ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Computer Vision on Piping and Instrumentation Diagrams: Towards the identification of their components

Μηχανική όραση σε Σχέδια Σωληνώσεων και Οργάνων: Προς την αναγνώριση των επί μέρους στοιχείων τους

Author: Anna Androvitsanea

Academic Supervisor:
K. Koutroumbas

Industrial Supervisor:
A. Karvounis, N. Koudounas, C. Merinopoulos

A Thesis submitted in fulfillment to the requirements for the degree of
Master of Science
in
Data Science
2020-2021
December 8, 2021

Aknowledgments

I would like to express my deepest gratitude to **Prof. Konstantinos Koutroumbas** for his continuous support and his detailed work on reviewing my thesis. It has been a great pleasure working with him, and a great lesson too.

I would also like to thank my industrial supervisor, Mr. Alexandros Karvounis,and the team of **Synthetica**, Nick & Chris, for the input, tipps, meeting and for their great support in general.

I also thank my classmate, Domna Ladopoulou, for helping me bridge the distance between Berlin and Athens, during the delivery of my thesis.

Finally, a special thank goes to **Prof. Vasilis Vassalos** for giving the chance to attend the Master Program and for bearing with the challenges I have faced at the very end.

Abstract

Piping and Instrumentation Diagrams (P&IDs) are schematic representations of equipment, pipelines, instrumentation, and control systems. They appear in process environments, such as Oil Refineries, Chemical Plants, Paper Mills, and Cement Plants, etc. The identification of each element constituting a P&ID, along with the way they are interconnected, is an important task that has not been automated yet. In this work we study a methodology and develop the respective algorithm towards the identification of these components. This identification aims to the classification of the elements based on their representation as images as well as to the identification and translation of the codes included in the diagrams.

In order to achieve this goal a combination of methods are employed. Using the `OpenCV` library the outlines of the P&ID are calculated. An algorithm is developed, which based on the coordinates of the outlines, delivers snapshots of the elements constituting the P&ID. In the sequel, these elements are classified by a suitably designed classifier, to one out of 53 classes. The classifier is a convolutional neural network (CNN), implemented using the `TensorFlow` and `Keras` libraries, which was trained on a data set of 2970 images that belong to one out of 53 classes.

Textual information contained in the P&ID are identified, using the `pytesseract` library and stored into an array. Then, they are passed to an algorithm that implements the ANSI/ISA-5.1.-1984 (R1992) standards and deciphers the textual tags, by providing as output the name, function, modifier etc. of each element.

The model is able to successfully identify an image and attribute it to the right class, which is a great step towards solving the challenging problem of the identification of the elements constituting a P&ID.

Περίληψη

Τα σχέδια σωληνώσεων και οργάνων (P&ID) είναι σχηματικές απεικονίσεις εξοπλισμού, σωληνώσεων, οργάνων και συστημάτων ελέγχου. Εμφανίζονται σε εγκαταστάσεις διεργασιών όπως Διυλιστήρια, Χημικά Εργοστάσια, Χαρτοποιεία και Τσιμεντοβιομηχανίες κ.λπ. Η αναγνώριση κάθε στοιχείου που συνιστά ένα τέτοιο σχέδιο, καθώς και η συνδεσμολογία αυτών μεταξύ τους, είναι κάτι πολύ σημαντικό που δεν έχει μέχρι σήμερα αυτοματοποιηθεί. Σε αυτή την εργασία προτείνουμε μια μεθοδολογία και αναπτύσσουμε τον αντίστοιχο κώδικα για την αναγνώριση αυτών των στοιχείων. Αυτή η ταυτοποίηση στοχεύει στην ταξινόμηση των στοιχείων με βάση την αναπαράστασή τους ως εικόνες καθώς και στην αναγνώριση και μετάφραση των κωδικών που περιλαμβάνονται στα σχέδια.

Για την επίτευξη αυτού του στόχου χρησιμοποιείτε ένας συνδυασμός μεθόδων. Χρησιμοποιώντας τη βιβλιοθήκη `OpenCV` υπολογίζονται το περίγραμμα του κάθε στοιχείου επί του σχεδίου. Αναπτύσσουμε ένας αλγόριθμος, ο οποίος με βάση τις συντεταγμένες που παρέχονται από τα περιγράμματα, δημιουργεί στιγμιότυπα των στοιχείων αυτών. Στην συνέχεια, αυτά τα στοιχεία κατηγοριοποιούνται από έναν κατάλληλα σχεδιασμένο ταξινομητή, σε μια από 53 κλάσεις. Ο ταξινομητής είναι ένα Συνελικτικό Νευρωνικό Δίκτυο, που υλοποιείτα με τη χρήση των βιβλιοθηκών `TensorFlow` και `Keras`, το οποίο εκπαιδεύουμε σε ένα σύνολο δεδομένων 2970 εικόνων που ανήκουν σε 53 τάξεις.

Οι πληροφορίες κειμένου που περιέχονται στο σχέδιο σωληνώσεων και οργάνων αναγνωρίζονται με τη χρήση της βιβλιοθήκης `pytesseract` και αποθηκεύονται σε έναν πίνακα. Στη συνέχεια, εφαρμόζουμε σε αυτά έναν αλγόριθμο που υλοποιεί τα πρότυπα ANSI/ISA-5.1.-1984 (R1992) και αποκρυπτογραφεί τις ετικέτες κειμένου, παρέχοντας ως αποτέλεσμα το όνομα, τη συνάρτηση, τον τροποποιητή κ.λπ. κάθε στοιχείου.

Το πρόγραμμα που αναπτύσσουμε επιτυγχάνει να ταυτοποιήσει ένα στοιχείο ενός P&ID και να το αποδόσει στην ορθή κλάση, γεγονός που αποτελεί ένα πρώτο ελπιδοφόρο βήμα προς την επίλυση του πολύ απαιτητικού προβλήματος της αναγνώσης στοιχείων ενός P&ID.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This work is developed in the framework of the Data Science of the Athens University of Economics and Business. It is a computer vision application where the aim is the identification of the objects and textual information constituting a piping and instrumentation diagram (P&ID). The project is developed in collaboration with the company Synthetica.

Synthetica has developed a product related to Alarm System and Event Signals. In that respect, the team collects and analyzes near-real time data in order to produce insights of various sorts. One of the ways to expand the situational awareness is to encode and digitize in detail the pipes and instruments that are reflected in the piping and instrumentation diagram. This information will flow into the Alarm System signals to empower descriptive, diagnostic, predictive and prescriptive capabilities. Apart from Shipping industry, the desired functionalities of the tool is expected to be proved very helpful in other industrial sectors.

A piping and instrumentation diagram (P&ID) is a detailed diagram used in the process industry. It depicts the **piping** and **process equipment** along with the **instrumentation** and **control devices**. It also shows the connections between the process equipment and the instrumentation used to control the process. The process industry uses a standardized set of symbols used to prepare a P&ID are standardized. They are generally based on International Society of Automation (ISA) Standard S5.1 [1].

In order to attack this problem, we apply object and text detection on engineering plans (Chap. 2 and 5). This enables the identification of the objects and labels, and therefore the creation a digital twin of the hard-copy diagram. The task is broken down in smaller parts and is developed as such. First, concerning the object identification task, we identify the outlines at the P&ID and therefore

---

[1] Piping and instrumentation diagram. In Wikipedia. Retrieved November 22, 2021.

the coordinates of the elements in the PID under study to be labeled. In parallel to that, a neural network classifier is defined and trained based on a suitably created synthetic dataset. Second, an optical character recognition scripts is developed that identify the text on the P&ID and decode its meaning. The synthesis of the above give rise to the proposed methodology towards the solution of the problem of the identification of the P&ID components (Chap. 6).

The related code is written in Python 3 and employs, among other, the `TensorFlow`, `Keras`, `sklearn`, `OpenCV` and `pytesseract`.

Chapter 2

Problem statement

The task considered in the present study is the digitization and interpretation of a piping and instrumentation diagram (P&ID). There are various types of symbols present in the P&ID sheets which represent certain instruments responsible for controlling the flow through pipelines and performing various tasks. These consist mainly of the **mechanical equipment**, the **piping**, including size and identification and the **process control instrumentation** and designation (names, numbers, unique tag identifiers) (Fig. 4.5).

The mechanical equipment (Fig. 2.2) includes pressure vessels, columns, tanks, pumps, compressors, heat exchangers, furnaces, wellheads, fans, cooling towers, turbo-expanders, pig traps, bursting discs, restriction orifices, strainers and filters, steam traps, moisture traps, sight-glasses, silencers, flares and vents, flame arrestors, vortex breakers, eductors etc.

The piping refers to type of pipe and line number, flow direction (Fig. 2.3), indicates the connection enabled by the pipe, insulation and heat tracing etc. The process control instrumentation and designation includes valves and their types and identifications, such as isolation, shutoff and safety valves. It also includes the control inputs and outputs, like e.g. sensors, as well as miscellaneous elements, such as vents, drains, flanges, special fittings, sampling lines, reducers and swages.

Instruments are frequently accompanied by text (Fig. 2.4) which bears coded information. This information is related to the variable linked to the instrument, e.g. temperature, pressure, the function of the instrument, e.g. alarm, well, the output functionality, meaning the results it produces, e.g. switch, transmit as well as modification of these functionalities, e.g. high, low, multi-function etc. The coding is performed according to the **ANSI/ISA-5.1.-1984 (R1992)** standards. At Figure 2.4 we can see for example the code FV-01. This gives us the information that this

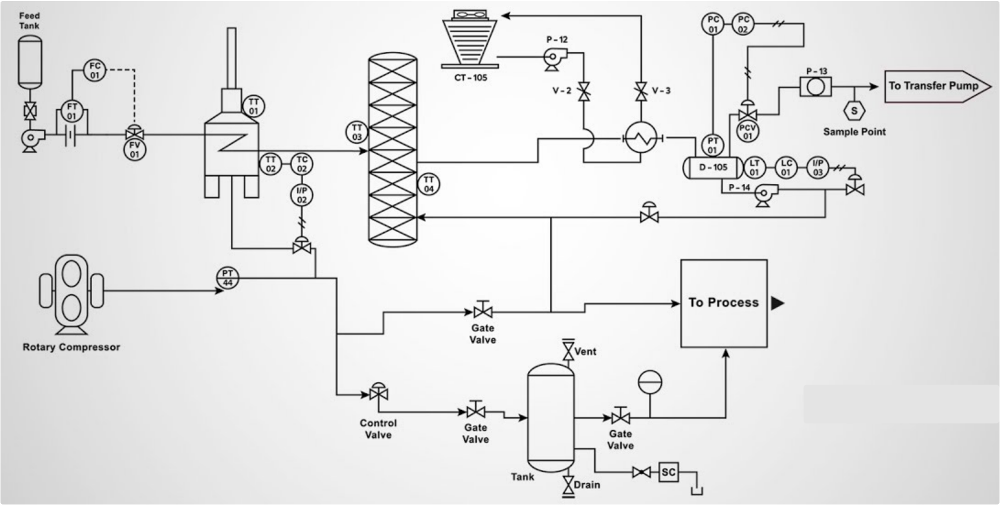


Figure 2.1: A Piping and Instrumentation Diagrams (Source: PID Symbols).

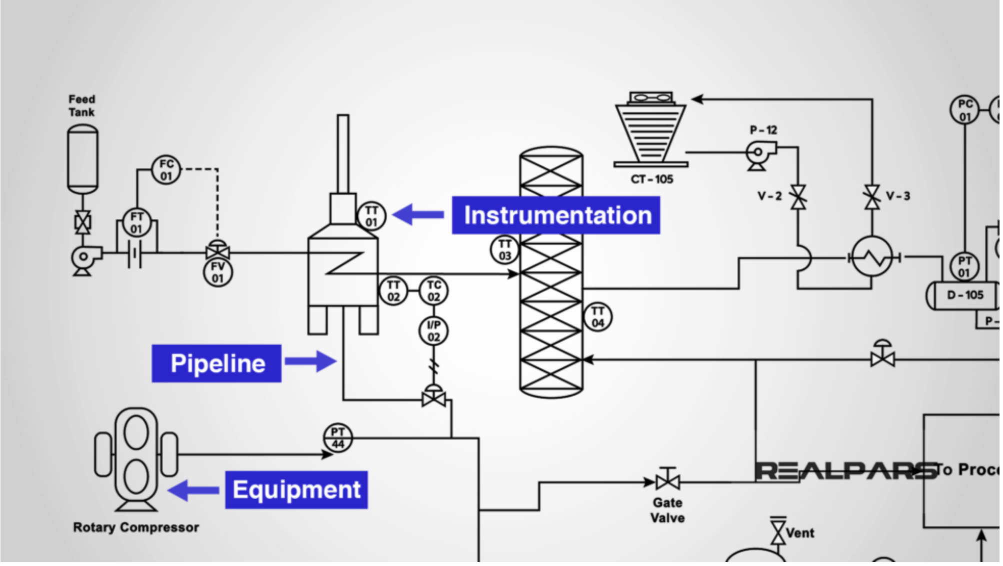


Figure 2.2: A Piping and Instrumentation Diagrams. Indication of the instrumentation, pipeline and equipment (Source: PID Symbols).

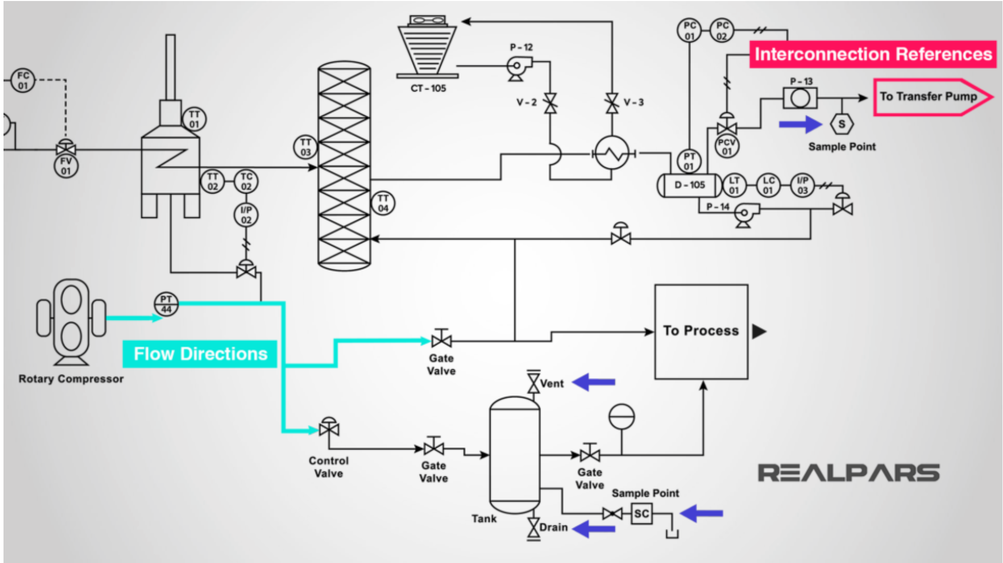


Figure 2.3: A Piping and Instrumentation Diagrams. Indication of the flow direction (Source: PID flow directions).

instrument is a valve that regulates the flow.

Our goal is to develop a methodology that takes as input a piping and instrumentation diagram (P&ID) in an image format, and delivers as output a list of objects constituting the diagram along with the textual information included. To achieve tis goal, the following steps are taken:

1. Import a scanned diagram and transform it into an array
2. Identify the flow diagram, meaning the lines, that connect all elements to one another within the P&ID, as well as the contours forming the diagram
3. Spot each instrument and identify its type via a suitably designed and trained convolutional neural network classifier.
4. Spot each label within the diagram and decipher it

This leads to the construction of a pipeline for information extraction from P&ID diagram via a combination of traditional vision techniques and a deep learning model, in order to (a) isolate and identify pipeline codes, inlets and outlets, and (b) detect symbols.

Concerning the identification of the mechanical equipment, in our case, we have 53 classes of symbols to detect and localise in the sheets, e.g. ball valve, check valve, globe valve, gate valve, filters, compressors etc (Fig. 2.5). These symbols have very low inter-class difference in

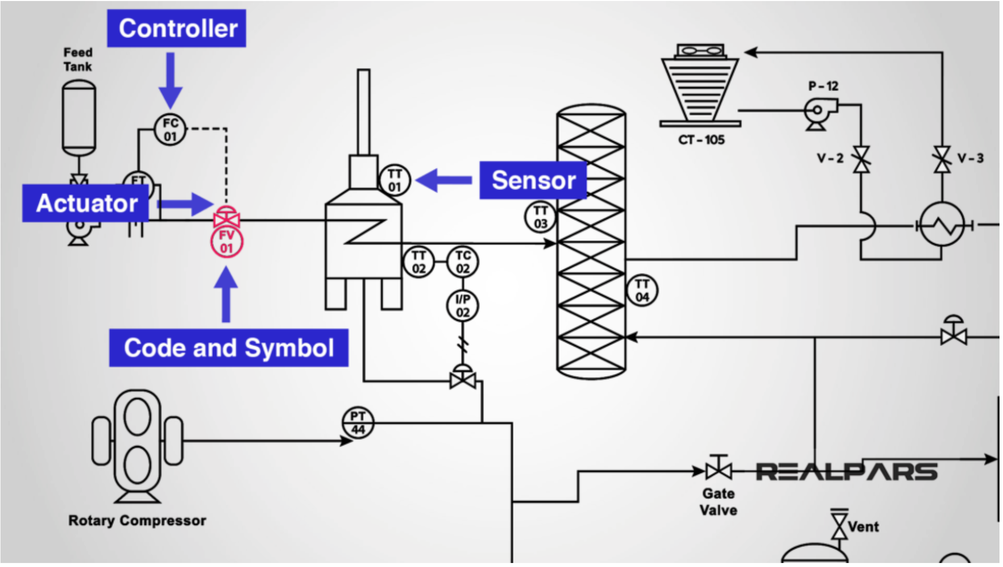


Figure 2.4: A Piping and Instrumentation Diagrams. Indication of the sensors, controllers, actuators and code/symbols (Source: PID Symbols and Codes).

visual appearances. So, standard deep networks for classification are not able to distinguish them correctly. Therefore, we implement a convolutional neural network (CNN)[1] classifier, which is able to discriminate among the various such symbols. The choice of a CNN classifier is justified from the fact that CNNs are vastly used in computer vision problems and they have been designed for processing structured arrays of data.

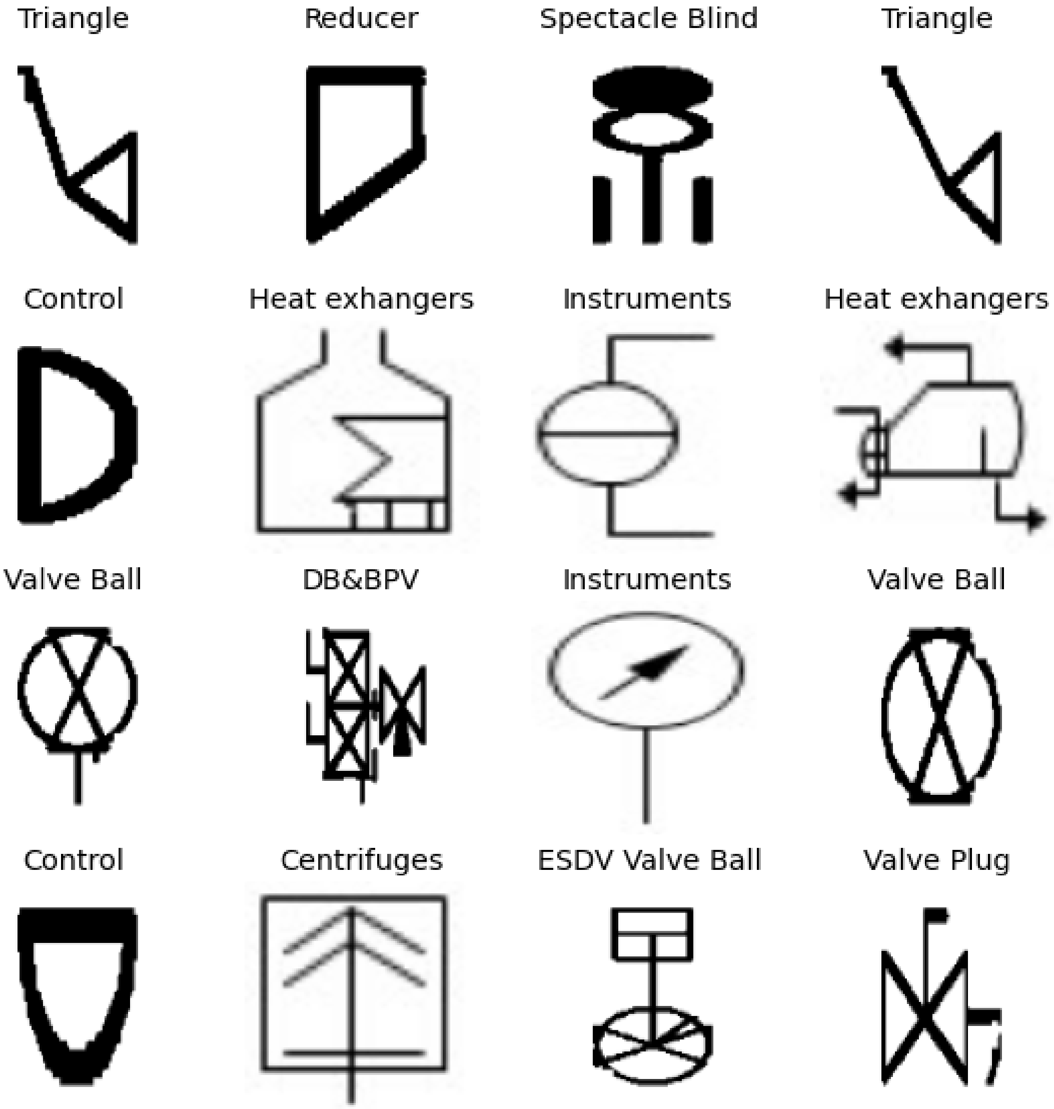


Figure 2.5: Sample of 16 images taken from the dataset used for the training, validation and testing of the model.

Chapter 3

Review of related literature

The digitization of Piping and Instrumentation diagrams (P&IDs) is a challenging problem studied by several researchers in the past. Rahul et al. [2] implemented a fully convolutional neural network (FCNN) building a pipeline for information extraction from P&ID sheets via a combination of traditional vision techniques and state-of-the-art deep learning models. The aim was the identification and isolation of pipeline codes, pipelines, inlets and outlets, and for detecting symbols. This information populates a tree-like data structure for capturing the structure of the piping schematics.

Rahul et al. [2] set as main goals the following:

- Determination of the flow from inlet to outlet: To this end, a combination of detection of different components of the process flow followed by their association with appropriate pipeline and representation in a tree-like data structure is performed.
- Detection and recognition of graphic objects (e.g., pipelines, inlets and outlets) presented in PID: To this end, conventional image processing and vision techniques were utilized.
- Detection of symbols in PID sheets: To this end, a fully convolutional neural network (FCN) based segmentation for detection of symbols in PID sheets at the pixel level were utilized. This choice was taken because of the very minute visual difference in appearance of different symbols, as the presence of noisy and textual information inside symbols makes it difficult to classify based on bounding box detection networks like Faster-RCNN

The symbol detection was implemented using a VGG-19 based FCN for training symbol detector. An input image of size 400×400 is fed to the network and it is trained using Adam optimizer with a learning rate of 0.0004 and batch size equal to 8.

Elyan et al. [3] implemented Generative Adversarial Networks (GAN) to address the challenge of class-imbalance in the dataset used during training. Generative Adversarial Networks consist of two contesting models the Generator (G), and the Discriminator (D). The discriminator is a classifier that receives input from the training set, which is an authentic content, and from the generator, which serves as fake input.

The authors the training process, the discriminator learns how to distinguish between authentic and fake input samples. The generator is trained to create samples that capture the underlying characteristics of the original data.

They continue performing symbol recognition employing the YOLO Algorithm for Object Detection. This enables the representation of the problem as a set of bounding box coordinates and class probabilities. The method is based on dividing the entire image into S × S grid, where each cell predicts B bounding boxes and confidence scores for those boxes. The method is evaluated on 172 P&ID sheets.

The approach of Elyan et al. [3] in regard to data exploration performs a segmentation of an image. The original P&ID sheets are large images, 7500 × 5250 pixels. To speed up the training process they divide the sheet into 6 × 4 grid, resulting in 24 sub-images (patches) with relatively smaller sizes compared to the original sheets (1250 × 1300). The annotation data for each patch is obtained using the annotations for the whole P&ID.

Mani et al. [4] implemented a CNN model that performed symbol detection, text recognition and association, as well as connection detection. The symbol detection is enabled via the Convolutional Neural Network (CNN) that performs a three-way classification task. It determines whether an input image contained a tag, linear matrix inequality (LMI), or no symbol. The architecture of the CNN consists of three convolutional layers, with ReLU activations and max pooling, and two fully-connected dense layers. It inspired by the LeNet architectures popularized for digit recognition.

An important part of the Mani et al. [4] approach is the augmentation of the training examples by rotating, shifting, shearing, zooming, and flipping. This aims to make the network invariant to these transformations. To launch the trained CNN in order to detect symbols in a new diagram, they first slide over the diagram image with a small stride length and generate all 100 x 100 pixel windows from the input diagram.

Mani et al. [4] take next the step of text recognition and association. They implement an Efficient and Accurate Scene Text Detector (EAST), a state-of-the-art pipeline which uses a neural network to produce bounding boxes where text is present in an image. For each symbol, they identify associated text based on the proximity of the symbol to text bounding boxes, using a distance threshold. Associated text for each symbol is then interpreted using Tesseract OCR, and the results are added to the extracted information in the asset hierarchy.

Finally, Mani et al. [4] work on the connection detection using a graph search approach. The thresholded diagram image is represented by a graph, whose nodes are individual pixels in the diagram, while each node contains information on whether it is black or white (based on its thresholded pixel intensity) and whether it is part of a symbol (and if so, which one). The graph's edges form links between neighboring pixels, with a maximum of eight edges per node. The symbols are represented in the graph as a collection of nodes corresponding to the pixels that form the symbol.

Moreno-García et al. [5] implement a methodology for **pre-processing**, shape detection and extraction of features. As a first step they binarize the image by performing image thresholding. This is useful for removing noise and enhancing object localisation. Thinning or skeletonisation is another pre-processing method used on image recognition systems to discard the volume of an object often considered as redundant information. Skew correction can be achieved through morphological operations to remove salt-pepper type noise or algorithms based on morphology. Then, the **shape detection** follows, in order to detect images such as arrowheads, cross-hatched areas, arcs, dashed and dot-dashed lines. Vertical and horizontal lines are detected using morphological operations.

Chapter 4

Data Collection and methods

The data used in the framework of this project is of various types, aiming to cover all aspects of the P&ID digitization process. Therefore, the following types of data were gathered, pre-processed and applied in the various stages of the proposed methodology:

1. Textual data
2. Images
    (a) Icons
    (b) Plans

4.1 Textual data

The textual data refer to the identification of each instrument and devise (Tables 4.1 and 4.2), transducer functions (Table 4.3), and fluid service (Table 4.4) by the use of a letter or code. This is performed according to the ANSI/ISA-5.1.-1984 (R1992) standards. These are the international standard developed by the International Society of Automation for creating an automated interface between enterprise and control systems.

Pressure indicators e.g. have the abbreviation PI and temperature indicators the abbreviation TI. The coding has a logical order, where flow and level indicators use the abbreviations FI and LI, respectively. Since most plants can have many instruments of the same type, it is important to be able to identify each on uniquely. To this end, number is therefore applied. This number is often referred to as the ”loop number”[1]. Thus, the device abbreviation and loop number become the unique ”tag number” (Figure 4.1).

The letters on the fist line are defined based on the ISA standards and provide unique information (Fig. 4.2). The first letter defines the variable that the certain instrument measures, while the second letter, which is optional, describes a modification to the fist letter. E.g. the code ”PDIT” is interpreted as following: First Letter stands for ”Pressure” and the second letter is a modifier for the first and stands for ”Differential”. The third letter, which is again optional, defines an indication functionality. In the example of the code ”PDIT”, ”I”, which is the third letter stands for ”Indicating” . The fourth letter defines an output functionality as for example again in the case of the code ”PDIT” the letter ”T” stands for ”Transmitter”.

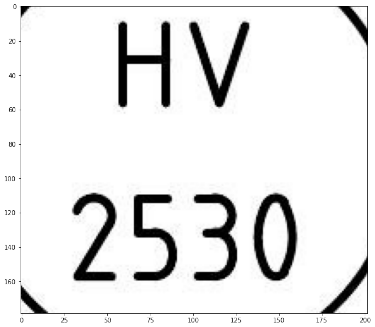
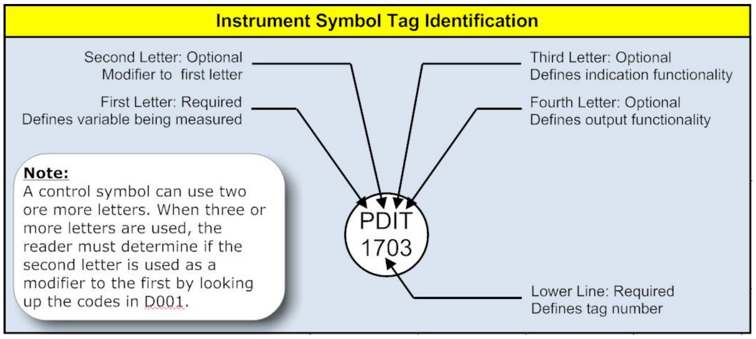


Figure 4.1: An example of a ”tag number”.



Note:
A control symbol can use two ore more letters. When three or more letters are used, the reader must determine if the second letter is used as a modifier to the first by looking up the codes in D001.

Figure 4.2: Instrument symbol tag identification (Source: aiche.org).

---

[1] Source: aiche.org

Table 4.1: Letters representing different variables.

Letter	**Variable**
A	ANALYZER
B	BURNER
C	USER’S CHOICE
D	USER’S CHOICE
E	VOLTAGE
F	FLOW
G	USER’S CHOICE
H	HAND
I	CURRENT
J	POWER
K	TIME
L	LEVEL
M	USER’S CHOICE
N	USER’S CHOICE
O	USER’S CHOICE
P	PRESSURE
Q	QUANTITY
R	RADIATION
S	SPEED
T	TEMPERATURE
U	MULTI-VARIABLE
V	VIBRATION
W	WEIGHT, FORCE
X	UNCLASSIFIED
Y	EVENT, STATE
Z	POSITION

Table 4.2: Letters representing different output functionalities.

Letter	Variable
B	USER'S CHOICE
C	CONTROL
K	CONTROL STATION
N	USER'S CHOICE
S	SWITCH
T	TRANSMIT
U	MULTI-FUNCTION
V	VALVE, DAMPER
X	UNCLASSIFIED
Y	RELAY, COMPUTE
Z	DRIVER, ACTUATOR UNCLASSIFIED FINAL CONTROL ELEMENT

Table 4.3: Letters representing different transducer functionalities.

Letter	Variable
E/E	VOLTAGE TO VOLTAGE
E/I	VOLTAGE TO CURRENT
E/P	VOLTAGE TO PNEUMATIC
I/P	CURRENT TO PNEUMATIC
P/I	PNEUMATIC TO CURRENT

Table 4.4: Letters representing different fluids.

Letter	Variable
ALM	ALUMINUM SULFATE
AMN	AMMONIUM NITRATE
AMH	AMMONIUM HYDROXIDE
ABF	AMMONIUM (BI)FLUORIDE
AMS	AMMONIUM SULFATE
ASO	ACID SOLUBLE ORGANICS
BAR	BACKWASH AIR
CAF	CALCIUM FLUORIDE
CAR	COMPRESSED AIR
CBW	CLEAN BACKWASH WATER
CFD	CAUSTIC RAW FEED
CO2	CARBON DIOXIDE
CHC	CALCIUM HYPOCHLORITE
CL2	CHLORINE
DBW	DIRTH BACKWASH WATER
DRN	PROCESS DRAIN
DSL	DIESEL FUEL
EFF	EFFLUENT (GENERAL USE)
FEC	FERRIC CHLORIDE
FEW	FILTER EFFLUENT WATER
FIW	FILTER INFLUENT WATER
FOL	FUEL OIL
HCL	HYDROCHLORIC ACID
HF	HYDROFLUORIC ACID
HPX	HYDROGEN PEROXIDE
IAR	INSTRUMENT AIR
IFD	INDUSTRIAL RAW FEED
LSY	LIME SLURRY
MEL	METHANOLS

Letter	Variable
NAG	NATURAL GAS
NIA	NITRIC ACID
N2	NITROGEN
OIL	OIL (GENERAL USE)
PAR	PROCESS AIR
PFD	POLYMER FEED
PHA	PHOSPHORIC ACID
KF	POTASSIUM FLUORIDE
KOH	POTASSIUM HYDROXIDE
PSL	PROCESS SLURRY/SLUDGE
PVP	PROCESS VAPOR
PWR	POTABLE WATER
SAH	SULFURIC ACID, >75%
SAL	SULFURIC ACID, <75%
SHC	SODIUM HYPOCHLORITE
SOC	SODIUM CARBONATE
SOH	SODIUM HYDROXIDE
SLP	STEAM, <125
SMB	SODIUM METABISULFITE
STM	STEAM, 125-220
SNY	SANITARY SEWER
STO	STORM DRAIN
SWR	SERVICE WATER
TFL	THERMAL FLUID
UAR	UTILITY AIR
UWR	UTILITY WATER
VNT	VENT (GENERAL USE)
WOL	WASTE OIL
WWR	WASTEWATER (GENERAL USE)

4.2 Images

Apart from the textual information, distinct drawings, representing each instrument, devise, pipe etc. In order to identify these components, diagrams are collected along with itemized icons for each instrument.

4.2.1 Icons

In the framework of this project, a synthetic dataset is created based on two main sources. The one source is the paper of Elyan et al. [6] which includes 2432 instances of engineering symbols scaled to 100 by 100 pixels. The second source is a list of PID Symbols provided by Projectmaterials. The 538 icons, deriving from the second source, are downloaded via a web crawling script developed for this purpose.

A total of 2970 icons belonging to 53 classes are finally gathered, e.g. ball valve, check valve, globe valve, gate valve, filters, compressors etc (Fig. 4.3). As can be seen in Table 4.5 and Figure 4.4, the dataset has not a good balance, in the sense that there are classes of icons that are underrepresented.

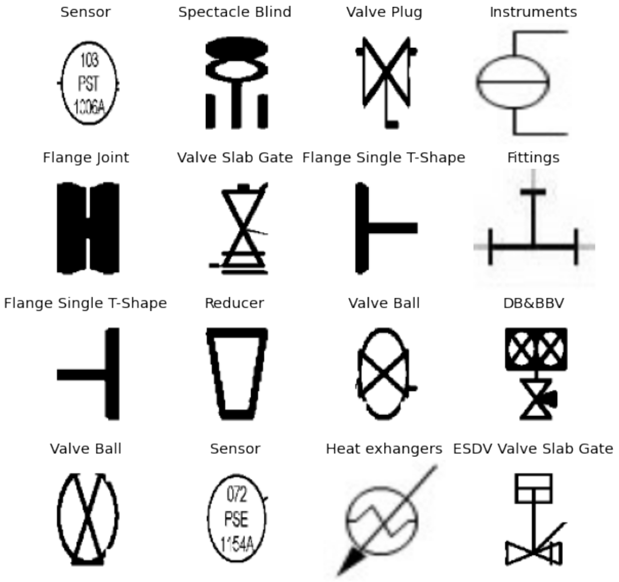


Figure 4.3: Sample of 16 images taken from the dataset used for the training, validation and testing of the model.

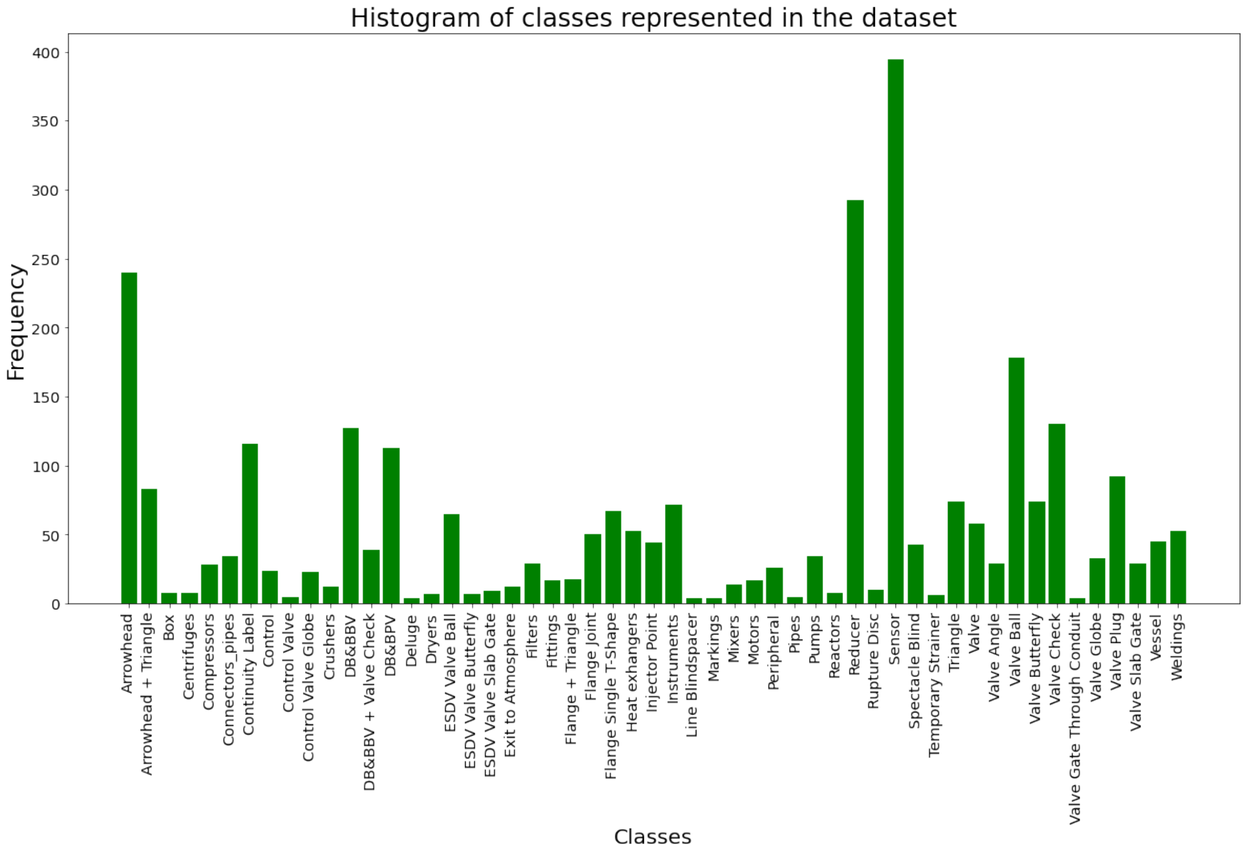


Figure 4.4: Histogram of classes represented in the dataset.

Table 4.5: Distribution of the classes.

Class	Frequency
Arrowhead	240
Arrowhead + Triangle	83
Box	8
Centrifuges	8
Compressors	28
Connectors pipes	34
Continuity Label	116
Control	24
Control Valve	5
Control Valve Globe	23
Crushers	12
DB &BBV	127
DB&BBV + Valve Check	39
DB&BPV	113
Deluge	4
Dryers	7
ESDV Valve Ball	65
ESDV Valve Butterfly	7
ESDV Valve Slab Gate	9
Exit to Atmosphere	12
Filters	29
Fittings	17
Flange + Triangle	18
Flange Joint	50
Flange Single T-Shape	67
Heat exchangers	53
Injector Point	44
Instruments	72
Line Blindspacer	4
Markings	4
Mixers	14
Motors	17
Peripheral	26
Pipes	5
Pumps	34
Reactors	8
Reducer	292
Rupture Disc	10
Sensor	394
Spectacle Blind	43
Temporary Strainer	6
Triangle	74
Valve	58
Valve Angle	29
Valve Ball	178
Valve Butterfly	74
Valve Check	130
Valve Gate Through Conduit	4
Valve Globe	33
Valve Plug	92
Valve Slab Gate	29
Vessel	45
Weldings	53

4.2.2 Piping & Instrumentation Diagram

In order to test the application created in the framework of this project real Piping & Instrumentation Diagrams (Fig. 4.5 and 4.6) are used. These are chosen based on the factor of complexity in order to start from simple examples and gradually move towards more complex diagrams.

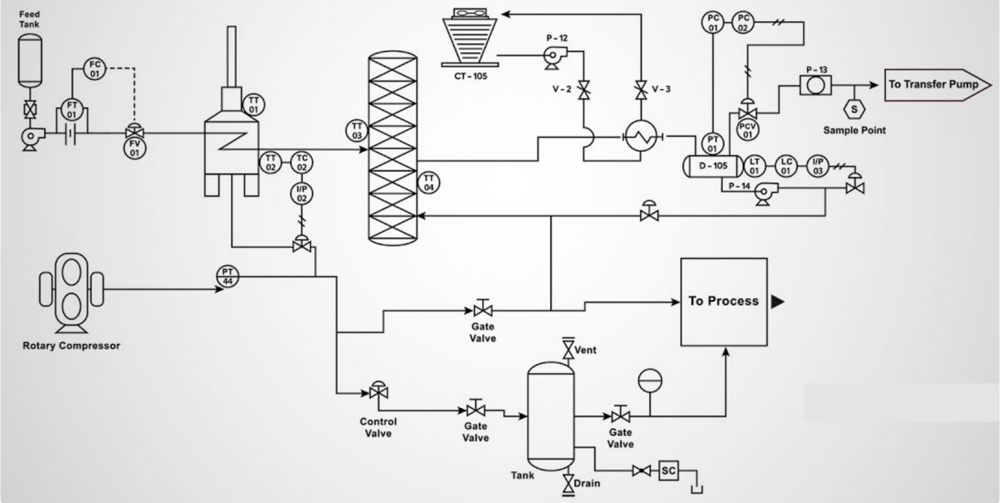


Figure 4.5: A Piping and Instrumentation Diagrams (Source: PID Symbols).

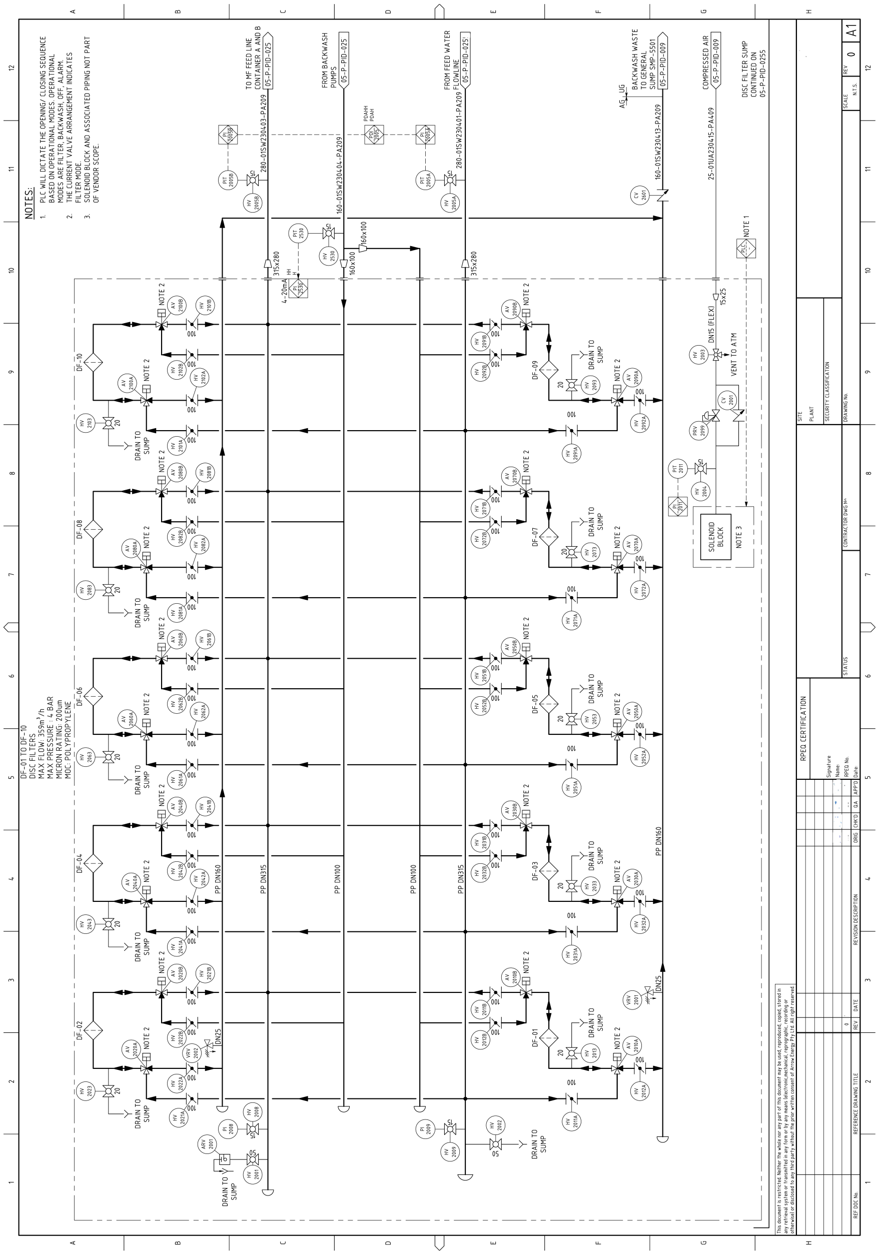


Figure 4.6: A Piping and Instrumentation Diagrams (Source: PID Symbols.

Chapter 5

Proposed methodology

As already mentioned in Chapter 1, the proposed methodology deals with the problem of the object and text detection on engineering plans. The methodology breaks down in several stages, starting with a simple identification of the outlines of the different elements of the P&ID, which yields the coordinates of the those to be labeled (Section 5.1). The next step is a convolutional neural network (Section 5.2.1) that is trained, validated and tested with a synthetic dataset (Section 4.2.1). Finally, an optical character recognition scripts system is developed that identifies the text on the P&ID and decodes its meaning (Section 5.3).

5.1 Identification of vertical and horizontal lines

The detection of the vertical and horizontal lines and therefore that of the outlines within the P&ID is performed in steps. First the image is imported and transformed from RGB format to grayscale format. This ends up with the 2-d dimensional digitized representation of the P&ID under study, that would simplify and enable the calculations. Thus, from now on, we process the two-dimensional array resulting from the transformation of the image to the gray-scale format. The library used at the step is `skimage`. Then, the image array is passed through the module `measure.find-contours` of the `skimage` library. This module identifies iso-valued contours in a 2D array for a given level value (Fig. 5.1). It uses the "marching squares" method[1] to compute the iso-valued contours of the image array. The array values are linearly interpolated to provide better precision in the determination of the output contours[2].

---

[1] Wikipedia. Retrieved November 24, 2021.

[2] Source: find-contours

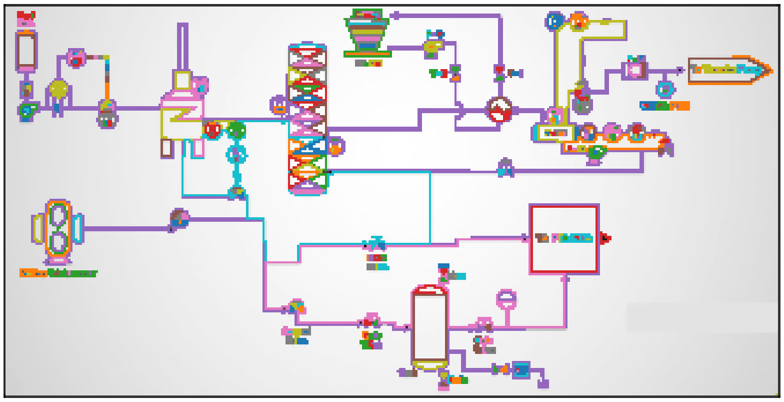


Figure 5.1: Contours extracted from the Piping and Instrumentation Diagrams of Fig. 4.5.

5.2 Identification of objects via a CNN model

This step consists of two main parts. The first part is the designation of the Convolutional Neural Network classifier that will classify the objects identified in the PID under study (Section 5.2.1). The second part consists of the process of extracting the distinct icons constituting the P&ID in order to pass them through the CNN model (Section 5.2.2).

5.2.1 CNN model

The classifier we define is a Convolutional Neural Network (Fig. 5.2) that consists entirely of connected layers. All convolution layers use the same kernel size (3x3) and all non-terminal layers use the rectified linear unit (relu) as the activation function. Downsampling layers use 2x2 pool dimensions.

The first (input) layer feeds into a 2-dimensional convolution layer (Conv2D) which is immediately followed by a downsampling step, effected by a 2-dimensional max pooling (MaxPooling2D) operation.

Next comes a 3-layer cascade of 2 identical (64-filter) Conv2D layers and MaxPooling2D. This layer cascade immediately repeats with larger (128) filter parameter values for the 2D convolution layers.

The outputs are then flattened and fed into a 2-layer cascade of a 128-unit densely-connected (Dense) layer kept in check by a 50% Dropout layer. The latter is connected to a similar 2-layer cascade comprising a 64-unit Dense layer followed by a Dropout layer configured with a 10% dropout rate.

The outputs of the model are produced by a Dense layer configured with as many units as there are classes and a softmax activation function for multiclass classification.

The model is trained, validated and tested based on the synthetic dataset presented in Section 4.2.1. The module `tf.keras.preprocessing.image_dataset_from_directory` is used in order to import the data. This module generates a `tf.data.Dataset` from image files in a directory, yielding batches of images from all the subdirectories, along with distinct labels[3]. The `tf.data.Dataset` consists of `Tensors`, which are multi-dimensional arrays with a uniform type.

The advantage of this call is that we can already incorporate some pre-processing steps, like the transformation from RGB format to gray-scale format, as well as the resizing of the image (Listing 5.1). Additionally, the batch size given as argument in this module, defines the grouping of data. The classes are inferred based on the names of the sub-folders.

```
data_ds = tf.keras.preprocessing.image_dataset_from_directory(path,
                                                    color_mode="grayscale",
                                                    image_size=(100, 100),
                                                    batch_size=32,
                                                    labels='inferred',
                                                    label_mode='categorical')
```

Listing 5.1: Import dataset from directory using the module `tf.keras.preprocessing.image_dataset_from_directory`.

Once the data are imported and transformed into batches of `Tensors`, we split them in a training, validation and test split. We chose a 70% - 15% - 15% size for splitting. In order to randomize the iteration order, we call `shuffle=True` as argument. By performing the split before importing the data in the model we ensure that each dataset (training, validation and test) is disjoint from the others and each one remains an unseen dataset until we decide otherwise.

Once the data is splitted, we proceed with the training and validation of the model. The input shape is `(100, 100, 1)` and the number of classes is 53, corresponding to the total of classes created at the step of the synthetic data preparation. We go on using the `EarlyStopping` callback,

---

[3] Source: tf.keras.utils.image-dataset-from-directory

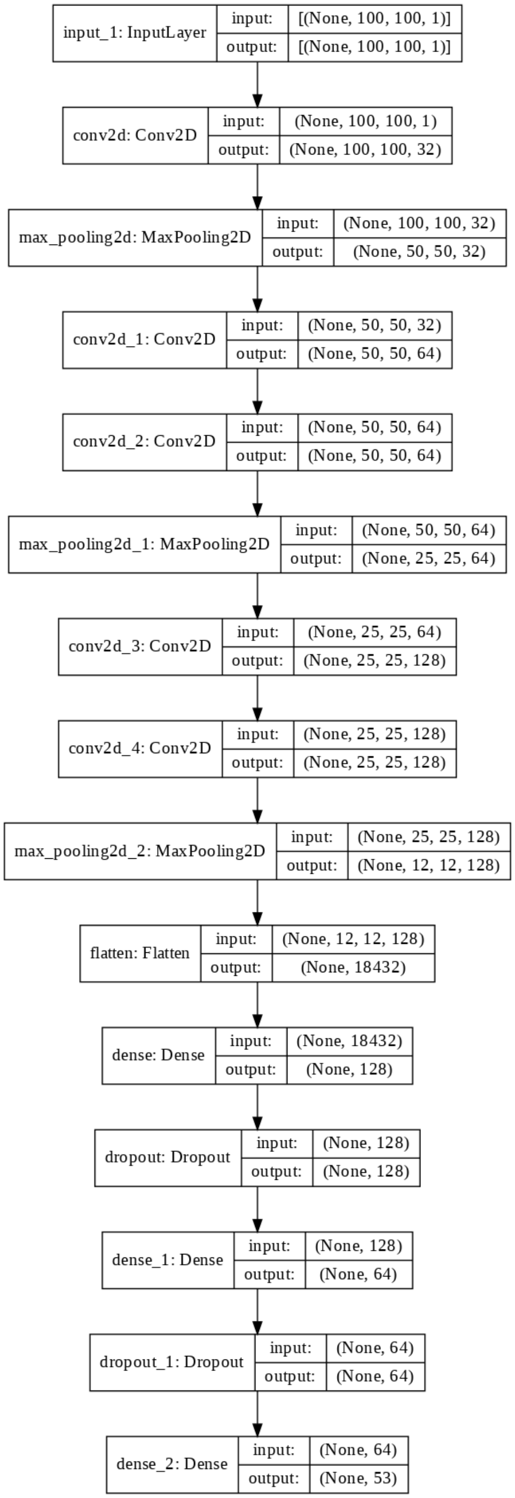

input_1: InputLayer
input: [(None, 100, 100, 1)]
output: [(None, 100, 100, 1)]
conv2d: Conv2D
input: (None, 100, 100, 1)
output: (None, 100, 100, 32)
max_pooling2d: MaxPooling2D
input: (None, 100, 100, 32)
output: (None, 50, 50, 32)
conv2d_1: Conv2D
input: (None, 50, 50, 32)
output: (None, 50, 50, 64)
conv2d_2: Conv2D
input: (None, 50, 50, 64)
output: (None, 50, 50, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 50, 50, 64)
output: (None, 25, 25, 64)
conv2d_3: Conv2D
input: (None, 25, 25, 64)
output: (None, 25, 25, 128)
conv2d_4: Conv2D
input: (None, 25, 25, 128)
output: (None, 25, 25, 128)
max_pooling2d_2: MaxPooling2D
input: (None, 25, 25, 128)
output: (None, 12, 12, 128)
flatten: Flatten
input: (None, 12, 12, 128)
output: (None, 18432)
dense: Dense
input: (None, 18432)
output: (None, 128)
dropout: Dropout
input: (None, 128)
output: (None, 128)
dense_1: Dense
input: (None, 128)
output: (None, 64)
dropout_1: Dropout
input: (None, 64)
output: (None, 64)
dense_2: Dense
input: (None, 64)
output: (None, 53)

Figure 5.2: Architecture of the model trained with raw data.

monitoring the validation loss and applying a patience of 5 epochs.

The compilation of the model uses the `Adam` optimizer with a rate of $1e-3$, monitors the loss based on the categorical cross-entropy, while calculating the `accuracy` of the model at each step.

Finally, the model fits to the training dataset, using the validation dataset for validating each wights at each epoch, while it runs for 50 epochs at the most, due to the call of the `EarlyStopping` callback.

An alternative approach is implemented, where the data used during training are augmented (Fig. 5.3). In this case, the architecture remained the same, but the input layer was passed through an augmentation process (Listing 5.2). This sequential model augments the data, by applying random contrast, flip (both vertical and horizontal), rotation, zoom as well as rescaling.

```
data_augmentation = tf.keras.Sequential(
    [# adjust the contrast of an image or images by a random factor
     layers.experimental.preprocessing.RandomContrast(factor=1.0,
                                                     seed=None),
    # randomly flip each image horizontally and vertically.
    layers.experimental.preprocessing.RandomFlip(mode="horizontal",
                                                 seed=123),
    # randomly flip each image horizontally and vertically.
    layers.experimental.preprocessing.RandomFlip(mode="vertical",
                                                 seed=123),
    # randomly rotate each image
    layers.experimental.preprocessing.RandomRotation(0.1, seed=123),
    # random zoom
    layers.experimental.preprocessing.RandomZoom(height_factor=(-1,1),
                                                 width_factor=None,
                                                 fill_mode='reflect',
                                                 interpolation='bilinear',
                                                 seed=None,
                                                 fill_value=0.0),
    # rescale an input in the [0, 255] range to be in the [0, 1] range, you would pass scale=1./255
    layers.experimental.preprocessing.Rescaling(1./255)])
```

Listing 5.2: Sequential model for data augmentation.

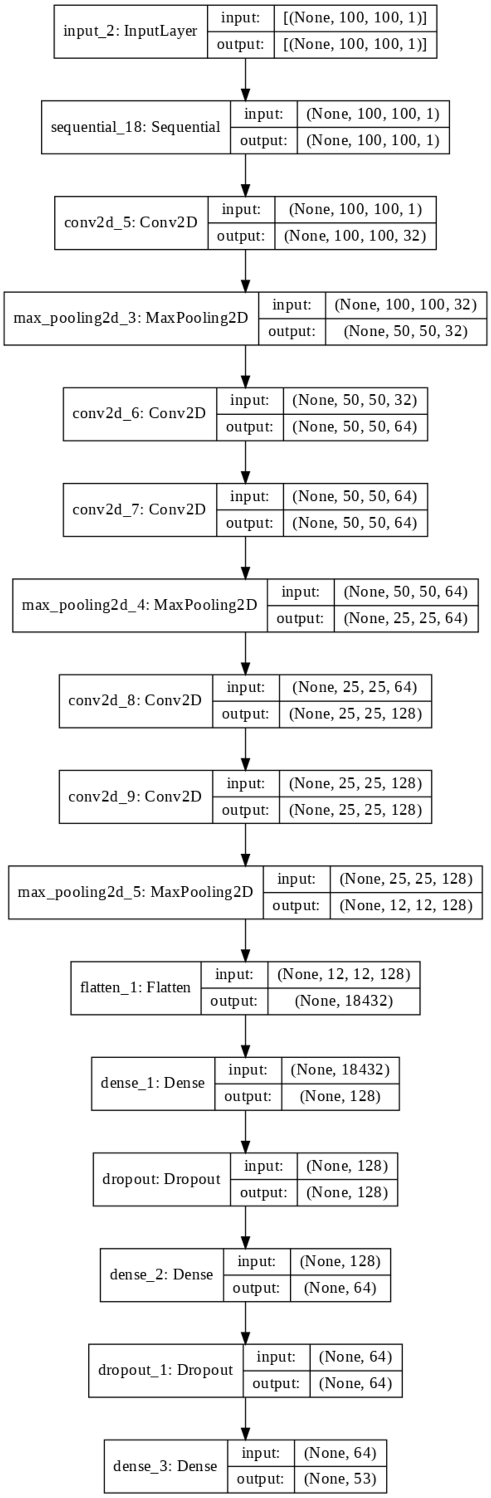


Figure 5.3: Architecture of the model trained with augmented data.

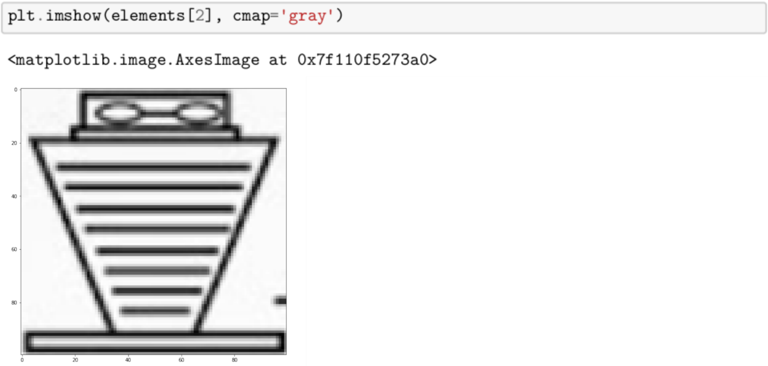
5.2.2 Extraction of P&ID's components

The extraction of the components encountered in a piping and instrumentation diagram is based on the contours created at previous step (Section 5.1). Four main lists are created, X_min, X_max, Y_min and Y_max. These list contain the minimum and maximum coordinate of each element stored in the contours. Then the image gets cropped iteratively based on a rolling window. This window has as coordinates the tuples (X_min, Y_min) and (X_max, Y_max). Then the output of this process gets resized at `(100, 100)`, in order to fit the input size of the CNN model (Listing 5.3).

```python
from skimage import color
elements = [] # each element of the PID  diagram
for i in coord_cont:
    image_cropped = img[X_min[i]:X_max[i],
                   Y_min[i]:Y_max[i]]
    resized = cv2.resize(image_cropped,
                         (100, 100),
                         interpolation = cv2.INTER_AREA)
    elements.append(resized)
```

Listing 5.3: Extraction of P&D elements.

In order to ensure that we fit in this window the whole instrument, devise etc, we expand its size by 50 pixels to the left and lower limit, and 100 pixels to the right and upper limit. We go on and do the same, by expanding the window 250 pixels on all directions. The additional cropped images are also stored. Each element that gets produced via this process, gets fed into the model in order to get a prediction on the class it belongs to.



```
plt.imshow(elements[2], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f110f5273a0>
```

Figure 5.4: Example of a cropped element taken from the Piping and Instrumentation Diagrams of Fig. 4.5.

5.3 Optical character recognition of instruments

The optical character recognition task is performed in steps. The first step is the pre-processing of the image, that includes the transformation from RGB format to grey-scale format, the resize as well as the smoothing of the image using `OpenCV` Gaussian Blur. The application of the Gaussian filter aims to the reducing of the noise at the image.

The next step is the identification of the circles in the P&ID. We already know that the textual information is included in circles (Section 4.1), located near the instrument they want to describe. We execute the module `cv2.HoughCircles` that takes as input the de-noised image and implements the `cv2.HOUGH_GRADIENT` method. We therefore need three parameters to define a circle, $C : (x_{center}, y_{center}, r)$, where (x_{center}, y_{center}) define the center position and r is the radius. Once the circles are identified, as in the case of Fig. 5.5, we first crop the image that lies beneath each circle, using the circles as the rolling window and working iteratively (Fig. 4.1). Upon each cropped image we run the module `pytesseract.image_to_string`. `pytesseract`, which is an optical character recognition (OCR) tool, recognizes and "reads" the text embedded in images (Fig. 5.6). The text extracted from the cropped images is stored in a list.

The next step is the decoding of the textual information, of the tags attributed to each instrument. The codes presented in Section 4.1 and included in the Tables 4.1, 4.2, 4.3 and 4.4 are set as dictionaries in Python. The rationale of the tag identification, again presented in Section 4.1, is implemented via a script that takes as input a **string**, which is the textual part of the tag and provides as output the translation of each letter (Listing 5.4). If we run, e.g. *decipher_tag("FV")* we take as output *"FLOW", "VALVE, DAMPER"*.

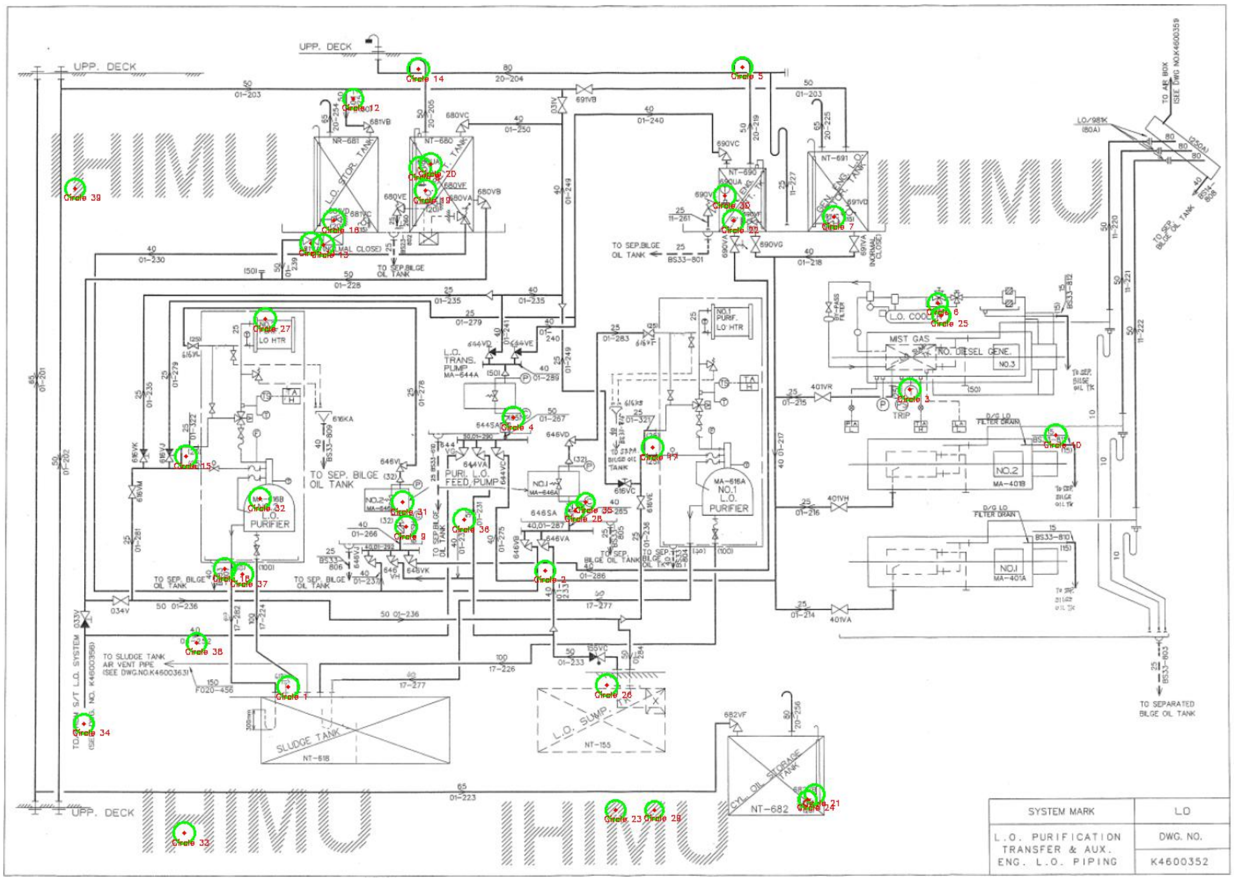


Figure 5.5: Example of the identification of the circles included at the Piping and Instrumentation Diagrams of Fig. 4.6.

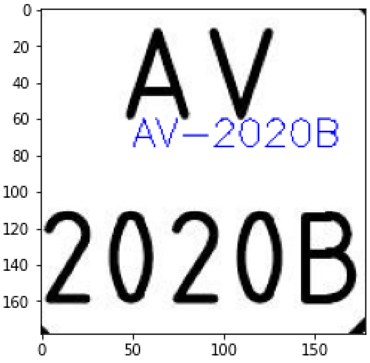


Figure 5.6: An example of a ”tag number” that goes through the optical character recognition process. The outcome of the process is plotted on the image.

```
def decipher_tag(text): # input is a string
    text = text.upper()
    variable = ''
    modifier = ''
    indication_functionality = ''
    output_functionality = ''
    modifier_of_functionality = ''
    tags = []

    if len(text) == 2:
        if text[0] in variables.keys():
            variable = text[0]
            tags.append(variables[variable])

        if text[1] in indication_functionalities.keys():
            indication_functionality = text[1]
            tags.append(indication_functionalities[indication_functionality])

        if text[1] in output_functionalities.keys():
            output_functionality = text[1]
            tags.append(output_functionalities[output_functionality])

    if len(text) == 3:
        if text[0] in variables.keys():
            variable = text[0]
            tags.append(variables[variable])

        if text[1] in indication_functionalities.keys():
            indication_functionality = text[1]
            tags.append(indication_functionalities[indication_functionality])

        if text[2] in output_functionalities.keys():
            output_functionality = text[2]
            tags.append(output_functionalities[output_functionality])

    if len(text) == 4:
        if text[0] in variables.keys():
            variable = text[0]
            tags.append(variables[variable])

```

```python
        if text[1] in modifiers.keys():
            modifier = text[1]
            tags.append(modifiers[modifier])

        if text[2] in indication_functionalities.keys():
            indication_functionality = text[2]
            tags.append(indication_functionalities[indication_functionality])

        if text[3] in output_functionalities.keys():
            output_functionality = text[3]
            tags.append(output_functionalities[output_functionality])

    if len(text) == 5:
        if text[0] in variables.keys():
            variable = text[0]
            tags.append(variables[variable])

        if text[1] in modifiers.keys():
            modifier = text[1]
            tags.append(modifiers[modifier])

        if text[2] in indication_functionalities.keys():
            indication_functionality = text[2]
            tags.append(indication_functionalities[indication_functionality])

        if text[3] in output_functionalities.keys():
            output_functionality = text[3]
            tags.append(output_functionalities[output_functionality])

        if text[4] in modifier_of_functionalities.keys():
            modifier_of_functionality = text[4]
            tags.append(modifier_of_functionalities[modifier_of_functionality])

    return tags
```

Listing 5.4: Instrument symbol tag identification

Chapter 6

Results

6.1 Convolutional Neural Network

The training and validation of the CNN was performed on the 70%-15% of the original dataset (Section 4.2.1. The accuracy during training reached 82.88 %, while during validation went up to 92.31 % (Tab. 6.1). The learning curve (Fig. 6.1) indicates that until approximately epoch 5 the model learns very quick, since the inclination of the curve is very steep. After epoch 5 the learning rate gets smaller and the model still learns but at a rather slower rate.

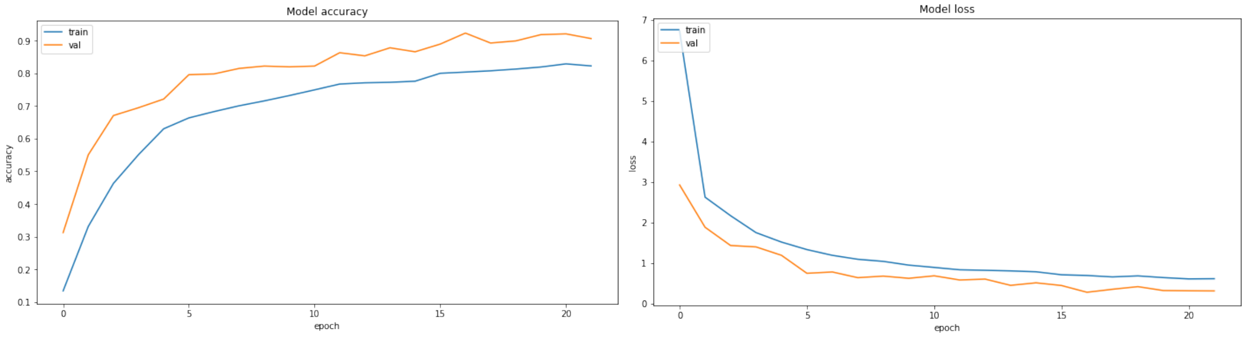
An interesting feature that we identify is the fact that the curve of validation lies above the one of the training. We know for sure that the validation data, are unseen data for the model during training. Therefore, we could assume that the model learns the features of the data during training really well, so that it can perform outstanding during validation. However, this issue needs some further investigation.

Parallel to the accuracy, the model calculates in each step the losses, which is the quantity that a model should seek to minimize during training (Fig. 6.1b and Tab. 6.1).

The evaluation of the model is performed based on the unseen test dataset (Fig. 6.2 and 6.3)

training	loss	accuracy
	6.72	82.88 %
validation	loss	accuracy
	2.93	92.31 %

Table 6.1: Training and validation loss and accuracy.
Model trained on raw data.

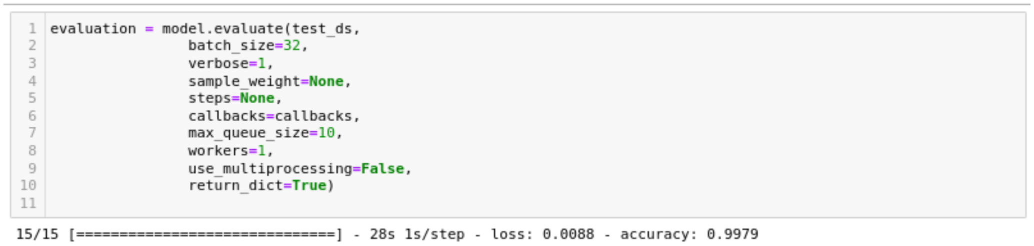


(a) Accuracy.

(b) Loss.

Figure 6.1: Learning curves of model trained with raw data.

and reached a 99.79 % of accuracy.



```
evaluation = model.evaluate(test_ds,
                    batch_size=32,
                    verbose=1,
                    sample_weight=None,
                    steps=None,
                    callbacks=callbacks,
                    max_queue_size=10,
                    workers=1,
                    use_multiprocessing=False,
                    return_dict=True)

15/15 [==============================] - 28s 1s/step - loss: 0.0088 - accuracy: 0.9979
```

Figure 6.2: Evaluation of the model.

The next step is to feed the trained model with an image extracted from a P&ID (Fig. 4.5). We chose the image of the compressor as presented at Fig. 6.4. The CNN model trained with the original data, not the augmented data, predicted that the image belongs to Class 5, which is a compressor, with a probability of 28.81 % (Fig. 6.5). The barplot of predictions for this image, meaning the probability that the model attributed to each class (Fig. 6.6), indicates that the model succeeds in attributing the object to the correct class, giving low probabilities (< 12 %) to the 52 other classes.

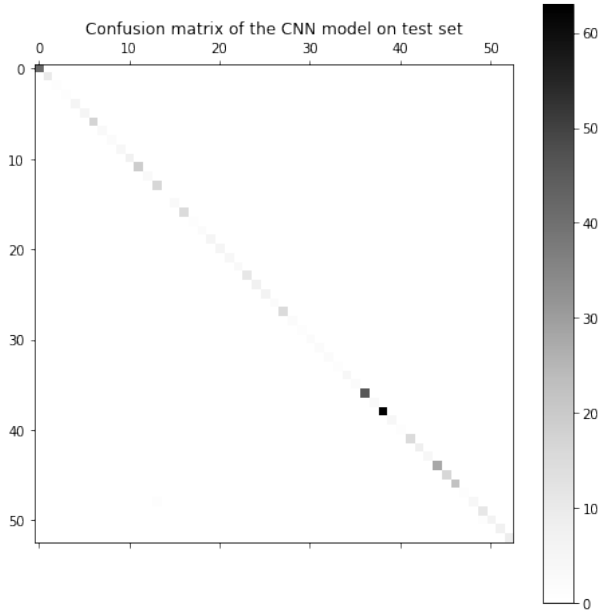


Figure 6.3: Confusion matrix of the CNN model on test set.

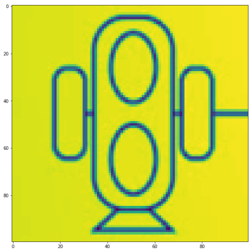
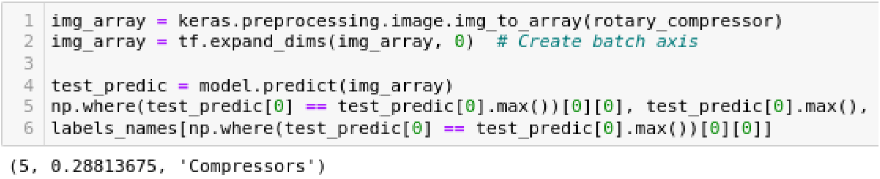


Figure 6.4: Instance of a P&ID (Fig. 4.5) representing a compressor.



```
img_array = keras.preprocessing.image.img_to_array(rotary_compressor)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

test_predic = model.predict(img_array)
np.where(test_predic[0] == test_predic[0].max())[0][0], test_predic[0].max(),
labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]]
```

```
(5, 0.28813675, 'Compressors')
```

Figure 6.5: Prediction of the class of an instance of a P&ID (Fig. 4.5 representing a compressor).

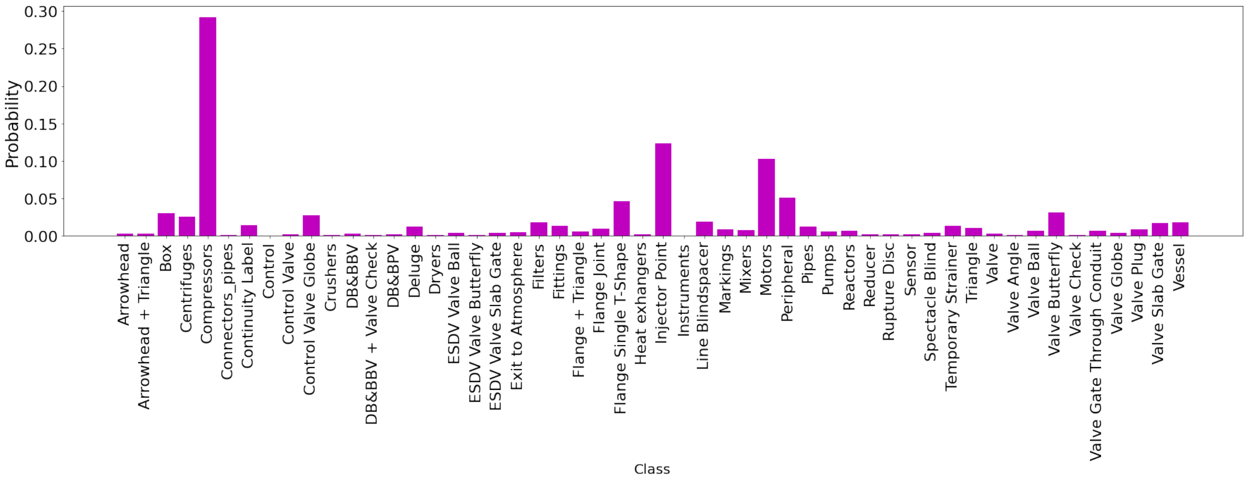
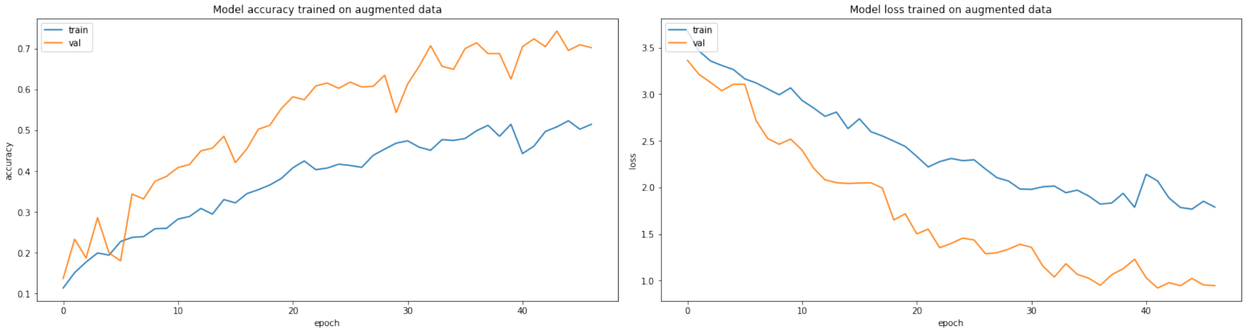


Figure 6.6: Barplot of the probability attributed to each class for the compressor.



(a) Accuracy.

(b) Loss.

Figure 6.7: Learning curves of model trained with augmented data.

training	loss 3.68	accuracy 52.31 %
validation	loss 3.37	accuracy 74.28 %

Table 6.2: Training and validation loss and accuracy.
Model trained on augmented data.

The model trained on the augmented data performs worse than the one presented before, during training and validation. It reaches an accuracy of 52.31 % during training and 74.28 % during validation (Tab. 6.2). These results make sense if we take into account that the augmentation alters significantly the images, making the training really difficult. However, the validation data, which are non-augmented data, seem to work better, bringing the accuracy quite high, compared to the training. The learning curves (Fig. 6.7) indicate again that the model still manages to learn during training so that it is in the position to perform better during validation. The inclination of the accuracy curve seems to be reaching almost to zero a little bit before epoch 50.

6.2 Optical character recognition

The methodology and algorithms described in Section 5.3 result to the identification of the tags of the different instruments and devices along with their deciphering (Listing 6.1). In P&ID of figure 4.6 97 circles are identified (Fig. 6.8). The text in these circles is given as input to the OCR algorithm and the labels are identified (Table 6.3). Out of the 97 tags, 69 tags (71.13%) is identified correctly in regard to its first part (e.g. Table 6.3, entry nr. 1) and 62 tags (63.92%) is identified correctly in regard to its second part (e.g. Table 6.3, entry nr. 2). A total of 8 tags (8.25%) are not identified at all (e.g. Table 6.3, entry nr. 18).

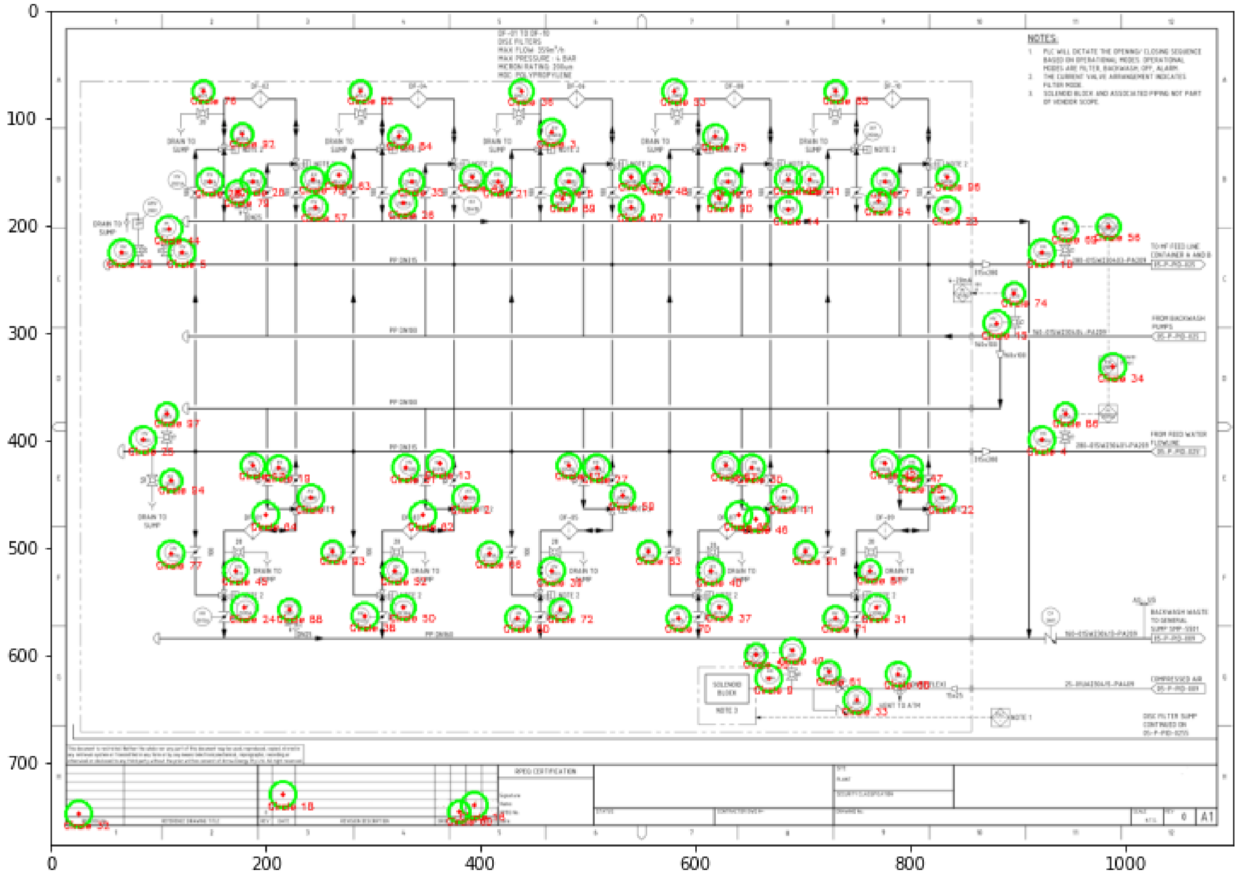


Figure 6.8: Identification of 97 circles found on the P&ID (Fig. 4.6).

The list of the tags or labels are passed to the algorithm that deciphers the text. On a total of 93 tags, 58 were deciphered (59.78%). 8 of the tags were totally missing as already discussed, while the other 31 were either lacking the first part or the letters were wrongly OCRed (Tables 6.4, 6.5, 6.6 and 6.7).

nr.	text OCR-ed
1	BN-
2	-LV5VD
3	AV-
4	QV-
5	N-XO
6	TS-OK
7	-RK
8	-OK
9	-LVUUF
10	-
11	CO-ZLVIVD
12	HVY-2052B
13	-031B
14	C-42VUUIV
15	HV-
16	TT-T
17	HV-20128J
18	-
19	V-
20	P-RK
21	N-OA
22	-
23	-421VIV
24	Y-0108
25	Q-
26	-
27	CHV-
28	-ER
29	-
30	HV-AOI
31	SY-00904
32	-REI
33	-2001
34	PDINF-WA
35	-CR
36	HV-2063
37	-
38	HV-03247
39	HV-
40	HV-
41	HV-
42	JN-20928
43	-2040B
44	PI-LVUVO0
45	HV-2013
46	OO-
47	V-
48	HVY-20BTAT
49	-2011
50	SN-C
51	HV-
52	HV-2033
53	HV-2083
54	HV-
55	ZFIN-NY
56	JPIN-X05
57	Y-
58	-
59	AV-
60	S-
61	PRV-2099J
62	-3
63	LN-
64	-1
65	-07
66	HV-
67	-
68	HV-
69	PIT-LUND
70	HV-
71	AVN-
72	YAV-2050A
73	AN-2060B
74	PIT-2530
75	AV-
76	HV-2023J
77	HV-
78	AV-
79	RV-2002
80	HV-
81	HV-2093
82	-2043
83	HV-20T1A
84	AV-20404
85	HYV-2103
86	PIT-2005A
87	-20728
88	VRV-2001
89	CU-2062A
90	-2082A
91	-20914
92	-2020A
93	HVY-2031A
94	HV-2002
95	AV-
96	A-2100B
97	PS-2009

Table 6.3: Result of the OCR performed on the P&ID of figure 6.8.

nr.	OCR-ed	deciphered
1	BN-	[’BURNER’, ”USER’S CHOICE”, ”USER’S CHOICE”]
2	-LV5VD	[]
3	AV-	[’ANALYZER’, ’VALVE, DAMPER’]
4	QV-	[’QUANTITY’, ’VALVE, DAMPER’]
5	N-XO	[]
6	TS-OK	[’TEMPERATURE’, ’SWITCH’]
7	-RK	[]
8	-OK	[]
9	-LVUUF	[]
10	-	[]
11	CO-ZLVIVD	[”USER’S CHOICE”, ’OFFICE’]
12	HVY-2052B	[’HAND’, ’RELAY, COMPUTE’]
13	-031B	[]
14	C-42VUUIV	[]
15	HV-	[’HAND’, ’VALVE, DAMPER’]
16	TT-T	[’TEMPERATURE’, ’TRANSMIT’]
17	HV-20128J	[’HAND’, ’VALVE, DAMPER’]
18	-	[]
19	V-	[]
20	P-RK	[]
21	N-OA	[]
22	-	[]
23	-421VIV	[]
24	Y-0108	[]
25	Q-	[]

Table 6.4: Part 1: Result of the deciphering of the OCR text performed on the P&ID of figure 6.8.

nr.	OCR-ed	deciphered
26	-	[]
27	CHV-	[”USER’S CHOICE”, ’VALVE, DAMPER’]
28	-ER	[]
29	-	[]
30	HV-AOI	[’HAND’, ’VALVE, DAMPER’]
31	SY-00904	[’SPEED’, ’RELAY, COMPUTE’]
32	-REI	[]
33	-2001	[]
34	PDINF-WA	[’PRESSURE’, ’DIFFERENTIAL’, ’INDICATE’, ”USER’S CHOICE”]
35	-CR	[]
36	HV-2063	[’HAND’, ’VALVE, DAMPER’]
37	-	[]
38	HV-03247	[’HAND’, ’VALVE, DAMPER’]
39	HV-	[’HAND’, ’VALVE, DAMPER’]
40	HV-	[’HAND’, ’VALVE, DAMPER’]
41	HV-	[’HAND’, ’VALVE, DAMPER’]
42	JN-20928	[’POWER’, ”USER’S CHOICE”, ”USER’S CHOICE”]
43	-2040B	[]
44	PI-LVUVO0	[’PRESSURE’, ’INDICATE’]
45	HV-2013	[’HAND’, ’VALVE, DAMPER’]
46	OO-	[”USER’S CHOICE”, ’OFFICE’]
47	V-	[]
48	HVY-20BTAT	[’HAND’, ’RELAY, COMPUTE’]
49	-2011	[]
50	SN-C	[’SPEED’, ”USER’S CHOICE”, ”USER’S CHOICE”]

Table 6.5: Part 2: Result of the deciphering of the OCR text performed on the P&ID of figure 6.8.

nr.	OCR-ed	deciphered
51	HV-	[’HAND’, ’VALVE, DAMPER’]
52	HV-2033	[’HAND’, ’VALVE, DAMPER’]
53	HV-2083	[’HAND’, ’VALVE, DAMPER’]
54	HV-	[’HAND’, ’VALVE, DAMPER’]
55	ZFIN-NY	[’POSITION’, ’RATIO’, ’INDICATE’, ”USER’S CHOICE”]
56	JPIN-X05	[’POWER’, ’INDICATE’, ”USER’S CHOICE”]
57	Y-	[]
58	-	[]
59	AV-	[’ANALYZER’, ’VALVE, DAMPER’]
60	S-	[]
61	PRV-2099J	[’PRESSURE’, ’RECORD’, ’VALVE, DAMPER’]
62	-3	[]
63	LN-	[’LEVEL’, ”USER’S CHOICE”, ”USER’S CHOICE”]
64	-1	[]
65	-07	[]
66	HV-	[’HAND’, ’VALVE, DAMPER’]
67	-	[]
68	HV-	[’HAND’, ’VALVE, DAMPER’]
69	PIT-LUND	[’PRESSURE’, ’INDICATE’, ’TRANSMIT’]
70	HV-	[’HAND’, ’VALVE, DAMPER’]
71	AVN-	[’ANALYZER’, ”USER’S CHOICE”]
72	YAV-2050A	[’EVENT, STATE’, ’ALARM’, ’VALVE, DAMPER’]
73	AN-2060B	[’ANALYZER’, ”USER’S CHOICE”, ”USER’S CHOICE”]
74	PIT-2530	[’PRESSURE’, ’INDICATE’, ’TRANSMIT’]
75	AV-	[’ANALYZER’, ’VALVE, DAMPER’]

Table 6.6: Part 3: Result of the deciphering of the OCR text performed on the P&ID of figure 6.8.

nr.	OCR-ed	deciphered
76	HV-2023J	[’HAND’, ’VALVE, DAMPER’]
77	HV-	[’HAND’, ’VALVE, DAMPER’]
78	AV-	[’ANALYZER’, ’VALVE, DAMPER’]
79	RV-2002	[’RADIATION’, ’VALVE, DAMPER’]
80	HV-	[’HAND’, ’VALVE, DAMPER’]
81	HV-2093	[’HAND’, ’VALVE, DAMPER’]
82	-2043	[]
83	HV-20T1A	[’HAND’, ’VALVE, DAMPER’]
84	AV-20404	[’ANALYZER’, ’VALVE, DAMPER’]
85	HYV-2103	[’HAND’, ’VALVE, DAMPER’]
86	PIT-2005A	[’PRESSURE’, ’INDICATE’, ’TRANSMIT’]
87	-20728	[]
88	VRV-2001	[’VIBRATION’, ’RECORD’, ’VALVE, DAMPER’]
89	CU-2062A	[”USER’S CHOICE”, ’MULTI-FUNCTION’, ’MULTI-FUNCTION’]
90	-2082A	[]
91	-20914	[]
92	-2020A	[]
93	HVY-2031A	[’HAND’, ’RELAY, COMPUTE’]
94	HV-2002	[’HAND’, ’VALVE, DAMPER’]
95	AV-	[’ANALYZER’, ’VALVE, DAMPER’]
96	A-2100B	[]
97	PS-2009	[’PRESSURE’, ’SWITCH’]

Table 6.7: Part 4: Result of the deciphering of the OCR text performed on the P&ID of figure 6.8.

```
import cv2
import numpy as np
import pytesseract
import matplotlib.pyplot as plt
import imutils
import re

def load_image(url):
    img_path = url
    img = cv2.imread(img_path, 1)
    return img

def preprocessing(img, resized_width): # try for resized_width = 1400
    # store height, width,
    # depth and ratio
    h, w, d = img.shape
    ratio = resized_width / w

    # resize image
    resized_image = imutils.resize(img, width=resized_width)

    # tranform to grayscale
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

    # blur image
    blur_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

    return resized_image, gray_image, blur_image, ratio

def find_circles(img, blur_image, resized_image, minRadius, maxRadius):
    # identify circles
    circles = cv2.HoughCircles(
        blur_image,
        method=cv2.HOUGH_GRADIENT,
        dp=0.1,
        minDist=10,
        param1=20,
        param2=10,
        minRadius = minRadius,
        maxRadius= maxRadius
    )

    # smooth to integer
    circles = np.uint16(np.around(circles))
    print(str(circles.shape[1]) + " circles found on the pi&d")

    count = 1
    img_circles = resized_image.copy()
    for circle in circles[0]:
        # Annotate circle and centroid
        cv2.circle(img_circles, (circle[0], circle[1]), circle[2], (0, 255, 0), 2)
        cv2.circle(img_circles, (circle[0], circle[1]), 2, (255, 0, 0), -2)

        # Annotate text
        offset_txt = int(circle[2] * 1.2)
        cv2.putText(
            img_circles,
            "Circle " + str(count),
            (circle[0] - offset_txt, circle[1] + offset_txt),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.3,
            (255, 0, 0),
            1,
        )
        count += 1

    plt.rcParams['figure.figsize'] = (16,9)
    plt.imshow(img_circles)
    plt.imsave(fname='img_output_circles.jpg',
               arr=img_circles)

    return circles

def ocr_text(img, circles, ratio):
    # Read information in every circle
    cropped_imgs = []
    cropped_imgs_txt = []
    img_circle_txt = img.copy()

```

```python
    circles_int = (circles[0] // ratio).astype(int)

    for circle in circles_int:
        up = ""
        low = ""

        x_offset_right = np.uint16(circle[2] * 0.79)
        x_offset_left = np.uint16(circle[2] * 0.65)
        y_offset_low = np.uint16(circle[2] * 0.72)
        y_offset_up = np.uint16(circle[2] * 0.56)

        cropped_img_lower = img_circle_txt[
            circle[1] : circle[1] + y_offset_low,
            circle[0] - x_offset_left : circle[0] + x_offset_right,
        ]
        cropped_img_upper = img_circle_txt[
            circle[1] - y_offset_up : circle[1],
            circle[0] - x_offset_left : circle[0] + x_offset_right,
        ]
        cropped_imgs.append(np.append(cropped_img_upper, cropped_img_lower, axis=0))

        upper = pytesseract.image_to_string(
            cropped_img_upper,
            lang="eng",
            config="--psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ",)
        lower = pytesseract.image_to_string(
            cropped_img_lower,
            config="--psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789",)

        r_upper = re.compile(r"[A-Z]+")
        r_lower = re.compile(r"[A-Z0-9]+")

        if not upper.isspace():
            if re.match(r_upper, str(upper)).group(0) is not None:
                up = re.match(r_upper, str(upper)).group(0)
        if not lower.isspace():
            if re.match(r_lower, str(lower)).group(0) is not None:
                low = re.match(r_lower, str(lower)).group(0)

        cropped_imgs_txt.append(up + "-" + low)

    return cropped_imgs, cropped_imgs_txt, img_circle_txt, circles_int

def read_text_in_circles(url, resized_width, minRadius, maxRadius):
    img = load_image(url)
    resized_image, gray_image, blur_image, ratio = preprocessing(img, resized_width)
    circles = find_circles(img, blur_image, resized_image, minRadius,maxRadius)
    cropped_imgs, cropped_imgs_txt, img_circle_txt, circles_int = ocr_text(img, circles, ratio)

    return circles, img_circle_txt, cropped_imgs, cropped_imgs_txt

circles, img_circle_txt, cropped_imgs, cropped_imgs_txt = read_text_in_circles('output1.jpg',

    1100, 10, 13)
```

Listing 6.1: Identification of circles and OCR of text of P&D elements.

Chapter 7

Limitations

The results presented at Chapter 6 show the potential of the methodology developed in the framework of this project to deal with the identification of the components that constitute a P&ID. In this Chapter we refer to the challenges faced in this project, the limitations of the methodology, as well as to suggest ideas on how to overcome and/or mitigate.

The method succeeds in identifying an element of the P&ID of Fig. 4.5, meaning the particular compressor, but fails to do the same for the other elements belonging to other classes. That means that the model currently isn't resilient enough on a dataset outside of the one we used. Let us recall that the synthetic dataset presented in Section 4.2.1 consists of data coming from two sources. This means that, on the one hand, we do have a total of 53 classes represented, we do however have the representation only based on two sources. This implies that there is not enough diversity in the data. Additionally, as already discussed in Section 4.2.1, the dataset is not in good balance, since many classes are under-represented. In order to address this challenge we could use a balanced dataset when training the model, and build a more diverse synthetic dataset based on data originating from different sources (e.g., by emplying a GAN neural network). This is a way to avoid overfitting of the model to the original data.

Another challenge faced is the identification of the size of the rolling window for each element across the P&ID. The methodology applied in order to extract the contour of each element on the P&ID, creats indeed some reference point, but identifies many lines for each object, meaning that each object isn't defined as one whole, but rather as an element consisting of many lines. This makes it hard to extract the most accurate representation of each element and leads to the workaround of expanding the minimum and maximum coordinates, as presented in Section 5.2.2. A way to overcome this would be to develop a more accurate algorithm that can indeed identify each element as one object, possibly by grouping line segments that share common points to one object.

Chapter 8

Conclusions

In this project we deal for a less than 4 months period a very challenging application: the identification of the components of a P&ID diagram, that takes as input a piping and instrumentation diagram (P&ID), provides insight on its textual and visual content, is developed. The methodology currently performs very well on the identification and decoding of the text included in the P&ID. It is however still a challenge the identification and classification of the components lying in it. The solid methodology developed in this project can be enriched and improved in order to address the topic of classification of the different components of the P&ID, following e.g., the directions outlined in Chapter 7.

Finally, it is important to notice that the proposed methodology and scope can be applied in other types of diagrams and drawing, such as those encountered in civil engineering, architecture etc. This could be an interesting aspect and approach, for fields that are not yet that linked to the world of Information Technology in regard to Machine Learning.

Bibliography

[1] S. Theodoridis. *Machine Learning*. 2020.

[2] Rohit Rahul, Shubham Paliwal, Monika Sharma, and Lovekesh Vig. *Automatic Information Extraction from Piping and Instrumentation Diagrams*. 2019. arXiv: 1901.11383 `[cs.CV]`.

[3] Eyad Elyan, Carlos Francisco Moreno-García, and Pamela Johnston. “Symbols in Engineering Drawings (SiED): An Imbalanced Dataset Benchmarked by Convolutional Neural Networks”. In: *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference*. Ed. by Lazaros Iliadis, Plamen Parvanov Angelov, Chrisina Jayne, and Elias Pimenidis. Cham: Springer International Publishing, 2020, pp. 215–224. isbn: 978-3-030-48791-1.

[4] Shouvik Mani, Michael A. Haddad, Dan Constantini, Willy Douhard, Qiwei Li, and Louis Poirier. “Automatic Digitization of Engineering Diagrams using Deep Learning and Graph Search”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, pp. 673–679. doi: 10.1109/CVPRW50498.2020.00096.

[5] Carlos Moreno-García, Eyad Elyan, and Chrisina Jayne. “New trends on digitisation of complex engineering drawings”. In: *Neural Computing and Applications* 31 (June 2019). doi: 10.1007/s00521-018-3583-1.

[6] Eyad Elyan, Laura Jamieson, and Adamu Ali-Gombe. “Deep learning for symbols detection and classification in engineering drawings”. In: *Neural Networks* 129 (2020), pp. 91–102. issn: 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.05.025. url: https://www.sciencedirect.com/science/article/pii/S0893608020301957.

Appendix A

Algorithm

```python

#!/usr/bin/env python3

##################################
##################################
######## P&ID digitization #######
##################################
##################################

##################################
########    Libraries   ##########
##################################
import cv2
import numpy as np
import pytesseract
import matplotlib.pyplot as plt
import imutils
import re
import sys
import cv2 as cv
from tqdm import tqdm
from skimage import morphology
from skimage.morphology import skeletonize

import numpy as np # linear algebra
import pandas as pd #CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #for plotting
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
import seaborn as sns
from random import randint
# imports for array-handling and plotting
import matplotlib
matplotlib.use('agg')
import os
import tensorflow as tf
from keras.models import Sequential, load_model
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import keras.applications

from subprocess import check_output
# hide warnings
import warnings
warnings.filterwarnings('ignore')

# solve cudNN initialisation
from tensorflow.compat.v1 import InteractiveSession
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)

# spit circles and crop images
import cv2
import numpy as np
import pytesseract
import matplotlib.pyplot as plt
import imutils


import tensorflow as tf
import tensorflow

```

```
from tensorflow import keras
from keras.layers import Dense

from tensorflow.keras.layers import Input, Dense

import tensorflow.keras as keras

import keras.layers

from keras.models import Sequential
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt


path = str(sys.argv[1]) # realpars_p_id_diag_mod.png, /home/jovyan/mezcla/Documents/Data_Science/capstone/pid-digitization/realpars_p_id_diag_mod.png

path2 = str(sys.argv[2]) # output1.jpg, /home/jovyan/mezcla/Documents/Data_Science/capstone/pid-digitization/output1.jpg

path3 = str(sys.argv[3]) # folder/to/images, '/home/jovyan/mezcla/Documents/Data_Science/capstone/pid-digitization/data/Symbol_pixels_blog_projectmaterials'

###################################
############# Step 1 #############
####### Contour detection ########
###################################

print("\n Countour detection started.")

```

```
def prepare_image(path):
    from skimage import color
    from skimage import io
    from skimage.color import rgb2gray
    # import image
    img = io.imread(path)
    imgGray = color.rgb2gray(img)
    return imgGray

# find contours
from skimage import measure
import matplotlib.pyplot as plt
def find_contours(r):
    # Find contours at a constant value of 0.8
    contours = measure.find_contours(r, 0.8)

    # Display the image and plot all contours found
    fig, ax = plt.subplots()
    ax.imshow(r, cmap=plt.cm.gray)

    for contour in contours:
        ax.plot(contour[:, 1], contour[:, 0], linewidth=2)
    ax.axis('image')
    ax.set_xticks([])
    ax.set_yticks([])
    plt.show()
    return contours

def execute_contours(url):
    img = prepare_image(url)
    contours = find_contours(img)
    return img, contours

img, contours = execute_contours(path)

print("\n Countour detection finished.")

#################################
############# Step 2 #############
######### Extraction of  #########
```

```python
######## PIDs elements ###########
###################################

## spot min, max coordinates

coord_cont = []
X_min = []
Y_min = []
X_max = []
Y_max = []
for contour in range(len(contours)):
    coord_cont.append(contour)
    X_min.append(math.floor(contours[contour][:,0].min()))
    X_max.append(math.ceil(contours[contour][:,0].max()))
    Y_min.append(math.floor(contours[contour][:,1].min()))
    Y_max.append(math.ceil(contours[contour][:,1].max()))

## crop elements along the PID diagram

from skimage import color
elements = [] # each element of the PID  diagram
for i in coord_cont:
    image_cropped = img[X_min[i]:X_max[i],
                       Y_min[i]:Y_max[i]]
    resized = cv2.resize(image_cropped,
                         (100, 100),
                         interpolation = cv2.INTER_AREA)
    elements.append(resized)

elements_50_100 = [] # each element of the PID  diagram
for i in coord_cont:
    if X_min[i] > 50 and Y_min[i] > 50 and X_max[i]+100 < img.shape[0] and Y_max[i]+100 < img.shape[1]:
        image_cropped = img[X_min[i]-50:X_max[i]+100,
                           Y_min[i]-50:Y_max[i]+100]
        resized = cv2.resize(image_cropped,
                             (100, 100),
                             interpolation = cv2.INTER_AREA)
        elements_50_100.append(resized)

```

```
elements_250_250 = [] # each element of the PID  diagram
for i in coord_cont:
    if X_min[i] > 250 and Y_min[i] > 250 and X_max[i]+250 < img.shape[0] and Y_max[i]+250 < img.shape[1]:
        image_cropped = img[X_min[i]-250:X_max[i]+250,
                       Y_min[i]-250:Y_max[i]+250]
        resized = cv2.resize(image_cropped,
                             (100, 100),
                             interpolation = cv2.INTER_AREA)
        elements_250_250.append(resized)


##################################
############# Step 3 #############
##### OCR text information #######
##################################

# import path2 image

# try to add skeletonize

import cv2
import cv2 as cv
import numpy as np
import pytesseract
import matplotlib.pyplot as plt
import imutils
import re
from tqdm import tqdm
from skimage import morphology
from skimage.morphology import skeletonize

def load_image(url):
    img_path = url
    img = cv2.imread(img_path, 1)
    return img

def preprocessing(img, resized_width): # try for resized_width = 1400
    print("\n Executing preprocessing()")
    # store height, width,
    # depth and ratio
    h, w, d = img.shape
#     resized_w = 1400
    ratio = resized_width / w

    # resize image
    resized_image = imutils.resize(img, width=resized_width)

    # tranform to grayscale
    gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)

    # blur image
    blur_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

    return resized_image, gray_image, blur_image, ratio


def find_circles(img, blur_image, resized_image, minRadius, maxRadius):
    print("\n Executing find_circles()")
    # identify circles
    circles = cv2.HoughCircles(
        blur_image,
        method=cv2.HOUGH_GRADIENT,
        dp=0.1,
        minDist=10,
        param1=20,
        param2=10,
        minRadius = minRadius,
        maxRadius= maxRadius
#         minRadius=11,
#         maxRadius=12,
    )

    # smooth to integer
    circles = np.uint16(np.around(circles))
    print(str(circles.shape[1]) + " circles found on the pi&d")

    count = 1
    img_circles = resized_image.copy()
    for circle in circles[0]:
        # Annotate circle and centroid
        cv2.circle(img_circles, (circle[0], circle[1]), circle[2], (0, 255, 0)
    , 2)
        cv2.circle(img_circles, (circle[0], circle[1]), 2, (255, 0, 0), -2)

        # Annotate text
        offset_txt = int(circle[2] * 1.2)
        cv2.putText(
            img_circles,
            "Circle " + str(count),
            (circle[0] - offset_txt, circle[1] + offset_txt),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.3,
            (255, 0, 0),
            1,
        )
        count += 1

    plt.rcParams['figure.figsize'] = (16,9)
    plt.imshow(img_circles)
    plt.imsave(fname='img_output_circles.jpg',
               arr=img_circles)

    return circles

def ocr_text(img, circles, ratio):
    print("\n Executing ocr_text()")
    # Read information in every circle
    cropped_imgs = []
    cropped_imgs_txt = []
    img_circle_txt = img.copy()

    circles_int = (circles[0] // ratio).astype(int)

    for circle in circles_int:
        up = ""
        low = ""

        x_offset_right = np.uint16(circle[2] * 0.79)
        x_offset_left = np.uint16(circle[2] * 0.65)
        y_offset_low = np.uint16(circle[2] * 0.72)
        y_offset_up = np.uint16(circle[2] * 0.56)


        cropped_img_lower = img_circle_txt[
            circle[1] : circle[1] + y_offset_low,
            circle[0] - x_offset_left : circle[0] + x_offset_right,
        ]
        cropped_img_upper = img_circle_txt[
            circle[1] - y_offset_up : circle[1],
            circle[0] - x_offset_left : circle[0] + x_offset_right,
        ]
        cropped_imgs.append(np.append(cropped_img_upper, cropped_img_lower, axis=0))

        upper = pytesseract.image_to_string(
            cropped_img_upper,
            lang="eng",
            config="--psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ",)
        lower = pytesseract.image_to_string(
            cropped_img_lower,
            config="--psm 7 -c tessedit_char_whitelist=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789",)

        r_upper = re.compile(r"[A-Z]+")
        r_lower = re.compile(r"[A-Z0-9]+")

        if not upper.isspace():
            if re.match(r_upper, str(upper)).group(0) is not None:
                up = re.match(r_upper, str(upper)).group(0)
        if not lower.isspace():
            if re.match(r_lower, str(lower)).group(0) is not None:
                low = re.match(r_lower, str(lower)).group(0)

        cropped_imgs_txt.append(up + "-" + low)

    return cropped_imgs, cropped_imgs_txt, img_circle_txt, circles_int


```

```
def read_text_in_circles(url, resized_width, minRadius, maxRadius):
    print("\n Executing read_text_in_circles()")
    img = load_image(url)
    resized_image, gray_image, blur_image, ratio = preprocessing(img,
   resized_width)
    circles = find_circles(img, blur_image, resized_image, minRadius,maxRadius
   )
    cropped_imgs, cropped_imgs_txt, img_circle_txt, circles_int = ocr_text(img
   , circles, ratio)

    return circles, img_circle_txt, cropped_imgs, cropped_imgs_txt

print("\n Reading text in circles started.")

circles, img_circle_txt, cropped_imgs, cropped_imgs_txt = read_text_in_circles
   (path2, 1400, 12, 14)

print("\n Reading text in circles finished.")

##################################
### Deciphering the P&IDs text ##
##################################

##################################
######## Dictionairies ###########
##################################

# instrument letter identification
# based on ANSI/ISA-5.1.-1984 (R1992)
# https://www.aiche.org/sites/default/files/ChEnected-Example-PIDs-and-Lead-
   Sheets.pdf

variables = {"A":"ANALYZER",
             "B":"BURNER",
             "C":"USER'S CHOICE",
             "D":"USER'S CHOICE",
             "E":"VOLTAGE",
             "F":"FLOW",
             "G":"USER'S CHOICE",
             "H":"HAND",
             "I":"CURRENT",
             "J":"POWER",
             "K":"TIME",
             "L":"LEVEL",
             "M":"USER'S CHOICE",
             "N":"USER'S CHOICE",
             "O":"USER'S CHOICE",
             "P":"PRESSURE",
             "Q":"QUANTITY",
             "R":"RADIATION",
             "S":"SPEED",
             "T":"TEMPERATURE",
             "U":"MULTI-VARIABLE",
             "V":"VIBRATION",
             "W":"WEIGHT, FORCE",
             "X":"UNCLASSIFIED",
             "Y":"EVENT, STATE",
             "Z":"POSITION"
            }

modifiers = {
            "C":"CONTROL",
            "D":"DIFFERENTIAL",
            "F":"RATIO",
            "J":"SCAN",
            "M":"MOMENTARY",
            "Q":"INTERGRATE/TOTALIZE",
            "R":"RELIEF",
            "S":"SAFETY",
            "X":"X-AXIS",
            "Y":"Y-AXIS",
            "Z":"Z-AXIS"
            }

indication_functionalities = {"A":"ALARM",
                              "B":"USER'S CHOICE",
                              "E":"PRIMARY ENTITY",
                              "G":"GLASS",
                              "I":"INDICATE",
                              "L":"LIGHT",
                              "N":"USER'S CHOICE",
                              "O":"OFFICE",
                              "P":"POINT TEST CONN.",
                              "R":"RECORD",
                              "U":"MULTI-FUNCTION",
                              "W":"WELL",
                              "X":"UNCLASSIFIED"
                              }

output_functionalities =     {"B":"USER'S CHOICE",
                              "C":"CONTROL",
                              "K":"CONTROL STATION",
                              "N":"USER'S CHOICE",
                              "S":"SWITCH",
                              "T":"TRANSMIT",
                              "U":"MULTI-FUNCTION",
                              "V":"VALVE, DAMPER",
                              "X":"UNCLASSIFIED",
                              "Y":"RELAY, COMPUTE",
                              "Z":"DRIVER, ACTUATOR UNCLASSIFIED FINAL CONTROL ELEMENT",
                              }

modifier_of_functionalities = {"B":"USER'S CHOICE",
                               "C":"CLOSE",
                               "H":"HIGH",
                               "L":"LOW",
                               "M":"MEDIUM",
                               "N":"USER'S CHOICE",
                               "O":"OPEN",
                               "U":"MULTI-FUNCTION",
                               "X":"UNCLASSIFIED"
                              }

# transducer functions
# based on ANSI/ISA-5.1.-1984 (R1992)
# https://www.aiche.org/sites/default/files/ChEnected-Example-PIDs-and-Lead-Sheets.pdf

transducer_functions = {"E/E":"VOLTAGE TO VOLTAGE",
                        "E/I":"VOLTAGE TO CURRENT",
                        "E/P":"VOLTAGE TO PNEUMATIC",
                        "I/P":"CURRENT TO PNEUMATIC",
                        "P/I":"PNEUMATIC TO CURRENT"
                       }

# fluid service codes
# based on ANSI/ISA-5.1.-1984 (R1992)
# https://www.aiche.org/sites/default/files/ChEnected-Example-PIDs-and-Lead-Sheets.pdf

fluid_service_codes = {"ALM":"ALUMINUM SULFATE",
                       "AMN":"AMMONIUM NITRATE",
                       "AMH":"AMMONIUM HYDROXIDE",
                       "ABF":"AMMONIUM (BI)FLUORIDE",
                       "AMS":"AMMONIUM SULFATE",
                       "ASO":"ACID SOLUBLE ORGANICS",
                       "BAR":"BACKWASH AIR",
                       "CAF":"CALCIUM FLUORIDE",
                       "CAR":"COMPRESSED AIR",
                       "CBW":"CLEAN BACKWASH WATER",
                       "CFD":"CAUSTIC RAW FEED (GEN. USE)",
                       "CO2":"CARBON DIOXIDE",
                       "CHC":"CALCIUM HYPOCHLORITE",
                       "CL2":"CHLORINE",
                       "DBW":"DIRTH BACKWASH WATER",
                       "DRN":"PROCESS DRAIN",
                       "DSL":"DIESEL FUEL",
                       "EFF":"EFFLUENT (GENERAL USE)",
                       "FEC":"FERRIC CHLORIDE",
                       "FEW":"FILTER EFFLUENT WATER",
                       "FIW":"FILTER INFLUENT WATER",
                       "FOL":"FUEL OIL",
                       "HCL":"HYDROCHLORIC ACID",
                       "HF":"HYDROFLUORIC ACID",
                       "HPX":"HYDROGEN PEROXIDE",
                       "IAR":"INSTRUMENT AIR",
                       "IFD":"INDUSTRIAL RAW FEED",
                       "LSY":"LIME SLURRY",
                       "MEL":"METHANOLS",
                       "NAG":"NATURAL GAS",
                       "NIA":"NITRIC ACID",
                       "N2":"NITROGEN",
                       "OIL":"OIL (GENERAL USE)",
                       "PAR":"PROCESS AIR",
                       "PFD":"POLYMER FEED",
                       "PHA":"PHOSPHORIC ACID",
                       "KF":"POTASSIUM FLUORIDE",
                       "KOH":"POTASSIUM HYDROXIDE",
                       "PSL":"PROCESS SLURRY/SLUDGE",
                       "PVP":"PROCESS VAPOR",
                       "PWR":"POTABLE WATER",
                       "SAH":"SULFURIC ACID, >75%",
                       "SAL":"SULFURIC ACID, <75%",
                       "SHC":"SODIUM HYPOCHLORITE",
                       "SOC":"SODIUM CARBONATE",
                       "SOH":"SODIUM HYDROXIDE",
                       "SLP":"STEAM, <125#",
                       "SMB":"SODIUM METABISULFITE",
                       "STM":"STEAM, 125-220#",
                       "SNY":"SANITARY SEWER",
                       "STO":"STORM DRAIN",
                       "SWR":"SERVICE WATER",
                       "TFL":"THERMAL FLUID",
                       "UAR":"UTILITY AIR",
                       "UWR":"UTILITY WATER",
                       "VNT":"VENT (GENERAL USE)",
                       "WOL":"WASTE OIL",
                       "WWR":"WASTEWATER (GENERAL USE)"
                       }

####################################
# Function to decipher textual tag #
####################################

# how to interpret the letter in the circles
# which refer to physical devices
# https://www.aiche.org/chenected/2010/09/interpreting-piping-and-instrumentation-diagrams-symbology

```

```python
# how to interpret the letter in the circles
# which refer to physical devices
# https://www.aiche.org/chenected/2010/09/interpreting-piping-and-instrumentation-diagrams-symbology

def decipher_tag(text): # input is a string
    text = text.upper()
    variable = ''
    modifier = ''
    indication_functionality = ''
    output_functionality = ''
    modifier_of_functionality = ''
    tags = []

    if len(text) == 2:
        if text[0] in variables.keys():
            variable = text[0]
#             print("First letter means: ", variables[variable])
            tags.append(variables[variable])

        if text[1] in indication_functionalities.keys():
            indication_functionality = text[1]
#             print("Second letter means: ", indication_functionalities[indication_functionality])
            tags.append(indication_functionalities[indication_functionality])

        if text[1] in output_functionalities.keys():
            output_functionality = text[1]
#             print("Second letter means: ", output_functionalities[output_functionality])
            tags.append(output_functionalities[output_functionality])

    if len(text) == 3:
        if text[0] in variables.keys():
            variable = text[0]
#             print("First letter means: ", variables[variable])
            tags.append(variables[variable])

        if text[1] in indication_functionalities.keys():
            indication_functionality = text[1]
```

```
#             print("Second letter means: ", indication_functionalities[indication_functionality])
            tags.append(indication_functionalities[indication_functionality])

        if text[2] in output_functionalities.keys():
            output_functionality = text[2]
#             print("Third letter means: ", output_functionalities[output_functionality])
            tags.append(output_functionalities[output_functionality])

    if len(text) == 4:
        if text[0] in variables.keys():
            variable = text[0]
#             print("First letter means: ", variables[variable])
            tags.append(variables[variable])

        if text[1] in modifiers.keys():
            modifier = text[1]
#             print("Second letter means: ", modifiers[modifier])
            tags.append(modifiers[modifier])

        if text[2] in indication_functionalities.keys():
            indication_functionality = text[2]
#             print("Third letter means: ", indication_functionalities[indication_functionality])
            tags.append(indication_functionalities[indication_functionality])

        if text[3] in output_functionalities.keys():
            output_functionality = text[3]
#             print("Fourth letter means: ", output_functionalities[output_functionality])
            tags.append(output_functionalities[output_functionality])

    if len(text) == 5:
        if text[0] in variables.keys():
            variable = text[0]
#             print("First letter means: ", variables[variable])
            tags.append(variables[variable])

        if text[1] in modifiers.keys():
            modifier = text[1]
#             print("Second letter means: ", modifiers[modifier])
            tags.append(modifiers[modifier])

        if text[2] in indication_functionalities.keys():
            indication_functionality = text[2]
#             print("Third letter means: ", indication_functionalities[indication_functionality])
            tags.append(indication_functionalities[indication_functionality])

        if text[3] in output_functionalities.keys():
            output_functionality = text[3]
#             print("Fourth letter means: ", output_functionalities[output_functionality])
            tags.append(output_functionalities[output_functionality])

        if text[4] in modifier_of_functionalities.keys():
            modifier_of_functionality = text[4]
#             print("Fifth letter means: ", modifier_of_functionalities[modifier_of_functionality])
            tags.append(modifier_of_functionalities[modifier_of_functionality])

#     print("Text tag is: ",variable,
#           modifier,
#           indication_functionality,
#           output_functionality,
#           modifier_of_functionality)

    return tags

print("\n Deciphering text in circles started.")

cropped_imgs_txt_deciphered = []
for text in cropped_imgs_txt:
    tag = text.split("-")[0]
    tags = decipher_tag(tag)
    cropped_imgs_txt_deciphered.append(decipher_tag(tag) )
print(cropped_imgs_txt_deciphered)
print("\n Deciphering text in circles finished.")

##################################
############# Step 4 #############
## Convolutional Neural Network ##
##################################

# import data

print("\n Import dataset from directory started.")
data_ds = tf.keras.preprocessing.image_dataset_from_directory(path3,
                                                           color_mode="grayscale",
                                                           image_size=(100,100),
                                                           batch_size=32,
                                                           labels='inferred',
                                                           label_mode='categorical')
type(data_ds)

print("\n Import dataset from directory finished.")

# create dict with labels

labels_names = {}
i = 1
for item in data_ds.class_names:
    labels_names[item] = i
    labels_names[i] = item
    i += 1

labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]]


# split train val test
print("\n Split train val test")
def split_train_val_test(ds,
                         train_split=0.7,
                         val_split=0.15,
                         test_split=0.15,
                         shuffle=True,
                         shuffle_size=10000):

    assert (train_split + test_split + val_split) == 1

    ds_size = 0
    for data, labels in ds:
        ds_size += 1

    if shuffle:
        # Specify seed to always have the same split distribution between runs
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    print("train_size=", train_size)
    val_size = int(val_split * ds_size)
    print("val_size=", val_size)
    print("test_size=", (ds_size - val_size - train_size))

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds

train_ds, val_ds, test_ds  = split_train_val_test(data_ds)
type(train_ds), type(val_ds), type(test_ds)

print("\n Compile and make the CNN model.")
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    print("inputs.shape", inputs.shape)
    x = inputs
    print("1x_data_augmentation.shape", x.shape)
    # Entry block
    #x = layers.experimental.preprocessing.Rescaling(1.0 / 255)(x)
    x = layers.Conv2D(filters=32,
                     kernel_size=(3,3),
                     activation='relu',
                     padding='same')(x)
    print("2x_Conv2D.shape", x.shape)
    x = layers.MaxPooling2D(pool_size=(2,2))(x)
    print("3x_MaxPooling2D.shape", x.shape)
    x = layers.Conv2D(filters=64,
                     kernel_size=(3,3),
                     activation='relu',
                     padding='same')(x)
    x = layers.Conv2D(filters=64,
                     kernel_size=(3,3),
                     activation='relu',
                     padding='same')(x)
    x = layers.MaxPooling2D(pool_size=(2,2))(x)

    x = layers.Conv2D(filters=128,
                     kernel_size=(3,3),
                     activation='relu',
                     padding='same')(x)
    x = layers.Conv2D(filters=128,
                     kernel_size=(3,3),
                     activation='relu',
                     padding='same')(x)
    x = layers.MaxPooling2D(pool_size=(2,2))(x)
    x = layers.Flatten()(x)
    # Densely connected layers
    print("x.shape", x.shape)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(0.5)(x)
    x = layers.Dense(64, activation='relu')(x)
    x = layers.Dropout(0.1)(x)

    # output layer
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model((100, 100, 1), 53)
# model = make_model((100, 100,), 53)
print(model.summary())

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
    # define the early stopping
    keras.callbacks.EarlyStopping(monitor='val_loss',
                                              patience=5)
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)

history = model.fit_generator(
           train_ds,
           epochs=50,
           callbacks=callbacks,
           validation_data=val_ds,
       )

print("Plot learning curves for training.")
plt.figure(figsize=(12, 6))


# print scores
def print_scores(history):
    print(history.history.keys())
    print(max(list(history.history.values())[0]), \
          max(list(history.history.values())[1]), \
          max(list(history.history.values())[2]), \
          max(list(history.history.values())[3]))
print_scores(history)


```

```
# plot curves
def plots_curves(history):
    plt.figure(figsize=(12, 6))
    # summarize history for accuracy
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.savefig("accuracy_training100.png")
    plt.show()
    # summarize history for loss
    plt.figure(figsize=(12, 6))
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.savefig("loss_training100.png")
    plt.show()
plots_curves(history)

# plot architecture of the model
from keras.utils.vis_utils import plot_model
plot_model(model, show_shapes=True, dpi=100)
plt.savefig("plot_model.png")

# evaluate the model

evaluation = model.evaluate(test_ds,
                batch_size=32,
                verbose=1,
                sample_weight=None,
                steps=None,
                callbacks=callbacks,
                max_queue_size=10,
                workers=1,
                use_multiprocessing=False,
```

```python
                return_dict=True)

print("Evaluation of the model:", evaluation)

# predict on the test set_xticks

predictions_test = model.predict(test_ds,
                batch_size=32,
                verbose=1,
                steps=None,
                callbacks=None,
                max_queue_size=10,
                workers=1,
                use_multiprocessing=False)

# print confusion  matrix on test set

predictions = np.array([])
labels =  np.array([])
for x, y in test_ds:
    predictions = np.concatenate([predictions,  np.argmax(model.predict(x), axis = -1)])
    labels = np.concatenate([labels, np.argmax(y.numpy(), axis=-1)])
cm = tf.math.confusion_matrix(labels=labels, predictions=predictions).numpy()

from sklearn.metrics import confusion_matrix
import pylab as pl
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (8,8)

pl.matshow(cm, cmap='Greys')
pl.title('Confusion matrix of the CNN model on test set')
pl.colorbar()
pl.show()

##################################
############# Step 5 #############
##### Predict on PID elements ####
##################################

```

```python
index_pred = []
probab_pred = []
for image in range(len(elements)):
    plt.imshow(elements[image], cmap='gray')
    plt.show()

    img_array = keras.preprocessing.image.img_to_array(elements[image])
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis
    test_predic = model.predict(img_array)
    print("The element belongs to class %d,%s with a probablity of %.2f %%." % (np.where(test_predic[0] == test_predic[0].max())[0][0],labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]], round(test_predic[0].max()*100,2)))

    index_pred.append(np.where(test_predic[0] == test_predic[0].max())[0][0])
    probab_pred.append(test_predic[0].max())

index_pred_50_100 = []
probab_pred_50_100 = []

for image in range(len(elements_50_100)):
    plt.imshow(elements_50_100[image], cmap='gray')
    plt.show()

    img_array = keras.preprocessing.image.img_to_array(elements_50_100[image])
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis
    test_predic = model.predict(img_array)
    print("The element belongs to class %d,%s with a probablity of %.2f %%." % (np.where(test_predic[0] == test_predic[0].max())[0][0],labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]], round(test_predic[0].max()*100,2)))


    index_pred_50_100.append(np.where(test_predic[0] == test_predic[0].max())[0][0])
    probab_pred_50_100.append(test_predic[0].max())

index_pred_250_250 = []
probab_pred_250_250 = []

```

```python
for image in range(len(elements_250_250)):
    plt.imshow(elements_250_250[image], cmap='gray')
    plt.show()

    img_array = keras.preprocessing.image.img_to_array(elements_250_250[image])
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis
    test_predic = model.predict(img_array)
    print("The element belongs to class %d,%s with a probablity of %.2f %%." % (np.where(test_predic[0] == test_predic[0].max())[0][0],labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]], round(test_predic[0].max()*100,2)))

    index_pred_250_250.append(np.where(test_predic[0] == test_predic[0].max())[0][0])
    probab_pred_250_250.append(test_predic[0].max())


# check examples from PIDs
# 1
# extract element

rotary_compressor = img[850:1150, 75:400]
rotary_compressor = cv2.resize(rotary_compressor,
                               (100, 100),
                               interpolation = cv2.INTER_AREA)
plt.imshow(rotary_compressor)

# pass it through the model

img_array = keras.preprocessing.image.img_to_array(rotary_compressor)
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

test_predic = model.predict(img_array)
np.where(test_predic[0] == test_predic[0].max())[0][0], test_predic[0].max(), labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]]
print("The element belongs to class %d,%s with a probablity of %.2f %%." % (np.where(test_predic[0] == test_predic[0].max())[0][0],labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]], round(test_predic[0].max()*100,2)))
```

```

def plot_histogram(predictions):
    plt.figure(figsize=(10,10))
    plt.hist(predictions, bins=53)
    plt.show()
plot_histogram(test_predic[0])


# 2
# pass it through the model

img_array = keras.preprocessing.image.img_to_array(elements[2])
img_array = tf.expand_dims(img_array, 0)  # Create batch axis

test_predic = model.predict(img_array)
np.where(test_predic[0] == test_predic[0].max())[0][0], test_predic[0].max()

print("The element belongs to class %d,%s with a probablity of %.2f %%." % (np.where(test_predic[0] == test_predic[0].max())[0][0]-1,labels_names[np.where(test_predic[0] == test_predic[0].max())[0][0]], round(test_predic[0].max()*100,2)))
```

Listing A.1: Algorithm of the application