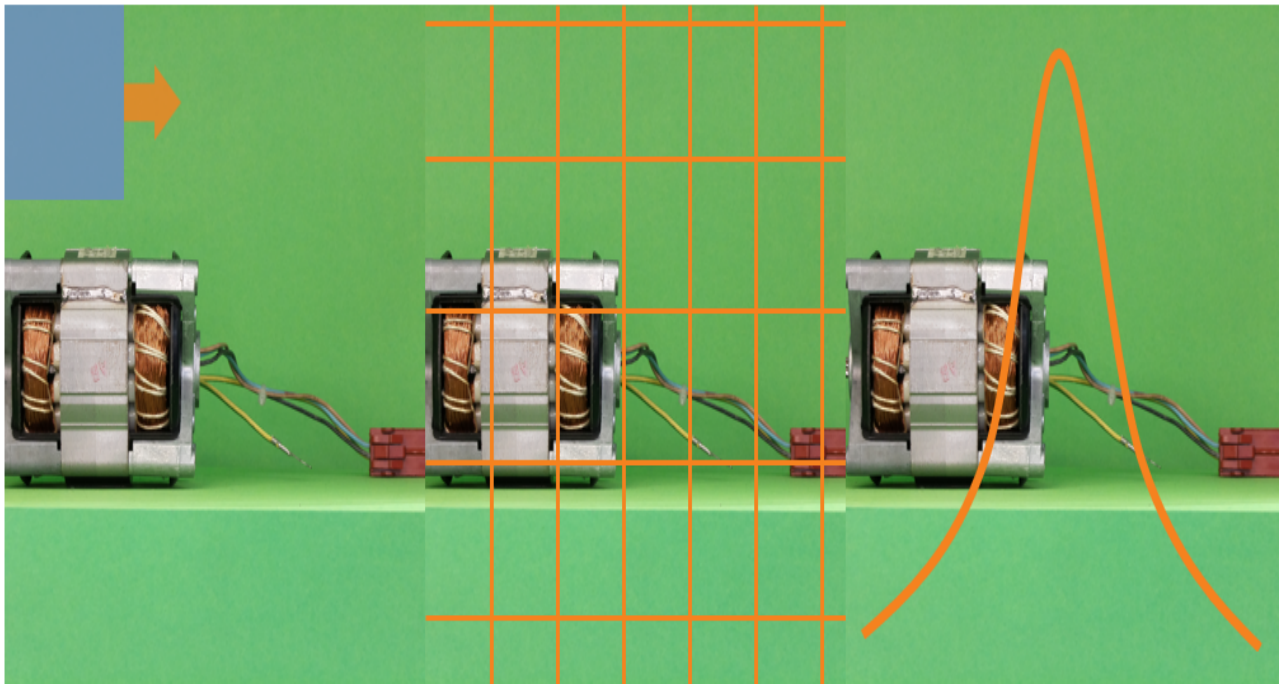
FAPS

Comparative Analysis of Deep Learning Methods for Automated Visual Inspection of Electric Motor Components

Scientific Report on the Applied AI Project
as part of the Master's Program in Artificial Intelligence

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institute for Factory Automation and Production Systems
Prof. Dr.-Ing. Jörg Franke



Author: Muhammad Abdullah, 23016193

Supervisor(s): Prof. Dr.-Ing. Jörg Franke
Andreas Mayr, M.Sc., M.Sc.

Submission date: 02.02.2024

Processing time: 6 months

Declaration

I hereby declare that I have written this project report without outside help and without using any sources other than those indicated. This report has not been submitted in the same or a similar version to any other examination office and been accepted as part of an examination performance. All statements that have been adopted literally or paraphrased are marked as such.

Erlangen, 02.02.2024

Muhammad Abdullah

Contents

List of Figures

List of Tables

List of Abbreviations

AE	Autoencoder
AI	Artificial Intelligence
ANN	Artificial Neural Networks
AVI	Automated Visual Inspection
BCE	Binary Cross Entropy
CNNs	Convolutional Neural Networks
CrossViT	Cross-Attention Multi-Scale Vision Transformer
DenseNets	Densely Connected Convolutional Networks
DL	Deep Learning
FC	Fully Connected Layer
MHA	Multi-Head Attention
MLP	Multilayer Perceptrons
NLP	Natural Language Processing
ResNet	Residual Networks
VAE	Variational Autoencoder
VBM	Variational Bayesian Methods
VI	Visual Inspection
ViT	Vision Transformers

1 Introduction

Visual inspection (VI) is the process of determining if a product deviates from a given set of specifications [11]. It generally involves the measurement of specific part features such as assembly integrity, surface finish, and geometric dimensions [11]. Human inspectors in many industries carry out the inspection whose performance is generally inconsistent and inadequate [12]. Many inspection tasks are also considered time-consuming and boring for humans [11]. Automated visual inspection (AVI) replaces human inspectors with machine vision that reduces labor costs and improves quality [11]. Other benefits of AVI include performing inspection in unfavorable environments, freeing humans from mundane work and better information storage for future management decisions [12].

AVI usually incorporates machine vision systems at three levels [13]. First in detecting and localization of defects, then performing pixel-wise image segmentation, and lastly in defects classification [13]. Traditional machine vision methods employ hand-crafted features computed by domain-specific industry experts based on texture, color, or shape [14]. These traditional methods are mostly regarded as trademarks [13] and involve a lengthy development cycle [15]. Recent advancements in deep learning (DL) have enabled us to learn important features directly from the raw data, eliminating the need for hand-crafted features [15]. DL methods have reached and even exceeded human-level performance on image-related tasks such as image classification [16]. The widespread adoption of deep learning methods still faces several challenges which include availability of datasets and manual work required for data labeling [15].

Convolutional neural networks (CNNs) are DL-based architectures that have shown remarkable performance in automatic feature extraction. After the successful classification of the ImageNet [17] dataset by AlexNet [18] in 2012, many new CNN architectures were proposed enabling even higher performance on ImageNet and many vision-related tasks. As the CNN-like architectures have been successful in vision, Transformer-based architectures [19] have shown great results in language modeling. Recently, Transformers have also been applied to vision tasks [20] performing similarly to CNNs that lead to more research in Vision Transformers (ViT). Unlike CNN and ViT, autoencoder-based [21] architectures have also been proposed to learn the representation of non-defective samples while skipping the need of training data for defective samples and considering them as out-of-distribution samples or anomalies [22].

This project compares different DL approaches on a small but complex multi-view dataset of electric motors. The objective is to detect surface defects by categorizing electric motor components such as screws, covers, winding heads, etc., into either defective or non-defective. A FAPS proprietary multi-view electric motor dataset [8] has been used for this study. Three different DL-based architectures incorporating a CNN, a ViT, and a variational autoencoder (VAE) [23] are used to perform defects classification and subsequently, their performances are evaluated. The report's structure is as follows: Chapter 2 is devoted to explaining fundamental background knowledge of DL approaches, related work in VI, and defining important terminology for the rest of the report. Chapter 3 discusses the specific problem at hand, i. e., electric motor dataset and identification of assembly faults. Chapter 4 explains the experimental process incorporating data preprocessing, network training, hyperparameter optimization, evaluation, and comparison. The final chapter provides a summary of the project and delves into the future outook.

2 Fundamentals, Related Work, and Research Objective

This chapter comprises an overview of DL methods used within the scope of this project and their development over the recent years. It references related work using similar methods for AVI within industrial settings and ultimately drives the research objective while outlining the selected approach.

2.1 DL Methods for AVI

One of the core principles of Industry 4.0 is to use artificial intelligence (AI) to make machines capable of self-acquisition, self-adjustment, self-determination, and self-control [24]. AVI is an important part of this transition towards smart manufacturing [25]. AVI systems can detect defects at a much higher speed matching the high speed production [12]. They increase consistency while reducing labor costs [11]. Data collected is also saved and used for future analysis [12] and overall process improvement. Figure 1 outlines the fundamental pipeline used in AVI systems.

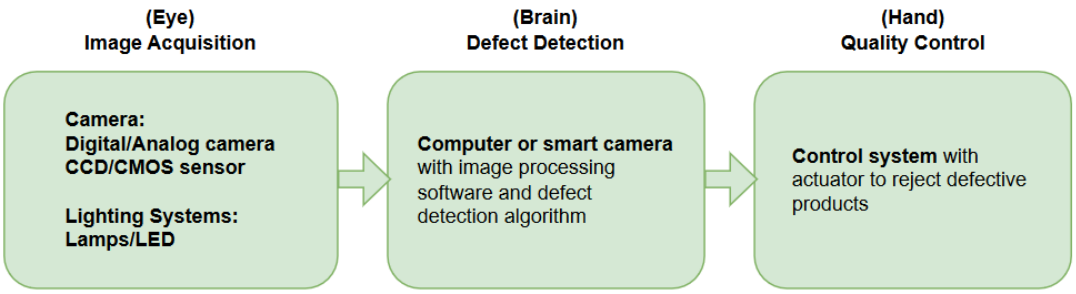


Figure 1: Pipeline for AVI systems based on [1].

Traditional defect detection algorithms in AVI systems have deployed hand-crafted features for the detection and classification of defects. These hand-crafted features are based on traditional computer vision and statistical signal processing [14]. Fourier transforms, Gabor filtering, and histograms are some of the common image processing techniques [14] that can be used to compute features to train a classifier for defect detection. Although traditional algorithms are still used in many industries, they require expert knowledge in specific industrial domains. The quality of features is insufficient to achieve satisfactory inspection performance [26] in many complex inspection tasks.

DL enables the computation of features in an automated and data-driven [15] manner. Data quality plays an important role in improving performance [27]. Interestingly, DL is built upon artificial neural networks (ANN) which on the contrary are quite old. Rosenblatt in his 1957 seminal paper [28] noted that it is hard for a general analytic program to recognize objects from different angles, positions, and postures without a large library of reference images. He suggested Perceptron, a probabilistic model that is closely analogous to the perceptual processes of the biological brain. Its performance can be described as a process of learning and

it gains its reliability from the properties of statistical measurements obtained from large populations of elements [28]. But this perceptron could not go far in research as it was pointed out that it was not able to solve the XOR problem [29]. Considering the failure of basic perceptron, researchers started exploring multilayer perceptron (MLP) by introducing hidden layers between input and output layers. MLP with hidden layers correctly solves the XOR problem. However, as the hidden layers grow, it is difficult to train their weights optimally. Rumelhart et al. introduced a gradient-based learning strategy that minimizes the error between actual output and expected output by updating the weights [30]. As a result, hidden units weights come to represent important features of task domain [30].

For images, automated features encapsulated in the hidden layers of ANN are mainly extracted by convolutional layers. These features can be used to train classifiers for downstream tasks. In the context of AVI, these automated features are superior to traditional hand-crafted features in many ways. First, they don't require specific domain experts to compute. They are computed directly from the training dataset in a supervised learning manner. Training these networks is generally computationally intensive and requires large amounts of labeled data. Labeling data is also a labor-intensive task and sometimes requires domain experts. More recently, unsupervised learning techniques which learn from unlabeled data, and semi-supervised learning techniques which learn from limited label data have been proposed and have shown promising results [31].

2.1.1 Convolutional Neural Network

CNNs are ANNs that use convolution operators instead of MLP-style matrix multiplication in hidden layers. They are used for data that has a grid-like structure [32]. Since images are composed of 2D or 3D (color images) grids, CNN is generally used in vision-related tasks. In a convolution operation, a small kernel or filter is convolved over the image resulting in a feature map. For example, if the Sobel filter is convolved with the image, it results in edges. We can use many different types of filters to compute our feature maps as the number of feature maps depends on the number of filters applied. But the computation of these filters is not manual, CNN learns them automatically through gradient-based backpropagation [30] using the same MLP-style training process. CNNs are translational equivariant as the same filter is convolved over the whole image (parameter sharing) [32]. The translational equivariant property of CNNs aids in improving their ability to generalize well on unseen data [5].

Figure [2] explains the architecture of LeNET-5 which was one of the first CNNs trained successfully to recognize hand-written digits. LeNET-5 contains 7 layers that include convolutional, sub-sampling, and fully connected layers (FC) [2]. The sub-sampling or pooling layers are applied after each convolutional layer to reduce the spatial features and introduce invariance to translation [2]. Max-pooling and average-pooling are some common types of pooling operations. Pooling layers also introduce invariance to unwanted transformations if performed along filters' dimension [32]. The last layers contain an FC layer and a non-linear activation layer that is applied after the FC layer [2]. For classification tasks, an additional softmax layer is applied at the end to compute probabilities for each class.

Training deep CNNs is very difficult because they require large computational resources and are prone to overfitting [33]. Overfitting occurs when the network performs well on the training dataset but not on the held-out test dataset. It is similar to memorizing a training dataset. As the networks become deeper, they have more parameters and can easily overfit.

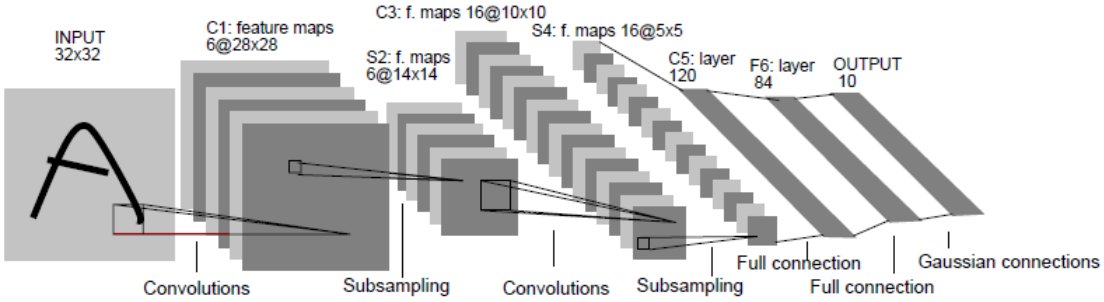


Figure 2: Architecture of LeNet5, a CNN used for hand-written digits recognition [2].

AlexNet [18] was the first network after LeNet-5 [2] which trained a large CNN to classify ImageNet [17] dataset. They used the GPU implementation of the convolution operation to parallelize the computations. To counter overfitting, they introduced a regularization technique called dropout [34] and inserted dropout layers between fully connected layers which randomly omit some neurons output and force all neurons to learn features that are generally helpful for finding correct answer [34]. Also, they noted that increasing the convolution layers and making CNN deeper has a positive impact on performance. However, they could not train further deeper CNNs because of the memory limitation of the available GPUs in 2012 [18].

VGGNet [35] was introduced in 2015 in the quest for deeper networks without increasing parameters. They reduced the convolutional filter dimensions and added more conv layers to increase the depth. By slightly modifying the architectures, they achieved a 16% performance boost on top-1 validation error [35]. But as networks started becoming deeper, they started becoming hard to train [36]. To enable training of deeper CNNs, residual networks (ResNet) [36] were introduced. ResNets introduced a parameter-free identity shortcut to form a residual block. By stacking multiple of these residual blocks, ResNets were able to overcome the optimization difficulty and demonstrate accuracy gains when the depth increases [36].

Densely connected convolutional networks (DenseNets) [3] were inspired by ResNets. Rather than adding identity shortcuts only to the last layer, they introduced identity shortcuts to all preceding layers within a denseblock [3]. DenseNets alleviate the vanishing-gradient problem and strengthen feature propagation as they allow the maximum information flow [3]. They also have fewer parameters as they use fewer convolutional filters (less wide) in each layer. Figure 3 shows a basic denseblock. Multiple of these dense blocks are stacked in a single DenseNet [3].

Around 2020, it became clear that network depth and width [37] play an important role in network architectures. EfficientNet [38] introduced a grid-search-based compound scaling method to empirically find the best network depth and width values. They slightly outperform ResNet and DenseNet while having fewer parameters. [38].

2.1.2 Vision Transformer

In the realm of visual recognition, the primary focus of advancements during the decade spanning from 2010 to 2019 predominantly centered on CNNs. And the best-performing CNN, EfficientNet, reached 84.4% top-1 accuracy on the ImageNet test set [38].

Around the same time, Transformer [19] architecture brought a revolution in sequence model-

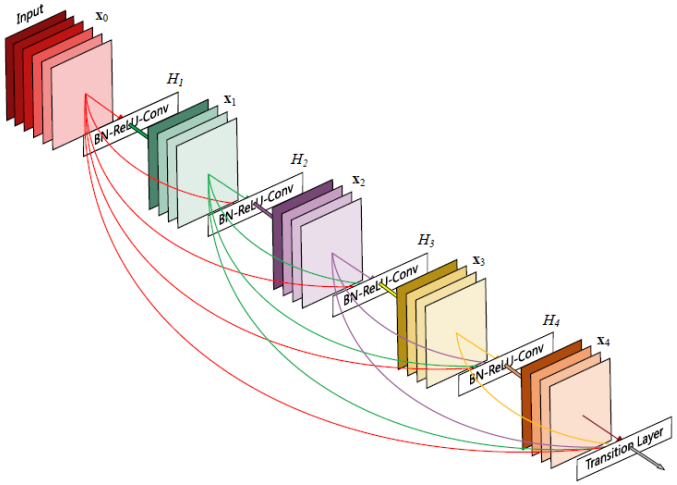


Figure 3: A 5-layer dense block based on [3]. Each layer takes all preceding feature maps as input. Here H denotes a composite function of three layers (batch normalization, ReLU, and conv layer).

ing and machine translation tasks in natural language processing (NLP). Transformers employ an encoder-decoder architecture [19]. They have a special attention mechanism allowing modeling of dependencies between different tokens without regard to their distance in the input or output sequences [19]. The illustration in Figure 4 portrays the Transformer architecture. The encoder block consists of a multi-head attention (MHA) module that contains several self-attention layers running in parallel. Self-attention is an attention mechanism based on scaled dot-product relating different positions of the input sequence to compute a representation of the sequence. These representations are passed through a feed-forward neural network that projects hidden dimensions back to input dimensions. These learned embeddings are passed to the decoder which also employs multi-head cross-attention between input embedding and predicted token embedding. The decoder also employs masked multi-head self-attention of predicted tokens in an auto-regressive manner. It means predicted tokens attend only to previously predicted tokens. Multiple encoder and decoder blocks are stacked to get deeper architectures. [19]

Figure 5 shows the architecture of scaled dot-product attention and multi-head attention. In self-attention, Query (Q), Key (K) and Value (V) vectors come from the same sequence which results in a sequence attending to itself. In cross-attention, the query comes from one sequence and key and value vectors come from another sequence which results in one sequence attending to the other. Cross-attention has also become a suitable conditioning method in many recent latent diffusion models [39].

Although transformer-based architecture has become state-of-the-art in many language modeling tasks, its vision application was not fully explored. In 2021, a ViT was proposed [5] containing only a transformer encoder. The basic idea is to convert the image into 16x16 patches, flatten these patches, and treat them like word sequences. An extra learnable embedding is added whose state at the output of the Transformer encoder serves as the image representation [5]. An extra MLP is added at the end to classify images based on image representation [5]. An exemplary ViT architecture is depicted in Figure 6.

Theoretically speaking, it is difficult for ViT to beat CNNs as CNNs learn inductive biases such

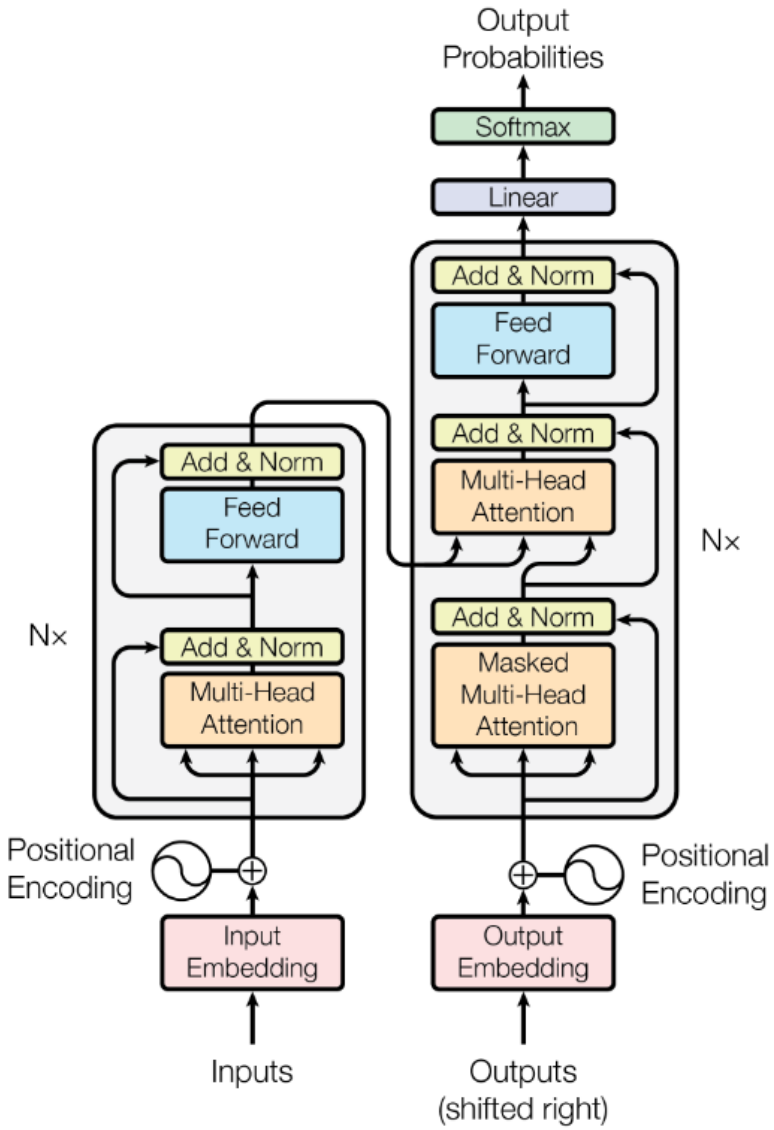
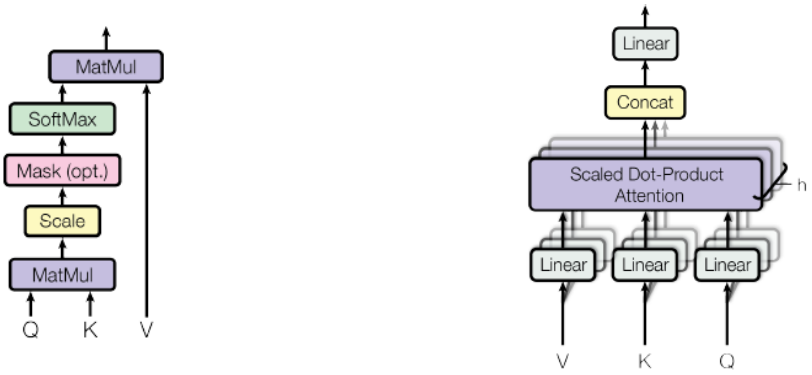


Figure 4: The illustration of the Transformer model architecture based on [4] is portrayed here. The encoder block is on the left side while the decoder block is on the right side.

as translation equivariance and locality [5]. The lack of these inductive biases in Transformer hinders its ability to generalize effectively when there is not enough data available [5]. But if ViT is trained on larger datasets, the large-scale training beats inductive biases and ViT can outperform CNNs [5].

Recently, many variations of vision transformers have been proposed. For example, a hierarchical transformer called Swin Transformer is proposed in [40]. Unlike vanilla ViT which has fixed 16x16 patches, the Swin Transformer constructs hierarchical feature maps starting from small-sized patches and gradually merging neighboring patches into large-sized patches in deeper Transformer layers. With these hierarchical feature maps incorporating image scale, it can serve as a general-purpose backbone for vision tasks [40]. Cross-attention multi-scale vision Transformer (CrossViT) [10] is another ViT architecture that takes different image scales into account. Unlike the Swin Transformer, which can have many scales, it employs a dual



a) Scaled dot-product attention *b) Multi-head attention*

Figure 5: Scaled dot-product attention and multi-head attention which consists of several attention layers running in parallel [4]

branch strategy incorporating only two scales. One ViT processes small patches, while the other processes large patches. Embedding for small and large patches are fused using cross-attention to complement each other [10]. Multi-scale ViTs, particularly CrossViT, are better in the sense that they require fewer parameters [10] and can be used if data is limited.

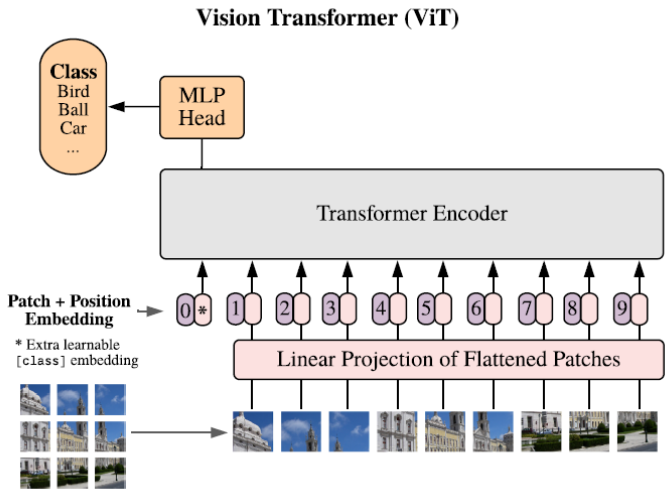


Figure 6: ViT based on [5] splits the image into patches, flattens the patches, adds position embedding to preserve the order of patches, and passes them through the standard Transformer encoder.

2.1.3 Variational Autoencoder

Training a network in a supervised learning fashion requires labeled data. Getting a sufficient amount of large clean labeled data might not be possible in many industries due to large outliers and pervasive noise [41]. An alternative approach is to treat correct samples as normal and all the faulty and substandard samples are considered as anomaly or abnormal. We try to learn the representation of normal samples by projecting them into a latent space and

then reconstructing them back into the original space. Normal samples tend to have lower reconstruction errors as compared to abnormal samples. By selecting a suitable threshold for reconstruction error, we can discriminate between normal and abnormal samples. It should be noted that this approach is not fully unsupervised as we still need labels for normal samples.

Autoencoder (AE) was originally proposed as a method to perform dimensionality reduction using neural networks [42]. It consists of an encoder block to learn robust features in latent space and a decoder block to reconstruct the input representation from latent space features. AEs have also shown robustness in denoising partially-destroyed inputs [43]. High dimensional inputs like images contain a lot of redundant data and can be easily recovered if some part of the image is hidden just like humans ability to recognize partially occluded or corrupted images [43]. An important aspect of AE is to prevent them from learning identity mapping. For this, the use of some form of regularization becomes essential to avoid uninteresting solutions where the AE could perfectly reconstruct the input without needing to extract any useful feature [44].

VAE [23] is a type AE that learns the probability distributions of sample representations rather than directly learning the latent features. It uses a variational Bayesian method (VBM) to estimate the intractable posterior $p_\theta(z|x)$ using stochastic variational inference. During training, we try to learn the representation of samples by projecting them into a parameterized latent space. Then we sample from that parameterized latent space and try to reconstruct them. Because a stochastic sampling operation is performed for reconstruction, a re-parameterization trick is used which rewrites sampling into learnable parameters(mean and variance) and a noise term. This noise term is generally sampled from a normal distribution with a mean of 0 and a variance of 1, making it independent of the gradient flow. We suggest the reader to [45, 23] for mathematical derivations. VAE is considered a generative model because we can sample from a learned probability distribution and pass it from the decoder to get a novel input. Figure 7 shows the process of reconstructing input using a VAE.

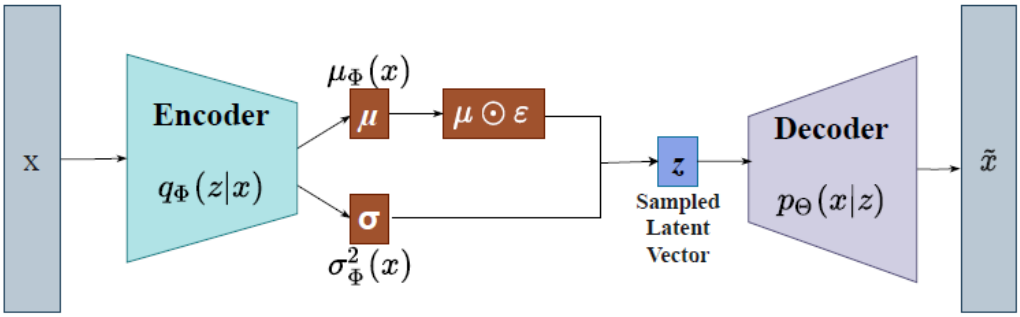


Figure 7: VAE architecture [6]

2.2 Related Work in AVI

DL-based methods have been proposed recently to detect surface defects for industrial quality control ranging from supervised to semi-supervised and unsupervised learning. Here, we list some of the papers that employ approaches similar to the ones used in this study for the classification of surface defects. Song et al. [46] use a custom three-layer CNN network to find surface defects in metal screws. The defects they considered include surface damage,

surface dirt, and stripped screws. They also employ a two-stage process. First, they detect screws irrespective of the defect and then classify them. Fu et al. [47] uses a CNN-based SqueezeNet [48] to classify surface defects in steel strips. Farnsworth et at. [49] use ResNet to find winding defects like gap, wire crossing and double winding in electrical coils. Avdelidis et al. [50] use DenseNet [3] to find damaged paint, scratches and, dents on the surface of unmanned aerial vehicle (UAV). ViTs are introduced quite recently and there are not many papers that use them in AVI context but the trend is shifting. Hütten et al. [51] compare ViT and CNN on three different datasets of visual damage assessment and concluded that ViT models achieve at least equivalent performance to CNNs in industrial applications with sparse data available, and significantly surpass them in increasingly complex tasks. DL has also triggered the development of numerous, novel anomaly detection methods specific to AVI [52]. Yu et al. [53] propose a stacked convolutional sparse denoising AE (SCSDAE) for wafer map pattern recognition (WMPR) in semiconductor manufacturing processes. Though there are a few other papers using ViT and AE for defect detection, they belong to different domains and are thus beyond the scope of this project report.

2.3 Research Objective and Applied Procedure

ViT and AE-based defect detection methods hold potential for AVI in production environments. However, as of now, no such methods have been identified for inspecting electric motor components. This project aims to use a ViT, an AE, and CNN-based feature extractor to perform binary classification for a variety of surface defects in a proprietary electric motor dataset which was created as a part of student preparatory work. The next chapter, in this report, will explain the nature of this dataset and the associated surface defects. Three methods are used to perform binary classification: A CNN-based DenseNet121 [3], a ViT-based CrossVit [10], and a VAE-based architecture. A performance comparison between different methods is provided at the end.

3 Task Description and Data Understanding

In this chapter, the focus is on discussing the characteristics of assembly defects in electric motors and the type of dataset used. It delves into the main components of an electric motor, the assembly procedure employed, and the potential defects that may arise during the process. Figure 8 shows an example of a fully assembled electric motor.

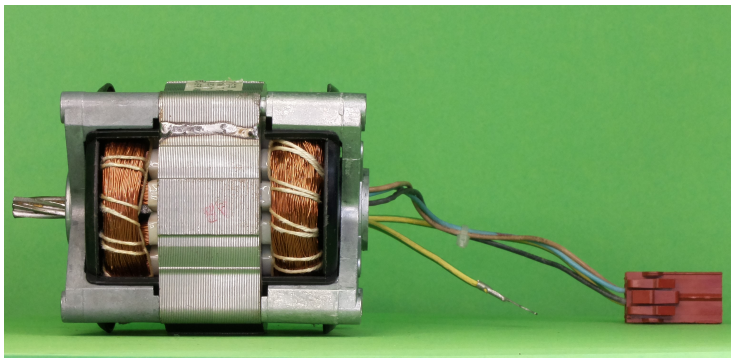


Figure 8: An exemplary assembled electric motor

3.1 Identification of Electric Motor Assembly Faults

In this section, the production methods used in assembling electric motors are discussed, along with the potential anomalies that may arise in the final assembly of the electric motors.

3.1.1 Production Processes of Electric Motor Assembly

There are several processes usually involved in the production of electric motors. It starts with housing and rotor cage production which involves producing an outer casing or structure that houses the motor's internal components. This can include techniques such as casting, machining, and cleaning. Shaft production also follows similar processes. The production of the laminated rotor and stator cores entails cutting thin layers of electrical steel, stacking, and joining them together, and then insulating them with varnish. Winding and contacting involve wrapping insulated copper or aluminum wires around the rotor and stator cores, and then connecting the wire ends to suitable leads for electrical conductivity. In permanent magnet synchronous (PSM) motors, permanent magnets are manufactured and mounted into the rotor structure. The final assembly involves bringing together all the motor components, including the rotor, stator, bearings, shaft, and housing. This step involves press-in operations, gluing, shrinking, screwing, and welding [7]. Finally, a visual inspection of the assembled motor is conducted to ensure that all components are free of defects. Figure 9 lists these different processes concisely.

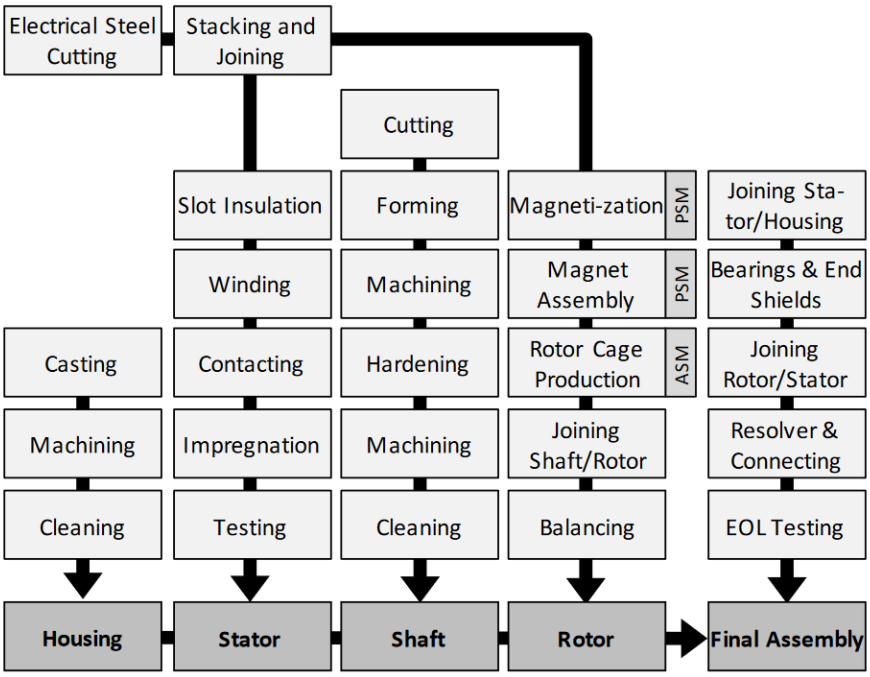


Figure 9: Simplified production process of electric motors based on [7]

3.1.2 Assembly Faults

There can be a lot of anomalies or faults introduced during the different production processes but the scope of this project is limited to assembly faults. Some of the faults that can occur during the assembly are listed below:

- Missing screws
- Screws not screwed properly
- Broken insulation cover
- Broken cable
- Missing cable lug
- Missing plug
- Broken bandaging threads or wire on winding head
- Gaps among sheets in sheet metal package

3.2 FAPS Proprietary Electric Motor Dataset

The FAPS proprietary electric motor dataset is a multi-view dataset containing images of the motor from six different views. The current version of the dataset was collected by Alexander Christoph during his master's thesis [8] and it contains 481 different assembled motor combinations. The purpose of collecting six different views is that some of the faults are view-dependent and can only be seen from certain views. Figure 10 shows these six views. The dataset contains 11 different object classes which are listed below:

- Cable lug

- Cable
- Cover
- Fixed end shield
- Floating bearing shield
- Plug red
- Rolling bearings
- Rotor shaft
- Screw
- Metal package
- Winding head

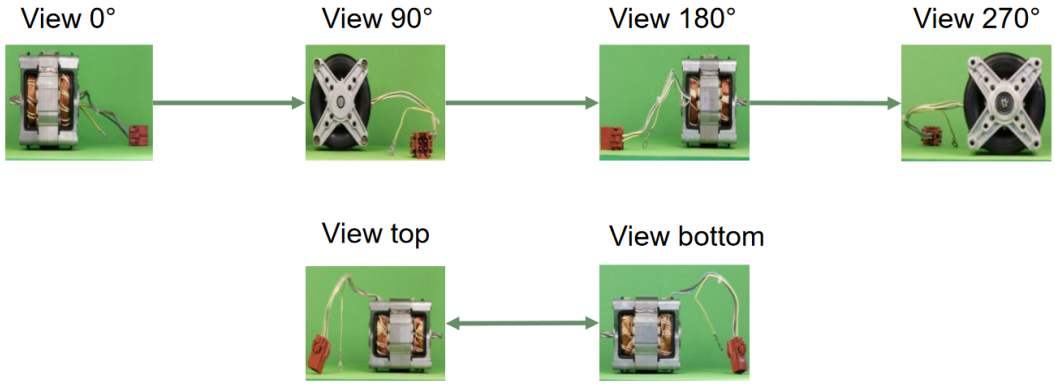


Figure 10: Six different views in the multi-view dataset collected by [8] For views 0-270°, motor is rotated 90° in clock-wise direction. To capture the top and bottom views, the motor is oriented such that its top and bottom sides face the camera.

Figure 11 an exemplary 0° view with annotations of different object classes. It can be seen that screws are not visible from this view but they can be seen from 90° view.

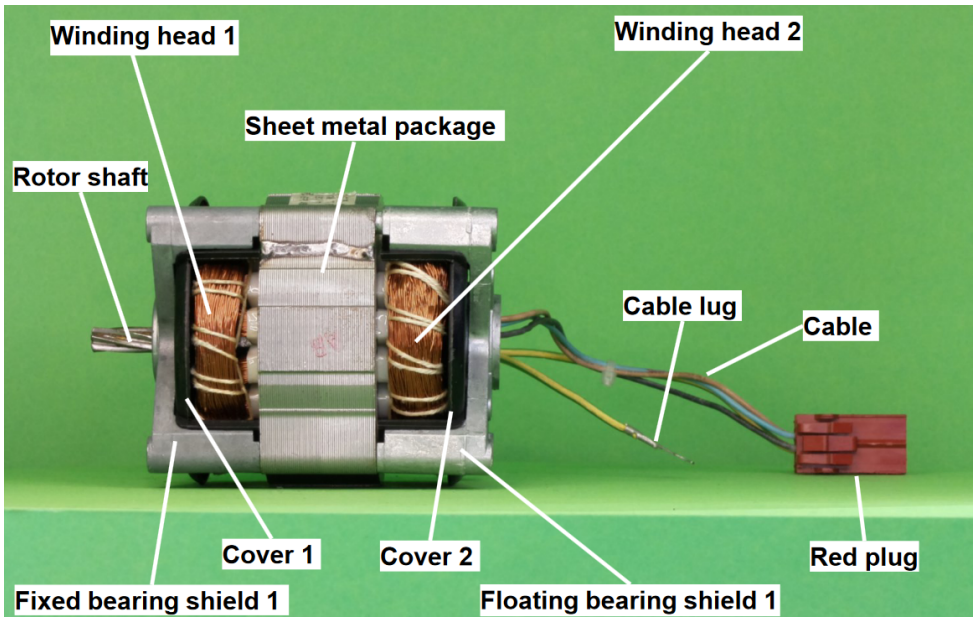


Figure 11: Annotation of different classes present in the multi-view dataset collected by [8]

Instead of directly classifying the object classes as defective or non-defective on the original images, a two-stage approach is considered [8]. The first stage involves using an object detector to detect 11 object classes irrespective of their surface damage. Binary classification on image crops obtained from the object detector is performed in the next stage. Alexander Christoph assessed this two-stage approach during his master's thesis, and the image crops utilized for binary classification were generated using his pipeline [8]. Figure 12 shows image crops obtained from the object detector.

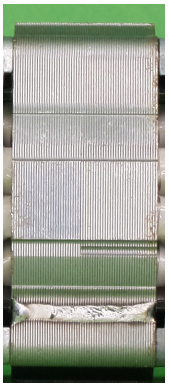
This project investigates the following five defects:

- Broken insulation cover
- Broken cable
- Screw not properly screwed
- Gaps in sheet metal package
- Broken bandaging threads on winding head

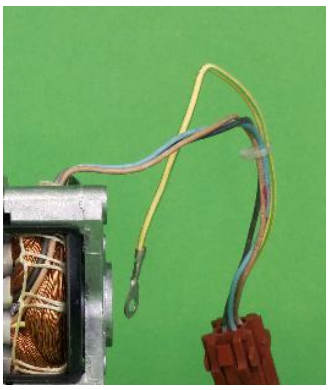
An illustration of these faults has been provided in Figure 13.

a) Cover crop

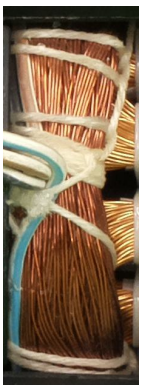
b) Screw crop



c) Sheet metal package crop



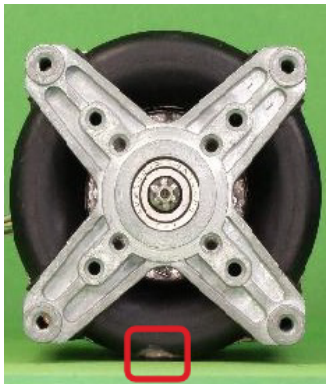
d) Cable crop



e) Winding head crop

Figure 12: Illustrations of image crops obtained from the object detector

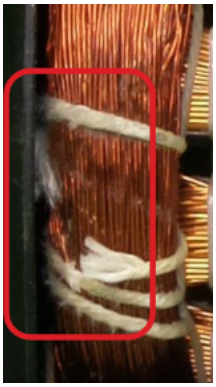
The dataset contains 481 different motor combinations. The motor components in these combinations are not unique as some of the components i.e. rotor, stator, and winding heads were reused in different combinations. The dataset is divided into training, validation, and test splits. These splits were created by Alexander Christoph [8]. The training split contains 328, the validation split contains 74, and the test split contains images from 79 combinations [8]. All splits contain images for all five surface defects though some of the surface defects have



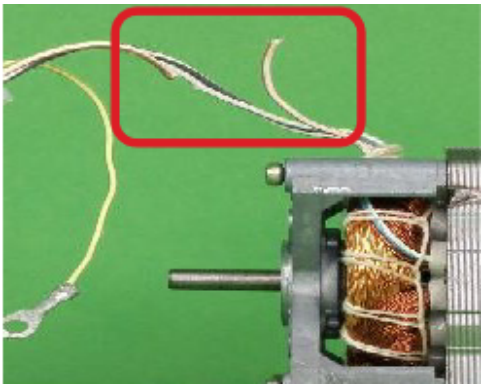
a) Broken insulation cover

b) Screws not properly screwed

c) Gaps among sheets in sheet metal package



d) Broken bandaging threads on winding wead



e) Broken cable

Figure 13: Illustrations of the five assembly defects considered for binary classification in this project

a largely imbalanced distribution between defective and non-defective images. Table 1 details the distribution of samples for each defect class in each split excluding augmented images as the image augmentations are performed in later steps. Table 2 shows that defective and non-defective images are not evenly distributed in each split but they are highly imbalanced. It can be seen that we have very few defective samples for all classes. The situation is worst for winding heads where there are only 66 samples with broken bandaging threads in the training set. Table 3 shows the distribution of unique defects counting for the fact that each defective component has been used in more than one motor combination. Here it can be seen that the situation is even worse for cable classification where we have just seven unique defects in the training set.

Table 1: Distribution of cropped images (excluding augmentations) in train, validation, and test split

Type	Train	Validation	Test
Cable	1999	441	444
Cover	2353	575	579
Screw	1132	250	269
Winding Head	2672	592	592
Sheet Metal Package	1336	296	296

Table 2: Distribution of defects in train, validation, and test split

Type	Train	Validation	Test
Cable	251 (12.5%)	53 (12%)	27 (6%)
Cover	174 (7.4%)	43 (7.5%)	42 (7.3%)
Screw	384 (34%)	83 (33.2%)	84 (31.2%)
Winding Head	66 (2.5%)	12 (2.0%)	25 (4.2%)
Sheet Metal Package	181 (13.5%)	23 (7.8%)	36 (12.2%)

Table 3: Unique defects in train, validation, and test split

Type	Train	Validation	Test
Cable	7	2	2
Cover	25	12	11
Winding Head	25	12	11
Sheet Metal Package	28	10	6

4 Data Preparation, Modeling, and Evaluation

This section discusses various data preprocessing steps implemented before feeding images into the DL network. Additionally, it covers the employed DL architectures and the achieved results of binary classification. Furthermore, it evaluates these results.

4.1 Data Preprocessing

As the dataset was highly imbalanced and scarce. Image augmentations are applied to counter data scarcity and oversampling of minority classes is done to handle imbalanced data.

4.1.1 Image Augmentations

Image augmentations have now become an important part of efficient DL pipelines [54]. They are used to handle data scarcity and introduce variations in input data that help in better generalization by improving regularization [54]. Simple image transforms such as mirroring or cropping and affine transforms, including horizontal and vertical translation or scaling can be applied to create augmented data which can be used to improve model robustness and increase accuracy [55]. The first kind of image augmentation is applied by extending the bounding boxes obtained from the object detector by 5, 20, and 40% respectively for each image. This is already performed by Alexander in an offline manner. We get these augmented images in our data. We additionally apply many augmentations mainly mirroring around the x-axis and y-axis, and affine transforms such as translation, rotation, and scaling. These augmentations are applied during training in an online manner. There are a total of five augmentations available, but randomly three augmentations are chosen for each image each time. We used a dedicated image augmentations library ImgAug [56] for this purpose. ImgAug can be integrated with Pytorch data loader and augmentations are performed in an online manner. The augmentation parameters are listed below:

- Mirror around the x-axis with probability 0.8
- Mirror around the y-axis with probability 0.8
- Translate image -5% to 5% (randomly selected) in both x and y direction
- Scale image with scaling coefficient ranging from 0.7 to 1.1
- Rotate image from -45° to 45°

4.1.2 Handling Imbalanced Data

Imbalanced data mainly cause two problems: (1) training is inefficient as most locations are easy samples that contribute no useful learning signal; (2) easy samples can overwhelm training and lead to a degenerate model [57]. As our data is highly imbalanced, further steps should be taken for efficient training. One approach to handle imbalanced data is bootstrapping where we oversample negative or hard samples and undersample easy samples. Another approach is to modify the loss functions directly so that it penalizes more for hard samples and less for easy samples while forcing the network to learn more discriminating features [57].

Methods that directly alter the loss functions encompass weighted cross entropy, typically with weights determined by inverse class frequency, and focal loss [57], where weighting is determined by employing two hyperparameters. In our experiments, we found it extremely difficult to efficiently fine-tune the focal loss hyperparameters. We ended up using the oversampling of defective samples. The probability of sampling is computed as the inverse of class frequency for both defective and non-defective samples.

4.2 Network Architectures

This section describes the network architectures we use to learn features and do binary classification for each defect class. For the CNN-based classifier, we selected DenseNets as dense connections in DenseNet have a regularizing effect that helps them counter overfitting in tasks with smaller training sets [3]. We use DenseNet121 [3] as a feature extractor and add one additional FC layer at the end to map output to 2 which is our number of classes for binary classification. For the ViT-based classifier, we use CrossViT [10] as a feature extractor as it achieves good performance even with fewer parameters [10]. We also modify the FC layer at the end of CrossViT. For VAE-based one-class anomaly detection, we used a vanilla VAE to reconstruct the image and used reconstruction loss to classify the image between defective and non-defective by applying a suitable threshold.

4.2.1 DenseNet121

Figure 14 shows the architecture of DenseNet121. DenseNet121 consists of 4 *Dense Blocks* denoted as DB and 3 transition layers denoted as TL. Each dense block consists of multiple dense layers. DB1 consists of 6 dense layers, DB2 consists of 12 dense layers, DB3 consists of 24 dense layers and DB4 consists of 16 dense layers. Every dense layer in every dense block contains convolution, batch-normalization, and ReLU operations and outputs 32 channels but takes input from all preceding layers and performs concatenation at channel dimension. Denseblocks do not change the spatial dimensions. Transition layers contain 1x1 convolutions followed by average pooling layers. Spatial down-sampling happens in the average pooling layer of transition layers. Adaptive average pooling is applied at the end before feeding learned features to an FC layer. Though the original DenseNet121 uses 224x224 input image dimensions, the network can accept variable input image dimensions due to the adaptive average pooling layer. In our training experiments, we used 256x256 input image dimensions. Also, we apply an additional 1000x2 FC layer at the end to project 1000-dimensional output to 2-dimensional output.

As we are performing binary classification, we apply Softmax at the output of the network and use CrossEntropy as the loss function during training. We initialize DenseNet121 from ImageNet pre-trained weights. We finetune all layers. We treat batch size, epochs, and initial learning rate as hyperparameters and use Optuna [58] for hyperparameter tuning. More details can be found in section 4.3.

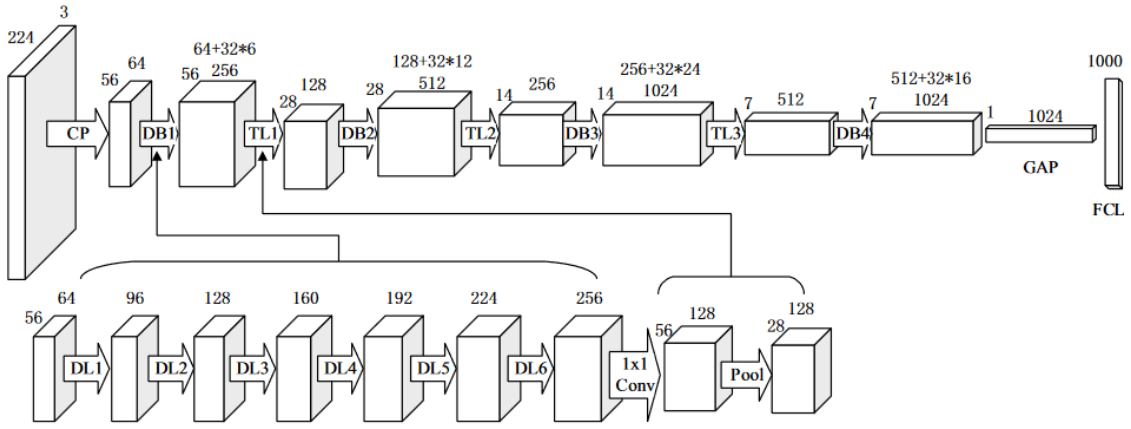


Figure 14: DenseNet121 [3] architecture [9]

4.2.2 CrossViT

Figure 15 shows the architecture of CrossViT. An image is divided into smaller and larger patches. The embedding of class tokens is initialized randomly and concatenated with the patch embeddings. One ViT processes smaller patch tokens and the other ViT processes larger patch tokens [10]. Both of these ViTs predict tokens for small and large patches. Remember, ViTs are still sequence to sequence mapper. The first output token is a class token. In an attempt to efficiently fuse representation for both scales, another cross-attention transformer is used that performs cross-attention between predicted small-patch class token and predicted larger patches and vice versa [10]. Class token embeddings for both smaller and larger patches after the cross-attention transformer are passed through an MLP which projects them from token dimension to output classes [10].

In CrossViT, many architecture-related parameters can be tuned for better performance. Table 4 displays the important architecture parameters and their value used during the training. We have selected these values from the CrossViT-S implementation specified in CrossViT paper [10]. The depth, token dimension, and patch size are the most important ones. Depth indicates the number of times a module was repeated. Further details on experiments and results are provided in section 4.3.

Table 4: Values used for CrossViT for different architecture parameters. These values (with slight modifications) are selected from the CrossViT-S implementation specified in the CrossViT paper. n/a indicates not applicable

Parameter	Small-patch ViT	Large-patch ViT	Cross-atten. ViT	Multi-scale embedder
Depth	2	3	2	3
Token dim	192	384	n/a	n/a
Patch size	16	64	n/a	n/a
No. Heads	8	8	8	n/a
Head dim	64	64	64	n/a
MLP dim	2048	2048	n/a	n/a

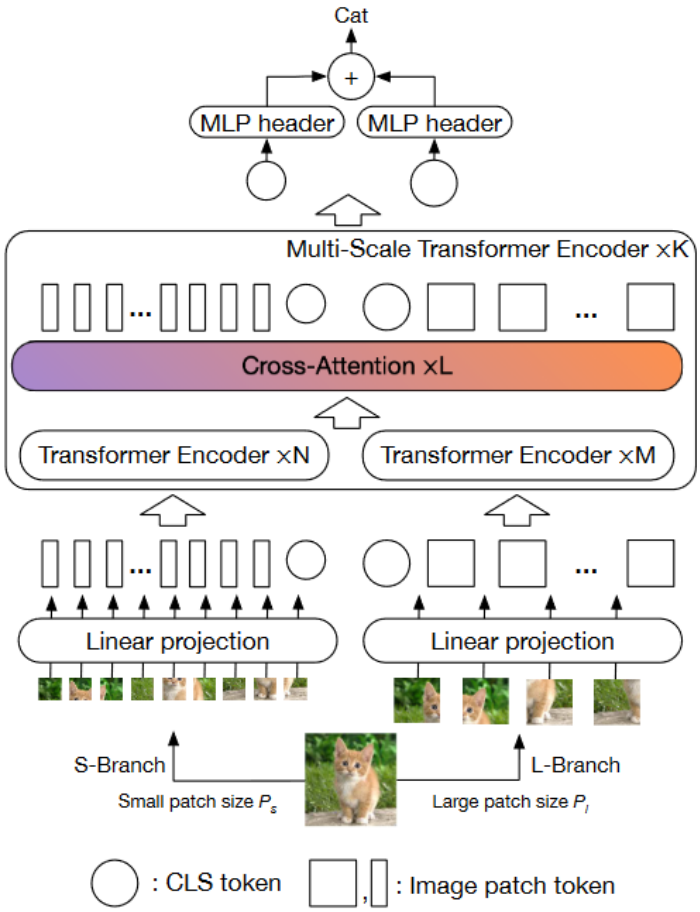


Figure 15: CrossViT. A dual branch architecture containing vision transformer for small and large patches [10]

4.2.3 Vanilla VAE

We have used a vanilla VAE for doing one-class classification [59]. During training, we use only correct (non-defective) samples and try to learn their representation. During inference time, we pass both defective and non-defective samples and try to discriminate them using reconstruction loss.

It should be noted that VAE training uses two loss terms. One is MSE Loss between input and reconstructed input and the other is Kullback–Leibler divergence loss [60] which calculates how close our learned distribution is to the actual distribution.

Our implementation (including code) of vanilla VAE was bootstrapped from Pytorch-VAE [61] project on GitHub. The architecture of VAE has already been shown in Figure 7. In the encoder block, we used 6 convolution layers with channels dimensions 32, 64, 128, 256, 512, 1024. Each convolution layer is followed by a batch normalization layer and a leaky ReLU layer. Two fully connected layers are used to compute the mean and variance from the bottleneck layer in latent space. Our latent space was 256 dimensions. The decoder block begins with mapping latent space back to the bottleneck layer representation. Then we have six upsampling layers where each layer consists of transposed convolutions for upsampling followed by batch normalization and leaky ReLU. The last layer is the convolution layer that maps 32

channels to 3 channels as we have in the input. More details about experiments and results are given in section 4.3.

4.3 Experiments and Results

Training deep neural networks is an iterative process and generally requires a lot of experiments with different hyperparameters. Different architectures have different hyperparameters but the learning rate [62], and choice of optimizer are generally shared among all deep neural network architectures.

4.3.1 Hyperparameters Optimization

Hyperparameter search is one of the most cumbersome tasks in machine learning projects [58]. It focuses on finding optimal values for hyperparameters from their search space. Hyperparameter tuning techniques include random search, grid search, and Bayesian [63]. Grid search experiments with all possible combinations of hyperparameters and select the best ones. Random search randomly samples different values for hyperparameters from their search space instead of doing an exhaustive search. But both grid search and random search are inefficient in the sense that they are uninformed from previous experiment results and spend a lot of time experimenting with bad values. In Bayesian optimization search, we keep track of previous experiments and their results and use this information for selecting values for the next experiment.

We used Optuna [58] for our experiments which provides an easy-to-use framework written in Python. Optuna also provides very nice dashboards for better visualization of experiments.

We have already discussed the model architectures in section 4.2. Now we discuss the hyperparameters and using Optuna to find their optimal values. For optimization, We used AdamW [64] optimizer. AdamW is the correct implementation of Adam optimizer in pytorch. We selected Adam because it is an adaptive gradient algorithm that adjust learning rate for each gradient coordinate according to the current geometry curvature of the objective function and achieve much faster convergence speed than a vanilla SGD in practice [65]. Below are the ranges of values for various hyperparameters that we used to explore and find the optimal values using Bayesian search with Optuna:

- Initial learning rate: 1e-3 to 1e-5
- Batch size: [16, 32, 64, 128]
- Optuna trials: 100
- Optuna criterion: minimize validation loss

Table 5 lists best-performing values of different hyperparameters on the validation set during DenseNet121 experiments.

For CrossViT experiments, we were again planning to use Adam[64] optimizer but the original implementation of CrossViT paper [10] use Adam for pretraining and SGD for fine-tuning on downstream tasks [10]. We decided to use SGD as we were training CrossViT on very small dataset. Here, We also want to highlight that while Adam [64] typically leads to faster convergence, it often exhibits poorer generalization performance compared to SGD [65]. In

our CrossViT experiments, we exclusively utilized SGD, although Adam could potentially yield similar results. Below are the ranges of values for various hyperparameters that we used to explore and find the optimal values using grid search:

- Initial learning rate: 1e-1 to 1e-6
- Batch size: [16, 32, 64, 128]
- Criterion: minimize the validation loss

Table 6 lists different hyperparameters and their best-performing values for CrossViT experiments.

For VAE experiments, hyperparameter optimization was not performed by us as we decided to use the default values provided in Pytorch-VAE [61] implementation. Table 7 lists down the different hyperparameter values used during reconstruction experiments using vanilla VAE.

Table 5: Different hyperparameters and their best-performing values during our binary classification experiments using densenet121. (*) indicates that only a single option was used for all experiments

Problem	Batch Size	Initial lr	Optimizer*	Epochs	Weight Decay*
Screw	32	1e-4	AdamW	10	0.05
Cable	32	1e-4	AdamW	30	0.05
Cover	32	1e-4	AdamW	40	0.05
Sheet Metal Package	32	1e-4	AdamW	30	0.05
Winding Head	32	1e-6	AdamW	30	0.05

Table 6: Different hyperparameters and their best-performing values during our binary classification experiments using CrossViT. (*) indicates that only a single option was used for all experiments

Problem	Batch Size	Initial lr	Optimizer*	Epochs	Momentum*
Screw	64	1e-3	SGD	30	0.9
Cable	32	1e-2	SGD	40	0.9
Cover	64	1e-4	SGD	30	0.9
Sheet Metal Package	64	1e-4	SGD	30	0.9
Winding Head	64	1e-2	SGD	40	0.9

4.3.2 Evaluation Metrics

We decided not to use accuracy as an evaluation metric due to the possibility of getting high scores even if all samples are predicted as non-defective. Instead, we decided to use the F1

Table 7: Different hyperparameter and their values during our reconstruction experiments using vanilla VAE. (*) indicates that only single option was used for all experiments

	Batch Size*	Initial lr*	Optimizer*	Epochs*	Weight Decay*	Latent Dim*
Same for all	64	0.005	Adam	100	0.0001	256

score, which considers both false positives and false negatives. In our scenario, prioritizing minimal false negatives (better recall) outweighs a slight increase in false positives. Therefore, achieving a balance between high recall and precision emerges as the main evaluation metric for our use case.

4.3.3 Results and Discussion

Tables 8 and 9 list down the binary classification results for all five defects obtained from DenseNet121 and CrossViT respectively. Confusion Matrices are also provided in the appendix B. Observations from our experiments reveal that DenseNet121 surpasses CrossViT in performance. Both methods demonstrate flawless performance for screws, largely due to the simplicity of binary classification for screws and the availability of more defective screw samples in our training set compared to other defective components. However, as the number of defective components in the training set decreases, the performance of CrossViT significantly declines. This suggests that CrossViT requires a greater number of training data compared to DenseNet for achieving better generalization. The necessity for additional data can also be attributed to the greater model capacity of CrossViT, which has 24 million parameters [10], compared to DenseNet121's 7 million parameters [3]. When considering defects other than screws, DenseNet121 demonstrates strong performance in cover classification, achieving perfect recall. However, the classification of broken cables, gaps in sheet metal packages, and broken bandaging threads on winding heads still require enhancement. Detecting gaps in sheet metal packages proves particularly challenging, as these gaps can be extremely small, occasionally leading human observers to mistake them for non-defective samples. The poorest performance is observed in the classification of broken bandaging threads on the winding head, primarily because our training set contained only 66 defective samples of this type.

Table 8: Binary classification results on test set using DenseNet121

Problem	Input Type	Precision	Recall	F1
Screw	RGB	1.0	1.0	1.0
Cover	Grayscale	0.86	1.0	0.92
Sheet Metal Package	Grayscale	0.61	0.66	0.63
Cable	Grayscale	0.80	0.60	0.69
Winding Head	RGB	0.19	0.51	0.28

The results for anomaly detection, employing VAE-reconstructed samples, are depicted in Figure 16. The reconstruction error, measured as binary cross entropy (BCE), is computed be-

Table 9: Binary classification results on test set using CrossViT

Problem	Input Type	Precision	Recall	F1
Screw	RGB	1.0	1.0	1.0
Cover	RGB	0.38	0.58	0.46
Sheet Metal Package	RGB	0.32	0.64	0.43
Cable	RGB	0.07	0.46	0.12
Winding Head	RGB	0.17	0.53	0.26

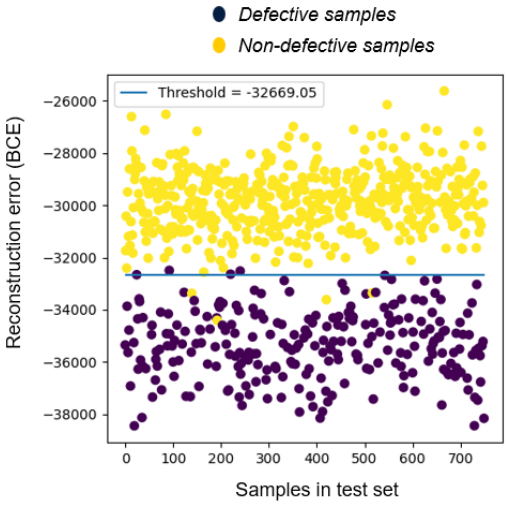
tween input samples and their reconstructed counterparts. Notably, only screws can be accurately classified by appropriately selecting a threshold for the reconstruction error. In the case of the remaining four defects, the reconstruction errors for both defective and non-defective samples fall within a similar range, making it challenging to differentiate between them solely based on the thresholding of the reconstruction error.

4.3.4 Comparison with Results of Alexander Christoph

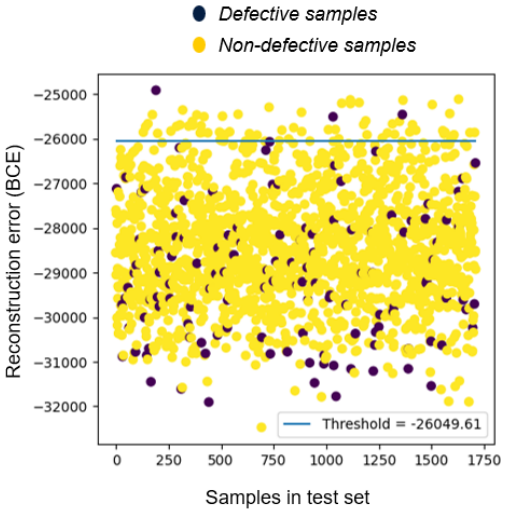
Table 10 compares the results of Alexander Christoph's EfficientNetV2-S [8] with those produced by our DenseNet121 model. While screw, cover, and sheet metal package classification exhibit similar performance, our model demonstrates superior performance in classifying broken cables and broken bandaging threads on winding heads. It's worth noting that achieving better recall with good precision serves as the appropriate evaluation metric in our scenario as we can tolerate a slight increase in false positives.

Table 10: Binary classification results of Alexander Christoph [8] using EfficientNetV2-S. The results are compared with those obtained from our DenseNet121 model. Red signifies inferior performance compared to ours, green indicates superior performance, and black denotes the same performance.

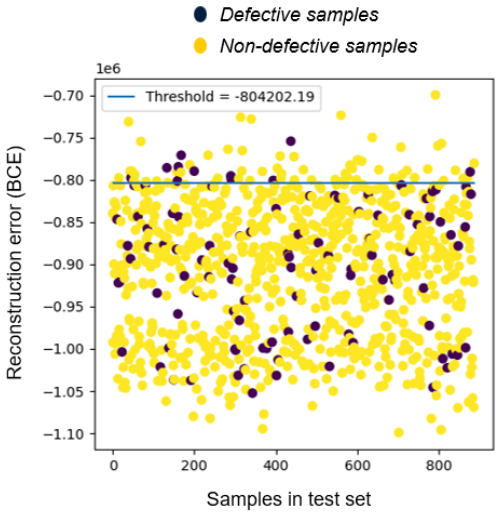
Problem	Precision	Recall	F1
Screw	1.0	1.0	1.0
Cover	0.94	0.97	0.95
Sheet Metal Package	0.47	0.74	0.57
Cable	0.69	0.35	0.46
Winding Head	0.30	0.27	0.28



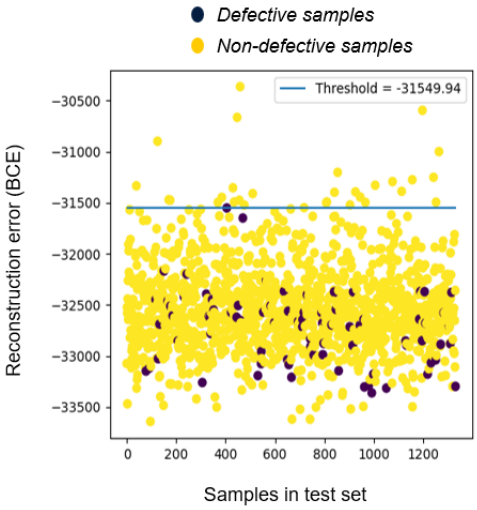
a) Screw



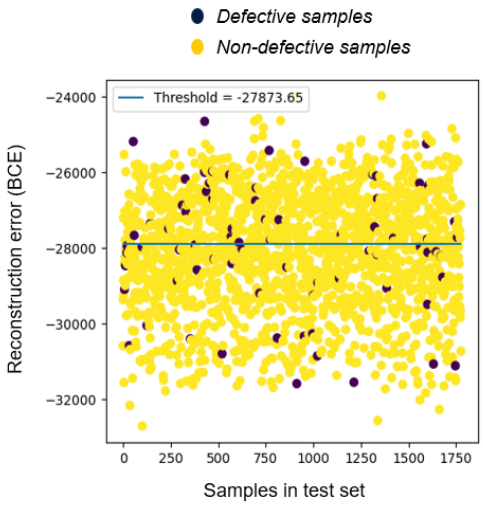
b) Insulation cover



c) Sheet metal package



d) Bandaging threads on winding head



e) Cable

Figure 16: Reconstruction error between input samples and VAE-reconstructed samples for five defects on test set samples (minus sign can be ignored)

5 Conclusion and Future Outlook

AVI tasks frequently encounter the problem of limited data availability. Recently, DL methods have been proposed for performing surface defect classification within the context of AVI. When comparing the performance of these methods on a proprietary electric motor dataset, CNNs surpass ViTs in tasks with limited data, despite the recent advancements in ViTs. Moreover, the deployment of ViTs presents difficulties due to their large model sizes, potentially hampering their practical implementation in production environments. While VAE-based anomaly detection holds promise, determining an appropriate threshold for reconstruction error remains a persistent challenge. Therefore, prioritizing the acquisition of high-quality data emerges as a crucial consideration, overshadowing extensive architectural exploration and emphasizing the indispensable role of data quality in model performance and generalization.

In the future, DL methods are expected to experience growing adoption within the AVI domain. However, there are many avenues of improvement as well. Firstly, leveraging self-supervised pretraining holds the potential for better performance for ViTs, offering a way of suitable performance for tasks with limited data. Additionally, knowledge distillation techniques present an opportunity to train smaller models that are more suitable for deployment in practical environments. Furthermore, generative models present an innovative approach to synthetic data creation, offering a means to augment datasets and enhance model robustness. These strategies collectively represent key areas of exploration that can significantly advance the efficacy and scalability of DL methods in AVI systems.

NOTE: Some of the sentences in the conclusion and future outlook were rephrased using ChatGPT from my original sentences for better wording. Citations are deliberately skipped here. A few sentences (5-10) in previous parts are also rephrased but overall more than 98% of this report has been written by a human (author) with proper citations. Grammarly was also integrated with the LaTeX environment during the writing of this report.

Bibliography

[1] X. Zheng, S. Zheng, Y. Kong, and J. Chen, “Recent advances in surface defect inspection of industrial products using deep learning techniques,” *The International Journal of Advanced Manufacturing Technology*, vol. 113, pp. 35–58, Mar. 2021.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” *arXiv preprint arXiv:1608.06993*, 2018.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *arXiv preprint arXiv:1706.03762*, 2023.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2021.

[6] D. Addo, S. Zhou, J. K. Jackson, G. U. Nneji, H. N. Monday, K. Sarpong, R. A. Patamia, F. Ekong, and C. A. Owusu-Agyei, “Evae-net: An ensemble variational autoencoder deep learning network for covid-19 classification based on chest x-ray images,” *Diagnostics*, vol. 12, no. 11, 2022.

[7] A. Mayr, J. Seefried, M. Ziegler, M. Masuch, A. Mahr, J. v. Lindenfels, M. Meiners, D. Kißkalt, M. Metzner, and J. Franke, “Machine learning in electric motor production - potentials, challenges and exemplary applications,” in *2019 9th International Electric Drives Production Conference (EDPC)*, pp. 1–10, 2019.

[8] A. Christoph, “Hybrid artificial intelligence in industrial production using the example of visual inspection of spatial assembly units,” Master’s thesis, Friedrich-Alexander University Erlangen-Nuremberg, 2023.

[9] Y.-D. Zhang, S. C. Satapathy, X. Zhang, and S.-H. Wang, “COVID-19 diagnosis via DenseNet and optimization of transfer learning setting,” *Cognitive Computation*, Jan. 2021.

[10] C.-F. Chen, Q. Fan, and R. Panda, “Crossvit: Cross-attention multi-scale vision transformer for image classification,” *arXiv preprint arXiv:2103.14899*, 2021.

[11] T. S. Newman and A. K. Jain, “A survey of automated visual inspection,” *Computer Vision and Image Understanding*, vol. 61, no. 2, pp. 231–262, 1995.

[12] R. T. Chin and C. A. Harlow, “Automated visual inspection: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 6, pp. 557–573, 1982.

[13] S. Arikan, K. Varanasi, and D. Stricker, “Surface defect classification in real-time using convolutional neural networks,” *arXiv preprint arXiv:1904.04671*, 2019.

[14] Y. Chen, Y. Ding, F. Zhao, E. Zhang, Z. Wu, and L. Shao, “Surface defect detection methods for industrial products: A review,” *Appl. Sci. (Basel)*, vol. 11, p. 7657, Aug. 2021.

[15] D. Martin, S. Heinzel, J. K. von Bischhoffshausen, and N. Kühl, “Deep learning strategies for industrial surface defect detection systems,” *ArXiv*, vol. abs/2109.11304, 2021.

[16] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer Cham, 2018.

[17] J. Deng, R. Socher, L. Fei-Fei, W. Dong, K. Li, and L.-J. Li, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, vol. 00, pp. 248–255, 06 2009.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *arXiv preprint arXiv:1706.03762*, 2023.

[20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2021.

[21] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” *arXiv preprint arXiv:2003.05991*, 2021.

[22] O. Rippel, P. Mertens, and D. Merhof, “Modeling the distribution of normal data in pre-trained deep features for anomaly detection,” in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 6726–6733, IEEE, 2021.

[23] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2022.

[24] Y. Lu, “The current status and developing trends of industry 4.0: a review,” *Information Systems Frontiers*, Nov. 2021.

[25] M. Babic, M. A. Farahani, and T. Wuest, “Image based quality inspection in smart manufacturing systems: A literature review,” *Procedia CIRP*, vol. 103, pp. 262–267, 2021. 9th CIRP Global Web Conference – Sustainable, resilient, and agile manufacturing and service operations : Lessons from COVID-19.

[26] Y. Liu, C. Zhang, and X. Dong, “A survey of real-time surface defect inspection methods based on deep learning,” *Artif. Intell. Rev.*, vol. 56, p. 12131–12170, apr 2023.

[27] L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, and H. Harmouch, “The effects of data quality on machine learning performance,” *arXiv preprint arXiv:2207.14529*, 2022.

[28] F. Rosenblatt, “The perceptron: A perceiving and recognizing automaton,” Tech. Rep. 85-60-1, Cornell Aeronautical Laboratory, Buffalo, New York, 1957.

[29] M. L. Minsky and S. Papert, *Perceptrons, An Essay on Computational Geometry*. Cambridge, MA: MIT Press, 1969.

[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[31] Y. Chen, M. Mancini, X. Zhu, and Z. Akata, "Semi-supervised and unsupervised deep visual learning: A survey," *arXiv preprint arXiv:2208.11296*, 2022.

[32] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. `http://www.deeplearningbook.org`.

[33] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," *arXiv preprint arXiv:1511.06068*, 2015.

[34] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2015.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[37] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2017.

[38] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2020.

[39] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," *arXiv preprint arXiv:2112.10752*, 2022.

[40] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *arXiv preprint arXiv:2103.14030*, 2021.

[41] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, (New York, NY, USA), p. 665–674, Association for Computing Machinery, 2017.

[42] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[43] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, (New York, NY, USA), p. 1096–1103, Association for Computing Machinery, 2008.

[44] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, (Madison, WI, USA), p. 833–840, Omnipress, 2011.

[45] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Journal of the American Statistical Association*, vol. 112, p. 859–877, Apr. 2017.

[46] L. Song, X. Li, Y. Yang, X. Zhu, Q. Guo, and H. Yang, "Detection of micro-defects on metal screw surfaces based on deep convolutional neural networks," *Sensors (Basel)*, vol. 18, p. 3709, Oct. 2018.

[47] G. Fu, P. Sun, W. Zhu, J. Yang, Y. Cao, M. Y. Yang, and Y. Cao, "A deep-learning-based approach for fast and robust steel surface defects classification," *Optics and Lasers in Engineering*, vol. 121, pp. 397–405, 2019.

[48] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[49] M. Farnsworth, D. Tiwari, Z. Zhang, G. W. Jewell, and A. Tiwari, "Augmented classification for electrical coil winding defects," *The International Journal of Advanced Manufacturing Technology*, vol. 119, pp. 6949–6965, Apr. 2022.

[50] N. P. Avdelidis, A. Tsourdos, P. Lafiosca, R. Plaster, A. Plaster, and M. Droznika, "Defects recognition algorithm development from visual uav inspections," *Sensors*, vol. 22, no. 13, 2022.

[51] N. Hütten, R. Meyes, and T. Meisen, "Vision transformer in industrial visual inspection," *Applied Sciences*, vol. 12, no. 23, 2022.

[52] O. Rippel and D. Merhof, "Anomaly detection for automated visual inspection: A review," in *Bildverarbeitung in der Automation* (V. Lohweg, ed.), (Berlin, Heidelberg), pp. 1–13, Springer Berlin Heidelberg, 2023.

[53] J. Yu, X. Zheng, and J. Liu, "Stacked convolutional sparse denoising auto-encoder for identification of defect patterns in semiconductor wafer map," *Computers in Industry*, vol. 109, pp. 121–133, 2019.

[54] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[56] A. B. Jung, K. Wada, J. Crall, S. Tanaka, J. Graving, C. Reinders, S. Yadav, J. Banerjee, G. Vecsei, A. Kraft, Z. Rui, J. Borovec, C. Vallentin, S. Zhydenko, K. Pfeiffer, B. Cook, I. Fernández, F.-M. De Rainville, C.-H. Weng, A. Ayala-Acevedo, R. Meudec, M. Laporte, *et al.*, "imgaug." `https://github.com/aleju/imgaug`, 2020. Online; accessed 01-Feb-2020.

[57] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *arXiv preprint arXiv:1708.02002*, 2018.

[58] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," *arXiv preprint arXiv:1907.10902*, 2019.

[59] P. Perera, P. Oza, and V. M. Patel, "One-class classification: A survey," *arXiv preprint arXiv:2101.03064*, 2021.

[60] A. G. de G. Matthews, J. Hensman, R. E. Turner, and Z. Ghahramani, "On sparse variational methods and the kullback-leibler divergence between stochastic processes," in *International Conference on Artificial Intelligence and Statistics*, 2015.

[61] A. Subramanian, "Pytorch-vae." `https://github.com/AntixK/PyTorch-VAE`, 2020.

[62] Y. Wu, L. Liu, J. Bae, K.-H. Chow, A. Iyengar, C. Pu, W. Wei, L. Yu, and Q. Zhang, "Demystifying learning rate policies for high accuracy training of deep neural networks," in *2019 IEEE International conference on big data (Big Data)*, pp. 1971–1980, IEEE, 2019.

[63] E. Elgeldawi, A. Sayed, A. R. Galal, and A. M. Zaki, "Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis," *Informatics*, vol. 8, no. 4, 2021.

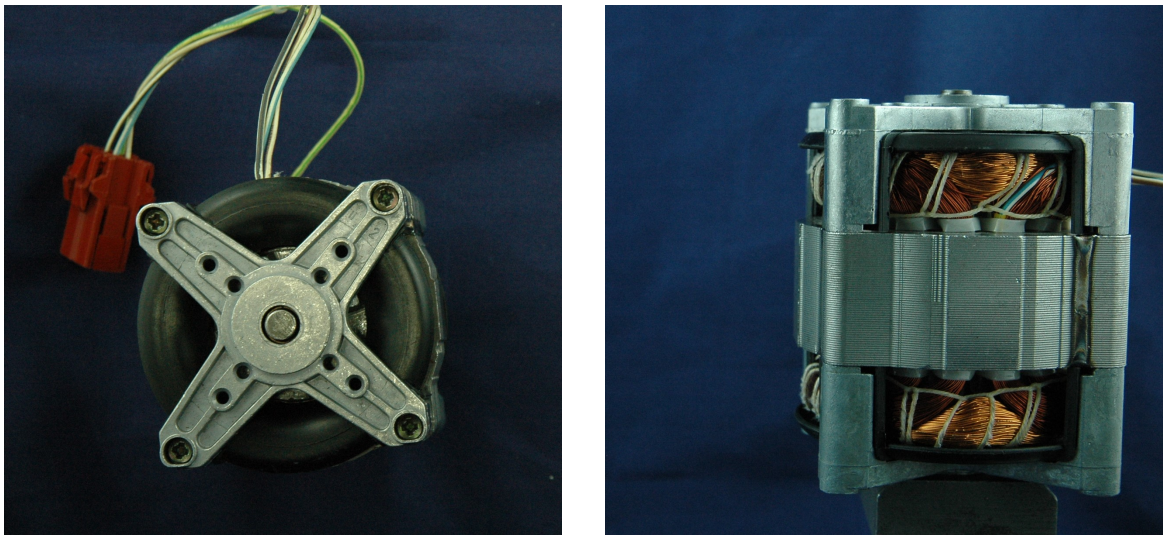
[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[65] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi, *et al.*, "Towards theoretically understanding why sgd generalizes better than adam in deep learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21285–21296, 2020.

A Appendix: Multi-view Learning Experiments on FAPS Old Electric Motor Dataset

In the context of this project, the author was working initially on an old FAPS multi-view electric motor dataset. That dataset contains two views (side and top) of the same motor. There are 13 surface defects but some of the defects are only visible from a single view. The task was multi-view learning. We take both views and pass them through a feature extractor. These features are fused (concatenated) together in a late-fusion manner and then passed through a classifier to find the correct class. The reason for not proceeding further with the dataset was that the top view and side for each motor were not properly linked. There was a chance of linking the side view of one motor to the top view of the other motor causing data leaks into the validation and test split.

Figure A.17 shows top and side views of an exemplary middle motor while Table A.17 explains the nature of defects in a 14-class classification problem. There are nearly 35 samples of each class from each view making 1006 top and side view pairs in total.



a) Top-view of a middle motor *b) Side-view of a middle motor*

Figure A.17: Top and side views of an exemplary middle motor from old electric motor dataset

Here, we again used DenseNet121 as a feature extractor. The training process is similar to the one explained in this report earlier for binary classification using DenseNet121 except we have 14 classes here and the classifier here adds three FC layers rather than one. In this dataset, we created the train, validation, and test split. There are 643 sample pairs in the training set, 161 in the validation set, and 201 in the test set. Our DenseNet121 with late-fusion achieved 97% overall accuracy with just 6 misclassifications in the test set. Results for 14-class classification has been written in Table B.22.

Table A.11: Explanation of 14 classes in the dataset (13 defects and one class for correct assembly)

Class	Explanation	Class	Explanation
C	Correct assembly. No defect.	MC_2	Both top and bottom covers are missing
MC_O	Top cover is missing	MC_U	Bottom cover is missing
MS_1	One screw is missing	MS_2I	Two adjacent screws are missing
MS_2X	Two diagonal screws are missing	MS_3	Three screws are missing
MS_4	Four screws are missing	NS_1	One screw is not properly screwed
NS_2I	Two adj. screws are not properly screwed	NS_2X	Two diag. screws are not properly screwed
NS_3	Three screws are not properly screwed	NS_4	Four screws are not properly screwed

Table A.12: Multiclass classification results for 14 classes in the dataset (13 defects and one class for correct assembly)

Class	Precision	Recall	F1
C	1.0	1.0	1.0
MC_2	1.0	1.0	1.0
MC_O	1.0	1.0	1.0
MC_U	1.0	1.0	1.0
MS_1	1.0	1.0	1.0
MS_2I	1.0	1.0	1.0
MS_2X	1.0	0.90	0.95
MS_3	0.95	1.0	0.97
MS_4	1.0	1.0	1.0
NS_1	0.93	1.0	0.97
NS_2I	0.81	1.0	0.89
NS_2X	0.93	0.93	0.93
NS_3	1.0	0.74	0.85
NS_4	1.0	1.0	1.0

B Confusion Matrices for DenseNet121 and CrossViT Results

B.1 DenseNet121 Confusion Matrices

Table B.13: DenseNet121: Screw confusion matrix

Actual defected	252	0
Actual non-defected	0	498
	Predicted defected	Predicted Non-defected

Table B.14: DenseNet121: Cover confusion matrix

Actual defected	126	0
Actual non-defected	21	1563
	Predicted defected	Predicted Non-defected

Table B.15: DenseNet121: Sheet metal package confusion matrix

Actual defected	71	37
Actual non-defected	45	735
	Predicted defected	Predicted Non-defected

Table B.16: DenseNet121: Cable confusion matrix

Actual defected	49	32
Actual non-defected	12	1239
	Predicted defected	Predicted Non-defected

Table B.17: DenseNet121: Winding head confusion matrix

Actual defected	38	37
Actual non-defected	163	1538
	Predicted defected	Predicted Non-defected

B.2 CrossViT Confusion Matrices

Table B.18: CrossViT: Screw confusion matrix

Actual defected	252	0
Actual non-defected	0	498
	Predicted defected	Predicted Non-defected

Table B.19: CrossViT: Cover confusion matrix

Actual defected	72	52
Actual non-defected	116	1456
	Predicted defected	Predicted Non-defected

Table B.20: CrossViT: Sheet metal package confusion matrix

Actual defected	68	39
Actual non-defected	142	631
	Predicted defected	Predicted Non-defected

Table B.21: CrossViT: Cable confusion matrix

Actual defected	37	44
Actual non-defected	509	738
	Predicted defected	Predicted Non-defected

Table B.22: CrossViT: Winding head confusion matrix

Actual defected	40	35
Actual non-defected	197	1504
	Predicted defected	Predicted Non-defected