Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Analysis of the Scalability of Siamese Neural Network for Performing Quality Inspection of the Welding Nuts

Priyadarshini Krishna Shobha

Course of Study:	Information Technology
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Prof. Dr. Marco Aiello, Daniel Díez Álvarez
Commenced:	November 8, 2021
Completed:	May 8, 2022

Acknowledgement

I would like to thank my supervisors Prof. Dr. Marco Aiello and Daniel Díez Álvarez, my colleague Felix Euteneuer and the entire team of Dr. -Ing. Matthias Reichenbach at Mercedes-Benz AG for all of the insightful discussions throughout the course of writing this thesis and their continued support which has greatly motivated me while working on this project.

Abstract

Fabrication is a common process in the manufacturing industry. It is important to inspect the nuts before welding in order to maintain product quality and avoid the need for rework. Nowadays, artificial intelligence based visual validation have demonstrated that machines can efficiently perform quality inspection. This thesis provides a siamese neural network based solution to identify incorrectly placed nuts before welding process. The network classifies incorrect nuts, missing nuts and flipped nuts as invalid cases. The network is designed in such a way that the model trained to perform quality analysis of one task can be scaled to a new task by using six or ten training images and one minute of training duration. The decision to use siamese network is made because of its ability to learn from semantic similarity. To validate the adaptability of the solution, we have focused on three use cases in two different environments. First case consists of test nuts which are different in size, shape and position. The second use case validates the model on different orientation of test nuts. In third case, the test nuts are captured in different environment conditions. We have proposed two different approaches, one with custom convolutional neural network and the other with EffecientNet-B0 as feature extractor. The model with custom convolutional feature extractor has better performance and scales to all the three use cases. The model with EfficientNet-B0 as feature extractor has exceptional performance in third use case which consists of data captured in factory environment. We evaluate the proposed solutions by recording the accuracy and confusion matrix of different use cases and architectures. However this approach is limited in terms of production efficiency and needs more validation in factory lighting conditions.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

Quality inspection plays a vital role in manufacturing industries. It ensures the products are defect free and meet the company requirements. Improper quality analysis will harm the customer safety and lead to huge financial compensations. Takata airbag recall due to defective airbags is one of the biggest recall in automotive industries which resulted in recall of more than 55 million vehicles around the world [3].

Production system involves welding process to join different materials. Welding of screws and nuts to body parts is a widely used process. There are wide range of approaches which inspects body parts before fabrication process. The manual inspection approach where people carefully assess the quality of each part is time consuming. Computer vision is an alternate approach where machines can inspect large quantity of products reliably and repeatedly. On the other hand, the conventional computer vision (CV) approaches uses feature descriptors like speeded up robust features (SURF) and scale-invariant feature transform (SIFT) combined with machine learning algorithms like support vector machines and and k-nearest neighbors for image classification. The developed CV algorithms requires expert analysis, high fine tuning and are domain specific [33]. The quality analysis performed on the image captured by sensors and processed using OpenCV libraries are specific to image characteristics like size, shape and colour [2].

Artificial intelligence (AI) has mobilized several industries, including the automotive sector. It is one of the key components of the visionary concept of Industry 4.0 [48]. In particular, deep learning has outperformed several state-of-the-art techniques in the field of computer vision. AI combined with machine vision can build applications which checks the quality to the finest details. Deep learning is well suited to handle complex patterns which vary in subtle but tolerable ways [21].

Several convolutional neural network architectures have been developed to solve real world problems. These standard architectures like EffecientNet-B7 trained on ImageNet [37] which attains top-1 accuracy of 84.3% with 66M trainable parameters [47] cannot be directly used to perform quality analysis on industrial applications due to variation in the datasets. Retraining of the pretrained network requires large amount of data. The developed neural networks are highly case specific and are not scalable to new applications.

Manufacturing industries have several processes which needs to be automated in order to increase the production efficiency. Whenever a new quality inspection problem has to be solved, the whole process of collecting and preprocessing the image data and training the neural network has to be repeated. This promotes the need for an engineer to be always present in the production loop.

1.2 Contribution

As mentioned in the previous section, provided the availability of a large amount of data, traditional deep neural networks attain good performance in classification tasks. However, with the increase in the number of classes in classification or the quality inspection of different use cases, the networks need redesigning, retraining and fails to adapt. The authors in [30] and [23] have proposed solutions to perform quality analysis in industrial applications using siamese twin networks. But these solutions does not concentrate on adapting the developed model to different inspection tasks. In this thesis, we propose a siamese neural network that addresses the problems of conventional deep learning networks and scales to unseen data with less effort and time.

The proposed model maps the images to embedding space and is trained to learn the semantic difference and similarity between the image pairs instead of the image characteristics required to recognize the image. In embedding space, similar images are mapped close to one another whereas dissimilar images are mapped far apart. Therefore, distance metrics which calculates the distance between the images is used to detect if the given images are similar or not. This behaviour helps the model to classify the images of unseen distribution. The classifier network is independent of the number of classes and hence the network needs no redesigning with increase in the prediction classes. The network detect flipped nuts, incorrect nuts and missing nuts as invalid nuts so that required changes can be done before welding. A baseline model is trained to perform quality analysis of a particular part in the industry. When a new quality inspection task needs to be solved, this model is loaded and retrained with 6 or 10 images of new use case with training duration of approximately one minute. The scalability of the solution is tested on its ability to generalize to nuts with varying shape, size, position, orientation and also the nuts in different environmental conditions. A detailed analysis is performed regarding the number of images and training duration required to adapt the model to new use case. Number of categories of nuts needed in training in order to test the model on a complete different nut is also evaluated. The model is designed so that a factory worker could use the developed system to switch the analysis to different task without the involvement of an engineer in the process.

1.3 Thesis Outline

The thesis structure consists of seven chapters which are organised as follows:

Chapter 2 provides brief introduction to fundamental concepts of deep learning starting with neural networks, learning algorithm, regularization techniques, architecture of convolutional neural networks and twin networks called siamese neural networks.

Chapter 3 discusses the existing state-of-the-art technologies and methodologies in similarity learning and their applications in various domains. Detailed numerical explanation is also provided for two papers which apply siamese neural network to perform industrial quality inspection.

Chapter 4 provides the architectural details of the proposed solution. The focus is on two main architectures with different feature extractors. A step by step procedure of the scalability approach is highlighted along with the choice of dataset to prove the hypothesis.

Chapter 5 illustrates the collection of image data and it's preprocessing. Experiments on two main architectures for the selected use cases with learning curves are also included.

Chapter 6 numerical proof for the suggested approaches are provided in this section. Accuracy of the proposed architectures along with confusion matrix are recorded.

Chapter 7 concludes the result of the thesis by indicating the potential future works and limitations of the project.

2 Background

In the field of computer science, artificial intelligence is defined as the science and engineering of making intelligent machines [27]. AI is being adapted in different sectors such as health care, finance, automotive industry, gaming and many more. Machine learning, natural language processing, computer vision, robotics, and other areas of AI have gained a lot of attention recently. This chapter briefly introduces the fundamental concepts of AI discussed in the thesis.

2.1 Machine Learning

Machine learning is a branch of artificial intelligence that analyses the patterns in the underlying data and performs predictive analysis. It imitates humans' ability to learn and improve their performance gradually. Machine learning is specifically important for understanding structured data where human intelligence fails to recognize the pattern. Depending on whether the machine learns with the knowledge of ground truth or not, the machine learning algorithms can be categorized as follows:

- **Supervised Learning:** In this method, each sample of training data is annotated with the expected solution, called ground truth. The algorithms will learn to map the input to output labels. Supervised learning can be further classified as:
  - **Classification:** Given the input with n features, model classifies it to one of the m classes i.e. $f : \mathbb{R}^n \rightarrow \{1, .., m\}$. For example, classifying a message as ham or spam.
  - **Regression:** The model will predict a numerical value given the input, thus mapping n dimensional feature vector to a single value i.e. $f : \mathbb{R}^n \rightarrow \mathbb{R}$. For example, predicting the price of a car given it's features.
- **Unsupervised Learning:** The data used for training is unlabeled. The model learns the hidden intrinsic patterns in the input data and performs decision making without any ground truth of the data samples.
- **Reinforcement Learning:** The learners, called agents, will observe the environment and perform actions for which they are given a positive or negative reward. It is similar to children learning things through observations. The agents must learn the best strategy, called policy, to maximize the reward.

2.2 Deep Learning

Deep learning is a sub section of machine learning which teaches the computer what comes intuitively to human beings. For years computers have outperformed human beings in tasks which are structured and can be represented by mathematical rules. Whereas human beings are more efficient in unstructured data like recognizing the voice, object identification etc. Given a set of features like location, height and area a machine learning model can predict the given mountain. But for example with images it is extremely complex to extract the features, the image might vary because of the lighting conditions, shadow and many other factors. With the advent of deep learning, machines are able to intelligently learn complex features by combining simple features, like recognizing the house in an image by learning edges and contours.

2.2.1 Artificial Neural Networks

The human brain is made up of billions of neurons which transmit signals from one part of the body to another using biochemical reactions. Inspired by this, Artificial Neural Networks (ANN) have been developed whose basic processing elements are called neurons. ANNs provide mathematical modelling of the human nervous system. Neural networks can be multiple layered whose architecture include input, hidden and output layers as shown in Figure 2.1. The goal of the neural network is to learn a mapping function f from input x to output y which can a be a classifier or a regressor. The neural network learns the parameter θ defined by the forward function $y = f(x;\theta)$ to best approximate the output [12]. The linear models are extended to non linear functions of x by activation functions. The detailed mathematical modelling of a neuron is depicted in Figure 2.2 .

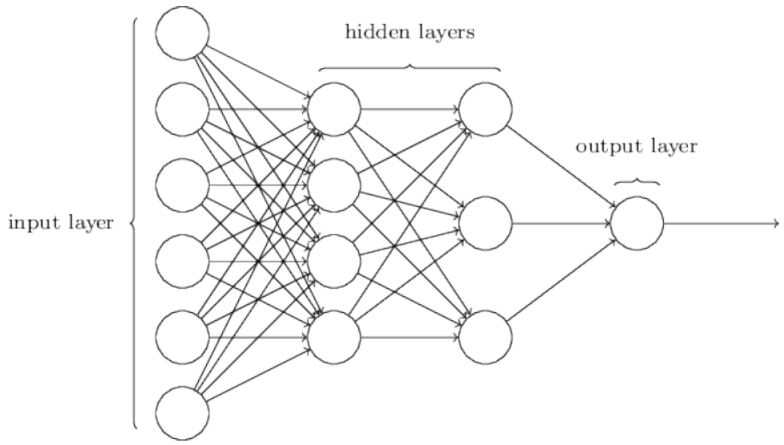


Figure 2.1: Neural network with a input layer, two hidden layers and a output layer [31].

As shown in the Figure 2.2 the inputs x_i from all the neurons in the previous layer will be multiplied with weight parameters $\sum_i w_i x_i$ and a bias term b will be added, which is then passed through the activation function and it is propagated to other nodes. During forward pass, this input is passed through several nodes of multiple hidden layers and arrives at the final layer. The prediction $\hat{y}$ made by the network is the approximation of the given ground truth y. The difference between the prediction and ground truth is called the loss function and is as represented in the equation

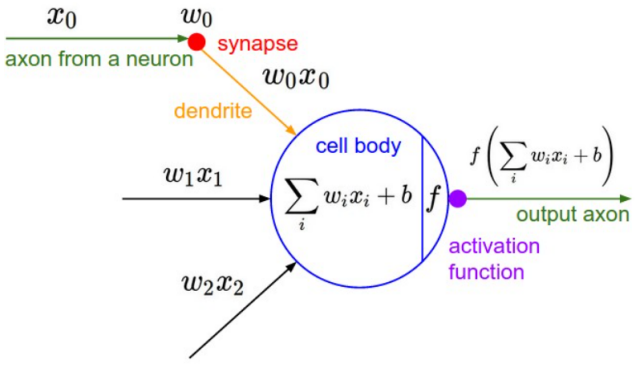


Figure 2.2: Mathematical model of a single neuron. The dot product is computed between weights and inputs to which a bias term is added and is followed by a non-linear activation function [41].

2.1. Higher loss values indicates erroneous model prediction, therefore the model will be trained to reduce the loss function and make the prediction as close to the ground truth as possible. The average loss of all the training samples is called cost function and is given by the equation 2.2. The gradient of cost function is calculated with respect to learnable parameters and the parameters are updated from output layer to input layer. This process is called gradient decent and backpropagation (explained in detailed in next section). The model is trained to learn optimized set of parameters and to reduce the cost function to zero, so that the prediction is close to ground truth (equation 2.3).

$$L(f(x_i;\theta), y_i) = ||f(x_i;\theta) - y_i||^2 \tag{2.1}$$

$$J(\theta) = \frac{1}{N}\sum_{i}^{N} ||\hat{y}_i - y_i||^2 \tag{2.2}$$

$$\nabla J = 0 \tag{2.3}$$

2.2.2 Activation Functions

As explained in the previous section, neurons calculate the weighted sum of the inputs and will also add a bias term. But the neurons do not have the ability to choose a firing pattern [32]. The activation function decides if the neuron needs to be activated or not and introduces non-linearity in the network, thus enabling the neural network to learn complex patterns. There are numerous activation functions in the deep learning domain but this section focuses on three important activation functions required for the proposed algorithm.

ReLU: Rectified Linear Unit (ReLU) is one of the most commonly used activation functions in convolutional neural networks. As it it can be observed in equation 2.4 The function returns zero if the input is negative and behaves linear if the input is positive. The behaviour of ReLU is exemplified in the Figure 2.3 (a).

$$f(x) = max(0, x) \tag{2.4}$$

LeakyReLU: It is based on ReLU activation function. Instead of having flat slope for negative values, it has a small slope. The coefficient of slope is not dynamically learnt but will be set before training (equation 2.5). The coefficient chosen in Figure 2.3 (b) is 0.1.

$$f(x) = max(0, x) + negative\_slope \times min(0, x) \tag{2.5}$$

Sigmoid: The activation curve looks like a S curve and restricts the output between 0 and 1. As it can be seen in Figure 2.3 (c) the output saturates at 0 and 1 and the functions keeps increasing for the values in between (equation 2.6). As the probability is always between 0 and 1, sigmoid is the most used activation function for models whose output is a probability.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

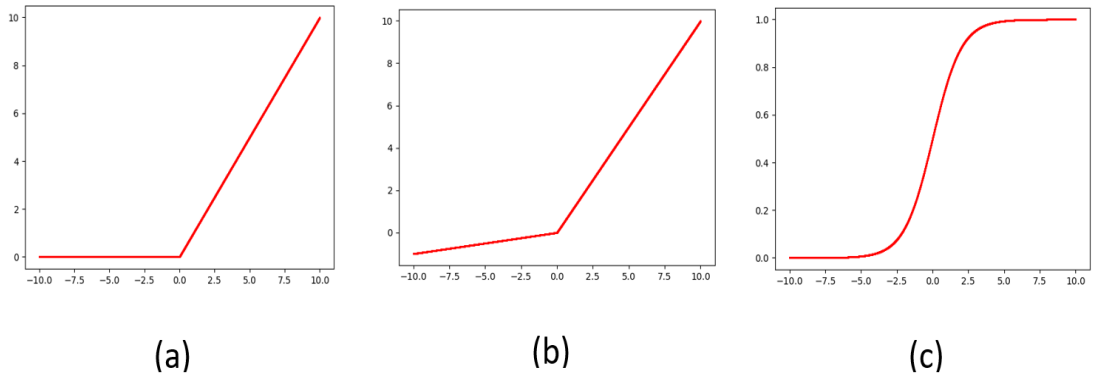


Figure 2.3: Illustration of different activation functions. **(a):** ReLU activation function. **(b)**: Leaky ReLU activation function with negative slope of 0.1. **(c):** Sigmoid activation function

2.2.3 Gradient Descent and Back Propagation

As highlighted in the previous section, the output of an untrained network will mostly deviate from the actual value. Gradient decent is an algorithm which focuses on minimizing the cost function by adjusting the network parameters. In calculus, gradient of a function is a vector which points in the direction where function increases and negated gradient points in the direction where function decreases. This property can be applied to find the updated parameters where the current parameters are updated with -∇J with learning rate η shown in equation 2.7

$$w_{i+1} = w_i - \eta \nabla J(w_i) \tag{2.7}$$

Backpropagation [36] is an important element in training a neural network. During backpropagation, gradient of the cost function is computed with respect to the learnable parameters (weights and biases) of the network. As explained in the equation 2.7 before, to avoid large changes in the parameters, gradient vector along with learning rate is used to update the parameters. To further understand this, let us consider a simple network with one input layer, two hidden layers and one output layer with sigmoid activation as shown in the Figure 2.4. The learnable parameters are $w^{(L-2)}$, $b^{(L-2)}$, $w^{(L-1)}$, $b^{(L-1)}$, $w^{(L)}$ and $b^{(L)}$ where w represents weight, b represents biases of different layers $L-n$. The activation of the last layer and last but one layer is $a^{(L)}$ and $a^{(L-1)}$ respectively. The desired output of the network is denoted by y and the given input is x.

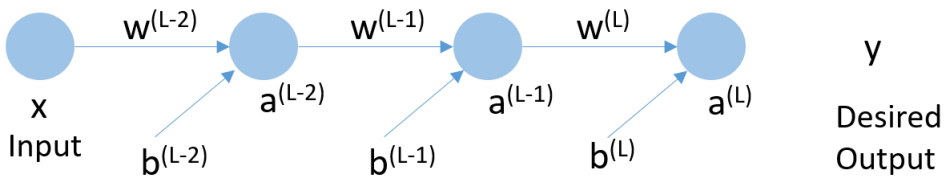


Figure 2.4: A simple neural network with hidden layers to illustrate back propagation. x and y indicate the network input and expected output respectively. The activation of $(L-n)^{th}$ layer is indicated by $a^{(L-n)}$. Weights and biases of the respective layers are represented by $w^{(L-n)}$ and $b^{(L-n)}$.

The $z^{(L)}$ is dependent on the parameters and activation of the previous layer $a^{(L-1)}$ (equation 2.8) which is in turn dependent on the activations of the previous layers. Sigmoid activation function σ is applied on $z^{(L)}$ to obtain the network output, $a^{(L)}$ and is shown in the equation 2.9. Equation 2.10 indicates the cost function of one sample.

$$z^{(L)} = w^{(L)} \cdot a^{(L-1)} + b^{(L)} \tag{2.8}$$

$$a^{(L)} = \sigma \cdot z^{(L)} \tag{2.9}$$

$$J(w,b) = \left(a^{(L)} - y\right)^2 \tag{2.10}$$

To understand how the cost function can be minimized by tweaking the learnable parameters let us consider the following equations. The cost function $J(w,b)$ is not directly dependent on weights and biases as explained in the previous equations. Therefore to evaluate the partial derivatives, let us consider the chain rule (equation 2.11). Each of the individual partial derivatives is shown in equations 2.12, 2.13 and 2.14. The partial derivative of the sigmoid activation function is σ'.

$$\frac{\partial J(w,b)}{\partial w^{(L)}} = \frac{\partial J(w,b)^{(L)}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{L}} \cdot \frac{\partial z^{(L)}}{\partial w^{L}} \tag{2.11}$$

$$\frac{\partial J(w,b)^{(L)}}{\partial a^{(L)}} = 2(a^{(L)} - y) \tag{2.12}$$

$$\frac{\partial a^{(L)}}{\partial z^{L}} = \sigma'(z^{(L)}) \tag{2.13}$$

$$\frac{\partial z^{(L)}}{\partial w^{L}} = a^{(L-1)} \tag{2.14}$$

The derivative of the cost function with respect to bias is also calculated in a similar way. The cost function can be calculated on several training samples by taking their average. As shown in equation 2.14, the partial derivative of the current layer's output is dependent on the activation of the previous layer ($a(L-1)$), which is in turn dependent on the parameters of the $L-1$ layer. By using the chain rule, the cost function can be propagated backwards from the last layer to the first layer. According to the equation 2.7, the parameters are updated in each layer with the computed gradient. For simplicity, the illustration of backpropagation in this section is done using a single neuron. But the same principle can be extended to train dense neural networks.

2.2.4 Overfitting and Underfitting

The actual data on which model is trained is called training data. The learnable parameters θ will be optimized to best predict the training data. The model performance will be frequently evaluated on the validation data or development data. It is used to fine tune the model hyperparameters but it will never learn from it. Once the model is completely trained on train data, it is tested on test data which provide the unbiased evaluation of the final trained model. The test data is curated in such a way that it covers real world situation.

The primary causes of poor performance of the model, affecting its generalizability, are overfitting and underfitting. **Overfitting** occurs when a model fails to perform well on unseen test data despite its good prediction on training data. That means, instead of improving its ability to solve problems, the model simply learns some random regularity in the training data [19]. Overfitting is characterized by low bias and high variance estimators. **Underfitting** is the opposite of overfitting, where a model becomes incapable of capturing the variability of training data, resulting in poor performance of both training data and test data. For example, trying to fit a linear classifier for non-linear data distribution. For example, trying to fit a linear classifier for non-linear data distribution. An illustration of model generalizability is shown in the Figure 2.5.

2.2.5 Regularization

Major concern in developing a machine learning model is do design an algorithm which not only has good performance on training data but also on the unseen data distribution. The adapted methodologies to decrease the test error could some time lead to increase in the training error. These strategies are collectively called as regularization [12]. This section will discuss about the popular regularization techniques used to reduce overfitting.

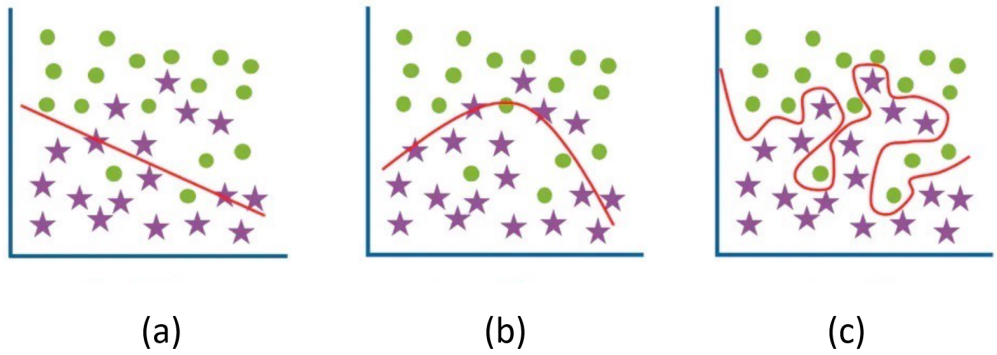


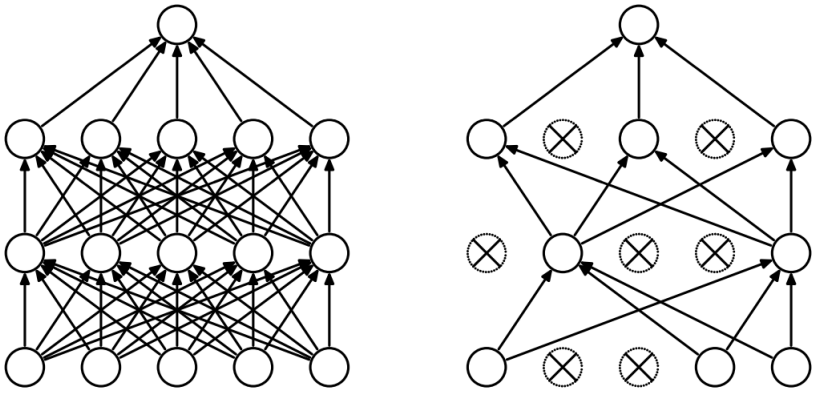
Figure 2.5: Example of a binary classifier with different generalizability. (a) Classifier which is not capable of learning training data and exhibiting underfitting. (b) Optimal classifier with good generalization. (c) Classifier memorizing the training data which is suffering from overfitting [29].

L1 and L2 Regularization: These regularizations methods reduce the variance by decreasing the parameter values. L1 regularization is called *Lasso Regression* and L2 regularization is called *Ridge Regression*. L1 regularization adds absolute value of model parameters to original loss function as penalty term (equation 2.15). Whereas, L2 regularization will add squared parameters to the penalty term (equation 2.16). The value of λ decides the amount of regularization. High λ value will add more weight to penalty term which leads to underfitting. Having said, λ value should be carefully chosen and can be tuned as an hyperparameter. As lasso regression can push the parameters to lower values, it can perform automatic feature selection.

$$CostFunction = LossFunction + \lambda \sum_{i}^{N} |\theta_i| \tag{2.15}$$

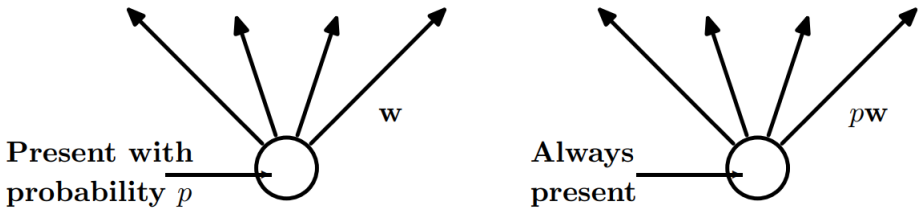
$$CostFunction = LossFunction + \lambda \sum_{i}^{N} \theta_i^2 \tag{2.16}$$

Dropout: It is a regularization technique that aims to reduce overfitting by randomly dropping a few neurons along with their inputs and outputs. The dropout applied to a dense neural network is depicted in the Figure 2.6. There are 5 neurons in the input layer and 5 neurons in the hidden layers, and some of the neurons are randomly dropped, thus reducing the training computation. Dropout creates a thin network from a deep neural network and prevents the neurons from co-adapting. The rate parameter decides what fraction of neurons should be dropped during training and can vary between 0 and 1. The dropout layer has different performance in the test phase compared to the training phase. A simple probabilistic method is used to evaluate the model on test data where the probability p with which the neuron is retained during training is multiplied by the weights of that neuron as shown in Figure 2.7. This method makes sure the predicted output is similar to the actual output during training.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.6: Dropout illustration in the dense neural network. **Left**: Standard neural network with two hidden layers. **Right**: Dropout applied to the left network resulting in thinned network. The crossed neurons indicates the dropped neurons. [40].



(a) At training time (b) At test time

Figure 2.7: Left: A neuron which is present with a probability p during training and connected with weight **w** with neurons in the next layer. **Right**: A neuron which is always present during testing and connected to the neurons in the next layers with probability p multiplied to the weights. [40].

Early Stopping: It is one of the most commonly used regularization methods. Early stopping prevents overfitting by restricting the optimization to a small area of parameter space. The number of training epochs plays an important role in training a neural network. Higher training epochs will result in the model overfitting to the training data, whereas lower values will lead to underfitting. With the validation set, it is possible to monitor the learning and generalizability of the model. With increasing epochs, training loss decreases and so does validation loss. But after some epochs, it can be observed in Figure 2.8 that the training loss keeps decreasing but the validation loss increases, resulting in the model overfitting on training data. The parameters can be stored and updated during every epoch. When the update does not result in the reduction of validation loss or error, the training can be stopped and the previously stored parameters can be used for evaluation.

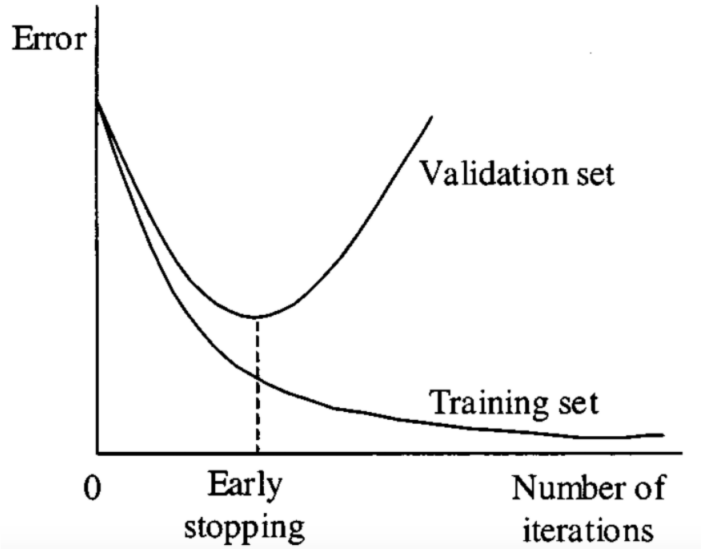


Figure 2.8: : Illustration of early stopping. As the loss or error on the validation dataset increases the training is stopped to avoid overfitting [10].

Data Augmentation: In the field of computer vision, data augmentation is a powerful regularization strategy. With increase in training epochs, the model memorizes parts of the images. To overcome this problem, random and logical transformations can be applied to the original dataset, which results in slightly different images from the original images. Transformations could be vertically or horizontally flipping the images, rotating the images by X degrees, adding noise to images and adjusting contrast, saturation or image brightness. If the model is trained on MNIST [9] dataset to recognize the digits, vertical flipping of 6 would result in 9 and thus affecting the model performance. Therefore set of transformations should be carefully chosen. Few examples of image augmentation is shown in the Figure 2.9.

2.3 Convolutional Neural Networks

Convolutional Neural Networks simply known as CNNs are part of artificial neural networks that explore the principle of linear algebra like matrix multiplication to deliver superior performance in computer vision and natural language applications. The major advantage of CNNs over ANNs are their reduced number of learnable parameters. The problems solved by CNNs are spatially independent. For example, the detection of car in an image is important, irrespective of it's position in the image. The three important layers in CNNs are convolutional layers, pooling layers and fully connected layers which are stacked on one another. With each layer the learning complexity increases enabling it to obtain abstract features. An example convolutional neural network with multiple layers is shown in Figure 2.10

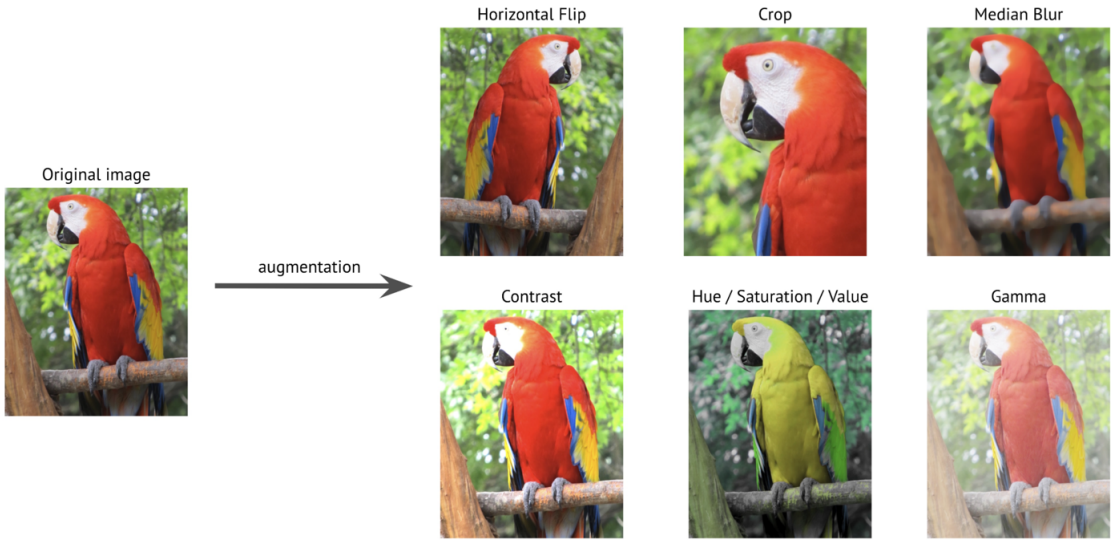


Figure 2.9: Left: Illustration of different data augmentations applied to the original image [1].

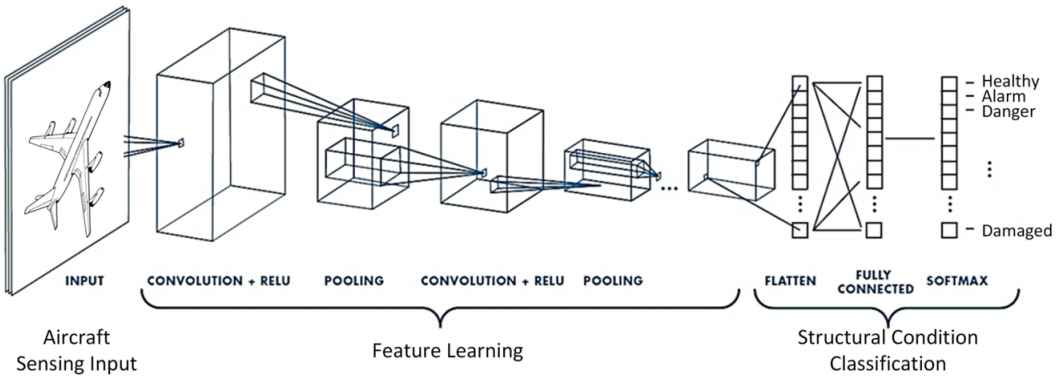


Figure 2.10: Illustration of a convolutional neural network with multiple layers [44].

2.3.1 Convolutional Layers

Convolutional layer is a computationally heavy layer which is also the core building block of CNNs. The learnable parameters of the convolutional layers are called filters or kernels. The size of the filter determine the receptive field of the input. Though the filter size is small spatially, it is convolved through the entire width, height and depth of the input image producing activation maps. The convolution operation between kernels and input image is shown in equation 2.17 and is represented in the Figure 2.11. The initial layers learn lower level features like edges and corners, while the middle layers learn to identify the objects in the image, for example, the eyes and nose of a dog. The final layer learns to recognize an entire object in different shapes and positions, like recognizing the animal in the given image is a dog. Number of filters, stride and zero-padding are the three main hyperparameters which determines the size of the output and must be defined before

training the CNNs [17]. The depth of the output is determined by number of filters. Stride is the number of pixels kernels move during the convolution, larger stride will result in smaller output. Zero-padding sets the elements outside the input matrix to zero if the kernels does not fit the matrix. *Valid padding* drops the last convolution in case of dimension mismatch. *Same padding* is used to produce the output layer which has same size as the input layer. *Full padding* add zeros to the border of input matrix resulting in the size of the layer output larger than input. The size of the output of the convolutional layer is as shown in equation 2.18. The output of the convolutional layers is passed through activation functions like ReLU to introduce non linearity in the network.

$$O(i,j,c) = (I * K)(i,j,c) = \sum_m \sum_n \sum_o I(i+m, j+n, c+o)K(m,n,o) \tag{2.17}$$

$$OutputSize = ([InputSize - KernelSize + 2 * Padding]/Stride) + 1 \tag{2.18}$$

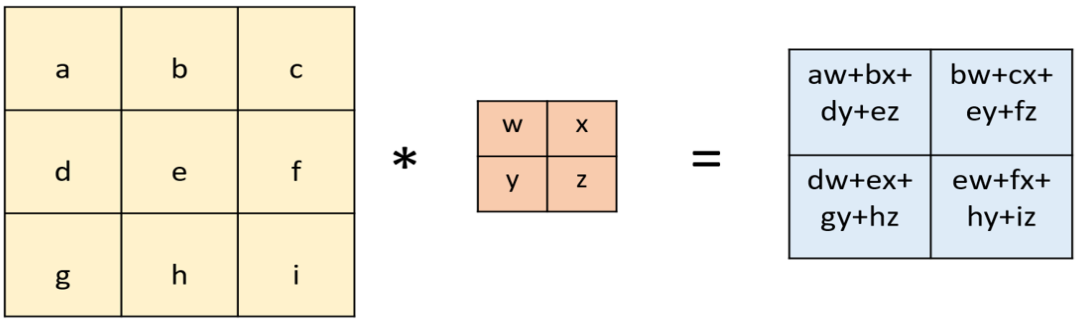


Figure 2.11: Convolution operation between input image and filter. The filter is convoled with the input image to produce corresponding value in the output channel.

2.3.2 Pooling Layers

Pooling layers or downsampling layers reduce the number of parameters in the input by performing dimensionality reduction. The operation of the pooling layer is similar to that of the convolutional layer where a filter is swept across the input to produce an output array, but the weights of this filter are not learnable; they only perform aggregation function in the receptive field. **Max pooling** is the most used pooling operation where it selects the maximum pixel value as it slides across the input and pass it to the output array. Figure 2.12 shows max-pooling operation with 2x2 filter and stride of 2. **Average pooling**, on the other hand, selects the average value of the receptive field to send it to the output array. Pooling operation looses a lot of information but helps reducing overfitting and increase the model performance.

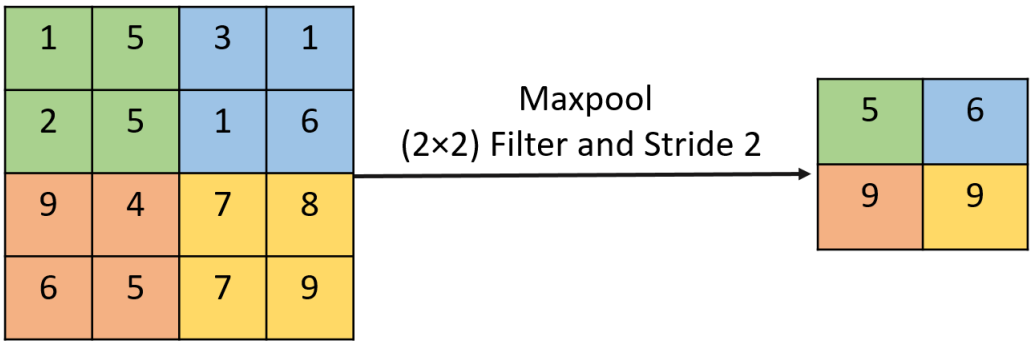


Figure 2.12: Example of max-pooling with filter of size (2x2) and stride of 2.

2.3.3 Fully Connected Layers

The output of convolutional layer or pooling layer is flattened and fed to fully connected (FC) layer. Unlinke convolutional layers, the nodes in the output layer is connected directly with the nodes in the previous layer. These layers forms the last few layers and are placed before the output layer. FC layer can be termed as classification layer as it performs classification on the features extracted from the previous layers. The general structure of a CNN architecture [41] is as follows:

Input–> Convolution–> Activation Function–> Pooling–> Fully Connected

Let us consider a 3D image of size 32x32x3 is passed thorough the network shown in the Figure 2.10. The output of the first convolutional layer with padding and stride as 1 and 10 filters of size 3x3x3 is 32x32x10. This output is passed through ReLU activation function and fed to pooling layer which performs dimensionality reduction and produce downsampled output. Depending on the network architecture, these feature maps are passed through stacked convolution layers, ReLU and pooling layers to extract more abstract features of the input image. These feature maps are finally flattend and fed to FC layers to compute the output of the CNN model.

2.4 Siamese Neural Networks

Humans have the inherent ability to learn and recognize new patterns, as well as to expand that knowledge to recognize variants on previously taught notions. People can use this ability to not only recognize taught features when presented with stimuli, but also to distinguish across unrelated categories. For example, different breeds of dogs are classified as dogs, while cats are classified as non-dogs. There have been extensive research in the field of computer science to mathematically and semantically compare two list of elements. Cosine distance, Manhattan distance or Euclidean distance could be a good choice to compare two elements statistically, whereas to compare the correlation between two elements, Pearson correlation coefficient, Kendall τ distance or Spearman's ρ rank coefficient can be a better choice. These approaches work well when two elements in comparison have same meanings and data types and fails when the list contains elements having

different meanings or data types [5]. An example for the latter use case is forgery recognition. Bromley and LeCun introduced Siamese neural network in the early 1990s as a solution for signature verification problem [4]. The following sections will describe the architecture and loss functions used in training Siamese neural networks.

2.4.1 Architecture

Siamese neural network consists of twin networks which are joined together at the end by energy function [24]. The weights of the neural networks are tied together, meaning that both the networks in the twin structure have the same weights. The weight tying ensures that if the two similar images are passed through the same network, their vectors in the feature space will not be mapped at different locations as the networks compute the same function. Different energy functions can be used based on the network architecture. The authors in [6] use contrastive energy function which decrease the energy between like pairs and increase the energy between unlike pairs. However the authors in [24] employ weighted L1 distance combined with sigmoid activation function between two feature vectors produced by twin networks. The output is mapped between the interval [0,1] where 0 indicates the images are dissimilar and 1 indicates similar images. The architecture of the siamese neural network with twin structure and distance layer with logistic prediction p for binary classification is as shown in the Figure 2.13.

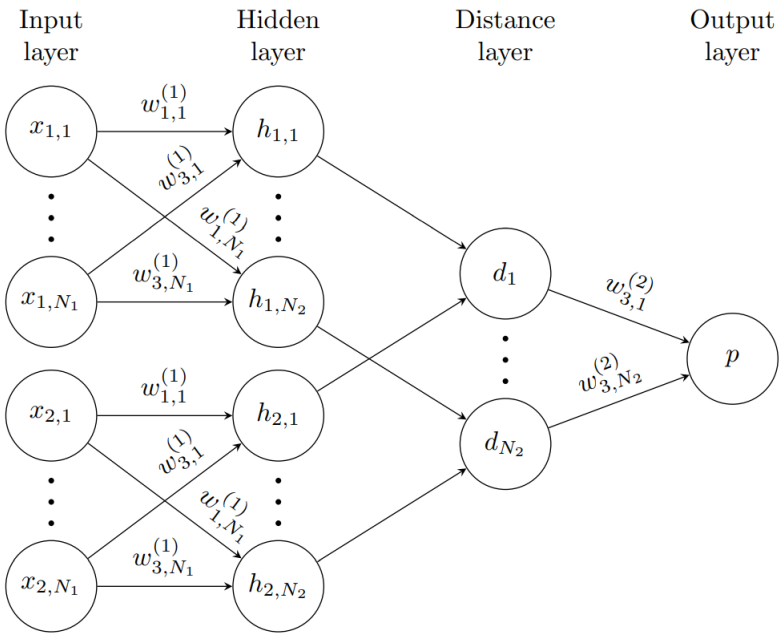


Figure 2.13: A simple siamese neural network with two hidden layers for binary classification. The twin network have same structure at top and bottom with shared weights [24].

2.4.2 Special Loss Functions

Special loss functions called ranking loss are used in training siamese nets. The objective of these loss functions are not to predict the label or a value but to determine the relative distance between two given inputs which is often termed as metric learning. Contrastive loss is used to train the network on pairs of data and triplet loss is used on triplets of training data. The feature embeddings of the input image is obtained using feature extractors and a metric function like Euclidean distance is employed to calculate the distance between pairs or triples of embeddings [11]. The Feature extractors, in this case, the twin networks, are trained to produce representation close to each other if the input images are similar and far apart if the input images are dissimilar. To train a contrastive loss, positive and negative samples of the training data points are used as input. Positive pair includes anchor sample and positive sample and negative pair is composed by anchor sample and negative sample. The objective of the loss function is to bring the positive pairs as close as possible and separate them from the negative pairs with a margin of m. Let r_0 and r_1 indicate the representation of image pairs and with y being a binary label with value as 1 for positive pair and 0 for negative pair, the loss function is given by the equation 2.19. Triplet loss on the other hand is trained on anchor sample, positive sample and negative sample whose representations are denoted as r_a, r_p and r_n respectively. The network is optimized in such a way that the distance between anchor-positive sample and anchor-negative sample is m. The equation for triplet loss is given by the equation 2.20. Figure 2.14 shows input samples and optimization approach for contrastive loss and triplet loss.

$$L(r_0, r_1, y) = y||r_0 - r_1|| + (1 - y)max(0, m - ||r_0 - r_1||) \tag{2.19}$$

$$L(r_a, r_p, r_n) = max(0, m + d(r_a, r_p) - d(r_a, r_n)) \tag{2.20}$$

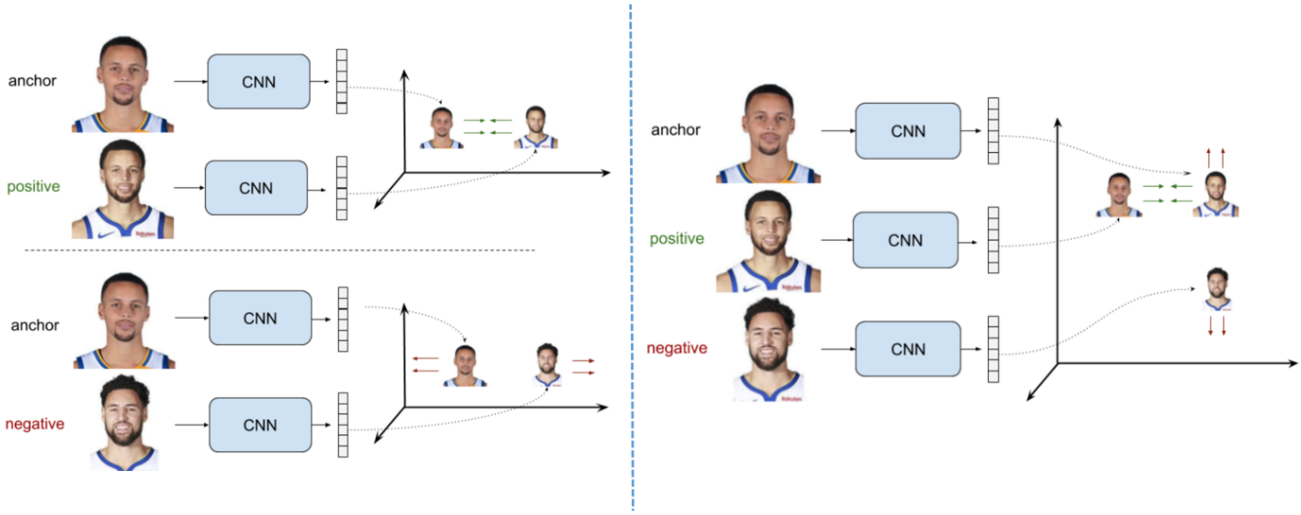


Figure 2.14: Representation of ranking loss. **Left:** Contrastive Loss which pushes positive pair closer and negative pair apart. **Right:** Triplet loss which pushes anchor and positive samples closer and anchor and negative samples away [11].

3 Related Work

Bromley and LeCun [4] first introduced siamese neural network to develop a signature verification system as image matching problem. Authors used 5990 Signature Capture Devices to collect the signature data. The proposed algorithm uses two separate sub network which extracts features from given pair of signatures and one output which indicates the similarity between two inputs. The distance between two feature vectors is calculated by cosine of the angle between them. During verification, features extracted from test signature is compared with stored feature vector of the signer. If the difference between them is greater than a threshold, then the signature is classified as forgery. This paper supports that siamese network are capable of classifying unseen signatures by comparing them with a reference image. As a result, in our implementation, we compare the test image with a positive reference image.

The authors in [24] have developed a siamese neural network for character recognition using Omniglot dataset [25]. The dataset consist of 50 different alphabets by 20 drawers. The features extracted by siamese convolutional neural networks are flattened and passed through a fully connected layer to obtain the representation of images in embedding space. L1 distance is computed between the images and passed through fully connected layer followed by sigmoid activation function which provides the similarity scores. 40 alphabets from 12 drawers are used in training to enable the model to learn discriminative feature. The verification task is performed on the remaining 10 alphabets drawn by 4 drawers from the evaluation set. The authors conclude that the proposed metric learning approach can produce human-level accuracy and can be extended to one-shot learning tasks in various image classification domains.

A special siamese convolutional neural network called Fusioning Convolutional Siamese Neural Network (FCSNN) is developed in [30] which identifies the defect types of new object classes without retraining. The proposed architecture uses VGG16 [37] as feature extractor. A pair of images are passed through feature extractors and a fully connected layer to obtain the feature vectors. The authors also perform fusion of feature vectors using fully connected layers to which the L1 distance between the vectors and concatenation of two vectors are fed as input. The similarity score is provided by the sigmoid layer where high score indicate the defects from identical attributes. The network is trained to recognize 8 defects from 21 types of traffic signs and a few defective samples from the casting dataset [8] and is evaluated on 25 different types of traffic signs and the remaining casting data. The performed evaluation is a 20-way one shot testing where the network has to pick the right pair of the test image given 20 support images. Through these tests, the authors prove that the proposed architecture has better performance on untrained object types. Following this work, we have experimented training the siamese network with pretrained model as feature extractor.

Siamese neural network with few shot learning is used to classify the steel surface defects in [23]. The North Eastern University surface defect dataset, which consists of 9 defects, is used for experiments. Twin CNN layers extract the features from the pair of images. Contrastive loss with a margin 1 which calculates the distance between the feature vectors is used to train the neural

network. In use case 1, the model is trained on 5 images from each class and tested on the remaining images. Whereas in use case 2, the model is trained on 5 images from only 8 classes and tested on one unseen class. An analysis is performed on how the accuracy of the model changes with different number of training images per class and total number of classes. The test results indicate siamese neural network not only require less images for training but also can predict the defect of unknown class distribution. The ability of the network to learn from few shots of training data has inspired our approach to scale the pretrained network to unseen data distribution using 6 or 10 images.

The authors in [50] use siamese neural network for retrieval of family members. Experiments have been conducted with different backbone architectures, loss functions and similarity computation. Resnet50 [15] and SENet50[16] are chosen as backbone and are trained with focal loss and BCE loss resulting in four types of models. The feature vectors from the backbone is extracted either through max pooling layer or average pooling layer. The feature 5000 pairs of family members are used as positive samples and 5000 pairs without kinship are used as negative samples. The similarity of features vectors are calculated by fully connected (FC similarity) layers or cosine similarity. The features vectors of the image pairs (x and y) in FC similarity are combined with two different operation as shown in equation 3.1 and equation 3.2 where $\oplus$ indicate concatenation.

$$(x^2 - y^2) \oplus (x - y)^2 \tag{3.1}$$

$$(x^2 - y^2) \oplus (x - y)^2 \oplus (x \cdot y) \tag{3.2}$$

The combined feature vectors are passed through fully connected layers to produce a scalar value in the range 0 and 1. The authors conclude that cosine similarity performed on the features extracted with max pooling and trained on binary cross-entropy (BCE) loss outperforms other experimental conditions. As discussed in this paper, we have used BCE loss to train our neural network.

[39] achieves state-of-the-art face recognition performance using embeddings $f(x)$ which maps the images to Euclidean space such that the squared distance between pairs of images of similar identities is small and dissimilar identities are large. The authors experiment with two different architectures, one based on ZeilerFergus [51] network and other is Inception [43] type network as core architectures. The network is fed with triplet images where image, x_i^a (anchor) of a person is pulled closer to all the other images of the same person, x_i^p (positive), and images of any other person, x_i^n (negative) are separated with a margin. The network is trained with triplet loss which minimizes the distance between anchor and positive samples and maximizes the distance between anchor and negative samples. To ensure faster convergence, the authors generate triplets online which picks all anchor-positive pair and selects only hard anchor-negative samples ($argmin_{x_i^n} ||f(x_i^a) - f(x_i^n)||_2^2$) in a mini-batch. The model's performance on various embedding dimensionalities was also investigated, with 128 and 256 dimensional vectors outperforming others. Based on this proof, the dimensionality of the vector in embedding space in our work is set to 256.

All of the works discussed in this section demonstrate that the siamese network can scale to unknown data distribution. But, very little research has been conducted on implementing siamese network based solutions for quality inspection in industrial applications. The model FCSNN [30] is trained to identify the casting defect in metal casting process. The model tested on the casting defects which are similar to training data has an accuracy of 99.50%. With 10 images in few-shot learning, the

proposed approach [23] classifies 2 classes of steel surface defects with an accuracy of 99.1%. These two papers prove that siamese network clasifiers will result in high test accuracies on industrial dataset. We propose to evaluate the test nuts of different parts by retraining the model trained to perform quality analysis of one part with 6 or 10 images. Our approach is more stable as we are scaling a trained baseline model to new test cases. As a result, the model is capable of classifying unseen nuts from different parts.

4 Methodology

4.1 Solution Realization

As mentioned in the previous sections, to design a system which can be used to perform quality analysis on various use cases, we have chosen siamese neural network based artificial intelligence solution. The proposed solution can be divided into four main modules, which are the input layer, feature extractor, distance metric and classifier as shown in Figure 4.1. Two main implementation approaches have been designed to develop a stable solution. In this section, we are going to focus on detailed architecture and internal details of the four sub systems in these two implementations.

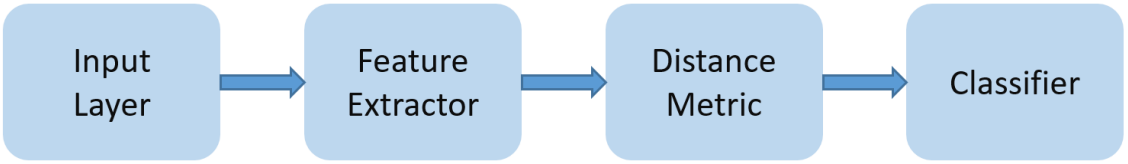


Figure 4.1: Flow chart of the proposed solution. The architecture is composed of four main sub modules namely input layer, feature extractor, distance metric and classifier.

4.2 Architecture-1

Input Layer

Unlike conventional CNNs, siamese networks take either pair of images or triplet images as input. The main principle behind this method is that the network is capable to learn the semantic similarity or differences between the images. This implementation is realized by training the network with pair of images. So, for a given nut, the correctly placed nut is taken as a reference nut. The image pair is considered valid or positive if reference image is compared with correctly placed nut and invalid or negative if it is compared with either incorrect nut or if the nut is flipped or if the nut is missing. The network is trained to predict 1 for valid case and 0 for invalid case. An example of the valid and invalid image pairs is shown in the Figure 4.2.

Feature Extractor

Custom CNN architecture is designed to extract features from the images. The pair of images are passed through twin networks to obtain the embedding space representation of the images. Grayscale image (single channel) of size 224x224 is fed to feature extractors. The image is passed through alternating layers of convolutions and sub sampling. As the layer deepens, the depth of the feature maps increases and spatial dimension decreases. The detailed network architecture is illustrated in

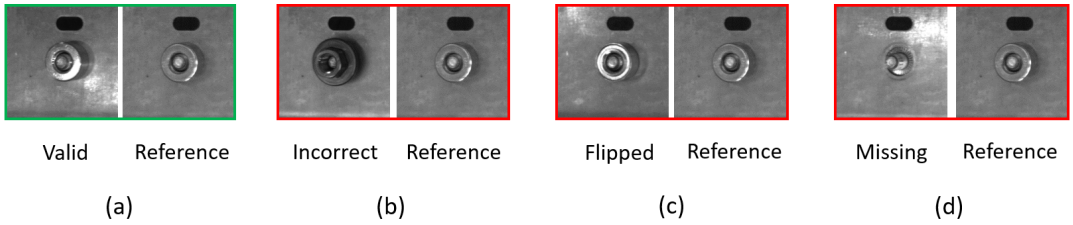


Figure 4.2: Valid Pair: (a) Reference image compared with correctly placed nut. **Invalid Pairs:** (b) Reference nut and incorrect nut. (c) Reference nut and flipped nut. (d) Reference nut and missing nut.

the Figure 4.3 and feature extractor in Figure 4.4. The Stanford course on 'Convolutional Neural Network for Visual Recognition' highlights the fact that ReLU activation functions are fragile and can die during training [42]. Therefore to mitigate the problem of 'dying ReLU', Leaky ReLU is used as activation function. Following [20] and [28], batch norm [18] is performed after activation function for better results. An interesting observation here is, instead of flatten layer, global average pooling layer is used in the final layer which produces 256 feature vectors. The global pooling reduces the overfitting of the model to training data and increases the model's generalizability. Output activation maps of first convolution and last convolution layer is plotted in Figure 4.5. It can be seen that the activation map closer to the input learn general features like corner, edges and all the related fine details. But as the network deepens, the model learns more abstract features and it is not possible to identify the nut in the last layer. This is because the model learns general details that is used in classification at deeper layers.

Distance Metric

Distance metric is used to calculate the similarity between the data points. Following the work of [52], we have experimented with cosine distance, L1 distance and Euclidean distance to calculate the distance between feature vectors in embedding space. In this architecture, the 256 feature vectors are split into 8 vectors. Therefore, both reference image and test image (valid or invalid) will produce 8 vectors each and the distance between them is calculated using Euclidean distance. Let a and the b be two vectors such that, $a = (x_1, x_2, ..., x_n)$ and $b = (y_1, y_2, ..., y_n)$ and Euclidean distance between two vectors is given by equation 4.1. The computed distance is fed to classifier to classify if the given pair of images are valid or invalid.

$$d(a, b) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{4.1}$$

Classifier

Fully connected layer (FCL) is used to detect if the nuts are misplaced or not. The classifier consists of two hidden layers with 8 neurons and 16 neurons respectively. The 8 Euclidean distance values computed between input image and reference image are passed as input to classifier. The final activation function of the last layer is sigmoid. Therefor the classifier produces a value bounded

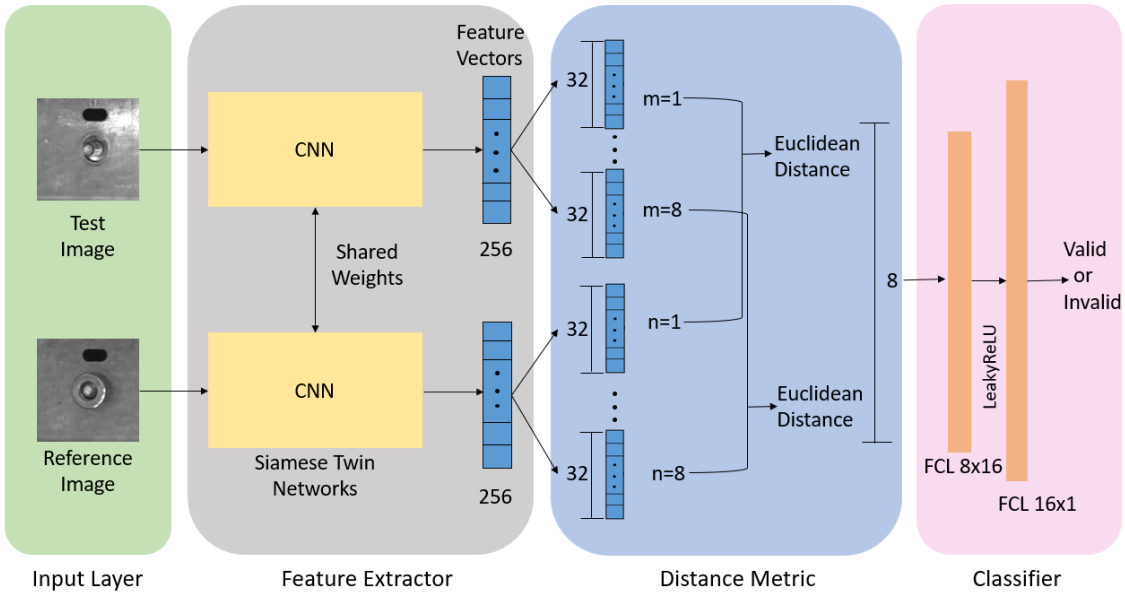


Figure 4.3: The proposed architecture-1 with four sub modules namely input layer, feature extractor, distance metric and classifier. The test image is an invalid image as the nut is missing. Therefore the network is expected to classify the test image as invalid

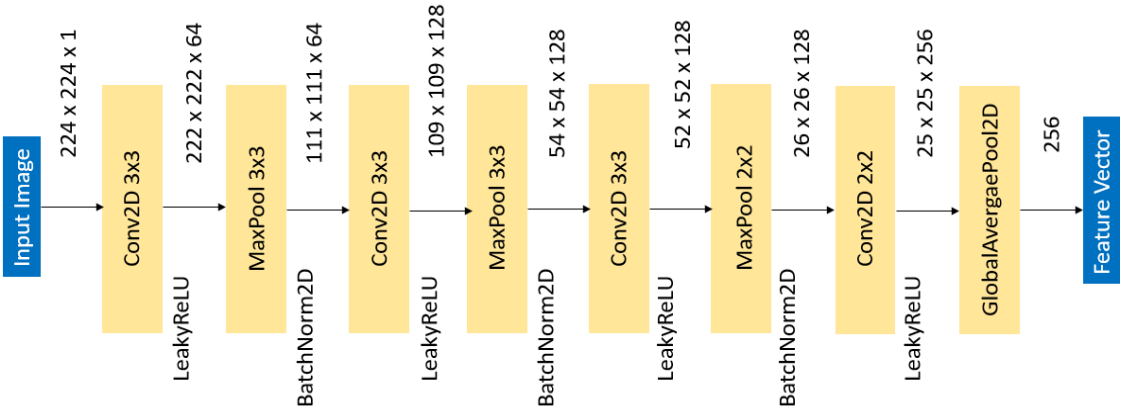


Figure 4.4: Detailed explanation of feature extractor used in architecture-1

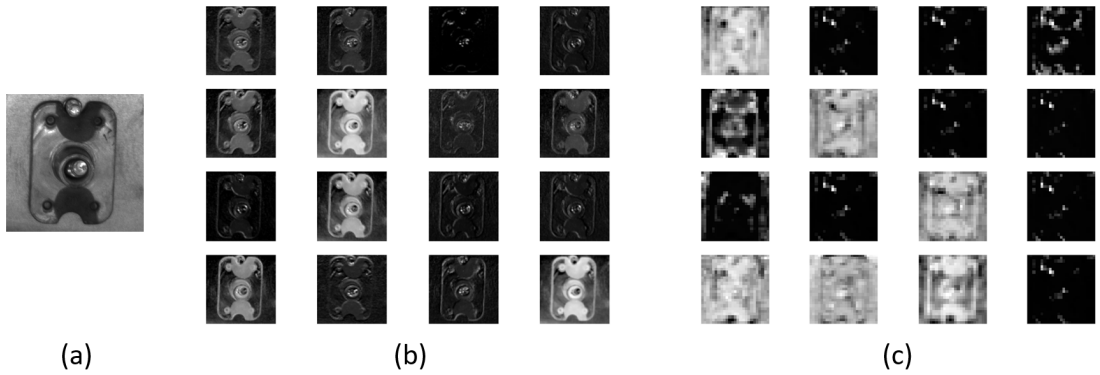


Figure 4.5: (a): Input image whose activation maps are visualized in CNN layers. **(b):** Activation map of the first convolutional layer **(c):** Activation map of the last convolutional layer.

between the interval [0,1]. The threshold is set to 0.5, if the classifier predicts a value greater than 0.5, the given pair of images are classified as valid. If the classifier prediction is less than 0.5, the input pair is classified as invalid pair.

4.3 Architecture-2

This implementation has architectural details similar to the previous implementation except for feature extractor module. Transfer learning is the method to reuse the pretrained model. It enhances the performance of learners on one domain by transferring the information from a related domain [49]. Transfer learning plays an important role when there is a limited amount of training data, which may be due to data being rare, inaccessible, or expensive. Andrew Ng, chief scientist at Baidu and professor at Stanford, said during NIPS 2016 tutorial [34] that transfer learning will be an important factor in the success of machine learning in industries [35]. This sections highlights the advantages of Effecientnet-B0 [47] as pretrained model and it's implementation as feature extractor in our proposed solution.

Feature Extractor

The authors in [47] have proposed a novel scaling approach by studying the impact of scaling width, depth and resolution of the model. The method aims to scale all the three dimensions with fixed set of scaling coefficients [45]. The relationship between scaling coefficients of different dimensions is computed by grid search of the baseline network. The baseline architecture is then scaled to required target size by applying these coefficients. The validity of the model scaling depends on the baseline network. The baseline architecture, EffecientNet-B0 is designed by performing neural architecture search using AutoML MNAS framework and has mobile inverted bottleneck convolution (MBConv), similar to MnasNet [46] and MobileNetV2 [38]. The model is light with 5.3M parameters and Top-1 accuracy of 76.3% and Top-5 accuracy of 93.2% on Imagenet [37]. As the architecture has good performance with less trainable parameters, tranfer learning is adapted and EffecientNet-B0 pretrained on ImageNet is used as feature extractor. The detailed architecture

is shown in the Figure 4.6. As shown in Figure 4.7, the output of feature extractor is 7x7x1280 which is passed through global average pooling layer resulting in 1280 feature vectors which is further sent to distance metric module explained in the previous architecture.

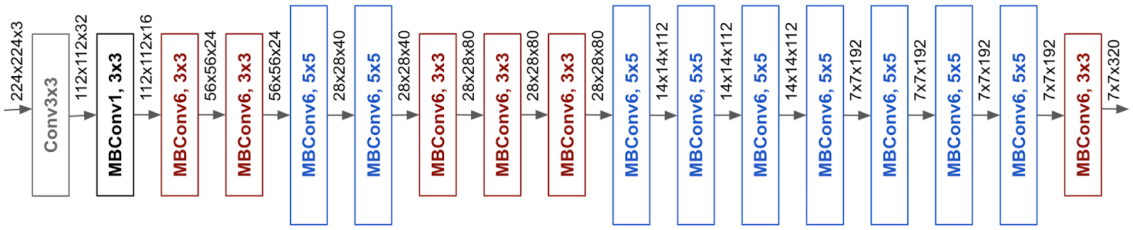


Figure 4.6: EffecientNet-B0 architecture. MBConv block indicate mobile inverted bottleneck convolution [45].

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 4.7: Architecture details where each stage i with $\hat{L}_i$ layers having $\hat{H}_i \times \hat{W}_i$ input dimension and output channel $\hat{C}_i$. $\hat{H}_i$ indicate height and $\hat{W}_i$ indicate width [47].

4.4 Solution Approach - Divide and Conquer

Divide and conquer is a popular problem solving strategy in computer science. The approach focuses on solving a complex problem by breaking it into smaller chunks. And by solving the smaller problems and combining their solutions together, the original complex problem is solved. Algorithm 4.1 [7] shows the step wise procedure for divide and conquer approach.

The same technique is followed in our implementation. To detect if the given body part has defective nuts, instead of inputting the entire image of the body part to neural network, each nut is considered as a separate problem. Individual nut is cropped from the image and is analysed if it is positioned correctly. For example as in Figure 4.8, if a part has 7 different nuts which needs to be checked before welding, single image of the part is taken and is cropped around each nut to produce 7 different images which is then tested. The problem is solved as a binary classification case where the output is 1 if the nut is valid and 0 if the nut is invalid.

Algorithm 4.1 Divide and conquer approach

Input: Problem to be solved **X**
Output: Combined solution of the problem
procedure DAC(X)
  if X is small enough **then**
    return Solution of X
  else
    Divide larger problem **X** into n smaller problems X_1,X_2,....,X_n
    DAC(X_i)
    return combined solution (DAC(X_1),(DAC(X_2),..,DAC(X_n))
  end if
end procedure

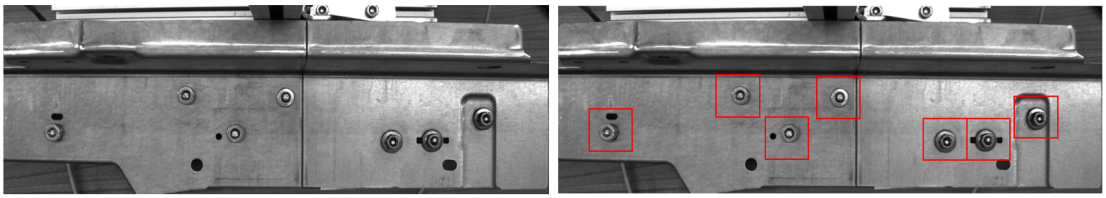


Figure 4.8: Left: Image of the entire part. **Right**: Image cropped around each nut resulting in 7 images

4.5 Scalability Method

Manufacturing industries have data redundancy, i.e. the same nut can be used in engine mount and seat tracks but can be produced in totally different production units. Therefore, we have proposed a solution which takes advantage of this fact and have designed a neural network which transfers the previously learned knowledge of the nuts to new use case with less training duration and image data. A baseline model is trained with commonly used nuts to perform quality analysis of one part. When the same model is used to detect incorrectly placed nuts in different part of the car, the model performance will decrease as the nut to be tested can be placed in different position and angle from the camera or the nut may differ in size or the part may contain few different nuts which model has never seen during training. The authors in [22] have proved that siamese network can learn with few shots of training data. So, when the requirement is to perform quality analysis of a new part, the baseline model is retrained with 6 (3 valid + 3 invalid) or 10 images (5 valid + 5 invalid), for a duration of less than one minute and the scaled model can be used to classify correct and incorrect nuts of a new part. These 6 or 10 images are addressed as scale images in the following sections. A flow chart indicating the step by step procedure of the proposed method is shown in the Figure 4.9

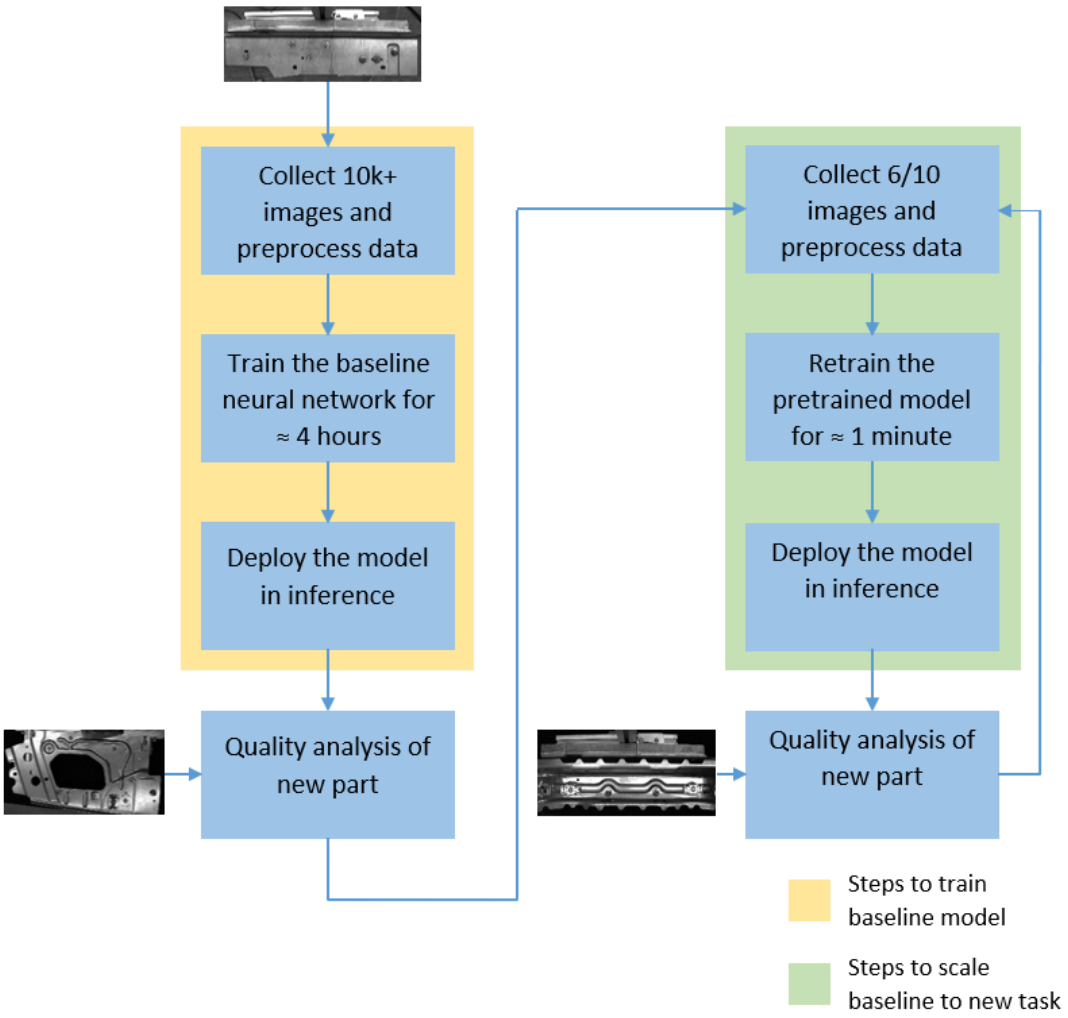


Figure 4.9: A step by step flow chart to illustrate the proposed solution. The process highlighted with the yellow background is used to train the baseline model and the one with green background is used to scale the trained baseline to new task.

4.6 Proof of Scalability

To test the scalability of the model, the test cases are chosen such that the nuts differ in shape and size compared to the nuts the baseline model is trained on. The test body part also contains a few nuts on which the model is trained, but these nuts are placed in different positions and orientations from the camera. As it was tedious to collect all the data from the production unit, we have designed a similar setup in the office environment to replicate the production process in the factory. So our dataset includes data from two lighting conditions (office and factory). Three use cases are used to determine the validity of the proposed solution, and this section highlights the details of the use cases.

4.6.1 Use Case 1:

In this case, the baseline model is trained on all the nuts (ID:1-ID:7) from part1 and only two nuts (ID:10 and ID:11) from part2. As we can see in Figure 4.10, the training data include three main types of nuts (ID:1, ID:5 and ID:10). The test data include completely different nuts (ID:8 and Id:13) and same nuts as training but in different position (ID:9, ID:12 and ID:14). the data from both the parts are collected in office lighting conditions. Few images of part1 from factory conditions are also included in training.

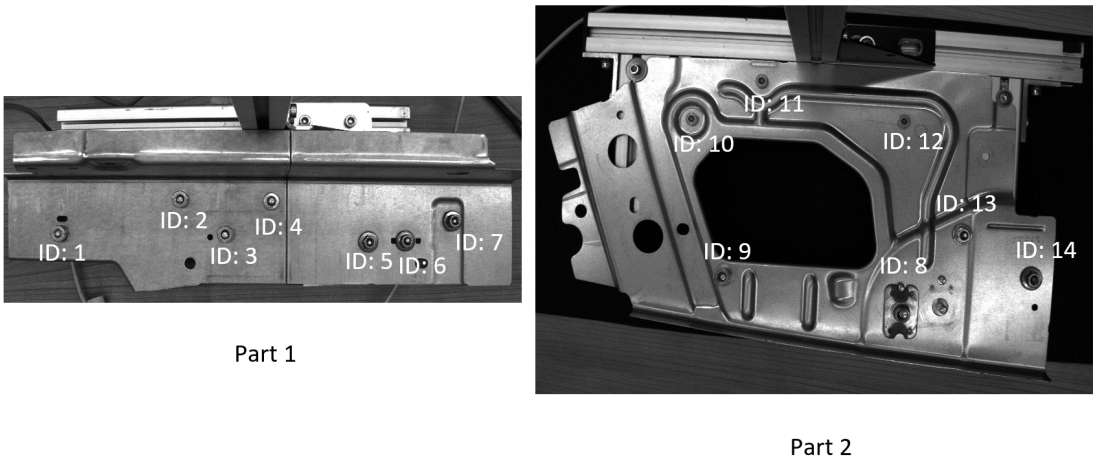


Figure 4.10: Two different parts which needs quality analysis before fabrication process. **Left:** Part 1 with two types of nuts which is completely used in training. **Right:** Part 2 with one category of nut in training and rest of the nuts in testing

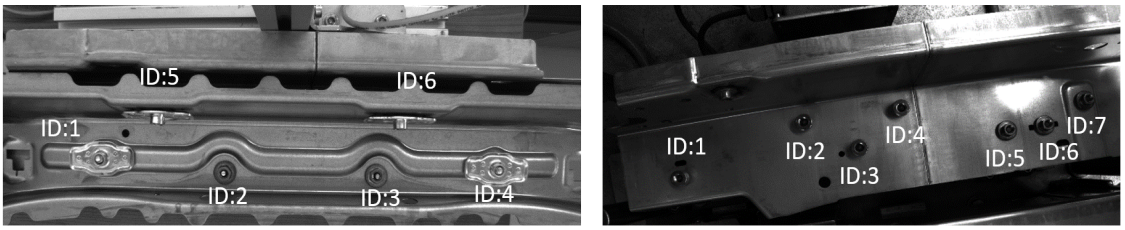
4.6.2 Use Case 2:

The focus of this use case is to validate the scalability of the siamese neural network on a new body part (Part3). The two nuts (ID:2 and ID:3) in the Figure 4.11 are the same nuts on which the model is trained. It is important to note that ID:5 and ID:6 are not just different nuts with bigger sizes like ID:1 and ID:4 but, the orientation at which the nuts are inspected is vertical, unlike the other nuts, which are placed horizontally. This test case evaluates the adaptability of the proposed model to varied orientations and sizes of the nut. The images of part 3 are collected in office environment.

4.6.3 Use Case 3:

As the brightness has large variations in factory and office use case, a separate baseline is trained for factory data. In this case, we have collected the data from part 1 in factory lighting conditions. As it can be seen in Figure 4.11, the image is too dark when compared to part 1 positioned in office lighting condition. This use case validates if the network is capable to predict with confidence if the

lighting condition changes. Due to the limitations of the availability of different parts in factory conditions, the scalability is evaluated on only one part. All the nuts except ID:2 and ID:7 are used in training to develop a baseline model which is later scaled to the test nuts ID:2 and ID:7.



Part 3

Part 1 in Factory Lighting

Figure 4.11: Left: Part 3 with six nuts. ID:2 and ID:3 is similar to ID:5 to ID:7 in part 1 shown in Figure 4.10 and ID:1, ID:4, ID:5 and ID:6 are unique nuts. **Right:** Part 1 with dark lighting conditions in the factory

5 Experiments and Results

5.1 Camera Specifications

Hardware and Firmware Details

GV-5280CP-M-GL (AB02020) industrial cameras with uEye+ standard from IDS Imaging Development Systems GmbH is used to capture image data. The GV code indicate that cameras are connected with Gigabit Ethernet.The firmware is supported with compact 2/3"global IMX264 shutter CMOS sensor from Sony and delivers almost noise-free images with high resolution [14]. The cameras are fast and reliable with extensive pixel pre-processing. Power over Ethernet (PoE) functionality enables single-cable operation upto 100 meters with GigE speed data transmission. The camera dimensions is 29x29x29 mm. The compact camera size makes it suitable for space critical applications like multi camera systems or on robot arms. The camera can be used in wide range of fields like automation, medical technologies, automotive, logistics and transport [13]. The camera is setup at 70 cm height from the body part and the aperture is set to F8.

Software Details

IDS peak is the free IDS camera software that is compatible with uEye+ industrial cameras. As a software development kit, it offers various software tools and programming interfaces required for programming the cameras. It offers programming interfaces in multiple languages like C, C++, C# with .NET and Python. We are using the Python interface to access the device and to adjust image acquisition settings. IDS Vision Cockpit tool is used to configure the standard camera parameters with a modern and interactive graphical user interface. It is used to control the brightness by adjusting the exposure time and to select the trigger mode in our application. The exposure time details for various use cases are explained in next section. To capture the images with software interface, the trigger mode is set to 'SW Trigger' mode. Figure 5.1 is the GUI of IDS vision cockpit.

5.2 Dataset Acquisition

All the images are monochrome or gray scale images. The camera is powered and transfers the data over ethernet. The camera is interfaced with an automated python script which runs on the PC and captures the images and stores the data in a local directory. During the acquisition, the valid images and invalid images are stored in a separate directory. The captured image is processed using OpenCV to crop each nut from the whole body part. Both the valid and invalid data folders contain a separate JSON file with details such as nut ID, width and height of the crop, x and y coordinates

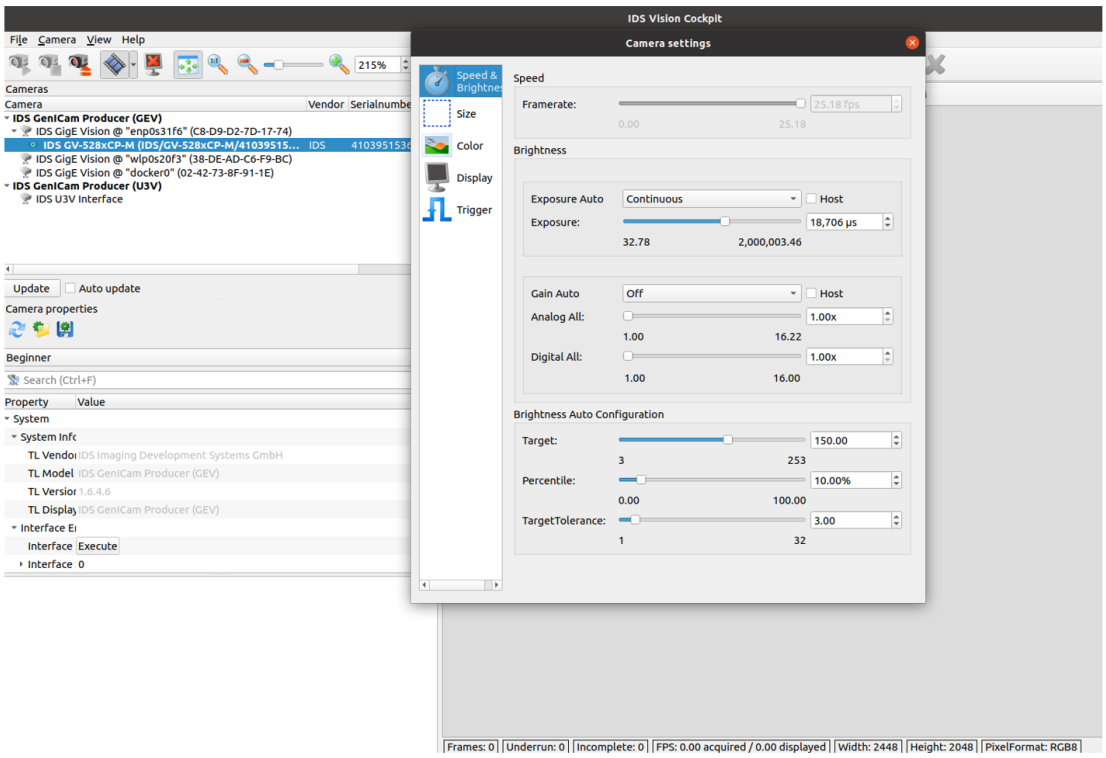


Figure 5.1: Graphical user interface of IDS vision cockpit used to set the exposure time and trigger mode.

of the crop center and label as 1 for valid cases and 0 for invalid cases. A python script loads the data from JSON file and crops each nuts from the image and stores each nut in a separate folder. As a result, both valid and invalid folders will further contain sub folders of each nut.

5.3 Dataset Details

We are using custom dataset for our experiments. As described earlier, we are focusing on three main use cases to prove the scalabilty of the designed siamese neural network. Use case 1 contains data from part 1 and part 2 (Figure 4.10) in office conditions. A 224x224 crop of each nut is used in training and testing the model. Total number of images in training dataset is 11837 in which 6028 are valid samples and 5809 are invalid samples. The test dataset contains 6288 samples with 3330 valid samples and 2958 invalid samples. Both training and test data are almost balanced between the number of valid and invalid samples, but the number of images from each nut is not the same. Among the 11837 training images, 2100 images of part 1 are captured in the factory environment and all other images are captured in office environment. In office conditions, the brightness is varied between +3000 and -3000 μs of the normal exposure conditions whereas in factory condition the

brightness varies between +20000 to -20000 μs of the normal exposure conditions. An additional 6 images of the test nuts is used to retrain the pretrained model. The scale images contain 3 valid and 3 invalid samples.

Use case 2 mainly aims to test whether the model used to solve use case 1 will be able to classify the nuts in a completely new part (part 3 in Figure 4.11) after scaling to use case 2. Therefore, all the acquired image for this use case are test images. The total number of test images are 2500. It contains 970 valid images and 1530 invalid images. 10 images of the nuts with 5 valid images and 5 invalid images are used in pretraining the model to scale the baseline model to use case 2. The size of the nuts ID:1, ID:4, ID:5 and ID:6 are too large and they do not fit in 224x224 crop. Therefore, 400x400 crop of these nuts are obtained which are resized to 224x224 crop during training and inference. In order to balance the size between all the nuts, ID:2 and ID:3 are cropped at 256x256 and are further resized to 224x224 for training and inference. All the images of part 3 are captured in the office environment with +10000 and -10000 μs of the normal exposure.

Use case 3 includes the images of part 1 in factory condition as shown in Figure 4.11. All the nuts are cropped at 224x224 pixels. The dataset contains 22052 training samples and 4358 test samples. The training dataset is imbalanced with 17480 valid images and 4572 invalid images whereas as test dataset contain 2561 valid images and 2011 invalid images. 10 images of the test nuts are used to scale the baseline model to use case 3 with 5 valid and 5 invalid images. The images in factory are captured with +20000 and -20000 μs of the normal exposure.

5.4 Architecture and Training Details

As mentioned in Section 4.1, we have two main architectures proposed in our solution. This section first discusses the data augmentation and training details common for training the baseline models in both the architectures, followed by the training details required to retrain architecture-1 on three use cases along with their learning curves. Different feature extractor networks have been experimented with before arriving at the proposed network in architecture 1. As a proof to show that architecture-1 is the best performing for our dataset, the learning curves of the experimental networks for use case 1 are also included here. This is followed by training details and learning curves for architecture-2.

5.4.1 Data Augmentation

To reduce the model overfitting and increase it's performance on validation and test dataset, data augmentation is used. Irrespective of the architecture and use case, same augmentation method is used in all our experiments. Data augmentation is performed in such a way that same augmentation is applied for both test image and reference image. The augmentation methods used are rotating a nut by $+/-5°$, $+/-10°$, $+/-12°$ and $+/-15°$, flipping the nut horizontally and vertically. One augmentation is chosen from the above methods and is applied for the image pairs.

5.4.2 Training Details for Baseline Models

Two baseline models, one for office lighting conditions and the other for factory lighting conditions, are trained for both architecture-1 and architecture-2. When the model needs retraining to scale to new test nuts, the baseline model is retrained with 6 or 10 images of test nuts. The training details of the baseline model is same for both the architecture and different use cases. AdamW [26] with weight decay of 0.01 is used as optimizer. The batch size is set to 32 and learning rate of 5×10^{-5} is used in training. The model is trained for 50 epochs. Since the train and test data belong to different distribution, 20% of the test data is used as validation data. As a data preprocess method, the images are standardized to mean and standard deviation of the training images. The test data is also standardized on training images to avoid any knowledge leak of the test data during training. For both valid and invalid nuts, a randomly chosen valid nut of the same nut is used as reference. When the model is inputted with an invalid pair, it is trained to predict a value close to 0 and for valid pair a value close to 1. The problem is a binary classification as the model needs to predict if the nut is valid or invalid. Therefore the model is trained with binary cross-entropy loss.

5.4.3 Architecture-1 and Use Case 1

The baseline model is trained on training images of the use case 1. The model consist of 354k trainable parameters. The model is trained for 50 epochs and the parameters are saved at every epoch. After analysing the model performance on the validation data, the parameters at the epoch where the validation loss is the lowest are considered as the optimal model parameters. The learning curve for use case 1 is plotted in the Figure 5.2. The deviation between training curve and validation curve is mainly because of the difference between training and test data. At epoch 40, it can be seen that the validation loss is lowest and accuracy is highest. When the baseline model is tested on the unseen test nuts at this epoch, the accuracy is 97.51%. To improve this performance, we scale the model to test images. This is done by loading the baseline model with weights at epoch 40 and scaling to identify incorrectly placed nuts of part 2 by retraining the model with 6 images (3 valid and 3 invalid) of each test nuts for 50 epochs (less than 1 minute). The model is retrained in a similar approach with the same hyperparameters as baseline model. Since the data size of the scale images is 30 (6 images from 5 test nuts), only the batch size is changed to 8 instead of 32. To prove that the results are statistically significant an average of three runs with different seed values are reported. Figure 5.3 represents the learning curves of three average runs. The solid line is the mean of the three runs and the shaded region is the min and max range on which aggregation is performed. As the baseline model itself has high accuracy on the test data, pretraining of the model on test data will have low starting validation loss and and high validation accuracy.

To analyse the model's performance on different training nuts, we have experimented by adding the nuts reserved for testing in training, and testing the model on the remaining test nuts. Initially ID:12 and ID:14 of part 2 in Figure 4.10 is moved from testing to training and the trained model is scaled to ID:8, ID:9 and ID:13 with 6 scale images each. The loss curve and accuracy curve for scaling the baseline to test nuts is shown in Figure 5.4. The model classifies valid and invalid nuts of test data with 99.47% with loss 0.0646.

In the next stage, all the nuts from part 1 and part 2 are included in training but only ID:8 of part 2 is in testing. This test is conducted to validate how many different types of nuts should be included in the training so that the model can be tested on a completely different nut. The learning curve of this

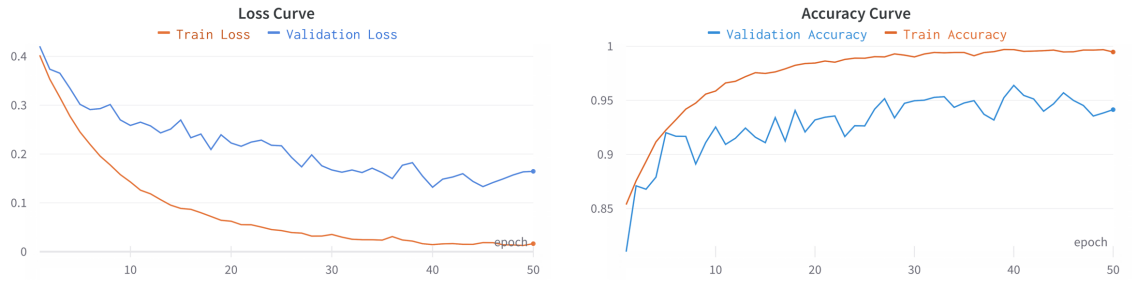
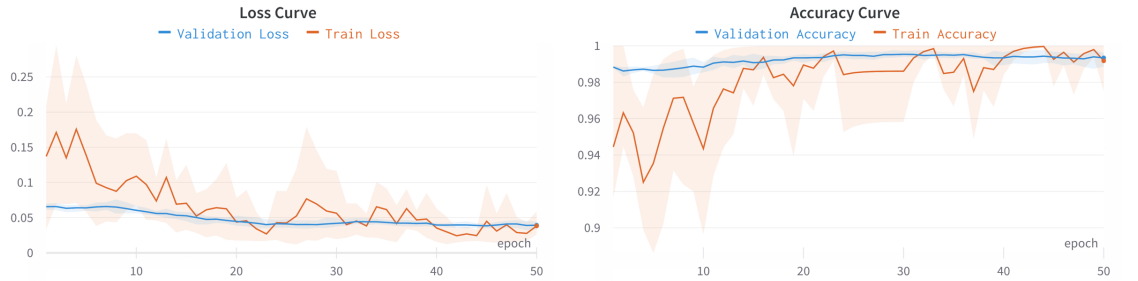


Figure 5.2: Learning curve for training the baseline model of architecture-1 and use case 1. **Left**: Loss curve of training and validation plotted against number of epochs. **Right**: Accuracy curve of training and validation plotted against number of epochs.



42

Figure 5.3: Learning curve for retraining the baseline model with 6 images of each test nut for use case 1. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation.

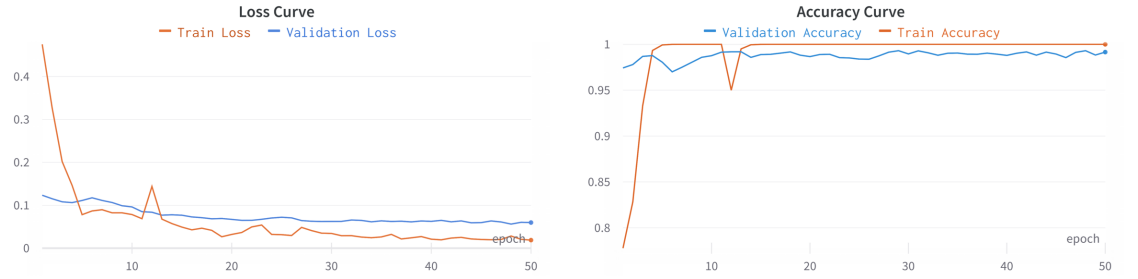


Figure 5.4: Learning curve for use case 1 by adding ID:12 and ID:14 in training. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation.

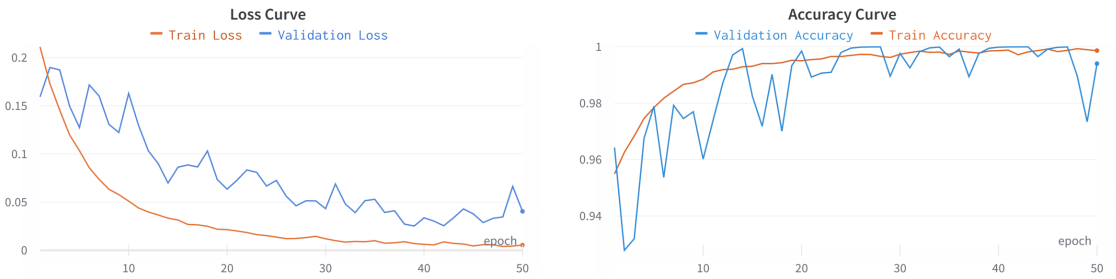


Figure 5.5: Learning curve for use case 1 by adding all the nuts from part 1 and part 2 except ID:8 in training. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation.

experiment in shown in the Figure 5.5. As it can be observed in the accuracy plot, the validation accuracy at 46^{th} epoch is 99.81%. This experiment proves that with different categories of training nuts, the model classifies the unseen test nut even without retraining the pretrained model with 6 images of test nut. The test accuracy of ID:8 is 99.86% with loss of 0.05618.

5.4.4 Architecture-1 and Use Case 2

The baseline model trained for use case 1 is pretrained on 10 images from each nut (5 valid and 5 invalid) with a batch size of 8, training duration of approximately 1 minute and with hyperparameters of the baseline model. The mean accuracy and loss curve of three runs is shown as a solid line in Figure 5.6 and the shaded region indicate the min and max range of the runs on which mean is calculated. The main difference from the previous use case is, during retraining the network, the images are standardaized to their mean and standard deviation values instead of mean and standard deviation of the scaling images as the brightness of the dataset varies greatly. It can be seen that train loss and validation loss decrease together which indicate the model is not overfitting on the training data. In contrast to use case 1, the images in this case have large variations in test nuts compared to training. Therefore, the loss at which retraining starts is higher and accuracy is lower.

5.4.5 Architecture-1 and Use Case 3

The aim of this use case is to validate if the proposed solution holds with the change in environment conditions. As the dataset had large variation in the lighting condition, a separate baseline model is trained for use case 3. The model trained to solve use case 1 is loaded as initial weight and is retrained with 22052 training images. The baseline model is trained on 5 nuts of part 1 and tested on the 2 unseen nuts. To improve the performance on the unseen nuts, 10 images with 5 valid and 5 invalid samples are used to retrain the model for training duration of less than 1 minute. To retrain the model with 20 scale images (10 images from each nut) a batch size of 4 is used and all other hyperparameters remains the same as the baseline model. As in the previous use cases, the learning curve of three runs with different seed is plotted in the Figure 5.7 for retraining the model with scale images.

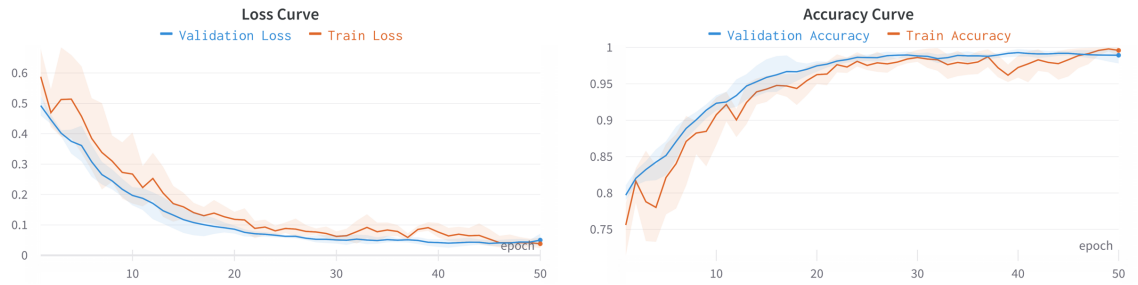


Figure 5.6: Learning curve for retraining the baseline model trained on use case 1 with 10 images from each test nut of use case 2. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation.

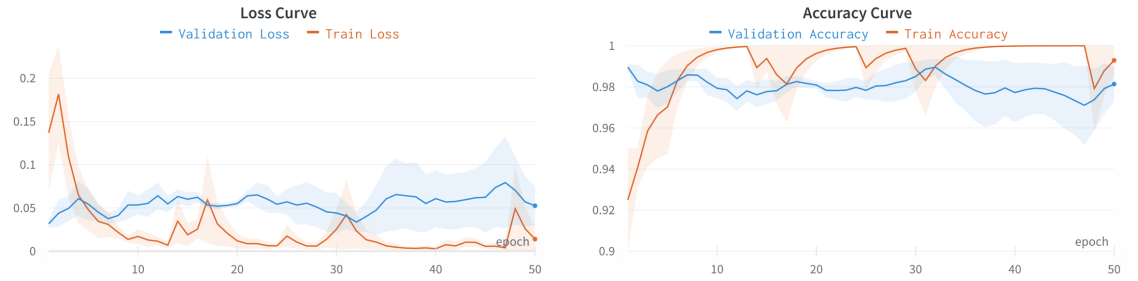


Figure 5.7: Learning curve for retraining the baseline model trained on training nuts from factory dataset with 10 images from each test nut of use case 3. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

5.4.6 Experiments on Feature Extractor of Architecture-1

The feature extractor in architecture-1 is developed from scratch and is not based on any standard architecture. Before concluding the proposed architecture is the best performing feature extractor, experiments have been conducted on different number of CNN layers, number of filters and ordering of the layers. In this section, three results are provided: one with a lower number of learnable parameters, one with a higher number of learnable parameters, and the last one with a different layer ordering than the proposed solution.

Lesser Complex Feature Extractor

The experimentation started with this extractor in twin network. As it can be seen in the Figure 5.8, in contrast to architecture-1, this feature extractor not only has only three convolutional layers, but number of filters (depth of CNN) is also less. Thus the total number of learnable parameters of the model is reduced to 25081. An important observation from the learning curve in Figure 5.9 is, There is very little deviation between training and validation curve, thus implying there is no overfitting in the model. But another significant information is that the learning saturates around

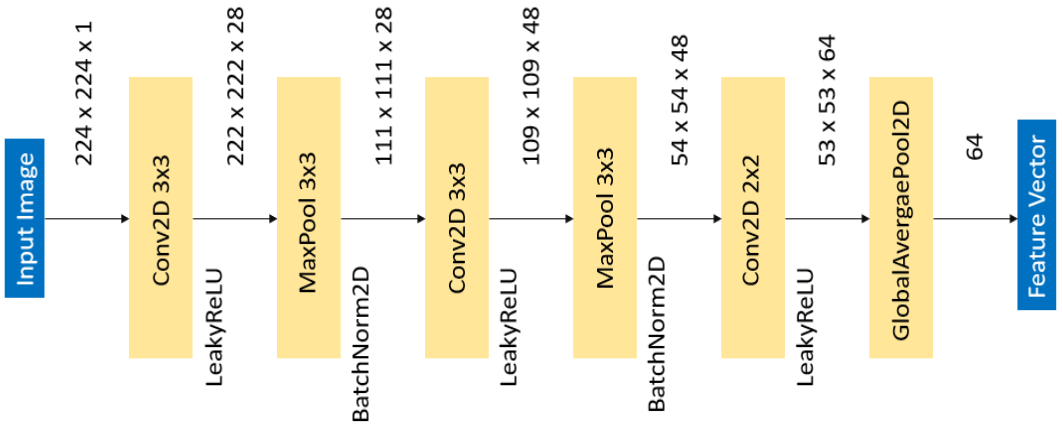


Figure 5.8: Feature extractor with less number of trainable parameters compared to feature extractor proposed in architecture-1

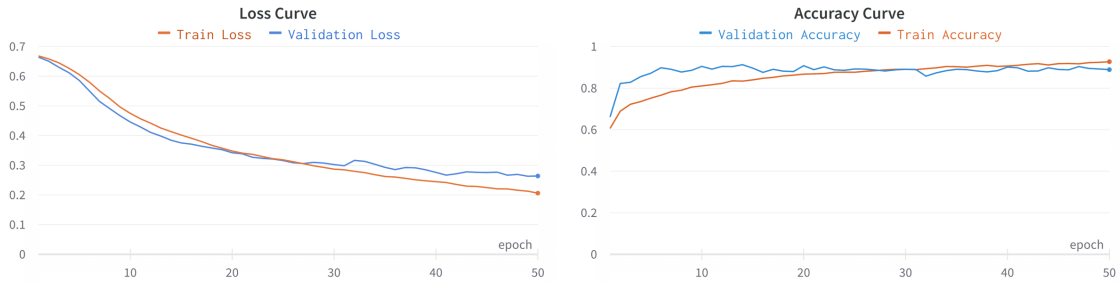


Figure 5.9: Learning curve for network architecture where feature extractor has less parameters in comparison to architecture-1. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

90%. This indicates the model is underfitting and does not have enough capacity to learn all the important features required to differentiate between valid and invalid nuts. This promoted to design a model with higher complexity.

More Complex Feature Extractor

Due to underfitting problem of feature extractor in Figure 5.8, the network in architecture-1 with increased number of filters and convolutional layers was designed. As the performance on test data increased with increase in number of learnable parameters, the network was further improved to check if further increase in model complexity would result in better performance than architecture-1. The network shown in Figure 5.10, have more number of filters compared to the proposed network. The learnable parameter is increased to 1.26M. The loss and accuracy curve of this model is plotted in the Figure 5.11. The training loss is decreased to an almost zero and accuracy reaches nearly 100% with increase in number of epochs. Therefore there is clearly no problem of underfitting as in architecture 5.8. But the network faces generalizabilty problem due memorization of the training data. In contrast to architecture-1 whose validation loss is less than 0.15 and accuracy is higher than

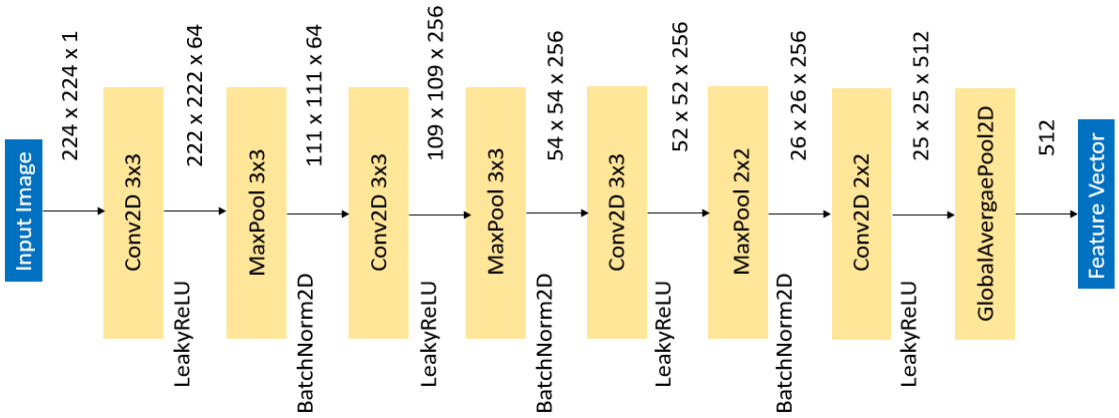


Figure 5.10: Feature extractor with more number of trainable parameters compared to feature extractor proposed in architecture-1

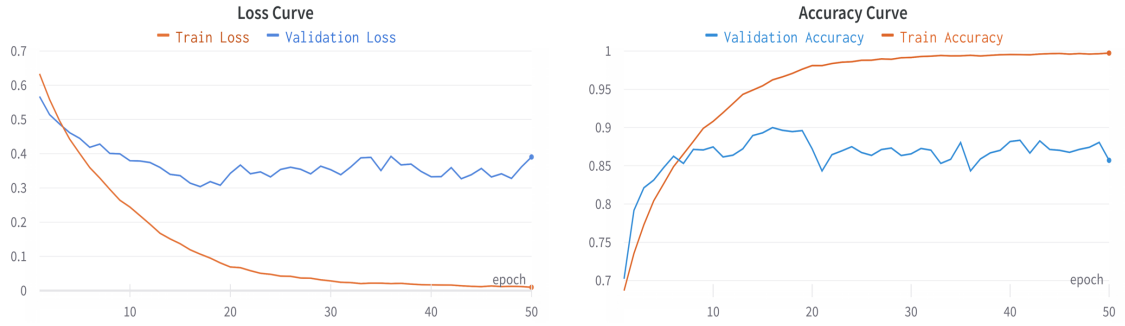


Figure 5.11: Learning curve for network architecture where feature extractor has more number of channels at deeper layers and is more complex in comparison to architecture-1. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

96%, the loss in this model is higher than 0.3 and accuracy is below 90%. This results indicated that architecture-1 had better performance on the unseen data compared to network with either lesser or more complex feature extractor.

Feature Extractor with Different Order of the Layers

According to Stanford course materials [41], stacking two convolutional layers and then performing max pool could result in better performance as the convolutional layers learn more complex features before pooling operation. Hence, feature extractor as described in Figure 5.12, is designed and trained to validate if this ordering could result in performance better than architecture-1. Unfortunately, for our use case, since the feature extractor is not very deep, this architecture did not improve the test accuracy compared to performance of architecture-1. The learning curves are shown in Figure 5.13. The total trainable parameters are 354k which is similar to architecture-1. This model also resulted in overfitting of the training data as the subsampling is reduced.

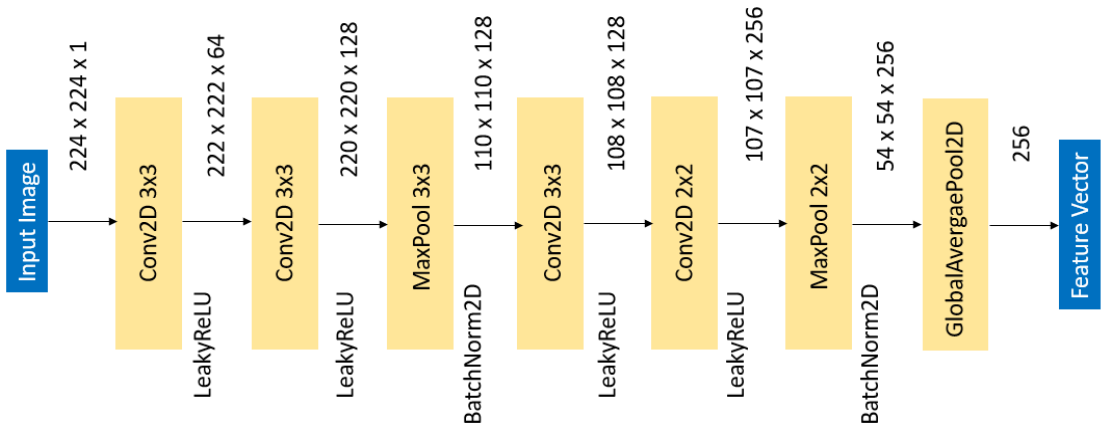


Figure 5.12: Feature extractor with different layer ordering compared to architecture-1 where two convolutional layer is followed by subsampling layer

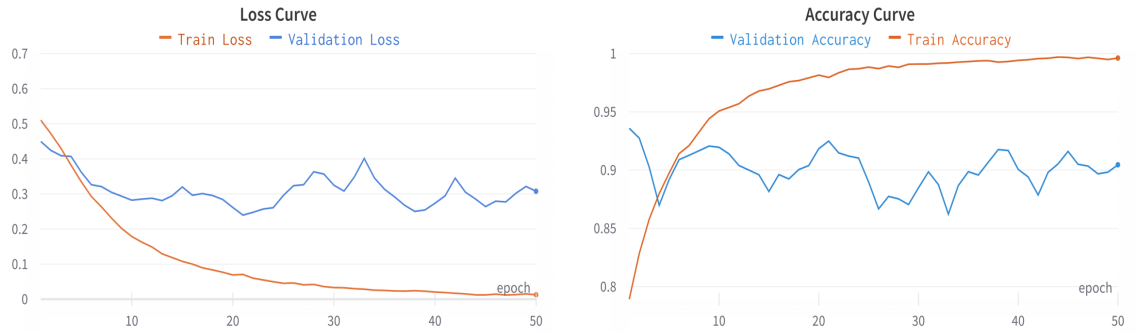


Figure 5.13: Feature extractor with different layer order compared to architecture-1. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

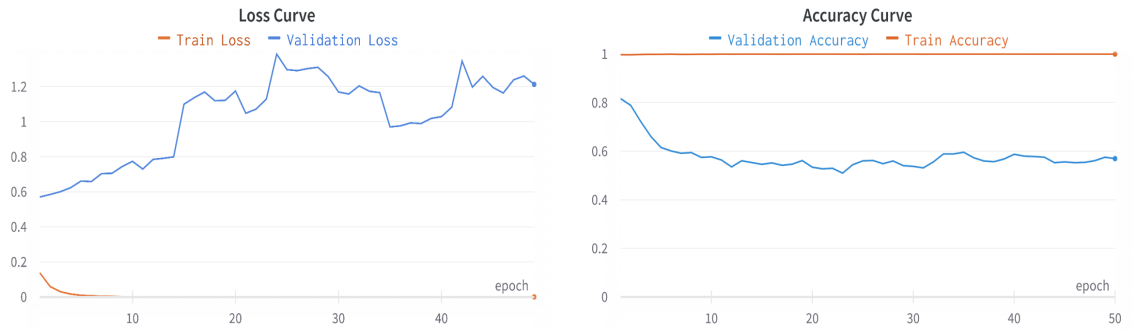


Figure 5.14: Learning curve of architecture-2 baseline model for use case 1. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

Figure 5.15: Learning curve of architecture-2 for retraining the baseline model on use case 2. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

5.4.7 Architecture-2 and Use Case 1

The baseline model is trained on training nuts of use case 1 similar to architecture-1. The feature extractor EfficientNet-B0 has 5.3M trainable parameters. Due to increased model complexity, the architecture-2 exhibits large overfitting to the training data of use case 1 resulting in poor performance on validation data. The learning curve plotted in Figure 5.14, proves that model overfits greatly to training data right from first epoch.

5.4.8 Architecture-2 and Use Case 2

The baseline model of use case 1 is loaded and retrained on the test nuts of use case 2 to validate the model performance on different size of the nuts and different orientation of the nut. As the architecture-2 based model trained on use case 1 was overfitting to training data, retraining this model on test nuts of use case 2 did not improve the performance and can be noticed in the learning curve plotted in the Figure 5.15

5.4.9 Architecture-2 and Use Case 3

The EffecientNet-B0 pretrained on Imagenet is used as feature extractor. The baseline model is obtained by retraining the EffecientNet-B0 on training images (22052) of use case 3. Architecture-1 requires 10 images (5 valid + 5 invalid) of the new test nuts to result in accuracy higher than 99%. But architecture-2 attains better results only with scale 6 images (3 valid + 3 invalid) of test nuts. There are two test nuts - Id:2 and ID:7 (Figure 4.11 Right), so six images from each nut resulted in 12 scale images in total. Due to this reason, the retraining of the model is done with batch size of 4. Learning curve for model retraining is shown in the Figure 5.16. When the baseline model is tested on the test nuts without retraining, accuracy obtained is 98.5% with a test loss of 0.0526. But to further improve the prediction accuracy over 99% retraining is done. An interesting fact to be noticed is the variation in the training curve. As the batch size is small, even one incorrect prediction will lead to large fluctuation in the accuracy and loss. For example, if only one sample is

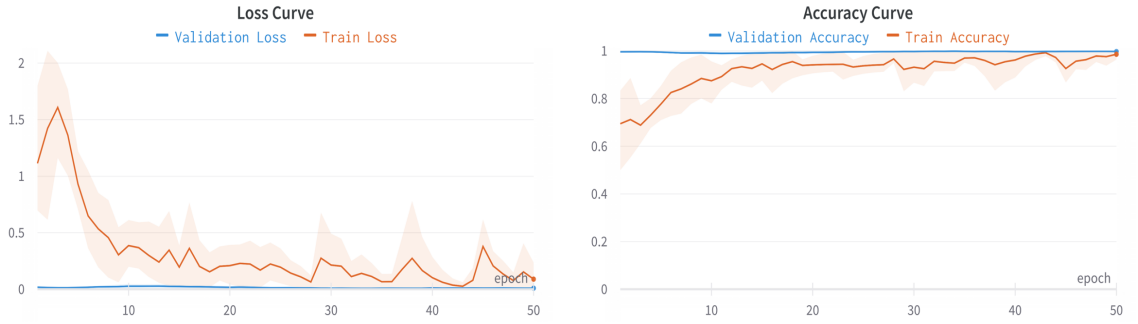


Figure 5.16: Learning curve for retraining architecture-2 with test nuts of use case 3 in factory condition. **Left**: Loss curve of training and validation. **Right**: Accuracy curve of training and validation

classified incorrectly, the accuracy is reduced to 3/4 = 75%. The retraining of the baseline model which already had better accuracy resulted in a model which predicted the incorrect nuts with production grade results which is discussed in the next section.

6 Evaluation of Solution

As discussed in the earlier sections, to support our proposition that Siamese neural network have the ability to classify unseen data and can be used to perform quality analysis of different tasks with less efforts, we have divided our test cases into three major ones. In the previous chapter, we have plotted the learning curves and examined the generalizability of the model on these three use cases. This section will focus on providing numerical proof to our thesis proposition. Most commonly used metric for classification model is accuracy and is defined as number of correct predictions divided by total number of samples. But, the dataset used in our experiments is not balanced in all the three use cases. Therefore we are reporting accuracy and confusion matrix to evaluate if the model is capable to predict samples from both the classes. For statistical evaluation, we have reported the average values of three runs with different seeds.

6.1 Architecture-1

Use Case 1

The test dataset is a balanced dataset i.e. it contains approximately 50% of valid images and 50% of invalid images. Therefore accuracy can be considered as a reliable metric to evaluate the model which is reported in the Table 6.1. The accuracy is the performance of the model retrained on 6 images (3 valid and 3 invalid) of each test nuts for less than one minute. As it can be seen, the loss is 0.0452 which tells the model is confident in classifying the valid and invalid images with the a high accuracy of 99.47%. Confusion matrix indicating the number of correct predictions (valid and predicted valid + invalid and predicted invalid) and incorrect predictions (valid but detected invalid + invalid but detected valid) is recorded in the Table 6.2. The model predicts only 2 valid nuts as invalid and 30 invalid nuts as valid. But is successful is classifying more than 99% of the samples correctly.

Use Case 2

Use Case	Number of Images	Accuracy	Loss
1	6	99.47%	0.0452
2	6	97.61%	0.0843
	10	99.16%	0.0498
3	6	98.68%	0.0394
	10	99.08%	0.0308

Table 6.1: Accuracy of architecture-1 for three different use cases. For use case 2 and use case 3 training the model with 10 images results in better accuracy than with 6 images.

Actual \ Predicted	Valid	Invalid
Valid	2699	2
Invalid	30	2323

Table 6.2: Confusion matrix of retraining the baseline model of architecture-1 for use case 1 with 6 images of each test nut.

Actual \ Predicted	Valid	Invalid
Valid	787	15
Invalid	0	1206

Table 6.3: Confusion matrix of retraining the baseline model of architecture-1 for use case 2 with 10 scale images.

The average accuracy of the model on the test data is shown in the Table 6.1. The retraining of the baseline model is done with 6 images and 10 images of the test nuts. The model retrained on 10 images has higher performance compared to the other. As the difference between train nuts from part 1 and part 2 and test nuts from part 3 is huge, the model requires more images to classify the nuts with high accuracy. The test dataset is not balanced therefore a careful analysis of confusion matrix Table 6.3 is conducted. None of the invalid images are predicted as valid but 15 valid samples are predicted as invalid. The high accuracy on the test set supports our proposition that the designed model is capable to perform quality analysis of a new part with nuts of different size and orientation with very less training images and training duration.

Use Case 3

The average accuracy and test loss of three runs are recorded in the Table 6.1. As it can be seen, unlike use case 1 which has high accuracy only with 6 images, this case also requires 10 images to have a comparable performance. The baseline model retrained with 6 images of test nuts have an average accuracy of 98.68%. In order to improve the accuracy above 99%, retraining is done with 10 images. Due to extreme lighting conditions in the factory, the model requires more images to classify the nuts correctly. Table 6.4 shows the confusion matrix for use case 3. Though the model classifies 16 valid images as invalid and 20 invalid images as valid, it is successful in classifying most of the images to right class.

6.2 Architecture-2

As discussed in the previous chapter, architecture-2 does not generalize to test data in use case 1 and use case 2. With EffecientNet-B0 as feature extractor, which has 5.3M learnable parameters, the model learns more specific features to classify a nut. Therefore the feature vectors in the embedding space are very specific to each nut rather than generic features which are required to differentiate

Actual \ Predicted	Valid	Invalid
Valid	2046	16
Invalid	20	1403

Table 6.4: Confusion matrix of retraining the baseline model of architecture-1 for use case 3 with 10 scale images.

Use Case	Number of Images	Accuracy	Loss
3	6	99.68%	0.0129
	10	99.84%	0.0099

Table 6.5: Accuracy of architecture-2 for use case 3. Average accuracy and test loss is reported for both 6 scale images and ten scale images.

between the test image and the reference image. This results in the model overfitting to the training data. As the test nuts in use case 1 and use case 2 vastly differ from training data, the model has poor performance on the test nuts.

Use Case 3

As observed in the previous section, architecture-1 requires 10 scale images for use case 3 in order to classify the nuts with higher confidence and accuracy. Architecture-2 has exceptional performance on use case 3. With 6 scale images, the model classifies most of the images to correct class with high confidence. The average accuracy of the architecture-2 with 6 scale images is reported in the Table 6.5. A comparison is also made with the model trained with 10 scale images. With 10 images the prediction accuracy is even more higher and the model is also more confident as the test loss in the order of 10^{-3}. Confusion matrix for the model trained with 6 scale images is reported in the Table 6.6. As it can be observed, there are 16 valid images which are predicted as invalid and all the other images are classified to the right class. The number of training samples used to train the baseline model of use case 3 is double compared to the previous two use cases. This results in less overfitting to the training data. The test nuts contain the same nuts as training but are positioned at different places compared to the nuts in training. The accuracy and confusion matrix prove that, architecture-2 is very efficient in classifying the test nuts of use case 3 with extreme lighting conditions and data distribution similar to train.

Actual \ Predicted	**Valid**	**Invalid**
Valid	2042	16
Invalid	0	1426

Table 6.6: Confusion matrix of retraining the baseline model of architecture-2 for use case 3 with 6 scale images.

7 Conclusions and Future Works

7.1 Conclusion

In this thesis, we have proposed a solution which significantly reduces the efforts of retraining a model to perform quality analysis of new parts. We have adressed the scalability issues of the connventional CNNs by designing a siamese neural network classifier. Two approaches have been proposed to scale a baseline model trained to detect the misplaced welding nuts of one part to different parts with unseen nuts. When compared to the model with EffecientNet-B0 as feature extractor, we have proved that the model with a custom convolutional feature extractor scales to nuts of different shapes, sizes, orientations, and lighting conditions more efficiently. The training of the baseline model in factory lighting requires more data to learn to distinguish between valid and invalid samples than the model trained in office environment. When the test nuts differ significantly compared to the nuts the baseline model is trained on, the rescaling requires 10 images. Otherwise, retraining the model with 6 images will result in test accuracy over 99%. In detecting invalid cases, model predicts few false positives (invalid but predicted valid) in classifying incorrect nuts and flipped nuts but will successfully identify the missing nuts in all the use cases as incorrect sample. In use case 3, we have trained the baseline on imbalanced training data with 80% of valid samples and 20% invalid samples. The test accuracy of 99.08% of architecture-1 and 99.68% of architecture-2 proves that the system can handle class imbalance. The architecture needed no redesigning with change in number of test nuts in different use cases.

7.2 Limitations

The model is evaluated on only one part in factory lighting condition. In order to deploy the model in production, more validation needs to be performed in terms of it's adaptability to factory environment and possible errors during manufacturing. The user who captures the scale images to retrain the baseline model should have the knowledge of what invalid cases can occur in the production system. Our approach compares the test image with a reference image in order to classify a nut as valid or invalid. Since the position at which the image of the part is captured for inference keep drifting in the production line, the reference image needs to be updated frequently.

7.3 Future Works

Manufacturing industries will have some common nuts used in production. We have demonstrated that with different categories of nuts in training, the proposed model can scale to completely different nut even without retraining. A dataset should be collected with regularly used nuts and

with possible lighting conditions. Instead of training the baseline with few nuts, it should be trained on this dataset. As a result, the system would be more dynamic as the baseline will have the knowledge of commonly used nuts and their possible errors. More evaluation needs to be conducted if the exceptional performance of architecture-2 on use case 3 is only due to similarity of the data distribution between train and test compared to other use cases or if the extreme lighting condition in factory needs more parameters to learn the variance in the data.

Bibliography

[1] Albumentations. *What is image augmentation and how it can improve the performance of deep neural networks*. URL: https://albumentations.ai/docs/introduction/image_augmentation/ (cit. on p. 28).

[2] N. M. Alie, M. S. Karis, G.-J. Wong, M. B. Bahar, M. Sulaiman, M. M. Ibrahim, A. F. Z. Abidin. "Quality checking and inspection based on machine vision technique to determine tolerance-value using single ceramic cup". In: *ARPN Journal of Engineering and Applied Sciences* 12.8 (2017), pp. 2737–2742 (cit. on p. 15).

[3] AutoVersed. *The Biggest Automotive Recalls Of The Past Decade*. URL: https://autoversed.com/the-biggest-automotive-recalls-of-the-past-decade/ (cit. on p. 15).

[4] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, R. Shah. "Signature verification using a 'siamese' time delay neural network". In: *Advances in neural information processing systems* 6 (1993) (cit. on pp. 31, 35).

[5] D. Chicco. "Siamese neural networks: An overview". In: *Artificial Neural Networks* (2021), pp. 73–94 (cit. on p. 31).

[6] S. Chopra, R. Hadsell, Y. LeCun. "Learning a similarity metric discriminatively, with application to face verification". In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546 (cit. on p. 31).

[7] CodeCrucks. *Divide and Conquer Strategy for Problem Solving*. URL: https://codecrucks.com/divide-and-conquer/ (cit. on p. 43).

[8] R. DABHI. *Casting product image data for quality inspection*. URL: https://www.kaggle.com/datasets/ravirajsinh45/real-life-industrial-dataset-of-casting-product (cit. on p. 35).

[9] L. Deng. "The mnist database of handwritten digit images for machine learning research [best of the web]". In: *IEEE signal processing magazine* 29.6 (2012), pp. 141–142 (cit. on p. 27).

[10] R. Gencay, M. Qi. "Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging". In: *IEEE Transactions on Neural Networks* 12.4 (2001), pp. 726–734. DOI: 10.1109/72.935086 (cit. on p. 27).

[11] R. Gómez. *Understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss and all those confusing names*. URL: https://gombru.github.io/2019/04/03/ranking_loss/ (cit. on pp. 32, 33).

[12] I. Goodfellow, Y. Bengio, A. Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 20, 24).

[13] *GV-5280CP-CAMERA FAMILY*. URL: https://en.ids-imaging.com/store/gv-5280cp.html (cit. on p. 49).

[14] *GV-5280CP-OVERVIEW*. URL: https://en.ids-imaging.com/store/gv-5280cp.html (cit. on p. 49).

[15] K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 36).

[16] J. Hu, L. Shen, G. Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141 (cit. on p. 36).

[17] IBMCloudEducation. *Convolutional Neural Networks*. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks/ (cit. on p. 29).

[18] S. Ioffe, C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456 (cit. on p. 40).

[19] H. Jabbar, R. Z. Khan. "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)". In: *Computer Science, Communication and Instrumentation Devices* 70 (2015) (cit. on p. 24).

[20] A. Kathuria. *Intro to Optimization in Deep Learning: Busting the Myth About Batch Normalization*. URL: https://blog.paperspace.com/busting-the-myths-about-batch-normalization/ (cit. on p. 40).

[21] J. Khan. *Everything you need to know about Visual Inspection with AI*. 2021. URL: https://nanonets.com/blog/ai-visual-inspection/ (cit. on p. 15).

[22] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, D. Krishnan. "Supervised contrastive learning". In: *arXiv preprint arXiv:2004.11362* (2020) (cit. on p. 44).

[23] M. S. Kim, T. Park, P. Park. "Classification of Steel Surface Defect Using Convolutional Neural Network with Few Images". In: *2019 12th Asian Control Conference (ASCC)*. 2019, pp. 1398–1401 (cit. on pp. 16, 35, 37).

[24] G. Koch, R. Zemel, R. Salakhutdinov, et al. "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. Vol. 2. Lille. 2015 (cit. on pp. 31, 35).

[25] B. Lake, R. Salakhutdinov, J. Gross, J. Tenenbaum. "One shot learning of simple visual concepts". In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 33. 33. 2011 (cit. on p. 35).

[26] I. Loshchilov, F. Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017) (cit. on p. 52).

[27] J. McCarthy. "What is artificial intelligence?" In: (1998) (cit. on p. 19).

[28] D. Mishkin. *caffenet-benchmark*. URL: https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md (cit. on p. 40).

[29] D. Monica. *Understanding Overfitting and Underfitting In Layman Terms*. URL: https://medium.com/mlearning-ai/understanding-overfitting-and-underfitting-in-layman-terms-e4c82a28e2d2 (cit. on p. 25).

[30] A. M. Nagy, L. Czúni. "Detecting Object Defects with Fusioning Convolutional Siamese Neural Networks." In: 2021 (cit. on pp. 16, 35, 36).

[31] M. A. Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, 2015 (cit. on p. 20).

[32] C. Nwankpa, W. Ijomah, A. Gachagan, S. Marshall. "Activation functions: Comparison of trends in practice and research for deep learning". In: *arXiv preprint arXiv:1811.03378* (2018) (cit. on p. 21).

[33] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, J. Walsh. "Deep learning vs. traditional computer vision". In: *Science and Information Conference*. Springer. 2019, pp. 128–144 (cit. on p. 15).

[34] S. Ruder. *Highlights of NIPS 2016: Adversarial learning, Meta-learning, and more*. URL: https://ruder.io/highlights-nips-2016/index.html#thenutsandboltsofmachinelearning (cit. on p. 42).

[35] S. Ruder. *Transfer Learning - Machine Learning's Next Frontier*. URL: https://ruder.io/transfer-learning/index.html#whatistransferlearning (cit. on p. 42).

[36] D. E. Rumelhart, G. E. Hinton, R. J. Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 23).

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252 (cit. on pp. 15, 35, 42).

[38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4510–4520 (cit. on p. 42).

[39] F. Schroff, D. Kalenichenko, J. Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823 (cit. on p. 36).

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958 (cit. on p. 26).

[41] Stanford. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/convolutional-networks/ (cit. on pp. 21, 30, 57).

[42] Stanford. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: https://cs231n.github.io/neural-networks-1/ (cit. on p. 40).

[43] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9 (cit. on p. 36).

[44] I. Tabian, H. Fu, Z. Sharif Khodaei. "A convolutional neural network for impact detection and characterization of complex composite structures". In: *Sensors* 19.22 (2019), p. 4933 (cit. on p. 28).

[45] M. Tan. *EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling*. URL: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html (cit. on pp. 42, 43).

[46] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le. "Mnasnet: Platform-aware neural architecture search for mobile". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2820–2828 (cit. on p. 42).

[47] M. Tan, Q. Le. “Efficientnet: Rethinking model scaling for convolutional neural networks”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6105–6114 (cit. on pp. 15, 42, 43).

[48] S. Tay, L. Te Chuan, A. Aziati, A. N. A. Ahmad. “An Overview of Industry 4.0: Definition, Components, and Government Initiatives”. In: *Journal of Advanced Research in Dynamical and Control Systems* 10 (Dec. 2018), p. 14 (cit. on p. 15).

[49] K. Weiss, T. M. Khoshgoftaar, D. Wang. “A survey of transfer learning”. In: *Journal of Big data* 3.1 (2016), pp. 1–40 (cit. on p. 42).

[50] J. Yu, G. Xie, M. Li, X. Hao. “Retrieval of family members using siamese neural network”. In: *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*. IEEE. 2020, pp. 882–886 (cit. on p. 36).

[51] M. D. Zeiler, R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833 (cit. on p. 36).

[52] D. Zhang, G. Lu. “Evaluation of similarity measurement for image retrieval”. In: *International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003*. Vol. 2. IEEE. 2003, pp. 928–931 (cit. on p. 40).

All links were last followed on May 07, 2022.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature