Specifying Key-properties to Improve the Recognition Skills of Neural Networks

Jahić Benjamin[1], Guelfi Nicolas[2], Ries Benoît[3]
University of Luxembourg
2, avenue de l'Université
L-4365, Esch-sur-Alzette
(+352) 46 66 44 6030
[1]benjamin.jahic@uni.lu, [2]nicolas.guelfi@uni.lu, [3]benoit.ries@uni.lu

ABSTRACT
Software engineers are increasingly asked to build datasets for engineering neural network-based software systems. These datasets are used to train neural networks to recognise data. Traditionally, data scientists build datasets consisting of random collected or generated data. Their approaches are often costly, inefficient and time-consuming. Software engineers rely on these traditional approaches that do not support precise data selection criteria based on customer's requirements. We introduced a software engineering method for dataset augmentation to improve neural networks by satisfying the customer's requirements. In this paper, we introduce the notion of key-properties to describe the neural network's recognition skills. Key-properties are used all along the engineering process for developing the neural network in cooperation with the customer. We propose a rigorous process for augmenting datasets based on the analysis and specification of the key-properties. We conducted an experimentation on a case study on the recognition of the state of a digital meter counter. We demonstrate an informal specification of the neural network's key-properties and a successful improvement of a neural network's recognition of the meter counter state.

CCS Concepts
• Software and its engineering → Software creation and management → Software development process management → Software development methods → Rapid application development.

Keywords
Software engineering; methods; neural networks; specification; key-properties; dataset augmentation.

1. INTRODUCTION
Deep Learning [1] focuses on approaches for engineering computer programs that simulate the behaviour of a simplified human brain. These programs, called neural networks, are in high demand in various domains such as autonomous vehicles, commercial, finance, etc. Large datasets are required to train these neural networks to recognise some data (e.g. recognising pedestrian, meter counter states, prices…). Data scientists are building these datasets by following traditional engineering approaches, which are mostly hand-crafted and empirical in order to collect, classify and split the data to obtain different types of datasets (e.g. training, testing and development dataset).

Due to the growing demand of neural networks in various domains, software engineers and data scientists face major challenges to build these datasets for engineering improved neural networks. Software engineers rely on traditional dataset engineering approaches, because there exists no software engineering methodology that supports dataset engineering based on customer's requirements. Traditional dataset engineering approaches do not support precise data selection criteria based on customer's requirements. Additionally, data scientists are often not trained to use software engineering methods. However, as it is important to engineer a neural network that satisfies the customer's needs, thus, there is a need for a software engineering methodology to support the efficient creation of a dataset to engineer a neural network based on the customer's requirements.

In [2], we introduced a first version of a software engineering methodology for dataset engineering based on customer's requirements. We presented a rigorous process for iteratively augmenting datasets with generated synthetic data to improve neural networks. We presented the relevant activities and techniques to support the efficient creation of datasets based on customer's requirement. At each process iteration, the results of the neural network's tests, called test monitoring data, are analysed to extract an informal list of potential dataset improvements. In general, these improvements are presented to a customer for being validated or they serve to specify the dataset augmentation for improving a neural network to satisfy the customer's requirements. However, analysing the test monitoring data and listing the dataset improvements is a very challenging task. We categorised the test data depending on the correctness of the classification (e.g. data labels) or the recognition (e.g. predicted data labels). The data categories are used to extract relevant information for listing some dataset improvement. The process was very simplistic and does not guide the engineer to produce effective dataset improvement for satisfying the customer's need. These dataset improvements are often very technical, not-well structured, time-consuming and complicated without precise data selection criteria. It becomes even more challenging to discuss these improvements with a customer without precise structure and adapted technical terminologies.

In this paper, we extend our iterative rigorous process to contribute to the issue of imprecise and complicated specification of dataset improvements. We introduce a rigorous process to support the engineers for analysing the test monitoring data. We introduce the notion of neural network's key-properties (KP) to describe its recognition skills and its improvements. The key-properties serve to define a list of neural network's strengths and weaknesses for a customer validation

process. Finally, they are used to specify a dataset augmentation [3] to improve the neural network.

In Section 2, we present briefly the related work. Section 3 presents our software engineering method formalized as a business process. Section 4 presents an experimentation of our approach conducted on our academic meter counter recognition case study. In Section 5, we highlight and discuss some important aspects of our process and propose some potential future work.

2. RELATED WORK

In this section, we present some recent studies related to our work. We focus on related works around software engineering methodological issues, and when possible on the particular phase of requirements engineering. We do not focus on the numerous machine learning works on the optimisation of the design of neural networks.

Laroca et al. [4], Nodari et al. [5] and Vanetti et al. [6] present different deep learning-based software systems for recognizing the state of a meter counters. Laroca et al. introduce a new dataset, called UFPR-AMR, of real-world images of meter counters for training and testing their neural networks. In their paper, they claim that it is difficult to recognize meter counter states of noisy (dirty, dark, bright) images. They use several techniques to generate randomly additional images by adjusting the brightness, adding noise, rotating images or segmenting images. The generated synthetic images are added to the initial datasets for training their new neural network architectures. Thus, they were able to improve the recognition of the meter counter states. We agree that the dataset augmentation can improve the recognition skills of a neural network. However, they do not follow a software engineering method to develop their neural networks based on requirements. We propose the usage of a software engineering method to engineering improved neural networks that satisfy the requirements of some customer. Our software engineering method includes precise data selection for augmenting dataset based on customer's requirements. The augmented dataset is used to retrain and improve the neural network to satisfy the customer's requirements.

Vogelsang and Borg [7]; Kostova et al. [8] present some advances on requirements engineering for machine learning-based software systems (e.g. deep learning-based systems). They claim that more research must be investigated to understand the need of requirements engineering for machine learning. In [7], they claim that requirements engineering must evolve in the machine learning domain. They interviewed several data scientist to answer questions about their background (e.g. domain, involved projects…), their concrete usage of requirements in their projects and their requirements engineering approaches. Based on the answers, they summarise the characteristics for different requirements engineering activities (e.g. specification, analysis, verification, validation…) in the machine learning engineering approaches. They claim that data scientists usually improve their machine learning-based systems by analysing technical concepts (e.g. accuracy, loss…). These concepts are often not understood by customers, who should validate the system. They claim that there is a need for requirements engineering methods to map these machine learning concepts to the customer's requirements. We agree that there is a need for methods to support the engineering of machine-learning based systems, such as neural network, based on customer's requirements. We think that software engineers require methods and tools to facilitate the development of machine learning system that satisfy the customer's requirements.

Amershi et al. [9] present a workflow for engineering their machine learning-based software systems used at Microsoft. Their workflow consists of several stages such as data-oriented (e.g. data collection, cleaning and labelling), and stages for engineering machine-learning based software systems (e.g. training, evaluation, deployment…). They claim that the work of a machine learning engineer mostly consists in improving the machine learning model's architecture (e.g. neural network) by adjusting their parameters (e.g. number of layers, number of neurons, activation function…). They build their dataset by following traditional dataset engineering approaches, such as random data gathering, random data generation and removal of inaccurate data (noisy data). Our process presented in this paper differs in the sense that it is developed to support software engineers to select and generate precisely the data needed for training and testing a neural network that satisfies the customer's requirements.

Hesenius et al. [10] present a software engineering process for engineering machine learning-based software, called EDDA. EDDA consists of six phase that are connected to the phases of the software engineering lifecycle [11]. They define the actors (e.g. software engineer, data scientist, domain expert and data domain expert) responsible for executing the tasks of their engineering phases (e.g. data exploration, model requirements, model development…). The data exploration phase is an iterative process for analyzing existing data to define the goals of the application. Afterwards, they define the requirements and develop the machine learning model. During the development, they follow an iterative process of feature engineering [12] from the data, updating the machine learning model, and evaluating the model. Our process presented in this paper differs in the sense that we are focusing exclusively on deep learning-based systems. In deep learning-based systems, the layers of the neural network are describing the features of the data [13]. The features are not designed by humans and they are learnt from data. Thus, we iteratively improve the recognition skills of a neural network by augmenting our datasets based on the customer's needs. We argue that neural networks can efficiently improve by improving the datasets.

3. SOFTWARE ENGINEERING METHOD

We previously introduced a software engineering method [1] formalized as a business process using the BPMN 2.0 [14] modelling language. The main purpose of the initial process is to support engineers for the efficient development of neural network-based software systems. We consider designing appropriate datasets to train and test neural network based on customer's requirements. Thus, the trained neural network should recognise the data based on the needs expressed by a customer. The initial process consists of three different types of activities and three different types of data objects, listed below.

- Activity types
  - dataset engineering activities
  - neural network software engineering activities
  - neural network execution activities.
- Data objects types
  - Datasets

- Neural Networks
- Neural Networks Data

Concretely, the initial process is designed as a cyclic flow graph consisting of 9 main activities and 10 data objects. To give you a concrete overview of our initial process, we summarise our process using these groups of activities:

1. Engineering datasets and a neural network
2. Analysing the trained neural network
3. Defining a dataset augmentation
4. Generating synthetic data
5. Reengineering the dataset with the synthetic data
6. Return to 1 and reengineer the neural network

When engineering a neural network with respect to our process, we usually have to test the trained neural network at some step. Neural networks are tested on a set of testing data. The dataset should consist of precisely selected data in order to verify the neural network's recognition skills. The results of the neural network's tests, called test monitoring data, need to be analysed to understand the neural network's recognition skills. In the current process, the way to perform this analysis remains unclear and quite complicated, we gather some test data information by decomposing it into the following four categories:

1. Correctly Recognised data (CR)
2. Incorrectly Recognised data (IR)
3. Correctly Classified data (CC)
4. Incorrectly Classified data (IC)

Engineers are free to specify the recognition skills in their own way. This leads often to confusions and issues, when presenting the neural network to the customer. It may also lead to incomplete and inconsistent specification. Since the specification is used to define a dataset augmentation, we might need to iterate many times through the process to obtain the customer's validation. Moreover, these specifications are often written at a technical level. Customers often lack in technical knowledge, which makes it hard to discuss the recognition skills with them.

In this section, we present a new version of this software engineering method for augmenting datasets based on the specification of the neural network's key-properties. Our goal is to improve the process of analysing the test monitoring data to obtain clear customer's feedback and improve the specification of the dataset augmentation. Therefore, we present our new process consisting of some modified and refined activities. We focus mainly on the customer's requirements and analysis of the test monitoring data. We introduce the key-properties to describe the recognition skills of the neural network.

We present a new subprocess for analysing the test monitoring data to specify the neural network's recognition skills. We use the specification to improve the customer's validation process and the dataset augmentation specification. We designed the new process using the BPMN 2.0 modelling language. Our new process still consists of 9 main activities. We modified two activities, the analysis of the test monitoring data and the dataset augmentation specification. We refined the activity for analysing the test monitoring data by defining a precise sub-process consisting of 4 activities. We introduce an additional data object, called the key-properties specification.

In the upcoming section, we present in detail the modified activities. Figure 1 shows an overview of the modified activities of the business process introduced in this paper. The neural *network execution activities* are represented in *green* and the *dataset engineering activities* are represented in *blue*.

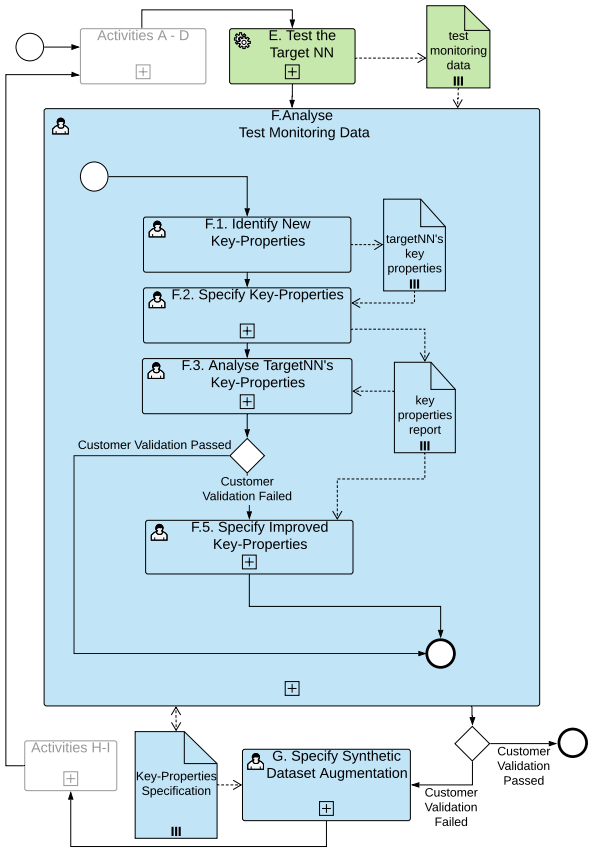

Figure 1. Business process for specifying the neural network's key-properties.

3.1 Data Input

In this section, we present the new process' data input. The process' Data input consists of a collection of classified data and a list of requirements to describe the customer's needs.

The collection of classified data remains unchanged except for a recommendation on the dataset construction that we would like to put emphasis on. The classified data is needed for training and testing the neural network. A classified data is any artefact that has been labelled with some relevant information (e.g. description of the content, integer…). The classified data serves to engineer some raw datasets for training and testing a target neural network. They may be collected or obtained from the customer. However, we recommend discussing with the customer to obtain a set of classified test data. This set should be used later in this process to test the targeted neural network. The data should be selected carefully in order to cover the most important test cases needed for validation of the neural network.

We add a list of requirements to the data input. The requirements are needed to define the initial customer's needs. This step is mandatory for being able to engineer a neural network that satisfies its requirements. Following software engineering best practices in requirements engineering, an engineer meets a customer to discuss the requirements of the

targeted neural network before executing our process. Based on the discussion, the engineer defines a list of initial requirements to be taken as input in our process.

3.2 Raw Datasets and targetNN Engineering

In this section, we present the dataset and neural network engineering initial activities of our process:

1. *Engineering raw datasets* consists in the creation of the equivalence classes and the required datasets. The equivalence classes are defined based on the classes used for classifying the input data. The datasets are called training, testing and development datasets. They are composed of a selection of input data. Usually, these datasets do not share any common data.
2. *Engineering the targetNN* consists in designing and implementing a neural network architecture (e.g. convolutional neural network). It includes choosing the appropriate layers, activation functions, loss function, …
3. *Train the targetNN* consists in training the implemented targetNN using the training and the development datasets. During the training, the neural network processes the training images and adjusts the weights on the neurons. This activity is executed multiple times until the accuracy and loss start to converge.
4. *Analysing the training monitoring data* focuses on the analysis of the results of the neural networks' training. The training monitoring data consists of the values of the accuracy and the loss during the training, the overall accuracy and loss after the training. The accuracy describes the amount of correctly recognised data. The loss is a measure to describe the precision of the recognised data. Usually, we analyse if the neural network is tending to over- or underfit [15].
5. *Test the targetNN* consists in verifying the targetNN. The neural network processes the images of the testing dataset. We verify the recognition skills of the neural network. We generate several diagrams, images grids and statistics to build the test monitoring data. The output of this activity is the test monitoring data.

3.3 Analyse the Test Monitoring Data

In this activity, we focus on analysing the test monitoring data. This activity has been modified and integrated in our new process. We present a new subprocess for analysing the test monitoring data, specifying the neural network's key-properties and validating the target neural network. The subprocess consists of four activities. We present in detail the activities of our process here.

3.3.1 Identify new key-properties

This activity's data input is the test monitoring data. During this activity, the engineer analyses the test monitoring to extract and specify the relevant information about the neural network's recognition skills. These recognition skills are described with the neural network's key-properties. In this activity, the engineer focuses on the identification of the key-properties by analysing the test monitoring data. As the instances of test monitoring data may vary depending on the executed neural network's tests, the engineers verify if the list of identified key-properties must be updated at each process iteration.

First, the engineer categorises the test monitoring data into two main categories, called the quantitative and qualitative data [16]. The engineer analyses the test monitoring data and determines in which category the data belongs to:

- **Quantitative data** is a subset of the test monitoring data used for identifying numerical key-properties.
- **Qualitative data** is a subset of the test monitoring data used for identifying key-properties represented in any textual or Boolean format.

In a second step, the engineer should analyse the categorised test monitoring data to identify the new key-property. The resulting key-properties are grouped into two main categories, the quantitative and qualitative properties. The engineer may subdivide the properties into further smaller subcategories to improve their structure. Table 1 shows an overview of our proposed key-properties categorisation.

- **Quantitative properties** are usually described as a numerical value. (e.g. dataset size, accuracy, loss, number of recognised data)
- **Qualitative properties** are usually represented as a textual or boolean expression.

We suggest categorising the key-properties into sub-categories to improve the structure. The two sub-categories for the quantitative properties are continuous and discrete.

- **Continuous properties** have their value belonging to a non-countable set. (e.g. accuracy, loss…)
- **Discrete properties** have their value belonging to a countable set such as dataset size, categorisations, number of correctly and incorrectly classified data...

We apply the same idea for the quantitative properties. We suggest categorising the qualitative key-properties in the categories of nominal, ordinal or logical properties.

- **Nominal properties** are characterized as textual representations. These properties are usually only named in textual format. (e.g. data description, image content, …)
- **Ordinal properties** are characterized as textual representation with a certain order. These properties are usually defined as a name, which is part of an ordered set. (e.g. categorical evaluation of data classification…)
- **Logical properties** are characterized typically as a boolean expression. These properties usually can only be true or false (e.g. correctness of data classification…)

Finally, the categorised key-property types are the activity's data output used in the next activity.

Table 1. Property categorisation

Key-properties categorisation				
Quantitative (QT)		Qualitative (QL)		
Continuous (C)	Discrete (D)	Nominal (N)	Ordinal (O)	Logical (B)

3.3.2 Specify key-properties

This activity's data input is the categorised key-property types. The engineer uses the key-property types to specify a list of key-properties of the trained neural network. At each iteration of the process, the engineer has to perform the following tasks:

1. Select a key-property type.
2. Select the most appropriate test monitoring data to define the key-property.
3. Analyse the selected test monitoring data with respect to the key-property type.
4. Specify the key-properties in different tables with respect to the categorisation of the key-property types.
5. Return to 1. and continue until all key-property types are covered.

The key-properties should be written in a similar format as the initial requirements in order to facilitate the verification of the satisfaction of the customer's requirements.

Finally, the key-property specification should be updated at each iteration of the process based on the identified key-property types. This allows us to track the evolution of the key-properties and to justify the satisfaction of the customer's requirements.

3.3.3 Analyse targetNN's key-properties

The engineer analyses the key-properties specified in the previous activity to define a list of strengths and weaknesses of the neural network. The strengths of the neural network should reflect and motivate the satisfaction of some customer's requirements. The weaknesses should show and validate the unsatisfied customer's requirements. Moreover, the engineer should propose some improvements in order to satisfy these requirements. The list of strengths, weaknesses and improvements should be less technical and written using a 'customer-friendly' vocabulary.

When the list of strengths, weaknesses and improvements is defined, it is then presented to the customer. The customer should reflect on the current version of the neural network. The engineer and the customer should discuss the strengths, weaknesses and improvements in order to clarify or propose some requirements. The goal is that the customer obtains an overview of his ordered product and participates to the evolution of the requirements. The customer may criticise some strengths, weaknesses or improvements. He should have the possibility to propose additional requirements, strengths, weaknesses or improvements. These proposals should be considered by the engineer during the further software construction.

Finally, the customer should validate the targetNN based on the presented information. Based on the customer's decision, we may stop the process or continue to improve the targetNN.

3.3.4 Specify improved key-properties

This activity is only run if the customer did not validate the targetNN. In this activity, the engineer analyses the discussion with the customer. The engineer should reflect on the improved key-properties from the customer's needs. These improved properties should be satisfied by the targetNN after a second process iteration. The specification of the improved key-properties serves to obtain a better overview of the desired targetNN. Thus, the engineer has a clear picture of the targetNN.

Finally, the output of this activity is the specification of the improved key-properties.

3.4 Specify Synthetic Dataset Augmentation

This activity's data input is the specification of the key-properties. In our initial definition of the process, we focused on analysing the correctly/incorrectly classified and recognised classes. Based on this analysis, we defined some dataset modifications to improve our neural network.

In the updated process presented in this paper, we define our dataset augmentation based on the specification of key-properties. The key-property specification might cover the analysis of the correctly/incorrectly classified and recognised classes. Thus, the engineer would also be able to follow the previous version of our process. However, the specification of the key-properties allows us to have a larger understanding of the neural network's recognition skills. Thus, this large understanding allows us to improve our dataset augmentation by considering the key-properties instead of only the correctly/incorrectly classified and recognised data.

The engineer analyses the specification of the key-properties and verifies if the current datasets are compatible with the specification. The detected issues of the datasets must be solved by specifying a dataset augmentation. The engineer analyses the key-property specification and the suggested improvements. Based on the analysis, the engineer specifies some dataset modifications and improvements. These dataset modifications and improvements are specified to obtain the dataset augmentation specification. The engineer might define operations such as:

- Generate additional data to strengthen the neural network's recognition skills
- Remove some unrecognizable data from the dataset
- Generate data to have more data variations (e.g. flip, rotate…)
- ….

Finally, the dataset augmentation specification is created. The dataset augmentation consists of a list of properties and tasks concerning the new datasets, e.g. "Generate 4 random images with shifted digits per equivalence class to strengthen the targetNN's recognition precision".

3.5 Synthesizer and Augmented Dataset Engineering

In this section, we present the remaining process activities before iterating the process. This part of the process focuses on engineering a data synthesiser and generating the synthetic data.

The first activity focuses on the engineering of a synthesizer. Based on the dataset augmentation, the engineer has to find the best synthesizer type for generating the required synthetic data. In our initial definition of the process, we suggested only the usage of a synthesizer neural network (synthesizerNN). In the updated process presented in this paper, we suggest these two different options:

- **Synthesizer** refers to a classical program. The program consists of functions that are able to manipulate data. (e.g. image filters, cropping, rotating…)

- **SynthesizerNN** refers to a neural network-based program for automatic data generation. The synthesizerNN may be, for instance, a generative adversarial network [17].

The second activity focuses on generating an augmented dataset. This activity remains unchanged as in [1]. The engineer starts with the synthetic data generation by executing the synthesizer. He monitors the synthesizer execution. Once the synthesizer has generated the synthetic data, the engineer evaluates the data manually or using some similarity function. The similarity function is used to compare the generated synthetic data with the original data and to sort out the synthetic data that differ to much from the original data. The engineer may define a tolerance threshold for the similarity of the synthetic and original data.

The accepted synthetic data is added to the different dataset with respect to the dataset augmentation specification.

4. EXPERIMENTATION

In this section, we present an experimentation conducted on the approach introduced in this paper. We describe an academic case study on the recognition of a meter counter state to experiment our approach. In this experiment, we instantiate our process to engineer a dataset and a neural network (NN), which recognizes the state of a two-digit mechanical meter counter, based on customer's requirements.

Before executing the process, we define a set of initial customer's requirements to be satisfied by the dataset and the neural network. Table 2, below, describes these functional requirements (FR):

Table 2. Customer's functional requirements

ID	Description
FR_1	A digital digit shall be represented with 7 segments.
FR_2	The meter counter state shall be represented as digital two-digit numbers from 00 to 99.
FR_3	The incrementation of a meter counter shall be represented as shifts of the incremented digit(-s) from top to down.
FR_4	Images shall be sharp, dark, bright or dusty images.
FR_5	The image size shall be fixed to 310x330 pixels.
FR_6	The NN's output shall be two integers values.
FR7	NN shall recognise the state of a two-digit meter counter by outputting the corresponding integer value.

Secondly, we define these non-functional requirements (NFR), as described in Table 3, below:

Table 3. Customer's non-functional requirements

ID	Description
NFR_1	NN shall recognise >99% of training data.
NFR_2	NN shall recognise >97% of development data.
NFR_3	NN shall recognise >95% of testing data.
NFR_4	NN's loss shall be less than 0.02.

Given the customer's requirements, we execute our process' activities to engineer the requested dataset and a neural network. In this experiment, we focus on the process' activity F (Act-F) which is our main contributions and summarize the other activities.

4.1 Raw Datasets and targetNN Engineering

The process' Data Input is a set of classified reference and test images provided by the customer and having these characteristics:

- 10 reference images, $img_{ref} \in [0, 255]^{310\times165}$, classified into 10 equivalence classes, $ec_{ref} = \{0,...,9\}$. The images represent a digital digit in [0...9]
- 172 testing images, $img_{test} \in [0, 255]^{310\times165}$, classified into 100 equivalence classes $ec_{counter} = \{ec_{ref}, ec_{ref}\}$. These images consist of a random selection of meter counter states with digital numbers in [00..99]. The images show sharp, dark, bright or dirty digits. Moreover, some images represent shifted digits to model the change to the next greater integer value.

The first process' activity focuses on engineering the raw datasets. Given the Data Input, we concatenate all possible combinations of two reference images to obtain the representative classified images of meter counter states. The concatenated images show all representative and sharp digital numbers in [00..99]. Figure 2 and 3 show the reference images and some random samples of generated images.

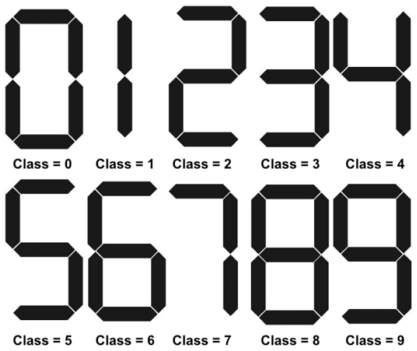

Figure 2. Reference images.

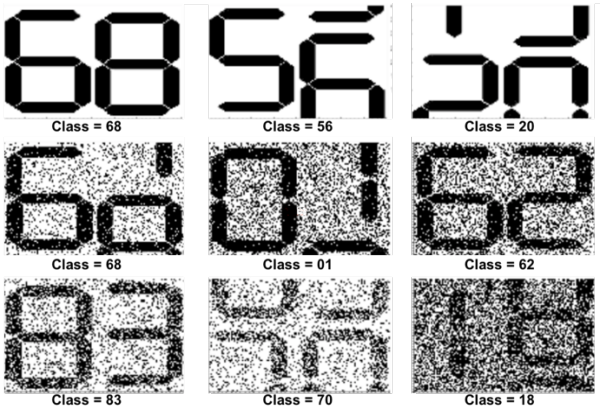

Figure 3. Random samples generated meter counter images.

Given the raw classified images dsraw and the classified test images, we performed the four tasks described in the approach to create our equivalence classes and datasets:

- The equivalence classes are defined as $ec_{counter} = \{ec_{ref,}$ $ec_{ref}\}$.
- The training dataset, $ds_{train} \in P([0, 255]^{310\times165} \times ec_{counter})$, consists of 645 random images from ds_{raw}.
- The development dataset, $ds_{dev} \in P([0, 255]^{310\times165} \times ec_{counter})$, contains the 40 remaining images from ds_{raw}.
- The testing dataset, $ds_{test} \in P([0, 255]^{310\times165} \times ec_{counter})$, contains all classified test images.

In the next process' activity, we focus on engineering a target neural network (targetNN). Laroca et al. [5] present a convolutional neural network (CNN) architecture for the recognition of a real-world meter counter state. Our CNN architecture and targetNN implementation in this experimentation is inspired from them. We implemented our CNN in Python [18] using the Keras [19] and Tensorflow libraries [20].

Our targetNN has 9 layers: 4 convolutional layers, 2 max-pooling layers, 2 fully connected layers with a random dropout of 30% and 1 output layer with randomly initialized weights. We use the activation function "Relu" except in the output layer, where we use the activation function "Sigmoid". The output layer has 20 neurons and outputs a probability distribution over 2 x 10 possible equivalence classes. The probability distribution describes the likelihood that a digit at the tens' and ones' position is recognized as an equivalence class. We selected the "binary cross-entropy" [21] function as our loss function.

The next process's activities focus on the targetNN training and the analysis of the training monitoring data. We trained our targetNN [1] for +/-4 hours and 25 epochs. We observed the targetNN's accuracy evolution on the training and development dataset. We decided to adjust the targetNN's parameters to reduce signs of over- and underfitting [22] based on our observations. Figure 4 shows the accuracy and loss evolution on the training and development dataset. After the training, we analysed our training monitoring data consisting of an accuracy and a loss value for the training and development dataset. Our targetNN reached the following accuracies and losses:

- ds_{train}: accuracy 100% and loss 0.002707.
- ds_{dev}: accuracy 99.37% and loss 0.024017.

From the analysis of these accuracies and loss values, we can conclude that it is unlikely that the targetNN is overfitting. Thus, we can stop the training, then accept and freeze the targetNN's architecture.

[1] The targetNN training has been performed on a machine having the following specs: 2,4Ghz - 32GB RAM;

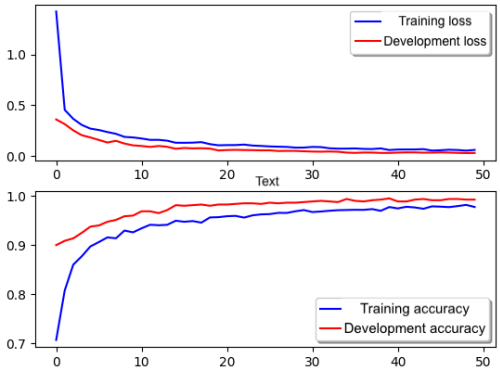

Figure 4. Accuracy and loss diagram for ds_{train} and ds_{dev}.

4.2 Test the targetNN

In this activity, we test our targetNN with the test images of dstest. As described in the previous section, our targetNN takes as input every test image and outputs a probability distribution over 2x10 equivalence classes. We decide to classify the left and right digits of the image separately in the equivalence classes with highest probability of the first 10 resp. last 10 equivalence classes.

Thanks to our selection criteria, we are able to compare the targetNN's recognized equivalence classes with the expected equivalence classes. It allowed us to generate these test monitoring data computed from the testing dataset using the bokeh library [23]:

- ds_{test}: accuracy 99.56% and loss 0.028118
- 3 confusion matrices for measuring quantitatively the correctly and incorrectly recognized images (CR and ICR images)
- 2 grids of correctly, resp. incorrectly, recognized images

Figure 5 shows some sample images of correctly and incorrectly recognized images.

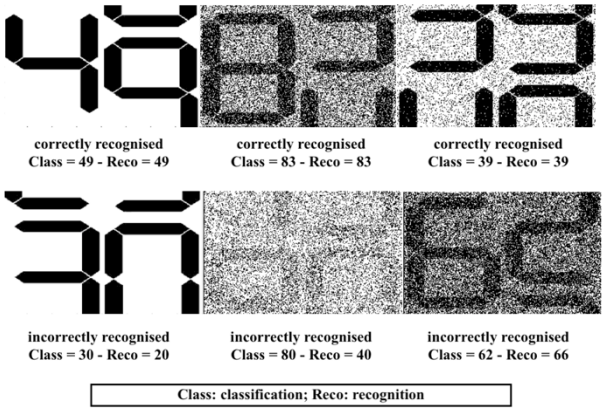

Figure 5. Random samples of correctly and incorrectly recognised images.

4.3 Analyse the Test Monitoring Data

In this activity, we present our analysis of the test monitoring data and the specification of the key-properties of our targetNN. Our first activity focuses on the identification of the new key-properties.

4.3.1 *Identify new key-properties*

The activity's Data input is the test monitoring data resulting from the previous activity. We classify the test monitoring data into the following two main categories, quantitative and qualitative data:

- Quantitative data
  - Accuracies
  - Losses
  - Confusion Matrices
- Qualitative data
  - Grids of CR and ICR recognised images
  - Accuracy and Loss evolution diagram

We use the categorised test monitoring data to identify a set of properties to be specified for our dataset and trained neural network. In this experiment, we list the most important properties to illustrate our approach. Table 4 shows a list of identified quantitative key-property types.

Table 4. List of quantitative key-property types

Category	Subcategory	Property
Quantitative	Discrete	Number of CR/IR images
Quantitative	Discrete	Number of CR/IR images per equivalence class
Quantitative	Discrete	Size of datasets
Quantitative	Continuous	Ratio of CR/IR images
Quantitative	Continuous	Ratio of CR/IR images per equivalence class
Quantitative	Continuous	Ratio of CR/IR images per equivalence class
Quantitative	Continuous	Neural network's recognition precision

Table 5 shows a list of identified qualitative key-property types. We use these key-property types to specify the key-property instances of our datasets and targetNN.

Table 5. List of qualitative property types

Category	Subcategory	Property
Qualitative	Logical	Data and classification correctness
Qualitative	Logical	Correctness of the classification and recognition
Qualitative	Nominal	Data consistency
Qualitative	Nominal	Groups of incorrectly recognised images
Qualitative	Ordinal	Recognition weaknesses and strengths
Qualitative	Ordinal	Threshold for belonging to an equivalence class

4.3.2 *Specify key-properties*

In this activity, we focus on the specification of the identified key-properties with respect to the list of identified key-properties. We determine the key-properties based on our observations on the test monitoring data. The key-properties of our targetNN and our datasets are specified in natural language.

Table 6 shows a list of specified quantitative key-properties. We mainly focus on specifying the amount of correctly and incorrectly recognised (resp. classified) data. Additionally, we specify statistics on the datasets and the targetNN's recognition precision.

Table 6. Quantitative key-properties at iteration 1

ID	Subcategory	Description
$KP_{v1,1}$	Continuous	NN recognises 99.97% of the training data
$KP_{v1,2}$	Continuous	NN recognises 99.25% of the development data
$KP_{v1,3}$	Continuous	NN recognises 99.50% of the testing data
$KP_{v1,4}$	Discrete	NN recognises the left digit on 8 images incorrectly.
$KP_{v1,5}$	Discrete	NN recognises the right digit on 3 images incorrectly.
$KP_{v1,6}$	Continuous	NN has an average loss of 0.02624
$KP_{v1,7}$	Discrete	320 (all) images have been correctly classified.
$KP_{v1,8}$	Discrete	The test dataset size is 320

Table 7 shows a list of specified qualitative key-properties. We focus on the specification of the following properties:

- Correctness of the recognised equivalence classes
- Tested equivalence classes
- TargetNN's recognition weaknesses and strengths

Table 7. Qualitative key-properties at iteration 1

ID	Subcategory	Description
$KP_{v1,9}$	Logical	NN recognises the left digit in $ec_{tens}=\{1,3,6,7\}$ correctly.
$KP_{v1,10}$	Logical	NN recognises the left digit in $ec_{ones}=\{0,2,4,5,8,9\}$ incorrectly.
$KP_{v1,11}$	Logical	NN recognises the right digit in $ec_{tens}=\{1,2,3,4,5,7,8,9\}$ correctly.
$KP_{v1,12}$	Logical	NN recognises the right digit in $ec_{ones}=\{0,6\}$ incorrectly.
$KP_{v1,13}$	Nominal	NN's recognition of $ec_{class} = \{30\}$ has not been tested.
$KP_{v1,14}$	Ordinal	The equivalence class to which an image will be declared to belong to is the one for which the recognition probability is the highest.
$KP_{v1,15}$	Nominal	NN recognises the sate of a meter counter at a 60% certitude, if both digits are shifted.
$KP_{v1,16}$	Nominal	NN does not always recognise images of shifted digits on sharp and dirty images.
$KP_{v1,17}$	Ordinal	NN recognises the right digit on an image very well.

Thanks to our specification of the key-properties of the dataset and the targetNN, we are able to obtain an overview of the targetNN's recognition skills. This allows us to perform the next activity of our software engineering process.

4.3.3 Analysing targetNN's key-properties

In this activity, we focus on the analysis of our specified key-properties. We present a list of strengths and weaknesses of our targetNN and our datasets that could be presented to some customer [2]. Additionally, we suggest some solutions to potentially improve our targetNN. Table 8 shows a list of weaknesses of our targetNN and proposed solutions.

Table 8. Detected weaknesses at iteration 1

Weakness	Proposed solution
Low number of test cases, since we have 2 images per equivalence class in average.	Augment the testing dataset with sufficient variations and at least 5 images per equivalence class
Recognition problems, if the digit at the ones place is a 0 or 9.	Augment the training dataset with images that contain a 9 or a 0 at the ones place.
Recognition precision (loss value not satisfied).	Augment the training dataset
Neural network's recognition policy (When do we consider that the neural network has recognised some data?)	The equivalence class to which an image will be declared to belong to is these ones for which the recognition probability is higher than 0.9.

Table 9 shows a list of strengths of our targetNN.

Table 9. Detected strengths at iteration 1

Strengths	Descriptions
Overall recognition correctness	The targetNN recognised correctly recognise more than 99% of the training, development and testing data.
targetNN's reliability	The targetNN is not showing signs of over- and underfitting.
targetNN's recognition precision	The targetNN's recognition precision can be improved to satisfy the initial requirement
Dataset augmentation simplicity	The dataset design allows us to generate efficiently synthetic data to augment the dataset

The results and the suggested improvements have to be presented to the customer for validating the neural network. The weaknesses and strengths are discussed with the customer to obtain a first feedback. A customer might suggest some weaknesses to be solved. A customer could claim that the recognition problems of our targetNN may lead to some severe financial problems in his institution. Let's suppose that it is mandatory for him to recognise the state of the meter counter at a precision of +/- 2. Table 10 summarises some possible customers weakness proposals.

Table 10. Customer's weakness proposals at iteration 1

Weakness	Proposed Solution
High differences in between the classification and recognition	Recognise the state of the meter counter at a precision of +/- 2.

As a result of the discussion, let's suppose that the targetNN has not been validated by the customer. Thus, we decide to improve the targetNN in a second process iteration to satisfy the customer's requirements.

4.3.4 Specify improved key-properties

In this activity, we focus on the specification of the improved key-properties based on the results of our discussion with the customer. Thus, we specify a new set of improved key-properties to be satisfied by the neural network.

Table 11. Specification of improved key-properties

ID	KP	Description
$KP_{v1,imp,1}$	$KP_{v1,14}$	The acceptance threshold that an image belongs to an equivalence class should be fixed to 0.9.
$KP_{v1,imp,2}$	$KP_{v1,8}$	The testing dataset size should be at least 1000. There should be at least 5 images per equivalence class.
$KP_{v1,imp,3}$	$KP_{v1,18}$	The right digit of an image can be recognised with a tolerance of +/- 2.

Table 11 shows a list of specified improved key-properties resulting from the discussion with the customer. The improved key-properties must be satisfied after the second process iteration. We can move forward to the next activity of our process.

4.4 Specify Synthetic Dataset Augmentation

In this activity, we focus on the specification of the synthetic dataset augmentation. We consider the discussed improved key-properties to specify our dataset augmentation. Table 12 shows a list of data generation operations to augment our datasets.

Table 12. Dataset augmentation specification

ID	Description
$DS_{aug,1}$	Generate 5 random images per equivalence class to be added to the testing dataset.
$DS_{aug,2}$	Generate 100 random images with a 9 or 0 at the ones place.
$DS_{aug,3}$	Generate 2 random images per equivalence class to be added to the training and development dataset to strengthen the targetNN's recognition.
$DS_{aug,4}$	Generate 4 random images with shifted digits per equivalence class to strengthen the targetNN's recognition precision.

[2] This is an academic case study, as such, no real customer was part of the experimentation. Thus, all mentions to a customer are assumptions to perform an interesting case study for the illustration of our approach.

We use the dataset specification augmentation in the next activity of our software engineering process.

4.5 Synthesizer and Augmented Dataset Engineering

In this activity, we focus on engineering a synthesizer to augment our datasets. We do not design a synthesizer neural network as originally described in Jahic-et-al's paper [1]. We implemented in Python a set of functions to generate data based on the dataset augmentation specification. We used different libraries (e.g numpy [24], skimage [25] and bokeh [23]) to support efficient data management and the generation of our synthetic image.

Thanks to the reference images, we are able to generate some sharp images of a meter counter. Additionally, we add different types of noise to the sharp images in order to simulate dirty images using the skimage library. They selected randomly some images and changed their brightness, which allowed us to generate additional data to be added to the dataset. The generated data is used to augment the dataset with additional images to satisfy the dataset augmentation specification.

Finally, we collect and sort all the generated data based on our dataset augmentation specification. The generated data is added to the corresponding datasets. Thus, we obtain new augmented training, development and testing datasets. These datasets are then used to run our process a second time and to improve the neural networks recognition.

4.6 Summary of 2nd Process Iteration

In this section, we summarise our activities of the 2nd process iteration. As in the previous process iteration, we retrain our targetNN on the same number of epochs. The resulting training monitoring data has been analysed in detail to detect some signs of over- or underfitting. We did not detect any sign of over- or underfitting after the second training. Afterwards, we execute our new targetNN (targetNNv2) to recognise the images of the test dataset. The resulting test monitoring data are analysed in activity "Analyse test monitoring data" of our process.

In activity "Identify new key-properties", we first start by identifying the key-properties. We categorise our test monitoring data into our previously defined categories. Based on our observations of our key-properties, we do not identify new types of key-properties. We accept the previously defined activity to move to the next process' activity, activity "specify key-properties". We focus on the specification of the key-properties of our targetNNv2 and datasetv2. Table 13 and 14 show our new list of key-properties of our targetNN and our datasets.

Table 13. Quantitative key-properties at iteration 2

ID	Description
$KP_{v2,1}$	NN recognises 100% of the training data
$KP_{v2,2}$	NN recognises 100% of the development data
$KP_{v2,3}$	NN recognises 99.59% of the testing data
$KP_{v2,4}$	NN recognises the left digit on 5 images incorrectly.
$KP_{v2,5}$	NN recognises the right digit on 2 images incorrectly.
$KP_{v2,6}$	NN has an average loss of 0.014919
$KP_{v2,7}$	600 (all) images have been correctly classified.
$KP_{v2,8}$	The test dataset size is 600
$KP_{v2,9}$	NN does not recognise 7 noisy images with dark or bright backgrounds
$KP_{v2,10}$	NN does not recognise 3 noisy images with dark background and dark digits
$KP_{v2,11}$	NN does not recognise 2 noisy images with bright background and bright digits
$KP_{v2,11}$	NN does not recognise 2 noisy images with bright background and dark digits

Table 14. Qualitative key-properties at iteration 2

ID	Description
$KP_{v2,9}$	NN recognises the left digit in $ec_{tens}=\{0,1,3,4,5,7\}$ correctly.
$KP_{v2,10}$	NN recognises the left digit in $ec_{ones}=\{2,6,8,9\}$ incorrectly.
$KP_{v2,11}$	NN recognises the right digit in $ec_{tens}=\{0,1,3,4,5,6,7,8\}$ correctly.
$KP_{v2,12}$	NN recognises the right digit in $ec_{ones}=\{2,9\}$ incorrectly.
$KP_{v2,13}$	All equivalence classes have been tested.
$KP_{v2,14}$	The equivalence class to which an image will be declared to belong to is the one for which the recognition probability is higher than 0.9.
$KP_{v21,15}$	NN recognises the sate of a meter counter at more than 90% certitude, if both digits are shifted.
$KP_{v2,16}$	NN does not always recognise images of shifted digits on sharp and dirty images
$KP_{v2,17}$	NN recognises the left and right digit on an image very well.

In the next process' activity "analyse targetNN's key-properties", we focus on the analysis of the key-properties. Again, we defined a list of strengths and weaknesses of our targetNN. Table 15 and Table 16 show a set of detected weaknesses and strengths of our targetNN.

Table 15. Detected weaknesses at iteration 2

Weakness	Proposed solution
High number of unrecognised very dirty images	Augment the datasets with very dirty images.

Table 16. Detected strengths at iteration 2

Strengths	Descriptions
Overall recognition correctness	The targetNN recognised correctly more than 99% of the training, development and testing data.
Specific recognition correctness	The targetNN recognised very well all kinds of images as described in the requirements.
targetNN's reliability	The targetNN is not showing signs

	of over- and underfitting.
targetNN's recognition precision	The targetNN's recognition precision satisfies the initial requirement.

These weaknesses and strengths have been discussed with our customer. Let's suppose that he claims that the very dirty images are even difficult for humans to be recognised. This led the customer to add a functionality request for the neural network which is to detect dirty meter counters in order to either clean them or replace them. Thus, we should specify a new equivalence class, called 'dirty images', in the next activity. We then reclassify some very dirty images into the new equivalence. A 3rd process iteration could be lunched to train the neural network to satisfy the remaining improved key-properties.

As a result of our discussion, let's suppose that the customer validates our targetNN. Thus, the targetNN can be deployed on the customer's desired platform.

5. DISCUSSION

In this section, we present two main points of discussion. We present the current problems and limitations. We propose some improvements and briefly describe the related work.

A first point to discuss is that in our process, we do not define precisely the notion of customer's satisfaction. In the process presented in this paper, a customer is only able to express his satisfaction using a binary decision. Thus, he can validate a key-property with a yes/no-answer or propose some alternative key-properties to be satisfied. In case of a negative feedback of customers, the engineer must reengineer the datasets and the neural network until the key-properties are validated by the customer. However, this process is time-consuming, and it is very difficult to validate the requirements. Moreover, it is difficult to keep track of the changed requirements during the evolution of the datasets and the neural network. Guelfi [26] presents a formal framework for dependability and resilience from a software engineering perspective, called DREF. He introduces the notion of satisfiability as a measure for evaluating a property (e.g. requirements…) of an entity (e.g. programs, neural networks…). The satisfiability is expressed as a real number of a user-defined grading scale. Moreover, he introduces the notion of tolerance threshold to describe precisely the strictness of the customer's satisfiability. He visualises the satisfiability for the evolution of each entity in time. We suggest as possible improvement to integrate the DREF framework into our process to precisely describe the satisfiability of the customer's key-properties. Thus, the engineer could focus on engineering a neural network for the most important key-properties as per the customer's satisfiability. Additionally, we would be able to visualize the evolution of the satisfiability of the key-properties. Thus, we could use these diagrams to discuss the neural network's development progress and key-property improvements with the customer.

A second point to discuss is that in our process, we specify our key-properties informally using tables and natural language. Moreover, the process presented in this paper involves a lot of manual steps to specify the neural network's key-properties. Since the key-properties are currently specified in natural language, the specification could lead to misunderstandings. Thus, the engineer requires support to precisely specify the neural network's key-properties. Most papers talking about specifications present domain-specific languages [27-30] for the construction or the execution of neural networks. Mostly, they present domain-specific languages for the specification of an architecture of a neural network facilitate the implementation. We argue that the usage of domain-specific languages to specify the neural network's key-properties would improve our process. The domain-specific language would ease the identification and specification of the key-properties. Useful features, such as specification templates, auto-completion, syntax highlighting, would support the engineer to specify precisely the key-properties. Additionally, a translation program that interprets the specification of the key-properties and generates a dataset augmentation specification would facilitate the design of a synthesizer.

6. CONCLUSION

In this paper, we have presented an updated version of a software engineering process for improving neural networks using dataset augmentation based on customer's requirements. We introduced the notion of key-properties for specifying the neural network's key-properties. We presented a subprocess for specifying, improving and validating the neural network's key-properties. The process has been formalized using the BPMN 2.0 modelling language. We have presented a concrete experimentation of our software engineering process. We conducted our experimentation on an academic case study on the recognition of a meter counter state. The experimental results show that our approach is promising, because we improved our neural network's recognition skills by augmenting our datasets based on the specification of the neural network's key-properties.

For future work, we will work on a formal definition of a domain-specific language for data scientists. The domain-specific language should support the specification of the neural network's key-properties. We will use model-driven engineering [31] methods for generating a dataset augmentation specification interpreted from the specification of the neural network's key-properties.

Another future work would be the integration of DREF framework [26] into our process. We could propose a process for specifying the resilience of neural network-based systems based on customer's requirements. Thus, the data scientist could specify precisely the satisfiability and the customer's acceptance tolerance for each requirement. The process would improve the key-properties specification and as a result also the dataset augmentation for engineering a neural network that satisfies the customer's requirements.

7. REFERENCES

[1] Goodfellow, I., Bengio, and Y., Courville, A., 2016. Deep Learning. MIT Press.

[2] Jahic, B., Guelfi, and N., Ries, B., 2019. Software Engineering for Dataset Augmentation using Generative Adversarial Networks. In 10th International Conference on Software Engineering and Service Sciences. 59-66. DOI=https://doi.org/10.1109/ICSESS47205.2019.9040806

[3] Wong, S. C., Gatt, A., Stamatescu, V., and McDonnell, M. D., 2016. Understanding data augmentation for classification: when to warp?. In *2016 international conference on digital image computing: techniques and applications*, 1-6. DOI=https://doi.org/ 10.1109/DICTA.2016.7797091

[4] Nodari, A., and Gallo, I., 2011. A Multi-Neural Network Approach to Image Detection and Segmentation of Gas Meter Counter. In *International conference on Machine Vision Applications*. 239-242.

[5] Laroca, R., Barroso, V., Diniz, M. A., Gonçalves, G. R., Schwartz, and W. R., Menotti, D., 2019. Convolutional neural networks for automatic meter reading. *Journal of Electronic Imaging*, *28*(1), 013023. DOI= https://doi.org/10.1117/1.JEI.28.1.013023

[6] Vanetti, M., Gallo, I., and Nodari, A., 2013. GAS meter reading from real world images using a multi-net system. *Pattern Recognition Letters*, *34*(5). 519-526. DOI= https://doi.org/10.1016/j.patrec.2012.11.014

[7] Vogelsang, and A., Borg, M. 2019. Requirements Engineering for Machine Learning: Perspectives from Data Scientists. In *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. 245-251.

[8] Kostova, B., Gurses, S., and Wegmann, A. 2020. On the Interplay between Requirements, Engineering, and Artificial Intelligence. In *25th International Working Conference on Requirements Engineering: Foundation for Software Quality - Workshops*.

[9] Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Besmira, N., and Zimmermann, T. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*. 291-300.

[10] Hesenius, M., Schwenzfeier, N., Meyer, O., Koop, W., and Gruhn, V., 2019. Towards a software engineering process for developing data-driven applications. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE),* 35-41.

[11] Sommerville, I. 2016. *Software Engineering*. Harlow: Pearson Education Limited.

[12] Zheng, A., and Casari, A. 2018. *Feature engineering for machine learning: principles and techniques for data scientists.* O'Reilly Media, Inc..

[13] LeCun, Y., Bengio, Y., and Hinton, G. 2015. Deep learning. *nature*, *521*(7553). 436-444.

[14] Object Management Group, 2011. Business Process Modeling Notation (BPMN) v2.0. Full Specification Formal/2011-01-03.

[15] Hawkins, D. M., 2004. The problem of overfitting. *Journal of chemical information and computer sciences*, *44*(1), 1-12.

[16] Driscoll, P., Lecky, F., and Crosby, M., 2000. An introduction to everyday statistics—1. *Emergency Medicine Journal*, *17*(3). 205-211.

[17] Mirza, Mehdi, and Simon Osindero., 2014. "Conditional generative adversarial nets." *arXiv preprint arXiv:1411.1784*.

[18] Summerfield, M. 2010. *Programming in Python 3: a complete introduction to the Python language.* Addison-Wesley Professional.

[19] Gulli, A., and Pal, S. 2017. *Deep learning with Keras.* Packt Publishing Ltd.

[20] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., and Isard, M., 2016. Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation* 16 (2016). 265–283

[21] De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. 2005. *A tutorial on the cross-entropy method*. Annals of operations research 134, 1 (2005). 19–67.

[22] Jabbar, H., and Khan, R. Z. 2015. *Methods to avoid over-fitting and under- fitting in supervised machine learning (comparative study)*. Computer Science, Communication and Instrumentation Devices (2015). 163–172.

[23] Jilly, K. 2018. *Hands-On Data Visualization with Bokeh: Interactive Web Plotting for Python Using Bokeh*. Packt Publishing Ltd, 2018.

[24] Oliphant, T. E., 2006. *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.

[25] Van der Walt et al, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart E, Yu, T. and the scikit-image contributors., 2014. scikit-image: image processing in Python. *PeerJ*, *2*, e453. DOI=https://doi.org/10.7717/peerj.453

[26] Guelfi, N., 2011. A formal framework for dependability and resilience from a software engineering perspective. *Open Computer Science*, *1*(3). 294-328.

[27] Sankaran, A., Aralikatte, R., Mani, S., Khare, S., Panwar, N., and Gantayat, N., 2017. DARVIZ: deep abstract representation, visualization, and verification of deep learning models. In *2017 IEEE/ACM 39th International Conference on Software Engineering: New Ideas and Emerging Technologies Results Track*. 47-50.

[28] Tamilselvam, S. G., Panwar, N., Khare, S., Aralikatte, R., Sankaran, A., and Mani, S., 2019. A visual programming paradigm for abstract deep learning model development. In *Proceedings of the 10th Indian Conference on Human-Computer Interaction*. 1-11.

[29] Zhao, T., and Huang, X. 2018. Design and implementation of DeepDSL: A DSL for deep learning. *Computer Languages, Systems & Structures*, *54.* 39-70.

[30] Elango, V., Rubin, N., Ravishankar, M., Sandanagobalane, H., and Grover, V. 2018. Diesel: DSL for linear algebra and neural net computations on GPUs. In *Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*. 42-51.

[31] Kent, S. 2002. Model driven engineering. In *International Conference on Integrated Formal Methods.* Springer, Berlin, Heidelberg. 286-298.