Tampere University

Joonas Hyvärinen

AUTOMATING CAD ASSEMBLY CONSTRAINTS USING SUPERVISED DEEP LEARNING

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiner: Professor Iñigo Flores Ituarte (PhD)
July 2021

ABSTRACT

Joonas Hyvärinen: Automating CAD assembly constraints using supervised deep learning
Master of Science Thesis
Tampere University
Master's Degree Programme in Mechanical Engineering
July 2021

Assembly constraints determine the relative position and allowed movements between CAD models in assemblies, which are an integral part of computer aided design. They comprise both the expertise of CAD designers and information on the purpose of the assembly. Automation of the assembly constraint adding process supports designers by decreasing the design time and enhancing the user experience. This thesis discusses and evaluates a method for automating assembly constraints using supervised deep learning and the convolutional neural network architecture. It investigates the most relevant information of a CAD model to maintain a real-time capable data presentation, as well as the requirements for dataset generation from pre-existing assemblies.

The most relevant information to be extracted from CAD models was investigated through literature review and empirical research. An adjacency graph-based approach was selected based on its benefits compared to other commonly used data presentations in CAD model classification. A method for converting the graph into a matrix presentation was discussed and validated. A data augmentation algorithm for cases in which exact normalization of graph data is difficult was tested to increase the classifier performance. The data extraction method was used on professionally designed CAD assemblies to collect datasets for training, validation and testing of the convolutional neural network model. To test the theory in a real environment, an initial software implementation was developed, and its real-time suitability validated as a part of the Vertex G4 mechanical CAD system.

The analysis showed a strong correlation between data quantity and classifier model performance. The results indicate that augmenting the collected data using the presented methods leads to superior performance compared to a baseline dataset. A classification accuracy of up to 95.46% was reached using the larger matrix presentation and the augmented dataset. Utilization of the presented augmentation method increases the classifier performance by up to 4.7% on average on the calculated performance metrics. Balancing the datasets by making the quantity of each constraint equal led to worse performance than non-augmented data especially in differentiating distance constraints from coincident constraints. Overall, it can be concluded that the presented method is adequate for assembly constraint classification and real-time CAD design workflow. The thesis advances the state of research in the CAD model related machine learning field.

Keywords: assembly constraint, deep learning, computer aided design, convolutional neural network, classification, machine learning

TIIVISTELMÄ

Joonas Hyvärinen: Kokoonpanoehtojen automatisointi ohjattua syväoppimista hyödyntäen
Diplomityö
Tampereen yliopisto
Konetekniikan DI-ohjelma
Heinäkuu 2021

Kokoonpanoehdot määrittävät CAD -mallien suhteellisen sijainnin ja sallitut keskinäiset liikkeet kokoonpanoissa, jotka ovat olennainen osa tietokoneavusteista suunnittelua. Ne sisältävät sekä suunnittelijoiden asiantuntemusta että tietoa kokoonpanon tarkoituksesta. Kokoonpanoehtojen asettamisprosessin automatisointi tukee suunnittelijoita vähentämällä suunnitteluaikaa ja parantamalla käyttäjäkokemusta. Tässä opinnäytetyössä käsitellään ja arvioidaan menetelmä kokoonpanoehtojen automatisoimiseksi hyödyntäen ohjattua syväoppimista ja konvoluutioneuroverkkoarkkitehtuuria. Työ tutkii CAD-mallien oleellisimpia tietoja reaaliaikaiseen toimintaan kykenevän datan muodolle sekä vaatimuksia jo olemassa olevien kokoonpanojen käytölle datajoukkojen luomisessa.

Merkityksellisimpiä CAD-malleista kerättäviä tietoja tutkittiin kirjallisuuskatsauksen ja empiirisen tutkimuksen avulla. Vierusgraafiin perustuva lähestymistapa valittiin sen hyötyjen perusteella muihin CAD-mallien luokittelussa yleisesti käytettyihin datan muotoihin verrattuna. Menetelmä graafin muuntamiseksi matriisiesitykseksi käsiteltiin ja validoitiin. Luokittelijan suorituskyvyn parantamiseksi testattiin datan lisäysalgoritmia tapauksissa, joissa graafimuotoisen datan tarkka normalisointi on vaikeaa. Datan keräysmenetelmää käytettiin ammattimaisesti suunnitelluille CAD-kokoonpanoille datan keräämiseksi konvoluutioneuroverkkomallin koulutusta, validointia ja testausta varten. Teorian testaamiseksi todellisessa ympäristössä kehitettiin alustava ohjelmistototeutus ja vahvistettiin sen reaaliaikainen soveltuvuus osana mekaanista CAD-järjestelmää, Vertex G4:ää.

Analyysi osoitti vahvan korrelaation datamäärän ja luokittelumallin suorituskyvyn välillä. Tulokset osoittavat, että kerätyn datan lisääminen esitetyillä menetelmillä johtaa parempaan suorituskykyyn verrattuna lähtötason datajoukkoon. Jopa 95,46%:n luokittelutarkkuus saavutettiin käyttämällä suurempaa matriisiesitystä ja laajennettua datajoukkoa. Esitetyn lisäysmenetelmän käyttö lisää suorituskykyä keskimäärin jopa 4,7% lasketuilla suorituskykymittareilla. Datajoukkojen tasapainottaminen tekemällä kunkin kokoonpanoehdon määrä yhtä suureksi johti huonompaan suorituskykyyn kuin alkuperäisellä datalla koulutettu malli. Etenkin mallin kyky erottaa etäisyysehtoja yhtenevyysehdoista oli huonompi. Kaiken kaikkiaan voidaan päätellä, että esitetty menetelmä on sopiva kokoonpanoehtojen luokittelulle ja reaaliaikaiselle CAD-suunnittelun työnkululle. Opinnäytetyö edistää tutkimustilaa CAD-malliin liittyvällä koneoppimisalalla.

Avainsanat: kokoonpanoehto, syväoppiminen, tietokoneavusteinen suunnittelu, konvoluutioneuroverkko, luokittelu, koneoppiminen

PREFACE

This Master of Science thesis was written as part of the master's programme at Tampere University. The thesis was commissioned by a Finnish software company, Vertex Systems. The writing process happened over the span of six months during the spring and early summer of 2021.

First, I would like to thank Timo Tulisalmi and Esa Mikkonen from Vertex Systems for the possibility to write the thesis, for following the writing process and for their assistance in the refinement of the thesis topic into its final form. I am also grateful for the freedom I was given to use my own creativity in the way the system was implemented.

Second, I want to express my gratitude to my thesis supervisors Iñigo Flores Ituarte and Hossein Mokhtarian from Tampere University for their support during the writing process. I am especially grateful for the outstanding atmosphere in our meetings and the ease of communication. It has been very pleasant working with you both.

Finally, I want to thank my friends for the peer support and the ability throw ideas and see what sticks. It has been immeasurably valuable for the thesis to reach its final shape.

Tampere, 5th July 2021

Joonas Hyvärinen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
BFS	Breadth First Search
CAD	Computer Aided Design
CNN	Convolutional Neural Network
DL	Deep Learning
DOF	Degrees of Freedom
ML	Machine Learning
NN	Neural Network
PWR	Part in Whole Retrieval
ReLU	Rectified Linear Unit
RPC	Remote Procedure Call
UI	User Interface

1. INTRODUCTION

Assemblies are an integral part of Computer Aided Design (CAD). Assemblies consist of CAD models as well as the information of the models' relationship in the form of assembly constraints. Assembly constraints comprise both the expertise of CAD designers and information on the purpose of the assembly. [1] However, applying assembly constraints is a time-consuming routine task in the CAD process, which additionally requires the designer to have proficiency in the CAD design. Automation of such tasks can support designers by decreasing the design time and enhancing the user experience [2].

Deep Learning (DL) is a powerful tool used to automate routine tasks in many fields, but the research in the CAD field focuses heavily on manufacturing feature classification, part type recognition and model retrieval [3]–[5]. While these applications have received large attention in the ML community in the recent years in studies such as [3]–[10], at the time of writing this thesis, there has been little to no work on assembly constraint classification. To explore the potential of DL in the automation of the assembly constraint applying process, it is important to develop and evaluate a methodology for assembly constraint classification.

The aim of this thesis is to present a novel assembly constraint automation method based on supervised deep learning and Convolutional Neural Networks (CNN), a widely used DL model in e.g. image classification and other CAD model related classification studies. Figure 1.1 presents the fundamental outline of the classifier system. The process begins with two CAD models with one selected geometric element from each, highlighted in orange. The elements go through the processing pipeline, which is connected to a CNN classifier. The classifier is presented as a black box model, which outputs the probability predictions of the assembly constraint type between the user-selected elements.

The thesis focuses heavily on the data collection and the processing steps, that convert the CAD models into a valid input to a CNN classifier, since using a solid CAD model directly is not feasible. Because the classifier system is developed to be associated with a mechanical CAD software, Vertex G4, and the company has access to professionally designed CAD assemblies, these pre-existing assemblies are used to collect the datasets required for the classifier creation. However, different constraints are not used equally during the design process, which is hypothesized to cause bias in the classifier perfor-

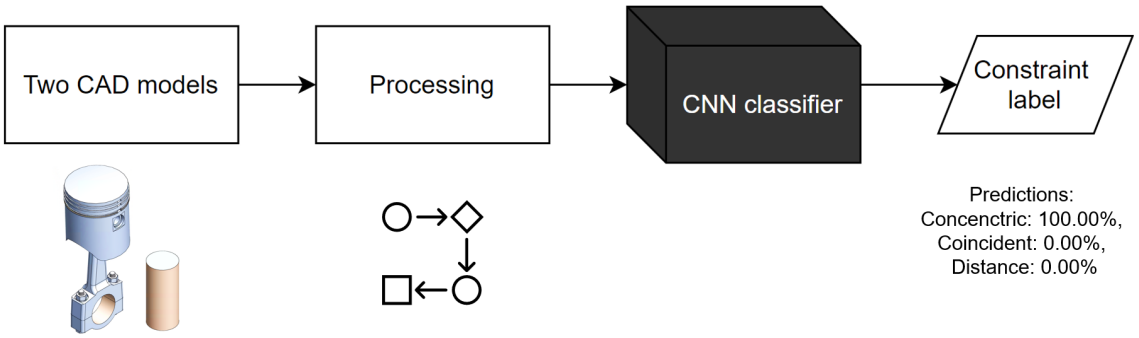


Figure 1.1. *Fundamental outline of the classifier system.*

mance. This is the second main topic the thesis will explore. Third, the whole pipeline presented in figure 1.1 needs to happen without major delays not to interrupt the CAD design workflow. For example, the CAD model classification approach used by Manda et al. [9] requires up to 15 s to classify a single model, which is inadequate for real-time CAD design. The thesis scope is focused on the assembly constraint classification based on the geometrical structure of the CAD model and thus other approaches are left out of the scope. While there exist other methods for CAD model related classification, this thesis focuses on the ones that are valid to be used with the CNN architecture. The thesis scope makes it impossible to discuss all the related techniques in-depth, and instead the aim is to present these concepts in an efficient and compact format.

The goal of the thesis is to answer the following research questions:

1. What is the most relevant information to be extracted from a CAD model for the purpose of training an assembly constraint classifier?
2. How can pre-existing CAD models be used to create datasets for CNN training, validation and testing and what challenges does this approach pose related to data bias?
3. How effectively can an assembly constraint classifier be incorporated to CAD design workflow?

Chapter 2 describes the fundamental basics required to understand the research topic and its importance. It also reviews the literature around CAD model related classification applications, due to the limited amount of literature around assembly constraint classification. Chapter 3 contains a method for CAD model data extraction, data processing and dataset generation. The chapter provides justifications for the selected data presentations and a method to increase data quantity algorithmically. The software system architecture, the CNN architecture, the DL model training process and the developed software implementation are described in chapter 4. Chapter 5 evaluates and discusses the obtained results and presents possible future improvements before the thesis is concluded in chapter 6.

2. BACKGROUND

2.1 Basic Concepts

2.1.1 Supervised Deep Learning

Machine Learning (ML) is a subsection of AI. Compared to traditional software systems, ML based systems do not implement logical operations explicitly. Instead, they recognize patterns in training data and generalize them to solve complicated problems [6]. Supervised learning is the most frequently used form of ML. Supervised learning can be described as learning from example, where during the training phase of the model the system is taught with data and ground truth pairs, which the system learns to generalize. Contrary to supervised learning, unsupervised learning systems are not trained using ground truth labeled data, which makes the main problem the identification of patterns in the data. [11]

Supervised DL is a subsection of ML that has reached state-of-the-art performance in recent years on multiple ML fields [4]. In supervised DL the system is trained through adjusting a large number of weights contained inside a deep Neural Network (NN). Deep NN structures are distinguished from older NNs through the lack of need for manual feature extraction and instead, they can operate on raw data [11].

Figure 2.1 shows a deep NN that consists of an input layer, three hidden layers and one output layer. The layer sizes and the number of hidden layers are dependant on the application, the data dimensionality and the size of the dataset [11]. The weight, bias and input values are left outside of the figure to make it clearer and are instead presented in figure 2.2. The network is fully connected, which means that each neuron in a layer is connected to each neuron in the following layer [12]. Each layer of a deep NN transforms the data into a higher abstraction level, which allows the network to learn complex functions [11].

Figure 2.2 depicts the role of a single neuron in a deep network. Each neuron sums its inputs based on the optimized weight value. The edges connecting the neurons contain weight values, which contain the relevant information of the model. The role can be presented as equation 2.1:

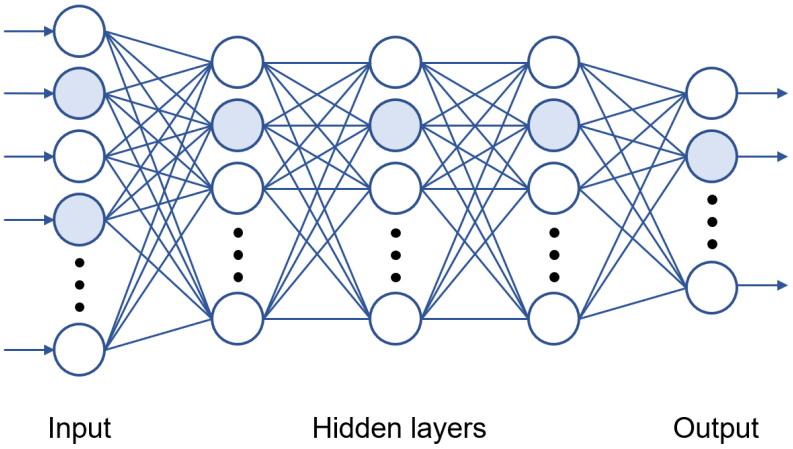


Figure 2.1. *An example of a deep NN excluding the weights, biases, inputs and outputs.*

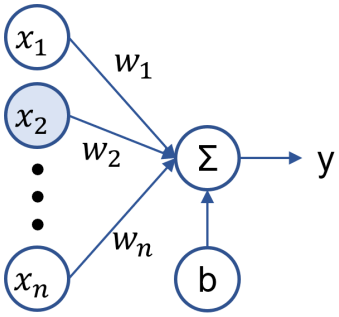


Figure 2.2. *A single neuron in a deep NN. Adapted from [11].*

$$y = \sum_{i=1}^{i=n} x_i w_i + b \tag{2.1}$$

where y is the output from the neuron, x_i is the input vector, w_i is the learned weight vector and b is the bias term, which tries to approximate the shifting needed for the activation function to work properly [13]. The output of the neuron is fed into an activation function. The Rectified Linear Unit (ReLU) is a popular activation function for DL. Other popular activation functions include the Sigmoid function and the hyperbolic tangent [11]. Even though there are other activation functions and there have been other improved variations of ReLU [14], ReLU will be used in this thesis as the main activation function due to its computational simplicity, fast learning and proven good performance compared to other common activation functions [11]. A reason to consider other activation functions is the dying problem of ReLU, in which neurons become inactive and never output anything but the value zero for any input [15]. The fast learning of ReLU is the main advantage that influences the selection over other approaches regardless of the problem. ReLU can be described as function:

$$z = max(0, y) \tag{2.2}$$

where z is the output and y is the input to the activation function. Activation functions add non-linearity, for the sake that a system consisting merely of linear activations could be reduced into a single neuron. [11] As presented in equation 2.2, the non-linearity is achieved through setting values below zero to zero. An exception to using ReLU is made in the last activation of the network, where Softmax is used instead. Softmax maps the output values into values between zero and one and is imperative for producing the final predictions for class values in multi-class classification [16]. Equation 2.3 presents the Softmax function: [17]

$$s_i = \frac{e^{y_i}}{\sum_{j=1}^{j=n} e^{y_j}} \tag{2.3}$$

where s_i is the Softmax output, y_i is the corresponding ith value in the output vector of the preceding layer, y_k is used to iterate over all the output vector values, n is the number of classes and e is the Euler's number.

The process of converting an input into an output by using equations 2.1 and 2.2 for each neuron starting from left to right, finishing with equation 2.3, is called the forward pass. After the forward pass, the last output layer in the network, the size of which is equal to the number of the target classes, outputs a vector containing probability values for each class label. In the training phase of the network these values are off by some amount depending on how far the network has been trained. Thus, after each forward pass, the weights of the NN are adjusted through a process called the back-propagation. In back-propagation the network weights are iterated backwards by calculating gradient values towards the steepest gradient using equation 2.4: [18]

$$W_i = W_{i-1} - \gamma \frac{dE}{dW} \tag{2.4}$$

where W_i is the new weight, W_{i-1} is the old weight, γ is the learning rate, also known as the step size and $\frac{dE}{dW}$ is the derivative of the loss with respect to the weight [18]. The categorical cross-entropy is a loss function, that is used to measure the prediction error of a DL model with multiple output classes in probability-based classification. Categorical cross-entropy is calculated using: [17]

$$E = -\sum_{i=1}^{i=n} \tau_i log(s_i) \tag{2.5}$$

where E is the categorical cross-entropy loss value, s_i is the output of equation 2.3 and τ_i depicts the probability distribution of of each training sample, where a value of one is given to the correct class and zero to all other classes [17]. The derivative of the loss described by equation 2.5 with respect to the weight in equation 2.4 is solved using the

chain rule, however the solution is not relevant to be discussed further in the scope of this thesis.

Adam is a widely used method for optimizing the individual learning rates for different parameters, with a proven performance on a variety of models and datasets [19]. Thus, Adam is used as the optimizer of choice in this thesis.

Dropout is a stochastic regularization method that is effective at preventing model overfitting [19]. Commonly the value for dropout is set to between 0.3 [6] and 0.5 [16], [20] meaning 30% to 50% of hidden neuron output values are set to zero. The dropped-out neurons do not participate in forward pass or back-propagation and thus prevent the network from relying on a single neuron too heavily. The downside of using a dropout layer in a deep NN is that it roughly doubles the training time of a model to converging. [16]

2.1.2 Convolutional Neural Network

Convolutional neural networks (CNN) are a type of deep feed-forward NN, which have achieved great success in image detection, segmentation, and recognition [11]. CNNs have been a state-of-the-art method in image recognition since AlexNet [16] was developed in 2012. While image recognition is the main application of CNNs, CNNs have been used to classify graphs as well [20] [21]. The network gets its name from a particular linear operation called the convolution. In addition to the convolution, an activation function and a pooling operation are generally applied to the data [18]. Figure 2.3 presents an example of a CNN architecture and positions the convolutional and pooling operations to the structure. The classification step at the end of the architecture contains a deep NN but is compressed instead and presented as a blank rectangle.

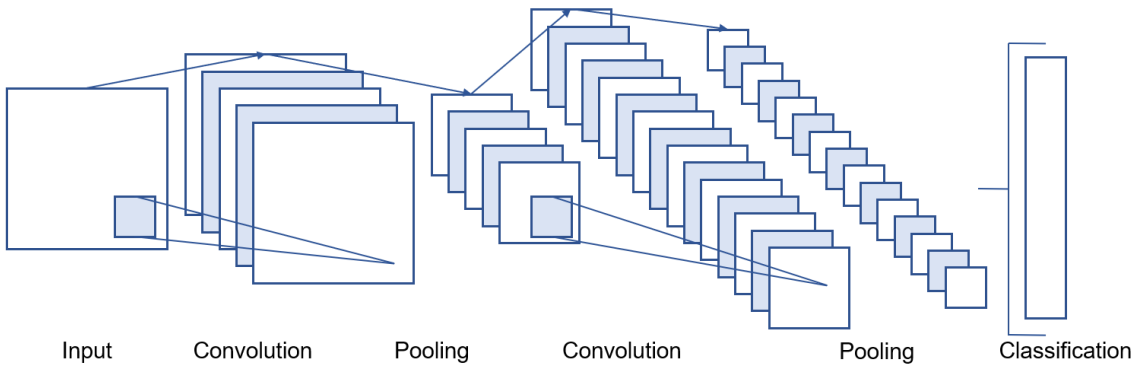


Figure 2.3. *An example of a CNN architecture. Adapted from [4], [14].*

A convolutional layer of a CNN consists of multidimensional matrices, called filters [20]. A filter size of 3×3 is commonly used in image classification applications, such as the [9]. The filters contain weights and biases similarly to the NN neurons and are updated using

the same back-propagation principle presented earlier in the equation 2.4. The filters are convolved over the input matrices to create a feature presentation, that is used as an input to the next layer in the architecture. The input matrix can be padded to retain the matrix size after the convolution operation. The padding operation is optional and is decided based on the context by the NN designer. The stride parameter describes how much the convolution filter is moved at each convolution. Figure 2.4 visualizes these parameters. The input matrix is presented in blue and the part of the matrix, which the filter is applied to is highlighted in green. The filter learned by the CNN is shown in a light orange color. The part of the input is multiplied by the filter using matrix dot product, and the resulting matrix is summed into a single value. The next value in the output matrix is calculated by moving the green input to the right equal to the stride parameter and repeating until the whole output has been covered. It can be seen, that without padding the matrix, the output dimension is smaller than the input dimension. [18]

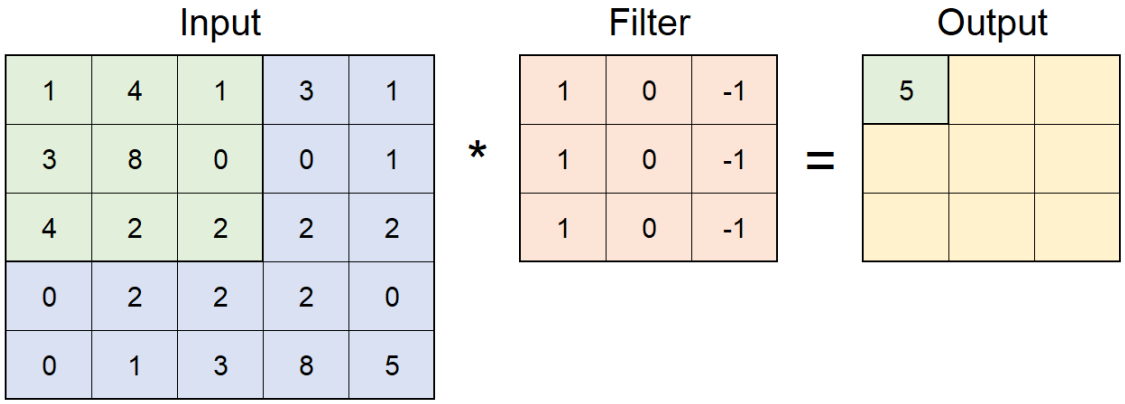


Figure 2.4. *The convolution operation on a 5 by 5 matrix with filter size of 3, no padding and stride of 1.*

Similarly to the deep NN, an activation function is required to add non-linearity to the system. The ReLU is used between convolution layers and convolution and pooling layers for the reasons stated in subsection 2.1.1. In the literature, the activation layer is sometimes referred to as the detector stage [18].

Pooling is the operation of reducing the convolutional layer output matrix dimensionality to combine semantically similar features [11]. Pooling is invariant to small translations in the input data, and thus the locations of elements in the matrix can vary between inputs [18]. Figure 2.5 describes a variation of pooling called the max pooling. Max pooling uses a kernel of size $N \times N$ over a matrix to choose the maximum value in each kernel. This diminishes the matrix size to N^2th of its original size. Another frequently used method in ML is average pooling, which follows the same principle, except instead of the maximum value it uses the average value in each kernel.

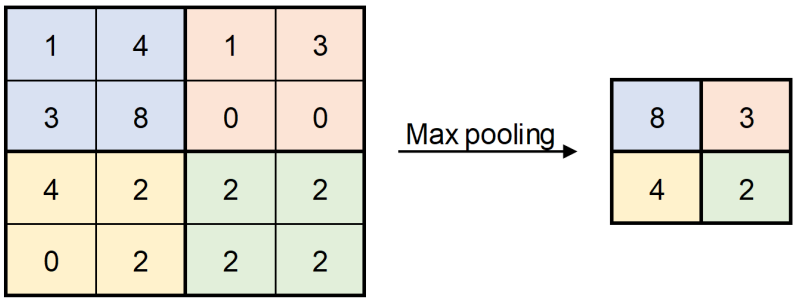


Figure 2.5. *The max pooling operation on a 4 by 4 matrix with a kernel size of 2.*

The result that is obtained by applying the presented steps according to figure 2.3 is a feature map that can be fed into a NN model. The NN, which does the final classification, was described more in-depth previously in section 2.1.1.

2.1.3 Computer Aided Design

Vertex G4 CAD models consist of elements, which can be either faces, edges or vertices. The different elements of the CAD model belong to the same volume, that acts as the base of the model. The division to faces and edges follows the same structure as other CAD model related ML studies, including Ma et al. [22]. However, a key difference compared to related studies are the point elements called vertices, which are not widely used in the literature or in other popular CAD systems. However due to Vertex G4 containing a point element, which assembly constraints can be set to, it is important to include points in the element types. Points are either the end points of edges or the center points of circles.

Attributes of a CAD model are divided into three categories in the literature: inherent attributes, derived attributes and the intended use. Inherent attributes are attributes related to the geometric or material properties of the model (length, diameter, face, edge, vertex, etc.), whereas derived attributes involve external disturbance (weight, motion, deformation, etc.). The intended use describes the use case of the model and is difficult to describe numerically. [23] In CAD model classification tasks, the inherent attributes are the most important because they contain all the parameters that are relevant to the structure of a CAD model, which is the most important aspect for assembly constraint classification. Inherent attributes will be used as the main method to depict a CAD model in this thesis, combined with a minor addition of derived attributes. The intended use is left out of the scope of this thesis.

In Vertex G4 assembly workflow, the user selects two or more elements from CAD models, which can be either parts or other assemblies. During each selection, the program checks the element types for validity to be used in the selected constraints. Other well-known commercial CAD products, such as Siemens NX [24], utilize a similar assembling process to Vertex G4. This process differs from some systems, such as Solidworks [25],

where the constraint type is selected after the elements have been selected. The main difference between these two approaches is that the first approach checks element type validity per constraint in the input phase, whereas the other presents the user only valid constraints between the two chosen elements. Figure 2.6 presents the process diagram for the assembling process, following the first presented method. The selected elements can be related to either parts or other assemblies. Assembly constraints determine the relative position and allowed movement between the entities [26]. Each constraint removes one to six degrees of freedom (DOF) from either one, or both parts. In the end the constraint and selected elements are passed onto a constraint solver that does the mathematical calculations related to the constraint. Adding too many constraints between entities makes the assembly over-defined causing the geometric constraint solver engine to be unable to solve the assembly.

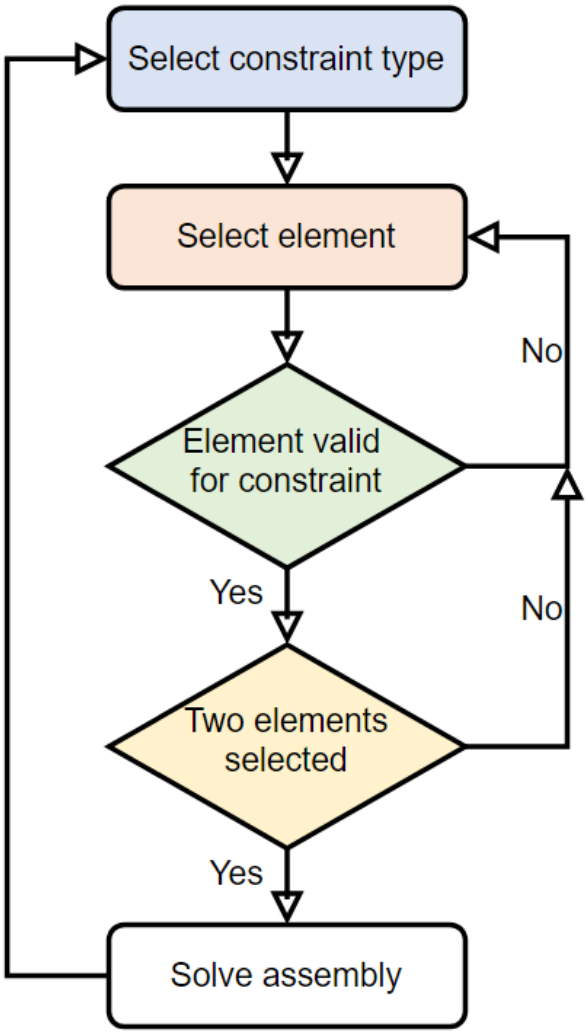


Figure 2.6. *Process diagram for product assembling process in Vertex G4.*

Assembly constraint types	Element types
Distance	Plane
Coincident	Cylinder
Angle	Sphere
Perpendicular	Torus
Concentric	Cone
Parallel	Spline surface
Tangential	Line
Equal distance	Circle
Symmetric	Spline curve
Handle linkage	Point

Table 2.1. *Assembly constraint types and element types combined.*

Table 2.1 combines the types of assembly constraints and elements into a single table. In total there are ten of both. Out of the ten assembly constraints, the first seven are constraints that connect two elements together. The last three constraints, highlighted in light orange color in the table, require more than two elements. The equal distance constraint needs to be set between four elements, and the symmetric constraint requires a third element that acts as a symmetry axis. The handle linkage is a unique constraint to the CAD system at hand, and it uses different elements altogether. The assembly constraint classifier presented in this thesis focuses on the constraints between two elements, due to the choices in data structures. Adding more elements would require changes in the architecture and creating the classifier based on the commonly used assembly constraint types makes the generalization of the system possible on other fields of CAD development. Thus these assembly constraints with more elements are not discussed further in this thesis.

Depending on the type of the chosen elements the assembly constraints can have slightly different behavior. For example, adding a distance constraint between two unparallel planar surfaces automatically includes the parallel constraint, which makes the distance constraint applicable. Due to the nature of these side effects, going over them in detail is not inside the scope of this thesis. The side effects are generally minor and dependant on the previously described attributes of the CAD model. Thus, the effects are redundant information to the CNN classifier.

2.2 Literature Review

Assembly constraint classification has not been widely studied in the ML community. However, many of the elements relevant to assembly constraint classification are present in other CAD model related classification applications. The most prominent are manufacturing feature classification [3], [6], [7], part type classification [4], [8], [9] and retrieval of similar parts based on geometry for design reuse purposes [5], [10]. This section reviews the literature around these applications, focusing on methods which either are or can be feasibly solved using CNN architectures. Thus, older studies that require manual feature extraction conducted before CNNs were a state-of-the-art method are left outside the scope of this thesis.

Zhang et al. [3] solve the problem of machining feature detection from mechanical parts using geometric data in voxelized form. Voxels (volumetric pixels) are 3D versions of pixels and can depict a CAD model as a 3D binary matrix grid. While the method reaches a high classification accuracy of up to 97.4% with 24 classes, it suffers from resolution limitations of the voxel structure and difficulties differentiating similar features such as chamfers and rounds. With fast computation times in classification, the method is viable for real-time use when the input data has been converted to voxelized form before the start of classification. [3]

Shi et al. [6] present a novel manufacturing feature recognition model utilizing heat kernel signatures. The model can recognize interacting machining features, which according to the study, is limited in other manufacturing feature recognition approaches in the literature. The method converts CAD models into heat persistence maps that are clustered by grouping similar heat persistence values. Then the clusters are combined with an attributed adjacency graph that contains the concave or convex relations of adjacent faces. Finally, the adjacency graph is processed into a lower dimensional matrix presentation that is fed into a CNN. The system achieved a total classification accuracy 98.8% with 11 classes and interacting features. [6]

Hao and Chi [7] present a solution for machining feature detection through scoring each face in a CAD model with a numerical value based on the concavity or convexity of the face and its edges, as well as the existence of possible inner loops. The model emphasizes the importance of the face more than its edges or inner loops by adding or subtracting from the total score more based on the face geometry. Eight most important faces related to the main face are selected and used as an input to a NN. However, the study does not validate the results on any larger dataset and thus the classification accuracy for the system is unknown. [7]

Dekhtiar et al. [4] convert the solid CAD model into N different images; viewpoints, which act as a degraded view to the model geometry. A novel data augmentation process is

used to make the data more resistant to noise and allow a more consistent behaviour in real life situations. For maximal strength 52 images are proposed per CAD model. The advantage of the technique is the possibility of using pre-trained image classification models, such as the GoogLeNet [27]. In a 30-class classification task the model reached top-1 accuracy of 82.81% and top-5 accuracy of 90.68% on the testing set. [4]

Hegde and Zadeh [8] propose a 3D model classifier for solving the Princeton ModelNet [28] classification challenge. The method combines the principles from Zhang et al. [3] and Dekhtiar et al. [4] to complement the multi-view image pixel data with the volumetric information of voxels. The method reached the highest classification accuracy of its time on the ModelNet dataset of 93.11% with 10 possible classes and 90.80% with 40 classes. [8]

Manda et al. [9] use a light field descriptor that captures 20 images from cameras distributed evenly around the CAD model. The study differentiates itself from other multi-view studies by the number of images taken per model to reduce redundancy. In addition, a post-processing scheme is applied to the CNN output to prevent incorrect classifications on some of the 20 images due to some classes looking similar from certain view directions. The post-processing increases the classification accuracy from 93.41% to 95.63% when classifying between 43 classes. [9]

Zhang et al. [5] proposed a generic face adjacency graph method for design reuse that considers the pre-existing constraints in assemblies through a system called the mating face pair. The generic face adjacency graph is based on the face adjacency graph originally presented by Ma et al. [22]. While neither of these studies uses the adjacency graphs for CNN input, the studies present methods for mapping the graphs into two dimensional presentations valid to be used with CNNs.

Muraleedharan et al. [10] approach the design reuse problem through Part in Whole Retrieval (PWR). They present a method for part segmentation and normalize the segmentation results through a Gauss map feature extraction process. The output is fed into a type of unsupervised NN called an autoencoder. However, no detailed description of autoencoders is presented in the scope of this thesis. The results of the study are validated manually due to the absence of a labelled dataset. The presented method has limitations with parts that have two segments with a hole going through them. [10]

The literature review demonstrated the theoretical framework for assembly constraint classification. It presented the commonly used methods based on adjacency graphs, voxelization and image-based methods. None of the discussed studies explored the assembly constraint classification problem, which shows a research gap to be filled. Because the CAD related DL field has attracted a lot of attention in the recent years, going over all the related studies is not possible in the scope of the review. Instead, the review focused on presenting a variety of different approaches for CAD model classification.

3. DATA COLLECTION

3.1 Feature Extraction

3.1.1 Adjacency Graph

CAD models are an instance of unstructured data, which means that their data cannot be presented as a matrix directly, without losing inner information. The opposite of this are for example pictures, where all the inner information can be presented by matrices consisting of pixel values. Unstructured data requires feature extraction, which is a highly complex and costly process. [4]

Converting CAD model data into a matrix form is required, because a solid model is not suitable for neural network input. Neural networks require a numerical representation of the CAD model, which can be achieved through four main methods: 2D projections, adjacency graphs, face score vectors and voxelization [2]–[4], [6], [29]. In addition to these, point clouds have recently been used in DL to present 3D model data [2]. Adjacency graphs, voxelization and 2D projections, also known as image-based approaches, are the state-of-the-art and the most used methods in CAD model classification. Each of these methods has its advantages and disadvantages, which are compared using three parameters: the number of parameters in the matrix presentation, the data extraction speed from a CAD model and the difficulty of augmentation. These parameters are collected to table 3.1. The justifications for the selected values are either based on the literature, or empirical testing done in the CAD environment.

Method	Number of parameters	Extraction speed	Augmentation
Adjacency graph	N^2	Depends on N	Medium
Voxelization	64^3 [3]	Fast [30]	Medium [31]
Image-based	$256 * 256 * 1$ [9]	Slow	Easy

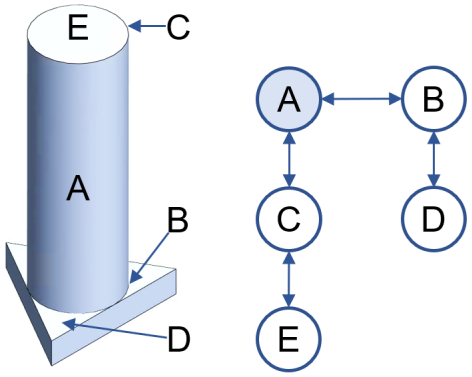
Table 3.1. *Method comparison for CAD model conversion to a matrix presentation based on important parameters for assembly constraint classification.*

The first two comparison parameters are related to the real-time performance of the assembly constraint classifier. For the system to be applicable to CAD design workflow, the feature extraction needs to happen seamlessly to the software user. This is used as

a baseline for the method comparison. Adjacency graphs are presented as a $N \times N$ matrix as described earlier. Thus, the number of parameters in the matrix is N^2. In the case of the assembly constraint classifier, the number of parameters can be selected based on how widely the surroundings of an element are described. Fewer nodes result in a very fast extraction speed. The augmentation of graphs is a problem in itself, and it will be discussed further in section 3.4, however at this point it is enough to state that the augmentation is similar to the one of voxelization and harder than the one of image-based classification. While the voxel form of a CAD model can be extracted fast [30], the number of parameters causes problems in the case of this thesis. Similar augmentation methods work for voxels as for images, however the extra dimension lowers the augmentation speed [31]. Both of those parameters lead to a requirement of a large dataset, which as discussed later in section 3.2, is not feasible. Image-based approaches have the benefit of easy data augmentation, but there are other problems for assembly constraint classification. Assemblies often contain parts that are hidden from different angles or are located inside other parts. This results in irrelevant data when viewed from some directions and makes some cases unable to be classified. The presented image extraction speed is based on pre-existing screenshot system in Vertex G4 CAD software, which is fluently able to capture 24 screenshots per second. According to literature, using 20 images leads to good classification performance, which would require around one second per CAD part [9]. This makes the use of image-based approaches illogical for real-time CAD design.

Another aspect to discuss in the assembly constraint classification problem is the method to present the selected elements of the CAD model to the CNN classifier. The selected elements are the most important item in the selection of the assembly constraints. For adjacency graphs the selected element can easily be enforced because the adjacency graph contains the elements as nodes. This is the most problematic in the voxelization approach, where elements occupy only a fractional section of a part of the voxels. For image-based approaches the selected element can be presented in a different color or in a similar manner. Based on all these discussed factors, adjacency graphs are selected as the data presentation in this thesis.

Adjacency matrices contain the connections between adjacent elements in an adjacency graph. The matrix presented in figure 3.1 presents an example of a simple graph and its adjacency matrix, which can be expanded to cover the adjacency matrices of CAD models. As described in subsection 2.1.3, CAD models consist of faces, edges, and points. Each of these elements is a node in the adjacency graph of the model. Similarly to A, B, C, D and E in figure 3.1, each element in the CAD model is given a unique identifier. The element connections are presented as edges in the adjacency matrix. Faces are generally connected to edges and edges are connected to points. Exceptions are made when a part is a sphere or a torus because they have no edge lines.



	A	B	C	D	E
A	0	1	1	0	0
B	1	0	0	1	0
C	1	0	0	0	1
D	0	1	0	0	0
E	0	0	1	0	0

Figure 3.1. *Visualization of a CAD model, its partial adjacency graph and the corresponding adjacency matrix presentation.*

Figure 3.1 contains a header row and column of the node identifiers and binary values for the connections between the nodes. The arrows between nodes present two-way connections, which make the graph traversable in both directions. A binary value of one presents a connection between nodes and value zero tells that there is no connection. A graph with two-way connections always results in a symmetric adjacency matrix. To check if e.g. nodes A and B are connected, the value on row A and column B is read. In the example matrix the nodes are connected by the binary value one in the respective location. The presented matrix is only one of multiple possibilities to present the data in the graph at issue. By default, there is no specific order for the nodes. The header column and row can contain values A, B, C, D and E in any permutation, as long as the order is same for both. This causes problems in CNN training and classification because the data does not contain spatial structure. This problem will be addressed in depth in section 3.3.

3.1.2 Feature Vector

As described in subsection 3.1.1, the adjacency graph describes only the relational data between CAD model faces, edges and points. However, only using the relational data is not enough to describe the complete inner geometrical structure of the model. Thus, additional information needs to be introduced to complement the geometrical presentation. This is done through feature vectors, which add additional data to each adjacency graph node. Feature vectors are used in graph-based spatial CNN studies, such as [20]. In this thesis, the feature vector of each node in the graph is split into four categories. The four categories are chosen to minimize the number of parameters in the presentation in a way that still provides an extensive depiction of the CAD model. The division to the categories is based on empirical research and recognized geometrical essentials of a CAD model from the literature. The first category describes the importance of a node based on its location related to the selected element of the assembly constraint [20]. The second category contains geometrical type information of the node [22] and the third category is used

Main class	Value	Subclass	Value
Face	1	Plane	1
		Cylinder	2
		Sphere	3
		Torus	4
		Cone	5
		Spline surface	6
Edge	2	Line	7
		Circle	8
		Spline curve	9
Vertex	3	Point	10

Table 3.2. *Geometrical type values in the feature vector.*

to describe the geometrical form of the node [23]. The fourth category contains the DOF data of the part [23].

The importance rating can be calculated through different means. As explained later in section 3.3, the commonly used methods for calculating the importance rating for nodes of a CAD model are not trivial. Instead, a Boolean value is used to impose the two selected elements in the assembly constraint. The value true is given to the respective nodes of those elements in the adjacency graph. For other nodes, the value false is given instead. The goal of this approach is to guide the CNN classifier towards giving more weight to the geometrical type information and the geometrical form of the node that is directly a part of the assembly constraint.

The geometrical type information is divided into geometrical main class and subclass. The main class contains the three main element types of a CAD models enumerated. The subclass contains all the subclass types, also known as the element shapes, enumerated. Table 3.2 presents the values used for the main and subclasses. The faces are divided into plane, cylinder, sphere, torus, cone and spline surface shapes. The spline surface covers elements that do not fit to the other face classes. The edges are divided into lines, circular lines and spline curves. For the vertices there is only one subclass, the point.

The geometrical form parameter is used to add a spatial component to the matrix presentation. Depending on the element subclass, normal vectors, direction vectors or 3D points are used. The normal vector is used for planar surfaces, due to it providing relevant information of the shape. The direction vector is used to present the geometrical form of cylinder, torus and cone surfaces and line edges. The 3D point is used to describe the spherical surfaces' and circular lines' center points, as well as the point element. The spline surface normal changes based on the location and similarly, the spline curve has multiple possible direction vectors. Thus, these elements are described with a zero vector

not to hinder the model performance. This decision is based on the fact that the spline elements are more infrequently used in assembly constraints than the other geometrical shapes.

To describe the DOFs of the CAD model involved in an assembly constraint, a vector of size 1×6 is used. Each degree of freedom is described by a binary value, where a value of one is used if the DOF is open and a value of zero if it is locked. To make the data able to be concatenated with the rest of the feature vectors it needs to be padded to size $N \times 6$ with zeros. The DOF information is necessary for the assembly constraint classifier due to some constraints only being sensibly applicable when some of the model's DOFs are locked. An example of this is the angle constraint, which is rarely used between two parts unless only one of the rotational freedoms is open and the other DOFs are locked. In other cases, the angle constraint is often dominated by the more frequently used coincidence and distance constraints.

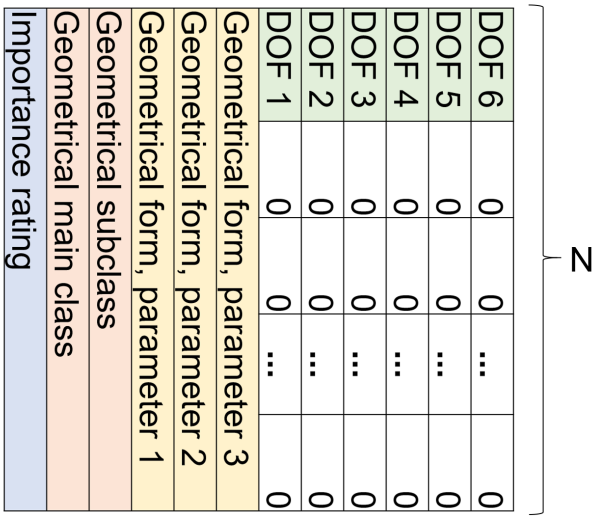


Figure 3.2. *Feature vector matrix presentation.*

Combining the importance rating, the geometrical type information, the geometrical form and the DOF information results in a $N \times 12$ matrix. The results are combined by concatenating all the information together. The resulting matrix containing the feature vectors is presented in figure 3.2. It is important to notice, that the geometrical type values are significantly higher than the other values in the feature vector. It is noted that this can theoretically influence the CNN model performance, however these effects are not discussed further in this thesis.

3.2 Dataset Generation

Even though 3D model related ML has increased in popularity in the last few years, the deficiency of datasets presents problems for developing more advanced methods. While there are some datasets, such as the ShapeNet dataset [32] and the DMU-Net dataset

[4], they often concentrate on the classification of different shapes and do not contain necessary assembly constraint information for the case of this thesis. Thus, pre-existing datasets will not be suitable, and data must be collected from a different source.

Classification using deep learning requires large amounts of data and the performance of a model is highly dependent on the quality and quantity of the data used [3]. Depending on the methodology the amount of data per class varies from tens for multi-view classification [4] to 5000 to 6000 for graph and voxel-based applications [6][3].

A general approach for generating more data is the geometric variation of 3D models within set limits. This approach works best for parametric CAD systems. However, the approach is not well suited for the classification approach used in this thesis because the data used is invariant to scale and only describes the relations between different elements in a CAD model. Thus, a different approach is needed.

The approach chosen for data collection is to use pre-existing assemblies that have been created by professional CAD designers. Vertex Systems has access to several professionally created assemblies that are used to test the functionality of the Vertex G4 CAD system. Some of the assemblies used are intellectual property of the customers and are thus privileged information and cannot be presented in depth in this thesis. However, due to the nature of deep learning this information can be used as training data. Other used assemblies consist of models created for the internal use of the company and will not thus be presented either.

The approach provides advantage compared to the geometric variation in data quality due to the variety of parts used in the assemblies, while the geometric approach provides more quantitative data. It also does not require manual work in ground truth annotation, or contain as large of a risk of misclassification, since the assembly constraints have been carefully selected by skilled designers during the assembly process. Extracting these values directly from the assemblies, and using them as the classifier ground truth values, increases the generalization ability of the system due to a broad pool of designers over multiple expertise fields. The goal of the approach is to capture the essence of CAD designers and their expertise inside the ML algorithm to make the design process more fluent.

Assemblies consist of parts and sub-assemblies, of which sub-assemblies can contain more assembly constraints than what are visible to the main level of an assembly. The sub-assemblies can contain additional sub-assemblies. To collect all constraints from an assembly all the sub-assembly constraints need to be collected as well to maximize the size of the dataset. Algorithm 1 describes two functions that can be used to collect constraints from assemblies. The first part is a recursive function that goes through all the sub-assemblies and collects them into a single data container. This function is called by the second function, that then loops through all the assemblies and all constraints that

belong to them. The constraints can be filtered using multiple criteria, however here the number of elements involved is used to filter out the constraints, that require three or more elements, such as the equal distance constraint.

Algorithm 1: Algorithm for collecting all constraints recursively from an assembly.

```
1  Function GetAllChildrenAssemblies (R);
   Input  : Root assembly R
   Output: List of children that are assemblies C
2  children = [ ];
3  for child in R.children do
       // Only assemblies have children
4      if not child of type assembly then
5          continue;
6      end
7      insert child to children;
8      insert GetAllChildrenAssemblies(child) to children;
9  end
10 return children;
11 Function GetAllConstraints (R);
   Input  : Root assembly R
   Output: List of assembly constraints C
12 children = GetAllChildrenAssemblies(R);
13 C = [ ];
14 for child in children do
15     c = get all assembly constraints from child;
16     for constraint in c do
17         if constraint is between two elements then
18             insert constraint to C;
19         end
20     end
21 end
22 return C;
```

As described in subsection 2.1.3 volumes act as the base for a CAD model. Thus, a volume will be used as the starting point for the data collection algorithm. The data collection is done in three main steps. The first step is to collect faces from a CAD model volume and add them to the graph. Respectively, the second step collects edges from faces and the third step collects points from edges and adds them to the graph. These steps are presented as pseudo-code in algorithm 2. The algorithm checks if a node already exists in the graph before creating another one to prevent the creation of duplicate nodes. If the node exists it is fetched from the graph. The result of the algorithm is an undirected adjacency graph in an object-oriented form.

Algorithm 2: Algorithm for creating an adjacency graph from a CAD model.

```
Function CreateAdjacencyGraphFromVolume (V);
Input  : Part model volume V
Output: Adjacency graph G
S = get surfaces from V;
for surface in S do
    create surface node sn;
    E = get edges from surface;
    for edge in E do
        if not edge in G then
            create edge node en;
            insert en to G;
        else
            get en from G;
        end
        P = get points from edge;
        for point in P do
            if not point in G then
                create point node pn;
                insert pn to G;
            else
                get pn from G;
            end
            connect pn to en;
        end
        connect en to sn;
    end
end
return G;
```

Five large scale assemblies were selected from Vertex libraries and the algorithms 1 and 2 were run on them. In total 2580 assembly constraints were collected initially from the assemblies. Figure 3.3 visualizes the division of assembly constraints between the selected assemblies sorted in descending order. Each assembly is named in numerical order to retain anonymity.

Distribution of constraints within possible categories in the five assemblies is presented in figure 3.4. The coincident constraint is the most used by the designers of the assemblies. This behavior is expected due to the constraint's flexibility. The parallel, distance and concentric constraints are used commonly as well, however not as commonly as the coincident constraint. The tangent constraint is used relatively little, while the angle and perpendicular constraints are used very infrequently or not at all.

As it is shown in figure 3.4, the constraint class distribution is imbalanced in the current dataset. This means that there is skewness in the data, which causes problems for the

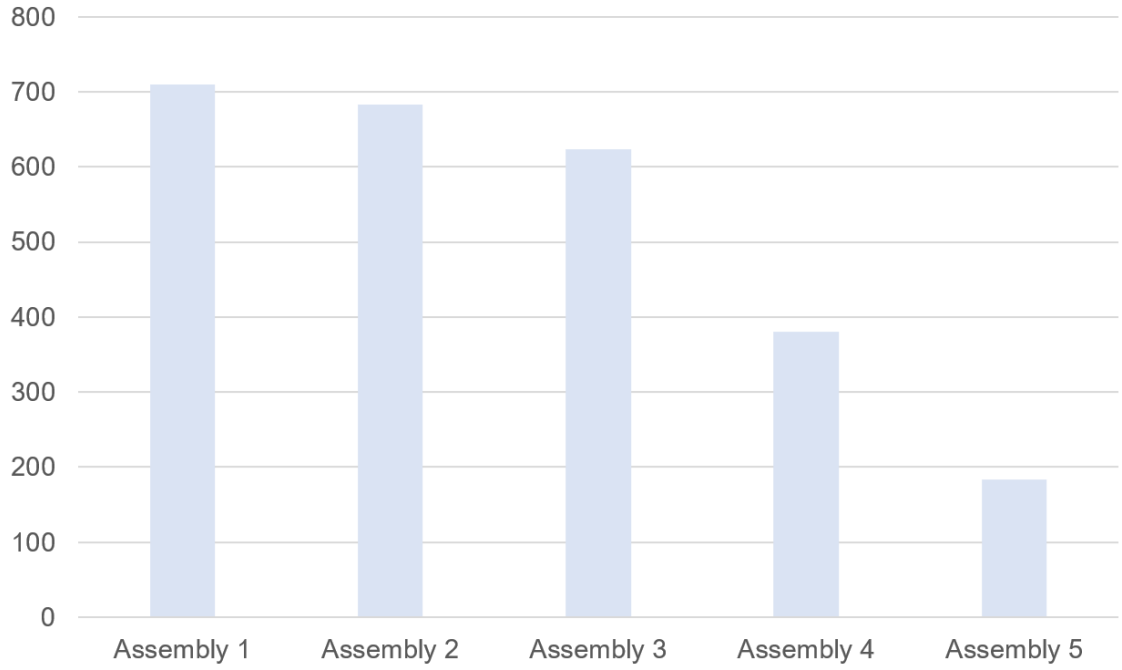


Figure 3.3. *Quantity of constraints extracted from each of five large assemblies.*

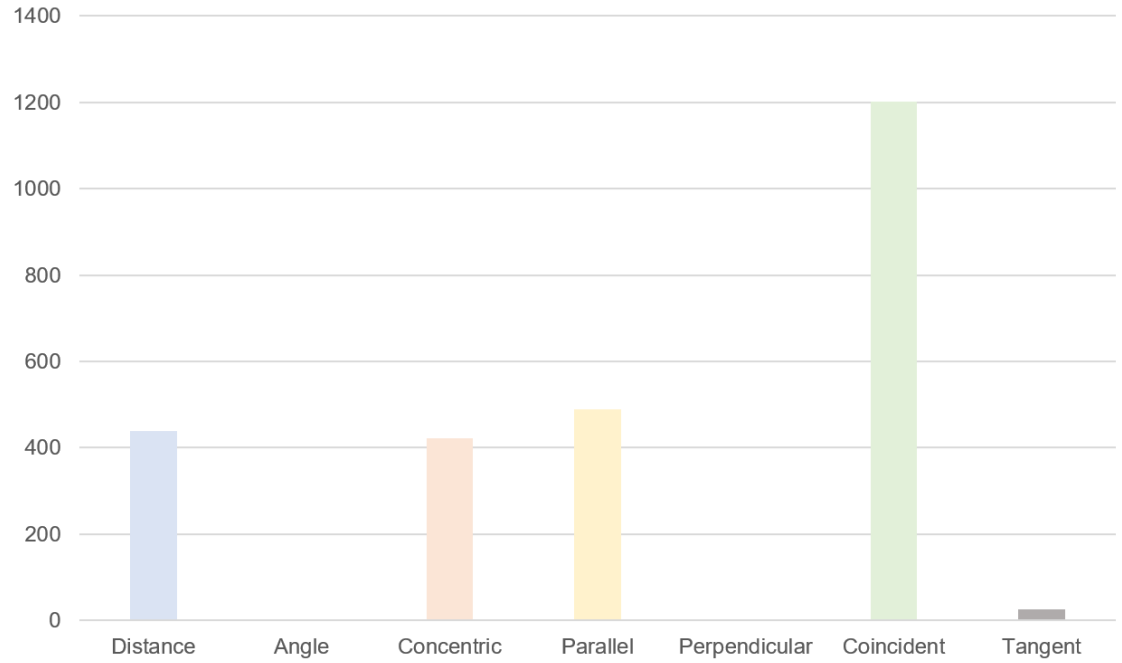


Figure 3.4. *Initial constraint distribution within data extracted from pre-existing assemblies.*

DL classification system. CNNs are not designed for imbalanced classification tasks, and when a model is trained on an imbalanced dataset, it is due to develop a bias towards the majority class and predict it more commonly than a minority class [33], [34]. In this case, the system is prone to learn to predict the coincident constraint in cases, where the other minority constraint classes are the correct choice. To improve the system, the quantity of the samples in the minority classes can be increased to balance the overall distribution of the data [33].

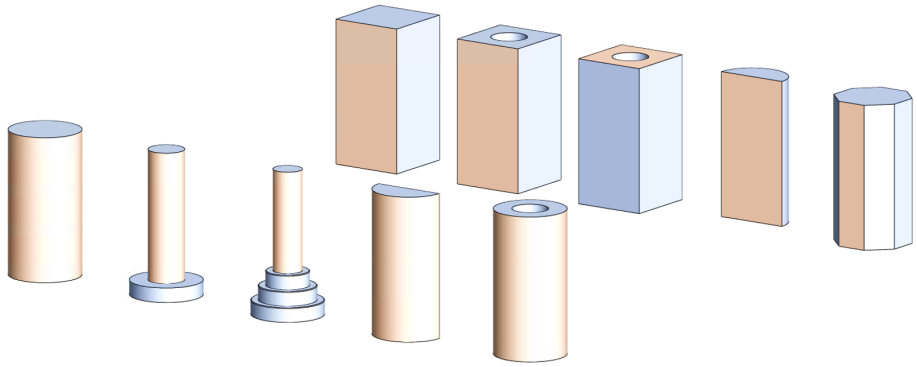


Figure 3.5. *Designed CAD models for the manual tangential constraint generation process.*

The angle constraint and the 90-degree special case for it, the perpendicular constraint, are virtually not used at all and thus generating samples for them could produce unwanted behavior in the trained classifier. Thus, those two constraint types are left out of the classification target classes. This leaves the tangent constraint as the main minority constraint class, for which data needs to be generated. A manual data generation approach is chosen, in which eight CAD models are designed and combined into assemblies using the tangential constraint. Due to the method's invariance to scale, no size variation of the models is required. Two of the models are used twice, resulting in a total of ten usable models. The models are visualized in figure 3.5. Each model is designed to present a variety of possible relations of cylindrical and planar surfaces to other surfaces. Tangential constraints can be placed between parts containing cylindrical surfaces (left) and parts containing planar surfaces (right). In this instance generating more data of tangential constraints required creating an assembly, which contained the ten models thrice. A variation of DOFs was generated for each three sets of ten models. The first had all DOFs free, the second limited all but one translation leaving the rotations free and the third had only the rotations free. Adding tangential constraints between one cylindrical surface of the models on the left in figure 3.5 with one planar surface of the models on the right in each variation of DOFs results in $3 * 5 * 3 * 5 = 225$ tangential constraints. The selected surfaces are colored with light orange color. The results of the tangential constraints data generation are shown in figure 3.6.

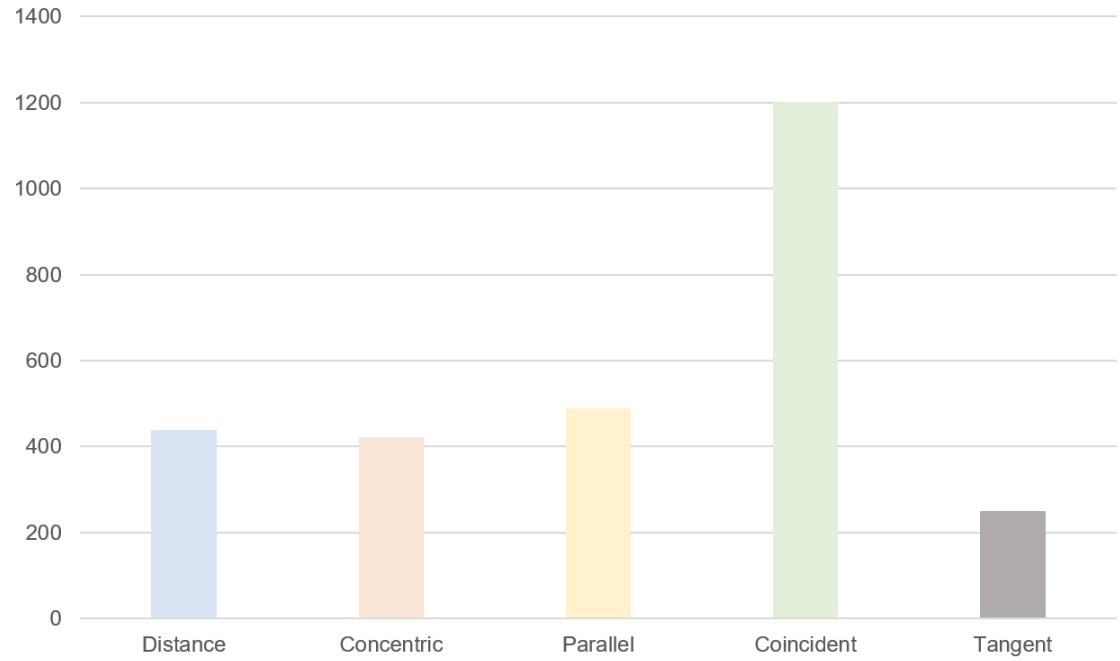


Figure 3.6. *Modified constraint distribution after manual tangential constraint generation and removal of angle-based constraints.*

While the constraint distribution is still slightly skewed, especially considering the quantity of coincident constraints compared to the other constraints, a slight bias might prove useful in the constraint classification task. The data collection from professionally designed CAD models proves that real-life designers have a bias towards the coincident constraint and while training the classifier on the current data can lead to bias in the system, the effects of this bias in the competence of the classifier should be studied. The amount of assembly constraint data can be increased using another method than manual generation. The method will be discussed later in section 3.4.

3.3 Data Processing

The data processing step is divided into two parts. The first part is the normalization of the adjacency graph data, which is done to be able to systematically convert similar assembly cases into similar two-dimensional presentations. The second part is combining the extracted features from section 3.1 into a 2D matrix form that is valid to be input into a CNN classifier.

CNNs are intended to be used with structural data with consistent internal order, which graphs by default do not contain [6]. While the adjacency graph created in subsection 3.1.1 already has structured data compared to a traditional CAD model, it still lacks the internal order aspect. Thus, normalization of the data is required. There are two main methods for creating internal order in a graph. The first is to rank each node in the graph with a value, such as the heat persistence value used by [6] and sort the adjacency

matrix in descending order based on the respective value. The second is to rank each edge in the graph using an importance rating, such as the betweenness centrality [35], which measures the centrality of a node based on the number of shortest paths that pass through that node. [20], [21]

A core difference between the two methods is the dimensionality of the output matrix. While the node ranking methods result in a two-dimensional adjacency matrix and a separate two-dimensional feature vector, the edge ranking methods result in a three-dimensional presentation instead [20]. Both the output matrices are valid for CNN input, but the latter requires an additional dimension in the CNN filters, which results in more weight parameters to learn in the CNN architecture. In adjacency graphs extracted from CAD models, the nodes, i.e. the CAD model faces, edges and points, contain the important geometric information of the model. The edges act as connectors between the nodes and assigning an importance rating to a specific connection through parameters such as the betweenness centrality is not relevant. For example, a square surface with a hole and five connected edges is far less central than a surface with ten holes that are directly connected to the surface node due to how the graph presentation is created. The ground truth constraint between these two cases is identical, however the centrality values for the edges are different. Based on these factors, the node ranking method is used in this thesis.

Breadth First Search (BFS) is used by Ma et al. [20] and Niepert et al. [21] to collect a local neighborhoods of central nodes. In this thesis, there is only one central node in each graph, which is the node that corresponds to the element belonging to the assembly constraint. Due to the structure of assembly constraints, the central node's local neighborhood contains the most relevant information for the selection of the assembly constraint. Thus, in this thesis, the rank of a node is solely based on its depth in relation to the central node. BFS is a graph traversal technique, that is used to collect N nearest neighbors for the assembly constraint node in order of their depth related to the central node. Algorithm 3 presents the principle of the traversal algorithm. The visitation status of each node is tracked and the nearest adjacent neighbor that has not been visited is chosen at each iteration [36]. This approach is prone to some randomization based on the implementation technique and the order in which the neighboring nodes are explored.

Once the N nearest neighbors have been collected from the adjacency graph, they can be converted into an ordered adjacency matrix. The order of the nodes in the first row and column is equal to the sorted order of the neighbors. The sorting is based on the depth from the central node, that is the number of edges between the central node and the specific node. If after this process the adjacency matrix is smaller than the wanted size, i.e. there are less nodes in the graph than the wanted result size, zero-padding is applied. In zero-padding the value zero is added to the last columns and rows of the matrix until the required dimensions are reached [21].

Algorithm 3: Algorithm for finding N nearest neighbors. Adapted from [36].

```
Function BreadthFirstSearch (G, s, N);
Input  : Graph G, start node s, neighbor count N
Output: List of N nearest neighbors
// Reset visited status for each node
for n in G do
    n.visited = false;
end
s.visited = true;
queue.push(s);
nearest = [ ];
while not queue.empty() do
    n = dequeue(queue);
    neighbors = n.neighbors;
    for adjacent in neighbors do
        if not adjacent.visited then
            adjacent.visited = true;
            queue.push(adjacent);
            nearest.push(adjacent);
        end
        if nearest.size > N then
            return nearest;
        end
    end
end
```

After the normalization process, the normalized adjacency matrices are combined with the feature vectors from subsection 3.1.2, the size of which are $N \times 12$. As presented earlier, since the feature vector contains node-wise parameters it cannot be combined with the adjacency matrix to create a three-dimensional matrix presentation. Instead, it is sorted to match the order of the adjacency nodes from BFS and concatenated directly to the adjacency matrix to create a two-dimensional $N \times (N + 12)$ matrix presentation, that contains both the adjacency data and the node-wise feature vectors. An assembly constraint connects two parts, both of which have an adjacency matrix and feature vectors. The matrix data of each part is stacked on top of each other to create the $(2N) \times (N+12)$ matrix presentation of an assembly constraint.

The values 24 and 12 are chosen for N for comparison in this thesis. This leads to matrix presentations of size 48×36 and 24×24 respectively. The value 12 is selected to create a square matrix, while the value 24 is expected to provide insight into the effect of increasing the neighbor count. The higher the value for N, the more the classifier will know of the surroundings of the selected elements of an assembly constraint. The increase in input size also increases the amount of data required to train and validate a classifier. With larger matrices, it is expected, that the performance of the classifier will decrease, due

to the amount of learning required and the relatively small dataset compared to other classification tasks. In addition, the increase of the nearest neighbor quantity is expected to increase the training time. Generally, the most important factors that determine the constraint applied are located nearer the constraint's selected elements and thus values higher than the ones tested contain redundant information for the constraint classification. The values are tested later in sections 4.3 and 5.2, and the hypotheses are analyzed. To demonstrate the effect of the normalization process, different constraint class samples are presented in figure 3.7. A value of $N = 24$ is selected for visualization purposes.

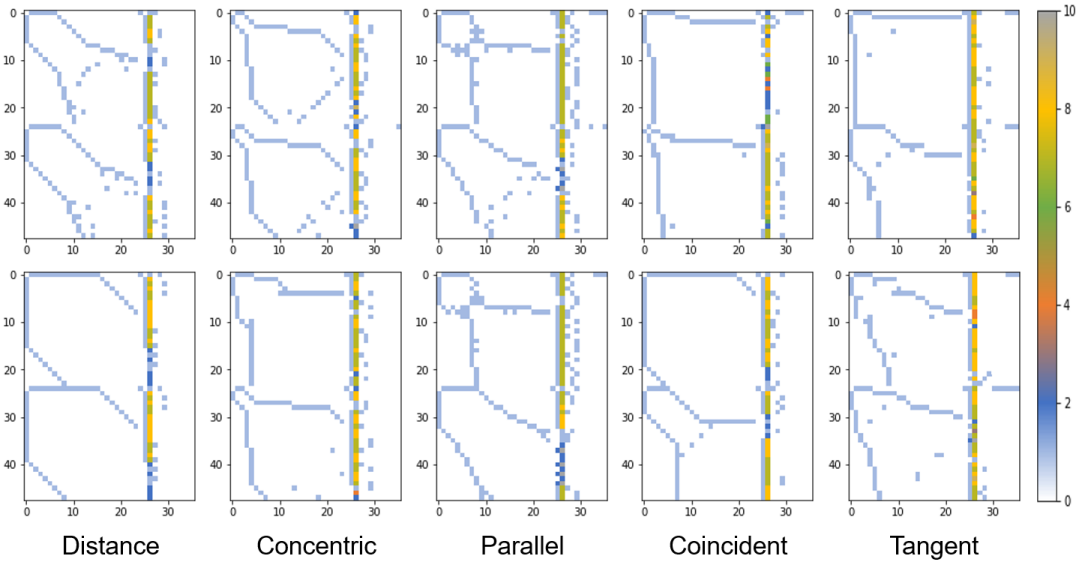


Figure 3.7. *Two randomly selected normalized constraints from each class from the generated dataset.*

Figure 3.7 shows randomly selected examples of the constraint classes in 48×36 matrix form. The colorbar describes the correspondence between these colors and the values described in section 3.1. The similarity between the matrices is not obvious to the human eye. To determine if the normalized values inside each class are similar, the mean absolute error between the matrices is calculated. For this calculation, 100 random samples from each class are selected from the original dataset. This corresponds on average to 23 % of each dataset and provides an adequate depiction of the whole dataset, while being computationally less expensive. Each sample is compared to each other sample inside the class and across other classes. Figure 3.8 shows the results of the calculations, averaged over the class pairs. The diagonal values describe the differences inside a class, while the rest of the values describe cross-class differences. The slightly smaller diagonal values, compared to the rest of the values, state that the normalization method is functional and indicates that the CNN is predicted to be able to differentiate different classes from one another. However, this will be validated later. It is important to note, that while the mean absolute error is not precise for this type of comparison, it provides some insight to the similarity of the matrices.

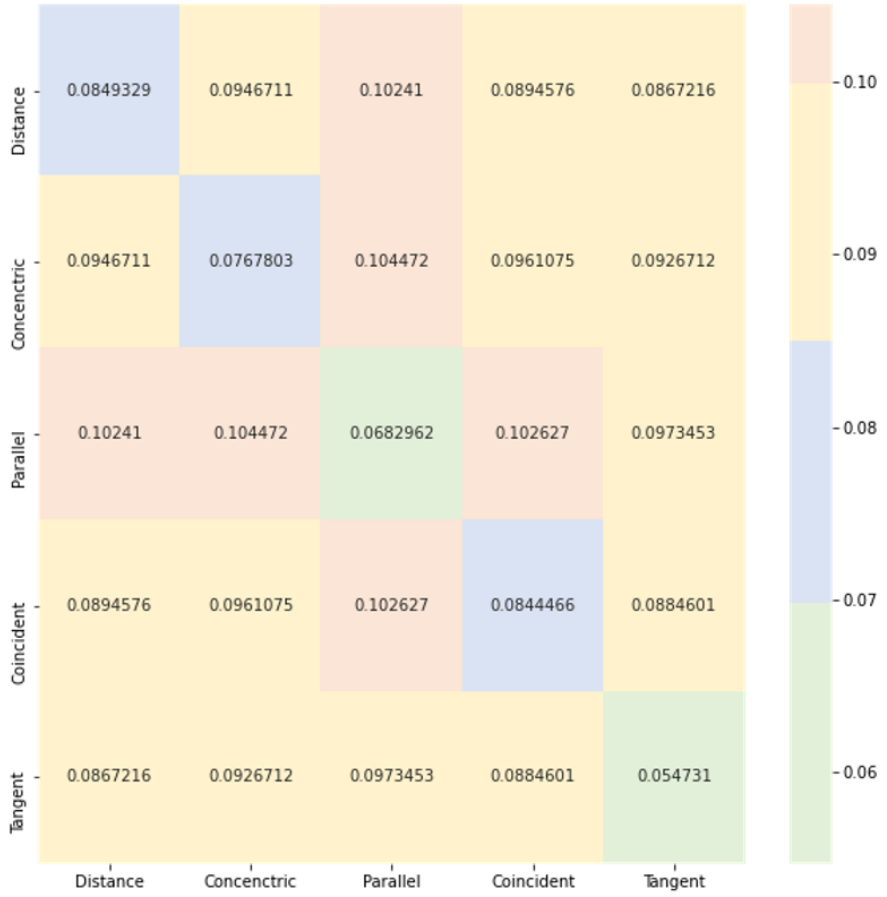


Figure 3.8. *Mean absolute error between class matrix presentations of 100 randomly picked samples per class.*

3.4 Data Augmentation

Data augmentation is common in image classification tasks. In data augmentation, operations such as mirroring, rotating and pixel shifting are performed on the image data to create a larger dataset for the image classifier. Even though the newly created data is fundamentally the same as the original data, the pixel values and their orientations are completely different while the ground truth values remain the same. Thus, the new data works as unseen data for the classifier training and validation and reduces the model sensitivity to specific situations and enhances the model robustness [4]. In CAD model classification the multi-view approaches, such as [4], are the main user of data augmentation. However, data augmentation offers great potential in other classification applications as well. As discussed at the end of section 3.3, the two parts' adjacency matrix and feature vector combinations are stacked on top of each other. However, this stacking is invariant to order and can be done in both orders to duplicate the number of assembly constraint data. This is computationally inexpensive compared to other augmentation approaches and should be used to maximize the amount of data.

While data augmentation for adjacency graphs has been studied relatively little, there are approaches such as the one developed by Zhao et al. [37] for general graph data augmentation. The problem in data augmentation for graphs is due to the graph structure being based on node connectivity instead of the position of a node. The approach for data augmentation in graphs is adding nodes or edges, of which adding possible edges and removing noisy edges is considered the state-of-the-art approach. [37] Adding or removing edges from an adjacency graph of a CAD model breaks the geometry presentation and can lead to unintended behavior. Thus, other approaches for adjacency data augmentation should be explored.

The use of data augmentation for uniformly normalized adjacency graph data is not rational when the edges or nodes of the graph cannot be altered. In the case of this thesis, this means that if each closely similar assembly constraint between two parts produces equal adjacency graph presentations, the problems in the classification of those situations are diminished and thus data augmentation is not as necessary. However, in cases, where the normalization of the graph contains random elements, data augmentation could be used to enhance classification model performance and robustness, and prevent overfitting caused by insufficient amounts of data.

The normalization method presented in 3.3 in its current form does not consider the importance of a node and treats each node at the same depth as as important. The adjacency matrix generated using algorithm 3 is dependent on the order the graph traversal algorithm explores the graph. As presented in section 3.3, the normalization of the CAD model adjacency graph in this thesis is based on the node's depth in relation to the central node. This is due to the problem of giving a higher importance rating to a single node compared to another. This opens an approach for augmentation, which requires the utilization of the random element in the node exploration order based on the internal implementation. Random walks are a method for graph exploration similar to graph traversal techniques such as BFS [36]. Combining the randomization aspect of random walks with the theory of BFS has potential to be a simple method for adjacency matrix augmentation, with possibility to generate large amounts of data.

Adding a shuffle method call to line ten of algorithm 3 randomizes the order, in which the nearest neighbors are explored. The algorithm can be run multiple times with different randomization seeds to collect different matrices from the same adjacency graph. The method retains the depth order of the graph causing small deviations to the original matrix data. These deviations are small enough to create a wider presentation of each constraint class, yet not too large to go over the class borders. The problem with the method comes in the form of possible duplicate data. In cases where nodes have only few neighbors, e.g. only zero to one neighbors, the randomization has little to no effect. For example, shapes such as the torus or the sphere contain this type of nodes. To prevent problems caused by redundant duplicate data, the created matrices are compared to the previous

Constraint type	Original count	Augmented count	Balanced augmented count
Distance	439	3480	1856
Concentric	423	3264	1856
Parallel	490	3664	1856
Coincident	1201	9608	1856
Tangent	250	1856	1856
Total	2803	21872	9280
Average	561	4374	1856

Table 3.3. *Constraints per category before and after the data augmentation operation.*

augmentation results of the specific constraint case, and exact duplicates are removed. Thus, the original count multiplied by the number of augmentations does not always equal to the value in the augmented count column of table 3.3.

The last column of table 3.3 describes an experimental approach to dataset balancing. Other approaches, such as [9], use data augmentation more on the classes with fewer data samples to balance the overall distribution. With the data augmentation method used in this thesis, this would easily lead to duplicate data, which is bad for the classifier model. The approach tested balances the augmented data count by clipping off the excess data from all but the class with the minimum number of data samples. This means that the balanced augmented count for each constraint type is limited to the minimum value of the augmented count for the constraint types. Using this approach balances the dataset, which helps eliminate bias in the classifier. However the clipping approach can only be used for the augmented dataset, because otherwise the data quantity is diminished too severely for the classifier to be trainable. The approach is tested to explore the effect of data skewness on the classifier bias, and if balancing the datasets increases or decreases the classifier performance.

4. METHOD FOR ASSEMBLY CONSTRAINT CLASSIFICATION

4.1 Software System Architecture

A key problem in the system architecture design for the software implementation exists in the programming language selection. Vertex G4 source code is mainly created using the C++ programming language. However, Python is commonly used to train ML algorithms due to its ease of use. There exist multiple easy-to-use models and libraries for Python, such as the TensorFlow Keras sequential model, that have most of the functionality related to ML training built in [38]. Based on these factors and the time constraints in the creation of this thesis, Python and TensorFlow Keras are used to train the classifier model. TensorFlow Keras also offers tools for early stopping and checkpoint creation of the training algorithm using callbacks. This is especially important in the use case of this thesis, due to the problem of multiple constraint classes fitting between similar elements, which can easily lead to over-fitting. However, C++ and Python are not compatible directly. Thus, running the classifier model requires discussing.

This thesis discusses two main approaches for running the classifier model. The first is running the model in Python separately from the C++ executable. The implementation of this approach is the least time consuming. The most critical disadvantage of this approach is that the software user needs to have a Python interpreter installed to be able to run the classifier. This requires extra work and expertise to get the system running, which makes it less likely that the software user gets involved with the new software feature. The second approach is freezing the python packages using an external software, such as the PyInstaller [39], and thus packaging the program into standard executables. This approach consumes more time than the first but is easier for the CAD user. Other options include integrating a Python interpreter to the CAD system codebase and running the classifier model as is or porting the Python implementation as whole to C++. These options require the largest amount of development to get working and thus will not be considered as the approach of choice. These approaches are left as future development targets when the implemented classifier is honed for a later release version. Based on these factors, the first approach is chosen to be used during development, which is running the model in Python separately from the C++ executable.

To run the Python classifier separately from the C++ code, gRPC [40] is used. gRPC is a modern, free, lightweight and open-source communication protocol developed by Google. It is based on the Remote Procedure Call (RPC) framework and protocol buffers [40]. The more in-depth structure of gRPC is not part of the scope of this thesis.

Figure 4.1 presents a process diagram of using a CNN classifier in assembly constraint classification. The diagram is split into two categories by a dotted line: the training phase and the prediction phase. The training phase contains the data collection and feature extraction from the CAD model library, described in sections 3.2 and 3.1, respectively. The CNN model is presented as a black box model, meaning it's only observed by its inputs and outputs. The black box classifier is trained using the matrix presentations and ground truth labels from the earlier phase. In the prediction phase, the software user follows the process presented earlier in figure 2.6. The two selected elements and their respective CAD models are fed through the feature extraction algorithm, that outputs the matrix presentations. Compared to the training phase, in the prediction phase the feature extraction algorithm is not aware of the ground truth label for the constraint. Instead, the fully trained classifier is used for predicting this constraint label, that is then automatically placed between the two models as an assembly constraint. Otherwise, the prediction follows the same principles as the training phase.

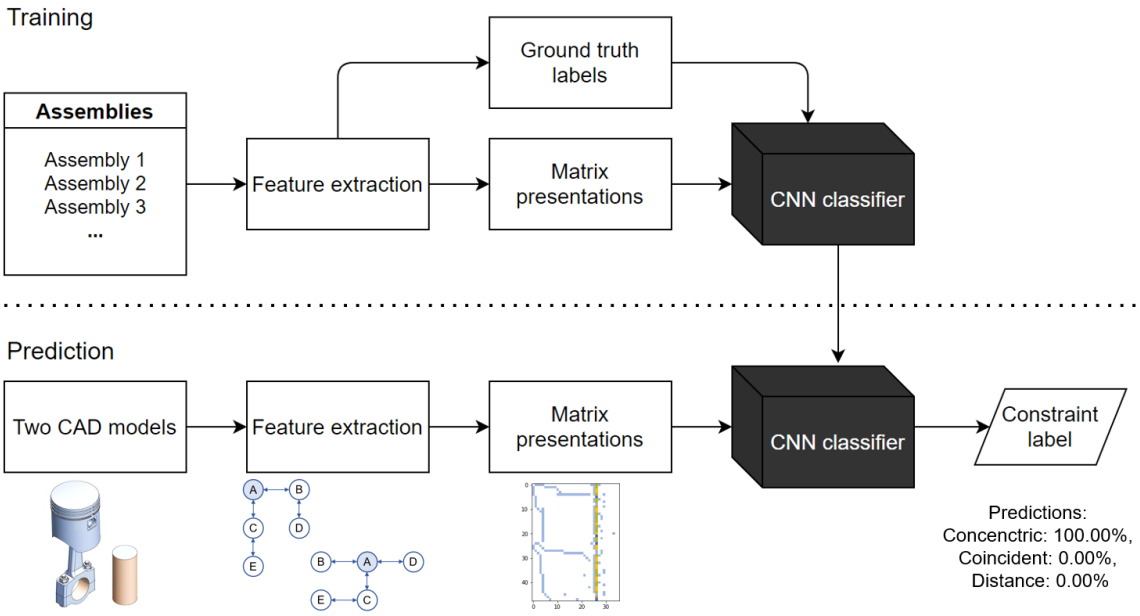


Figure 4.1. *The training and prediction process of the classifier.*

The feature extraction from the CAD models and assemblies is done in the CAD system source code in C++. In the initial implementation the extracted matrix presentations are saved to the system hard drive as comma-separated value files, from which they are read in Python. In the training phase all the matrices are read simultaneously, whereas in the prediction phase only one matrix is read at once.

4.2 CNN Architecture

In recent years, the trend in CNN architectures has been to create deeper networks rather than wider. Deeper CNN models have more layers with fewer filters per layer. [9] The network deepness is limited by the input size and the number of convolution and max pooling layers in the network. As presented earlier, nearest neighbor counts of $N = 24$ and $N = 12$ are used for the normalization of the adjacency graph. This results in input sizes of 48×36 and 24×24 to the classifier. These input sizes are relatively small, compared to e.g. image classification tasks, where the input size commonly is a 256×256 image [9]. Due to the size difference and the size diminishing feature of the convolutional layer and the max pooling layer, the maximum depth of the model is limited. However, the number of values in the matrix is also significantly smaller compared to image data. The number of values in the input matrices in this thesis are 1728 and 576 respectively, compared to the 65536 parameters for 256×256 images. Thus, the model depth requirement is diminished. The chosen model architecture for this thesis contains three convolutional layers, with 128, 128 and 256 filters respectively, totaling up to 512 filters and 512 * 9 = 4608 parameters with a filter size of 3×3. The classifier architecture is presented in figure 4.2. Other architectures with a larger quantity of convolutional layers were initially tested, but the presented model performs better in the application of assembly constraint classification. This is most likely due to the small input size to the network.

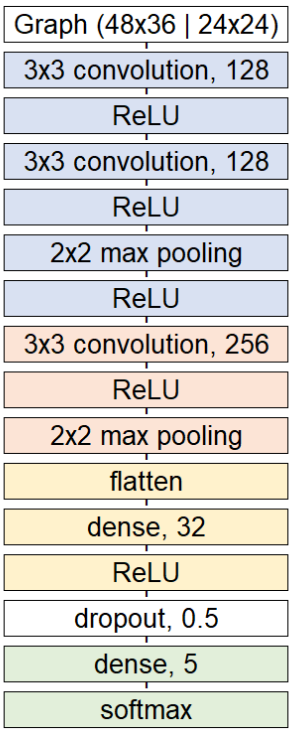


Figure 4.2. *Presented CNN classifier architecture.*

The network begins with the input matrix. The matrix is fed into a convolutional layer, the filter size of which is 3×3. The same filter size is used for all the following convolutional layers as well and will not be mentioned again in this section. The first convolutional layer has 128 filters. The layer is followed by a ReLU layer and a second convolutional layer that matches the first one. ReLU layers are used as the activation function between layers, as presented in subsection 2.1.2. A max pooling layer of size 2×2 is used to select the most important values found by the convolutional layer. Similarly to the size of the convolutional filters, the max pooling size is also fixed. The model up to this part is highlighted in a blue tone, disregarding the model input. The network is continued with a wider convolution layer, presented in an orange tone. This layer has 256 filters. The layer is followed by an activation layer and a max pooling layer.

A yellow tone is used to present the DL NN part of the CNN. The flatten-layer converts the multi-dimensional output of the CNN into a one-dimensional vector. The vector is then fed into a dense layer with 32 hidden neurons. The hidden layer is followed by an activation and the dropout layer, described in white. The dropout value 0.5 is selected based on its good performance to prevent over-fitting in the literature. Thus, 50% of hidden neuron output values are randomly set to zero to prevent over-fitting. The network ends with a green tone. A dense layer with neurons equal to the number of output classes is connected to a Softmax activation layer, that converts the class-wise outputs into values between zero and one. These values correspond to the probabilities of each class.

Equal network architectures are used for both input sizes, even though adding more layers for the larger input is possible. This makes the comparison of the classifiers trained on different input sizes more systematic and limits the variance caused by the model architecture, which allows the selection of the best model regardless of the input size.

4.3 Model Training

Before initiating the model training process, the trained models need to be defined. The division of data into three different datasets was presented previously in table 3.3. The original non-augmented data is used as a baseline to validate the effectiveness of the data augmentation method presented earlier in section 3.4. The effect of data skewness on the bias of the classifier is investigated by training a model on both the augmented dataset and the balanced augmented dataset, where each class has the same number of samples. To determine the relationship between the number of graph size and model performance, models are trained with nearest neighbor counts of $N = 24$ and $N = 12$. Permutations of these parameters are presented in table 4.1 and are used as the model definitions. Because the clipped dataset is only created from the augmented dataset and not the original dataset, six permutations of the dataset parameters can be created, which corresponds to six trainable models.

Name	Nearest neighbors	Augmented	Clipped
Model 1	24	Yes	Yes
Model 2	24	Yes	No
Model 3	24	No	No
Model 4	12	Yes	Yes
Model 5	12	Yes	No
Model 6	12	No	No

Table 4.1. *Definitions of the trained models.*

Hyperparameters are high level parameters that relate directly to the structure or performance of a ML model [4]. These parameters are tuned based on the literature and experiments and expectations of the system performance. Table 4.2 presents the relevant hyperparameters for the assembly constraint classifier model, and the selected values for those parameters. In medical image classification learning rate value of 0.0001 has resulted in better classification accuracies than the default value of 0.001 used by Adam [19] and Tensorflow Keras [38] on smaller batch sizes such as 64 [41]. Batch size describes how many data samples are used to train a single forward or backward pass. A value of 64 is selected based on its general good performance [41]. Both learning rate values are experimentally tested on a 20-epoch test training of a single model. Model two, presented in table 4.1, is used for the conducted test based on its large data quantity. On the initial test train, the higher learning rate of 0.001 resulted in 11.9% higher accuracy and 41.7% lower loss on the training dataset and 6.9% higher accuracy and 25.9% lower loss on the validation dataset, respectively. Based on these values, the 0.001 learning rate is selected to be used in this thesis. Each model is trained to 100 epochs to investigate the model performance over a longer training time. In addition, a longer training process presents a more in-depth description of model performance related to over-fitting. The exponential decay rates β_1 and β_2 and the ϵ value, that prevents division by zero, are selected to be the same as with the original Adam optimizer [19].

Hyperparameter	Value
Learning rate	0.001
Batch size	64
Epoch count	100
β_1	0.9
β_2	0.999
ϵ	10^{-8}

Table 4.2. *Essential hyperparameters and their values.*

Early stopping is a method to stop the model training process if a fitness metric of the model is not improving during the training process [4]. To research the model performance over the 100-epoch period and make the comparison of different models systematic, an early stopping is not used directly during the training process in this thesis. However, keeping track of the point of lowest validation loss is used and model checkpoints are saved at points of lowest validation loss. This provides the means for returning to these points later and the comparison of performance metrics at these points, as well as the training times to reach these points. These comparisons are done later in section 5.2.

The three datasets presented earlier in table 3.3 are each divided into a training, validation and testing dataset. The training set is used to fit the classifier model and it is the data the model learns from. The validation set is used for model evaluation simultaneously to the training process. The model is not trained on this data and thus the validation set does not directly affect the model performance. The testing set is only used to evaluate the performance of the final, fully trained model. [12] To retain the maximum number of training set samples, an 80%, 10% and 10% split is used to divide the data into the training, validation and testing sets, respectively. Each dataset is randomized to reduce the inner bias learned by the classifier [4]. The training and validation sets are used during the training process in this section, while the testing set is used later in section 5.2.

The accuracies of popular image classification systems trained on ImageNet [42] dataset are benchmarked using two parameters: top-1 accuracy and top-5 accuracy [43]. In top-k accuracy ($k = 1$ or $k = 5$ in this case) the k most probable labels given by the CNN are compared to the correct ground truth label. If any of the top-k labels is correct, the prediction is identified as correct [16]. Due to the nature of assembly constraints described previously, top-k accuracy provides a more accurate description of the model. While a value of five is commonly used for k, due to the low number of target classes for the classifier, a lower value of two should be used instead. Top-2 accuracy is rarely used in the literature around CNNs, but there are some studies such as [44], that use it in combination with top-1 accuracy due to fewer target classes. In this thesis, the top-1 accuracy, referred to as accuracy later in this thesis, is combined with the top-2 accuracy due to there only being five constraint classes to classify between.

Figure 4.3 describes the model accuracy (top), top-2 accuracy (middle) and loss (bottom) on the training dataset. For the accuracy plots, the higher the value is, the better the model performance is. For the loss plots lower values are preferred. Categorical cross-entropy (Softmax) is used as the loss function, as explained in subsection 2.1.1. Each of the six models presented in table 4.1 is trained for 100 epochs and the epoch-wise metrics are drawn in blue, orange, yellow, green, grey and purple colors for the six models, respectively. On the accuracy plot, the augmented grey and orange models' series ascend quickly to the approximately maximum value resulting in a steep curve. The baseline purple and yellow models reach a lower accuracy at a relatively similar rate, while the

augmented and balanced blue and green models achieve the lowest accuracies. It can be seen, that the nearest neighbor counts of $N = 12$ perform better than their $N = 24$ counterparts in the accuracy department. The top-2 accuracy plot is similar to the accuracy plot, where the yellow, blue and green models' performances remain comparable, yet lower than the other three models'. The values for the top-2 accuracy are always higher than or equal to the accuracy, when compared on the same epoch values. On the loss plot, the order of the model performance is similar to the accuracy plots. Overall, the augmented models perform the best during the training process followed by the baseline models. The augmented and clipped datasets perform the worst.

Figure 4.4 describes the model accuracy (top), top-2 accuracy (middle) and loss (bottom) on the validation dataset. The accuracy and top-2 accuracy follow a similar trend to the models trained on the training dataset. The main difference is that the models reach higher accuracies faster than on the training dataset. This is caused by the dropout layer used on the training dataset models. On the validation dataset all the hidden neuron values are used because over-fitting is not an issue, and thus the model performance is better initially. In addition, there is more fluctuation in the model accuracies on the validation dataset. The model validation loss has the most fluctuation out of all the plots. The point at which the model validation loss starts consistently increasing is the point where the model starts over-fitting. This is a problem especially on a smaller dataset, such as the models with no data augmentation [45]. The bottom plot depicts that the model two, highlighted in orange, performs best on the model validation loss compartment, with only little increase in the loss over the 100 epochs. All the other models perform similarly, with high fluctuation in the validation loss value over the presented scope.

Table 4.3 presents the model training and validation accuracies and losses at the point of the minimum validation loss. Due to the high fluctuation, the minimum validation loss is calculated by comparing each loss value to the five next consecutive values and taking the first point, at which there have been five consecutive increases in the loss value compared to the original value. Top-k accuracy is left out of this table due to space constraints and because the training and validation accuracies provide more variation in the description of the model performance. The table shows that at the point of minimum validation loss the model validation accuracies are slightly higher than the training accuracies, suggesting that at that point in the training process the models are slightly under-fitted. Interestingly, the fifth model, which has performed nearly as well as the second model on the previous metrics, starts over-fitting soon after epoch 12. This conveys that it is not necessary to train all the models for the whole 100 epochs and sufficient results can be reached with a lower epoch count. The table shows that even the model two with the largest dataset and data presentation gains its lowest validation loss at 19 epochs, which is a similar rate to the other models. However, the performance is better on both the training and the validation metrics compared to the other models. To keep the model discussion

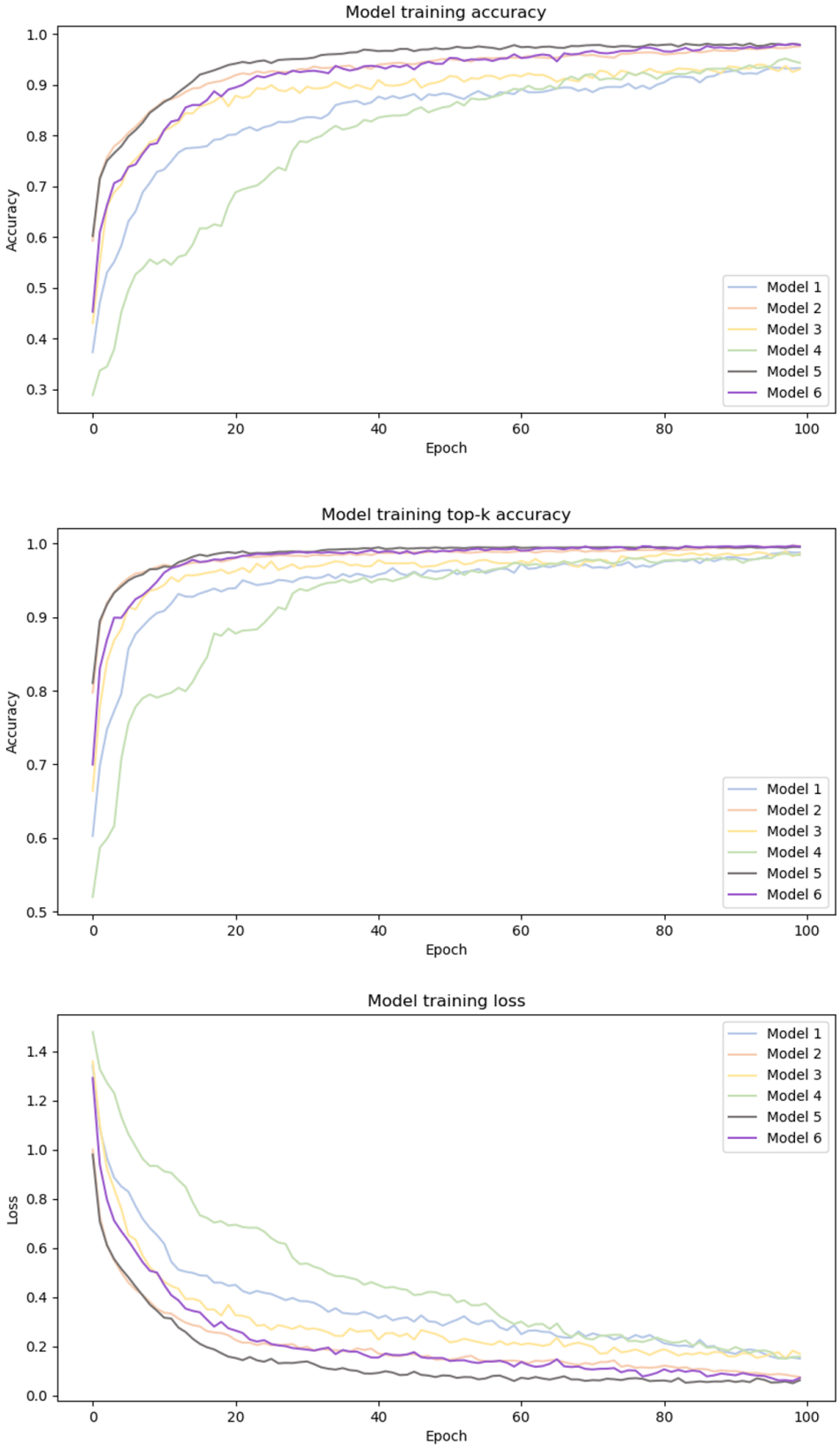


Figure 4.3. *Comparison of model accuracy, top-k accuracy and loss on the training dataset.*

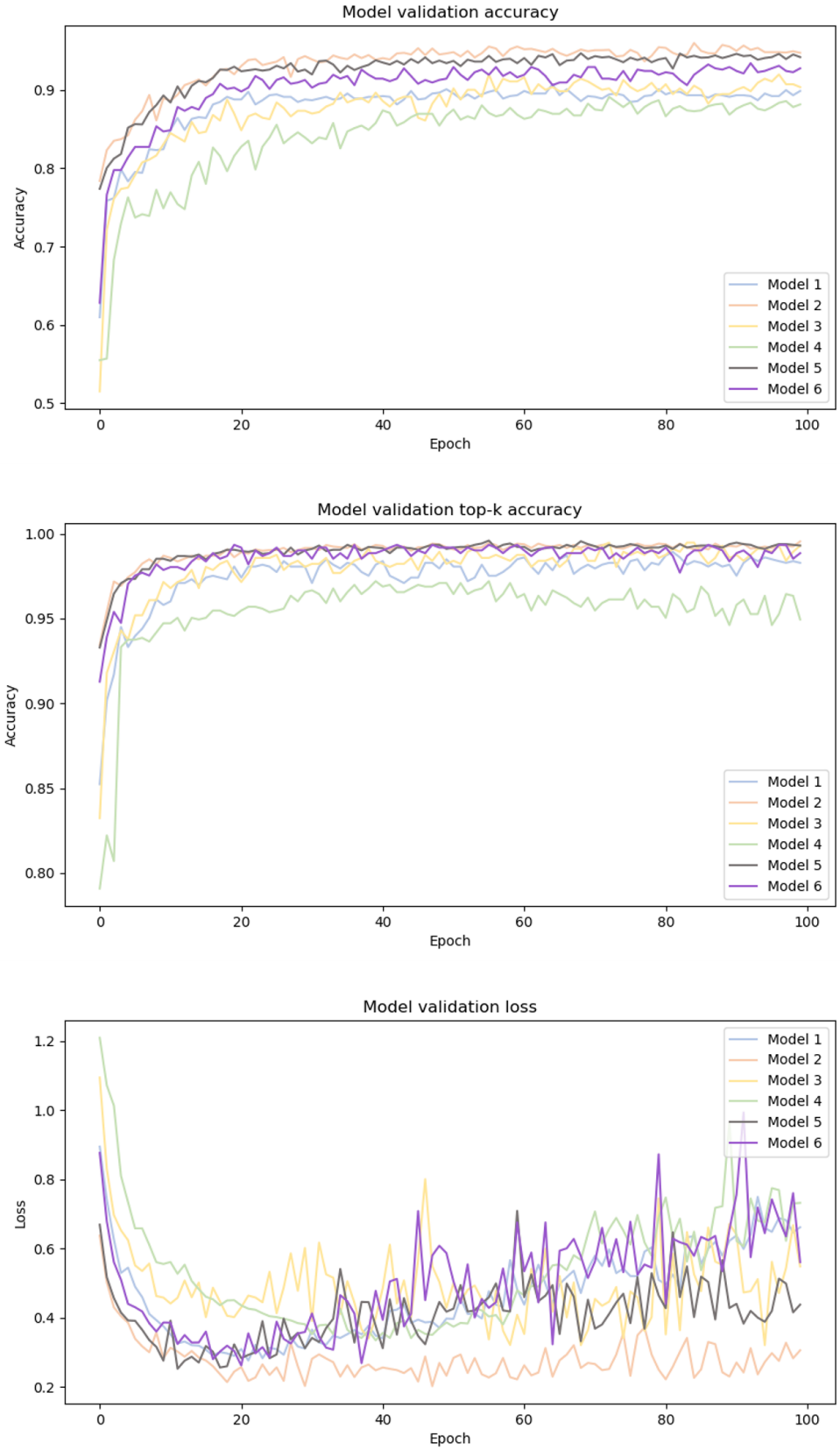


Figure 4.4. *Comparison of model accuracy, top-k accuracy and loss on the validation dataset.*

Name	Epoch	Training acc.	Training loss	Validation acc.	Validation loss
Model 1	22	0.8113	0.4253	0.8976	0.2760
Model 2	19	0.9074	0.2547	0.9268	0.2137
Model 3	20	0.8573	0.3691	0.8699	0.4015
Model 4	40	0.8276	0.4613	0.8578	0.3344
Model 5	12	0.8737	0.3135	0.9046	0.2521
Model 6	21	0.8947	0.2647	0.8980	0.2618

Table 4.3. *Model metrics at the point of minimum validation loss with five epochs of patience for the early stopping algorithm.*

Name	Epoch train time (s)	Train time to minimum loss (s)	Total train time (s)
Model 1	88	1936	8800
Model 2	216	4104	21600
Model 3	53	1060	5300
Model 4	26	1040	2600
Model 5	75	900	7500
Model 6	17	357	1700

Table 4.4. *Model training time comparison to the point of minimum validation loss and to the point of 100 epochs.*

systematic further on in this thesis, the models are examined over the whole 100-epoch period instead of comparing them further at the point of minimum validation loss.

The model is trained on a system consisting of an Intel i5-9600k processors, a GTX 970 graphics card and 16 gigabytes of RAM. Table 4.4 presents the training times per epoch, the training times to reach the previously described points of minimum validation loss and the total training times to the point of 100 epochs. The training time increases linearly between models trained on the nearest neighbor counts $N = 24$ and $N = 12$. Models with the higher nearest neighbor count take longer to train than their counterparts with fewer nearest neighbors.

4.4 User Interface

A User Interface (UI) allows the software user to pass information to and interact with a computer system [46]. The usability of the software implementation is the main focus in the initial UI design of the classifier system. Usability is described as the extent to which a system can be used by its users to achieve the goal of assembly constraint inputting with effectiveness, efficiency and satisfaction in the use setting [47]. The UI is designed to reduce the number of dialog boxes the user must go through during the assembly process.

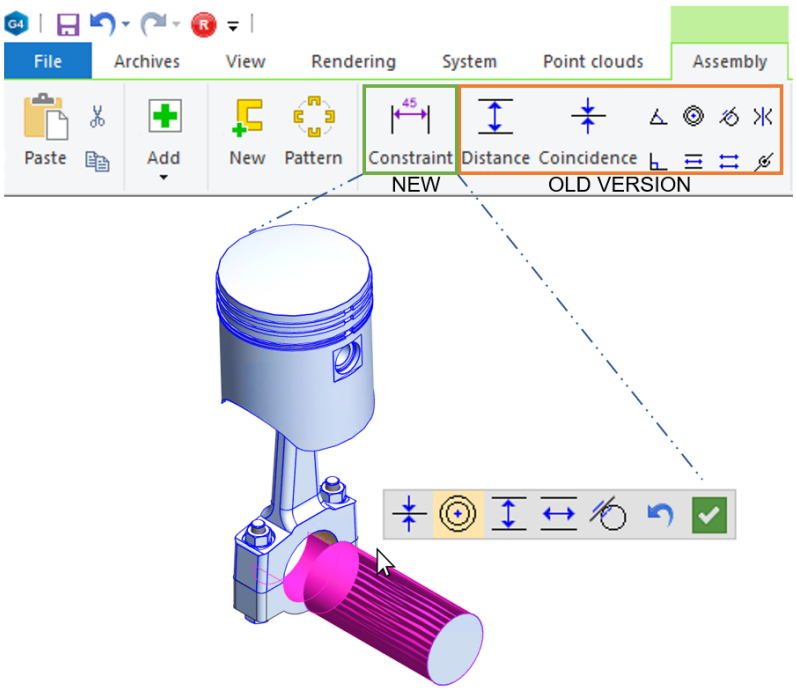


Figure 4.5. *Initial user interface for the classifier system inside the Vertex G4 CAD software.*

The main problem with the former interface was that the user had to repeatedly start and close functionalities when they wanted to switch between different constraint types. This is visible in figure 4.5, where the buttons highlighted in orange are used to start the input of different assembly constraints in the old software version. In the introduced UI all the different constraint types can be placed through a single functionality. The "Constraint"-button, highlighted in green, is used to start the new ML-based functionality. After the user has selected the elements, between which an assembly constraint is required, the data in transferred to the CNN classifier using gRPC. The classifier responds with the probability values of each constraint type that can fit between the selected elements. Then, the predictions go through a filtering system, that removes the predictions that are invalid between the elements based on their geometric form or that already exist between the elements. The most probable of the remaining constraints is automatically applied between the elements. Simultaneously the CAD parts are animated and moved to their corresponding places. In addition, a small ribbon is shown to the user at the location of their cursor, where different constraint types are shown, and the applied constraint is highlighted in yellow. This allows the software user to change the constraint type fluently, if necessary, and reduces the required amount of mouse movement. The ribbon can also be used to undo previous constraints or to stop the assembly functionality altogether. In an ideal case the ML algorithm can predict most of the constraints correctly and the incorrect classifications are corrected using the geometric filtering system in the CAD software.

5. RESULTS AND DISCUSSION

5.1 Evaluation Criteria

Before evaluating the results, it is important to determine a foundation for the criteria of evaluation. The end goal in the assembly process is to remove DOFs until the CAD designer reaches the desirable state. Some constraints, such as the coincident constraint, are more commonly used and remove more DOFs than others and fit between a larger variety of elements. Model accuracy is the most used metric for a classification model's performance [48]. The accuracy works best to characterize a model that has been trained on an evenly distributed dataset. With skewed data, other metrics are used to gain more in-depth information of the performance. For example, if majority of the data belongs to one class, the model gains high classification accuracy by always predicting that a sample belongs to that class. In such a case, the accuracy gives a falsely positive portrayal of the state of the model. A confusion matrix is a $C \times C$ matrix, where C is the number of classes, that holds four main parameters: the number of true positive predictions, the number of true negative predictions, the number of false positive predictions and the number of false negative predictions. True positives and true negatives are correct classifications of a sample either belonging or not belonging to a certain class, respectively. False positives are samples that were incorrectly classified as belonging to a class, while false negatives were not recognized belonging to a class. The significance of these values for the case of assembly constraint classification is discusses later in section 5.2. [48]

For multi-class classification, such as the instance of assembly constraint classification in this thesis, there are five commonly used performance metrics for a systematic analysis: accuracy, error rate, precision, recall and F-score. Three of these measures: precision, recall and F-score, are averaged using micro-averaging, which considers the distribution of the data and thus favors larger classes, and using macro-averaging, which is the traditional averaging method. [48]

Model accuracy describes the average relation of correct classifications to the incorrect classifications. Using the four confusion matrix values it can be presented in mathematical terms: [48]

$$\frac{\sum_{i=1}^{i=n} \frac{TP_i + TN_i}{TP_i + FN_i + FP_i + TN_i}}{n} \tag{5.1}$$

where TP is the number of true positive, TN is the number of true negative, FP is the number of false positive, FN is the number of false negative values and n is the number of classes. The error rate of a model is directly related to the accuracy and describes the average number of classification errors. The formula for error is equal to one minus accuracy and can be presented as: [48]

$$\frac{\sum_{i=1}^{i=n} \frac{FP_i + FN_i}{TP_i + FN_i + FP_i + TN_i}}{n} \tag{5.2}$$

Precision is a metric, that describes the quantity of correct positive classifications in relation to the total number of positive classifications. It presents the proportion of positive predictions that were correct. Micro precision P_μ is calculated based on sums of per-sample values, while macro precision P_M is calculated as an average of per-class precisions. They are defined as: [48]

$$P_\mu = \frac{\sum_{i=1}^{i=n} TP_i}{\sum_{i=1}^{i=n} TP_i + FP_i} \tag{5.3}$$

$$P_M = \frac{\sum_{i=1}^{i=n} \frac{TP_i}{TP_i + FP_i}}{n} \tag{5.4}$$

Recall, also called the model sensitivity, describes the ratio of correct classifications to the total number of positive data samples. It presents the proportion of positive ground truths that were identified correctly. Micro recall R_μ and macro recall R_M are similar to the precision metrics, but instead of the false positive values the false negatives are used, such that: [48]

$$R_\mu = \frac{\sum_{i=1}^{i=n} TP_i}{\sum_{i=1}^{i=n} TP_i + FN_i} \tag{5.5}$$

$$R_M = \frac{\sum_{i=1}^{i=n} \frac{TP_i}{TP_i + FN_i}}{n} \tag{5.6}$$

Precision and recall are both invariant to the number of true negative classifications. Improving of precision generally decreases the recall and vice versa. F-score F is a performance metric that combines the precision and recall values to summarize a model's

performance into a single metric. The F-score is valid for comparing constraint classifiers trained on similar datasets. However, as described later, there is a slight difference between false positives and false negatives especially on some of the constraint classes. Whereas in e.g. medical classification this difference is magnified, and a false negative can lead to not doing further research on a disease, while a false positive can lead to unnecessary testing, and thus the importance of false negatives is higher than the false positives. Micro F-score and macro F-score are calculated using the same formula, where the precision and recall are used with the μ or M subscript based on the case, such that: [48]

$$F = \frac{2PR}{P + R} \tag{5.7}$$

Equations 5.1 - 5.7 will be used to calculate performance metrics for each trained classifier and used as a baseline for their comparison. All the metric values belong to range $[0.0, 1.0]$. High accuracy, precision, recall and F-score values refer to high model performance.

5.2 Classification Model Validation

The goal of this section is to compare the six trained classification models' performance metrics. First, the confusion matrices are presented and analyzed to obtain understanding of how the matrix values relate to the classification performance of the models. Second, the confusion matrices are used to calculate the performance metrics discussed in equations 5.1 - 5.7 and are presented side by side for comparison. In addition, the slowest of the presented model's performance is measured by clocking the duration of different steps in the implementation and the results are generalized to cover the other models' performance as well.

Figure 5.1 presents the normalized confusion matrices for the trained models. The confusion matrix describes the ratio of ground truth labels on the y-axis to the classifier predictions on the x-axis. The normalization is done by scaling the values on the prediction axis between zero and one. The diagonal values correspond to the class-wise recall values of the classifier. A perfect classifier predicts 100% of the the class labels correctly, resulting in an identity matrix. In analyzing the confusion matrices, most focus is placed on the model's ability to differentiate distance constraints from coincident constraints, which depicting the confusion matrices, is the most major difference between the different models. Values outside the diagonal on the vertical axis are interpreted as false positive values, as presented earlier in section 5.1. Respectively, the values on the horizontal axis are interpreted as false negatives. Both are bad for the performance of the classifier, and thus the model that minimizes these values is the optimal choice for the classifier.

The model performance in relation to other values than distance and coincidence is nearly equivalent in figure 5.1. The models can be roughly divided into two categories. Models one and four have approximately the same number of false positives and false negatives, within $\pm 5\%$. This means that the model classifies distance constraints as coincident constraint approximately as often as vice versa. Models two, three, five and six form the second category, where the number of coincident constraints classified as distance constraints is negligible, but some of the distance constraints are classified as coincident constraints. At this point, it is important to discuss a limitation in the dataset creation. Sometimes Vertex G4 CAD users prefer to use distance constraint with a value zero instead of coincidence constraints. Thus, some of the values where the ground-truth value is distance and the prediction is coincident, are falsely too high. However, correcting these values manually is impossible because the dataset does not contain the constraint distance values. It is expected that each of the model datasets has a relatively equal number of these constraints proportionately to the dataset size, and thus this limitation does not hinder the ability to compare the models. This is further discussed in section 5.4.

Overall, model two performs the best out of the six models. It has the highest diagonal values with only 13% of the distance constraints classified as coincident. Model five performs relatively similarly, with a 21% value for the respective miss-classifications. Models three and six follow in performance, with a larger number of coincident constraints classified as distance constraints. The effect of dataset balancing can be seen in the performance of models one and four, in which 17% and 21% of coincident constraints are classified as distance constraints, respectively. This is caused by the absence of model bias on models one and four, whereas the other models' biases aid the classifier by weighing the coincident constraint more heavily.

The confusion matrices are used as the basis for the metrics calculations. The accuracy is calculated using equation 5.1, the precisions and the recalls using equations 5.3 - 5.4 and 5.5 - 5.6, respectively, and the F-score using equation 5.7. These metrics are presented in figure 5.2. The metric values correspond to the earlier confusion matrix analysis. Models two and five achieve the highest performance values on all the metrics, followed by models three and six. Model two reaches a performance of equal or over 95% on each metric. Respectively, models one and four trained on the balanced dataset perform the worst, reaching under 90% performance on each metric. The best performing model, model two, performs on average 6.2% better than the worst performing model, model four. The averaging is done over all the metrics simultaneously. The increased data quantity and the corresponding generalization ability achieved through the presented augmentation method leads to the superior performance of models two and five. While the models one and four are trained on more data than models three and six, the generalization ability is diminished by the non-existent bias. The metrics are discussed further in section 5.3.

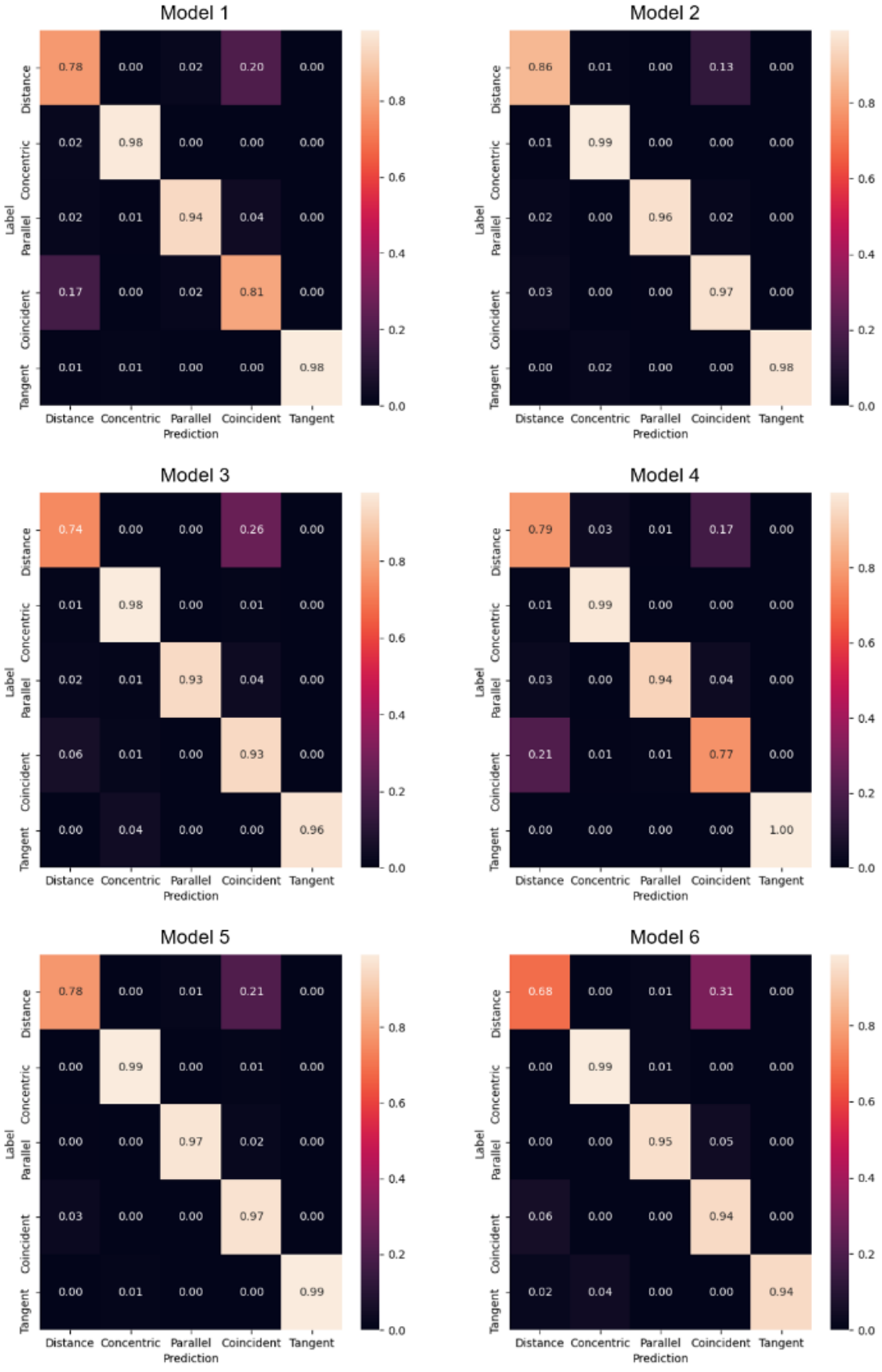


Figure 5.1. *Comparison of normalized model confusion matrices.*

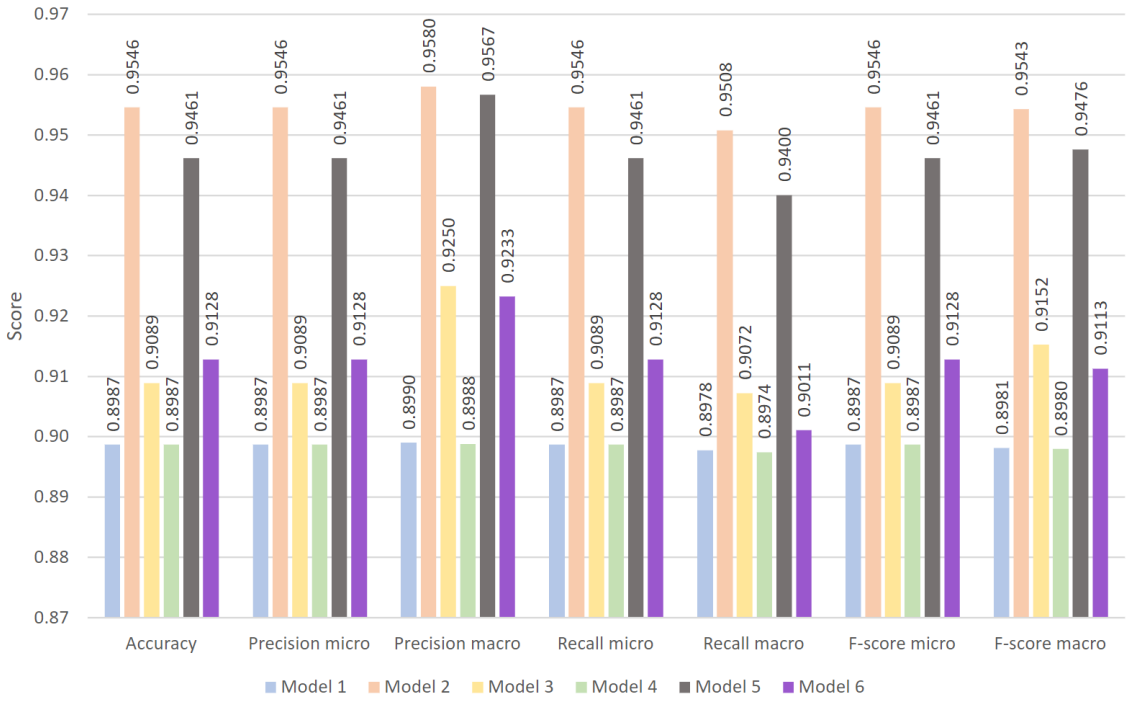


Figure 5.2. *Side by side comparison of model test set accuracy, precision, recall and F-score.*

It is debatable, that classifying distance constraints as coincident constraints is better than vice versa, based on three reasons. First, the coincident constraint is more widely used by the CAD users, as is demonstrated previously in figure 3.4. Second, the coincident constraint is generally more applicable especially between planar elements than a distance constraint. And third, based on the zero-distance constraint usage mentioned before, the number of distance constraints classified as coincident constraint cannot be zero for even the best model without over-fitting. Based on these arguments, the minimum value for the false positives especially on the coincident constraint is preferred, which as described by equations 5.5-5.6, equals to a higher recall value. Table 5.1 presents the model recall values by assembly constraint type. The coincident class recall values for model two and model five are emboldened to emphasize the highest values in the class. The concentric, parallel and tangent constraints have similar performance between models, within 6.3%. The distance and coincident constraints have a lot greater variation and the distance class recall values can be used to differentiate between the best performing models two and five. Model two performs around 9.9% better than model five on the distance recall, which is a significant difference compared to the average performance difference on all the metrics between the best and worst models.

To validate if the model is feasible for fluent CAD design workflow, the speed of the model needs to be measured. The model two is used for the measuring due to its good performance and its larger input matrix size of the two choices. Thus, the model classification time is higher, than the models with the smaller input size. Respectively, if it

Name	Distance	Concentric	Parallel	Coincident	Tangent
Model 1	0.7784	0.9786	0.9379	0.8092	0.9848
Model 2	0.8571	0.9911	0.9617	**0.9677**	0.9764
Model 3	0.7391	0.9780	0.9333	0.9292	0.9565
Model 4	0.7884	0.9897	0.9351	0.7740	1.0000
Model 5	0.7800	0.9913	0.9727	**0.9668**	0.9895
Model 6	0.6824	0.9863	0.9541	0.9414	0.9412

Table 5.1. *Model recall values by assembly constraint type. The emboldened values are the highest recall values in the coincident constraint class.*

can be validated that this model is applicable to a fluent workflow, the rest of the models can be determined to be as well. The measurements are taken on a medium sized CAD assembly by clocking the three phases of the classification: the data collection and data saving phase, the gRPC communication and classification phase and the constraint adding phase. These groupings are based on the software components related to each task, which makes clocking these tasks efficient. In the measurement process 20 assembly constraints are added between different elements and each of the three phases are clocked using std::chrono C++ library and documented for each of the constraints. Based on the measurements, the data collection and saving to the system hard drive takes the lowest amount of time, averaging to 24.6 ms. The process of C++ to Python communication using gRPC and the constraint classification process take the longest, averaging to 102.0 ms per constraint. This is an adequate classification duration for real-time CAD design, which makes it unnecessary to do a more in-depth trade-off analysis between the different models on the classification speed department. Adding the constraint between the elements takes on average 44.2 ms by the CAD system. In total, the average time to add a constraint using the presented system is 170.8 ms. At minimum adding a constraint took 94 ms, at maximum 256 ms and the standard deviation was 41.9 ms.

The presented ML-based system takes on average 3.9 times the amount of time to add a constraint than adding a constraint entirely by the program. However, this increase in duration to add a constraint is insignificant, because there are multiple points in the workflow of the CAD design process where the user saves time using the novel system. These include thinking of a constraint type, selecting the functionality, possibly opening, and closing a constraint dialog box depending on the constraint type and stopping and restarting the process if the constraint type needs to be changed. To have a point of comparison, the time to open and close a dialog box for the constraint adding was measured by the writer through repeating the operation 20 times. On average opening and closing a dialog box took 1068 ms, with a standard deviation of 298 ms. While this number is influenced by the personal skills of an average CAD user, it gives a point of comparison to the usability of the classifier.

5.3 Discussion

The thesis trained six CNN models on different datasets collected from professionally designed CAD assemblies. Model two, which was trained on nearest neighbor count $N = 24$ on the augmented dataset, outperformed the other models on each of the calculated confusion matrix-based metrics. In addition, it achieved the lowest and the most stable validation loss. The results suggest that model two is the most suitable for assembly constraint classification. This can be validated by comparing the model to the baseline models trained on the original datasets. Compared to the baseline models three and six, model two achieved on average 4.7% and 4.6% better performance, respectively. The recall value was determined to be more important than the other metrics. Model two outperformed the baseline models by 5.0% and 4.6% on the micro recall and 4.8% and 5.5% on the macro recall. This demonstrates a clear correlation between data quantity and model performance, but the distribution in the data is important as well. This is established by the lower performance metrics of the first and fourth classifier model, trained on the balanced datasets.

The metrics presented earlier in figure 5.2 visualize that the micro precision, recall and F-score values for the performance metrics of a model are the same. This is a characteristic of using the micro values in multi-class classification, and is caused by each prediction error of class A classified as belonging to class B, where A and B are arbitrary classes, being a false negative for A and a false positive for B. These values also equal to the model accuracy. Thus, the macro metrics provide more variety to the model performance.

To combat the classifier bias caused by using pre-existing assemblies, an experimental approach was selected to balance the constraint datasets by performing a clipping operation. The performance metrics indicate that this approach is not recommended for the case of assembly constraint classification. Based on the distribution of the data collected during the data collection phase, the coincident constraint is the most used constraint with nearly as many cases as the other constraints combined. While this skewness in the distribution effects the classifier's predictions by favoring the coincident constraint over the others, this bias is considered positive for the classifier's performance. Because the data has been collected from professionally designed CAD assemblies, the distribution is able to be generalized to cover a variety of other CAD assemblies as well. The results indicate that removing the bias from the dataset using the presented methodology diminishes the classifier accuracies and increases the categorical loss significantly on both the training and validation datasets.

The hypothesis, which stated that the models trained on nearest neighbor counts $N = 12$ would perform better than the models trained on nearest neighbor counts $N = 24$, was deemed to be only partially correct. For the hypothesis to be correct, the models four, five and six would have trained faster and exceeded the performance of models one, two and

three due to fewer parameters and the expectation that the most important parameters for the selection of an assembly constraint would be located within the 12 nearest elements from the selected elements. Contrary to the hypothesis, model two performed better than model five on the validation accuracy and validation loss. Of the models trained on the augmented and balanced datasets, model one trained faster and reached higher accuracies than model four. While the classifier solves the problem of assembly constraint classification successfully with the used nearest neighbor counts, the performance increase caused by the higher neighbor count on some models presents an opportunity for future research. Nearest neighbor counts $N = 36$ and $N = 48$ are logical next options to explore. The model two performance compared to the model five performance, especially on the validation loss department, signals that the higher neighbor count makes the model less prone to over-fitting.

The thesis determined that assigning importance ratings to edges of the adjacency graph, similarly to what was done by Gilsing et al. [35], is difficult for the case of a CAD model. Thus, a similar approach to the one used by Shi et al. [6] was selected. The main difference in the approach was to rank each node in the adjacency graph based on their depth from the single central node, which was the element selected to be a part of the assembly constraint by a professional CAD designer. The approach led to a random element, caused by equal depth values on multiple nodes. This random element opened a possibility for a novel augmentation method, that to the best of the writer's knowledge, has not been used in a similar context before. Compared to the graph data augmentation methods discussed by Zhao et al. [37], this thesis discussed a graph data augmentation algorithm using BFS and randomization in situations where exact normalization of graph data is difficult. The main difference between these approaches is that in the latter, the graph data remains unaltered during the augmentation process. While the approach used in this thesis is not usable for data augmentation on graph neural networks, it is valid for the case of converting the adjacency graph data into a normalized adjacency matrix based around a single central node.

The presented augmentation method was divided into two sections. Four variations of the neighborhood were collected and combined with the bidirectional stacking of the two CAD models to achieve the eight augmented final matrix presentations. Combining the augmentation algorithm with an adequate data presentation, such as model two, results in a converged model with less over-fitting than corresponding non-augmented cases on relatively low quantity of training data. The results of using the data augmentation algorithm over the baseline methods indicate, that the algorithm is successful at improving model performance in situations where exact normalization of graph data is difficult.

A central goal of the designed classifier is to reduce the time the CAD user takes to go through the steps of assembly constraint adding. Based on the measurements taken and the overall accuracy of the classifier, it can be stated that the classifier fulfills this goal

Study	Method	Accuracy	Classes
Zhang et al. [3]	Voxels	97.40%	24
Shi et al. [6]	Graphs	98.80%	11
Dekhtiar et al. [4]	Image-based	82.81%	30
Hegde and Zadeh [8]	Voxels and image-based	93.11%	10
Manda et al. [9]	Image-based	95.63%	43
This thesis	Graphs	95.46%	5

Table 5.2. *Recap of CAD model related classification results achieved with different data presentations and different number of target classes.*

adequately. As presented earlier, the total duration to classify a CAD assembly constraint is 170.8 ms on average on a reasonably average system, which is sufficiently fast for fluent CAD design. The presented software implementation solves the identified issues in the original system, and thus improves the usability of the software during the assembly process.

Comparing the achieved classification results to literature is difficult, due to there being little to no prior studies on assembly constraint classification. A major limitation for the model performance was the data quantity, which is clearly presented by the performance of the augmented models compared to the other models. This limitation also makes the comparison to other CAD model classification applications difficult. As presented earlier, the graph and voxel-based applications are commonly trained with 5000 to 6000 per class data samples [6][3]. Whereas in this thesis, the classes have on average 561 to 4374 samples as presented previously in table 3.3. However, these averages are heavily impacted by the large number of coincident constraints. For reference, these values vary between 401 and 3066 when the coincident constraints are not considered. In either case, the number of samples is low. Correspondingly, the number of classes is also lower than in other CAD model classification studies. Table 5.2 recaps the accuracies and number of classes for different methods in addition to the results achieved in this thesis. The comparison is based on the results of the literature review presented earlier in section 2.2. The comparison provides insight into the model performance, but it is important to note that many of the other studies had a larger number of target classes, which makes the classification task more difficult. The method presented in this thesis provides comparable results to the image-based classification results and is bypassed by the voxel and graph-based methods considering the sheer classification accuracy.

5.4 Future Improvements

While the classifier performance in the assembly design process has been validated earlier in this thesis, the software is not ready for deployment as is. The main future improvements required are related to the software system C++ to Python communication, the classifier's ability to differentiate coincident constraints from distance constraints and the initial UI.

As it was proven in section 5.3, gRPC is valid for real-time communication between C++ and Python. The problem with the current implementation is that it requires the user to manually install a Python interpreter. To make the software as easy to install as possible, this approach requires improving. Different approaches have been discussed earlier in section 4.1. In addition, improving the file transferal system to Python can be improved. The current implementation relies on saving the matrix presentations on the system hard drive, whereas direct transfer without physical saving would be preferred.

The classifier's ability to differentiate coincident constraints from distance constraints is hindered by illogical behaviour of CAD users on the assemblies used as training data. In the history of Vertex G4, the type of CAD constraint was unable to be changed after the constraint had been applied. Thus, some of the clientele used distance constraints with distance value zero instead of coincident constraints for future-proofing the designed assemblies. This behaviour is still precedent in some of the senior CAD designers and the zero-distance constraint was used within some of the assemblies in the created datasets. The current structure of the classifier makes the classifier unable to differentiate these cases correctly and causes uncertainty in the behaviour of the classifier in relation to these two constraint types. This especially affects the model comparison metrics of precision, recall and F-score because it causes inaccuracies in the number of true negatives and false positives. To improve this behaviour in the future, a pre-processing step should be added to the classifier pipeline. Each zero-distance constraint ground truth should instead be saved as a coincident constraint. However, this change would require a total recollection of the datasets and a retraining of the classifiers and is thus left as a future improvement.

The UI designed during the thesis is an initial version that is mainly designed to validate the convenience of the classifier in a real-time use situation. The usability of the UI can be improved through small improvements, such as disabling the constraints that are not available in certain situations. In addition, the CAD users' attitude towards the novel UI design choices can be validated by customer surveys and improved based on user feedback.

6. CONCLUSIONS

6.1 Summary

The goal of this thesis was to present a novel method for automating the CAD assembly process by classifying the assembly constraint types between two geometrical elements that collectively comprise an assembly constraint. The thesis provided solutions to the three stated research questions and contributed to the research around CAD model related classification using DL and CNNs. At the time of writing the thesis, the state of assembly constraint classification was deficient in the literature. To the best of the writer's knowledge, the problem of selecting an assembly constraint between two geometrical elements of two CAD models using ML had not been previously addressed in the literature.

The thesis aimed to identify the most relevant information to be extracted from a CAD model for the purpose of teaching an assembly constraint classifier. Based on extensive literature review and empirical research, it was concluded that the adjacency matrices of both CAD models belonging to an assembly constraint, combined with the geometrical element type information, the geometrical form and DOF data, provides an adequate depiction of a CAD model. The classification results indicate that the presented method captures the relevant information of the CAD model and presents the constrained elements, and their respective neighborhoods, in a compact and efficient format. The element neighborhood presentation was normalized using the BFS algorithm, which provided the ability to augment the data quantity through randomizing the exploration order and retaining depth information. Based on the findings, the data augmentation algorithm can be used to increase the data quantity by eight-fold. Respectively, this increase in data quantity improved the classifier performance on the confusion matrix metrics by up to 5.5% and diminished the validation loss calculated using categorical cross-entropy.

In this thesis, algorithms for assembly constraint collection from assemblies and adjacency graph creation from volumes were developed. The output of the algorithms combined with the utilized feature vectors facilitate the use of pre-existing CAD assemblies in the creation of datasets for the classifier development. This approach enabled the utilization of the proficiency of CAD designers, provided a method to avoid the laborious manual design process and provided advantage over geometric variation due to the variety of parts used in assemblies. While this approach caused the classifiers to learn bias,

the bias was demonstrated to be beneficial to the classifier performance by comparing the results to the results of classifiers trained on datasets with a balanced distribution of constraint classes. The models with no bias learned to misclassify distance constraints as coincident constraints significantly more often than the biased models.

The presented theory was used in practice to automate the time-consuming routine task of assembly constraint applying to support CAD designers by decreasing the design time and enhancing the user experience [2]. As part of the thesis, an initial software implementation of the classifier system was created into a mechanical CAD software, Vertex G4. By measuring the average duration to classify a constraint from within the software system, this thesis has validated that the presented classifier can effectively be incorporated to the fluent CAD design workflow. The combined duration of C++ to Python communication using gRPC and the constraint classification using the best performing model was calculated to be 102 ms on average per assembly constraint on the test system, which is adequate for real-time CAD design.

To summarize, the main new contributions of this thesis in a descending order of importance are:

1. Training six CNN classifier models on different datasets collected from professionally designed CAD assemblies and comparing the models based on common performance metrics, such as the test set accuracy, precision, recall and F-score. In addition, exploring an experimental approach of bias removal from the data distribution through this comparison.
2. Presenting a method for collecting relevant data from a CAD model into a matrix presentation that is valid for assembly constraint classification.
3. Implementing an assembly constraint classifier into Vertex G4 CAD software and validating its performance in real-time CAD design workflow based on measurements.
4. Providing a method for graph data augmentation using BFS and randomization in situations where exact normalization of graph data is difficult.
5. Collecting and comparing the CAD model related classification applications from the literature.

The importance order is based on how essential each contribution is for answering the research questions. The contributions that are directly related to answering the research questions are rated more highly. The last two contributions are an important by-product that are necessary for answering the research questions. In addition to the contributions, it is important to discuss the limitations of the thesis. To summarize, the most important limitations of this thesis in an arbitrary order are:

1. The developed classifier's inability to occasionally differentiate coincident and distance constraints, due to an error in the dataset generation. The error is caused by CAD designers using the zero-distance constraint instead of a coincident constraint due to old habits and/or possibly old CAD assemblies being used as the basis for the dataset generation. This limitation is present in all the developed models similarly, which allows the models to be compared and validated correctly regardless of the limitation.
2. The limited data quantity and moderately difficult manual data generation to classes with lower data quantity. While the models performed adequately regardless of this limitation, it is important to observe, that the model performance is predictable to improve if trained on a larger dataset. This is validated by the results of this thesis as well.

6.2 Future Research

The selection between the utilized graph-based approach and the voxel-based approaches was heavily influenced by the large number of parameters in the voxel presentation, the larger dataset requirement and the problems in enforcing the selected elements in the dataset for the classification purpose. However, based on the CAD model related classification results achieved by Zhang et al. [3] and Hegde and Zadeh [8], voxels have immense potential as a CNN input. Even with the disadvantages, the potential of voxels in CAD assembly constraint classification requires further research. This thesis provides a point of comparison in the presented graph-based approach and thus enables further studies to compare their results to the results of this thesis.

Due to very limited amount of angle constraints and perpendicular constraints in the original dataset, it was determined that retaining these constraints in the dataset would harm the model performance. Having more samples of these constraints in the training data would allow the classifier to learn to classify these constraints as well, marginally increasing the applicability. Having a larger dataset would also open the possibility to explore the effect of the adjacency graph nearest neighbor count further. As discussed in section 5.3, the models with lower nearest neighbor counts did not always outperform the models with higher nearest neighbor counts.

The scope of the thesis did not allow an in-depth investigation into the usability improvement of the initial software implementation over the old system. Such an investigation could be conducted for example through user interviews and user experience surveys. Conducting a study that compares the original assembly constraint method to the improved AI based UI would produce information of the system usability and especially the experiences of new CAD users with no bias towards the old system.

REFERENCES

[1] Z. Han, R. Mo, H. Yang, and L. Hao, "Cad assembly model retrieval based on multi-source semantics information and weighted bipartite graph", *Computers in Industry*, vol. 96, pp. 54–65, 2018, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2018.01.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361517303780.

[2] C. Krahe, A. Bräunche, A. Jacob, N. Stricker, and G. Lanza, "Deep learning for automated product design", *Procedia CIRP*, vol. 91, pp. 3–8, 2020, Enhancing design through the 4th Industrial Revolution Thinking, ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2020.01.135. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827120307769.

[3] Z. Zhang, P. Jaiswal, and R. Rai, "Featurenet: Machining feature recognition based on 3d convolution neural network", *Computer-Aided Design*, vol. 101, pp. 12–22, 2018, ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2018.03.006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448518301349.

[4] J. Dekhtiar, A. Durupt, M. Bricogne, B. Eynard, H. Rowson, and D. Kiritsis, "Deep learning for big data applications in cad and plm – research review, opportunities and case study", *Computers in Industry*, vol. 100, pp. 227–243, 2018, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2018.04.005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0166361517305560.

[5] J. Zhang, Z. Xu, Y. Li, S. Jiang, and N. Wei, "Generic face adjacency graph for automatic common design structure discovery in assembly models", *Computer-Aided Design*, vol. 45, no. 8, pp. 1138–1151, 2013, ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2013.04.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448513000535.

[6] Y. Shi, Y. Zhang, and R. Harik, "Manufacturing feature recognition with a 2d convolutional neural network", *CIRP Journal of Manufacturing Science and Technology*, vol. 30, pp. 36–57, 2020, ISSN: 1755-5817. DOI: https://doi.org/10.1016/j.cirpj.2020.04.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1755581720300298.

[7] Y. T. Hao and Y. M. Chi, "Ann-based feature recognition to integrate cad and cam", English, *Applied Mechanics and Materials*, vol. 55-57, p. 1269, May 2011. [Online]. Available: https://www-proquest-com.libproxy.tuni.fi/scholarly-

journals/ann-based-feature-recognition-integrate-cad-cam/docview/1443592186/se-2.

[8] V. Hegde and R. Zadeh, *Fusionnet: 3d object classification using multiple data representations*, 2016. arXiv: 1607.05695 [cs.CV].

[9] B. Manda, P. Bhaskare, and R. Muthuganapathy, "A convolutional neural network approach to the classification of engineering models", *IEEE Access*, vol. 9, pp. 22 711–22 723, 2021. DOI: 10.1109/ACCESS.2021.3055826.

[10] L. P. Muraleedharan, S. S. Kannan, and R. Muthuganapathy, "Autoencoder-based part clustering for part-in-whole retrieval of cad models", *Computers & Graphics*, vol. 81, pp. 41–51, 2019, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2019.03.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849319300391.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://doi.org/10.1038/nature14539.

[12] J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural Networks*, vol. 61, pp. 85–117, 2015, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135.

[13] P. Suresh Kumar, H. Behera, A. K. K, J. Nayak, and B. Naik, "Advancement from neural networks to deep learning in software effort estimation: Perspective of two decades", *Computer Science Review*, vol. 38, p. 100 288, 2020, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100288. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013720303889.

[14] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. S. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures", *Electronics*, vol. 8, no. 3, 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8030292. [Online]. Available: https://www.mdpi.com/2079-9292/8/3/292.

[15] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples", *Communications in Computational Physics*, vol. 28, no. 5, pp. 1671–1706, Jun. 2020, ISSN: 1991-7120. DOI: 10.4208/cicp.oa-2020-0165. [Online]. Available: http://dx.doi.org/10.4208/cicp.OA-2020-0165.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://dl.acm.org/doi/10.1145/3065386.

[17] K. Ghiasi-Shirazi, "Competitive cross-entropy loss: A study on training single-layer neural networks for solving nonlinearly separable classification problems", *Neural Processing Letters*, vol. 50, no. 2, pp. 1115–1122, Oct. 2019, ISSN: 1573-773X.

DOI: 10.1007/s11063-018-9906-5. [Online]. Available: https://doi.org/10.1007/s11063-018-9906-5.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[20] T. Ma, H. Wang, L. Zhang, Y. Tian, and N. Al-Nabhan, "Graph classification based on structural features of significant nodes and spatial convolutional neural networks", *Neurocomputing*, vol. 423, pp. 639–650, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.10.060. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220316374.

[21] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs", *CoRR*, vol. abs/1605.05273, 2016. arXiv: 1605.05273. [Online]. Available: http://arxiv.org/abs/1605.05273.

[22] L. Ma, Z. Huang, and Y. Wang, "Automatic discovery of common design structures in cad models", *Computers & Graphics*, vol. 34, no. 5, pp. 545–555, 2010, CAD/GRAPHICS 2009 Extended papers from the 2009 Sketch-Based Interfaces and Modeling Conference Vision, Modeling & Visualization, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2010.06.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849310000853.

[23] S. Finger, T. Tomiyama, and M. Mäntylä, Eds., *Knowledge Intensive Computer Aided Design*. Springer US, 2000. DOI: 10.1007/978-0-387-35582-5. [Online]. Available: https://doi.org/10.1007%5C%2F978-0-387-35582-5.

[24] Siemens. (2021). "Siemens NX home page", [Online]. Available: https://www.solidworks.com (visited on 04/26/2021).

[25] D. S. S. Corporation. (2021). "Solidworks home page", [Online]. Available: https://www.solidworks.com (visited on 04/26/2021).

[26] K. Lupinetti, J.-P. Pernot, M. Monti, and F. Giannini, "Content-based cad assembly model retrieval: Survey and future challenges", *Computer-Aided Design*, vol. 113, pp. 62–81, 2019, ISSN: 0010-4485. DOI: https://doi.org/10.1016/j.cad.2019.03.005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448518305451.

[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[28] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, *3d shapenets: A deep representation for volumetric shapes*, 2015. arXiv: 1406.5670 [cs.CV].

[29] B. R. Babic, N. Nesic, and Z. Miljkovic, "Automatic feature recognition using artificial neural networks to integrate design and manufacturing: Review of automatic feature recognition systems", English, *Artificial Intelligence for Engineering Design,*

Analysis and Manufacturing : AI EDAM, vol. 25, no. 3, pp. 289–304, Aug. 2011. [Online]. Available: https://www.researchgate.net/publication/220306695_Automatic_feature_recognition_using_artificial_neural_networks_to_integrate_design_and_manufacturing_Review_of_automatic_feature_recognition_systems.

[30] Y. Zhang, S. Garcia, W. Xu, T. Shao, and Y. Yang, "Efficient voxelization using projected optimal scanline", *Graphical Models*, vol. 100, pp. 61–70, 2018, ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2017.06.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S152407031730053X.

[31] A. Neb, I. Briki, and R. Schoenhof, "Development of a neural network to recognize standards and features from 3d cad models", *Procedia CIRP*, vol. 93, pp. 1429–1434, 2020, 53rd CIRP Conference on Manufacturing Systems 2020, ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2020.03.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827120305552.

[32] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, *Shapenet: An information-rich 3d model repository*, 2015. arXiv: 1512.03012 [cs.GR].

[33] I. Valova, C. Harris, T. Mai, and N. Gueorguieva, "Optimization of convolutional neural networks for imbalanced set classification", *Procedia Computer Science*, vol. 176, pp. 660–669, 2020, Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.09.038. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920319335.

[34] P. Branco, L. Torgo, and R. Ribeiro, *A survey of predictive modelling under imbalanced distributions*, 2015. arXiv: 1505.01658 [cs.LG].

[35] V. Gilsing, B. Nooteboom, W. Vanhaverbeke, G. Duysters, and A. van den Oord, "Network embeddedness and the exploration of novel technologies: Technological distance, betweenness centrality and density", *Research Policy*, vol. 37, no. 10, pp. 1717–1731, 2008, Special Section Knowledge Dynamics out of Balance: Knowledge Biased, Skewed and Unmatched, ISSN: 0048-7333. DOI: https://doi.org/10.1016/j.respol.2008.08.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004873330800190X.

[36] M. Kurant, A. Markopoulou, and P. Thiran, "On the bias of bfs (breadth first search)", in *2010 22nd International Teletraffic Congress (ITC 22)*, 2010, pp. 1–8. DOI: 10.1109/ITC.2010.5608727.

[37] T. Zhao, Y. Liu, L. Neves, O. Woodford, M. Jiang, and N. Shah, *Data augmentation for graph neural networks*, 2020. arXiv: 2006.06830 [cs.LG].

[38] TensorFlow Core. (2021). “Tensorflow sequential model”, [Online]. Available: https://www.tensorflow.org/guide/keras/sequential_model (visited on 03/08/2021).

[39] P. D. Team. (2021). “PyInstaller home page”, [Online]. Available: http://www.pyinstaller.org (visited on 04/30/2021).

[40] gRPC Authors. (2020). “gRPC documentation”, [Online]. Available: https://grpc.io/docs/ (visited on 04/21/2021).

[41] I. Kandel and M. Castelli, “The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset”, *ICT Express*, vol. 6, no. 4, pp. 312–315, 2020, ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.04.010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519303455.

[42] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database”, in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[43] Papers With Code. (2021). “Image classification on imagenet”, [Online]. Available: https://paperswithcode.com/sota/image-classification-on-imagenet (visited on 03/04/2021).

[44] D. J. V. Lopes, G. W. Burgreen, G. dos Santos Bobadilha, and H. M. Barnes, “Automated means to classify lab-scale termite damage”, *Computers and Electronics in Agriculture*, vol. 168, p. 105 105, 2020, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2019.105105. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919313687.

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

[46] “Iso/iec/ieee international standard - systems and software engineering – vocabulary”, *ISO/IEC/IEEE 24765:2010(E)*, pp. 1–418, 2010. DOI: 10.1109/IEEESTD.2010.5733835.

[47] “Ergonomics of human-system interaction — human-centred design for interactive systems”, en, International Organization for Standardization, Standard ISO 9241-210:2019, Jun. 2019. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en.

[48] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks”, *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009, ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2009.03.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457309000259.