

CNN-BASED SYMBOL RECOGNITION AND DETECTION IN PIPING

DRAWINGS

by

Yuxi Zhang

A Thesis

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science



Department of Lyles School of Civil Engineering

West Lafayette, Indiana

August 2019

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Hubo Cai, Chair

Lyles School of Civil Engineering, Purdue University

Dr. Dulcy M. Abraham

Lyles School of Civil Engineering, Purdue University

Dr. Charles A. Bouman

School of Electrical and Computer Engineering, Purdue University

Approved by:

Dr. Dulcy M. Abraham

Head of the Departmental Graduate Program

ACKNOWLEDGMENTS

I am deeply thankful to my advisor, Professor Cai. He is knowledgeable and patient in guiding me to solve a problem. With his encouragement and supervision, I learned a lot in technical knowledge and critical thinking. His passion, hardworking and extraordinary vision make him an outstanding scientist and engineer, which affects me in my research life. I appreciate his support, which provides me a comfortable environment for research.

It is my privilege to have Prof. Abraham, Prof. Bouman in my thesis committee, who are always willing to help in their expertise – Prof. Abraham provides her insights in construction management and project lifecycle; Prof. Bouman leads me to the field of image processing.

I am so honored as a member of the Lab of Computer Integrated Infrastructure Informatics (LCIII) and indebted to all the lab members and alumni in Prof. Cai's research group. It has been a wonderful experience to have such a close relationship with so many people. Dr. Chenxi Yuan are my previous colleagues and we share so many good memories when we worked together in INDOT projects. Besides, it is always inspiring and pleasant to discuss with group members such as Xin Xu, Jiannan Cai, JungHo Jeon, and Liu Yang.

I would also like to take the time to acknowledge my parents who have been supporting me mentally and financially throughout my student life in China and my Master abroad. They have been extremely patient and understanding and I would like to thank them for all the amazing opportunities they have given me over the years.

TABLE OF CONTENTS

LIST OF TABLES	7
LIST OF FIGURES	8
ABSTRACT	9
CHAPTER 1. INTRODUCTION	11
1.1 Background and Problem Statement	11
1.2 Literature Review	15
1.2.1 Traditional Statistical Methods	16
1.2.2 Structural or Syntactic Methods	18
1.2.3 Review of Deep Neural Networks.....	21
1.2.3.1 Principles of Neural Networks	21
1.2.3.2 Related Works on CNN-based Object Classification and Detection	23
1.2.4 Review of Data Augmentation	25
1.3 Goal and Objectives	26
1.4 Significance and Research Contributions	26
1.5 Organization of the Thesis	27
CHAPTER 2. CNN-BASED SYMBOL RECOGNITION	29
2.1 Introduction	29
2.2 Data Preparation	30
2.2.1 Normal Synthetic Symbols.....	30
2.2.2 Affine Transformation.....	31
2.2.3 Additional Variations	32
2.3 Methodology	33
2.3.1 The Design of the Architectures of the Basic CNN	33
2.3.2 The Design of the Architectures of the CNN + STN Model	36
2.3.3 The Design of the Loss Function	38
2.3.4 The Selection of the Optimization Methods.....	39
2.4 Experiments.....	40
2.4.1 Experimental Setup	40

2.4.2	Experimental Results.....	41
2.5	Generalization Capabilities of CNNs to Rotations	43
2.5.1	Rotation Invariance	43
2.5.2	Generalization Capabilities of CNNs to Rotations.....	44
2.6	Summary and Conclusions.....	46
2.6.1	Summary	46
2.6.2	Conclusions	47
2.6.3	Limitations.....	48
2.6.4	Future Works	48
CHAPTER 3.	CNN-BASED SYMBOL DETECTION	49
3.1	Introduction	49
3.2	Methodology	49
3.2.1	The Convolutional Network.....	50
3.2.2	The Region Proposal Network	51
3.2.3	ROI Pooling.....	53
3.2.4	Classification	53
3.3	Data Preparation and Experiments	54
3.3.1	Data Preparation	54
3.3.2	Experimental Setup	55
3.3.3	Experimental Results.....	56
3.4	Summary and Conclusions.....	59
3.4.1	Summary	59
3.4.2	Conclusions	60
3.4.3	Limitations.....	60
3.4.4	Future Works	60
CHAPTER 4.	SUMMARY AND CONCLUSIONS.....	61
4.1	Summary and Conclusions.....	61
4.2	Limitations	61
4.3	Future Works	61
APPENDIX A.	CODE	63
APPENDIX B.	IMAGE SAMPLES	118

REFERENCES	127
------------------	-----

LIST OF TABLES

Table 1.1 Sub-problems of technical drawings interpretation and methods.....	13
Table 1.2 Pixel-based methods for symbol recognition in drawings.....	17
Table 2.1 Parameters for affine transformation	32
Table 2.2 Noise and distortion additions and dilation/erosion operations.....	33
Table 2.3 The architecture of the basic CNN	34
Table 2.4 The architecture of the localisation network.....	37
Table 2.5 Experimental setup	41
Table 2.6 Experimental results – recognition accuracies.....	42
Table 2.7 The average and standard deviation of recognition accuracies for five repeats	42
Table 2.8 The confusion matrix for the CNN model	43
Table 2.9 The confusion matrix for the CNN + STN model	43
Table 3.1 The architecture of VGG16 (Simonyan and Zisserman 2016)	50
Table 3.2 Results for piping symbols with Faster RCNN detectors and VGG16.....	57

LIST OF FIGURES

Figure 1.1 Parts of a plumbing diagram in the elevation view	12
Figure 1.2 General pipeline for region-based symbol detection.....	16
Figure 1.3 The taxonomy of primitive-based symbol recognition and detection methods	19
Figure 1.4 (a) multilayer neural networks and (b) backpropagation (LeCun et al. 2015)	22
Figure 1.5 Architecture of LeNet-5 (LeCun et al. 1998).	23
Figure 2.1 The pipeline of generating symbols	30
Figure 2.2 Example of parameters in a normal synthetic symbol.....	31
Figure 2.3 A convolutional layer using a 3x3 filter	35
Figure 2.4 Max pooling using a 2 x 2 window and a stride size of one	36
Figure 2.5 The architecture of CNN+STN model.	37
Figure 2.6 Sample images used in the dataset for symbol recognition.....	40
Figure 2.7 Predictions of CNN and CNN + STN models.....	45
Figure 3.1 The architecture of the Faster RCNN network (Ren et al. 2016).....	50
Figure 3.2 The architecture of RPN	51
Figure 3.3 The anchors which scale $\in \{128, 256, 512\}$ and aspect ratio $\in \{0.5, 1, 2\}$	51
Figure 3.4 The architecture of ROI pooling.....	53
Figure 3.5 Seven symbols used in drawings.....	54
Figure 3.6 The distribution of the numbers of symbols for each class	55
Figure 3.7 The samples of true positive and false positive.....	58

ABSTRACT

Author: Zhang, Yuxi. M.S.

Institution: Purdue University

Degree Received: August 2019

Title: CNN-based Symbol Recognition and Detection in Piping Drawings

Major Professor: Hubo Cai

Piping is an essential component in buildings, and its as-built information is critical to facility management tasks. Manually extracting piping information from legacy drawings that are in paper, PDF, or image format is mentally exerting, time-consuming, and error-prone. Symbol recognition and detection are core problems in the computer-based interpretation of piping drawings, and the main technical challenge is to determine robust features that are invariant to scaling, rotation, and translation. This thesis aims to use convolutional neural networks (CNNs) to automatically extract features from raw images, and consequently, to locate and recognize symbols in piping drawings.

In this thesis, the Spatial Transformer Network (STN) is applied to improve the performance of a standard CNN model for recognizing piping symbols, and the Faster Region-based Convolutional Neural Network (Faster RCNN) is adopted to exploit its capacity in symbol detection. For experimentation, the synthetic data are generated as follows. Two datasets are generated for symbol recognition and detection, respectively. For recognition, eight types of symbols are synthesized based on the geometric constraints between the primitives. The drawing samples for detection are manually sketched using AutoCAD MEP software and its piping component library, and seven types of symbols are selected from the piping component library. Both sets of samples are augmented with various scales, rotations, and random noises.

The experiment for symbol recognition is conducted and the accuracies of the recognition accuracy of the CNN + STN model and the standard CNN model are compared. It is observed that the spatial transformer layer improves the accuracy in classifying piping symbols from 95.39% to 98.26%. For the symbol detection task, the experiment is conducted using a public implementation of Faster RCNN. The mean Average Precision (mAP) is 82.8% when Intersection over Union (IoU) threshold equals to 0.5. Imbalanced data (i.e., imbalanced samples in each class) led to a decrease in the Average Precision in the minority class. Also, the symbol library, the small dataset, and the

complex backbone network limit the generality of the model. Future work will focus on the collection of larger set of drawings and the improvement of the network's geometric invariance.

CHAPTER 1. INTRODUCTION

1.1 Background and Problem Statement

As-built information on the piping system is critical to many facility management (FM) tasks, but unfortunately, that information is typically in paper-format piping drawings (Joseph and Pridmore 1992, Rahul et al. 2019) and is very difficult to access in a timely manner. A piping drawing is a schematic representation of the flow and the constitution of a piping system, such as a plumbing system that is an essential component in buildings. The piping diagram uses a graphical language that is composed of schematic lines and graphical symbols and annotations to illustrate the piping process with instrumentations and control devices. Figure 1.1 shows the snapshots of a plumbing drawing in the elevation view. Users interpret the meaning of graphics based on the legend table, annotations, and geometric and topological relationships of graphics. With the rich context and spatial information, piping drawings are used to locate specific piping components and extract product information such as product models, pipeline orientation, and piping components' locations, all of which has been identified as required information for many FM tasks, including maintenance (Javier et al. 2012; Cavka et al. 2015), operations (Sattenini et al. 2011; Mayo and Issa 2016), asset management (Becerik-Gerber et al. 2012), and energy monitoring (Volk et al. 2014; Yalcinkaya and Singh 2015). Therefore, information extraction from piping drawings is a key task in facility management.

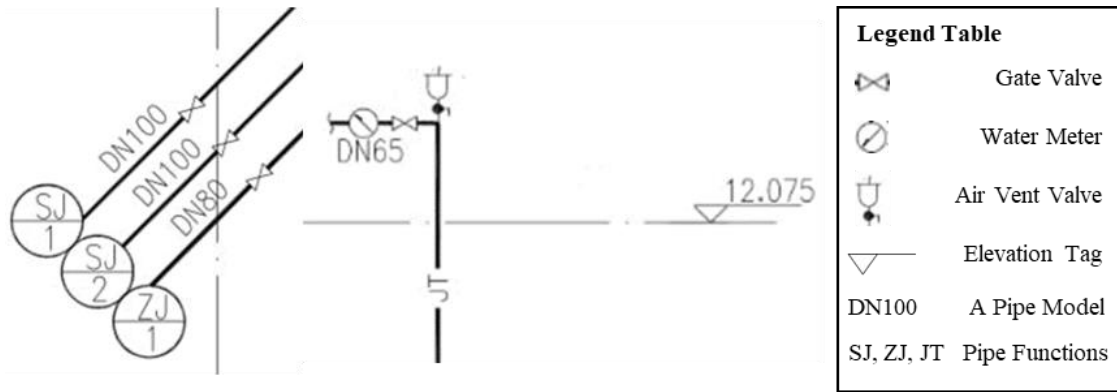


Figure 1.1 Parts of a plumbing diagram in the elevation view

However, manually extracting information from legacy drawings that are in paper, PDF, or image format is mentally exerting, time-consuming, and error-prone. According to a report by the National Institute of Standards and Technology (NIST), the estimated cost of inadequate interoperability in the U.S. capital facilities industry is \$15.8 billion per year; about 57.8% of this cost is borne by owners and operators during facility O&M (O’Conner 2004). Some of these costs arise from the information retrieval process in many existing buildings. Volk et al (2014) proposed that the high conversion efforts from drawings to semantic objects is the main challenge in building information integration. Paradoxically, with the accumulation of recreated models, concern about information overload is also evolving. From the view of lean management, many researchers (e.g., Jylhä and Suvanto 2015; Gerrish et al. 2017) recognize that the irretrievable and non-indexed files lead to inefficiencies in facility management, meaning that poor document management is the key deficiency in the current practice. Therefore, to eliminate the tedious process of manually rebuilding piping design or repeatedly digging into the superfluous documents, there is a critical need for developing an automated method to extract semantic information from piping drawings.

In today’s market, tools for interpreting drawings are still stuck in a low automated level. Existing tools can be categorized into two types: drawing vectorization and navigation. AutoCAD Raster Design and Bluebeam Revu are representatives in these two types, the former emphasizes the vectorization process and needs further correction and recognition by users, and the latter one integrates some useful measures and tracking tools based on the users’ markers.

From a scientific point of view, automated interpretation of general engineering drawings is an active topic in graphics recognition and document analysis community, but over the years end-to-

end applications have been regarded as a utopia and abandoned (Doermann and Tombre 2014). In fact, focuses have been set on the subparts of the problem, including text-graphics separation, primitive extraction and vectorization, symbol recognition and detection and knowledge modeling. The methods for each sub-problem are categorized in Table 1.1.

Table 1.1 Sub-problems of technical drawings interpretation and methods

Sub-problem	Methods
Text-graphics separation	Heuristics: connected component analysis + Hough transform + criterion Sparse representations: morphological component analysis; K-SVD Deep learning: connectionist text proposal networks (CTPN)
Primitive extraction and vectorization	Model fitting and voting: least square; random sample consensus (RANSAC); Hough transform (HT) Hypothesis testing: line support region growing + <i>a contrario</i> approach Line/arc decomposition: recursively significance comparison
Symbol recognition	Statistical: statistical features (geometric moments, R-signatures, etc.) + similarity measures Structural: structural descriptors (usually visual primitives or their spatial relations) and graph-based matching Syntactic: spatial predicates (usually primitives' relations) and inductive learning programming Deep learning: convolution neural networks (CNN)
Symbol detection	Region-based: region proposals (CC, grid partitions, sliding windows, region of interests) and similarity measures Line primitive based: graph embedding and indexing Deep learning: fully convolution networks (FCN)
Knowledge modeling	Descriptive knowledge: geometric and topological reasoning based on rule sets, ontologies or meta-models Control knowledge: image analysis methods selection and sequential ordering based on strategy rules, ontologies or meta-models

For the text-graphics separation problem, the baseline methods are the heuristic ones, which iteratively group connected components bounding boxes based on their properties (similar size, centers alignment, the histogram and aspect ratio of the elongated rectangle formed by aligned boxes). Their main limitation is in the case of text touching graphics (Fletcher and Kasturi 1988; Tombre et al. 2002). To overcome this limitation, some researchers proposed methods based on the sparse representation, which consider texts and graphics as two types of signals and separate them based discriminative dictionaries (Hoang and Tabbone 2010; Do et al. 2012). Recently, deep

learning approaches for text detection, such as CTPN (Tian et al. 2016), have been proposed, to seamlessly connect CNN and recurrent neural networks (RNN). CTPN has been adopted to solve text-graphics separation in drawings (Rahul et al. 2019) since it takes advantage of capturing the sequential context information.

Primitive extraction and vectorization are low-level processing steps and usually used as preliminaries to symbol recognition and detection. Least square, RANSAC and HT are generally applied as complements of vectorization or symbol recognition (Lamiroy and Guebbas 2010, Coustaty et al. 2011; Boumaiza and Tabbone 2012). Another interesting perspective on primitive extraction is based on the Helmholtz principle, which formulates lines/arcs as *meaningful events* (Desolneux et al. 2001). Based on this principle, a serial of researches on line/arc detection has been proposed (e.g., Von Gioi et al. 2010; Pătrăucean et al. 2012; Akinlar and Topal 2013). The significant advancement is that these approaches enable the control of false detection by computing the expectation of the number of event occurrences. Line/arc decomposition is a typical problem in drawing vectorization, which can transform the skeletons or edges into either lines or arcs. Rosin’s method has been widely adopted in this field, which extracts line and arc segments by recursively comparing segments’ significance (Rosin and West 1989). The main advantage of this method is that only a limited set of parameters are used so that it can be generalized into many applications. The common drawback of raster-vector conversion is error-prone (Santosh and Wendling 2015), and it is not trivial to jointly consider vectorization and further analysis.

Symbol recognition and detection methods are the core problems of drawings interpretation and can be roughly classified into two categories: with primitive extraction (structural or syntactic) and without primitive extraction (statistical). Here is a brief summary, and more details about this problem will be discussed in Section 2. Overall, the benefit of statistical methods is low computation cost, but the main challenge is the difficulty in feature selection which is critical to discriminative power and robustness. Structural and syntactic methods are much closer to human interpretation, and these methods outperform the statistical ones in the case of complex and composite symbols. However, the performance is limited by the precision of the vectorization process, and matching based on graph representation needs high computation cost.

Because of the dual image/language nature of drawings (Dori and Tombre 1995), knowledge is considered necessary to interpret some of the contextual information. On the whole, the required knowledge can be classified into two levels: descriptive knowledge and control knowledge of

descriptive ones as explained in Table 1. Many researches have shown the strengths of modeling descriptive and control knowledge using rules and grammars (Joseph and Pridmore 1992; Lu et al. 2009). Lu et al. (2009) proposed many implicit composition rules, such as reference and inheritance, which can be applied in the interpretation of drawings. Based on that, the recent trend on knowledge modeling is the inference of implicit knowledge based on ontologies and meta-models (Raveaux 2010; Bhatt et al. 2012; de las Heras et al. 2017). The ontology-based methods and meta model-based methods are complementary to each other, in that the former takes advantage of the expressive power of the domain ontologies, while the latter is not sensitive to the variations introduced in the low-level processing.

Therefore, there is a need to design an automated piping drawings interpretation method. Based on the review of state of the art, the sub-problems seem to reach a plateau, and the boundaries of these sub-problems have begun to blur. Therefore, this study focuses on symbol recognition and detection tasks. The reasons are mainly in twofold: (1) they are core sub-problems in this field since both low-level processing and high-level interpretation are tightly connected with them; (2) recent researches on deep learning have shown excellent performance on feature extraction, and its connectionist architecture enables the exploration of end-to-end applications.

1.2 Literature Review

Over the years, great efforts have been poured into the symbol recognition and spotting in technical drawings. Studies on symbol recognition and localization in technical drawings are based on either: (1) pixel-based descriptors or (2) primitive-based descriptors, including lines, arcs as well as the relations between them. They are commonly referred to as statistical and structural/syntactic methods (Doermann and Tombre 2014; Santosh and Wendling 2015), which will be reviewed in 1.2.1 and 1.2.2, respectively.

These traditional methods still face challenges in robust representations and computation complexity. With recent progress in deep neural networks, classification and object detection for natural images can be achieved via the connectionist architecture, which can break through the plateau of conventional approaches. In this paper, deep neural networks are applied for symbol classification and detection, and principles and recent studies will be discussed in 1.2.3. Besides,

data augmentation techniques are applied in this study to enrich the dataset, so related works in data augmentation will be briefly reviewed in 1.2.4

1.2.1 Traditional Statistical Methods

For methods in the statistical group, Figure 1.2 shows the general process in region-based symbol detection, including three steps:

- (1) Region detection: minimum bounding regions (e.g. rectangular, circular) enclosing the target symbols are detected;
- (2) Feature extraction: an optimal set of region-based features are selected and normalized by estimating the centroid position, orientation, and scale;
- (3) Classification: a proper classifier or a fusion of classifiers is designed.

In the remaining part of this subsection, pixel-based symbol recognition methods will be first reviewed, and then a discussion of related studies and challenges in region detection will be followed.

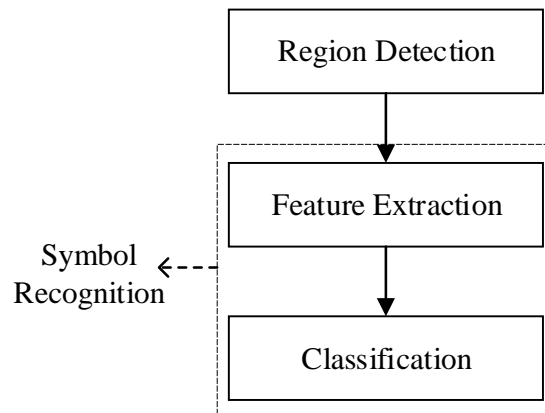


Figure 1.2 General pipeline for region-based symbol detection

As indicated in Figure 1.2, symbol recognition can be regarded as a sub-task of detection based on the assumption that the symbol is already well segmented; thus, only feature extraction and classification are needed in this task. However, it is challenging to develop an optimal set of features as well as classifiers for discriminative and robust symbol recognition because of the high variability nature of symbols, such as rotation, scaling, deformation, intra-class, and inter-class

variations. (Santosh and Wendling 2015). Table 1.2 shows some benchmark methods as well as recent studies for symbol recognition in technical drawings.

Table 1.2 Pixel-based methods for symbol recognition in drawings

Features	Classifiers	Advantages	Limitations	References
Radon transform + Fourier transform / Generic Radon transform	Euclidean distance	Invariant to rotation and scaling	Sensitive to noise and deformation	Tabbone et al. (2006) Hoang and Tabbone (2012)
Histogram of Radon transform	Dynamic time warping	Invariant to rotation, scaling; Robust to distortion and degradation	High computation cost	Tabbone et al. (2008) Santosh et al. (2013)
Moments (e.g. Hu moments, Zernike moments)	Locality-Sensitive Hashing	Invariant to rotation and scaling	Not suitable for complex symbols	Dutta et al. (2013)
Generic Fourier transform + normalization	City block distance	Invariant to rotation and scaling; Robust to distortion	Less discriminative power	Zhang and Lu (2002a) Zhang and Lu (2002b)
Blurred shape model (BSM) / BSM + Active appearance model	Adaboost / SVM	Invariant to rotation; Robust to deformation	Less discriminative power	Escalera et al. (2009) Almazán et al. (2012)
Shape context descriptors + Sparse representation	Term frequency and inverse document frequency (tf-idf)	Invariant to rotation and scaling	Not suitable for complex symbols	Do et al. (2016)

As illustrated in Table 1.2, related studies have provided significant insights into the fusion of features and the proper selection of classifier to balance similarity invariance and discriminative power. However, state-of-the-art approaches in symbol recognition community addressed slightly on affine invariance, which may become a trend with the popularity of mobile devices (Doermann and Tombre 2014).

In addition to symbol recognition, the region detection problem is crucial since the accuracy of symbol recognition highly relies on the segmentation. Compared to the intensive studies on symbol recognition, little works have been done on region detection in technical drawings due to its textureless nature, which only shape information is available (Nibal Nayef 2012). There are two

main types of region detection, one is window filtering, and the other one is grouping (Hosang et al. 2015). The window filtering method is based on a grid or sliding window, which works like a correlation filter, but this method is sensitive to size and rotation variation (Maclean and Tsotsos 2009, Escalera et al. 2011). The grouping methods are mainly applied in the primitive-based symbol detection task, which will be discussed in Section 1.2.2.

Above all, the challenges in the pixel-based symbol recognition and detection mainly lie on two aspects: (1) the selection of an optimal set of features and classifiers; (2) efficient and robust symbol localization which is invariant to scaling and rotation. It seems that traditional pixel-based approaches face the bottleneck that lacks useful high-level representations for symbols, resulting in that the interest on primitive-based descriptors is increasing substantially.

1.2.2 Structural or Syntactic Methods

Symbol patterns, which are of the strong structural nature, can be regarded as the composition of visual primitives (e.g., lines, arcs, meaningful regions) and represented by graph or grammar structures. This kind of representation enables it to capture high-level features of symbols. However, primitive-based symbol recognition is not trivial. Instead of formulating this problem as an exact graph matching based on an ideal model, researchers often focus on developing robust representations and efficient and error-tolerant matching/indexing methods. The reasons are mainly in twofold: (1) the intrinsic variability of the patterns, noise in the processing steps, and nondeterministic representations; (2) high computational complexity in the matching process (Conte et al. 2004). As an analogy to structural pattern recognition, syntactic methods are also adopted in this field. Different from structural ones, syntactic approaches often consider the primitives as grammars and interpret patterns with parsing technologies rather than matching. Figure 1.3 shows the taxonomy of related works.

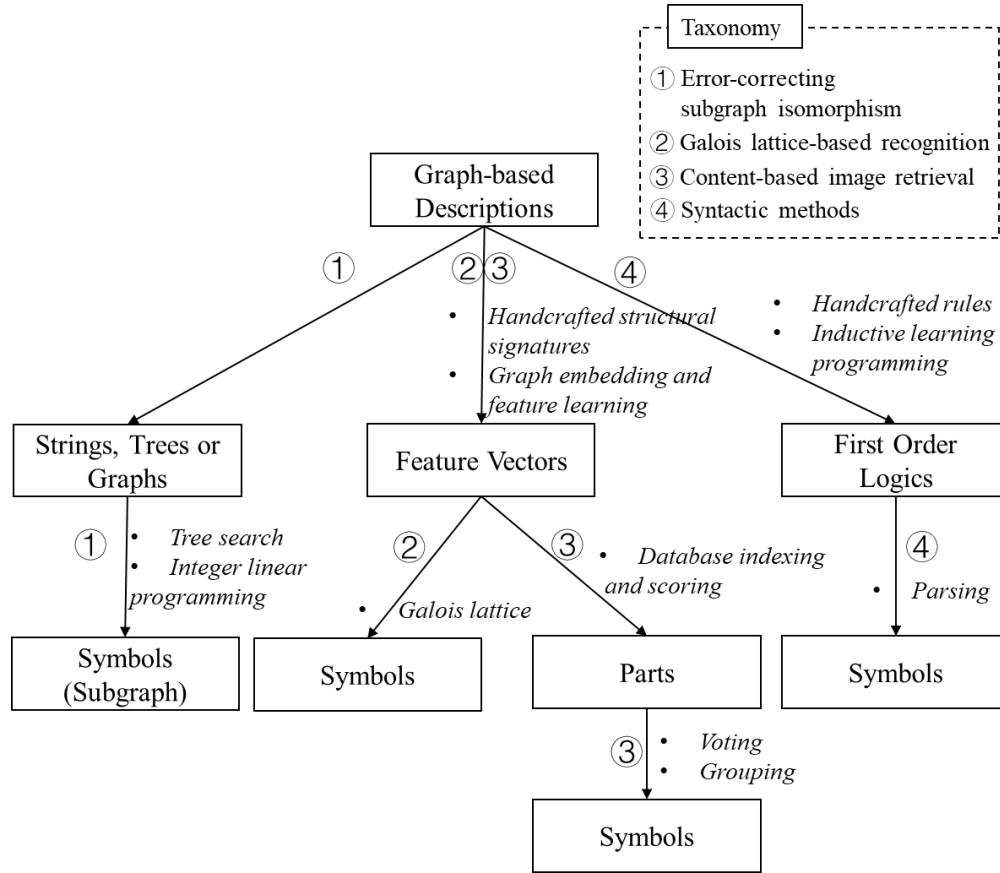


Figure 1.3 The taxonomy of primitive-based symbol recognition and detection methods

According to Figure 1.3, the pipeline of structural/syntactic methods is not straightforward. First, the graph-based descriptions vary widely, such as attributed relational graphs (ARGs) (Bunke and Messmer 1995; Luqmen et al. 2013; Santosh et al. 2014), region adjacency graphs (RAGs) (Lladós 2001; Le Bodic et al. 2009) and proximity graphs, encoding the fundamental parameters of primitives as well as geometric and topological relationships between them. After graph construction, there are four categories of methods widely adopted in the next steps.

Previously, researchers formulated this problem as error-correcting subgraph isomorphism. By minimizing the graph/string edit cost between the prototype and the input, the optimal subgraph can be detected, and tree search algorithms are applied to reduce the search space. For instance, Lladós (2001) encoded regions with boundary strings and represented the whole circuit diagram as a RAG, and then matched strings (at a local level) and the graph (at a global level) via minimizing edit costs and the branch and bound search algorithm. Similar researches with various

graph representations and tree search algorithms (Messmer and Bunke 1995; Bunke and Messmer 1998) can also be found. Another interesting perspective is to formulate it as an integer linear program, which enables error-tolerant on the vertex and edge labels (Le Bodic et al. 2009). The main limitation of these methods is the high computation cost, which is not suitable for large graphs.

To take advantage of low computation cost in statistical methods, a strategy that is mapping a graph to a low-dimensional vector is developed. The traditional structural signatures are handcrafted, such as parallelism, connectivity, relative lengths and angles between two lines (Dosch and Lladós 2003; Rusiñol and Lladós 2005; Coustaty et al. 2011). In contrast, recent studies on representation engineering are using graph embedding, which is to define a mapping function that can well describe the properties of the graphs, such as pair of paths, acyclic paths, node degrees, etc. Then, the compressed feature vectors can be learned offline via well-studied statistical methods, forming a large database. Lastly, the problems can be solved by database indexing and scoring. To spot the symbols in the drawings, researchers focus on developing a further step, and there are two kinds of methods. One is voting the hypothetic center and clustering, which uses the spatial relationships between symbol parts (Rusiñol et al 2010; Santosh et al. 2014; Dutta et al. 2013); the other one is grouping based on the confidence value, which needs the rich representations on each node (Luqmen et al. 2013). Overall, depending on how to design the features, these methods vary widely.

Studies in the second group, Galois lattice-based approaches, are also creative and insightful in mining the semantics behind the patterns. Although until now researches in this group can only tackle the symbol recognition problem, they introduce a novel view on the design of classifiers. Galois lattice can be regarded as a classifier since it indicates the correspondence between two partially ordered sets – shared attributes of a set of objects and shared objects of a set of attributes (Jaoua and Elloumi 2002). Based on that, the classification can be achieved by navigating on the Galois lattice, which is much closer to the human reasoning process. The attributes are similar to the features used in the third group, which has already been reviewed. Recent advances can refer to these related works (Coustaty et al. 2011; Visani et al. 2011; Boumaiza and Tabbone 2011; Boumaiza and Tabbone 2012).

Similarly, syntactic methods can be further divided into two: handcrafting and learning. The conventional rules are various, including but not limited to the spatial predicates between

primitives and the geometric information of a primitive with respect to the whole symbol (Yu et al. 2007). Recent studies using syntactic approaches is to automatically learn implicit descriptions of symbols using inductive logic programming based on background knowledge about spatial relationships (Santosh et al. 2009). In general, it is not trivial to transform structural signatures into grammars to robustly represent a symbol.

On the whole, structural or syntactic methods are more powerful in higher-level representations, conveying how parts are connected. With the growing interest in graph embedding and representation learning, recent advances provide significant insights into symbol recognition and detection. However, it is still far away from end-to-end drawings interpretation since the components in existing methods are separately trained and tuned. In the next section, related works on deep learning will be reviewed to explain how to solve these problems via an elegant connectionist architecture.

1.2.3 Review of Deep Neural Networks

1.2.3.1 Principles of Neural Networks

Compared to traditional statistical methods, which are limited by hand-engineered features, a deep neural network takes advantage of its multiple layer structure to extract higher-level features. Without the vectorization process and non-trivial graph embedding, representations at an abstract level can be automatically learned based on the training data. Figure 1.4 shows the multilayer neural networks and backpropagation.

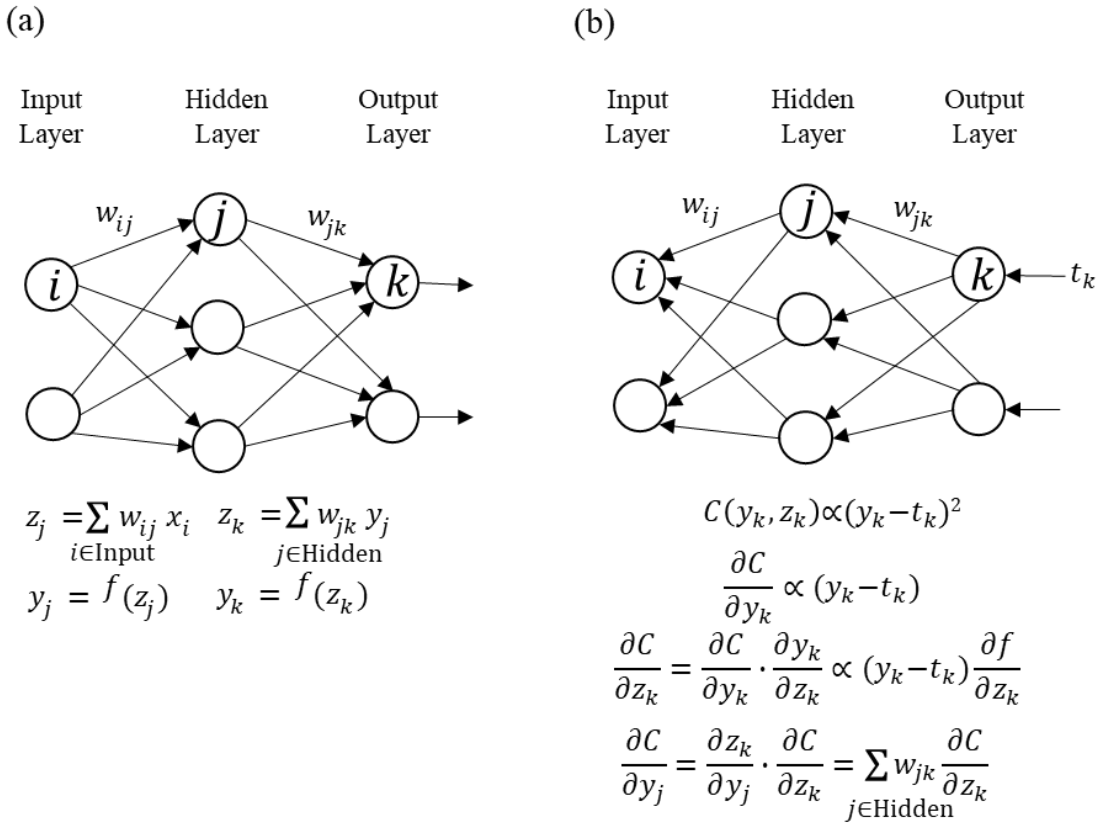


Figure 1.4 (a) multilayer neural networks and (b) backpropagation (LeCun et al. 2015)

Figure 1.4 (a) shows the multilayer neural networks with one hidden layer and one output layer. At each layer, the total input z is first computed, which is a weighted sum of the outputs y from the lower layer and weights w . Then a non-linear function $f(\cdot)$ is applied to z to get the output of the unit. This is the forward pass of neural networks, which can transform the input space into a distorted space. Next, a continuous optimization problem is formulated, which is to learn the weights in the neural networks so that the networks can be used as a feature extractor. Figure 1.4 (b) shows the backward pass. The value of cost function $C(y_k, z_k)$, averaged over all the training examples, is proportional to the square errors (t_k indicates the target value). To find the weights resulting in the local minimum of the cost function, the gradient descent algorithm is commonly applied. Using the chain rule of derivatives, the gradient can be computed according to equations in Figure 1.4 (b) so that weights between layers are available. In practice, to accelerate the training

process, stochastic gradient descent is usually applied, which is a method to repeat the optimization process for many small sets of examples until the average of the loss function converges.

Taking into account the local correlations and high dimensions of images, convolutional neural networks (CNNs) are widely adopted because of the four key ideas: local connections, shared weights, pooling and the use of many layers (LeCun et al. 2015). Figure 1.5 shows a classical CNN architecture: LeNet-5. The convolution operation is to multiply weights by the input values from local receptive fields, which can extract the locally sensitive features called feature maps, such as oriented edges and corners. In the framework of general multilayer neural networks, the convolution is equal to share weights for these repeated blocks. Next, the subsampling, or pooling, is conducted to reduce the resolution of feature maps, reducing the sensitivity of the output to shifts and distortions (LeCun et al. 1998). In the end, fully connected layers are connected as a classifier so that the feature extractor and the classifier can be trained together.

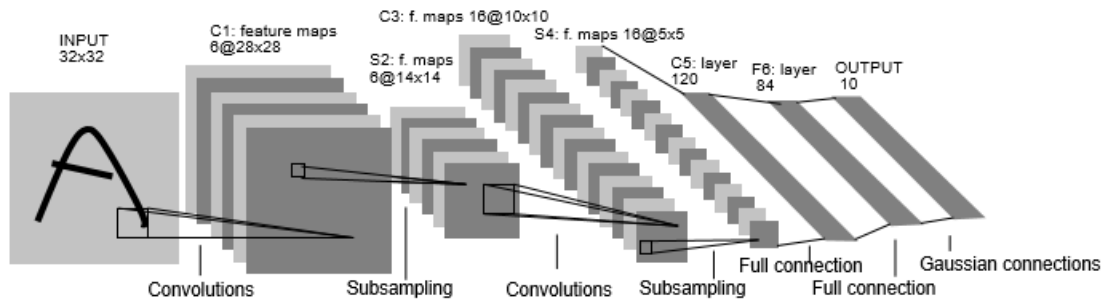


Figure 1.5 Architecture of LeNet-5 (LeCun et al. 1998).

Convolution neural networks are remarkable since it enables to extract high-level features as well as end-to-end learning framework, which breaks through the plateau in traditional statistical methods and structural methods. Inspired by the significant performance improvements, related works are in explosive growth.

1.2.3.2 Related Works on CNN-based Object Classification and Detection

The overall evolution of the CNNs architectures for image classification is going deeper and deeper, from LeNet-5 (LeCun et al. 1998) to AlexNet (Krizhevsky et al 2012), ZFNet (Zeiler and Fergus 2014), VGGNet (Russakovsky et al. 2015), GoogleNet (Szegedy et al. 2014) and ResNet

(He et al. 2016). In addition to deep structures, recent advances lie in many aspects including convolutional layer, pooling layer, activation function, loss function, regularization and optimization (Gu et al. 2018).

To focus the review in symbol classification and detection, only studies related to the challenges in this field will be reviewed. Some studies have been focused on to improve CNNs in the invariance to scaling, rotation and translation. Lenc and Vedaldi (2015) proposed a transformation layer to study the equivariance in CNNs, and the results showed the invariance is not always increasing with depth. Some researchers focused on further generalizing CNNs, which can extract feature maps invariant to the symmetry transform (Gens and Domingos 2014; Cohen and Welling 2015). Recently, the spatial transformer network (STN), a differentiable module which can be injected into CNNs, was developed by Jaderberg et al. (2015) to increase the networks performance related to geometric invariance. The invariance is achieved by three components: (1) localization network – estimates the parameters of affine transformation; (2) grid generator – generates the coordinates after affine transformation; (3) bilinear sampler – uses bilinear interpolation to round the generated coordinates to the integers. This spatial transformer will be applied in this paper to test its performance improvement in the symbol classification task.

In addition to learning features invariant to affine transform, another challenge lies in the symbol detection. As mentioned in 1.2.1, symbol detection in technical drawings is not trivial owing to the textureless nature of drawings, but recent advances in CNNs enable to encode rich representations in local regions, which can solve this problem. One of the famous object detectors is Region-based CNN (R-CNN), in that Selective Search is applied to extract region proposals, and then the object in the proposed region is classified using CNNs (Girshick et al. 2014). But the limitation of this network is that the input image must have a fixed size. To overcome this limitation, He et al. (2015) proposed spatial pyramid pooling network (SPP), which can encode the entire image into a fixed-length vector, but the training process is still in multiple stages. Fast RCNN (Girshick 2015) and Faster RCNN (Ren et al. 2015) are end-to-end solutions, and Faster RCNN is better since it spots the regions based on a region proposal network instead of Selective Search, reducing the computation burden. Furthermore, recent studies in object detection are notable for the one-shot paradigm, such as SSD (Liu et al. 2016) and YOLO (Redmon et al. 2015; Redmon and Farhadi 2017; Redmon 2018), which are much faster.

Recently, some researchers have begun to apply state-of-the-art networks for symbol recognition and detection. Elyan et al. (2018) presented a semi-automatic and heuristic-based approach to localize symbols in the drawings, and CNNs are applied in the symbol classification. Rahul et al. (2019) proposed an end-to-end pipeline to extract information in the piping and instrumentation diagram (P&ID), and symbol detection is achieved via FCN. Both experiments showed relatively good performance on their own datasets.

In conclusion, studies on CNNs have proved that the connectionist architecture significantly boosts the performance in image classification and object detection, as well as in symbol recognition and detection. In this paper, CNNs augmented with a spatial transformer will be applied to classify symbols in various scales and orientations. Besides, Faster-RCNN will be implemented for fast symbol spotting in piping drawings.

1.2.4 Review of Data Augmentation

For symbol classification and detection, one of the limitations of basic CNN architecture is that the learned features are not invariant to geometric transformation; thus, this study will use data augmentation to improve the invariance of the model and enrich the dataset. Data augmentation is widely applied to create a larger dataset and alleviate the overfitting issue, and the following is a brief introduction about the state-of-the-art methods.

Traditional data augmentation is achieved by performing affine transformations, brightness and contrast changes, and noise addition. Affine transformation means that the images in the small dataset are duplicated by rotation, scaling, translation, and shearing. As an approximation of real-world scenarios, brightness and contrast changes are simulations of physical illumination. Similarly, noise addition is also necessary, and the noise types include but not limited to Gaussian, speckle, and pepper and salt. Another group of transformation derives from the prior domain knowledge, i.e. representations of symbols should be invariant to changes in line thickness, changes in angles between lines, etc. These augmentation strategies enable the learned model robust to different data.

Recent advance in data augmentation is to use generative adversarial networks (GANs) (Goodfellow et al. 2014), which can learn domain-invariant knowledge. For instance, Antoniou et al. (2017) proposed a generative model to do data augmentation. This model can take data from a

source domain and generate other within-class data items based on conditional GANs, and the experiment results showed the accuracy increase in many benchmark datasets.

In this paper, traditional data augmentation will be used in this paper to generate a larger dataset for robust symbol classification and detection.

1.3 Goal and Objectives

The overarching goal is to fully automate the process of detecting and classifying symbols in piping drawings. The specific objectives include 1) classification of piping symbols and 2) detecting and labeling piping symbols from drawings.

The first objective is to classify the piping symbols using convolutional neural networks, which can automatically learn the representations from the data. The networks used in this part are CNN and CNN + STN, and datasets are synthetic, including eight types of symbols in piping drawings. Then, an experiment about comparing the performance between CNN and CNN+STN will be designed to test the performance improvement of STN in the context of symbol classification problem. Chapter Two of this thesis is devoted to this problem.

The second objective is applying Faster RCNN to detect symbols in the drawings, considering the clutter scenario that symbols are connected with pipelines. Compared to the first objective, the symbol detection task is more advanced, including localizing the regions highly likely to contain symbols and classifying different symbols. The original dataset is created by sketching in AutoCAD software to simulate the real-world drawings, and seven symbols are used in this task. Besides, this dataset is augmented via affine transformation and noise addition. The experiment is conducted with a public repository of Faster RCNN on Tensorflow platform, and the model is trained and tested using the dataset of piping drawings.

1.4 Significance and Research Contributions

The research involves the design of CNN-based methods to recognize and spot symbols in the piping drawings. Compared to conventional approaches, this study explores the ability of CNNs for symbol recognition and detection in cluttered engineering drawings. Two specific outcomes are summarized:

First, CNN and STN are adopted for symbol classification. This combination exhibits the ability to learn invariance to affine transformation, which captures the characteristic of symbols in technical drawings. Specifically, two outstanding merits to apply this architecture in the context of piping symbols recognition are as follows: 1) the convolutional neural network can extract the features in an abstract level, and it enables to jointly train the feature extractor and the classifier; 2) the use of STN increases the ability of the model to learn invariance to translation, scale, rotation, and more generic warping. The adaptability of CNNs in symbol recognition is evaluated by recognition accuracy.

Second, Faster RCNN is applied to detect symbols in cluttered drawings. The benefits of this network are in threefold: 1) instead of searching for candidate boxes using selective search, the location of candidate boxes is based on the classification of background and foreground using feature maps; 2) it tunes the locations of the bounding box using twice regression, which can increase the accuracy of localization; 3) this network consists of spatial pooling layers, which is capable of detecting symbols in multiple scales. It overcomes the challenge in the traditional statistical methods, such as finding local descriptors for the textureless drawings and choosing the window size.

The importance of fulfilling these two objectives is acknowledged in a wide range of fields, including the reduction of the data inaccessibility issues in facility management, the reconstruction of 3D models for existing buildings, and knowledge discovery in the design of piping systems. With accurately locating symbols in piping drawings, facility staff will be free from the tedious process of repeatedly digging into the superfluous documents so that the facility management process will be smoothed and streamlined.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows.

Chapter 2 aims to fulfill the objective of symbol recognition. The process of synthesizing the piping symbols is introduced. Then, the methods used to recognize the symbols in technical drawings are explained, including the design of network architecture, the loss function and the selection of the optimization method. Next, experiments are conducted to test the model performance of two models. Lastly, the rotation invariance of learned representation is evaluated

by the predicted score of the correct class, showing that learning rotation-invariant features is the challenge of using CNN models in symbol recognition.

Chapter 3 aims to explore the capability of Faster RCNN in the symbol detection problem. The architecture of Faster RCNN is introduced, and four main modules of this model are clearly explained. The next part includes data preparation and experiments. Data preparation includes the process of synthesizing piping drawings. For experimentation, the parameters and evaluation matrices used in this study are introduced. Based on the results, the conclusions and limitations of using Faster RCNN in symbol detection are discussed.

Chapter 4 includes summary, conclusions, limitations and future works. In this chapter, potential improvements on the dataset, the network architectures, and invariance measurements are discussed to increase the adaptability of models in recognizing and detecting symbols in piping drawings.

CHAPTER 2. CNN-BASED SYMBOL RECOGNITION

2.1 Introduction

In this chapter, the problem to be addressed is the symbol recognition in technical drawings. Symbol recognition is a task that is fundamental to drawing interpretation. The challenge in the computer-based symbol recognition in engineering drawings is to extract a set of robust features, which is invariant to geometric transformation, noise, and distortion, from variations/noises introduced during the symbol sketching process. Collectively, recognizing symbols and learning invariant representations are the focuses of this chapter.

In Section 2.2, the process of generating synthetic data is introduced. The synthetic data is used to explore the capability of CNN models in the context of symbol recognition. There are eight types of symbols in the dataset, including various valves, water pressure meter, and the indicator. The number of symbols in each class is 1000. The parameters for generating samples take account of the variations of symbols in the real piping drawings, including the rotation, scaling, translation, noises, etc., and the distribution of the parameters will be introduced in this section.

In Section 2.3, the methodology of symbol recognition is introduced. There are two models applied in this study, including a basic CNN model and the CNN model augmented with a spatial transformer network (STN) (Jaderberg et al. 2015). This section covers the design of the basic CNN and CNN+STN models, including the layers, the loss function, and the selection of the optimization method.

In Section 2.4, experiments based on two models are conducted on the Tensorflow platform. Random subsampling is used to provide an accurate estimate of the model performance. The experiment results show that the spatial transformer layer can improve the recognition accuracy from 95.39% to 98.26%.

In Section 2.5, sensitivity analysis is used to test the generalization of trained models to rotation transformation. By rotating the sample, the curve of the predicted score of the ground-truth class is generated, which measures the degree of rotation invariance of predictions using two models. The experimental results show that although data augmentation technology is applied, both models

lack the generalization capabilities to rotation transformation. Even a negligible rotation can significantly decrease the predicted score.

2.2 Data Preparation

In this section, the simulated datasets for experiments include eight symbols in total. The pipeline for synthesizing is to generate a normal symbol with parameters to control the variations, apply an affine transformation to this symbol, and then add noise, distortion, and dilation/erosion. The output is an image showing a symbol connected with pipelines, since in practice the symbol is not always well segmented, and bounding boxes are usually not rotated (except for some researches using rotated bounding boxes). Also, the simulated symbols are free of shearing and stretching because these transformations are scarcely seen in drawings. The overall flowchart is illustrated in Figure 2.1, showing the process of generating piping symbol datasets. The output is in a black background because the zero value is taken to be black.

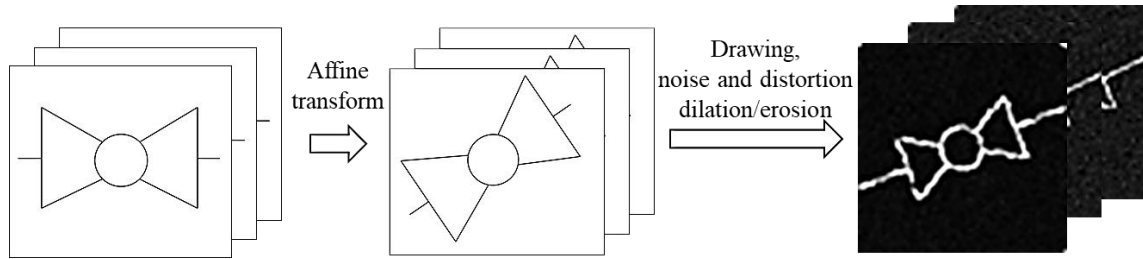


Figure 2.1 The pipeline of generating symbols

In the next subsections, the parameters used in the simulation process are discussed to explain how to control variations in the synthetic datasets.

2.2.1 Normal Synthetic Symbols

The normal synthetic symbol is that the symbol is not rotated, scaled, translated or noise added, etc., but there is still a need to customize the parameters of primitives and stroke width in this step since symbols are various from libraries to libraries. All symbols are sketched on a white canvas, which the size is 105 x105. The example of the simulation parameters is shown in Figure 2.2.

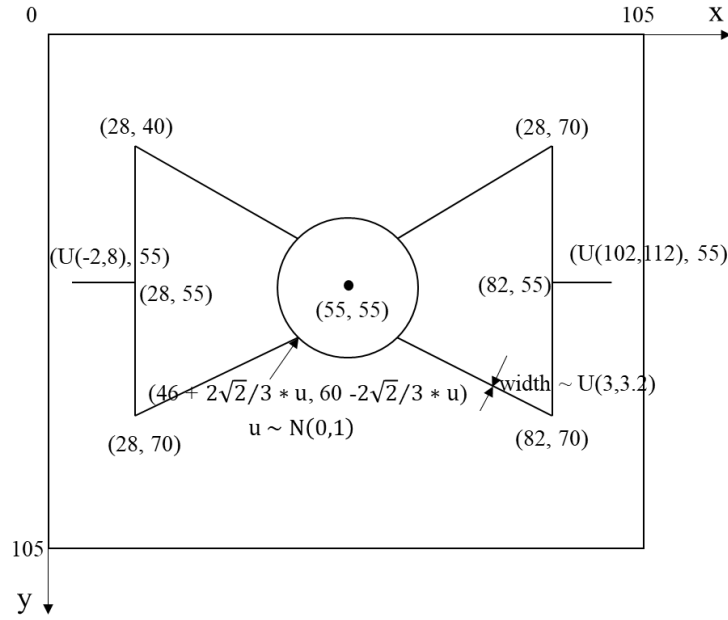


Figure 2.2 Example of parameters in a normal synthetic symbol

In this case, this symbol is symmetric. The parameter u , which is in a normal distribution, controls the diameter of the circle in the middle. Besides, the length of the pipeline connected to this symbol is uniformly distributed, and the stroke width is uniformly distributed from 3 to 3.2, which is visually similar to the stroke width in real drawings. For other types of symbols, there are other variations for primitive parameters, and most of them are consistent with piping symbols in the real world in the aspect of symmetricity and geometric constraints. The code for generating all symbols are listed in Appendix A.

2.2.2 Affine Transformation

Affine transformation is a necessary step for simulation since the symbols are usually in different orientations and scales. The affine transform matrix used in this study is indicated in Equation 2.1.

$$\begin{bmatrix} x_o \\ y_o \\ 1 \end{bmatrix} = \begin{bmatrix} s * \cos \alpha & \sin \alpha & t_x \\ -\sin \alpha & s * \cos \alpha & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \quad (2.1)$$

In Equation 2.1, (x_i, y_i) means the coordinates of the input, and (x_o, y_o) are the coordinates of the output. s is the scale value and α indicates the rotation angle, (t_x, t_y) is a pair of parameters, meaning the translation distance with respect to the origin. Since this step is performed before rasterizing, the interpolation is not needed. The parameters for affine transformation for all symbols are the same as listed in Table 2.1.

Table 2.1 Parameters for affine transformation

Parameter	Distribution
s	N (1.2, 1)
α	U (0, 360)
t_x	N (0, 400)
t_y	N (0, 400)

This dataset is designed for symbol recognition, which assumes that the symbol is segmented. Therefore, the variance of the scale of the symbol is small. The rotation angle is uniformly distributed from 0° to 360° , meaning that the generated samples can be presented in all orientations to ensure the model can be possibility generalized to rotation transformation. The translation parameters are selected considering the size of the canvas. Finally, the plot symbol will be resized into 60 x 60 and rasterized for further operations.

2.2.3 Additional Variations

Taking account of the real-world conditions, other variations are added, including noise, distortion and dilation/erosion operations. The ratios of augmented images and the parameters to control the strengths of these operations are listed in Table 2.2.

Table 2.2 Noise and distortion additions and dilation/erosion operations

Operation	Ratio	Type	Parameters
Noise	0.25	Speckle ($J = I + n * I$)	n is uniformly distributed with mean 0 and variance 0.05.
	0.25	Salt and pepper	Noise density=0.05
	0.5	Gaussian white noise	$N(0, 0.01)$
Distortion	0.5	Elastic deformation (Simard et al. 2003)	7 x 7 Gaussian filter with $\sigma = 10$ Scalar = 200
Dilation or Erosion	Dilation: 0.25	Line-shape structuring element	Line length =2 Orientation $\sim U(0, 360)$
	Erosion: 0.25	Line-shape structuring element	Line length =3 Orientation $\sim U(0, 360)$

The criteria for selecting parameters are to increase the randomness of samples but to ensure that the symbols can be still interpreted by a human.

2.3 Methodology

To recognize the piping symbols in paper drawings, the basic CNN model and CNN+STN model are used in this paper. The customization parts are the design of layers and the loss function, and selection of the optimization method. For these parts, some recent advances in designing the layers, such as batch normalization and ReLU, are applied in this model to improve the performance. Also, L2-norm is used in this study to reduce overfitting issues. Since learning is a process to minimize the loss function, the proper optimization method also needs to be selected.

2.3.1 The Design of the Architectures of the Basic CNN

The basic CNN architecture is shown in Table 2.3.

Table 2.3 The architecture of the basic CNN

Basic Convolution Network
Input 60x60 grayscale images
3 x 3 conv. 8 BN ReLU
2 x 2 Max pooling
3 x 3 conv. 16 BN ReLU
2 x 2 Max pooling
3 x 3 conv. 32 BN ReLU
2 x 2 Max pooling
2048 to 64 Dense
64 to 8 Dense
Softmax

$M \times M$ conv. N : a convolution layer (LeCun et al. 1998). $M \times M$ is the size of filters, and N is the number of filters convolving over the image or feature maps. Figure 2.3 shows how the convolution layer works. The operation is to multiply the weight and the corresponding pixel value in the input image, sum them and add a bias. The result will be the output. The padding area is used to ensure the size of the output is the same as the input. The tricky part of this operation is that it can encode the neighboring values into the center, which can capture the information in a local region. Also, a convolutional layer is another type of fully connected layer, which the weights are shared with other units. Since the grayscale image is used in this study, the number of outputs (feature maps) is equal to the number of the filters used. In this layer, weights and biases are the parameters; thus, millions of parameters will be generated from a deep neural network.

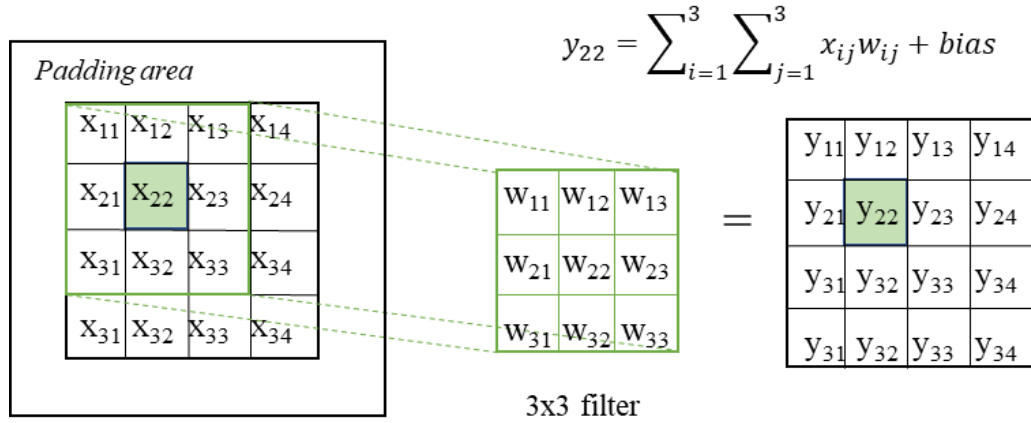


Figure 2.3 A convolutional layer using a 3x3 filter

BN: a batch normalization layer (Loffe and Szegedy 2015). The batch normalization layer is used to reduce the internal covariate shift, which is caused by the various distributions of input data from different batches. In deep networks, the change in the inputs will result in a problem, because all layers need to adapt to the new distribution, and the effects of distribution shift will be amplified down the network. So, the batch normalization layer is used to normalize batches into zero mean and unit variance approximately through several updates. The algorithm can be referred to this work (Loffe and Szegedy 2015).

ReLU: a rectified linear unit (Nair and Hinton 2010). It acts as an activation function, which can prune the negative values to zero and retain positive values. It is used to add the nonlinearity of the model.

$M \times M$ Max pooling: a subsampling layer. It takes the maximum value in a sub-region, which can reduce the features and computation complexity of the network. $M \times M$ is the region size. Figure 2.4 shows the mechanism of the Max pooling layer. The maximum value in the 2×2 matrix is taken as the output. So, after this layer, the width and height of the outputs will be reduced to half.

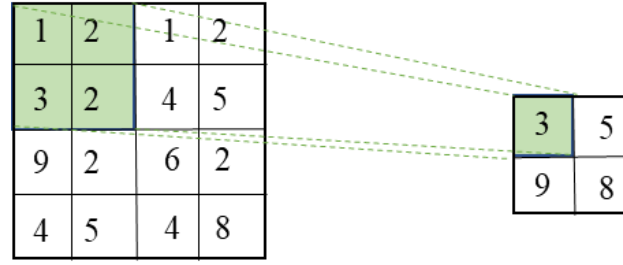


Figure 2.4 Max pooling using a 2 x 2 window and a stride size of one

Dense: a fully connected layer. M is the number of the input neurons, and N is the number of the output neurons.

Softmax: a classifier. It applies a standard exponential function to each element from input vectors and then normalizes these values to ensure the sum of the output equals to one. Equation 2.3 illustrates this process. So, the output of the softmax layer is the probability of the input belonging to each class, and the dimension of the output in this study is eight. In Equation 2.2, K is the number of the class. z is the input of the softmax layer, and $\sigma(z)_i$ is the output, which is the predicted score of the i^{th} class.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}} \quad \text{for } i = 1, \dots, K \text{ and } z = (z_1, \dots, z_K) \in R^K \quad (2.2)$$

2.3.2 The Design of the Architectures of the CNN + STN Model

The spatial transformer network (STN) is proposed by Jaderberg et al. (2015) to facilitate the geometric invariance learning by an embedded module. In this study, the spatial transformer network is embedded after the input layer as shown in Figure 2.5, so the size of the input and the size of the output of this network are the same.

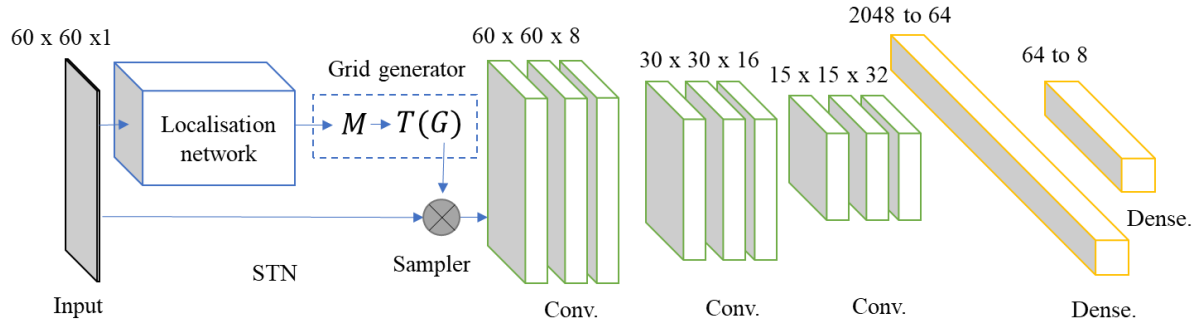


Figure 2.5 The architecture of CNN+STN model.

STN is composed of the localisation network, the grid generator and the sampler. M is the transformation matrix, and $T(G)$ represents the transformation operation is applied on the regular grid G .

This network consists of three components: (1) a localisation network to output the six parameters in affine transform; (2) a grid generator to produce the coordinates of the image after the transformation; (3) a bilinear sampler to round the coordinates to integers using the bilinear interpolation. The localisation network can be either a standard convolutional network or a fully connected network.

The architecture of the localisation network used in this paper is shown in Table 2.4.

Table 2.4 The architecture of the localisation network

The localisation network
Input 60x60 grayscale images
3 x 3 conv. 8 BN ReLU
2 x 2 Max pooling
3 x 3 conv. 16 BN ReLU
2 x 2 Max pooling
3600 to 64 Dense
64 to 6 Dense

The grid generator is to apply the pointwise transformation to the input image, and the transformation is illustrated in Equation 2.3.

$$\begin{pmatrix} x_o \\ y_o \end{pmatrix} = M \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \quad (2.3)$$

In Equation 2.4, M is the transformation matrix, which is regressed from the localisation network. (x_i, y_i) represents the coordinates of the input image. The coordinates of the output image, (x_o, y_o) , are the product of M and the pixel values in the input image.

The creativity of the spatial transformer network is that it builds up a differentiable module, so it can actively transform the images or feature maps to minimize the overall loss function.

2.3.3 The Design of the Loss Function

In addition to the design of networks architecture, the loss function is also critical to the performance of the model. The loss function used in this paper is illustrated in Equation 2.4. In this equation, θ represents all the parameters used in the model.

$$\text{loss function} = \sum_{i=1}^N H(\mathbf{y}_i, \hat{\mathbf{y}}_i) + 0.01 * \|\theta\|_2 \quad (2.4)$$

The fitting term, $H(\mathbf{y}, \hat{\mathbf{y}}_i)$, in the loss function, is the cross-entropy, which is commonly used in the multilabel classification problem. The principle of cross entropy is illustrated in Equation 2.5, which can be explained from the view of maximum likelihood.

$$\begin{aligned} \Pr(\hat{\mathbf{y}}_i | \theta) &= [\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^K] \text{ where } \sum_{j=1}^K \hat{y}_i^j = 1 \\ \Pr(\mathbf{y}_i | \theta) &= [y_i^1, y_i^2, \dots, y_i^K] \text{ where } \sum_{j=1}^K y_i^j = 1 \\ -\log(\Pr(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \dots, \hat{\mathbf{y}}_N | \theta)) &= -\log \left(\prod_{i=1}^N \Pr(\hat{\mathbf{y}}_i | \theta) \right) \\ &= -\sum_{i=1}^N \log \Pr(\hat{\mathbf{y}}_i | \theta) \end{aligned} \quad (2.5)$$

$$\begin{aligned}
&= - \sum_{i=1}^N \sum_{j=1}^K y_i^j \log \hat{y}_i^j \\
&= \sum_{i=1}^N H(\mathbf{y}, \hat{\mathbf{y}}_i)
\end{aligned}$$

In Equation 2.5, the index of the sample is denoted by i , and the index of the class is denoted by j . The number of classes is denoted by K . $\hat{\mathbf{y}}_i$ is the prediction vector in K dimensions, which indicates the probability of K classes for the i^{th} sample, In Equation 2.6, the vector \mathbf{y}_i is the ground-truth vector, which is a one-hot encoding of the i^{th} sample, in which one represents the label category and zero represents other categories.

In Equation 2.7, the number of samples is denoted by N . This equation shows that the sum of cross entropy, $H(\mathbf{y}, \hat{\mathbf{y}}_i)$, is equal to the negative log-likelihood over N samples, which means that to minimize the entropy is equal to maximize the log-likelihood of parameters. That is the reason why the cross-entropy is usually used as the fitting term in the cost function. In addition, the benefits of the log operation are in twofold: (1) it can reduce the production to summation, which is much easier to calculate; (2) if the number is very small, it will run out of the floating-point precision in the computer.

In addition to the fitting term, the L2 norm regularizer, $\|\boldsymbol{\theta}\|_2$, is added in the loss function to reduce the overfitting issues. L2 norm is used as a shrinkage of weights, which makes parameters much closer to zeros (but not exact zeros). This can reduce the complexity of the model and reduce the variances in the view of bias-variance tradeoff. So, adding the regularization term can help alleviate the overfitting issues.

2.3.4 The Selection of the Optimization Methods

The optimization method applied in this study is Adam (Kingma and Ba 2015). The main challenge in the optimization is to choose a proper direction and step size in each update. Adam is a complex algorithm, which takes advantage of both RMSProp and Momentum (Qian 1999). In the Momentum method, the update is based not only on the gradient but also on the previous movement. RMSProp is proposed by Hinton, which is to divide the learning rate by the root mean square (RMS) of the multiplication of the gradients and previous decayed gradients. Besides, the

originality of Adam is that it also incorporates a bias-correction step. Adam algorithm is computationally efficient and widely used in training deep neural networks.

2.4 Experiments

2.4.1 Experimental Setup

The application of the CNN is demonstrated in classifying eight piping symbols, including valves, the water pressure meter and the indicator, which are commonly used in the building plumbing system. The sample images for eight classes are shown in Figure 2.6.

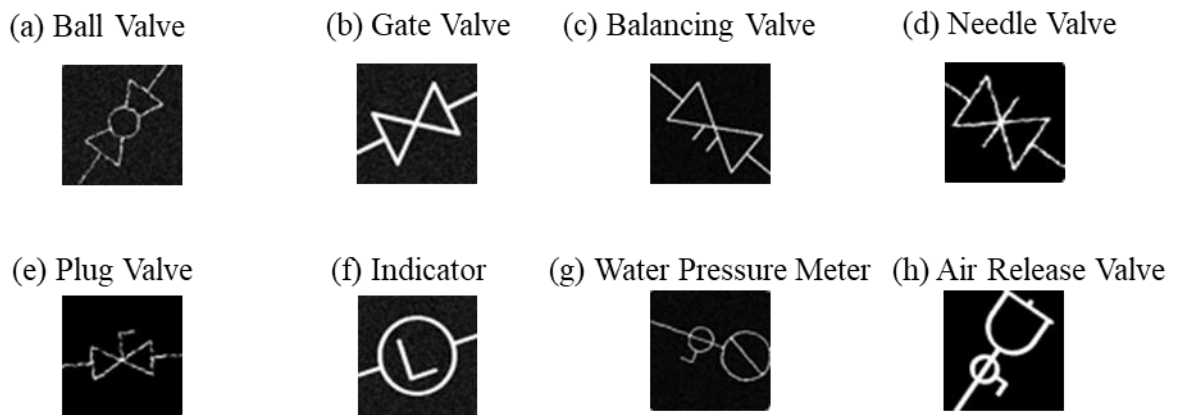


Figure 2.6 Sample images used in the dataset for symbol recognition

The synthetic dataset includes 1000 samples for each class, and the size is $8 * 1000 = 8000$ in total. The ratio to split the training set, validation set, and testing set is 3:1:1, and the size of the input image is 60×60 . All the input images are grayscale images, and the pixel intensity was scaled down to $[0, 1]$. The dataset was shuffled first and then randomly split into training, validation, and testing set for five times. This can reduce the biases in reporting the result, which gives a realistic estimation of the predictions. The CNN is developed using Tensorflow API, which is an open source platform for machine learning. Multiple experiments were performed to select the parameters in the neural networks, such as the number of layers, learning rate, batch size and so on. Based on the experiments, the best set of parameters are selected for two models, which is listed in Table 2.5.

Table 2.5 Experimental setup

Model	Learning rate	Batch size	Epoch
CNN	0.01	256	50
CNN + STN	0.0005	256	70

The learning rate for training the CNN + STN model is much smaller than a standard CNN model. The reason is that training with a larger learning rate will cause the loss function diverged. The batch size for both models is 256, which is selected as a compromise of the generalization capability and training efficiency. A larger batch size will make the loss converge faster. However, using a larger batch degrades generalization abilities of models (Keskar et al. 2016). For a standard CNN model, the loss is converged between 30 to 50 epochs; while for a CNN + STN model, the loss is converged between 50 to 70 epochs. So, the number of epochs is selected based on the loss convergence.

2.4.2 Experimental Results

The evaluation matrix is the accuracy and confusion matrix. The calculation of accuracy is shown in Equation 2.6.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.6)$$

In Equation 2.6, TP denotes the number of true positives, TN denotes true negatives, FP refers to the number of false positives, FN refers to the number of false negatives. The results of these are listed in Table 2.6, including the training, validation, and testing accuracies for five repeats. The mean and standard deviation for the accuracy is illustrated in Table 2.7.

Table 2.6 Experimental results – recognition accuracies

No.	CNN			CNN + STN		
	Training	Validation	Testing	Training	Validation	Testing
1	95.94	95.31	95.94	99.65	99.50	99.22
2	96.16	94.84	94.44	98.37	97.92	97.22
3	97.29	96.29	94.88	98.46	97.64	98.05
4	96.85	96.29	95.88	98.89	98.68	97.22
5	97.07	96.22	95.81	99.37	99.24	99.61

Table 2.7 The average and standard deviation of recognition accuracies for five repeats

Model	Training		Validation		Testing	
	Avg. (%)	Std. (%)	Avg. (%)	Std. (%)	Avg. (%)	Std. (%)
CNN	96.66	0.58	95.79	0.68	95.39	0.68
CNN +STN	98.94	0.56	98.59	0.81	98.26	1.11

For the standard CNN, the average training accuracies is 96.66%, while the average of the testing accuracies is 95.39%, which is a little smaller than the training accuracies. For CNN + STN, the average training accuracies is 98.94% and the average of testing accuracies is 98.26%. Based on the testing accuracies, the CNN + STN model improves the accuracy from 95.39% to 98.26%, which increases by 2.87%.

The visualization of the output of the spatial transformation network is attached in Appendix B, which some symbols showing interesting patterns of transformation.

The confusion matrices for CNN and CNN + STN are shown in Table 2.8 and Table 2.9.

Table 2.8 The confusion matrix for the CNN model

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
(a)	214	0	0	0	0	0	0	0
(b)	0	179	0	0	11	0	0	0
(c)	0	17	156	12	5	0	0	0
(d)	0	2	2	181	10	0	0	0
(e)	0	2	1	0	196	0	0	0
(f)	0	0	0	0	0	218	0	1
(g)	0	0	0	0	0	0	203	0
(h)	0	0	0	0	0	0	0	190

Table 2.9 The confusion matrix for the CNN + STN model

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
(a)	210	0	0	0	0	0	0	0
(b)	0	195	0	0	0	0	0	0
(c)	0	0	197	1	0	0	0	0
(d)	0	1	3	205	1	0	0	0
(e)	0	0	0	0	178	0	0	0
(f)	0	0	0	0	0	201	0	0
(g)	0	0	0	0	0	0	194	0
(h)	0	0	0	0	0	0	0	214

Based on Table 2.8, (b) gate valve, (c) balancing valve and (d) needle valve are easily misclassified. The reason may be that these three types of valves are visually similar. Based on Table 2.9, the model with spatial transformer shows a better result that the misclassification among these three types of valves is reduced. Therefore, the trained CNN + STN model is more robust to classifying visually similar valves.

2.5 Generalization Capabilities of CNNs to Rotations

2.5.1 Rotation Invariance

The formula for invariance is illustrated in Equation 2.7.

$$f(T(x)) = f(x) \quad (2.7)$$

Formally, f is invariant to transformations, if the output is identical for all transformations T of the input x (Schmidt and Roth 2012).

Generally, researchers focus on the invariance properties of features. In CNNs, the rotation invariance is not inherently satisfied, but the “approximately” invariance can be learned from the excessive training data (Schmidt and Roth 2012). So, data augmentation technologies are commonly used to enrich the data with various rotations. However, the shortcoming of this approach is that it is difficult to understand what invariant features are learned.

Learning robust representations invariant to rotations is a crucial problem in symbol recognition. Different from the objects in the natural scene, the symbols in drawings are likely to be rotated in different angles. However, without the transformation-invariant features, CNNs can be fooled with a simple transformation (Engstrom et al. 2018). Therefore, learning rotation-invariant descriptors is necessary for the symbol recognition problem.

In this paper, the prediction correctness of models reflects of rotation invariance of features. If the predictions are not invariant to rotations, it means the features learned from networks are not rotation invariant, which is the limitation of CNN-based methods.

2.5.2 Generalization Capabilities of CNNs to Rotations

In Section 2.4, the CNN + STN model shows the accuracy of 98.26% on the testing data, however, it cannot ensure the generality of the trained model. Inspired by the work of Azulay and Weiss (2018), the predicted score vector is used to analyze the sensitivity of both CNN and CNN + STN models to rotations. It is designed to test if the generalization capabilities of CNNs can be obtained using data augmentation and spatial transformer module.

The predicted score vector is the output of a model, which indicates the probability of the symbol belonging to the corresponding class.

The experiment tests if rotation transform attacks the CNN and CNN + STN models, which have been trained in Section 2.4. Three samples are used, and each sample is rotated counterclockwise with an interval of five degrees. The range of rotation is from 0° to 180° . By feeding the samples into the network, the predicted scores of the correct class can be extracted from the outputs of the softmax layer. Figure 2.7 shows the predictions of CNN and CNN + STN models.

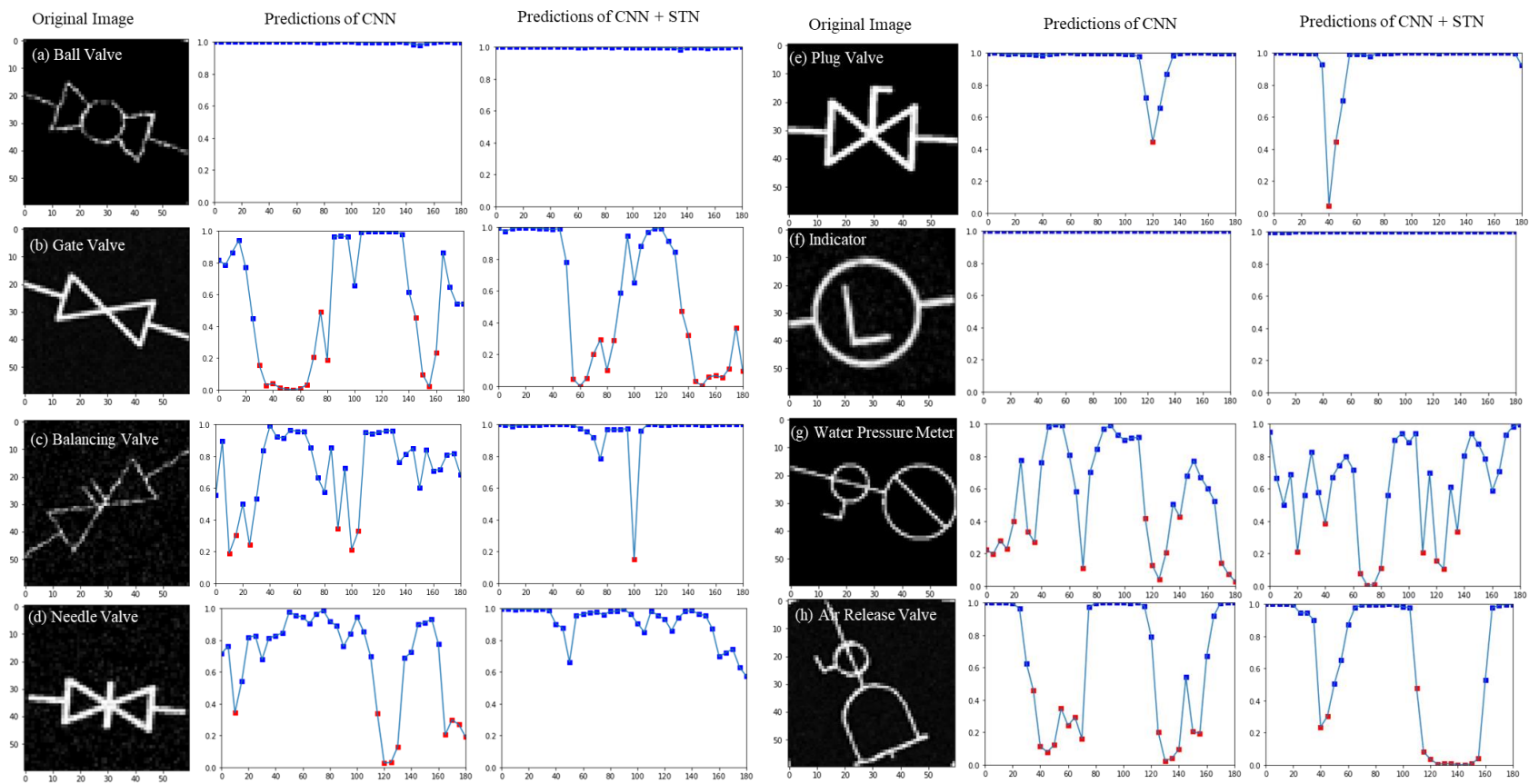


Figure 2.7 Predictions of CNN and CNN + STN models

Figure 2.7 shows the samples tested, the predicted scores of the correct class using the standard CNN model and the CNN + STN model. For the curves, the horizontal axis of the plots represents the rotation angle, and the vertical axis means the probability of the sample recognized as the correct class. The curve shows how the predictions change with rotations. The color of the square indicates if the rotated sample is correctly recognized or not. Blue means it is correctly classified, while red means it is wrongly classified as one of the other classes. For (a) ball valve and (f) indicator, the trained CNN + STN model shows a better performance in generalization to rotation transform, because the curve is almost a straight line and the probability of correct classification equals to one. The possible reason is that salient features are extracted from circles (in different scales for these two types of symbols), so the effects of rotation are reduced since the circle is inherently invariant to rotations. For other types of symbols, both models show jagged curves in predicting a correct class, meaning that they generalize poorly to rotation transformation. Also, the sharp shape (i.e sample (c) from 100° to 105°) in Figure 2.7 means that even a negligible rotation can significantly decrease the predicted score.

Generally speaking, this experiment shows the difficulty of convolutional networks in learning rotation-invariant features, and the invariance properties of learned features depend on the geometric pattern of symbols.

2.6 Summary and Conclusions

2.6.1 Summary

The focus of this chapter is to recognize symbols in piping drawings using CNNs. The symbols dataset is simulated based on the geometric constraints, and then augmented with scaling, rotation, translation, and noises. Two models, CNN and CNN+STN, are applied to compare the model performances in symbol recognition. The experiments are conducted based on the synthetic dataset, and the results show that the accuracy improves 2.87% using the spatial transformer module. The numbering and contents of sections are summarized in the following.

In Section 2.2, the data preparation process is introduced. It covers how to generate synthetic data and the internal parameters for samples. Taking account of the variations in stroke width, the geometric relationship between primitives, affine transformation, noises, and distortions, the synthetic dataset is generated for experiments.

In Section 2.3, the methodology of symbol recognition is covered, including the design of network architectures, the cost function and the selection of the optimization method. The state-of-the-art models, CNN and CNN+STN, are used to recognize symbols in piping drawings. The principles of these methods are emphasized and clearly elaborated in this section.

In Section 2.4, experiment setup and results are stated. Experiments are implemented on the Tensorflow platform based on the CNN and CNN + STN models. Recognition accuracy is used as the evaluation matrix to compare the model performances. To further understand the spatial transformer network, the output of this module is visualized in Appendix B.

In Section 2.5, the generalization capabilities of CNNs to rotations are discussed. The limitation of CNN-based approaches is proposed by counterexamples, showing how prediction correctness changes with rotations.

2.6.2 Conclusions

This chapter presents a CNN-based method for recognizing symbols in piping drawings, and the spatial transformer is applied in this study. The basic CNN model consists of three layers of convolution, batch normalization, Relu, and max-pooling, and two fully connected layers, followed by a softmax layer for classification. The spatial transformer module is injected after the input layer and consists of three components: a localisation network, a grid generator and a sampler. The localisation network consists of two layers of convolution, batch normalization, Relu, and max-pooling, and two fully connected layers to output the six parameters in the affine matrix. This spatial transformer is differentiable, so it can actively transform the images or feature maps to help minimize the overall loss function of the network during training (Jaderberg et al. 2015).

Previous CNN-based approaches on symbol recognition only adopted data augmentation technologies to improve the generalization capabilities of CNNs to the rotation, scaling, and translation. The proposed method shows a gain of 2.87% with the spatial transformer in the recognition accuracy. Also, the CNN + STN model is more robust to classifying visually similar valves.

Compared with the traditional statistical methods, the main advantage of CNN-based methods is that the classification network can be potentially embedded into a larger network for the detection task, which will be discussed in Chapter 3.

2.6.3 Limitations

The limitation of this study is that the synthetic dataset is used to train and test the CNN models so that the trained models cannot be directly used on the real piping drawings.

2.6.4 Future Works

The first thing is to collect the real drawings because it is essential in symbol recognition whichever method is used. Also, there is a need to improve the network architecture to learn rotation-invariant descriptors, which is a challenge in CNNs. Recently, researchers have begun to study the transformation invariance in CNNs using rotated filters for convolutional layers (Marcos et al. 2016) or a transition layer which can transform the image into the Fourier space (Chidester et al. 2018). These are the candidate solutions to improve the model in learning rotation-invariant representations. Also, to measure the transformation invariance of the features, the quantified approaches, such as measuring the linearity of learned features under the transformation (Lenc and Vedaldi 2015), are needed.

CHAPTER 3. CNN-BASED SYMBOL DETECTION

3.1 Introduction

Symbol detection in piping drawings is the problem to be addressed in this chapter. Given a drawing in whole or in part, it can identify where the symbol is. The challenge in symbol detection derives from the paradox: to correctly recognize the symbols, one should be able to segment them; but to correctly segment them, one needs to recognize the symbols (Doermann and Tombre 2014). In this paper, the state-of-the-art CNN-based object detection methods are used to detect symbols in piping drawings and to explore the potential improvements.

In this paper, Faster Region-based Convolutional Neural Networks (Faster-RCNN) (Ren et al. 2016) is applied to detect symbols in piping drawings. Section 3.2 covers the methodology of Faster-RCNN, including the components of this network. By digging into the architecture of Faster-RCNN, the adaptability of this model to detect symbols can be discussed.

In Section 3.3, the data preparation and experiments are stated. The drawings are sketched using AutoCAD MEP software, and symbols are selected from the piping component library. Next, the sketched drawings are cropped into several patches and augmented with affine transformation and noise addition. For experimentations, a public Tensorflow repository of Faster RCNN (Chen and Gupta 2017) is used for training and testing the generated dataset. The model performance is evaluated using the mean Average Precision (mAP) under three levels of Intersection over Union. The results show that piping symbols in a cluttered drawing can be detected and classified; however, some classes have low Average Precision (AP) owing to the small size of data.

3.2 Methodology

There are four components in Faster RCNN (Ren et al. 2016), including a basic convolutional network to extract feature maps, a region proposal network (RPN) to classify the background and foreground, an ROI pooling layer to generate fixed-length feature vectors, and a classification module. The feature maps generated by the basic convolutional network are share for both RPN and ROI pooling. Figure 3.1 illustrates the architecture of Faster RCNN.

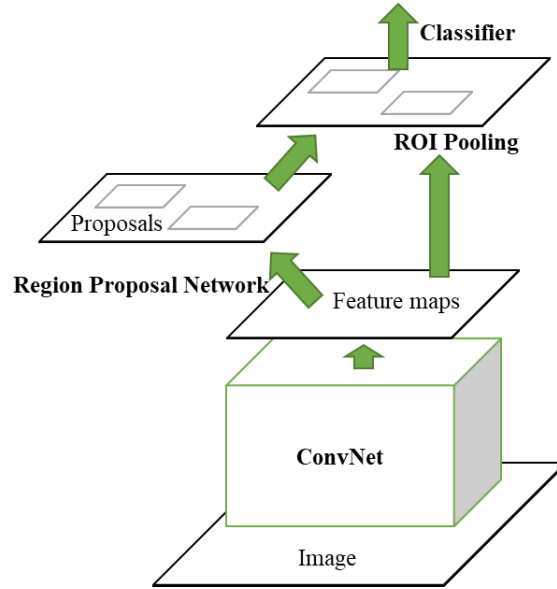


Figure 3.1 The architecture of the Faster RCNN network (Ren et al. 2016)

3.2.1 The Convolutional Network

The convolution network can be one of the popular deep CNNs, such as VGG, ZF, and ResNet. It is used to extract features from images, so the fully connected layers are dropped, and the outputs are feature maps. The convolutional network used in this paper is VGG16, since it shows good performance on the ImageNet dataset and the architecture is listed in Table 3.1.

Table 3.1 The architecture of VGG16 (Simonyan and Zisserman 2016)

VGG16
2 * conv3-64
max-pooling
2 * conv3-128
max-pooling
3 * conv3-256
max-pooling
3 * conv3-512
max-pooling
3 * conv3-512

After the backbone network, the size of the output feature maps is downscaled by after four layers of max pooling.

3.2.2 The Region Proposal Network

The Region Proposal Network (RPN) is critical to accelerating the region detection process. The architecture of this network is shown in Figure 3.2.

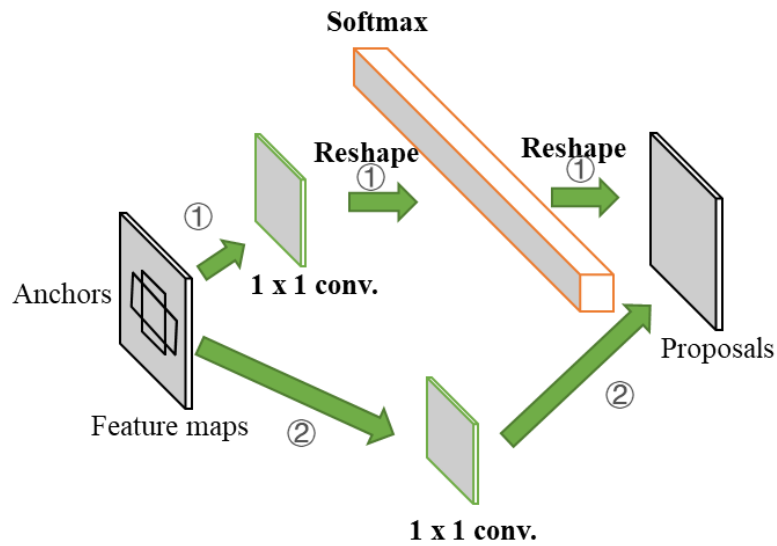


Figure 3.2 The architecture of RPN

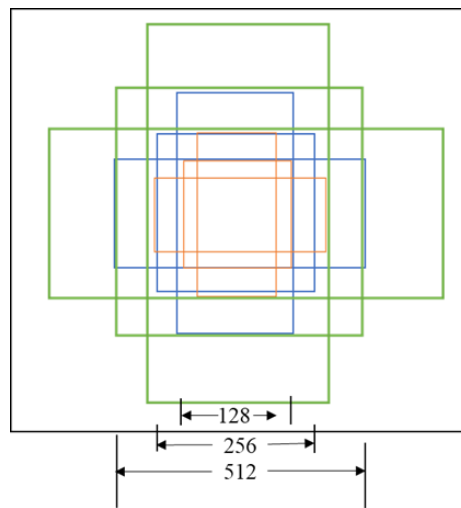


Figure 3.3 The anchors which scale $\in \{128, 256, 512\}$ and aspect ratio $\in \{0.5, 1, 2\}$

Based on Figure 3.2 and 3.3, 9 anchors are assigned to each point in feature maps, which equals to generate anchors with a stride of 16 on the original image. The aspect ratio of anchors belongs to $\{1:1, 1:2, 2:1\}$. Then, there are two paths. The first one is a classification layer, which can classify the foreground and background. So, the depth of feature maps is 18 (9 anchors x 2 labels), and the flattened feature vectors can be fed into the softmax layer for binary classification. The second path is for the bounding box regression. The proposals are adjusted by estimating the parameters of scaling and translation, as shown in Equation 3.1 and 3.2. (Girshick et al.2013)

$$\begin{aligned} A &= (A_x, A_y, A_w, A_h) \\ G &= (G_x, G_y, G_w, G_h) \end{aligned} \quad (3.1)$$

$$F(A_x, A_y, A_w, A_h) = (G'_x, G'_y, G'_w, G'_h) \approx (G_x, G_y, G_w, G_h)$$

$$\begin{aligned} G'_x &= A_w \cdot d_x(A) + A_x \\ G'_y &= A_h \cdot d_y(A) + A_y \\ G'_w &= A_w \cdot \exp(d_w(A)) \\ G'_h &= A_h \cdot \exp(d_h(A)) \end{aligned} \quad (3.2)$$

(x, y, w, h) represents the coordinates of the center, the width and the height of boxes. A represents the anchor box, and G represents the ground-truth box. G' is the regressed bounding box using function F . Then, the problem is formulated as a prediction of the transformation, which is (d_x, d_y, d_w, d_h) . (t_x, t_y, t_w, t_h) is similar to (d_x, d_y, d_w, d_h) but it is associated with G , not G' . The loss function and the smooth L1 loss is illustrated in Equation 3.3 (Girshick 2015).

$$\begin{aligned} L_{loc}(t, d) &= \sum_{i \in \{x, y, w, h\}} smooth_{L1}(t_i - d_i(A)) \\ smooth_{L1}(x) &= \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \end{aligned} \quad (3.3)$$

The loss function in this component consists of the loss of the bounding box location and the binary classification, and a parameter λ is used to balance these two tasks losses. In this study, $\lambda=1$. Based on the foreground anchor and bounding box regression, proposals are generated for the ROI pooling layer.

3.2.3 ROI Pooling

ROI pooling is applied to generate fixed-length feature vectors. The proposals are in different sizes, which cannot be fed into the convolutional networks directly. The feature maps in the region proposal are separated into the same number of grids, and a max pooling operation is applied on each grid. This can generate fixed-length representations, which can remove the fixed-size constraint of the network. The architecture of ROI pooling is shown in Figure 3.4.

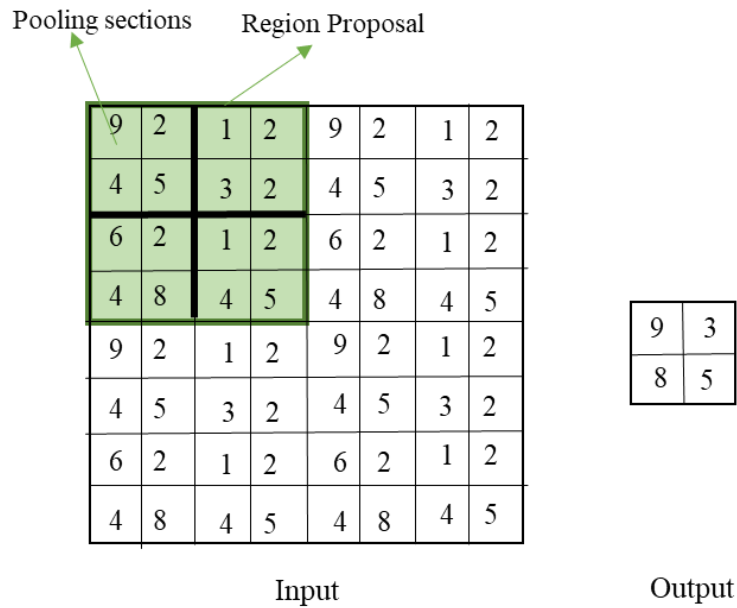


Figure 3.4 The architecture of ROI pooling

3.2.4 Classification

The classification is based on the feature maps of region proposals, including two modules which are classification and bounding box regression. The classification is achieved by a fully

connected layer and a softmax layer. The location of the bounding box is further tuned using regression, resulting in the final output.

3.3 Data Preparation and Experiments

3.3.1 Data Preparation

The data is generated by sketching a piping plan using AutoCAD MEP software, and seven classes of symbols in drawings are chosen from the default library of piping components. Sketching takes account of the connectivity between symbols and pipelines, the cluttered environment with the symmetry axis and terminal symbols, to simulate the real drawings. The total number of drawings is fourteen with different scales, and then they are cropped into patches and augmented with rotation, scaling, and noise addition. Finally, 225 images are generated for training, validation, and testing.

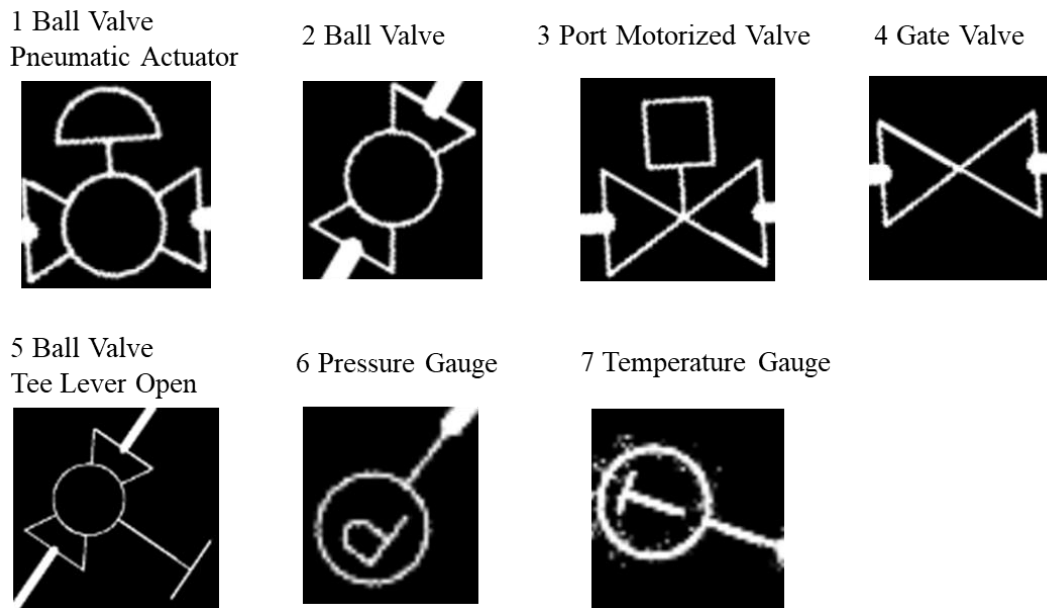


Figure 3.5 Seven symbols used in drawings

PASCAL VOC format is used to store the dataset, including a folder for images, a folder for XML files, and a folder for TXT files. PASCAL VOC is a dataset which is a popular used in detection and segmentation. XML files store all the information about an image, such as the classes

of objects and the locations of the bounding boxes. TXT files indicate which set and which class the sample image belonging to. LabelImg, a graphical image annotation tool, is used for labeling. The annotations of the objects can be saved as XML files in PASCAL VOC format. Then, based on the XML files, the corresponding TXT files are generated automatically by parsing the information in XML files. This is the general pipeline of generating a dataset in PASCAL VOC format.

The numbers of symbols for each class are summarized in Figure 3.6.

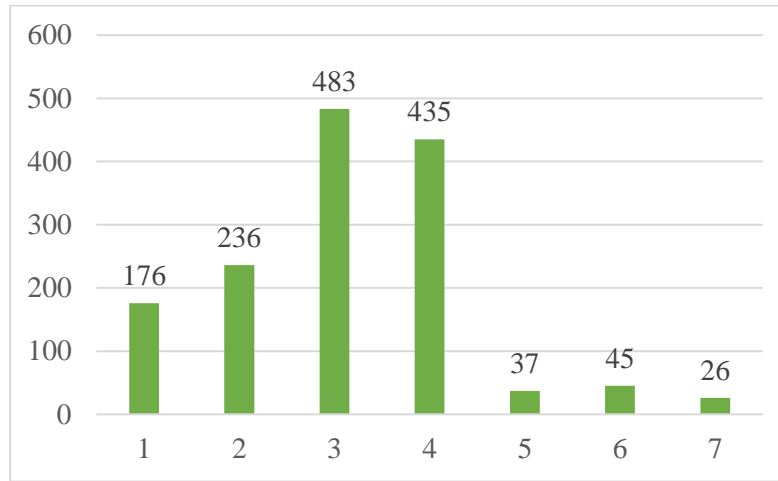


Figure 3.6 The distribution of the numbers of symbols for each class

Based on Figure 3.6, the size of this dataset is small, especially for symbol 5, 6 and 7. This means the trained model is likely to be overfitting. This dataset is used to test the performance of this model in symbol detection, and it will be enlarged in the future.

3.3.2 Experimental Setup

Several parameters are needed to set up the experiment. First, the ratio between the validation set and training set is 0.3, and the ratio between the testing set and the sum of the validation set and training set is 0.3. This model is trained with a full batch, and the total number of iterations is 70000. The optimization method used in this chapter is the stochastic gradient descent with momentum method (SGDM) (Qian 1999), and the initial learning rate is set to 0.001. The scales of anchors are {128, 256, 512} and the aspect ratios are {0.5, 1, 2}, so 9 anchors are generated for each grid.

The Average Precision (AP) is the area under the precision-recall curve. The evaluation matrix for symbol detection is the mean Average Precision (mAP), which is the mean of APs for seven classes. Formally, the formula for precision, recall, AP and mAP is illustrated in Equation 3.4.

$$\begin{aligned}
 \text{Precision} &= \frac{TP}{TP+FP} \\
 \text{Recall} &= \frac{TP}{TP+FN} \\
 \text{AP} &= \int_0^1 p(r)dr \\
 \text{mAP} &= \frac{\sum_{i=1}^K AP_i}{K}, \text{ K is the number of classes}
 \end{aligned} \tag{3.4}$$

The localization task is typically evaluated on the Intersection over Union threshold (IoU). The formula for IoU is illustrated in Equation 3.14. Area of interaction means the overlap of the predicted bounding box and the ground truth, and area of a union means the union of the predicted bounding box and the ground truth.

$$\text{IoU} = \frac{\text{Area of Interaction}}{\text{Area of Union}} \tag{3.5}$$

3.3.3 Experimental Results

The experiment is designed to detect seven symbols in piping drawings, and the IoU levels are set to 0.5, 0.6 and 0.7 so that the average precision is calculated based on these three levels. These IoU levels are commonly used in the researches using the PASCAL VOC dataset. Table 3.2 shows the experiment results, including the mAP and the AP for each class.

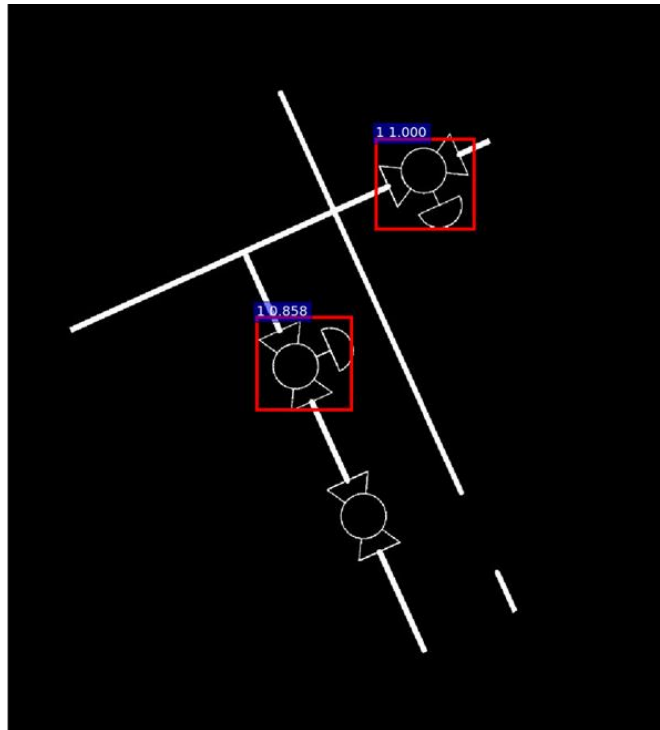
Table 3.2 Results for piping symbols with Faster RCNN detectors and VGG16.

No.	$AP^{IoU=0.5}$	$AP^{IoU=0.6}$	$AP^{IoU=0.7}$
Symbol 1	90.9	90.9	90.9
Symbol 2	90.8	90.8	90.4
Symbol 3	90.5	90.5	90.1
Symbol 4	88.9	88.9	80.9
Symbol 5	67.8	67.8	31.6
Symbol 6	94.7	94.7	59.1
Symbol 7	56.2	56.2	45.5
mAP	82.8	82.8	69.8

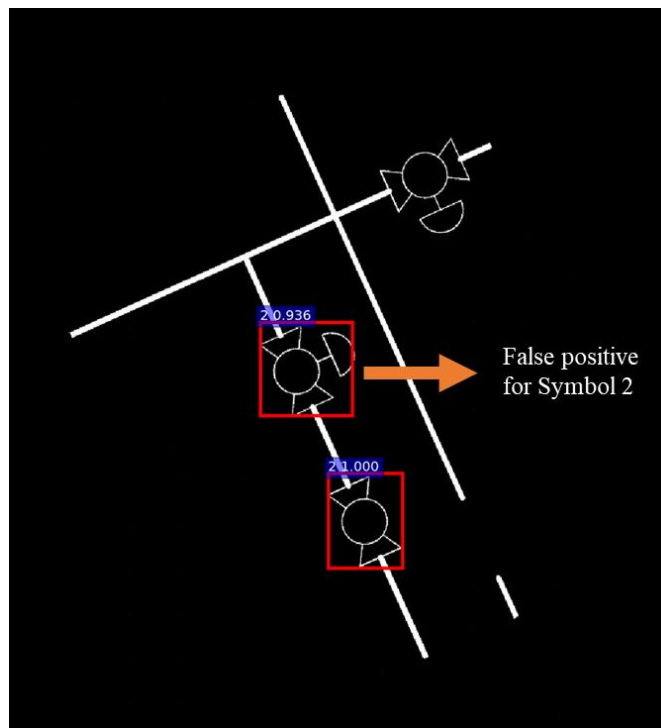
Based on Table 3.2, Faster RCNN achieves good results on the symbol 1-4 with APs are greater than 0.8 when IoU equals to 0.7. For symbol 6, the AP dropped abruptly when IoU changes from 0.6 to 0.7. It means the predicted class is correct, but the location of the bounding box is not regressed well. Relatively, Faster RCNN model shows the undesired result on symbol 5-7. The reasons for this failure might be the imbalanced data or the small size of the data as shown in Figure 3.6. Thinking of the symbol 7, which contains only 27 samples. Suppose the ratio for training, validation, and testing for this symbol is the same the ratio for drawing images. In this case, only 6 samples are used for testing, the estimation will introduce a large bias. Also, the instances of the minority class are oversampling, leading to overfitting issues.

Overall, this problem can be solved by increasing the number of symbols in a small size, so data augmentation technologies can be used to enrich the dataset. This issue will also occur in real drawings. Some of the classes are frequently used in drawings, but some are rarely used. Therefore, this issue needs to be solved whether synthetic data or the real data are used.

The samples for true positive and false positive are shown in Figure 3.7 to visualize the results.



(a) Symbol 1 detections with $p(\text{symbol 1} | \text{box}) \geq 0.8$



(b) Symbol 2 detections with $p(\text{symbol 2} | \text{box}) \geq 0.8$

Figure 3.7 The samples of true positive and false positive

There is an interesting observation that the symbol 2 is misclassified as symbol 1, but the locations of bounding boxes of this symbol in true positive and false positive cases are the same. The possible reason is that they are originally from different anchors, and finally adjusted to this location after the regression. The reason of this kind of errors needs to be further studied with visualizing how the location of the bounding box changes in different stages (i.e comparing the outputs of the bounding box locations after the first regression and the second regression).

3.4 Summary and Conclusions

3.4.1 Summary

The focus of this chapter is to detect symbols in piping drawings using Faster RCNN. The dataset is generated by sketching in AutoCAD MEP software, and symbols are selected from the default library. The results of experiments on Faster RCNN show its effectivity in the symbol detection task, but the Average Precision of the minority class is severely dropped by imbalanced data or small size of samples. Therefore, future works can be focused on labeling more objects and reducing the effects of the imbalanced data in drawings.

In Section 3.2, the methodology of symbol detection is introduced. The model used for symbol detection is Faster RCNN, which is a unified network composed of four modules. The principles of the four modules are clearly explained. The advantages of this model are discussed in this section.

In Section 3.3, the dataset is prepared by sketching in AutoCAD software, and symbols are selected from the default piping library. Next, the sketched drawings are cropped into several patches and augmented with affine transformation and noise addition. For the experiment part, the public Tensorflow repository of Faster RCNN is used in this study. Using this library, the model can be trained using the dataset of piping symbols. The results show that piping symbols in a cluttered drawing can be detected and classified using Faster RCNN; however, some classes have low AP owing to the imbalanced data, and this factor also needs to be considered when using real drawings.

3.4.2 Conclusions

This chapter mainly focuses on exploring the capability of Faster RCNN on the symbol detection task. The backbone network used in this experiment is VGG16. The scales of anchors are {128, 256, 512} and the aspect ratios are {0.5, 1, 2}. SGDM is applied to optimize the loss function, which consists of the loss for bounding box regression and the loss for classifications (binary in the region proposal network and multiclass in the classification module).

This method shows a good result with a mAP of 82.8% (IoU=0.5), which can be used in practice. The generalization of the model can be improved by labeling more symbols in various templates and libraries.

Region detection using traditional statistical methods is challenging owing to its textureless nature. Comparatively, Faster RCNN takes advantage of the regression of the bounding box, which makes the size and the location of windows more flexible. Also, it combines the location loss and classification loss into the cost function, so Faster RCNN can provide an accurate prediction of both the bounding box location and the symbol class via jointly minimizing the loss of locations and classification.

3.4.3 Limitations

This study is an exploratory work of symbol detection. The limitations are in two aspects: (1) the dataset is not balanced and too small, and it is not collected from real drawings; (2) the backbone network is too deep, leading to an overfitting problem.

3.4.4 Future Works

Future works are mainly in threefold: (1) collecting real drawings with different templates/libraries and labeling more symbols to enrich the dataset; (2) replacing the current backbone network into a smaller one, specifically, the number of layers is smaller; (3) combining other modules, which can improve the capabilities of learning invariance to geometric transformation.

CHAPTER 4. SUMMARY AND CONCLUSIONS

4.1 Summary and Conclusions

This study presents two topics, symbol classification, and detection based on CNN methods.

Chapter One of the thesis is devoted to providing an overview of the research problems, related works, and challenges that underpin my research significance and thereby the research objectives.

Chapter Two of the thesis proposes a CNN model, which is augmented with a Spatial Transformer Network (STN) for recognizing piping symbols. The merits of STN are dual: (1) it is a differentiable module, which can be embedded in CNNs; (2) it can actively learn the parameters in affine transformation matrix in the training process. The improvement of the proposed CNN model is evaluated using a synthetic dataset, including eight piping symbols. Compared with previous CNN-based approaches on symbol recognition, the proposed method shows a gain of 2.87% with the spatial transformer in the recognition accuracy. Also, the CNN + STN model is more robust to classifying visually similar valves.

Chapter Three of this thesis explores the capability of Faster RCNN model for detecting symbol in drawings. The dataset is generated by sketching drawings in AutoCAD software, selecting seven symbols from the piping component library and data augmentation. The experiment is conducted based on a public implementation of Faster RCNN, and the results are evaluated by mean Average Precision (mAP). The experiment results show a good result with a mAP of 82.8% (IoU=0.5).

4.2 Limitations

The limitation of this study lies into two points: (1) the dataset is not generated from real drawings with various templates/libraries, and the current dataset is too small for detection; (2) The network architecture and some parameters for symbol detection are not the best for this application.

4.3 Future Works

Future works on symbol recognition and detection are in four directions:

(1) Collect real drawings and enrich the dataset. The drawback of this study is that the synthetic data is used for training and testing, which is not substantial enough to support the conclusions. Although drawings are printed and human-crafted, there is still a need to collect data from various piping libraries.

(2) Improve the model in learning transformation invariance and measure the invariance of learned features. Researches related to improving the equivariance and invariance of CNNs are a hot topic in recent years. Researchers have begun to study the transformation invariance in CNNs using rotated filters for convolutional layers (Marcos et al 2016) or a transition layer which can transform the image into the Fourier space (Chidester et al. 2018). Sabour et al. (2017) proposed the CapsuleNet, which can actively predict the transformation matrix. These are the candidate solutions to improve the model in learning rotation-invariant representations. Also, to measure the transformation invariance of the features, the quantified approaches, such as measuring the linearity of learned features under the transformation (Lenc and Vedaldi 2015), are needed.

(3) Improve symbol detection via selecting a simple backbone network and proper parameters. The currently used backbone network is VGG16, which is commonly used on the ImageNet dataset. For piping symbols, the proper backbone network will be selected based on the experimentations.

(4) Improve symbol detection using the priors of the neighboring objects information. The information from neighbors is commonly used in scene understanding problems. Drawings are rich in the context information, such as the orientation of pipelines and the proximity of neighboring symbols. This kind of information can be incorporated into a detection network to improve symbol detection.

APPENDIX A. CODE

Matlab Code for Chapter 2 – Synthetic Symbols

```

main.m

%% parameters

clear all

close all

% author:Yuxi Zhang

% 1 Symbol

m=1;

K=1000;

for k=1:K

    u=randn*1/3;

    stk{m,1}{k,1}{1,1}=[(46+u*2*sqrt(2)),(60-u*2);28,70;28,40;(46+u*2*sqrt(2)),(50+u*2)];

    drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1,1},1,1);    %default    intv=1,

width=1

    stk{m,1}{k,1}{2,1}=[(46+2*sqrt(2)*u),(50+u*2);(46+u*2*sqrt(2)),(60-u*2);(64-

2*sqrt(2)*u),(60-u*2) ; (64-2*sqrt(2)*u),(50+u*2);(46+2*sqrt(2)*u),(50+u*2)];

    drawing{m,1}{k,1}{2,1}=sub_stroke_arc(stk{m,1}{k,1}{2,1},1,1);    %default    intv=1,

width=1

    stk{m,1}{k,1}{3,1}=[(64-2*sqrt(2)*u),(60-u*2);82,70;82,40;(64-2*sqrt(2)*u),(50+u*2)];

    drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3,1},1,1);    %default    intv=1,

width=1

    stk{m,1}{k,1}{4,1}=[82,55;82+20+rand*10,55];

    drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4,1},1,1);    %default    intv=1,

width=1

    stk{m,1}{k,1}{5,1}=[28,55;28-20-rand*10,55];

```

```

drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1);    %default    intv=1,
width=1

aff=true;
if (aff)
    close all
    tran_x=normrnd(0,1)*20;
    trans_y=normrnd(0,1)*20;
    theta=rand*360;
    scale=1.2+normrnd(0,0.05);
    interval=1;
    width=3+1/5*rand;
    fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
    drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

axis tight
axis equal
axis off;
Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels
%[Left Bottom Right Top] spacing
NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]
set(gca, 'Position', NewPos);
set(gcf,'color','w');
f = getframe(gcf); %# Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
% Distortion
improc=image{m,1}{k,1};

```



```

% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field
nx = norm(dx);
ny = norm(dy);
dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field
sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor
fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx
fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: 2 * 3*ceil(std2(dx)) + 1
% = 3 sigma pixels in each direction + 1 to make an odd integer
fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

% The resulting displacement
[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels
improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end

```

```

% erosion or dilation
v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);
    improc = imdilate(bw,se);
end
improc=uint8(255 * improc);
%add noise
u=rand;
if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end
rot_angle(m,k)=theta;
improc=imresize(improc,[60 60]);
imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
close all
end

% 2 Symbol
m=2;
for k=1:K
    u=randn*2;

```

```

stk{m,1}{k,1}{1,1}=[55,55;28,(70-2*u);28,(40+2*u);55,55];
drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1},1,1); %default in tv=1,
width=1
stk{m,1}{k,1}{2,1}=[55,55;82,(70-2*u);82,(40+2*u);55,55];
drawing{m,1}{k,1}{2,1}=sub_stroke_line(stk{m,1}{k,1}{2},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{3,1}=[82,55;82+20+rand*10,55];
drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{4,1}=[28,55;28-20-rand*10,55];
drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1); %default intv=1,
width=1

aff=true;
if (aff)
    close all
    tran_x=normrnd(0,1)*20;
    trans_y=normrnd(0,1)*20;
    theta= rand*360;
    scale=1.2+normrnd(0,0.05);
    interval=1;
    width=3+1/5*rand;
    fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
    drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

axis tight
axis equal
axis off;

```

Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any axis labels

%[Left Bottom Right Top] spacing

NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]

set(gca, 'Position', NewPos);

set(gcf,'color','w');

f = getframe(gcf); %# Capture the current window

im{m,1}{k,1}=rgb2gray(f.cdata);

image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);

improc=image{m,1}{k,1};

% Compute a random displacement field

u=rand;

if u>0.5

dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)

dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

nx = norm(dx);

ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1

dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution

alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx

fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: $2 * 3 * \text{ceil}(\text{std2}(\text{dx})) + 1$

% = 3 sigma pixels in each direction + 1 to make an odd integer

```

fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

% The resulting displacement
[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels
improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end
% erosion or dilation
v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);
    improc = imdilate(bw,se);
end
improc=uint8(255 * improc);
%add noise
u=rand;
if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end

```

```

    rot_angle(m,k)=theta;
    improc=imresize(improc,[60 60]);
    imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
    close all

end

% 3 Symbol
m=3;
for k=1:K
    u=randn;
    stk{m,1}{k,1}{1,1}=[55,55;28,(70-2*u);28,(40+2*u);55,55];
    drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{2,1}=[55,55;82,(70-2*u);82,(40+2*u);55,55];
    drawing{m,1}{k,1}{2,1}=sub_stroke_line(stk{m,1}{k,1}{2},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{3,1}=[(50+2*u),(55-(15-2*u)*(5-2*u)/27);(50+2*u),40];
    drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{4,1}=[(60-2*u),(55-(15-2*u)*(5-2*u)/27);(60-2*u),40];
    drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{5,1}=[82,55;82+20+rand*10,55];
    drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{6,1}=[28,55;28-20-rand*10,55];
    drawing{m,1}{k,1}{6,1}=sub_stroke_line(stk{m,1}{k,1}{6},1,1);    %default    intv=1,
width=1
    aff=true;
    if (aff)

```

```

close all
tran_x=normrnd(0,1)*20;
trans_y=normrnd(0,1)*20;
theta=normrnd(0,0.2)*360;
scale=1.2+normrnd(0,0.05);
interval=1;
width=3+1/5*rand;
fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

```

```

axis tight
axis equal
axis off;
Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels
%[Left Bottom Right Top] spacing
NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]
set(gca, 'Position', NewPos);
set(gcf,'color','w');
f = getframe(gcf); %# Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
improc=image{m,1}{k,1};
% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

```

```

% Normalizing the field
nx = norm(dx);
ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx
fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: 2 * 3*ceil(std2(dx)) + 1
% = 3 sigma pixels in each direction + 1 to make an odd integer

fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

% The resulting displacement

[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels

improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end

```



```

% erosion or dilation
v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);
    improc = imdilate(bw,se);
end
improc=uint8(255 * improc);
%add noise
u=rand;
if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end
rot_angle(m,k)=theta;
improc=imresize(improc,[60 60]);
imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
close all

end

% 4 Symbol
m=4;
for k=1:K

```

```

u=randn*2;
stk{m,1}{k,1}{1,1}=[55,55;28,(70-2*u);28,(40+2*u);55,55];
drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1},1,1); %default in tv=1,
width=1
stk{m,1}{k,1}{2,1}=[55,55;82,(70-2*u);82,(40+2*u);55,55];
drawing{m,1}{k,1}{2,1}=sub_stroke_line(stk{m,1}{k,1}{2},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{3,1}=[55,(70-2*u);55,(40+2*u)];
drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{4,1}=[82,55;82+20+rand*10,55];
drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{5,1}=[28,55;28-20+rand*10,55];
drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1); %default intv=1,
width=1
aff=true;
if (aff)
    close all
    tran_x=normrnd(0,1)*20;
    trans_y=normrnd(0,1)*20;
    theta= rand*360;
    scale=1.2+normrnd(0,0.05);
    interval=1;
    width=3+1/5*rand;
    fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
    drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

axis tight
axis equal

```

```

%xlim([-50 150])
%ylim([-50 150])
%u=randn;
set(gcf, 'Position', [100, 100, 150+5*u, 150+5*u]);
axis off;
Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels
%[Left Bottom Right Top] spacing
NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]
set(gca, 'Position', NewPos);
set(gcf, 'color', 'w');
f = getframe(gcf); %% Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
improc=image{m,1}{k,1};
% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

nx = norm(dx);
ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

```

```

sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx
fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: 2 * 3*ceil(std2(dx)) + 1
% = 3 sigma pixels in each direction + 1 to make an odd integer

fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

% The resulting displacement

[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels

improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end
% erosion or dilation
v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);

```

```

        improc = imdilate(bw,se);
    end
    improc=uint8(255 * improc);
    %add noise
    u=rand;
    if u<0.25
        improc=imnoise(improc,'speckle');
    elseif u>0.75
        improc=imnoise(improc,'salt & pepper');
    elseif u<0.5 && u>0.25
        improc=imnoise(improc,'gaussian');
    end
    rot_angle(m,k)=theta;
    improc=imresize(improc,[60 60]);
    imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
    close all

end

% 5 Symbol
m=5;
for k=1:K
    u=randn;
    stk{m,1}{k,1}{1,1}=[55,55;28,(70-2*u);28,(40+2*u);55,55];
    drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1,1},1,1);    %default    in    tv=1,
width=1
    stk{m,1}{k,1}{2,1}=[55,55;82,(70-2*u);82,(40+2*u);55,55];
    drawing{m,1}{k,1}{2,1}=sub_stroke_line(stk{m,1}{k,1}{2,1},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{3,1}=[55,55;55,(80-2*u);(70+2*u),(80-2*u)];

```

```

        drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3},1,1);    %default    intv=1,
width=1
        stk{m,1}{k,1}{4,1}=[82,55;82+20+rand*10,55];
        drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1);    %default    intv=1,
width=1
        stk{m,1}{k,1}{5,1}=[28,55;28-20-rand*10,55];
        drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1);    %default    intv=1,
width=1
        aff=true;
        if (aff)
            close all
            tran_x=normrnd(0,1)*20;
            trans_y=normrnd(0,1)*20;
            theta= rand*360;
            scale=1.2+normrnd(0,0.05);
            interval=1;
            width=3+1/5*rand;
            fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
            drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
        end

        axis tight
        axis equal
        axis off;

        Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels

        %[Left Bottom Right Top] spacing
        NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]
        set(gca, 'Position', NewPos);
        set(gcf,'color','w');

```

```

f = getframe(gcf); %# Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
    improc=image{m,1}{k,1};
% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

nx = norm(dx);
ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx
fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: 2 * 3*ceil(std2(dx)) + 1
% = 3 sigma pixels in each direction + 1 to make an odd integer

fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

```

```

% The resulting displacement

[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels

improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end
% erosion or dilation
v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);
    improc = imdilate(bw,se);
end
improc=uint8(255 * improc);
%add noise
u=rand;
if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end

```



```

    rot_angle(m,k)=theta;
    improc=imresize(improc,[60 60]);
    imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
    close all

end

% 6 Symbol
m=6;
for k=1:K
    u=rand;
    v=rand;
    stk{m,1}{k,1}{1,1}=[55,25;25,55;55,85;85,55;55,25];
    drawing{m,1}{k,1}{1,1}=sub_stroke_arc(stk{m,1}{k,1}{1},1,1);    %default    in    tv=1,
width=1
    stk{m,1}{k,1}{2,1}=[(45-2*u),(40-2*v);(45-2*u),(70+2*v)];
    drawing{m,1}{k,1}{2,1}=sub_stroke_line(stk{m,1}{k,1}{2},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{3,1}=[(45-2*u),(40-2*v);(65+2*u),(40-2*v)];
    drawing{m,1}{k,1}{3,1}=sub_stroke_line(stk{m,1}{k,1}{3},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{4,1}=[85,55;85+20+rand*10,55];
    drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{5,1}=[25,55;25-20+rand*10,55];
    drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1);    %default    intv=1,
width=1
    aff=true;
    if (aff)
        close all
        tran_x=normrnd(0,1)*20;

```

```

trans_y=normrnd(0,1)*20;
theta=rand*360;
scale=1.2+normrnd(0,0.05);
interval=1;
width=3+1/5*rand;
fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

```

```

axis tight
axis equal
axis off;

Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels

%[Left Bottom Right Top] spacing
NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]

set(gca, 'Position', NewPos);
set(gcf,'color','w');
f = getframe(gcf); %# Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
improc=image{m,1}{k,1};

% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

```

```

nx = norm(dx);
ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',7); % 2-D Gaussian filtering of dx
fdy = imgaussfilt(dy,sig,'FilterSize',7); % 2-D Gaussian filtering of dy

% Filter size: 2 * 3*ceil(std2(dx)) + 1
% = 3 sigma pixels in each direction + 1 to make an odd integer

fdx = alpha*fdx; % Scaling the filtered field
fdy = alpha*fdy; % Scaling the filtered field

% The resulting displacement

[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));

% Applying the displacement to the original pixels

improc = griddata(x-fdx,y-fdy,double(improc),x,y);
improc(isnan(improc)) = 0;
end
% erosion or dilation

```

```

v=rand;
if v<0.25
    bw=imbinarize(improc,0.7);
    se=strel('line',3,rand*360);
    improc = imerode(bw,se);
elseif v>0.75
    bw=imbinarize(improc,0.7);
    se=strel('line',2,rand*360);
    improc = imdilate(bw,se);
end
improc=uint8(255 * improc);
%add noise
u=rand;
if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end
rot_angle(m,k)=theta;
improc=imresize(improc,[60 60]);
imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
close all

end

% 7 Symbol
m=7;
for k=1:K
    stk{m,1}{k,1}{1,1}=[55,55;55,30-rand*10];

```

```

drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1},1,1); %default in tv=1,
width=1
stk{m,1}{k,1}{2,1}=[55,40;50,45;55,50;60,45;55,40];
drawing{m,1}{k,1}{2,1}=sub_stroke_arc(stk{m,1}{k,1}{2},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{3,1}=[55,55;45,65;55,75;65,65;55,55];
drawing{m,1}{k,1}{3,1}=sub_stroke_arc(stk{m,1}{k,1}{3},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{4,1}=[60,45;65,45;65,40];
drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1); %default intv=1,
width=1
stk{m,1}{k,1}{5,1}=[60,(65+5*sqrt(3));50,(65-5*sqrt(3))];
drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5},1,1); %default intv=1,
width=1
aff=true;
if (aff)
  close all
  tran_x=normrnd(0,1)*20;
  trans_y=normrnd(0,0.1)*20;
  theta=rand*360;
  scale=1.8+normrnd(0,0.05);
  interval=1;
  width=3+1/5*rand;
  fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
  drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

axis tight
axis equal
axis off;

```

```

Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any
axis labels
%[Left Bottom Right Top] spacing
NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]
set(gca, 'Position', NewPos);
set(gcf, 'color', 'w');
f = getframe(gcf); %# Capture the current window
im{m,1}{k,1}=rgb2gray(f.cdata);
image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);
improc=image{m,1}{k,1};
% Compute a random displacement field
u=rand;
if u>0.5
dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)
dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

nx = norm(dx);
ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1
dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution
alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',15); % 2-D Gaussian filtering of dx

```

```
fdy = imgaussfilt(dy,sig,'FilterSize',15); % 2-D Gaussian filtering of dy
```

```
% Filter size: 2 * 3*ceil(std2(dx)) + 1
```

```
% = 3 sigma pixels in each direction + 1 to make an odd integer
```

```
fdx = alpha*fdx; % Scaling the filtered field
```

```
fdy = alpha*fdy; % Scaling the filtered field
```

```
% The resulting displacement
```

```
[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));
```

```
% Applying the displacement to the original pixels
```

```
improc = griddata(x-fdx,y-fdy,double(improc),x,y);
```

```
improc(isnan(improc)) = 0;
```

```
end
```

```
% erosion or dilation
```

```
v=rand;
```

```
if v<0.25
```

```
    bw=imbinarize(improc,0.7);
```

```
    se=strel('line',3,rand*360);
```

```
    improc = imerode(bw,se);
```

```
elseif v>0.75
```

```
    bw=imbinarize(improc,0.7);
```

```
    se=strel('line',2,rand*360);
```

```
    improc = imdilate(bw,se);
```

```
end
```

```
improc=uint8(255 * improc);
```

```
%add noise
```

```
u=rand;
```

```

if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end
rot_angle(m,k)=theta;
improc=imresize(improc,[60 60]);
imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
close all

end

% 8 Symbol
m=8;
for k=1:K
    u=rand;
    v=rand;
    stk{m,1}{k,1}{1,1}=[55,55;55,30-rand*10];
    drawing{m,1}{k,1}{1,1}=sub_stroke_line(stk{m,1}{k,1}{1},1,1);    %default    in    tv=1,
width=1
    stk{m,1}{k,1}{2,1}=[55,40;50,45;55,50;60,45;55,40];
    drawing{m,1}{k,1}{2,1}=sub_stroke_arc(stk{m,1}{k,1}{2},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{3,1}=[65,65;55,55;45,65];
    drawing{m,1}{k,1}{3,1}=sub_stroke_arc(stk{m,1}{k,1}{3},1,1);    %default    intv=1,
width=1
    stk{m,1}{k,1}{4,1}=[60,45;65,45;65,40];
    drawing{m,1}{k,1}{4,1}=sub_stroke_line(stk{m,1}{k,1}{4},1,1);    %default    intv=1,
width=1

```



```

stk{m,1}{k,1}{5,1}=[45,65;45,(75+2*u)];
drawing{m,1}{k,1}{5,1}=sub_stroke_line(stk{m,1}{k,1}{5,1},1,1);    %default    intv=1,
width=1
stk{m,1}{k,1}{6,1}=[65,65;65,(75+2*u)];
drawing{m,1}{k,1}{6,1}=sub_stroke_line(stk{m,1}{k,1}{6,1},1,1);    %default    intv=1,
width=1
stk{m,1}{k,1}{7,1}=[(44-2*v),(75+2*u);(66+2*v),(75+2*u)];
drawing{m,1}{k,1}{7,1}=sub_stroke_line(stk{m,1}{k,1}{7,1},1,1);    %default    intv=1,
width=1
stk{m,1}{k,1}{8,1}=[55,(75+2*u);55,(77+2*u)];
drawing{m,1}{k,1}{8,1}=sub_stroke_line(stk{m,1}{k,1}{8,1},1,1);    %default    intv=1,
width=1

aff=true;
if (aff)
    close all
    tran_x=normrnd(0,1)*20;
    trans_y=normrnd(0,1)*20;
    theta=rand*360;
    scale=1.8+normrnd(0,0.05);
    interval=1;
    width=3+1/5*rand;
    fnc= @(stk) affine_fun(stk, tran_x,trans_y,55,55,theta,scale,scale,interval,width);
    drawing{m,1}{k,1}=apply_to_nested(drawing{m,1}{k,1},fnc);
end

axis tight
axis equal
axis off;

```

Tight = get(gca, 'TightInset'); %Gives you the bording spacing between plot box and any axis labels

%[Left Bottom Right Top] spacing

NewPos = [Tight(1) Tight(2) 1-Tight(1)-Tight(3) 1-Tight(2)-Tight(4)]; %New plot position
[X Y W H]

set(gca, 'Position', NewPos);

set(gcf,'color','w');

f = getframe(gcf); %% Capture the current window

im{m,1}{k,1}=rgb2gray(f.cdata);

image{m,1}{k,1}=imresize(im{m,1}{k,1},[224 224]);

improc=image{m,1}{k,1};

% Compute a random displacement field

u=rand;

if u>0.5

dx = -1 + 2*rand(size(improc)); % dx ~ U(-1,1)

dy = -1 + 2*rand(size(improc)); % dy ~ U(-1,1)

% Normalizing the field

nx = norm(dx);

ny = norm(dy);

dx = dx./nx; % Normalization: norm(dx) = 1

dy = dy./ny; % Normalization: norm(dy) = 1

% Smoothing the field

sig = 10; % Standard deviation of Gaussian convolution

alpha = 200; % Scaling factor

fdx = imgaussfilt(dx,sig,'FilterSize',15); % 2-D Gaussian filtering of dx

```
fdy = imgaussfilt(dy,sig,'FilterSize',15); % 2-D Gaussian filtering of dy
```

```
% Filter size: 2 * 3*ceil(std2(dx)) + 1
```

```
% = 3 sigma pixels in each direction + 1 to make an odd integer
```

```
fdx = alpha*fdx; % Scaling the filtered field
```

```
fdy = alpha*fdy; % Scaling the filtered field
```

```
% The resulting displacement
```

```
[y,x] = ndgrid(1:size(improc,1),1:size(improc,2));
```

```
% Applying the displacement to the original pixels
```

```
improc = griddata(x-fdx,y-fdy,double(improc),x,y);
```

```
improc(isnan(improc)) = 0;
```

```
end
```

```
% erosion or dilation
```

```
v=rand;
```

```
if v<0.25
```

```
    bw=imbinarize(improc,0.7);
```

```
    se=strel('line',3,rand*360);
```

```
    improc = imerode(bw,se);
```

```
elseif v>0.75
```

```
    bw=imbinarize(improc,0.7);
```

```
    se=strel('line',2,rand*360);
```

```
    improc = imdilate(bw,se);
```

```
end
```

```
improc=uint8(255 * improc);
```

```
%add noise
```

```
u=rand;
```

```

if u<0.25
    improc=imnoise(improc,'speckle');
elseif u>0.75
    improc=imnoise(improc,'salt & pepper');
elseif u<0.5 && u>0.25
    improc=imnoise(improc,'gaussian');
end
rot_angle(m,k)=theta;
improc=imresize(improc,[60 60]);
imwrite((255-improc),[num2str(m),'_',num2str(k),'.png']);
close all

end

```

sub_stroke_line.m

```

function stk=sub_stroke_line(p,intv,width)
n=length(p);
stk=cell(n-1,1);
for i=2:n
    %% compute distance between each point
    dist = zeros(n,1);
    x1 = p(i,:);
    x2 = p(i-1,:);
    dist(i) = norm(x1-x2);

    %% Generate uniform points
    nint = round(dist(i)/intv);
    nint = max(nint,2);
    xi = linspace(0,dist(i),nint); %nint: number of points
    stk{i-1} = interp1([0;dist(i)],[p(i-1,:); p(i,:)],xi);
    if(stk{i-1}(end,:)~=p(i,:))

```

```

        stk{i-1}=[stk{i-1};p(i,:)];
    end
    plot(stk{i-1}(:,1),stk{i-1}(:,2),'k','LineWidth', width);
    hold on
    plot(p(i-1,1),p(i-1,2),'.','MarkerEdgeColor','black','MarkerSize', pi*width);
    hold on
end

```

```

%% Plot

```

```

plot(p(n,1),p(n,2),'.','MarkerEdgeColor','black','MarkerSize', pi*width);

```

```

set(gca,'XTick',[],'YTick',[]);

```

```

xlim([1 105]);

```

```

ylim([1 105]);

```

```

hold on

```

sub_stroke_arc.m

```

function stk = sub_stroke_arc(p,intv,width)

```

```

n=length(p);

```

```

if n<3

```

```

    msg=['At least 3 points'];

```

```

    fprintf(1,[msg,'\n']);

```

```

end

```

```

stk=cell(n-1,1);

```

```

xcyc=cell(n-2,1);

```

```

R=cell(n-2,1);

```

```

for i=1:n-2

```

```

%% Compute Circle Center and Radius R, xcyc

```

```

% FIT_CIRCLE_THROUGH_3_POINTS

```

```

% Mathematical background is provided in

```

<http://www.regentsprep.org/regents/math/geometry/gcg6/RCir.htm>

```

%
% Input:
%
% ABC is a [3 x 2n] array. Each two columns represent a set of three points which lie on
% a circle. Example: [-1 2;2 5;1 1] represents the set of points (-1,2), (2,5) and (1,1) in
Cartesian
% (x,y) coordinates.
%
% Outputs:
%
% R is a [1 x n] array of circle radii corresponding to each set of three points.
% xcyc is an [2 x n] array of the centers of the circles, where each column is [xc_i;yc_i]
where i
% corresponds to the {A,B,C} set of points in the block [3 x 2i-1:2i] of ABC
%
% Author: Danylo Malyuta.
% Version: v1.0 (June 2016)
% -----
% Each set of points {A,B,C} lies on a circle. Question: what is the circles radius and center?
% A: point with coordinates (x1,y1)
% B: point with coordinates (x2,y2)
% C: point with coordinates (x3,y3)
% ===== Find the slopes of the chord A<-->B (mr) and of the chord B<-->C (mt)
% mt = (y3-y2)/(x3-x2)
% mr = (y2-y1)/(x2-x1)
% /// Begin by generalizing xi and yi to arrays of individual xi and yi for each {A,B,C} set
of points provided in ABC array

x1 = p(i,1:2:end);
x2 = p(i+1,1:2:end);
x3 = p(i+2,1:2:end);

```

```

y1 = p(i,2:2:end);
y2 = p(i+1,2:2:end);
y3 = p(i+2,2:2:end);
% /// Now carry out operations as usual, using array operations
mr = (y2-y1)./(x2-x1);
mt = (y3-y2)./(x3-x2);
% A couple of failure modes exist:
% (1) First chord is vertical    ==> mr==Inf
% (2) Second chord is vertical  ==> mt==Inf
% (3) Points are collinear      ==> mt==mr (NB: NaN==NaN here)
% (4) Two or more points coincident ==> mr==NaN || mt==NaN
% Resolve these failure modes case-by-case.
idf1 = isinf(mr); % Where failure mode (1) occurs
idf2 = isinf(mt); % Where failure mode (2) occurs
idf34 = isequaln(mr,mt) | isnan(mr) | isnan(mt); % Where failure modes (3) and (4) occur
% ===== Compute xc, the circle center x-coordinate
xcyc{i}(1) = (mr.*mt.*(y3-y1)+mr.*(x2+x3)-mt.*(x1+x2))./(2*(mr-mt));
xcyc{i}(idf1) = (mt(idf1).*(y3(idf1)-y1(idf1))+(x2(idf1)+x3(idf1)))/2; % Failure mode (1)
==> use limit case of mr==Inf
xcyc{i}(idf2) = ((x1(idf2)+x2(idf2))-mr(idf2).*(y3(idf2)-y1(idf2)))/2; % Failure mode (2)
==> use limit case of mt==Inf
xcyc{i}(idf34) = NaN; % Failure mode (3) or (4) ==> cannot determine center point, return
NaN
% ===== Compute yc, the circle center y-coordinate
xcyc{i}(2,:) = -1./mr.*(xcyc{i}-(x1+x2)/2)+(y1+y2)/2;
idmr0 = mr==0;
xcyc{i}(2,idmr0) = -1./mt(idmr0).*(xcyc{i}(idmr0)-
(x2(idmr0)+x3(idmr0))/2)+(y2(idmr0)+y3(idmr0))/2;
xcyc{i}(2,idf34) = NaN; % Failure mode (3) or (4) ==> cannot determine center point, return
NaN
% ===== Compute the circle radius

```

```

R{i} = sqrt((xcyc{i}(1,:)-x1).^2+(xcyc{i}(2,:)-y1).^2);
R{i}(idf34) = Inf; % Failure mode (3) or (4) ==> assume circle radius infinite for this case

```

```

%% Plot arc

```

```

intv_d=2*asin((1/2)/R{i})*intv;

if(i==1)
    %first segment
    if(p(i,2)<xcyc{i}(2,:))
        theta1=2*pi-acos((p(i,1)-xcyc{i}(1,:))/(R{i}));
    else
        theta1=acos((p(i,1)-xcyc{i}(1,:))/(R{i}));
    end
    if(p(i+1,2)<xcyc{i}(2,:))
        theta2=2*pi-acos((p(i+1,1)-xcyc{i}(1,:))/(R{i}));
    else
        theta2=acos((p(i+1,1)-xcyc{i}(1,:))/(R{i}));
    end
    if theta2>theta1
        theta2=theta2-2*pi;
    end
    th = theta2:intv_d:theta1;
    xunit = R{i} * cos(th) + xcyc{i}(1,:);
    yunit = R{i} * sin(th) + xcyc{i}(2,:);
    stk{2*i-1} = [xunit;yunit]';
    if(stk{2*i-1}(end,:)~=p(i,:))
        stk{2*i-1} = [stk{2*i-1};p(i,:)];
    end
    plot(xunit, yunit,'k','LineWidth', width);
    hold on
    plot(p(i+1,1), p(i+1,2),'.','MarkerEdgeColor','black','MarkerSize', pi*width);

```



```

    hold on
    plot(p(i,1), p(i,2), '.', 'MarkerEdgeColor', 'black', 'MarkerSize', pi*width);
    hold on
end
%second segment
if(p(i+1,2)<xcyc{i}(2,:))
    theta2=2*pi-acos((p(i+1,1)-xcyc{i}(1,:))/(R{i}));
else
    theta2=acos((p(i+1,1)-xcyc{i}(1,:))/(R{i}));
end
if(p(i+2,2)<xcyc{i}(2,:))
    theta3=2*pi-acos((p(i+2,1)-xcyc{i}(1,:))/(R{i}));
else
    theta3=acos((p(i+2,1)-xcyc{i}(1,:))/(R{i}));
end
if theta3>theta2
    theta3=theta3-2*pi;
end
th = theta3:intv_d:theta2;
xunit = R{i} * cos(th) + xcyc{i}(1,:);
yunit = R{i} * sin(th) + xcyc{i}(2,:);
stk{i+1}=[xunit;yunit]';
if(stk{i+1}(end,:)~=p(i+1,:))
    stk{i+1} = [stk{i+1};p(i+1,:)];
end
plot(xunit, yunit, 'k', 'LineWidth', width);
hold on
plot(p(i+1,1), p(i+1,2), '.', 'MarkerEdgeColor', 'black', 'MarkerSize', pi*width);
hold on
plot(p(i+2,1), p(i+2,2), '.', 'MarkerEdgeColor', 'black', 'MarkerSize', pi*width);
hold on

```

end

```
set(gca,'XTick',[],'YTick',[]);
xlim([1 105]);
ylim([1 105]);
hold on
```

affine_fun.m

```
function stk_affu=affine_fun(stk, tx,ty,cx,cy,theta,w,h,dint,width)
% Create translation matrix
tm= @(x,y)[1 0 0;0 1 0; x y 1];

% Create rotation matrix
rm = @(x,y,theta) tm(-x,-y)*[cosd(theta) -sind(theta) 0; sind(theta) cosd(theta) 0;0 0
1]*tm(x,y);

% Create scaling matrix
sm=@(x,y,w,h) tm(-x,-y)*[w 0 0;0 h 0; 0 0 1]*tm(x,y);

R=sm(cx,cy,w,h)*rm(cx,cy,theta)*tm(tx,ty);

% Rotate your point(s)
stk(:,3)=1;
stk_aff = stk*R;
stk_aff(:,3)=[];

% Plot
n=length(stk_aff);
j=2;
rm=[];
```

```

for i=1:n-1
    if(j-1<=n-1)
        i=j-1;
    else
        break
    end
    dist1(i+1)=0;
    while (dist1(i+1)<dint && j<=n)
        x1 = stk_aff(j,:);
        x2 = stk_aff(i,:);
        dist1(i+1) = norm(x1-x2);
        if(dist1(i+1)<dint && j<n)
            rm=[rm j];
        end
        j=j+1;
    end
end
stk_affu=stk_aff;
stk_affu(rm,:)=[];
[l m]=size(dist1);
rm=rm(rm<=m-1);
dist1(rm+1)=[];
cumdist = cumsum(dist1);
start_dist = cumdist(1);
end_dist = cumdist(end);
x = cumdist(:);
nint = round(end_dist/dint);
nint = max(nint,2);
xi = linspace(start_dist,end_dist,nint); %nint: number of points
stk_affu = interp1(x,stk_affu,xi);
plot(stk_aff(:,1),stk_aff(:,2),'k','LineWidth', width);

```

```

hold on
plot(stk_aff(1,1),stk_aff(1,2),'.','MarkerEdgeColor','black','MarkerSize', pi*width);
hold on
plot(stk_aff(end,1),stk_aff(end,2),'.','MarkerEdgeColor','black','MarkerSize', pi*width);
hold on
set(gca,'XTick',[],'YTick',[]);
xlim([1 105]);
ylim([1 105]);

```

Python Code for Chapter 2 – TFrecord file Generation

Datasetgenerator.py

```

#encoding=utf-8
import os
import tensorflow as tf
from PIL import Image

cwd = os.getcwd()

classes = {'1','2','3','4','5','6','7','8'}
def create_record():
    writer = tf.python_io.TFRecordWriter("train_final4.tfrecords")
    for index, name in enumerate(classes):
        class_path = cwd + "/STN_1000/" + name + "/"
        for img_name in os.listdir(class_path):
            img_path = class_path + img_name
            img = Image.open(img_path)
            img_raw = img.tobytes()
            # print (index,img_raw)
            example = tf.train.Example(
                features=tf.train.Features(feature={

```

```

        "label": tf.train.Feature(int64_list=tf.train.Int64List(value=[index])),
        'img_raw': tf.train.Feature(bytes_list=tf.train.BytesList(value=[img_raw]))
    )))
    writer.write(example.SerializeToString())
writer.close()

create_record()

```

Python Code for Chapter 2 – CNN

```

import tensorflow as tf
from tensorflow.keras import models, layers
import numpy as np
from stn import spatial_transformer_network as transformer
import matplotlib.pyplot as plt

def _parse_function(example_proto):
    features=tf.parse_single_example(
        example_proto,
        features={
            'label':tf.FixedLenFeature([],tf.int64),
            'img_raw': tf.FixedLenFeature([],tf.string)
        }
    )
    label=features['label']
    img=features['img_raw']
    img=tf.decode_raw(img,tf.uint8)
    img=tf.reshape(img,[60, 60, 1])
    img=tf.cast(img, tf.float32)*(1./255)
    label=tf.cast(label,tf.int32)
    return img, label

```

```
def data_iterator(tfrerecords, batch_size):
    full_dataset = tf.data.TFRecordDataset(tfrerecords)
    full_dataset = full_dataset.shuffle(buffer_size=8000, seed=1)
    train_dataset = full_dataset.take(train_size)
    test_dataset = full_dataset.skip(train_size)
    val_dataset = test_dataset.skip(val_size)
    test_dataset = test_dataset.take(test_size)
    train_dataset = train_dataset.map(_parse_function)
    val_dataset = val_dataset.map(_parse_function)
    test_dataset = test_dataset.map(_parse_function)
    train_iterator = train_dataset.repeat().batch(batch_size).make_initializable_iterator()
    val_iterator = val_dataset.repeat().batch(batch_size).make_initializable_iterator()
    test_iterator = test_dataset.batch(test_size).repeat().make_initializable_iterator()
    return train_iterator, val_iterator, test_iterator
```

```
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    axis = list(range(len(x.get_shape()) - 1))
    mean, var = tf.nn.moments(x, axis)
    size = len(mean.get_shape())
    scale = tf.Variable(tf.ones([size]))
    shift = tf.Variable(tf.zeros([size]))
    epsilon = 0.001
    x = tf.nn.batch_normalization(x, mean, var, shift, scale, epsilon)
    return tf.nn.relu(x)
```

```
def maxpool2d(x, k=2):
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME')
```

```

def conv_net(x, weights, biases):
    # here we call the conv2d function we had defined above and pass the input image x, weights
    # wc1 and bias bc1.
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and
    # outputs a 14*14 matrix.
    pl1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    # here we call the conv2d function we had defined above and pass the input image x, weights
    # wc2 and bias bc2.
    conv2 = conv2d(pl1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and
    # outputs a 7*7 matrix.
    pl2 = maxpool2d(conv2, k=2)

    #conv3 = conv2d(pl2, weights['wc3'], biases['bc3'])
    # Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and
    # outputs a 4*4.
    #pl3 = maxpool2d(conv3, k=2)

    #catlayer=tf.concat([tf.contrib.layers.flatten(pl1),tf.contrib.layers.flatten(pl2),tf.contrib.layers.flat
    #ten(pl3)],1)
    #fc1 = tf.reshape(catlayer,[-1, weights['wd1'].get_shape().as_list()[0]])
    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(pl2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])

```

```

# Output, class prediction
# finally we multiply the fully connected layer with the weights and add a bias term.
out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
return out, conv1

```

```
DATASET_SIZE=8000
```

```

train_size = int(0.6 * DATASET_SIZE)
val_size = int(0.2 * DATASET_SIZE)
test_size = int(0.2 * DATASET_SIZE)

```

```
batch_size=256
```

```

weights = {
    'wc1': tf.get_variable('W0', shape=(3,3,1,8), initializer=tf.contrib.layers.xavier_initializer()),
    'wc2': tf.get_variable('W1', shape=(3,3,8,16), initializer=tf.contrib.layers.xavier_initializer()),
    'wc3': tf.get_variable('W2', shape=(3,3,16,32), initializer=tf.contrib.layers.xavier_initializer()),
    'wd1': tf.get_variable('W3', shape=(15*15*16,64),
initializer=tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('W4', shape=(64,8), initializer=tf.contrib.layers.xavier_initializer()),
}
biases = {
    'bc1': tf.get_variable('B0', shape=(8), initializer=tf.contrib.layers.xavier_initializer()),
    'bc2': tf.get_variable('B1', shape=(16), initializer=tf.contrib.layers.xavier_initializer()),
    'bc3': tf.get_variable('B2', shape=(32), initializer=tf.contrib.layers.xavier_initializer()),
    'bd1': tf.get_variable('B3', shape=(64), initializer=tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('B4', shape=(8), initializer=tf.contrib.layers.xavier_initializer()),
}

```



```

# both placeholders are of type float
x = tf.placeholder("float", [None, 60,60,1])
y = tf.placeholder("float", [None, 8])
pred, conv1 = conv_net(x, weights, biases)
cost = (tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
        +0.01*tf.nn.l2_loss(weights['wc1'])
        +0.01*tf.nn.l2_loss(weights['wc2'])
        #+0.01*tf.nn.l2_loss(weights['wc3'])
        +0.01*tf.nn.l2_loss(weights['wd1'])
        +0.01*tf.nn.l2_loss(weights['out'])
    )

learning_rate=0.01
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

#Here you check whether the index of the maximum value of the predicted image is equal to the
actual labelled image. and both will be a column vector.
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
#calculate accuracy across all the given images and average them out.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

train_tfrecords='train_final4.tfrecords'
train_iterator,val_iterator,test_iterator=data_iterator(train_tfrecords,batch_size)
init = tf.global_variables_initializer()
train=train_iterator.get_next()
val=val_iterator.get_next()
test=test_iterator.get_next()
saver = tf.train.Saver()
with tf.Session() as sess:
    sess.run(init)
    train_loss = []

```

```

val_loss=[]
train_accuracy = []
val_accuracy=[]
summary_writer = tf.summary.FileWriter('./Output', sess.graph)
sess.run(train_iterator.initializer)
sess.run(val_iterator.initializer)
for epoch in range(50):
    sum_loss=0
    sum_acc=0
    avg_loss=0
    avg_acc=0
    sum_val_loss=0
    avg_val_loss=0
    sum_val_acc=0
    avg_val_acc=0
    for iteration in range(train_size//batch_size):
        train_x,train_y=sess.run(train)
        train_y=tf.one_hot(train_y, 8).eval()
        opt = sess.run(optimizer, feed_dict={x: train_x, y: train_y})
        loss, acc = sess.run([cost, accuracy], feed_dict={x: train_x, y: train_y})
        sum_loss=loss+sum_loss
        avg_loss=sum_loss/(iteration+1)
        sum_acc=acc+sum_acc
        avg_acc=sum_acc/(iteration+1)
        val_x,val_y=sess.run(val)
        val_y=tf.one_hot(val_y, 8).eval()
        valid_acc,valid_loss = sess.run([accuracy,cost], feed_dict={x: val_x,y : val_y})
        sum_val_loss=valid_loss+sum_val_loss
        avg_val_loss=sum_val_loss/(iteration+1)
        sum_val_acc=valid_acc+sum_val_acc
        avg_val_acc=sum_val_acc/(iteration+1)

```

```

train_loss.append(avg_loss)
val_loss.append(avg_val_loss)
train_accuracy.append(avg_acc)
val_accuracy.append(avg_val_acc)
print("Epoch " + str(epoch+1) + ", Loss= " + \
      "{:.6f}".format(avg_loss) + ", Training Accuracy= " + \
      "{:.5f}".format(avg_acc))
print("Epoch " + str(epoch+1) + ", Loss= " + \
      "{:.6f}".format(avg_val_loss) + \
      ", Validation Accuracy:", "{:.5f}".format(avg_val_acc))
summary_writer.close()
save_path = saver.save(sess, "./tmp/model4.ckpt")
sess.run(test_iterator.initializer)
test_x, test_y = sess.run(test)
test_y = tf.one_hot(test_y, 8).eval(session=sess)
test_acc, test_loss = sess.run([accuracy, cost], feed_dict={x: test_x, y: test_y})
print(test_acc)
print(test_loss)

```

Python Code for Chapter 2 – CNN +STN

```

import tensorflow as tf
from tensorflow.keras import models, layers
import numpy as np
from stn.spatial_transformer import transformer
import matplotlib.pyplot as plt

def _parse_function(example_proto):
    features = tf.parse_single_example(
        example_proto,
        features={

```

```

        'label':tf.FixedLenFeature([],tf.int64),
        'img_raw': tf.FixedLenFeature([],tf.string)
    }
)
label=features['label']
img=features['img_raw']
img=tf.decode_raw(img,tf.uint8)
img=tf.reshape(img,[60, 60, 1])
img=tf.cast(img, tf.float32)*(1./255)
label=tf.cast(label,tf.int32)
return img, label

def data_iterator(tfrecords,batch_size):
    full_dataset =tf.data.TFRecordDataset(tfrecords)
    full_dataset = full_dataset.shuffle(buffer_size=8000,seed=1)
    train_dataset = full_dataset.take(train_size)
    test_dataset = full_dataset.skip(train_size)
    val_dataset = test_dataset.skip(val_size)
    test_dataset = test_dataset.take(test_size)
    train_dataset=train_dataset.map(_parse_function)
    val_dataset=val_dataset.map(_parse_function)
    test_dataset=test_dataset.map(_parse_function)
    train_iterator=train_dataset.repeat().batch(batch_size).make_initializable_iterator()
    val_iterator=val_dataset.repeat().batch(batch_size).make_initializable_iterator()
    test_iterator=test_dataset.batch(test_size).repeat().make_initializable_iterator()
    return train_iterator, val_iterator, test_iterator

def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)

```

```

axis = list(range(len(x.get_shape()) - 1))
mean,var = tf.nn.moments(x, axis)
size = len(mean.get_shape())
scale = tf.Variable(tf.ones([size]))
shift = tf.Variable(tf.zeros([size]))
epsilon = 0.001
x = tf.nn.batch_normalization(x, mean, var, shift, scale, epsilon)
return tf.nn.relu(x)

```

```

def maxpool2d(x, k=2):
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1],padding='SAME')

```

```

def conv_net(x, weights, biases,loc_weights,loc_biases):
    h_fc1=loc_net(x, loc_weights, loc_biases)
    h_trans = transformer(x, h_fc1,[60,60])
    # here we call the conv2d function we had defined above and pass the input image x, weights
wc1 and bias bc1.
    conv1 = conv2d(h_trans, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and
outputs a 14*14 matrix.
    pl1 = maxpool2d(conv1, k=2)
    # Convolution Layer
    # here we call the conv2d function we had defined above and pass the input image x, weights
wc2 and bias bc2
    conv2 = conv2d(pl1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and
outputs a 7*7 matrix.
    pl2 = maxpool2d(conv2, k=2)
    conv3 = conv2d(pl2, weights['wc3'], biases['bc3'])

```

Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 4*4.

```
p13 = maxpool2d(conv3, k=2)
```

Fully connected layer

Reshape conv2 output to fit fully connected layer input

```
fc1 = tf.reshape(p13, [-1, weights['wd1'].get_shape().as_list()[0]])
```

```
fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
```

Output, class prediction

finally we multiply the fully connected layer with the weights and add a bias term.

```
out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
```

```
return out, h_fc1, h_trans
```

```
def loc_net(x, weights, biases):
```

here we call the conv2d function we had defined above and pass the input image x, weights wc1 and bias bc1.

```
conv1 = conv2d(x, weights['wc1'], biases['bc1'])
```

Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 14*14 matrix.

```
p11 = maxpool2d(conv1, k=2)
```

Convolution Layer

here we call the conv2d function we had defined above and pass the input image x, weights wc2 and bias bc2.

```
conv2 = conv2d(p11, weights['wc2'], biases['bc2'])
```

Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 7*7 matrix.

```
p12 = maxpool2d(conv2, k=2)
```

Max Pooling (down-sampling), this chooses the max value from a 2*2 matrix window and outputs a 4*4.

Fully connected layer

Reshape conv2 output to fit fully connected layer input

```

fc1 = tf.reshape(pl2, [-1, weights['wd1'].get_shape().as_list()[0]])
fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
# Output, class prediction
# finally we multiply the fully connected layer with the weights and add a bias term.
out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
return out

```

DATASET_SIZE=8000

```

train_size = int(0.6 * DATASET_SIZE)
val_size = int(0.2 * DATASET_SIZE)
test_size = int(0.2 * DATASET_SIZE)

```

```

batch_size=256
train_tfrecords='train_final4.tfrecords'
train_iterator,val_iterator,test_iterator=data_iterator(train_tfrecords,batch_size)
initial = np.array([1,0,0,0,1,0],dtype='float32')

```

```

weights = {
    'wc1': tf.get_variable('W0', shape=(3,3,1,8), initializer=tf.contrib.layers.xavier_initializer()),
    'wc2': tf.get_variable('W1', shape=(3,3,8,16), initializer=tf.contrib.layers.xavier_initializer()),
    'wc3': tf.get_variable('W2', shape=(3,3,16,32), initializer=tf.contrib.layers.xavier_initializer()),
    'wd1': tf.get_variable('W3', shape=(8*8*32,64), initializer=tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('W4', shape=(64,8), initializer=tf.contrib.layers.xavier_initializer()),
}

```

```

biases = {
    'bc1': tf.get_variable('B0', shape=(8), initializer=tf.contrib.layers.xavier_initializer()),
    'bc2': tf.get_variable('B1', shape=(16), initializer=tf.contrib.layers.xavier_initializer()),
    'bc3': tf.get_variable('B2', shape=(32), initializer=tf.contrib.layers.xavier_initializer()),
    'bd1': tf.get_variable('B3', shape=(64), initializer=tf.contrib.layers.xavier_initializer()),
}

```

```

    'out': tf.get_variable('B4', shape=(8), initializer=tf.contrib.layers.xavier_initializer()),
}

loc_weights = {
    'wc1': tf.get_variable('loc_W0', shape=(3,3,1,8),
initializer=tf.contrib.layers.xavier_initializer()),
    'wc2': tf.get_variable('loc_W1', shape=(3,3,8,16),
initializer=tf.contrib.layers.xavier_initializer()),
    '#wc3': tf.get_variable('loc_W2', shape=(3,3,16,32),
initializer=tf.contrib.layers.xavier_initializer()),
    'wd1': tf.get_variable('loc_W3', shape=(15*15*16,64),
initializer=tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('loc_W4', initializer=tf.Variable(tf.zeros([64, 6]))),
}

loc_biases = {
    'bc1': tf.get_variable('loc_B0', shape=(8), initializer=tf.contrib.layers.xavier_initializer()),
    'bc2': tf.get_variable('loc_B1', shape=(16), initializer=tf.contrib.layers.xavier_initializer()),
    '#bc3': tf.get_variable('loc_B2', shape=(32), initializer=tf.contrib.layers.xavier_initializer()),
    'bd1': tf.get_variable('loc_B3', shape=(64), initializer=tf.contrib.layers.xavier_initializer()),
    'out': tf.get_variable('loc_B4', initializer=initial),
}

# both placeholders are of type float
x = tf.placeholder("float", [None, 60,60,1])
y = tf.placeholder("float", [None, 8])
pred,h_fc1,h_trans = conv_net(x, weights, biases,loc_weights,loc_biases)
cost = (tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
        +0.01*tf.nn.l2_loss(weights['wc1'])
        +0.01*tf.nn.l2_loss(weights['wc2'])
        +0.01*tf.nn.l2_loss(weights['wc3'])
        +0.01*tf.nn.l2_loss(weights['wd1'])
        +0.01*tf.nn.l2_loss(weights['out']))

```



```

    #+0.01*tf.nn.l2_loss(loc_weights2['wc1'])
    #+0.01*tf.nn.l2_loss(loc_weights2['wc2'])
    #+0.01*tf.nn.l2_loss(loc_weights['wc3'])
    #+0.01*tf.nn.l2_loss(loc_weights2['wd1'])
    #+0.01*tf.nn.l2_loss(loc_weights2['out'])
    +0.01*tf.nn.l2_loss(loc_weights['wc1'])
    +0.01*tf.nn.l2_loss(loc_weights['wc2'])
    #+0.01*tf.nn.l2_loss(loc_weights['wc3'])
    +0.01*tf.nn.l2_loss(loc_weights['wd1'])
    +0.01*tf.nn.l2_loss(loc_weights['out'])
)
learning_rate=0.0005
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

#Here you check whether the index of the maximum value of the predicted image is equal to the
actual labelled image. and both will be a column vector.
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
#calculate accuracy across all the given images and average them out.
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
init = tf.global_variables_initializer()
train=train_iterator.get_next()
val=val_iterator.get_next()
test=test_iterator.get_next()
saver = tf.train.Saver()
for d in ['/gpu:0']:
    with tf.device(d):
        sess=tf.Session()
        sess.run(init)
        train_loss = []
        val_loss=[]
        train_accuracy = []

```

```

val_accuracy=[]
summary_writer = tf.summary.FileWriter('./Output', sess.graph)
sess.run(train_iterator.initializer)
sess.run(val_iterator.initializer)
sess.run(test_iterator.initializer)
for epoch in range(70):
    sum_loss=0
    sum_acc=0
    avg_loss=0
    avg_acc=0
    sum_val_loss=0
    avg_val_loss=0
    sum_val_acc=0
    avg_val_acc=0
    for iteration in range(train_size//batch_size):
        train_x,train_y=sess.run(train)
        train_y=tf.one_hot(train_y, 8).eval(session=sess)
        opt = sess.run(optimizer, feed_dict={x: train_x, y: train_y})
        loss, acc = sess.run([cost, accuracy], feed_dict={x: train_x, y: train_y})
        sum_loss=loss+sum_loss
        avg_loss=sum_loss/(iteration+1)
        sum_acc=acc+sum_acc
        avg_acc=sum_acc/(iteration+1)
        val_x,val_y=sess.run(val)
        val_y=tf.one_hot(val_y, 8).eval(session=sess)
        valid_acc,valid_loss = sess.run([accuracy,cost], feed_dict={x: val_x,y : val_y})
        sum_val_loss=valid_loss+sum_val_loss
        avg_val_loss=sum_val_loss/(iteration+1)
        sum_val_acc=valid_acc+sum_val_acc
        avg_val_acc=sum_val_acc/(iteration+1)
        print("Iter " + str(iteration+1) + ", Loss= " + \

```

```

        "{:.6f}".format(avg_loss) + ", Training Accuracy= " + \
        "{:.5f}".format(avg_acc))
    train_loss.append(avg_loss)
    val_loss.append(avg_val_loss)
    train_accuracy.append(avg_acc)
    val_accuracy.append(avg_val_acc)
    print("Epoch " + str(epoch+1) + ", Loss= " + \
        "{:.6f}".format(avg_loss) + ", Training Accuracy= " + \
        "{:.5f}".format(avg_acc))
    print("Epoch " + str(epoch+1) + ", Loss= " + \
        "{:.6f}".format(avg_val_loss) + \
        ", Validation Accuracy:", "{:.5f}".format(avg_val_acc))
summary_writer.close()
save_path = saver.save(sess, "./tmp/model_stn3.ckpt")

```

Python Code for Chapter 3 – xml2txt_pascalvoc

```

import os
import xml.etree.ElementTree as ET
import pandas as pd
import numpy as np
txtpath='C:/Users/Yuxi/Desktop/Research/dataset/dataset_processed/txt/';
xmlpath='C:/Users/Yuxi/Desktop/Research/dataset/dataset_processed/xml/';
filetype='val'
f = open(txtpath+filetype+'.txt', 'r')
a = np.array(f.read().split("\n"))
b = a[range(len(a)-1)]
df1=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32'))).T,columns=['filename','TF'])
df2=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32'))).T,columns=['filename','TF'])

```

```

df3=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32')))).T,columns=['filename','TF'])
df4=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32')))).T,columns=['filename','TF'])
df5=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32')))).T,columns=['filename','TF'])
df6=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32')))).T,columns=['filename','TF'])
df7=pd.DataFrame(np.vstack((b,-np.ones([1,len(a)-
1],dtype='int32')))).T,columns=['filename','TF'])
for i in range(len(a)-1):
    filename=str(df1.loc[i,'filename'])+'.xml'
    tree= ET.parse(xmlpath+filename)
    root = tree.getroot()
    obj=root.findall('object')
    for j in range(len(obj)):
        idx=obj[j].find('name').text
        if idx==str(1):
            df1.loc[i,'TF']=1
        if idx==str(2):
            df2.loc[i,'TF']=1
        if idx==str(3):
            df3.loc[i,'TF']=1
        if idx==str(4):
            df4.loc[i,'TF']=1
        if idx==str(5):
            df5.loc[i,'TF']=1
        if idx==str(6):
            df6.loc[i,'TF']=1
        if idx==str(7):
            df7.loc[i,'TF']=1

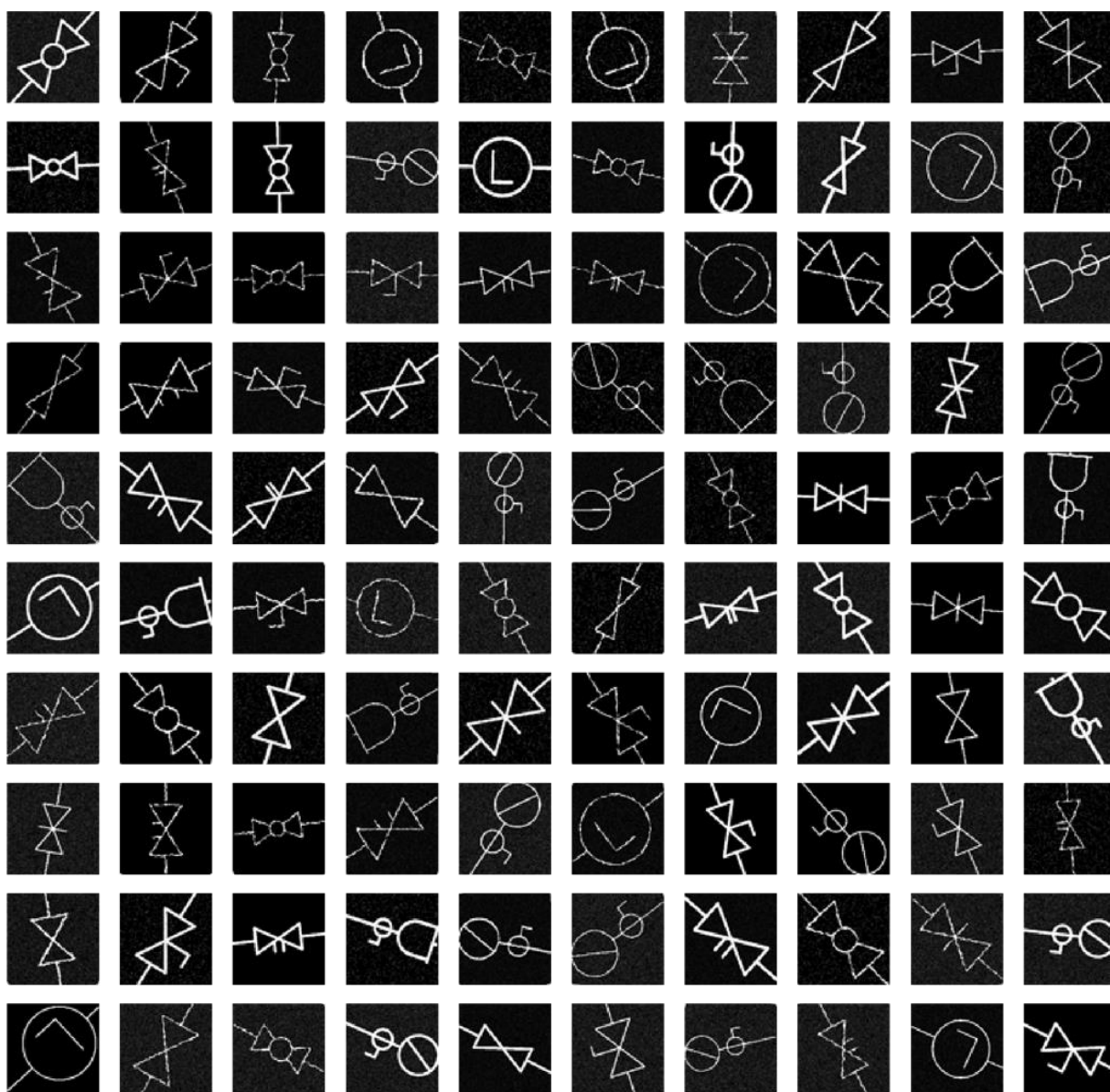
```

```
d = {}  
d=df1,df2,df3,df4,df5,df6,df7  
for n in range(1,8):  
  
np.savetxt(r'C:/Users/Yuxi/Desktop/Research/dataset/dataset_processed/txt/'+str(n)+'_'+filetype+  
' .txt', d[n-1].values,fmt='%s')
```

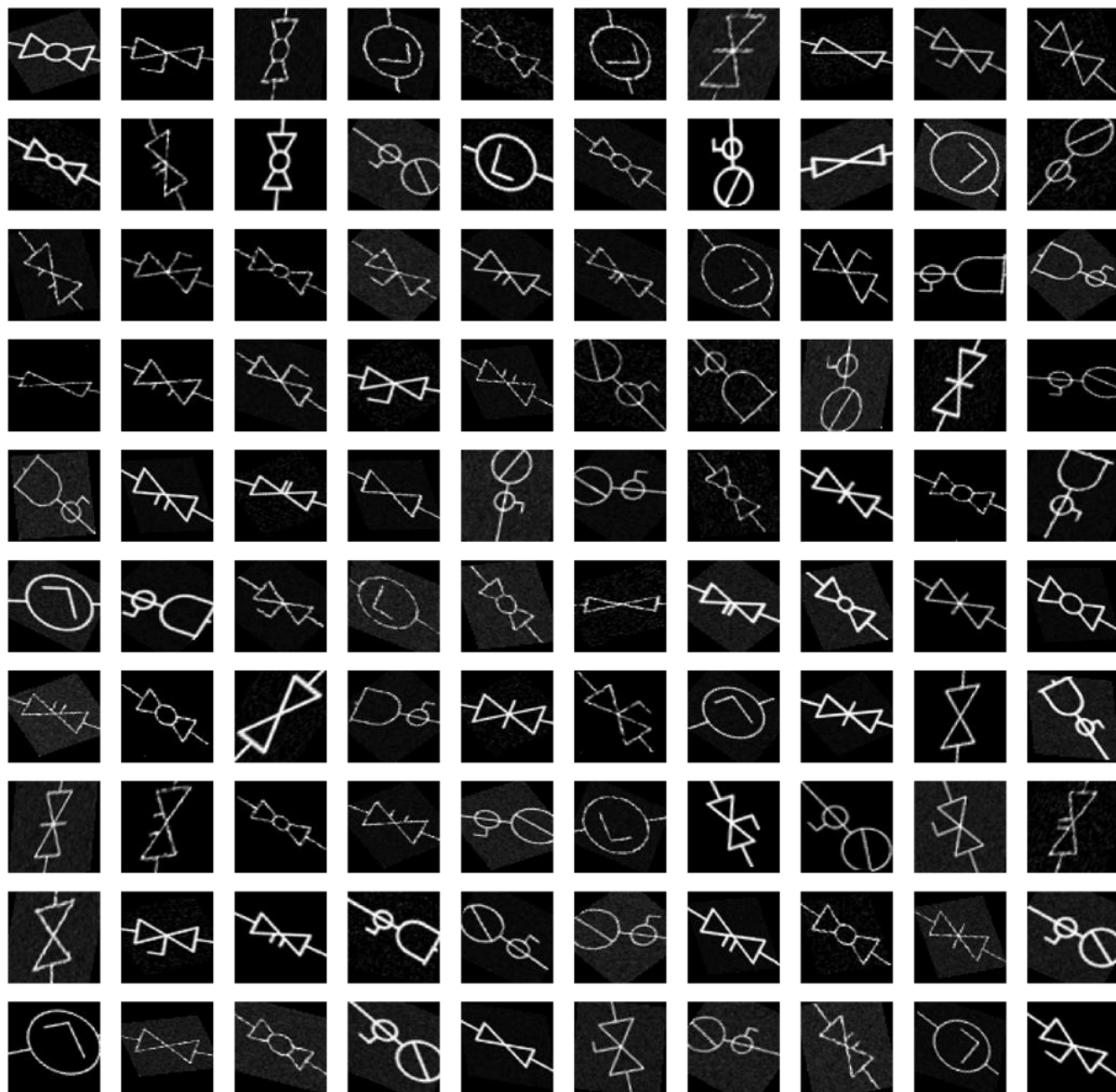
APPENDIX B. IMAGE SAMPLES

The Input and Output of STN for Chapter 2

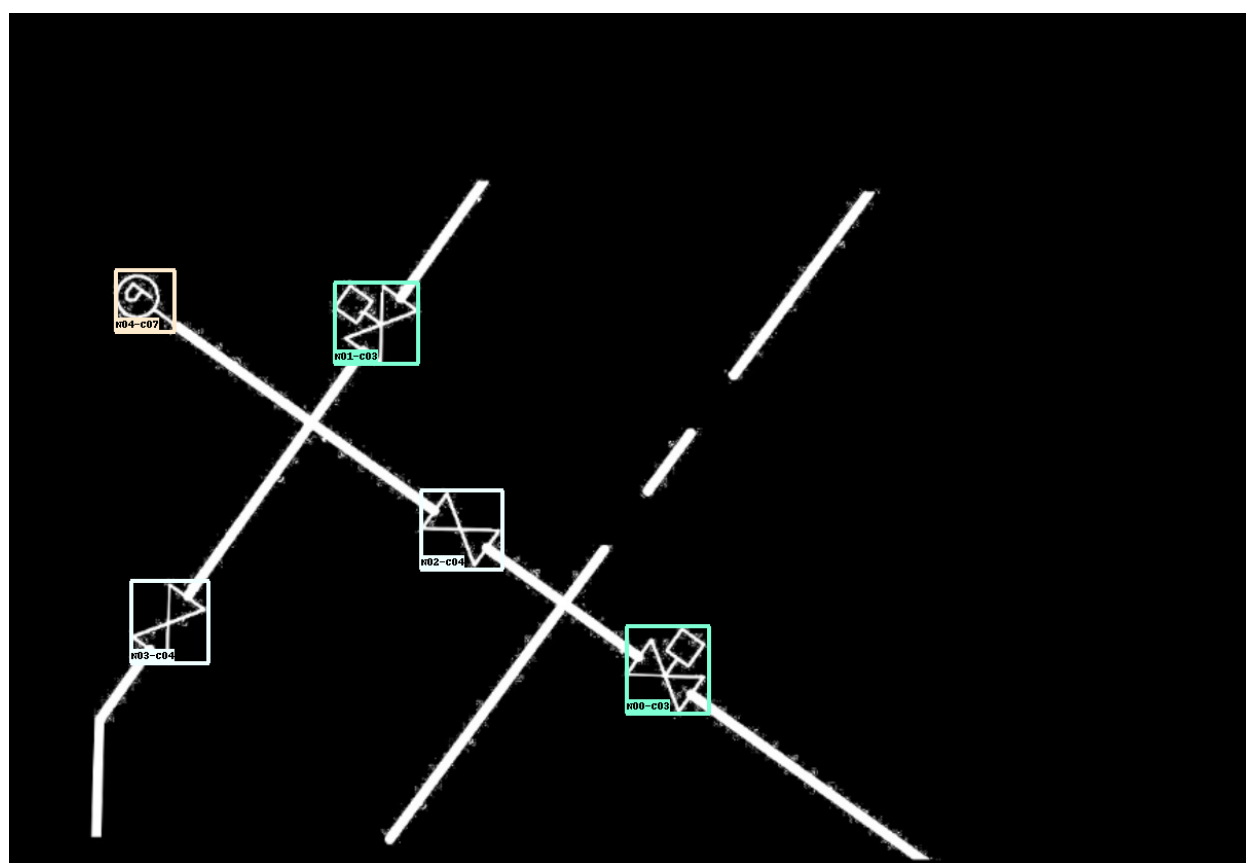
Input

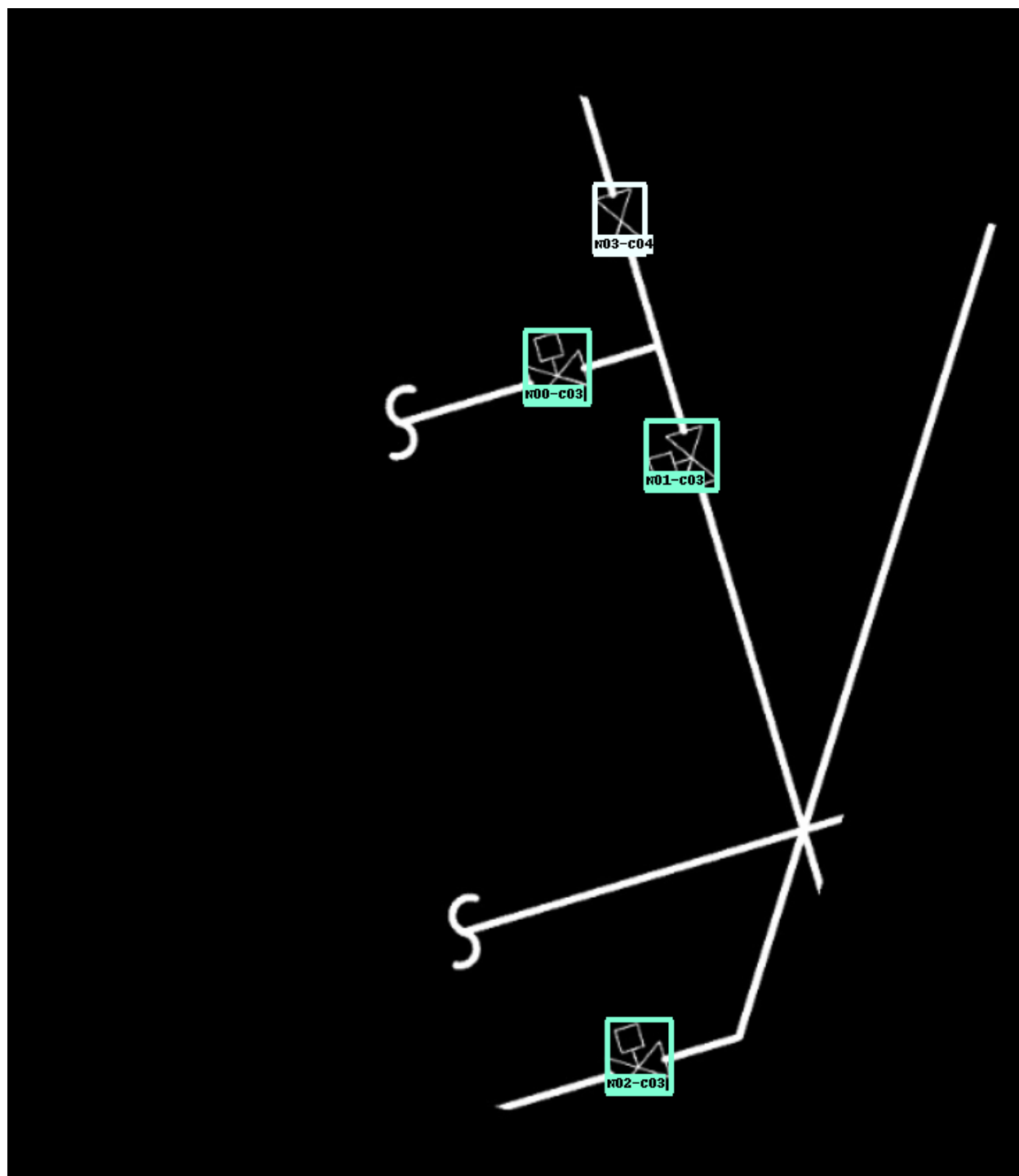


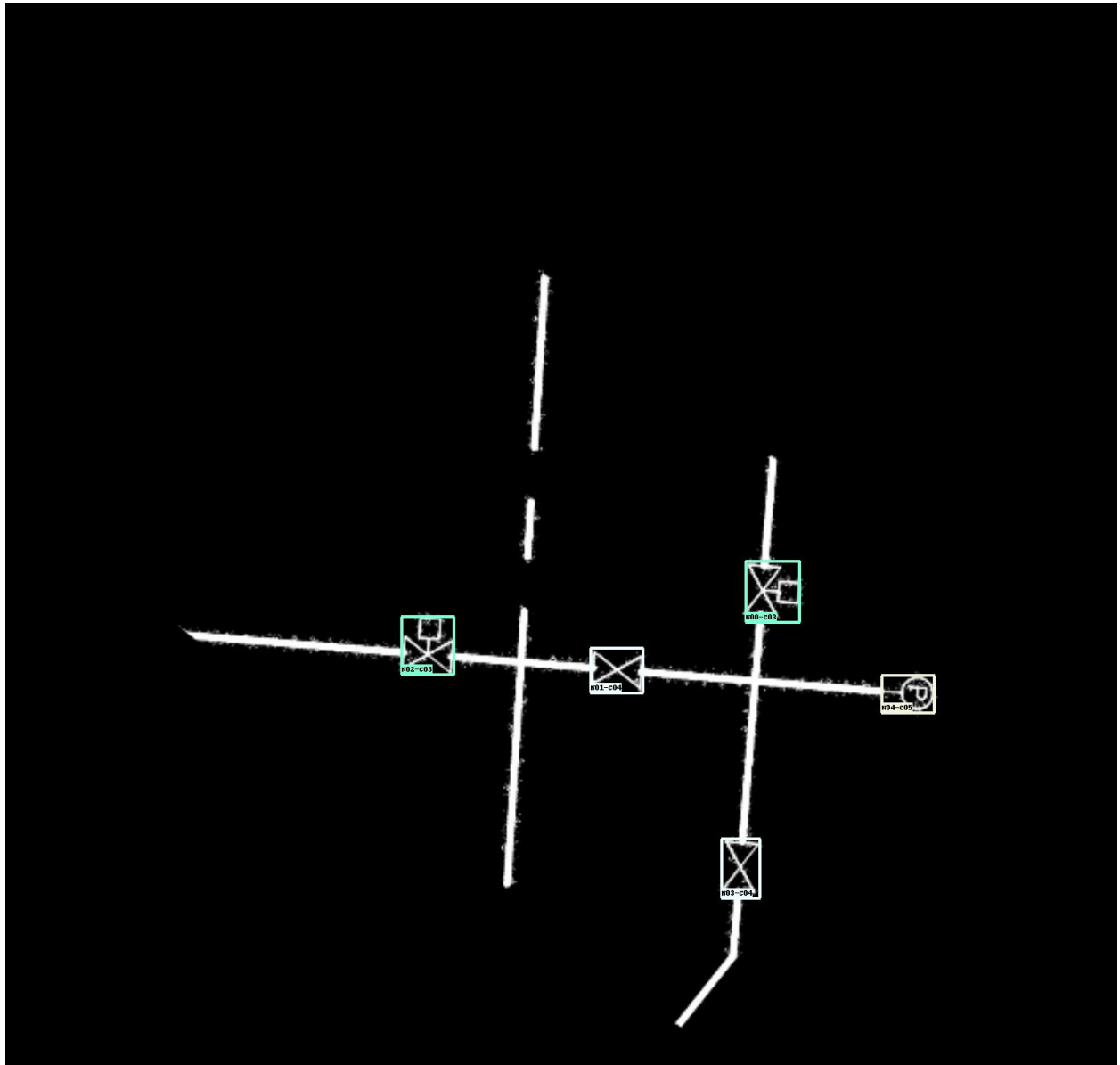
Output

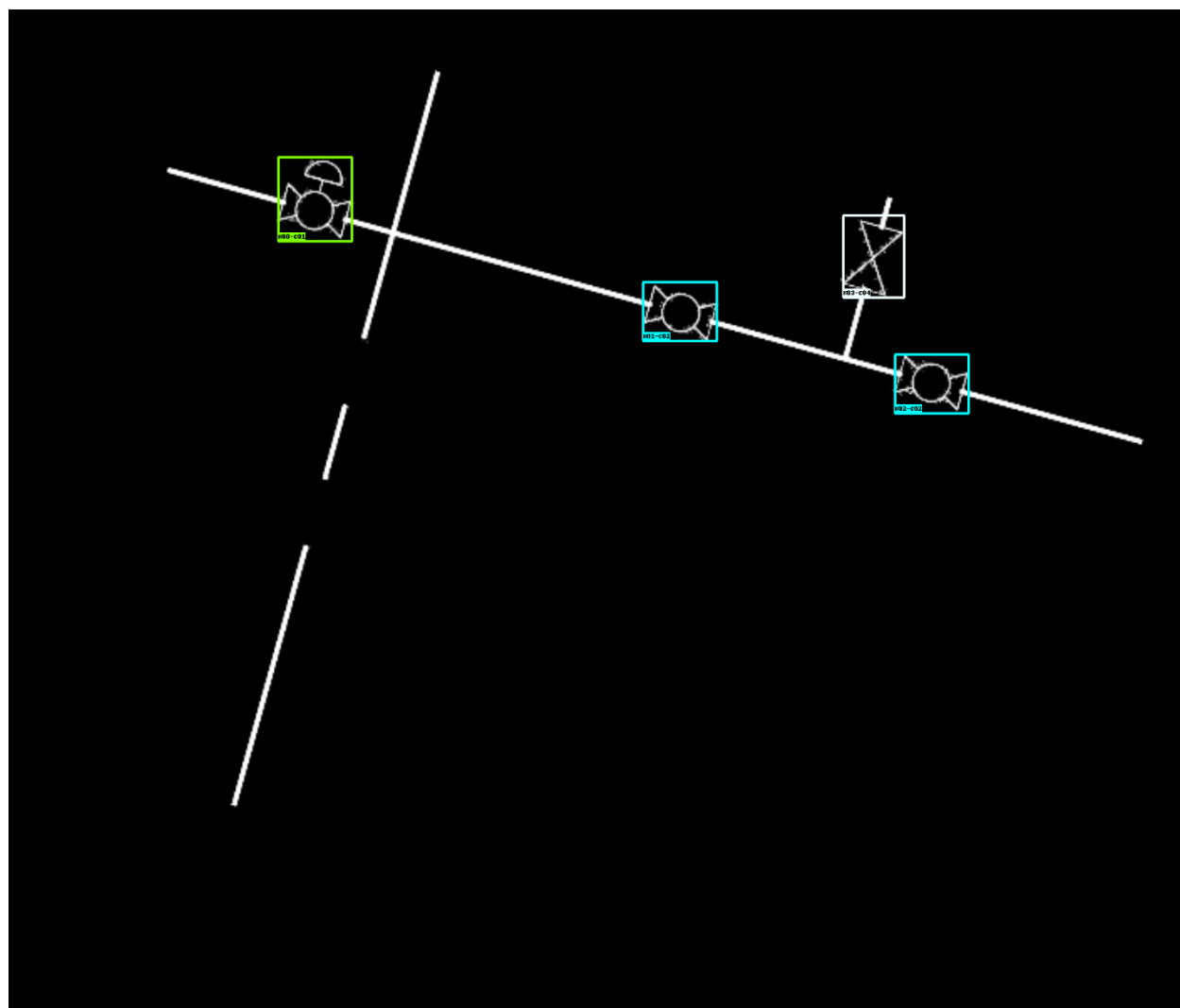


The Ground Truth of Detection for Chapter 3

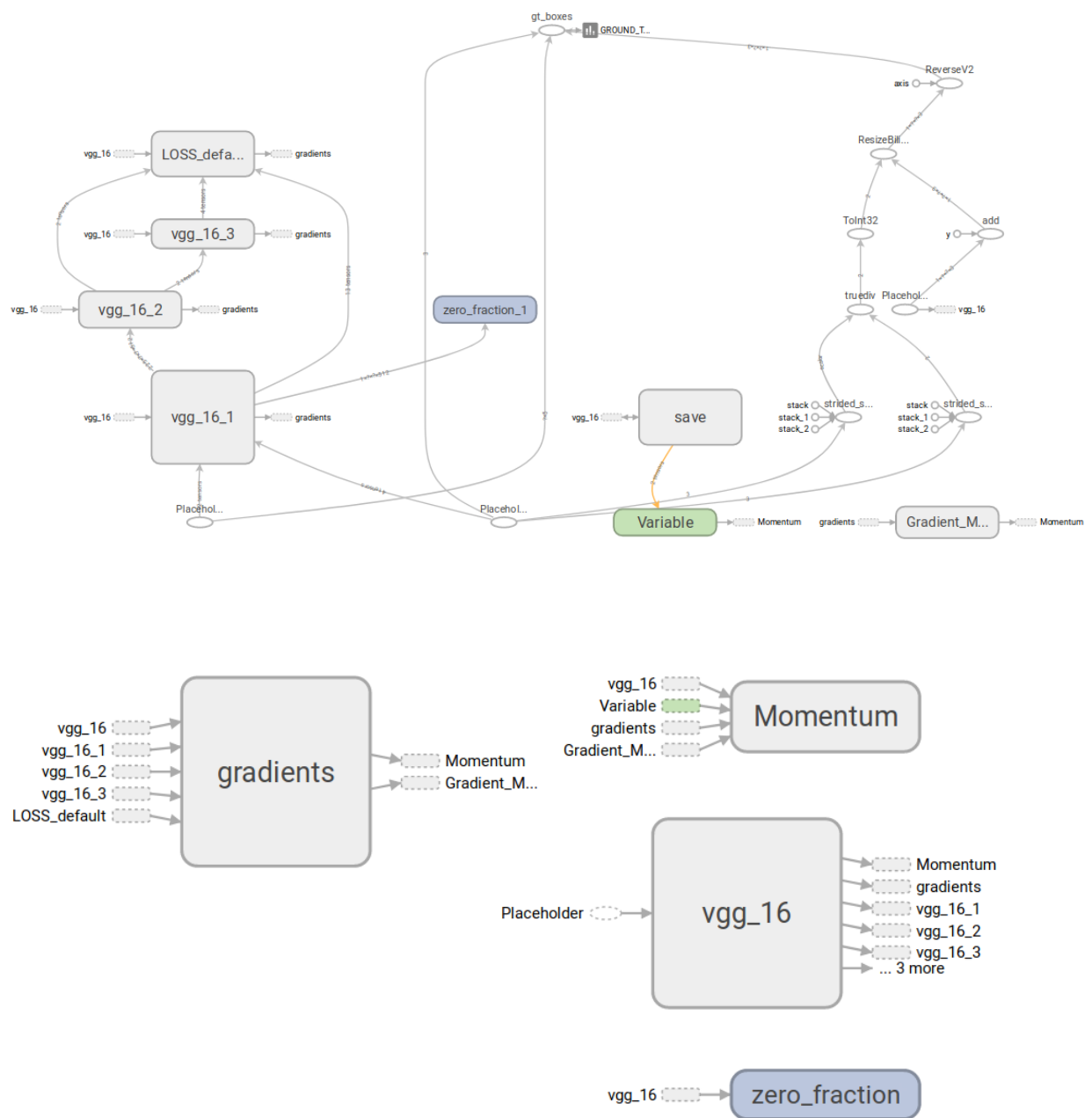




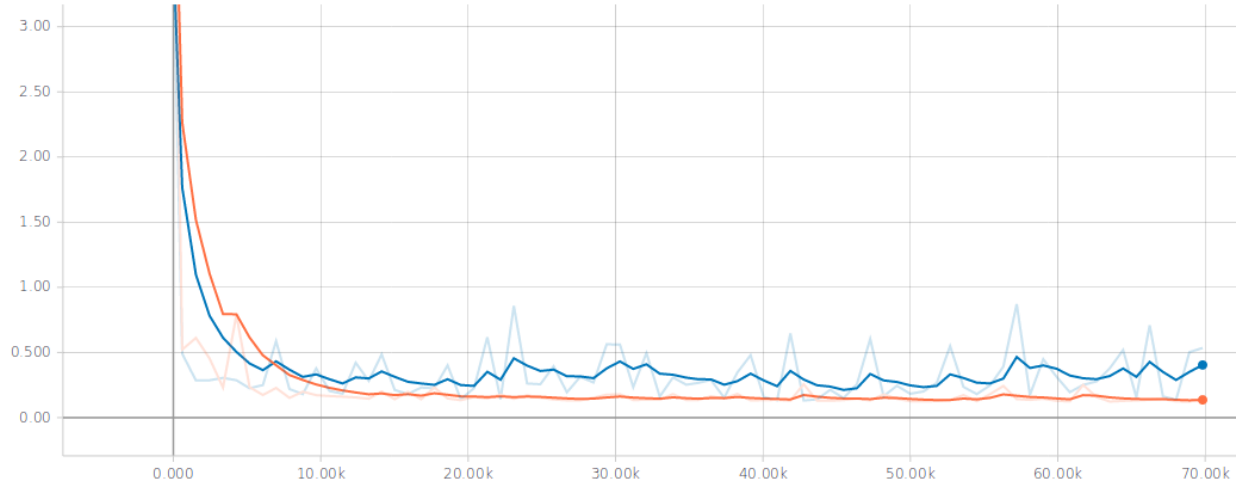




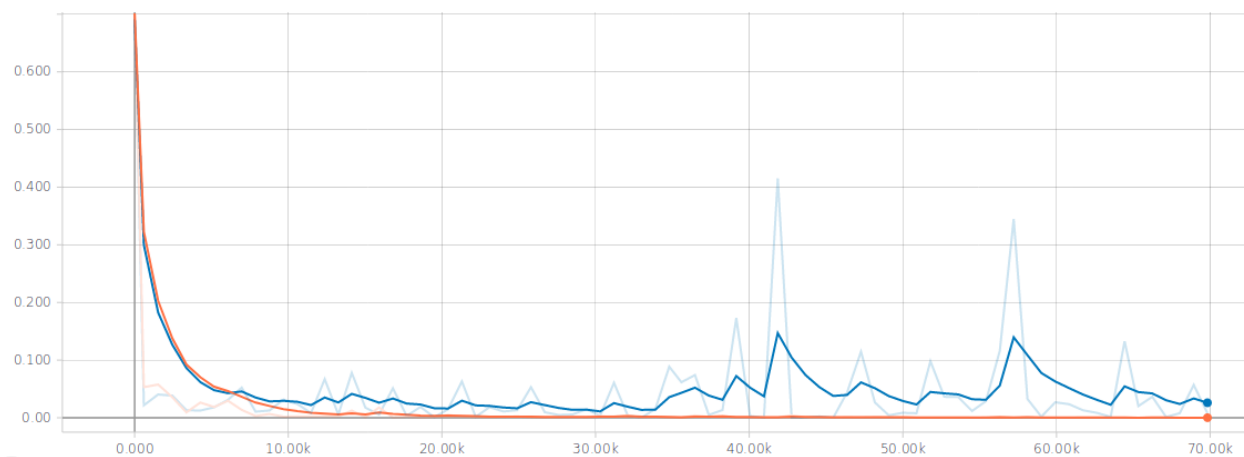
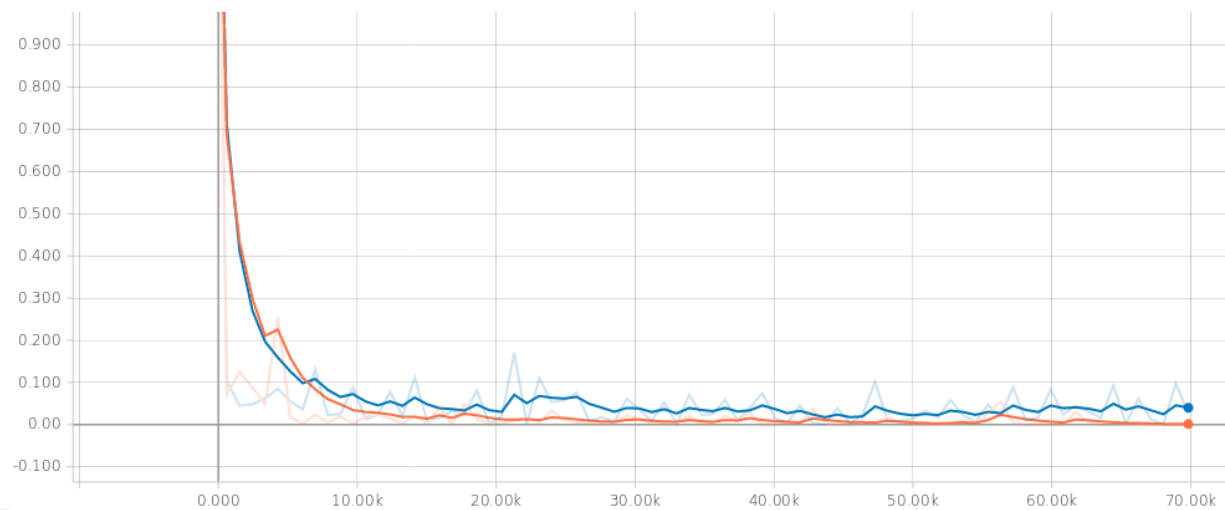
Network Architecture and Loss Curve for Chapter 3

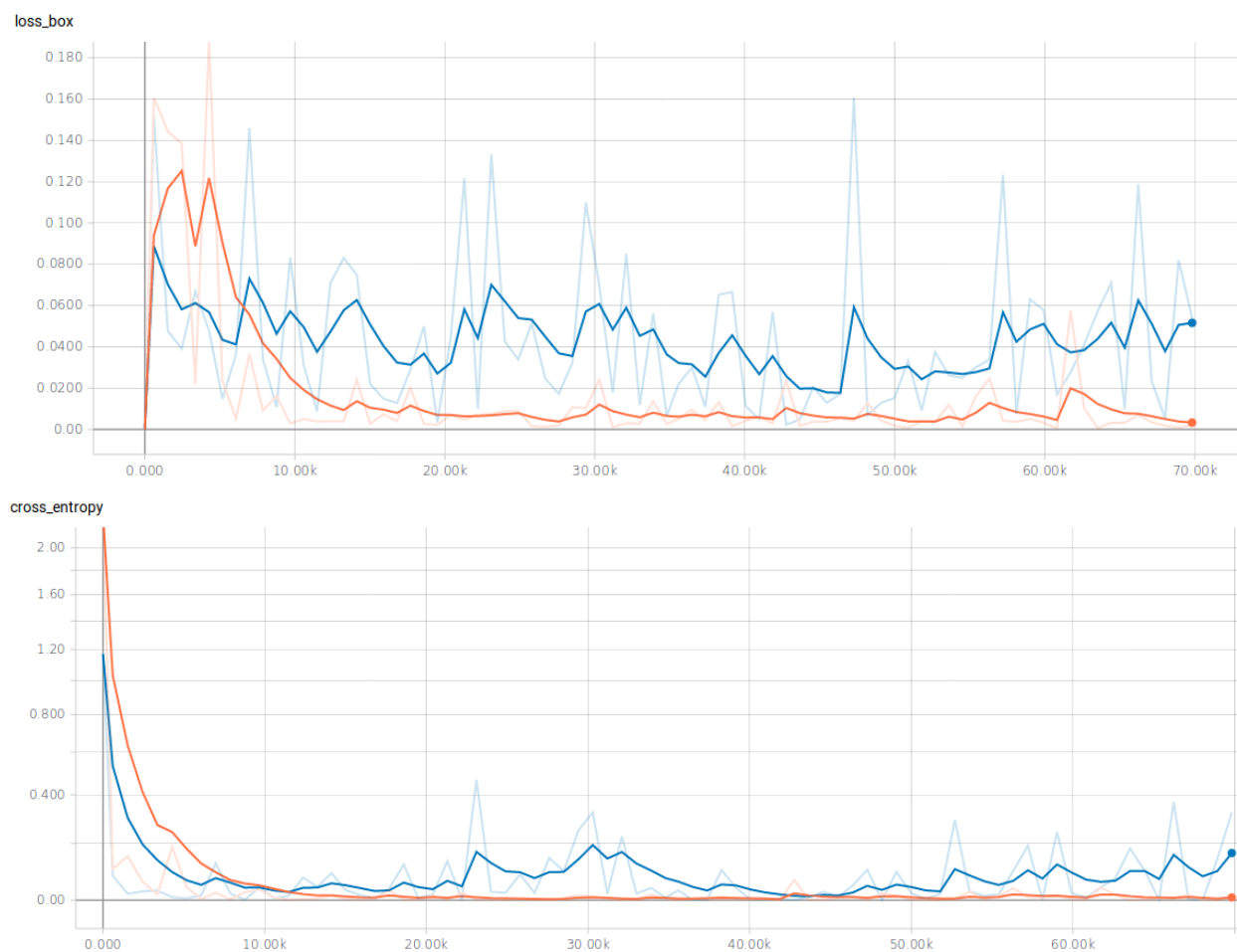


total_loss



rpn_loss_box





REFERENCES

- Akinlar, C., & Topal, C. (2013). EDCircles: A real-time circle detector with a false detection control. *Pattern Recognition*. <https://doi.org/10.1016/j.patcog.2012.09.020>
- Almazán, J., Fornés, A., & Valveny, E. (2012). A non-rigid appearance model for shape description and recognition. *Pattern Recognition*, 45(9), 3105–3113. <https://doi.org/10.1016/j.patcog.2012.01.010>
- Antoniou, A., Storkey, A., & Edwards, H. (2017). Data Augmentation Generative Adversarial Networks, 1–14. Retrieved from <http://arxiv.org/abs/1711.04340>
- Azulay, A., & Weiss, Y. (2018). Why do deep convolutional networks generalize so poorly to small image transformations? Retrieved from <http://arxiv.org/abs/1805.12177>
- Becerik-Gerber, B., Jazizadeh, F., Li, N., & Calis, G. (2011). Application Areas and Data Requirements for BIM-Enabled Facilities Management. *Journal of Construction Engineering and Management*. [https://doi.org/10.1061/\(asce\)co.1943-7862.0000433](https://doi.org/10.1061/(asce)co.1943-7862.0000433)
- Bhatt, M., Hois, J., & Kutz, O. (2012). Ontological modelling of form and function for architectural design. *Applied Ontology*. <https://doi.org/10.3233/AO-2012-0104>
- Boumaiza, A., & Tabbone, S. (2011). A novel approach for graphics recognition based on Galois lattice and bag of words representation. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. <https://doi.org/10.1109/ICDAR.2011.170>
- Boumaiza, A., & Tabbone, S. (2012). Symbol recognition using a Galois lattice of frequent graphical patterns. In *Proceedings - 10th IAPR International Workshop on Document Analysis Systems, DAS 2012*. <https://doi.org/10.1109/DAS.2012.83>
- Cavka, H., Staub-French, S., & Pottinger, R. (2015). Evaluating the Alignment of Organizational and Project Contexts for BIM Adoption: A Case Study of a Large Owner Organization. *Buildings*. <https://doi.org/10.3390/buildings5041265>
- Chen, X., & Gupta, A. (2017). An Implementation of Faster RCNN with Study for Region Sampling, 1–3. Retrieved from <http://arxiv.org/abs/1702.02138>
- Chidester, B., Do, M. N., & Ma, J. (2018). Rotation Equivariance and Invariance in Convolutional Neural Networks. Retrieved from <http://arxiv.org/abs/1805.12301>
- Cohen, T. S., & Welling, M. (2014). Transformation Properties of Learned Visual Representations, (3), 1–11. Retrieved from <http://arxiv.org/abs/1412.7659>

- Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*. <https://doi.org/10.1142/s0218001404003228>
- Coustaty, M., Bertet, K., Visani, M., & Ogier, J. M. (2011). A new adaptive structural signature for symbol recognition by using a Galois lattice as a classifier. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*. <https://doi.org/10.1109/TSMCB.2011.2108646>
- Do, T. H., Tabbone, S., & Ramos Terrades, O. (2016). Sparse representation over learned dictionary for symbol recognition. *Signal Processing*. <https://doi.org/10.1016/j.sigpro.2015.12.020>
- Do, T.-H., Tabbone, S., & Ramos-Terrades, O. (2012). Text/graphic separation using a sparse representation with multi-learned dictionaries. *Pattern Recognition (ICPR), 2012 21st International Conference On, (Icpr)*, 689–692.
- Doermann, D., & Tombre, K. (2014). *Handbook of Document Image Processing and Recognition*. *Handbook of Document Image Processing and Recognition*. <https://doi.org/10.1007/978-0-85729-859-1>
- Dori, D., & Tombre, K. (1995). From engineering drawings to 3D cad models: are we ready now? *Computer-Aided Design*. [https://doi.org/10.1016/0010-4485\(95\)91134-7](https://doi.org/10.1016/0010-4485(95)91134-7)
- Dosch, P., & Lladós, J. (2010). Vectorial Signatures for Symbol Discrimination. https://doi.org/10.1007/978-3-540-25977-0_14
- Dutta, A., Lladós, J., & Pal, U. (2013). A symbol spotting approach in graphical documents by hashing serialized graphs. *Pattern Recognition*. <https://doi.org/10.1016/j.patcog.2012.10.003>
- Escalera, S., Fornés, A., Pujol, O., Lladós, J., & Radeva, P. (2011). Circular blurred shape model for multiclass symbol recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*. <https://doi.org/10.1109/TSMCB.2010.2060481>
- Escalera, S., Fornés, A., Pujol, O., Radeva, P., Sánchez, G., & Lladós, J. (2009). Blurred Shape Model for binary and grey-level symbol recognition. *Pattern Recognition Letters*. <https://doi.org/10.1016/j.patrec.2009.08.001>
- Fletcher, L. A., & Kasturi, R. (1988). A Robust Algorithm for Text String Separation from Mixed Text/Graphics Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/34.9112>
- Gallaher, M. P., O'Connor, A. C., Dettbarn, Jr., J. L., & Gilday, L. T. (2004). Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry. <https://doi.org/10.6028/NIST.GCR.04-867>

- Gerrish, T., Ruikar, K., Cook, M., Johnson, M., Phillip, M., & Lowry, C. (2017). BIM application to building energy performance visualisation and management Challenges and potential. *Energy and Buildings*. <https://doi.org/10.1016/j.enbuild.2017.03.032>
- Girshick, R. (2015). Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*. <https://doi.org/10.1109/ICCV.2015.169>
- Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. <https://doi.org/10.1109/CVPR.2014.81>
- Goodfellow, I. J. (2014). Generative Adversarial Nets. *Corrosion*. <https://doi.org/10.1016/B978-0-408-00109-0.50001-8>
- Grompone Von Gioi, R., Jakubowicz, J., Morel, J. M., & Randall, G. (2010). LSD: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/TPAMI.2008.300>
- Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*. <https://doi.org/10.1016/j.patcog.2017.10.013>
- He, K., Zhang, X., Ren, S., & Sun, J. (2015). Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/TPAMI.2015.2389824>
- Hoang, T. V., & Tabbone, S. (2012). The generalization of the R-transform for invariant pattern representation. *Pattern Recognition*, 45(6), 2145–2163. <https://doi.org/10.1016/j.patcog.2011.11.007>
- Hoang, T. V., & Tabbone, S. (2010). Text extraction from graphical document images using sparse representation. <https://doi.org/10.1145/1815330.1815349>
- Hosang, J., Benenson, R., Dollar, P., & Schiele, B. (2016). What Makes for Effective Detection Proposals? *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/TPAMI.2015.2465908>
- Irizarry, J., Gheisari, M., Williams, G., & Roper, K. (2014). Ambient intelligence environments for accessing building information. *Facilities*. <https://doi.org/10.1108/F-05-2012-0034>
- Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial Transformer Networks, 1–15. Retrieved from <http://arxiv.org/abs/1506.02025>

- Jaoua, A., & Elloumi, S. (2002). Galois connection, formal concepts and Galois lattice in real relations: Application in a real classifier. In *Journal of Systems and Software*. [https://doi.org/10.1016/S0164-1212\(01\)00087-5](https://doi.org/10.1016/S0164-1212(01)00087-5)
- Joseph, S. H., & Pridmore, T. P. (1992). Knowledge-directed interpretation of mechanical engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/34.161351>
- Jylhä, T., & Suvanto, M. E. (2015). Impacts of poor quality of information in the facility management field. *Facilities*. <https://doi.org/10.1108/F-07-2013-0057>
- Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On Large Batch Training for Deep Learning. *ICLR*.
- Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic gradient descent. *ICLR: International Conference on Learning Representations*.
- Lamiroy, B., & Guebbas, Y. (2010). Robust and precise circular arc detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-642-13728-0_5
- Le Bodic, P., Locteau, H., Adam, S., Héroux, P., Lecourtier, Y., & Knippel, A. (2009). Symbol detection using region adjacency graphs and integer linear programming. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. <https://doi.org/10.1109/ICDAR.2009.202>
- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. <https://doi.org/10.1109/5.726791>
- Lenc, K., & Vedaldi, A. (2019). Understanding Image Representations by Measuring Their Equivariance and Equivalence. *International Journal of Computer Vision*. <https://doi.org/10.1007/s11263-018-1098-y>
- Lins, R. D., & Lamiroy, B. (2017). Preface. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9657 LNCS, V–VI. <https://doi.org/10.1007/978-3-319-52159-6>
- Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-46448-0_2
- Lladós, J., Martí, E., & Villanueva, J. J. (2001). Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/34.954603>

- Lu, T., Tai, C.-L., Yang, H., & Cai, S. (2009). A novel knowledge-based system for interpreting complex engineering drawings: theory, representation, and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. <https://doi.org/10.1109/TPAMI.2008.161>
- Luqman, M. M., Ramel, J. Y., Lladós, J., & Brouard, T. (2013). Fuzzy multilevel graph embedding. *Pattern Recognition*. <https://doi.org/10.1016/j.patcog.2012.07.029>
- MacLean, W. J., & Tsotsos, J. K. (2008). Fast pattern recognition using normalized grey-scale correlation in a pyramid image representation. *Machine Vision and Applications*. <https://doi.org/10.1007/s00138-007-0089-8>
- Marcos, D., Volpi, M., & Tuia, D. (2017). Learning rotation invariant convolutional filters for texture classification. In *Proceedings - International Conference on Pattern Recognition*. <https://doi.org/10.1109/ICPR.2016.7899932>
- Marcos, D., Volpi, M., & Tuia, D. (2017). Learning rotation invariant convolutional filters for texture classification. In *Proceedings - International Conference on Pattern Recognition*. <https://doi.org/10.1109/ICPR.2016.7899932>
- Mayo, G., & Issa, R. R. A. (2015). Nongeometric Building Information Needs Assessment for Facilities Management. *Journal of Management in Engineering*. [https://doi.org/10.1061/\(asce\)me.1943-5479.0000414](https://doi.org/10.1061/(asce)me.1943-5479.0000414)
- Messmer, B. T., & Bunke, H. (1998). clustering and error-correcting matching of graphs for learning and recognition of symbols in engineering drawings. https://doi.org/10.1142/9789812797704_0006
- Messmer, B. T., & Bunke, H. (1995). Automatic learning and recognition of graphical symbols in engineering drawings. https://doi.org/10.1007/3-540-61226-2_11
- Munz, E. D. (2017). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey. *Nervenheilkunde*. <https://doi.org/10.1007/s13398-014-0173-7.2>
- Nair, V., & Hinton, G. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning*.
- Nayef, N. (2012). Geometric-based Symbol Spotting and Retrieval in Technical Line Drawings, (December).
- Niimi, T., Hayashi, Y., & Sekiguchi, K. (2003). A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations Logan. *Journal of Biological Chemistry*. <https://doi.org/10.1074/jbc.M212578200>

- Pătrăucean, V., Gurdjos, P., & Von Gioi, R. G. (2012). A parameterless line segment and elliptical arc detector with enhanced ellipse fitting. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). https://doi.org/10.1007/978-3-642-33709-3_41
- Pedro, D., & Gens, R. (2012). Deep Symmetry Networks. Foundations and Trends® in Machine Learning. <https://doi.org/10.1561/22000000044>
- Rahul, R., Paliwal, S., Sharma, M., & Vig, L. (2019). Automatic Information Extraction from Piping and Instrumentation Diagrams. Retrieved from <http://arxiv.org/abs/1901.11383>
- Raveaux, R. (2010). Graph Mining and Graph Classification : application to cadastral map analysis.
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. <https://doi.org/10.1109/CVPR.2016.91>
- Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. In Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017. <https://doi.org/10.1109/CVPR.2017.690>
- Redmon, J., & Farhadi, A. (2017). YOLOv3. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). <https://doi.org/10.1109/CVPR.2017.690>
- Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. <https://doi.org/10.1109/TPAMI.2016.2577031>
- Rosin, P. L., & West, G. A. (1989). Segmentation of edges into lines and arcs. Image and Vision Computing. [https://doi.org/10.1016/0262-8856\(89\)90004-8](https://doi.org/10.1016/0262-8856(89)90004-8)
- Rusiñol, M., Borràs, A., & Lladós, J. (2010). Relational indexing of vectorial primitives for symbol spotting in line-drawing images. Pattern Recognition Letters. <https://doi.org/10.1016/j.patrec.2009.10.002>
- Rusiñol, M., & Lladós, J. (2006). Symbol spotting in technical drawings using vectorial signatures. https://doi.org/10.1007/11767978_4
- Rusk, N. (2015). Deep learning. Nature Methods, 13(1), 35. <https://doi.org/10.1038/nmeth.3707>
- Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic Routing Between Capsules, (Nips). Retrieved from <http://arxiv.org/abs/1710.09829>
- Santosh, K. C., Lamiroy, B., & Ropers, J. P. (2009). Inductive logic programming for symbol recognition. In Proceedings of the International Conference on Document Analysis and Recognition, ICDAR. <https://doi.org/10.1109/ICDAR.2009.166>

- Santosh, K., Lamiroy, B., & Wendling, L. (2013). DTW–Radon-Based shape descriptor for pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*. <https://doi.org/10.1142/s0218001413500080>
- Santosh, K. C., Lamiroy, B., & Wendling, L. (2014). Integrating vocabulary clustering with spatial relations for symbol recognition. *International Journal on Document Analysis and Recognition*. <https://doi.org/10.1007/s10032-013-0205-4>
- Sattenini, A., Azhar, S., & Thuston, J. (2017). Preparing A Buindling Information Model For Facility Maintenance And Management. In 28th International Symposium on Automation and Robotics in Construction (ISARC 2011). <https://doi.org/10.22260/isarc2011/0024>
- Schmidt, U., & Roth, S. (2012). Learning rotation-aware features: From invariant priors to equivariant descriptors. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. <https://doi.org/10.1109/CVPR.2012.6247909>
- Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*. <https://doi.org/10.1109/ICDAR.2003.1227801>
- Tabbone, S., Ramos Terrades, O., & Barrat, S. (2009). Histogram of radon transform. A useful descriptor for shape retrieval. <https://doi.org/10.1109/icpr.2008.4761555>
- Tabbone, S., Wendling, L., & Salmon, J. P. (2006). A new shape descriptor defined on the Radon transform. *Computer Vision and Image Understanding*. <https://doi.org/10.1016/j.cviu.2005.06.005>
- Tian, Z., Huang, W., He, T., He, P., & Qiao, Y. (2016). Detecting text in natural image with connectionist text proposal network. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-319-46484-8_4
- Tombre, K., Tabbone, S., Péliissier, L., Lamiroy, B., & Dosch, P. (2007). Text/Graphics Separation Revisited. https://doi.org/10.1007/3-540-45869-7_24
- Visani, M., Bertet, K., & Ogier, J.-M. (2011). NAVIGALA: an original symbol classifier based on based on navigation through a galois lattice. *International Journal of Pattern Recognition and Artificial Intelligence*. <https://doi.org/10.1142/s0218001411008634>
- Volk, R., Stengel, J., & Schultmann, F. (2014). Building Information Modeling (BIM) for existing buildings - Literature review and future needs. *Automation in Construction*. <https://doi.org/10.1016/j.autcon.2013.10.023>

- Webster, J. G., Santosh, K. C., & Wendling, L. (2015). Graphical symbol recognition. In Wiley Encyclopedia of Electrical and Electronics Engineering. <https://doi.org/10.1002/047134608x.w8260>
- Yalcinkaya, M., & Singh, V. (2015). Patterns and trends in Building Information Modeling (BIM) research: A Latent Semantic Analysis. Automation in Construction. <https://doi.org/10.1016/j.autcon.2015.07.012>
- Zhang, D., & Lu, G. (2002). Generic Fourier descriptor for shape-based image retrieval. In Proceedings - 2002 IEEE International Conference on Multimedia and Expo, ICME 2002. <https://doi.org/10.1109/ICME.2002.1035809>