Solder Inspection for Pseudo Error Reduction in Printed Circuit Board production using Unsupervised Learning

Master's Thesis

for the degree of

Master of Science (M.Sc.)

Data Science

at the Faculty of Sciences of
Friedrich-Alexander-Universität Erlangen-Nürnberg

in Cooperation with
Siemens AG, Berlin, Germany

submitted on 22-10-2024

by **Rohit Potdukhe**

Advisor: Prof. Dr. Enrique Zuazua
Supervisor: Dr. Majid Mortazavi

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Non-disclosure notice

The Academic Work contains confidential information that is subject to secrecy. Therefore, the Academic Work must not be duplicated or published without prior written approval of Siemens AG.

Sperrvermerk

Die Arbeit beinhaltet vertrauliche Informationen, die der Geheimhaltun unterliegen. Aus diesem Grund darf die Arbeit ohne vorherige schriftliche Zustimmung der Siemens AG nicht vervielfältigt oder veröffentlicht werden.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Majid Mortazavi, for his invaluable guidance, support, and encouragement throughout the course of my research. His insights and expertise were crucial to the successful completion of this work. I am also deeply grateful to my advisor, Prof. Dr. Enrique Zuazua, for his continuous support, constructive feedback, and for always being available to discuss ideas and provide guidance. I would like to extend my gratitude to Siemens AG, Berlin, Germany, for providing me with the opportunity to conduct my research in collaboration with their team and for providing me with the required resources. Finally my heartfelt thanks go to my family and friends for their unwavering support, patience, and understanding throughout this journey. Their encouragement has been a source of strength for me.

Abstract

The inspection of the solder joint in the manufacturing of printed circuit boards (PCBs), ensures the quality and reliability of the product. Traditionally, it is done by manual examination or by training a supervised machine learning model where an expert is used to label the dataset manually. This dependency on labeled data forms a bottleneck in efficiency and, hence, scaling in production. As PCB designs become more complex and the demand for shorter production cycles intensifies, innovative inspection methods must be developed to detect solder-joint defects without using an extensive prelabeled dataset. Our research addresses this challenge using unsupervised learning techniques for solder joint defect detection. We have leveraged the Anomalib library, a comprehensive collection of a wide range of anomaly detection algorithms, to implement and evaluate multiple models like PatchCore and DFM, among others. Our approach involves training or inferencing using these models on an image dataset of solder joints. The models then label the solder joints as normal or abnormal, with a corresponding prediction score for each image. This, therefore, removes the need for a manually labeled training dataset and can be more scalable and adaptable for inspection requirements on PCBs. The PatchCore model provided very promising results, showing an accuracy of 91.25%. During the course of the thesis, we encountered few challenges that provided us with a lot of information regarding the practical application of these techniques. The most prominent ones are related to significant training times for some models, which may hinder their feasibility in real-time manufacturing environments. Looking towards future developments, a more extensive use of Vision Transformers will bring huge potential to this application. Although some ViT models are explored in our current work to some extent, there is much space for a more elaborate exploration of their potential in unsupervised defect detection. Further efforts could thus be oriented toward developing such models, optimized for faster training and inferencing times, and investigating their generalization capability for different solder joint defects.

Contents

Chapter 1

Introduction

This thesis explores and compares the performance of various unsupervised anomaly detection models to automate the identification of faulty soldered pins on Printed Circuit Board (PCB) assemblies. Our goal with this research is to use unsupervised models to eliminate the dataset's time-intensive and costly manual labeling process to train any supervised learning model. We want to find a solution that can substitute the supervised learning model used by Siemens with state-of-the-art anomaly detection techniques. The outcome of this research will be a proposed unsupervised model capable of accurately identifying faults, ultimately improving production efficiency while reducing pseudo-errors and manual intervention.

Traditionally, inspection of solder joints has been carried out manually or through supervised machine learning models that rely on labeled data. However, the manual labeling process for training a supervised model is time-consuming and expensive. Due to this challenge, unsupervised anomaly detection can become a promising solution. This thesis uses Anomalib[1] to explore various unsupervised models. Anomalib is a library of state-of-the-art anomaly detection algorithms, such as PatchCore[2], Deep Feature Modeling (DFM)[3], FastFlow[4] among others. We focus on determining which models best detect defective solder joints, providing a good alternative to the current supervised model.

The dataset we use for this study comprises images of soldered joints on PCB boards provided by Siemens. This dataset includes images showing both the defect-free, called normal(FC) and defective, called anomalous(NG) soldered joints, with dimensions of 512×512 pixels. We divided the dataset into a training set and a testing set, with an 80-20 split for a thorough model evaluation. The experiments carried out on this dataset include benchmarking multiple Anomalib models to evaluate their effectiveness in detecting anomalies without needing labeled data.

Amongst the tested models, PatchCore emerged as the best performing model, achieving an accuracy of 91.25%. Its success can be attributed to its use of locally aware patch features, coreset sampling, and memory bank of nominal patch features, allowing for an efficient comparison between new samples and known defect-free examples. The experiments conducted in this thesis provide a comprehensive

comparison of unsupervised anomaly detection models. The results suggest that while PatchCore is the most effective unsupervised model for detecting solder joint defects, other models like DFM, EfficientAD, and FastFlow also offer unique advantages that could be leveraged depending on specific use-case requirements. Incorporating these models into an industrial setting could significantly improve the fault detection process, reducing the dependency on manual inspection and ultimately leading to more efficient production cycles.

1.1 Current Solution at Siemens

Before presenting the problem statement, we first give an overview of the current supervised solution implemented in Siemens. This current solution is described in the figure 1.1.1. This solution pipeline begins with first capturing high-resolution images of soldered joints on PCB assemblies using an Automated Optical Inspection (AOI) machine, which captures images in a batch process and takes under 5 minutes. These images are sent to a server for processing. Then, the server runs the You Only Look Once (YOLO)v8 model, which has been trained on a manually labeled dataset to detect any defect in the image and classify it as pass or fail. Due to the supervised nature of this model, a significant amount of labeled data is required to achieve higher accuracy, and experts with domain specific knowledge must perform the labeling process of the dataset. This dependency on labeling not only increases costs but also limits the scalability of the inspection process.

The current solution also involves human inspectors who verify the model's recommendations and make final repair decisions. The operator is responsible for verifying the model's prediction and making the final decision on whether a repair is needed or not. This manual intervention, while still necessary to ensure product quality, adds a layer of complexity and delay to the inspection process. As production demands increase, the dependence on human inspectors can become a bottleneck, preventing the desired levels of efficiency and throughput. Moreover, the manual nature of this process makes it difficult to achieve consistent inspection quality, as human inspectors may vary in their judgment and expertise.

Given these challenges, there is a clear need for an alternative approach to overcome the limitations associated with supervised learning. The unsupervised models explored in this thesis offer a promising solution by eliminating the need for labeled data and providing a more flexible, scalable approach to defect detection. Integrating an unsupervised anomaly detection model into the existing pipeline instead of the baseline YOLO model could significantly reduce the time and cost associated with labeling.

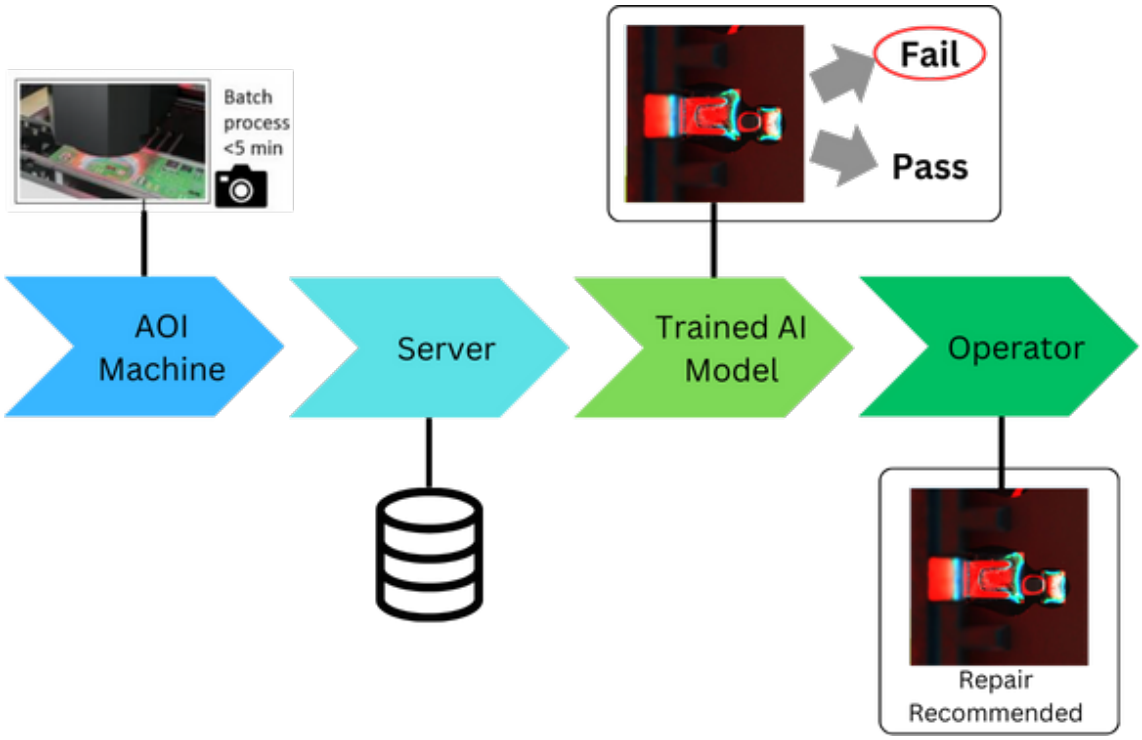

Figure 1.1.1: Current Siemens solution pipeline

1.2 Thesis Objective

The objective of this thesis is to develop a trained, unsupervised anomaly detection model for identifying faulty soldered pins in PCB assemblies. This approach aims to reduce the dependency on manual labeling processes while maintaining high accuracy, thereby improving the scalability of the visual inspection process in electronics manufacturing. Specifically, this research seeks to:

1. Evaluate the feasibility of using unsupervised anomaly detection models to identify defects in solder joints without requiring labeled datasets.
2. Benchmark various models from the Anomalib library, PatchCore, DFM, Deep Feature Kernel Density Estimation (DFKDE), EfficientAD, and FastFlow, to assess their performance in anomaly detection.
3. Optimize the model's performance through hyperparameter tuning and evaluate the models based on accuracy, precision, recall, and F1-score metrics.

Chapter 2

Theoretical Background

2.1 Related Works

Image Classification has experienced significant growth over the last few decades, driven by increased computational capabilities, large amounts of available data, and the development of sophisticated algorithms. This section provides an overview of the major advancements in Image Classification, especially for its applications in industrial environments, such as electronics manufacturing. We will also explore the increasing significance of anomaly detection, emphasizing the transition from conventional supervised learning approaches to modern unsupervised learning methods.

2.1.1 Evolution of Image Classification Techniques

Image Classification is a critical task in the field of Computer Vision (CV), with numerous applications ranging from medical diagnostics to autonomous driving. Early image classification algorithms mainly relied on manual feature extraction, which involves domain experts designing these features, which could be used to differentiate between classes of images. These features were subsequently inputted into traditional machine learning algorithms like Support Vector Machines (SVMs) or k-Nearest Neighbors (K-NN) for classification [5]. Nevertheless, these techniques were limited by their dependence on handcrafted features, which could not effectively reflect intricate changes observed in real-world scenarios.

Subsequently, the introduction of Convolutional Neural Networks (CNNs) was a significant milestone in this field. They have introduced the concept of feature learning, whereby the network learns to extract relevant features from the raw image data using numerous layers of convolutional filters [6]. This groundbreaking study in [6] on AlexNet architecture delivered exceptional results on the ImageNet dataset for image classification. This success triggered various research into network architectures that are deeper and more complex, such as VGGNet [7], GoogLeNet [8], and ResNet [9].

Among these, the work on the introduction of ResNet by [9] was highly significant as it addressed the issues of vanishing gradients that tormented earlier deep networks [7] [8]. ResNet made it possible to train much deeper networks by incorporating residual connections, resulting in substantial improvement in accuracy over various image classification benchmarks. Thus, CNNs were firmly established as the leading approach for tasks like image classification.

2.1.2 Applications of Image Classification in Industrial Settings

In the industrial sector, special attention is given to image classification applications in quality control processes, particularly in automated visual inspection. Traditionally, the visual inspection process was carried out manually, where we had to rely on the experience of quality inspectors to ensure product quality. As a result of the varying levels of expertise among inspectors and the limitations of human abilities, this approach exhibits low efficiency, low accuracy, and inadequate real-time performance on a large-scale manufacturing process [10].

With the rise of Deep Learning (DL), automated visual inspection systems are developed using CNNs for real-time defect detection. One of the most notable models in industrial applications is YOLO, which is used for real-time object detection and has gained popularity in tasks requiring rapid image processing. The design of YOLO enables it to perform object detection in a single iteration over the network, rendering it highly efficient in cases where speed is critical. For example, during the PCB manufacturing process, YOLO can rapidly identify solder joints that do not meet quality standards. This offers a significant decrease in inspection times compared to traditional manual techniques [11].

Although CNN-based models such as YOLO have demonstrated their effectiveness in most industrial applications, they have certain limitations. One major obstacle is the requirement for a large labeled dataset to train such models properly. In many industrial applications, such as PCB inspection, defects are rare, and gathering labeled instances in large enough quantities for training purposes may become prohibitively expensive. In such instances, the labeling process is typically labor- and knowledge-intensive, resulting in high costs and time consumption[12].

2.1.3 Anomaly Detection in Electronics Manufacturing

In certain scenarios, due to the lack of labeled data, supervised learning methods are not feasible. As a result, there has been a growing interest in exploring unsupervised learning methods for anomaly detection. Anomaly detection refers to finding patterns in data that deviate from expected behavior. This makes it a suitable solution in industrial applications for identifying defects where normal instances are well represented, but anomalies are rare and diverse [13].

In electronics manufacturing, anomaly detection is essential for tasks like solder

joint inspection. Traditional techniques are based on probabilistic and distance-based models. These techniques are effective when dealing with simpler anomaly patterns but often face difficulties when dealing with complex and high-dimensional data, as explained by [14]. This article points to the limitation of these methods because they rely on predefined thresholds and basic data assumptions that fail to generalize on a wide variety of anomalies.

Further expanding on this foundation, recent advancements in DL introduced more sophisticated methods, particularly by employing autoencoders[15]. Neural networks, namely Variational Auto-Encoders (VAEs)[16], provide substantial improvements in presenting complex data distributions. These autoencoders learn compression and reconstruction of inputs through training on normal operational data, thereby building a model of 'normality' that allows it to find those anomalous instances with higher reconstruction errors indicating a deviation from the normal[15]. This approach improves the detection of complex patterns of defects to help maintain the integrity of products produced in the manufacturing processes. Other variants of autoencoders, such VAE, deep autoencoders, have been investigated for anomaly detection and have shown promising results [16].

Another increasingly popular method for anomaly detection is the Generative Adversarial Networks (GANs)[17]. GANs consists of two networks, namely a generator and a discriminator, both of which are trained in parallel. The generator tries to produce data that closely resembles the real one, whereas the discriminator strives to differentiate between real and generated data. In anomaly detection, the generator is trained to generate normal instances, while the discriminator learns to identify deviations from this normal distribution as anomalies [18].

2.2 Supervised Image Processing

Supervised image processing is a subset of Machine Learning (ML) where a model is trained using a labeled dataset. Labeled dataset means that each image in that dataset is tagged with its correct output label or category. With this approach, the model can learn to map inputs to specific outputs [19].

2.2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANN) are computational processing systems inspired by how biological nervous systems like the human brain function. ANN mainly consists of numerous interconnected computational nodes, known as neurons, which are intertwined in a distributed fashion to collectively learn from the input and optimize the final output [20].

Figure 2.2.1 shows the basic structure of a ANN. Input data will be loaded as a multidimensional vector into the input layer. Then, it will be distributed into the

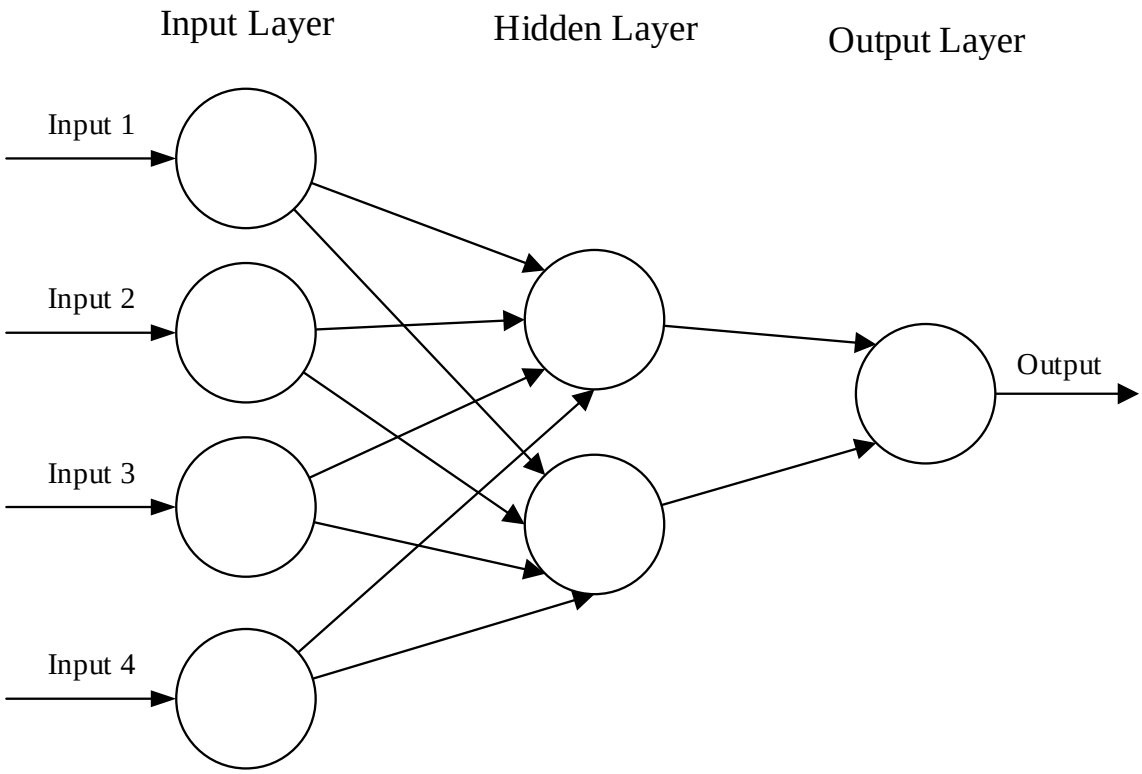

Figure 2.2.1: ANN is a three-layer FNN made up of an input, a hidden, and an output layer [20].

hidden layer. Then, hidden layers will make decisions based on the previous layer and evaluate how a stochastic change improves the final output, this process is known as learning. When multiple hidden layers are stacked next to each other, it is known as DL [20].

2.2.2 Convolutional Neural Networks (CNNs)

CNNs[21] is an extended version of ANN, which is primarily used for feature extraction from a grid-like matrix dataset [22]. CNNs is similar to traditional ANN as they consist of neurons that self-optimize through learning. Each neuron will receive input and perform operations like scalar product followed by a non-linear function, which is the basis of many ANN. The only significant difference between CNNs and traditional ANN is that CNNs are mainly used on images in the field of pattern recognition. This enables the encoding of image-specific features into the architecture, making it more suitable for image-focused tasks while reducing the parameters required for model configuration [20].

The term CNNs indicates that the network uses a mathematical operation called **convolution**. CNNs are neural networks that, in place of general matrix multiplication, use convolution in at least one of the layers [23].

CNN Architecture :

Basic CNN architecture consists of three types of layers they are **convolutional layers**, **pooling layers**, and **fully connected layers**. A CNN architecture is formed when these layers are stacked together. Figure 2.2.2 shows CNN architecture for MNIST[24] classification.

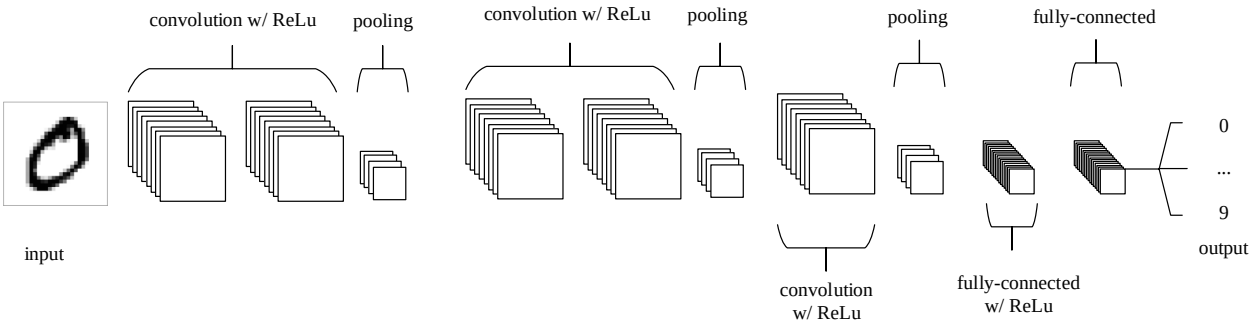

Figure 2.2.2: A common CNN architecture [20]

The functionality of CNN architecture shown in the above figure can be broken down into four key areas.

1. Similar to other types of ANN, the input layer holds the images pixel values [20].

2. The convolutional layer will determine the output of neurons linked to local input regions by calculating the scalar product of their weights and the corresponding input volume region. The Rectified Linear Unit (ReLu) aims to apply an 'elementwise' activation function like sigmoid to the output generated by the preceding layer's activation [20].

3. The pooling layer will perform downsampling along the spatial dimensions of the input. This further reduces the number of parameters in the activation [20].

4. The fully connected layers will then perform the same functions as that in a standard ANN, aiming to produce class scores from the activations for classification purposes. To improve performance, it is also recommended to use ReLu between these layers [20].

By using this simple transformation technique, CNNs can transform the original input layer by layer by using convolutional and downsampling techniques, producing class scores for classification and regression purposes [20]. Let's look at the main components of the CNN architecture in detail.

Convolutional Layer :

The convolutional layer plays an essential role in the CNNs functionality. The layer's parameters concentrate on using learnable **kernels**. These kernels are the set of learnable parameters.

These kernels are usually small in spatial dimensions but spread entirely along the input depth. Upon the data entering a convolutional layer, it convolves each filter across the spatial dimensions of the input to generate a 2D activation map as shown in figure 2.2.3 [20].

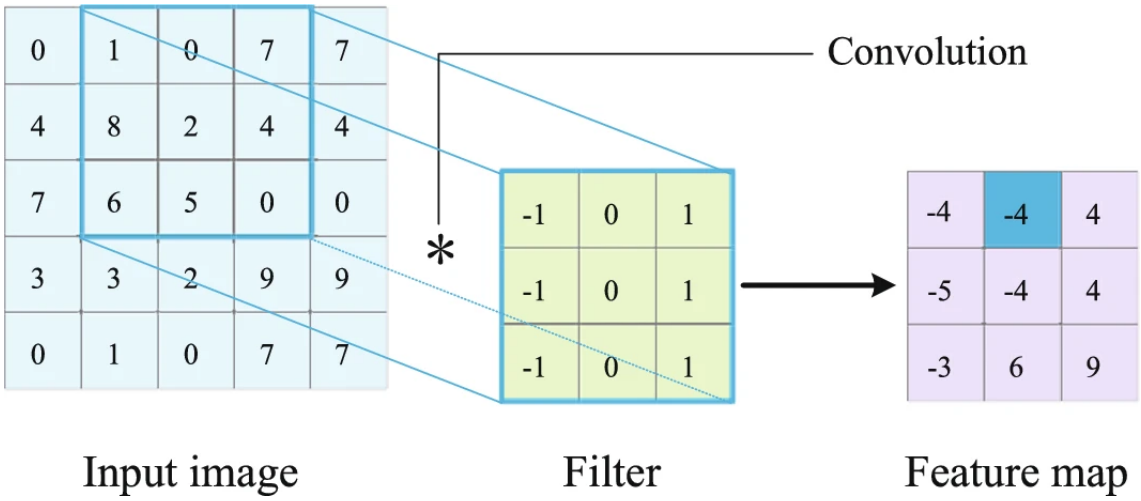

Figure 2.2.3: The convolution operation involves the sliding of a convolution kernel(filter) over the input vector, where the kernel is multiplied by the pixel value at the corresponding positions of the input, and summing them to produce a feature map [25].

As the kernel glides through the input, it calculates the scalar product for each value in that kernel. Through this, the network will learn kernels that activate upon they detect a specific feature at a given spatial position of the input. These are commonly referred to as activations. Each kernel will have a corresponding activation map, stacked along the depth dimension to form a complete output volume from the convolutional layer [20].

To mitigate the problem of ANN where the models get too big to train effectively due to the full-connected nature of standard ANN neurons, each neuron in a convolutional layer is connected only to a small region of the input, known as **receptive field**. The connectivity depth is almost always equal to the input depth [20]. To understand this, let's consider an example: if the network receives an input image measuring $64 \times 64 \times 3$(representing an RGB image), with the receptive field being of the size 6×6, each neuron will have a total of 108 weights within the convolutional layer($6 \times 6 \times 3$, with 3 representing the amount of connectivity across the depth of the volume). Whereas a standard neuron in other forms of ANN would have $12,288$ weights each [20].

Optimizing its output convolutional layers can also significantly reduce the complexity of the model. Three hyperparameters—depth, stride, and setting zero-padding—optimize these layers.

The **depth** of the output volume can be set manually using the number of neurons within the convolutional layer corresponding to the same region in the input. Although reducing this hyperparameter can significantly reduce the total number of neurons

of the network, it can also significantly reduce the model's pattern recognition capabilities [20].

Stride is the number of rows and columns the receptive field will move across the input's spatial dimension [25]. For example, with stride set to 1, then the significantly overlapping receptive field will produce extensive activations. At the same time, a larger value of stride will reduce overlapping and produce an output of lower spatial dimensions [20].

Zero-Padding is a simple process of padding the border of the input with zeros. It is an effective way to enhance control of the dimensionality of the output volumes [20].

We can calculate the spatial dimensionality of convolutional layer output by using the following formula:

$$\frac{(V-R)+2Z}{S+1}$$

Here, V denotes the input volume size(height×width×depth), R denotes the receptive field size, Z denotes the zero-padding set, and S denotes the stride. If the result from the above formula is not a whole integer, then the stride has been set incorrectly, and the neurons cannot fit across the given input [20].

Despite these optimizations, models can still be huge using high-dimensional input like images. To further reduce the number of parameters significantly within the convolutional layer, **parameter sharing** is used. It works on the assumption that if a feature from one spatial region is useful for computation, then it is likely to be useful in another region as well [20].

Pooling layer :

The pooling layers help reduce the dimensionality of the feature representation, thus reducing the number of parameters and computational complexity of the model. The pooling layer operates on each activation map in the input and reduces its dimensionality using the 'Max' function. **Max-pooling layer** is one of the most common types of pooling used in CNNs. It selects the maximum activity value of all neurons to represent the particular region and extracts the most essential feature from the input feature map [25]. A 2×2 kernel is applied with a stride of 2 across the spatial dimensions of the input, reducing the activation map by 75% of the original size while preserving the depth volume of the feature map [20].

Average pooling computes the arithmetic mean of all the elements within the region, resulting in the mean value of the local response from the extracted feature mapping [25]. Figure 2.2.4 shows the max pooling operation where the maximum value is used, and figure 2.2.5 shows the average pooling operation, which performs the average operation to get the value.

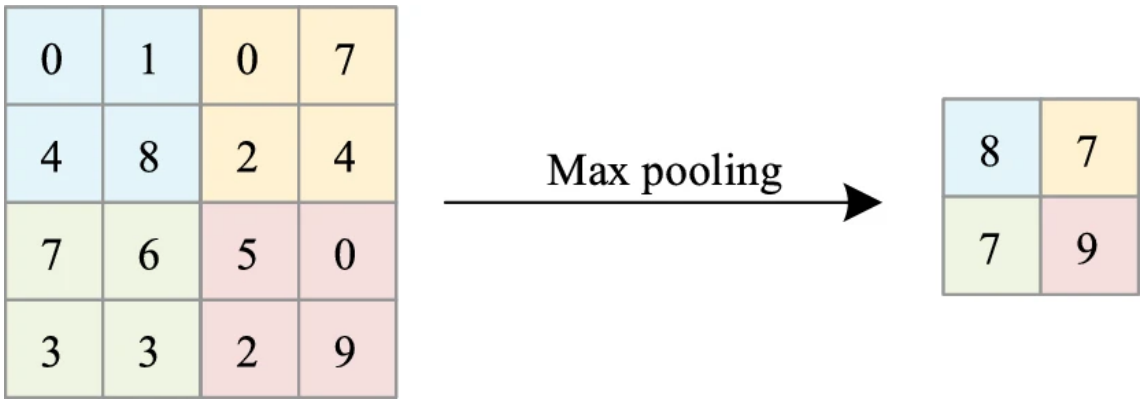

Figure 2.2.4: Max pooling [25]

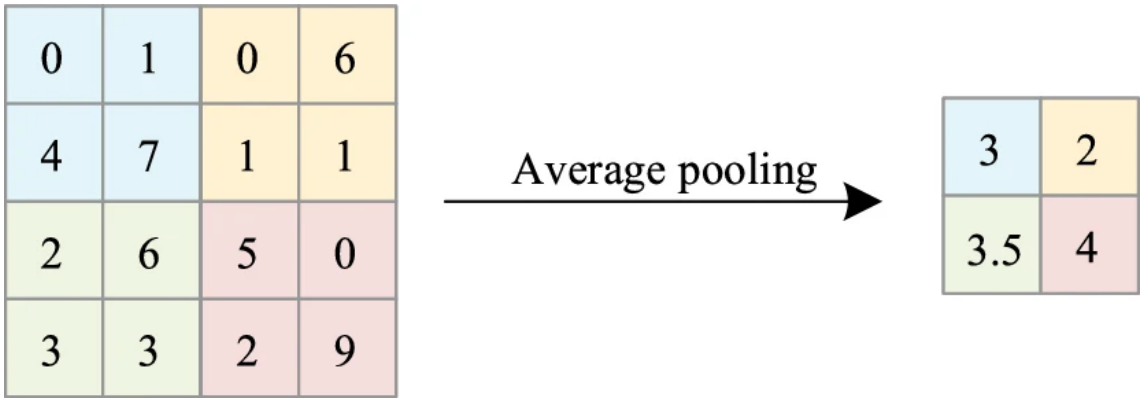

Figure 2.2.5: Average pooling [25]

Fully-Connected Layer :

The fully connected layer contains neurons directly connected to the neurons in the two neighboring layers, with no connections to any neurons within itself. It is similar to the way neurons are arranged in traditional ANN as can be seen in the figure 2.2.1 [20].

Activation Function :

Activation functions are essential to neural networks, strengthening the network's representational and learning capabilities by learning the abstract features through nonlinear transformation [26]. With activation functions, the neural network can approximate any nonlinear function, making it applicable to a wide range of nonlinear models [25]. Some of the activation functions common properties are:

1. It should add the nonlinear curvature in the optimization landscape to enhance training convergence of the network;

2. It should not significantly increase the computational complexity of the model;
3. During training, it should not hamper the gradient flow;
4. It should preserve the data distribution and facilitate better network training [26].

Here, we will look at the most common and widely used activation functions, including Sigmoid, Tanh, Softmax, ReLu, and Leaky ReLu.

1. Sigmoid Activation Function :

The Sigmoid function, or the logistic function, ranges from 0 to 1. As seen from the figure 2.2.6, sigmoid can be used to normalize the output and probability-based predictions [25]. The following is the sigmoid functions mathematical formula :

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

Figure 2.2.6 shows that the sigmoid gradient is smooth, preventing output values from jumping. Nevertheless, Sigmoid has many problems; for example, the chances of the vanishing gradient are high when the activation is near 0 or 1. There is also slow gradient descent convergence due to non-zero-centered output. Finally, due to the sigmoid function's exponential operation, the computation time of the model increases as well [25].

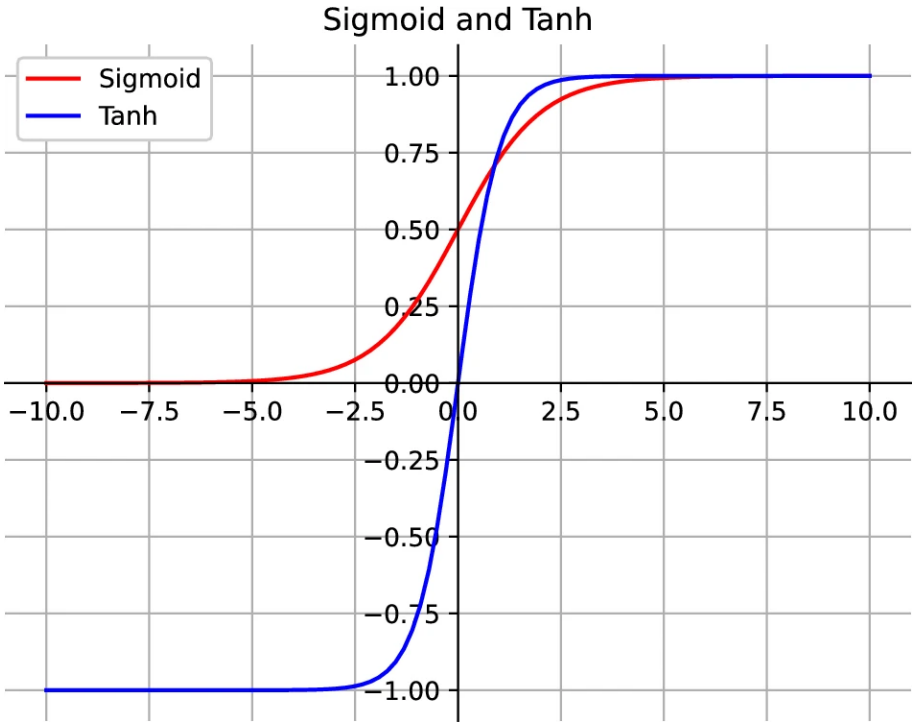

Figure 2.2.6: Sigmoid and Tanh activation function curve [25]

2. Tanh Activation Function :

The hyperbolic tangent activation function(HTAF), or tanh, compresses the input

vector in the range of -1 to 1 and offers a zero-centered output. The below formula and the figure 2.2.6 shows the tanh curve and mathematical representation:

$$f(x) = \frac{2}{1+e^{-2x}} - 1$$

From the figure 2.2.6, we can see that tanh and sigmoid function are relatively similar S-shaped curves. Tanh and Sigmoid have the following relationship:

$$Tanh(x) = 2Sigmoid(2x) - 1$$

In practice, tanh is used more than sigmoid as it solves the sigmoid functions problem of not centering the output to zero. However, like sigmoid, tanh also suffers from vanishing gradient problems for extreme input values [25].

3. Softmax Activation Function :

A softmax activation function is used in multi-class classification problems. It compresses the input vectors into probabilities from 0 to 1, all of which sum up to 1. In the K classification task, the generated probabilities can represent each category, with the larger value indicating the higher probability that it belongs to that particular category [25]. Figure 2.2.7 shows the softmax function curve, and the below formula shows how softmax is formulated mathematically:

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

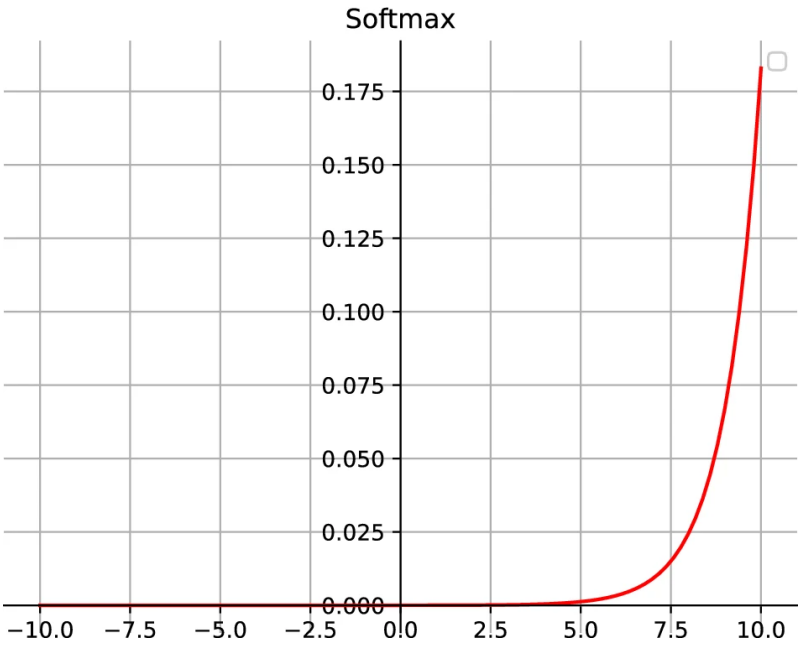

Figure 2.2.7: Softmax activation function curve [25]

The gradient becomes zero when the softmax function encounters a negative input value. Therefore, the weights for activation in that region will not update throughout backpropagation, resulting in a dead neuron that never activated [25].

4. ReLU Activation Function :

ReLU, or Rectified Linear Unit, is a segmented linear function, as shown in figure 2.2.8. It is a fast, simple activation function which is essentially a ramp function given by the following formula:

$$f(x) = max(0, x)$$

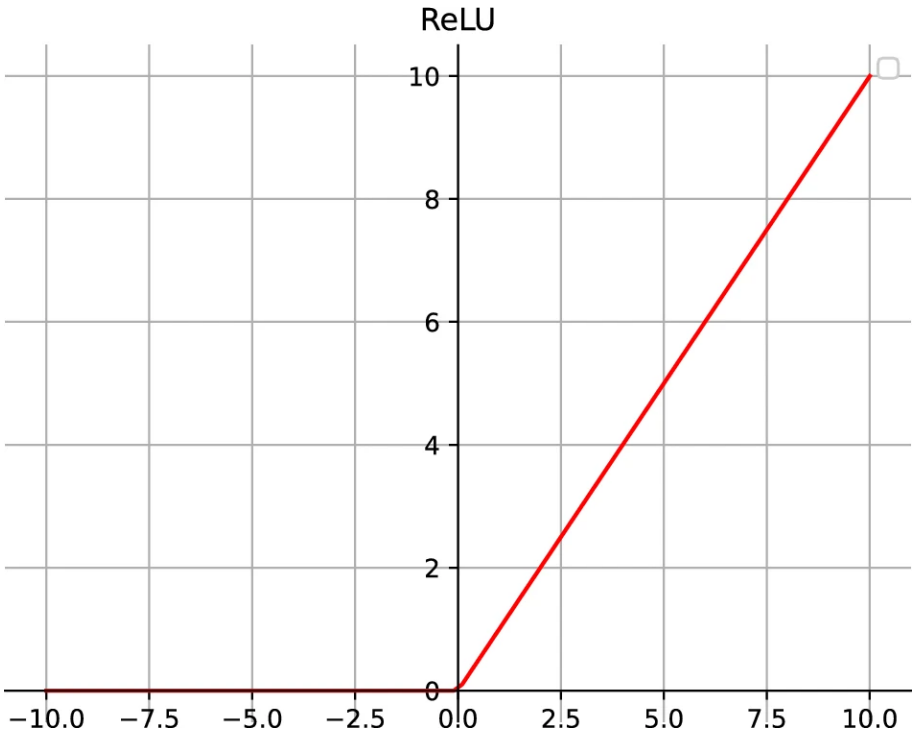

Figure 2.2.8: ReLU function curve [25]

For positive input, the derivative is 1, which improves the vanishing gradient problem and speeds up the gradient descent convergence. It is also faster than the sigmoid and tanh functions because the ReLU function only has linear relationships. However, this function suffers from a dying ReLU problem, which is when the input is negative, the gradient will be precisely zero, and the neurons will most likely die during training [25].

5. Leaky ReLU Activation Function :

Leaky ReLU addresses the dying ReLU problem to some extent by introducing very small linear components for negative inputs to solve zero gradients associated with negatives and extending the range of ReLU. Although Leaky ReLU has all the features of ReLU, in practice, it is not always the case that Leaky ReLU is better than ReLU

[25]. Below is the mathematical formulation of Leaky ReLU, and figure 2.2.9 shows its function curve.

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

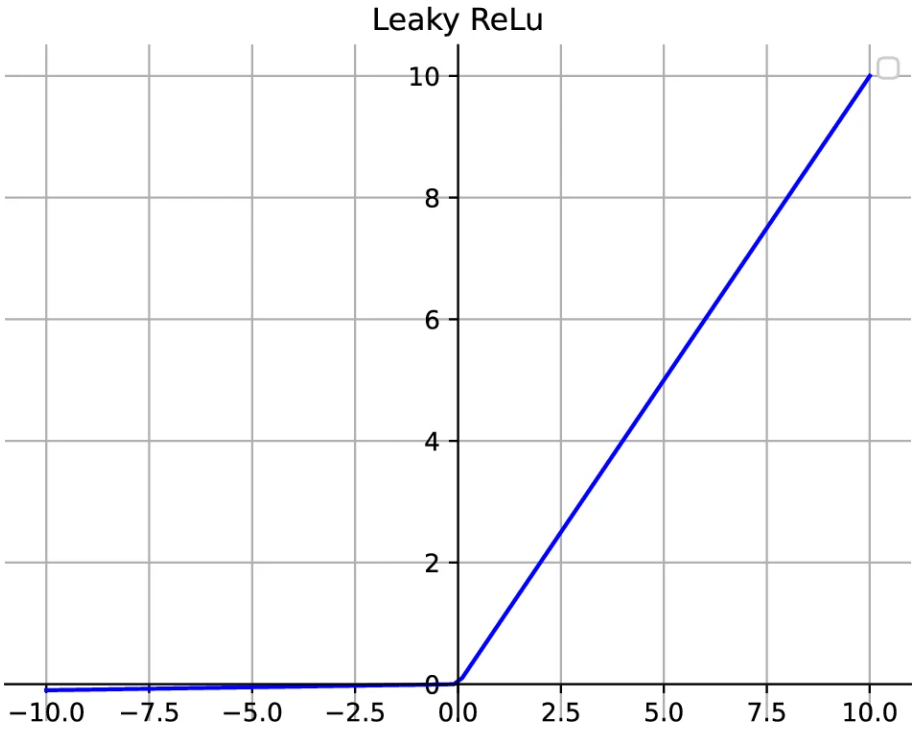

Figure 2.2.9: Leaky ReLU function curve [25]

2.2.3 ResNet

Deep Convolutional neural networks have led to numerous advancements in image classification. The network's depth plays a crucial role in this achievement, but deeper networks still face a critical problem of **degradation**. The degradation problem occurs when deeper neural networks start to converge: with increasing network depth, the accuracy gets saturated and then rapidly declines. This is caused by an increase in number of layers, which results in higher training errors. To address this degradation problem, a deep residual learning framework was proposed. In this framework, instead of relying on each stacked layer to directly match a specific underlying mapping, the layers are designed to fit a residual mapping [9].

Residual Learning :

As mentioned above, instead of expecting stacked layers to approximate mapping function H(x), where x denotes the inputs to the first of this stacked layer. The

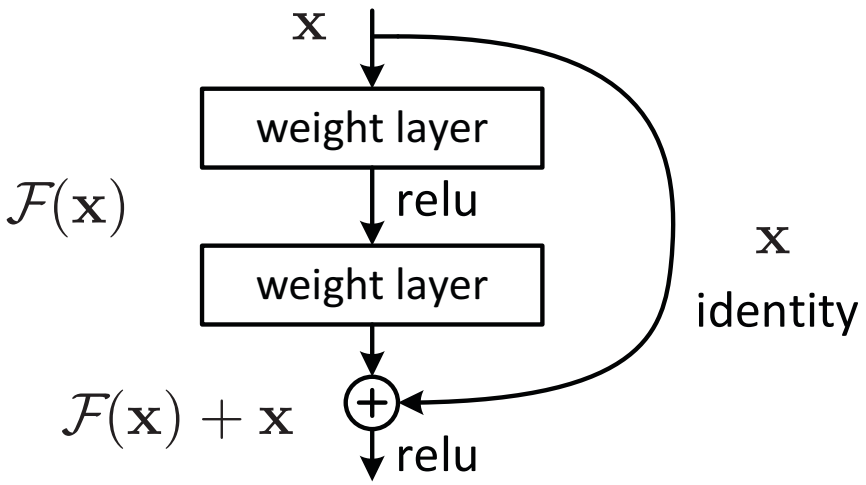

Figure 2.2.10: A residual block [9]

layers are explicitly let to approximate a residual function $F(x) := H(x) - x$. The original function then becomes $F(x)+x$. This makes the learning process much easier, although it is expected that both forms can asymptotically approximate the desired functions [9].

The above mentioned residual learning is applied to every few stacked layers. The structure of a residual block is shown in the figure 2.2.10. The basic building block can be defined for this approach as:

$y = F(x, \{W_i\}) + x.$

Here, the input and output vectors are represented by x and y of the considered layers; the function $F(x, \{W_i\})$ is the residual mapping that needs to be learned. The operation $F + x$ is carried out by a **shortcut connection**, also known as skip connection, and element-wise addition as shown in figure 2.2.10. This shortcut connection does not introduce any extra parameters or computational complexity except for the minor element-wise addition. This ensures a fair comparison between plain and residual networks, which have the same number of parameters, width, depth, and computational cost [9].

Network Architecture :

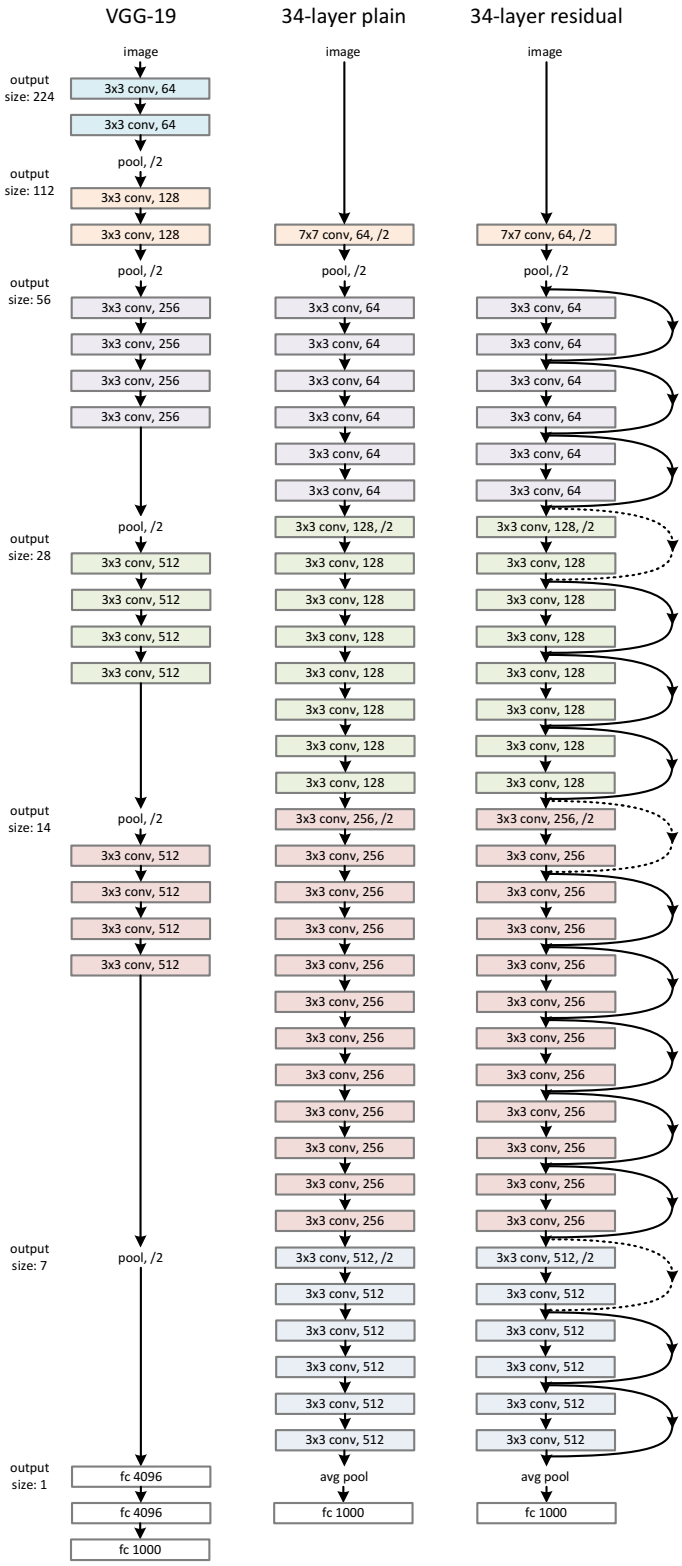

Figure 2.2.11: Network architecture example: Left: VGG-19 model serves as a reference. Middle: a plain network consisting of 34 parameter layers. Right: a residual network with 34 parameter layers [9].

Two models are described in the figure 2.2.11. The plain baseline network(figure 2.2.11, middle) draws inspiration from the philosophy of VGG nets[7] (Figure 2.2.11, left), which mainly uses 3×3 filters. It maintains the time complexity per layer by doubling the number of filters when the feature map size is halved. Downsampling is performed by convolutional layer with stride 2. The network finishes with a global average pooling layer followed by a 1000-way fully connected layer with softmax [9].

Short connections are added based on the plain network, which transforms the network into its counterpart residual version. Identity shortcuts match input-output dimensions(solid line shortcuts in figure 2.2.11). For when dimensions increase (dotted line shortcuts in figure 2.2.11), either the shortcut performs identity mapping, with zero-padding, or projection shortcuts(1×1 convolutions) are used [9].

Performance :

The 18-layer and 34-layer Residual Neural Network (ResNet) show that the deeper 34-layer ResNet performs better than the 18-layer ResNet by 2.8%. Also, the 34-layer ResNet significantly reduces training error and shows good generalization to the validation data. This shows that the degradation problem is well tackled in this setting, leading to accuracy gains from increased depth. Compared to the plain network(figure 2.2.11, middle), the 34-layer ResNet reduced the top-1 error by 3.5%; this verifies the efficacy of residual learning on very deep networks. It was also observed that 18-layer plain networks are more accurate, but the 18-layer ResNet converges faster [9].

Deeper Bottleneck Architectures :

To scale the networks to 50, 101, and 152 layers, a bottleneck design is used, where the building blocks are modified as bottleneck layers, in which the residual function is stacked with three layers(1×1, 3×3, 1×1 convolutions). The 1×1 are responsible for dimensions reduction filled by restoration, which creates a bottleneck in the 3×3 layer with smaller input and output dimensions [9].

50-layer ResNet: In this architecture, each 2-layer block in a 34-layer network is replaced with a 3-layered bottleneck block, which results in a 50-layer ResNet [9].

101-layer & 152-layer ResNets: 3-layer blocks are used for the construction of 101-layer and 152-layer ResNets. Even when the depth significantly increases, the 152-layer ResNet still has lower complexity than VGG-16/19 networks [9].

The 50/101/152-layer ResNets show significantly higher accuracy compared to the 34-layer versions. No degradation problem is observed, due to which high accuracy can be achieved [9].

2.2.4 Vision Transformer

Transformers introduced by [27] has become the go-to model in Natural Language Processing (NLP) related tasks. Transformer's computational efficiency and scalability allow for exceptional model training with over 100B parameters. However, when it comes to CV, convolutional architectures like ResNet[9] are still dominant. Therefore, inspired by the success of transformers in NLP, the standard transformer was directly applied to images with minimal modifications. For this, images were split into patches, and then the transformer was given the linear embeddings of these patches as input. These image patches are treated the same way as a token(words) as in NLP tasks, and the training is carried out in a supervised fashion [28].

Architecture :

Figure 2.2.12 shows an overview of the model's architecture. The standard transformer takes a 1D sequence of token embeddings as input. To process 2D images, firstly, the image is reshaped into flattened 2D patches called the patch embeddings [28].

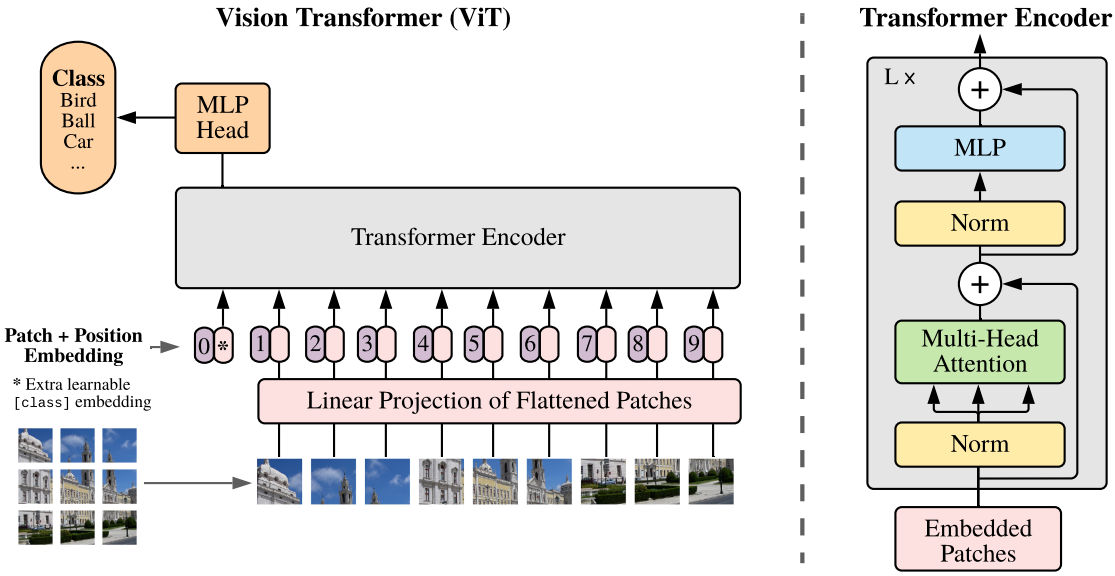

Figure 2.2.12: This is the proposed ViT architecture. Here, an image is split into fixed-sized patches, each of them is then linearly embedded, with that positional embedding is added, and then the resulting sequence is given as input to a standard transformer encoder. A "classification token" is added to the sequence to perform classification [28].

Similar to BERT's [class] token, a learnable classification token is prepended to the sequence of the patch embeddings. Its output from the transformer encoder is used as image representation for classification. During both training and fine-tuning, a classification head is attached to this output, which consists of a Multi-Layer

Perceptron (MLP) with one hidden layer during pre-training and a single linear layer at fine-tuning time. To keep the positional information, positional embeddings are added along with the patch embeddings. The transformer encoder, as seen on the right side of the figure 2.2.12, has alternating layers of Multiheaded Self-Attention (MSA) and MLP blocks. Each block also includes Layer Normalization (LN), and residual connections, these are applied before each block and after each block respectively. The MLP consists of two layers with a GELU activation function. These layers process the sequence of patch embeddings and the [class] token [28].

2.2.5 Data-efficient image Transformers (DeiT)

Data-Efficient Image transformer (DeiT) was introduced in the paper [29], which, unlike ViT[28] does not need large-scale training and uses only Imagenet as a training set, and also need fewer computational resources. DeiT introduced a token-based distillation strategy to distill information into the model [29].

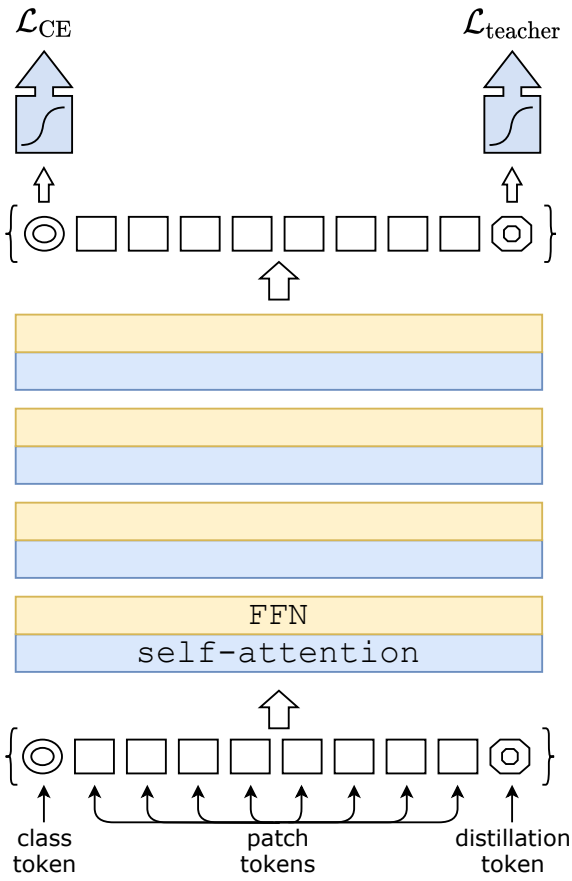

Figure 2.2.13: DeiT distillation procedure[29].

Distillation strategy :

For a teacher model, a strong image classifier is used, it can be either a CNN or a mixture of classifiers. One key innovation in DeiT is the introduction of a "distillation token". This procedure is shown in figure 2.2.13. This distillation token is used similarly to the class token, which is added to the initial embeddings. It interacts with other tokens using the self-attention. The distillation token embedding is output by the network after the last layer and optimized by using the distillation component of the loss. This distillation embedding helps the model learn from the teacher's output, which remains complimentary to the class embedding [29].

2.2.6 Class-Attention in Image Transformers (CaiT)

Class-Attention in Image Transformer (CaiT) is a vision transformer type. It makes two main modifications to the original ViT architecture. Firstly, a new layer scaling method termed LayerScale is used, which adds a learnable diagonal matrix at the output of each residual block. This matrix is initialized close to 0, which improves the training dynamics. Secondly, class-attention layers are also introduced. This creates an architecture in which the transformer layers, which involve self-attention between the patches, are separated from the class-attention layers, which are devoted to extracting the content of the processed patches into a single vector, so it can be linear classifiers input [30].

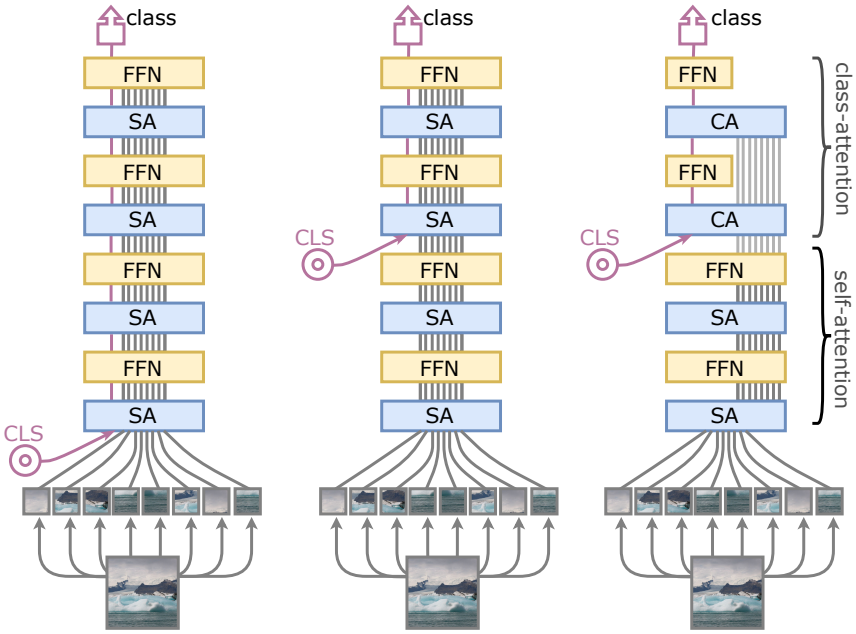

Figure 2.2.14: In this figure, the original ViT is on the left, here the CLS token is inserted with patch embeddings. This is counterproductive, as the same weights are used for two purposes. In the middle architecture, the author shows that inserting the CLS token later on improves performance. Moreover, further in the CaiT architecture(right), the patch embeddings are frozen when inserting a CLS token to save computational resources. So the last part of the network, usually the last two layers, is fully focused on summarizing the information that will be given as an input to the linear classifier [30].

Architecture :

The CaiT consists of two processing stages, as seen in the figure 2.2.14.

Self-Attention Stage : In this stage, the CLS class token is inserted later in the transformer. This resolves the inconsistency of the first layers of the transformer, which are then entirely used for performing self-attention between patches only [30].

Class-Attention Stage : A set of layers compiles the patch embeddings into a class embedding CLS that is then given to a linear classifier. Here, only the class embeddings are updated. However, the main difference is that, within CaiT architecture, the information is not copied from class embedding to the patch embeddings during the forward pass.

2.2.7 You Only Look Once (YOLO)

The human visual system operates with amazing speed and accuracy, which helps us perform complex tasks like driving with minimal conscious thought. Similarly, fast and accurate algorithms for object detection could facilitate computers driving a car without the need for specialized sensors and assistive devices to provide real-time scene information to human users. This could pave the way for better responsive robotic systems. The object detection algorithms before YOLO used classifiers and repurposed them to detect objects by evaluating them across different locations and scaling them in a test image. These methods were relatively slow and complex to optimize [11].

YOLO approaches object detection by treating it as a single regression problem, where it directly mapping image pixels to bounding box coordinates and class probabilities. That means you only look once (YOLO) at an image to predict and present what the object is and what its location is. As shown in the figure 2.2.15, a single CNN can predict multiple bounding boxes and, at the same time, their class probabilities. It is trained on full images and focuses on optimizing detection performance directly [11].

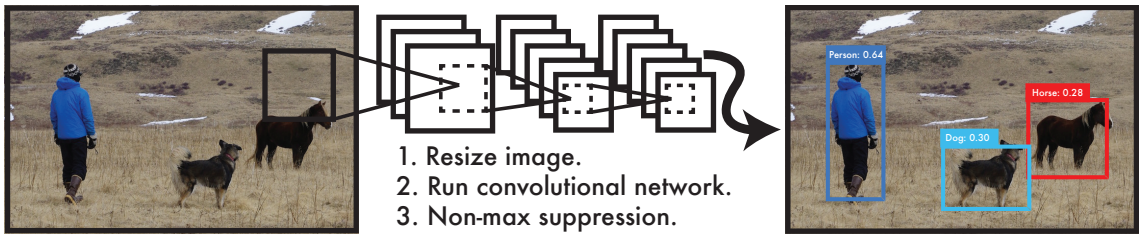

Figure 2.2.15: YOLO Detection System: YOLO makes the image processing very simple. It first resizes the image, then runs a single CNN on an image, and then gives thresholding value to the detections based on the model's confidence [11].

How YOLO works :

As stated, YOLO unifies the separate object detection components into a single neural network. This network uses features from the complete image to predict all bounding boxes across all classes simultaneously. YOLO enables training from start to finish and achieves real-time speeds, all while ensuring a high level of average precision. YOLO divides the input image into an $S \times S$ grid. Every grid cell is responsible for detecting an object whose center falls into that grid cell. Every grid cell predicts bounding boxes and its confidence scores. This score reflects the model's confidence in the box containing an object and how accurate the predicted box is. If the object is detected, the confidence score should be equal to Intersection Over Union (IOU) between the ground truth and the predicted box; if no object is detected, the confidence score should be zero. At the test time, the conditional class probabilities and the box confidence predictions are multiplied to get the final detection as shown in the figure 2.2.16 [11].

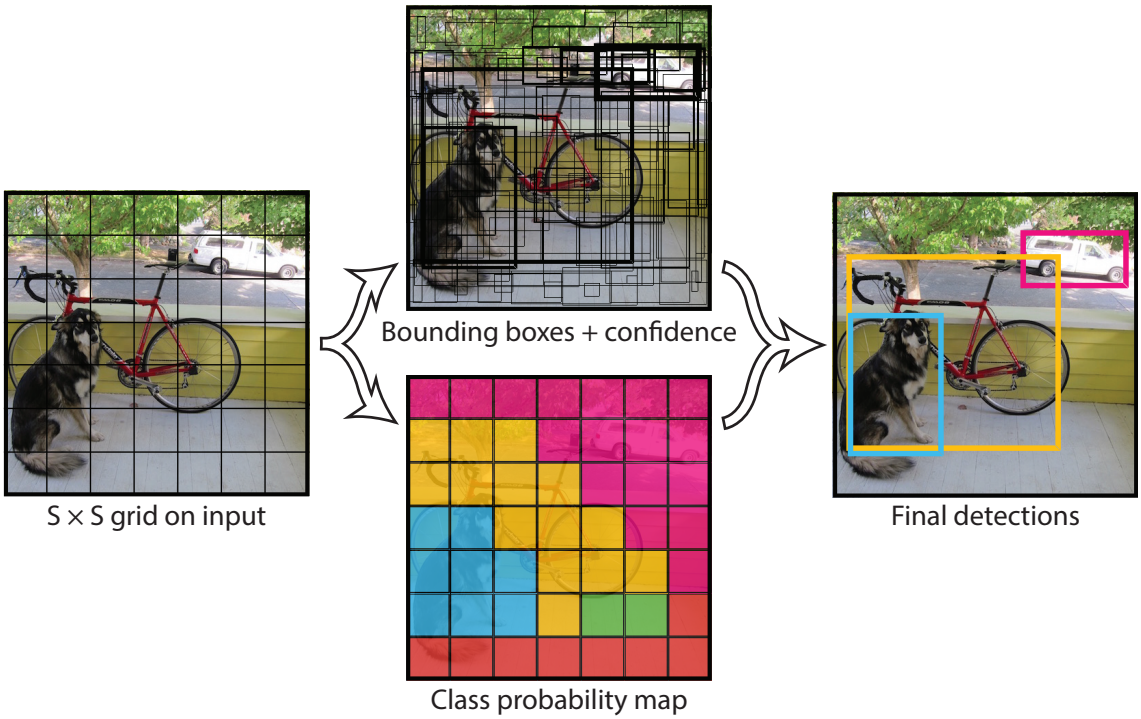

Figure 2.2.16: This figure shows the working of YOLO model [11].

Limitations of YOLOv1 :

YOLO imposes significant spatial constraints on bounding box predictions by allowing each grid cell to predict only one class and two bounding boxes. This reduces the model's ability to predict multiple nearby objects, like small objects that can appear in a group like flocks of birds [11].

YOLO struggles when generalizing objects in new or unusual aspect ratios or configurations. Lastly, YOLO's loss function does not differentiate between small and large bounding boxes. A small error in a large box does not have much impact, but a small error in a small box affects the IOU. The leading cause of this problem is the YOLO's incorrect localization [11].

YOLO's versions:

YOLOv2[31], also known as YOLO9000. It integrated several existing techniques of that time. The entire object detection architecture was converted into a full convolutional network, which helps achieve high accuracy and speed. Later, the high-resolution and low-resolution features were combined, and an anchor-based prediction method was adopted. Due to its simple input and output formats, YOLOv2 remains one of the mainly used object detection methods in the maintenance and development of various industrial settings, particularly on low-end devices which has very limited computing capabilities [32].

YOLOv3[33], integrated advanced technology of the existing object detection and made the necessary optimizations to one-stage object detectors. YOLOv3 has architecture which combines Feature Pyramid Network (FPN), allowing for simultaneous predictions across multiple scales. YOLOv3 made notable changes to the label assignment task. There are two changes in the YOLOv3. The first change involves assigning a ground truth to a single anchor, and the second change involves transitioning from soft label to hard label for IOU-aware objectness. YOLOv3 is still the most popular version of YOLO series [32].

Scaled-YOLOv4 [34], can be used for edge and clound computing both. Due to the efforts of the DarkNet and PyTorch YOLOv3 communities, scaled-YOLOv4 can forgo the pre-training steps necessary with ImageNet and instead directly use a train-from-scratch method to achieve high-quality object detection outcomes. Scaled-YOLOv4 has introduced CSPNet into Path Aggregation Network (PAN), significantly improving the speed, parameters, accuracy, and number of calculations. Scaled-YOLOv4 also introduced model scaling methods for different edge devices and offers three types of models: P5, P6, and P7. During training, it also used the decoder and label assignment strategy introduced in the initial version of YOLOv5 [32].

There are now many more such YOLO versions with incremental updates such as YOLOv5 [35], YOLOv8 [36] which is used in the current solution at Siemens as mentioned in section 1.1 and latest being YOLOv10 [37].

2.3 Unsupervised Image Processing

Unsupervised image processing is also part of ML, but unlike the supervised learning we saw in section 2.2, it does not require any labeled data. Instead, it focuses on clustering similar data points, discovering patterns and relationships, or detecting anomalies without explicitly telling it where the defect is [19]. This approach is beneficial when labeled data is scarce or expensive to obtain.

Below, we will discuss and understand the unsupervised models we used in our experiments.

2.3.1 PatchCore

PatchCore is a state-of-the-art approach developed for the efficient detection of anomalies in industrial settings, especially when there is a lack of defective samples or they are undefined. This approach has been specifically tailored to tackle the challenges of the cold-start problem in which models are exclusively trained on non-defective(nominal) images. PatchCore excels by utilizing a memory bank of nominal patch features and techniques like locally aware patch features and coreset subsampling, which are explained below [2]. Figure 2.3.1 shows the overview of the PatchCore model.

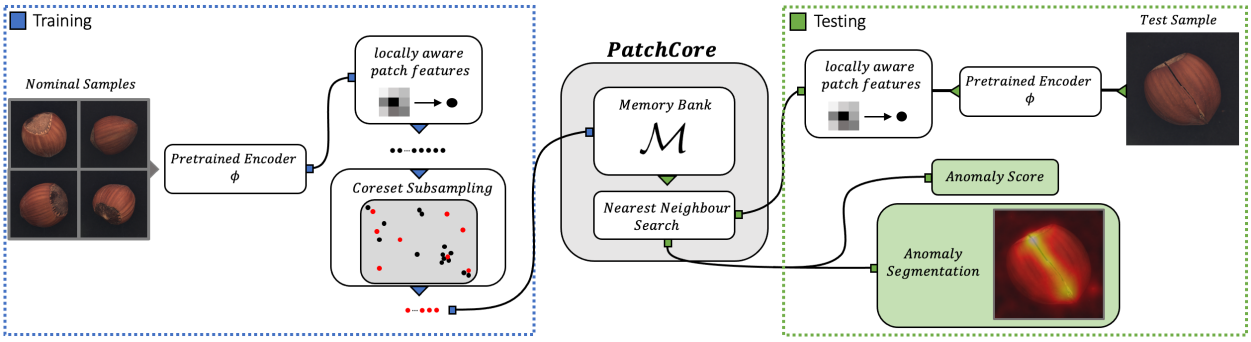

Figure 2.3.1: Overview of PatchCore: The nominal samples are decomposed into a memory bank of neighborhood-aware patch-level features. To minimize redundancy and the time required for inference, this memory bank is downsampled using greedy coreset subsampling. During the testing phase, images are classified as anomalies if any single patch is anomalous, and pixel-level anomaly segmentation is generated by assigning a score to each patch feature [2].

Locally Aware Patch Features :

The key advancement of PatchCore is the utilization of locally aware patch features. Contrary to traditional approaches that utilize global image features, PatchCore targets local patches of the image. It extracts the mid-level features that capture the contextual and spatial relationships within these patches. By maintaining local awareness, the model can preserve important details that might get lost when replying on the generalized global features [2].

The locally aware features are derived from the intermediate layers of a pre-trained CNN [21], namely WideResNet-50 [38]. Focusing on mid-level features lets PatchCore avoid the pitfalls of over-generalization and inherent ImageNet class bias, which are common problems when relying on deeper, high-level features. The outcome is a more nuanced and contextually rich representation of the nominal data, which is essential for identifying subtle anomalies that could otherwise go unnoticed [2].

Memory Bank and Coreset Subsampling :

PatchCores memory bank is built using these locally aware patch features, serving as a repository of nominal patch features, which are then used to compare against test images. However, handling a large memory bank in real-time industrial applications can be computationally expensive. In order to mitigate this issue, PatchCore uses coreset subsampling, a method that reduces the memory bank size by selecting the most representative features without compromising the model's performance [2].

Coreset subsampling employs a greedy selection algorithm, which ensures that the patches retained in the memory bank most accurately represent the overall distribution of the nominal data. The decrease in memory bank size is essential for achieving faster inference times, making PatchCore both accurate and highly efficient [2].

Anomaly Detection and Localization :

During testing, PatchCore compares the patch features of a test image and the features stored in the memory bank. The metric Euclidean distance is used to compare each patch in the test image with the nearest patch in the memory bank. If any patch shows a significant deviation from the stored nominal patches, the image is flagged as anomalous. The anomaly score, which serves as a robust measure for anomaly detection, is determined by calculating the maximum distance observed across all patches [2].

PatchCore extends this approach to localize anomalies by creating a detailed segmentation map. A score is assigned to each patch in the test image depending on its proximity to the nearest nominal patch. These scores are subsequently mapped back to the original image, resulting in a localization map that precisely highlights the areas where the anomalies occur [2]. This functionality is especially essential in industrial settings, where the ability to not only detect anomalies but also accurately determine their exact positions for quality control and remediation.

Performance and Applications :

PatchCore has been rigorously tested across multiple benchmark datasets, such as MVTec AD dataset [39], where it delivered exceptional performance, achieving an Area Under the Receiver-Operating Curve (AUROC) of up to 99.6%. This result significantly improves existing methods, effectively reducing detection error rates by half. Due to its ability to achieve high accuracy with minimal training data, PatchCore is especially well-suited for industrial settings where it can be challenging to gather a large number of defective samples [2].

2.3.2 Deep Feature Modeling (DFM)

DFM is an efficient method for anomaly detection by utilizing DL to extract feature representations and to model the distribution of normal data. This method excels in situations where the anomalies are rare and not well-defined, making it highly valuable for a range of industrial applications [3].

The key concept of DFM involves utilizing Deep Neural Networks (DNN) high-dimensional features from normal data and subsequently modeling the distribution of these features. The assumption is that the anomalies will significantly deviate from the learned distribution, facilitating accurate detection. DFM leverages the ability of DL to automatically learn complex feature representations from raw data, in contrast to traditional methods that rely on manually crafted features [3].

DFM is based on the notion that the normal data can be well represented in a lower dimensional feature space, while the anomalies stand out as outliers. By modeling the distribution of normal data features, DFM can accurately identify points that deviate

from the norm as anomalies [3].

Feature Extraction and Representation :

In DFM, a DNN most often a CNN is used to extract features from the input data. These features are usually extracted from the intermediate layers of the network, which capture varying levels of abstraction, ranging from low-level edges and textures to more complex semantic information. The model's performance can be greatly influenced by the choice of network architecture and the specific layer from which features are extracted [39].

These features are then utilized to create a feature representation space. In this space, the normal data points are expected to cluster together, whereas anomalies should appear distant from these clusters. To enhance visualization and anomaly detection[3], the dimensionality of this feature space is often reduced using methods like Principal Component Analysis (PCA)[40] or t-Distributed Stochastic Neighbor Embedding (t-SNE)[41].

Density Estimation and Anomaly Scoring :

DFM uses density estimation methods to model the distribution of normal data after constructing the feature representation space during the training process. Two commonly used methods for this purpose are Gaussian Mixture Models (GMM) or Kernel Density Estimation (KDE), which estimate the probability density function of normal data. The idea here is that the areas in the feature space that have a high density of normal data, whereas regions with a low concentration indicate possible anomalies [3].

A likelihood-based anomaly score is assigned to each new data point based on its deviation from the modeled distribution. Anomalies are identified by flagging points with low likelihood under the modeled distribution. This method is highly effective because it does not require labeled data for anomalies. It is well-suited for unsupervised learning situations where only normal data is accessible during training [3].

Anomaly Detection and Evaluation :

During testing, the trained model is given new data points to extract their feature representations, which are then evaluated against the density model to calculate an anomaly score. Again, anomalies are identified as points for which the model assigns low likelihood [3].

DFM has been extensively tested on several benchmark datasets, proving its effectiveness in detecting anomalies across various domains. The evaluation of the model's performance is done using metrics such as the AUROC, Area Under the

Precision-Recall Curve (AUPR), and accuracy. DFM demonstrated competitive results with AUROC score of about 98%, outperforming traditional methods for anomaly detection [3].

2.3.3 EfficientAD

EfficientAD is an effective anomaly detection model for real-world computer vision applications focusing on computational efficiency and a lightweight feature extractor that can process an image under a millisecond using a modern GPU. EfficientAD employs a student-teacher (S-T) approach to detect anomalous features. It establishes new standards for detecting and localizing the anomalies, coupled with its low error rate. This makes EffienctAD an economical option for real-world applications [42].

Core Concept :

EffientAD sets new benchmarks for both the accuracy of anomaly detection and the speed of inference. In order to detect anomalous features, a S-T approach is used, where the student network is trained to predict features computed by a pre-trained teacher network on normal training images. EffienctAD introduces training loss that prevents the student network from imitating the teacher network beyond the normal images. This loss enables the use of efficient network architecture for both the student and the teacher while improving anomaly detection [42]. The components of EfficientAD can be divided into three main components:

1. Efficient Patch Description Network (PDN),
2. Lightweight Student-Teacher model,
3. Autoencoder for Logical Anomaly Detection,

Efficient Patch Description Network (PDN) :

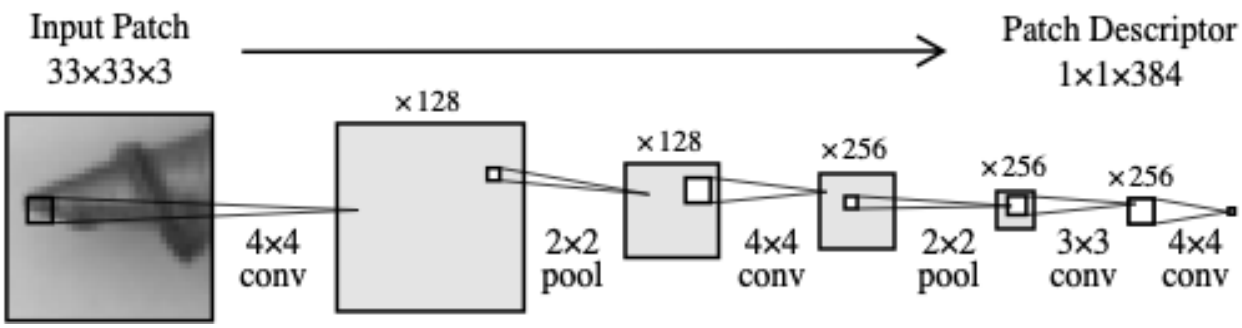

Figure 2.3.2: PDN architecture for EfficientAD. All features can be obtained in a single forward pass using it on an image in a fully convolutional manner [42].

Recent anomaly detection methods often employ deep pre-trained networks for feature extraction. EffienctAD employs a feature extractor network of drastically reduced depth. This feature extractor is referred to as PDN, and its architecture, as shown in figure 2.3.2, contains only four convolutional layers. The receptive field of each output neuron is 33×33 pixels, which makes PDN possible to generate all feature vectors in a single forward pass for images of different sizes [42].

To overcome the large feature map size, high computational costs, and memory requirement problems, PDN performs early stage downsampling by strided average-pooling layers after the first and second convolutional layers as shown in figure 2.3.2. To generate expressive features, PDN is trained by distilling a deep pre-trained classification network into it. PDN trained in images from ImageNet by reducing the mean squared difference between its output and the features extracted from the pre-trained network. Besides higher efficiency, this has another benefit in that the features generated by PDN only depend on local pixels, avoiding long-range dependencies in other methods. This ensures that an anomaly detected in one area of the image cannot activate anomalous feature vectors in other distant areas, which can compromise the localization of anomalies [42].

Lightweight Student-Teacher model :

For anomalous feature vector detection, EfficientAD uses a S-T approach, with the teacher being the PDN. The student network also uses the teacher's architecture to maintain low overall latency. The EfficientAD model introduces a training loss, substantially improving anomaly detection without affecting the computational test time requirements. The goal is to provide the student with enough data so that it can mimic the teacher on normal images while avoiding generalization on anomalous images. Therefore, the student's loss is restricted to the most relevant parts of an image, i.e., the patches where the student mimics the teacher the least. A hard feature loss is proposed, which primarily uses the output elements with the highest loss for the purpose of backpropagation [42].

Apart from hard feature loss, EfficientAD adds a loss penalty during training that further hinders the student's ability to imitate the teacher on images that are not included in the normal training images. The teacher network is either pre-trained on an image classification dataset or is a distilled version of a pre-trained network. However, the student is only trained on the application's normal images. To hinder the student's ability to generalize its imitation of the teacher to out-of-distribution images, a penalty in terms of using images from the teacher's pretraining during the training of student is proposed [42].

Autoencoder for Logical Anomaly Detection :

For learning logical anomalies of the training images and detecting any violations of these constraints, EfficientAD uses an autoencoder on training images. Figure

2.3.3 illustrates the anomaly detection pipeline for EfficientAD. It consists of S-T pair and an autoencoder. The autoencoder is trained to predict the output generated by teacher's output. A standard convolutional autoencoder is used, which consists of strided convolutions in the encoder and bilinear upsampling in the decoder [42].

Usually, for both the logical anomalies as well as normal images, the autoencoder fails to generate the accurate latent code needed for reconstruction of the image in the teacher's feature space, as can be seen in the figure 2.3.3. Using the difference between the teacher's output and the autoencoder's reconstruction as an anomaly map might cause false-positive detections. To avoid this, the student's output channels are doubled and trained to predict the output of the autoencoder in addition to the output of the teacher [42].

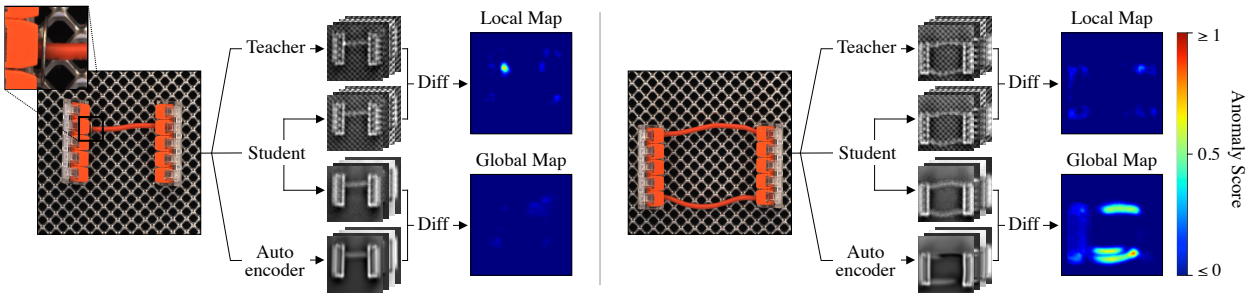

Figure 2.3.3: EfficientAD pipeline

The student learns the systematic reconstruction errors of the autoencoder on normal images. However, it does not learn the reconstruction errors for anomalies as they are not part of the training set. This makes the difference between the student's and the autoencoder's output suitable for computing the anomaly map, which is calculated as the squared difference between the two outputs and then averaged across all channels. EfficientAD produces two anomaly maps, the one generated by the autoencoder-student pair is called a global anomaly map, and the one generated by S-T pair is called a local anomaly map as seen in the figure 2.3.3. These two maps are then normalized to a similar scale and averaged to generate the combined anomaly map. Its maximum value is used as the image-level anomaly score [42].

Performance :

EfficientAD achieves an impressive image-level AUROC score of 99.8% on MVTec AD dataset[39] with early stopping enabled. For the overall anomaly detection performance, EfficientAD achieves very strong image-level detection and pixel-level localization performance with the highest score of 98.1 for VisA[43] dataset. The computational cost of EfficientAD was measured using the metrics latency and throughput, with the EfficientAD showing latency of 2.2ms and throughput of 614 img/s [42].

2.3.4 FastFlow

Most existing representation-based methods use a deep convolutional neural network to extract normal image features and then characterize this distribution through non-parametric distribution estimation methods. However, these methods fail to map features effectively to a tractable base distribution and ignore the relationship between the local and global features, which are essential for anomaly detection. Therefore, FastFlow is proposed to mitigate these problems. FastFlow is an unsupervised anomaly detection and localization model which is built with 2D normalization flows as its probability distribution estimator. Experimental findings show that FastFlow outperforms previous state-of-the-art methods in terms of both accuracy and inference efficiency. It achieves 99.4% AUC in anomaly detection [4].

Core Concept :

Earlier unsupervised approaches used non-parametric methods, while recent ones started using normalization flow to model the distribution of features for normal images. However, these one-dimensional normalization flow models require flattening of the two-dimensional input feature into a one-dimensional vector to estimate the distribution, which destroys the positional relationship of the 2D image and limits the ability of the flow model. These models also used a sliding window approach to extract features from a large number of image patches and detect anomalies for each patch. This led to high inference complexity. Therefore, to address the above problems, FastFlow extends the normalizing flow to two-dimensional space by using fully connected neural networks as the subnet, which can maintain the relative position of the space and support end-to-end inference of the whole image. This improves the anomaly detection performances and gives the detection and localization results at once for the whole image to improve inference efficiency [4].

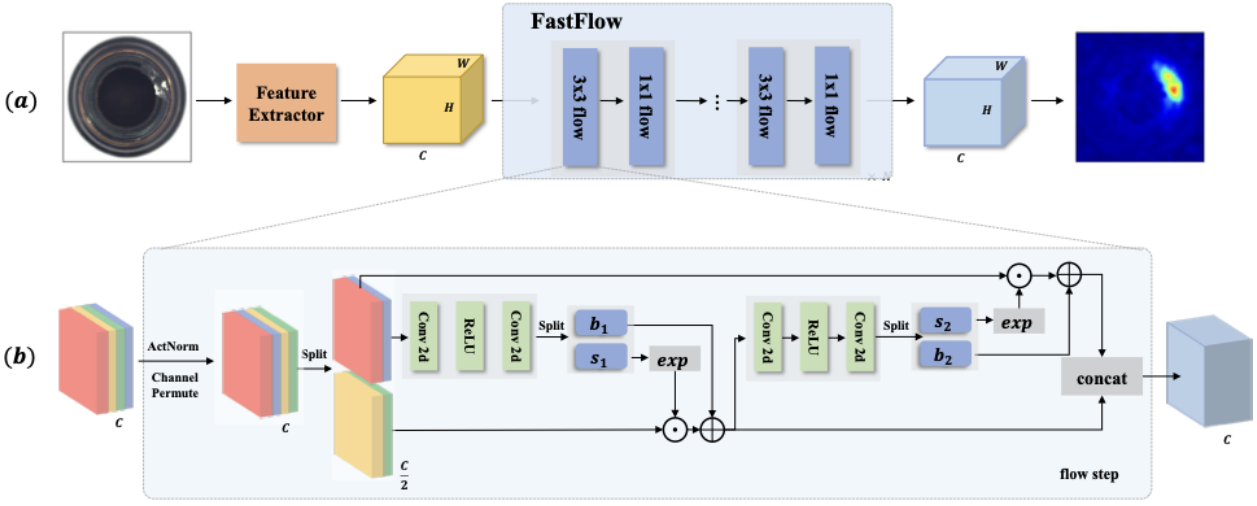

Figure 2.3.4: FastFlow pipeline[4]

Figure 2.3.4 shows the FastFlow pipeline, in which first, the visual features are extracted using the feature extractor and then passed as an input to the FastFlow module for probability density estimation. During training, FastFlow is trained using normal images to convert the normal distribution into a standard normal distribution in a 2D manner. For inferencing, an anomaly score is assigned to each location on the two-dimensional feature based on its probability values [4].

The FastFlow pipeline consists of two main components, they are:

1. Feature Extractor,
2. 2D Normalization Flow

Feature Extractor :

The first step in the whole pipeline involves extracting representative features from the input image using either ResNet as explained in section 2.2.3 or ViT. In ViTs as a feature extractor, features from only one layer are extracted because of their capability to capture the relationship between local patches and global features. In the case of ResNet, features are taken directly from the last layer in the first three blocks, and then these features are inputted into three corresponding FastFlow models [4].

2D Normalization Flow :

The 2D flow function is used to project the image features into the hidden variable using a bijective invertible mapping. At the time of inference, the features of anomalous images should be out of distribution and, therefore, will have lower likelihoods than normal images. This likelihood can be used as the anomaly score. Specifically, the sum of the 2D probabilities of each channel is done to obtain the final probability map, and then it is upsampled to the input image resolution using bilinear interpolation. In order to convert the original normalization flow into a 2D format, alternate 3×3 and 1×1 convolutional layers are used in the default subnet as shown in the figure2.3.4 to retain the spatial information in the flow model and the loIn the feature extraction stage, the features are extracted using DNN as the backbone, such as ResNet[mention reference], which was pre-trained on ImageNet[44] dataset [45]., CIFAR-10[46], BTAD[47] datasets. For the MVTec AD dataset, the FastFlow model was compared with many state-of-the-art models with two metrics image-level Area Under the Curve (AUC) and pixel-level AUC. FastFlow demonstrates exceptional performance in anomaly detection, achieving a 99.4 image-level AUC and 98.5 pixel-level AUC, surpassing all the other models. In the case of the CIFAR-10 dataset, FastFlow achieves an AUC 66.7, which is the best performing model when compared with others. For the BTAD dataset, FastFlow again surpasses all the compared models and achieves a pixel-level AUC of 97.0 [4].

2.3.5 Deep Feature Kernel Density Estimation (DFKDE)

DFKDE is a fast one-class anomaly detection model. It consists of two stages:

1. Feature extraction stage,

2. Anomaly detection stage,

In the feature extraction stage, the features are extracted using DNN as the backbone, such as ResNet[mention reference], which was pre-trained on ImageNet[44] dataset [45]. The penultimate layer, which is the average pooling layer of the backbone, is used to obtain a semantic feature vector of a fixed length of 2048 [1].

In the anomaly detection stage, once the features are extracted, then these features undergo dimensionality reduction using PCA[48] to get the first 16 principal components. PCA is among the simple and most straightforward ways of doing dimensionality reduction. It is a technique that reduces high-dimensional data into a lower-dimensional representation by using the dependencies between variables without losing too much information [49]. After the features are reduced using PCA, then on these reduced features, Gaussian KDE is applied. The main idea behind KDE is that the training datasets will follow some random distribution, and this distribution can be modeled by employing KDE [45].

During inference, if a lower probability density is observed below the threshold determined by the training dataset, this indicates the existence of an anomaly compared to the data distribution learned from the training data [45]. The DFKDE gives competitive results on MVTec AD[39], for the metric AUROC DFKDE gives the highest of 96.5 score [1].

Chapter 3

Methods

This section outlines the experimental procedures and processes carried out to conduct the research study. It includes datasets, the process of training & testing the model, and metrics used to evaluate the method's performance.

3.1 Dataset

The dataset provided by Siemens consists of images of Solder joints on a PCB. It has a total of 6014 images, each of 512×512 dimensions, divided into Fully Connected (FC) and Not Good (NG). The FC class contains images without defects, while the NG class contains images with various defects. The images are of two types, one is a colored image, as shown on the left of figures 3.1.1 & 3.1.2, and the other is a heat image, which is the red one on the right of the figures 3.1.1 & 3.1.2. This variation of images makes it suitable for evaluating various images for which a model can be used. The dataset is split into training and testing sets using an 80-20 split. A separate set of 160 images is also reserved for testing, finding accuracy, and visualizing classification and segmentation results.

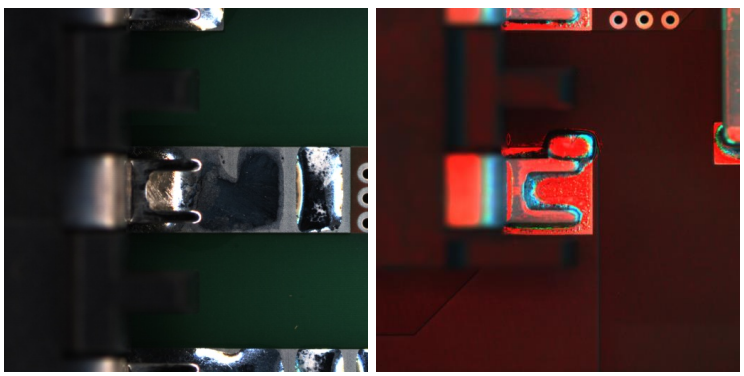

Figure 3.1.1: Examples of defective solder joints in the dataset.

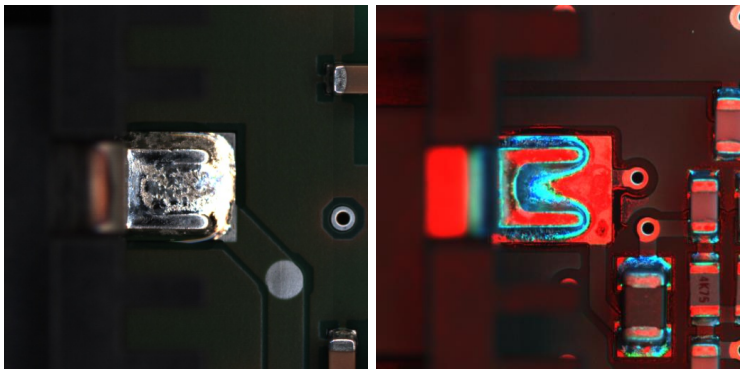

Figure 3.1.2: Examples of defect-free solder joints in the dataset.

As can be seen in figure 3.1.1 and figure 3.1.2, finding a defect without expertise can be challenging and time-consuming.

The table 3.1 provides a detailed breakdown of the dataset distribution across subsets:

Dataset	**FC (No Defects)**	**NG (Defective)**	**Total**
Training	2,971	1,712	4,683
Testing	743	428	1,171
Custom Test	80	80	160

Table 3.1: Distribution of Images in the Dataset

3.2 Experiment

Experiments were carried out using models mentioned in the section 2.3. All the models belong to Anomaly detection and are part of a deep learning library which has a collection of state-of-the-art anomaly detection algorithms called Anomalib. There are multiple steps taken in this experiment and they will be explained below.

3.2.1 Data Loading

To load and preprocess our dataset, we use Anomalib dataloader by importing **Folder** from **Data** class, specifically designed for handling our custom dataset, allowing for efficient data management and preprocessing. We create a Folder datamodule and configure several parameters as described below.

Name : It is the dataset configuration identifier, reflecting the specific dataset used for the experiment.

Root Directory : It points to the directory containing the organized dataset folders.

Normal & Abnormal Directories : Models mostly require anomalous free images for training, which in our case is FC, and anomalous images for testing(not used while

training), which is NG. So those are provided in these two fields.

Normal Split Ratio : The datamodule employs a 0.2 split ratio for validating normal images, which ensures a sufficient number of samples for model tuning and validation.

Image Size : All images are resized to 256 × 256 pixels during preprocessing, balancing the need for detailed feature representation with computational efficiency.

Batch Size : Both training and evaluation processes use different batch sizes for different models to optimize Graphics Processing Unit (GPU) memory usage while maintaining effective gradient updates.

Task Type : There are two types of tasks available: classification and segmentation. For this thesis, for all the models, we have set the parameter to "CLASSIFICATION" and it directs the model to perform binary classification between the FC and NG categories.

Number of Workers : Based on the GPU and the model being used, the number of workers varied between 0 to 8 for helping the computational resources to parallelize data loading.

3.2.2 Model Training or Inferencing

The training takes place in two steps here: first is loading the model, and second is the training part, which is explained below.

Model Loading : Firstly, the model of our choice is imported. The models used in this thesis are explained in 2.3. Each model is initialized with a specific backbone, such as 'ResNet18', 'ResNet50', 'WideResNet50' etc. These backbones are critical for feature extraction. The selection of the backbone and the specific layers used for feature extraction are the key factors that influence the model's ability to detect anomalies.

Model Training or Inferencing : Once the model is loaded, the next steps involve training or inferencing using the **Engine** class from Anomalib. Depending on the model we need to either train the model or just perform inferencing(extracting the features). The engine is configured to manage the complete process, ensuring that the model is trained effectively and can accurately infer from the data provided.

The key components of training/inferencing configuration are:

Thresholding : A threshold value determines the anomalous label for the calculated anomaly scores. There are two types of thresholding methods available; one is '**ManualThreshold**' where we have to set our own threshold value; this is beneficial in situations where there is a lack of appropriate representative validation data[50]. The second one we have used is '**F1AdaptiveThreshold**'. The algorithm optimizes the value of the threshold to determine the optimal f1_score and saves the calculated value of the adaptive threshold.[1]

Task Type : The task is usually set to '**CLASSIFICATION**', which directs the model to categorize images into predefined classes (ex., normal and anomalous). Depending on the requirements, the task can also be configured to '**SEGMENTATION**' to pinpoint

where the defect is there in the image.

Image Metrics : The model's performance is evaluated by '**AUROC**' and '**F1Score**'. These are standard metrics used for assessing classification accuracy and balance between precision and recall. These metrics are explained in section 3.4.

Accelerator : Depending on the available resources, the engine can run on a **'GPU'**, **'Central Processing Unit (CPU)'**, **'Tensor Processing Unit (TPU)'**, **'Intelligence Processing unit (IPU)'**, also there is an **'auto'** which chooses the appropriate resource automatically based on the availability.

Devices : This configuration allows us to provide the engine with the available number of devices(ex., CPUs, GPUs) to use during training and inference. The **'auto'** option sets the optimal devices automatically for the engine.

Validation Frequency : After every epoch **'check_val_every_n_epoch=1'**, this option validates the model, ensuring that the performance metrics are tracked throughout the training process.

Max Epochs : Depending on the model we are using, it can either required to be trained or do inferencing on. When the model performs inferencing at that time, the **'max_epochs'** parameter is always set to **1**. Moreover, the models requiring training can be adjusted depending on the complexity of the model.

Validation Check Interval : The parameter **'val_check_interval'** sets when the validation checks are performed, in our case, it's set to **1.0**, meaning the validation checks are performed after every epoch.

After providing these training configurations to the engine class, we can call the **fit** function to start with the training or inferencing. Here, we provide the model and the datamodule components. Here, the **Normal** directory is used, which is the anomaly free images. After the training/inferencing is finished, the next step is to perform testing, which is explained in the section below.

3.2.3 Testing

After the training/inferencing is finished, we can move to the next critical step, i.e., to evaluate the model's performance. For this, the **test** function from the engine class is used, and similar to the fit function, we again need to pass the model and the datamodule components. Here, the **Abnormal** directory is used, which has the anomalous images.

After the testing, its results are displayed, with two metrics mentioned in section 3.4 AUROC and F1-Score.

3.2.4 Model Exporting

Exporting a trained model is a very important task, as the model can then be deployed in real-world applications. This process involves converting the trained model into a format optimized for efficient inference on various hardware, like CPUs, GPUs etc. Thus, the model can be incorporated into production environments, operating at scale by processing new data and identifying anomalies in real time.

The Anomalib library lets us convert the models into formats like OpenVINO, ONNX, etc. We export our models into OpenVINO format. OpenVINO is an open-source software toolkit used to optimize and deploy DL models. It minimizes resource requirements and effectively deploys on various platforms like the cloud. OpenVINO™ enables inference on several hardware platforms, including CPUs, GPUs.[51]

The export process is initialized by setting the desired export format via the **'ExportType'** variable. The trained model and the destination where the model should be exported are then passed to the **'Export'** function present in the engine class, which is responsible for the conversion. In the specified export location, three files are saved there:

1. 'metadata.json', which stores the threshold value as mentioned in the section 3.2.2 based on which the classification can be done.

2. 'model.bin' is a binary file that contains the weights and biases of the model.

3. 'model.xml' this file stores the model's architecture. It contains the neural network structure, including layers, the connections between them, etc.

3.3 Inference

In the inference phase, the model is utilized to make predictions on new and unseen data. This section documents the approaches for achieving inference using a trained model, mainly focusing on two tasks: **'Image Classification'** and **'Image Segmentation'**. Both tasks are crucial in assessing the effectiveness of the model in detecting and localizing anomalies in solder joints on PCBs.

Firstly, the trained model is loaded using the **'OpenVINOInferencer'** or **'TorchInferencer'** class, which can enable efficient inference on CPUs or GPUs respectively. Here, we will be using OpenVINOInferencer by providing the model and metadata paths of the trained model as explained in the section 3.2.4. Once we are done creating the **'Inferencer'** object, we can move on to performing image classification and segmentation as described below.

3.3.1 Image Classification

Image Classification is an important task within CV because it helps identify and recognize the object present in an image. It involves labeling input images based on their likelihood of being anomalous or not [52].

As mentioned in section 3.3, after creating the inference object then, we pass the input image to its **'predict'** function, which calculates the predicted label and the anomalous score. The predicted results are then visualized using **'ImageVisualizer'** class, which is configured for **CLASSIFICATION** mode, so it overlays the classification results on the input image, making it simpler to analyze the results.

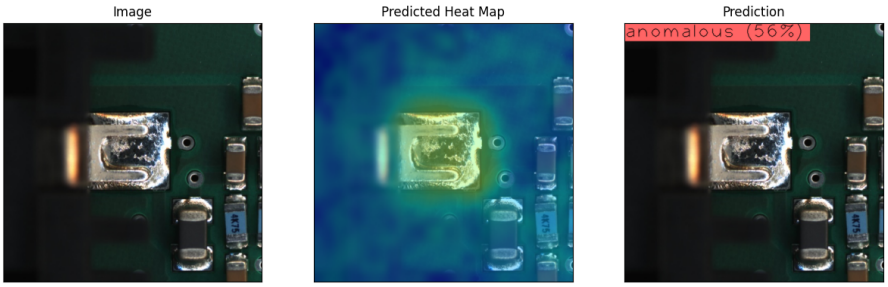

Figure 3.3.1: Image Classification on NG image

Figure 3.3.1 shows the result of image classification, where it gives three images as an output: first one is the input image, the second one shows the predicted heat map, which shows where the possible anomaly is and the third gives the prediction label and the score.

3.3.2 Image Segmentation

Image segmentation is a method of analyzing images at the pixel level. It involves utilizing multiple techniques to label each pixel as part of a particular class or instance[53]. So, using image segmentation, we can point where the model has predicted the anomaly to be as shown in the figure 3.3.2.

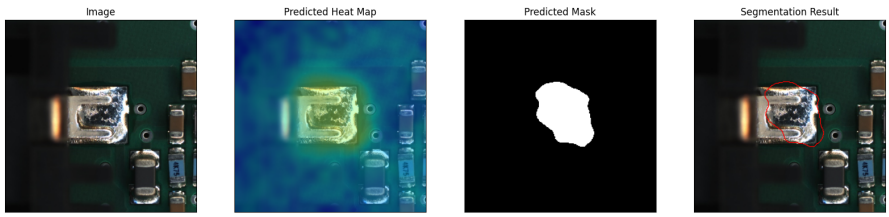

Figure 3.3.2: Image Segmentation on NG image

3.4 Evaluation Metrics

Evaluation metrics play a crucial role in determining how well the ML models have performed. These metrics aid in quantifying the model's ability to differentiate between anomalous and normal images, offering valuable insights into the model's effectiveness and areas for improvement. This section outlines the primary evaluation metrics used in this thesis: AUROC, Accuracy, Recall, Precision, F1-Score, and Confusion Matrix.

Area Under the Receiver-Operating Curve (AUROC)

The AUROC measures the ability of a classification model to differentiate across different classes. It has been particularly useful in measuring trade-offs of the True Positive Rate (TPR) (sensitivity) versus the False Positive Rate (FPR) (1-specificity) across different threshold settings. A higher AUROC value indicates better model performance, with a value of 1 representing a perfect model and a value of 0.5 indicating that the model that performs no better than random chance[54]. This is extremely important for quality control applications because missing an anomaly can have severe consequences for a manufacturing process.

Accuracy

Accuracy is the rate of correct predictions made by the model. It is often assessed using an unseen dataset that was never utilized throughout the learning process[55], like the custom test set mentioned in the table 3.1. It is expressed as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad [56] \tag{3.1}$$

Where TP, TN, FP, and FN are True Positives, True Negatives, False Positives, and False Negatives, respectively. It gives an overall view of the performance of the model.

Recall

Recall, also known as sensitivity, refers to the ratio of accurately predicted positive cases to the total number of actual positive cases [57]. It is expressed as:

$$\text{Recall} = \frac{\text{TP}}{TP + FN} \quad [57] \tag{3.2}$$

A higher Recall value means that the model is good at detecting most anomalies, but it does not consider the number of FP.

Precision

Precision, also known as confidence, refers to the proportion of accurately predicted positive cases that are actually correct [57]. It is expressed as:

$$\text{Precision} = \frac{\text{TP}}{TP + FP} \quad [57] \tag{3.3}$$

This metric is very important where the cost of false positives is high, as it indicates how well the model can prevent the incorrect labeling of normal cases as anomalies.

F1-Score

The F1-score is a metric that achieves a balance between precision and recall. The calculation involves taking the harmonic mean of precision and recall. The F1-score is a valuable metric for achieving a trade-off between high precision and recall. It effectively penalizes extreme negative values of either component [56]. It is expressed as:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad [56] \tag{3.4}$$

It makes this metric valuable since it evaluates a models capability to detect anomalies(Recall) and its precision in identifying true positives without an excessive number of false positives.

Confusion Matrix

A confusion matrix simply shows the predicted and actual classifications made by the model [55]. It is a table containing the TP, TN, FP, and FN values. Subsequently, a Confusion Matrix can help explain the errors a model makes so that improvements can be made specifically within the same area.

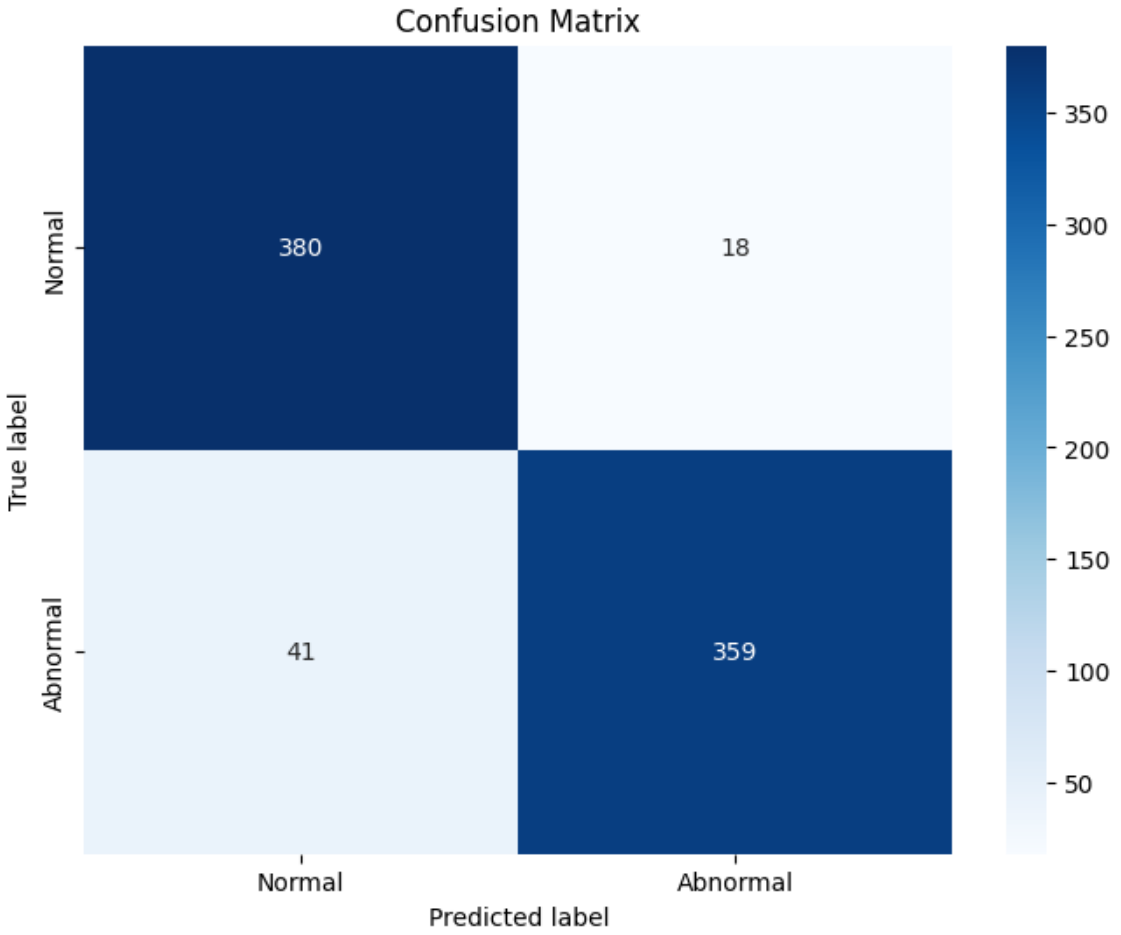

Figure 3.4.1: Example of Confusion Matrix

Figure 3.4.1 shows a confusion matrix from one of our models. Here, the TP is the top-left corner dark blue cell, TN is the bottom-right dark blue cell, FP is the bottom-left light blue cell, and FN is the upper-right light blue cell.

Chapter 4

Results

This section presents the detailed findings and results of our experiments using various unsupervised anomaly detection models for the task of identifying faulty solder joints in PCB. The models which we have evaluated here are PatchCore(section 2.3.1), DFM(section 2.3.2), DFKDE(section 2.3.5), EfficientAD(section 2.3.3), FastFlow(section 2.3.4), which uses different hyperparameters, feature extraction backbones and its layers. We aim to study the effectiveness of these models in comparison to the current baseline models used by Siemens YOLO(section 2.2.7) while also the factors affecting their performances.

4.1 Overall Performance of Models

The table 4.1 and the bar chart 4.1.1 represent each model's overall accuracy performance, giving an initial overview of how these models compare in terms of how they classify anomalies correctly.

Model	**Accuracy**
Yolov8-L	**95%**
Yolov8-M	**93.75%**
PatchCore	**91.25%**
Deep Feature Modeling(DFM)	72.85%
Deep Feature Kernel Density Estimation(DFKDE)	70.71%
FastFlow	66.87%
EfficientAD	60%

Table 4.1: Overall comparison of different models accuracy

The chart 4.1.1 provides a graphical representation of the same data of each model's accuracy, where we can clearly visualize the performance of different models. As can be seen in table 4.1 and bar chart 4.1.1 the baseline model YOLOv8-L achieves the highest accuracy of 95%, while **the best performing unsupervised model is**

PatchCore closely following with an accuracy of 91.25%, with only a 2.5% difference in the accuracy of baseline model YOLOv8-M. Other models, such as DFM, DFKDE, FastFlow, and EfficientAD, show moderate performance. Here, none of the unsupervised anomaly detection models could overtake the supervised baseline model. One of the reasons for this is that the baseline model was trained in a supervised fashion, where it had access to all the labels of the image dataset. Whereas the anomaly detection models were not given any labels for the images, and while training, only anomaly free data was used, it had to figure out on its own which images were anomalous and which images were normal.

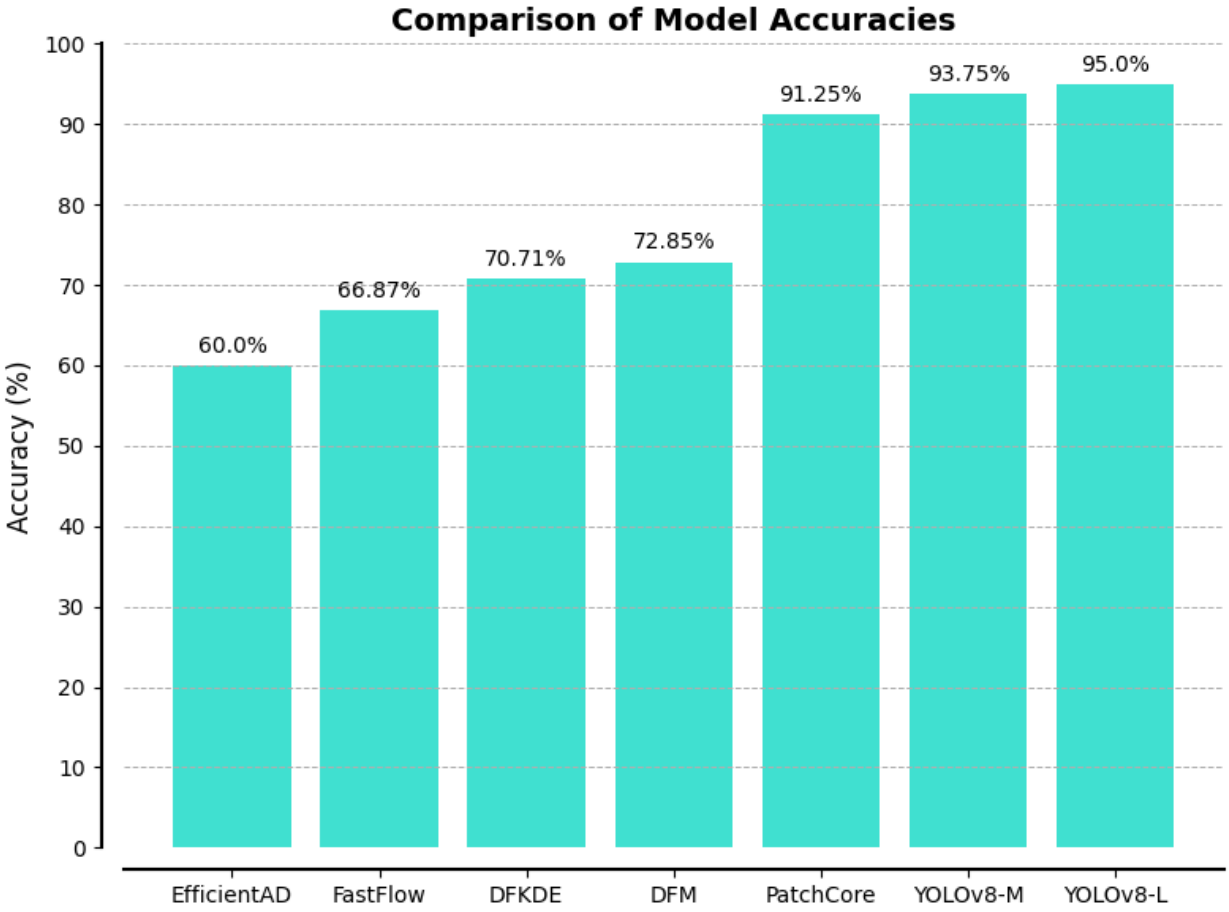

Figure 4.1.1: Bar chart representation of the different models accuracy like YOLO(baseline), our approach including PatchCore, DFM, DFKDE, FastFlow, and EfficientAD.

4.2 Model-wise breakdown of results

PatchCore

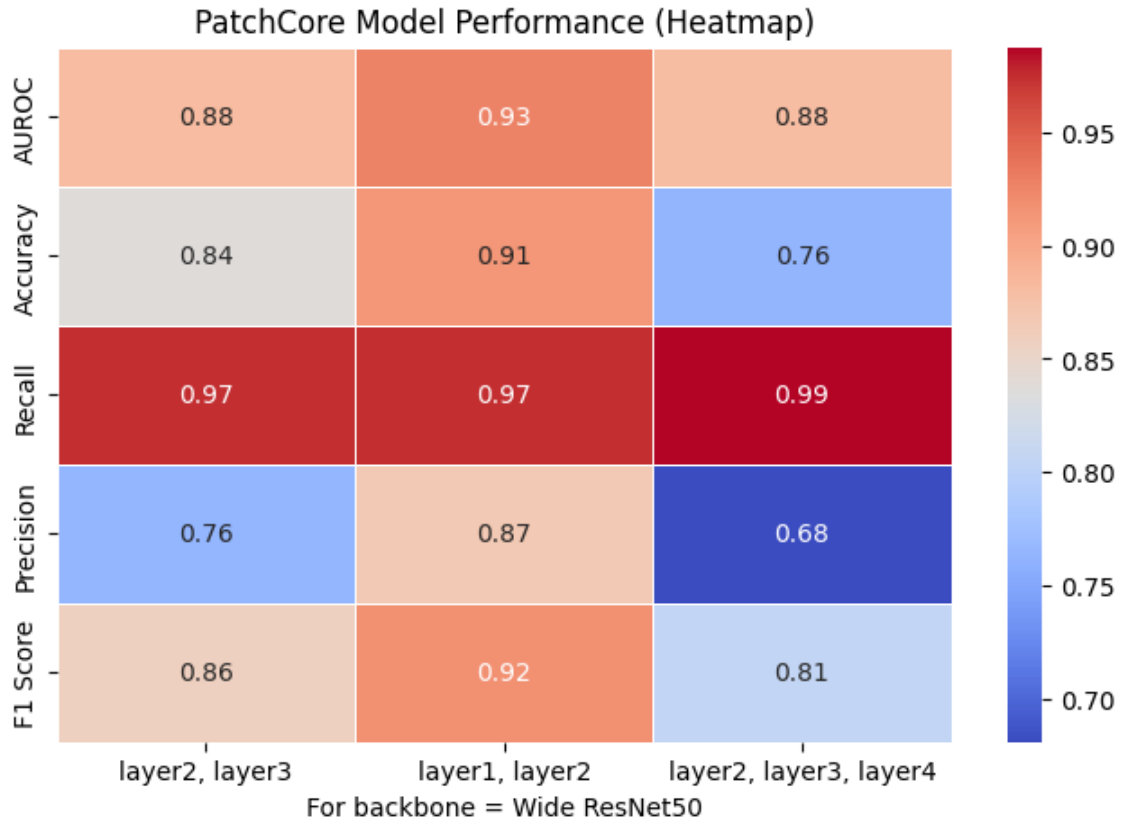

Figure 4.2.1: PatchCore heatmap for the experiment where backbone "Wide Resnet50" and different layers were used.

For this experiment, we employed PatchCore with `wide_resnet50_2` backbone as feature extraction for performing anomaly detection. Firstly, the model extracts feature representations from specific layers of the backbone and then compares the patch-level features of test images with those stored in a memory bank built from normal data. PatchCore uses K-NN retrieval for detecting deviations from nominal behavior by calculating the anomaly score based on the distance between patches from the test image and their closest counterparts in the memory bank. This allows PatchCore to perform well when the labeled data is not abundantly available. This model is explained in more detail in 2.3.1.

For the first configuration, we have used backbone `wide_resnet50_2` with layer2 and layer3 as shown in the heatmap 4.2.1. This configuration achieves an AUROC score of 0.8816 and an overall accuracy of 83.75%. The F1-score was relatively strong at 0.8571, suggesting that the model maintained a balance between precision and recall.

The model had a precision of 0.7647, suggesting that sometimes the model classified normal data as anomalous, which resulted in moderate occurrence of false positives, as can be confirmed by the confusion matrix 4.2.2. Whereas the high recall value of 0.975 indicates that the model was highly accurate in detecting almost all the anomalies as anomalies, with some false negatives, as can be seen in the figure 4.2.2.

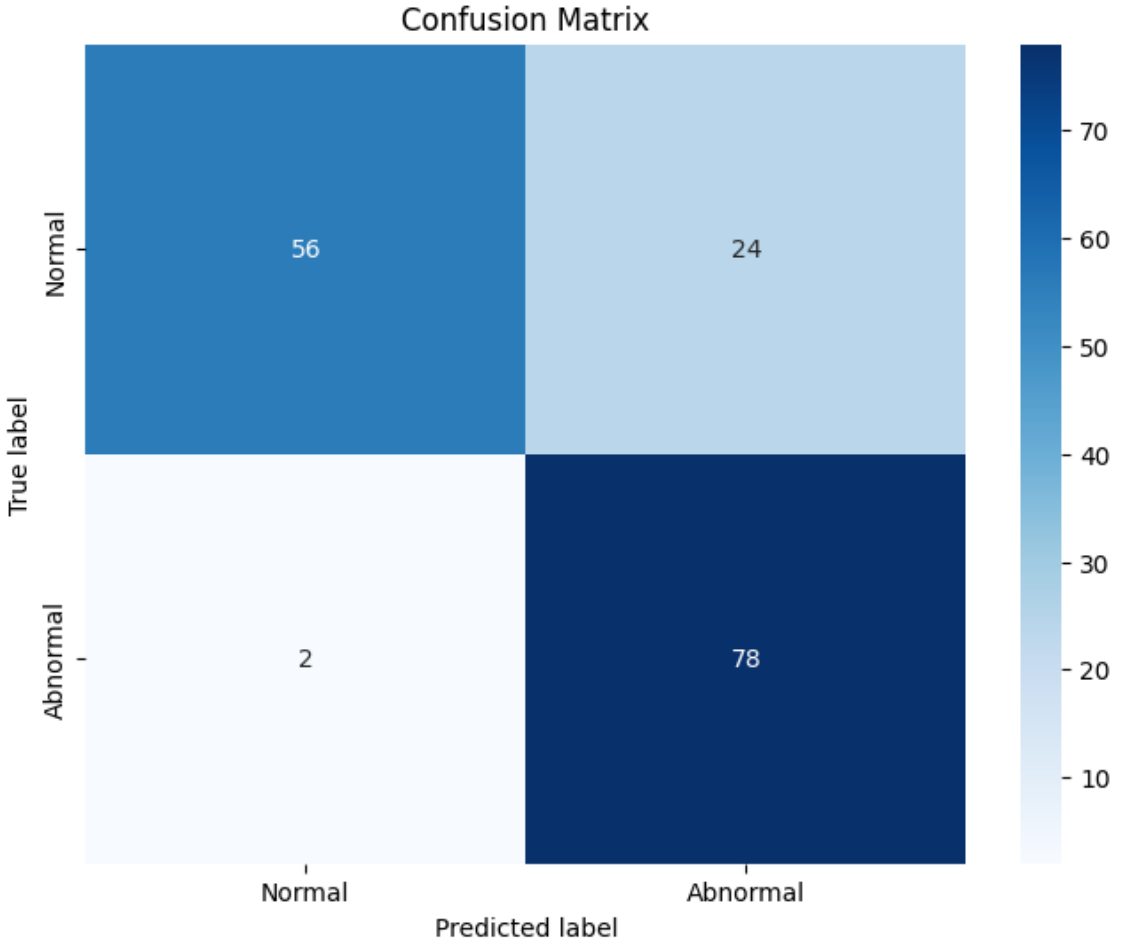

Figure 4.2.2: Confusion matrix for the first configuration where backbone "Wide Resnet50" and layers "2, 3" were used.

The second configuration resulted in the best overall performance across all metrics, where the feature extraction was carried out by layer1 and layer2 of the same backbone `wide_resnet50_2`. This configuration resulted in a high AUROC score of 0.9271, with a significant increase in accuracy of 91.25% as shown in the heatmap 4.2.1. We also saw improvement in F1-score by about 6.6% from the previous configuration, reaching 0.9176, indicating an even better balance between precision and recall. The precision improved to 0.8667, indicating a reduction in false positives as seen in the confusion matrix 4.2.3 where the false positives were reduced by 50% from the first configuration. While the recall remained high but the same as the first configuration at 0.975, meaning the model continued to detect almost all anomalies,

as shown in the figure 4.2.3. The improved performance of this configuration can be due to the use of features from the combination of layer 1 and layer 2, which incorporates the combination of both low-level and mid-level features. Low-level features might allow the model to detect fine-grained details, while layer 2 might provide the more complex structures needed to detect anomalies. These features can provide a richer, more detailed representation of normal data, making the detection of anomalies without overfitting normal data easier.

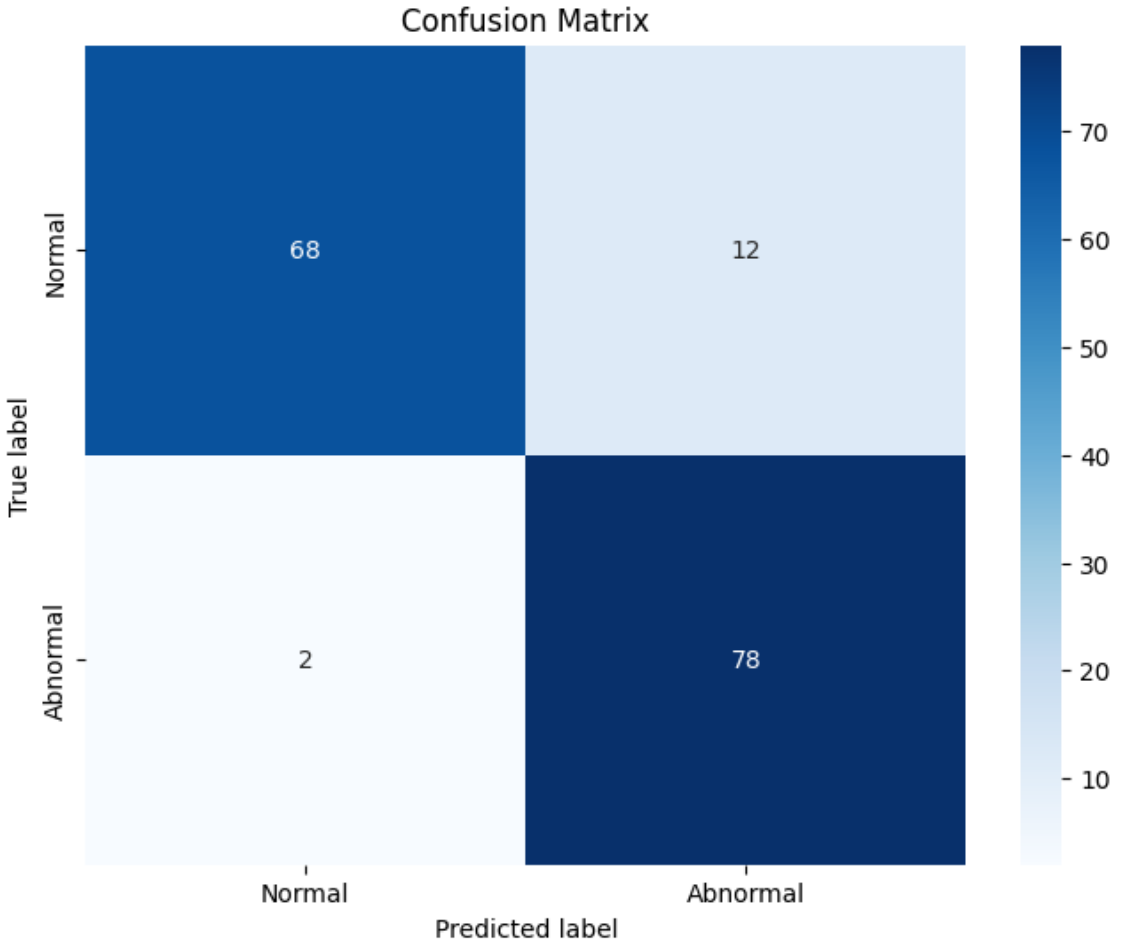

Figure 4.2.3: Confusion matrix for the second configuration where backbone "Wide Resnet50" and layers "1, 2" were used.

In the third configuration, layer 2, layer 3, and layer 4 were used for the backbone `wide_resnet50_2`, it resulted in a lower AUROCscore of 0.8807 which is almost equal to the AUROC score for first configuration, and the accuracy we got is 76.25% which is the lowest of all the configuration. The F1-score also decreased slightly to 0.8061 due to a drop in precision value to 0.6810. This lower precision indicates that the model produced a higher number of false positives, i.e., more normal images were classified as anomalous, as can be seen in confusion matrix 4.2.4 where out of 80 normal images, 37 were classified as anomalous. However, the recall was the highest at 0.9875,

which means that the model was still able to detect almost all of the anomalies, as seen in the figure 4.2.4 out of 80, 79 were correctly classified as anomalous with only one being misclassified as normal. The inclusion of deeper layers, like layer 4, probably introduced more abstract features, which may have been less effective for the detection of fine-grained anomalies, resulting in the reduction of overall performance. This configuration highlights the importance of extraction of features from appropriate layers for ensuring balance between detecting true anomalies and avoiding excessive false positives.

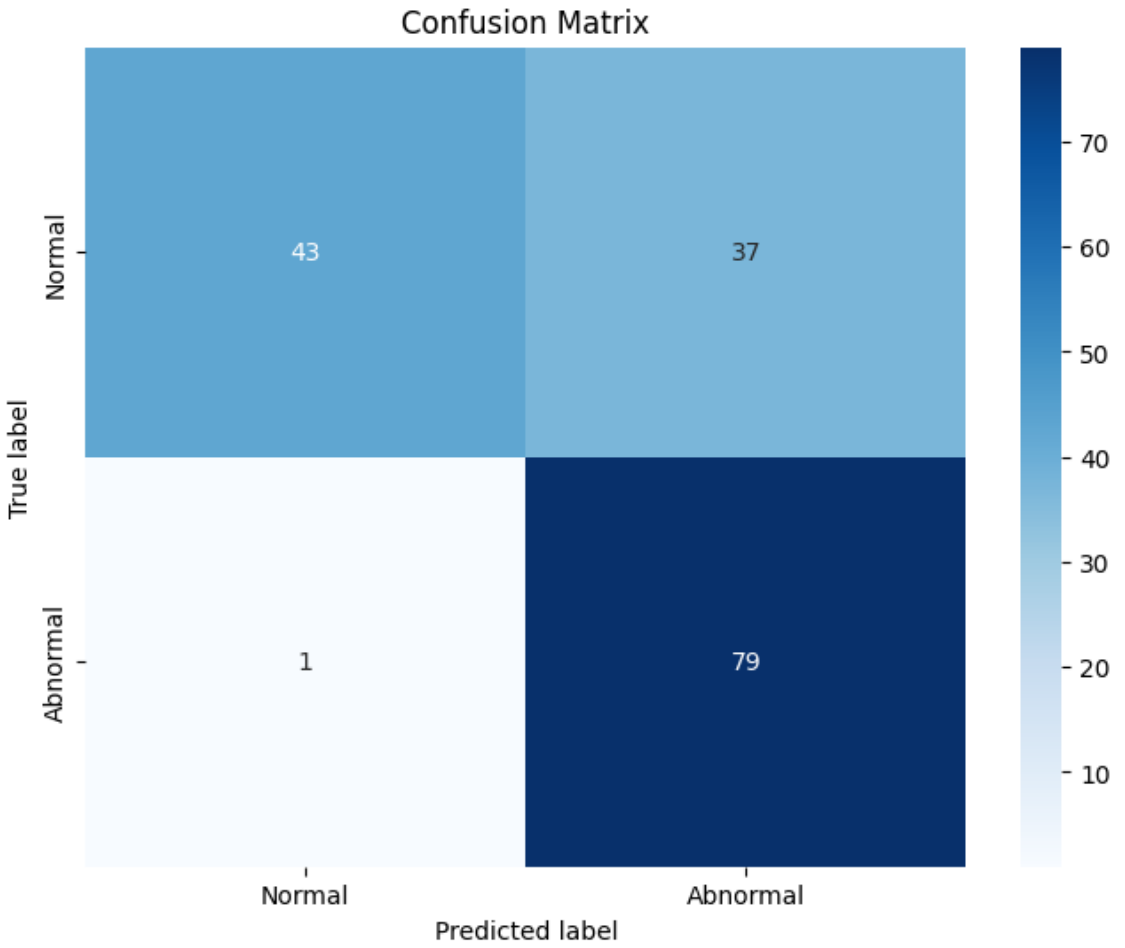

Figure 4.2.4: Confusion matrix for the third configuration where backbone "Wide Resnet50" and layers "2, 3, 4" were used.

Now let's look at how the best performing configuration of PatchCore image classification results. When performing image classification, three images are generated one is the original image which we provided to perform classification on, then the predicted heat map generated by the model is shown, and finally the predicted label for the image and its score is given, as shown in the figure 4.2.5. When the image is predicted as normal, then no heat map is generated.

Lastly, we also performed Image Segmentation using the best performing

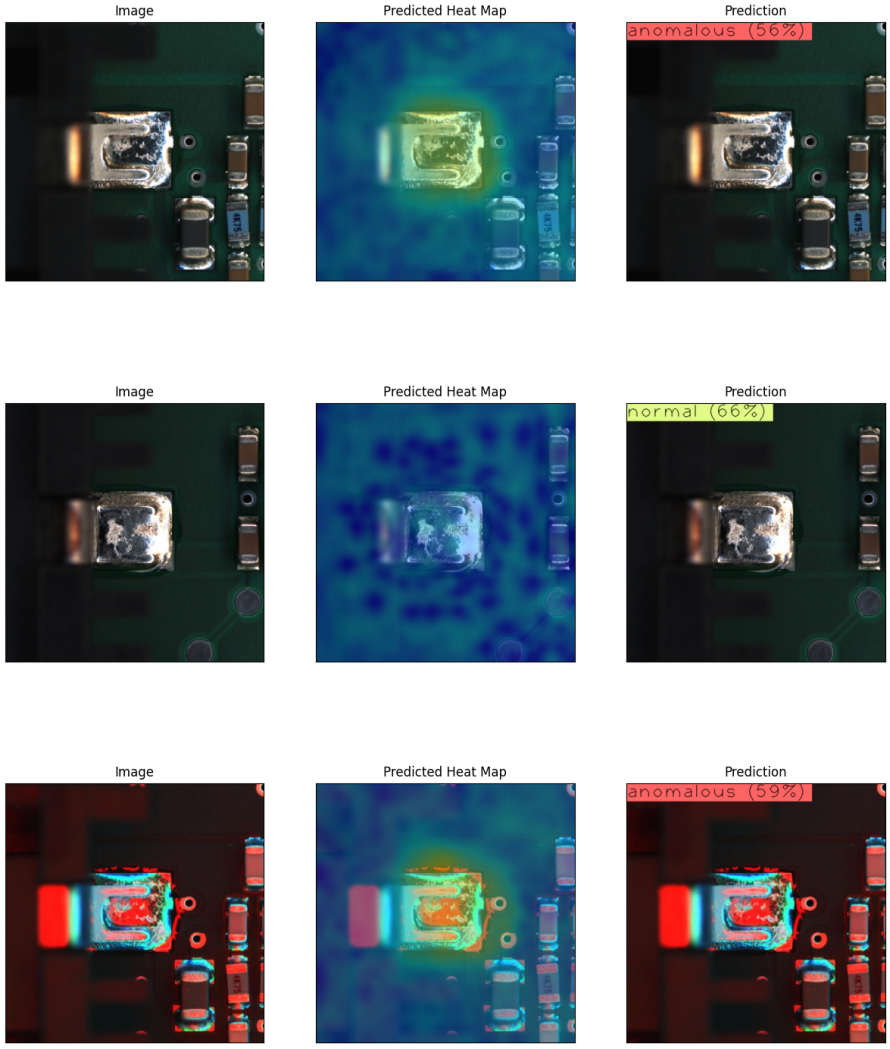

Figure 4.2.5: Results of image classification using second configuration(best performing) for PatchCore model.

configuration of PatchCore model, an example of its results can be seen in the figure 4.2.6. For the image segmentation, four images are generated in total. The first two are similar to the image classification results, but as the task here is segmentation, when the image is predicted as anomalous, it also generates a predicted mask of where the anomaly is, as shown in the last image of the top of the figure 4.2.6 marked with a red line. And when there is no anomaly detected it is just blank.

Both the image classification and segmentation results, helps us visualize how the model is performing in real world where we can see what is the predicted label and the anomaly score given for that image, with that also if the image is anomalous, with the help of the model we can pinpoint where the anomaly must be.

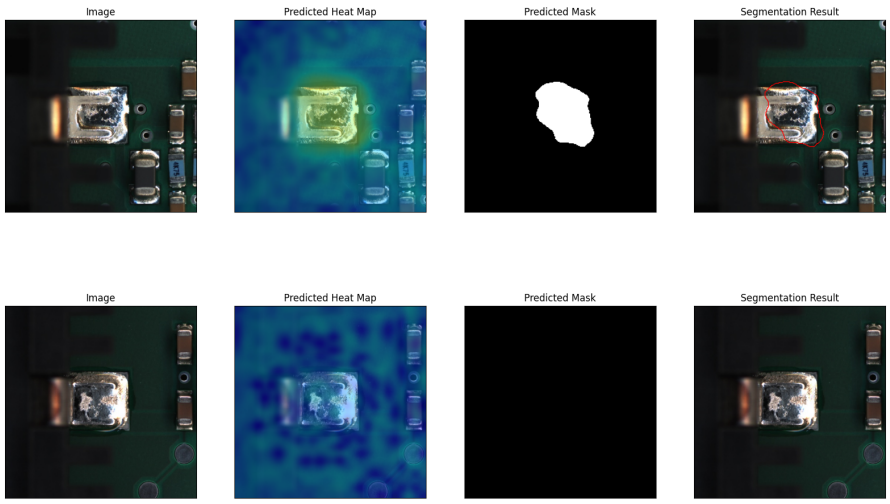

Figure 4.2.6: Results of image segmentation using second configuration(best performing) for PatchCore model.

YOLO

Two different model sizes of YOLOv8 [36] were evaluated for this experiment: YOLOv8-M(medium) and YOLOv8-L(large). Both the models were trained for 200 epochs on the training dataset.

The **YOLOv8-L** model, which is larger and more complex, reached an accuracy of 95% after 200 epochs. The model's high accuracy could be due to its increase in the depth and number of parameters, which allows it to capture more detailed and complex patterns in the dataset. Also it being a supervised model, having access to all the labels while training helps it to learn better and fit the model well to the data.

The confusion matrix, as shown in figure 4.2.7, provides a more detailed look into how well the model performs in the classification task. We can see that the model was able to correctly classify all the normal(FC) images as normal. While only 6 were false positives, i.e., anomalous(NG) images were classified as normal images, the rest of the 74 were correctly classified as NG, i.e., True Negatives. These values are consistent with the precision, recall, and F1-score results as shown in table 4.2. The precision of 1.0 highlights the model's reliability in predicting normal(FC) instances, as it did not make any incorrect predictions for that class. The recall of 0.925, while still quite good, indicates that the model misclassified a small number of anomalous(NG) as normal(FC). With the impressive F1-score of 0.9615, shows the models well-rounded performance.

The **YOLOv8-M** model, which is smaller and more lightweight when compared to YOLOv8-L, reaches an accuracy of 93.75%. This is slightly lower than its bigger model, as seen from the table 4.2, but its overall performance was still quite impressive.

The confusion matrix in figure 4.2.8 shows almost similar results to that of YOLOv8-L.

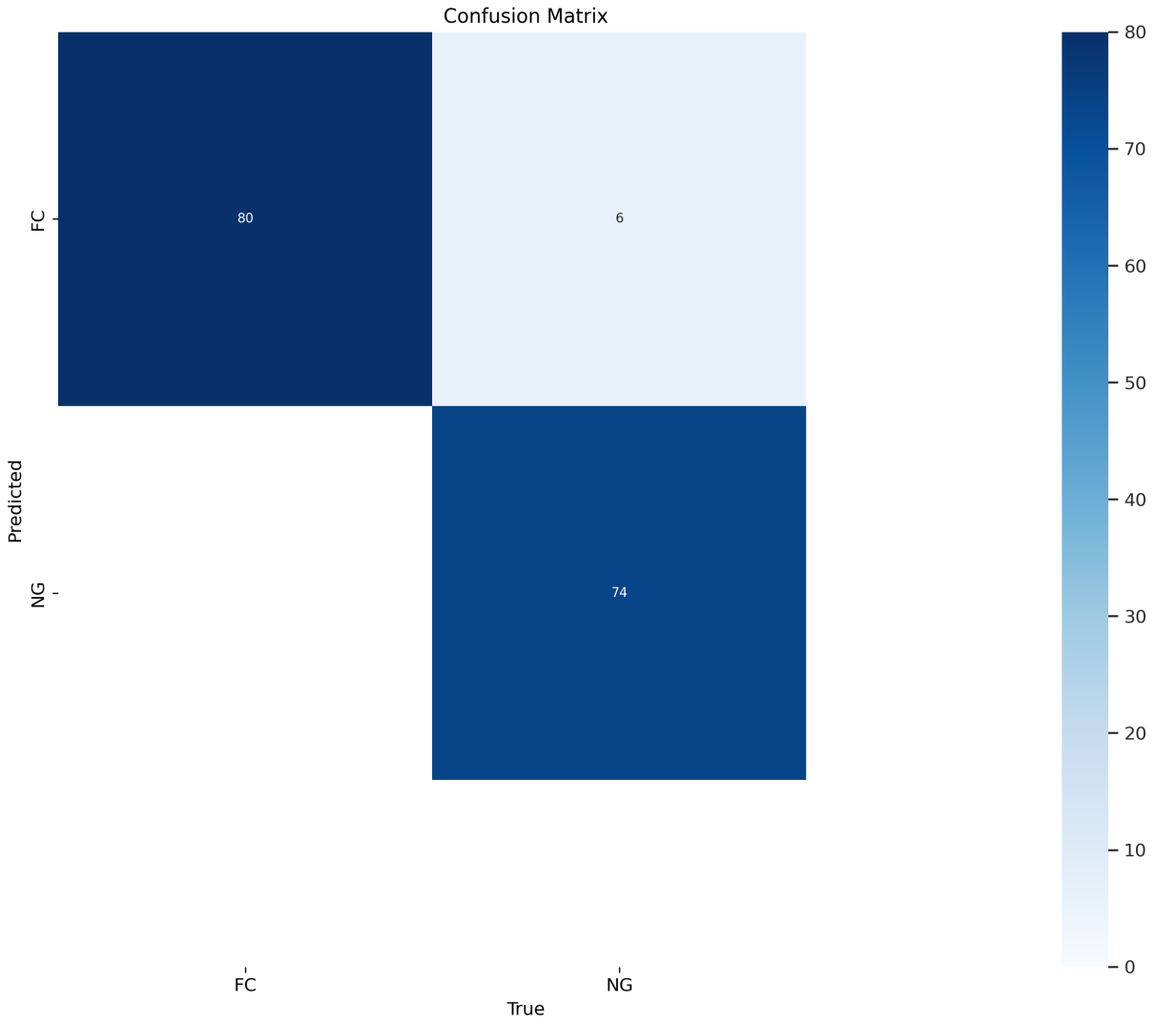

Figure 4.2.7: Confusion matrix for the baseline model YOLOv8-L, its accuracy is 95%

Model	Accuracy	Precision	Recall	F1-score
YOLOv8-M	93.75%	0.9867	0.925	0.9547
YOLOv8-L	95%	1.0	0.925	0.9615

Table 4.2: Comparison of YOLOv8-M and YOLOv8-L Model Performance

With 79 True positives and 74 true negatives, while only making 6 false positives and 1 false negative prediction, apart from the 1 misclassified normal image as anomalous, the results are similar to that of YOLOv8-L. This indicates that the performance of the YOLOv8-M, though highly accurate, still falls just short of YOLOv8-L's performance due to its smaller capacity for learning complex patterns. As can be seen from the table 4.2, the model achieved a precision of 0.9867, meaning that almost all of the predictions of normal(FC) images were correct. The recall for both the YOLOv8-L and

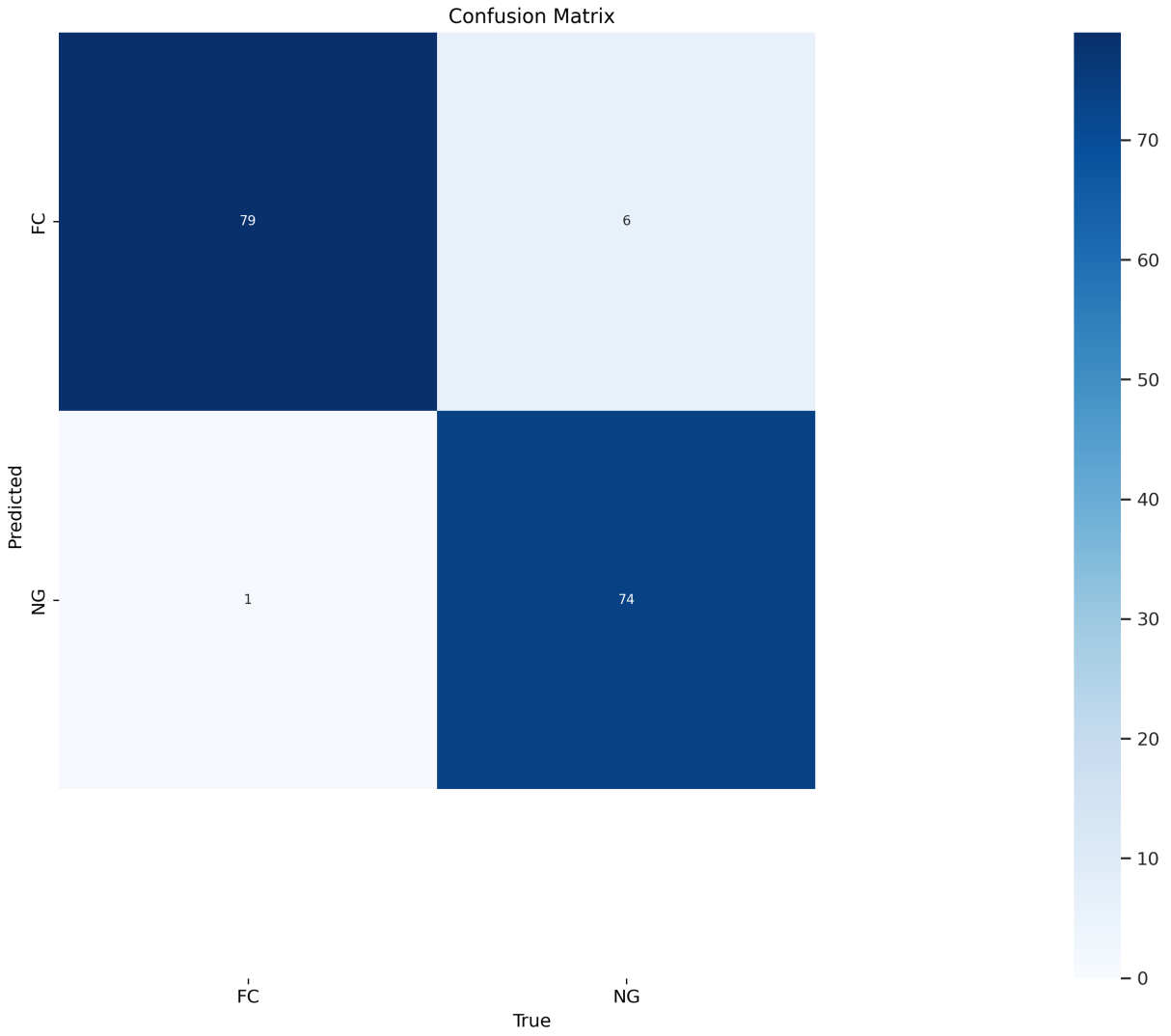

Figure 4.2.8: Confusion matrix for the baseline model YOLOv8-M, its accuracy is 93.75%

YOLOv8-M matches and is equal to 0.925, indicating that both models successfully detected the same number of FC cases. The F1-score of the model was 0.9547, which is slightly lower than YOLOv8-L due to the slight decrease in the precision. While the model was highly accurate, it did produce a small number of false positives, in this case, 6 anomalous(NG) images were misclassified as normal(FC), which reduced its precision slightly.

DFM

In this section, we will look into the results obtained from various experiments using DFM. DFM is explained in the section 2.3.2, and in our experiments, we evaluated DFM

performance using various backbones, their layers, and pooling techniques to explore the robustness and accuracy of the model. Below the results are divided based on the backbone used.

ResNet50

Here, we used ResNet50's layer4 for feature extraction along with Negative Log-Likelihood (NLL) as the scoring type. This gave us the AUROC score of 0.6309 and F1-score of 0.6689, with an overall accuracy of 50.7%, which is not very good, as can be seen from the table 4.3, it is basically doing random guessing. The low F1-score suggests that while the model was good at detecting anomalies, it struggled with precision because of the high number of false positives. These findings suggest that using features extracted from higher layers, like layer 4, may have led the model to focus on more abstract, high-level features that are less effective at determining anomalies from normal samples.

Model	**Accuracy (%)**	**AUROC**	**Precision**	**Recall**	**F1-score**
ResNet50	50.71%	0.6309	0.5036	1.0	0.6699
Wide-ResNet50-2 (layer2, nll)	60%	0.6382	0.5556	1.0	0.7143
Wide-ResNet50-2 (layer2, fre)	62.12%	0.5795	0.5681	1.0	0.7254
Wide-ResNet50-2 (layer3)	62.86%	0.7265	0.5738	1.0	0.7292

Table 4.3: Results for ResNet and Wide-ResNet50-2 Backbones

Another configuration that we tried was using a different variant of ResNet50 called Wide-ResNet50-2 and performing feature extraction using layer 2 while keeping the PCA level at 0.97 and using NLL score type. This resulted in a slight improvement in both the accuracy and AUROC score of 60% and 0.6382, respectively. The precision improved to 0.5556, while the recall stayed at 1.0, resulting in an F1-score of 0.7143, as shown in the table 4.3. Along with that, we also performed a similar experiment by keeping the backbone and layer the same but changing the PCA level to 0.995 and scoring type to Frequentist (FRE). This resulted in much better compared to when we used scoring type as NLL, where accuracy increased by about 2% to 62.12% while seeing a drop in AUROC to 0.5795. The recall stayed the same at 1.0, but a slight increase in precision was observed to 0.5681, therefore increasing the F1-score to 0.7254. This improvement in both precision and F1-score shows that lower-level features from layer 2 were more effective at capturing variations between normal and anomalous images. Finally, after seeing better results with FRE scoring type, we experimented with one more layer of the same backbone. The layer used here was layer 3, and with this, we again saw slight improvement across all the metrics. The accuracy came out to be 62.86% which is a small incremental update, while with AUROC score we saw an improvement of about 25% when compared to Wide-ResNet50-2 (layer2, FRE) of the table 4.3, and saw a rise of about 14% when compared with Wide-ResNet50-2 (layer2, NLL). Other layers were also tried for both

the ResNet50 and Wide-ResNet50-2 backbone, but all of them resulted in the model just randomly guessing the predictions, i.e., the accuracy was 50%.

Densenet

Next, we conducted experiments using different DenseNet backbones. Firstly, we used DenseNet121, which extracted features from the features.norm5 layer and the FRE score type. This configuration improved the model's performance further, with an AUROC score of 0.7309 and an accuracy of 61.43%. The model reached a precision of 0.5645 and a recall of 1.0, giving an F1-score of 0.7216. These improved scores suggest that the DenseNet architecture, which relies on dense connections and feature reuse, contributed to improved feature representations for anomaly detection. Table 4.4 shows the results of different backbones of DenseNet.

Model	AUROC	Accuracy	Precision	Recall	F1-score
DenseNet121	0.7310	0.6143	0.5645	1.0	0.7216
DenseNet169 (norm5)	0.7456	0.6571	0.5932	1.0	0.7447
DenseNet169 (denseblock3, 0.97 PCA)	**0.7203**	**0.7286**	0.6538	0.9714	0.7816
DenseNet169 (denseblock3, 0.995 PCA)	0.7190	0.7214	0.6422	1.0	0.7821
DenseNet201 (denseblock3, 0.995 PCA)	0.7259	0.7143	0.6389	0.9857	0.7753

Table 4.4: Results for DenseNet Backbones

The best performing configuration for the DenseNet architecture comes from the DenseNet169 backbone, with features extracted from features.denseblock3 layer, a PCA level of 0.97, and the FRE score type as shown in the table 4.4. This configuration gave an AUROC score of 0.7203 and an accuracy of 72.86%. Precision also improved to 0.6538. While recall reduced slightly, it remained high at 0.9714, resulting in an F1-score of 0.7816. This improvement in both the precision and the F1-score suggests that the DenseNet169's larger size allowed for better feature extraction and classification, particularly when extracting features from the third dense block. The high recall score indicates that the model was able to detect most of the anomalies, as can be seen in the confusion matrix 4.2.9, while the increase in precision value indicates a significant reduction in the number of false positives relative to other configurations.

After getting good results with the configuration explained above, we thought of checking another variation of the DenseNet169 backbone while keeping the feature extraction layer and scoring type the same as before but using a higher PCA level of 0.995, allowing for a greater variance in the data. However, the results were more or less similar to the previous configuration, with a slightly lower AUROC of 0.7190 and an accuracy of 72.14%, along with a precision of 0.6422 and an F1-score of 0.7821. The recall remained at 1.0, but the decrease in precision compared to the previous configuration indicates that the increased retained variance caused the model to classify a higher number of normal images as anomalous. Finally, DenseNet201 was also tested with the same feature extraction layer, keeping the other hyperparameters

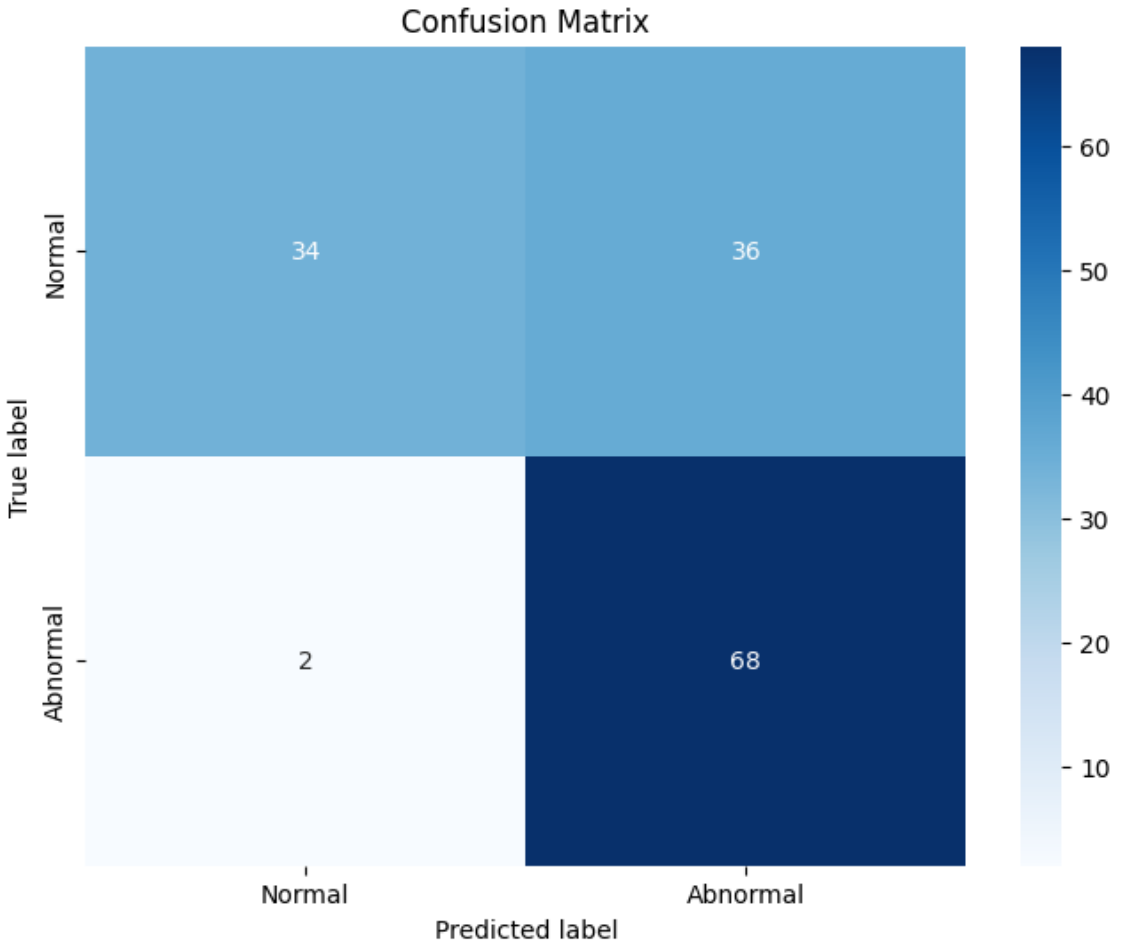

Figure 4.2.9: Confusion matrix for the best performing configuration for DFM model with DenseNet169 backbone, its accuracy is 72.86%

the same. This configuration also gave similar results to previous ones highlighted in the table 4.4.

DFKDE

Here, we discuss the results obtained for the model DFKDE. It combines the power of DL with statistical methods like PCA and KDE. The model first extracts robust features using a pre-trained deep neural network, then reduces their dimensionality by using PCA, and finally applying KDE to model the distribution of normal data. Two backbones were tested in this experiment, namely ResNet18 and Wide-ResNet50. The overview of the results for different configurations in the form of a bar chart is shown in figure 4.2.10.

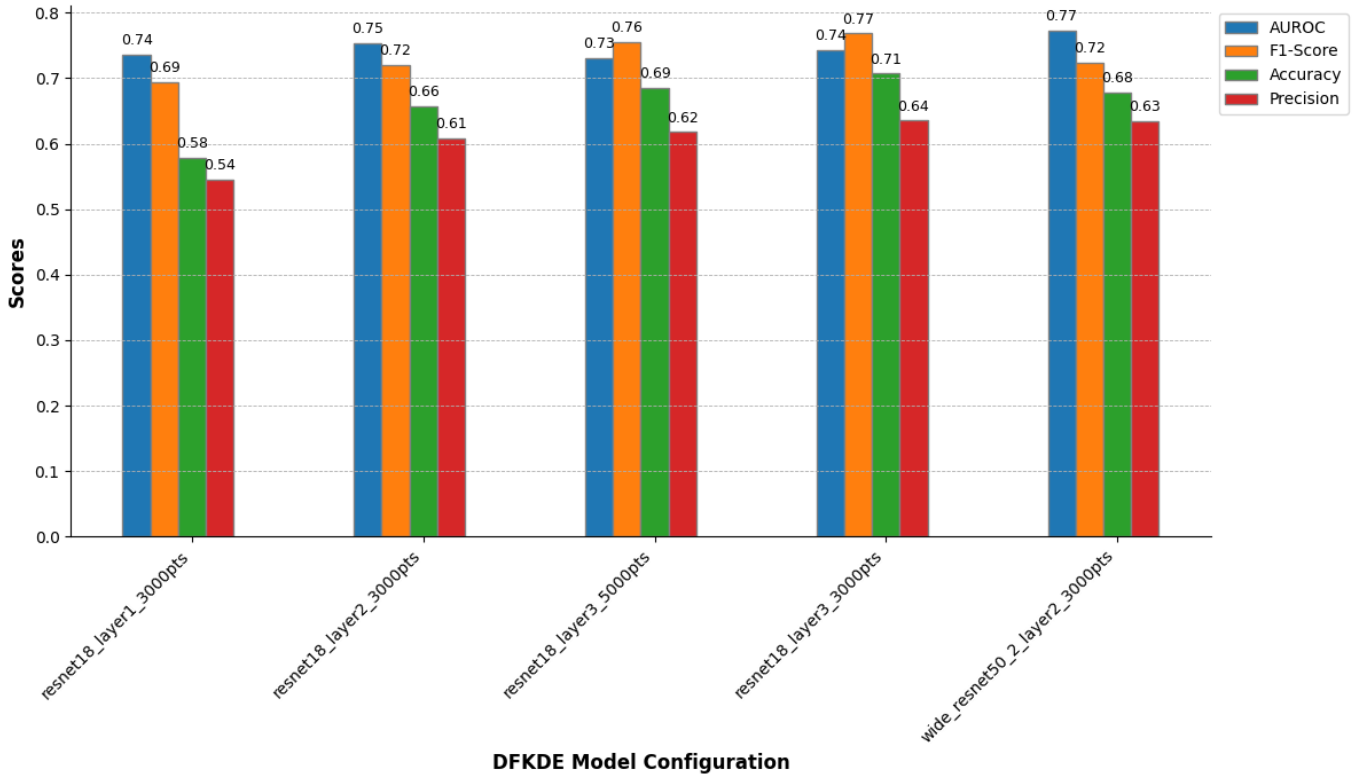

Figure 4.2.10: DFKDE model results for different backbones. The values shown here include AUROC score, F1-score, accuracy, and precision.

ResNet18

For the first experiment with the ResNet18 backbone, we use the feature extraction layer 1. This configuration aims at capturing the lower-level basic features and is trained with a maximum of 3000 training points to fit the KDE model.

Its results are shown in the table 4.5. As can be seen, the performance is not quite good, with AUROC score of 0.7364 and an accuracy of 57.86%. Precision is quite low at 0.5447, while recall is high at 0.9571, resulting in an F1-score of 0.6943. The lower precision value indicates that the features extracted from the lower layer lead to the introduction of noise, leading to higher false positives. While the high recall value suggests that the model was still able to detect most of the anomalies correctly. However, the overall performance of the model still suffered as low-level features are less effective in anomaly detection for DFKDE model.

For the second experiment with the ResNet18 backbone, we used feature extraction layer 2, and the number of maximum data points for KDE model remained the same as before at 3000.

The model achieved an AUROC score of 0.7535, which is slightly better than when features were extracted from layer 1, an accuracy of 65,71% is observed, which is about 14% higher than that of configuration 1, as can be seen in the table 4.5. A

Configuration	AUROC	Accuracy(%)	Precision	Recall	F1-Score
Layer1, max training points 3000	0.7364	57.86%	0.5447	0.9571	0.6943
Layer2, max training points 3000	0.7535	65.71%	0.6078	0.8857	0.7209
Layer3, max training points 3000	0.7428	70.71%	0.6355	0.9714	0.7684
Layer3, max training points 5000	0.7310	68.57%	0.6182	0.9714	0.7556

Table 4.5: Results for DFKDE Model with ResNet18 Backbone

precision of 0.6078 and a recall of 0.8857, resulting in an F1-score of 0.7209. This improvement in AUROC score and precision indicates that the mid-level features are more effective in anomaly detection for DFKDE model as they maintain the balance of the low-level details and high-level semantic information. Even though recall is slightly reduced compared to layer 1, the improvement in precision indicates a reduction in false positives.

For the third experiment, we select layer 3 for feature extraction of the backbone ResNet18, which is responsible for capturing higher-level semantic information. This layer offers a more abstract representation of the input data, which can be particularly suitable for anomaly detection tasks. This configuration was the best performing one in the DFKDE model.

As shown in the table 4.5 an AUROC score of 0.7427, an accuracy of 70.71% which is more than 22% higher than configuration 1, and more than 7% higher than configuration 2. A further improvement in precision was observed at 0.6355, and a recall of 0.9714, leading to an F1-score of 0.7684. The layer 3 features give more abstract representations of the data, which helped reduce false positive rates further while also maintaining high recall. However, a slight dip in AUROC compared to layer2 configuration can indicate that while the high-level features are useful, they might lack some of the finer details necessary for achieving good anomaly detection performance.

Next, for the fourth experiment, to examine the effects of increasing the number of training points, we increased the max training points parameter to 5000 points to fit the KDE model while keeping the backbone and the feature extraction layer the same as ResNet18 layer3. The goal was to see if more training data would improve KDE's ability to model normal distribution, as layer 3 gave the best results.

In table 4.5, we can see that the model reached an AUROC score of 0.7310, with an accuracy of 68.57%, which is lower than the third experiment, and with precision and recall of 0.6182 and 0.9714 respectively, leading to an F1-score of 0.7556. The performance was slightly worse than that of configuration 3, where layer 3 with 3000 max training points was used. This indicates that the number of training points may be less important than the layer from which the features are extracted, at least beyond a threshold.

Wide-ResNet50

The second backbone we experimented with was Wide-ResNet50-2, which is a wider version of the standard ResNet50 architecture. In this experiment, features were extracted using layer 2 of the backbone. This configuration was chosen because the middle layer performed better for the ResNet18 backbone experiments and with the wider architecture, which can capture more detailed feature representations, which could possibly improve the anomaly detection performance.

Configuration	AUROC	Accuracy(%)	Precision	Recall	F1-Score
Layer2, max training points 3000	0.7725	67.86%	0.6344	0.8429	0.7239

Table 4.6: Results for DFKDE Model with Wide-ResNet50-2 Backbone

This configuration showed improved performance in AUROC score and precision as seen from the table 4.6 of 0.7725 and 0.6344, respectively. The accuracy was 67.86%, and the recall fell to 0.8429, resulting in the F1-score of 0.7239. The wider architecture and mid-level features from layer 2 did help the model capture more subtle patterns in the data, which resulted in a higher AUROC score when compared to ResNet18 experiments. The precision also slightly improved, indicating that the model was comparatively more accurate in detecting true anomalies. However, the lowest recall value of all the experiments combined from resNet18 and Wide-ResNet50 indicates that the model became slightly less sensitive to anomalies, likely as a result of the increase in specificity brought by the wider backbone.

FastFlow

For this experiment, we used the FastFlow model with ResNet18, Wide-ResNet50-2, DeiT, CaiT as backbones for feature extraction. An overview of all the results is shown in the form of a bar chart in the figure 4.2.11. The best performing configuration in terms of accuracy was the one where DeiT architecture was used for feature extraction, achieving 66.87%. Below, we will discuss all the experiments and their results in detail, grouped based on the backbones used. Here, all the models were trained for 50 epochs except for when callback functionality was used. Then, it is trained until a condition is met.

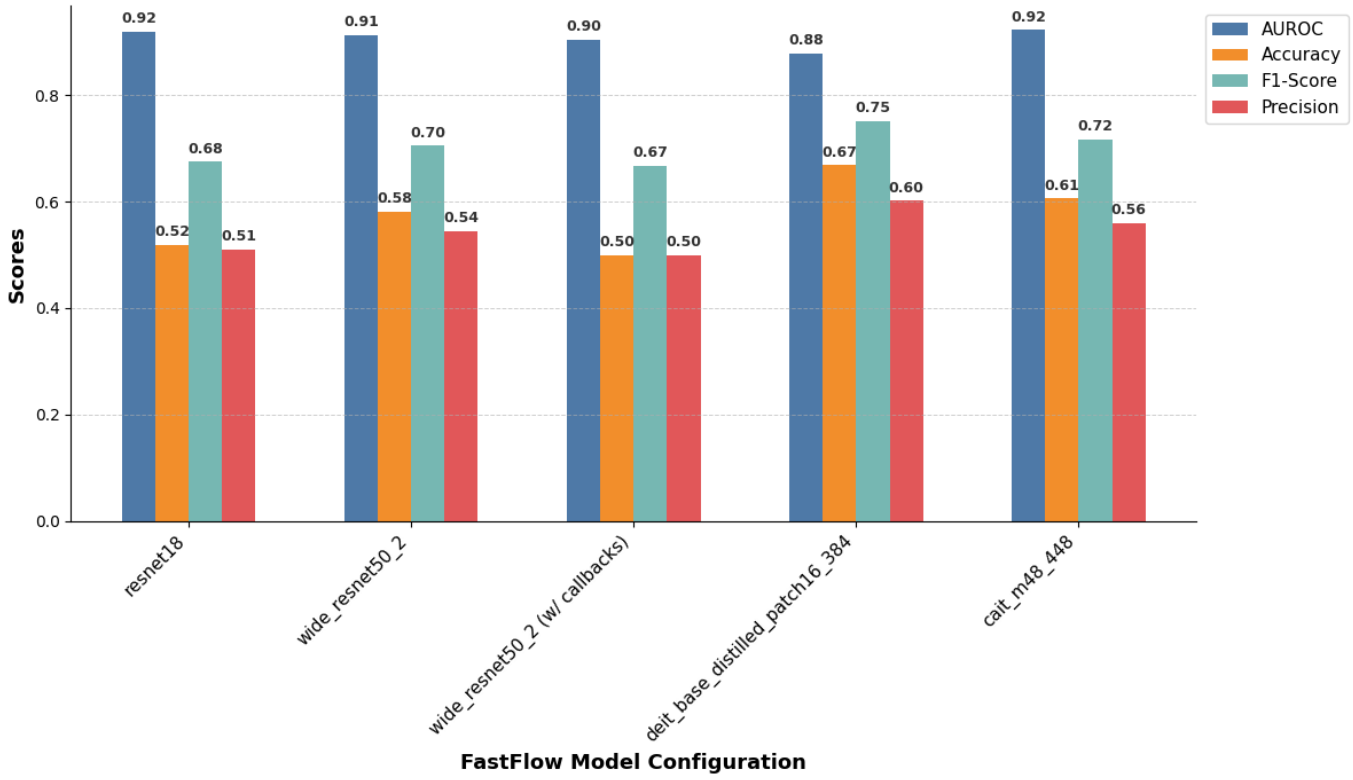

Figure 4.2.11: FastFlow model's results bar-chart for different configurations and backbones used for feature extraction.

ResNet18

For the first experiment, we have used ResNet18 as the backbone for feature extraction. This configuration achieved an AUROC score of 0.9199, which is quite good, and it demonstrates the models effectiveness in distinguishing normal images from abnormal ones on training and testing dataset. But, the overall accuracy on the unseen custom dataset is as good as random guessing, equating to 51.88%, reflecting imbalanced classification performance, as shown in the table 4.7. The recall of 1.0 shows that the model was able to detect all anomalous instances. However, the precision was low, at 0.5096, giving an F1 score of 0.6751.

Backbone	Accuracy (%)	AUROC	Precision	Recall	F1-Score
ResNet18	51.88%	0.9199	0.5096	1.0	0.6751

Table 4.7: Results for FastFlow model with ResNet18 Backbone

The confusion matrix results show that out of 80 normal samples, only 3 were correctly classified, while 77 were misclassified as abnormal. This significant misclassification of normal samples can be due to simplicity of the ResNet18 architecture, which might lack the depth required to capture subtle differences between normal and anomalous features correctly. Due to this, the model's ability to generalize across the dataset

declined, showing its bias toward identifying anomalies but at the cost of higher false positives.

Wide-ResNet50

For the second experiment, we used Wide-ResNet50-2 backbone for feature extraction. Along with that, we employed a callback mechanism. Here, we used early stopping, which monitors the image_AUROC while training, and when it stays the same for five consecutive epochs, then the training stops and those weights are saved. As can be seen in the table 4.8, the AUROC score dropped slightly to 0.9041 but again achieved 1.0 recall, indicating it was able to classify all the anomalous images correctly. However, the accuracy remained at 50%, which means the model effectively did not learn anything and is just making random guesses. The precision was 0.5, resulting in an F1-score of 0.666.

The confusion matrix results show that the performance was terrible, as the model classified all the images, whether normal or abnormal, as abnormal.

Backbone	**Accuracy (%)**	**AUROC**	**Precision**	**Recall**	**F1-Score**
Wide-ResNet50-2 (with callbacks)	50.00%	0.9041	0.5000	1.0	0.6667
Wide-ResNet50-2	58.13%	0.9125	0.5442	1.0	0.7048

Table 4.8: Results for FastFlow with Wide-ResNet50-2 Backbone

Looking at the above results, we decided to see the effect on the FastFlow model by keeping all the hyperparameters the same but removing the callback mechanism and letting the training continue for 50 epochs. The AUROC score increased to 0.9125, suggesting that the model's longer training led to better feature extraction. Along with that, the accuracy improved to 58.13%, with precision reaching 0.5442 and recall of 1.0, resulting in an F1 score of 0.7048.

The results of the confusion matrix showed improvement as well. The number of correctly classified normal samples increased to 13. However, 67 normal samples were still misclassified as abnormal. This suggests that longer training helps the model differentiate between normal and anomalous samples better. This performance improvement can also be due to the backbone's ability to capture more better features over a longer training period, refining the normalizing flow process and providing a more detailed understanding of normality in the dataset. Further, the increase in epochs did not show much improvement from then on.

DeiT_base

For the fourth experiment, the "DeiT Base Distilled Patch16_384" backbone was used for feature extraction, and the model was trained for 50 epochs. This configuration resulted in an AUROC of 0.8787, which is slightly lower than the previous experiments. However, despite that, the model achieved the highest accuracy for the FastFlow

model, at 66.88%. With a precision and recall of 0.6015 and 1.0, respectively, resulting in an F1 score of 0.7512 which is also the highest in all the experiments, as shown in the table 4.9.

Backbone	Accuracy (%)	AUROC	Precision	Recall	F1-Score
DeiT Base Distilled Patch16 384	66.88%	0.8787	0.6015	1.0	0.7512

Table 4.9: Results for FastFlow with DeiT_Base Backbone

The confusion matrix results show that 27 normal images were correctly classified, with 53 being misclassified as abnormal, reflecting its improved performance in detecting normal images compared to earlier configurations. The transformer-based DeiT backbone most likely contributed to these results, as its global attention mechanism enables the model to capture more contextual information across images. However, the slight drop in AUROC suggests that, while the model is capable of extracting features from normal samples, it might struggle more with identifying anomalies, which can be more subtle and context-dependent.

CaiT

For the fifth and final experiment, the "CaiT-M48" backbone was used and trained for 50 epochs. As can be seen from the table 4.10, this model achieved the highest image AUROC across all configurations, at 0.9223. The accuracy was 60.63%, which is about 9% lower than the configuration with DeiT as the backbone, with a precision of 0.5594 and an F1 score of 0.7175.

Backbone	Accuracy (%)	AUROC	Precision	Recall	F1-Score
CaiT M48 448	60.63%	0.9223	0.5594	1.0	0.7175

Table 4.10: Results for FastFlow with CaiT_M48_448 Backbone

The confusion matrix results showed that 17 normal images were correctly classified, while 63 were misclassified as abnormal. Despite no improvement in accuracy over the DeiT configuration results, the CaiT backbone performed better at maintaining a balance between precision and recall.

EfficientAD :

EfficientAD is a novel approach to visual anomaly detection, specifically developed to operate at millisecond-level latencies. This model was introduced as a lightweight alternative to existing anomaly detection architectures. EfficientAD is built on a S-T framework, where the teacher model learns representations from a more complex and larger model while the student attempts to mimic the teacher's performance using fewer resources.

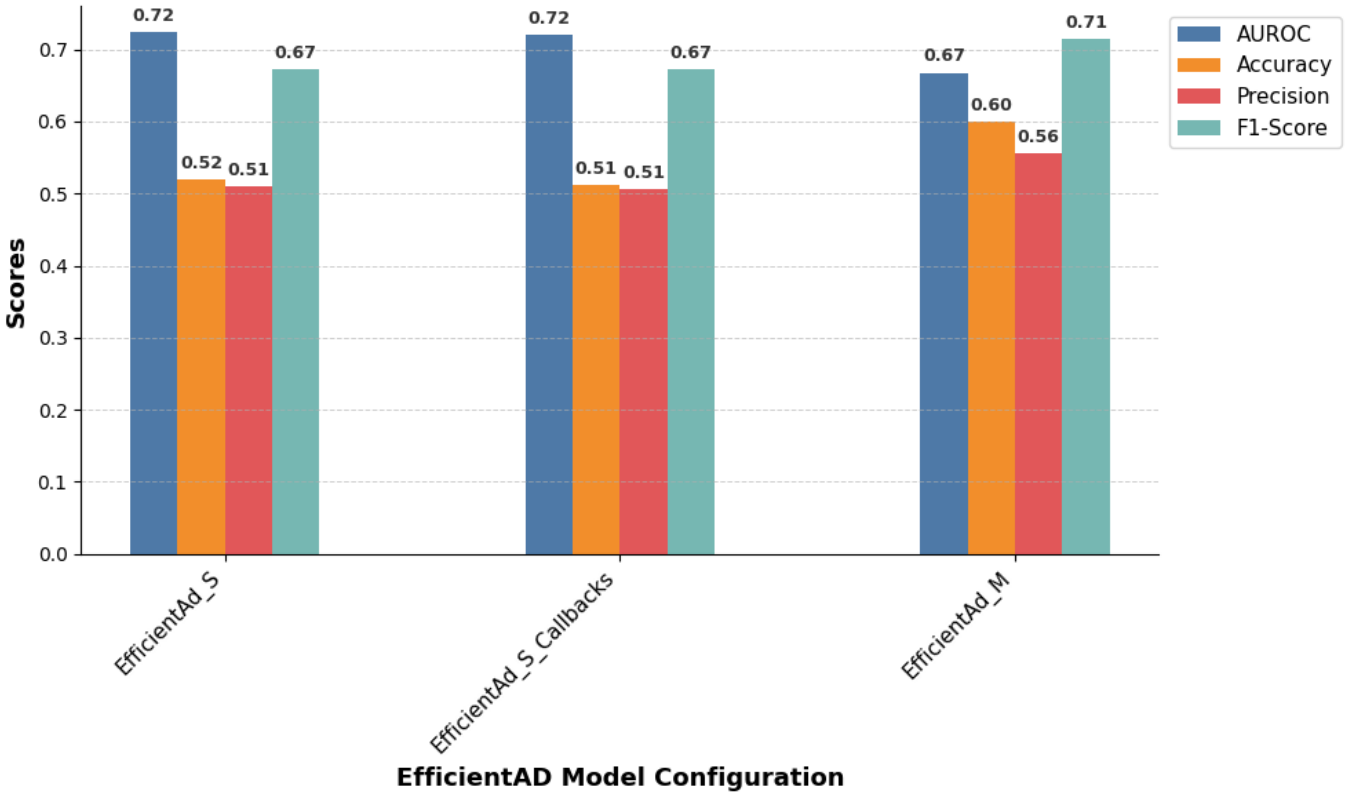

Figure 4.2.12: EfficientAD models result in bar-chart for different configurations used.

While the EfficientAD model claims to outperform existing models, such as PatchCore[2], as given in the paper [42], the experimental results conducted in this study suggest otherwise. The performance of EfficientAD was found to be suboptimal when evaluated on unseen data, as can be seen in the diagram 4.2.12. This section will discuss the results of different configurations in detail.

These experiments were carried out using three different configurations of the EfficientAD model. There are two sizes available for the model, which are EfficientAD_S(small) and EfficientAD_M(medium). The model size was constant for the first two configurations, but a callback mechanism was added in the second configuration. The Performance metrics used are the same as the rest of the models, namely AUROC, accuracy, precision, recall, and F1-score. Also, confusion matrices were generated to provide visual insights into how well the model performed in terms of classification.

In the first configuration, the base model (EfficientAdModelSize.S) was used without callbacks, as shown in the table 4.11. This gave the AUROC of 0.7244. However, the model's accuracy was only 51.87%, which clearly highlights its struggles in correctly classifying the majority of images. More specifically, out of 80 normal images, only 4 were correctly classified, with the remaining 76 being classified as abnormal, i.e., false positives. But on the other hand, the model performed quite well in detecting abnormal instances, with 79 out of 80 correctly classified as abnormal, resulting in

a recall of 0.9875. But the precision is quite low at 0.5097, showing the imbalance between the true positives and false positives. The resulting F1-score was 0.6723, indicating a tendency towards recall, with the model detecting most anomalies but at the cost of misclassifying a large number of normal images as abnormal. Here, all the models were trained for 50 epochs except for when callbacks functionality is used. Then, it is trained until a condition is met.

Model Size	**AUROC**	**Accuracy(%)**	**Precision**	**Recall**	**F1-Score**
EfficientAd-S	0.7244	51.88%	0.5097	0.9875	0.6723
EfficientAd-S (with callbacks)	0.7203	51.25%	0.5063	1.0	0.6723
EfficientAd-M	0.6678	60%	0.5556	1.0	0.7143

Table 4.11: Performance of EfficientAD Models with Different Sizes

For the second configuration, the same model size (EfficientAdModelSize.S) was used, but ModelCheckpoint and EarlyStopping callbacks were added. The purpose of these callbacks was to improve the training process by ensuring that the model did not overfit and preserving the best weights during the training. Despite these improvements, the model's AUROC dropped slightly to 0.7203, and an overall accuracy of which relatively remained the same as before at 51.25%, as seen in the table 4.11. This result suggests that the callbacks had minimal to no impact on the model's ability to generalize on unseen data. In terms of precision and recall, the scores were 0.5063 and 1.0, respectively, this shows us that while the model was able to classify all the abnormal images correctly, but it struggled a lot at classifying the normal images correctly and ended up with lots of false positives. The confusion matrix values further confirm these findings, with only 2 normal samples being correctly classified, with the rest, 78 being misclassified as abnormal. The F1-score came out to be 0.6723, which is identical to that of the first configuration with no callbacks.

In the third and final configuration, the larger variant of the model (EfficientAdModelSize.M) was used. An AUROC score of 0.6678 was observed, which is less than the smaller model configurations scores, indicating a further decline in the model's ability to differentiate between normal and abnormal instances. Even then, an increase in overall accuracy was observed at 60% with 16 normal samples correctly classified as normal, with 64 still being misclassified as shown in the confusion matrix 4.2.13. Although this is an improvement compared to other configurations, the model still struggles with a high number of false positives. The recall of 1.0 and increased precision of 0.5556 indicate an improved ability to correctly classify more normal images. The resulting F1-score is 0.7143, which is an improvement over smaller models, but it is still far from ideal.

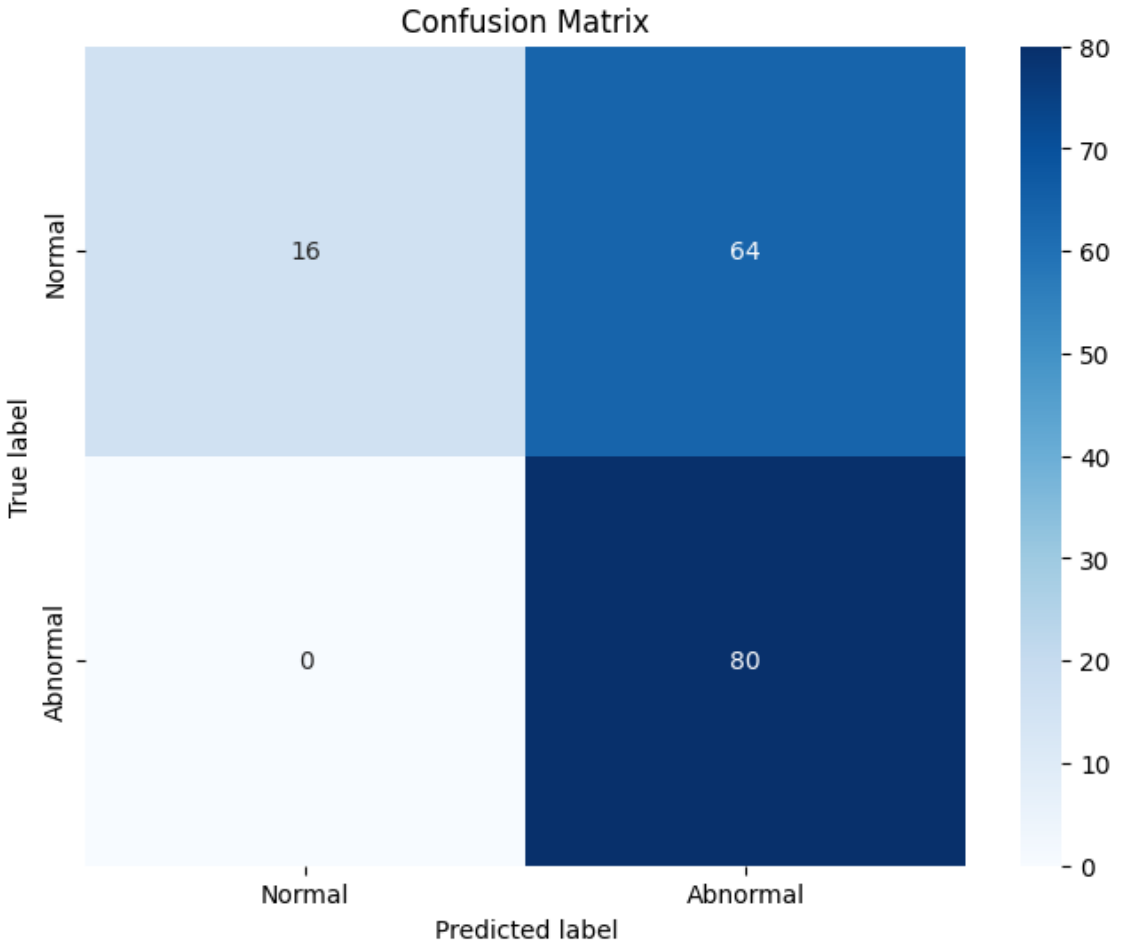

Figure 4.2.13: Confusion matrix for the best performing configuration for EfficientAD model with size EfficientAdModelSize.M, its accuracy is 60%

Chapter 5

Discussion

In this section, we will provide a more deeper understanding of the results. Particularly, we focus on comparing the performance of supervised and unsupervised approaches, also PatchCore's performance relative to the baseline YOLO model, and to try and examine the reasons behind the underperformance of newer models such as EfficientAD when compared to PatchCore. Also, we will investigate the consistently high recall value observed in most anomaly detection models in our experiments.

Supervised vs. Unsupervised Learning

One of the key objectives of this thesis was to evaluate the performance differences between the supervised and unsupervised models for anomaly detection. Supervised learning methods, in our case YOLO, usually require a large amount of labeled training data. This can be challenging for the PCB manufacturing, where defects can be rare, and labeling requires domain experts, which can be time-consuming and expensive. Whereas unsupervised models such as PatchCore, DFM, and EfficientAD use patterns from normal data to identify anomalies, which reduces the dependency on labeled datasets. **In anomaly detection, unsupervised learning means that the model is trained only on normal data, allowing it to learn the features of normal data and flag any deviations from it as anomalies. Even though the model is only exposed to normal samples during training, it does not have access to labeled information about defects or a mask indicating their location, which is required during supervised training methods. This makes it unsupervised as the model is not explicitly taught where defects are but instead learns to detect anomalies from normal patterns on its own.** The results also demonstrate that despite having no information about the defect location, it can still perform quite well, with PatchCore reaching an accuracy of 91.25%, which is quite good when compared to the baseline YOLOv8-m with only 2.5% difference in their accuracy. These results show the effectiveness of unsupervised learning models, which can also come close to the performance of supervised models.

Performance of PatchCore vs Baseline YOLO

The PatchCore model achieved an accuracy of 91.25%, showing its effectiveness in anomaly detection. While YOLOv8-M and YOLOv8-L achieved higher overall accuracy scores of 93.75% and 95%, respectively, YOLO models showed better results in most aspects, particularly in achieving fewer false positives, as evident from their confusion matrices. **PatchCore, however, provided a competitive approach**, particularly in its unique patch-level analysis. Unlike YOLO, which processes the entire image, PatchCore analyzes smaller patches of an image, which can be advantageous in certain scenarios. Despite this, the YOLO models generally outperformed PatchCore.

PatchCore's results are significant in addressing the 'cold start' problem, which refers to the lack of anomalous samples during training. By leveraging only normal data, **PatchCore provides a competitive alternative approach for industrial defect detection, successfully detecting anomalies without needing extensive labeled defect data like in supervised learning approaches**. This characteristic makes PatchCore suitable for quality assurance in high-volume production environments, where the ability to detect rare faults with minimal labeled data is crucial. Additionally, since no extensive manual labeling is required, this approach can help in significantly reducing costs.

Limitations of Newer Models

Although models like EfficientAD and FastFlow are more recent, with specific strengths, such as computational efficiency and speed, they still could not surpass PatchCore's performance in terms of accuracy. There could be several reasons for this outcome. Firstly, PatchCore's dependency on memory banks of patch-level features allows for a rich representation of normal patterns, which facilitates robust anomaly detection. The coreset subsampling method used in PatchCore reduces redundancy in the feature space, which helps improving it's efficiency when comparing new data with the stored nominal patches.

In contrast, even though EfficientAD is computationally efficient and fast due to its S-T network structure, it lacks the detailed patch-level comparison that PatchCore employs. EfficientAD focuses on lightweight architectures that excel in computational efficiency, but this can often come at the cost of the deep representational power needed to identify subtle anomalies in complex datasets. Similarly, DFM relies on probabilistic modeling of deep features. However, it lacks the fine granularity offered by PatchCore's patch-wise anomaly scoring, making it less effective when dealing with highly localized defects. FastFlow, which uses normalizing flows, also struggled to outperform PatchCore due to its limited ability to capture complex defect patterns. FastFlow's dependence on a more generalized feature extraction approach makes it less capable of identifying complex or subtle anomalies compared to PatchCore's specialized patch-level method.

In order to further highlight the differences in model performance, we compared key metrics such as accuracy, recall, and precision across all the unsupervised models. PatchCore achieved an accuracy of 91.25%, indicating its strong ability to detect defects. YOLOv8-M and YOLOv8-L, while fast and achieving accuracies, showed better results in capturing anomalies with fewer false positives. EfficientAD, with its focus on computational efficiency, demonstrated quick inference times but could not match PatchCore's accuracy, primarily due to its lightweight architecture. DFM, which uses probabilistic deep feature modeling, also fell short in accuracy compared to PatchCore, struggling with the complex and diverse nature of the defect patterns. This comparative analysis highlights the trade-offs between computational efficiency and detection accuracy among the different models.

High Recall in Anomaly Detection Models results

A notable observation across the evaluated models was the consistent high recall, often reaching 1.0. This behavior can be explained by the nature of anomaly detection tasks, in which the models are generally trained to identify any deviations from the learned distribution of normal data. In many industrial applications, it is crucial to capture all possible defects, even at the risk of producing false positives, to ensure product reliability. This drives the models towards maximizing recall, ensuring that no defective sample is overlooked. High recall is particularly important in situations where the cost of missing a defect is far greater than the cost of examining a false positive.

However, this emphasis on recall may come with the drawback of lower precision, as observed in some models. Precision measures the ability to correctly classify anomalies without misclassifying normal samples, and a lower precision value indicates a higher rate of false positives. In practical terms, while a high recall ensures comprehensive defect coverage, the accompanying lower precision necessitates additional post inspection checks to filter out the false positives.

Final Thoughts

While none of the unsupervised learning models could outperform the baseline YOLO model, but PatchCore still provided competitive results. Its ability to correctly detect anomalies without the need for labeled data or a mask during training makes it a valuable alternative, especially in settings where labeling is costly or impractical. This not only helps reduce the dependency on extensive manual labeling but also significantly cuts down the associated costs, making PatchCore a practical choice for industrial applications.

Chapter 6

Future Works

This thesis has explored different aspects of anomaly detection, focusing on supervised and unsupervised models such as YOLOv8, PatchCore, EfficientAD, and more. However, there are several directions for extending and improving the work, as outlined below:

Exploring Newer Unsupervised Models: Future work should explore newer and more advanced unsupervised models, particularly those utilizing vision transformer architectures, such as DINOv2[58], in the paper it has shown promising results even with just the pre-trained model. Vision transformers have shown promising results in various computer vision tasks, and their ability to capture global context could enhance anomaly detection.

Evaluating Performance on Larger Datasets and Faster GPUs: One of the limitations of models like PatchCore is the requirement for substantial memory to store patch-level features, which can lead to longer training and inference times. This is also true for other models we tested along with different backbones, which sometimes take days to train. Therefore, future research should evaluate these models on larger datasets, using faster GPUs with more memory to better understand their scalability and performance in high-volume industrial applications. Evaluating their performance on larger and more diverse datasets would also help to determine their robustness and adaptability to various defect types.

Deployment in Real-World Scenarios: Finally, deploying these models in real-world scenarios is an important future direction. Testing the models in an actual production environment would provide insights into their real-time performance, robustness, and ability to adapt to changing conditions. Challenges such as integration with existing quality assurance pipelines, real-time inference, and handling of various product types and defect characteristics should be explored.

Chapter 7

Conclusion

The goal of this thesis was to compare different unsupervised learning models for image classification of PCB images, with the objective to outperform the baseline YOLO model, which is currently being used by Siemens. YOLO, which is a supervised learning model, requires labeled datasets, which can be quite time consuming and expensive because of the requirement of subject experts for the annotation, it can be particularly challenging when the defects are quite rare. To address these limitations, we explored unsupervised learning models for anomaly detection. We experimented with several unsupervised models, including PatchCore, DFM, DFKDE, EfficientAD, and FastFlow. We also evaluated their performance on different metrics like accuracy and F1-score, to name a few. Among all the models, PatchCore gave the most competitive results, providing a great alternative to the baseline model.

While none of the unsupervised models could outperform the baseline YOLO model, this study remains significant in highlighting the potential of unsupervised anomaly detection. In particular, PatchCore provided competitive performance without the need of labeled data. EfficientAD, while being more computationally efficient, struggled to match PatchCore's accuracy because of its lightweight architecture. This makes it more suitable for scenarios where speed is prioritized over accuracy. DFM and DFKDE, while showing moderate performance, were less effective in capturing complex defect patterns compared to PatchCore. These insights highlight that while YOLO remains the most accurate, unsupervised models like PatchCore offer a valuable trade-off between performance and the need for labeled data, particularly in environments where labeling is infeasible.

Looking ahead, the findings of this thesis can be highly useful in real-world applications, particularly in quality assurance processes where anomaly detection is critical. This work lays the foundation for moving towards an unsupervised version of image classification for PCBs in industrial settings. This study shows that unsupervised models like PatchCore can serve as a good approach for anomaly detection without relying on labeled datasets. By evaluation of these models, we demonstrated how effective anomaly detection can be achieved in environments with limited data, overcoming challenges such as high labeling costs, and need for subject-

matter expertise. The insights gained from these models provide a foundation for developing more scalable, autonomous, and cost-effective quality control solutions for manufacturing, where balancing accuracy, efficiency, and cost is essential.

7.1 Summary

- **Chapter 1: Introduction:** In this chapter, the motivation for improving the efficiency of solder joint inspection in PCB manufacturing is introduced. This discussion highlights the limitations of current inspection methods, including manual examination and supervised learning, are discussed, establishing the need for unsupervised learning as a viable alternative to reduce dependency on labeled data.
- **Chapter 2: Theoretical Background:** This chapter lays out the theoretical knowledge required to understand the research. It covers the concepts of supervised and unsupervised image processing, highlighting the advantages of using unsupervised learning for anomaly detection. It also introduces relevant models and techniques, such as convolutional neural networks (CNNs), PatchCore, EfficientAD, etc., used in supervised and unsupervised approaches.
- **Chapter 3: Methods:** The methodology chapter describes the dataset used in this study, which consists of images of solder joints that are either normal or defective. The experimental setup includes training various unsupervised models from the Anomalib library, such as PatchCore, DFM, and EfficientAD. The processes for hyperparameter tuning, data preparation, and model evaluation are outlined, along with the metrics used for evaluating model performance.
- **Chapter 4: Results:** This chapter presents the results obtained from evaluating the performance of the different unsupervised models on our dataset. PatchCore emerged as the best-performing model. Comparative analysis of DFM, EfficientAD, and other models is also provided, highlighting the strengths and weaknesses of each model in anomaly detection.
- **Chapter 5: Discussion:** The discussion chapter explores the implications of the results, emphasizing the scalability and efficiency of unsupervised methods for anomaly detection in PCB manufacturing. Challenges faced during the research, such as the long training times of some models, are also addressed.
- **Chapter 6: Future Works:** This chapter discusses the potential future directions for this research. Suggestions include exploring the use of Vision Transformers for enhanced anomaly detection and optimizing current methods to further reduce training times and improve accuracy. This chapter also highlights the broader applicability of unsupervised learning techniques in other defect detection tasks.
- **Chapter 7: Conclusion:** The conclusion summarizes the key findings, highlighting the benefits of using unsupervised learning for anomaly detection in PCB manufacturing. It also highlights how unsupervised models can make the inspection process more scalable and reduce the dependence on manual labeling. This chapter concludes by discussing future research opportunities, including the integration of advanced models like Vision Transformers.

Acronyms

PCB Printed Circuit Board

FC Fully Connected

NG Not Good

GPU Graphics Processing Unit

CPU Central Processing Unit

TPU Tensor Processing Unit

IPU Intelligence Processing unit

ResNet Residual Neural Network

AUROC Area Under the Receiver-Operating Curve

AUPR Area Under the Precision-Recall Curve

DL Deep Learning

CV Computer Vision

TPR True Positive Rate

FPR False Positive Rate

TP True Positive

FP False Positive

TN True Negative

FN False Negative

SVM Support Vector Machine

K-NN k-Nearest Neighbors

CNN Convolutional Neural Network

YOLO You Only Look Once

VAE Variational Auto-Encoder

GAN Generative Adversarial Network

DFM Deep Feature Modeling

DNN Deep Neural Networks

PCA Principal Component Analysis

t-SNE t-Distributed Stochastic Neighbor Embedding

GMM Gaussian Mixture Models

KDE Kernel Density Estimation

PDN Patch Description Network

S-T student-teacher

ViT Vision Transformer

AUC Area Under the Curve

DFKDE Deep Feature Kernel Density Estimation

ANN Artificial Neural Networks

FNN Feed Forward Neural Networks

ReLu Rectified Linear Unit

IOU Intersection Over Union

FPN Feature Pyramid Network

PAN Path Aggregation Network

NLL Negative Log-Likelihood

FRE Frequentist

DeiT Data-Efficient Image transformer

CaiT Class-Attention in Image Transformer

AOI Automated Optical Inspection

NLP Natural Language Processing

MLP Multi-Layer Perceptron

MSA Multiheaded Self-Attention

LN Layer Normalization

ML Machine Learning

Bibliography

[1] Anomalib. Anomalib documentation, 2024. URL https://anomalib.readthedocs.io/en/latest/index.html.

[2] Karsten Roth, Latha Pemula, Joaquin Zepeda, Bernhard Schölkopf, Thomas Brox, and Peter Gehler. Towards total recall in industrial anomaly detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14318–14328, 2022.

[3] Nilesh A Ahuja, Ibrahima Ndiour, Trushant Kalyanpur, and Omesh Tickoo. Probabilistic modeling of deep features for out-of-distribution and adversarial detection. *arXiv preprint arXiv:1909.11786*, 2019.

[4] Jiawei Yu, Ye Zheng, Xiang Wang, Wei Li, Yushuang Wu, Rui Zhao, and Liwei Wu. Fastflow: Unsupervised anomaly detection and localization via 2d normalizing flows. *arXiv preprint arXiv:2111.07677*, 2021.

[5] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Xiaoyuan Gong, Yuewei Bai, Yiqun Liu, and Hua Mu. Application of deep learning in defect detection. In *Journal of Physics: Conference Series*, volume 1684, page 012030. IOP Publishing, 2020.

[11] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[12] Olga Fink, Qin Wang, Markus Svensen, Pierre Dersin, Wan-Jui Lee, and Melanie Ducoffe. Potential, challenges and future directions for deep learning in prognostics and health management applications. *Engineering Applications of Artificial Intelligence*, 92:103678, 2020.

[13] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[14] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal processing*, 99:215–249, 2014.

[15] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[16] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[18] Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International conference on information processing in medical imaging*, pages 146–157. Springer, 2017.

[19] GeeksforGeeks. Supervised and unsupervised learning, 2024. URL https://www.geeksforgeeks.org/supervised-unsupervised-learning/.

[20] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. URL https://arxiv.org/abs/1511.08458.

[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[22] GeeksforGeeks. Introduction to convolutional neural networks, 2024. URL https://www.geeksforgeeks.org/introduction-convolution-neural-network/.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[24] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi: 10.1109/MSP.2012.2211477.

[25] Xiaojie Zhao, Lin Wang, Yan Zhang, et al. A review of convolutional neural networks in computer vision. *Artificial Intelligence Review*, 57:99, 2024. doi: 10.1007/s10462-024-10721-6. URL https://doi.org/10.1007/s10462-024-10721-6. Accepted: 4 February 2024, Published: 23 March 2024.

[26] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[27] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[28] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[29] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Training data-efficient image transformers & distillation through attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/touvron21a.html.

[30] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 32–42, 2021.

[31] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[32] Chien-Yao Wang and Hong-Yuan Mark Liao. Yolov1 to yolov10: The fastest and most accurate real-time object detection systems. *arXiv preprint arXiv:2408.09332*, 2024.

[33] Joseph Redmon. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[34] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 13029–13038, 2021.

[35] Glenn Jocher. Ultralytics yolov5, 2020. URL https://github.com/ultralytics/yolov5.

[36] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL https://github.com/ultralytics/ultralytics.

[37] Ultralytics. Yolov10 model documentation, 2024. URL https://docs.ultralytics.com/models/yolov10/. Accessed: 5 September 2024.

[38] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017. URL https://arxiv.org/abs/1605.07146.

[39] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9592–9600, 2019.

[40] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[41] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[42] Kilian Batzner, Lars Heckler, and Rebecca König. Efficientad: Accurate visual anomaly detection at millisecond-level latencies, 2024. URL https://arxiv.org/abs/2303.14535.

[43] Yang Zou, Jongheon Jeong, Latha Pemula, Dongqing Zhang, and Onkar Dabeer. Spot-the-difference self-supervised pre-training for anomaly detection and segmentation. In *European Conference on Computer Vision*, pages 392–408. Springer, 2022.

[44] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[45] Yona Falinie A Gaus, Neelanjan Bhowmik, Brian KS Isaac-Medina, Hubert PH Shum, Amir Atapour-Abarghouei, and Toby P Breckon. Region-based appearance and flow characteristics for anomaly detection in infrared surveillance imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2995–3005, 2023.

[46] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *University of Toronto*, 2009.

[47] Pankaj Mishra, Riccardo Verk, Daniele Fornasier, Claudio Piciarelli, and Gian Luca Foresti. Vt-adl: A vision transformer network for image anomaly detection and localization. In *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*. IEEE, June 2021. doi: 10.1109/isie45552.2021.9576231. URL http://dx.doi.org/10.1109/ISIE45552.2021.9576231.

[48] IBM. Principal component analysis, December 2023. URL https://www.ibm.com/topics/principal-component-analysis. Published: 8 December 2023.

[49] Cosma Shalizi. Advanced data analysis from an elementary point of view. *Citeseer*, 2013.

[50] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A deep learning library for anomaly detection. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 1706–1710. IEEE, 2022.

[51] OpenVINO Toolkit. Openvino toolkit: Optimized deep learning inference, 2024. URL https://github.com/openvinotoolkit/openvino.

[52] Weili Fang, Peter ED Love, Hanbin Luo, and Lieyun Ding. Computer vision for behaviour-based safety in construction: A review and future directions. *Advanced Engineering Informatics*, 43:100980, 2020.

[53] IBM. What is image segmentation?, 2024. URL https://www.ibm.com/topics/image-segmentation.

[54] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8): 861–874, 2006.

[55] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30:271–274, 1998. doi: 10.1023/A:1017181826899. URL https://link.springer.com/article/10.1023/A:1017181826899.

[56] Stephen M. Walker II. Accuracy, precision, recall, and f1 score - explained, 2024. URL https://klu.ai/glossary/accuracy-precision-recall-f1.

[57] David M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation, 2020. URL https://arxiv.org/abs/2010.16061.

[58] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

Declaration

I hereby certify that I have written this thesis independently and that I have not used any sources or aids other than those indicated, that all passages of the work which have been taken over verbatim or in spirit from other sources from other sources have been marked as such and that the work has not yet been submitted to any examination authority in the same or a similar form.

Erlangen, November 14, 2024 Rohit Potdukhe