Unsupervised and hybrid vectorization techniques for 3D reconstruction of engineering drawings

Salwan Alwan

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Signal, Image, Vision*

Par
Salwan ALWAN

Unsupervised and Hybrid Vectorization Techniques for 3D reconstruction of Engineering Drawings

Thèse présentée et soutenue à Brest, le 28 Juin 2021
Unité de recherche : Lab-STICC, UMR CNRS 6285
Thèse N° : 2021IMTA0254

Rapporteurs avant soutenance :

Jean-Paul HATON	Professeur Emerite à l'Université de Lorraine
Titus ZAHARIA	Professeur à l'Institut Polytechnique de Paris, Télécom SudParis

Composition du Jury :

Président :	Philippe CARRE	Professeur à Laboratoire XLIM, UMR CNRS 7252
Examinateurs :	Xavier HILAIRE	Professeur associé à ESIEE Paris
	Gérard LE MEUR	Responsable Ingénierie Produit à Thalès DMS
	Jean-Paul HATON	Professeur Emerite à l'Université de Lorraine
	Titus ZAHARIA	Professeur à l'Institut Polytechnique de Paris, Télécom SudParis
Dir. de thèse :	Jean-Marc LE CAILLEC	Professeur à IMT Atlantique

Invité(s)

Anne BRONNEC	Architecte de solutions de formation à Thalès DMS

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Prof. Jean-marc LE CAILLEC for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Philippe CARRE, Prof. Jean-Paul HATON, Prof. Titus ZAHARIA, Prof. Xavier HILAIRE and Mr. Gerard LE MEUR, not only for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

A special thanks to Mr. Gerard LE MEUR and Ms. Anne BRONNEC from THALES DMS who helped me a lot during my thesis and I am grateful to Thales who sponsored this research.

Last but not the least I would like to thank my parents, brothers, sister, and my fiance for supporting me spiritually throughout writing this thesis and my life in general.

Contents

List of Figures

List of Tables

Long Résumé

Le dessin est un outil utilisé pour communiquer depuis longtemps. Il s'agit d'un outil permettant d'expliquer une idée à l'aide de dessins tels que les anciens dessins chinois et égyptiens. Cependant, actuellement, le dessin est utilisé pour expliquer comment différents objets tels que les machines, les bâtiments et l'électronique sont construits. Cet outil est connu comme le langage commun utilisé par les ingénieurs.

Avant l'apparition des logiciels de dessin tels qu'AutoCAD, les ingénieurs utilisaient de grands papiers et des planches à dessin pour produire des dessins techniques (voir figure 0.0.1). En outre, différents types de matériel étaient utilisés pour réaliser les dessins, tels que des crayons de différentes qualités, une équerre, des gommes, des règles et un compas. Outre l'énorme avantage de trouver une façon commune d'expliquer comment les choses sont faites, ce type de dessins présente de nombreux inconvénients, comme le temps considérable nécessaire pour réaliser le dessin, la difficulté de modifier un dessin après l'avoir mis sur papier, et la difficulté de conserver ce type de données.

Figure 0.0.1: Tamron's drafting department in the late 1970s [1]

Avec l'augmentation des technologies et de la fabrication des ordinateurs, Ivan Sutherlend crée en 1963 un logiciel appelé Sketchpad pour le dessin, comme le montre la figure

0.0.2. Ce logiciel était un logiciel simple qui permettait aux utilisateurs de créer des tracés x y. Sketchpad n'est plus utilisé de nos jours, mais il a marqué le début du développement des logiciels de conception assistée par ordinateur actuels.

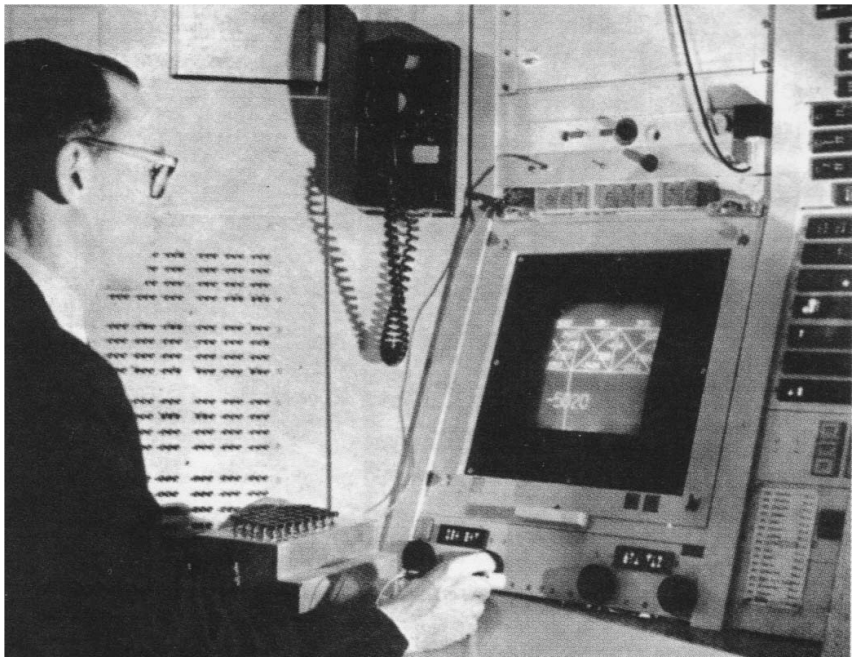


Figure 0.0.2: Sketchpad, Ivan Shuterland 1963 [2]

Dans les années 1960, de nombreuses entreprises telles que Ford, MIT, GM et Boeing ont mené des recherches financières et intellectuelles sur les logiciels de CAO. Ces recherches visaient à simplifier les conceptions automobiles et aérospatiales. Dans la moitié suivante du siècle, et grâce à la loi de Moor et à la croissance de l'électronique, les capacités de la CAO s'étendent régulièrement, et la fondation Autodesk apparaît.

Néanmoins, avec tous les progrès réalisés dans le domaine des logiciels de conception assistée par ordinateur, l'utilisation des logiciels de CAO se généralise à la fin des années 1980 et au début des années 1990. Avec l'augmentation et l'innovation avancée des logiciels de CAO, les technologies de réalité augmentée et de réalité virtuelle sont utilisées dans différents domaines tels que le divertissement, la fabrication et la formation.

La forte concurrence dans les secteurs des affaires et de la fabrication pose des problèmes difficiles, l'un des principaux étant de fabriquer des produits innovants en peu de temps. De plus, la plupart des grandes entreprises ont des filiales dans le monde entier, ce qui rend difficile le déplacement de certains produits et machines d'un endroit à l'autre. Tous ces problèmes et d'autres peuvent être résolus en utilisant les technologies de RV et de RA pour simuler et améliorer ces processus de fabrication et ces phases de test.

Les améliorations rapides des logiciels de CAO et des technologies VR/AR créent un nouveau problème, à savoir la conversion des dessins enregistrés sous forme d'images en données vectorielles pour pouvoir les modifier et les convertir en modèles 3D à utiliser dans les applications VR/AR. Il y a des milliers de dessins sur papier qui n'ont pas leurs fichiers CAO, et beaucoup de dessins anciens sont principalement enregistrés sous forme d'images. Ces dessins doivent être convertis en données vectorielles 2D pour pouvoir les modifier et les mettre à jour facilement. De plus, les données vectorielles peuvent être converties en modèles 3D et utilisées dans des applications AR/VR.

La conversion des vues orthographiques en modèles 3D peut être réalisée en utilisant principalement l'une des méthodes suivantes :

Représentation des limites (Brep)

La plupart des travaux précédents visant à reconstruire des solides en 3D à partir de vues orthographiques sont basés sur l'approche B-rep. L'approche B-rep a été proposée pour la première fois par *Idesawa* [3] et formalisée par *Markowsky, et Wesley* [4, 5]. Cette approche repose sur quatre étapes principales :

1. Convertir les jonctions 2D en sommets 3D.
2. Générer des arêtes 3D à partir de sommets 3D.
3. Construire des faces 3D à partir d'arêtes 3D.
4. Construire des objets 3D à partir de faces 3D.

Différentes méthodes prolongent les travaux de Markowsy et Wesley et augmentent l'efficacité, la précision et la robustesse de l'approche B-rep, comme *Sakurai et al.* [6], *Yan et al.* [7], *Shin et al.* [8], *Kuo et al.* [9].

Géométrie solide constructive (GSC)

L'approche CSG [10, 11, 12, 13] est une autre stratégie de modélisation utilisée pour reconstruire un solide 3D à partir de vues orthographiques. Cette approche suppose que le solide est construit par un ensemble d'objets solides primitifs et d'opérations booléennes. Cette approche commence par des primitives simples telles que des blocs, des pyramides et des sphères, puis un nouvel objet est construit en fusionnant deux primitives à l'aide d'opérations booléennes telles que l'intersection, l'union et la soustraction. L'objet construit peut être combiné avec d'autres primitives, et le processus se poursuit jusqu'à l'obtention d'un modèle complet (voir figure 0.0.3).

Néanmoins, les deux approches ont besoin de données vectorielles en entrée. Ainsi, la nécessité d'un algorithme de vectorisation pour convertir les dessins sur papier en données

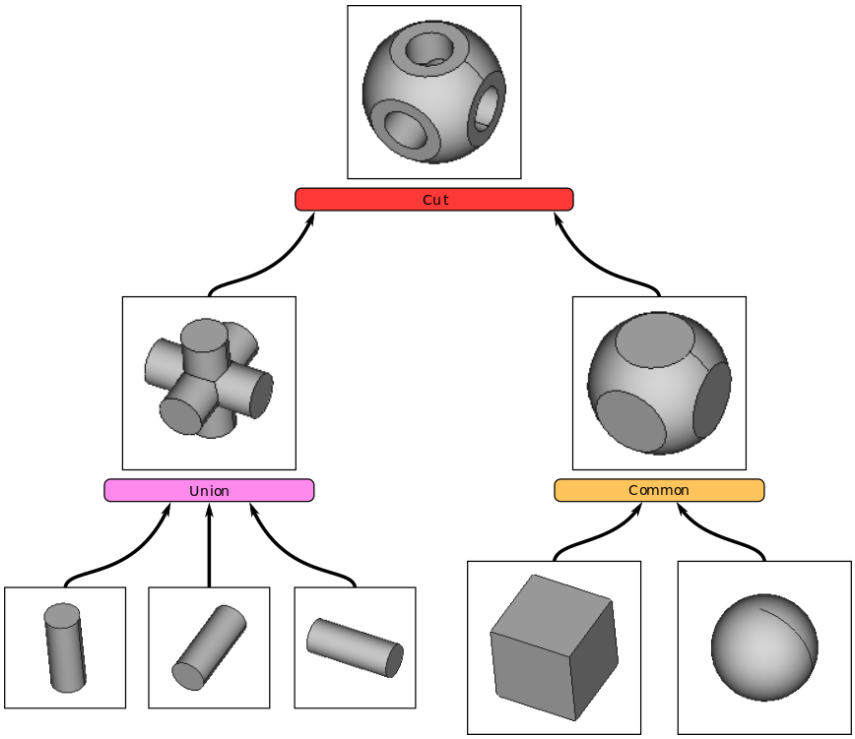


Figure 0.0.3: Exemple de CSG [14]

vectorielles est une étape clé à résoudre efficacement. Le concept de vectorisation peut être utilisé pour vectoriser différents types de dessins tels que des dessins à la main, des dessins techniques, des logos et d'autres images. Chaque type de dessin est vectorisé avec différents types de représentation. Par exemple, le dessin de la main et les logos peuvent être vectorisés et représentés à l'aide de courbes de Bézier. Les courbes peuvent également représenter les dessins techniques ; cependant, la manière la plus simple de reconstruire des modèles 3D est de représenter les dessins techniques avec des primitives de base telles que les lignes, les cercles et les arcs, qui peuvent être utilisées dans les approches B-rep et CSG. Plusieurs reconstructions de solides 3D basées sur différentes approches et avec différentes spécifications ont déjà été proposées et montrent des résultats remarquables ; pour plus de détails, consultez [15]. Par conséquent, ce travail vise à se concentrer sur la vectorisation des dessins techniques. Dans le paragraphe suivant, nous décrivons la structure du manuscrit et les résumé des chapitres.

Structure du manuscrit et résumé des chapitres

Ce manuscrit est divisé en quatre chapitres, et il est organisé de la manière suivante :

Chapter 1 - Background

Dans le premier chapitre, nous donnons un aperçu des méthodes utilisées dans cette thèse. La première section de ce chapitre présente la carte de distance utilisée pour squelettiser l'image. La deuxième section présente brièvement la théorie de l'apprentissage profond. De plus, nous présentons le concept CNN et les principales couches utilisées pour construire un réseau dans la deuxième section.

Chapter 2 - Preprocessing of Engineering Drawings

Dans le deuxième chapitre, nous avons proposé un cadre complet pour préparer les dessins techniques au processus de vectorisation. Le cadre reçoit un modèle de dessin d'ingénierie matriciel et le traite pour supprimer les bordures et regrouper les différentes vues. (voir la figure 0.0.4)

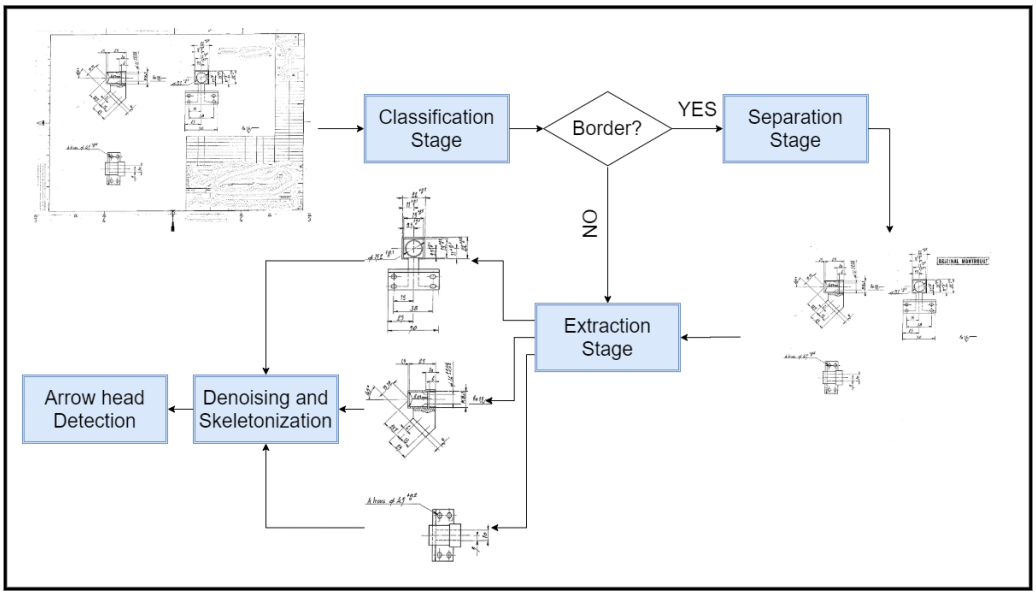


Figure 0.0.4: Preprocessing stage overview

L'algorithme commence par l'étape de classification dans laquelle nous classons les dessins techniques en "dessins avec bordure" et "dessins sans bordure". Ensuite, nous détectons le rectangle de la plus grande surface dans l'image du dessin technique en calculant les contours (courbes qui joignent des pixels connectés ayant la même intensité) [16]. Si le rectangle détecté est égal ou supérieur à 80% (sur la base d'expériences) de la surface de l'image, alors le dessin contient une bordure. Sinon, le dessin ne comporte que des vues orthogonales.

Ensuite, les dessins techniques avec des bordures sont traités pour séparer les éléments graphiques et éliminer les autres informations.Ainsi, nous détectons tous les contours dans

l'image. La deuxième plus grande zone de contour sépare les vues et la bordure, et le tableau.

Après avoir séparé les éléments graphiques des autres éléments, nous cherchons à séparer les vues. Comme mentionné précédemment, le nombre de vues peut varier d'un dessin à l'autre. Nous avons donc besoin d'une méthode non supervisée capable de séparer automatiquement les vues. Le clustering est une méthode non supervisée utilisée pour séparer les données en groupes basés sur des propriétés similaires telles que la distance entre les points ou les densités de points. Dans notre cas, nous choisissons l'algorithme k-means [17] et l'algorithme Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [18] car la distribution des données semble correspondre à l'hypothèse sous-jacente de ces algorithmes où chaque vue est séparée de l'autre par un espace blanc. Ainsi, la k-means fonctionne lorsque ses centroïdes sont placés au centre de chaque vue, et le DBSCAN fonctionne parce que les pixels de chaque vue sont très proches alors que chaque vue est séparée de la seconde par une grande distance. Après avoir comparé k-means et DBSCAN, nous choisissons le dbscan car il fonctionne mieux dans tous les cas, y compris lorsque les vues sont proches.

Ensuite, chaque vue est débruitée à l'aide du modèle U-net, qui est entraîné, testé et comparé à d'autres méthodes en utilisant les métriques PSNR et DRD. Le réseau formé atteint une précision de 99,97%. Les expériences montrent que le modèle U-net entraîné surpasse les autres algorithmes traditionnels de suppression du bruit utilisés pour le débruitage des dessins techniques (qualitativement et quantitativement).

Le réseau entraîné nettoie les images d'entrée tout en préservant les bords lisses qui sont importants pour le processus de squelettisation. La squelettisation est basée sur la distance de chanfrein 3,4 ; ce processus diminue la complexité du problème de la vectorisation à la segmentation des courbes.

Enfin, nous proposons une méthode de détection des pointes de flèches basée sur un sac de mots visuels. Cette méthode est basée sur deux éléments principaux : la détection des caractéristiques et la classification. Nous avons comparé différentes combinaisons d'algorithmes de détection de caractéristiques et d'algorithmes de classification bien connus. Nous avons constaté que la combinaison de KAZE comme détecteur de caractéristiques et de perceptron multicouche comme classificateur est plus performante que les autres combinaisons. La méthode proposée est testée avec différentes images provenant de différentes sources et avec différentes résolutions et niveaux de bruit. Les résultats montrent que la méthode proposée est précise dans la détection des pointes de flèches. Cependant, cette méthode nécessite un nouveau jeu de données pour détecter différents types de flèches.

Chapter 3 - Unsupervised Vectorization

Dans le troisième chapitre, nous avons proposé une méthode basée sur un algorithme génétique. L'entrée de cette méthode est l'image squelettisée et étiquetée du chapitre 2.(voir la figure 0.0.5)

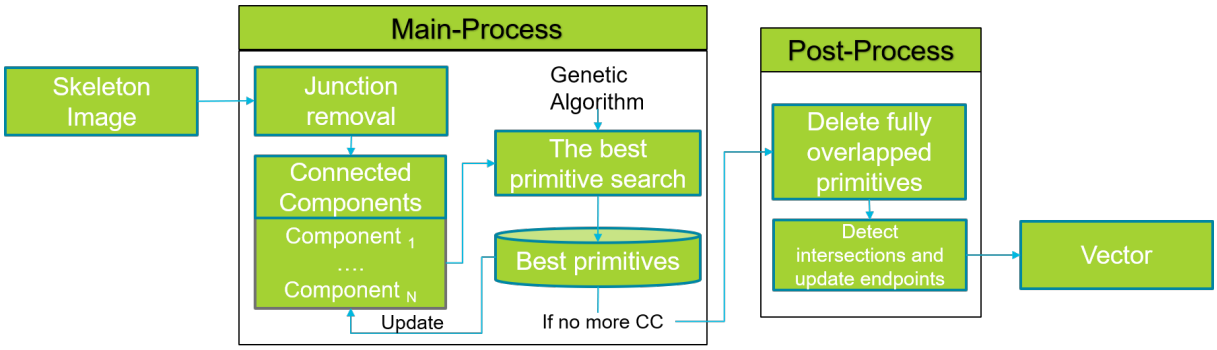


Figure 0.0.5: Unsupervised vectorization method overview

L'algorithme extrait les pixels de jonction ; nous obtenons un ensemble de branches à partir du squelette où chaque branche est un ensemble de pixels. A chaque fois, nous sélectionnons une branche et utilisons les pixels comme chromosomes pour générer différentes possibilités de primitives qui correspondent à cette branche. Les chromosomes sont de deux types différents : les chromosomes de ligne droite et les chromosomes d'arc de cercle. L'algorithme génétique détecte les primitives qui réduisent l'erreur d'ajustement et augmentent la longueur (nombre de pixels appartenant à la primitive). L'algorithme continue à détecter les primitives jusqu'à ce que la plus grande branche devienne plus petite qu'un certain seuil de pixels.

La méthode proposée ne peut détecter que les lignes droites, les cercles et les arcs ; cependant, la méthode peut être étendue pour détecter d'autres types de primitives comme l'ellipse en ajoutant des chromosomes d'ellipse et en ajoutant les paramètres de l'ellipse. La méthode proposée estime la largeur de chaque branche et met à jour les seuils des paramètres des primitives. La méthode proposée est comparée à différents algorithmes de vectorisation de dessins techniques et les surpasse en termes d'indice de récupération des vecteurs et de résultats visuels.

De plus, nous avons comparé l'algorithme proposé avec d'autres types de vectorisation, comme la vectorisation de dessins à la main en utilisant l'indice d'interaction sur l'union et le nombre de primitives nécessaires pour vectoriser une image matricielle. L'intersection sur les métriques d'union montre la précision du modèle de vectorisation, et le nombre de primitives nécessaires pour représenter une image matricielle montre la complexité de la sortie vectorielle. Les résultats montrent que notre méthode est compétitive avec certaines méthodes et surpasse d'autres méthodes. Outre la complexité de calcul élevée de

la méthode proposée, le principal inconvénient de notre méthode est la forte possibilité de détecter un petit coin comme un arc. De plus, la méthode de post-traitement proposée ne peut traiter que l'intersection de deux primitives, alors que dans certains cas, nous avons plus de deux primitives qui se croisent.

Chapter 4 - Hybrid Vectorization

Dans le quatrième chapitre, nous étudions la capacité à séparer les couches de primitives en utilisant différentes méthodes de segmentation par apprentissage profond. (voir la figure 0.0.6)

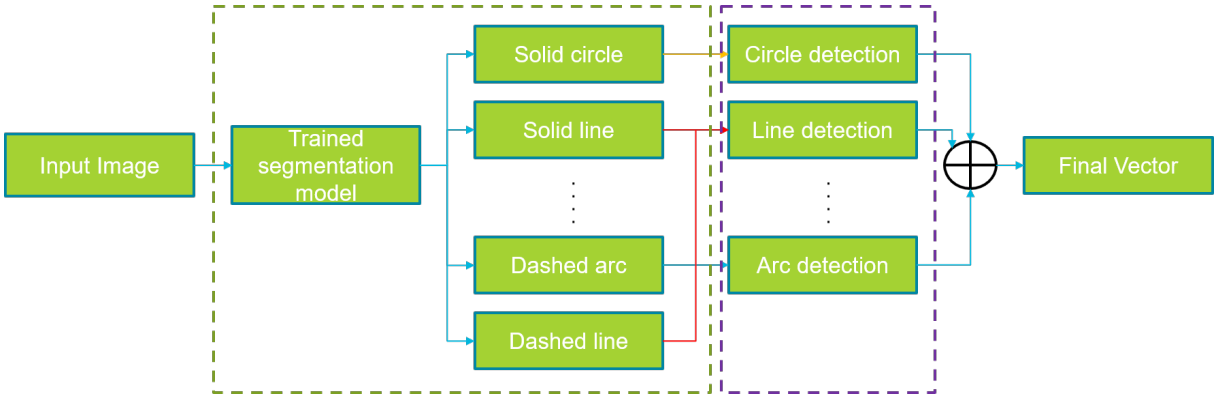


Figure 0.0.6: Hybrid vectorization method overview

Nous avons entraîné onze réseaux différents pour segmenter le dessin d'ingénierie et séparer les couches. Cinq des onze modèles sont entraînés à séparer le dessin technique d'entrée en huit couches différentes, à savoir : la couche d'arrière-plan, la couche de lignes droites, la couche d'arcs, la couche de cercles, la couche de lignes pointillées, la couche de cercles pointillés, la couche d'arcs pointillés et la couche d'intersection.

Nous évaluons ces cinq modèles à l'aide de la métrique intersection sur union, nous avons testé le modèle en utilisant 2000 images, et le meilleur modèle (Resnet U-net) atteint environ 66% d'IoU. Ce pourcentage est considéré comme faible, et nous ne pouvons pas nous y fier pour séparer les couches. Ensuite, nous avons entraîné les cinq mêmes modèles à séparer le dessin technique d'entrée en quatre couches : la couche d'arrière-plan, la couche des lignes droites, la couche des cercles/arcs et la couche des intersections. Nous utilisons la même métrique et le même nombre d'images de test, et le meilleur modèle (U-net) atteint une moyenne de 82% d'IoU. Ce pourcentage est considéré comme acceptable et capable de séparer les couches. En outre, nous avons entraîné le réseau Mask-Rcnn+ Pointrend pour évaluer la capacité de séparer les instances. Pour avoir une comparaison équitable avec les autres réseaux, nous convertissons les résultats de la segmentation des instances en résultats de la segmentation sémantique. Nous les convertissons deux fois,

d'abord pour comparer avec un réseau à huit étiquettes et ensuite pour comparer avec un réseau à quatre étiquettes.

Les résultats montrent que le meilleur réseau et le plus acceptable est le modèle U-net à quatre étiquettes. Ensuite, nous proposons une méthode basée sur ce réseau pour vectoriser les dessins. La méthode commence par séparer l'image d'entrée en une couche de cercle/arc, une couche de ligne droite et une couche d'intersection. Nous fusionnons la couche d'intersection avec les deux autres couches, puis nous détectons les lignes droites et les cercles/arcs. Cette méthode réduit le problème de la vectorisation de la segmentation des courbes à la détection des lignes et des arcs. Le principal inconvénient de cette méthode est la dimension limitée de l'image d'entrée, qui est de 512 * 512. Dans le cas d'une image de plus petite dimension, nous pouvons ajouter un remplissage blanc, et dans le cas d'une image de plus grande dimension, nous pouvons ajouter un remplissage et la décomposer en patches de segment 512*512 et les réunir. Cependant, la probabilité de fragmenter les primitives augmente lorsque nous décomposons l'entrée en patches car un patch peut passer par un cercle ou un arc.

De plus, nous testons la capacité de reconstruire des modèles 3D directement à partir d'images 2D basées sur l'apprentissage profond. Nous utilisons le modèle pix2vox, qui est l'un des algorithmes les plus performants pour cette tâche. Les résultats ne sont pas aussi bons que prévu, et ce pour plusieurs raisons. L'une des principales causes est que ce type d'algorithme est entraîné pour un seul type d'objet, comme les chaises et les avions ; or, le dessin technique ne présente pas de fortes similitudes de forme.

Introduction

Drafting is a tool used to communicate for a long time. It is the tool of explaining the idea using drawings such as the ancient Chinese and Egyptian drawings. However, currently drafting is used to explain how different objects such as machines, buildings, and electronics are constructed. This tool is known as the common language used between engineers.

Before the appearance of drafting software such as AutoCAD, engineers use large paper and drawing boards to generate engineering drawings (see figure 0.0.7). Besides, different equipment types were used to achieve the drawings, such as pencils with different grades, T square, erasers, rulers, and compass. Besides the huge advantage of finding a common way to explain how things are done, there are many disadvantages for such type of drawings, such as the considerable time needed to complete the drawing, the difficulty of modifying a drawing after committed to paper, and the difficulty of preserving this kind of data.

Figure 0.0.7: Tamron's drafting department in the late 1970s [1]

With the increase of technologies and computer manufacturing, Ivan Sutherlend cre-

ates a software called Sketchpad for drafting in 1963, as shown in figure 0.0.8. This software was a simple software that allows users to create x y plots. Sketchpad is no more used nowadays; however, it was the beginning of developing the current computer-aided design software.

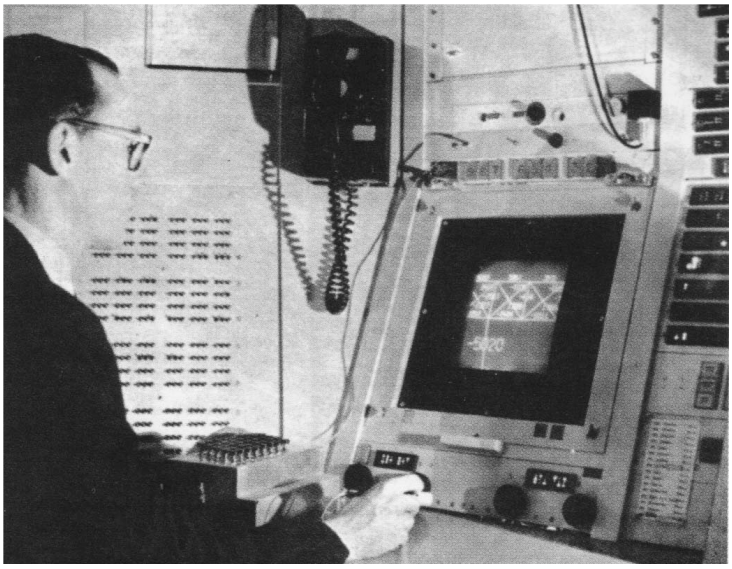


Figure 0.0.8: Sketchpad, Ivan Shuterland 1963 [2]

During the 1960s, many companies such as Ford, MIT, GM, and Boeing investigate financially and intellectually CAD software. Those investigations aimed to simplify automotive and aerospace designs. In the next half of the century, and due to Moor's law and electronics growth, CAD capabilities expanded steadily, and the Autodesk foundation appears. Nevertheless, with all the progress done in the field of computer-aided design software, CAD software became widely used in the late 1980s and early 1990s.

With the increase and advanced innovation in CAD software, Augmented Reality and Virtual Reality technologies are being used in different domains such as entertainment, manufacturing, and training.

The high competition in business and manufacturing areas produces challenging problems, one of the main problems is to produce innovative products in a short time. Also, most of the huge companies have different branches worldwide; thus, some products and machines are challenging to move from one place to another. All these problems and other problems can be solved using VR and AR technologies to simulate and improve these manufacturing processes and testing phases.

The fast improvements in CAD software and VR/AR technologies create a new problem which is converting drawings that are saved as images into vector data to be able to modify them and convert them into 3D models to be used in VR/AR applications.

There are thousands of paper drawings that do not have their CAD files, and a lot of old drawings are mostly saved as images. Those drawing should be converted into 2D Vector data to be able to modify and update them easily. Moreover, Vector data can be converted to 3D models and used in AR/VR applications.

The conversion from orthographic views into 3D models can be completed by using mainly one of the following methods:

Boundary Representation (Brep)

Most of the previous works to reconstruct 3D solid from orthographic views are based on the B-rep approach. The B-rep approach was first proposed by *Idesawa* [3] and formalized by *Markowsky, and Wesley* [4, 5]. This approach is based on four main steps:

1. Convert 2D junctions to 3D vertices.
2. Generate 3D edges from 3D vertices.
3. Build 3D faces from 3D edges.
4. Construct 3D objects from 3D faces.

Different methods extend the work of Markowsy and Wesley and increase the efficiency, precision, and robustness of the B-rep approach, such as *Sakurai et al.* [6], *Yan et al.* [7], *Shin et al.* [8], *Kuo et al.* [9].

Constructive Solid Geometry (CSG)

The CSG approach [10, 11, 12, 13] is another modeling strategy used to reconstruct 3D solid from orthographic views. This approach assumes that the solid is constructed by a set of primitive solid objects and boolean operations. This approach starts with simple primitives such as blocks, pyramids, and spheres, and then a new object is constructed by fusing two primitives using boolean operations such as intersection, union, and subtraction. The constructed object can be combined with other primitives, and the process continues until reaching a complete model (see figure 0.0.9).

Nevertheless, both approaches need vector data as input. Thus, the need for a vectorization algorithm to convert paper drawings into vector data is a key step to be efficiently solved. The vectorization concept can be used to vectorize different types of drawings such as hand drawings, engineering drawings, logos, and other images. Each type of drawing is vectorized with different types of representation. For example, the hand drawing and logos can be vectorized and represented using bezier curves. Curves can also represent engineering drawings; however, the easiest way to reconstruct 3D models is to represent engineering drawings with basic primitives such as lines, circles, and arc, which

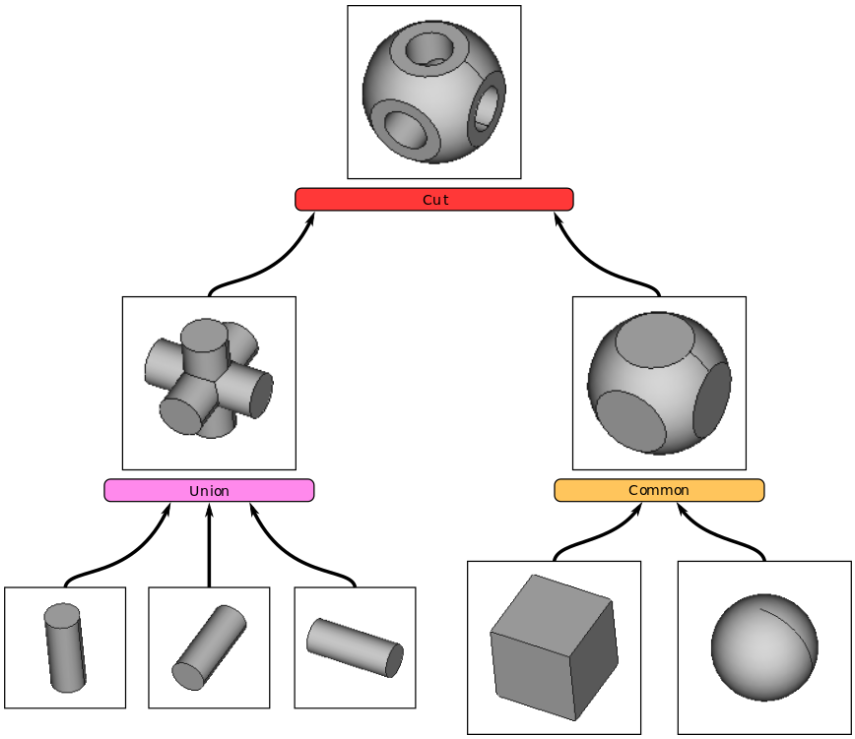


Figure 0.0.9: CSG example [14]

can be used in both B-rep and CSG approaches. Various 3D solid reconstructions based on different approaches and with different specifications are already proposed and show outstanding results; for more details, check [15]. Therefore, this work aims to focus on the vectorization of engineering drawings. In the following paragraphs, we describe the thesis contributions and the manuscript organization.

Thesis contributions

Towards these objectives, I proposed several original contributions in the framework of this PhD thesis. These contributions are summarized below.

1. In Chapter 2, I proposed a framework to pre-process technical drawing. This framework aims to prepare technical drawing data to be vectorized. The proposed algorithm starts by separating the template from graphical entities when available. The next stage aims to separate orthogonal views. Next, the proposed algorithm cleans each view separately and detects arrowheads.

2. Chapter 3 is our main contribution, I propose a new vectorization algorithm that is robust to noise and rotation. The algorithm is based on a genetic algorithm and refers to the line width and primitive parameters to predict the best primitive that fit the drawing based on a set of points.

3. In Chapter 4, I propose a vectorization method based on deep learning methods. In

this chapter, we study the behavior of different deep learning algorithms in terms of separating raster engineering drawing layers. We select the best model that can separate layers and then use the algorithm proposed in Chapter 3 to vectorize each layer separately. Moreover, we study the ability of 3D deep learning algorithm to convert 2D raster views into 3D model without passing through the intermediate vectorization stage.

Manuscript Organization

This manuscript is divided into four chapters, and it is organized in the following manner:

Chapter 1 gives an introduction to some methods that are used in the later chapter. The first part focuses on the thinning algorithm, whereas the second part focuses on the deep learning concepts.

Chapter 2 presents a full framework to prepare paper drawings to be vectorized. The chapter consists of three main sections: The first section receives an input image and separate views. The second section denoises each view using a deep learning network. The third section detects arrowheads using Bag of visual words. All these methods are optional; thus, we can use all of them or some of them when needed.

Chapter 3 introduces a new vectorization algorithm based on a genetic algorithm. The chapter starts by introducing the problem and then describes the proposed method used to vectorize engineering drawings. The proposed method is compared with different methods that are recently proposed using different metrics, which show the stability, robustness, and precision of our method.

Chapter 4 presents another vectorization method based on deep learning methods. This chapter aims to study the ability of different deep learning models to separate drawing layers such as straight lines and curves. Moreover, we study the ability to reconstruct 3D models directly from raster orthographic views using the pix2vox model.

The manuscript ends by summarizing the outcome of this thesis and discusses the future improvements, suggestions and works to be done.

Chapter 1

Background

Contents

The vectorization of documents is a fundamental problem in computer vision. This problem has been solved using different approaches during the last decade. In our work, we use the technique of thinning and then tracking. Thus, in this chapter, we explain more about the thinning method used. Besides, we use deep learning networks for segmenting images. Therefore, we also present the theory behind deep learning.

1.1 Thinning Approaches

Thinning in computer vision is the process of sampling the binary image to obtain a one-pixel wide representation. Thinning is also known as skeletonization or medial axis transformation, is widely used in vectorization algorithms before the process of shape detection.

The thinning procedure can be done mainly using two different approaches:

- Morphological approach
- Distance transform approach

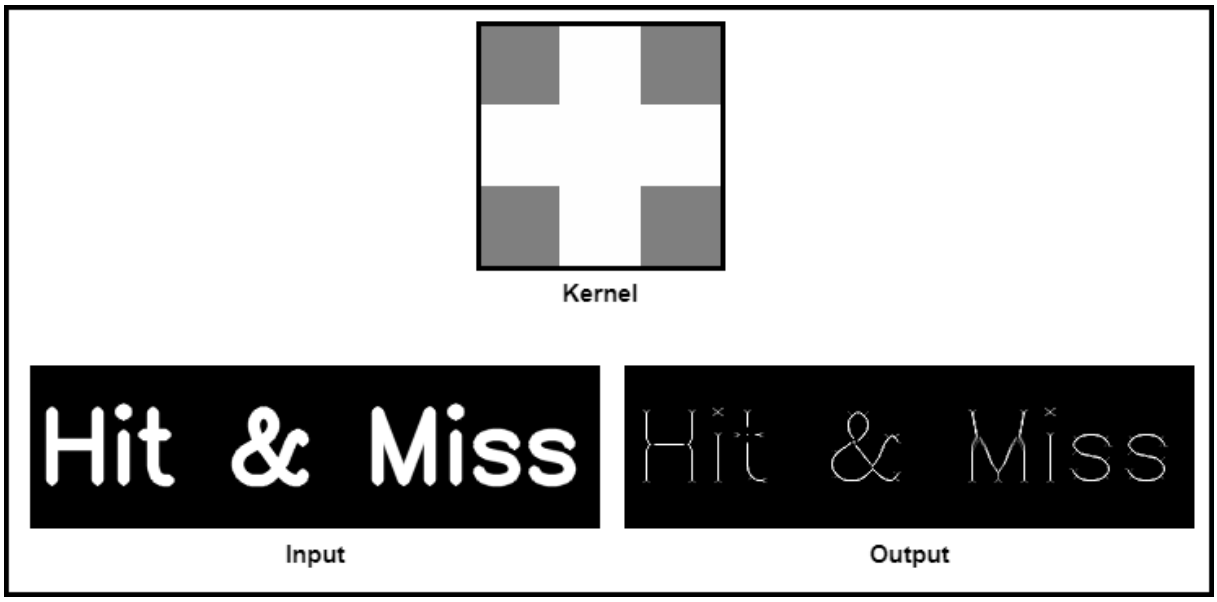


Figure 1.1.1: Hit-and-miss thinning example

1.1.1 Morphological Approach

The morphological approach is based hit-and-miss algorithm, which uses a structuring element of 0s and 1s to detect a specific pattern in a binary image. The hit-and-miss algorithm is also used for performing thinning using iterative steps. In each iteration, some different structuring elements are used to identify the edge pixels to be removed. The thinning process can be formulated as follow:

$$I'_k = f(I_k)$$
$$I_{k+1} = I_k \cap \bar{I'_k}$$
$$k = k + 1$$

where I_k,I'_k and $f(.)$ are respectively the image at k iteration, the image after the hit-and-miss operation at k iteration, and the structuring element used for the hit-and-miss process. The structuring elements are rotated to check whether other edge pixels can be removed. The algorithm continues looping until all the edge pixels are removed. The final shape is the skeleton of the input image, as seen in Figure 1.1.1.

1.1.2 Distance Transform Approach

The input for a distance transform algorithm is a binary image, and the output is a distance map. In fact, this approach aims to extract the frontiers between two blocks 0 and 1 by estimating the distance between these blocks. Distance transform algorithms are processed using two main stages:

- Forward pass: for every pixel in the foreground, the distance from the top and left foreground boundaries is determined.

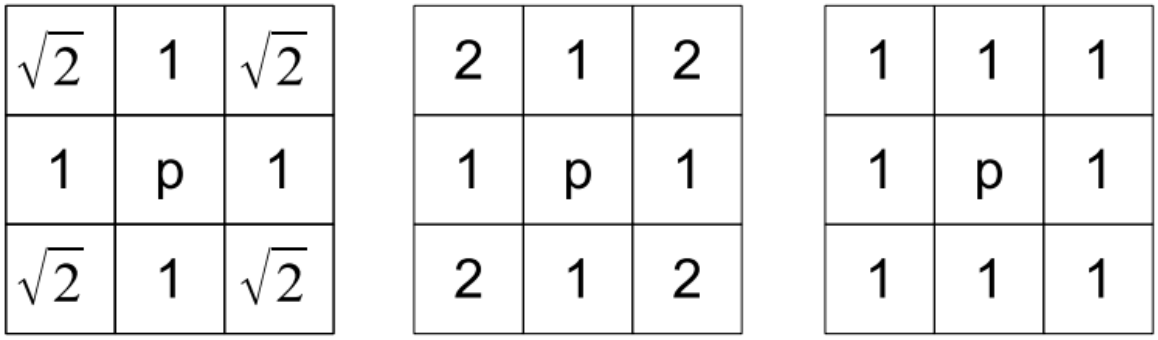


Figure 1.1.2: Distance transform filter: the left one compute the euclidean distance, the middle one compute the City block distance, and the right one compute the chess-board distance

- Backward pass: for every pixel in the foreground, the distance from every pixel in the foreground to the bottom and right boundaries.

Different distance metrics can be used to compute the distance map. Assume that we have two point $p = (x_p, y_p)$ and $q = (x_q, y_q)$, thus the distance between the two points is computed as follow (see figure 1.1.2):

- Euclidean distance $d_E(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$
- City block $d_{city}(p,q) = |x_p - x_q| + |y_p - y_q|$
- Chess-board $d_{chess}(p,q) = max(|x_p - x_q|, |y_p - y_q|)$

One of the well known and usable metrics is 3,4 chamfer distance which has many advantage such as the high speed and simplicity. Moreover, unlike euclidean distance, 3,4 chamfer distance is a local distance which permits to deduce a distance from the distances of close neighbors (refer to [19]). Thus in chapter 3, we used the 3,4 chamfer distance metric. As seen in figure 1.1.3, the algorithm of chamfer distance map receive a binary input image. The algorithm replace all 1s by infinity. Next the forward pass start, if $p > 0$, the value of p is changed as seen in algorithm 1 (line 3.). After scanning the image with the forward filter, the backward filter start to scan, if $p > 0$, the value of p is changed as seen in algorithm 1 (line 4.). The result of this process is a distance map as seen in figure 1.1.3

After we get the distance map, the skeleton is extracted using center of maximal disk detection.

1.2 Supervised Learning

Supervised learning is a well-known method in computer vision. The supervised learning algorithms learn a function f that can maps the input data X into the output data Y.

Algorithm 1: 3,4 Chamfer Distance

p is the current pixel;
N_1 to N_4 are the neighbors pixels;
$w_1 = w_3 = 4$ and $w_2 = w_4 = 3$ - see figure;

1. Replace the background pixels value by zero;
2. Replace the foreground pixels value by infinity;
3. Forward pass through all the the image from pixel $(0, 0)$ to $(max(x), max(y))$; if $p > 0$, $p = min(N_i + w_i)$ for $i = 1, 2, 3, 4$;
4. Backward pass through all the the image from pixel $(max(x), max(y))$ to $(0, 0)$; if $p > 0$, $p = min(p, min(N_i + w_i))$ for $i = 1, 2, 3, 4$;

Result: Distance map;

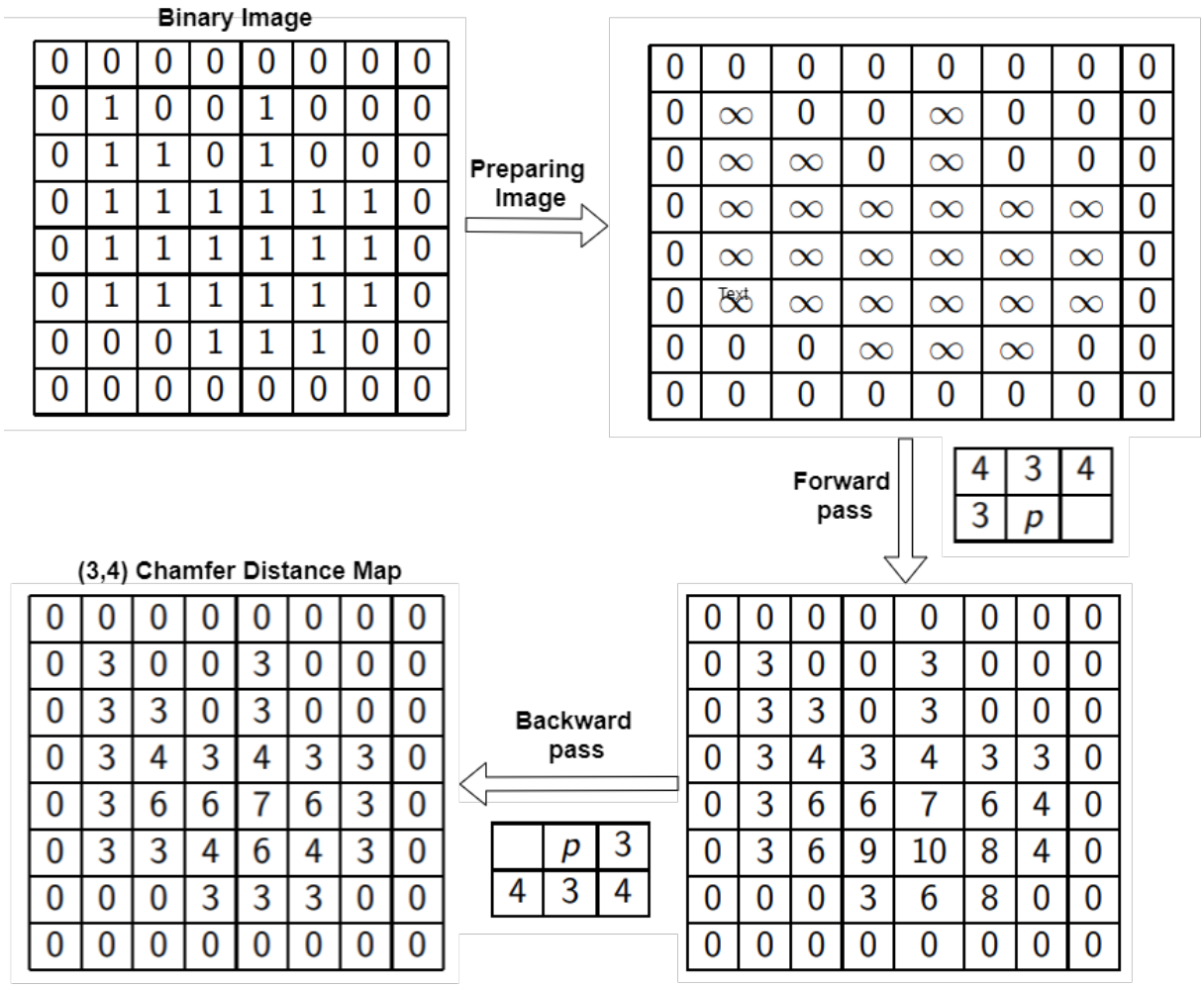


Figure 1.1.3: Example of (3,4) chamfer distance map

In the segmentation problem, the input data are images, and the output data are labeled images. Thus, the algorithm learns a function f that can map an unlabeled image to a labeled image. In the following sections, we explain how the network is trained to find the function f. Next, we introduce the convolutional neural network, a well-known concept used to train image data.

1.2.1 Learning algorithm

The learning stage in deep learning is an algorithm used to find the parameters that predict the best approximation to the input label. To reach this goal, we define a loss function J which compute the distance between the ground truth and the predicted data on the overall training set. J is minimized using two major steps:

- Forward propagation
- Backpropagation

The learning algorithm starts by initialization the model parameters, which are usually initialized using Random initialization. Random initialization is like injecting random noise into the model. Assume the number of epochs is equal to N. At each epoch, the algorithm starts by predicting the output $\hat{y}_i$ of the input x_i for all the training sets. The algorithm compare the ground truth output y_i with $\hat{y}_i$ using the loss function J where $J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i)$ where L computes the distance between the ground truth and the predicted data on a single sample. Based on J, the algorithm updates the parameters using the backpropagation algorithm.

To understand how the training stage learns, we explain the theory behind forward and backpropagation in the following sections [20].

1.2.1.1 Forward Propagation

We start by giving a simple example as shown in figure 1.2.4 to understand some concept which will be used during the explanation. In Figure 1.2.4, the red neurons are the input, and the green one is the output. The output neuron consists of two main blocks, z and a. Those two blocks can be defined as follow:

$$z = \sum_i w_i * x_i + b$$

$$a = \psi(z)$$

. w_i, b, and ψ are respectively the weights, the bias, and the activation function.

Assume we have the neural network shown in Figure 1.2.5 where i is the number of layers and j is the number of nodes in a layer i we denote:

$$W^{[i]} = [w_1^{[i]}, w_2^{[i]}, ..., w_{n_i}^{[i]}]$$

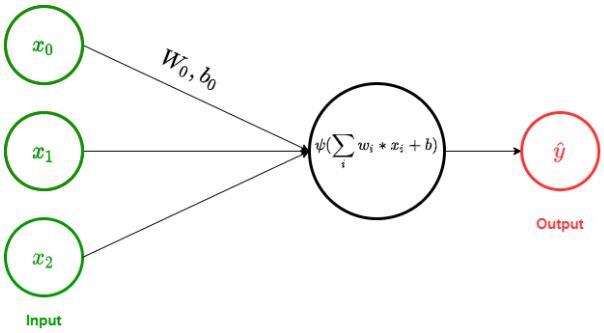


Figure 1.2.4: Simple example of neural network

$$b^{[i]} =^T [b_1^{[i]}, b_2^{[i]}, ..., b_{n_i}^{[i]}]$$
$$Z^{[i]} =^T [z_1^{[i]}, z_2^{[i]}, ..., z_{n_i}^{[i]}]$$
$$A^{[i]} =^T [a_1^{[i]}, a_2^{[i]}, ..., a_{n_i}^{[i]}]$$
$$A^{[i]} = \psi^{[i]}(Z^{[i]}) =^T [\psi^{[i]}(z_1^{[i]}), \psi^{[i]}(z_2^{[i]}), ..., \psi^{[i]}(z_{n_i}^{[i]})]$$
$$\implies A^{[i]} = \psi^{[i]}(W^{[i]^T} A^{[i-1]} + b^{[i]})$$

Thus during the training, assume $a^{[0]} = x^{(j)}$, for each layer i we compute:

$$z^{[i][j]} = W^{[i]^T} a^{[i-1][j]} + b^{[i]}$$

$$a^{[i][j]} = \psi^{[i]}(z^{[i][j]})$$
$$\implies \hat{y}^{[j]} = \psi^{[M]}(a^{[M]})$$

where M is the maximal number of layers

1.2.1.2 Backpropagation

The backpropagation algorithm aims to minimize the loss function by comparing the output $\hat{y}$ from the forward propagation with the ground truth label y and updating the weights. There are different methods to update weights. One of the well-known methods is called gradient descent, and this algorithm finds the new weight based on the derivative of the loss function with respect to the weight. As shown in figure 1.2.6, the aim is to find the best value for parameter w^{new} to minimize the loss function j. Thus, we calculate the new parameter w_1^{new} ass follows:

$$w_1^{new} = w - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 = w - \alpha \frac{\partial J}{\partial w_1^{new}}$$

where α is the learning rate. The gradient decent keep trying to find the w that minimize J until finding the minimum J or until reaching the maximal number of iteration. Moreover, while computing the new parameters we have to use the chain rule to calculate the

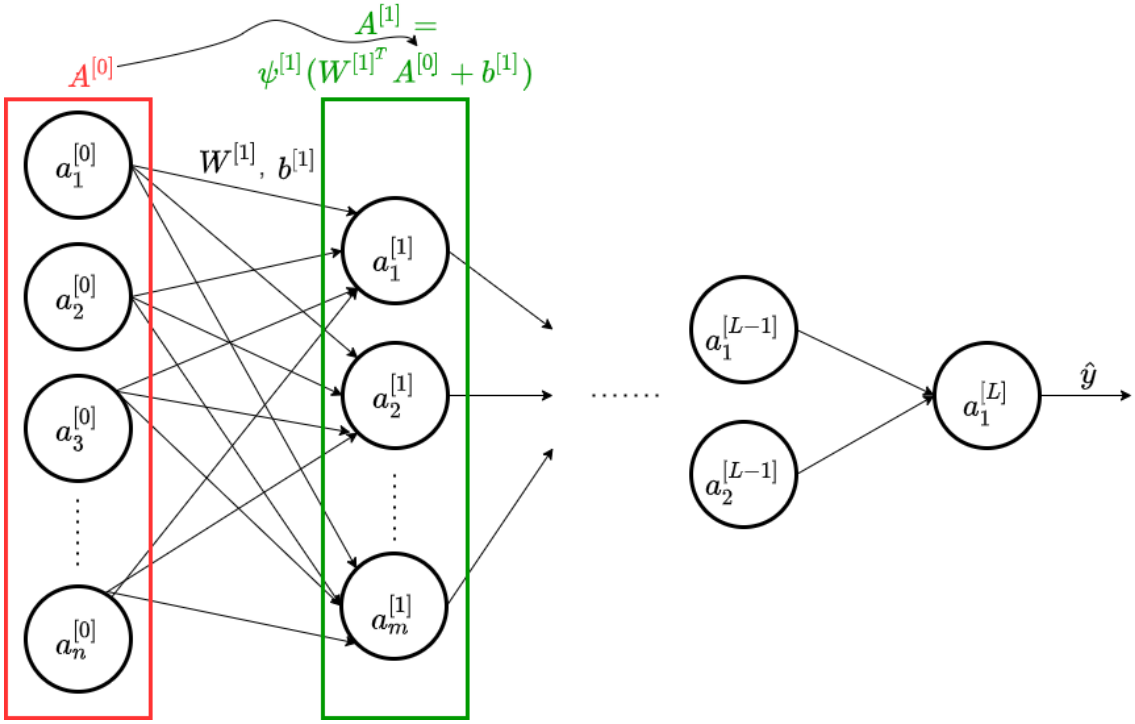


Figure 1.2.5: Forward propagation

derivative of a function. For example in figure 1.2.7, if we want to update the value of weight x we can not directly compute $\frac{\partial f}{\partial x}$. Thus, backpropagation use the chain rule to calculate it as follow:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \cdot \frac{\partial q}{\partial x} + \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x}$$

we already have:

$$\frac{\partial f}{\partial q} = z = -4$$

$$\frac{\partial f}{\partial z} = q = 3$$

$$\implies \frac{\partial f}{\partial x} = z.1 + q.0 = z = -4$$

To summarize, we explained in the previous sections how the neural network works. Algorithm 2 presents an overview of the learning process. In the next section, we present the convolutional neural networks used in chapter 4.

1.2.2 Basic CNN Components

One of the most widely Deep learning techniques used in the computer vision area is CNN. Based on the paper entitled "Receptive fields of single neurons in the cat's striate cortex." [21] published by David Hubel and Torsten Wiesel—in 1959, *Fukushima et al.*

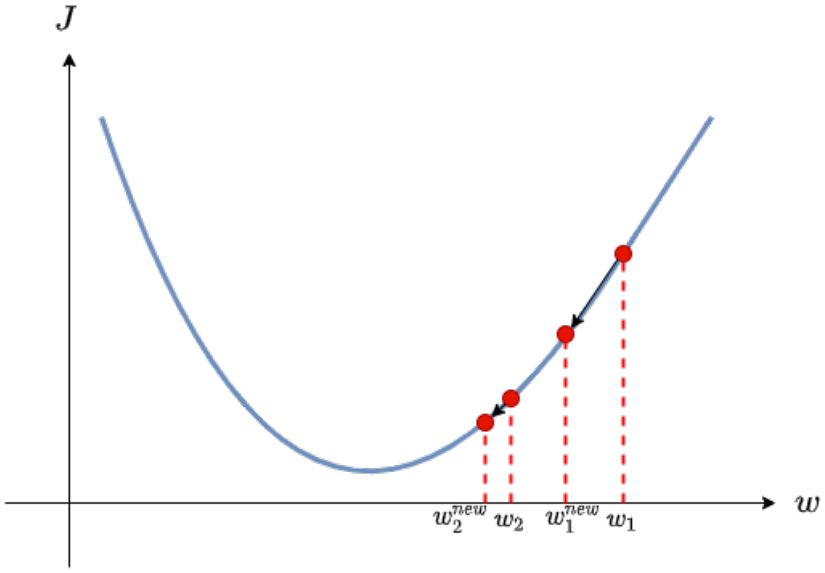


Figure 1.2.6: Gradient descent example

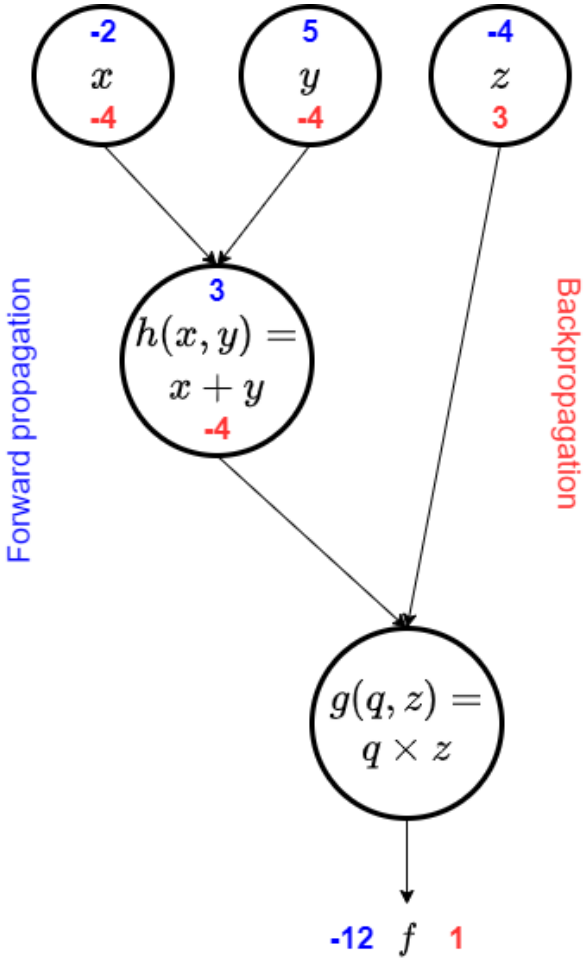


Figure 1.2.7: Simple neural network with froward propagation and backpropagation

Algorithm 2: Training Algorithm

Initialize model parameters;
Number of epochs = N;
For $i = 1$ to N;

1. $\forall i$ predict $\hat{y}_i$ from the input x_i;
2. Evaluate the Loss function j between the ground truth label y and the predicted output $\hat{y}$;
3. Apply descent method to update parameters and minimize the loss function;

Result: The trained Model;

introduce CNNs in his seminal paper on the "Neocognitron" [22]. Next, *LeCun et al.* developed convolutional neural networks that achieved outstanding results in various pattern recognition tasks [23, 24] using the error gradient. However, 2012 was a turning point in CNN history, when *Alex Krizhevsky* [25] won the ImageNet [26] competition by dropping the image classification error from 26% using traditional computer vision methods to 15% using CNN. Besides, the availability of large datasets and the increase in computing resources allow researchers to create complex CNN that was impossible previously.

CNN architecture is generally formed by succeeding convolution and pooling layers followed by one or more fully connected layers. This architecture consists mainly of two stages: feature extractor and classifier. The convolution and pooling layers extract features from input images, and then the fully connected layers classify them. In the following sections, we explain each element used to design a CNN in detail.

1.2.2.1 Convolutional Layer

The convolution layer consists of a set of filters that compute the convolution of the input images while scanning the input image concerning its dimensions. This layer has different variables to select, such as the size of filters and stride. The output of this layer is a feature map that represents the input image. In other words, the main idea of the convolution layer is to extract features from the input using the convolution operator with a set of filters.

1.2.2.2 Pooling Layer

The pooling layer is usually used after the convolution layer and does not have any parameters that need optimization. This layer reduces the computation and the number of parameters, which reduce the chance of over-fitting. The most common types of pooling layer are the max-pooling and average pooling. The max-pooling layer is a window that slides over the feature map and selects the maximum value inside the moving window;

however, the average polling is a window that slides over the feature map and averages the values inside the window.

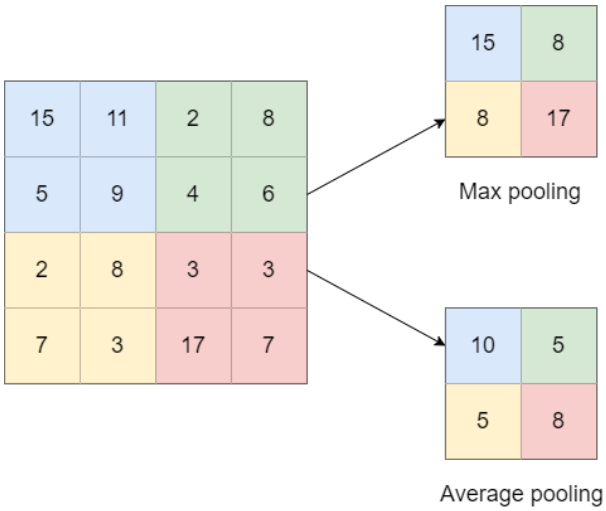


Figure 1.2.8: Pooling layer

1.2.2.3 Activation Function

The activation function is a decision function that helps the model to learn complex patterns. By choosing the suitable activation function, the learning process can be accelerated. Different activation functions are used in literature, such as sigmoid, tanh, maxout, Rectified Linear Unit (ReLU), leaky ReLU, Exponential Linear Unit (ELU), and Parametric Rectified Linear Unit (PReLU). Those functions train a non-linear combination of features. ReLu and its variants are the most preferred activation functions because they help in overcoming the vanishing gradient problem [27, 28].

1.2.2.4 Batch Normalization

Batch normalization is used to reduce the internal covariance, which accelerates the training of the model. This layer use normalization to fixes the means and variances of layer inputs. By diminishing the dependence of gradients on the scale of the parameters, batch normalization improves the gradient flow through the network, which allows the use of higher learning rates without the risk of divergence.

1.2.2.5 Dropout

Dropout introduces regularization within the network, which ultimately improves generalization by randomly skipping some units or connections with a certain probability. In Neural Networks, multiple connections that learn a non-linear relation are sometimes co-adapted, which causes over fitting [29]. This random dropping of some connections or units produces several thinned network architectures, and finally, one representative

network is selected with small weights. This selected architecture is then considered as an approximation of all of the proposed networks [30].

1.2.2.6 Fully Connected Layer

Fully connected layer is mostly used at the end of the network for classification. Unlike pooling and convolution, it is a global operation. It takes input from feature extraction stages and globally analyses the output of all the preceding layers. Consequently, it makes a non-linear combination of selected features, which are used for the classification of data [31].

1.3 Summary

In this chapter, we give a background of the methods that are used in this thesis. The first section of this chapter presents the distance map used to skeletonize the image. Whereas the second section briefly presents the theory behind deep learning. Moreover, we present the CNN concept and the main layers used to build a network in the second section.

In the next chapter, we are going to present a framework that prepares drawing to be vectorized.

Chapter 2

Preprocessing of Engineering Drawings

Contents

2.1 Introduction

The technical drawing is a visual language that describes how something functions or is constructed; it is a tool for communicating industry and engineering ideas. This language starts by using old technical drawing instruments such as T-square, technical pen, and compasses; Next, Computer-Aided Design (CAD) systems are used to automate and

accelerate the process of representing engineering drawings. Those systems are used later to generate 2-D and 3-D designs that contain more detailed information.

Companies have an extensive database of old manufactured designs without the associated CAD files. Paper-based drawings are highly available based on [32]; these files are usually scanned and saved as images. We aim to modify, update existing components, or produce 3D models from orthogonal views automatically from those drawings. To the best of our knowledge, the CAD software and the 3-D reconstruction algorithms, such as [33, 34, 13] required vector data as input. Therefore, the increased need for a computer-aided system converting paper drawings to vector data is essential.

The paper-based engineering drawing consists of three main parts: borders, tables, and graphics (different views). Oftentimes, scanned and old paper-drawings have broken or distorted information. Thus, we aim to build an automated process that identifies and cleans graphic elements for generating vector data. This process starts by separating graphic elements from other elements such as table and border, then denoising each view, and finally detecting arrowheads location (arrowheads can be used as hint to extract dimension set). In other words, the input is a paper-based technical drawing; the output is a set of clean views ready for the vectorization process. Figure 2.1.1 presents an overview of the preprocessing method, which is explained in detail in the following sections.

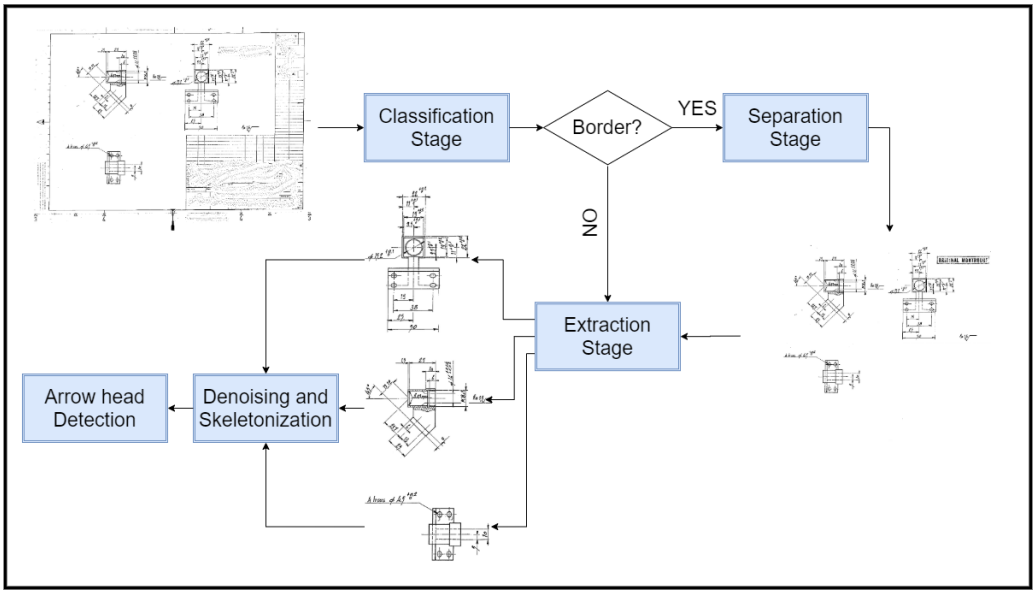


Figure 2.1.1: Preprocessing stage overview

This chapter outline is given as follows: Section 2.2 presents the proposed method, which extracts orthogonal views from drawing templates using the DBSCAN clustering algorithm. In Section 2.3, We trained a U-net model to clean each orthogonal view extracted from the section 2.2. Section 2.4 describes the process of locating arrowheads in each orthogonal view using the concept of Bag of Visual Words.

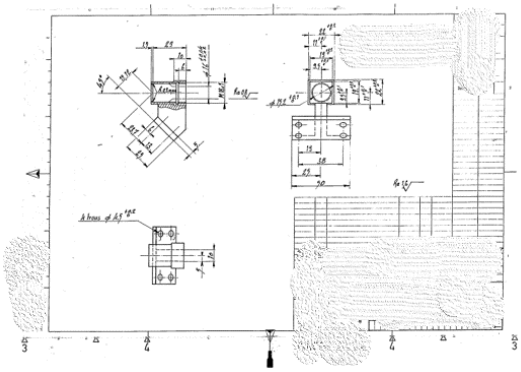
2.2 Views Extraction

Engineering drawings have different templates; they can be landscape or portrait, and the table size and view distribution can vary from one to another. The common thing between all templates is that the table is connected to the border of the template. Nevertheless, some engineering drawings do not contain any border and table; they only have the orthogonal views. Moreover, the number of projected views is indeterminate. Thus, we proposed an automated method that detects either paper-drawing containing border or not, removes the border and table when needed, and separates views. In other words, this method input is a paper drawing, and the output is a set of different views images extracted from the paper drawing.

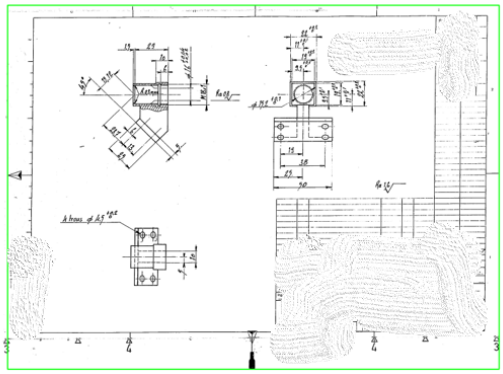
• ***Classification Stage:*** We begin by classifying engineering drawings into "drawings with border" and "drawings without border". To reach our goal, we detect the rectangle of the largest area in the engineering drawing image by computing the contours (curves that are joining connected pixels that have the same intensity) [16]. If the detected rectangle is equal or greater to 80% (based on experiments) of the image area, then the drawing contains a border. Otherwise, the engineering drawing has only orthogonal views. Figure 2.2.2 shows an example of classification stage. Figure 2.2.2a and figure 2.2.2c represent the input of classification stage, figure 2.2.2b shows that the detected rectangle has an area greater to 80% of the image area. Besides, figure 2.2.2b shows that the detected rectangle has an area smaller to 80% of the image area. Thus, figure 2.2.2b belong to the class "drawings with border", whereas figure 2.2.2b belongs to the class "drawings without border".

• ***Separation Stage:*** Engineering drawings with borders are treated to separate graphics elements and eliminates other information. To achieve our goal, we detect all the contours in the image. The second-largest contour area separate views and the border, and the table. Figure 2.2.3a shows the original image used as input for the separation stage, figure 2.2.3b represents the detected rectangle with largest area, figure 2.2.3c represents the detected contour with second largest area and figure 2.2.3d shows the information that will be eliminated to preserve only graphics elements.

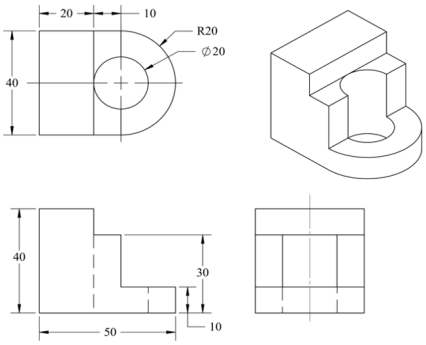
• ***Extraction Stage:*** After separating graphic elements from other elements, we are looking to separate the views. As mentioned before, the number of views can vary from one drawing to another. Thus, we need an unsupervised method ables to automatically separate views. Clustering is an unsupervised method used to separate data into groups based on similar properties such as distance between points or densities of points. In our case, we select the k-means algorithm [17] and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm [18] because the data distribution seems to match the underlying hypothesis of these algorithms where each view is separated from



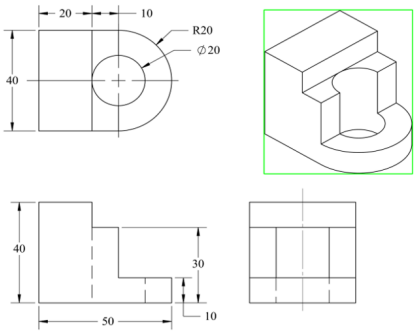
(a) Original image



(b) Detected rectangle in green - Drawing with border



(c) Original image



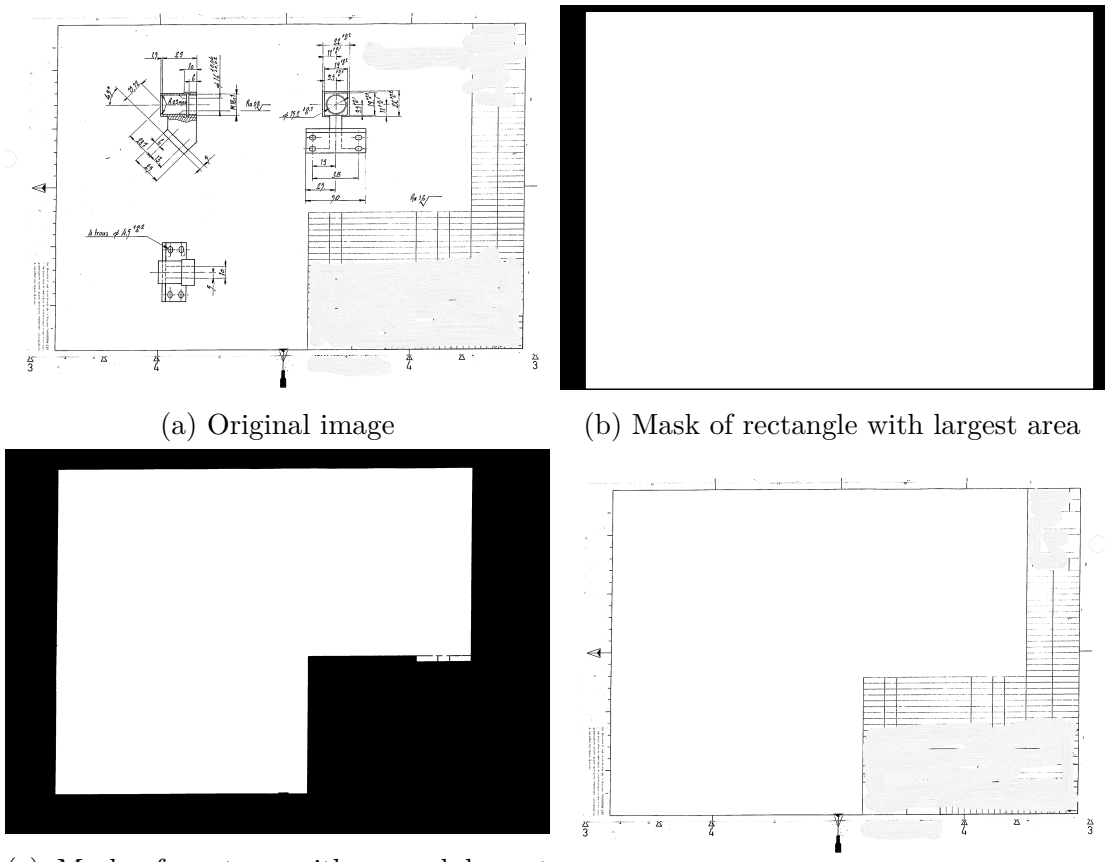
(d) Detected rectangle in green - Drawing without border

Figure 2.2.2: Classification stage example

the other by white space. Thus, K means works when its centroids are placed in the center of each view, and the DBSCAN works because the pixels of each view are very close while each view is separated from the second view with a large distance. Hence we briefly describe and compare the two different clustering methods in the following paragraphs.

• *K-means:* K-means clustering[35, 17] is an unsupervised machine learning method widely used due to its simplicity and efficiency in different fields. K-means algorithm intends to group n points into k distinct non-overlapping clusters, where each data point belongs only to one cluster. In other words, the k-means algorithm assigns the data point to a cluster that has a minimum squared distance between the data point and the centroid (center of the cluster) of the cluster.
The k-means algorithm requires two input parameters: the data and the number of clusters k. The algorithm works as follows:

1. Generate k random centroids called c_k



(a) Original image

(b) Mask of rectangle with largest area

(c) Mask of contour with second largest area

(d) Separated elements - border and table

Figure 2.2.3: Separation stage example

2. Find the nearest centroid c_k for each point p_i using the following equation

$$\arg\min_k D(p_i, c_k)$$

where D is the euclidean distance between p_i and c_k

3. Assign each data point p_i to the nearest cluster (nearest centroid)

4. Compute new location for each centroid c_k by computing the mean of all points p_i assigned to cluster k in the previous step:

$$c_k = \frac{1}{k} \sum_{p_i \in c_k} p_i$$

5. Repeat from number 2 until we reach the maximum iteration or reach the centroid stability (the centroid location does not change while iterating).

• *DBSCAN :*DBSCAN [35, 36, 18] is a well known density-based clustering algorithm, which group points based on Euclidean distance and the minimum number of neighbors

points. The DBSCAN algorithm can assign points as outliers. Moreover, the formed clusters can have varying shapes based on density area.

In addition to the dataset, the DBSCAN algorithm requires two other parameters defined as follows:

- $eps(\epsilon)$: The maximum Euclidean distance between two points to be considered as neighbors. q and p are said to be neighborhood when $dist(p, q) \leq \epsilon$.
- $minPts$: The minimum number of points required to represent a dense area.

Based on the input parameters, the algorithm groups points into three different categories:

- $Core$: The core point is a point that have at least $minPts$ neighborhood (included itself) inside a circle of radius eps.
- $Border$: The border point is a point that is within a neighborhood of core point but does not have $minPts$ neighborhoods. It is called a border point because it represents the border of a dense region.
- $Outlier$: The outlier point is a point that is neither a border point nor a core point. In other words, the noise point is not within a neighborhood of the core point and does not have a $minPts$ neighborhood to form a new cluster.

To understand the algorithmic steps for DBSCAN clustering, we first introduce the following concepts:

- Directly Density Reachable: if a point p belongs to the neighbors of point q and q is a core point, then the point p is directly density-reachable from a point q
- Density reachable: if a set of points $p_1, \ldots, p_n$ where $p_1 = q$ and $p_n = p$ such that p_{i+1} is directly-reachable from p_i, then the point p is density reachable from the point q
- Density Connected: if a point p and a point q are density reachable from a third point o, then the point p is density connected to the point q.

The DBSCAN algorithm works as follows:

1. Select a random point p from the input dataset such that p has not been visited.
2. Find all points density reachable from p
3. If p is a core point, a cluster formation starts. Otherwise, the point p is labeled as outliers. However, the point p might be a part of a cluster when it is density connected to another point. In both cases, that point is marked as visited.

4. If p is a core point, all points directly density reachable from p is added to the cluster along with their own ϵ neighborhood. Repeat the process of adding all points in the ϵ neighborhood to the cluster for all new points that have been just added to the cluster group.

5. Repeat steps 3 and 4 until the density-connected cluster is wholly found. In other words, repeat steps 3 and 4 until all points in the cluster are assigned and visited.

6. When the current cluster is completely found, Repeat all steps until all points in the dataset are visited and labeled.

Figure 2.2.4 shows an example of the DBSCAN algorithm. Assume $minPts = 4$; the red points are core points because they have at least four points within a circle of radius ϵ (including the point itself). The yellow points are border points because they are directly Density Reachable from core points, but they do not have sufficient neighbors. The blue point is labeled as outliers because it is not a core point, does not have sufficient neighbors, and is not density reachable from another point

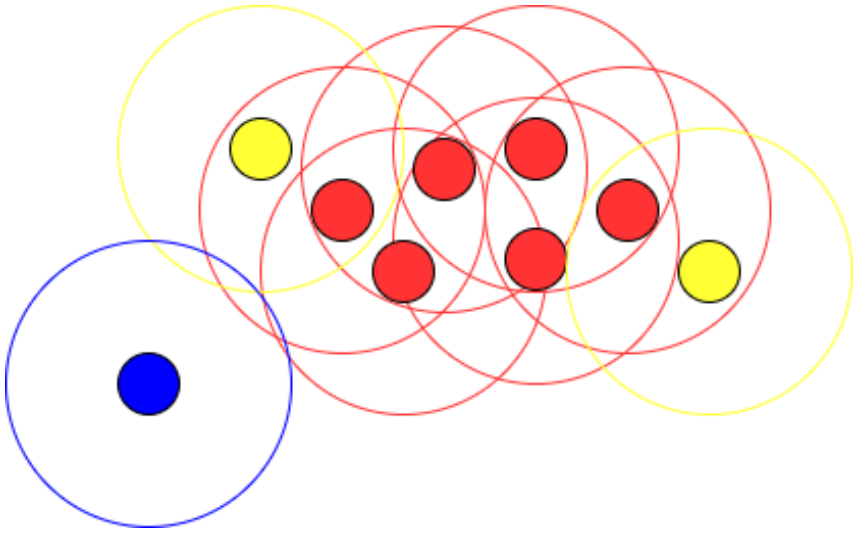
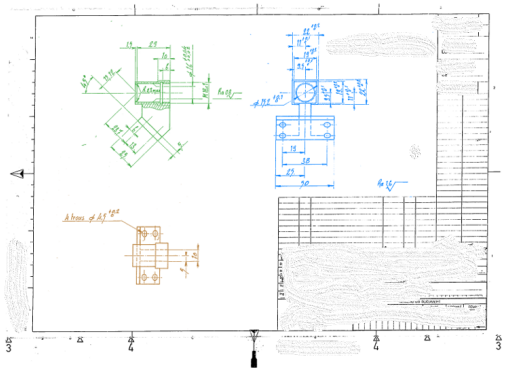


Figure 2.2.4: DBSCAN clustring example

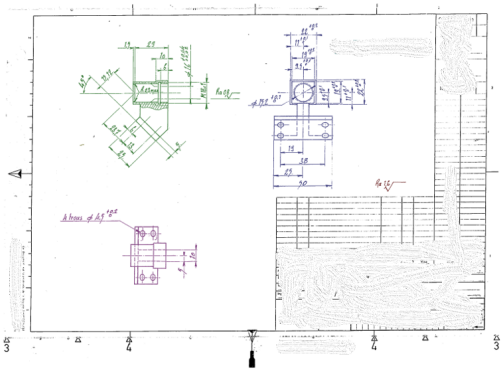
The k-means and DBSCAN clustering methods are tested on different images. The DBSCAN input parameters ϵ and $minPts$ are respectively 30 and 75; these values are selected based on experiments. However, for the k-means algorithm, the best number of clusters is that equal to the number of views. Thus, to determine the best cluster number k, we introduce a metrics called "inertia" which is the sum of samples squared distances to their closest cluster center (centroid). The high value of inertia indicates that samples in the cluster are far from each other, in other words, samples are less similar to each other. The k-means algorithm is computed with k between one to ten, and the inertia of each k is stored. The best number of clusters is the one that minimizes the inertia value

and minimizes the number of clusters. Thus, we use the elbow methods to determine the optimal k. For each tested image, the value of k might change.

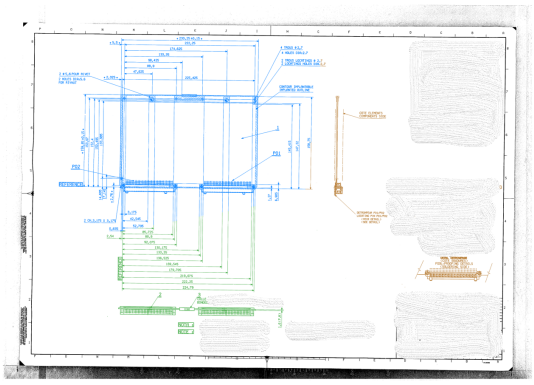
After determining input parameters, we found that both methods work well, as shown in Figure 2.2.5a and Figure 2.2.5b . These methods are able to separate views perfectly due to the white space that separates them. However, DBSCAN clustering outperforms k-means clustering when the white space is smaller between views. Figure 2.2.5c shows that the k-means algorithm fails to separate views; however, figure 2.2.5d shows that the DBSCAN method separates views correctly. Moreover, it is obvious from the small object (colored blue in figure 2.2.5a and brown in figure 2.2.5b) above the table in figure 2.2.5a and 2.2.5b that k-means algorithm add noisy data to the views. However, the DBSCAN cluster only views and creates another cluster for noisy data, which can be rejected later.



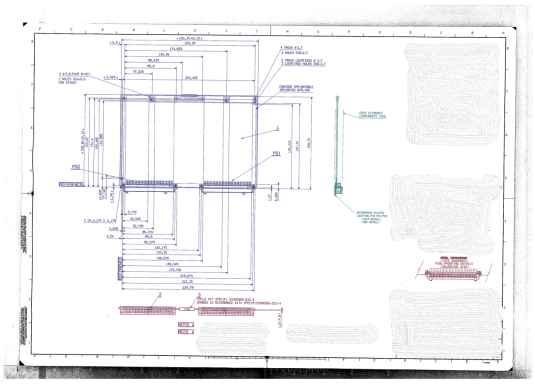
(a) K-means clustering



(b) DBSCAN clustering



(c) K-means clustering



(d) DBSCAN clustering

Figure 2.2.5: Clustering stage example

After extracting views from the input image, the next stage is to denoise and skeletonize them as introduced in section 2.3. The denoise and skeletonize stage is the most important one because they should clean and skeletonize the views while preserving all sensitive information, as explained in the following section.

2.3 Denoising and Skeletonization

With the increasing usage of images in different domains such as medical, satellite, and others; Image denoising becomes a more significant research area in computer vision—the noise infection of images increases due to transmission error or compression. When coming to the engineering drawing area, the noises usually come from the scanning process that digitizes paper drawings. The scanning process of engineering drawing usually generates salt and pepper noises. In scanned engineering drawings, the noisy pixels connected to graphical elements are the most harmful because they might fragment the primitive by losing important pixels or combining two or more primitives.

Nowadays, image denoising is essential in the image processing area. It aims to reduce noisy pixels and make images clearer to understand. Several methods were proposed to solve this problem, and for different types of images, [37, 38, 39]. More specifically, many denoising methods are proposed in the literature to reduce the salt and pepper noises in technical drawings and documents.

One of the well-known filters used in previous works is the Median filter [40]; it is a basic filter used to remove salt pepper noise from images. This filter slides a $(k \times k)$ window over the input image and replaces the window center with the median of the input window values. The main drawback of this method is the high distortion of thin lines and corners because they are surrounded by white pixels, which lead to loose important information and thus affect the vectorization process. This would obviously deteriorate the performance of vectorization and 3D reconstruction as detailed below.

Moreover, the Center Weighted Median (CWM) filter [41] is inspired by the median filter. The purpose of this filter is to retain details such as thin lines. Therefore, the proposed method gives the center more weight compared to the neighborhood. Another type of filer is the Morphological one [42]. This filter computes a combination of dilation or erosion to remove salt and pepper noise for images. The main disadvantage of this method is the distortion of thin lines and the high sensitivity when primitives are close (can lead to join close primitives). KFill [43, 44] and EnhKfill [42] algorithms are based on sliding $k \times k$ window over the image and deciding either to flip or not the pixel based on neighborhood. The drawback of those methods is selecting the k value; if k is small then the one-pixel-wide lines are shortened from their endpoints; otherwise, if k is large then some graphical elements are extensively eroded.

More recently, the Active Detector filter proposed by *Simard and Malvar* [45] is based on connecting component labeling; It considers the specification of documents and avoids thin line distortions. However, this algorithm is not able to remove noises connected to graphics elements. Moreover, The Track And May Delete (TAMD) algorithm proposed by *Al-Khaffaf et al.* [46] is a post-processing algorithm that can be added to other denoising algorithms such as Active detector. TAMD enhances salt and pepper noise removal algorithm by tracking and analyzing thin graphical elements and deciding whether to preserve

or remove these elements.

Most traditional methods have difficulties solving the problem entirely because they are based on thresholds, and some methods can lead to loose important information. Lately, deep convolution neural networks (CNN) have shown outstanding results in several image processing tasks, such as object detection and classification. Besides, different deep learning methods [47, 48, 49] were used to clean images. In this work, we choose the U-net architecture proposed for medical image segmentation where preserving edges and boundaries is highly important, similarly to engineering drawings.

This section presents a U-net model trained and tested to denoise images and compared our results with previous methods. In section 2.3.1, we introduce the U-net architecture. Section 2.3.2 describes the generated dataset. Section 2.3.3 introduces the distortion measurements used to compare the trained model with previously proposed methods. In section 2.3.4, we discuss the results and compare them with previous methods.

2.3.1 U-net

U-net is a fully convolutional network created for segmenting biomedical images such as tumors in the lungs or brains [50] by *Olaf Ronneberger et al.* [51]. The "U" in its name is inspired by its shape, as shown in figure 2.3.6. The outstanding performance of this architecture convinces researchers to use it in other fields [52, 53].

The U-net mainly consists of two parts, the encoder and the decoder. While the encoder generates the image features map, the decoder converts the feature map into the segmented image. The encoder is composed of four blocks, where each one consists of two (3×3) convolution layers followed by a ReLu activation function and batch normalization. The last layer in each block is (2×2) max-pooling layer. The decoder is also composed of four blocks. Each block consists of a deconvolution layer (stride of 2), a concatenation with the corresponding cropped feature map from the encoder, and two (3×3) convolution layers followed by a ReLu activation function and batch normalization.

The connections between the encoder path and decoder path layers lead to a good prediction due to the combined information about localization and context. Besides, this architecture accepts the different sizes of images as input due to the absence of a dense layer. Moreover, this architecture can be trained only with few numbers of images.

2.3.2 Dataset Description

Our dataset is generated using the ABC dataset [55] and FreeCAD software. The 3D model from the ABC dataset is rotated to extract three orthogonal view images of size 1024×1024 using FreeCAD python scripting. The number of generated images is 3800. To generate noisy images, we add salt and pepper noise for each one with a random ratio between 5% and 40%. Thus, we obtain a dataset of 7600 images of size 1024×1024 where 3600 are clean images, and 3600 are noisy. An example of the dataset is shown in Figure

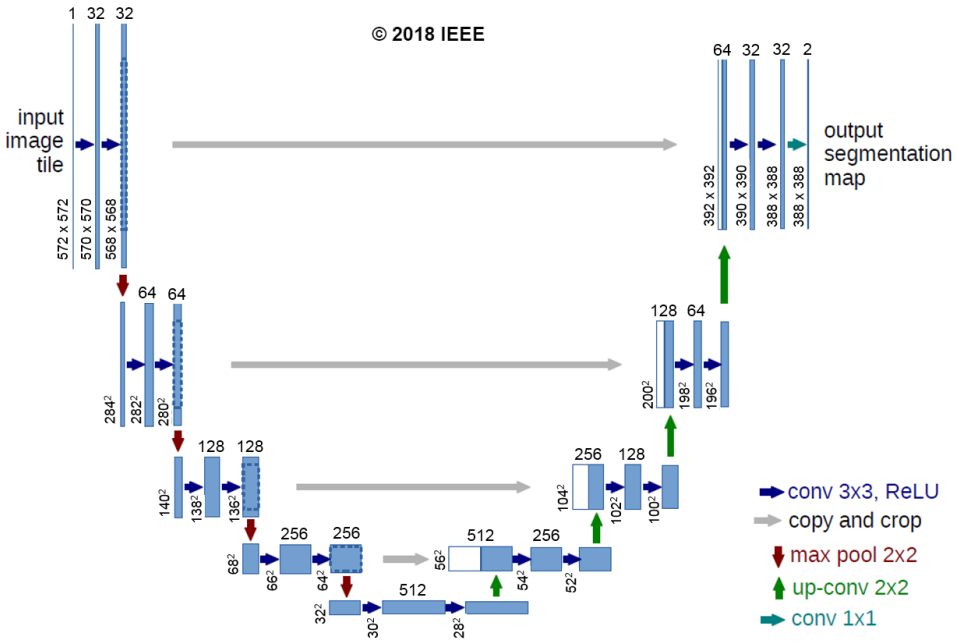


Figure 2.3.6: U-net architecture [54]

2.3.7, where Figures 2.3.7a and 2.3.7c are the clean images and Figures 2.3.7b and 2.3.7d are the noisy images.

2.3.3 Distortion Measurements

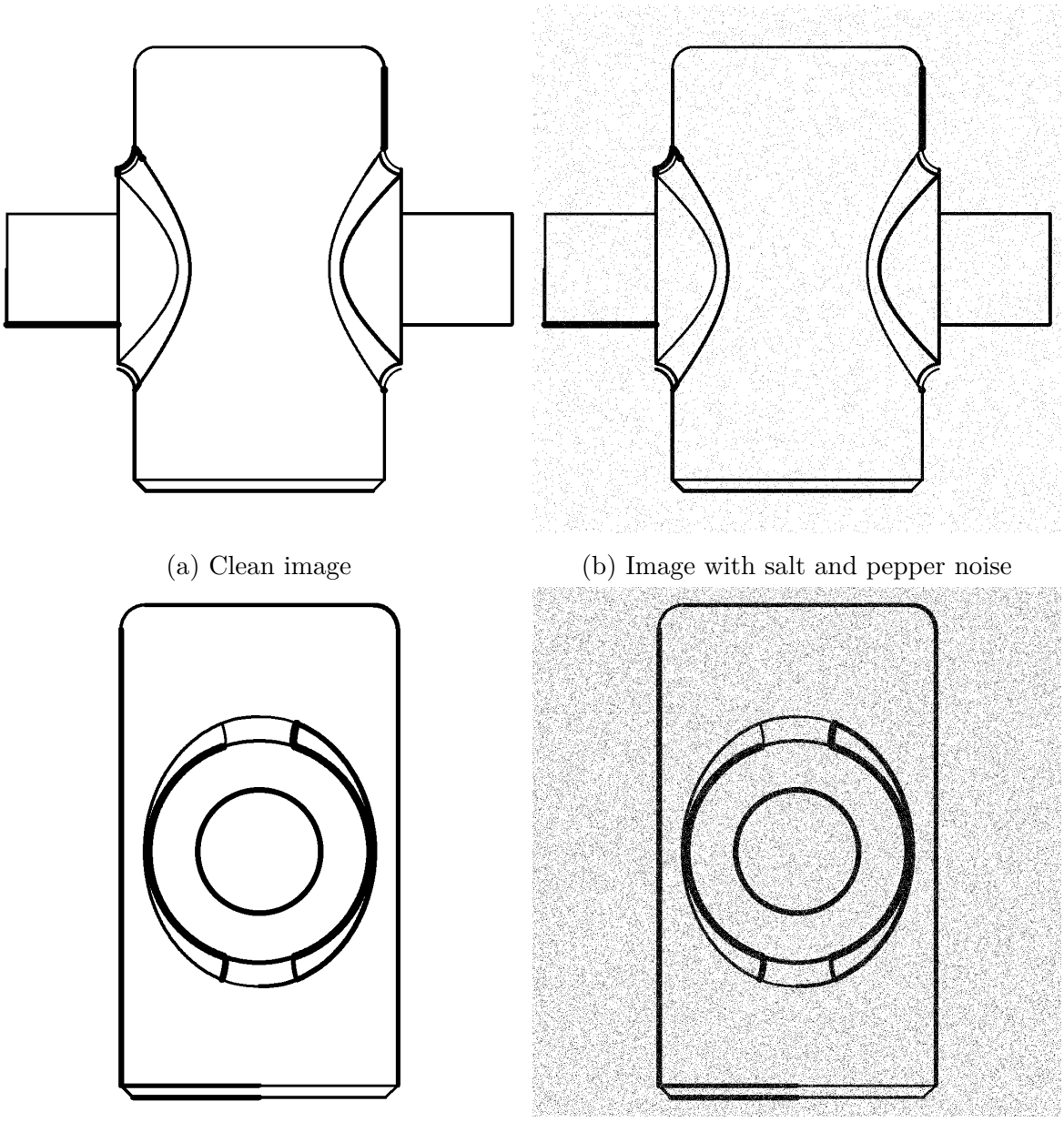
Distortion measurements are a set of mathematical formula used to estimate the similarity between images; these measurements are widely used to compare the efficiency of noise reduction algorithms. These measurements are used to find the best filter that can reduce the noise and prevent drawing fragmentation to increase the accuracy of the vectorization process. To evaluate the performance of the proposed method and compare it with the previous method, we describe two distortion measurements tools as follows:

- Peak Signal-to-Noise Ratio (PSNR) [56]: The PSNR is a well-known measure used to evaluate filters performance by comparing the restored image with the ground truth image. The PSNR value is based on the number of changed pixels between the ground truth image and the restored image. The PSNR is represented as follows :

$$PSNR = 10 \times log_{10}(\frac{MAX}{\sqrt{MSE}}) \tag{2.1}$$

  where MAX is the maximum intensity and MSE is the mean square error, which is estimated as follows:

$$MSE = \frac{1}{hw} \sum_{i=1}^{h} \sum_{j=1}^{w} \left(X_{(i,j)} - Y_{(i,j)}\right)^2 \tag{2.2}$$



(a) Clean image (b) Image with salt and pepper noise

(c) Clean image (d) Image with salt and pepper noise

Figure 2.3.7: Genrated dataset example

where X is the original image and Y is the restored image. h and w represent respectively the height and width of image X (Assuming that X and Y have the same height and width). A higher PSNR value indicates a better-restored image quality.

- Distance Reciprocal Distortion (DRD) : The DRD measurement method is proposed by *Haiping et al.* [57, 58]; this method evaluates the amount of distortion in binary document images. This method is based on human visual perception, which more sensitive when the flipped pixel has neighbors with close pixel values (the black pixel connected to a border is less seen from a black pixel surrounded by white

pixels). This method measure the distortion between an input image called f and the filtered image g. The distortion for all the S flipped pixels (changed from white to black or vice versa) is computed as follows:

$$DRD = \frac{\sum_{k=1}^{S} DRD_k}{NUBN} \tag{2.3}$$

where k is the number of flipped pixels $k = 1, 2, \ldots, S$ and NUBN is the number non uniform 8×8 blocks in the input image f.

The algorithm computes the distortion DRD_k of the k-th flipped pixel $(x, y)_k$ in $g(x, y)$ from an $m \times m$ block B_k centered at $(x, y)_k$ in $f(x, y)$ and using a 5 x 5 normalized weight matrix W_{Nm} (refer to [57]) as follow:

$$DRD_k = \sum_{i,j} [|B_k(i, j) - g[(x, y)_k]| \times W_{Nm}(i, j)] \tag{2.4}$$

where $1 \leq i, j \leq 5$ as defined in [57].

2.3.4 U-net Validation for Denoising

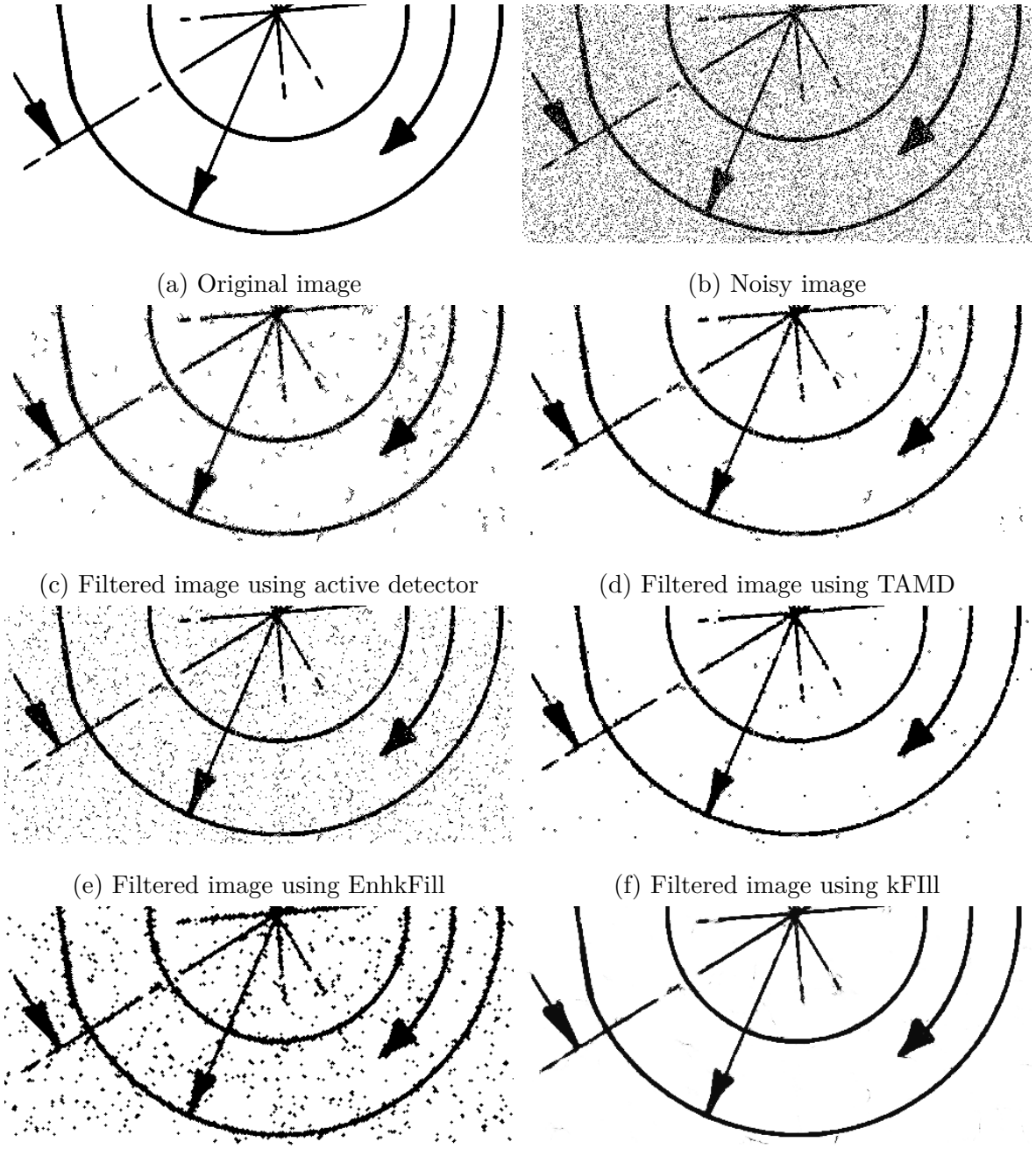
In this experiment, we trained a U-net network to denoise technical drawings using the described dataset. The model is trained using the MSE as loss function, the ADAM optimizer, the learning rate is set to 10^{-4}, batch size is equal to two, the steps per epoch is set to 2000, and the number of epochs is set to 30. The input image should be a binary image of size 1024×1024. If the image is not binary then the Otsu algorithm [59] is used to binarize it. If the input size is smaller than 1024×1024, we add white borders around the input. In the tested images, the size of input does not exceed 1024×1024. The trained model reaches 99.97% of accuracy.

The algorithms are evaluated using images from different International Association for Pattern Recognition (IAPR) contests on Graphics RECognition (GREC): GREC 2003, GREC 2005 [60] where:

1. GREC 2003 contains four scanned drawings (named as '1','2','3','4') in 256 grayscales and binarized with moderate thresholds. Images '1' and '4' are binarized again with a higher threshold to generate images '1_230' and '4_230' with thicker line width. However, images '2' and '3' are binarized with smaller threshold to generate images '2_100' and '3_100'. Moreover, images '1_n4', '2_n4', '3_n4' and '4_n4' are generated by adding synthesized noises [61] .

2. GREC 2005 contains six real scanned drawings (named as '5','6','7','8','9','10') where each one is used to generate two images one with random noise ('5_rn',...,'10_rn') and the other with salt and pepper noise ('5_sp',...,'10_sp').

We used only clean images of GREC 2003/2005 and thin/thicker images of GREC 2003 to thin and thicker. For each image, we add uniform salt-and-pepper noise with different noise levels (5%, 10%,15%,20%, 30%, and 40%), then we add white borders such that each image size reaches 1024×1024. The trained model is compared with previous methods using a qualitative metric (visually) and quantitative metrics (PSNR, DRD detailed in the previous section) .



(a) Original image (b) Noisy image

(c) Filtered image using active detector (d) Filtered image using TAMD

(e) Filtered image using EnhkFill (f) Filtered image using kFIll

(g) Filtered image using open-close (h) Filtered image using U-net

Figure 2.3.8: Qualitative results - Part of image 6.TIF cleaned by different algorithms

Quantitatively, Figures 2.3.9a and 2.3.9b shows the PSNR and DRD results of differ-

ent algorithms with different images. The U-net algorithm has the highest PSNR values and the lowest DRD values for all images; thus, the U-net outperforms all other methods. In addition, to evaluate the algorithm stability in different noise levels, we filtered the same image (1.tif) with varying levels of noise. Figure 2.3.9c shows the DRD results of computing different noise removal algorithms on one image with a different noise level. Almost all tested algorithms are stable when the noise level is less than 15%. Nevertheless, Kfill and Active detector are more stable than other algorithms, except for U-net when coming to a high noise level. Lastly, the U-net algorithm is the most stable as the difference between the DRD of 40% noise level, and the DRD of 5% noise level is the smallest.

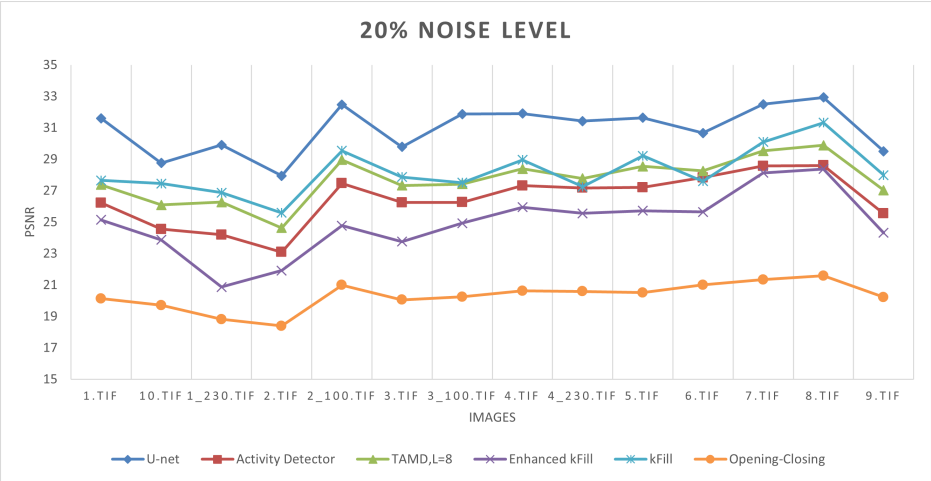
Furthermore, Figure 2.3.8 shows a qualitative comparison between different algorithms. Figure 2.3.8a shows a partial part of the ground truth image 6.tif and figure 2.3.8b shows the noisy image with noise level equal to 30% . Figure 2.3.8f and 2.3.8d shows that Kfill and TAMD algorithms have good performance in removing salt and pepper noises in high noise level, but these algorithms distort edges. Moreover, Figure 2.3.8c shows that the active detector algorithm is not able to remove attached pixels to the graphical elements, which highly affects the vectorization stage as previously stated. Besides, Figure 2.3.8g shows that the open-close filter is not able to remove noises. The best overall result is the filtered image with U-net shown in Figure 2.3.8h, the graphical elements are smoothed and well preserved even when the noise level is high.

To summarize, the U-net model is trained using the described dataset. The experiments show that the trained U-net model outperforms all other traditional noise removal algorithms (qualitatively and quantitatively). However, we should highlight that the main drawback of the trained model is the input size, which should be less than or equal to 1024×1024. The next stage after denoising is the skeletonization process, which is described in the following section.

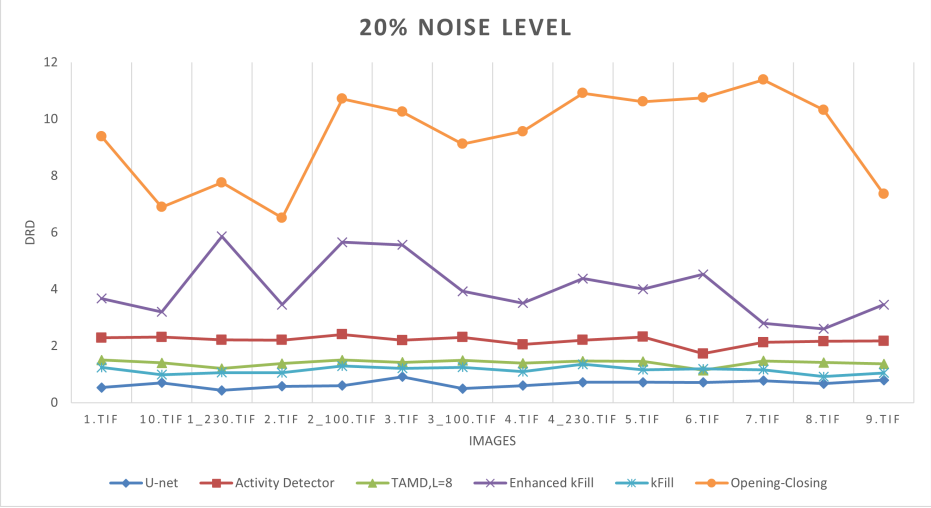
2.3.5 Skeletonization

In engineering drawings, the thickness of primitives might reflect important information; for example, the dimension lines are usually thinner than other lines. Besides, the accuracy of estimating a thin line equation is usually higher, then estimating a thicker line equation due to the additional unuseful information in thicker lines. Thus, we introduce the skeletonization process, which aims to reduce foreground regions within a binary image to a skeletal remnant while preserving the extent and connectivity of the original region. Skeletonization algorithms are widely used in raster vector conversion [62, 63, 64, 65] because it reduces the problem of vectorization to segmenting a 2D discrete curve into meaningful features.

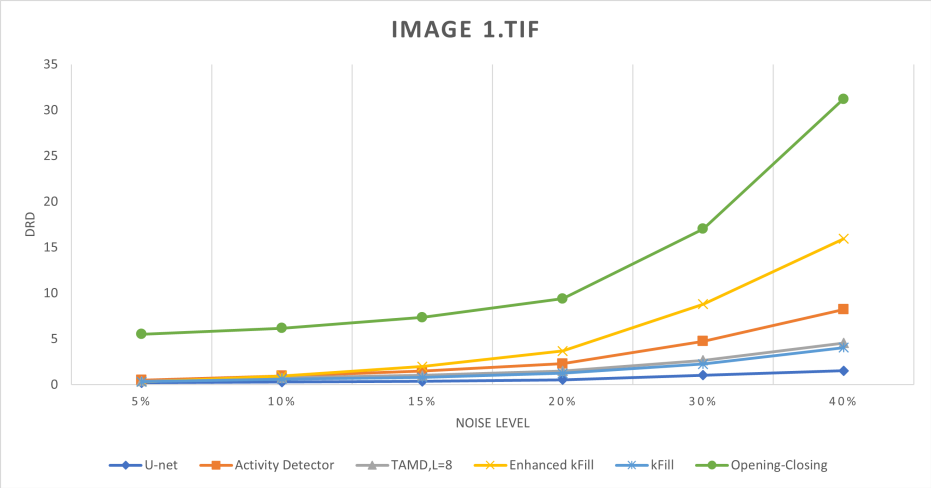
There are mainly three skeletonization techniques: skeletonization using distance map, skeletonization using morphological operation, and skeletonization using Voronoi diagram.



(a) PSNR of different algorithms with noise level = 20%



(b) DRD (5∗5 window size) of different algorithms with noise level = 20%



(c) DRD of image 1.tif of different algorithms and different noise level

Figure 2.3.9: Quantitative results

Chamfer distance skeletonization proposed by *Di Baja* [66] is used in our method for different reasons (see section 1.1.2). First, this skeletonization method produces a wellshaped skeleton. Second, (3,4) chamfer distance skeletonization is robust to noise. Moreover, this algorithm is reversible so we can predict the original thickness of the line drawing. Those features are discussed in detail in [65].

After generating the skeleton, the algorithm detects and labels the junctions of the skeleton. The algorithm [67] labels each pixel in the skeleton by counting its neighbors (8-neighbor):

- If one neighbor is detected, the selected pixel is classified as an endpoint.
- If the number of neighbor pixels is two, the pixel is classified as core-point.
- If more than two neighbor pixels are detected, the pixel is classified as a junction-point.

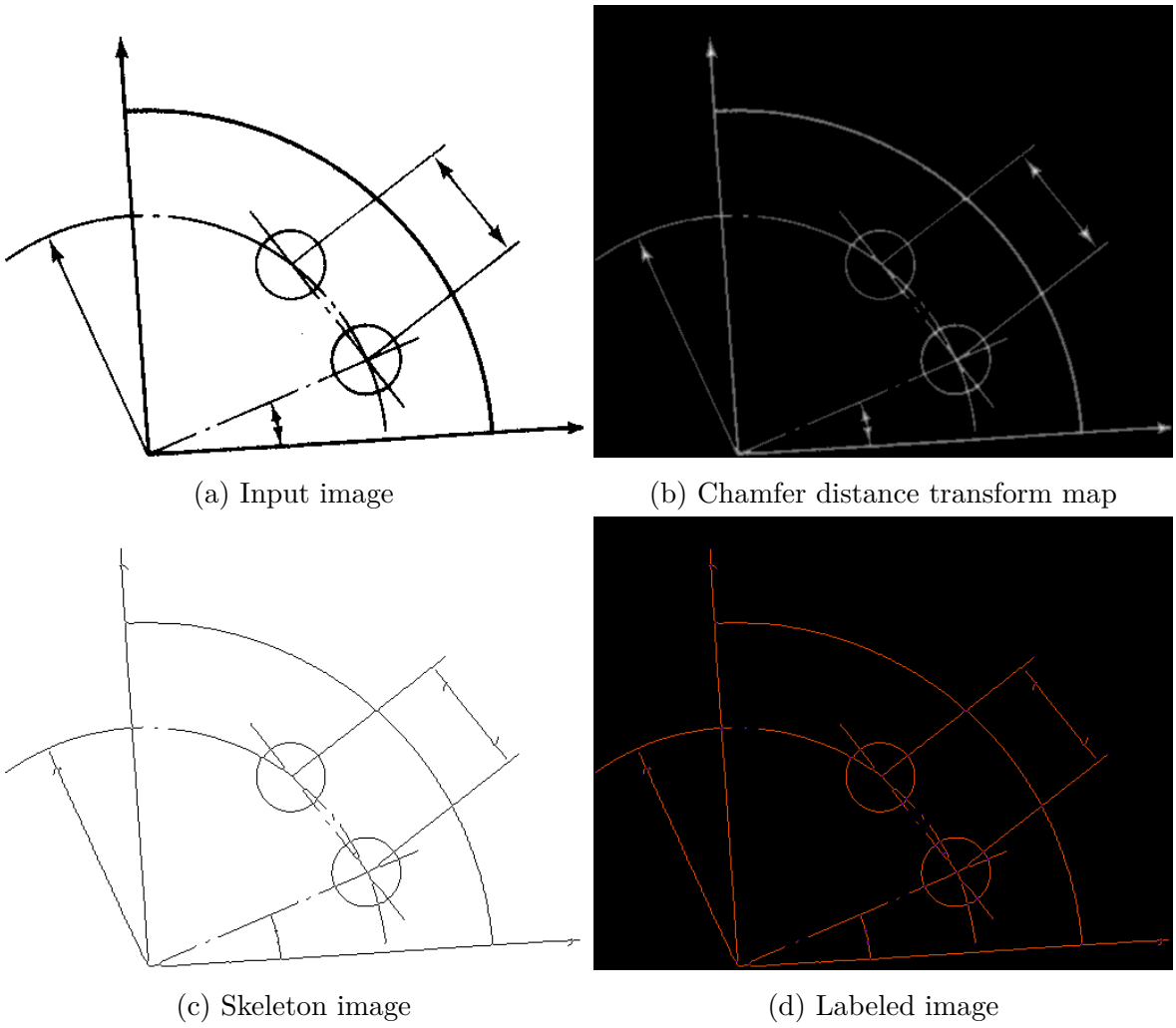
Figure 2.3.10 is an example of the skeletonization and labeling process. In figure 2.3.10a, we present the input Image. Next, the Chamfer distance map of the input image is computed, as shown in figure 2.3.10b. Figure 2.3.10c presents the well-shaped skeleton computed from the chamfer distance map. Finally, the labeled skeleton is shown in figure 2.3.10d, where the core points are orange, the endpoints are blue, and the junction points are purple.

2.4 Arrow Head Detection

In engineering drawing, it is well known that each entity brings essential information to understand the drawing. For example, the detection of borders and tables in templates leads to extracting different views, as shown in Section 2.2. The arrowhead is another precious information; its detection is often used as knowledge for detecting dimension sets. However, arrowheads may have varying size and orientation and thus to automate the arrowheads detection process, we need to design a robust algorithm working with different conditions.

In computer vision, the arrow head detection is considered as an object detection problem. Object detection is a well-known research area in computer vision; it is used in different areas such as face detection [68], cancer detection [69], and others. In addition, object detection has been used in technical drawings to detect symbols [70, 71, 72] such as doors, electronic components, and other entities. Despite the large number of research that turns around technical drawing understanding, few methods focus on detecting arrows head in line drawings.

The author in [73] proposes a method that searching for a combination of three points. Those three points will be classified as an arrow or not based on a set of merged criteria using Choquet Integral. The author in [74] proposed a method where the input is a



(a) Input image (b) Chamfer distance transform map

(c) Skeleton image (d) Labeled image

Figure 2.3.10: Chamfer distance skeletonization process

grayscale image. OTSU is the multilevel algorithm used to binarize the image. A morphological operation chosen based on the type of arrow to detect is applied. A post-processing method based on erosion and dilation operation is applied to avoid false-positive arrows. The author in [75] proposed a method based on convolution networks to localize and classify characters in engineering drawings (Not only arrows).

In this section, we proposed a method based on Bag of Visual Word (BoVW) to detect arrowheads. This method has two main stages: the training stage and testing stage. The training phase use BoVW to train a model to classify images by extracting features from images in the dataset and train a classification model to classify them as arrowhead or non-arrowhead. Whereas, the testing stage classifies new images by extracting their features and uses the trained classification model to classify it.

The proposed method reduces the problem from searching over all the image with different scales to an image classification problem by eliminating thin edges and classify only thick objects which usually represent noises, junction or arrowheads. The algorithm uses the 3,4 chamfer distance map generated from the skeletonizantion process to compute

the value of thick pixels (see previous section). Next, we generate $m \times m$ windows centered at thick pixels where m is the value of pixel (in distance map). Those windows are classified as arrow or non arrow.

The following section describes a proposed method based on Bag of visual word to detect arrowheads in images. In section 2.4.1, we are going to explain the idea of Bag of words briefly. In section 2.4.1.1, we are going to describe the dataset used to train the model. In sections 2.4.1.2 and 2.4.1.3, we are going to introduce the features and classifier used. Section 2.4.4 discusses the results.

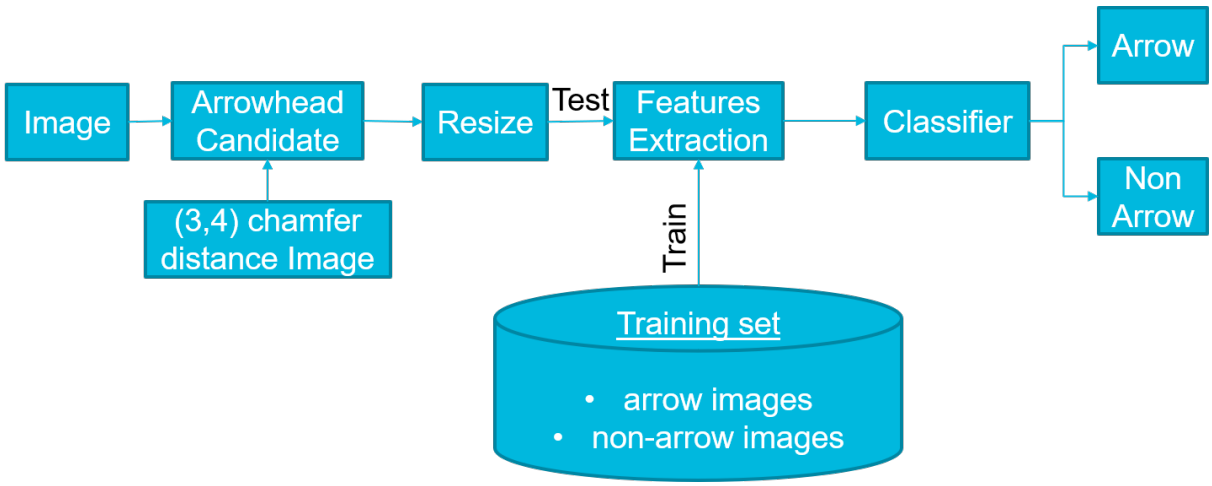


Figure 2.4.11: Arrow head detection algorithm

2.4.1 Bag of Visual Word

Bag of Visual word (BoVW) is a well-known method used in image classification. The idea behind it is similar to the idea of Bag of Words in natural language processing. The main concept of BoVW is to represent images as a set of features.

BoVW is a technique used to describe and classify images. This method extracts features from input images, generates visual words by quantizing features space using a clustering algorithm and extracting center points of the cluster, which are considered visual words that describe an image. After extracting the visual words, we train a classifier using extracted visual words to classify new test images.

To build a BoVW, we need to generate a dataset, select a features detector and descriptor and select a classifier to train extracted features. All these steps are explained in detail in the following sections.

2.4.1.1 Dataset Description

We generate a dataset that consists of two different groups: Arrow images and Non-Arrow images. It contains 3312 arrow images and 3341 non-arrow images (synthetic and real). The real samples are cropped from real images and classified manually, noting that the

real samples represent only about 10% of the dataset. However, the synthetic dataset is generated using a python script that generates arrow images and random line (Non-arrow images) images of random size between 16×16 to 64×64. The real and synthetic images are then resized to 128×128 using nearest-neighbor interpolation and binarized using OTSU algorithm. Figure 2.4.12 shows samples from the dataset.

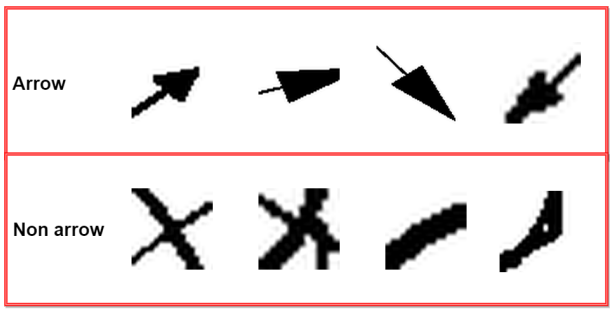


Figure 2.4.12: Arrow head dataset

2.4.1.2 Local Feature

After generating the dataset, the visual words are build by detecting features, extracting descriptors from images, making clusters from descriptors, using the center of each cluster as a visual dictionary vocabularies, and making frequency histogram from the vocabularies and the frequency of the vocabularies in the image. Thus, we use SIFT and KAZE as feature extractors, and descriptors based on their performance in literature reviews [76]. The two feature extraction algorithms are explained in details in the following paragraphs:

(i) Scale Invariant Feature Transform (SIFT) :
Lowe [77] presented the Scale Invariant Feature Transforms (SIFT) algorithm, one of the best feature extraction algorithm. SIFT shows its robustness in detecting features with different scales, rotation, and illumination. SIFT is made up of four main steps, as shown in Figure 2.4.13 and described in the following paragraphs.

- Scale-space: In this paragraph, we are going to describe the scale-space extrema detection. As shown in Figure 2.4.14, it is clear that we are not able to identify key points with a different scale. Thus, this stage search over all scales and image locations to detect scale and orientation invariant key points.

  SIFT method use the Laplacian of Gaussian function that acts as a blob detector. Thus, $L(x, y, \sigma)$ is the function that represents the scale space of an image, $L(x, y, \sigma)$ is written as follows:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{2.5}$$

  where $I(x, y)$ is the input image, (x, y) are the coordinates of the key point, σ is the scaling parameter, $*$ is the convolution operation in x and y, and $G(x, y, \sigma)$

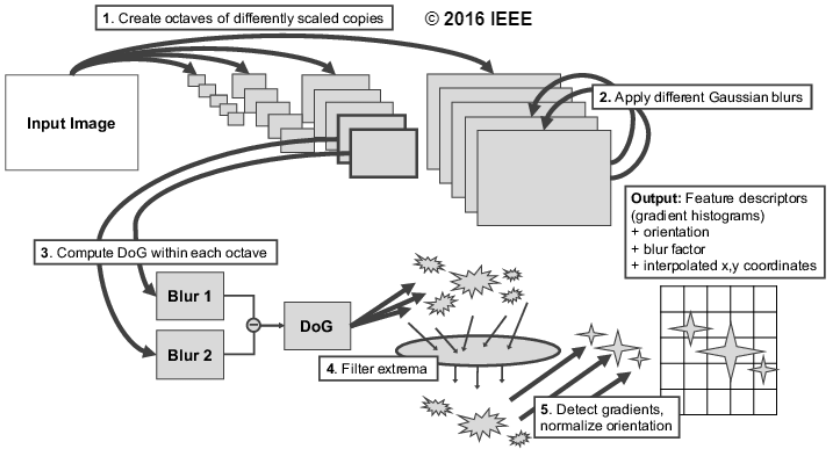


Figure 2.4.13: The four major stages of SIFT object recognition algorithm [78]

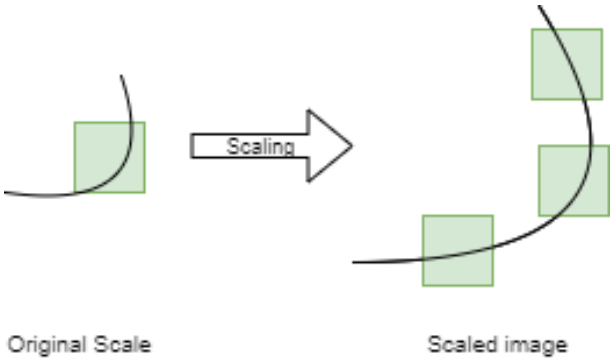


Figure 2.4.14: Corner detection

is the variable scale Gaussian kernel which is represented as follows:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{2.6}$$

However, Laplacian of Gaussian (LoG) is a costly function; *Lowe* [79] proposed to use an approximation for LoG which is the Difference of Gaussian (DoG), represented in Figure 2.4.15 and expressed as follows:

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{2.7}$$

where k is a constant multiple factor that separates two nearby scales.

According to *Lindeberg* [80] the difference-of-Gaussian function is a close approximation to the scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$. Referring to the heat diffusion equation we have:

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G \tag{2.8}$$

From the above equations, we obtain:

$$\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma} \tag{2.9}$$

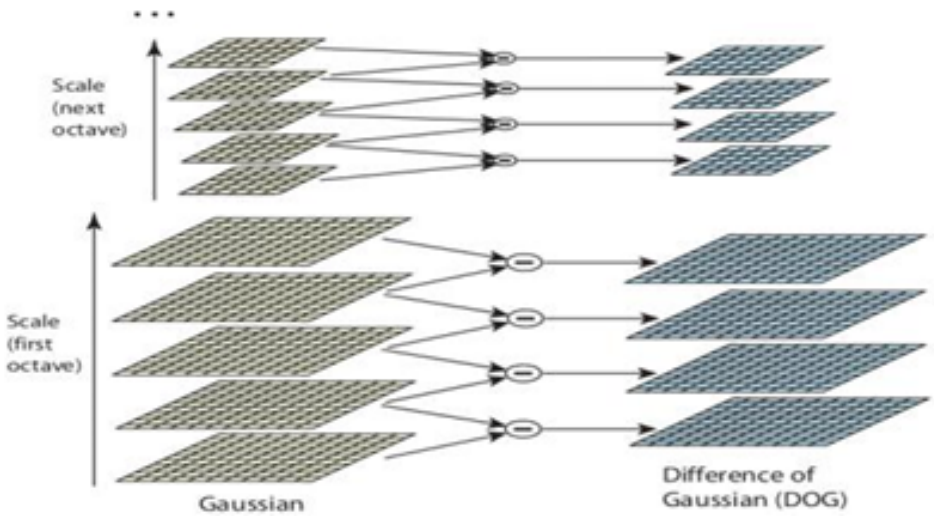


Figure 2.4.15: Detection of extrema of the DoG images [77]

This implies that:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx \sigma\nabla^2 G(k-1) \tag{2.10}$$

where $(k-1)$ in equation 2.10 will not be affected by extrema location because it is a constant over all scales.

The next step is to detect the local extrema. As shown in Figure 2.4.16, there are three images with three different scales. The SIFT algorithm selects a point

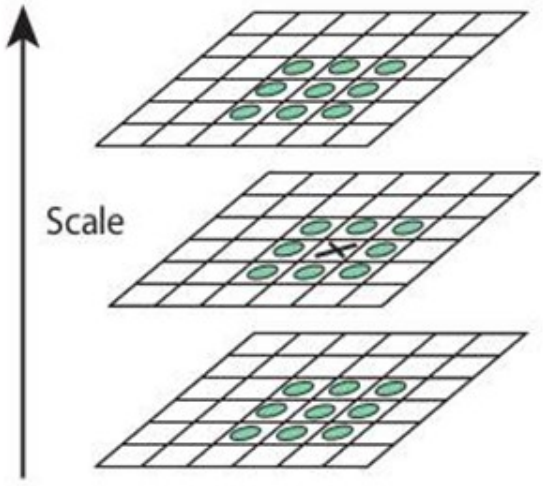


Figure 2.4.16: Detection of extrema of the DoG images [77]

(X) and looks at 3×3 neighborhoods at the same scale, 3×3 neighborhoods on the higher scale, and 3×3 neighborhoods on the lower scale. Thus, 26 neighborhoods (each scale 3×3 and 3 scale $\implies$ $3x3x3 = 27$ $\implies$ $26 + 1$ the

selected pixel and 26 neighborhoods) are obtained, and the 27 is the selected pixel. If this pixel is larger or smaller than all other 26 pixels, then the point (X) is considered a key-point.

- Key-point Localization: In this part, the algorithm localizes the best scale, location, and curvature for each key-point. Next, the algorithm rejects some key-points that lie along an edge or do not have sufficient contrast. Those points are detected by applying the Taylor series expansion of the scale-space function, $D(x, y, \sigma)$, shifted so that the origin is at the sample point

$$D(x) = D + \frac{\partial D^T}{\partial x}x + \frac{1}{2}x^T\frac{\partial^2 D}{\partial x^2}x \tag{2.11}$$

To find the extrema, the previous equation is differentiated with respect to x and set it to zero , where $x = (x, y, \sigma)^T$ is a sample point:

$$\hat{x} = -\frac{\partial^2 D^{-1}}{\partial x^2}\frac{\partial D}{\partial x} \tag{2.12}$$

If $D(\hat{x})$ is below a threshold, then this point is rejected. In addition, DoG has a higher response for edges. To get rid of those key-points, the Hessian matrix H of D is computed as follows:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{2.13}$$

then, the matrix is evaluated by calculating its trace and determinant:

$$\mathrm{Tr}(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{2.14}$$

$$\mathrm{Det}(H) = D_{xx}D_{yy} - \left(D_{xy}\right)^2 = \alpha\beta \tag{2.15}$$

where α is the largest eigenvalues and β is the smaller one.

After calculating the trace and the determinant for the matrix, the ratio of the squared trace $Tr(.)$ over the determinant $Det(.)$ should be less than a threshold r. Otherwise, the key-point is rejected:

$$\frac{Tr(H)^2}{\mathrm{Det}(H)} < \frac{(r+1)^2}{r} \tag{2.16}$$

where r is the ratio between the largest magnitude eigenvalue and the smaller one, so that $\alpha = r\beta$ (The experiments in this thesis use a value of $r = 10$)[77].

- Orientation Assignment: In this part, we talk about the orientation assignment, which makes the key-point invariant with the location. Using the scale of the key point to select the Gaussian smoothed image, the gradient magnitude

$m(x, y)$ and the orientation $\theta(x, y)$ for each image sample $L(x, y)$ are computed as follows:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (2.17)$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1))/(L(x+1, y) - L(x-1, y))) \quad (2.18)$$

After this step, each sample is weighted by its gradient magnitude and scale σ. As shown in Figure 2.4.17, the algorithm builds an orientation histogram in a neighborhood of the key-point, which has 36 bins covering 360 degree range of orientation (each bin cover 10 degree).

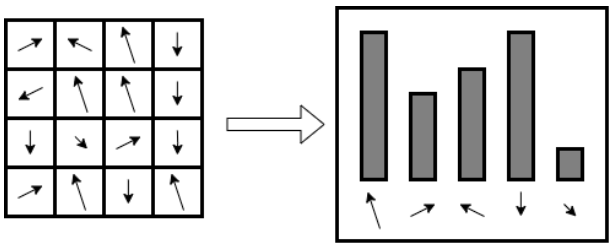


Figure 2.4.17: Gradient histogram

After Building the histogram, SIFT selects the highest peak and take into consideration all other local peaks that are within 80% of the highest one. Each peak creates a key-point with the same location and scale but different directions.

- Key-point Descriptor: In the final step, the keypoint descriptor shown in Figure 2.4.18 is generated. The local descriptor is obtained by sampling the magnitudes and the orientations of the image gradients in a neighborhood of each key point, then by building the smoothed orientation histograms that contain the important aspect of the neighborhood. A local descriptor is made up of a 4×4 array (histogram). An eight-orientation vector is associated for each coordinate of this array. A 128-elements $8\times(4\times4)$ vector is then built for each key point. In other words, each image im is described by a set of invariant features $X(im) = \{k_i = (s_i, sc_i, x_i, y_i) | i = 1 : N(im)\}$ where s_i is the 128-elements SIFT invariant descriptor computed near the key point k_i, (x_i, y_i) is the position of k_i in the original image im, sc_i is the scale, and $N(im)$ is the number of detected key points in image im. The extracted features are invariant to image scaling and rotation and partially invariant to change in illumination and 3D camera view point.

(ii) KAZE:

KAZE approach is quite similar to the SIFT method. However, the difference is

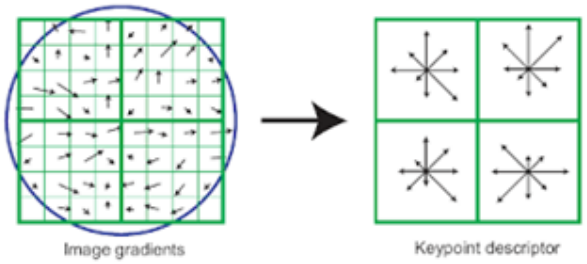


Figure 2.4.18: Key-point descriptor [77]

in the approach of building the scale-space where KAZE uses a nonlinear diffusion filtering operation instead of the Gaussian approach. Unlike the Gaussian approach, which blurs both noise and details and may lead to lose boundaries, the nonlinear diffusion filtering method preserves details (blur noises only) by locally blurring (adaptive blurring) the image data. The KAZE method is made up of four main steps described in the following paragraphs.

- Scale-space: KAZE uses a scheme based on Additive Operator Splitting (AOS) techniques to build the nonlinear scale space, this scheme is totally stable for any step size.
  The classical nonlinear diffusion is expressed as follows :

$$\frac{\partial X}{\partial t} = \mathrm{div}(c(x, y, t) \cdot \nabla X) \tag{2.19}$$

  where X represent an image with spatial coordinates (x, y), the time t is the scale parameter, $div(.)$ is the divergence, ∇ is the gradient operators and c is the conductivity function which controls the diffusion to the image structure. To preserve edges and blur the inside regions, *Perona and Malik* [81] propose to make the function c dependent on the gradient magnitude, where c can be represented as follows:

$$c(x, y, t) = g\left(|\nabla X_\sigma(x, y, t)|\right) \tag{2.20}$$

  where ∇X_σ represents the gradient of a Gaussian smoothing of version of the original image X. To promote wide region, the selected conductivity function g is written as follow:

$$g = \frac{1}{1 + \frac{|\nabla X_\sigma|^2}{k^2}} \tag{2.21}$$

  where the contrast factor k decides which have to be enhanced and which have to be canceled. This algorithm obtains an empirical value for k as the 70% percentile of the gradient histogram of a smoothed version of the original image.
  Using a vector-matrix notation, the discretization of Equation 2.19 can be

written as follows:

$$\frac{X^{i+1} - X^i}{\tau} = \sum_{l=1}^{m} A_l \left(X^i\right) X^{i+1} \tag{2.22}$$

where A_l is a matrix that encodes the image conductivities for each dimension. Thomas algorithm [82] is used to solve the tridiagonal linear system for computing X^{i+1} from X^i in the equation 2.22

The method discretizes the scale space in logarithmic steps that are arranged in a series of O octaves and S sub-levels. The set of octaves and sub-levels are identified by a discrete octave index o and a sub-level one s. The octave and the sub-level indexes are mapped to their corresponding scale σ through the following formula:

$$\sigma_i(o, s) = \sigma_0 2^{o+s/S}, o \in [0 \dots O-1], s \in [0 \dots S-1], i \in [0 \dots N] \tag{2.23}$$

where the base scale level is denoted by σ_0 and the total number of filtered images is denoted by N. Unlike SIFT, KAZE does not perform any down-sampling at each new octave, it always works with the original image. As nonlinear diffusion filtering is defined in time terms, the scale units σ_i should be converted to time units t_i which is expressed as follow:

$$t_i = \frac{1}{2}\sigma_i^2, i = \{0 \dots N\} \tag{2.24}$$

- Key-point detection: After building the scale space, the algorithm detects key-points by computing the determinant of the Hessian at multiple scale levels which is represented as follows:

$$X^i_{\text{Hessian}} = \sigma^2 \left(X_{xx} X_{yy} - X_{xy}^2\right) \tag{2.25}$$

where the second order horizontal, vertical and cross derivatives are respectively denoted by X_{xx}, X_{yy} and X_{xy}. the 3×3 Scharr filters approximate the set of first and second order derivatives; This filter outperform other popular filter in terms of rotation invariance.

- Orientation Assignment: KAZE algorithm approximates the dominant orientation in a local neighborhood centered at the keypoint location to retrieve rotation invariant descriptors. To detect the dominant orientation, KAZE computes the first-order derivatives X_x and X_y in a circular neighbored of radius $6\sigma_i$ where σ_i is the scale of the keypoint. Next, the first order derivatives of each sample in the circular area are weighted with a Gaussian centered at the key point. The derivatives responses are represented as points in vector space. The dominant orientation is obtained by summing the derivatives within a sliding circle window covering an angle of $\frac{\pi}{3}$ and select the longest vector.

- Keypoint Descriptor :After estimating each keypoint orientation, the KAZE algorithm constructs their descriptors using the M–SURF descriptor adapted to its nonlinear scale space. The first order derivatives X_x and X_y for each key point are computed over a $24\sigma_i \times 24\sigma_i$ rectangular region. This region is divided into 4×4 subregions of size $9\sigma_i \times 9\sigma_i$ with an overlap of $2\sigma_i$. In each subregion, the derivatives responses are weighted with a Gaussian ($\sigma = 2.5\sigma_i$ refer to [83]) centered on the subregion center and summed into a descriptor vector. Next, each subregion vector is weighted using another Gaussian ($\sigma = 1.5\sigma_i$ refer to [83]) defined over a 4×4 mask and centered on the key point. To determine the dominant orientation of the keypoint, the algorithm takes into consideration both the computation of derivatives and samples of the $24\sigma_i \times 24\sigma_i$ region. Lastly, the algorithm normalizes the descriptor vector of length 64 into a unit vector to achieve invariance to contrast.

2.4.1.3 Classifiers

(i) Decision Tree:

Decision Tree (DT) is a very effective classification algorithm that aim to divide the dataset into homogeneous groups to make the prediction easier. To understand how the algorithm works, Suppose we have a training set T and a set of classes C1, C2,. . . ., Cn. The algorithm will perform the decision according the content of T. First, concerning that T contains samples that belong to the same class Ck , the decision tree for this training set will be a leaf categorizing class Ck. Conversely, if T contains samples that belongs to different classes, T must be refined into subsets that are aiming to be single class-groups of samples. A test is chosen, based on single attribute, that exclusive outcomes O1, O2, ... On. T will be divided it into subsets T1,T2. . . ..Tn and for each Ti , an outcome Oi of the chosen test will be obtained. T will be a node classifying the test and one branch for each possible outcome.

(ii) Multi-Layer Perceptron:

Multi-Layer Perceptron (MLP) is one of the most efficient machine learning methods currently known; it also shows high classification rates in previous work. Perceptrons are made to solve separable datasets linearly. Thus, to classify nonlinear datasets, *Rumelhart et al.* [84] developed MLP. MLP classifier classifies instances based on the back propagation algorithm, which is a network of simple units that produce a complex output by working together. The back-propagation algorithm train multi-layer feed-forward neural network. MLP consists of units that are connected using weighted connections. We have three different types of units, as shown in Figure 2.4.19, which are the input units, the hidden units, and the output units.

The input units are weighted and fed simultaneously to the hidden layer. The outputs of the hidden layer can be inputs for another hidden layer. After reaching the output layer, the back propagation will calculate the amount of error in the predicted output compared to the actual output and change the weights according to this amount. (see section 1.2.1)

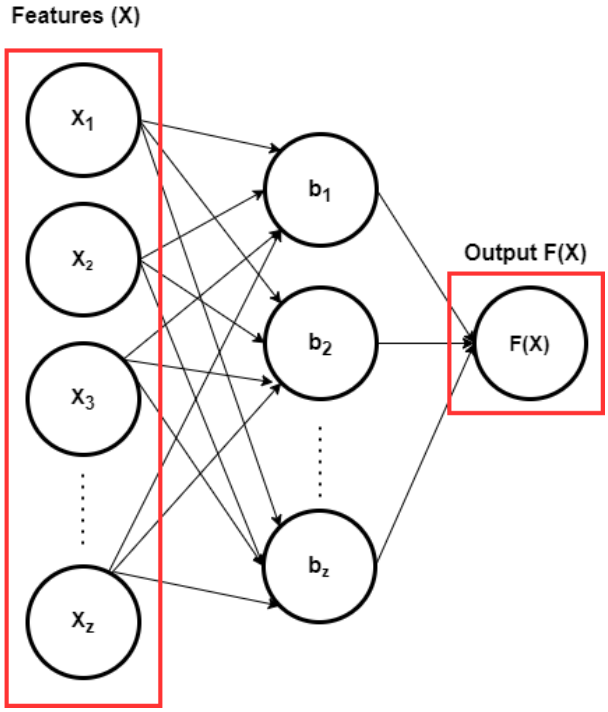


Figure 2.4.19: MLP architecture

(iii) Support Vector Machine:
Support Vector Machine (SVM) has been proposed by *Vapnik* [85] and is based on the structural risk minimization principle from statistical learning theory. SVM expresses predictions in terms of a linear combination of kernel functions centered on a subset of the training data, known as support vectors (SV).

Suppose we have a training set x_i, y_i where x_i is the training pattern and y_i the label. For problems with two classes, with the classes $y_i \in \{-1, 1\}$, a support vector machine algorithm works as follows. First, the training points x_i are projected into a space H (of possibly infinite dimension) by means of a function $\phi(.)$. The second step is to find an optimal decision hyperplane in this space. The criterion for optimality will be defined shortly. Note that different transformations $\phi(.)$ may lead to different decision functions for the same training set. A transformation is achieved in an implicit manner using a kernel $K(.,.)$ which can have different forms such as:

- The polynomial kernel where

$$K(x, y) = (\gamma < x, y >)^p \tag{2.26}$$

where p is represent the degree of the polynomial function and γ is the kernel coefficient.

- The sigamoid function kernel where:

$$K(x, y) = \tanh(\gamma\langle x, y\rangle + r) \tag{2.27}$$

where γ is the kernel coefficient and $\gamma > 0$

- The linear kernel where:

$$K(x, y) =< x, y > \tag{2.28}$$

This later kernel is being devoted to linearly separable, while the previous two are designed for non separable cases (as seen below). Consequently the decision function can be defined as:

$$f(x) = \langle w, \phi(x)\rangle + b = \sum_{i=1}^{l} \alpha_i^* y_i K\left(x_i, x\right) + b \tag{2.29}$$

where $\alpha_i^* \in \mathcal{R}$ The values w and b are the parameters defining the linear decision hyperplane. In SVMs, the optimality criterion to maximize the margin is the distance between the hyperplane and the nearest point $\phi(x_i)$ of the training set. The α_i^* which optimizes this criterion are obtained by solving the following problem:

$$\begin{cases} \max_{\alpha_i} \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{ij=1}^{l} \alpha_i \alpha_j \gamma_i K\left(x_i, x_j \gamma_j\right) \\ \text{with constraints} \\ 0 \leq \alpha_i \leq C \\ \sum_{i=1}^{l} \alpha_i \gamma_i = 0 \end{cases} \tag{2.30}$$

where C is a penalization coefficient for data points located in or beyond the margin and provides a compromise between their numbers and the width of the margin.

2.4.2 Cross Validation

Cross-validation [86] is an algorithm used to evaluate a classifier model using a limited data sample to prevent the over-fitting problem. The k-folds Cross-validation algorithm divides the N instances of the data set into k equal or approximately equal-sized subsets or folds. Next, the classification model is trained using $k-1$ folds, tested using one fold (kth), and the model performance measures are stored. However, this process is repeated k times by using in each iteration a different fold for testing and the remaining $k-1$ folds for training the classification model. Once the algorithm completes all iterations, the overall performance measures are computed by the stored measures at each iteration. The cross-validation algorithm is described in Figure 2.4.20.

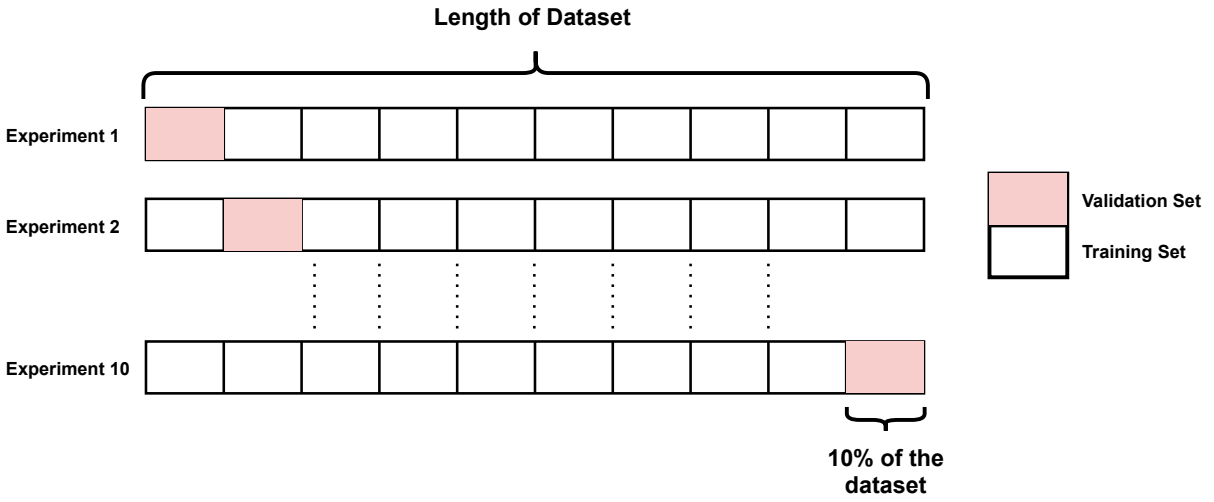


Figure 2.4.20: Cross validation

2.4.3 Confusion Matrix

After applying the classification (arrowhead or not in our case), the confusion matrix is used to measure the performance of the classifier. The confusion matrix is a matrix with four different combinations of predicted and actual values, as shown in Figure 2.4.21; these four values are as follows:

- The True Positive (TP) which is the number of correctly classified positive instances
- The False Positive (FP) which is the number of negative instances that are incorrectly classified as positive instances
- The False Negative (FN) which is the number of positive instances that are incorrectly classified as negative instances
- The True Negative (TN) which is the number of correctly classified negative instances

Those values are extremely useful for calculating classification performance measures such as precision, recall, and f1-measure. The precision is the ratio of correctly predicted instances observations to the total predicted positive instances. The precision reflects the ability of the classifier not to label as positive an instance that is negative. The equation of precision is written as follows:

$$Precision = \frac{TP}{(TP + FP)} \tag{2.31}$$

The Recall is the ratio of correctly predicted positive instances to all instances in the same class. The Recall reflects the ability of a classification model to the classifier ability to find all the positive instances. The equation of Recall is written as follows:

$$Recall = \frac{TP}{(TP + FN)} \tag{2.32}$$

The F1-score is the harmonic mean of recall and precision; it reflects a classification model overall performance. The equation of the F1 score is written as follows:

$$F1 = 2 \times \frac{Recall \times Precision}{(Recall + Precision)} \tag{2.33}$$

	Predicted Positive	Predicted Negative
Actual True	True Positive	False Negative
Actual False	False Positive	True Negative

Figure 2.4.21: Confusion matrix

2.4.4 Discussion

In this section, we discuss the results of the proposed method. All the results in this section are obtained using the dataset and 10-cross validation method. Table 2.4.1 shows the confusion matrix of SIFT and KAZE features with the five classifiers. The receiver operating characteristic curve (ROC) is a popular measure for evaluating classifier performance. The ROC curve is used to show the trade-off between sensitivity and specificity. Moreover, the ROC curve is used to calculate the Area Under the ROC Curve (AUC), which reflects the performance of the model in discriminating between arrowheads and non arrowheads. Thus, an AUC of 0.5 suggests no discrimination, whereas an auc greater than 0.9 is considered excellent and shows the excellent performance of the model in classifying arrowheads and non arrowheads. Moreover,figure 2.4.22a represents the ROC curves of SIFT features with all classifiers, and figure 2.4.22b represents ROC curves of KAZE features with all classifiers. Based on the confusion matrix, it is clear that KAZE + MLP model is the best combination since it has the smallest number of False-negative and the smallest number of False-positive instances. Thus, we calculate F1,precision and recall of this model.

$$Recall_{KAZE+MLP} = 0.99 \qquad Precision_{KAZE+MLP} = 0.99 \qquad F1_{KAZE+MLP} = 0.99$$

The calculated metrics show that the classification model is very accurate.

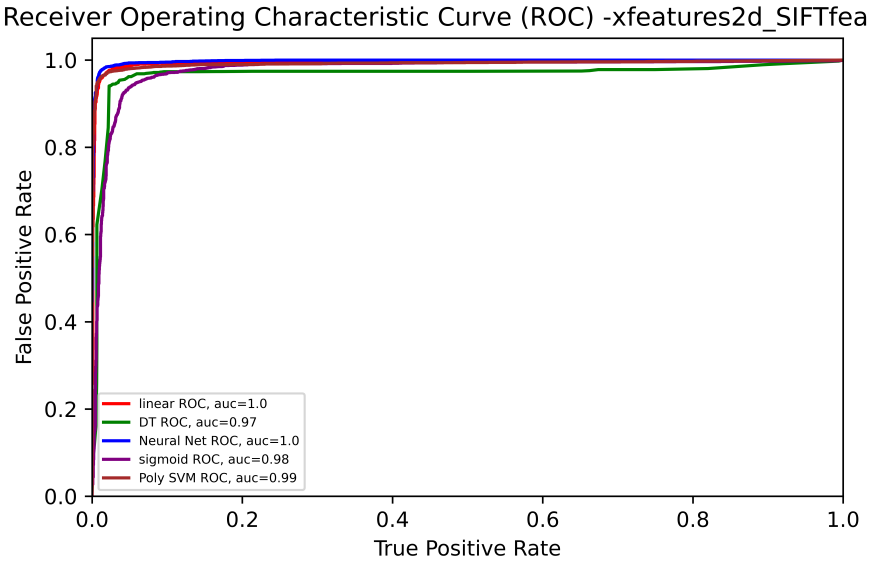
KAZE outperforms SIFT, as shown in the confusion matrix; the main reason is the ability of KAZE to detect more features which is critical because arrow images do not have a lot of details. Moreover, SIFT finds features in the gaussian scale space, and gaussian blurring does not respect the natural boundaries of the object. Unlike SIFT, KAZE finds features in a nonlinear scale-space that keep important details. As the boundaries in our case are significant, we can understand why KAZE outperforms SIFT (see Fig.2 in [83]).

Next, we test the selected model on different numbers of images from different sources and with different sizes. The model could detect arrows correctly in most images except in image (a), where the model classifies some non-arrows head images as arrowheads.

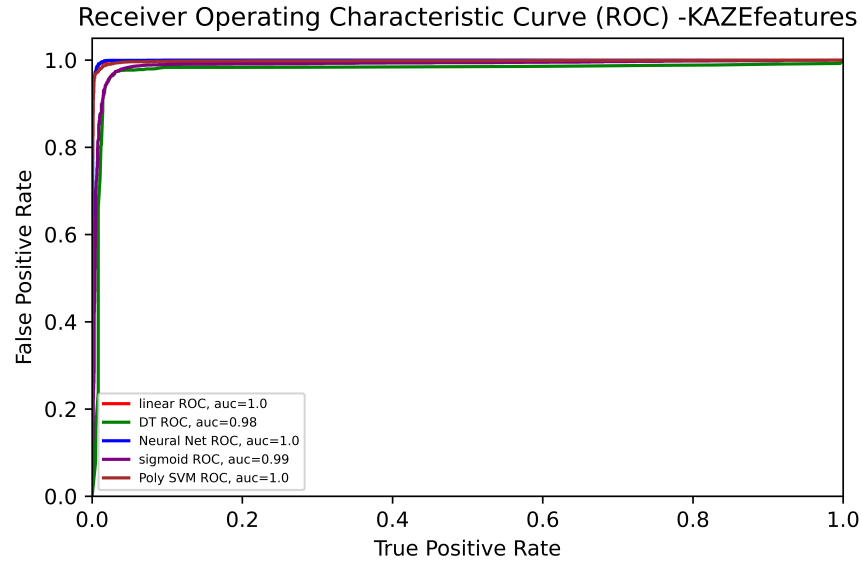
The proposed method is robust to noises and has the ability to detect arrowheads with various rotations. However, the main drawback of this method is the inability to detect arrows with different shapes.

		Linear SVM		**Sigmoid SVM**		**Poly SVM**		**MLP**		**DT**	
		A	*N*	*A*	*N*	*A*	*N*	*A*	*N*	*A*	*N*
SIFT	*A*	3252	60	3077	235	3266	46	3262	50	3146	166
	N	70	3267	220	3117	122	3215	84	3253	170	3167
KAZE	*A*	3289	23	3237	75	3280	32	3292	20	3243	69
	N	35	3302	74	3263	63	3274	30	3307	115	3222

Table 2.4.1: Confusion matrix of different models

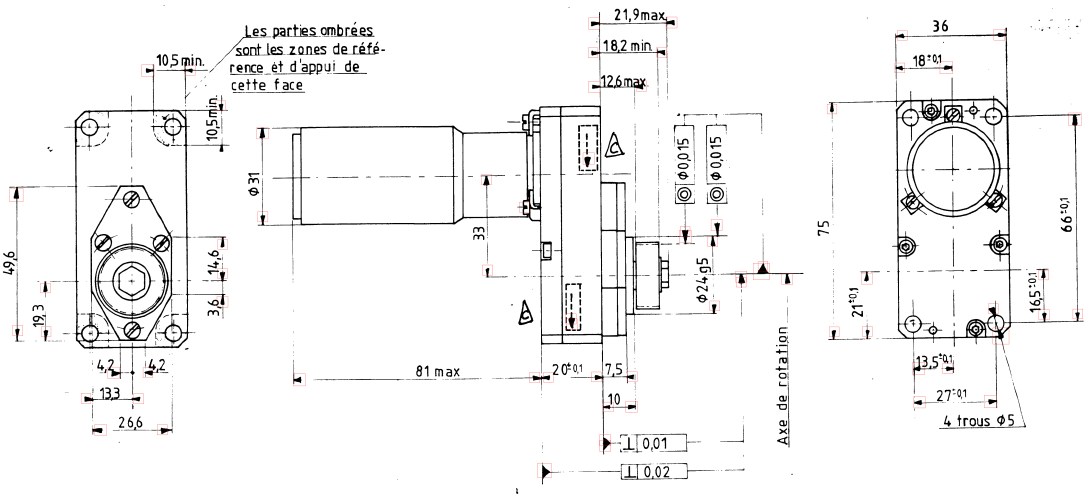


(a) ROC - SIFT

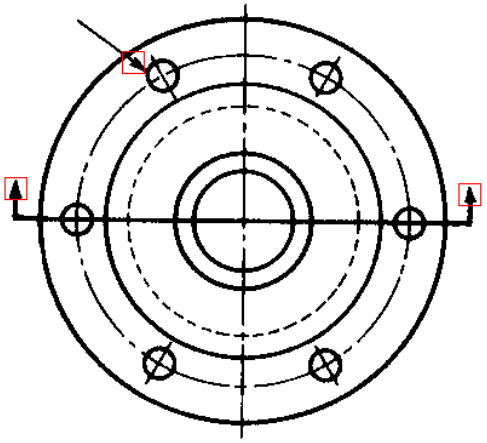


(b) ROC - KAZE

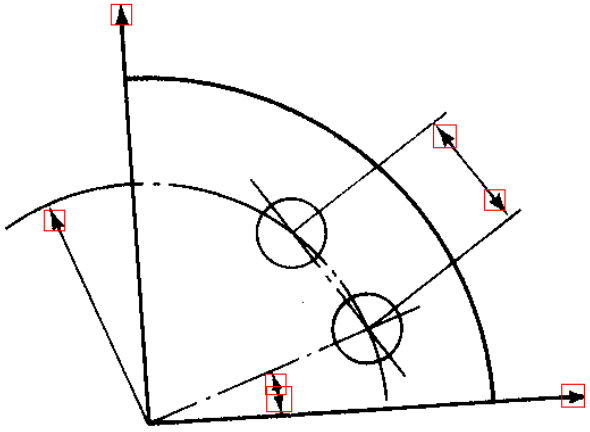
Figure 2.4.22: ROC curves



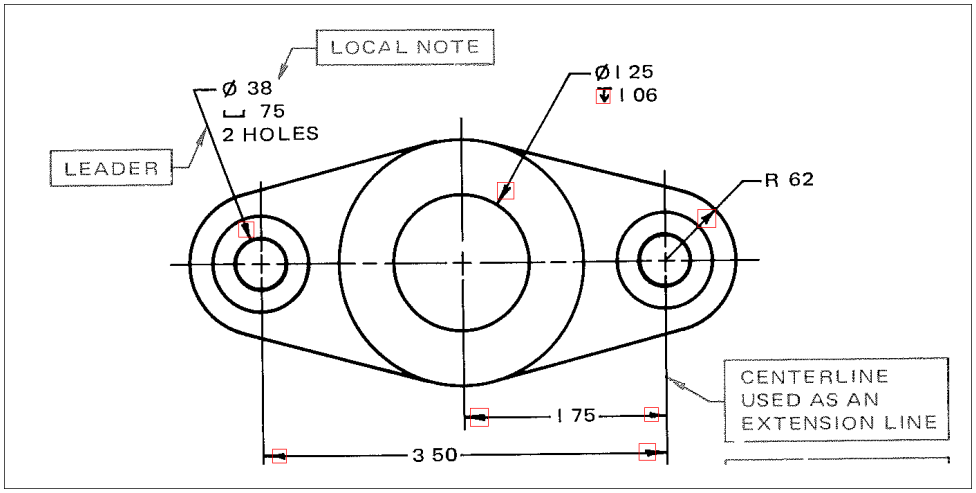
(a)



(b)



(c)



(d)

Figure 2.4.23: Arrow heads detection

2.5 Summary

In this chapter, we proposed a method to prepare engineering drawings for vectorization. The proposed algorithm:

1. Separates drawings with borders and drawings without borders.
2. Removes borders when needed and then clusters the views to separate them.
3. Cleans each view using the trained Unet network.
4. Skeletonizes each view using 3,4 chamfer distance.
5. Detects arrowheads in each view (This arrowheads detection algorithm facilitates the extraction of dimension sets. However, due to the limited time, we did not implement the extraction step.).

Thus, the input of the next step is described either from an unsupervised point of view (chapter 3) or hybrid point of view (chapter 4) is a binary image with its skeleton. The vectorization step needed for 3D reconstruction aims to identify segments of drawing with numerical features.

The advantage of the proposed algorithm is the high ability to work with different types of engineering drawings. Nevertheless, the clustering process might cluster some noisy data, which will be considered as a view. Thus, the proposed algorithm needs more process to separate noisy data from correct views and then label the views (top, side, bottom ...). Moreover, the algorithm of detecting arrowheads can be completed by detecting the arrowhead direction to be able to detect the dimension sets.

All the steps are optional for the following chapters except the skeletonization stage. The input of the proposed method in this chapter is a raster drawing with a full template, and the output is a set of views, cleaned and skeletonized. Moreover, for each extracted view, we have the location of arrowheads.

Chapter 3

Unsupervised Vectorization

Contents

3.1 Introduction

In the past thirty years, and with the increased usage of 2D CAD Softwares, the interest in converting paper-based technical drawing to vector drawing starts rising. Due to the profuse number of engineering drawings exits without the associated CAD data, in addition to the copious number of hand made drafting scanned and saved as raster drawing. Compared to raster images, vector representations are scale-independent, much more compact, and, most importantly, support easy primitive-level editing. Besides the importance of vector representation, the converted data can be used to update old drawings easily, make new drawings derived from old ones, or even convert old drawings into 3D models to use them in VR/AR applications.

Conversion of paper-based drawing to vector-based drawing is a computer vision problem that aims to convert pixels into meaningful 2D data. This problem need to propose

automated algorithms that have the ability to understand and convert the raster drawing to 2D vector drawing.

Different researcher addressed this topic due to its importance. Several kinds of images such as cartoon images [87], hand drawings [88, 89], natural images [90], maps [91] and characters [92] can be vectorized. Hence, our focus will be on vectorization of technical drawings such as mechanical and electrical engineering drawings.

Yuan Chen et al. [93] developed P-RENDER (post process of RENDER[94]), which is a vectorization system showing interesting results. However, this method is sensitive to noise and variation of resolution. Additionally, the inability to recreates all arcs is another weakness of this approach.

Orthogonal zig-zag (OZZ)[95] is a method proposed by Dov Dori and inspired by a light beam conducted by optic fiber. This technique outperforms the Hough transform method in terms of execution time and vectorization results. However, this method does not detect arcs, this being a heavy drawback when dealing with engineering drawing.

In [96], a sparse pixel vectorization algortithm (SPV) inspired from OZZ is proposed. Instead of visiting all points (like OZZ), SPV visit only points of medial axis. This method detects only lines, however arcs are detected as polylines.

One of the most interesting and robust algorithms was proposed by *Hilaire et Tombre* [65] and is one of the most accurate and robust algorithms that uses 3-4 chamfer distance for extracting skeleton and maintaining all geometrical features. By using random sampling and fuzzy primitives, the skeleton is segmented. This method is accurate and robust to noise; however, this method can fragment primitives (such as lines, arcs and circles) due to the followed strategy, which assumes that a primitive starts and ends between two junction pixels and then unify primitives. In addition, this method needs to pre-define primitive parameters which usually change from one image to another.

Mandal et al. [97] vectorize horizontal and vertical line segments by using mathematical morphological tools. Also, some digital geometric properties of straightness are used to vectorize inclined and curve lines.

Recently, *Parmita De et al.* [98] proposed a method called ASKME, which also uses 3-4 chamfer distance to skeletonize. *Parmita De et al.* proposed to segment primitives by labeling connected components after deleting all junctions. An adaptive sampling method is used to detect and classify primitives based on mechanical domain knowledge. This method has several notable features, but still, *Hilaire and Tombre* method is more accurate.

A method [99] based on ternary segmentation and soft computing is proposed. This method shows good results however it requires a gray scale image (because it works based on gray scale values) which not always available.

These systems yield good results, but they rely on some thresholds and specific res-

olutions which lead to fragment primitives. Obviously, the accuracy of the vectorization step strongly impacts the quality of the 3-D reconstruction; This motivate us to propose a robust vectorization method which avoids primitives fragmentation and extracts accurate primitives.

In this chapter, we propose an unsupervised vectorization method due to the lack of labeled data and the intensive resources needed for labeling data. Thus in this chapter, we proposed a vectorization algorithm based on the genetic algorithm. The proposed algorithm uses the skeleton image generated in chapter 2, segmenting it based on junction points and vectorizing it based on the genetic algorithm approach. The current method detects only lines, circles, and arcs. However, it can be extended to identify more primitives. The chapter presents the proposed method in detail, then compares the proposed method with previous works and evaluates it based on hyperparameters tuning.

3.2 Proposed Method

The proposed vectorization method is based on the genetic algorithm, which is a heuristic search method inspired by evolutionary biology and is widely used in image processing and pattern recognition fields see [100, 101].

The genetic algorithm generates multiple primitives that represent a set of pixels and selects the highest ranked solution based on a fitness function. Thus, the genetic algorithm increases the accuracy by selecting the best primitives representing the drawing. The input of this stage is labeled pixels and the output is a vectorized image (in format .dxf, .svg and .vec).

As seen below, our algorithm aims to classify the connected components into a set of predefined primitives,these primitives having a certain degree of freedom. The pseudo-code of the main procedure is available in Algorithm 3 where NC, NG are respectively the Number of Chromosomes and the Number of Generations. In addition, $thresh$ is the minimum number of pixels possible to generate a circle or an arc which is equal to 3.

After the prepossessing stage (detailed in previous chapter), we have a labeled skeleton (each pixel in the skeleton is labeled as: endpoint or core point or junction point – refer to section 2.3.5) image called CCI that contains one or more connected components (in a 8-neighborhood meaning) as shown figure 3.2.2a. Assume that I_1 is a connected component that belong to CCI (see figure 3.2.2b). The following discussion is applied for each I that belong to CCI.

We start by generating a static copy of I_1 called I'_1 which does not change when I_1 is updated, the junction pixels are extracted from I_1, then I_1 is a set of connected components (see figure 3.2.2d) such as $I = \{C_1, C_2, ..., C_k\}$. In particular, C_3 is connected component since we are in a 8 neighborhood framework. The algorithm selects the longest connected component LC in I_1 (for example $LC = C_3$ in figure 3.2.2d) such that $LC = \{p_1, p_2, ...p_n\}$ is a set of connected pixels P_i and LC is used to generate the population. This strategy

Algorithm 3: Main algorithm

```
CCI = {I_1, I_2, .., I_n};
foreach I ∈ CCI do
    I' = I.copy();
    RemoveJunction(I);
    I = {C_1, C_2, .., C_n};
    LC = longest(C ∈ I);
    while |I| > Thresh and |LC| > Thresh do
        w = Widthestimation(LC);
        α = ⌈w/2⌉ + 1;
        Population = ChromoGeneration(LC, NC);
        solution.append(Fitness(Population));
        for Kinrange(NG) do
            NewPopulation = Crossover(Population);
            solution.append(Fitness(NewPopulation));
        end
        solution = [solution ≥ 80%];
        solution = sort(solution, Fitness);
        Bestsolution = max(solution);
        Vector.append(Bestsolution);
        I = Update(I);
        if |I| > Thresh then
            LC = longest(CC ∈ I);
            if |LC| > Thresh then
                contine
            else
                break
            end
        else
            break
        end
    end
end
Result: Vector;
```

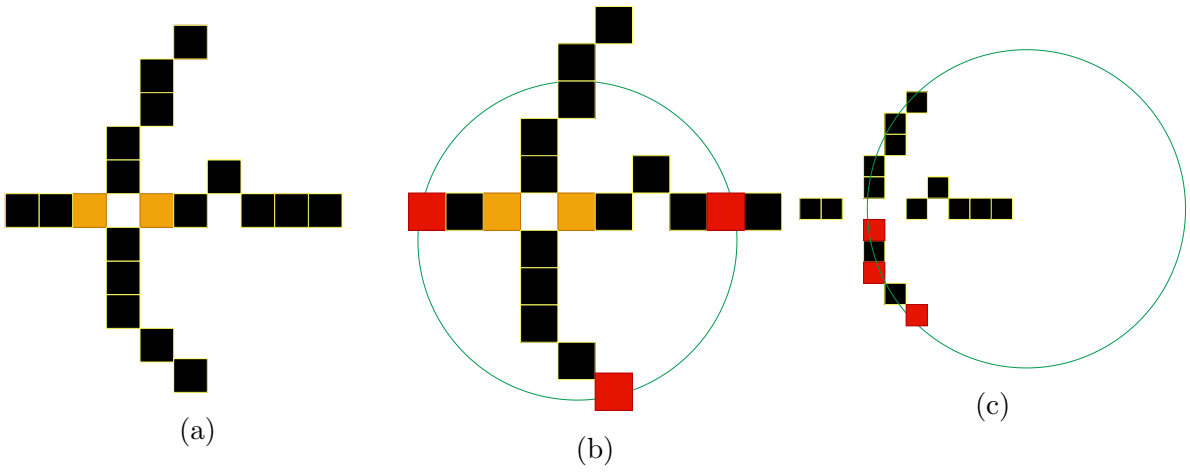


Figure 3.2.1: Example of the proposed decomposition strategy

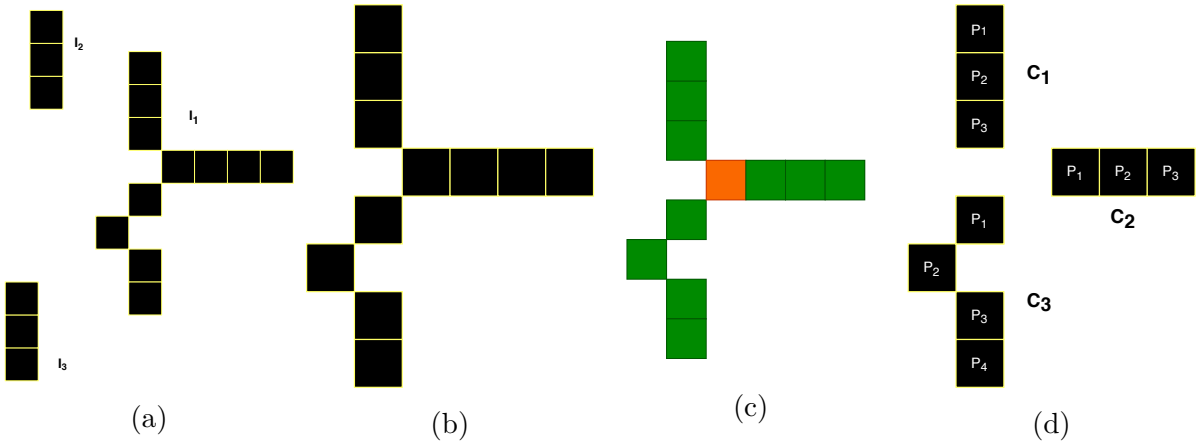


Figure 3.2.2: (a) Connected component in skeleton image, (b) One of the connected component I_1, (c) Labeled connected component I_1, (d) Set of connect component generated from I_1 after removing junction pixel

of decomposing I_1 and generating the population using LC decreases the probability of selecting a set of pixels that does not represent any primitives. Figure 3.2.1 illustrates the importance of the followed strategy, assume that figure 3.2.1a represents a connected component I belonging to LC. It is obvious that without decomposing I (see figure 3.2.1b), we have more possibilities to generate useless primitives such as the green circle in figure 3.2.1b. However, when we follow the decomposition strategy I and select chromosomes from one branch, we have more possibilities to generate useful and realistic primitives such as the green circle in figure 3.2.1c.

3.2.1 Population Generation

The population is a set of chromosomes where each chromosome represents a primitive (set of two pixels representing a line and set of three pixels representing a circle). The

Algorithm 4: Width estimation

$N_8(p)$ = 8 neighbor pixels of p;
$V_{3,4}(p)$ = value of pixel p in (3,4)-DT;
foreach *pixel* $p \in LC$ **do**
  K=[ ];
  foreach *pixel* $q \in N_8(p)$ **do**
    if $V_{3,4}(q) \geq V_{3,4}(p)$ **then**
      K.append(q);
    end
  end
  if $|K| \geq 3$ **then**
    T.append($V_{3,4}(p) + \frac{3}{2}$);
  else
    T.append($V_{3,4}(p)$);
  end
end
Result: $width = \frac{2}{3} \times Median(T) - 1$;

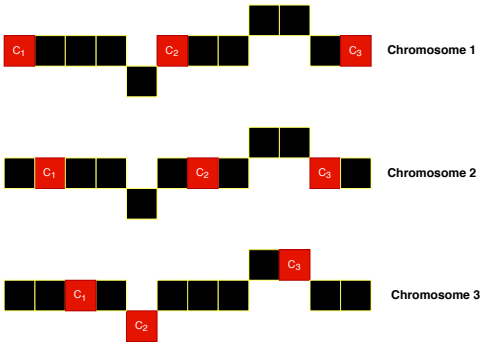


Figure 3.2.3: Population generation

algorithm generates the population using pixels of LC; the chromosomes are chosen following a down-sampling (among the pixels of the connected component) selection strategy (see figure 3.2.3) such that:

$$Population = \{(p_1, p_n), (p_1, p_{\lceil \frac{n}{2} \rceil}, p_n), (p_2, p_{n-1}), (p_2, p_{\lceil \frac{n}{2} \rceil + 1}, p_{n-1}), (p_3, p_{n-2}), (p_3, p_{\lceil \frac{n}{2} \rceil - 1}, p_{n-2}), ...\}$$

where n is the number of pixels knowing that those pixels are sorted clockwise (see Figure 3.2.3 where n=12). As detailed below in our approach chromosomes are pairs or triplets of pixels, since the searched primitives are lines or circles.

3.2.2 Fitness Function

The fitness function aims to select the best chromosome representing LC. The width w of LC is calculated using algorithm 4 proposed by [65] and α is the tolerance that defines the inliers and outliers pixels for each chromosome where $\alpha = \lceil \frac{w}{2} \rceil + 1$ ($\lceil x \rceil$ maps x to the least integer greater than or equal to x). Two different strategies are available to calculate the fitness function:

1. If the chromosome is a set of two pixels, then the line equation $y = mx + b$ that passing through those pixels is computed (Different pixels combination can represent the same equation, thus each equation is computed only once). All pixels in I'_1 located between $y = mx + b + \alpha$ and $y = mx + b - \alpha$ are nominated for the next step (see figure 3.2.5).

2. If the chromosome is a set of three pixels, then the circle equation $(x-x_c)^2+(y-y_0)^2 = r^2$ that passing through those pixel is computed (Different pixels combination can represent the same equation, thus each equation is computed only once). All pixels in I'_1 that are located between $(x-x_c)^2+(y-y_0)^2 = (r+\alpha)^2$ and $(x-x_c)^2+(y-y_0)^2 = (r-\alpha)^2$ are nominated for the next step (see figure 3.2.5).

As stated in introduction, the dictionary of the features can be enriched with other parametric, thus showing that our approach is very generic. The nominated pixel list is written as follow:

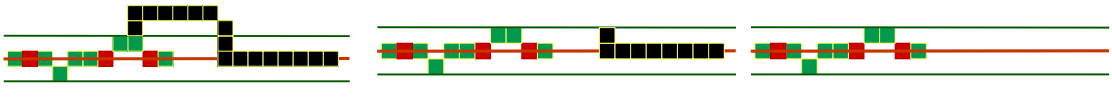
$$\text{NominatedList} = \begin{cases} \forall (x,y) \in I', b - \alpha \leq y - mx \leq b + \alpha, & \text{if } |chromo| = 2 \\ \forall (x,y) \in I', (r-\alpha)^2 \leq (x-x_c)^2 + (y-y_c)^2 \leq (r+\alpha)^2, & \text{if } |chromo| = 3 \end{cases} \tag{3.1}$$

We compute the connected components of pixels that belong to $Nominatedlist$ called $NominatedCC$ such that $NominatedCC = \{ CN_1, \ldots, CN_n \}$. In fact, disjoint connected component can verify equation (3.1). The accepted connected component called $AcceptList$ is that having the maximum intersection with LC. Thus, the $AcceptList$ should respect the following equation:

$$AcceptedList = \max\{|CN_1 \cap LC|, \ldots, |CN_n \cap LC| : n = |Nominatedlist| \; and \; |CN \cap LC| \neq 0\} \tag{3.2}$$

In figure 3.2.4, we depicted an example that shows the importance of the previous step. Figure 3.2.4a shows that the $nominatedList$ is a set of two connected components (CN_n where n=2). If the connected component with an empty intersection with LC is selected, the code will update I_1,re-select the same LC, generate the same solution and keep looping infinitely.

We compute the endpoints of each component that represents a circle or an arc in the $AcceptedList$, which can be rejected based on the equation 3.3. However, all the components representing a line in $AcceptedList$ are always accepted.



(a) (b) (c)

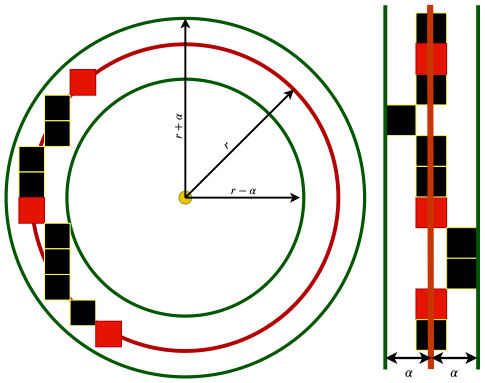
Figure 3.2.4: (a) General image view with nominated and non nominated pixel, (b) Pixels in *NominatedList*, (c) Pixels in *AcceptedList*

$$\text{AcceptedList} \begin{cases} \text{rejected}, & \text{if raduis} < 3\alpha \textit{ or } \text{Sagitta} \leq 2\alpha \\ \text{small arc}, & \text{if Sagitta} < \text{raduis } \textit{and } \text{Sagitta} > 2\alpha \\ \text{big arc}, & \text{if Sagitta} \geq \text{raduis } \textit{and } \text{Sagitta} > 2\alpha \\ \text{circle} & \text{if distance(Endpoint1, Endpoint2)} \leq 2\alpha \textit{ and } \text{Sagitta} > 2\alpha \end{cases} \tag{3.3}$$

Next, the fitness is calculated for accepted solution where

$$F = \frac{1}{N} \sum_{i=0}^{N} dist(AcceptedList[i])^2 \tag{3.4}$$

where $N = |AcceptedList|$ and $dist(.)$ is the euclidean distance between each pixel in *AcceptedList* and the computed equation.The accepted primitives are saved in *solutionList* with their fitness function.



(a) (b)

Figure 3.2.5: (a)Circle fitting, (b) Line fitting

3.2.3 Selection and Crossover

The algorithm saves the results and generates a new population. It uses random selection to select chromosomes; each chromosome being selected once. This method is chosen because we aim to generate the largest number of possibilities and do not stuck on one primitive knowing that each *LC* do not always represents only one primitive (see Figure 3.2.4a). Every two parent chromosomes selected should be of same type (for example

line-line or Circle-Circle) and generate two new children using the single point crossover. This type of crossover split the parental chromosome randomly into two parts. The child is created by adding the first part of the first parent with the second part of the second parent. As an example, assume that $P1 = [p_1, p_2, p_3]$ and $P2 = [p_4, p_5, p_6]$ are the parents; the generated child are $C1 = [p_5, p_2, p_3]$ and $C2 = [p_4, p_1, p_6]$. The mutation stage is not used in our model because after some generations it might introduce a chromosome that does not represent LC and cause an infinite loop as explained in ***Fitness Function*** paragraph.

The previous steps are repeated until the number of generations is done (Condition A in Algorithm 3). This generates a set of possible solutions for the best primitive that represents LC. Normally, primitives with less number of pixels might have a better fitting value. Thus, based on experimental results only primitives greater than or equal to 80% of the largest detect primitives are accepted as shown equation 3.5.

$$\text{solution} = \begin{cases} \text{rejected}, & if\ \{\ \frac{|PS|}{max(solution)} \leq 0.8; \forall PS \in solution\} \\ \text{accepted}, & otherwise \end{cases} \tag{3.5}$$

The best solution ($Bestsolution$) that represents LC is the one having the maximum fitness function F in the list $solution$. The width is recomputed for estimating the final width of $Bestsolution$; and calculate endpoints when needed. The detected pixels are deleted from I_1 (and not from I_1') and the connected component in I_1 are updated; the algorithm re-select the new longest connect component and search for the new best primitive. The algorithm selects a new component from CCI when I_1 is null, or it contains only connected components smaller than 3 pixels (Condition B in Algorithm 3). Moreover, the algorithm stops when each component in CCI is null or smaller than 3 pixels.

3.2.4 Post Processing

Owed to the tolerance α in the vectorization stage, some primitives might overlap (share same pixels). Thus, a post processing method is proposed to process overlapping and find accurate endpoints and junctions.

The method begins by removing the fully overlapped primitives; Assume that SA is the set of pixel that belong to a primitive A and SP is a set of pixels that belong to all other primitives (except A). if $|SP \cap SA| = |SA|$ then the primitive A is deleted (see figure 3.2.6d). Next, assume we have a primitive A that partially intersect a primitive B where $|SA \cap SB| \neq \emptyset$, different scenarios can occur:

1. Circle case:
   If the pixels of the circle and the endpoints pixel of line or arc intersects, the intersecting pixels between the arc or line and the circle is computed, and the point that first intersects the circle is selected to be the new endpoint of the line or arc. If

the pixels of the circle and Midpoint pixels of line or arc intersects nothing happens (see figure 3.2.6a).

2. Endpoint-Midpoint (except circles):
   If the endpoints pixels of the first primitive and the midpoint of the second primitive intersects, the intersection pixels are computed, and the point that first intersects the midpoint is selected; the endpoint of the first primitive is updated (see figure 3.2.6b).

3. Endpoint-Endpoint (except circles):
   If the endpoints pixels of the first and the second primitive intersects, the intersection pixels are computed, and the best point that minimize the distance between the two primitives is selected. The endpoints of the first and second primitives are updated (see figure 3.2.6c).

Finally, the algorithm calculates the exact endpoint of each primitive by finding the nearest points (float point) to the pixel endpoint (integer) that belong to the primitive equation.

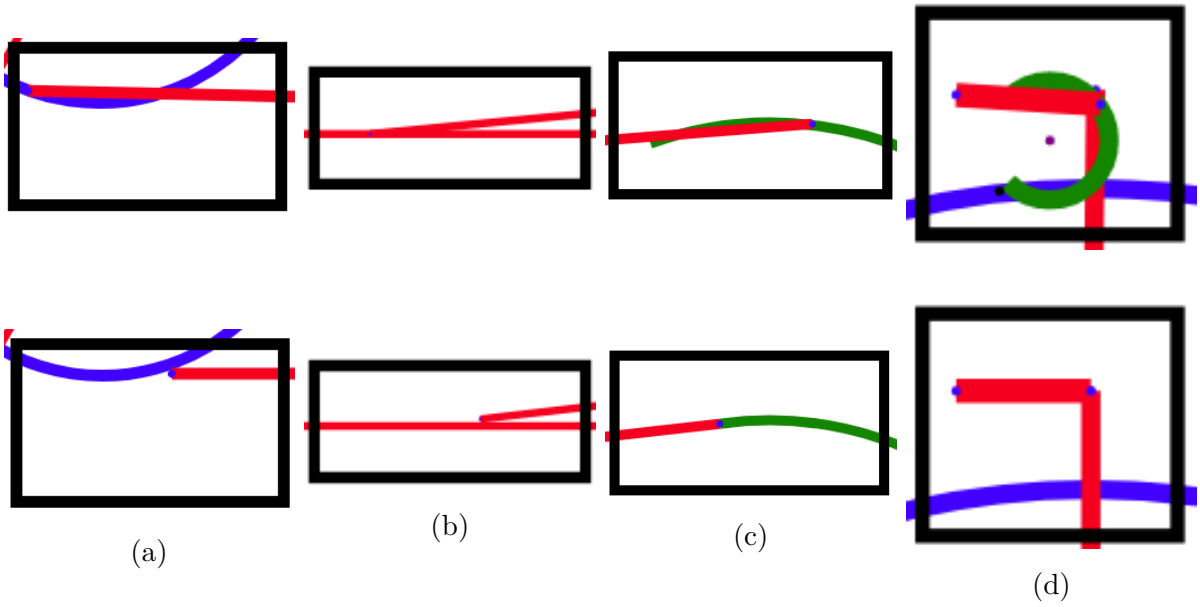


Figure 3.2.6: Before (top images) and after (bottom images) pre processing stage: (a) Circle case, (b) Endpoint-Midpoint case, (c) Endpoint-Endpoint case, (d) Fully Overlapped case

3.3 Results and Discussion

The algorithm is evaluated using images from different International Association of Pattern Recognition (IAPR) contests on graphics recognition: GREC 2003, GREC 2005 [60],

GREC 2007 and GREC 2013 [102]. In addition to our own dataset collected from different sources like books and CAD digitized images. These datasets contain noisy and different resolution data, old and new drawings.

3.3.1 Comparison with Previous Methods

The proposed algorithm is compared with previous methods using a qualitative metric (visually) and a quantitative metric that is the Vector Recovery Index (VRI) [103]. VRI is a metric that compares the vectorized drawing with the ground truth vectors. This metric is compatible with different types of primitives such as straight lines, arcs, and circles (solid and dashed). The VRI value score is between 0 and 1, where the higher is better. Moreover, the VRI score takes into consideration both the detection and false alarm rates. In this section, we compare our work with previous algorithms that are evaluated during an arc detection competition. Thus in the following evaluation, the computed VRI takes into consideration only circles and arcs. This evaluation uses the fifth and sixth contests of graphic recognition datasets where:

1. GREC 2003 contains four scanned drawings (named as '1','2','3','4') in 256 grayscales and binarized with moderate thresholds. Images '1' and '4' are binarized again with a higher threshold to generate images '1_230' and '4_230' with thicker line width. However, images '2' and '3' are binarized with smaller threshold to generate images '2_100' and '3_100'. Moreover, images '1_*n*4', '2_*n*4', '3_*n*4' and '4_*n*4' are generated by adding synthesized noises [61] .

2. GREC 2005 contains six real scanned drawings (named as '5','6','7','8','9','10') where each one is used to generate two images one with random noise ('5_rn',...,'10_rn') and the other with salt and pepper noise ('5_sp',...,'10_sp').

Our algorithm is compared with TIF2VEC [104], Song method [105], Hilaire [65, 106], Daniel [107] and ASKME [98]. The results presented in tables 3.3.1 and 3.3.2, are the average results of ten runs where the number of chromosomes and number of generations are fixed to 50.

As shown in tables 3.3.1 and 3.3.2 our system achieved the best average VRI comparing with all the other methods. However, it is noticeable that Hilaire method outperforms our method with images ('9','9_rn' and '9_sp'). The lack of accuracy in our method is caused due to the preprocessing stage, where Hilaire method fills holes and ours does not. A visual comparison is done in figure 3.3.7 to highlight the problem in our method. The blue circle in figure 3.3.7a shows the unconnected pixels of the arc in the source image, in figure 3.3.7c it is obvious that the hole is filled before the vectorization process. Figure 3.3.7d shows that our method detects two arcs instead of one due to the missing pixels in the original image. In other words, our method is fragmenting one arc into two which leads to the decrease our VRI score.

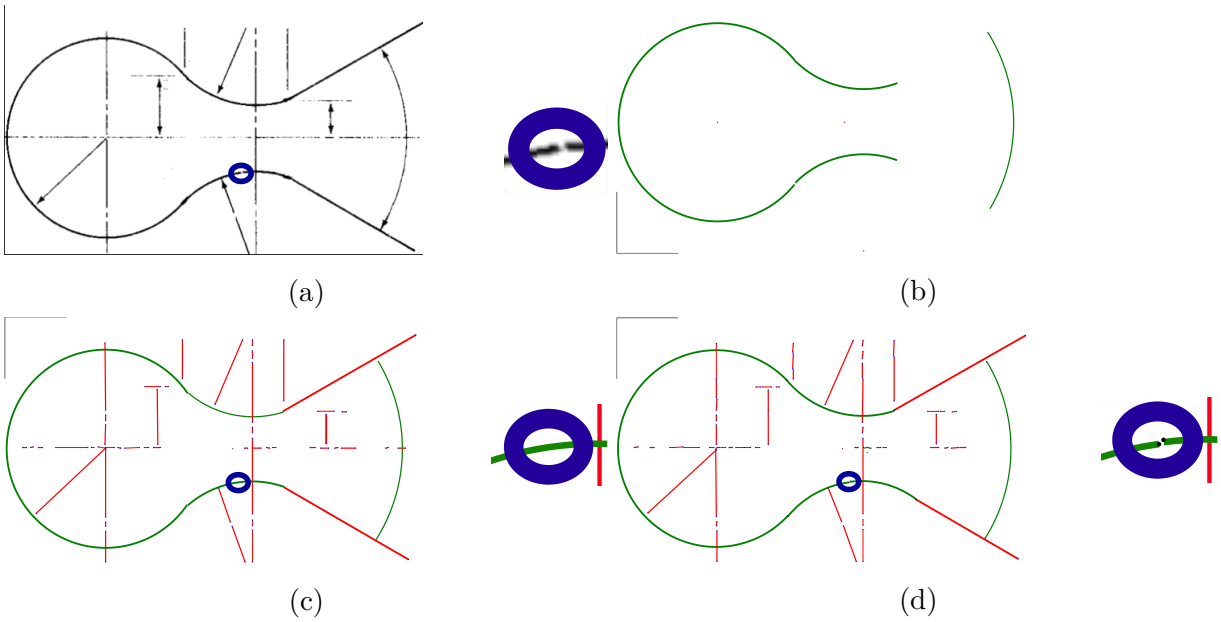


Figure 3.3.7: (a) Source image, (b) Ground truth , (c) Hilaire method, (d) Our method

One of highest VRI reached comparing with the other methods is for images ('8','8_rn','8_sp'). As shown in figure 3.3.8, TIF2VEC method does not detect circles however for arc detection it works better. Keysers approach unlike TIF2VEC detects circles correctly but miss-detects arcs. Hilaire method detects arcs and circles; however, the default parameters of circles bounds [106] lead the algorithm to fragment the largest arc as shown in Figure 3.3.8e. Nevertheless, our method, since it automatically tune parameters (estimate parameter for each primitive), detects arcs and circles accurately and without any fragmentation but fails to detect the smallest arc due to arrows (in source image), which leads to miss-estimate the width; Thus, the arc solution is rejected because $Sagitta \leq 2\alpha$ (see equation 3.3). This example shows the high ability of our method in avoiding fragmentation. In the other hand, it shows the influence of arrows head which increase the width estimation (particularly when the primitive is small comparing to arrow heads) and reject a correct solution.

Another interesting example to show the robustness of our method is image '5' (see figure 3.3.10). Our algorithm finds all circles successfully but Hilaire method miss-detects small circles. The fragmentation of primitives in Hilaire method is due to extracting primitives between two junctions separately, where our method performs a global search (not only between two junctions) to select the best primitive. Nevertheless, Hilaire method detects all the arcs when our method miss-classify two of them because the estimation of $Sagitta$.

Figure 3.3.9 shows the visual comparison, we see that for image '1' (first row) our algorithm estimates the upper arc more accurately than the ASKME method and middle circles better than Hilaire-Tombre method. For the image '2_n4' (second row), only our

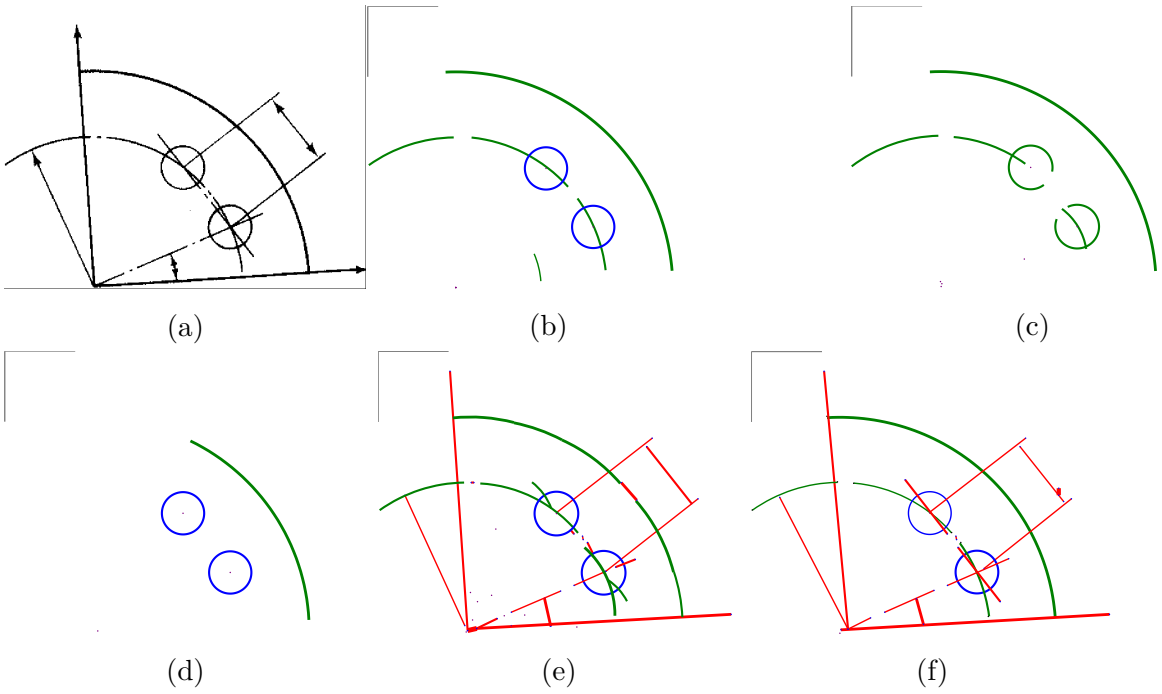


Figure 3.3.8: (a) Source image, (b) Groud truth, (c) TIF2VEC method, (d) Keysers method, (e) Hilaire method, (f) Our method

method manages to detect the small arc connected to the largest one.

Based on our knowledge, the method proposed by [65] achieves the best VRI average until know; and it is obvious that ASKME has the better visual results due to primitive classification and arrow heads recovery. Nevertheless, our method outperforms other methods based on average VRI reflecting the high accuracy of our method. In addition, our method generates good visual unfragmented results.

3.3.2 Algorithm Robustness and Hyper-parameters Effects

In this section, we are going to evaluate the effect of hyper-parameters on our algorithm. We start by evaluating the effect of the number of chromosomes and generations on the accuracy of our algorithm. Next, we assess the impact of changing the width parameter α. These two evaluation helps us to select the best parameters for vectorizing input images. In addition, we study the robustness of our algorithm in terms of noise, rotation, and scale. This evaluation is performed to ensure that our algorithm can vectorize scanned drawings that can be rotated slightly or affected by salt and pepper noise during the scanning process.

3.3.2.1 Number of Chromosomes and Generation Effects

The only parameters needed to this algorithm are the number of generation and number of chromosomes as seen in section 3.2, therefor it is important to study the evolution of results when these values changes. We start our test by fixing the number of chromosomes

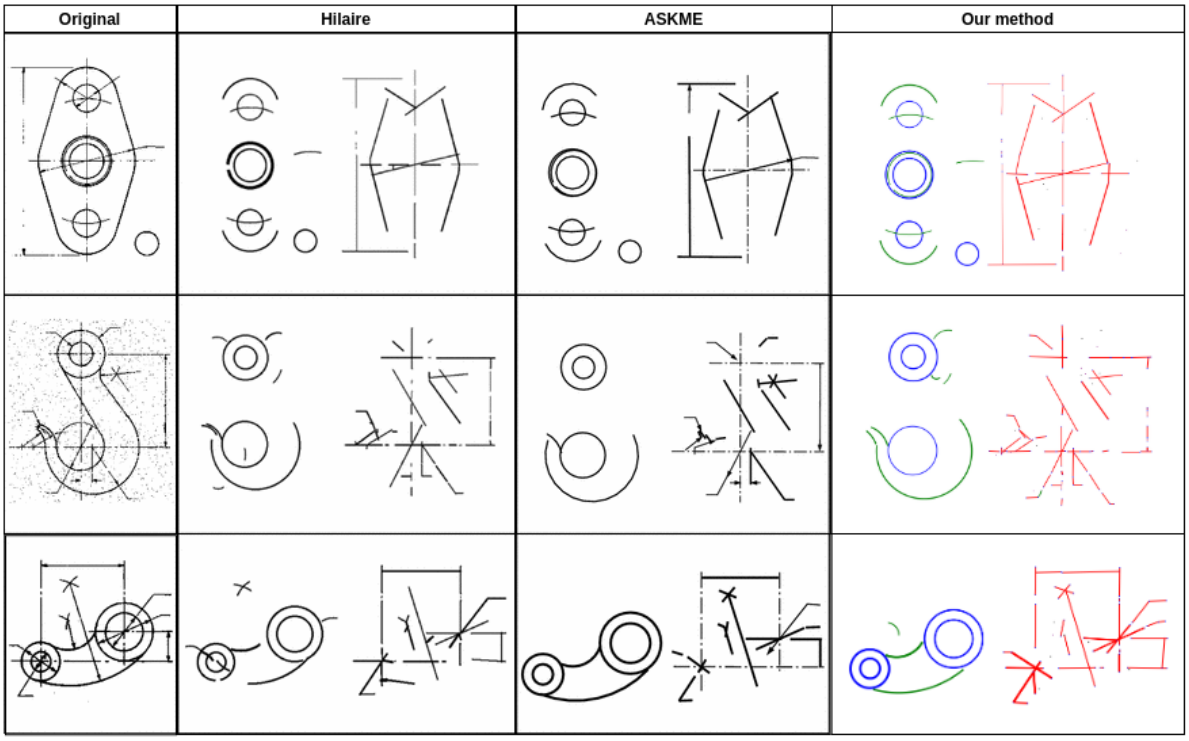
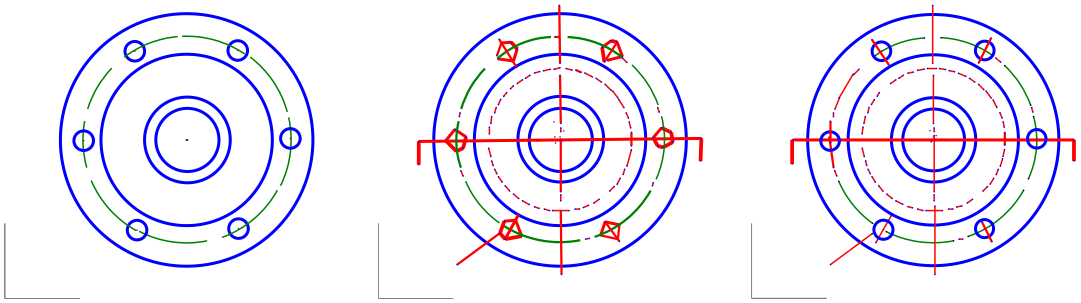


Figure 3.3.9: Visual comparison



(a) (b) (c)

Figure 3.3.10: (a): Ground truth , (b): Hilaire method, (v): Our method

to 50 and changing the number of generation from 5 to 10,30 and 50.The average VRI (of 10 loops) of images in table 3.3.1 goes from 0.8 for 5 generation and almost stabilize at 0.83 for generation 10,30 and 50. Moreover, for images in table 3.3.2 the average VRI (of 10 loops) starts 0.78 for 5 generations, slightly increases to reach 0.79 for 10 generation, it continues slightly increasing to reach 0.81 for 50 generations. Next, we fixed the number of generations and change the number of chromosomes from 5 to 10 and 30. The average VRI (of 5 loops) of images in table 3.3.1 starts 0.76 for 5 chromosomes, increases to 0.82 for 10 chromosomes and reaches 0.83 for 30 chromosomes. For images in table 3.3.2 the average VRI increases from 0.75 for 5 chromosomes to 0.79 for 10 chromosomes and reach 0.81 for 30 chromosomes. From this evaluation, we can conclude that the number

Images	TIF2VEC [104]	SONG J. [105]	Hilaire [65]	ASKME [98]	Our Method
1	0.567	0.641	0.756	0.769	**0.853**
1_230	0.589	0.64	0.752	0.745	**0.895**
1_n4	0.664	0.509	0.762	0.714	**0.873**
2	0.439	0.753	0.653	0.654	**0.813**
2_100	0.513	**0.786**	0.707	0.613	0.771
2_n4	0.612	0.703	**0.791**	0.615	0.768
3	0.272	0.532	0.724	0.645	**0.781**
3_100	0.519	0.301	0.74	0.69	**0.817**
3_n4	0.451	0.224	0.697	0.635	**0.783**
4	0.5	0.735	0.663	0.747	**0.862**
4_230	0.323	0.79	0.795	0.739	**0.886**
4_n4	0.399	0.688	0.782	0.727	**0.862**
Average	0.487	0.609	0.735	0.691	**0.831**

Table 3.3.1: The fifth international workshop on graphics recognition GREC 2003

Images	TIF2VEC [104]	Daniel Keysers [107]	Xavier Hilaire [106]	Our Method
5	0.119	0.591	0.904	**0.954**
6	0.896	0.796	0.939	**0.936**
7	0.092	0.268	0.404	**0.458**
8	0.760	0.729	0.736	**0.923**
9	0.855	0.611	**0.970**	0.877
10	0.458	0.614	**0.862**	0.786
5_rn	0.111	0.615	**0.898**	0.868
6_rn	0.852	0.774	0.943	**0.95**
7_rn	0.126	0.347	**0.444**	0.424
8_rn	0.658	0.717	0.693	**0.87**
9_rn	0.722	0.704	**0.930**	0.754
10_rn	0.585	0.576	**0.866**	0.809
5_sp	0.119	0.591	0.910	**0.951**
6_sp	0.841	0.797	**0.959**	0.956
7_sp	0.099	0.265	0.415	**0.419**
8_sp	0.727	0.729	0.732	**0.919**
9_sp	0.764	0.732	**0.961**	0.887
10_sp	0.466	0.614	**0.856**	0.854
Average	0.514	0.615	0.801	**0.811**

Table 3.3.2: The sixth international workshop on graphics recognition GREC 2005

Images / Methods	FvS [109]	CHD [110]	PVF [88]	DVTD [108]	Ours (α_1)	Ours (α_2)	Ground Truth
00050080	67% / 79	67% / 108	95% / 9.5k	86% / 139	83% / 77	84% / 93	139
00050063	86% / 88	68% / 211	96% / 20k	82% / 147	83% / 103	84% / 112	122
00050012	74% / 433	64% / 994	91% / 43k	76% / 579	81% / 478	81% / 629	973
00050084	-	-	-	71% / 89	60% / 66	63% / 84	157
00050026	-	-	-	85% / 84	82% / 64	89% / 69	89
00050024	-	-	-	67% / 25	61% / 24	70% / 31	57
094_clean	-	52% / 349	-	78% / 368	76% / 165	79% / 179	-
107_clean	-	44% / 226	-	84% / 174	72% / 117	78% / 212	-
101_clean	-	38% / 230	-	81% / 187	78% / 125	79% / 135	-

Table 3.3.3: Qualitative comparison on ABC and Real Images IoU% / number of primitives

of chromosomes affect the results more than the number of generations.
This shows that the algorithm start to saturate when the number of generations reaches 10 and number of chromosomes reaches 30. we noticed that this values are affected by the length of the primitive, for a long primitive we need more chromosomes and/or generations while for a short primitive we need less chromosomes and/or generations.

3.3.2.2 Width (α) Effects

Moreover, we compared the proposed method with recently proposed methods such as [108, 109, 110, 88], those methods are used for vectorizing different types of drawing and use not only line and circles but also curves. This comparison uses two metrics reflecting the ability to have accurate results with a minimal number of primitives (low complexity). We rasterize the generated vector, and we compute the intersection over the union (IoU) between the ground truth image and the rasterized vector (generated using our method). Assume we have two raster images or rasterized vector drawings R_1 and R_2, the IoU is computed using the equation $IoU = \frac{R_1 \cap R_2}{R_1 \cup R_2}$. We use two of the datasets used in [108] which are the ABC dataset and real images (we use the directly cleaned images used in [108]). As in this experiments the intersection over union is more important than detecting primitives we compute our algorithm using two different tolerance α_1 and α_2 where $\alpha_1 = \lceil \frac{w}{2} \rceil + 1$ and $\alpha_2 = \lceil \frac{w}{2} \rceil$ (α_1 is similar to the tolerance α used in all previous evaluations). All the tests are done using $NG = 5$ and $NC = 50$.

Based on table 3.3.3, our method outperforms CHD method [110] in real images and ABC datasets. Moreover, the PVF method [88] shows very high accuracy, but it generates vectors with a very high number of primitives. Moreover, our method is very competitive with FvS [109] and DVTD [108] methods, and we reach almost the same IoU with the same number of primitives. Moreover, our method with α_1 is less complex in terms of the number of primitives than the results of our method with α_2. However, our method with α_2 is more accurate in terms of IoU than our method with α_1. Nevertheless, the IoU between our method with α_1 and α_2 are almost very close; thus, we will be able to use

the same tolerance in all other tests. In figure 3.3.13 we visualize the ground truth data with the rasterized vector result.

3.3.2.3 Rotation, Scale and Noise Effects

While scanning documents, different distortion types might occur, such as rotating the paper a little bit or adding pepper and salt noise caused by the scanner. Moreover, the drawing might be of a different scale. Thus to demonstrate the robustness of the algorithm, we analyze its behavior with different scales, rotation angle and noise level. Figure 3.3.12 shows the vectoriztion results of an image with different scales (150 dpi, 300 dpi and 600 dpi) and rotations ($-5°$, $5°$, $10°$, $20°$ and $30°$). Based on table 3.3.1 and table 3.3.2 the VRI score is almost stable for original and noisy image. Moreover, figure 3.3.11 shows that our algorithm is stable for different noise level. Thus, our vectorization method is invariant to different rotation, resolution and noise level, this being a key part for operational purpose.

3.3.2.4 3D Reconstruction

At the end of this chapter, we show two examples of reconstructing 3D models using our vectorized data. The tool is called Qrec, which:

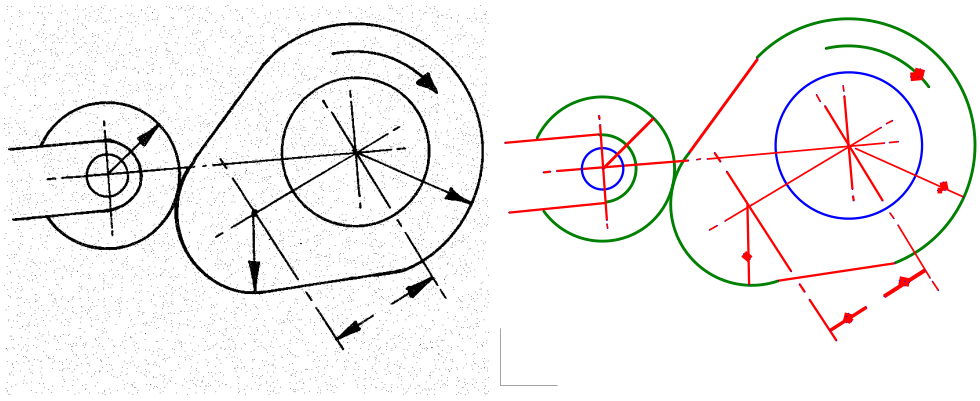
- Labels different view (top, front, bottom, ..) using the method proposed in [111]
- Reconstructs 3D models using the method proposed in [12]

The code of the Qrec tool is published on GitHub: `https://github.com/elrinor/qrec`. The description of the project on GitHub shows that this tool is just a proof of concept, and they do not guarantee to work on any drawings but the ones in the repository.

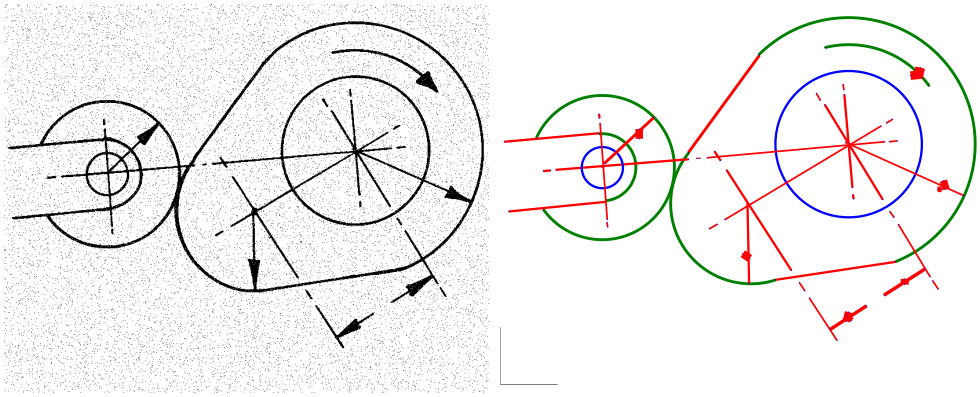
Thus, we convert the drawings in the repository into images, vectorize them using our method, and reconstruct the 3D model using our vectorized data.

Our algorithm considers all entities as a solid line, and thus it cannot label center line, dashed lines, and hashed area, which makes the reconstruction limited in some cases. Moreover, the Qrec tool assumes that lines are split on intersections, which is not considered in our vectorization post-processing stage. Therefore, to overcome those two limitations, in figure 3.3.14d, we manually remove centerlines from the vectorized drawing (as mentioned in this paragraph, the centerlines are vectorized as solid lines which crash the Qrec application). Moreover, in figure 3.3.15d, we split the lines manually at their intersections to reconstruct the model shown in figure 3.3.15e. The reconstructed model is missing the holes because dashed small lines are detected as solid lines. Thus, we manually label small solid lines as dashed lines and recompute the 3D reconstruction. The result shows that the algorithm is able to detect two of three holes.

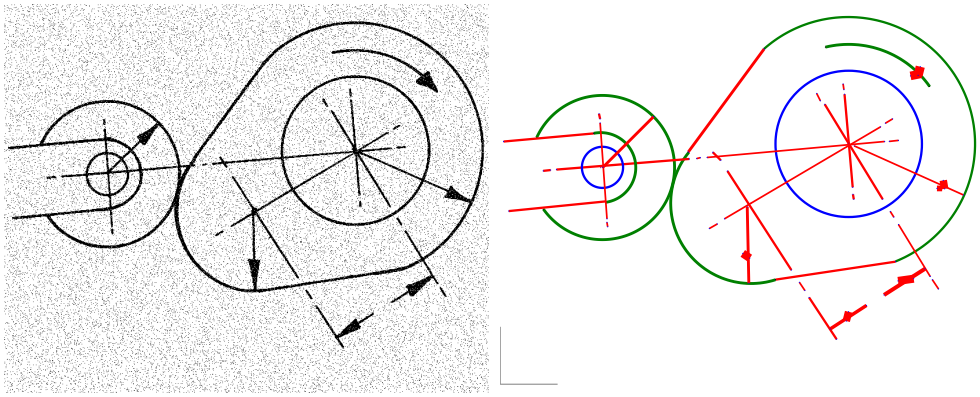
The main concept of applying these two examples (in figures 3.3.15 and 3.3.14) is to show that our vectorization algorithm can vectorize raster images accurately. Nevertheless, the vectorization algorithm still needs some post-processing steps to be able to reconstruct 3D models automatically, in particular labeling dashed lines and hashed areas.



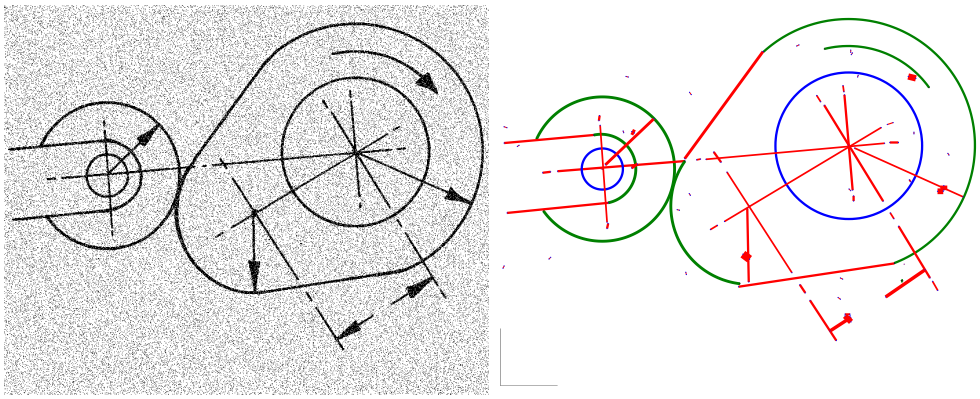
(a)



(b)



(c)



(d)

Figure 3.3.11: (a) 2% of Salt and Pepper noise, (b) 5% of Salt and Pepper noise, (c) 10% of Salt and Pepper noise, (d) 20% of Salt and Pepper noise

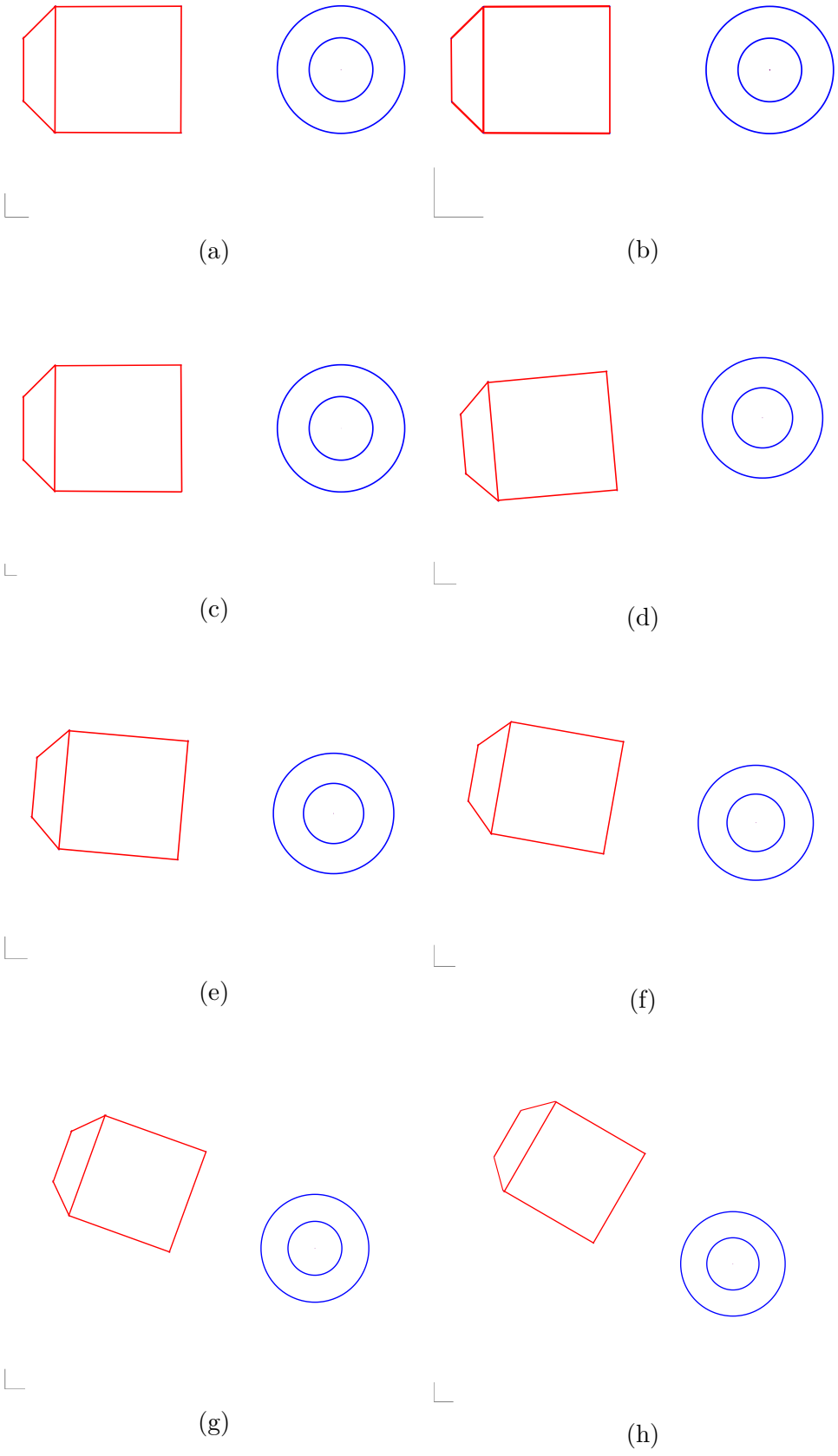


Figure 3.3.12: Vectorization result of image (a): 300dpi , (b): 150 dpi, (c): 600dpi,(d): $-5°$ rotation,(e):$5°$ rotation,(f): $10°$ rotation,(g): $20°$ rotation,(h):$30°$ rotation

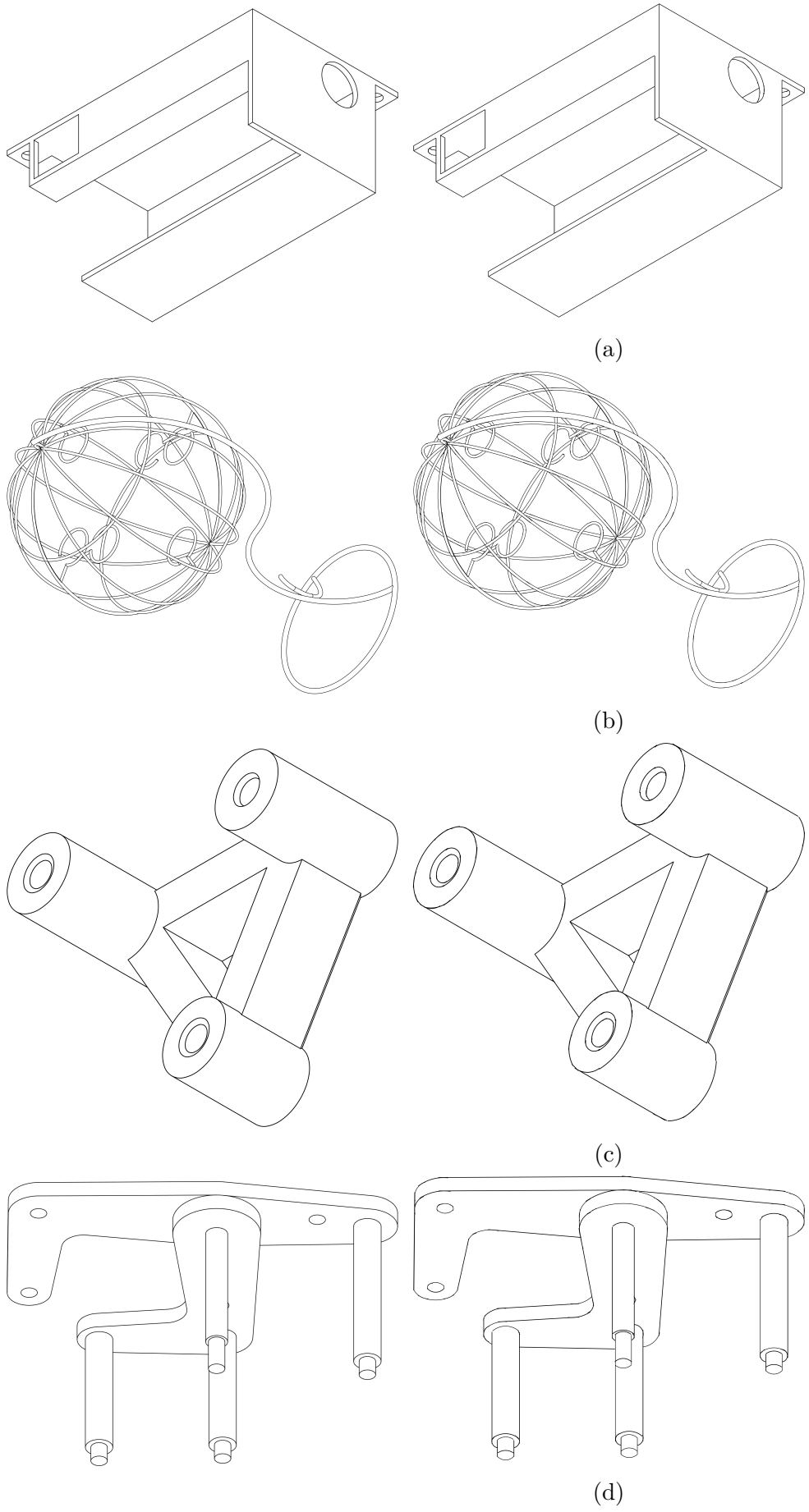
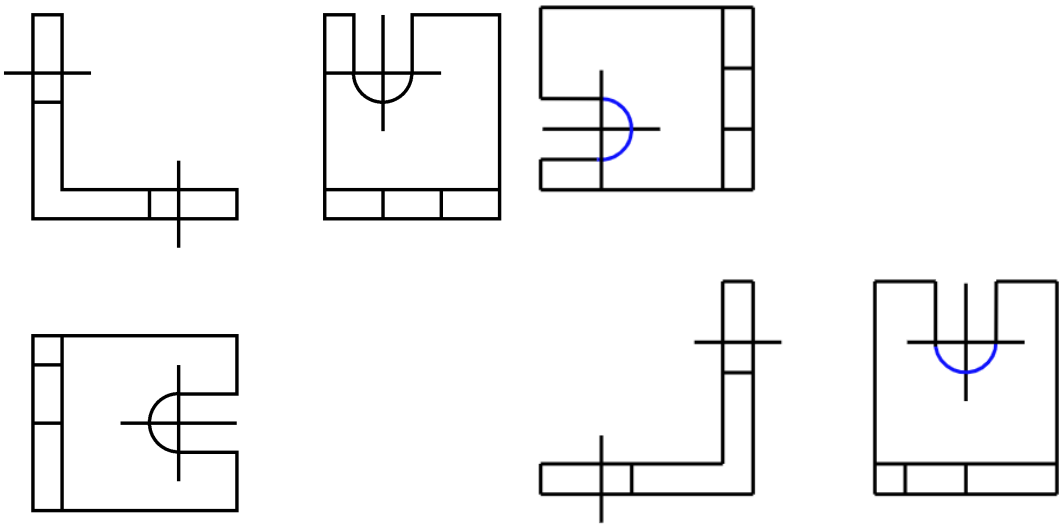
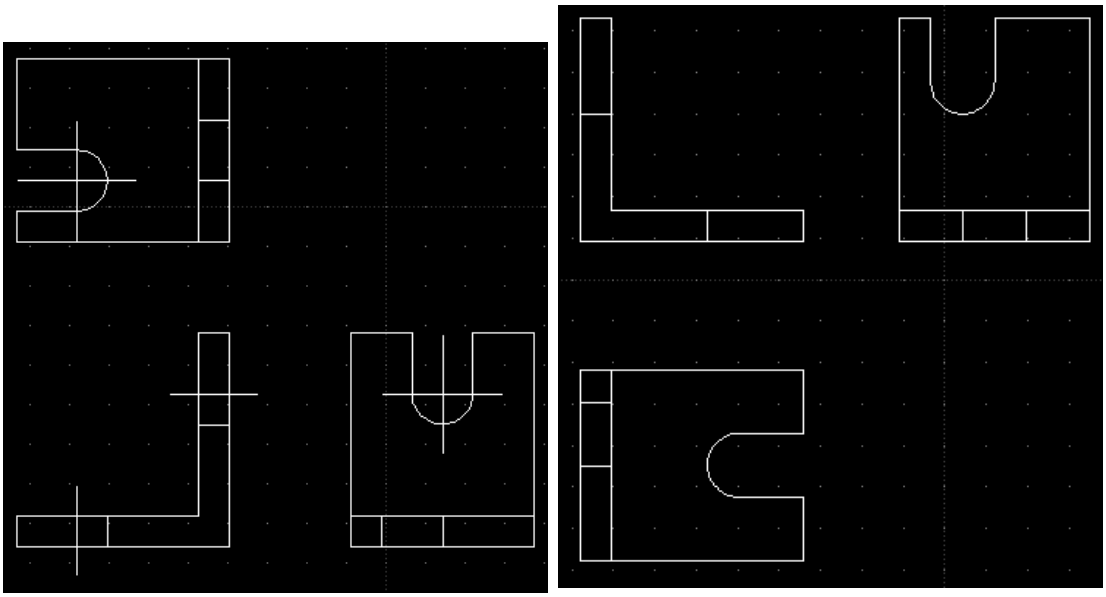


Figure 3.3.13: (a) ABC image 00050001, (b) ABC image 00050012, (c) ABC image 00050080, (d) ABC image 00050002 (left images are ground truth, right images are rasterized vector computed using our method (α_1))



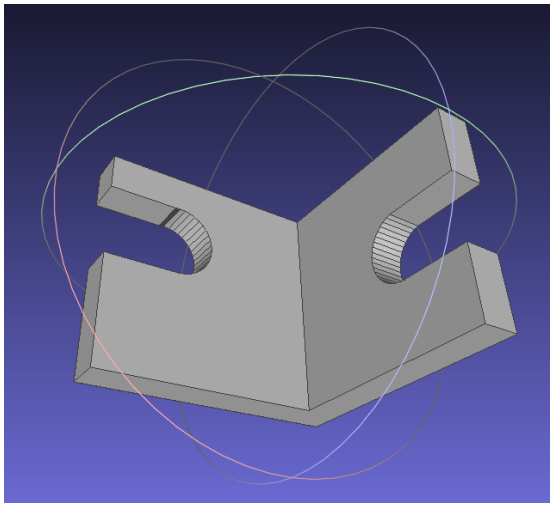
(a) Input raster image

(b) Vectorized - SVG format



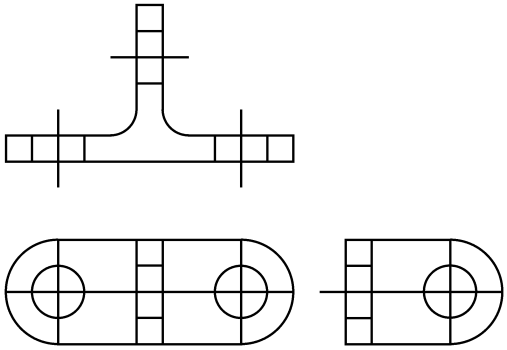
(c) Vectorized - CAD format

(d) Manually modify CAD: rotate the drawing by −90° and remove centerlines

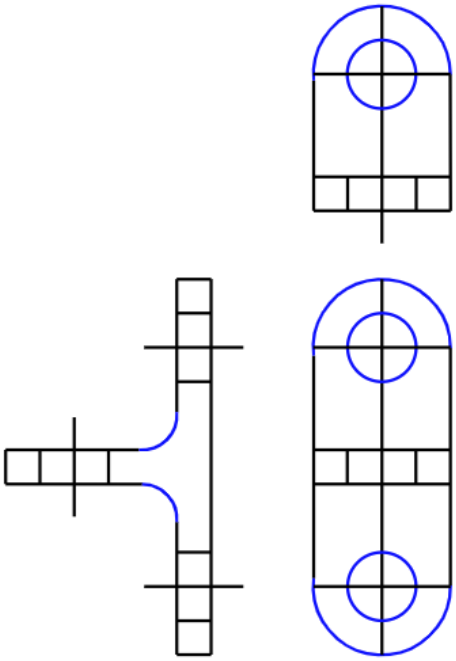


(e) Reconstructed 3D models

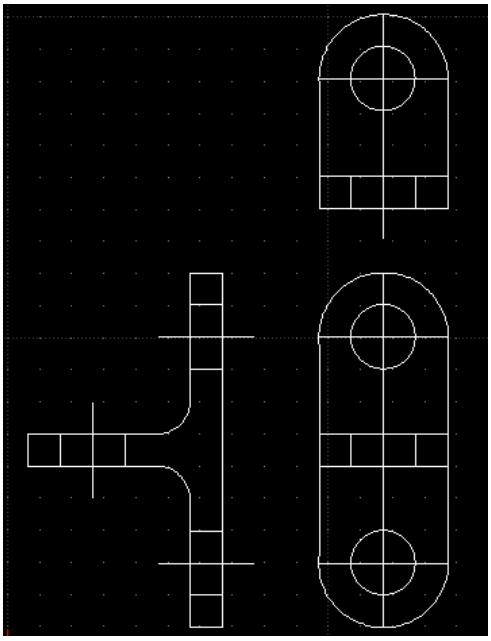
Figure 3.3.14: Example of 3D model reconstruction from our vectorized data using Qrec tool



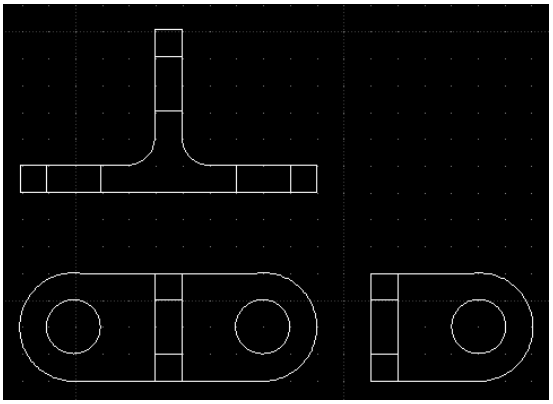
(a) Input raster image



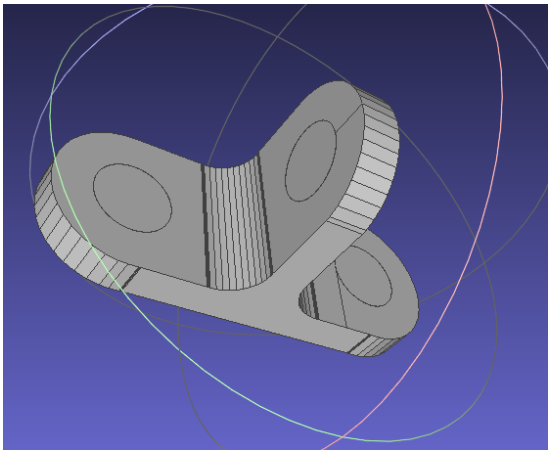
(b) Vectorized - SVG format



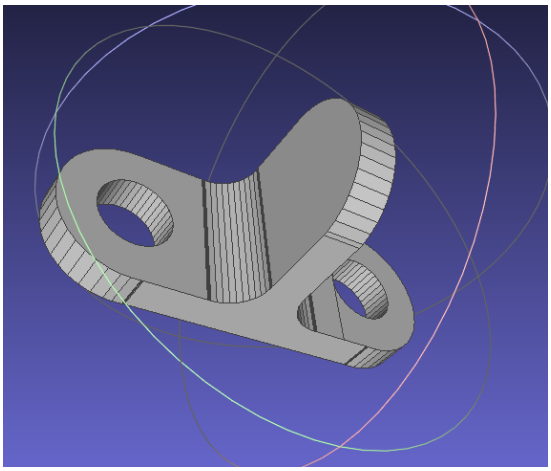
(c) Vectorized - CAD format



(d) Manually modify CAD: rotate the drawing by $-90°$, remove centerlines,split lines at intersection



(e) Reconstructed 3D models



(f) Reconstructed 3D models after changing the solid small lines to dashed

Figure 3.3.15: Example 2 of 3D model reconstruction from our vectorized data using Qrec

3.4 Summary

In this chapter, we proposed a new unsupervised vectorization method based on a genetic algorithm. The proposed method is compared with different vectorization methods and with different metrics. The proposed use OTSU algorithm to binarize input images when needed. The binarized image is skeletonized using the 3,4 chamfer distance method. After the skeletonization, the algorithm separates the skeleton based on junction pixels. Pixels in each skeleton branch are used to generate chromosomes for the genetic algorithm. The algorithm keeps running and detecting primitives until all pixels are represented by a set of primitives. The proposed algorithm detects only straight lines, arcs, and circles.

By evaluating our algorithm, we demonstrate that it is robust to noise and rotation. Moreover, the proposed algorithm can vectorize different types of drawings. The proposed algorithm outperforms other methods in detecting primitives. Moreover, the proposed algorithm is competitive with deep learning methods.

The main advantage of our algorithm is the ability to update the parameters based on primitives width. The proposed algorithm can work with different types of drawings without the need for training data. Moreover, our method does not fragment the vector. However, the main disadvantage of the method is the high computation complexity which is related to the number of iteration and the relation between the number of iteration and the accuracy. In addition, the post-processing methods treat only the problem of intersecting two primitives (in some cases, more than two primitives might intersect).

In the following chapter, we aim to decrease the complexity of the vectorization problem. In this chapter, we proposed a method that segment 2D discrete curve into meaningful features. In contrast, in the following chapter, we propose a technique that separates primitives layers and convert the problem from segmenting 2D discrete curve into primitive detection.

For the sake of exhaustivity, in the next chapter, we compare our unsupervised method with supervised ones widely used in image processing these last years. Thus, we propose a hybrid method taking advantage of the supervised/unsupervised approaches.

Chapter 4

Hybrid Vectorization

Contents

4.1 Introduction

As already stated in the previous chapter, computer vision is a well-known research area that aims to help the computer to understand images or videos and convert them into meaningful data. Deep learning has made a revolution in the computer vision area. It

solved various computer vision problems such as semantic segmentation, instance segmentation, object detection, and more.

The vectorization problem and the 3D reconstruction problem can be processed based on deep learning algorithms:

1. For the vectorization problem, the aim is to convert image pixels into meaningful mathematical equations such as line, arc, and circle equations as already discussed in the previous chapter. This problem can be treated using instance or semantic segmentation, where the instance segmentation model labels each pixel to the instance to which it belongs—however, semantic segmentation labels each pixel to the class to which it belongs. For example, assume we have an image that contains two lines and a circle, as shown in figure 4.1.1. The two lines will be labeled as a line in semantic segmentation, whereas in instance segmentation, each line will be labeled as an instance that belongs to the line class. The proposed method in this chapter is a hybrid method where we separate layers based on a deep learning segmentation model, and then we vectorize each layer alone based on an unsupervised approach.

2. For the 3D reconstruction problem, the aim is to directly convert raster orthogonal views into a 3D model. This problem is well-known in computer vision and deep learning under the area of 3D reconstruction using multi-view images, which convert a bunch of 2D images to a 3D model. The main idea is to study the ability to reconstruct 3D solids from 2D views without passing through vectorization.

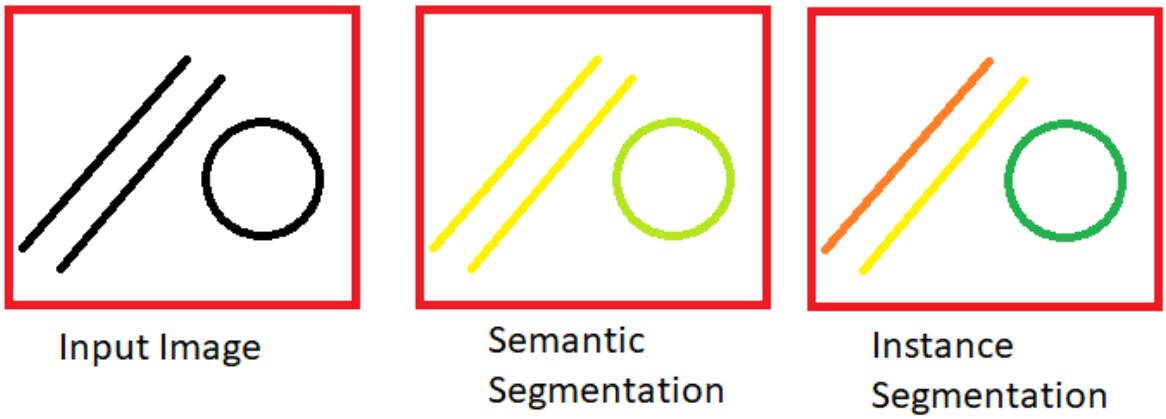


Figure 4.1.1: Semantic vs instance segmentation

In this chapter, we proposed a hybrid vectorization method which decreases the complexity of vectorization problem of segmenting 2D curves by detecting simple primitives such as straight line, circle, and arcs. Moreover, the supervised stage of the proposed

method automatically learns and identifies patterns to separate different types of primitives.

The proposed method is based on two main stages: Segmentation and detection. In the segmentation stage, we tested five different semantic segmentation models and one instance segmentation model on a dataset of eight labels. Besides, we tested five different semantic segmentation models and one instance segmentation model on a dataset of four labels. We study the effects of increasing the number of labels on different models and the performance of different instance and semantic segmentation models. The second stage is the detection stage; we use the best model selected from the segmentation phase to separate layers and then detect primitives from each layer separately. Finally, we study the possibility of reconstructing 3D solids from 2D views without passing through the vectorization process. The proposed algorithm is a hybrid algorithm as detailed in section 4.6. The first stage uses a supervised approach (convolutional neural network) to segment images. The second stage uses an unsupervised approach (used in chapter 3) to vectorize images.

4.2 Literature Review

Vectorization

The image vectorization research area has been one of the most important areas between 1990 and 2010 with the rise of CAD software. This research area aimed to convert the paper-based drawing into vectors to be used by different software such as AutoCAD. Moreover, image vectorization is used in various areas such as logos vectorization and hand drawing vectorization. Nevertheless, we are focusing on vectorization of CAD drawings; thus, we discussed different unsupervised vectorization methods in chapter 3. However, in this chapter, we are focusing on supervised methods of paper-based drawing vectorization.

Different supervised methods were proposed in the literature review. *Liu et al.* [112] proposed a method to convert the raster floor plan into vector format by training a model to detect a limited set of junctions and use integer programming to convert the labeled junctions into a set of lines or boxes. This method reaches high accuracy of vectorization. However, the main drawback of this method is the inability to handle curved and diagonal lines. *Kim et al.* [92] proposed a technique to segment semantically line drawings using neural networks. This method is based on two concepts, the Pathnet, and the Overlapnet. This method predicts the possibility that two pixels occur on the same path and then semantically segment the line drawings using graph cut. The main drawbacks of this method are the computation complexity because it is related to the number of pixels in the image and the inaccuracy of segmenting complicated junction. *Guo et al.* [113] proposed a method to vectorize drawings based on two phases. The first subdivides the lines into partial curves by ignoring the junctions, and the second reconstructs the topology at junctions by predicting the line connectivity. This method solves the problem of computation complexity and junction inaccuracy caused by the technique of *Kim et*

al. [92]. However, this method is susceptible to noise. Different works focus on vectorizing sketches. Nevertheless, these methods produce less desirable results for technical line drawings.

Closer to our work, *Sharma et al.* [114] propose the CSGNet method which is recursive neural network model to generate a program that defines the relation between primitive based on CSG modeling. This method shows accurate results; however, the possible positions and sizes of the primitives are limited to the size of primitives. Moreover, *Huang et al.* [115] proposed a method to detect and separate primitive layers, which outperforms other methods in terms of computational complexity and accuracy. However, this method does not work with highly overlapping images. Recently, *Egiazarian et al.* [108] proposed a method for vectorizing engineering drawings using deep learning. The proposed method first cleans the raster drawing, split images into patches, and predicts primitives. Next, the detected primitives in each patch are refined using iterative optimization and merged together to generate a vector image. However, this method is limited to detect line segments and quadratic Bezier curves.

3D reconstruction

As our final goal is to reconstruct 3D models, different proposed methods discussed the reconstruction 3D model issue from different 2D images, which has been an important research area for a long time. This area contains different sub domains such as reconstruction 3D models using natural 2D images from different views, 3D object reconstruction using calibrated camera pose, and others. Previously, most proposed methods were based mainly on dense feature extraction and matching [116]. Those methods present good performance when a small margin separates the multiple viewpoints. Newly, the high availability of large 3D datasets such as Shapenet pushes the researchers to investigate in 3D shape reconstruction from single or multiple 2D images using deep learning. Many attempts have been addressed this problem, such as Pixel2Mesh [117], 3D-R2N2 [118], AttSets [119] and SMDVP [120]

Back to the 3D model reconstruction from the orthogonal views, usual methods [13, 33, 121, 12, 122, 123, 124] are considered sufficient, accurate, and suitable when we have clean data because the problem can be well defined mathematically. Different techniques were proposed to solve this problem based on Boundary representation (Brep) and Constructive solid geometry (CSG). *Liu et al.* [122] proposed a method based on Brep concept, where this method needs three orthographic projections to generate a 3D model. This model can handle sectional views, dead end holes, and hidden lines. Moreover, *Ciceck et al.* [33] and *Dimitri et al.* [12] proposed two different methods based on CSG concepts. Like the *Liu et al.* [122] method, these methods can handle sectional views, dead-end holes, and hidden lines. In addition, they accept two and more views.

During 2020, *Seff et al.* [125] published a dataset called SketchGraphs that con-

tains sketches extracted from real-world CAD models. Therefore, we propose a hybrid vectorization method to take advantage of labeled data and improve the unsupervised vectorization approach proposed in chapter 3. In 2019, *Koch et al.* [55] published a dataset called ABC which contains a collection of one million Computer-Aided Design (CAD) 3D models. Thus we render ABC 3D models and generate raster orthographic views to evaluate the performance of a supervised model that reconstructs 3D models from different 2D views on this dataset.

4.3 Semantic Segmentation

Semantic image segmentation is the task of classifying each pixel in an image from a predefined set of classes. The supervised semantic segmentation aims to learn a function f that can map an unlabeled image into a labeled one. In the following paragraphs, we briefly describe a set of networks that are tested and have good performance during the Imagenet competition.

4.3.1 Visual Geometry Group Networks

K. Simonyan and A. Zisserman [126], from the university of Oxford, designed a convolutional neural network model called Visual Geometry Group (VGG). The proposed model participated and won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by achieving 92.7% top-5 test accuracy in the ImageNet, which contains over 14 million images belonging to 1000 classes.

The VGG group designed a collection of convolutional neural networks to study the relation between network depth and accuracy. The collection starts with VGG11 and ends with VGG19, where the number connected to VGG reflects the number of layers. For example, VGG11 consists of 8 convolutional layers and three fully connected layers, whereas VGG19 has 16 convolutional layers and three fully connected layers.

In this work, we used VGG16 architecture that contains 13 convolutional layers and three fully connected layers. The VGG16 architecture consists of two blocks, where each one contains two convolutional layers and a 2×2 max-pooling layer, and followed by three blocks, where each one contains three convolutional layers with and a 2×2 max-pooling layer. After the final pooling layer, the network has three fully connected layers.

4.3.2 Residual Networks

Residual Networks (ResNet) is a powerful deep neural network designed by *Kaiming He et al.* [128]. It won the ILSVRC 2015 in image classification, detection, and localization. Besides, ResNet architecture is the winner of the MS COCO 2015 detection and segmentation competition. There are different networks that belong to the family of ResNet

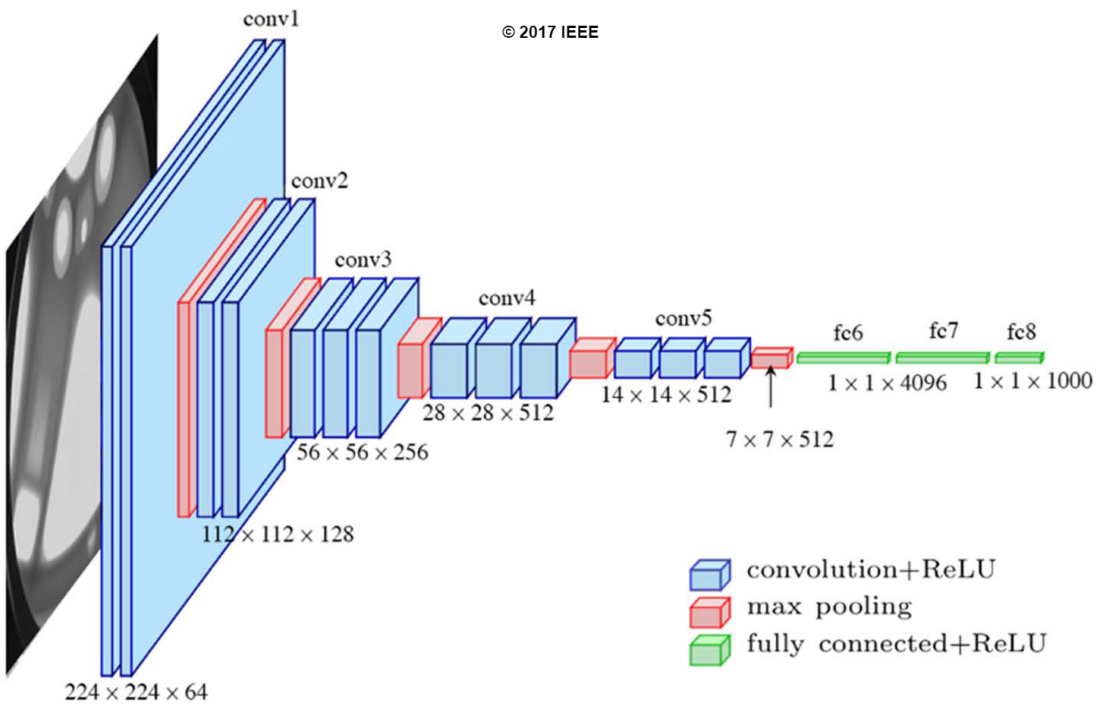


Figure 4.3.2: An overview of the VGG-16 model architecture, this model uses simple convolutional blocks to transform the input image to a 1000 class vector [127]

such as ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110 and ResNet-152. The number that follows the name "ResNet" refers to the number of neural network layers used.

Theoretically, a neural network performance improves or stays the same by increasing the number of layers. However, experimentally, when the network depth increase, the performance gets saturated or degrade rapidly, this is known by the vanishing gradient problem. Thus, the core idea behind ResNet is to overcome the vanishing gradient problem by using skip connections. The skip connection skips training from a few layers and joins directly to the output. This connection helps the network skip any layer that reduces the architecture performance; thus, the accuracy might get saturated but not degrade.

In this paragraph, we are going to describe ResNet50 architecture, as shown in figure 4.3.3. It consists of a convolutional layer with a kernel size 7x7 and 64 different kernels with a stride of size (2), followed by a max-pooling layer with a stride of size 2. Next, there are three blocks where each one consists of three convolutional layers with kernel size 1x1, 3x3, and 1x1, respectively, and the numbers of kernels are respectively 64,64,256. The previous three blocks are followed by four blocks where each one consists of three convolutional layers with kernel size 1x1, 3x3, and 1x1, respectively, and the numbers of kernels are respectively 128,128,512. Subsequently, there are six blocks where each one consists of three convolutional layers with kernel size 1x1, 3x3, and 1x1, respectively, and

the numbers of kernels are respectively 256,256,1024. Alongside, there are three blocks where each one consists of three convolutional layers with kernel size 1x1, 3x3, and 1x1, respectively, and the numbers of kernels are respectively 512,512,2048. In the end, there is an average pooling layer followed by a fully-connected layer with a softmax layer to classify the input.

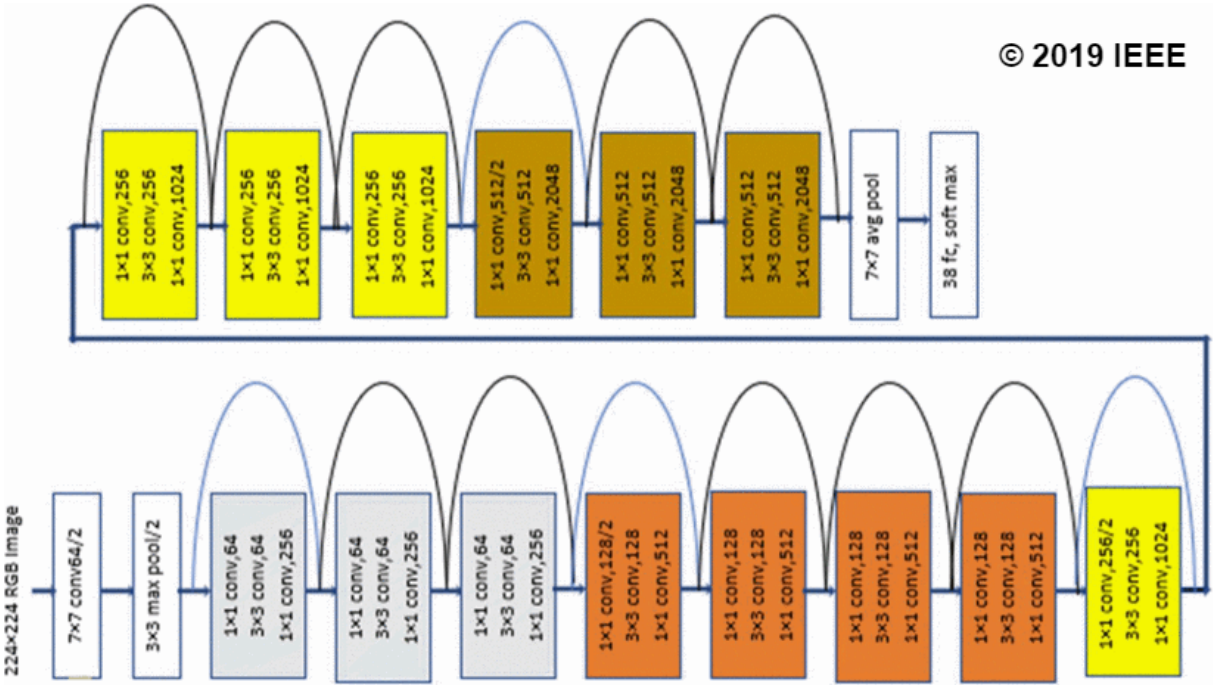


Figure 4.3.3: Residual network [129]

4.3.3 SegNet

SegNet [130, 131] is designed for semantic segmentation; it is a performant architecture for pixel-wise semantic segmentation. The SegNet architecture consists mainly of two main parts the encoder network and the decoder network, as shown in figure 5. The encoder extracts the features vector, whereas the decoder project the discriminative features learned by the encoder onto the pixel space to get a dense classification.

4.4 Instance Segmentation

Instance segmentation is one of the most complex and challenging tasks in the computer vision area. This task is a combination of object detection and semantic segmentation tasks. In other words, Instance segmentation aims to localize and mask pixels of each instance in a specific class. Unlike semantic segmentation, instance segmentation does not label all pixels (pixel-wise), it detects the object and is labeled (the areas that do not contain any object are not labeled). Moreover, instance segmentation can distinguish between instances of the same class that is not possible in semantic segmentation.

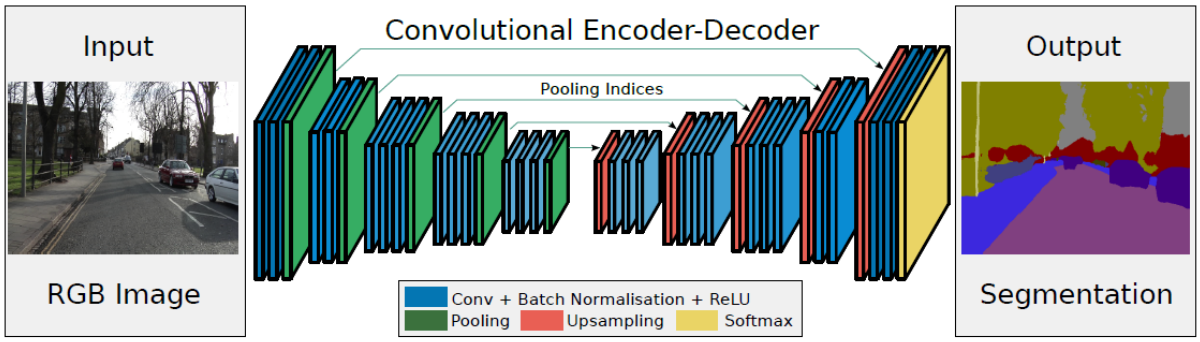


Figure 4.3.4: Segnet architecture [131]

4.4.1 Mask Regional Convolutional Neural Network

Mask regional convolutional neural network (Mask R-CNN) [132, 133] is a deep neural network that addresses the instance segmentation problem in the computer vision area. This algorithm process an input image or video and generates an output with object bounding boxes, classes, and masks.

Mask R-CNN architecture consists mainly of two stages:

1. Generate proposals about the regions of interest, where there might be an object based on the input image.

2. Predicts class of detected object, refine bounding box and predict pixel-wise mask (segment pixel that belongs to the detected object)

To understand the concept behind these two stages, we introduce the backbone network, which is a Feature Pyramid Network (FPN) that consists of:

1. The bottom-up pathway which extracts features from the input image.

2. The top-bottom pathway which generates a feature pyramid map.

FPN maintains strong semantically features at different resolution scales. Thus, the first stage of Mask R-CNN, scan all FPN top-bottom pathway using a region proposal network. As its name shows, this network proposes regions that probably contain objects. Next, the anchor concept is introduced to bind scanned features to their location in the raw image. Anchors are a set of boxes with predefined locations and scales relative to images. Region Proposal Network (RPN) uses the Anchors to figure out whereof the feature map 'should' get an object and what size of its bounding box is.

At the second stage, the method is almost similar to RPN. Instead of using anchors, the network uses a method called Region of interest align (RoIAlign) to locate the relevant areas of the feature map. Finally, a branch of the network generates masks for each object at the pixel level.

The main advantage of Mask Rcnn is its high capabilities to to learn features with different scales.

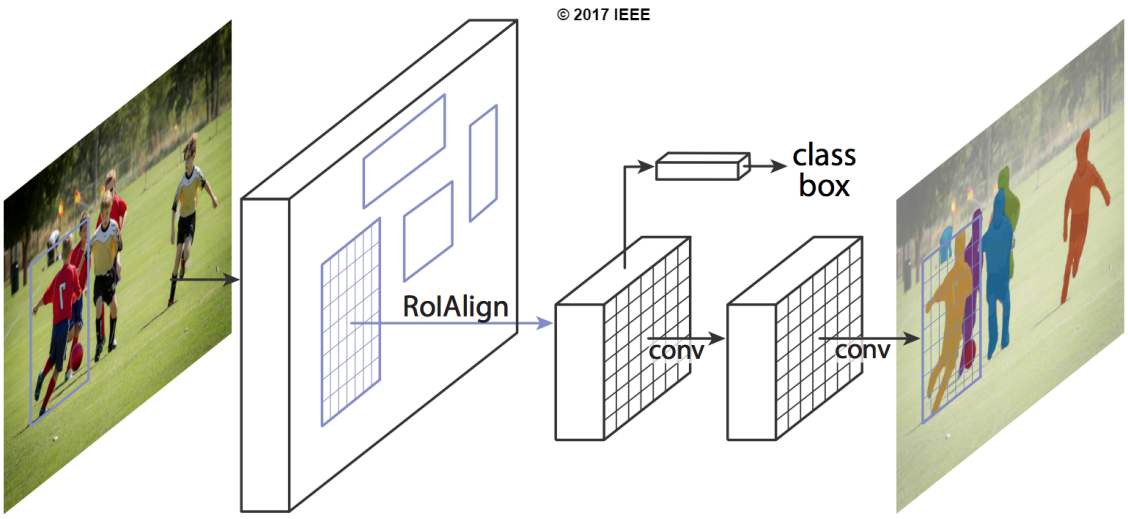


Figure 4.4.5: Mask Rcnn architecture [132]

4.4.2 PointREND

PointRend [134, 133] is a module that treats the segmentation problem as an image rendering problem to render high-quality label maps efficiently. This module can be used with semantic and instance segmentation algorithm to increase the accuracy of masks.

PointRend uses a subdivision strategy to adaptively select a non-uniform set of points at which to compute labels. PointRend is a module that treats the segmentation problem as an image rendering problem to render high-quality label maps efficiently. This module can be used with semantic and instance segmentation algorithm to increase the accuracy of masks.

PointRend uses a subdivision strategy to adaptively select a non-uniform set of points at which to compute labels.

4.5 Performance Metrics

Performance metrics are a set of measures used to evaluate the trained model by comparing the predicted data with ground-truth data. Different metrics are used for different tasks:

Semantic Segmentation evaluation: In the semantic segmentation task, there are a bunch of famous evaluation metrics used to evaluate the model, such as Accuracy, Precision, Recall, F1-score, and Intersection over Union. All these metrics are based and calculated using the confusion metrics. The confusion matrix contains four main elements:

1. True positive (TP) represents the number of correctly labeled pixels

2. True Negative(TN) represents the number of correctly identified pixels as not belonging to a specific class.

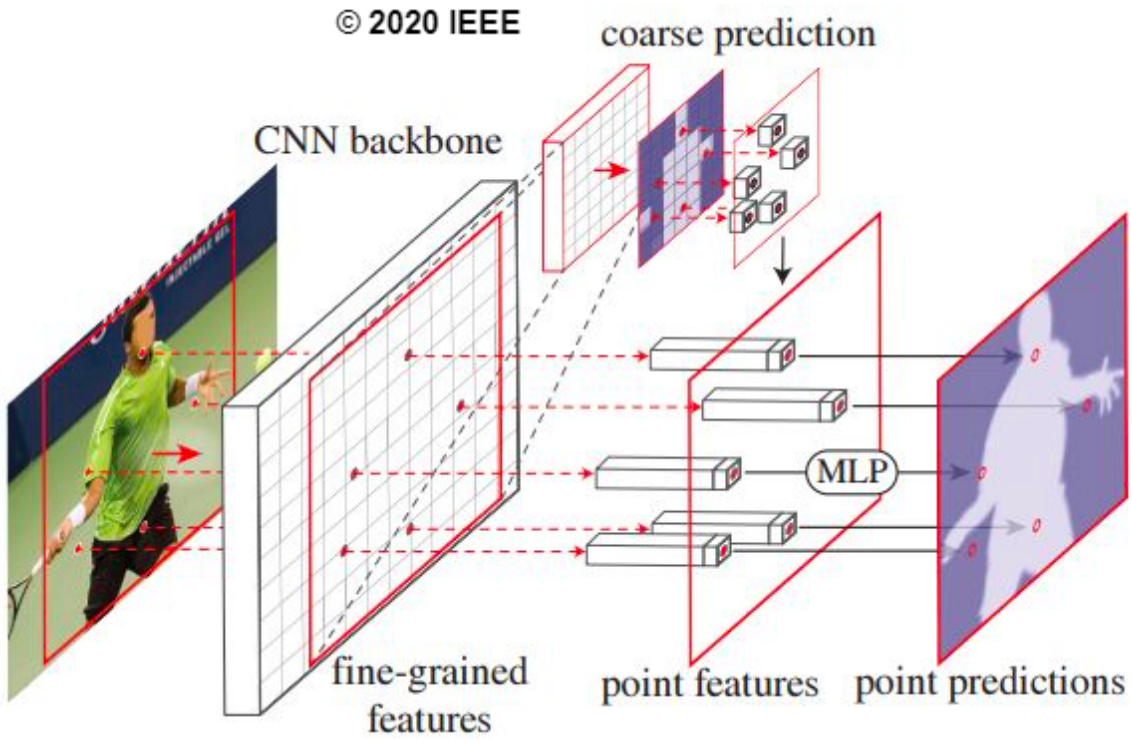


Figure 4.4.6: Point Rend architecture [134]

3. False Positive (FP) represents the number of pixels belonging to other classes misclassified as the target class

4. False Negative (FN) represents the number of pixels that belong to the target class but are misclassified as belonging to other classes

After defining the confusion matrix elements, we can compute a well-known metric used in the semantic segmentation problem, the Intersection over Union, also known as the Jaccard index. The IoU metric represent the similarity ratio between the predicted image and the ground truth image. As stated in section 3.3.2.2, the IoU is the intersection over the union of two images. Thus, the IoU metric can be written in function of confusion matrix as follows:

$$IoU = \frac{TP}{FP + TP + FN} \tag{4.1}$$

where TP is the number of correctly classifies pixels which implies the correct intersection between two images. Moreover, the union of two images is expressed as $FP + TP + FN$ (see to figure 4.5.7)

Instance Segmentation evaluation: The evaluation of instance segmentation models is more complicated than the evaluation of semantic segmentation models. In semantic segmentation, the output is a single mask describing one class, whereas, in instance segmentation, the output is a multi-mask that describes each object in the same class.

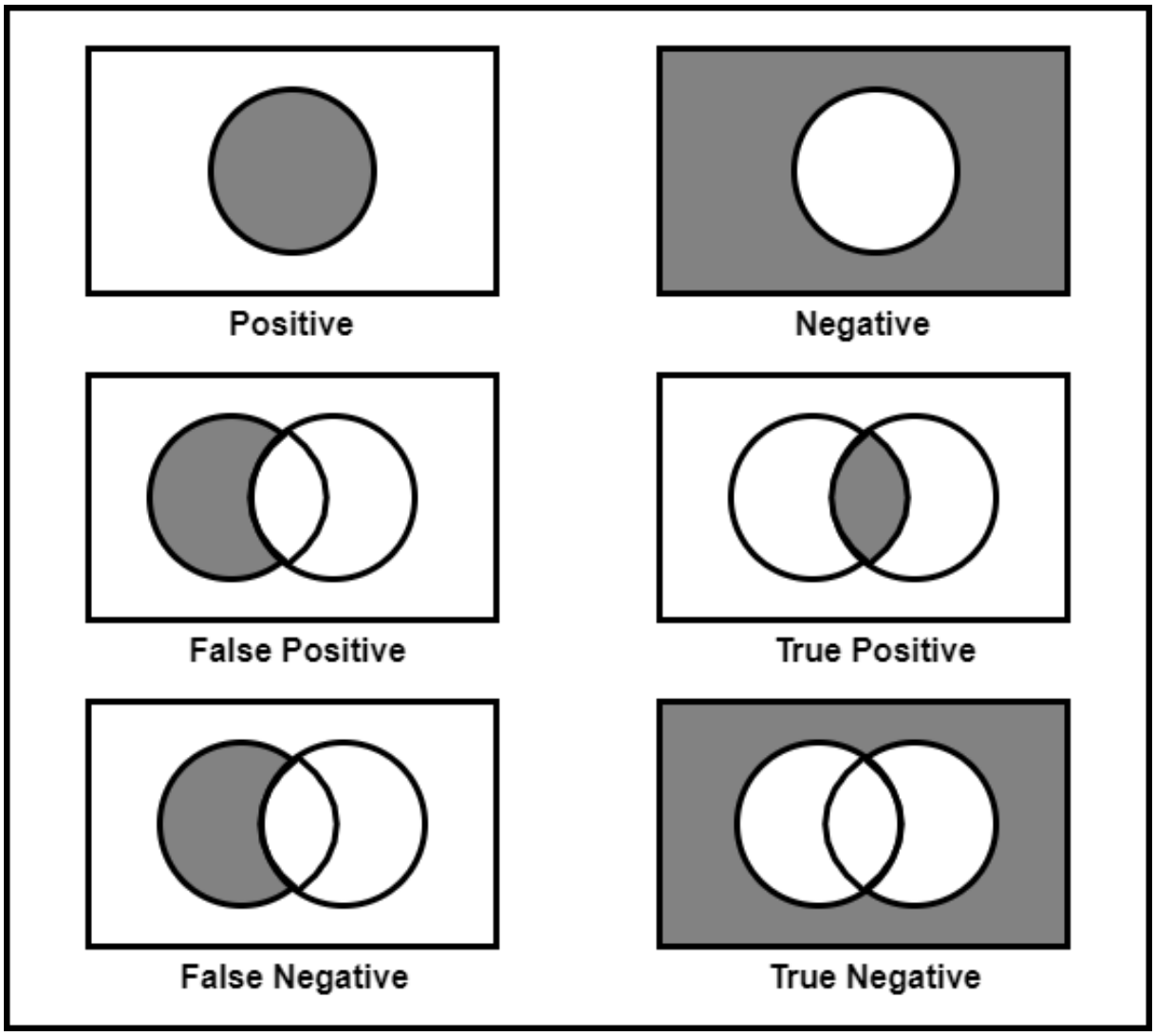


Figure 4.5.7: IoU in function of confusion matrix

In the past years, most instances segmentation algorithm uses the Average precision metrics to evaluate the trained model. However, in instance segmentation and unlike semantic segmentation, the IoU is computed with a different threshold. For example, the Average precision for a fixed threshold such as IoU= 0.8 is written as AP80. AP80 means that only predicted bounding boxes that intersect with the ground truth bounding box and IoU>80% is considered correctly classified. And thus, the instance segmentation is evaluated using the intersection over the union threshold ranges from 0.5 to 0.95 with a step size of 0.05.

Nevertheless, to have a fair evaluation, we convert the instance segmentation into semantic segmentation, and we evaluate them using the same IoU metric used for semantic segmentation.

4.6 Hybrid Vectorization Algorithm

The proposed hybrid vectorization algorithm consists of two main stages:

- The first stage separates different layers where each layer contains one primitive type. For example, the circle layer includes only circles (Basically, we detect the same primitives as in the previous chapter with some refinements as detailed below). The first stage is based on a supervised method that segments different layers using deep learning networks.
- In the second stage, we process each layer based on its type. For example, for the circle layer, we use a circle detection algorithm to vectorize the layer. This stage is based on a modified version of the proposed method in chapter 3, which is an unsupervised approach. In this step, the type of primitive is known, it remains only to estimate the parameters to vectorized.

4.6.1 Segmentation Phase

In this section, we are going to train and evaluate different models with different scenarios. First, we train and evaluate a set of deep learning models using a dataset that aims to segment an input image into eight labels: Straight Line, Arc, Circle, Dashed Line, Dashed Arc, Dashed Circle, Intersection, and Background. Next, we train and evaluate the same models to segment an input image into four labels: Straight Line, Arc/Circle, Intersection, and Background. In the third scenario, we use an instance segmentation algorithm to segment each instance and classify it. In the three scenarios, we use the same datasets; however, the data are differently labeled for each scenario.

4.6.1.1 Dataset Description

SketchGraphs dataset [125] consists of 15 million sketches extracted from real-world CAD models. In this dataset, each drawing is represented as a geometric constraint graph where edges denote designer-imposed geometric relationships between primitives and the graph nodes.The proposed dataset aims to help research in Artificial intelligence and computer-aided design areas. Each sketch in the dataset can contain six different primitives: straight lines, Arcs, Circles, Dashed Lines, dashed arms, and Dashed Circles.

We use 16k sketches in our scenarios where 11k sketches are used for training, 2k sketches are used for validation, and 2k sketches are used for testing. Each sketch of the 16k chosen sketches contains at least three different primitives to ensure a certain balance in instances. All scenarios use the same dataset; however, the labeling process only changes from one scenario to another. First, we generate an image for each sketch; Simultaneously, we generate x images where each image contains only one primitive, and x is the number of the primitive in each sketch. For example figure 4.6.8a represents the

generated sketch and figure 4.6.8b to figure 4.6.8k represents the x images where $x = 10$ and each figure represent one labeled instance that belongs to the generated sketch in figure 4.6.8a.

Scenario 1: In the first scenario, we label the data by converting the input image to binary using OTSU thresholding, and then we create a segmentation map where each pixel value is equal to the integer that represents its class. Thus, pixels that belong to the background has a value of 0, pixels that belong to lines have a value of 1, pixels that belong to arcs have a value of 2, pixels that belong to circles have a value of 3, pixels that belong to dashed lines have a value of 4, pixels that belong to dashed arcs have a value of 5, pixels that belong to dashed circles have a value of 6, pixels that belong to the intersection of two or more primitives have a value of 7. The figure 4.6.9 represents the segmentation masks of each class. We should highlight that the intersection class contains the pixels that belong to more than one category. This may occur in the case of overlapping primitives.

Scenario 2: In the second scenario, we label the data by converting the input image to binary using OTSU thresholding, and then we create a segmentation map where each pixel value is equal to the integer that represents its class. Thus pixels that belong to the background has a value of 0, pixels that belong to lines, dashed lines, dashed circles, and dashed arcs have a value of 1, pixels that belong to arcs and circles have a value of 2, pixels that belong to the intersection of two or more primitives have a value of 3. Similar to scenario 1, the intersection class contains the pixels that belong to more than one category. Figure 4.6.10 represents the masks of different classes.

Scenario 3: In the third scenario, we label data using the Microsoft Common Objects in COntext (COCO) dataset format [135] for instance segmentation. COCO stores annotations in a JSON file of multi blocks as follows:

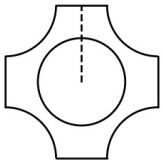
1. Information block includes the year, version, description, contributor, URL, and data created.

2. License block provides a list of image licenses such as license id, license name, and license URL.

3. Categories block contains an id for each category, the category name, and the super category name if needed.

4. Images block contains list of all image information such as image id, width, height, file name, license, and some optional data such as flickr URL, coco URL, and date of capture

5. Annotations block consists of a list of individual instance annotations from each image. For each instance, we have the id, the bounding box coordinates, the bound-



(a) Initial Sketch

(b) Instance 1 - Arc

(c) Instance 2 - Arc

(d) Instance 3 - Arc

(e) Instance 4 - Arc

(f) Instance 5 - Circle

(g) Instance 6 - Dashed Line

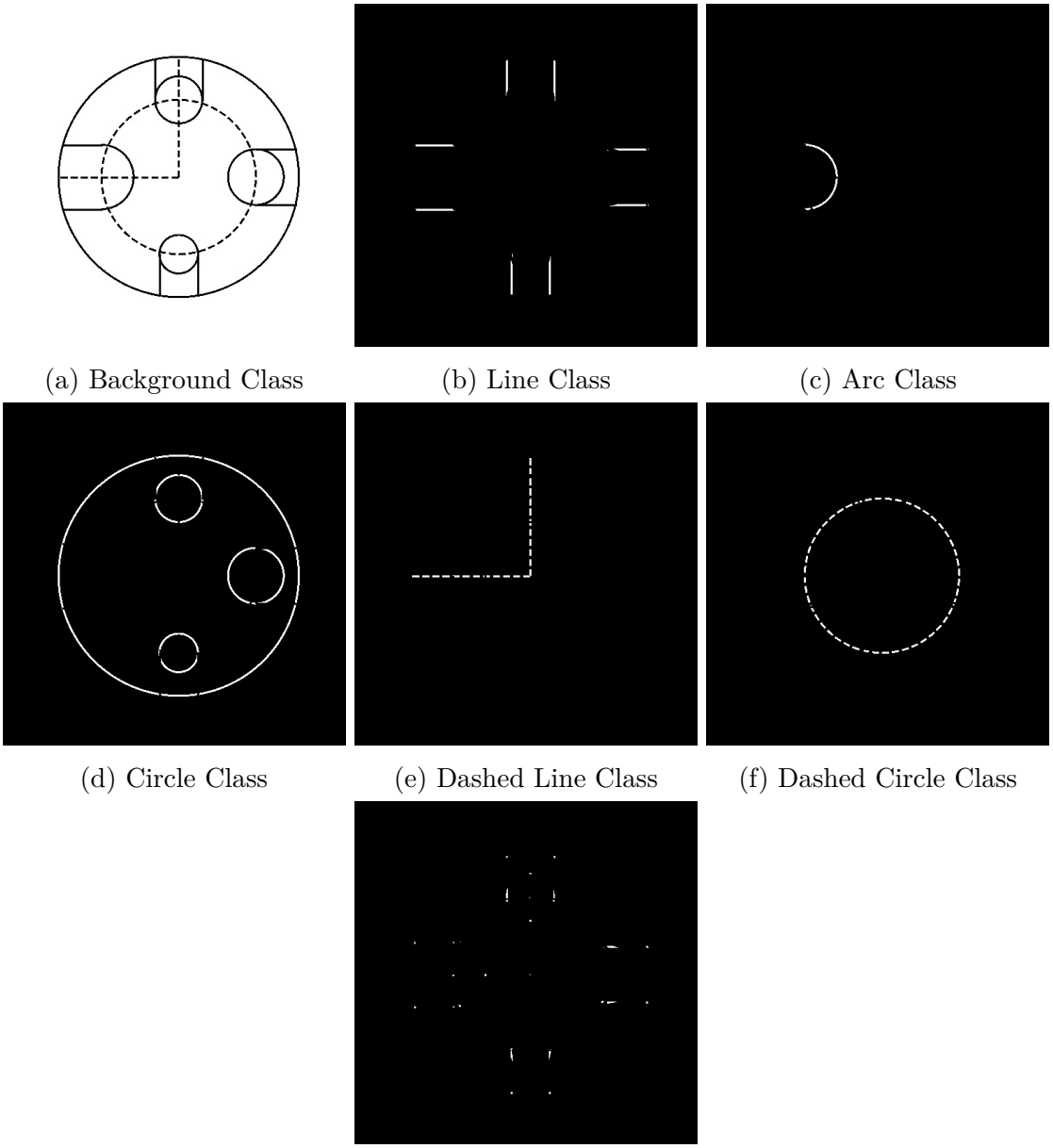
(h) Instance 7 - Line

(i) Instance 8 - Line

(j) Instance 9 - Line

(k) Instance 10 - Line

Figure 4.6.8: Figure (a) represents the generated sketch and figures (b) to (k) represent the instances that belong to figure (a)

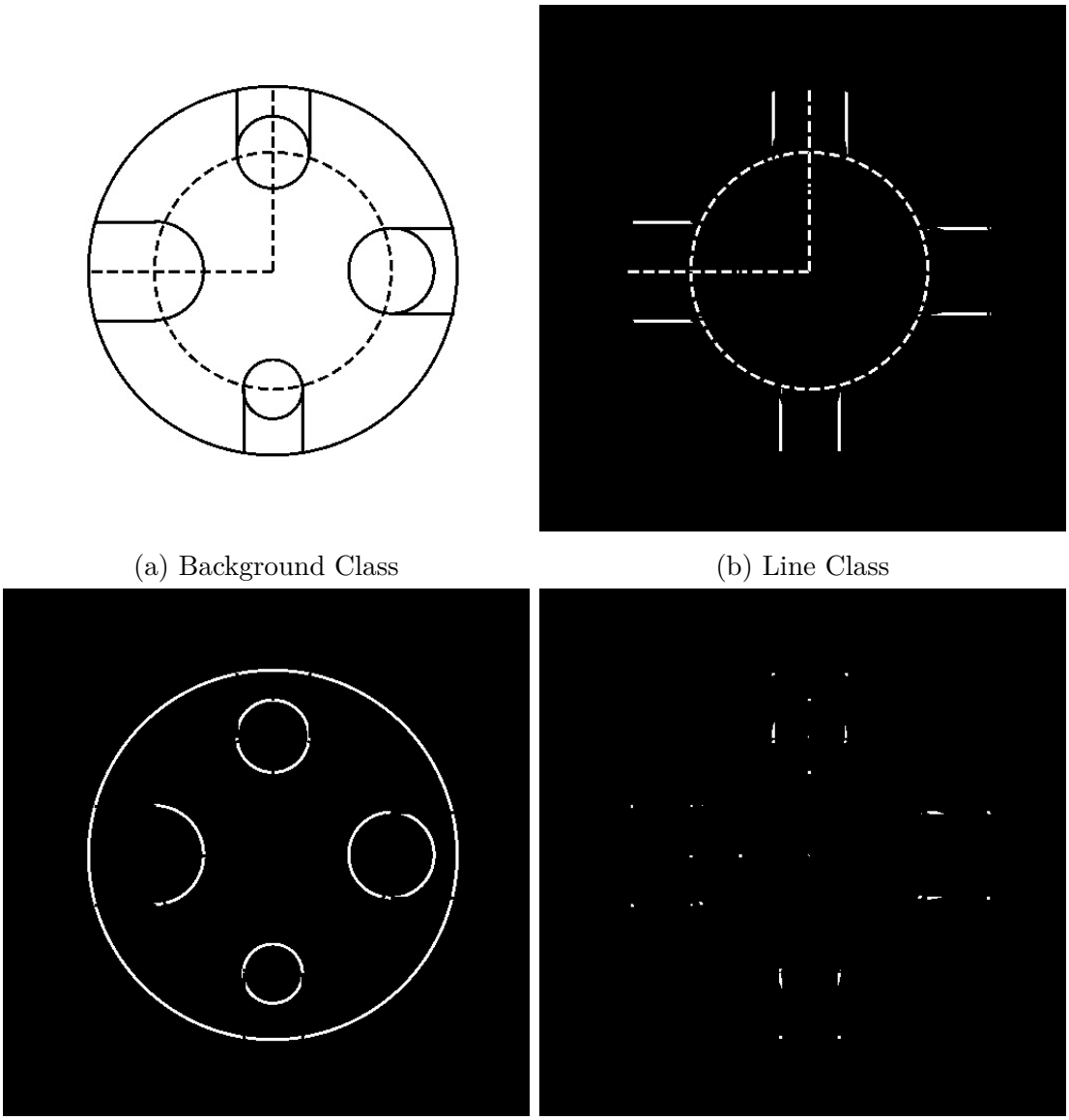


(a) Background Class (b) Line Class (c) Arc Class

(d) Circle Class (e) Dashed Line Class (f) Dashed Circle Class

(g) Intersection Class

Figure 4.6.9: Segmentation masks used in scenario 1 (eight classes)



(a) Background Class

(b) Line Class

(c) Circle/Arc Class

(d) Intersection Class

Figure 4.6.10: Segmentation masks used in scenario 2 (four classes)

ing box area, the list of coordinates of the segmentation mask (polygon), and the boolean variable is-crowd, which is always 0 in our case (0 if the masked area contains only one object and one otherwise). In addition to those data, we also have the image id and category id to which the instance belongs.

The ideas behind using three scenarios are the following:

1. Scenario 3 is a dataset that is created to be used for instance segmentation and not for semantics segmentation. In instance segmentation, the algorithm detects the object and then labels it. Thus, instance segmentation does not label all pixels in the image, unlike semantic segmentation, where each pixel in the image is labeled. To summarize, the main aim behind the scenario 3 dataset is to compare the performance of the instance segmentation model with semantic segmentation models.

2. Datasets of scenario 1 and scenario 2 is created to be used for semantic segmentation. The difference between them is the number of labels. Scenario 1 has eight labels, whereas scenario 2 has only four labels. The main idea behind creating those two datasets is to study the performance of semantic segmentation models with different numbers of labels.

4.6.1.2 Segmentation Phase

In this experiment, we train and evaluate eleven segmentation models, where ten are semantic segmentation models, and one is instance segmentation models, as detailed below. These experiments aim to find the best way to separate layers and facilitates the vectorization process. Thus, instead of having an input image with different types of primitives, we will have multi-layers for one image where each layer contains only one type of primitives. For example, by applying the segmentation using the scenario 2 dataset, each input will generate four layers: the background layer, the straight line layer, the circle/arc layer, and the intersection layer. Therefore, instead of vectorizing the input image directly, we vectorize the straight line layer and circle/arc layer separately, then combine them to obtain a fully vectorized image. Using this strategy, we decrease the complexity of the vectorization problem.

In scenarios 1 and 2, we train the same models with differently labeled data and compare them. The models are based on U-net and SegNet architecture. We trained five models:

1. Unet Basic (refer to section 2.3.1)

2. Unet with VGG16 as a base model

3. Unet with ResNet50 as a base model

4. SegNet Basic

5. SegNet with ResNet50 as a base model

These models are trained using Adam optimizer, a batch size equal to four, 100 maximum epochs, 2500 steps per epoch, a learning rate equal to 0.001, and L2 regularization. The loss between ground truth and the predicted label is computed using categorical cross-entropy loss. The categorical cross-entropy loss is one of the most commonly used loss functions for image segmentation and it can be written as follows:

$$CCE = -\frac{1}{N}\sum_{i=0}^{N}\sum_{j=0}^{J} y_j \cdot \log\left(\hat{y}_j\right) + \left(1 - y_j\right) \cdot \log\left(1 - \hat{y}_j\right) \tag{4.2}$$

where N is the total number of pixels, J is the number of classes, y is the ground truth label and $\hat{y}$ is the predicted label.

Besides, in scenario 3 we use the Mask-RCNN model with the PointRend module. The Mask-RCNN is one of the most important models used for instance segmentation, and the PointRend module helps increase the accuracy of border detection that is important in our case. The training and evaluation are done using Detectron2 PointRend implementation with the configuration "pointrend_rcnn_X_101_32x8d_FPN_3x_coco". In the following section we are going to discuss the results of different scenarios.

4.6.1.3 Segmentation Results and Discussion

In this section, we discuss the results of the segmentation with different architectures using the three scenarios exposed in section 4.6.1.1.

First, we compute the normalized confusion matrix for 2000 test images with different architectures. Figure 4.6.11 represents the confusion matrix of the five trained architectures. Figure 4.6.11a represents the confusion matrix of the Resnet50 SegNet networks and shows that the accuracy of detecting background pixels is almost 100%, which is logical due to the type of images where the background is almost all white pixels. Moreover, the accuracy of line and dashed lines are 96% and 94%, respectively, which shows the high performance of the architecture in terms of detecting lines and dashed lines. Nevertheless, the accuracy of detecting arcs is almost 85%, and the accuracy of detecting circles is 64%. Thus, the confusion matrix shows that almost 27% of pixels that should be classified as circles are classified as arcs, which shows that the trained network is confused between circles and arcs. This confusion is caused by the high similarity between an arc and a circle knowing that the intersection class makes them more similar by the discontinuation of circles. Also, the accuracy of detecting dashed arcs is very low, and the dashed arcs are classified as dashed lines and dashed circles. The accuracy of detecting dashed circles is better than the accuracy of detecting dashed arcs; however, almost 15% of dashed circle pixels are labeled as dashed lines.Finally, the accuracy of detecting intersection pixels is low, and it is reasonable because this class is the pixels that belong to two or more

classes. Thus the confusion in labeling intersection pixels does not necessarily mean that the trained network is wrong, but it may mean that those pixels are labels to one of the classes.

On the same hand, other confusion matrices in figure 4.6.11a except figure 4.6.11c show that almost all the architectures fail to label dashed arcs and circles correctly and confuse between arcs and circles detection. However, the figure 4.6.11c shows that the trained network is not stable, and the accuracy of detection is low compared to other networks.

The confusion matrix does not reflect the overall performance of the architectures; thus table 4.6.1 shows the IoU metrics for each class (circles, arcs, ...). The table shows that the IoU of the background class in all trained networks is highly accurate. Nevertheless, the line class reaches an IoU greater than 90% in all architectures except SegNet. In addition, the IoU score of the dashed lines class is acceptable in all U-net based architectures and not sufficient in SegNet based architectures. We notice that the IoU of dashed arcs is almost bad, and the reason is clear from the previous paragraph. The Resnet U-net architecture outperforms other trained architectures in terms of average IoU, which is the addition of all IoU for each class over the number of classes. The weighted IoU considers the number of pixels in each class; thus, it is high in almost all architectures because the number of pixels of the background class is higher than the number of pixels of other classes, and the IoU of the background class is high in all architectures. To summarize, the best architecture in terms of the IoU metric is the resnet U-net architecture, which reaches an average of 66% of accuracy. This average is not satisfactory to say that the trained networks can separate classes, and therefore we proposed another scenario by merging confused classes while preserving the ability to benefit from this segmentation in the vectorization process.

Trying to solve the issues that lead to a non-satisfactory accuracy (in terms of separating layers which will affect the vectorization process) of trained networks in scenario 1, we propose scenario 2 labeling protocol which merges confusing classes. Based on scenario 1 results, we combine the circle and arc classes into one class, lines, dashed lines, dashed circle, and dashed arcs classes into one class. In addition to the intersection and background classes which have the same task as the previous scenario.

Similarly to scenario 1, we compute the confusion matrix of five different architectures as shown in figure 4.6.12. All confusion matrices in figure 4.6.12 have almost the same percentage of accuracy for the three main classes (background, line, circle/arc) except the SegNet architecture where the accuracy of detecting circle/arc pixels decrease from almost 95% in other architectures to 70%.

To study the behavior more accurately, we refer to the table 4.6.2 which contains the IoU of each class. The table shows the architecture based on U-net outperforms the architectures based on SegNet, and the U-net based architectures have almost the same average

	unet	vgg unet	resnet unet		segnet	resnet segnet
Background	0.99	0.99	0.99		0.99	0.99
Line	0.93	0.95	0.94		0.85	0.91
Arc	0.55	0.43	0.65		0.4	0.57
Circle	0.58	0.56	0.73		0.38	0.59
Dashed Line	0.87	0.86	0.86		0.58	0.67
Dashed Circle	0.51	0.83	0.72		0.55	0.63
Dashed Arc	0.04	0.01	0.079		0	0.07
Intersection	0.32	0.39	0.31		0.21	0.24
Average	0.6	0.63	**0.66**		0.5	0.59

Table 4.6.1: IoU metric of scenario 1

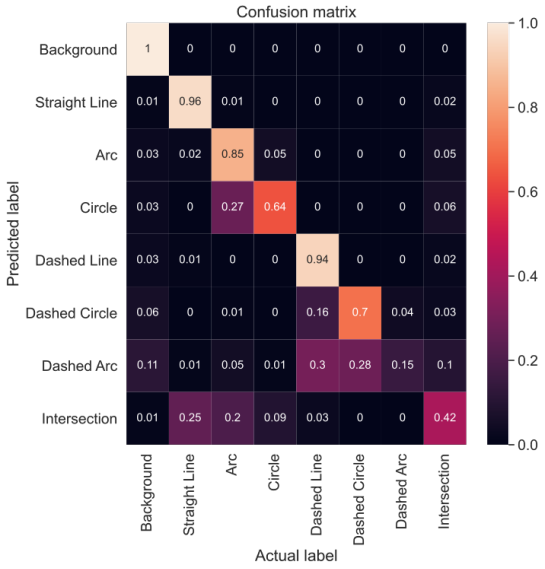
	unet	vgg unet	resnet unet		segnet	resnet segnet
Background	0.99	0.99	0.99		0.99	0.99
Line	0.95	0.95	0.92		0.78	0.89
Circle/Arc	0.94	0.93	0.91		0.67	0.87
intersection	0.39	0.36	0.38		0.16	0.28
Average	**0.82**	0.81	0.8		0.65	0.76

Table 4.6.2: IoU metric of scenario 2

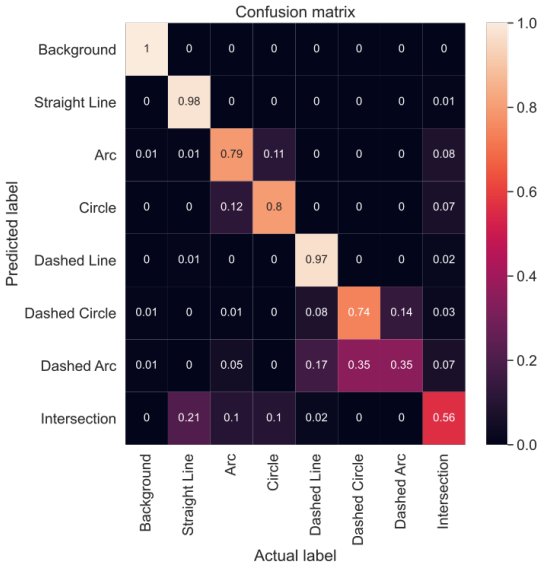
IoU score, which is almost 81% of accuracy. This average shows that the trained network can correctly separate classes because those architectures can separate main classes with an IoU score above 90%. However, as mentioned previously, the intersection class IoU score does not reflect the network performance.

Figure 4.6.13 presents the behavior of loss in terms of epochs of the trained networks in the first and second scenarios. We can notice that each architecture in both scenarios is trained with a different number of epochs because we used the early stopping criteria while training which prevent from the over fitting problem. This criterion will stop the training process when the loss gets worth while increasing the number of epochs. In other words, if the loss keeps increasing for a certain number (in our case 5) of epochs, then the network stops training and saves the network with the minimum loss. In both scenarios 1 and 2, the U-net and Resnet U-net reach the lowest loss value with almost 55 epochs.

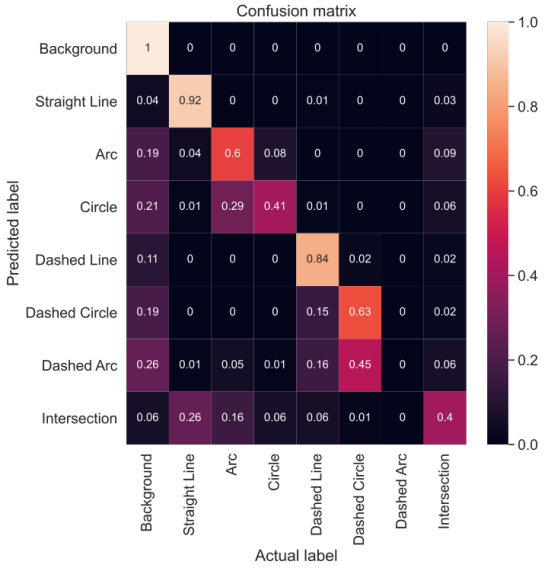
To be able to compare scenario 3 (instance segmentation) with scenarios 1 and 2 (semantic segmentation), we convert the instance segmentation masks in scenario 3 into semantic segmentation masks. To do so, all the instance masks that belong to the same class are converted to the same color. We firstly convert them into eight classes to compare



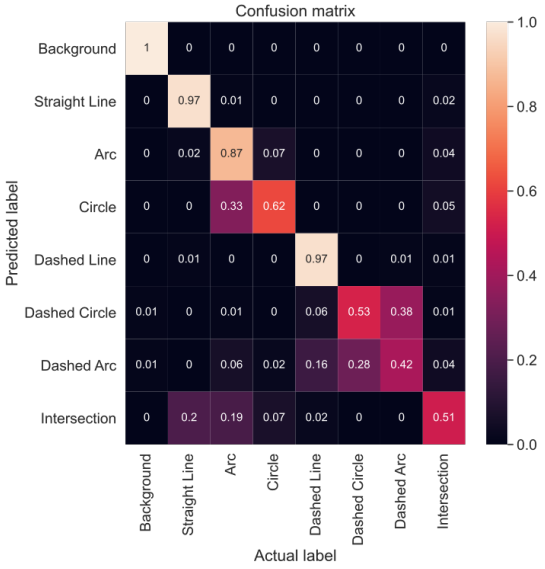
(a) Confusion Matrix - Resnet50 Segent architecture



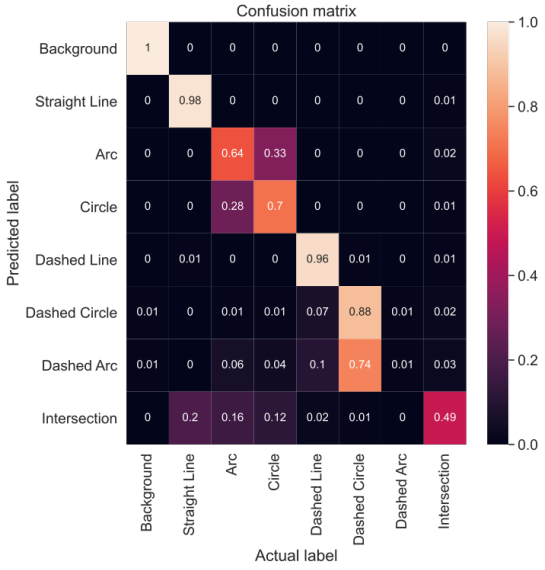
(b) Confusion Matrix - Resnet50 Unet architecture



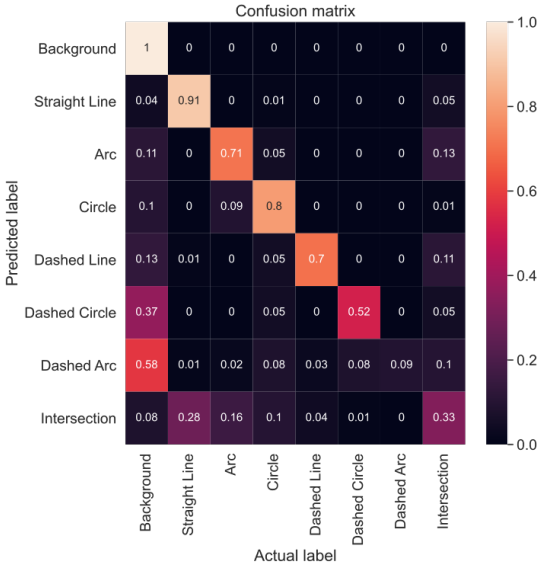
(c) Confusion Matrix - Segent architecture



(d) Confusion Matrix - Unet architecture

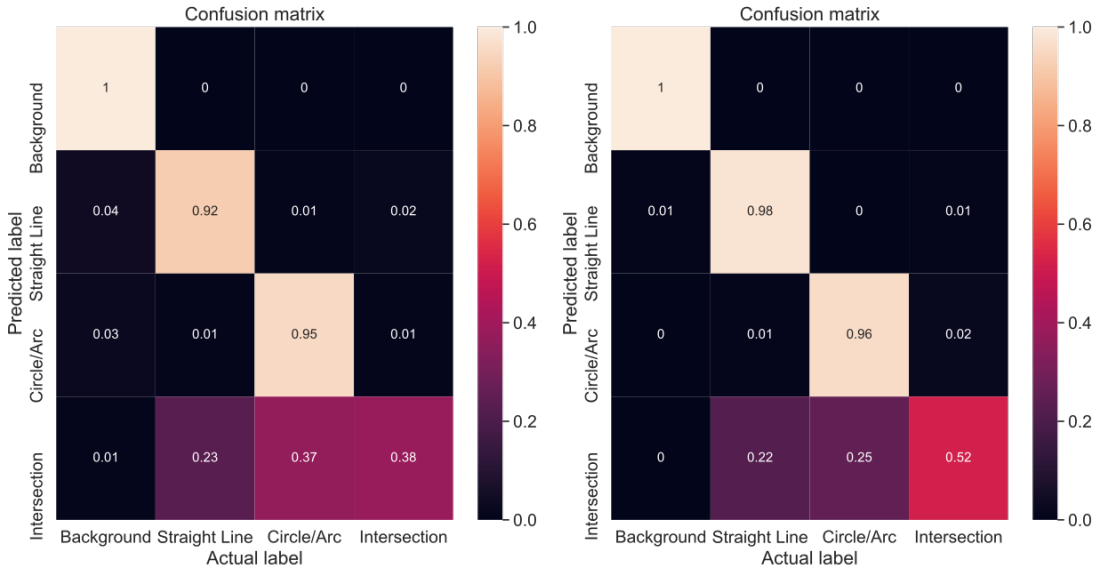


(e) Confusion Matrix - VGG Unet architecture



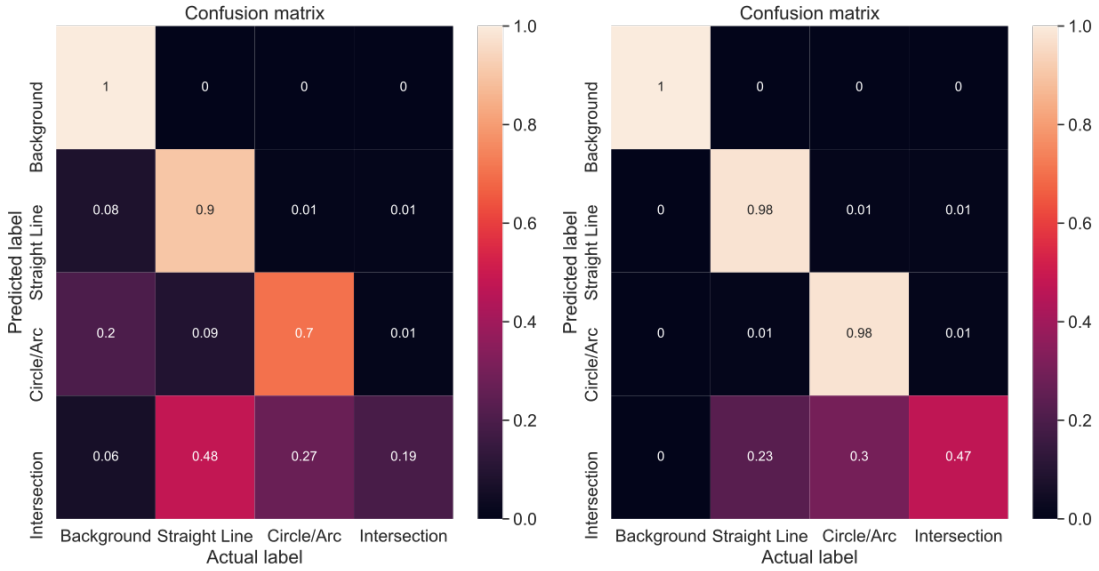
(f) Confusion Matrix - MaskRcnn architecture

Figure 4.6.11: Normalized confusion matrices of scenario 1



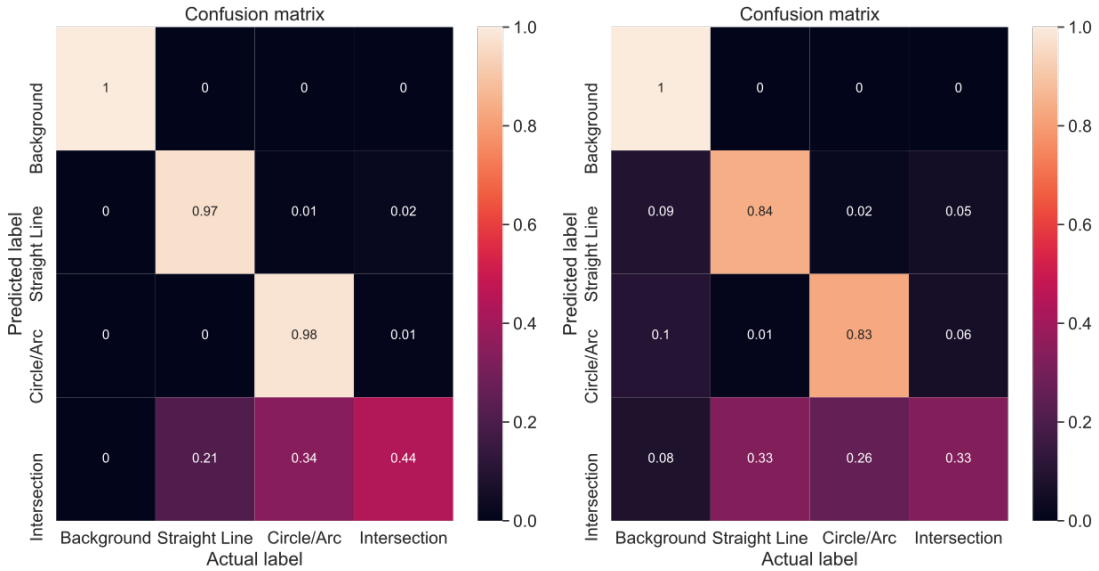
(a) Confusion Matrix - Resnet50 Segent architecture

(b) Confusion Matrix - Resnet50 Unet architecture



(c) Confusion Matrix - Segent architecture

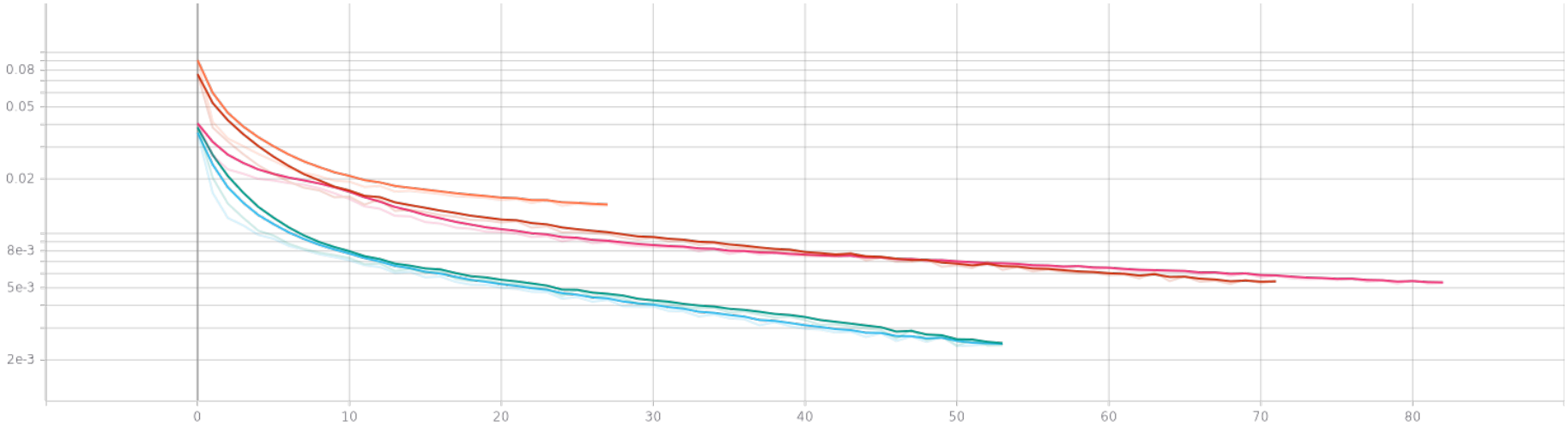
(d) Confusion Matrix - Unet architecture



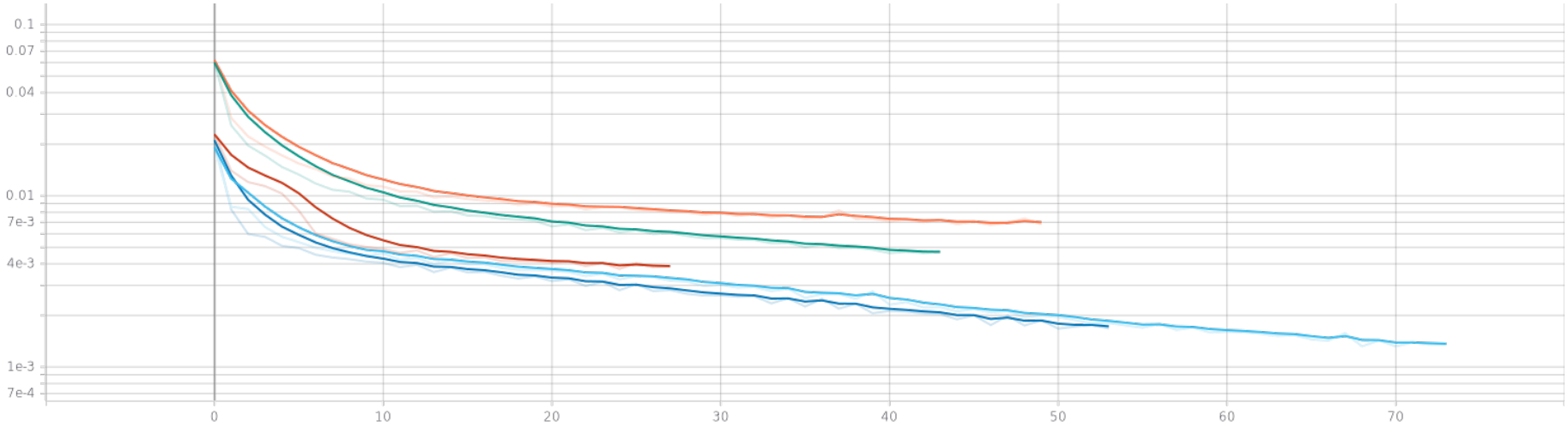
(e) Confusion Matrix - VGG Unet architecture

(f) Confusion Matrix - MaskRcnn architecture

Figure 4.6.12: Normalized confusion matrices of scenario 2



(a) Losses of different architectures in terms of epochs for scenario 1



(b) Losses of different architectures in terms of epochs for scenario 2

Figure 4.6.13: Losses of different architectures

with scenario 1, and then we convert them into four classes and compare with scenario 2. Figure 4.6.11f and Figure 4.6.12f shows the confusion matrices of converted masks respectively for 8 classes and 4 classes. The confusion matrix in figure 4.6.11f shows that the method used in scenario 3 outperforms other methods in terms of detecting circles, but it is less accurate in detecting dashed entities. Moreover, the confusion matrix in figure 4.6.12f shows that the trained model has competitive results compared to other results, but still, the U-net model outperforms other methods.

To summarize, based on results, the models trained with eight labels (scenario 1 and scenario 3 - 8 Labels) are not sufficient to separate the layers. In contrast, the models trained with four classes (scenario 1 and scenario 3 - 4 Labels) reach better results with high accuracy, which is sufficient to say that the trained model is able to separate the layers. The best model based on IoU metrics shows in table 4.6.2 is the U-net model. Thus, the U-net model (4 classes) will be used in the next stage to separate layers before vectorizing.

4.6.2 Detection Phase

The proposed method is based on deep learning to vectorize drawing, the aim of this method is to reduce the complexity of the vectorization problem by separating layers (as detailed bellow) and then detecting lines and circles in each layer separately. The overview of the algorithm is shown in figure 4.6.14. The input image is segmented using the trained U-net architecture, and the output image is separated based on colors to generate four layers: the background layer, Circle/arc layer, straight lines layer, and intersection layer. The intersection layer is added to straight lines and circle/arc layers because the intersections of primitive reduce fragmentation. As shown in figure 4.6.14, a circle detection algorithm processes the merged circle/arc layer and intersection layer, and a line detection algorithm processes the merged straight line layer and intersection layer. Both segmentation and detection algorithms are then merged to generate the final vector file.

The circle detection algorithm and line detection algorithm used in this stage is a modified version of the algorithm presented in chapter 3 such that it detects only lines for line detection and only circles/arc for circle detection. Nevertheless, we can use other algorithms such as hough lines and hough circles or the RANdom SAmple Consensus (RANSAC) algorithm.

4.6.3 Hybrid Vectorization Results

As we used the same algorithm used in the previous chapter to detect line and circles, we do not expect a high difference in terms of accuracy. To evaluate the hybrid algorithm, we use the GREC datasets used in chapter 3. we evaluate only images of size smaller than 512*512 by adding a white border. Nevertheless, images with a size larger than (512

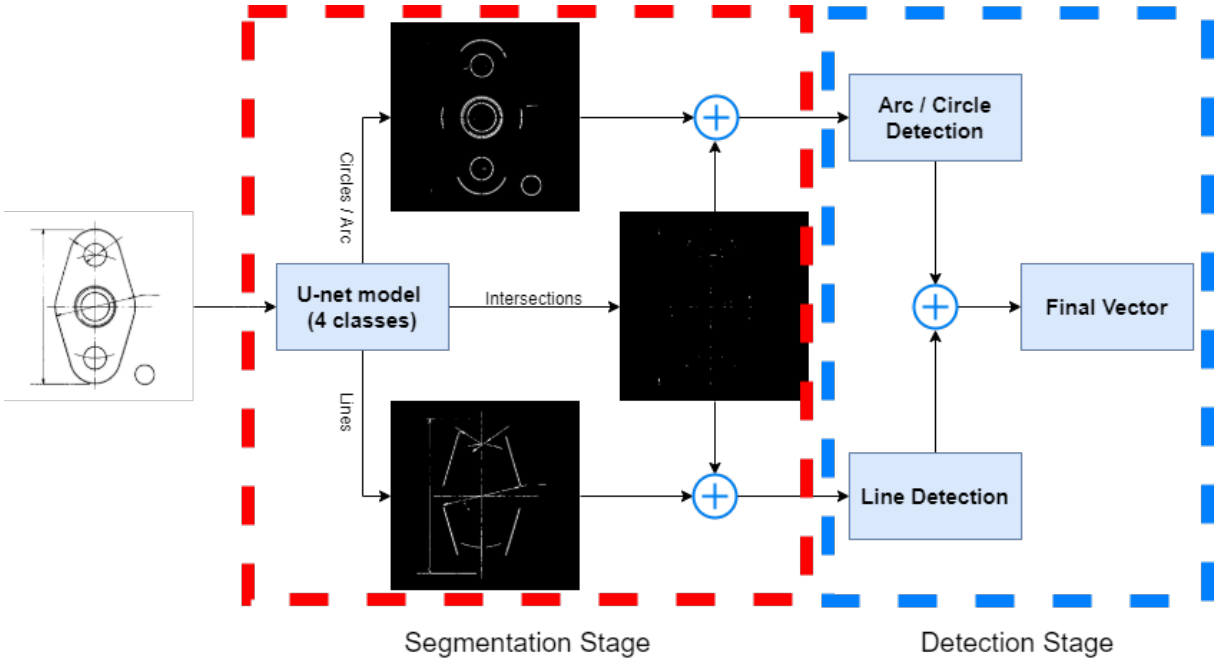


Figure 4.6.14: Vectoriztion algorithm overview

x 512) can be decomposed into patches of (512 x 512), segment each patch separately, and rejoin them. Assume we have an image of size (1024 x 1024), we decomposed into 4 images of size (512 x 512), segment each image separately and we rejoin the 4 images of size (512 x 512) into 1 image of (1024 x 1024).

Figure 4.6.15 , shows the VRI results of different GREC images with the same settings of the algorithm tested in chapter 3. Overall the proposed method in chapter 3 outperforms this method. However, we have two critical cases to study, which are image 2 and image 8. For image 2, we can see that the hybrid method outperforms the unsupervised method due to the correct segmentation of small primitives, which are detected as arcs in the unsupervised method and detected correctly as straight lines due to the segmentation stage in the hybrid method. Nevertheless, the unsupervised method highly outperforms the hybrid method for image 8 because the hybrid algorithm did not segment the large arc correctly and fragment it into small lines.

We also compare the hybrid method with the unsupervised method using the intersection over union metric and the data generated for testing the hybrid algorithm (scenario 2). In this experiment, we set both the number of chromosomes and number of generation to 10, and we select $\alpha = \lceil \frac{w}{2} \rceil$ as stated from the results of chapter 3. We can see from figure 4.6.16 that both algorithms work perfectly and reach an average IoU equal to 84%. However, moving to figure 4.6.17, we can see that the hybrid algorithm is twice faster than the unsupervised method. As we separate layers in the hybrid method, we can detect lines and circles in parallel, which is impossible for the unsupervised method. Some results are shown in figures 4.6.18, 4.6.19, and 4.6.20.

It is clear that the segmentation layer highly affects the vectorization output. The

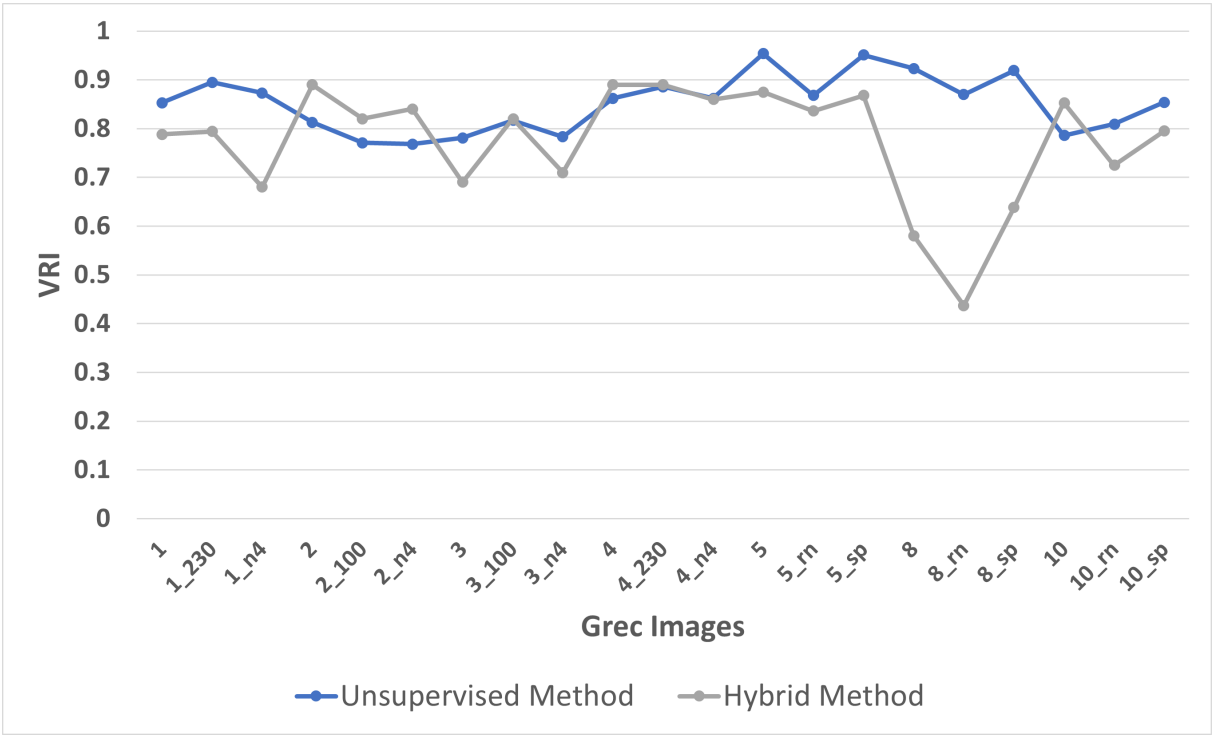


Figure 4.6.15: The VRI comparison between the unsupervised method and the hybrid method

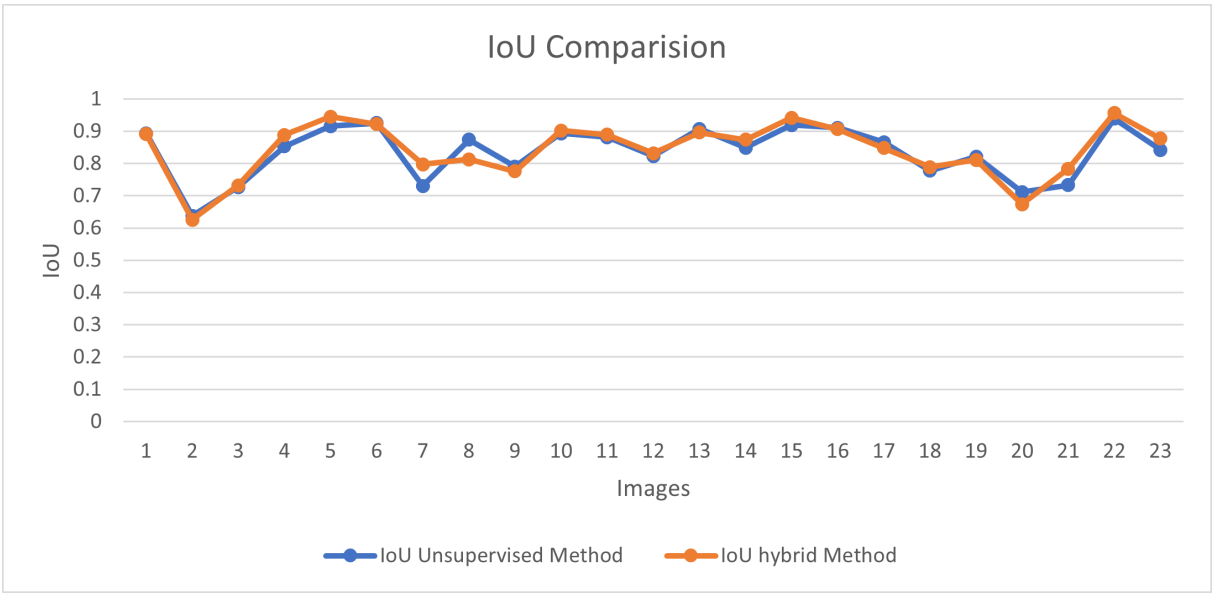


Figure 4.6.16: The intersection over union (IoU) comparison between the unsupervised method and the hybrid method

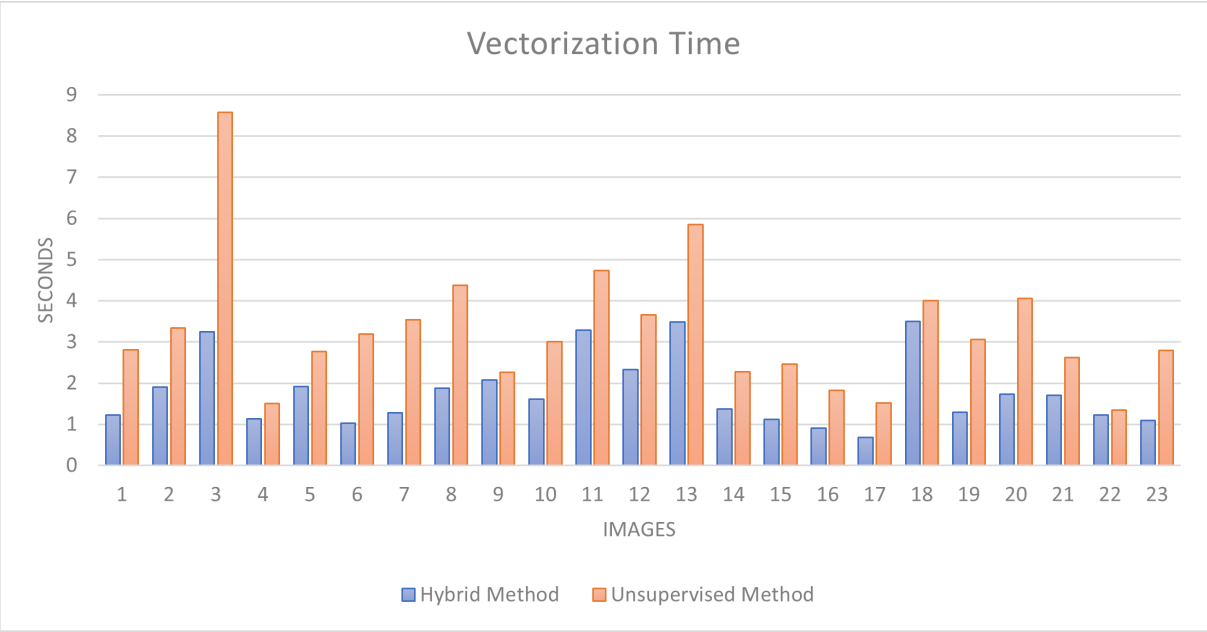


Figure 4.6.17: The running time comparison between the unsupervised method and the hybrid method

trained model segments synthetic data generated using the same method accurately. However, while testing the model with other data such as GREC images, the model has less accuracy due to different causes:

1. GREC images are more complicated than the data used for training. In GREC images, there are more intersections between primitives. In the results, we notice that the intersection results play a major role in preventing primitives from fragmenting. This is the major cause that decreases the accuracy of evaluation on GREC images. For example, assume we are vectorizing only the figure 4.6.10c, this leads to detect circles as set of arcs because they are not connected. Nevertheless, by merging figure 4.6.10c and 4.6.10d, we are able to detect the circles correctly without fragmentation.

2. The maximum line width used while training is not sufficient. While generating the data, we put a threshold to the line width to guarantee the generation of good drawings with primitives overlapping. This affects the segmentation on the GREC dataset, which has variant width size.

3. Generating the training data of the same shape and type. All the generated data have the same width and height, and this problem highly affects the segmentation of data from different shapes.

We mainly have different solutions for these three problems, such that:

1. Data augmentation: we use a data augmentation algorithm that adds different noises to the image, such as salt and pepper, rotation, blurring, or scaling. Moreover, the

second idea is to merge randomly two images to increase the number of primitives and their intersections.

2. The second idea is to generate images with more different line widths. Besides, we can also generate data of different shapes and rescaled during the training to increase the accuracy of segmentation for data from different shapes.

To summarize, the proposed method shows promising results in terms of intersection over union and VRI. In addition, the main advantage is the running time which is twice faster when both the number of chromosomes and generation are fixed to 10. This gap in running time is more significant when we increase the number of chromosomes and generation. Nevertheless, the main drawback of this algorithm is the limited size of the image, which is 512x512. Even if we can crop large images into different blocks of 512x512, however, this may cause fragmentation during the joining process of different blocks. Moreover, downscaling large images into 512x512 lead to lose important data due to the type of engineering drawings that may have close primitives.

4.7 3D reconstruction

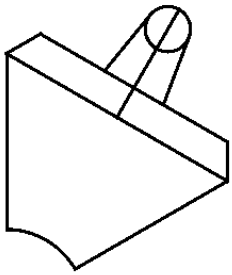
The last section aims to study the ability of the deep learning model to learn and generate the 3D models directly from 2D raster views, unlike the two previous (unsupervised and hybrid) approaches that first extract primitives.

The deep learning method has been widely used for 3D reconstruction from single or multiple 2D images. However, 3D data can be represented in different formats where each representation has its advantages and disadvantages.

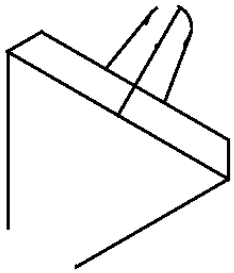
The first representation is the rasterized form called voxel grids. This representation is the expansion of spatial grid pixels into volume grid voxels. With this representation, the CNN can be directly applied; however, this representation needs many resources to produce high-resolution shapes. In other words, this method can cause the loss of details of 3D objects due to the vast memory required.

The second representation is the geometric forms such as meshes and point clouds. The mesh representation is a set of vertices, edges, and faces representing a surface of a 3D object. Nevertheless, point clouds are a list of points defined by 3D coordinates (x,y,z) where these points represent a 3D object. By increasing the number of points, the 3D shape details increase. The main drawback of this representation is the inability to apply CNN directly; however, this representation focuses more on details of a 3D object.

Both representations (voxel/mesh) need post-processing to be used for VR and AR applications. Still, it is interesting to test the problem of 3D model reconstruction from different orthographic views using deep learning methods. The problem can be formulated as follows: we assume $I = k, k = 1, ..n$ a set of orthogonal views of an object O. The learning process aims to predict a function f that reconstructs a 3D solid O' as close as

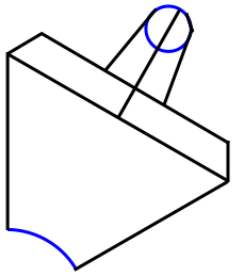


(a) Image 19

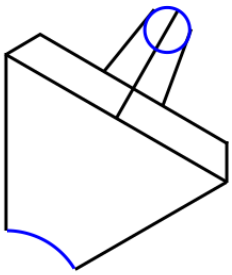


(b) Line Class - Segmentation Phase

(c) Circle/Arc Class - Segmentation Phase

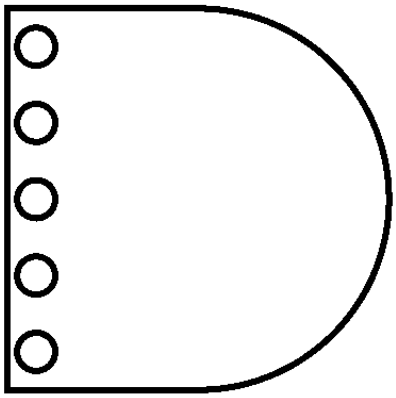


(d) Hybrid Vectorization

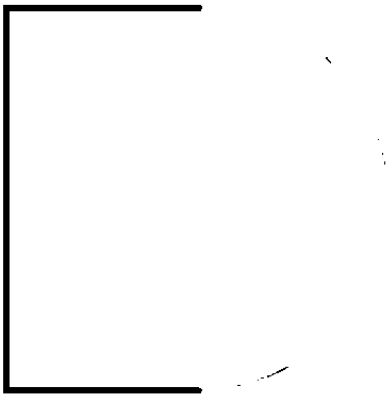


(e) Unsupervised Vectorization

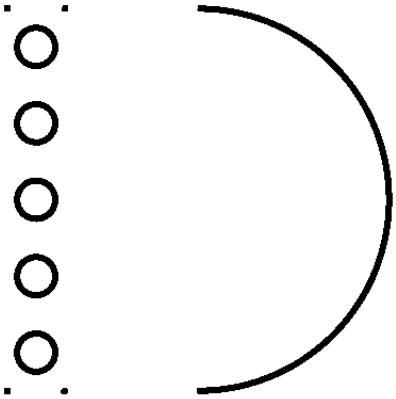
Figure 4.6.18: Hybrid and unsupervised results of image 19



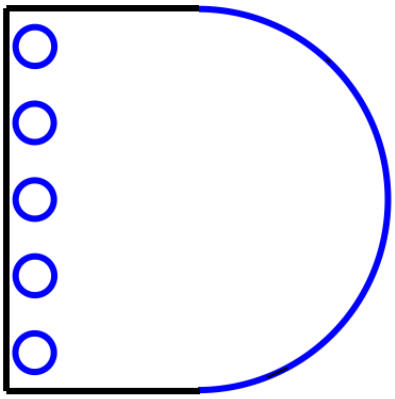
(a) Image 5



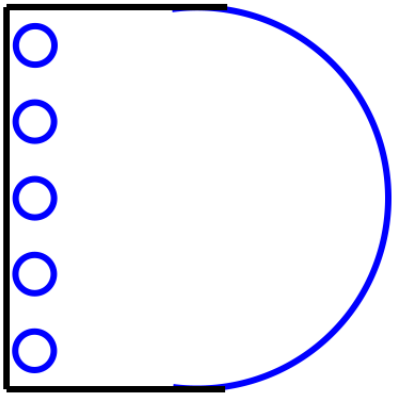
(b) Line Class - Segmentation Phase



(c) Circle/Arc Class - Segmentation Phase

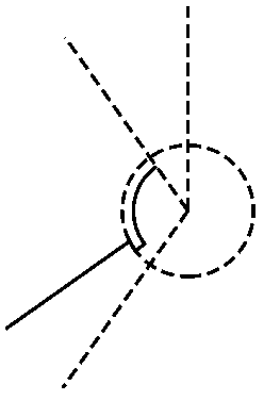


(d) Hybrid Vectorization

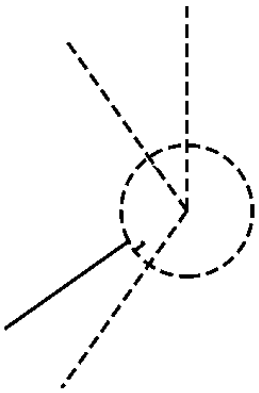


(e) Unsupervised Vectorization

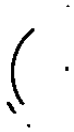
Figure 4.6.19: Hybrid and unsupervised results of image 5



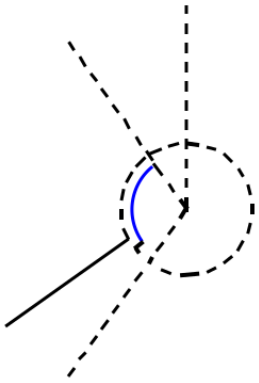
(a) Image 2



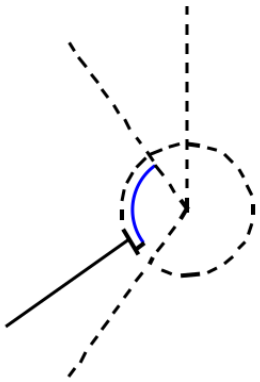
(b) Line Class - Segmentation Phase



(c) Circle/Arc Class - Segmentation Phase



(d) Hybrid Vectorization



(e) Unsupervised Vectorization

Figure 4.6.20: Hybrid and unsupervised results of image 2

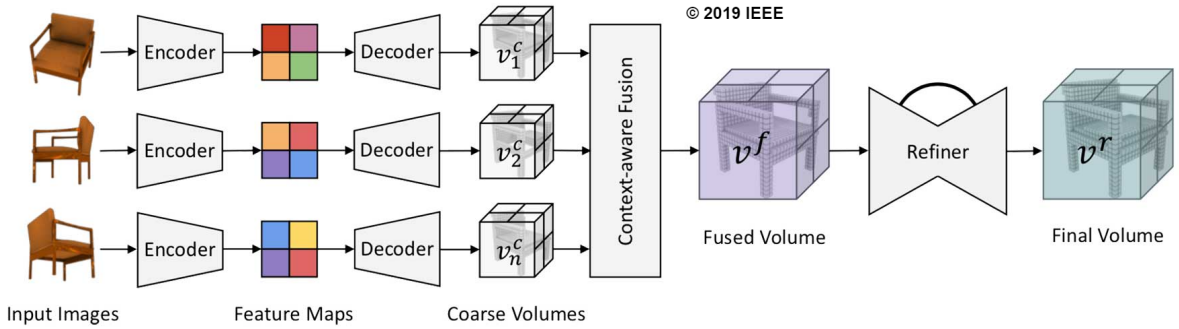


Figure 4.7.21: Pix2vox architecture [137]

possible to ground truth 3D model O. To summarize, we aim to predict a function f that minimize the loss function L such as $L(I) = d(f(I), O)$, where $O' = f(I)$ and $d(.,.)$ is the distance between the ground truth 3D model O and the predicted 3D model O'.

After defining the problem and defining the 3D data representation, we will test one of the most performant architectures in reconstructing 3D models from different views called pix2vox++ using orthographic views instead of natural images.

4.7.1 Pix2Vox Model

Pix2vox is a deep learning network proposed by *Xie et al.* [136] to reconstruct a 3D shape from one or more images. This method is based on voxel representation and outperforms other previous methods in terms of accuracy. The network consists of four main parts:

- The encoder which generates features maps from input images
- The decoder decodes the feature map and generates 3D volume for each input image.
- The context-aware fusion model fuses the output of decoders.
- The refiner module refines the 3D shape generated by the context-aware module and produces the final 3D shape.

To test this algorithm, we generate 2D orthographic view projection from 3D models in the ABC dataset. For each 3D object, we rotate it and generate top, front and side views using Blender scripting. Besides, we generate 32x32x32 voxels of the 3D object. The size of the generated images is 224x224. In figure 4.7.22, we present an example of the generated images and voxels. The generated dataset consists of 27233 voxel and three images with different views for each model. 21785 data are used for training, 2724 data are used for testing, and 2724 data are used for validation.

We train the network using the same configuration used on the original paper. Figure 4.7.23a shows the encoder and decoder loss and Figure 4.7.23b shows the Refiner loss. The average Intersection over union ratio reaches 65% as shown in figure 4.7.23c. The result

was expected for different reasons. One of them is the heterogeneity of the trained dataset, unlike other datasets that consist of one object such as shapenet-chair or shapenet-airplane datasets.

This experiment is done to study the possibility of going from 2D raster orthographic views to 3D models directly. The results were promising; however, such a deep learning model have many limitations:

1. The generated voxels need a post-processing stage to be smoothed and used in VR/AR applications.

2. In our case, we trained the model with 32*32*32 voxel size, which huge amount of details are lost. The trained model needs a massive amount of GPU memory to reconstruct 128*128*128 models, which can be considered acceptable in terms of details.

3. The ability of handling sectional views should be tested with this method.

4. The generalization of this method needs a huge number of data and resources.

Thus based on the previous four limitations, we believe that passing through the vectorization process before 3D reconstruction is required.

4.8 Summary

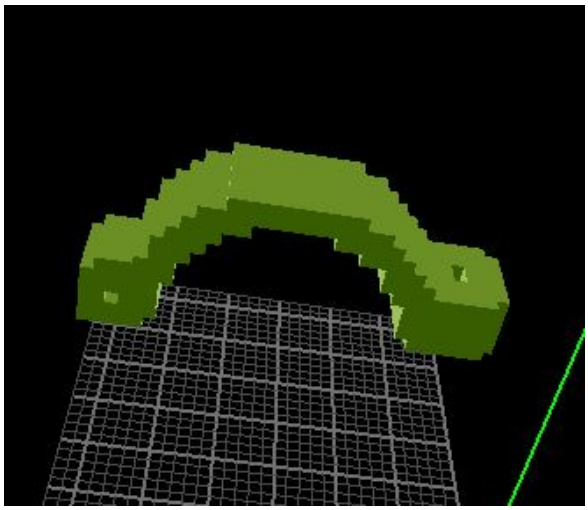
This chapter discusses the possibility of training a model to segment different primitives and generate different layers where each layer contains only one type of primitives. We test eleven different architectures with the same dataset and different labeling strategies. We found that the best model that is able to separate layers is the U-net model with four layers. This model treats an input image of size 512*512 and generates four different layers: the background layer, the straight lines layer, the circles/arc layer, and the intersection layer. We proposed a method to vectorize engineering drawing based on the trained model, where we generate the layers and fuse them as follow: First, the straight lines layer is fused with the intersection layer, and we detect straight lines. Second, the circles/arc layer is fused with the intersection layer, and we detect circles and arcs. Finally, both detected lines and circles/arc are fused to generate the final vector data. The method shows promising results; however, it has some limitations. The main limitation is the limited size of the input. Moreover, the segmentation model sometimes leads to fragment primitives, making it hard to recover them and detect them correctly.

Moreover, we test the ability to use a deep learning model to convert 2D raster orthographic views to 3D models without passing through the vectorization process. The trained model shows promising results, and the intersection over union ratio reaches 65%. Nevertheless, our final goal is to use the generated model for AR/VR applications. Thus,

(a) Rendered image view 1

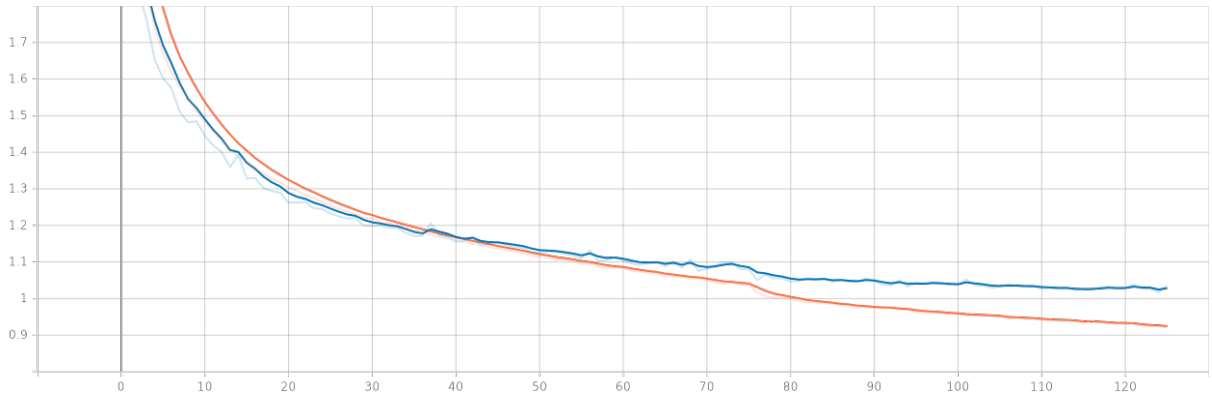
(b) Rendered image view 2

(c) Rendered image view 3

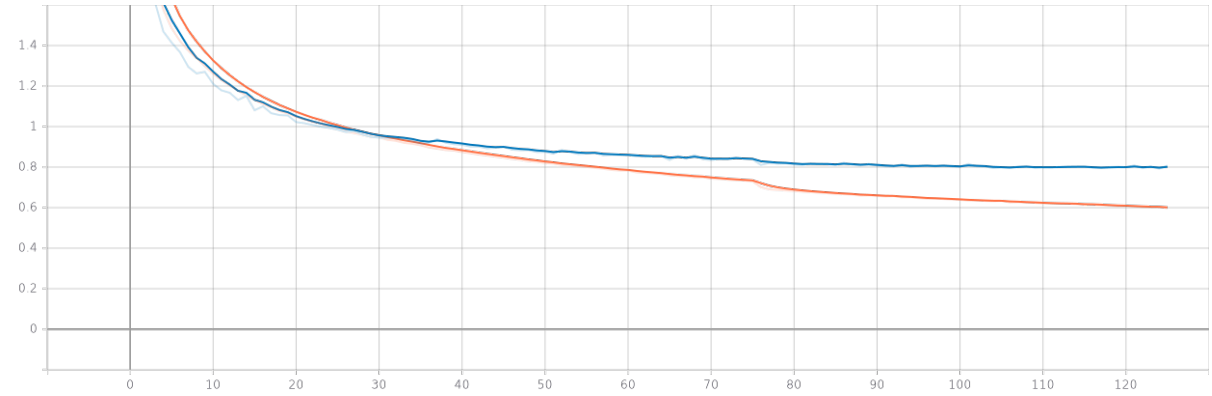


(d) Voxel representation

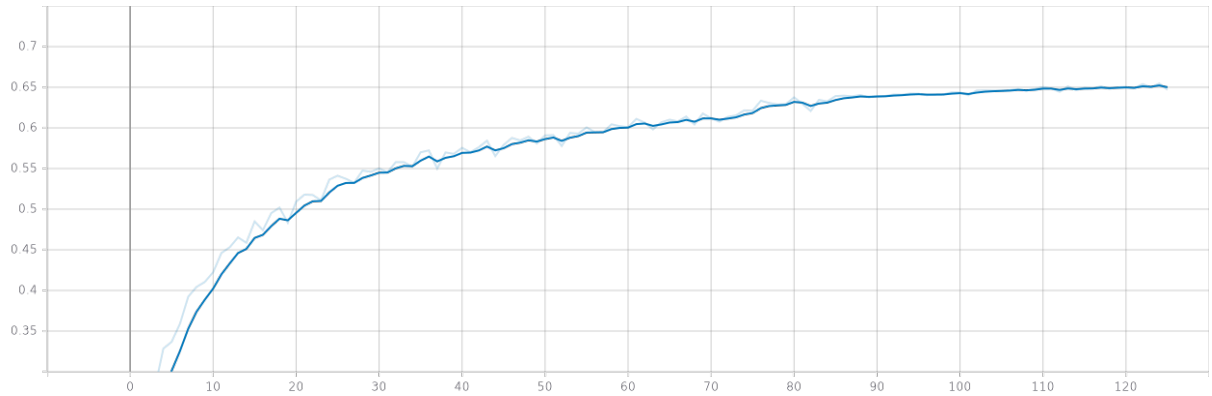
Figure 4.7.22: Example of rendered data



(a) Pix2Vox++ Encoder and decoder loss



(b) Pix2Vox++ Refiner loss



(c) Pix2Vox++ IoU

Figure 4.7.23: Evaluation of Pix2Vox model

the generated model is a Voxel, which needs much processing to generate a smooth model for AR/VR applications. Moreover, the voxel-based representation decreases the level of the details, in particular with low-resolution voxels, and high-resolution voxel models need a massive amount of GPU memories to train and test a model. Besides, we cannot guarantee that this type of learning can also handle sectional views.

Conclusions and perspectives

Conclusions:

This thesis work has been devoted to propose different vectorization methods that can be developed and used for 3D reconstruction from orthogonal views. The manuscript started with a brief introduction which explains the problem briefly. In each chapter of this thesis, we proposed a method to solve a problem.

Starting with the second chapter, we proposed a full framework to prepare engineering drawings for the vectorization process. The framework receives a raster engineering drawing template and treats it to remove border and cluster different views using DBSCAN cluster method. Next, each view is denoised using the U-net model, which is trained, tested, and compared with other methods using PSNR and DRD metrics. The trained network clean input images while preserving smooth edges that are important for the skeletonization process. The skeletonization is based on 3,4 chamfer distance; this process decreases the complexity of the problem from vectorization to curve segmentation. Finally, we propose a method for detecting arrowheads based on bag of visual words. The proposed method is tested with different images from different sources and with different resolutions and noise levels. The results show that the proposed method is accurate in detecting arrowheads. However, this method needs a new dataset to detect different types of arrows.

In the third chapter, we proposed a method based on a genetic algorithm. The input for this method is the skeletonized and labeled image from chapter 2. The algorithm extracts junctions pixels; we obtain a set of branches from the skeleton where each branch is a set of pixels. Each time we select a branch and use the pixels as chromosomes to generate different possibilities of primitives that fit this branch. The chromosomes consist of two different types: the straight line chromosomes and Circle arc chromosomes. The genetic algorithm detects the primitives that decrease the fitting error and increase the length (number of pixels that belong to the primitive). The algorithm continues to detect primitives until the largest branch becomes smaller than a certain threshold of pixels. The proposed method can detect only straight lines, circles, and arcs; however, the method can be extended to detect other types of primitives such as ellipse by adding

ellipse chromosomes and adding the parameters of the ellipse. The proposed method estimates the width of each branch and updates the parameter thresholds of the primitives. The proposed method is compared with different engineering drawing vectorization algorithms and outperforms them in terms of vector recovery index and visual results. Moreover, we compared the proposed algorithm with other types of vectorization, such as vectorization of hand drawings using the interaction over union index and the number of primitives needed to vectorize a raster image. The intersection over union metrics shows the precision of the vectorization model, and the number of primitives needed to represent a raster image shows the complexity of the vector output. The results show that our method is competitive with some methods and outperforms other methods. Besides the high computation complexity of the proposed method, the main drawback of our method is the high possibility of detecting a small corner as an arc. Moreover, the post-processing method proposed can handle only the intersection of two primitives, whereas in some cases, we have more than two primitives that intersect.

In the fourth chapter, we study the ability to separate primitives layers using different deep learning segmentation methods. We trained eleven different networks to segment engineering drawing and separate layers. Five of the eleven models are trained to separate the input engineering drawing into eight different layers, which are: the background layer, straight-line layer, arc layer, circle layer, dashed line layer, dashed circle layer, dashed arc layer, and intersection layer. We evaluate those five models using the intersection over union metrics, we tested the model using 2000 images, and the best model (Resnet U-net) reaches about 66% IoU. This percentage is considered low, and we cannot rely on it to separate layers. Next, we trained the same five models to separate the input engineering drawing into four layers: the background layer, straight-line layer, circle/arc layer, and intersection layer. We use the same metric and the same number of test images, and the best model (U-net) reaches an average of 82% IoU. This percentage is considered acceptable and able to separate layers. In addition, we trained Mask-Rcnn+ Pointrend network to evaluate the ability of separate instances. To have a fair comparison with other networks, we convert the instance segmentation results into semantic segmentation results. We convert them twice, first to compare with eight labels network and second to compare with four labels network. The results show that the best and most acceptable network is the U-net model with four labels. Next, we propose a method based on this network to vectorize drawings. The method starts by separating the input image into circle/arc layer, straight-line layer, and intersection layer. We merge the intersection layer with the two other layers, and then we detect straight lines and circles/arc. This method reduces the problem of vectorization from curve segmentation into line and arc detection. The main drawback of this method is the limited dimension of the input image, which is 512 * 512. In case of having an image with a smaller dimension, we can add white padding, and in case of having an image with a larger dimension, we can add padding and decompose it into patches of 512*512 segment and rejoin them. However, the probability of

fragmenting primitives increases when we decompose input into patches because a patch might pass through a circle or an arc. In addition, we test the ability to reconstruct 3D models directly from 2D images based on deep learning. We use the pix2vox model, which one of the most performing algorithms for this task. The results were not good enough as expected due to many causes. One of the main causes is that this kind of algorithm is trained for one kind of object, such as chairs and airplanes; however, engineering drawing does not have strong shape similarities.

The main aim of proposing those methods is to use them to reconstruct 3D models and use them in VR/AR applications. Thus, in the next paragraph, we will list some perspective work that will help us reach our goal.

Perspectives:

As our goal is to reconstruct 3D models from orthographic views and based to the contributions done, we are going to propose several ideas for future work and further investigation:

Regarding proposed method in this thesis:

1. The detection arrowheads method needs more improvements such as detecting the arrowhead direction, detecting the dimension set, and separating them from the drawing.

2. The proposed method in chapter 3, can be extended by a preprocessing stage to treat more efficiently intersection between primitives. Moreover, we highly need an algorithm that classifies different entities such as dashed lines, dashed circles, dashed arcs, hashed areas, and centerlines.

3. The hybrid method proposed in chapter 4 can be improved by augmenting data and training the models with more complex drawings. In addition, the proposed method can be extended to the first segment dashed and hashed areas, segment dimension sets, and then segment solid primitives.

Regarding the final goal:

1. To the best of our knowledge, the proposed 3D reconstruction algorithms need a perfect vector input. We suggest studying the ability to propose an unsupervised 3D reconstruction algorithm that accept imperfect input drawings, facilitating the possibility of building 3D models using vectorized engineering drawings.

2. Referring to the method proposed by *Zhou et al.* [138] that detect lines using a fully supervised method, we suggest extending this method to vectorize other types of primitives and adapted to the drawing vectorization.

3. We suggest building a model based on deep learning to convert vectorized data into a 3D model.

4. We already study the ability to reconstruct 3D models directly from raster orthographic views. However, it is intersecting to project views into a voxel cube and study the ability to complete the missing voxels. Unlike pix2vox, which reconstructs 3D models from each view and then merges them, we propose to train a model that learns how to map pixels in the same voxel grid to build a 3D model. This way, the model can benefit from different views at the same time to learn.

Bibliography

[1] "Tamron's drafting department in the late 1970s." [Online]. Available: https://rarehistoricalphotos.com/life-before-autocad-1950-1980

[2] I. E. Sutherland, "Sketchpad a man-machine graphical communication system," *Simulation*, vol. 2, no. 5, pp. R–3, 1964.

[3] M. IDESAWA, "A system to generate a solid figure from three view," *Bulletin of JSME*, vol. 16, no. 92, pp. 216–225, 1973.

[4] G. Markowsky and M. A. Wesley, "Fleshing out wire frames," *IBM Journal of Research and Development*, vol. 24, no. 5, pp. 582–597, 1980.

[5] M. A. Wesley and G. Markowsky, "Fleshing out projections," *IBM Journal of Research and Development*, vol. 25, no. 6, pp. 934–954, 1981.

[6] H. Sakurai and D. C. Gossard, "Solid model input through orthographic views," *ACM SIGGRAPH Computer Graphics*, vol. 17, no. 3, pp. 243–252, 1983.

[7] Q.-W. Yan, C. P. Chen, and Z. Tang, "Efficient algorithm for the reconstruction of 3d objects from orthographic projections," *Computer-Aided Design*, vol. 26, no. 9, pp. 699–717, 1994.

[8] B.-S. Shin and Y. G. Shin, "Fast 3d solid model reconstruction from orthographic views," *Computer-Aided Design*, vol. 30, no. 1, pp. 63–76, 1998.

[9] M. Kuo, "Reconstruction of quadric surface solids from three-view engineering drawings," *Computer-Aided Design*, vol. 30, no. 7, pp. 517–527, 1998.

[10] S. S. Shum, W. Lau, M. M. Yuen, and K.-M. Yu, "Solid reconstruction from orthographic views using 2-stage extrusion," *Computer-Aided Design*, vol. 33, no. 1, pp. 91–102, 2001.

[11] B. Aldefeld, "On automatic recognition of 3d structures from 2d representations," *Computer-Aided Design*, vol. 15, no. 2, pp. 59–64, 1983.

[12] J. Dimri and B. Gurumoorthy, "Handling sectional views in volume-based approach to automatically construct 3d solid from 2d views," *Computer-Aided Design*, vol. 37, no. 5, pp. 485–495, 2005.

[13] H. Lee and S. Han, "Reconstruction of 3D interacting solids of revolution from 2D orthographic views," *Computer-Aided Design*, vol. 37, no. 13, pp. 1388–1398, 2005.

[14] "Constructive solid geometry." [Online]. Available: https://wiki.freecadweb.org/Constructive_solid_geometry

[15] M. A. Fahiem, S. A. Haq, and F. Saleemi, "A review of 3d reconstruction techniques from 2d orthographic line drawings," in *Geometric Modeling and Imaging (GMAI'07)*. IEEE, 2007, pp. 60–66.

[16] "Contours hierarchy." [Online]. Available: https://docs.opencv.org/master/d9/d8b/tutorial_py_contours_hierarchy.html

[17] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding, p 1027–1035," in *SODA'07: proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. Society for Industrial and Applied Mathematics, Philadelphia, PA*, 2007.

[18] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[19] É. Rémy and É. Thiel, "Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2d or 3d," *Pattern Recognition Letters*, vol. 23, no. 6, pp. 649–661, 2002.

[20] I. Mebsout. (2020) Deep learning's mathematics. [Online]. Available: https://towardsdatascience.com/deep-learnings-mathematics-f52b3c4d2576

[21] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[22] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[23] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

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

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[27] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[28] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning arxiv prepr 181103378," 2018.

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

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[31] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[32] T. C. Henderson, *Analysis of engineering drawings and raster map images*. Springer, 2014.

[33] A. Çıçek and M. Gülesın, "Reconstruction of 3d models from 2d orthographic views using solid extrusion and revolution," *Journal of materials processing technology*, vol. 152, no. 3, pp. 291–298, 2004.

[34] J. Liu and B. Ye, "New method of 3D reconstruction from mechanical engineering drawings based on engineering semantics understanding," in *International Conference GraphiCon'2005*, 2005.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[37] R. Yan, L. Shao, L. Liu, and Y. Liu, "Natural image denoising using evolved local adaptive filters," *Signal Processing*, vol. 103, pp. 36–44, 2014.

[38] V. Jain and S. Seung, "Natural image denoising with convolutional networks," in *Advances in neural information processing systems*, 2009, pp. 769–776.

[39] S. Anwar and N. Barnes, "Real image denoising with feature attention," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 3155–3164.

[40] M. Sonka, V. Hlavac, and R. Boyle, "Image processing, analysis, and machine vision second edition," *International Thomson*, vol. 2, 1999.

[41] S.-J. Ko and Y. H. Lee, "Center weighted median filters and their applications to image enhancement," *IEEE transactions on circuits and systems*, vol. 38, no. 9, pp. 984–993, 1991.

[42] K. Chinnasarn, Y. Rangsanseri, and P. Thitimajshima, "Removing salt-and-pepper noise in text/graphics images," in *IEEE. APCCAS 1998. 1998 IEEE Asia-Pacific Conference on Circuits and Systems. Microelectronics and Integrating Systems. Proceedings (Cat. No. 98EX242)*. IEEE, 1998, pp. 459–462.

[43] L. O'Gorman, "Image and document processing techniques for the rightpages electronic library system," in *11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems*, vol. 1. IEEE Computer Society, 1992, pp. 260–261.

[44] G. A. Story, L. O'Gorman, D. Fox, L. L. Schaper, and H. Jagadish, "The rightpages image-based electronic library for alerting and browsing," *Computer*, vol. 25, no. 9, pp. 17–26, 1992.

[45] P. Y. Simard and H. S. Malvar, "An efficient binary image activity detector based on connected components," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3. IEEE, 2004, pp. iii–229.

[46] H. S. Al-Khaffaf, A. Z. Talib, and R. Abdul, "Salt and pepper noise removal from document images," in *International Visual Informatics Conference*. Springer, 2009, pp. 607–618.

[47] H. Li, "Deep learning for image denoising," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 7, no. 3, pp. 171–180, 2014.

[48] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, "Deep learning on image denoising: An overview," *Neural Networks*, 2020.

[49] B. Fu, X. Zhao, Y. Li, X. Wang, and Y. Ren, "A convolutional neural networks denoising approach for salt and pepper noise," *Multimedia Tools and Applications*, vol. 78, no. 21, pp. 30 707–30 721, 2019.

[50] G. Gaál, B. Maga, and A. Lukács, "Attention u-net based adversarial architectures for chest x-ray lung segmentation," *arXiv preprint arXiv:2003.10304*, 2020.

[51] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[52] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[53] A. G. Smith, J. Petersen, R. Selvan, and C. R. Rasmussen, "Segmentation of roots in soil with u-net," *Plant Methods*, vol. 16, no. 1, pp. 1–15, 2020.

[54] B. Huang, K. Lu, N. Audeberr, A. Khalel, Y. Tarabalka, J. Malof, A. Boulch, B. Le Saux, L. Collins, K. Bradbury *et al.*, "Large-scale semantic classification: outcome of the first year of inria aerial image labeling benchmark," in *IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2018, pp. 6947–6950.

[55] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo, "Abc: A big cad model dataset for geometric deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9601–9611.

[56] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim," in *2010 20th international conference on pattern recognition*. IEEE, 2010, pp. 2366–2369.

[57] H. Lu, A. C. Kot, and Y. Q. Shi, "Distance-reciprocal distortion measure for binary document images," *IEEE Signal Processing Letters*, vol. 11, no. 2, pp. 228–231, 2004.

[58] H. Lu, J. Wang, A. C. Kot, and Y. Q. Shi, "An objective distortion measure for binary document images based on human visual perception," in *Object recognition supported by user interaction for service robots*, vol. 4. IEEE, 2002, pp. 239–242.

[59] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[60] GREC-Dataset, "Grec'03 dataset," http://www.cs.cityu.edu.hk/~liuwy/ArcContest/test-images-2003.zip, 2003, accessed: 2018-07-16.

[61] J. Zhai, L. Wenyin, D. Dori, and Q. Li, "A line drawings degradation model for performance characterization," in *Proceedings of the Seventh International Conference on Document Analysis and Recognition-Volume 2*, 2003, p. 1020.

[62] R. D. Janssen and A. M. Vossepoel, "Adaptive vectorization of line drawing images," *Computer vision and image understanding*, vol. 65, no. 1, pp. 38–56, 1997.

[63] R. Kasturi, S. T. Bow, W. El-Masri, J. Shah, J. R. Gattiker, and U. B. Mokate, "A system for interpretation of line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 978–992, 1990.

[64] E. Bodansky and M. Pilouk, "Using local deviations of vectorization to enhance the performance of raster-to-vector conversion systems," *International Journal on Document Analysis and Recognition*, vol. 3, no. 2, pp. 67–72, 2000.

[65] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 6, pp. 890–904, 2006.

[66] G. S. di Baja, "Well-shaped, stable, and reversible skeletons from the (3, 4)-distance transform," *Journal of visual communication and image representation*, vol. 5, no. 1, pp. 107–115, 1994.

[67] I. Arganda-Carreras, R. Fernández-González, A. Muñoz-Barrutia, and C. Ortiz-De-Solorzano, "3d reconstruction of histological sections: application to mammary gland tissue," *Microscopy research and technique*, vol. 73, no. 11, pp. 1019–1029, 2010.

[68] A. Kumar, A. Kaur, and M. Kumar, "Face detection techniques: a review," *Artificial Intelligence Review*, vol. 52, no. 2, pp. 927–948, 2019.

[69] Z. Hu, J. Tang, Z. Wang, K. Zhang, L. Zhang, and Q. Sun, "Deep learning for image-based cancer detection and diagnosis- a survey," *Pattern Recognition*, vol. 83, pp. 134–149, 2018.

[70] E. Elyan, L. Jamieson, and A. Ali-Gombe, "Deep learning for symbols detection and classification in engineering drawings," *Neural networks*, vol. 129, pp. 91–102, 2020.

[71] E. Elyan, C. M. Garcia, and C. Jayne, "Symbols classification in engineering drawings," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[72] C. F. Moreno-García, E. Elyan, and C. Jayne, "New trends on digitisation of complex engineering drawings," *Neural computing and applications*, vol. 31, no. 6, pp. 1695–1712, 2019.

[73] L. Wendling and S. Tabbone, "A new way to detect arrows in line drawings," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 7, pp. 935–941, 2004.

[74] A. M. Intwala, K. Kharade, R. Chaugule, and A. Magikar, "Dimensional arrow detection from cad drawings," *Indian J. Sci. Technol*, vol. 9, no. 21, pp. 1–7, 2016.

[75] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Convnet-based optical recognition for engineering drawings," in *37th Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2017, pp. DETC2017–68 186.

[76] O. Andersson and S. Reyna Marquez, "A comparison of object detection algorithms using unmanipulated testing images: Comparing sift, kaze, akaze and orb," 2016.

[77] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[78] M. Plauth, W. Hagen, F. Feinbube, F. Eberhardt, L. Feinbube, and A. Polze, "Parallel implementation strategies for hierarchical non-uniform memory access systems by example of the scale-invariant feature transform algorithm," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 1351–1359.

[79] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157.

[80] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.

[81] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 12, no. 7, pp. 629–639, 1990.

[82] W. Lee, "Tridiagonal matrices: Thomas algorithm," *MS6021, Scientific Computation, University of Limerick*, 2011.

[83] P. Fernández Alcantarilla, A. Bartoli, and A. Davison, "Kaze features," 10 2012.

[84] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[85] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[86] S. Arlot, A. Celisse *et al.*, "A survey of cross-validation procedures for model selection," *Statistics surveys*, vol. 4, pp. 40–79, 2010.

[87] S. Zhang, T. Chen, Y. Zhang, S. Hu, and R. R. Martin, "Vectorizing cartoon animations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 4, pp. 618–629, 2009.

[88] M. Bessmeltsev and J. Solomon, "Vectorization of line drawings via polyvector fields," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 1, p. 9, 2019.

[89] J. Chen, Q. Lei, Y. Miao, and Q. Peng, "Vectorization of line drawing image based on junction analysis," *Science China Information Sciences*, vol. 58, no. 7, pp. 1–14, 2015.

[90] J. Xie, H. Winnemöller, W. Li, and S. Schiller, "Interactive vectorization," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 6695–6705.

[91] K.-H. Lee, S.-B. Cho, and Y.-C. Choy, "Automated vectorization of cartographic maps by a knowledge-based system," *Engineering Applications of Artificial Intelligence*, vol. 13, no. 2, pp. 165–178, 2000.

[92] B. Kim, O. Wang, A. C. Öztireli, and M. Gross, "Semantic segmentation for line drawing vectorization using neural networks," in *Computer Graphics Forum*, vol. 37, no. 2. Wiley Online Library, 2018, pp. 329–338.

[93] Y. Chen, N. A. Langrana, and A. K. Das, "Perfecting vectorized mechanical drawings," *Computer Vision and Image Understanding*, vol. 63, no. 2, pp. 273–286, 1996.

[94] V. Nagasamy and N. A. Langrana, "Engineering drawing processing and vectorization system," *Computer Vision, Graphics, and Image Processing*, vol. 49, no. 3, pp. 379–397, 1990.

[95] D. Dori, "Orthogonal zig-zag: an algorithm for vectorizing engineering drawings compared with hough transform," *Advances in Engineering Software*, vol. 28, no. 1, pp. 11–24, 1997.

[96] D. Dori and W. Liu, "Sparse pixel vectorization: An algorithm and its performance evaluation," *IEEE Transactions on pattern analysis and machine Intelligence*, vol. 21, no. 3, pp. 202–215, 1999.

[97] S. Mandal, A. K. Das, and P. Bhowmick, "A fast technique for vectorization of engineering drawings using morphology and digital straightness," in *Proceedings of the Seventh Indian Conference on Computer Vision, Graphics and Image Processing*. ACM, 2010, pp. 490–497.

[98] P. De, S. Mandal, P. Bhowmick, and A. Das, "Askme: adaptive sampling with knowledge-driven vectorization of mechanical engineering drawings," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 19, no. 1, pp. 11–29, 2016.

[99] D. R. Kasimov, A. V. Kuchuganov, V. N. Kuchuganov, and P. Oskolkov, "Vectorization of raster mechanical drawings on the base of ternary segmentation and soft computing," *Programming and Computer Software*, vol. 43, no. 6, pp. 337–344, 2017.

[100] E. Lutton and P. Martinez, "A genetic algorithm for the detection of 2D geometric primitives in images," in *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1. IEEE, 1994, pp. 526–528.

[101] L. Zhang, W. Xu, and C. Chang, "Genetic algorithm for affine point pattern matching," *Pattern Recognition Letters*, vol. 24, no. 1-3, pp. 9–19, 2003.

[102] GREC-Dataset, "Grec'13 dataset," https://sites.google.com/a/iupr.com/grec2013_arc_and_line_seg_contest/, 2013, accessed: 2018-07-16.

[103] L. Wenyin and D. Dori, "A protocol for performance evaluation of line detection algorithms," *Machine Vision and Applications*, vol. 9, no. 5-6, pp. 240–250, 1997.

[104] D. Elliman, "Tif2vec, an algorithm for arc segmentation in engineering drawings," in *International Workshop on Graphics Recognition*. Springer, 2001, pp. 350–358.

[105] J. Song, M. R. Lyu, and S. Cai, "Effective multiresolution arc segmentation: Algorithms and performance evaluation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 11, pp. 1491–1506, 2004.

[106] X. Hilaire, "Ranvec and the arc segmentation contest: second evaluation," in *International Workshop on Graphics Recognition*. Springer, 2005, pp. 362–368.

[107] D. Keysers and T. M. Breuel, "Optimal line and arc detection on run-length representations," in *International Workshop on Graphics Recognition*. Springer, 2005, pp. 369–380.

[108] V. Egiazarian, O. Voynov, A. Artemov, D. Volkhonskiy, A. Safin, M. Taktasheva, D. Zorin, and E. Burnaev, "Deep vectorization of technical drawings," in *European Conference on Computer Vision*. Springer, 2020, pp. 582–598.

[109] J.-D. Favreau, F. Lafarge, and A. Bousseau, "Fidelity vs. simplicity: a global approach to line drawing vectorization," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 4, pp. 1–10, 2016.

[110] L. Donati, S. Cesano, and A. Prati, "A complete hand-drawn sketch vectorization framework," *Multimedia Tools and Applications*, vol. 78, no. 14, pp. 19 083–19 113, 2019.

[111] J.-H. Gong, H. Zhang, B. Jiang, and J.-G. Sun, "Identification of sections from engineering drawings based on evidence theory," *Computer-Aided Design*, vol. 42,

no. 2, pp. 139–150, 2010, aCM Symposium on Solid and Physical Modeling and Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S001044850900147X

[112] C. Liu, J. Wu, P. Kohli, and Y. Furukawa, "Raster-to-vector: Revisiting floorplan transformation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2195–2203.

[113] Y. Guo, Z. Zhang, C. Han, W. Hu, C. Li, and T.-T. Wong, "Deep line drawing vectorization via line subdivision and topology reconstruction," in *Computer Graphics Forum*, vol. 38, no. 7. Wiley Online Library, 2019, pp. 81–90.

[114] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji, "Csgnet: Neural shape parser for constructive solid geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5515–5523.

[115] J. Huang, J. Gao, V. Ganapathi-Subramanian, H. Su, Y. Liu, C. Tang, and L. J. Guibas, "Deepprimitive: Image decomposition by layered primitive detection," *Computational Visual Media*, vol. 4, no. 4, pp. 385–397, 2018.

[116] S. Fuhrmann, F. Langguth, and M. Goesele, "Mve-a multi-view reconstruction environment." in *GCH*. Citeseer, 2014, pp. 11–18.

[117] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang, "Pixel2mesh: Generating 3d mesh models from single rgb images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 52–67.

[118] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," in *European conference on computer vision*. Springer, 2016, pp. 628–644.

[119] B. Yang, S. Wang, A. Markham, and N. Trigoni, "Robust attentional aggregation of deep feature sets for multi-view 3d reconstruction," *International Journal of Computer Vision*, vol. 128, no. 1, pp. 53–73, 2020.

[120] J. Delanoy, M. Aubry, P. Isola, A. A. Efros, and A. Bousseau, "3d sketching using multi-view deep volumetric prediction," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 1, no. 1, pp. 1–22, 2018.

[121] D. Feng, S. Lee, and B. Gooch, "Perception-based construction of 3d models from line drawings," in *ACM SIGGRAPH 2006 Sketches*, 2006, pp. 53–es.

[122] S.-X. Liu, S.-M. Hu, C.-L. Tai, and J.-G. Sun, "A matrix-based approach to reconstruction of 3d objects from three orthographic views," in *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*. IEEE, 2000, pp. 254–261.

[123] J.-H. Gong, H. Zhang, G.-F. Zhang, and J.-G. Sun, "Solid reconstruction using recognition of quadric surfaces from orthographic views," *Computer-Aided Design*, vol. 38, no. 8, pp. 821–835, 2006.

[124] M. C. Cooper, "Wireframe projections: physical realisability of curved objects and unambiguous reconstruction of simple polyhedra," *International Journal of Computer Vision*, vol. 64, no. 1, pp. 69–88, 2005.

[125] A. Seff, Y. Ovadia, W. Zhou, and R. P. Adams, "Sketchgraphs: A large-scale dataset for modeling relational geometry in computer-aided design," in *ICML 2020 Workshop on Object-Oriented Learning*, 2020.

[126] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[127] M. Ferguson, R. Ak, Y.-T. T. Lee, and K. H. Law, "Automatic localization of casting defects with convolutional neural networks," in *2017 IEEE international conference on big data (big data)*. IEEE, 2017, pp. 1726–1735.

[128] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[129] I. Z. Mukti and D. Biswas, "Transfer learning based plant diseases detection using resnet50," in *2019 4th International Conference on Electrical Information and Communication Technology (EICT)*. IEEE, 2019, pp. 1–6.

[130] V. Badrinarayanan, A. Handa, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling," *arXiv preprint arXiv:1505.07293*, 2015.

[131] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[132] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[133] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2," https://github.com/facebookresearch/detectron2, 2019.

[134] A. Kirillov, Y. Wu, K. He, and R. Girshick, "Pointrend: Image segmentation as rendering," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9799–9808.

[135] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[136] H. Xie, H. Yao, S. Zhang, S. Zhou, and W. Sun, "Pix2vox++: multi-scale context-aware 3d object reconstruction from single and multiple images," *International Journal of Computer Vision*, vol. 128, no. 12, pp. 2919–2935, 2020.

[137] H. Xie, H. Yao, X. Sun, S. Zhou, and S. Zhang, "Pix2vox: Context-aware 3d reconstruction from single and multi-view images," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 2690–2698.

[138] Y. Zhou, H. Qi, and Y. Ma, "End-to-end wireframe parsing," 2020.

DOCTORAT BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Techniques de vectorisation hybride et non supervisée pour la reconstruction 3D de dessins d'ingénierie

Mots clés : Vectorisation, Dessins techniques, Deep Learning, Segmentation, Reconstruction 3D

Résumé : Les technologies de conception assistée par ordinateur se sont fortement améliorées au cours de la dernière décennie. Cette révolution a créé un fossé entre les anciens dessins techniques (faits à la main et enregistrés sous forme d'images matricielles) et les nouveaux dessins techniques (enregistrés sous forme de données vectorielles), qui peuvent être modifiés et enregistrés facilement. De plus, ce fossé s'est encore creusé avec l'essor des applications de réalité virtuelle et augmentée. Différents algorithmes permettent de reconstruire des modèles 3D à partir de dessins techniques vectoriels. Par conséquent, un système de conception assistée par ordinateur pour convertir les images matricielles de dessins techniques en données vectorielles est plus que jamais nécessaire. Dans cette thèse, une étape de prétraitement est d'abord proposée afin de préparer les données au processus de vectorisation. Dans cette étape, on extrait les informations graphiques du dessin technique, sépare les différentes vues à l'aide d'une approche de regroupement et débruite chaque vue séparément en adaptant un réseau d'apprentissage profond. En outre, cette étape génère le squelette (au sens de la morphologie mathématique) de l'image, qui est ensuite utilisé dans le processus de vectorisation. Enfin, les pointes de flèches sont détectées, ces flèches pouvant être utilisées pour détecter les dimensions du graphique. L'approche par algorithme génétique est adaptée à la vectorisation des dessins techniques. Cette méthode adapte les paramètres géométriques en fonction de la largeur de ligne et de segment estimée, ce qui diminue la possibilité de fragmenter les primitives. L'évaluation sur des dessins techniques montre la robustesse et les effets des hyperparamètres sur l'approche de vectorisation non supervisée proposée. Une méthode hybride est ensuite proposée afin de réduire la complexité du problème de vectorisation. L'étape supervisée utilise des réseaux d'apprentissage profond pour segmenter l'image d'entrée en différentes couches où chaque couche ne contient qu'un seul type de primitives (comme la couche des lignes droites et la couche des cercles). L'étape non supervisée détecte la primitive dans chaque couche séparément. L'approche de vectorisation hybride convertit le problème de la segmentation des courbes (dans l'approche de vectorisation non supervisée) en détection de primitives. De plus, l'approche de vectorisation hybride diminue la complexité informatique grâce à la détection simultanée de différents types de primitives.

Title : Unsupervised and Hybrid Vectorization Techniques for 3D reconstruction of Engineering Drawings

Keywords : Vectorization, Technical drawings, Deep Learning, Segmentation, 3D reconstruction

Abstract : Computer-aided design technologies are highly improved during the last decade. This revolution created a gap between old technical drawings (hand-made saved as raster images) and the new technical drawings (saved as vector data), which can be modified and saved easily. Moreover, this gap grows up with the rising of virtual and augmented reality applications. Different algorithms can reconstruct 3D models from vector technical drawings. Therefore, a computer-aided design system for converting raster images into vector data is needed more than ever. In this thesis, a preprocessing framework is proposed to prepare data for the vectorization process. The framework extracts graphical information from the engineering drawing template, separates different views using a clustering approach, and denoises each view separately by adopting a deep learning network. Moreover, the framework generates the skeleton of the image, which is used in the vectorization process. Finally, the framework detects arrowheads where arrowheads can be lately a pattern to detect dimension sets. The genetic algorithm approach is adapted to vectorize technical drawings. This method tune geometric parameters based on the estimated width, which decreases the possibility of fragmenting primitives. The evaluation shows the robustness and effects of hyperparameters on the proposed unsupervised vectorization approach. A hybrid method is proposed to reduce the complexity of the vectorization problem. The supervised stage uses deep learning networks to segment the input image into different layers where each layer contains only one type of primitives (such as straight-line layer and circle layer). The unsupervised stage detects the primitive in each layer separately. The hybrid vectorization approach converts the problem from curve segmentation (in unsupervised vectorization approach) into primitive detection. Moreover, the hybrid vectorization approach decreases the time complexity due to simultaneously detecting different types of primitives.