Transforming RMG Printing with Computer Vision: Achieving Unmatched Precision in Fabric Defect Detection

by

Rayhan Sharif Sadif
21101063
Towhidul Islam Pranto
20301215
Aadarsh Mishra
21101116
Mahatuk Islam
20241017

A project submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

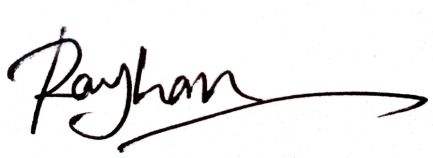
Department of Computer Science and Engineering
Brac University
September 2025

Declaration

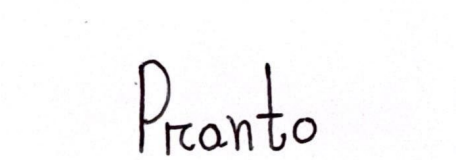
It is hereby declared that

1. The project submitted is my/our own original work while completing degree at Brac University.
2. The project does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The project does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

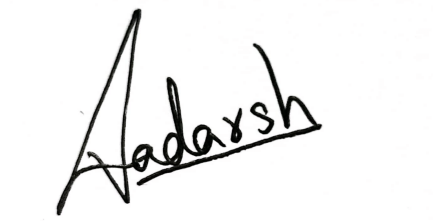
Students Full Name & Signature:



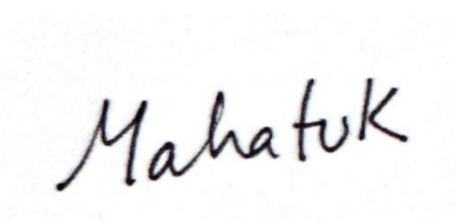
Rayhan Sharif Sadif
21101063



Towhidul Islam Pranto
20301215



Aadarsh Mishra
21101116



Mahatuk Islam
20241017

Approval

The project titled Transforming RMG Printing with Computer Vision: Achieving Unmatched Precision in Fabric Defect Detection submitted by

1. Rayhan Sharif Sadif (21101063)
2. Towhidul Islam Pranto (20301215)
3. Aadarsh Mishra (21101116)
4. Mahatuk Islam (20241017)

Of Summer, 2025 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on September 16, 2025.

Examining Committee:

Supervisor:
(Member)

Md. Sabbir Ahmed
Lecturer
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

Rafeed Rahman
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

This project investigates the application of advanced computer vision methods for detecting and sorting printing blights in the ready- made garments(RMG) sector, with a particular emphasis defect detection. The primary goal is to reduce the time and expense involved in manual checking by achieving the level of perfection in quality control procedures. This study evaluates recent advancements in computer vision operations for disfigurement discovery, relating the limitations and challenges in current methodologies. Our research leverages state- of- the-art im- age recognition algorithms, including convolutional neural networks (CNN) and deep learning models like Simple CNN, MobileNet and EfficientNet to accurately identify printing defects. The primary areas of focus include the development of a comprehensive dataset of publishing blights, the creation of robust image process- ing channels, and the integration of these systems into existing RMG product lines. Additionally, the significance of this exploration lies in its implicit to transform the quality control process within the garments industry. By automating disfigurement discovery, we aim to markedly reduce the reliance on human or manual examination, therefore adding product effectiveness and lowering costs. Moreover, this advancement will not only improve the overall quality of the products but also provide a competitive advantage to manufacturers in the global demand. Our find- ings show how we can achieve advanced discovery rates and accuracy compared to conventional styles, in this manner streamlining the quality control process. Models like Simple CNN, MobileNet and EfficientNet using epoch value 10 provided maximum levels of accuracy, scoring 85.8%, 97.6% and 98.47% respectively. Our future exploration will concentrate on refining these algorithms and examining their scalability and rigidity to colorful types of publishing blights and garments.

Keywords: Computer Vision, RMG Printing, Simple CNN, MobileNet, EfficientNet, Quality Control, Image Recognition, Productivity, Cost Reduction.

Table of Contents

List of Figures

Chapter 1

Introduction

In the modern days, the production of clothing is being replaced by new technology. Additionally, two fields that are gaining popularity in the apparel sector to enhance quality control are computer vision and artificial intelligence (AI). A key component of maintaining good standards in the ready-made garment (RMG) sector is the accuracy of colors and designs. Generally, quality inspections have always been conducted by human beings. However, this approach turned out to be unreliable and less accurate. And this is where computer vision and artificial intelligence are useful. Along with accuracy and cost-effectiveness, it guarantees a quicker inspection process.

In order to enhance the detection of fabric flaws in RMG printing, this study focuses on deep learning, more especially on convolutional neural networks (CNNs) models. With manual checks, the possibility of human errors while checking fabric flaws like color mismatches or design mistakes are frequently overlooked. The goal of the project is to increase production efficiency and decrease material waste by using AI to automate this process. These technologies help save time and costs by identifying flaws early on in the production process.

The goal of this project is to create a more reliable system for quality control. The system will also be flexible enough to handle different kinds of cloth textures and printing problems. This technology would lower the possibility of man-made errors while raising the standard of the final product. The clothing sector may see major improvements in manufacturing speed, accuracy and cost reductions through the integration of AI into the manufacturing control process.

1.1 Problem Description

According to the study, the current methods for detecting errors in manufacturing processes like 3D printing and textile printing are unable to maintain up with the rapidly changing requirements of the sector. The study also draws attention to the difficulties posed by material variation and the complexity of flaws, difficulties that conventional manual inspections are unable to adequately address. Because of this, automated solutions that can identify and fix errors in real time are desperately needed in order to increase production efficiency and accuracy. Using these, we will also be able to notice changes in productions accuracy and efficiency.

Among the various problems in defect detection we faced during our research, the most important one was the different types of fabrics having different types of print-

ing defects due to differences in fabric material or texture. Moreover, in some cases the differences in fabric textures makes it more difficult to detect the defects.

Furthermore, we can not ignore the fact that the traditional process of manually detecting defects in garments is often slow and prone to human errors which also results in not having the kind of end product which we expect. To keep up with such upcoming complexity, automated solutions are a must to increase the defect detection process.

Considering all the things mentioned above, the way these sectors production processes have developed highlights how critical it is to have integrated systems that can automatically identify defects. Additionally, to identify problems, such technology would allow for real-time adjustments during output, which will result in improved quality and less waste while also being quick.

1.2 Research Objective

The main goal of our project is to meet the growing demand of improved quality control. Throughout the whole production phase, we try to improve fault detection. To achieve this, we are relying on advanced computer vision and artificial intelligence (AI). To automate the processes, we are opting for Real-time feedback mechanisms and deep learning approaches. By integrating these mechanisms into the system, it will be able to enhance the detection accuracy as well. To deal with this, Convolutional neural networks (CNNs) are one of the models that can help us to fulfill our goal. This model can ensure that the overall output accuracy is improved. Moreover, it can also lower the material waste along with reducing human error. Our primary aim for the following study are:

- The first primary goal is to create a convolutional neural network (CNN) model according to the requirement. This model will be optimized to handle the various fabric textures and defects. The improved CNN model will have more accurate outputs of detecting defects compared to the VGG16 and VGG19 which are used currently.

- In order to reduce material waste and increase overall print quality, we are refining and optimizing a deep learning-based defect detection system. Specifically, we are trying to improve the precision and recall metrics in real-time monitoring and parameter modification.

- In order to decrease human inspection mistakes and improve the speed and accuracy of quality assessments, it is intended to automate the defect identification procedure during the garment cutting phase utilizing advanced image processing and object detection algorithms.

- The aim is to include feedback mechanisms into AI-driven defect detection systems, which provide dynamic modifications based on real-time data. This would improve the manufacturing processes flexibility and efficiency in various industries.

Chapter 2

Literature Review

In recent years, significant advancements have been made in the field of computer vision, particularly in defect detection and sorting within the textile and garment industries. This review discusses the latest research on deep learning techniques and transfer learning. The review also highlights the effectiveness of these models in terms of improving the color accuracy. Dealing with the design precision is another sector in RMG (ready-made garment) printing which is highlighted in this literature review.

The paper published by Singh et al.(2023), mainly focused on understanding the role of AI models and computer vision to modify the structure of manufacturing industries[21]. It highlighted the importance of how AI based technologies like artificial neural networks, enhanced automation, increased accuracy, and improved decision making can reshape the structure of manufacturing processes. For this, approaches like convolutional neural networks (CNN) and recurrent neural networks (RNN) were used to detect trends, outliers etc. Furthermore, the study also emphasizes how implementing neural networks can achieve the goal of improved production efficiency. The authors have highlighted a number of case studies which tells us how AI integration can impact quality assurance and predictive maintenance. The application of AI which was used in this study brought a promising improvement in overall productivity and correction. This indicates that a major change is possible to the existing system that weve been following in manufacturing processes.

"The study in[9] discusses how computer vision and algorithms were used. It explains how color information was integrated into a 3D object recognition system. This was achieved through image processing techniques. The study used algorithms such as color thresholding and segmentation. The sole purpose of using these algorithms is to detect and distinguish colors effectively. These methods were tested in different environments and provided promising results in terms of color-based object recognition. In addition, they also proved useful for improving human-computer interaction as well. This could benefit areas like automated vehicle tracking and medical imaging in the future.

Aydin et al. (2022)[8] studied fabric and production defect detection in the garment sector. This was achieved by applying a data mining approach. The study was done using models like decision trees, support vector machines (SVM), and k-nearest neighbors (KNN) techniques. The primary goal was to localize the problems on the fabric during the multiple stages production.The system was trained to identify diverse types of faults, including tears, stains, and defective patterns, through image processing and data analysis. The study also indicated better performance metrics

in defect detection using the proposed system as opposed to using the manual process, thus increasing the quality and efficiency in cost. Besides, addressing quality problems using data mining together with image processing seems to bring in freshness in the field. A theology direction is introduced by focusing on deep learning models for more robust and flexible defect detection tools based on the assumptions put forth.

Chen et al. (2019) [15] developed the image registration methods to the level of automatic defect detection systems applied to the region of printed garments. In this case, the use of contour point distribution and edge gradient analysis improves the defect identification process in terms of accuracy and speed, particularly with high-resolution images. In their work, they focused on how the use of machine vision could improve the quality controlling measures when printing. Besides, they have built a deep-learning based fault detection technique for knitted fabrics that utilizes the 4 point system and Isl-Knit benchmark dataset in their work. The model they worked on gave results in detecting the differences of fabrics. Also, these results helped them to have error free defect detection and solve technical problems immediately. For fabric quality assurance and error detection in fabrics, uses of deep learning approaches are proved to be highly effective.

Villalba-Diez et al. (2019) stated that it is possible to use deep neural nets to detect detects that exhibit an average accuracy of 98.4 percent while cutting down human effort and cost of production. The importance of using computer vision surveillance systems for fabric alignment and print accuracy enhancement during production has also been noted. Further, in the recent past, RPDNet has been reported which applies CNNs and pattern horizontal structure for the improvement of anomalous detection. The research has also discussed the defect detection systems that are based on computer vision and artificial neural networks (ANNs). The method improves automation of the classification of defects across different fabrics in the same fabric type. Furthermore, it has been underlined the importance of investigation into unsupervised learning concerning model generalization improvements for mass production. [2]

According to study by Shahrabadi et al. (2022), automation works with a higher degree of reliability than manual operators, reaching a percentage above 98 with the use of models such as CNN, AlexNet, and VGG16 in the recognition of defects like holes and stains. Similarly, a deep learning based model which is described as able to discriminate between various fabric defects with a percentage accuracy of over 95 is also presented thus also bringing to light the effectiveness of efficiently trained CNN models. Furthermore, the incorporation of computer vision with the concept of Industry 4.0, explaining how effective imaging methods could help improve productivity and decrease the number of defects in the production of product elements. In sum, while these works confirm the improvement of accuracy and efficiency of processes as a result of their studies, it however provides a caveat of needing to continue the investigation of practical applications of synchronization for example, between AI and IoT, for more effect.[11]

Paraskevoudis et al. (2020)[6] have proposed a method of detecting defects in 3D printing processes that is based on artificial intelligence and computer vision that operates in real time. To address this problem, they decided to focus on stringing

variation by implementing a Deep Convolutional Neural Network that was fed with images showcasing these defects. The model was integrated into production with the objectives of monitoring defects through video feeds and allowing to change print parameters on the go. Setting an Intersection over Union (IoU) threshold of 0.4, the method obtained favorable results, precision was 0.44 and recall was 0.69. This approach facilitates repairs that can be done on the spot enabling better identification of defects while minimizing wastage of materials as well as enhancing the quality of prints.

Chakraborty et al. (2020) discussed the detection of defects on printed fabrics using CNN. It is a highly complicated assignment because different fabric textures possess many distinct types. The objective of this research was to develop a specialized database of print fabrics followed by deep CNN applications. In special defect detection including spot mismatches and print errors. A CNN model was developed through 380 fabric images in their study which led to evaluation of performance against already established VGG16 and VGG19 models[7]. The research group reported their CNN achieved 48.5% accuracy identification. The VGG16 attained 62.28% accuracy and the VGG19 reached 60% accuracy. It provides information about the automated real-time defect detection in the textile manufacturing industry.

Moreno et al. (2017)[3] developed the FabricVision, program which employs computer vision technology. It detects fabric defects during the assembly cutting process. The system employs image capture together with object detection and segmentation techniques. It automates manual inspection procedures which results in increased efficiency and accuracy. The system uses a Raspberry Pi 3-based image processing system and operates through histogram of oriented gradients approach. The gradient method helps enhance detection accuracy of defects. The inspection processing speed escalated after implementing this method. Experimental findings reveal that the proposed algorithm successfully detects mistakes at a rate of 98.07%. This proves its high degree of effectiveness in identifying defects.

Saberironaghi et al. (2023) focused on using deep learning techniques for detecting defects in industrial products. It addresses challenges like noise, lighting issues, and complex textures. The paper covers different datasets used for defect detection, including the MVTec AD and Northeastern University datasets. The review explores supervised, unsupervised, and semi-supervised approaches, highlighting methods like CNNs, autoencoders, and GANs. These techniques are effective in detecting defects in real-time with high accuracy, such as 99% on the MVTec dataset. Deep learning models outperform older techniques, offering better accuracy and flexibility across industries like textiles and electronics. However, the paper also points out challenges such as imbalanced datasets, small sample sizes, and the high computational demands of some methods[13].

The paper by Sneha et al. (2024) [16] presented a fabric defect detection system utilizing transfer learning, addressing the issue of limited labeled data in textile manufacturing. The system applies a pre-trained deep learning model, fine-tuned using a fabric defect dataset that includes various types of defects such as holes and color inconsistencies. However, the dataset is imbalanced, with 83 percent of damaged samples. The methodology includes data preprocessing, feature selection

and testing of the model. This helped to achieve high accuracy in detecting complex defects, especially in jacquard fabrics. Despite these advancements, the model is liable for bias due to class imbalance.Future improvements could focus on balancing the dataset, enabling real-time processing and integrating edge computing.

Liu et al. (2022) presented a real-time defect detection system in regards to digital fabric printing. The primary goal is to design a model which can identify common surface issues such as white silk, spots, and wrinkles etc. The system utilizes an enhanced Speeded-Up Robust Features (SURF) algorithm to achieve precise image registration along with feature matching. The system detects defects efficiently by using bidirectional matching and a differential algorithm. The method provided a 12 percent increase in detection accuracy in the testing results. This result proved to be 42.81 milliseconds faster in processing time compared to traditional template matching methods which is used generally. The system proved to be very suitable for industrial applications[9] as this system managed to secure a 98 percent accuracy rate [10]. However, the system fails to detect text-based defects. But this issue can be resolved by integration of Optical Character Recognition (OCR) techniques in the near future.

Alam et al. (2020) [5] introduced an automated quality assurance system especifically for printed textile images. It enabled effective defect identification through the implementation of graph matching and pixel computation algorithms. These algorithms maintained error margins within the range of 0.16 percentage to 8.4 percentage. This system employs color coordinate frameworks, edge detection methodologies and image segmentation techniques. These help to facilitate accurate defect recognition. By integrating an Automated Robot Vision System (ARVIS) along with Optical Character Recognition (OCR), the methodology exhibited a remarkable 88.9 percentage accuracy rate in quality evaluation. The research underscores the significance of minimizing human intervention in the domain of textile quality control. Hence, production expenses are reduced and enhancing operational efficiency. Investigations seek to refine algorithmic accuracy. It takes into account environmental variables such as lighting conditions.

Zhu et al. (2007) introduced a Pix2Pix GAN-based method to improve color prediction in digital printing of silk fabrics.This specifically addressed the unique challenges of the texture and reflective properties of silk [17]. Future improvements could include better dataset balancing. Real-time processing could also be a focus. Edge computing is another area which can be integrated.. By integrating textile-specific texture and color data, this method reduces manual testing and increases color fidelity. The research builds on previous studies on fabric texture and color perception, highlighting the importance of advanced chromatic evaluations such as CIEDE2000 in digital textile printing. The future work will focus on improving the model for wider textile applications.

Golnabi et al. (2007) [1] examined the design and use of machine vision systems in industrial automation and emphasized how these systems can increase the effectiveness and quality of products. The study outlines a modular strategy that allows for flexibility in response to a range of industrial demands by addressing picture capture, preprocessing, feature extraction, and classification. Real-time monitoring and item detection are enhanced by methods like 3D laser scanning and sophisti-

cated sensors. Robotic guiding, process control, and automated visual inspection all employ machine vision to cut down on errors and manual labor. It is suggested that combining AI with machine learning would enhance flexibility and pattern detection in the future.

From the above discussion, advancements in defect detection for textile manufacturing and 3D printing have shown the effectiveness of deep learning techniques, especially Convolutional Neural Networks (CNNs). Studies using models like VGG16 and AlexNet have achieved high accuracy in identifying fabric defects, with some reaching up to 98.4 percent. In 3D printing, CNNs help detect defects like stringing in real-time to improve print quality. Systems such as FabricVision, powered by a Raspberry Pi, have automated textile defect detection with impressive success rates. Despite challenges like imbalanced datasets, approaches like transfer learning offer potential for improving quality control in manufacturing

Chapter 3

Methodology

The workflow of our proposed approach for transforming how RMG printing is carried out based on modern computer vision starts with data gathering, by using high quality with defective and non defective garment images. Firstly, the data is augmented and prepared by pre-processing. This ensures compatibility for accurate analysis. Special colored patterns are used to train the system. It also uses a custom model to detect color mismatching or design differences. To determine the color accuracy, the model validates with the test data set . In terms of design alignments; it checks for misalignments or elements that are missing. After this procedure, garments are divided into two categories. Either the product can be pass or it can be reject. The method helps us to achieve maximum accuracy along with long term printing quality

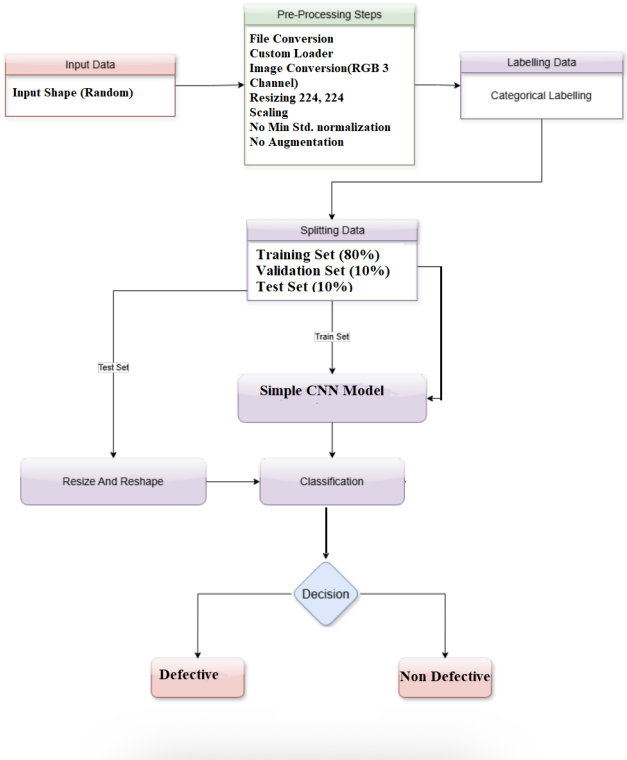


Figure 3.1: Workflow

3.1 Data and Dataset Description

In this part, we are providing infromation about dataset description, data sample, and dataset pre-processing.

3.1.1 Dataset Description

First, we have collected fabric dataset from different resources, which were then merged to create a comprehensive balanced dataset for analysis. The first source [18] includes a custom dataset of fabric defects was constructed, utilizing images captured from garments with various imperfections. The dataset comprises 2468 high-quality images and each image to ensure quality.

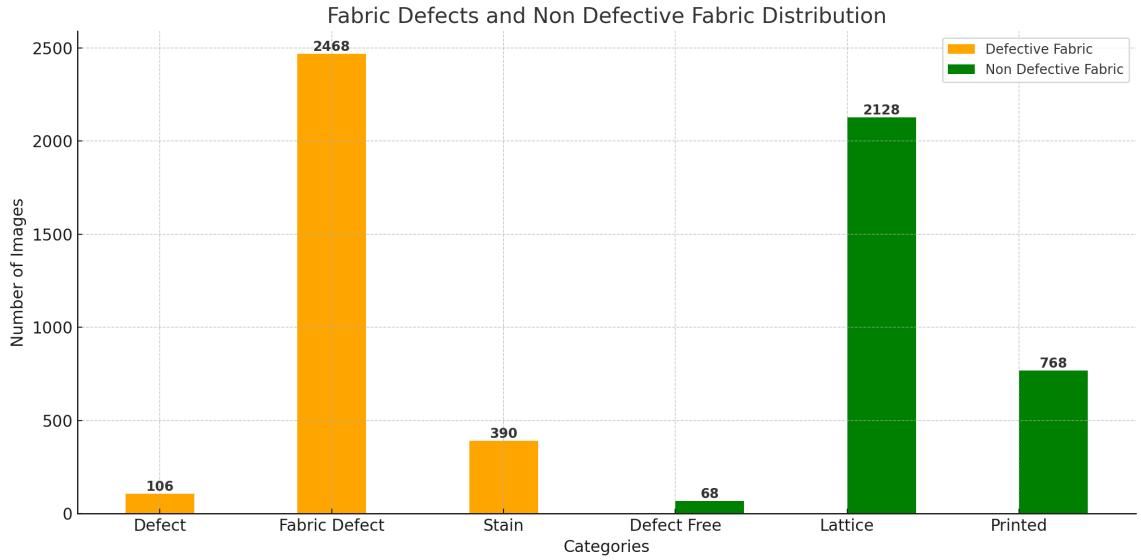


Figure 3.2: Distribution Fabric Defects And Non Defective Fabric Images

The second source was a [12] fabric dataset named Non Defective Fabric Image Data for textile image classification. The dataset contains alots of images like lattice, pattern, solid and stripe but we have used 2896 images where pattern has 768 and lattice has 2128 pictures. The third source [4] datasets is the AITEX Fabric Image Database contains defect images, mask images and no defect images, we have used 106 defect images, which is centered on fabric defect detection. It includes various textures, colors, and types of defects. The last source [19] which contains stain have 390 images and defect free have 68 images. For our study we have merge this dataset in Defective Fabric 2964 images and Non Defectice Fabric 2964 images then we have total 5928 Fabric images.

3.1.2 Data Sample

Below are images of different fabric of readymade garments industry. The pictures display the images of defected fabric and non defected fabric. As we can see defective fabric sample contains broken button and stain on the fabric and on other hand non defective fabric sample no defect on fabric. These samples demonstrate the difference in quality control and manufacturing processes within the textile industry. Defective

fabrics may result from improper handling, machine errors, or material flaws during production, whereas non-defective fabrics maintain high standards of quality and finish.
In addition to the broken buttons and stains, other potential defects in fabrics may include uneven dyeing, fabric pilling, tears, or misaligned seams. On the contrary, the non-defective fabrics may be free from such imperfections, ensuring a longer lifespan and better appeal in the final product.

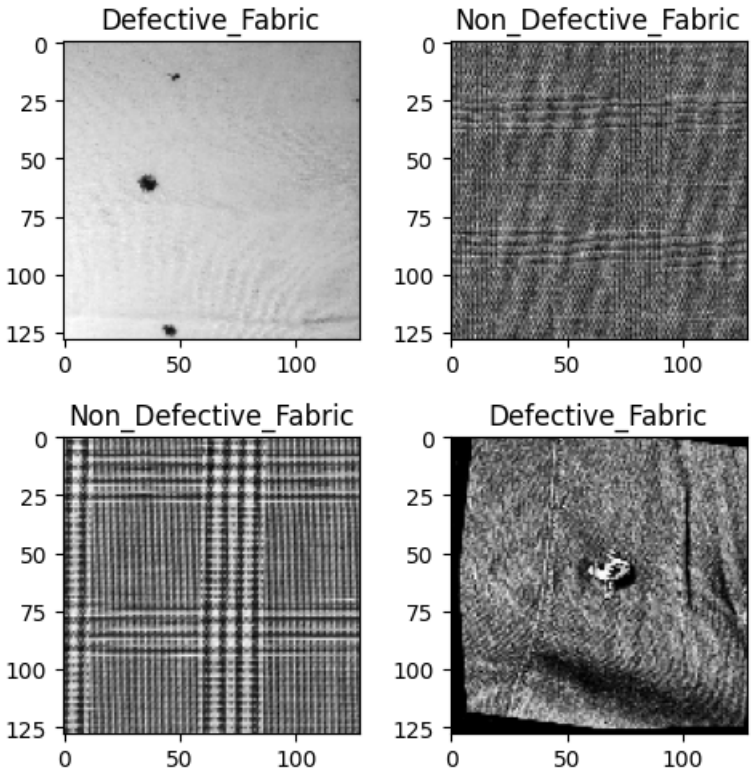


Figure 3.3: Data Sample

3.1.3 Dataset Preprocessing

This code will be used to preprocess and prepare a fabric image dataset that will be trained on the deep learning model, that specifically performs fabric defect detection. It also handles the dataset and makes transformations using PyTorch, a global deep learning framework. How the dataset will be loaded and accessed is defined in a class that inherits torch.utils.data.Dataset, FabricDataset. The image directory path and a list of the names of the classes (which are to reflect the names of the defective fabric and non defective fabric) are all passed to the constructor of this class, along with optional transformations to the images. Using the directory structure of the image dataset, the image paths list and labels list fill respectively. It traverses through the directory and adds the path of the image along with the correct class label (0 as defective and 1 as non defective) to the lists by writing in each of the lists one by one, each record by record.
Opening a particular image and its label based on an index is under the control of getitem method. When an image is loaded, it is loaded in Python Imaging Library

(PIL) and converted to RGB image. Where specified transformations (e.g. resizing, conversion to a tensor) are to be done, they are done to the image. The logo and their label are subsequently brought back as a pair. In case a problem arises during opening the image, an exception is caught and the message is printed. Transformations in the code are also applied with the torchvision.transforms. It converts it to PyTorch tensors and resizes the images to 224x224 and scales the pixels to the range in PyTorch tensor require values to be within range [0, 1] Once the dataset is prepared, it instantiates DataLoader objects on the training, validation and test sets. In this case the train, test, split is not employed and Train/Validation/Test sets are created manually via random split with proportions 80/10/10. DataLoader helps in making sure that images can be loaded in batches, efficiently when training the model, and the training set is shuffled in order to avoid any form of bias to the data.
Lastly, structure and the dimensionality of the dataset is validated through the printing of the shape of one batch of images and the matching labels. The images are mapped in a group of 32 and the form of the images is ascertained to be [32, 3, 224, 224] which means 32 images, 3 color channels (RGB) and resized as 224x224 pixel. The dataset has 5928 total images with 4742 as training, 592 as validation and 594 as testing, making there a good balance between these subsets. The structure of the code is such that it can be applied to large datasets in an efficient manner and ready them to use a models to detect fabric defects.

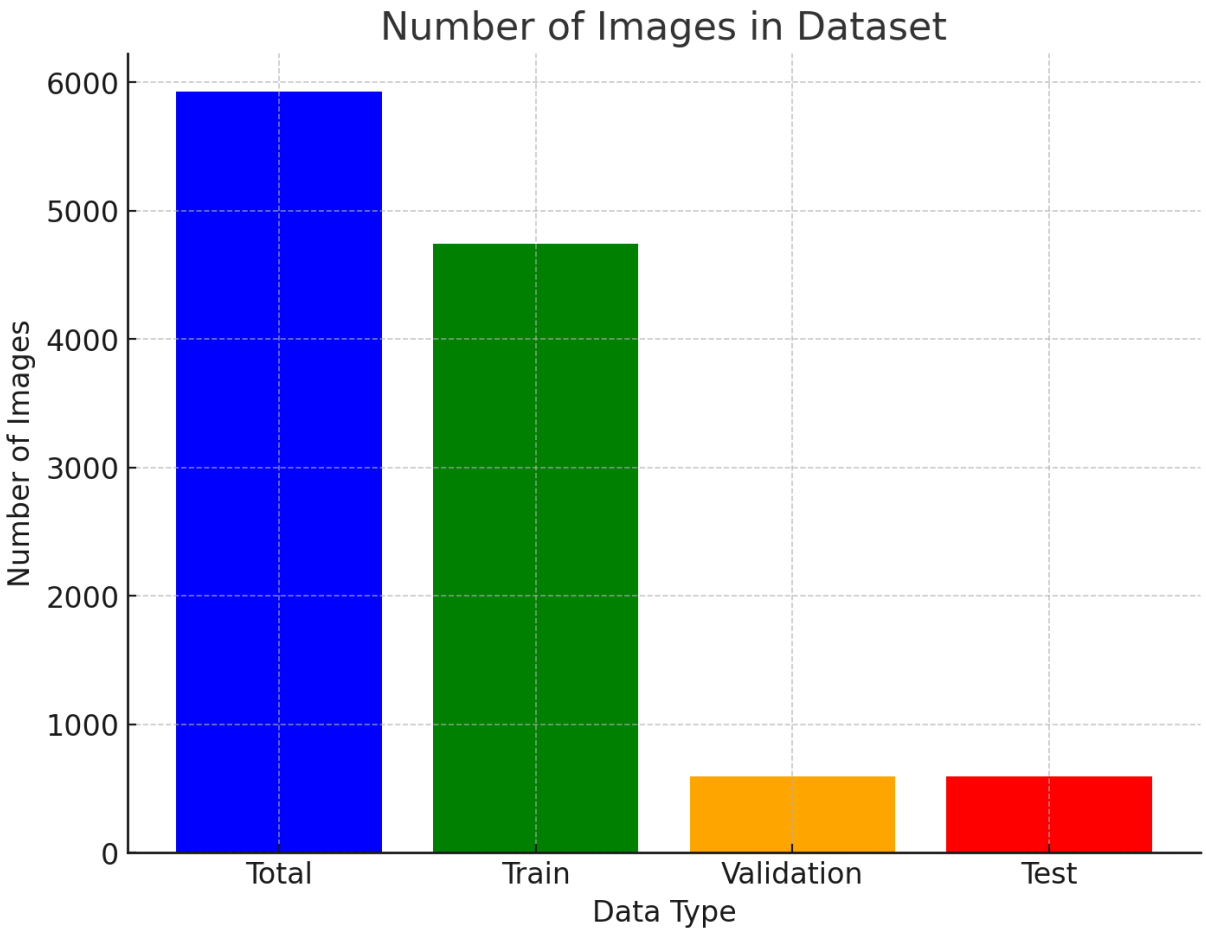


Figure 3.4: Train Test split of Dataset

3.2 Model Implementation

For this project we have implemented three models like Simple CNN, MobileNet and EfficientNet.

3.2.1 Model Architecture Using Simple CNN

Using the deep learning method, computers can be trained in a way that they learn to behave like human beings. It represents a type of machine learning, which is based on a three-layered neural network. Deep learning has made extensive use of CNNs to process visual data. A large portion of the computations is carried out by the CPU that is the main component of a CNN. After passing the processing stages, the CNN is able to identify the intended item and locate it after identifying the larger features or shapes.

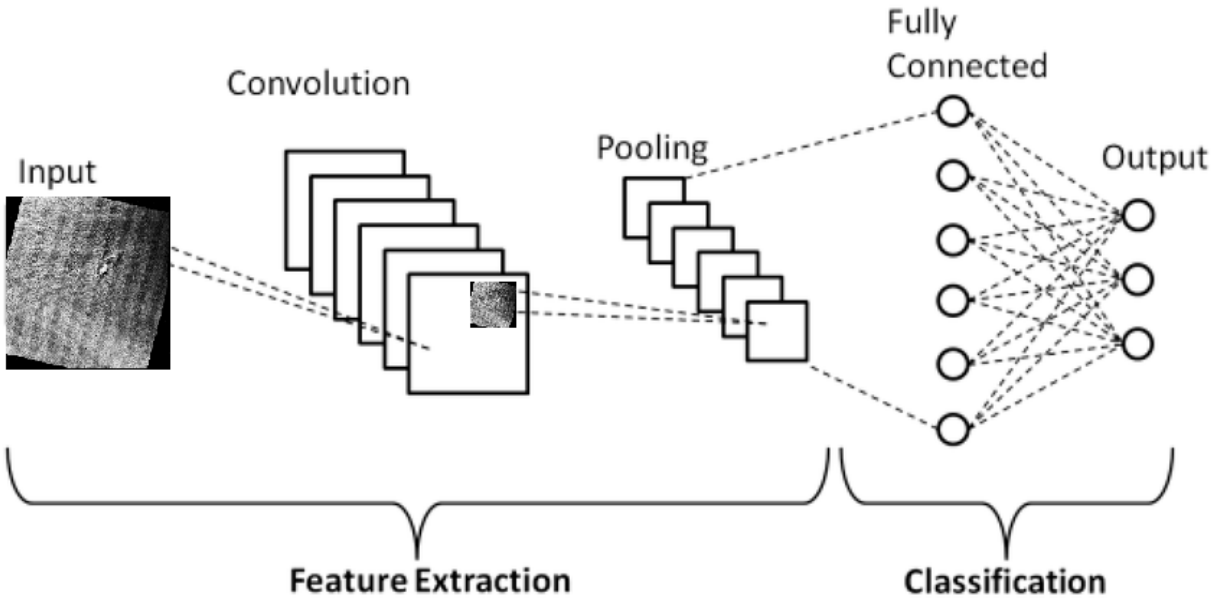


Figure 3.5: Working Process of Simple CNN

3.2.2 Proposed CNN Model

CNN models are created to work with data which can be expressed in grid-like form, including images. The given model will attempt to differentiate defective and non defective images using a deep multi-layered convolutional neural network (CNN). The CNN architecture consists of many layers: Convolutional Layers, Pooling Layers, Fully Connected (FC) Layers, and other components, such as Batch Normalization and Dropout layers. Each layer of the model is described as below:

3.2.3 Convolutional Layer:

Convolutional layer is an essential part of CNN that requires the use of filters (also referred to as kernels) to operate on the input images. The filters used in this layer will be smaller than the image size, and this enables CNN to identify significant features of the image, like edges, textures and patterns within the image. The convolution operation detects different characteristics of the image using multiple filters. The goal of this layer is to highlight relevant features, such as edges, blurring, and sharpening. For our model, the Conv2D layer from the torch library was used. In total, the model incorporates three Conv2D layers to capture different hierarchical features in the input images.

3.2.4 Batch Normalization:

To normalize the neuron activations of all the layers, batch normalization is used. The method is useful to stabilize and expedite to train the process by scaling and adjusting the input of one of the layers according to the mini-batch of current data. The operations carried out by this layer are:
It estimates the variance between the standard deviation and the mean of mini-batch. It uses a learnable scale (gamma) and shift (beta) to normalize the data, which enables the model to fit at training time. In such a way, by using batch normalization, the efficiency of learning is enhanced and the problem of vanishing or exploding gradients is prevented. The method is implemented following every convolutional layer of the model.

3.2.5 Max Pooling Layer:

To make the feature maps generated by the convolutional layers spatial (height and width) small, max pooling is applied. This layer operates by summarizing a block in the feature map by choosing the largest value in a specified window (e.g. a 2x2 window). Pooling serves two purposes:
Dimensionality reduction: It is used to reduce the number of features and therefore the number of parameters required to be trained.
Computational efficiency: Pooling can be used to decrease the computation cost during training by increasing the size of feature maps. Three max-pooling layers are used in this model with each image followed by a convolutional layer to down-sample the feature maps, and decrease the number of trainable parameters.

3.2.6 Fully Connected Layer:

The layer that directly comes after the convolutional and pooling layers is the fully connected layer (also referred to as the Dense layer), with every neuron in this layer having input to all the neurons in the previous layers. The flattening of the final convolutional or pooling layer is into a single dimension vector, which is then fed into the fully connected layer. The result of this transformation enables the network to acquire non-linear combinations of features. Our model includes:

- Flatten Layer: After the three BatchNormalization layer, a flatten operation is applied to convert the multi-dimensional tensor into a single 1D array.

- Dense Layer: There are two dense layers in the fully connected portion of the network. These layers enable the network to learn difficult patterns in the data by connecting all the neurons from the earlier layer to every neuron in the present layer.

- Dropout Layer: To prevent overfitting, we use one dropout layer. The dropout layer randomly deactivate a percentage (50%) of neurons during training, forcing the network to generalize better. The dropout probability is set to 0.5.

3.2.7 Proposed Model Summary :

The model architecture will comprise of a number of layers: Convolutional layers, Batch Normalization, Max Pooling, and Fully Connected layers. Convolutional Layers (Conv2d) are based on filters that identify features in the input image, which continues to gradually deepen the level of feature map. The use of Batch Normalization takes place following the convolution procedure to stabilize the training procedure. The layers of max pooling are apply to downsize the size of the feature maps and enhance computational efficiency. The result of the last pooling layer is flattened and inputted into Fully Connected Layers (Linear) where the model trains to categorize the features. Intermediate layers are used to apply dropout to avoid overfitting. The last layer gives out the binary classification outcome. The model contains 93,954 non-trainable parameters and it has no non-trainable parameters at all, and therefore it is a fully trainable network to be used in image classification.

Layer (type)	**Output Shape**	**Param #**
Conv2d-1	[-1, 32, 224, 224]	896
BatchNorm2d-2	[-1, 32, 224, 224]	64
MaxPool2d-3	[-1, 32, 112, 112]	0
Conv2d-4	[-1, 64, 112, 112]	18,496
BatchNorm2d-5	[-1, 64, 112, 112]	128
MaxPool2d-6	[-1, 64, 56, 56]	0
Conv2d-7	[-1, 128, 56, 56]	73,856
BatchNorm2d-8	[-1, 128, 56, 56]	256
MaxPool2d-9	[-1, 128, 28, 28]	0
Dropout-11	[-1, 128]	0
Linear-12	[-1, 2]	258
Total params		**93,954**
Trainable params		**93,954**
Non-trainable params		**0**

Table 3.1: Simple CNN Model Architecture Summary

3.3 Pre-Trained Models

In this part, we are going to tell about verison of the models like MobileNetV2 and EfficientNetB0.

3.3.1 MobileNetV2

MobileNetV2 [20] is a classification architecture that was designed by Google. It offers real-time classification features within computing constraints in gadgets such as smart phones. This adoption uses ImageNet transfer learning to our dataset. It is pre-trained on a large dataset, ImageNet, which allows it to recognize a wide variety of general features in images. The model consists of several layers, and the last layer (mobilenetv2.classifier) is a fully connected layer that outputs the classification result. Since MobileNetV2 was originally trained for 1000 classes in ImageNet, we need to modify the last layer to match the number of classes in our dataset (the number of fabric categories, such as "Defective Fabric" and "Non Defective Fabric"). This is done by replacing the original final layer with a new one that has an output size equal to the number of classes in our dataset (num classes).
After modifying the final layer, an Adam optimizer is set up with a learning rate of 0.001. The Adam optimizer helps adjust the weights of the model during training to minimize the loss and improve the models accuracy over time. This setup prepares MobileNetV2 to be fine-tuned on our specific fabric dataset for defect detection.

3.3.2 EfficientNetB0

EfficientNet is a family of convolutional neural networks (CNNs) [14] whose goal is to be highly performing and require fewer computational resources than the earlier networks. EfficientNetB0 is a deep learning network that has significantly different network scale. It does not randomly decide the depth and width and resolution of the network but adjusts all three of these in an equal measure using a single compound coefficient. This approach helps the model become more efficient by finding the optimal balance between depth, width, and resolution, rather than increasing any one dimension too much. This method allows EfficientNetB0 to achieve high accuracy while keeping the model small and computationally efficient. We are using EfficientNetB0, which is the smallest model in the EfficientNet family. It comes pre-trained on a large dataset called ImageNet, which helps the model learn general patterns that can be useful for many tasks. We then modify the final layer of the model so that it can predict whether a fabric is defective or not, based on our specific classes: 'Defective Fabric' and 'Non Defective Fabric'. This is accomplished by adjusting the output layer to the number of our classes (2 in this case). This is followed by the configuration of the model using loss function (CrossEntropyLoss) to classify with multiple classes and Adam optimizer to modify the parameters of the models in training. This setup is ready to train on our fabric images to detect defects.

Chapter 4

Result Analysis and Discussion

For result analysis, we are going looking into model performances and discuss about it.

4.1 Model Performance

For model, we got performances for: Simple CNN

4.1.1 Training and Testing on Simple CNN

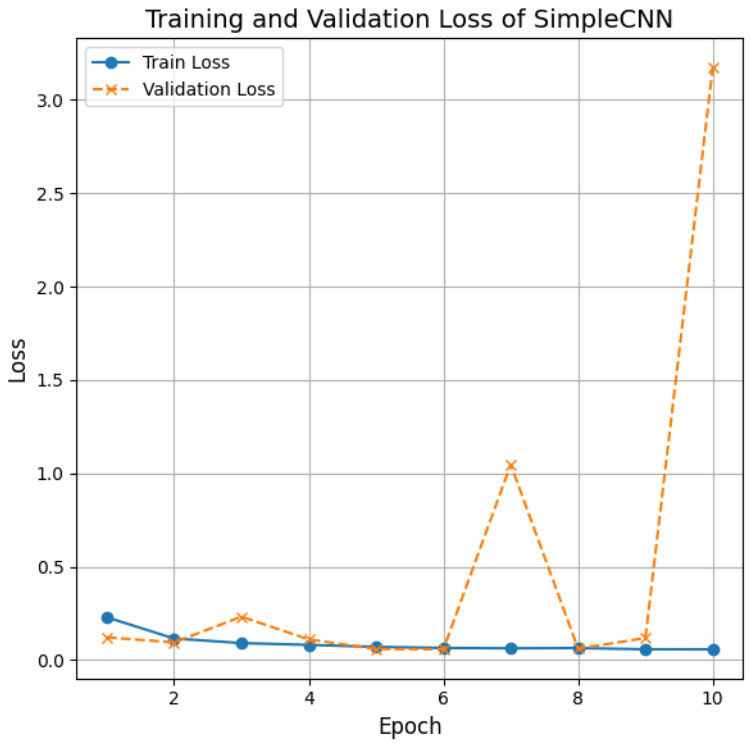


Figure 4.1: Simple CNN Train and Validation Loss

In this graph, training and validation loss of a SimpleCNN model with 10 epochs is presented. The blue line depicts the loss on training, which is decreasing with time showing that the model is training. The red and orange dashed line indicates the validation loss and it varies and peaks at epoch 6 indicating that this model is likely

to be overfitting. On the whole, the model has stronger performance on training data compared to the validation data.

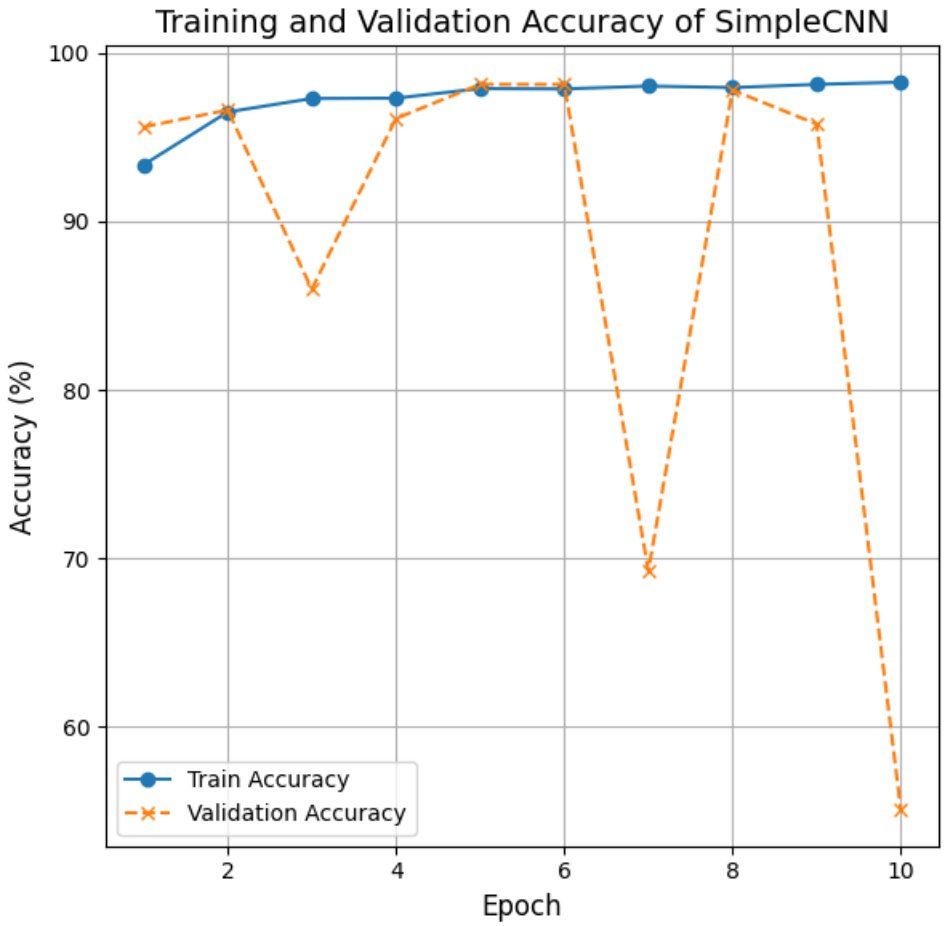


Figure 4.2: Simple CNN Train and Validation Accuracy

The graph indicates the accuracy of Training and validation of the SimpleCNN model in 10 epochs. The blue line refers to the accuracy of the training which remains high and consistent. The orangish line of the dash shows the accuracy of the validation which varies tremendously particularly in epoch number 6 implying that the model is not very good at generalization. On the whole, this model works well on train data but not on the validation data.

Class	Precision	Recall	F1-Score	Support
0	1.00	0.71	0.83	293
1	0.78	1.00	0.88	299
Accuracy			0.86	592
Macro avg	0.89	0.86	0.85	592
Weighted avg	0.89	0.86	0.85	592

Table 4.1: Simple CNN Model Evaluation Metrics

This is the matrix of the predictions of the SimpleCNN model on the test set. The matrix displays the quantity of correct and faulty yearnings of the two categories: Defective Fabric and Non Defective Fabric. The model was able to forecast 209

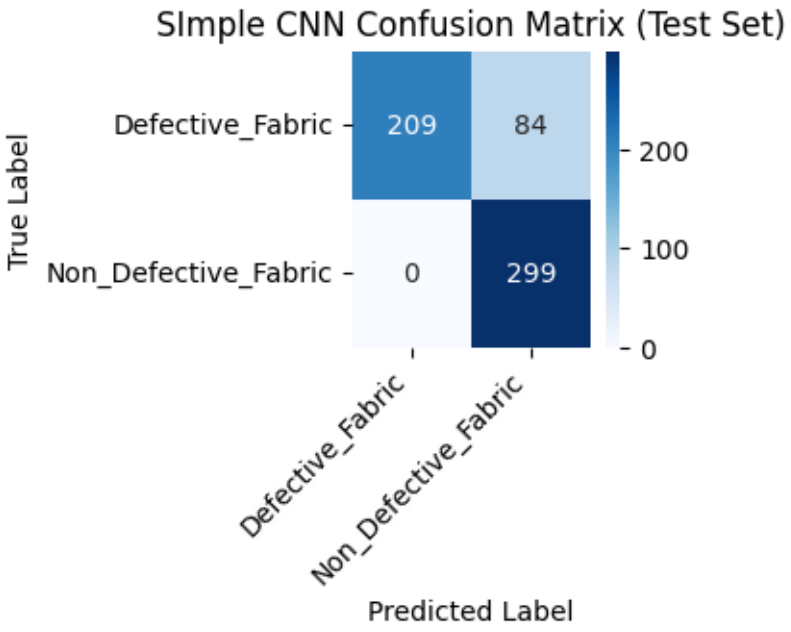


Figure 4.3: Simple CNN Confusion Matrix

defective fabrics as well as 299 non-defective fabrics correctly, whereas it classified 84 non-defective fabrics as defective. The defective fabric category does not have any false negatives.

4.2 Pre-Trained Model Performance

For model, we got performances for: MobileNet and EfficientNet

4.2.1 Training and Testing on MobileNetV2

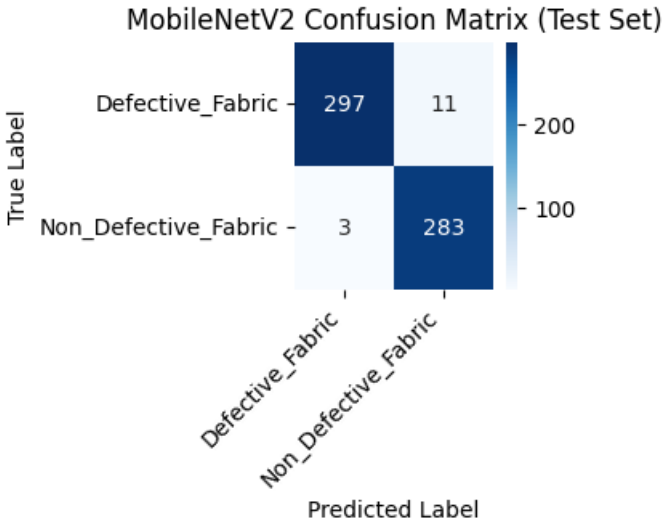


Figure 4.4: MobileNetV2 Confusion Matrix

This is a confusion table of the prediction of the MobileNetV2 model on test. The model also accurately forecasted 297 and 283 defective and non-defective fabrics respectively. It misclassified 11 defective fabrics as non-defective and 3 non-defective fabrics as defective. MobileNetV2 is generally performed well.

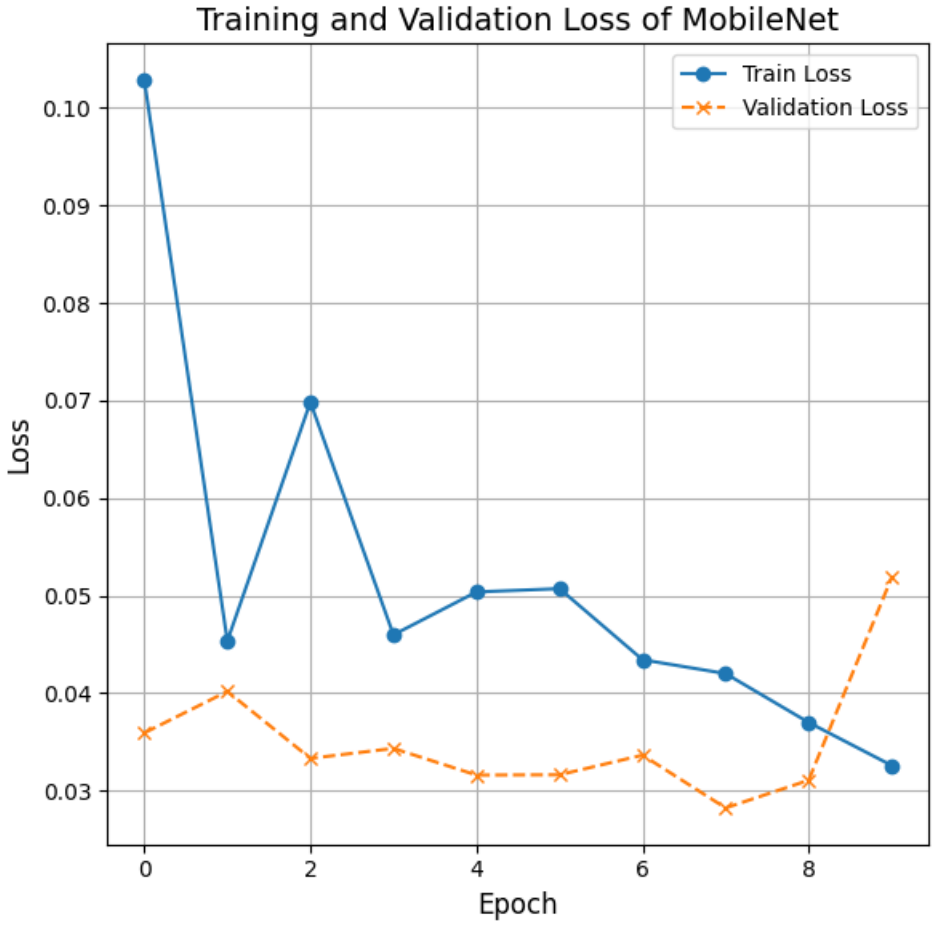


Figure 4.5: MobileNetV2 Train and Validation Loss

The above graph reveals train and validation loss of the MobileNet model during 10 epochs. The blue line is the loss during training and it decreases sharply and then levels off because the learning is good. The orange dashed line is the validation loss that varies more, with the highest values at the first three epochs, then drops. The overall model demonstrates a gradual decrease in train and validation loss.

Class	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.99	0.96	0.98	308
1	0.96	0.99	0.98	286
Accuracy			0.98	594
Macro avg	0.98	0.98	0.98	594
Weighted avg	0.98	0.98	0.98	594

Table 4.2: MobileNetV2 Model Evaluation Metrics

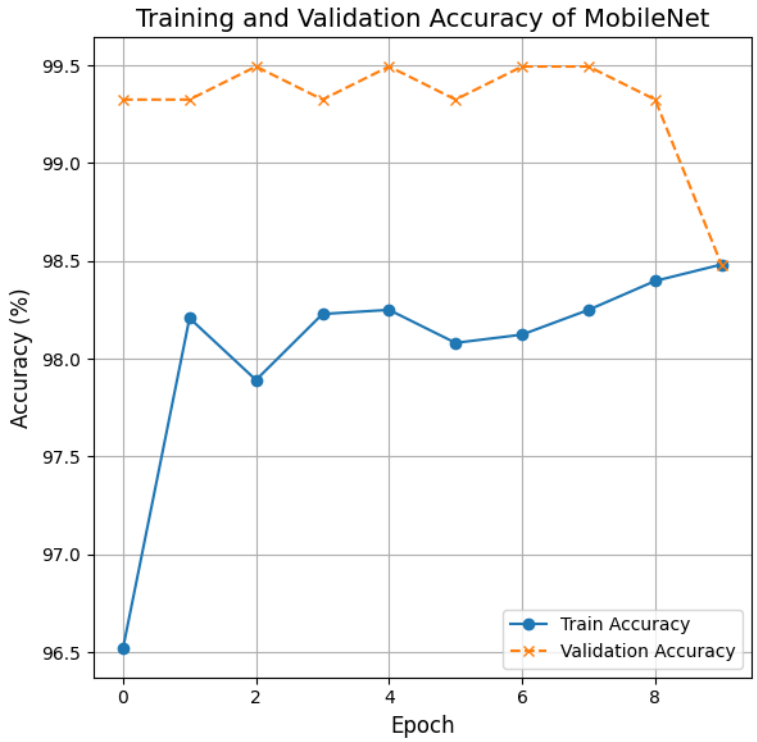


Figure 4.6: MobileNetV2 Train and Validation Accuracy

The graph below shows the validation accuracy and training of mobileNet model during 10 epochs. It has a blue line indicating the training accuracy in which it starts with 96.5 % and slowly increases to around 98.5%. The orange dashed line is a sign of the validation accuracy which initially has high value of approximately 98% but varies and decreases slightly at the end. This means that the model would be good when trained on the training data but it would be inconsistent when applied to the validation data.

4.2.2 Training and Testing on EfficientNetB0

Class	Precision	Recall	F1-Score	Support
0	0.97	1.00	0.99	306
1	1.00	0.97	0.98	286
Accuracy			0.98	592
Macro avg	0.99	0.98	0.98	592
Weighted avg	0.99	0.98	0.98	592

Table 4.3: EfficientNet Model Evaluation Metrics

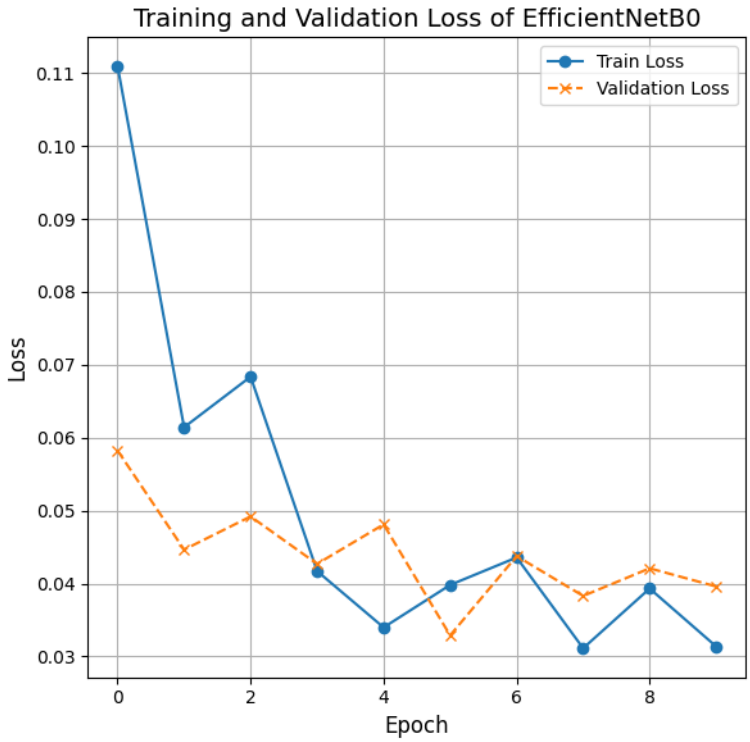


Figure 4.7: EfficientNetB0 Train and Validation Loss

This graph indicates that the train and validation loss of the EfficientNetB0 model is decreasing throughout the 10 epochs. The blue line takes the form of training loss which begins with high score but reduces steeply and leveled at around 0.03. The orange dotted line is the loss of validation, which varied slightly, but it was relatively low, it stayed between 0.04. The model achieves high learning rates, and both training and validation losses reduce steadily as time goes by.

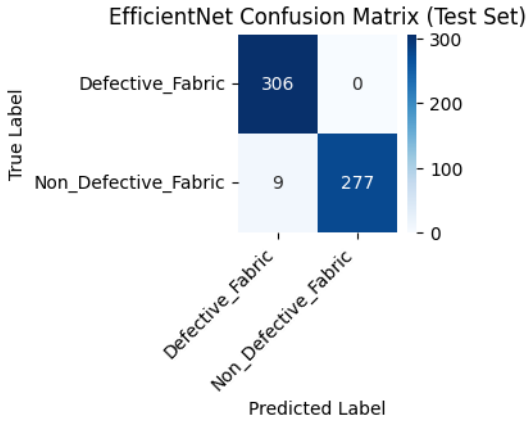


Figure 4.8: EfficientNetB0 Confusion Matrix

This confusion matrix shows the performance of the EfficientNet model on the test set. The model correctly predicted 306 defective fabrics and 277 non-defective fabrics. It misclassified 9 non-defective fabrics as defective, but there were no false

negatives for defective fabrics. Overall, the model performs well with a small number of misclassifications.

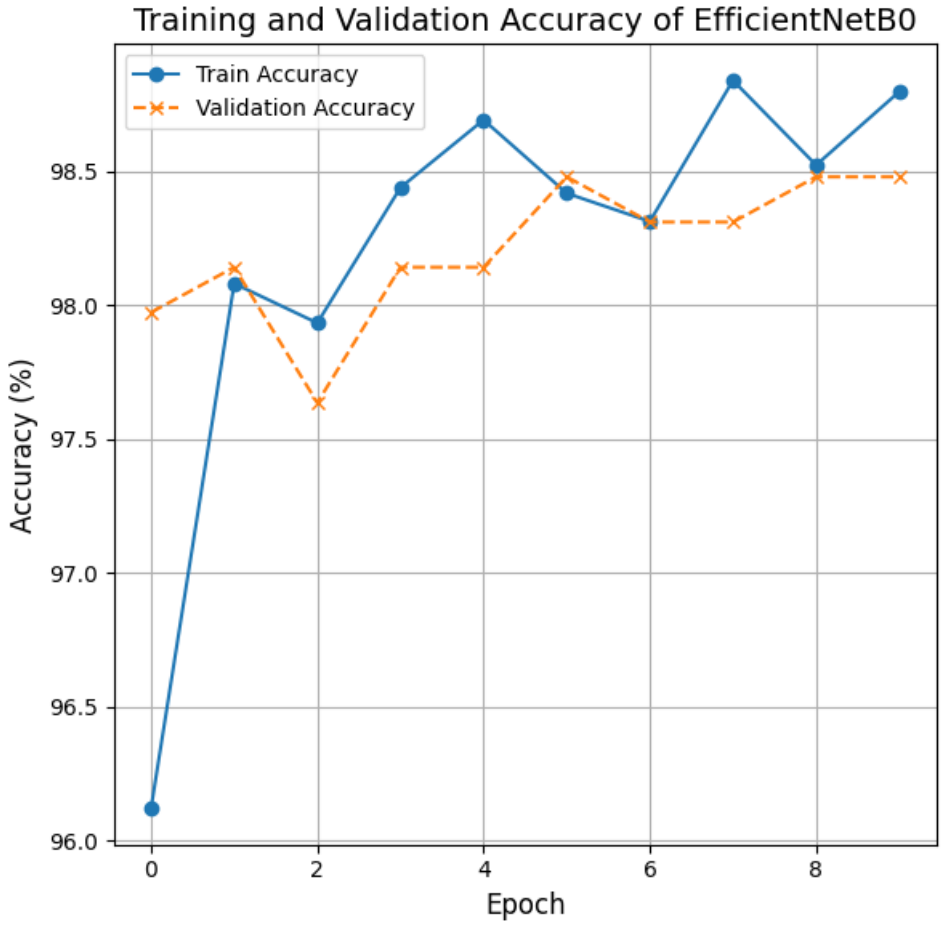


Figure 4.9: EfficientNetB0 Training and Validation Accuracy

The accuracy of the model in terms of training and validation is depicted in this graph with 10 epochs using EfficientNetB0. The blue line shows the training accuracy, and it begins with an accuracy of 96% and progressively rises to about 98.5%. The orange dashing line signifies the accuracy of validation which begins with a good value, but varies with a slight variation. The model works very well on the training set and validation set with a few discrepancies in the validation accuracy.

4.3 All Models Comparison and Analysis

Here we are going to compare the results of three other models: Simple CNN, MobileNetV2, and EfficientNetB0 in terms of their accuracy and their results on the test set in terms of classification are as follows:

- Simple CNN: 85.8%
- MobileNetV2: 97.64%
- EfficientNet: 98.47%

Simple CNN model attained a 86 percent accuracy. It did highly in predicting class 0 (with a perfect accuracy of 1.00) but not so with class 1, whereby it had a lower precision (0.78) but full recall (1.00). The f1-score in class 1 was more than in class 0 indicating that the model had a high recall with class 1, but this came at the cost of accuracy.

Simple CNN did not perform as well as the MobileNetV2 model, as the accuracy of the latter was an impressive 98%. This model recorded a balanced performance in both classes with the precision of 0.99 in the case of class 0 and 0.96 in class 1. Its recall values were also very high with a recall of 0.96 in class 0 and 1.00 in class 1 resulting in a very high f1-score. This general model was very reliable in the two classes.

EfficientNet also demonstrated great results of 98% accuracy, as it does MobileNetV2. Nevertheless, EfficientNet had an ideal recall (1.00) on class 0, and the precision of the class 1 was still very high at 1.00. It had a slight higher f1-scores than MobileNetV2, indicating that it had a little more superior balance in terms of precision and recall on the two classes.
Fianlly, Simple CNN is a useful solution when it comes to classifying class 0 with a high accuracy, but MobileNetV2 and EfficientNet offer a significantly more reliable overall performance, with consistently high precision and recall of both classes, and with EfficientNet having a minor advantage in balance and consistency.

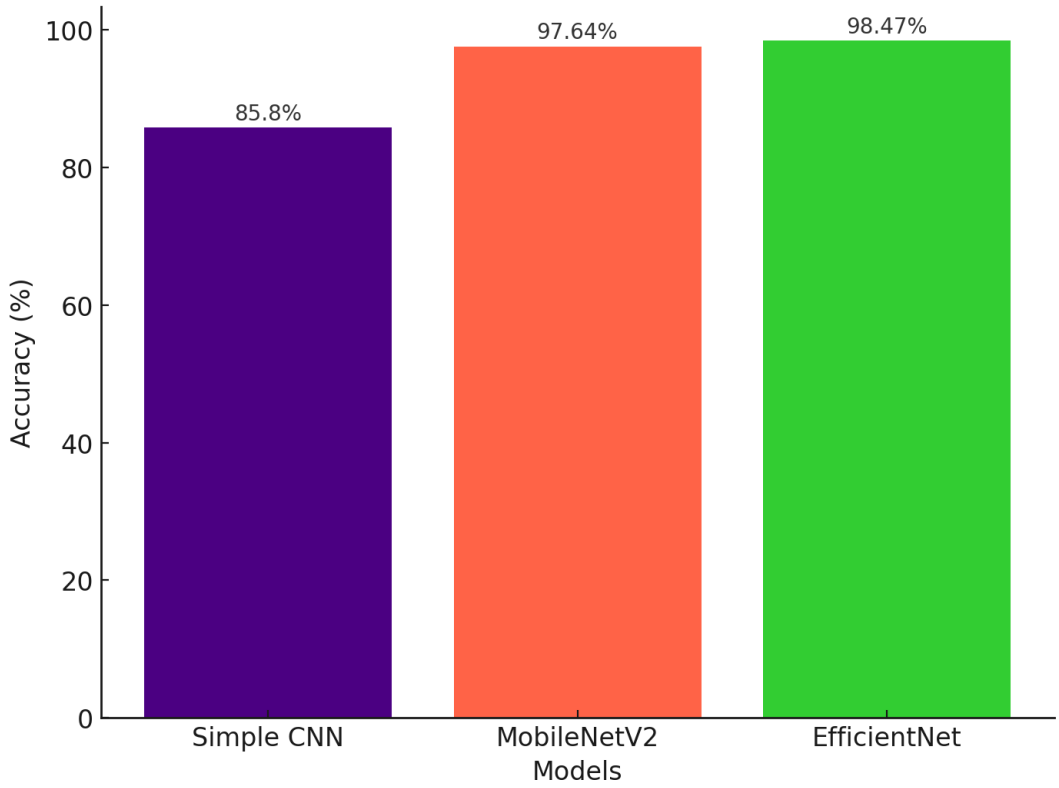


Figure 4.10: Model Accuracy Graph

4.4 Application for our Proposed Models

Our application is totally based on python language. For UI, we have used a python library Gradio. We have choosen gradio because it is the fastest way to demo our models with a friendly web interface. We used keras save model to save the model into .h5, .pth and .pkl format. And load it into the coding file using logics. The model is loaded only once and will be used in our web interface. We can predict the defectiveness of any fabric by either uploading from the system or webcam or copy and paste image and preprocess it just like it was done with the training images. After choosing image, then we will select our model and will click predict button and get results as image is defective fabric or non defective fabric. Web interface also show us predicted probability, elapsed and image size.

- Defective Fabric Detection: In our web interface,we have selected a EfficientNetB0 model for prediction. Then we have uploaded a images and then we click on predit button. The prediction shows that it is 100% defective fabric, predicted probability is 0.9998, elapsed time is 0.166s and image size is 1984*1468.

- Non-Defective Fabric detection: In our web interface,we have selected a EfficientNetB0 model for prediction. Then we have uploaded a images and then we click on predit button. The prediction shows that it is 100% non defective fabric, predicted probability is 1.000, elapsed time is 0.066s and image size is 420*570.

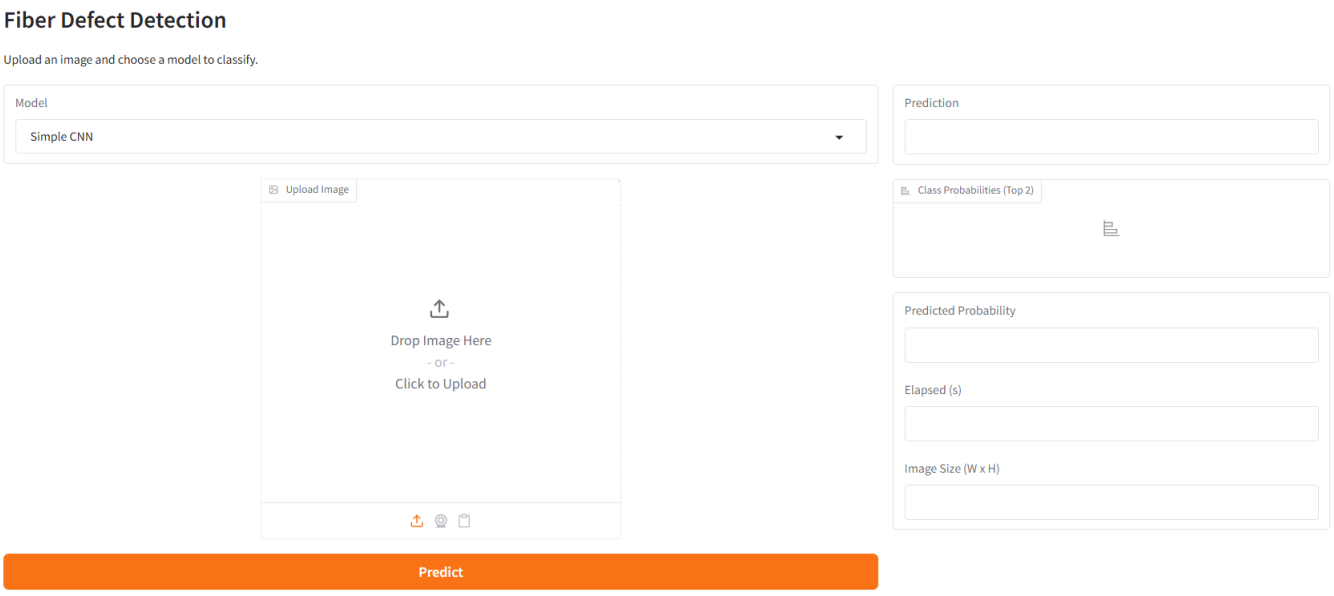


Figure 4.11: Frontend of our UI

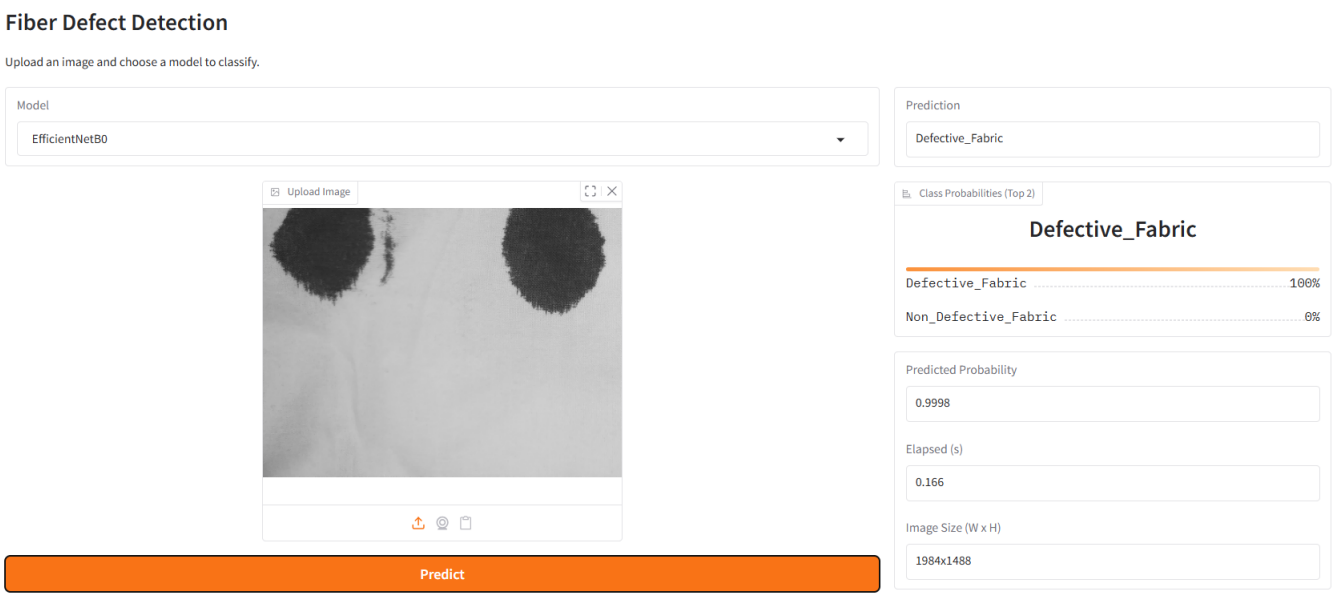


Figure 4.12: Defective Fabric Detection

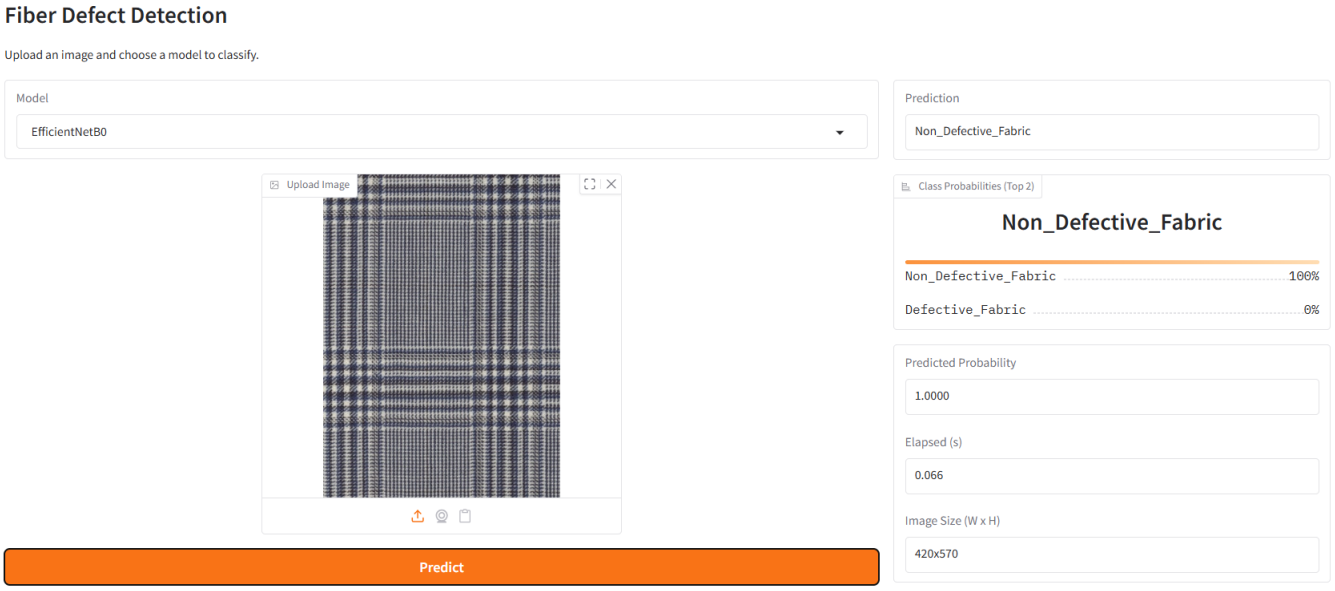


Figure 4.13: Non-Defective Fabric Detection

Chapter 5

Conclusion

5.1 Conclusion

In this study, we highlighted the transformative role of AI and computer vision in enhancing quality control in RMG printing. By developing advanced CNN models we addressed key challenges like defect de- tection. The system significantly outperforms traditional methods, reducing material waste, improving product quality, and mini- mizing human error. The findings demonstrate the potential for substantial gains in production efficiency and consistency. Models like Simple CNN, MobileNet and EfficientNet using epoch value 10 provided maximum levels of accuracy, scoring 85.8%, 97.64% and 98.47% respectively. Although the results are promising, future work should explore integrating the system with technologies like IoT and expanding its application across various industries. Au-tomating quality control in garment manufacturing is a crucial step toward achieving higher efficiency and consistent standards.

5.2 Future Work

We plan to find and collect more diverse datasets of defects in fabrics with more variations in them which we plan to train our models with. We also plan to incorporate newer and more advanced models to help with our proposed work. We also plan to enhance the models capabilities in handling real-time processing and incorporate color and design accuracy detection to our models.

Bibliography

[1] H. Golnabi and A. Asadpour, "Design and application of industrial machine vision systems", *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 6, pp. 630–637, 2007, 16th International Conference on Flexible Automation and Intelligent Manufacturing, ISSN: 0736-5845. DOI: https://doi.org/10.1016/j.rcim.2007.02.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584507000233.

[2] A. Islam, S. Akhter, and T. E. Mursalin, "Automated textile defect recognition system using computer vision and artificial neural networks", *International Journal of Materials and Textile Engineering*, vol. 2, no. 1, pp. 110–115, 2008.

[3] J. Moreno, A. Aguila, E. Partida, O. Morales, and R. Tejeida, "Fabricvision: System of error detection in the manufacture of garments", *International Journal of Advanced Computer Science and Applications*, vol. 8, no. 11, 2017.

[4] J. Silvestre-Blanes, T. Albero-Albero, I. Miralles, R. Prez-Llorens, and J. Moreno, *Afid: A public fabric image database for defect detection*, 2019. [Online]. Available: https://content.sciendo.com/view/journals/aut/ahead-of-print/article-10.2478-aut-2019-0035.xml.

[5] S. M. J. Alam, G. Hu, and S. Roy, "Analysis of a printed complex image quality checking method of fabric cloth for development of an automated quality checking system", *Signal, Image and Video Processing*, vol. 14, pp. 1341–1351, 2020. DOI: 10.1007/s11760-020-01737-w.

[6] K. Paraskevoudis, P. Karayannis, and E. P. Koumoulos, "Real-time 3d printing remote defect detection (stringing) with computer vision and artificial intelligence", *Processes*, vol. 8, no. 11, p. 1464, 2020.

[7] S. Chakraborty, M. Moore, and L. Chapman, *Automatic defect detection of print fabric using convolutional neural network*, Jan. 2021. DOI: 10.48550/arXiv.2101.00703.

[8] T. Ersz, H. Zahoor, and F. Ersz, "Fabric and production defect detection in the apparel industry using data mining algorithms", *International Journal of 3D Printing Technologies and Digital Industry*, vol. 5, no. 3, pp. 742–757, 2021.

[9] D. T Joy, G. Kaur, A. Chugh, and S. B. Bajaj, "Computer vision for color detection", *International Journal of Innovative Research in Computer Science & Technology (IJIRCST) ISSN*, pp. 2347–5552, 2021.

[10] H. Liu, "Real-time image defect detection system of cloth digital printing machine", *Paladyn, Journal of Behavioral Robotics*, vol. 14, pp. 1–7, 2022. DOI: 10.1515/pjbr-2022-0099. [Online]. Available: https://doi.org/10.1515/pjbr-2022-0099.

[11] S. Shahrabadi, Y. Castilla, M. Guevara, L. G. Magalhes, D. Gonzalez, and T. Ado, "Defect detection in the textile industry using image-based machine learning methods: A brief review", in *Journal of Physics: Conference Series*, IOP Publishing, vol. 2224, 2022, p. 012 010.

[12] R. Liu, Z. Yu, Q. Fan, *et al.*, *The improved method in fabric image classification using convolutional neural network*, 2023. DOI: 10.1007/s11042-023-15573-w. [Online]. Available: https://doi.org/10.1007/s11042-023-15573-w.

[13] A. Saberironaghi, J. Ren, and M. El-Gindy, "Defect detection methods for industrial products using deep learning techniques: A review", *Algorithms*, vol. 16, no. 2, p. 95, 2023. DOI: 10.3390/a16020095.

[14] GeeksforGeeks. (Jun. 3, 2024). Efficientnet architecture. Last updated: June 3, 2024, [Online]. Available: https://www.geeksforgeeks.org/computer-vision/efficientnet-architecture/.

[15] A. D. Gupta, Z. Sadek, M. S. Hossain, T. R. Toha, A. Mondol, S. U. Habiba, and S. M. M. Alam, "An approach to automatic fault detection in four-point system for knitted fabric with our benchmark dataset isl-knit", *Heliyon*, vol. 10, no. 17, 2024.

[16] S. K, S. V, S. B, and D. P, "Fabric defect detection using transfer learning", *International Journal of Research In Science Engineering*, vol. 4, pp. 62–71, Sep. 2024. DOI: 10.55529/ijrise.45.62.71.

[17] W. Zhu, Z. Wang, Q. Li, and C. Zhu, "A method of enhancing silk digital printing color prediction through pix2pix gan-based approaches", *Applied Sciences*, vol. 14, no. 1, 2024, ISSN: 2076-3417. DOI: 10.3390/app14010011. [Online]. Available: https://www.mdpi.com/2076-3417/14/1/11.

[18] R. Islam, S. U. Sajjad Ullah, S. Rahaman, J. Hossain Sheebly, and S. R. Shaon, *Fabric defect*, Mendeley Data, V1, 2025. DOI: 10.17632/y62b4pfyz2.1.

[19] P. Pathirana, *Fabric stain dataset*, Accessed: 2025-08-09, 2025. [Online]. Available: https://www.kaggle.com/datasets/priemshpathirana/fabric-stain-dataset.

[20] Roboflow, *Mobilenet v2 classification*, Accessed: 2025-08-09, 2025. [Online]. Available: https://roboflow.com/model/mobilenet-v2-classification.

[21] N. D. Kulkarni and S. Bansal, "Revolutionizing manufacturing: The integral role of ai and computer vision in shaping future industries",

Appendix

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
!pip install -q keras
```

```
!pip install -q tensorflow
```

```
!pip install -q torchsummary
```

```
import tensorflow as tf

path = "/content/drive/MyDrive/T2420376_Thesis/Final_Dataset_for_Model"

data = tf.keras.utils.image_dataset_from_directory(directory = path, image_size = (128, 128), batch_size = 32)
```

```
Found 5928 files belonging to 2 classes.
```

```
class_names = data.class_names
print(class_names)
```

```
['Defective_Fabric', 'Non_Defective_Fabric']
```

```
import numpy as np
label_map = {int(m): n for m, n in zip(np.arange(len(class_names)), class_names)}

print(label_map)
```

```
{0: 'Defective_Fabric', 1: 'Non_Defective_Fabric'}
```

Figure 5.1: Code Image - 1

```python
import os
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, random_split, Dataset
from torchvision import transforms
from PIL import Image


class FabricDataset(Dataset):
    def __init__(self, image_dir, class_names, transform=None):
        self.image_dir = image_dir
        self.class_names = class_names
        self.transform = transform
        self.image_paths = []
        self.labels = []


        for label, class_name in enumerate(class_names):
            class_dir = os.path.join(image_dir, class_name)

            for root, dirs, files in os.walk(class_dir):
                for file in files:

                    file_path = os.path.join(root, file)

                    if file.lower().endswith(('.png', '.jpg', '.jpeg')):
                        self.image_paths.append(file_path)
                        self.labels.append(label)


    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_path = self.image_paths[idx]
        try:
            image = Image.open(img_path).convert('RGB')
            label = self.labels[idx]

            if self.transform:
                image = self.transform(image)

            return image, label
        except Exception as e:
            print(f"Error opening image file: {img_path} - {e}")

            return None, None

```

Figure 5.2: Code Image - 2

```
                return None, None


transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])


dataset = FabricDataset(image_dir="/content/drive/MyDrive/T2420376_Thesis/Final_Dataset_for_Model",
                        class_names=['Defective_Fabric', 'Non_Defective_Fabric'], transform=transform)

print(f"Length of dataset: {len(dataset)}")


train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = random_split(dataset, [train_size, val_size, test_size])


train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)


for images, labels in train_loader:
    print(images.shape)
    print(labels)
    break
```

```
Length of dataset: 5928
torch.Size([32, 3, 224, 224])
tensor([0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1,
        0, 0, 1, 0, 1, 0, 1, 1])
```

Figure 5.3: Code Image - 3

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SimpleCNN(nn.Module):
    def __init__(self, num_classes=2):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)


        self.fc1 = None
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = self.pool2(F.relu(self.bn2(self.conv2(x))))
        x = self.pool3(F.relu(self.bn3(self.conv3(x))))


        flattened_size = x.size(1) * x.size(2) * x.size(3)

        if self.fc1 is None:
            self.fc1 = nn.Linear(flattened_size, 128).to(x.device)

        x = x.view(-1, flattened_size)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x

model = SimpleCNN(num_classes=len(dataset.class_names))


criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

print("PyTorch Model Defined:")
print(model)
```

```
PyTorch Model Defined:
```

Figure 5.4: Code Image - 4

```python
import torch
import torch.nn as nn
import torchvision.models as models

mobilenet_v2 = models.mobilenet_v2(pretrained=True)

num_ftrs = mobilenet_v2.classifier[1].in_features
num_classes = len(dataset.class_names)
mobilenet_v2.classifier[1] = nn.Linear(num_ftrs, num_classes)


print("Modified MobileNetV2 Model:")
print(mobilenet_v2)

optimizer_mobilenet = torch.optim.Adam(mobilenet_v2.parameters(), lr=0.001)
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to /root/.ca
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:208: UserWarning: Th
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/torchvision/models/_utils.py:223: UserWarning: Ar
```

Figure 5.5: Code Image - 5

```python
!pip install -q torchinfo
```

```python
!pip install -q torchsummary
```

```python
from torchvision.models import efficientnet_b0
import torch.nn as nn
import torch.optim as optim
from torchinfo import summary
import torch


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = efficientnet_b0(weights='DEFAULT')


num_ftrs = model.classifier[1].in_features
model.classifier[1] = nn.Linear(num_ftrs, len(class_names))


model.to(device)


criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

print("--- EfficientNetB0 Model Summary ---")
summary(model, input_size=(1, 3, 224, 224))
```

```
Downloading: "https://download.pytorch.org/models/efficientnet_b0_rwightman-7f5810bc.pth" to /roo
100%|██████████| 20.5M/20.5M [00:00<00:00, 55.6MB/s]
--- EfficientNetB0 Model Summary ---
==========================================================================================
Layer (type:depth-idx)                                  Output Shape              Param #
==========================================================================================
```

Figure 5.6: Code Image - 6

```python
import time

num_epochs = 10

for epoch in range(num_epochs):
    start_time = time.time()  # Start time of the epoch
    model.train()
    running_loss = 0.0
    correct_train = 0
    total_train = 0

    for i, (inputs, labels) in enumerate(train_loader):

        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item() * inputs.size(0)
        _, predicted = torch.max(outputs.data, 1)
        total_train += labels.size(0)
        correct_train += (predicted == labels).sum().item()


    epoch_loss = running_loss / len(train_loader.dataset)
    epoch_acc = 100 * correct_train / total_train
    print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {epoch_loss:.4f}, Train Acc: {epoch_acc:.2f}%')


    model.eval()
    running_val_loss = 0.0
    correct_val = 0
    total_val = 0
    with torch.no_grad():
        for inputs_val, labels_val in val_loader:

            outputs_val = model(inputs_val)
            val_loss = criterion(outputs_val, labels_val)

            running_val_loss += val_loss.item() * inputs_val.size(0)
            _, predicted_val = torch.max(outputs_val.data, 1)
            total_val += labels_val.size(0)
            correct_val += (predicted_val == labels_val).sum().item()

    epoch_val_loss = running_val_loss / len(val_loader.dataset)
    epoch_val_acc = 100 * correct_val / total_val
    print(f'Epoch [{epoch+1}/{num_epochs}], Val Loss: {epoch_val_loss:.4f}, Val Acc: {epoch_val_acc:.2f}%')

    end_time = time.time()  # End time of the epoch
    epoch_duration = end_time - start_time
    print(f'Epoch [{epoch+1}/{num_epochs}] took {epoch_duration:.2f} seconds')


print('Finished Training')
```

```
Epoch [1/10], Train Loss: 0.2482, Train Acc: 92.58%
```

Figure 5.7: Code Image - 7

```python
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, train_losses, label='Train Loss', linestyle='-', marker='o')
plt.plot(epochs, val_losses, label='Validation Loss', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('Training and Validation Loss of SimpleCNN', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)


plt.subplot(1, 2, 2)
plt.plot(epochs, train_accs, label='Train Accuracy', linestyle='-', marker='o')
plt.plot(epochs, val_accs, label='Validation Accuracy', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Accuracy (%)', fontsize=12)
plt.title('Training and Validation Accuracy of SimpleCNN', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)

plt.tight_layout()
plt.show()
```

Figure 5.8: Code Image - 8

```python
import matplotlib.pyplot as plt


plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(train_losses_mobilenet, label='Train Loss', linestyle='-', marker='o')
plt.plot(val_losses_mobilenet, label='Validation Loss', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('Training and Validation Loss of MobileNet', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)


plt.subplot(1, 2, 2)
plt.plot(train_accuracies_mobilenet, label='Train Accuracy', linestyle='-', marker='o')
plt.plot(val_accuracies_mobilenet, label='Validation Accuracy', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Accuracy (%)', fontsize=12)
plt.title('Training and Validation Accuracy of MobileNet', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)

plt.tight_layout()
plt.show()
```

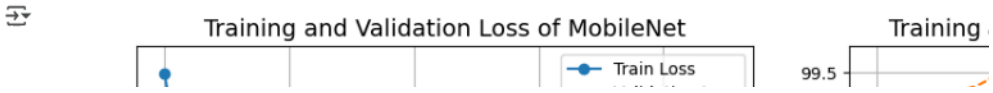


Figure 5.9: Code Image - 9

```python
import matplotlib.pyplot as plt

# Plot training and validation loss
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(train_losses_efficientnet, label='Train Loss', linestyle='-', marker='o')
plt.plot(val_losses_efficientnet, label='Validation Loss', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Loss', fontsize=12)
plt.title('Training and Validation Loss of EfficientNetB0', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)

# Plot training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(train_accuracies_efficientnet, label='Train Accuracy', linestyle='-', marker='o')
plt.plot(val_accuracies_efficientnet, label='Validation Accuracy', linestyle='--', marker='x')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Accuracy (%)', fontsize=12)
plt.title('Training and Validation Accuracy of EfficientNetB0', fontsize=14)
plt.legend(fontsize=10)
plt.grid(True)

plt.tight_layout()
plt.show()
```

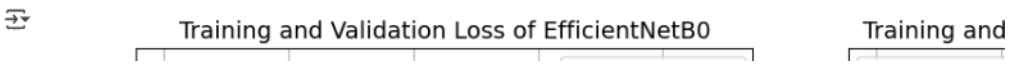


Figure 5.10: Code Image - 10

```python
model.eval()
predictions_list = []
true_labels_list = []

with torch.no_grad():
    for inputs_val, labels_val in val_loader:


        outputs_val = model(inputs_val)
        _, predicted = torch.max(outputs_val.data, 1)

        predictions_list.extend(predicted.cpu().numpy())
        true_labels_list.extend(labels_val.cpu().numpy())

predicted_classes = np.array(predictions_list)
true_classes = np.array(true_labels_list)


cm = confusion_matrix(true_classes, predicted_classes)


class_labels = dataset.class_names


plt.figure(figsize=(len(class_labels), len(class_labels)))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)

plt.title('SImple CNN Confusion Matrix (Test Set)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()


print("Simple CNN Accuracy Score (Test Set):")
print(accuracy_score(true_classes, predicted_classes))
print("\nSimple CNN Classification Report (Test Set):")
print(classification_report(true_classes, predicted_classes, zero_division=0))
```

```
/tmp/ipython-input-2257158941.py:39: UserWarning: Tight layout not applied. The left and right margins cannot be
  plt.tight_layout()
```


Figure 5.11: Code Image - 11

```python
mobilenet_v2.eval()
predictions_list_mobilenet = []
true_labels_list_mobilenet = []

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
mobilenet_v2.to(device)

with torch.no_grad():
    for inputs_test, labels_test in test_loader:

        inputs_test, labels_test = inputs_test.to(device), labels_test.to(device)

        outputs_test = mobilenet_v2(inputs_test)
        _, predicted_test = torch.max(outputs_test.data, 1)

        predictions_list_mobilenet.extend(predicted_test.cpu().numpy())
        true_labels_list_mobilenet.extend(labels_test.cpu().numpy())

predicted_classes_mobilenet = np.array(predictions_list_mobilenet)
true_classes_mobilenet = np.array(true_labels_list_mobilenet)


cm_mobilenet = confusion_matrix(true_classes_mobilenet, predicted_classes_mobilenet)

class_labels = dataset.class_names


plt.figure(figsize=(len(class_labels), len(class_labels)))
sns.heatmap(cm_mobilenet, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)

plt.title('MobileNetV2 Confusion Matrix (Test Set)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()


print("MobileNetV2 Accuracy Score (Test Set):")
print(accuracy_score(true_classes_mobilenet, predicted_classes_mobilenet))
print("\nMobileNetV2 Classification Report (Test Set):")
print(classification_report(true_classes_mobilenet, predicted_classes_mobilenet, zero_division=0))
```

```
/tmp/ipython-input-143028486.py:42: UserWarning: Tight layout not applied. The left and right margins cannot be made lar
  plt.tight_layout()
```

Figure 5.12: Code Image - 12

```
model.eval()
predictions_list = []
true_labels_list = []

with torch.no_grad():
    for inputs_val, labels_val in val_loader:
        inputs_val = inputs_val.to(device)
        labels_val = labels_val.to(device)

        outputs_val = model(inputs_val)
        _, predicted = torch.max(outputs_val.data, 1)

        predictions_list.extend(predicted.cpu().numpy())
        true_labels_list.extend(labels_val.cpu().numpy())

predicted_classes = np.array(predictions_list)
true_classes = np.array(true_labels_list)


cm = confusion_matrix(true_classes, predicted_classes)


class_labels = class_names


plt.figure(figsize=(len(class_labels), len(class_labels)))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_labels, yticklabels=class_labels)

plt.title('EfficientNet Confusion Matrix (Test Set)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()


print("EfficientNet Accuracy Score  (Test Set) :")
print(accuracy_score(true_classes, predicted_classes))
print("\nEfficientNet Classification Report  (Test Set) :")
print(classification_report(true_classes, predicted_classes, zero_division=0))
```

```
/tmp/ipython-input-2315679254.py:40: UserWarning: Tight layout not applied. The left and right margins cannot be m
  plt.tight_layout()
```

Figure 5.13: Code Image - 13

```python
import torch


model_save_path = "/content/drive/MyDrive/T2420376_Thesis/simple_cnn_trained.pth"

torch.save(model.state_dict(), model_save_path)

print(f"SimpleCNN model saved successfully to {model_save_path}")
```

```
SimpleCNN model saved successfully to /content/drive/MyDrive/T2420376_Thesis/simple_cnn_trained.pth
```

```python
import os
import torch

save_path = '/content/drive/MyDrive/T2420376_Thesis/simple_cnn_model.h5'


os.makedirs(os.path.dirname(save_path), exist_ok=True)

torch.save(model.state_dict(), save_path)

print(f"Model saved in H5 format at '{save_path}'")
```

```
Model saved in H5 format at '/content/drive/MyDrive/T2420376_Thesis/simple_cnn_model.h5'
```

```python
import torch
import pickle
import os


save_path = '/content/drive/MyDrive/T2420376_Thesis/simple_cnn_model.pkl' l

os.makedirs(os.path.dirname(save_path), exist_ok=True)

with open(save_path, 'wb') as f:
    pickle.dump(model.state_dict(), f)
print(f"Model saved in Pickle format at '{save_path}'")
```

```
Model saved in Pickle format at '/content/drive/MyDrive/T2420376_Thesis/simple_cnn_model.pkl'
```

Figure 5.14: Code Image - 14

```python
import torch

model_save_path = "/content/drive/MyDrive/T2420376_Thesis/mobilenet_v2_trained.pth"

torch.save(mobilenet_v2.state_dict(), model_save_path)

print(f"MobileNetV2 model saved successfully to {model_save_path}")
```

```
MobileNetV2 model saved successfully to /content/drive/MyDrive/T2420376_Thesis/mobilenet_v2_trained.pth
```

```python
import os
import torch

save_path = '/content/drive/MyDrive/T2420376_Thesis/mobilenet_model.h5'

os.makedirs(os.path.dirname(save_path), exist_ok=True)

torch.save(mobilenet_v2.state_dict(), save_path)

print(f"Model saved in H5 format at '{save_path}'")
```

```
Model saved in H5 format at '/content/drive/MyDrive/T2420376_Thesis/mobilenet_model.h5'
```

```python
import torch
import pickle
import os

save_path = '/content/drive/MyDrive/T2420376_Thesis/mobillenetnet_model.pkl'

os.makedirs(os.path.dirname(save_path), exist_ok=True)

with open(save_path, 'wb') as f:
    pickle.dump(mobilenet_v2.state_dict(), f)
print(f"Model saved in Pickle format at '{save_path}'")
```

```
Model saved in Pickle format at '/content/drive/MyDrive/T2420376_Thesis/mobillenetnet_model.pkl'
```

Figure 5.15: Code Image - 15

```python
import torch

model_save_path = "/content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_trained.pth"

torch.save(model.state_dict(), model_save_path)

print(f"EfficientNetB0 model saved successfully to {model_save_path}")
```

```
EfficientNetB0 model saved successfully to /content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_trained.pth
```

```python
import os
import torch

save_path = '/content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_model.h5'

os.makedirs(os.path.dirname(save_path), exist_ok=True)

torch.save(model.state_dict(), save_path)

print(f"Model saved in H5 format at '{save_path}'")
```

```
Model saved in H5 format at '/content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_model.h5'
```

```python
import torch
import pickle
import os

save_path = '/content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_model.pkl'

os.makedirs(os.path.dirname(save_path), exist_ok=True)

with open(save_path, 'wb') as f:
    pickle.dump(model.state_dict(), f)
print(f"Model saved in Pickle format at '{save_path}'")
```

```
Model saved in Pickle format at '/content/drive/MyDrive/T2420376_Thesis/efficientnet_b0_model.pkl'
```

Figure 5.16: Code Image - 16

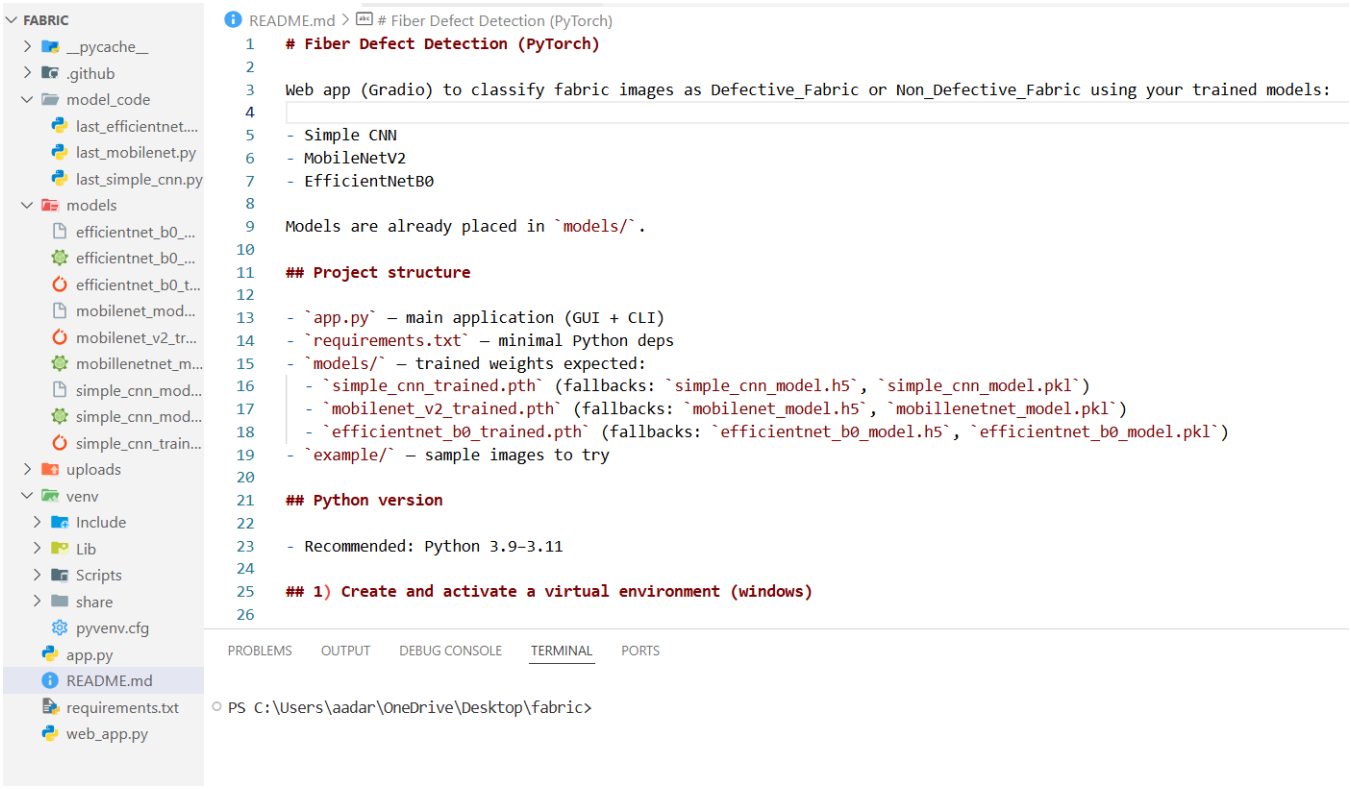


Figure 5.17: Code Image - 17

app.py > ...

```python
#!/usr/bin/env python3
"""
Fiber Defect Detection

- Loads one of three models: Simple CNN, MobileNetV2, EfficientNetB0
- Accepts image input and predicts: Defective_Fabric or Non_Defective_Fabric

Models expected in: server/models/
  - simple_cnn_trained.pth
  - mobilenet_v2_trained.pth
  - efficientnet_b0_trained.pth

Fallbacks: if a .pth isn't found, will try .h5 (torch.save state_dict) then .pkl

"""
from __future__ import annotations
import argparse
import os
import pickle
import sys
import time
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, Optional, Tuple

import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from PIL import Image
from torchvision import transforms as T
from torchvision import models as tv_models


# ----------------------------
# Constants and Labels
# ----------------------------
```

Figure 5.18: Code Image - 18

app.py > ...

```python
CLASSES = ["Defective_Fabric", "Non_Defective_Fabric"]
IDX2LABEL = {0: CLASSES[0], 1: CLASSES[1]}
INPUT_SIZE = (224, 224)

MODEL_KEYS = [
    "Simple CNN",
    "MobileNetV2",
    "EfficientNetB0",
]


# ----------------------------
# Model Definitions (Inference)
# ----------------------------
class SimpleCNNFixed(nn.Module):
    """Simple CNN architecture matching the training code with fixed fc1 size.

    Training code used:
      3 conv blocks (3->32, 32->64, 64->128), each with BN + ReLU + MaxPool(2)
      Input size 224x224 => after 3 pools => 28x28 spatial
      Flattened size = 128 * 28 * 28 = 100352
      fc1: 100352 -> 128, dropout(0.5)
      fc2: 128 -> 2
    """

    def __init__(self, num_classes: int = 2):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(32)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(64)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
```

Figure 5.19: Code Image - 19

app.py > SimpleCNNFixed > __init__

```python
class SimpleCNNFixed(nn.Module):
    def __init__(self, num_classes: int = 2):
        self.bn3 = nn.BatchNorm2d(128)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)

        flattened_size = 128 * 28 * 28
        self.fc1 = nn.Linear(flattened_size, 128)
        self.dropout = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.pool1(F.relu(self.bn1(self.conv1(x))))
        x = self.pool2(F.relu(self.bn2(self.conv2(x))))
        x = self.pool3(F.relu(self.bn3(self.conv3(x))))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x


@dataclass
class ModelBundle:
    name: str
    model: nn.Module
    device: torch.device
    transform: T.Compose
    classes: Tuple[str, str]


def _build_model(name: str) -> nn.Module:
    name = name.strip()
    if name == "Simple CNN":
        return SimpleCNNFixed(num_classes=2)
    elif name == "MobileNetV2":
        m = tv_models.mobilenet_v2(weights=None)
        in_feats = m.classifier[1].in_features
```

Figure 5.20: Code Image - 20

app.py > SimpleCNNFixed > __init__

```python
def _build_model(name: str) -> nn.Module:
        in_feats = m.classifier[1].in_features
        m.classifier[1] = nn.Linear(in_feats, 2)
        return m
    elif name == "EfficientNetB0":
        m = tv_models.efficientnet_b0(weights=None)
        in_feats = m.classifier[1].in_features
        m.classifier[1] = nn.Linear(in_feats, 2)
        return m
    else:
        raise ValueError(f"Unknown model name: {name}")


def _find_weight_file(models_dir: Path, name: str) -> Optional[Path]:
    preferred = {
        "Simple CNN": ["simple_cnn_trained.pth", "simple_cnn_model.h5", "simple_cnn_model.pkl"],
        "MobileNetV2": ["mobilenet_v2_trained.pth", "mobilenet_model.h5", "mobillenetnet_model.pkl"],
        "EfficientNetB0": ["efficientnet_b0_trained.pth", "efficientnet_b0_model.h5", "efficientnet_b0_model.pkl"],
    }
    for fname in preferred.get(name, []):
        p = models_dir / fname
        if p.exists():
            return p
    return None


def _load_state_dict(path: Path) -> Dict[str, torch.Tensor]:
    ext = path.suffix.lower()
    if ext in {".pth", ".h5"}:
        return torch.load(path, map_location="cpu")
    elif ext == ".pkl":
        with open(path, "rb") as f:
            return pickle.load(f)
    else:
        raise ValueError(f"Unsupported weight file extension: {ext}")
```

Figure 5.21: Code Image - 21

app.py > load_model_bundle

```python
def load_model_bundle(model_name: str, models_dir: Path) -> ModelBundle:
    device = torch.device("cpu")
    model = _build_model(model_name)
    weight_file = _find_weight_file(models_dir, model_name)
    if weight_file is None:
        raise FileNotFoundError(
            f"Weights not found for '{model_name}' in {models_dir}."
        )
    state = _load_state_dict(weight_file)
    model.load_state_dict(state, strict=True)
    model.eval()

    transform = T.Compose([
        T.Resize(INPUT_SIZE),
        T.ToTensor(),
    ])

    return ModelBundle(
        name=model_name,
        model=model.to(device),
        device=device,
        transform=transform,
        classes=(CLASSES[0], CLASSES[1]),
    )


def predict(bundle: ModelBundle, image_path: Path) -> Dict[str, object]:
    t0 = time.time()
    img = Image.open(image_path).convert("RGB")
    tensor = bundle.transform(img).unsqueeze(0).to(bundle.device)
    with torch.no_grad():
        logits = bundle.model(tensor)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
        pred_idx = int(np.argmax(probs))
        pred_prob = float(probs[pred_idx])
        pred_label = IDX2LABEL[pred_idx]
    elapsed = time.time() - t0
```

Figure 5.22: Code Image - 22

app.py > load_model_bundle

```python

def predict(bundle: ModelBundle, image_path: Path) -> Dict[str, object]:
    t0 = time.time()
    img = Image.open(image_path).convert("RGB")
    tensor = bundle.transform(img).unsqueeze(0).to(bundle.device)
    with torch.no_grad():
        logits = bundle.model(tensor)
        probs = torch.softmax(logits, dim=1).cpu().numpy()[0]
        pred_idx = int(np.argmax(probs))
        pred_prob = float(probs[pred_idx])
        pred_label = IDX2LABEL[pred_idx]
    elapsed = time.time() - t0
    return {
        "pred_label": pred_label,
        "pred_index": pred_idx,
        "probability": pred_prob,
        "probs": probs.tolist(),
        "elapsed_sec": elapsed,
        "image_size": img.size,
    }


```

Figure 5.23: Code Image - 23

web_app.py > ...

```python
#!/usr/bin/env python3
"""
Fiber Defect Detection - Web UI (Gradio)

Reuses the inference logic and model weights in server/models/.
Run:
  python server/web_app.py
Then open http://127.0.0.1:7860
"""
from __future__ import annotations
import time
from pathlib import Path
from typing import Dict

import gradio as gr
import numpy as np
import torch
from PIL import Image


from app import load_model_bundle, predict, MODEL_KEYS

# Cache loaded models to avoid reloading per request
_BUNDLE_CACHE: Dict[str, object] = {}


def get_bundle(model_name: str):
    if model_name in _BUNDLE_CACHE:
        return _BUNDLE_CACHE[model_name]
    models_dir = Path(__file__).resolve().parent / "models"
    bundle = load_model_bundle(model_name, models_dir)
    _BUNDLE_CACHE[model_name] = bundle
    return bundle

```

Figure 5.24: Code Image - 24

web_app.py > ...

```python
def predict_ui(model_name: str, img: Image.Image):
    if img is None:
        return "No image provided", {"Defective_Fabric": 0.0, "Non_Defective_Fabric": 0.0}, "-", "-", "-"

    bundle = get_bundle(model_name)
    t0 = time.time()

    # Save uploaded image under server/uploads/ (inside this folder)
    base_dir = Path(__file__).resolve().parent
    uploads_dir = base_dir / "uploads"
    uploads_dir.mkdir(parents=True, exist_ok=True)
    # Unique filename by timestamp
    filename = f"upload_{int(time.time()*1000)}.png"
    save_path = uploads_dir / filename
    img.convert("RGB").save(save_path)
    res = predict(bundle, save_path)

    # Build outputs
    probs = res.get("probs", [0.0, 0.0])
    label_scores = {
        "Defective_Fabric": float(probs[0]),
        "Non_Defective_Fabric": float(probs[1]),
    }

    return (
        res.get("pred_label", "-"),
        label_scores,
        f"{res.get('probability', 0.0):.4f}",
        f"{res.get('elapsed_sec', 0.0):.3f}",
        f"{res.get('image_size', (0,0))[0]}x{res.get('image_size', (0,0))[1]}",
    )

```

Figure 5.25: Code Image - 25

web_app.py > ...

```python
def predict_ui(model_name: str, img: Image.Image):
    )


def build_ui():
    # Custom CSS to hide Gradio footer and other elements
    custom_css = """
    footer {visibility: hidden}
    .gradio-container .footer {visibility: hidden !important}
    .gradio-container .built-with {visibility: hidden !important}
    """

    with gr.Blocks(title="Fiber Defect Detection", css=custom_css) as demo:
        gr.Markdown("# Fiber Defect Detection\nUpload an image and choose a model to classify.")
        with gr.Row():
            with gr.Column(scale=2):
                model = gr.Dropdown(choices=MODEL_KEYS, value=MODEL_KEYS[0], label="Model")
                image = gr.Image(type="pil", label="Upload Image", height=400, width=400)
                btn = gr.Button("Predict", variant="primary")
            with gr.Column():
                pred = gr.Textbox(label="Prediction", interactive=False)
                label = gr.Label(label="Class Probabilities (Top 2)", num_top_classes=2)
                prob = gr.Textbox(label="Predicted Probability", interactive=False)
                elapsed = gr.Textbox(label="Elapsed (s)", interactive=False)
                size = gr.Textbox(label="Image Size (W x H)", interactive=False)
        btn.click(predict_ui, inputs=[model, image], outputs=[pred, label, prob, elapsed, size])
    return demo


def main():
    demo = build_ui()
    demo.launch(server_name="127.0.0.1", server_port=7860, show_api=False)


if __name__ == "__main__":
    main()

```

Figure 5.26: Code Image - 26

README.md app.py 9+ web_app.py requirements.txt

requirements.txt

```
#  the Fiber Defect Detection app
# Tip: If you hit platform-specific issues with torch, see README.md for alternatives.
torch>=2.1.0
torchvision>=0.16.0
Pillow>=9.4.0
numpy>=1.24.0
gradio>=4.0.0

```

Figure 5.27: Code Image - 27