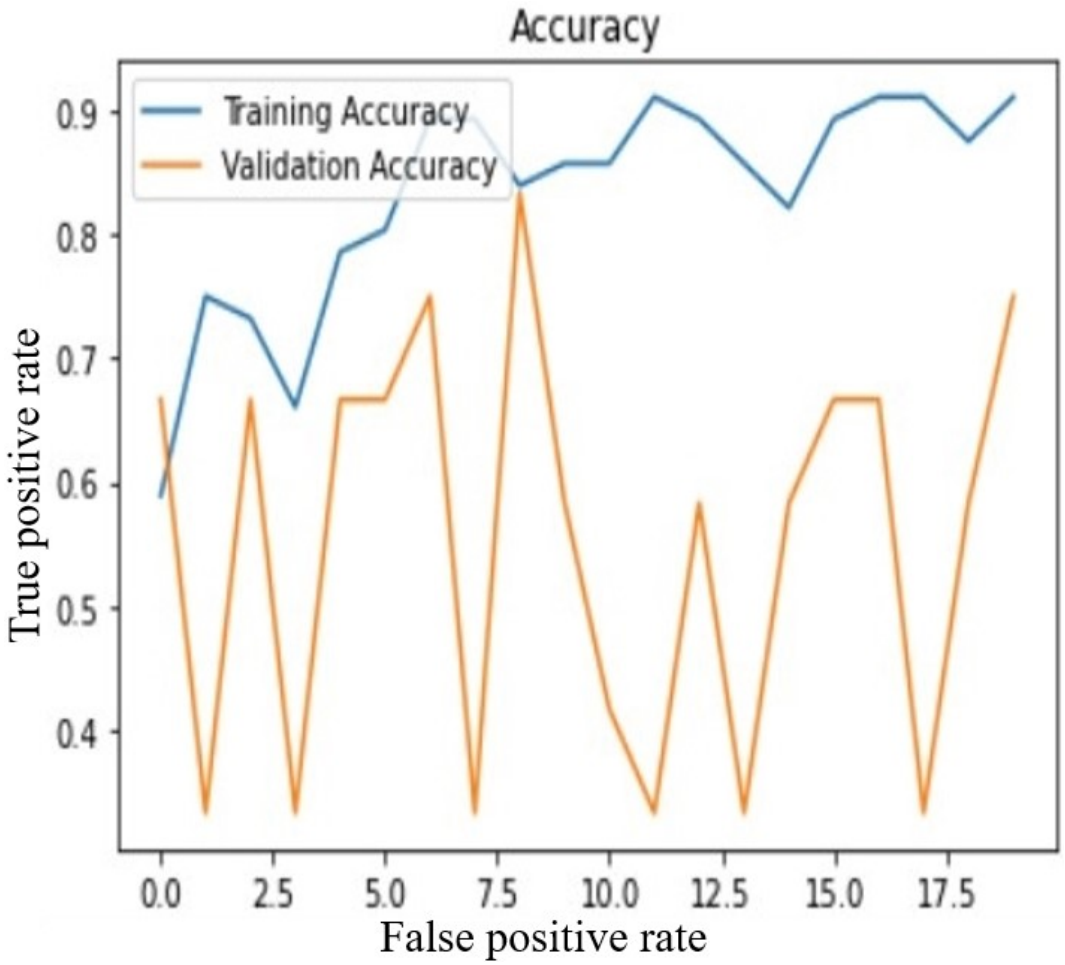


Figure 6.5: plotting for accuracy

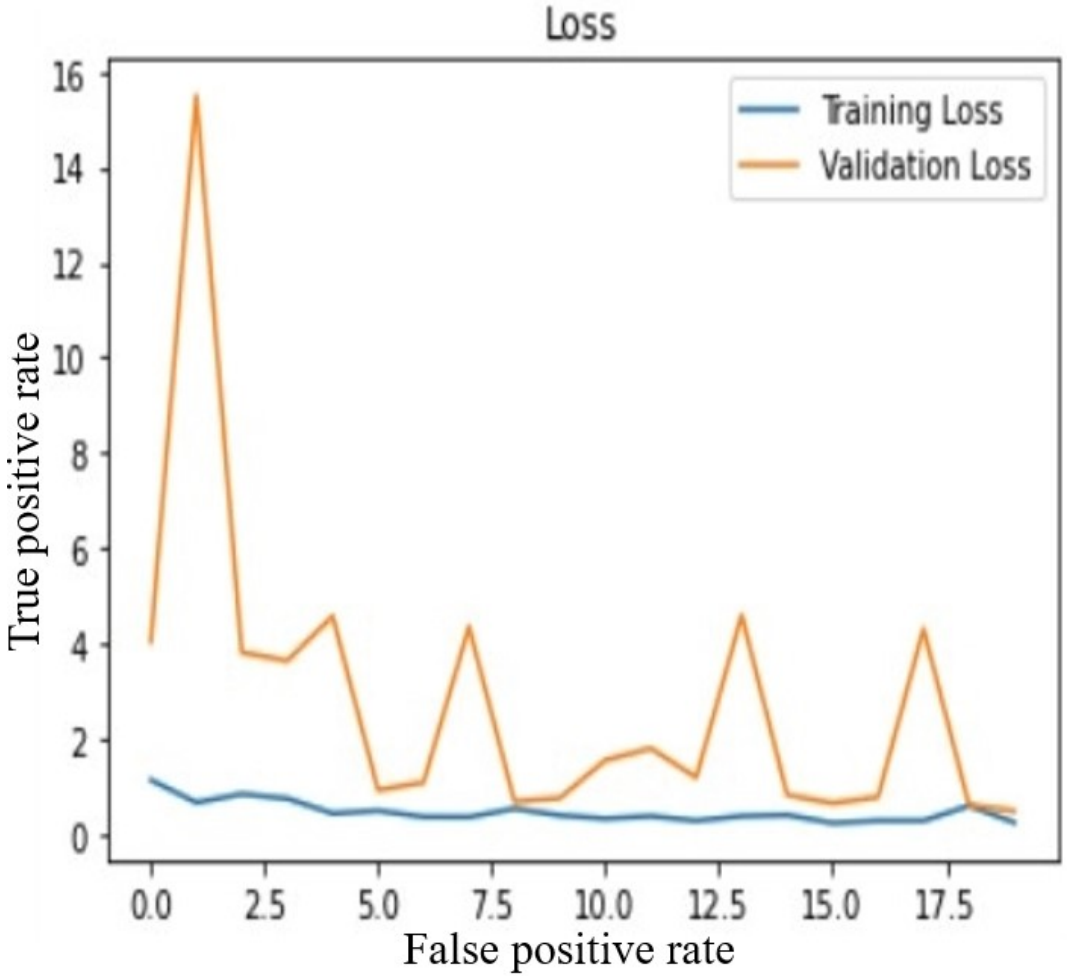


Figure 6.6: plotting for loss

CHAPTER 7

DISCUSSION

For automatic data generation, an important direction of future research will be to devise unique data augmentation methods for data generation. Also, it will be vital to develop automated annotation methods for facilitating weakly supervised learning as manual annotation would not be feasible on such a big scale. Provided ample grading data from domain experts, there is a need to design methods that can interpolate on that data and learn to quantify leather quality. This problem can be solved with effective data augmentation techniques, and especially generative adversarial networks may prove to be quite useful for this problem, due to their recent success in similar settings [76]. According to this proposal, the generator network generates a rating for the leather quality, where the discriminator network attempts to discriminate it in terms of whether it is from the human expert or the generator network. To counter the challenge of simultaneous multi-defect detection, it is vital to resort to CNNs, owing to their inherent capabilities and success in recent literature. CNN's are an inherent choice for simultaneous detection of multi-class categories, which is considered a fact after their triumph on large scale data having thousands of classes. Hence, the design and development of CNN based machine vision systems for robust visual inspection of leather and hides is an important future research direction. For challenging cases, where a leather sample contains several different types of a defect having a high degree of variability, the traditional CNN based methods may not obtain the best results. For industrially applicable deep learning in such scenarios, the problem of defect detection may be decoupled from the problem of leather quality grading. This can be achieved through an ensemble of CNNs. This project presented a methodical and detailed review on automatic defect detection system on finished leathers using deep learning A detailed review of the image analysis based leather defect inspection methods that generally belong to the class of heuristic or basic machine learning techniques was presented. Owing to the recent success of deep learning methods in various related intelligent applications, deep learning architectures tailored to image classification, detection and segmentation are discussed. A detailed review of the role of deep learning methods in general visual inspection applications was presented, where recent CNN architectures are classified and compared. The recent triumph of CNN based methods for general defect inspection is a driver for their application in leather defect detection/classification. Also, these CNN architectures can act as a source of guidelines for the design and development of novel solutions for leather defect inspection.

In this work, we highlighted the challenges that exist in the design and development of CNN based solutions for leather defect inspection, where ample training data, quantification of leather quality and high variability of defects are some of the greatest challenges. We also presented research directions for fellow researchers that should be investigated in the future to overcome these challenges and enable advancements in this very important area of research.

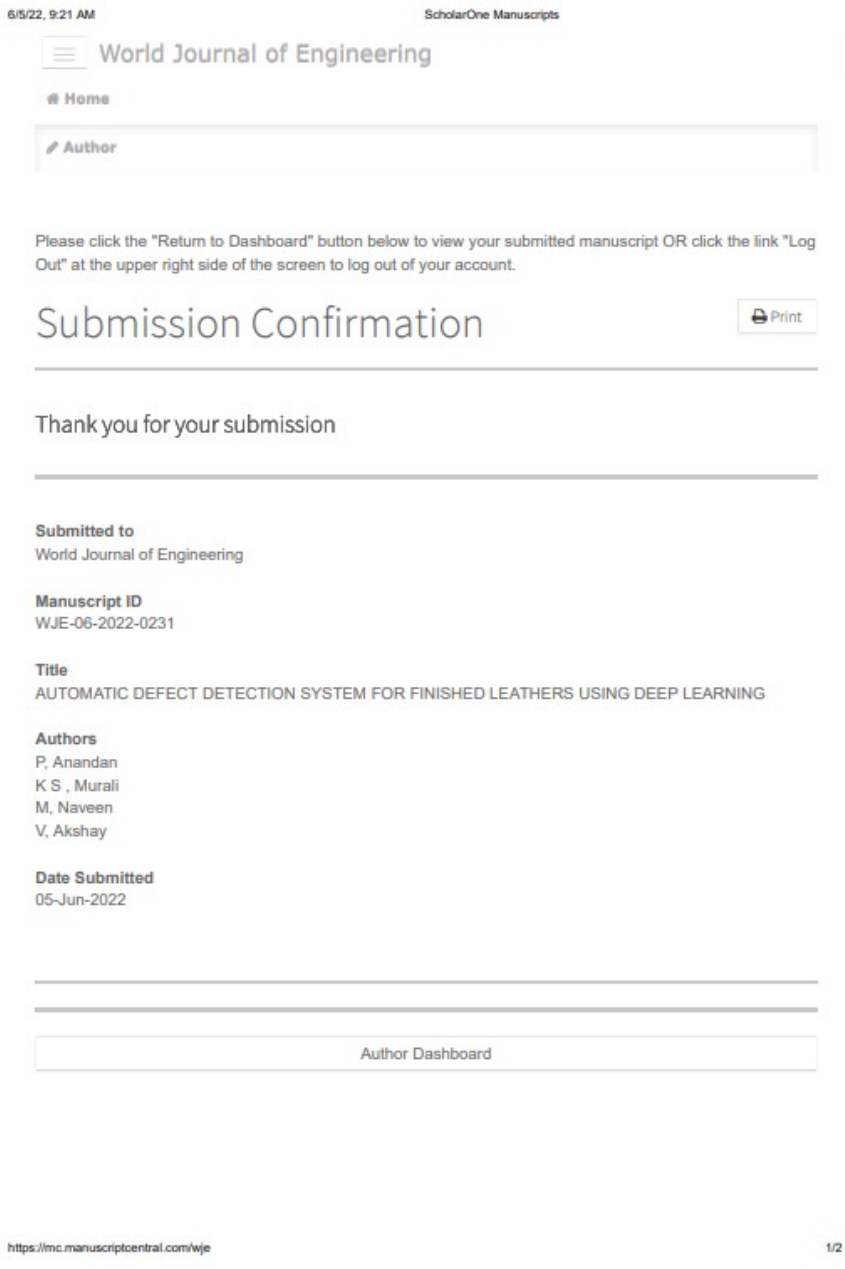
CHAPTER 8

CONCLUSION

This work provided a binary classification method to determine whether a leather picture contains a defect or not. Extensive tests and analysis were performed to validate the resilience of the suggested technique. Overall, when preprocessing approach and neural network classifier used, good results are obtained. As a consequence, when CNN is used as classifier, the best classification accuracy attained is 95 percent. Because this experiment was confined to the defect type, further study in this area might involve the construction of a classification or segmentation system to detect the defect kinds such as scratches, closed scratches, wrinkles, hovels, and itch. In addition to evaluating the fault type, the experiment may be expanded to test different leather types such as lamb, crocodile, and snake hides. In the future, a completely automated hardware setup consisting of the tasks of acquiring leather image patches, recognizing problematic spots, and laser cutting of the leather can be built.

CHAPTER 9

PUBLICATION PROOF OF PAPER



6/5/22, 9:21 AM ScholarOne Manuscripts

World Journal of Engineering

Home

Author

Please click the "Return to Dashboard" button below to view your submitted manuscript OR click the link "Log Out" at the upper right side of the screen to log out of your account.

Submission Confirmation

Print

Thank you for your submission

Submitted to
World Journal of Engineering

Manuscript ID
WJE-06-2022-0231

Title
AUTOMATIC DEFECT DETECTION SYSTEM FOR FINISHED LEATHERS USING DEEP LEARNING

Authors
P, Anandan
K S , Murali
M, Naveen
V, Akshay

Date Submitted
05-Jun-2022

Author Dashboard

https://mc.manuscriptcentral.com/wje 1/2

© Clarivate Analytics | © ScholarOne, Inc., 2022. All Rights Reserved.
ScholarOne Manuscripts and ScholarOne are registered trademarks of ScholarOne, Inc.
ScholarOne Manuscripts Patents #7,257,767 and #7,263,655.

@ScholarOneNews | System Requirements | Privacy Statement | Terms of Use

REFERENCES

[1] Y.S.Gan,Wei-Chuen Yau,Sze-Teng Liong,and Chih-Cheng Chen "Automated Classification System for Tick-Bite Defect on Leather", Mathematical Problems in Engineering, Volume 22, Article ID 5549879 (2022).

[2] M. Aslam, T. M. Khan, S. S. Naqvi, G. Holmes, and R. Naffa, "On the application of automated machine vision for leather defect inspection and grading: a survey," IEEE Access, vol. 7, Article ID 176065, (2019)

[3] M. Jawahar, N. K. Babu, K. L. J. A. Vani, L. J. Anbarasi, and S. Geetha, "Vision Based Inspection System for Leather Surface Defect Detection Using Fast Convergence Particle Swarm Optimization Ensemble Classifier Approach," Multimedia Tools and Applications, vol. 80, no. 3, pp. 4203–4235, (2020).

[4]] S. T. Liong, Y. S. Gan, Y. C. Huang, C. A. Yuan, and H. C. Chang, "Automatic Defect Segmentation on Leather with Deep Learning," March 2019, https://arxiv.org/abs/1903. 12139,(2019)

[5] Y. Gan, S.-S. Chee, Y.-C. Huang, S.-T. Liong, and W.-C. Yau, "Automated leather defect inspection using statistical approach on image intensity," Journal of Ambient Intelligence and Humanized Computing, vol. 12, pp. 1–17, (2020).

[6] JIEHANG DENG,JIAXIN LIU,CHANGZHENG WU1,TAO ZHONG1 , GUOSHENG GU 1 , AND BINGO WING-KUEN LING 3 , (Senior Member, IEEE)," A Novel Framework for Classifying Leather Surface Defects Based on a Parameter Optimized Residual Network", volume 8. IEEE access (2020)

[7] MASOOD ASLAM, TARIQ M. KHAN, (Member, IEEE), SYED SAUD NAQVI , GEOFF HOLMES, AND RAFEA NAFFA,"Ensemble Convolutional Neural Networks With Knowledge Transfer for Leather Defect Classification in Industrial Settings",volume 8 IEEE Access,(2020).

[8] Prahar M. Bhatt, Rishi K. Malhan,Pradeep Rajendran, Shantanu Thakar"Image-Based Surface Defect Detection Using Deep Learning", Journal of Computing and Information Science in Engineering,volume 21,(2021)

[9] Edmilson Q. Santos Filho,Pedro Henrique Feijó de Sousa,Pedro P. Rebouças Filho2 · Guilherme A. Barreto,Victor Hugo C. de Albuquerque."Evaluation of Goat Leather Quality Based on Computational Vision Techniques" springer (2019)

[10] Sze-Teng LIONG, Y.S. GAN , Kun-Hong LIU, Tran Quang BINH, Cong Tue LE, Chien An WU, Cheng-Yan YANG , Yen-Chang HUANG. “Efficient Neural Network Approaches for Leather Defect Classification”,ARXIV (2019)

[11]] Murinto Murinto,Adhi Prahara,Sri Winiari,Dewi Ismi.”Pre-Trained Convolutional Neural Network for Classification of Tanning Leather Image”, RESEARCH GATE(2018)

[12]] Evaluation of Goat Leather Quality Based on Computational Vision Techniques,”Evaluation of Goat Leather Quality Based on Computational Vision Techniques”ARXIV,(2019)

[13]] SHIH-YU CHEN, (Member, IEEE), YU-CHIH CHENG, WEN-LONG YANG, AND MEI-YUN WANG, Surface Defect Detection of Wet-Blue Leather Using Hyperspectral Imaging, IEEE,2021.

[14]]Renato F. Pereira,Claudio M. S. Medeiros,Pedro P. Rebouc¸as Filho, Goat Leather Quality Classification Using Computer Vision and Machine Learning,IEEE,2018.

[15]]Murinto Murinto, Adhi Prahara,Sri Winiari,Dewi Ismi, Pre-Trained Convolutional Neural Network for Classification of Tanning Leather Image, RESEARCH GATE,2018.

[16]]Qingsheng Jiang, Dapeng Tan, Yanbiao Li, Shiming Ji, Chaopeng Cai and Qiming Zheng, Object Detection and Classification of Metal Polishing Shaft Surface Defects Based on Convolutional Neural Network Deep Learning, MDPI,2019.

[17]] Kaiming He,Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, international journal of computer integrated manufacturing ,2016