Tampere University

Mika Iitti

BRICK CLASSIFICATION USING A CONVOLUTIONAL NEURAL NETWORK

Master's Thesis
Faculty of Engineering and
Natural Sciences

Examiners:
Tarmo Lipping
Jari Turunen
3 2022

ABSTRACT

Mika Iitti: Brick classification using a convolutional neural network
Master's Thesis
Tampere University
Master's Degree Programme in Automation Engineering
March 2022

Machine learning is a field of artificial intelligence which can be utilised in image recognition applications. By means of machine learning, software capable of solving complex tasks can be developed. In this work, a convolutional neural network was used as the machine learning method. The feasibility of using convolutional neural networks in masonry brick classification was studied in this Master's thesis. The aim was to discover the most efficient network model by testing different network structures.

According to the preliminary development plan, the classification of brick types can be done in the brick factory. This is conducted by imaging bricks with a machine vision camera while the bricks are moving on a conveyor belt. After the imaging, a machine learning algorithm identifies different brick classes from the images. The bricks are sorted into different classes at the end of the production line, using conveyor belts and industrial robots.

The content of the Master's thesis can be divided into three sections: the theoretical part, the image acquisition part and the results achieved with the network model. At first, topics related to the camera model, image formation and optics are covered. This is followed by the theory part, regarding machine learning and neural networks. Also, the network model used is presented. After the theoretical part, the equipment and the methods used in image acquisition are introduced. Finally, the results obtained with the network model are presented, and the suitability of the model for brick classification is evaluated.

The bricks were pre-divided into three classes at the brick factory, based on different quality requirements. The imaging of the bricks was done in a laboratory, where 1105 bricks were imaged three times. A machine vision camera and lighting system were assembled on a conveyor belt. By using this system, each brick was imaged from the top, bottom and side surfaces. The machine vision camera used was controlled by vision software. With this equipment, the images were captured automatically as the bricks moved along the conveyor belt.

After the image acquisition, the image data was modified into a suitable format for the machine learning algorithm. Two image datasets were used to test the convolutional neural network. The original dataset included 1800x1500 pixel size images. The second dataset included the same images. Additionally, in the second dataset images, the excessive background areas around the bricks were cropped. Both datasets contained 1105 images, consisting of the three brick classes.

A pretrained VGG16 convolutional neural network was chosen as the network model. A transfer learning method was utilised with this network model. In transfer learning, pretrained network parameter values are used. Pretrained parameters were obtained for the VGG16 by training the network with the ImageNet database. Generally, network performance in a classification task can be improved by using transfer learning. This is especially the case when the number of images to be used in a classification task is limited.

According to the results obtained using two different image size datasets, it can be concluded that the convolutional neural network is well suited for the classification of brick images. When performing image acquisition in factory conditions, it is important to keep the ambient lighting conditions constant during shooting. Acquiring a high-quality image dataset is an essential part of developing the classification system.

Keywords: Machine learning, Convolutional neural network, Machine vision, Transfer learning

TIIVISTELMÄ

Mika Iitti: Tiilien luokittelu konvoluutioneuroverkolla
Diplomityö
Tampereen yliopisto
Automaatiotekniikan diplomi-insinöörin tutkinto-ohjelma
Maaliskuu 2022

Koneoppiminen on tekoälyyn sisältyvä osa-alue, jota voidaan hyödyntää kuvantunnistukseen liittyvissä sovelluksissa. Koneoppimisen avulla voidaan kehittää ohjelmistoja, jotka ovat kykeneviä ratkaisemaan monimutkaisia tehtäviä. Tässä työssä käytettiin koneoppimismenetelmänä konvoluutioneuroverkkoa. Diplomityössä tutkittiin, soveltuuko konvoluutioneuroverkko tiiliskivien luokitteluun. Testaamalla erilaisia verkkorakenteita pyrittiin löytämään mahdollisimman tehokas verkkomalli.

Kehitysvaiheessa olevan suunnitelman mukaisesti, tiilityyppien luokittelu voidaan toteuttaa tiilitehtaalla, kuvaamalla liukuhihanalla kulkevat tiiliskivet konenäkökameran avulla. Kuvauksen jälkeen koneoppimis-algoritmi tunnistaa erityyppiset tiiliskivet valokuvien perusteella. Tiiliskivet lajitellaan eri laatuluokkiin linjaston loppupäässä, hyödyntäen liukuhihnoja ja teollisuusrobotteja.

Diplomityön sisältö voidaan jakaa kolmeen osaan: teoriaosuuteen, valokuvien hankintaan sekä verkkomallilla saatuihin tuloksiin. Työssä käsitellään ensin kameran mallinnukseen, kuvanmuodostukseen sekä optiikkaan liittyviä aiheita. Tämän jälkeen käydään läpi koneoppimiseen ja neuroverkkoihin sisältyvää teoriaa, sekä esitellään työssä käytetty verkkomalli. Teoria-osuuden jälkeen esitellään käytetyt laitteistot ja menetelmät, joilla valokuva-aineisto kerättiin. Lopuksi käsitellään verkkomallin avulla saatuja tuloksia ja arvioidaan mallin soveltuvuutta tiiliskivien tunnistukseen.

Tiiliskivien valokuvaus suoritettiin laboratoriossa, jossa 1105 kpl tiiliskiviä kuvattiin kolmeen kertaan. Tiiliskivet oli valmiiksi jaoteltu tehtaalla kolmeen luokkaan erilaisten laatuvaatimusten perusteella. Valokuva-aineiston keräämiseksi, konenäkökamera ja valaistusjärjestelmä asennettiin liukuhihnalle. Kuvausjärjestelmän avulla jokainen tiili kuvattiin yläpuolelta, alapuolelta sekä sivusta. Kuvaukseen käytettyä konenäkökameraa ohjattiin siihen liitetyllä ohjelmistolla. Laitteiston avulla valokuvaus tapahtui automaattisesti, tiiliskivien kulkiessa liukuhihnalla.

Valokuvien keräyksen jälkeen, aineistoa muokattiin sopivaan muotoon, jotta valokuvia pystyttiin syöttämään koneoppimis-algoritmiin. Konvoluutioverkon testauksessa käytettiin kahta eri valokuvakokoelmaa. Alkuperäinen kuvakokoelma sisälsi 1800x1500 pikselin kokoisia valokuvia. Toinen kuvakokoelma sisälsi samat valokuvat, joista ylimääräiset taustat tiiliskivien ympäriltä oli leikattu pois. Molemmat kokoelmat sisälsivät 1105 kpl valokuvia, jotka koostuivat kolmesta eri tiililuokasta.

Verkkomalliksi valittiin esiopetettu VGG16-konvoluutioneuroverkko, jonka avulla hyödynnettiin siirto-oppimiseen liittyvää menetelmää. Siirto-oppimisessa käytetään esiopetettuja parametriarvoja, jotka VGG16 verkon osalta oli saatu opettamalla verkkomalli ImageNet-valokuvakokoelmalla. Menetelmällä voidaan yleisesti parantaa verkon luokitustulosta, erityisesti jos tunnistuksessa käytettäviä valokuvia on saatavilla rajoitetusti.

Kahdella eri kuvakokoelmalla saatujen tulosten perusteella voidaan todeta, että konvoluutioneuroverkko soveltuu hyvin tiiliskivien luokitteluun. Valokuvattaessa tunnistettavia tiiliskiviä tehdasolosuhteissa, on tärkeää pitää ympäristön valaistusolosuhteet vakioina kuvauksen aikana. Hyvälaatuisen valokuvakokoelman hankinta on oleellinen osa toimivan tunnistusjärjestelmän kehittämistä.

Avainsanat: Koneoppiminen, Konvoluutioneuroverkko, Konenäkö, Siirto-oppiminen

PREFACE

I would like to thank the supervisors of the Master's thesis, Professor Tarmo Lipping and University Lecturer Jari Turunen for their good and close cooperation during the work. In addition, I would like to thank Jani Uusitalo, Jere Grönman and Pauli Valo, for their good collaboration during the thesis project.

Pori, 19 March 2022

Mika Iitti

CONTENTS

1. INTRODUCTION 1
2. IMAGE ACQUISITION 3
    2.1 Pinhole camera model 3
    2.2 Camera parameters 5
    2.3 Distortions 7
    2.4 Machine vision camera 8
    2.5 Optics 11
3. MACHINE LEARNING 14
    3.1 Types of machine learning 14
    3.2 Neural networks 18
    3.3 Training of neural networks 22
    3.4 Convolutional neural networks 24
    3.5 VGG16 model 27
    3.6 Transfer learning 28
    3.7 Data augmentation 31
    3.8 Performance metrics and overfitting 32
4. TEST IMPLEMENTATION 34
    4.1 Initial steps to gather data 34
    4.2 Test equipment 37
    4.3 Collected data 41
    4.4 Modifications to collected data 44
5. CLASSIFICATION TEST RESULTS 47
    5.1 Cross-validation and oversampling 47
    5.2 Results 49
    5.3 Original image size with all layers open 50
    5.4 Cropped image size 52
    5.5 Cropped image size with all layers open 54
    5.6 Discussion of the results 56
6. CONCLUSIONS 59
REFERENCES 61

LIST OF SYMBOLS AND ABBREVIATIONS

ANN	Artificial neural networks
CE	Cross-entropy error function
CIFAR-10	Dataset consisting of 60000 32x32 pixel colour images
CMOS	Complementary metal-oxide semiconductor
CNN	Convolutional neural network
CPU	Central processing unit
C++	Programming language
DOF	Depth of field
f-number	Focal length value, divided by the diameter of the aperture
FOV	Field of view
GD	Gradient descent algorithm
GPU	Graphics processing unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LED	Light-emitting diode
MAE	Mean absolute error
MLP	Multilayer perceptron
MP	Megapixel
MSE	Mean squared error
Nf	Number of pixels to map the smallest feature
NN	Neural network
PCA	Principal component analysis
PC	Personal computer
Rc	Camera resolution
R-CNN	Region-based convolutional neural network
ReLU	Rectified linear unit
RGB	Red, green and blue colour channels
Rs	Spatial resolution
$\mathbb{R}^n$	N-dimensional real number space
Sf	Size of the smallest feature
SGD	Stochastic gradient descent algorithm
SMOTE	Synthetic minority oversampling technique
SOM	Self-organizing map
SSL	Semi-supervised learning
SVM	Support vector machine
Tanh	Hyperbolic tangent
TPR	True positive rate
WDH	Width, depth and height
WD	Working distance, object's distance to the camera
y^{GT}	Ground truth value, correct value for a certain datapoint
$\hat{y}$	Estimated value for a certain datapoint
2D	Two-dimensional
3D	Three-dimensional

1. INTRODUCTION

Bricks belong to the oldest building elements in the world. The earliest findings of bricks can be dated to 7000 BC in southern Turkey, near the city of Jericho. Ancient bricks were made by hardening mud bricks in sunlight, in the warm regions of the world. The usage of fired bricks started in about 3500 BC, which also made it possible to manufacture bricks in cooler climate regions [1].

The objective of this thesis was to examine the quality of masonry bricks, with an appropriate machine learning model. The topic of this thesis was suggested by a local engineering company, which received a commission from a brick factory to improve the defect detection of masonry bricks. The plan was to implement quality inspection from high quality images, captured by a machine vision system. The research questions were firstly to examine whether a quality inspection can be conducted using machine learning. Secondly, if a feasible model was found, the challenge was to create a useful application. Also, guidelines for appropriate imaging setup were to be planned, to be able to collect images for inspection.

In factory facilities, masonry bricks are inspected manually by an operator and then divided into three different categories. There is growing interest in upgrading the automation level in the brick factory to enable more efficient production. In the factory in question, there are over 10 brick colour options to choose from. Additionally, the brick size varies, and the outer surface can have five different patterns. One brick type was chosen for examination. The plan was to train the learning model using one brick type and then, if successful, it would be easy to train other types with the same model.

The early plan for the final implementation is illustrated in Figure 1. The configuration consists of robots, conveyor belts and an imaging system on the production line. A robot arm would pick a single brick at a time from the pile, as it is discharged from the oven. This is followed by an imaging setup including line scan cameras, where bricks are imaged from three directions (Figure 1). The three images are combined into one image, which is processed by a learning algorithm. Eventually, the brick is sent to the correct conveyor, according to the brick’s classification result.

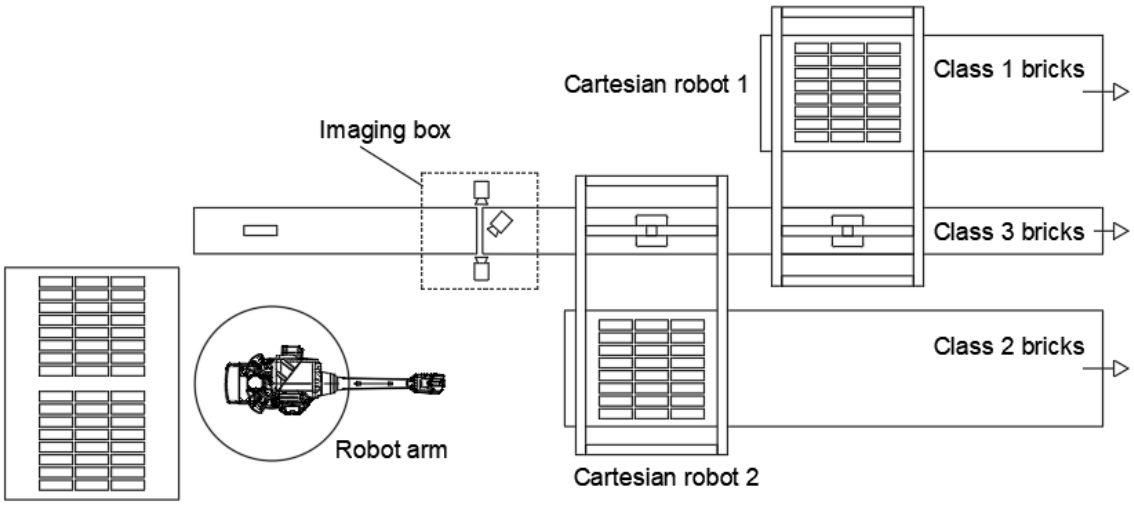

Figure 1. *Factory layout plan consists of an imaging setup, robots and conveyor belts. A robot arm picks bricks to the conveyor, one at a time. Cartesian robots stack bricks onto another conveyor, according to their class.*

During this thesis work, different kinds of tasks were conducted. The main stages of the work are listed as follows:

- An imaging setup was built by imaging 2400 kg of bricks three times. A dataset was collected, consisting of 3315 images representing the three classes.
- The images were arranged in the right order, and excessive data samples were removed. Only three images per single brick were stored.
- A custom-made program was used to crop external, white background areas from the original images.
- Two Python programs were implemented to combine three brick images into one image. The first program was used to combine the original size images, with large background areas. Another program was used to combine the cropped size images.
- A convolutional neural network (CNN) model for the classification was implemented, using the Python programming language.
- Cross-validation and oversampling algorithms were implemented in Python to evaluate the CNN model's performance.

This thesis is divided into three main parts. Chapters 2 and 3 cover topics related to the camera model, image formation, optics and machine learning. Methods for image acquisition and the equipment used in imaging are introduced in the test implementation chapter. At the end of the thesis, in Chapters 5 and 6, the results of the image classification tasks are covered.

2. IMAGE ACQUISITION

In this chapter, the pinhole camera model and camera parameters are introduced. There will be an explanation of the formulas related to the matrix representation of the camera parameters. This is followed by a section which describes the two common distortion types affecting image quality. At the end of the chapter, some machine vision camera and optics related topics are introduced.

2.1 Pinhole camera model

With a pinhole camera model, image formation from a 3D space to a 2D image plane can be represented. This simple camera model was introduced by Brunelleschi early in the fifteenth century [2]. The model assumes that a single ray of light that goes through the camera aperture at one time connects points between the image plane and the scene. In fact, a cone of light rays forms an object in the image plane. This simplified pinhole model, sometimes called a central perspective projection model, is often accurate enough to approximate the imaging process mathematically [2]. An image of the object lying in the 3D space is projected to the 2D image plane, as shown in Figure 2.

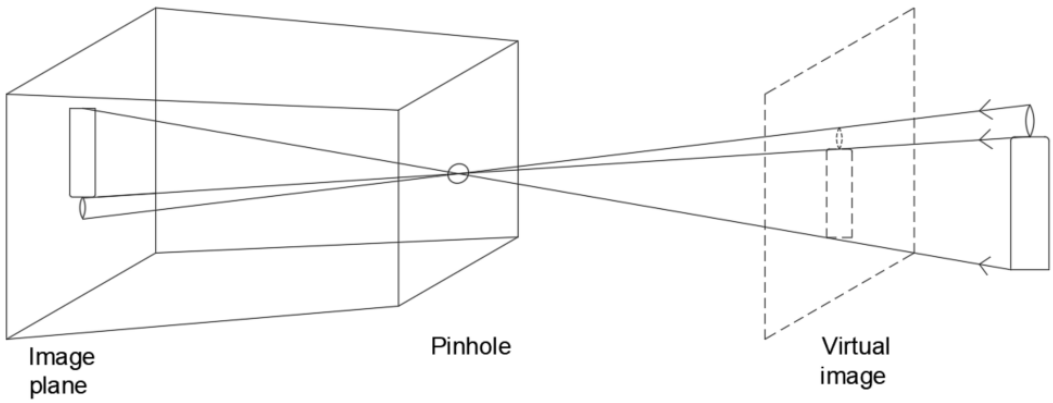

Figure 2. *Image formation in a pinhole camera model [2].*

A virtual plane can be placed at an equal distance from the camera's aperture as the image plane, but in front of the aperture. This way the image is represented in the same direction as the actual object in space (virtual image in Figure 2). The preceding procedure makes matrix calculations easier to compute. In the following section, the image plane is assumed to be in front of the camera.

In an ideal pinhole camera model, it is assumed that the camera coordinate system is aligned with world coordinates, and the camera's centre is at the world coordinate origin. In Figure 3, point X in the 3D space is mapped to point x in the camera's 2D image plane, described by the line going through the camera centre. The distance from the image plane to the camera centre is called the focal length f. The line passing perpendicularly from the camera centre to the image plane is called the principal axis, and it meets the image plane at the principal point p [3].

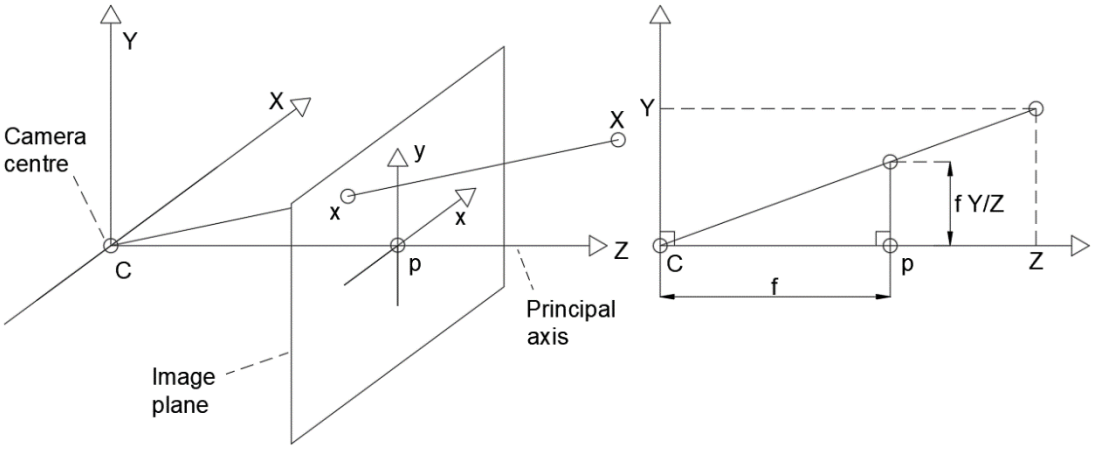

***Figure 3**. Projection in a pinhole camera model [3].*

With two triangles formed, shown on the right side of Figure 3, it is possible to compute the mapping from the 3D space to the 2D image plane. The mapped image point locations x and y can be presented as a relation, which is called mapping between the $\mathbb{R}^3$ Euclidean space and $\mathbb{R}^2$ Euclidean space [3]. In Equation 1, the image point x is marked as $\mathrm{fX/Z}$ and y as $\mathrm{fY/Z}$:

$$(\mathrm{X}, \mathrm{Y}, \mathrm{Z})^{\mathrm{T}} \rightarrow \left(\frac{\mathrm{fX}}{\mathrm{Z}}, \frac{\mathrm{fY}}{\mathrm{Z}}\right)^{\mathrm{T}}. \tag{1}$$

While making this Euclidean mapping from 3D to 2D (also called projective transformation), one dimension and some information is lost. The object's size in the image plane depends on the distance to the camera. In the image, the lines' length and angles are not preserved, but lines' straightness is preserved. By using homogeneous coordinates with transformations, matrix calculations become easier. In homogeneous coordinates, n-dimensional position vectors are presented as (n+1)-dimensional homogeneous vectors [4].

In Equation 2, the image point $\{x_1, x_2\}$ is expressed by multiplying the 3x4 projection matrix by the world point $\{X_1, X_2, X_3\}$ in the homogeneous coordinates:

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix}. \quad (2)$$

Equation 2 can be written in a more compact way, where the camera properties are represented by the camera projection matrix P. This describes the effects of the camera on the projection (Equation 3). The 2D point in the homogeneous coordinates is denoted as vector x, the 3D point in the homogeneous coordinates is denoted as vector X and matrix P is a 3x4 homogeneous projection matrix [3]:

$$x = PX . \quad (3)$$

2.2 Camera parameters

There are two different coordinate systems: camera coordinates and world coordinates. With an ideal pinhole camera, the camera's origin is placed at the world coordinates' origin and the two are aligned with each other. In practice, when a photo of an object is taken from space, there can be rotation and translation between these coordinate systems. It is useful to divide the camera matrix P into two matrices, one describing translation and rotation between coordinates and the other matrix describing the camera's intrinsic parameters:

$$P = K[R|t] = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}, \quad (4)$$

where K is a camera calibration matrix, describing the camera's intrinsic parameters, such as lens distortions, pixel dimensions and focal length. The properties of matrix K depend on the hardware. Different camera models have their own unique parameters. Parameter f in Equation 4 denotes focal length, which is the distance of the image plane to the pinhole. The pixel values p_x and p_y indicate the principal points' transition along the x and y coordinates, respectively, from point $\{0,0\}$ on the image plane. In an ideal pinhole model, the principal point is mapped to coordinate $\{0,0\}$.

The parameters in matrix K are referred to as internal camera parameters. Additionally, focal lengths and principal point transitions along the x and y directions can be presented in pixel coordinates instead of world coordinates:

$$K = \begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}, \quad (5)$$

where parameters $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the focal length coordinates in pixels. K-matrix elements, $x_0 = m_x p_x$ and $y_0 = m_y p_y$ represent the principal points' transition in pixels. In some cases a skew parameter s is added to the element in row 1 column 2 in matrix K, denoting that the axes of the image plane are not perpendicular [5], [3]. Extrinsic parameters due to the coordinate system translation and rotation are expressed by translation vector t and rotation matrix R, respectively. In Equation 6, the rotation matrix R element values α, β and γ indicate rotation around the Z, Y and X axes, respectively [5]. From the rotation matrix R values, it can be interpreted in which direction the camera is pointing in the real world:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) =$$

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \quad (6)$$

The translation vector t expresses the movement of the camera and world coordinates' origins with respect to each other. The homogeneous transformation matrix containing rotation and translation (Figure 4) is also used in robotics, when estimating the translation and rotation of the robot end effectors from the robot's world coordinate origin, which is usually located on the robot's mounting plane.

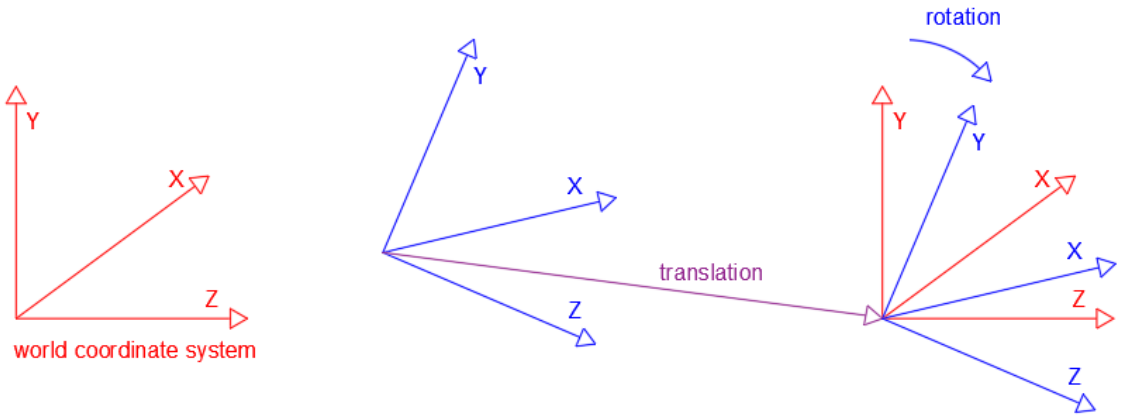

Figure 4. *Translation t and rotation R in world coordinate system.*

Finally, the mapping of the point $\{X_1, X_2, X_3\}$ in the 3D world to the camera's 2D image plane $\{x_1, x_2\}$ in homogeneous coordinates becomes:

$$\begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ 1 \end{bmatrix}. \quad (7)$$

A pinhole camera model approximates an actual camera, and the model can be used to describe real cameras' properties. The procedure to define a camera's intrinsic and extrinsic parameters is called camera calibration [6].

2.3 Distortions

In practice, an ideal pinhole camera model does not take into account light ray distortions caused by the lens. In the model it is assumed that the lens is attached perfectly to the camera. When a lens gathers light coming from the environment, radial and tangential distortions may occur. Radial distortion can be divided into barrel and pincushion distortion.

Barrel distortion distracts pixels in the image away from the image's centre, while pincushion distortion moves the object's pixels towards the image centre (Figure 5). These effects cause straight lines to look bent, and this phenomenon is proportional to the distance from the image origin. At the outer edges of the image, straight lines look like they are more bent. Corrected pixel values can be calculated with the following formulas [3]:

$$\hat{x} = x_c + L(r)(x - x_c) \quad (8)$$

$$\hat{y} = y_c + L(r)(y - y_c) \quad (9)$$

$$r^2 = (x - x_c)^2 + (y - y_c)^2, \quad (10)$$

where $\{\hat{x}, \hat{y}\}$ denote corrected pixel values and $\{x, y\}$ denote original distorted pixel values. Point $\{x_c, y_c\}$ is the centre of radial distortion, which is usually assumed to be at the principal point. The distortion function $L(r)$ is dependent on the radial distance r. Parameter r indicates the radial distance of distorted values x and y from the centre of radial distortion [3], [6]. Another disturbance type is tangential distortion, which can occur in situations when the camera lens is not parallel to the image plane [4]. This can cause the image to look tilted, with some parts of the object closer to the camera than others.

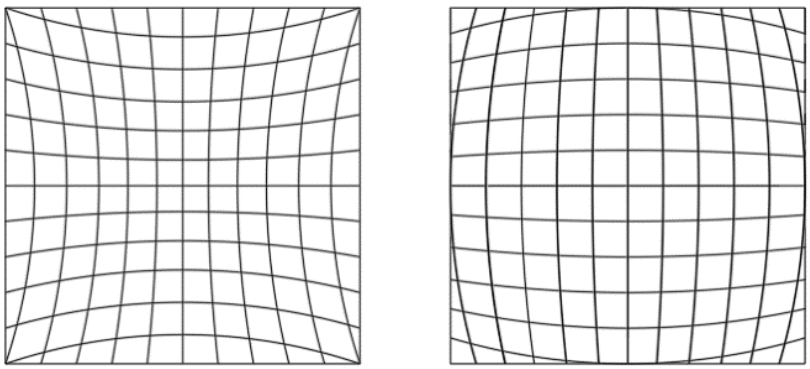

Figure 5. *Pincushion and barrel distortions* [6].

2.4 Machine vision camera

The first stage in this thesis work was to acquire an image dataset, consisting of several hundred images. This was done with a specially constructed machine vision system, comprising a conveyor belt, machine vision camera and a lighting arrangement. In this section, the basic properties of the vision system and optics are covered.

When choosing a machine vision camera, two commonly used types available are line scan and area scan cameras. The area scan camera captures a square form area from a scene, and a matrix of pixels constitutes an image. Area scan cameras are the most popular, and these types are suitable for situations when the target is stationary or the whole object fits in the camera's field of view at one time. When imaging non-moving objects using an area camera, system implementation is usually easier compared to a line scan camera [7], [8].

Images are formulated in a different way in a line scan camera, where the sensor consists of a single row of pixels. Multiple pixel lines are captured from the target, and a two-dimensional image can be formed by combining these rows of pixels. To be able to create a 2D image, either the target or the camera must move with respect to one other. In cases when the observed target at a scene is constantly moving in some direction, or when the whole object does not fit into the camera's field of view at one time, a line scan camera may be more suitable.

Line scan cameras can be used in high-speed applications and camera type resolution is determined by means of the camera's scan rate. Compared to area scan cameras, higher resolution can be achieved by using the line scan technique. A typical application of a line scan camera is the inspection of a rolling cylinder. Also, line scan cameras can be installed to observe a continuous material flow [8].

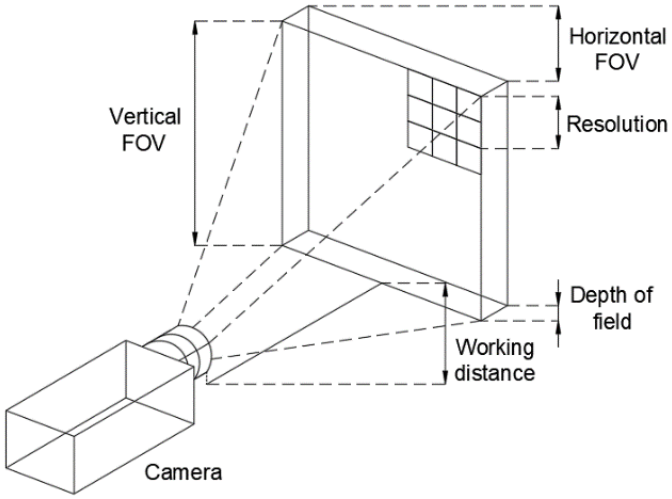

Figure 6. *Machine vision system consisting of optical parameters* [9].

A camera's field of view (FOV) determines the area which can be seen by a vision system (Figures 6, 7). The area marked 1 illustrates the size of an object in the scene. Rotated and translated, the same object is shown as 2, describing the object's possible movement while imaging. When the size of the object and possible movement have been taken into account, a dotted square marked 3 in Figure 7, can be drawn. There must also be some margins to ensure that objects remain in the field of view, when considering image processing and camera installation. The area marked 4 indicates the necessary field of view. After taking the camera's aspect ratio into account, the FOV is determined as 5 in Figure 7 [7].

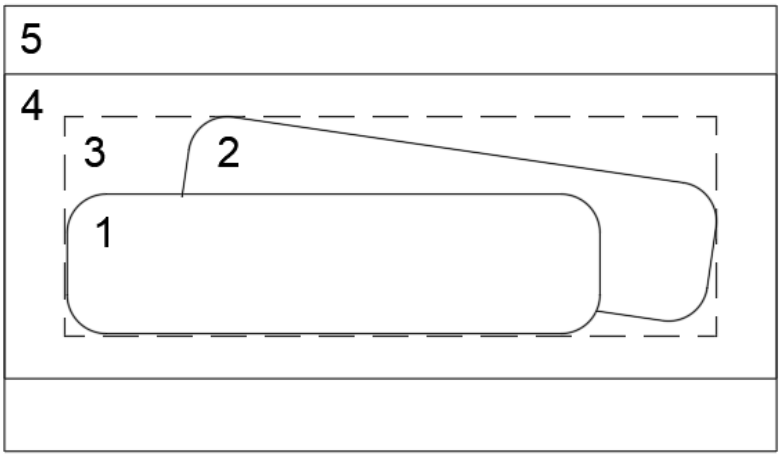

Figure 7. *Determining a camera's field of view* [7].

There is a relation between the camera's focal length and field of view. Using a shorter focal length gives a wider FOV, but objects appear shorter in the image plane. A longer focal length enables bigger magnification, and in this case the field of view becomes narrower. The distance between the camera's lens and the object in the environment is determined as the working distance (WD) [8].

The next stage in image acquisition is to define the proper camera resolution. Regarding resolution, three concepts can be differentiated (Table 1). Camera sensor resolution is the number of pixels in a physical image sensor. This value is expressed in pixels. Spatial resolution refers to how an object in the scene is mapped to a camera's image sensor and the value is given in mm/pixel. Spatial resolution depends on the camera's resolution and field of view. In cases when the image sensor's pixels are not square, spatial resolution must be calculated along the x and y directions differently. Measurement accuracy is the size of the smallest feature which can be observed. When determining adequate resolution, the number of pixels necessary to present this smallest feature must also be examined. The system's total measuring accuracy depends on the spatial resolution, the software algorithms used and the image contrast. It is easier to spot features needed for recognition from high contrast images [7].

Table 1. *Parameters used to calculate resolution for a camera application* [7]*:*

Name	Variable	Unit
Camera resolution	Rc	pixel
Spatial resolution	Rs	mm/pixel
Field of view	FOV	mm
Size of the smallest feature	Sf	mm
Number of pixels to map the smallest feature	Nf	pixel

When the smallest feature size and the number of pixels needed to map the smallest desired feature are known, spatial resolution can be defined by:

$$\mathrm{Rs} = \frac{\mathrm{Sf}}{\mathrm{Nf}}. \tag{11}$$

The relation between the field of view (FOV) and spatial resolution (Rs) defines the camera resolution, which must be calculated along horizontal and vertical directions:

$$\mathrm{Rc} = \frac{\mathrm{FOV}}{\mathrm{Rs}} = \mathrm{FOV} \times \frac{\mathrm{Nf}}{\mathrm{Sf}}. \tag{12}$$

2.5 Optics

The amount of light passing through the lens to an image plane is controlled by the camera's aperture. A larger aperture size allows more light rays to travel to the sensor; this enables the camera to be used in low lighting conditions. Conversely, in bright lighting, a smaller aperture can be applied. The aperture's functionality can be compared to the pupil in the human eye. The aperture size is defined as the $\mathrm{f-number}$, also called the $\mathrm{f-stop}$.

The $\mathrm{f-number}$ is defined as $\mathrm{f/d}$, where f means the focal length and d the diameter of the aperture. The value of the $\mathrm{f-number}$ is inversely proportional to the aperture size: a lower $\mathrm{f-number}$ indicates a larger aperture. The relation between the $\mathrm{f-number}$ and the focal length f is usually presented as the $\mathrm{f/f-number}$, as shown in Figure 8. When the focal length value is divided by the $\mathrm{f-number}$, the calculation gives the aperture diameter [10].

Another method for controlling brightness is varying the exposure time. The shutter speed (exposure time) is related to the amount of time that the camera's sensor is exposed to light rays. While imaging high speed objects in a scene, a fast shutter speed is required to avoid blurry images. Shutter speed is usually given in fractions of a second (e.g. 1/125 s, 1/250 s).

As the aperture's area increases by a factor of 2, the ratio between the focal length and the aperture diameter, the $\mathrm{f-number}$, increases by a factor of $\sqrt{2}$. The same amount of light can be given to an image plane by multiplying the exposure time by 2 [10].

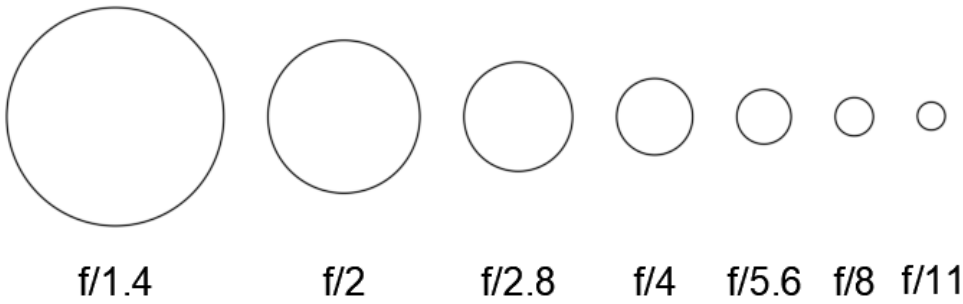

Figure 8. *Effect of increasing the f-number on the aperture size.*

A camera's ability to focus on a target in multiple distances must be considered, and this is defined as the depth of field (DOF). This parameter indicates how much the working distance may increase or decrease while still able to produce a focused image without blur. The depth of field is affected by the values of focal length, aperture size and working distance. A narrow depth of field can be used in situations when focusing on a certain

object in the scene, and when background information is not essential. When the aperture size becomes smaller, the value of the $\mathrm{f-number}$ is increased, and the depth of field becomes wider. Also, by changing the size of the focal length, the DOF can be controlled. By decreasing the focal length, the depth of field becomes wider. A long depth of field can be used, for example, when taking an image of scenery. The working distance also affects the camera's DOF: when imaging a target at further away, the depth of field becomes wider [8].

There are several geometrically formed lens types available, and a camera lens can be a combination of different lens types. In most cases, the lens is spherically curved. When the lens surface is bent toward the centre, it is called a concave lens. When the lens surface is bent away from the lens centre, the lens is called convex [11].

Basic image formation can be modelled by a thin converging lens, where thin means that the lens's focal length is considerably greater than its width. In a converging lens, the light rays viewed parallel with respect to the optical axis towards the lens, are bent through the lens. The light rays are converged behind the lens, on the focal point (beam 2 in Figure 9) [6].

Also, light which is pointed through a focal point comes out of the lens parallel to the optical axis (beam 3 in Figure 9). The centre of the lens is called the optical centre while the distance from the focal point to the optical centre is defined as the focal length [6]. By means of the thin lens equation (Equation 14), the focal length can be calculated. In Equation 14, g represents the object's distance, and b the image plane's distance from the lens centre along the optical axis. This model is an approximation of light refraction from the centre of the lens, and the model can be used in automated inspection tasks [6]. Image formation in the case of a biconvex, converging lens is presented in Figure 9.

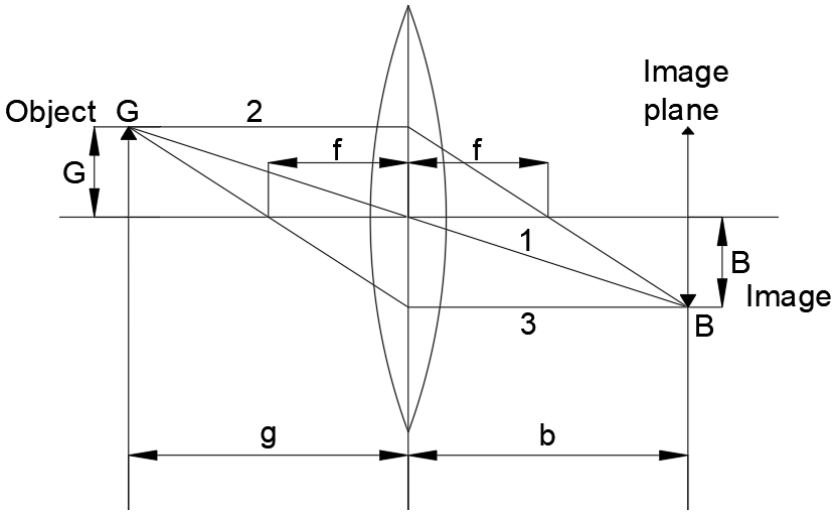

***Figure 9**. Image formation with a thin lens* [6].

A light ray reflected from the object and passing through the lens at its optical centre goes through without changing direction (beam 1 in Figure 9). This light ray is not refracted and is called the principal ray. The relation between the object's distance in space and the image's distance from the optical centre can be written as [6]:

$$\frac{G}{g} = -\frac{B}{b}\ , \tag{13}$$

where G represents the object point's height and g the distance to the middle of the lens along the optical axis. The image point's height is indicated as B and the distance to the lens centre as b. When using the thin lens equation, certain sign conventions must be taken into account. Positive values are given in the following conditions [6]:

- The object's horizontal distance g to the principal point is on the left side of the lens.
- The object point G is located vertically above the optical axis.
- The image's horizontal distance b to the principal point is located on the right side of the lens.
- The image point B is located vertically above the optical axis.
- The image-side focal point is located on the right side of the lens.

As the positions of the image and the object with respect to the lens are taken into account with the sign rules above, the relation between the focal length f, the object's distance g and the image's distance b from the lens centre can be written as follows:

$$\frac{1}{f} = \frac{1}{g} + \frac{1}{b}\ . \tag{14}$$

The magnification of the lens system V is written as follows [6]:

$$V = \frac{B}{G} = -\frac{b}{g} = -\frac{f}{g-f}\ . \tag{15}$$

Greater magnification can be achieved when the object's distance g remains constant with respect to the lens with longer focal length f values.

3. MACHINE LEARNING

At the beginning of this chapter, different types of machine learning are presented. This is followed by a brief introduction to neural networks, and to the procedure used in network training. Also, the VGG16 neural network model and transfer learning method are introduced. At the end of the chapter, data augmentation and the performance metrics used in deep learning are described. Machine learning can be categorized into different types, depending on the task to be solved. The types of learning are supervised learning, unsupervised learning, semi-supervised learning and reinforced learning.

3.1 Types of machine learning

The most frequently used type of machine learning is supervised learning [12]. In this type of learning, the input datapoints are labelled as belonging to a certain category, for example, the CIFAR-10 dataset consisting of 60000 32x32 pixel colour images [13]. Dataset images are categorised into 10 classes, with 6000 images per class. One class contains only images of cats, dogs, cars, ships etc. An algorithm is trained with data consisting of input images with known labels. The model is required to learn patterns from the images representing different classes. Eventually, in the test phase, an unlabelled image is fed as a model input. The algorithm is required to correctly predict the class of the unlabelled input image.

The input data can be expressed in a vector format, such as feature vector x. A learning algorithm's output Y can be expressed as the function $\mathrm{Y} = \mathrm{f}(\mathrm{x})$. This function can be improved by updating the model parameters in the training phase to find the best solution [14]. Supervised learning can be divided into classification or regression tasks. In a classification problem, all input datapoints are labelled in advance as belonging to a certain discrete category [14]. This is the case with the previous example concerning the CIFAR-10 dataset, where 60000 images are labelled as belonging to different classes. Typical classification algorithms are k-nearest neighbours, support vector machines (SVM), logistic regression, decision trees and naive bayes.

Another type of supervised learning problem is regression. In this case, an algorithm tries to predict a numerical output for a continuous variable, using given input variables. An example of a regression problem would be estimating the price of a vehicle. The data, which is given to the model, can be the year of construction and the number of kilometres driven. In a classification task, an input is predicted to belong to a certain discrete class.

In the case of a regression problem, the algorithm gives a numerical estimate of the output, based on the input values [15].

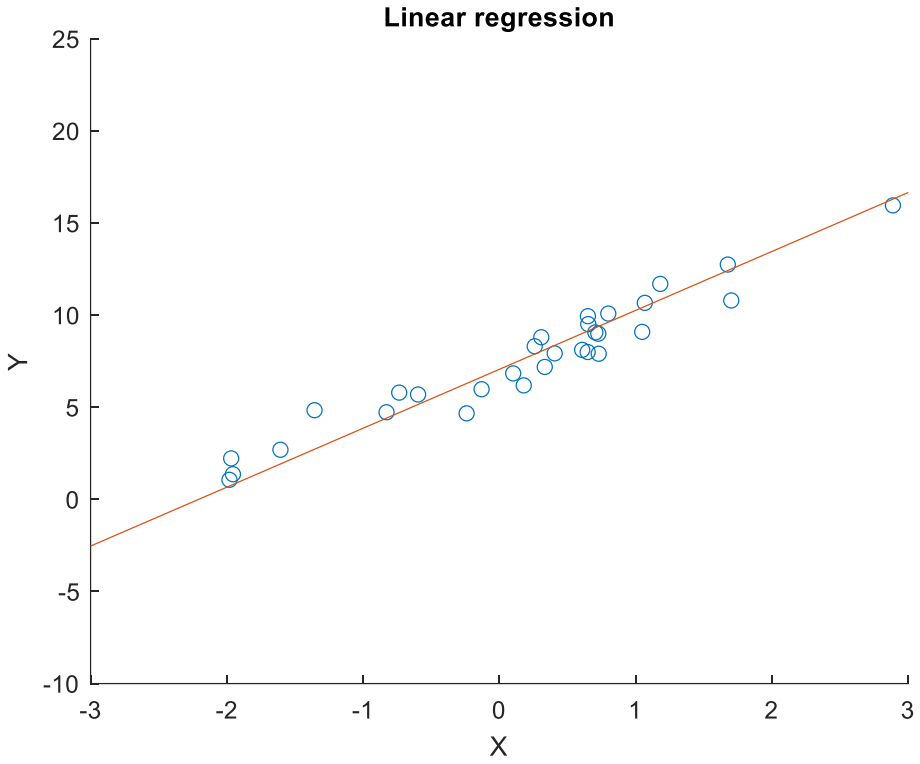

Figure 10. *Linear regression tries to predict Y values, with given values of X* [16].

In a basic linear regression model, the relation between the input variable vector $\mathrm{x} \in \mathbb{R}^{\mathrm{n}}$ and output scalar value $\mathrm{y} \in \mathbb{R}$ is assumed to be linear (Figure 10). Functions that predict the output are marked as $\hat{\mathrm{y}}$ and the parameter vector is defined as $\mathrm{w} \in \mathbb{R}^{\mathrm{n}}$. After adding a bias parameter b, the linear regression model can be written as:

$$\hat{\mathrm{y}} = \mathrm{w}^{\mathrm{T}}\mathrm{x} + \mathrm{b} \, . \tag{16}$$

Every element in a vector w is multiplied by the rows of vector x and summed up. The values of vector w are called weights. A greater magnitude of a certain weight has a bigger effect on the prediction [15]. This linear model concept is also applied in neural networks and discussed in the following sections.

After having estimated n pieces of $\hat{\mathrm{y}}$ training samples, a procedure to measure model efficiency can be performed. In the case of a supervised learning problem, correct values for each training samples are given, marked as vectors y^{GT}. Model performance can be measured by comparing the difference between correct values and predicted values with

an error function. This function is also known as a cost function. Mean squared error, expressed as:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i^{GT})^2, \tag{17}$$

is one of the loss functions used, which measures the Euclidean distance between the estimated and ground truth values [15].

In unsupervised learning, only data samples are given to the algorithm, without knowledge of the correct output labels. While having only an input vector x and no corresponding true values, the learning algorithm is assumed to be able to find associations from the data structure by itself.

In general, when the quantity of features (dimensions) in the data is increased, the efficiency of a learning algorithm decreases. The technique that aims to prevent the phenomenon is called a dimension reduction. A typical feature reduction technique is the unsupervised method known as principal component analysis (PCA). In PCA, data is converted to lower dimensions, to visualise the dimension with the most variance in the data structure [17].

In clustering, the aim is to discover similarities from the input data. By means of these same properties, data can be divided into groups, called clusters [18].

The reduction of the data dimensionality usually takes place before clustering [19]. In K-means clustering, data is divided into k clusters. An example of a clustering-algorithms output is illustrated in Figure 11. Another used unsupervised algorithm is the self-organising map (SOM), which can be used for dimension reduction.

Semi-supervised learning (SSL) is a combination of supervised and unsupervised learning. Only a certain amount of data is labelled, as in supervised learning. Labelling of data in advance can be expensive and the data can be hard to obtain, especially when the number of examples is extensive. A vast amount of unlabelled data is faster and cheaper to obtain. Semi-supervised leaning can be utilised in learning tasks such as speech recognition or classification. Because the unlabelled part of the data contains less information, a lot of unlabelled data is needed to improve SSL algorithm accuracy [20].

In reinforced learning, an agent explores its surroundings with actions. The goal is to maximise a reward signal while moving in a defined space called an environment. No instructions are given in advance to the agent on how to solve the problem. The agent is assumed to be able to learn, based on feedback from the performed actions. After performing a correct action, a reward is received. This reward can be achieved by one action, or by combinations of several actions.

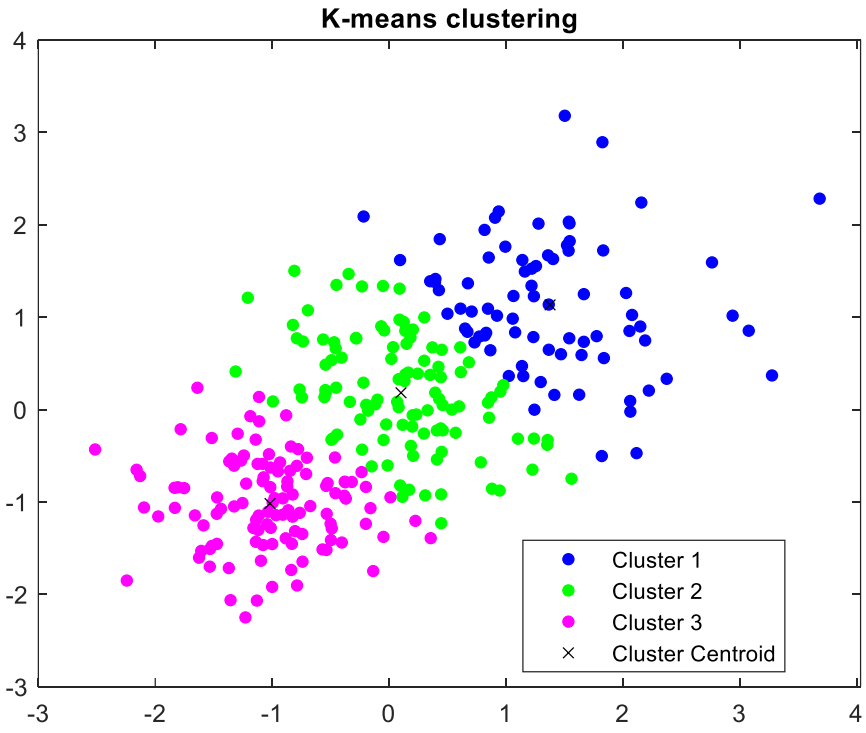

Figure 11. *Data divided into three groups* [21].

A reinforced learning system can be referred to as a trial-and-error type of learning. The system has the ability to gain a reward based on the correct actions [22]. The agent's current state defines which actions must be made to gain the maximum reward. Cases when the performed actions do not help to receive a reward will give negative feedback to the agent. When repeating multiple actions in various states, the agent is assumed to be able to learn. These series of actions can constitute the maximal reward.

One challenge regarding reinforced learning is how to exploit a past successful experience to gain new rewards. New explorations must be made to receive more knowledge and better results. Balancing between the exploration of an environment and exploitation of already received experiences must be made to find the optimal solution [22]. As in unsupervised learning problems, there are no correct answers given to a reinforcement algorithm. Nevertheless, these learning concepts differ from each other. The unsupervised approach aims to predict the structure of the data, whereas a reinforced learning algorithm tries to achieve the maximal reward [22].

An example of how to solve problems with reinforced leaning is a robot moving in a room comprising corridors and obstacles. The robot receives information from the surroundings and improves its performance while moving around. The goal could be to find an exit door, which is placed in one corner of the room. By gaining new experiences from the surroundings, the robot eventually learns to find a way out.

3.2 Neural networks

Neural networks (NN, and also artificial neural networks ANN) are powerful computational models used in machine learning applications. The first model of a neural network was introduced in 1943 by McCulloch and Pitts. Influenced by their earlier study, Rosenblatt later (1958, 1962) introduced the concept of a perceptron, which enabled a model to learn [15], [23]. The development of a neural network has been inspired by the functionality of nerve cells (neurons) in the human nervous system. A network neuron model can be said to be a very simplified version of a biological neuron [24].

There are three typical structures related to neural networks. Firstly, the number of layers, and the quantity of neurons in each layer. Secondly, the learning system applied to update the weights of the neurons. Finally, different activation functions can be used in a network [25].

A neural network consists of an input layer and an output layer. In between there are hidden layers, sometimes called black boxes. NN layers are fully connected, which means that every neuron is connected to every neuron in the next layer. The strength of each connection is called the weight. When altering the number of layers and the number of neurons in each layer, these modifications affect the model efficiency.

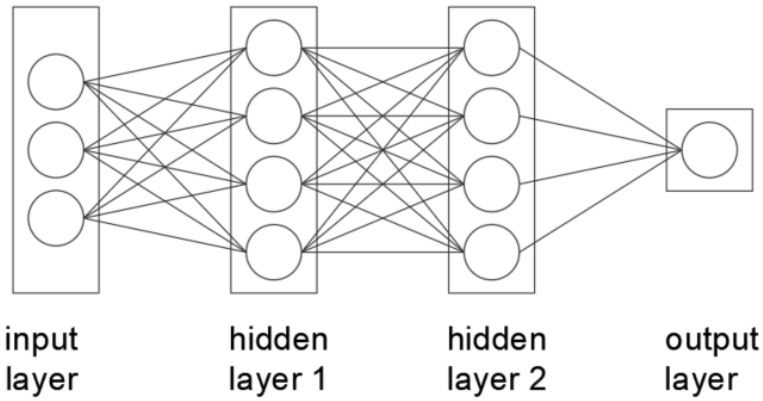

Figure 12. *Fully connected layers. Each circular neuron is connected to every neuron in the following layer* [26].

The NN structure as shown in Figure 12 is a feedforward neural network (multilayer perceptron or MLP). The data, which is fed to the input layer, propagates through the network in only one direction. There are only connections in the forward direction, with no feedback connections to previous layers. A network structure where feedback connections also exist is called a recurrent neural network [15].

A single computational unit in a network layer is a neuron, as illustrated in Figure 13. Neurons are connected to previous layer neurons with inputs. The output value is connected to the input of the next layer's neurons. Every input has its own weight value, and this value corresponds to how much a certain input activates a single neuron. The weight value indicates how important a certain input is with respect to the neuron's output.

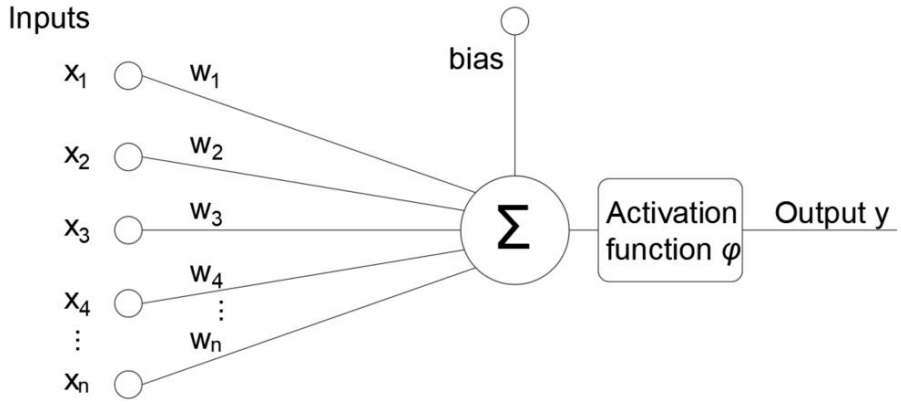

Figure 13. *Model of a single neuron. Input value strength is marked as weight and a bias term is added. After the activation function, output y is formed.*

Inputs x and weights w are presented in a vector format as $X = [x_1, x_2, x_n]$ and $W = [w_1, w_2, w_n]^T$. The sum of dot products between these vectors' elements is calculated, and a single bias term is added to the neuron. By summing a single neuron's weight and bias values, a neuron can be presented as a linear model: $y = \sum w_i * x_i + b$.

In the next phase, a nonlinear activation function is applied to produce a neuron output y:

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) . \quad (18)$$

By adding a nonlinear activation function, a linear model is transformed into nonlinear form. Without this nonlinearity, a whole network of connected neurons would be described as a single linear regression model [27]. An activation function determines how the output of a neuron in produced. It enables a model to be used to learn complex structures.

In Equation 18, the activation function is marked as φ, and w denotes weight values, which are multiplied by input data values x and added with bias b. The number of inputs and corresponding weights w connected to the neuron is represented as index i. When starting to train a multilayer neural network, usually small initial weight values from uniform distribution are randomly selected [28].

There are several activation functions used in feedforward networks. The first perceptron model was introduced by Rosenblatt in 1958, where a single layer neuron output was formulated as a step function. This function can be used only in linear separable binary classification tasks, and the output is determined with a single threshold value. If the threshold value is 0, the activation rule can be written as:

$$f(x) = \begin{cases} 0, \text{if } x \leq 0 \\ 1, \text{if } x > 0 \end{cases}. \tag{19}$$

Nowadays in advanced networks, other subsequent activation functions are applied. The logistic sigmoid function resembles a step function, but it has a wider decision boundary. The structure is like a smoothed step function, as illustrated in Figure 14. Function output values are in the range of [0,1] instead of only 0 or 1. High negative values are converted to 0, and high positive values are converted to 1. A sigmoid is used to predict probabilities in binary classification problems. The SoftMax activation function (Equation 20) is a multiclass generalized version of a sigmoid [14]. It is usually applied to the last layer in a network, with a multiclass classification task. SoftMax computes the probability distribution over different classes [29], [15]:

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}, \tag{20}$$

where x_i is the network's output value for a certain class, and $\sum_j \exp(x_j)$ is the sum of all output values. SoftMax calculates a probability value for a certain class x_i.

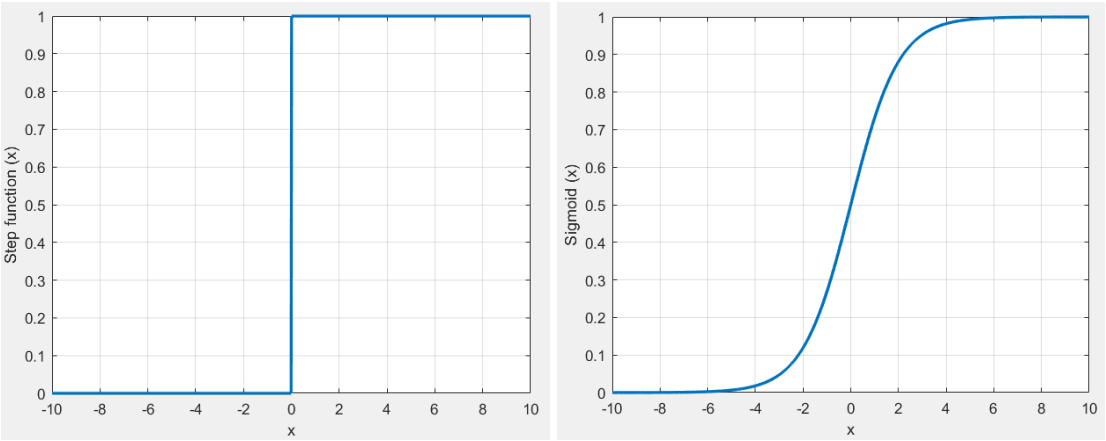

Figure 14. *A step function (Heaviside function) and a logistic sigmoid function.*

The hyperbolic tangent (tanh) activation function is a scaled version of a sigmoid and centred to zero, with the output varying in the range of [-1,1] [23]. The advantage when using a tanh function instead of a sigmoid is a zero-centred output. This improves the backpropagation in the training phase [29]. The tanh activation function is illustrated in Figure 15.

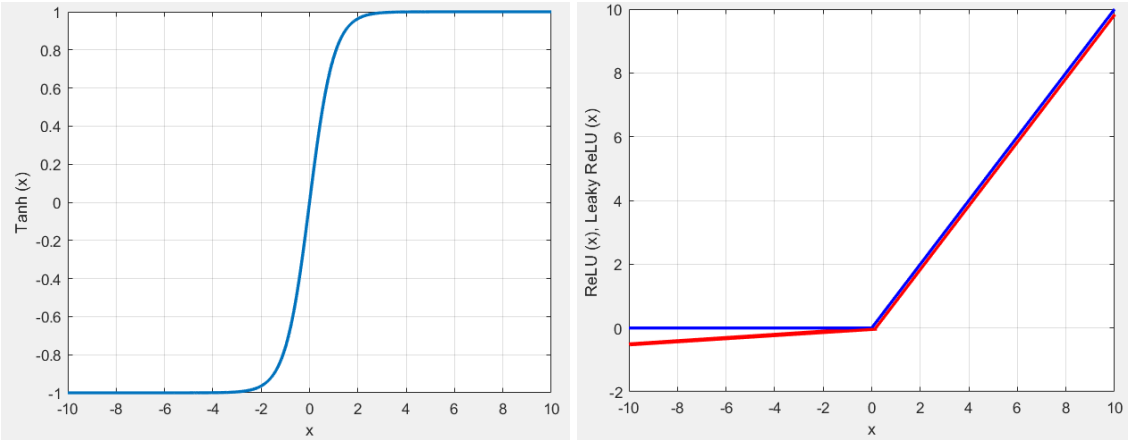

Figure 15. *On the left, an output of the tanh function. On the right side, ReLU (blue) and Leaky ReLU (red) activation functions.*

A rectified linear unit (ReLU) is a commonly used activation function. The output is set to zero with negative input vales, and a positive input results in a linear output. Due to the function's simple structure, it is fast to compute, and the function's performance is high. Rectified linear units have been applied in Alexnet by Krizhevsky et al. in [30]. In their study, the ReLU activation function turned out to be several times faster in training compared to tanh activations.

A vanishing gradient problem is a phenomenon where a gradient value becomes smaller when backpropagation is performed in the training phase (section 3.3). A gradient value decreases in hidden layer neurons, when moving backwards from the output to the input layer. This causes an early layer's neurons to learn more slowly, while gradient values become smaller [23].

One benefit concerning ReLU activation is that the function works better with a vanishing gradient problem while training, as compared for example to sigmoid, where an output is saturated with high positive or negative input values. This sets the gradient values in sigmoid close to zero. ReLU sets positive inputs as a linear output and a gradient exist. With negative values, the neuron is not activated [23]. The ReLU activation function is defined in:

$$\mathrm{ReLU}(x) = \begin{cases} x, \text{if } x > 0 \\ 0, \text{if } x \leq 0 \end{cases}. \tag{21}$$

A modified version of ReLU is the Leaky ReLU, where a low slope is added for input values less than zero (Figure 15). With a traditional ReLU, high negative values are set to zero. This can cause a dying ReLU problem, because of the zero gradient in the training phase. A leaky ReLU also sets a small output for a negative input value [31]. This prevents the whole neuron from dying while training.

3.3 Training of neural networks

After having introduced the basic structures of a neural network, the principles of NN learning are covered in this section. To enable a neural network algorithm to learn, the objective is to minimise the value of a loss function in the training phase. By tuning weight vectors w parameters and biases b, the error between a correct input value y^{GT} and an estimated value $\hat{\mathrm{y}}$ can be minimised. At the beginning of training, the network weights can be initialised to random values. Alternatively, uniquely designed weight initialisation methods can be used to improve network performance [32].

In the next phase, a forward pass to the training data is performed. As a result, the output values for the weights are generated. Then by means of a cost function, the error between a true and an estimated value is calculated. Different cost functions can be applied to calculate the cost. Commonly used cost functions are mean squared error (section 3.1, Equation 17) or mean absolute error, defined as:

$$\mathrm{MAE} = \frac{1}{\mathrm{n}} \sum_{\mathrm{i}=1}^{\mathrm{n}} \left| \hat{\mathrm{y}}_{\mathrm{i}} - \mathrm{y}_{\mathrm{i}}^{\mathrm{GT}} \right| , \tag{22}$$

where y^{GT} is a correct value and $\hat{\mathrm{y}}$ is an estimated value. The index of the test sample is marked as i. Another type of error function is a binary cross-entropy, which is used to solve binary classification tasks. In the case of a multiclass classification task, a categorical cross-entropy error function can be used. The cross-entropy error function is defined in Equation 23 [14], where y^{GT} is a ground truth value and $\hat{\mathrm{y}}$ is an estimated value:

$$\mathrm{CE} = -\sum_{\mathrm{i}=1}^{\mathrm{n}} \{ \mathrm{y}_{\mathrm{i}}^{\mathrm{GT}} \ln \hat{\mathrm{y}}_{\mathrm{i}} + (1 - \mathrm{y}_{\mathrm{i}}^{\mathrm{GT}}) \ln(1 - \hat{\mathrm{y}}_{\mathrm{i}}) \} . \tag{23}$$

Optimisation algorithms are used to minimise the cost function; one commonly used algorithm is a gradient descent (GD). The next procedure is to find the negative gradient of this cost function, which describes how the weights and biases must be changed to minimise the cost function. The gradient is a vector value, indicating the direction where a function grows fastest. The goal is to minimise the gradient of the error function value to zero by taking a small step (learning rate) towards the negative gradient. Backpropagation is a method for calculating these gradients. The partial derivatives with respect to each weight in a training set are calculated. Gradient values are computed from output layers to input layers for each weight, and the chain rule in derivative calculus is utilised.

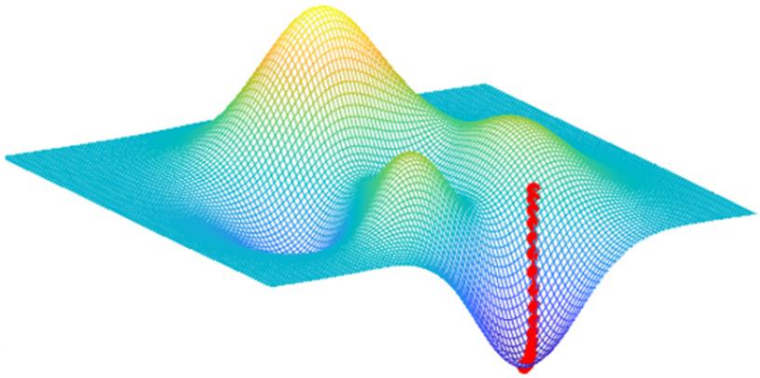

Figure 16. *Optimiser algorithm reaches the global minimum point* [33].

After the gradients are calculated, every weight parameter on each layer in the network is updated according to:

$$w_{i\_new} = w_i - \alpha \frac{dE}{dw_i}, \tag{24}$$

where α denotes the learning rate, indicating the step size in the parameter update procedure, dE/dw_i is the partial derivative of the cost function with respect to the corresponding weight and w_i is the weight parameter to update.

In a batch gradient descent algorithm, all training samples are used to calculate a single weight update. In contrast, in a stochastic gradient descent (SGD) algorithm, only one randomly selected training sample at a time is used to evaluate the cost function and weight update after one iteration. This makes the training process more efficient. Furthermore, when a certain set of training samples are used at a time to update the weights, the algorithm is called a minibatch gradient descent [34].

In Figure 16, the cost function is reduced by a parameter optimisation. This process is repeated until the global minima of the cost function is reached. An optimal learning rate hyperparameter can be selected by tuning this parameter in the training phase. When the learning parameter is set too small, model learning can be slow [15]. By setting the learning rate too high, an algorithm may not reach the minimum point, as the valley bottom shown in Figure 16. Another learning parameter is an epoch. One epoch is performed when all training samples have been used once. Weights are updated and backpropagation is continued until the model error reaches the desired level or another stopping criterion occurs.

3.4 Convolutional neural networks

Early development of a convolutional neural network (CNN) was based on studies by LeCun et al. in [35], where handwritten digits were detected using a backpropagation algorithm. Later LeCun and his colleagues introduced an enhanced method to recognize handwritten digits with CNN [36]. CNN is commonly used in the field of computer vision, in object detection and classification tasks.

The basic structure of NN architecture is also exploited in convolutional neural networks (Figure 17), where one or several fully connected layers are replaced with convolutional layers. A CNN model requires less computational power compared to NN. In a CNN model, only a small part of adjacent layers are fully connected at one time.

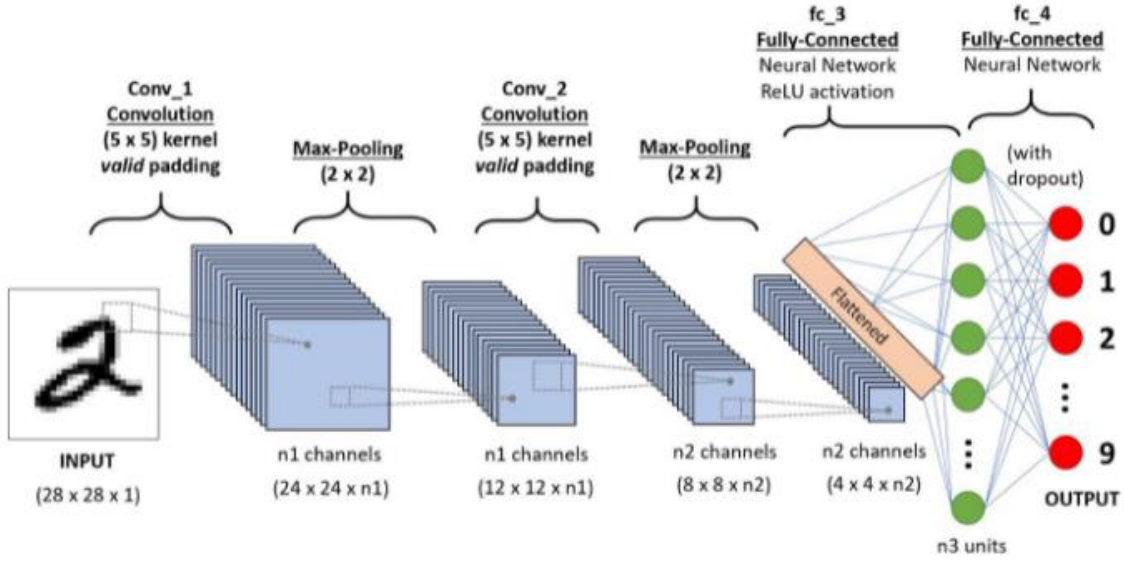

Figure 17. *An example structure of CNN, input image size 28x28x1 (1 colour channel input image)* [37].

A CNN takes an image as input and represents convolutional layer neurons in three dimensions: width, height and depth. A convolutional layer receives values from only a certain region at a time from the previous layer, instead of all pixel values as in the case of a fully connected layer [26].

In a convolutional layer, a two-dimensional surface area from an input image is called the receptive field, which also takes the image's depth dimension into account, for example, a 6x6x3 region where 3 represents the input image's dimensions in 3 RGB channels. All 108 pixel values of this volume are fully connected to each neuron in a convolutional layer [38].

An input image can be interpreted as a matrix containing the pixel values. In the case of an RGB image, 3 matrices, one for each colour channel, are used. A square-shaped receptive field called a kernel slides (convolves) horizontally over the rows of the input volume, until the whole input has been walked through with this window (Figure 18). Simultaneously with element-wise multiplication, the sum of dot products between the kernel and the corresponding input image region is calculated [39]. This operation creates a feature map, also called a channel. For example, the first convolutional layer in VGG16 consists of 64 different feature maps.

Different filters can learn various features from an input image. As we go deeper into the CNN, the features of previous layers are combined in deeper layers. Filters in deeper layers will construct more complex features, learned by the network. Also, the number of filters is increased as we go deeper into the network. The primitive features of the early layers can be presented with fewer filters. Complex features are presented with more filters, deeper in the network.

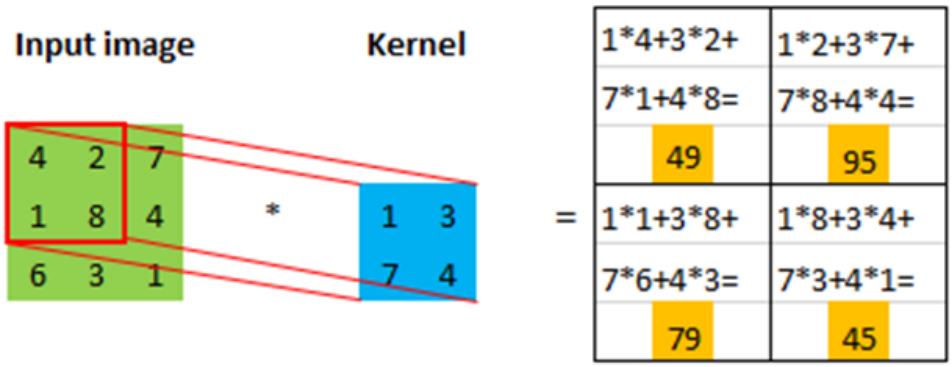

Figure 18. *A kernel slides over one colour channel input image, with a stride of 1.*

There are some hyperparameters which quantify the size of the output volume in a convolution layer: depth, stride and zero padding. By determining a depth value, the number of filters to be used in a convolutional layer can be chosen. Every filter looks for a different kind of feature from an input image. To make CNN more effective, the number of network parameters is reduced with parameter sharing. This is done by keeping the kernel weight values constant within the same feature map [26].

The stride value determines how many pixels at a time a kernel window moves on each slide. The stride value affects the output. Higher stride values produce fewer overlapped kernels while sliding, and a smaller spatial image is formed as output [38].

A stride is used with a padding to adjust the kernel movement along the input. Hyperparameter zero padding is a method to determine the spatial size of the output volume. By setting zero values (Figure 19) around the input volume, the output size can be adjusted [26]. As a kernel movement with a certain stride must fit in the spatial frame, zero values can be added. No zero values are added in a valid padding, and a part of the original image may be cropped. This occurs when the kernel movement does not fit the input size.

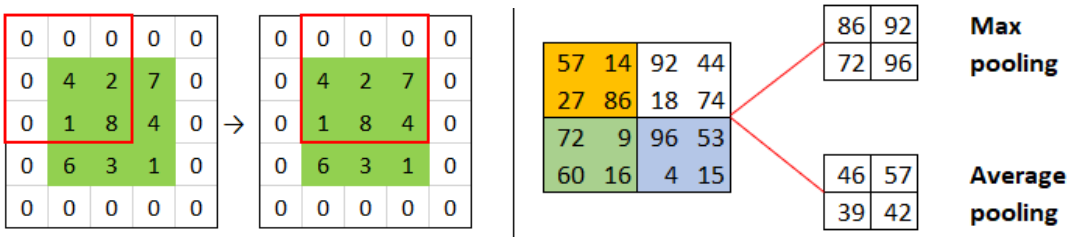

Figure 19. *On the left, a stride of 1 and a zero padding of 1 are used. The kernel is marked as a red area. On the right, max pooling and average pooling operations.*

The basic layer structures in CNN are:

- A pooling layer makes a down-sampling operation and shrinks an image along its height and width. As a result, it produces a down-sampled image, as shown in Figure 19. Average pooling calculates the average value of an input window area. Max pooling calculates the maximum value from the same region.
- In a dropout layer, a set of neurons are dropped from the training phase with a specified probability value (e.g. 0.5). By adding a dropout layer to a network, overfitting can be reduced.
- A rectified linear unit (ReLU) activation function layer converts all negative values to zero and has no effect on the input volume size [26].
- A fully connected layer type is similar to regular NN, in that every neuron in this layer is connected to every output at the previous layer [26]. The classifier part at the end of the network consists of fully connected layers.
- At the final output layer of CNN, the image is represented as a single vector containing scores for every class. Final probability scores for each class can be calculated by using the SoftMax activation function.

3.5 VGG16 model

VGG16 is a deep convolutional neural network introduced in a publication [40] by Karen Simonyan and Andrew Zisserman from the University of Oxford. The model achieved high rankings in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014, achieving over 92 percent top-5 accuracy with a test dataset. The ImageNet classification dataset applied in the ILSVRC contest (2014) included 1000 varied classes with 12.8 million training, 50 000 validation and 100 000 test images [41].

A few different versions of the VGG16 structure are presented in [40], with the main differences between the versions concerning the structure of the convolutional layers. The CNN version, consisting of 16 weight layers was applied in the experiments of this thesis. The network is presented in Figure 20. The network receives an image size of 224x224 pixels as an input, followed by two adjacent convolutional layers. These layers have a small receptive field of 3x3 pixels, and the kernel performs a stride of 1, resulting in 64 channels.

As we go deeper into the network structure, a stack of convolutional layers is followed by a max-pooling layer, which performs spatial pooling over an area of 2x2 pixels operated with a stride value of 2 [42]. Additionally in all the hidden layers, the Rectified Linear Unit (ReLU) is used as the activation function (not visible in Figure 20). The classifier part, consisting of two fully connected layers including 4096 channels, and the final dense layer with 1000 channels are predicted by a SoftMax layer.

In the brick quality inspection experiments, the last fully connected layer size (Figure 20) was modified to 2 or 3, to maintain the brick type class-wise comparison. During the classification tests with Python programming, different-sized input images were also tested. This was enabled by changing the last max-pooling layer into a global average pooling layer.

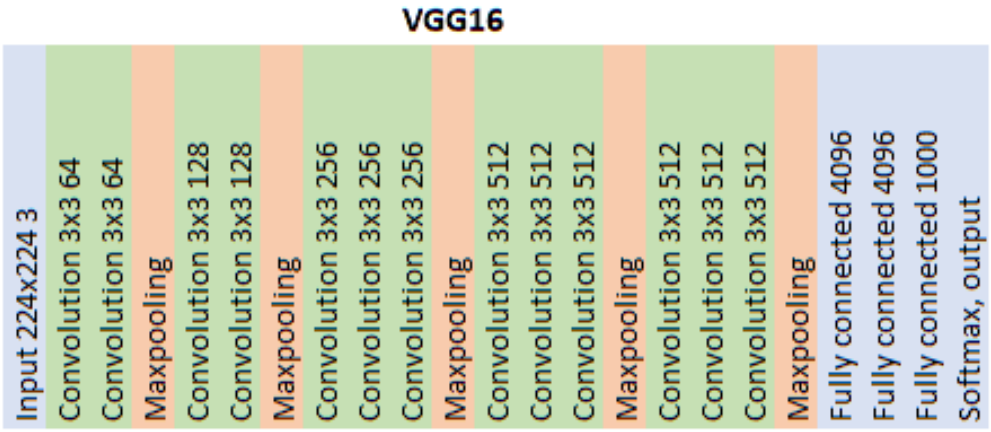

Figure 20. *The layer structure of the VGG16 CNN model* [40].

3.6 Transfer learning

Developing a CNN classifier from scratch can be a challenging task. This may be due to the size of the dataset, computation power or time required to train the model. Transfer learning is a design approach associated with machine learning to overcome these challenges.

There may be a certain task related to supervised learning, for example an image classification with a deep learning model. Additionally, a domain can be defined as the origin of the data e.g. ImageNet. Transfer learning can be used in a situation where the aim is to exploit knowledge gained from a certain task for another similar task in a new domain [43]. In the thesis case, the weights of VGG16 pretrained by ImageNet were exploited in brick quality detection.

The differences between a source and a target domain affects the usability of transferred features. Despite the differences, by using transferred features, the network's generalization performance can be improved as compared to random weight initialization [44], [45]. In this case, the brick data and the ImageNet data domains are not assumed to be very similar. A fine-tuning method was performed to alter the weights in the pretrained VGG16 layers.

CNN early layer filters are closer to the input image, and these weights represent easily interpretable basic features. These feature maps consist of horizontal and vertical edges, lines, and combinations of these added with different colours. As we go deeper into the network, the features become more complex, as combinations of weights learned by the previous layers [46], [47].

Concerning the masonry brick data, the images consist of basic features like blobs, cracks and edges with colour variations. In the early convolutional layers in pretrained VGG16, the layers are assumed to have more valuable information regarding the brick classification task. Complex feature maps in the deeper part of the network are assumed to be less important for brick classification.

The classifier part at the end of the original VGG16 is presumed not to have very usable weights, because of the difference between the training images in the brick data and ImageNet data. The thesis classification experiments were conducted with two scenarios. These were opening only the classifier part of VGG16 for training, and alternatively opening all layers for training with the brick data.

With Python programming, the Keras framework can be applied to develop a deep learning model. In relation to transfer learning, there are several pretrained models available in Keras. Table 2 lists some of these pretrained models.

Top-1 accuracy is used to evaluate classifier performance. This indicates how many test samples are correctly classified out of a certain dataset. Nevertheless, in some cases datasets can have multiple class instances in the same test image, for example, a dog and a cat in the same image. Top-1 accuracy gives only the highest prediction with a single class. Top-5 accuracy represents the top five predictions, which are present in the test image, also revealing multi-class instances from the test image. The classification result is considered to be true if the correct class is included in the top-5 predictions [48]. Top-5 accuracy marked in Table 2, is achieved with the ImageNet dataset.

Table 2. *Examples of deep learning models available in Keras* [49].

Model	Size (MB)	Top-5 Accuracy	Parameters	Depth (Layers)	Time (ms) / inference step (GPU)
MobileNetV2	14	0.901	3,538,984	88	3.83
VGG16	528	0.901	138,357,544	23	4.16
InceptionV3	92	0.937	23,851,784	159	6.86
Xception	88	0.945	22,910,480	126	8.06
InceptionResNetV2	215	0.953	55,873,736	572	10.02

The feature visualization tool used to generate Figure 21 was the Deep Dream technique in the Matlab Deep learning toolbox [50]. The feature maps learned by the VGG16 network are visualised in Figure 21. The CNN was pre-trained with the ImageNet database.

Layers between the input and output layers are referred to as hidden layers. By visualizing feature maps of different layers as plotted images, we can examine what kind of patterns and features each filter in the different layers is looking for. The idea is to produce a certain image, which corresponds to the activation on each feature map.

Using this technique enables the visualization of what kind of features the network has learned to recognise in different layers of CNN. Deeper layers will construct more complex features, learned by the network. Figure 21 is formed by visualising four feature maps, 92-95, in the 4th, 7th, 10th and 13th convolutional layers. A brick image was used as the VGG16 network's input image.

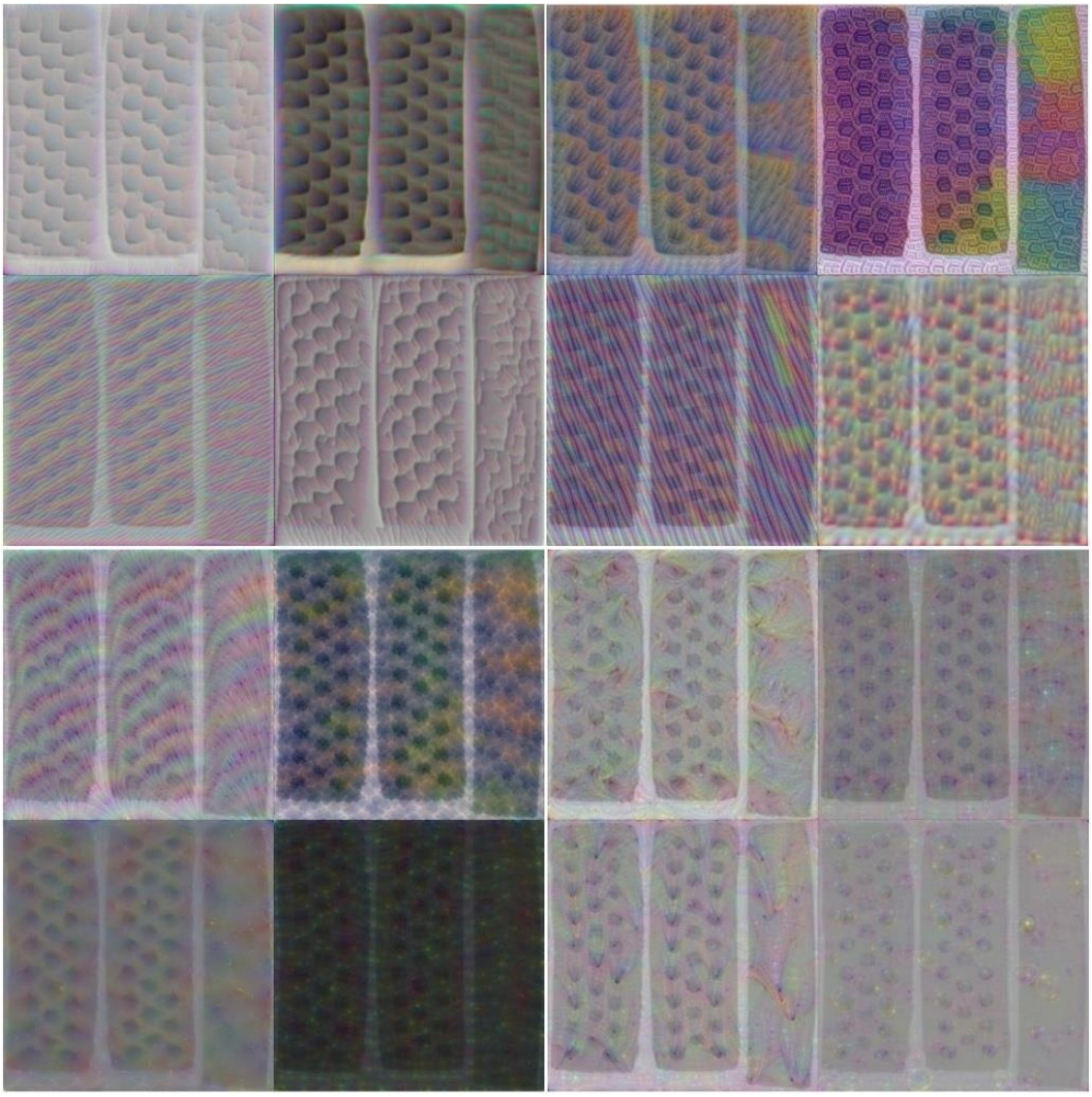

Figure 21. *The features learned by VGG16, feature maps 92-95 in the 4th, 7th, 10th and 13th convolutional layers. From left to right, the top row presents convolutional layers 4 and 7, and the lower row presents layers 10 and 13. Matlab Deep learning toolbox was used to generate this image, representing the activation of different feature maps.*

3.7 Data augmentation

Essential factors when building a CNN model are the quality and quantity of the images in the dataset. On many occasions, the number of images available to be exploited in a learning process is limited. By applying more diverse items to a training dataset, augmentation can improve the classifier's performance with a better generalization ability [51]. Data augmentation is a technique for artificially expanding the size of the training dataset at hand. This is done in an image classification task, by altering the properties of the original images. Slightly varied versions of the original images are added to the training dataset.

Images can be processed for example by flipping horizontally or vertically, translating in certain directions, rotating with various angles, altering a colour channel's RGB values, cropping a part of the image away or alternatively adding some Gaussian noise to the image [52]. The Keras deep learning library contains the *ImageDataGenerator* -class, which has various augmentation parameters available. The augmentation methods used in the thesis tests were rotations (lower row in Figure 22) and flips in horizontal and vertical directions. Rotations in both directions were used only with the original image size. The cropped image size could not be rotated since part of a brick would have been outside the image frame. Neither colour variations by altering the brightness of the brick, or shear range modes were used, to avoid brick misclassifications.

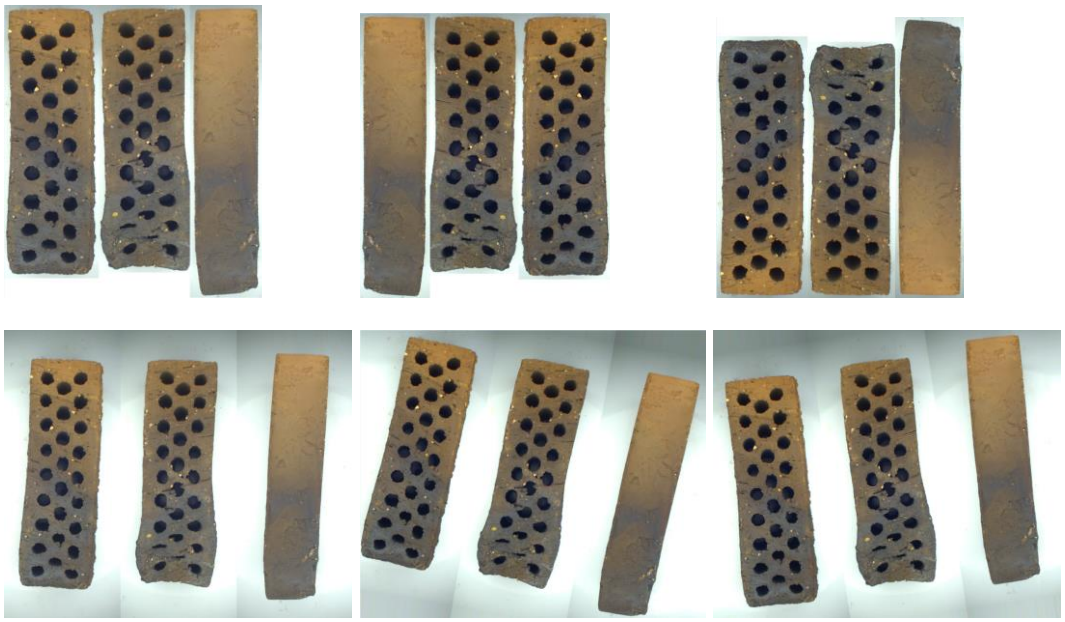

Figure 22. *Top row, a cropped image augmented with horizontal and vertical flips. Below, the same brick with the original image size, rotations applied.*

3.8 Performance metrics and overfitting

Model accuracy is a frequently used metric for evaluating the performance of a learning algorithm. Accuracy indicates the ratio between the correct predictions and all predictions, calculated as:

$$\text{Accuracy} = \frac{\text{True Positive+True Negative}}{\text{Number of predictions}} . \quad (25)$$

Precision, recall and $\text{F1} - \text{score}$ are other estimation metrics which can be exploited regarding machine learning model evaluation. Depending on the machine learning task, a certain kind of error type may become more important than others.

An example of a confusion matrix in a binary classification task is presented in Figure 23. Predicted labels are marked as the matrix columns, and the rows represent the actual labels. Correctly classified observations are indicated in dark green on the confusion matrix diagonal. In a situation where the classifier output is positive, and the actual label is negative, the prediction is defined as a false positive. Similarly, when the classifier predicts the output as negative with a true positive observation, the prediction is a false negative [53].

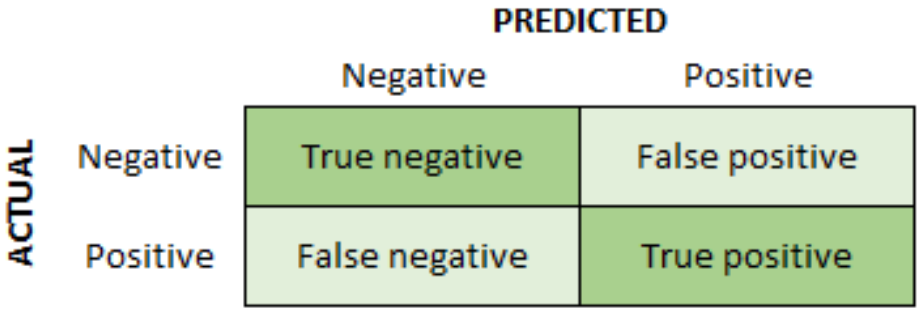

Figure 23. *A confusion matrix in a binary classification task.*

Precision is a metric revealing the ratio of how many true positive predictions are estimated out of all positive predictions:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive+False Positive}} . \quad (26)$$

Precision is useful in situations where a false positive prediction is harmful. An example of a false positive is when a patient is examined for a disease and gets a positive test result although the person is healthy.

Recall (also, true positive rate or TPR) is used to indicate the ratio of true positives compared to actual positive and false negative predictions:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive+False Negative}}. \tag{27}$$

In occasions where a false negative prediction must be avoided, recall is a useful metric. As in the patient health examination example, a false negative occurs when a patient receives a negative test result, although they still have a disease.

Precision and recall can be combined into one performance metric: the $\text{F1} - \text{score}$ which indicates the harmonic mean of these metrics. The $\text{F1} - \text{score}$ can be calculated as follows:

$$\text{F1} - \text{score} = 2 * \frac{\text{Precision}*\text{Recall}}{\text{Precision+Recall}}. \tag{28}$$

In situations when either the precision or recall value is more important, the $\text{F}\beta - \text{score}$ can be used. If setting a β-value of 1, the metric is used as the $\text{F1} - \text{score}$ (Equation 28). When using a lower β-value (e.g. F0.5), precision is considered more important. Alternatively, with a higher β-value such as F2, recall is given more weight [54]. The $\text{F}\beta - \text{score}$ is formulated as follows:

$$\text{F}\beta - \text{score} = (1 + \beta^2) * \frac{\text{Precision}*\text{Recall}}{(\beta^2*\text{Precision})+\text{Recall}}. \tag{29}$$

Overfitting is a problem related to supervised machine learning. When considering a classification task, a model may obtain good performance in the training phase. Conversely, when the same model fails to classify unseen datapoints correctly in the test phase, overfitting may have occurred. An overfitted model is too complex and learns details from the training data. The model also learns noise, instead of only the essential signal. An overfitted model suffers from an inability to generalize, which leads to poor performance with the new datapoints in the test phase. The objective in the training of a machine learning model is to achieve a model with a good generalization ability. In the case of a classification task, good generalization enables the model to classify correctly unseen datapoints in a test set also. Procedures to avoid overfitting in machine learning include cross-validation, regularization and the usage of dropout layers. Also, adding more datapoints to the dataset and an early stopping of training before the model overfits, can be used to reduce overfitting [55], [56].

4. TEST IMPLEMENTATION

The objective of this thesis was to study the feasibility of a machine learning application in brick quality inspection. A brick manufacturing company were interested in improving the detection of quality differences in various brick types. If a suitable detection method could be found, the manufacturer would also be interested in increasing the degree of automation in the production. The plan was to implement quality inspection with a machine vision system and detect defects from the images of the bricks. A convolutional neural network was chosen as the most suitable machine learning method to evaluate images collected by automated image capturing.

4.1 Initial steps to gather data

The first goal was to collect a uniform, high-quality dataset of images. It was desirable that the lighting conditions should remain constant during the imaging of multiple bricks.

A visit to the brick manufacturer production facilities was made to start planning the image data acquisition. The conditions at the production location did not allow the collection of a set of high-quality images. This conclusion was drawn because there was no proper conveyor belt where a machine vision camera could be installed. Other options for suitable data collection implementation were then studied.

One option was to build an imaging box, where all the bricks would be placed one by one for shooting. This approach would enable uniform lighting conditions during the imaging. The downside of this system was that it would be very time-consuming. All bricks had to be imaged from three different directions and changing the position of the bricks and triggering the camera would have to be done manually.

The second option to collect images was to build a test arrangement in a laboratory and transport a set of bricks there for imaging. This approach turned out to be the best option. Suitable test equipment was found at the Satakunta University of Applied Sciences (SAMK) RoboAI laboratory, where the brick imaging was performed.

After choosing the data collection equipment and the location, the next thing to decide was the brick type, and the quantity of bricks to be transported for imaging. There were over 10 brick colour options with different sizes, and additionally there were five options for the bricks' outer surface. The outer surface (which is visible after masonry) can vary from smooth to rough with different variations. Dark brown bricks with dimensions of

285x85x60 mm (WDH) were chosen for imaging; the weight of each brick was about 2.1 kg.

In image classification tasks implemented with CNNs, there is usually a vast number of images in the datasets. For example, in the CIFAR-10 dataset, there are 6000 images in each class. Choosing the number of bricks for the examination was a challenging task, because there had to be enough imaging data for the learning algorithm. Another aspect was, that all bricks would be manually processed on the conveyor belt three times. This made restrictions for the number of bricks to be imaged. In the case of too few gathered test samples, data augmentation was planned for the collected images. Augmentation can be used to artificially generate more images by rotating, flipping and zooming the original images.

Bricks are sorted into three different categories in production: class 1 represents only good quality items and class 2 contains bricks having minor defects. Class 3 bricks have such major defects that these low category products are not used, and the items are sorted for recycling. Class 2 category products can be used for masonry but are sold at a lower price. Different class images are presented in Figures 24, 25 and 26.

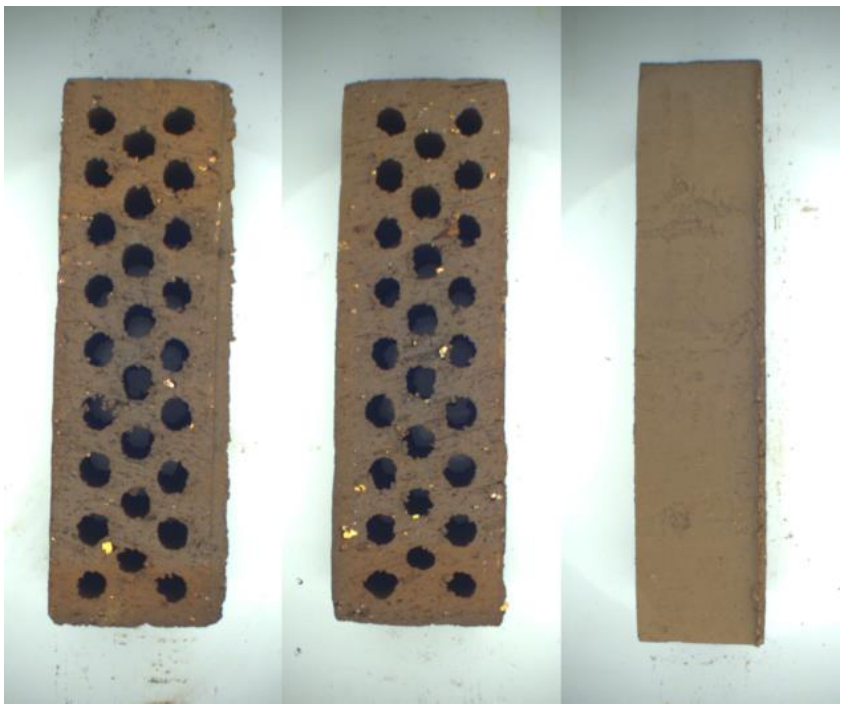

Figure 24. *A class 1 brick imaged from top, bottom and outer side surface.*

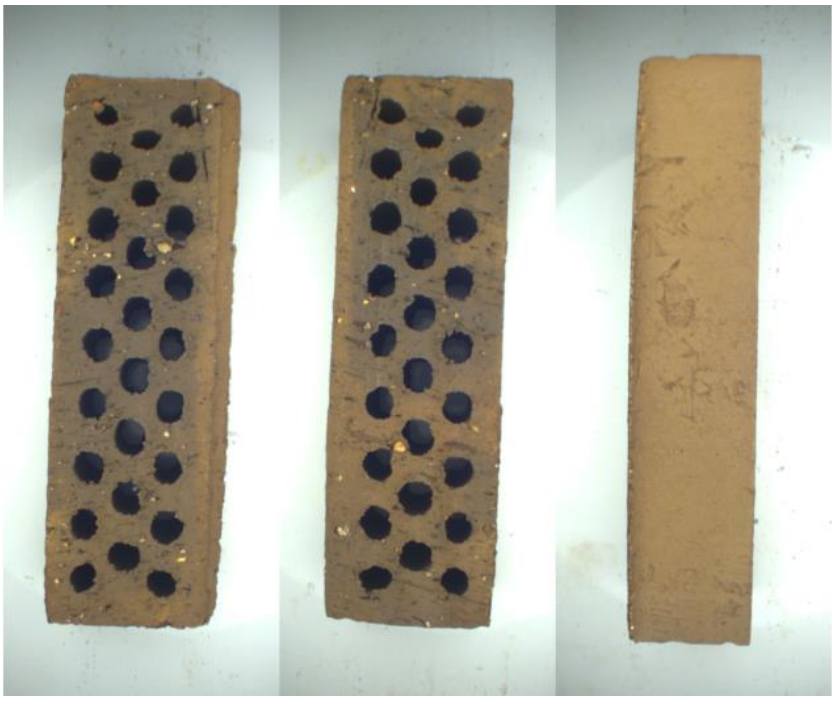

Figure 25. *Class 2 brick.*

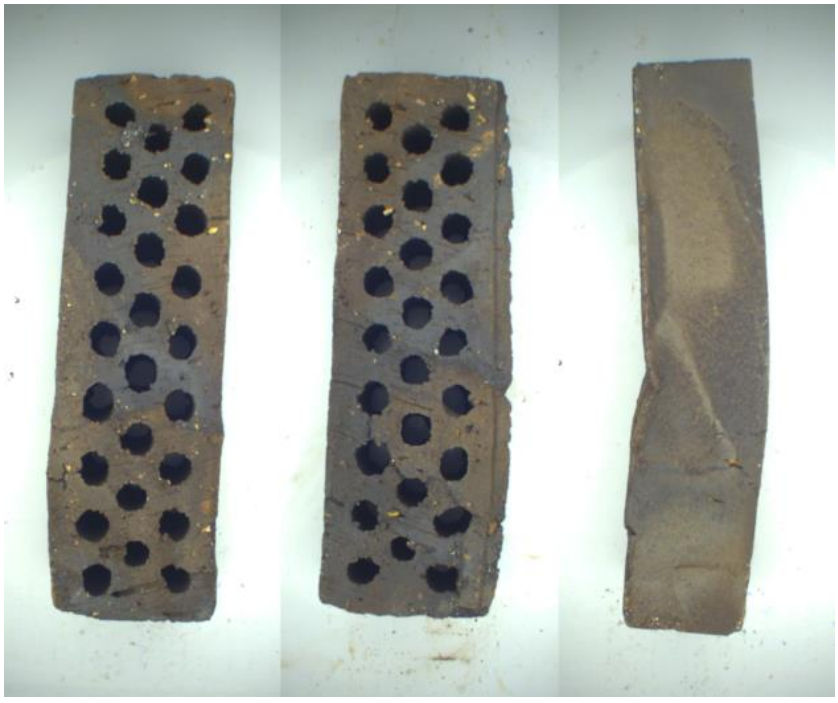

Figure 26. *Class 3 brick.*

The objective was to collect enough data from all three brick classes. Finally, the decision was made to collect 250 bricks from classes 1 and 2, with an additional 500 bricks from class 3, to have more items representing the major defects. The plan was to image all the bricks from three different sides: top, bottom and from the side surface. (Figures 24, 25, 26). If a small defect were at the end (the smallest side) of the brick, the end side would be visible only when the brick was placed at the corner of a wall. In that case the brick can be rotated before masonry. A situation where both ends would have defects is very rare. The bricks are packed in strings at the production site, each string made up of 128 individual bricks. The delivered brick shipment consisted of 2x128 class 1 bricks, 2x128 class 2 bricks and 5x128 class 3 bricks.

4.2 Test equipment

Imaging took place in the SAMK RoboAI laboratory and was started on 29th July, with the last set of images collected on 20th August 2021. Data acquisition was performed over 14 different days. The imaging setup consisted of a machine vision camera, a laptop with Halcon imaging software, a lighting module, a power source and a conveyor belt, as shown in Figure 27.

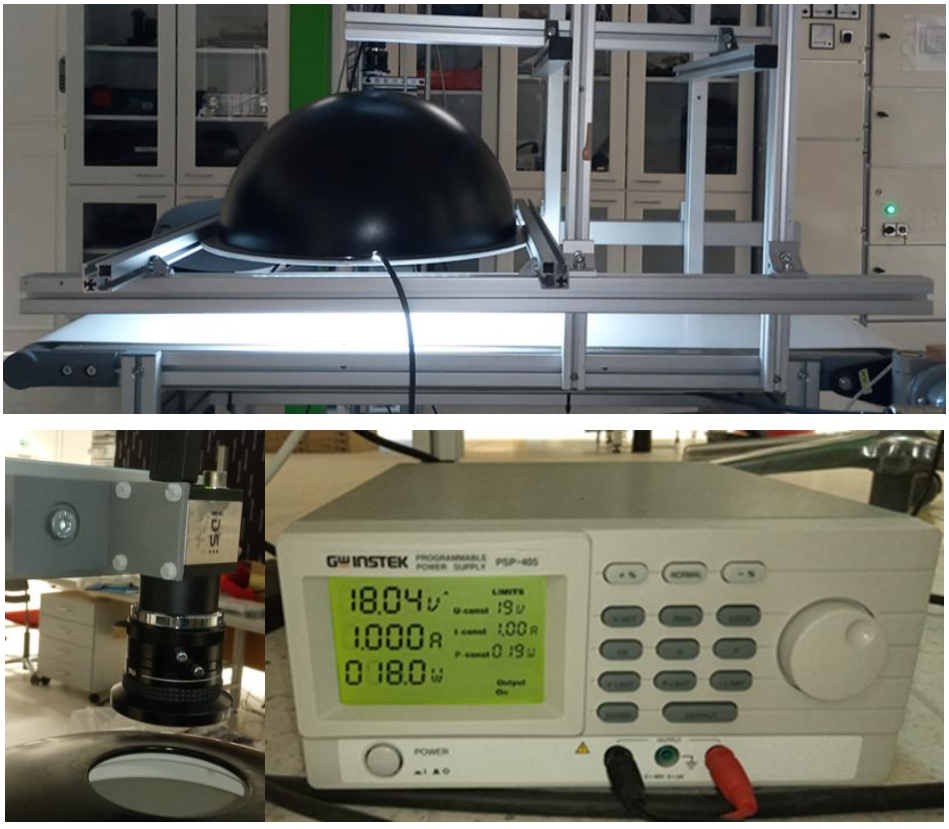

Figure 27. *The conveyor belt with installed camera. The dome lighting's power source, tuned to imaging settings.*

There were several machine vision cameras available in the laboratory for use for imaging, and IDS area camera model UI-3590CP-C-HQ was chosen to collect the data. This 18 MP resolution area camera is equipped with a CMOS colour sensor and a rolling shutter. A colour sensor was chosen because colour difference was one of the defect types to be inspected.

The goal was to build a lighting arrangement where the amount of light would remain constant during the shooting procedure, regardless of the environment's lighting conditions. When starting to plan the lighting arrangement, two options were possible. One option was a setup with frontal lighting, where a set of LEDs would be installed above the conveyor. This approach would have required a box around the camera, to keep the lighting conditions stable. The second option was to use diffused dome LED illumination, where a dome structure efficiently blocks reflections from the surroundings. Eventually, the dome option was selected for testing. One advantage with using this setup was that there was no need to build an external imaging box around the camera.

After several aluminium support frame installations, the correct height for the dome was tuned (Figure 27). The ideal height was set to the level where a single brick would be able to move under the rails in all shooting positions. Additionally, some free space needed to remain between the rail and the brick. That was for the case of a bent class 3 brick, which also had to be able to move freely under the rails.

The next phase was to install the camera in the correct position. The vision camera was mounted at the same level as the hole on top of the dome, and test imaging was then performed. The inspection of test images revealed that there were blue reflections in the images (Figure 28). This occurred due to a led string light, pointing towards the camera sensor. To solve this reflection problem, the camera position was shifted 3 cm above the top hole of the dome (Figure 29). This procedure helped to get rid of the blue dots in the images. There were also noticeable reflections in the test images caused by the indoor fluorescent lighting at the laboratory. It was possible to switch off the room top lighting during the imaging, and in this way the amount of extra light caused by the environment was minimised.

The conveyor belt used in this setup could be run in both directions, with multiple speed values. Images were captured mainly by running the conveyor in the same (forward) direction. Occasionally, when being imaged the bricks cracked into two pieces, or with other less common defects, both conveyor directions were used to obtain multiple images.

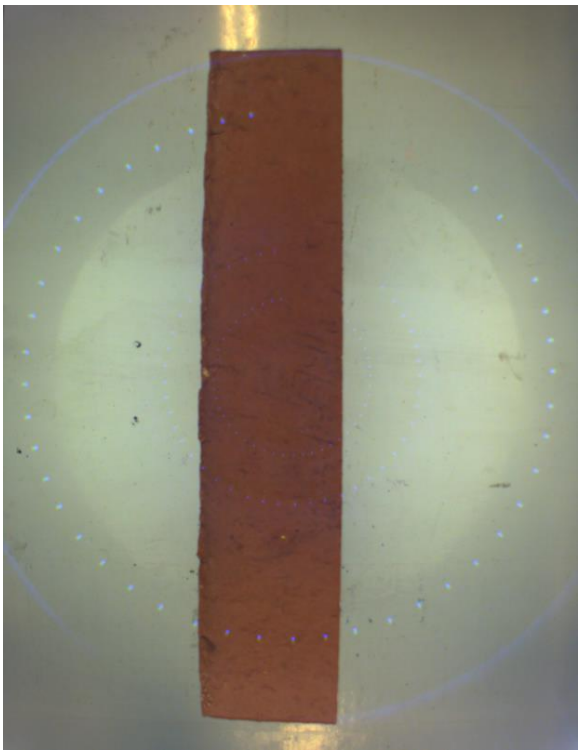

***Figure 28**. In the test shooting, a low camera installation position incurred blue reflections in the image. Also, indoor fluorescent lighting caused yellow reflections in the image, visible at both ends of the brick.*

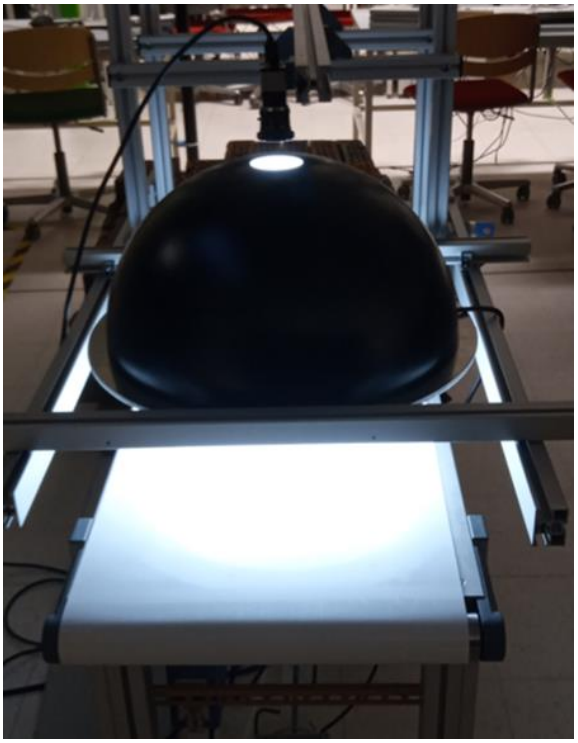

***Figure 29.** Camera installed above the dome light, with overhead lighting switched off during shooting.*

The camera was controlled by Halcon machine vision software. The code used for imaging and the camera parameters was tuned to be suitable for the bricktype used. The Halcon program controlled the image capturing with several parameters. The shape of the detected brick and the colour were supposed to be within certain limits to trigger the camera automatically. The number of images to be captured while a single brick passed through the camera's field of view was also one adjustable parameter. At first, all bricks were imaged once from three directions. Later, after all 3 classes had been imaged, some extra images were captured with a multiple image capturing program. This was made to collect more data on the already imaged bricks, for possible later usage.

Especially when imaging class 3 (rejected quality) bricks, in some cases the brick colour on one of the three sides was too light, and the image was not captured. The same effect occurred if the program's area parameter was not fulfilled because a large part of the brick was missing. In these situations, bricks were shifted to the side and marked. Afterwards the program parameters were also changed to collect images of these bricks.

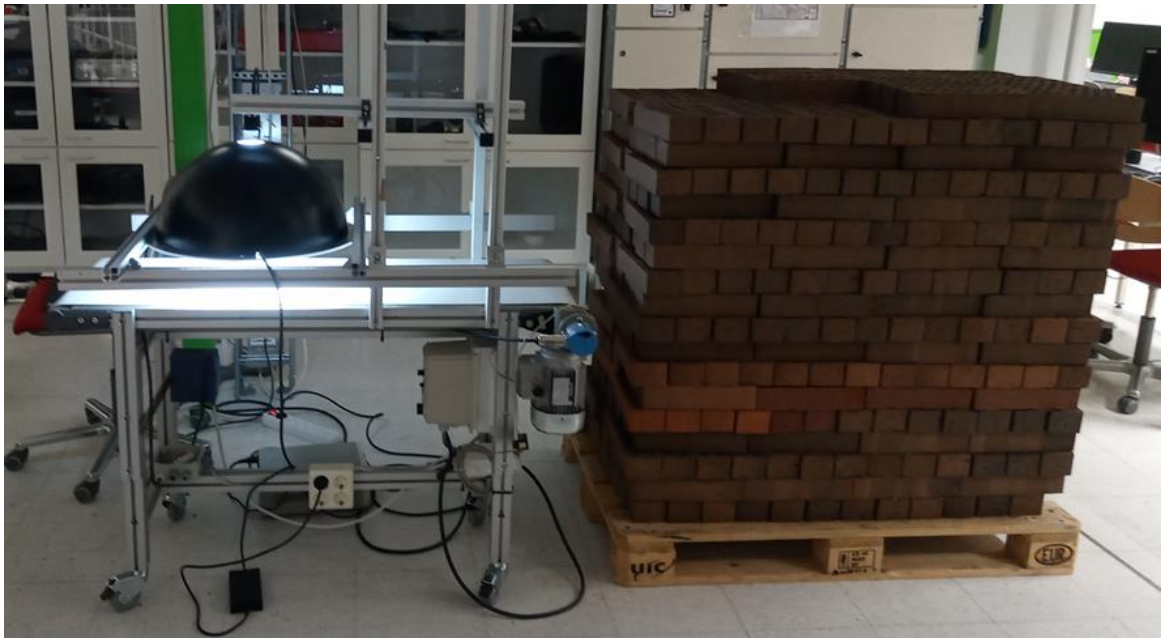

Figure 30. *The imaging setup in the laboratory, and a pile of approximately 500 imaged bricks. From the 10th row upwards, the pile is filled with class 1 bricks, and the lower rows consist of class 2 bricks. The colour hue difference between the two classes is noticeable.*

The imaging procedure was as follows. A pallet stacked with bricks was transported to the left side of the conveyor. An empty pallet for imaged bricks was placed at the right side of the conveyor. Bricks were fed one at the time, through the camera's field of view

on the conveyor, while the belt speed and direction remained constant. After passing the camera, a brick was manually picked up, rotated to the other side, and fed again to the conveyor. This picking sequence was repeated twice per brick to collect data from all three sides. Then the brick was placed on the right side pallet (Figure 30) and so on.

When processing multiple bricks at the lab, there was a constant need to clean the conveyor belt several times every day. This was an attempt to keep the backgound colour in the images constant. Also, the floor and the camera lens needed to be cleaned occasionally. Usage of the same kind of dome lighting in the factory would need an extra cover to prevent dust spreading inside the dome. Additionally, a proper air blowing mechanism could be used to avoid dust disruption.

4.3 Collected data

The total amount of collected images is presented in Table 3 and all the imaged bricks are shown in Figure 31. At the first stage, all bricks were fed to the conveyor for imaging. Some of the bricks were not captured by the vision system, because all the camera program triggering parameters (e.g. colour hue, shape) were not fullfilled. In the second stage, the program parameters were changed to also capture images of light-coloured bricks. The third stage of shooting consisted of light-coloured bricks, and also cracked bricks with one end missing. During the third shooting, the camera program parameters also needed to be altered, to gain images of different-shaped bricks.

After the data acquisition, the images of all the classes were inspected, duplicate images were deleted and the data was fixed in the right order. At the end of the inspection, in the class data folder there were only three photos for the same brick in the right order. The two first images presented the upper and underside of the brick (sides with holes), and the third photo was from the side surface.

Python 3.8 software was implemented to concatenate three images together, resulting in one image of each brick. File paths to find the images to be combined and saved were given to the program. The order of the images to be combined had to be correct in the data folder. The program combined three images arranged in ascending numeric values at the end of the image name. In cases where the number of images in the data folder were not divisible by three, the additional images remained in the original shape and were not combined.

Table 3. *The total number of images acquired in each phase.*

Class	1st stage captured images	2nd stage captured images	3rd stage captured images	Images after inspection	Final combined images
1	768			768	256
2	765		6	768	256
3	1820	79	94	1779	593

Figure 31*. Bricks were piled on pallets after imaging. The two left pallets are filled with class 3 bricks, and the right-side pallet is stacked with class 2 bricks (9 lower rows), with class 1 bricks on top.*

There were several damage types in brick classes 2 and 3 (Figure 32). The main defect to differentiate a class 2 brick from a class 1 brick was the difference caused by the lighter colour hue. Also, some minor chips and cracks were noticed in class 2 bricks. In several cases, a class 2 brick resembled a class 1 brick so much that there was no noticeable difference between the classes. This caused challenges for classification,

which is covered in the following chapters. After inspection of class 2 and class 3 bricks, the overall defect types can be listed as follows:

- difference in colour
- minor chipping
- part of brick missing
- minor cracks
- whole brick cracked
- bent brick
- twisted brick.

When looking at a brick from above or underneath, black holes can be seen. If a minor crack occurs in between two holes, the brick can be categorised as class 1. However, if the minor crack extends between three holes, for durability reasons the brick is categorized as class 3. A black colour in the brick images indicates that the brick was exposed to a high temperature in the oven. In this case, the brick may also be twisted or bent.

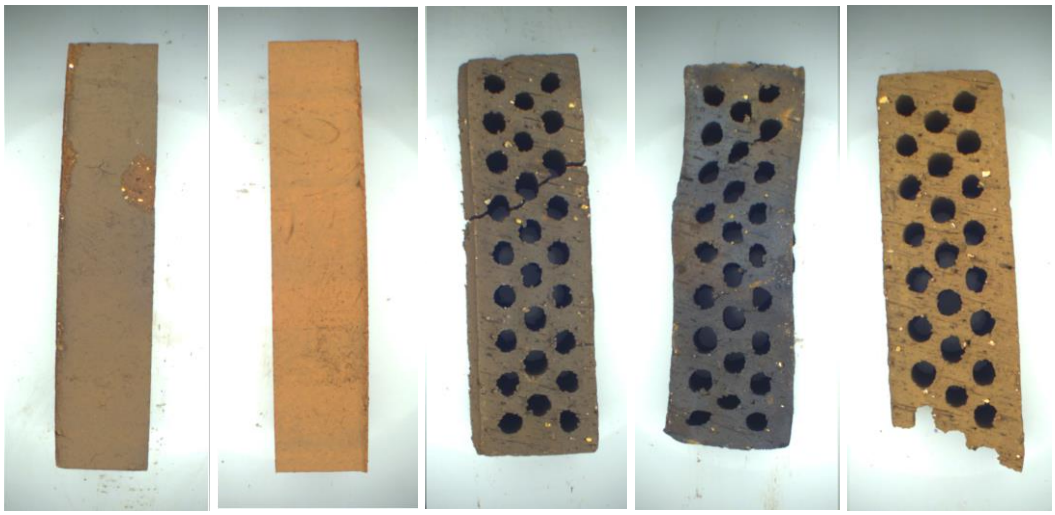

Figure 32. *Defect types: the two images on the left represent class 2, with chipping and a colour error. Other images: class 3 with a major crack, a bent/colour error, and a part of the brick missing with a colour error.*

4.4 Modifications to collected data

At this stage, the image data consisted of 256 classes 1 and 2, and 593 class 3 TIFF format images, with a resolution of 1800x1500 pixels. There were large white background areas in the images, some of them having fine-grained brick dust in the background. The next procedure was to cut off the majority of the white areas from each original and discrete 600x1500 pixel size images. Then three cropped images were combined to represent one individual brick.

The CNN model VGG16 was used in this thesis work for classification. The input image was resized to 224x224 pixels in the network. The goal was to test the network's performance with the combined original size and the combined cropped size images.

Image cropping was performed with a tailor-made tester program, implemented by a local engineering company using C++ programming language. In the program, the original image is first turned into grayscale, and convolved with a Sobel kernel to highlight the edge features. Strong enough edges are taken through the Hough transform to find the most dominant linear lines, both (nearly) horizontally and vertically. These four edges of the brick are then offset by some pixels, to make sure no actual brick area is touched (Figure 33). Then the original colour image is clipped and straightened by using the intersection points of the edges.

A new combining program was implemented, and three cropped images were combined into one image, as earlier. The size of the cropped images varied because the size of the bricks was not exactly uniform. To preserve the original brick image aspect ratio, in the program the maximum height value of three images was detected. Then the images were combined from left to right, and the empty space below was filled with a white colour (Figure 34). The resolution of the combined images varied by a small amount. The overall combined image size was about 1150x1300 pixels.

By cropping the original size brick images, the situation where the CNN would be able to learn features from the background was eliminated. In the dataset images in this thesis the background was relatively clean, due to the constant cleaning of the conveyor belt in the laboratory. In factory conditions, there would be more brick dust in the background, and image cropping would be advisable. Another aspect is that the cropped images were also rectified vertically as compared to the original photos.

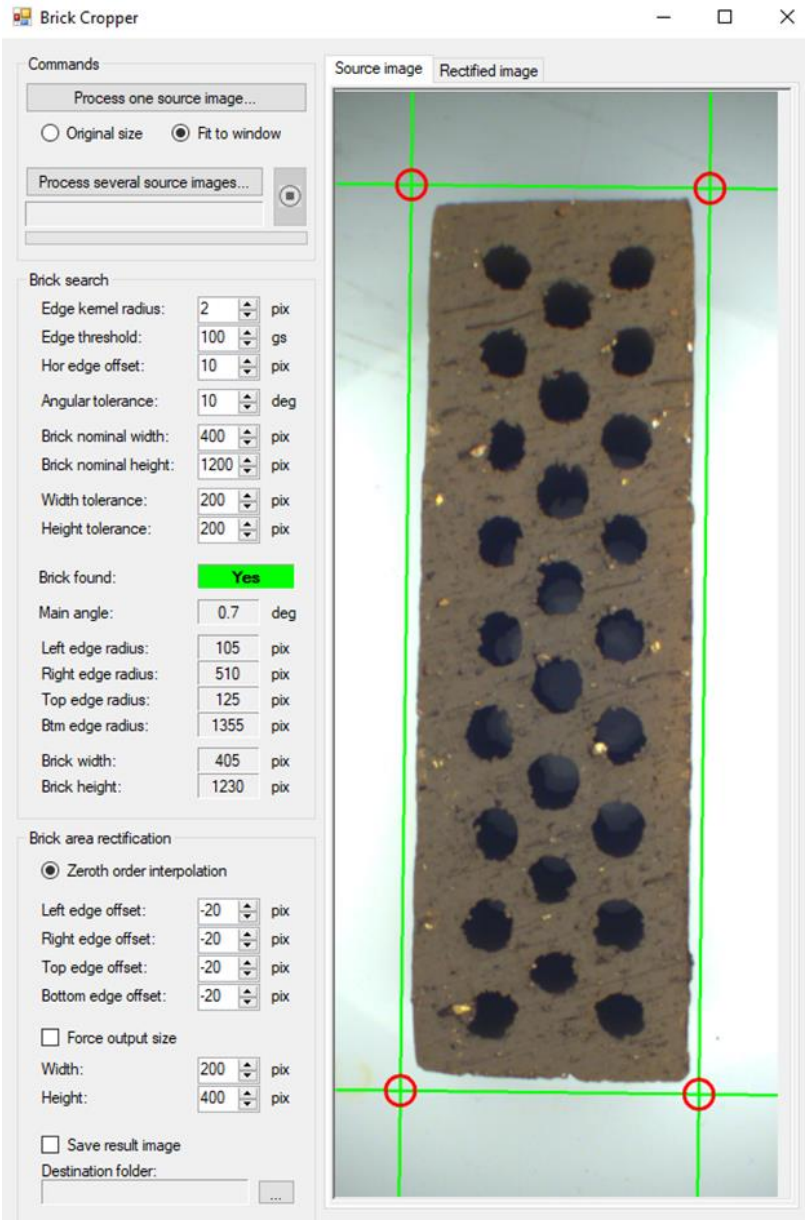

Figure 33. *Cropping program interface. Detected original image was cropped along the green lines.*

In the combined images (Figure 34), the brick side surface photo seems to be longer than the other sides. This is due to the camera's projective transformation in the imaging phase. The side surface of the brick was closer to the camera when imaging the brick on the conveyor belt.

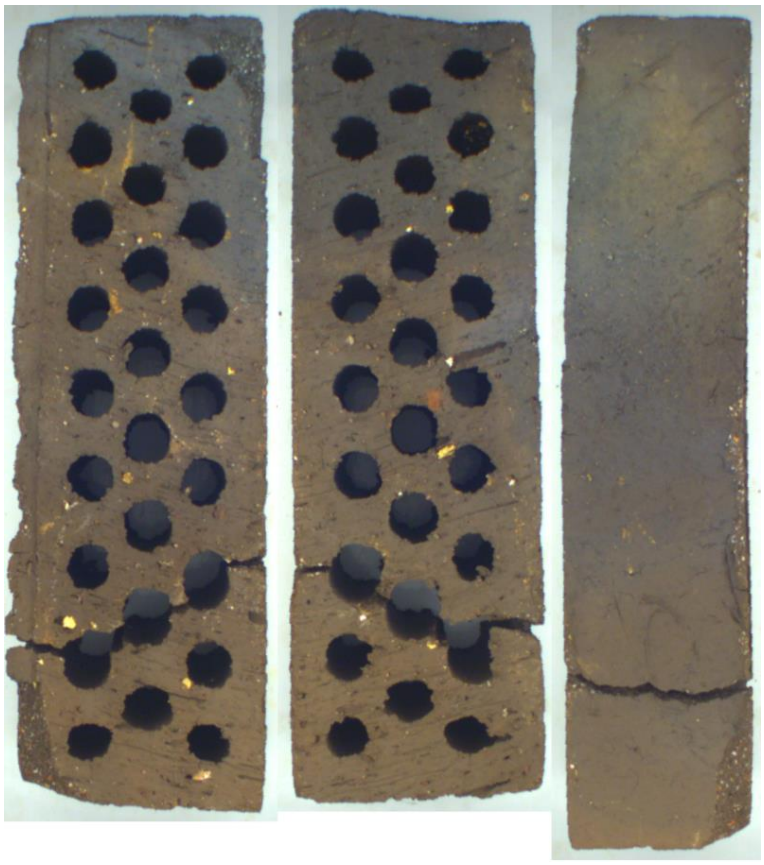

Figure 34. *Cropped and combined class 3 brick.*

5. CLASSIFICATION TEST RESULTS

The next phase in the thesis work was to examine the ability of a convolutional neural network to categorize different brick classes by means of the collected data. In this chapter, the results of three different tests are presented. The Python programming language was chosen to be used in this project to develop a machine learning model. Initially, Anaconda open-source distribution with Python version 3.8, TensorFlow and Keras frameworks were installed on a Windows PC. Anaconda comes with the Jupyter and Spyder programming interfaces, which were used during the code development.

The training phase in a neural network demands a lot of computational power, so it is useful to operate with a GPU (Graphics Processing Unit) rather than a CPU (Central Processing Unit). The necessary GPU drivers were installed in the system from the TensorFlow website [57]. During the tests, as expected, the GPU computation turned out to be several times faster compared to only CPU operation.

At the beginning of the CNN design phase, several approaches were available. One option was to design the network all the way from the beginning to the end and update the network structure with the help of the results received from the test runs. Initial tests were made with this method, by changing the network's layer structure and parameters. Having a relatively small amount of image data, a need for more effective ways to achieve a powerful classifier arose.

The transfer learning method was exploited to achieve more efficient results for brick classification. In this learning procedure, pretrained network weights can be used as feature extractors and only the fixed classifier part at the top of the network is trained with new data. Also, a fine-tuning method can be used to alter the weights in the base model's convolutional layers [58]. In the Keras framework, there are CNN models with pretrained weights available for use with transfer learning. A VGG16 pretrained CNN model was chosen to be used in the thesis experiments.

5.1 Cross-validation and oversampling

Cross-validation is a technique used in machine learning to evaluate model performance more reliably. The basic idea in cross-validation is to divide an available dataset into different train/test splits with several allocations. By altering the division of training and test samples in the dataset, more robust results can be achieved. In K-fold cross-validation, the dataset is randomly divided into K-folds. K-1 divisions are used in training and

one group is used for testing at a time. If K is chosen to be 5, four different groups at a time are used in training and one group is used for testing (Figure 35) [14].

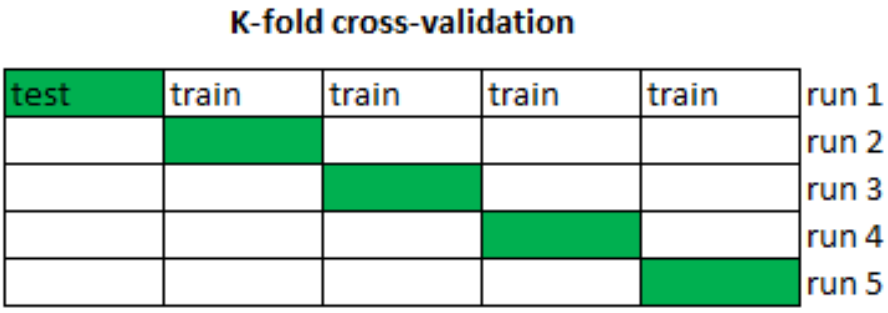

***Figure 35**. K-fold cross-validation, where K=5* [14].

Uneven class distribution can have a harmful effect on the classification with CNNs, but the imbalance can be improved with oversampling or undersampling procedures. In oversampling, a certain class has fewer samples than the others. The number of data samples in this class is increased to have an even amount of data, as compared to the other classes. Also, by performing undersampling, the amount of data points in the majority classes can be reduced. As a downside of undersampling, by removing samples from the majority class, valuable information can be lost.

In a systematic study by Buda et al. [59], the conclusions were that oversampling does not cause overfitting when applied in a CNN model. Oversampling can be performed by randomly adding copies of existing datapoints to the minority classes. A more progressive oversampling technique is the Synthetic Minority Oversampling Technique (SMOTE), where new datapoints are constructed synthetically for the class.

In the SMOTE technique, the feature vectors of oversampled class datapoints are visualised and the k-nearest neighbours of these datapoints are spotted. A new datapoint is created on a line between the original and the k-nearest datapoint in the feature space. A synthetic datapoint is created by multiplying the difference between the datapoints by a number between 0 and 1 and adding this value to the original sample [60].

Oversampling was implemented to maintain an even class distribution between the training samples in classes 1-3. A separate code was created to make copies of class 1-2 data samples, and a cross-validation method was also performed before the oversampling. The procedure was as follows:

- Count the random cross-validation by 50/50 data division between the training and test set folder images, concerning all classes. At this stage, 128 images were

in the class 1 and 2 training folders. The class 3 training folder contained 297 images.

- Oversample class 1 and class 2, by making copies of all 128 datapoints. Then randomly choose 41 more training samples, to achieve even class distribution in the training set images. At this stage, 297 images were in the class 1, 2 and 3 training folders.
- The class 1 and 2 test folders contained 128 images. The class 3 test folder contained 296 images.
- Perform training 10 times. After 10 runs, calculate the test accuracy mean and standard deviation (Tables 4, 6, 8). Also, calculate the other performance metrics from the 10 test runs (Tables 5, 7, 9).

When performing training of the network, the training set was divided by 50/50 division into training and validation sets. In the results listed further on in this chapter, oversampling was performed for the training set, and cross-validation was performed before the oversampling.

5.2 Results

During the development of the CNN structure, several tests were performed by altering the hyperparameters of the network. Different learning rates, batch sizes, epochs, image pre-processing methods and layer structures in the classifier part of the network were tested. Also, many variations were examined concerning the training/validation/test divisions. Different input image sizes and data augmentation techniques were also tested. No data augmentation (except oversampling), was applied to the image data concerning the results in this chapter.

10 tests were performed between classes 1 and 2, 1 and 3, 2 and 3 and for all classes 1-2-3. Experiments were conducted with the original size 1800x1500 pixel images and with the cropped image size dataset. In the VGG16 model, these different-sized input images were resized to 224x224 pixels. Section 5.4 covers the tests where all convolutional layers were frozen during training, and only the classifier part was open for training. The tests covered in sections 5.3 and 5.5 were conducted with all VGG16 layers open for training.

Network parameters were set to constant during the tests to achieve comparable results. The loss function used during the tests was categorical cross-entropy. Stochastic gradient descent (SGD) was utilised as an optimiser, with a learning rate of 0.0001.

5.3 Original image size with all layers open

The results in this section present experiments where the original size images were used as input images. Pretrained VGG16 weights were utilised, and every layer in the VGG16 model was opened for training. By opening the convolutional layers for training, high accuracy scores in all class-wise tests were achieved (Table 4). The reason for this is assumed to be the difference between pretrained ImageNet images and the brick data images. Adjusting the weights also in the convolutional layers improved the model's performance. The whole network was trained with the brick data and the training was performed with 15 epochs.

Table 4. *Training, validation, test accuracies and standard deviations (SD) of 10 class-wise tests, with the original 1800x1500 image size.*

10 Testruns	Training accuracy	Training SD	Validation accuracy	Validation SD	Test accuracy	Test SD
Classes 1-2-3	0.99	0.02	0.94	0.04	0.89	0.06
Classes 1-2	0.99	0.01	0.99	0.02	0.90	0.02
Classes 1-3	1.00	<0.01	0.95	0.02	0.95	0.01
Classes 2-3	1.00	<0.01	0.96	0.02	0.94	0.01

The results in Table 4 indicate, that 89 percent accuracy was achieved in the tests where all three classes were used. Class-wise 1-2 tests achieved almost the same accuracy. Class-wise tests between two classes, which were conducted with class 3 images, achieved over 90 percent accuracy. The training curve in Figure 36 is rather smooth, so the model is not assumed to have improved learning after 15 epochs. In the performance metrics shown in Table 5, F1 score values indicate that class 3 brick images were the easiest to classify. In the same table, class 2 bricks received the lowest F1 score values.

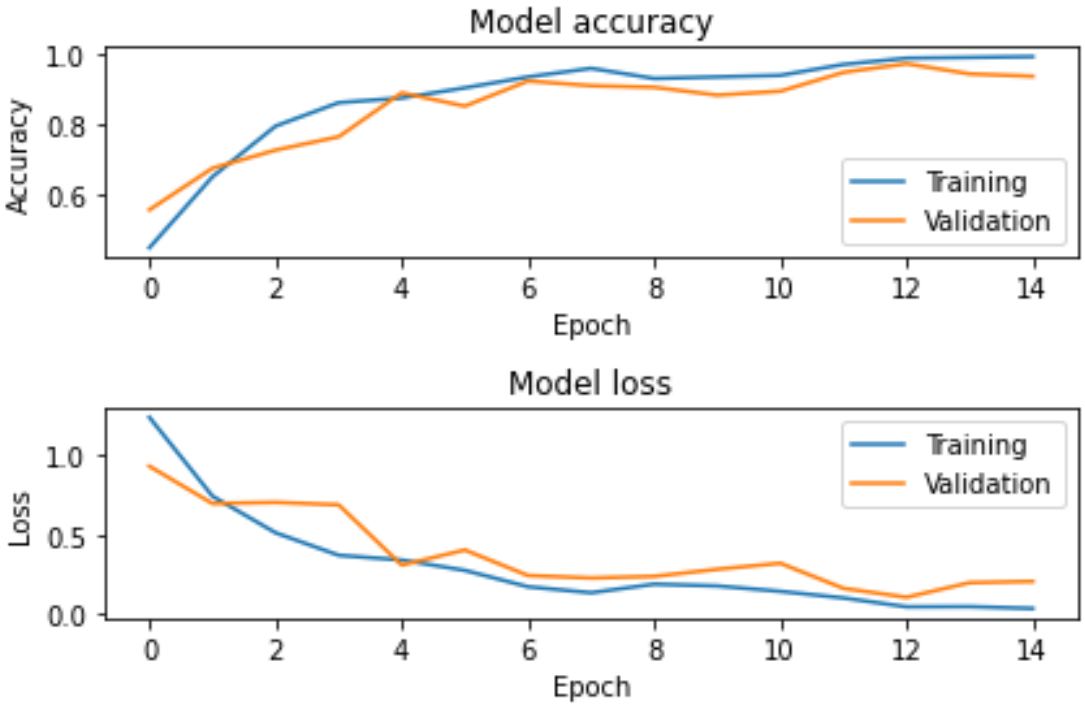

Figure 36. *Model learning curves with the original image size. Accuracy and loss as a function of epoch. The graph concerns one training phase between classes 1-2-3.*

In Figure 36, the learning curve presents a single 15 epoch training run between classes 1-2-3. In the test set, there was over twice the amount of test samples in class 3, as compared to classes 1-2. This can be noticed in the images column in Table 5.

Table 5. *Class-wise model performance metrics as the mean of 10 tests of 1-2-3 classification; 15 epoch training conducted before each test.*

10 Testruns	Precision	Recall	F1 score	Images
Class 1	0.84	0.93	0.88	128
Class 2	0.82	0.86	0.84	128
Class 3	0.98	0.89	0.93	296

5.4 Cropped image size

The results in this section present experiments where cropped size images were used as the input images. VGG16 model weights were utilised, and only the classifier part of the pretrained model was opened for training. Layers 1-19 were frozen during training. The classifier part of the network was trained with the brick data and the training was performed with 15 epochs.

Table 6. *Training, validation, test accuracies and standard deviations (SD) of 10 class-wise tests, with the cropped image size (approximately 1150x1300 pixels).*

10 Testruns	Training accuracy	Training SD	Validation accuracy	Validation SD	Test accuracy	Test SD
Classes 1-2-3	0.99	0.02	0.92	0.02	0.85	0.02
Classes 1-2	0.98	0.02	0.98	0.02	0.84	0.03
Classes 1-3	1.00	<0.01	0.95	0.02	0.94	0.02
Classes 2-3	1.00	<0.01	0.92	0.01	0.90	0.02

The results in Table 6 indicate, 85 percent accuracy in the test where all three classes were used. Class-wise 1-2 tests achieved almost the same accuracy. The best class-wise tests between two classes, which were conducted with class 3 images, achieved 94 percent accuracy. The training curve in Figure 37 is rather smooth, therefore the model is not assumed to have improved learning after 15 epochs. In the performance metrics shown in Table 7, F1 score values indicate that class 3 brick images were the easiest to classify. In the same table, class 2 bricks received the lowest F1 score values.

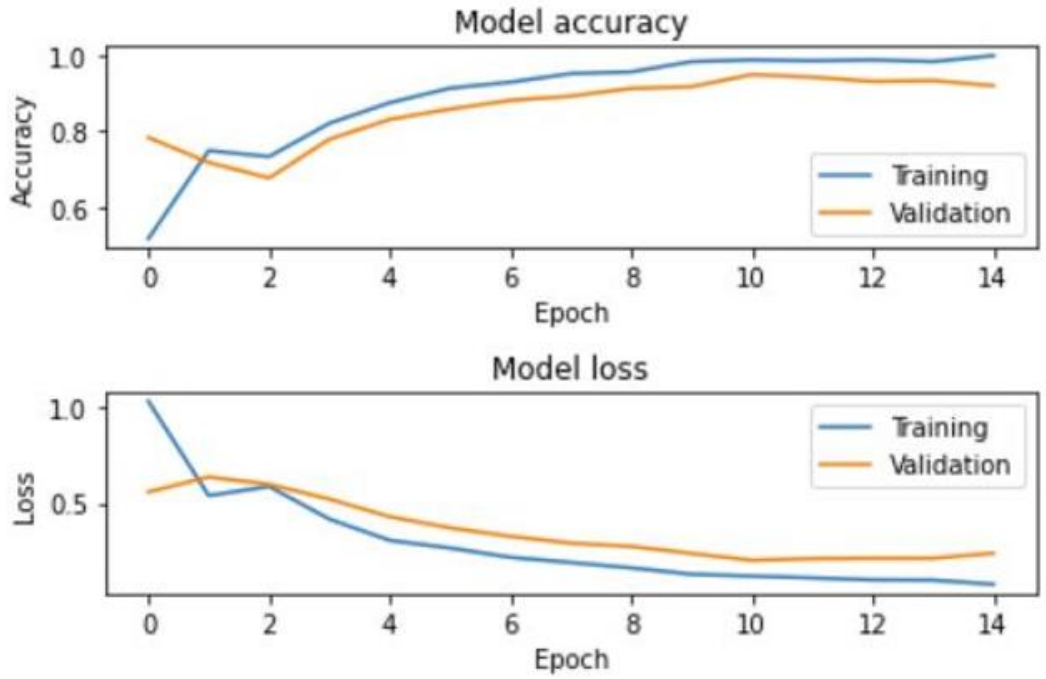

Figure 37. *Model learning curves with the cropped image size. Accuracy and loss as a function of epoch. The graph concerns one training phase between classes 1-2-3.*

In Figure 37, the learning curve presents a single 15 epoch training run between classes 1-2-3. In the test set, there was over twice the amount of test samples in class 3, as compared to classes 1-2. This can be noticed in the images column in Table 7.

Table 7. *Class-wise model performance metrics as the mean of 10 tests of 1-2-3 classification; 15 epoch training conducted before each test.*

10 Testruns	Precision	Recall	F1 score	Images
Class 1	0.77	0.86	0.81	128
Class 2	0.71	0.78	0.74	128
Class 3	0.98	0.87	0.92	296

5.5 Cropped image size with all layers open

The results in this section present tests where cropped size images were used as the input images. Pretrained VGG16 model weights were utilised, and every layer in the VGG16 was opened for training. High accuracy scores in all class-wise tests were achieved by opening convolutional layers for training too (Table 8). The reason for this is assumed to be the difference between pretrained ImageNet images and the brick data images. Adjusting the weights also in the convolutional layers improved the model's performance. The whole network was trained with the brick data and the training was performed with 15 epochs.

Table 8 *Training, validation, test accuracies and standard deviations (SD) of 10 class-wise tests, with the cropped image size (approximately 1150x1300 pixels).*

10 Testruns	Training accuracy	Training SD	Validation accuracy	Validation SD	Test accuracy	Test SD
Classes 1-2-3	1.00	<0.01	0.95	0.02	0.89	0.02
Classes 1-2	1.00	<0.01	1.0	<0.01	0.90	0.03
Classes 1-3	1.00	<0.01	0.96	0.01	0.96	0.01
Classes 2-3	1.00	<0.01	0.94	0.03	0.92	0.05

The results in Table 8 indicate 89 percentage accuracy in the test where all three classes were used. Class-wise 1-2 tests achieved approximately the same accuracy. The best class-wise tests between two classes, which were conducted with class 3 images, achieved 96 percent accuracy. The training curve in Figure 38 is rather smooth, the model is not assumed to have improved learning after 15 epochs. In the performance metrics shown in Table 9, the F1 score values indicate that class 3 brick images were the easiest to classify. In the same table, class 2 bricks received the lowest F1 score values.

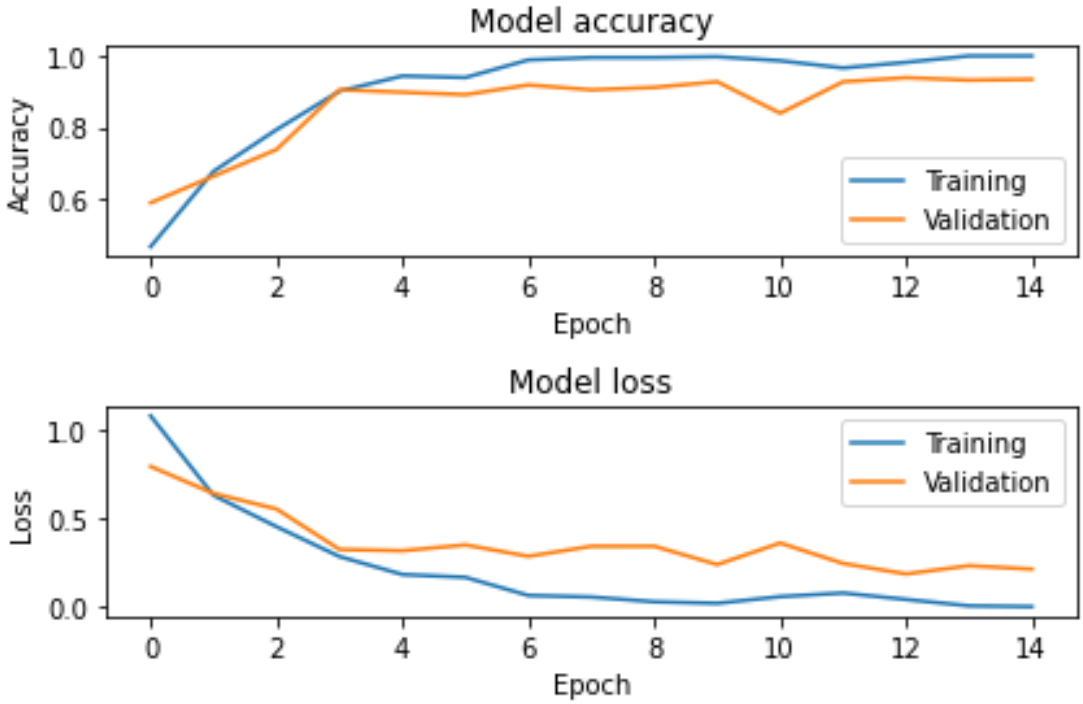

Figure 38. *Model learning curves with the cropped image size. Accuracy and loss as a function of epoch. The graph concerns one training phase between classes 1-2-3.*

The learning curves shown in Figure 38, and the confusion matrix in Figure 39 present a single training (Figure 38) and test run (Figure 39) between classes 1-2-3. In the test set, there was over twice the amount of test samples in class 3, as compared to classes 1-2. This can be noticed in the images column in Table 9. Also, the amount of class-wise test samples is marked in the rows in the confusion matrix.

Table 9. *Class-wise model performance metrics as the mean of 10 tests of 1-2-3 classification; 15 epoch training conducted before each test.*

10 Testruns	Precision	Recall	F1 score	Images
Class 1	0.86	0.91	0.88	128
Class 2	0.75	0.85	0.80	128
Class 3	0.99	0.89	0.94	296

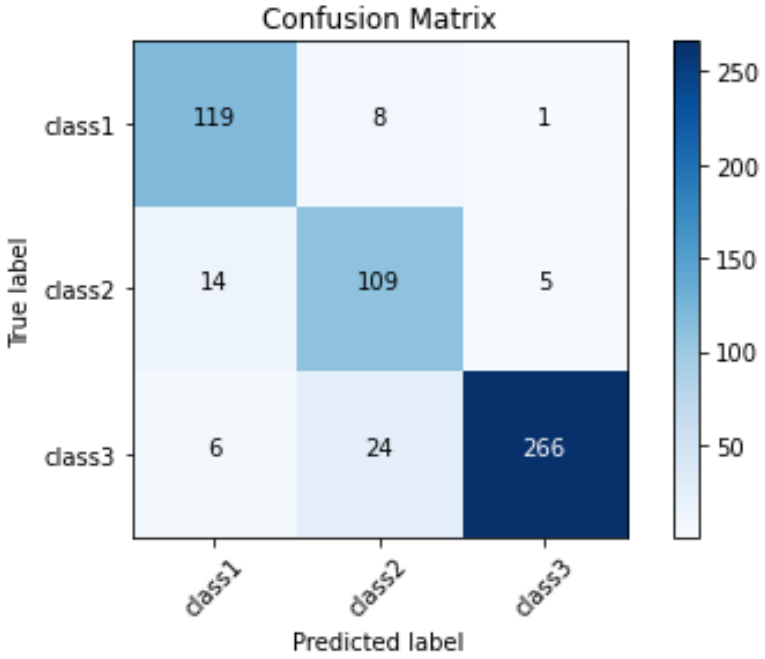

Figure 39. *Confusion matrix after 128 class 1-2 and 296 class 3 test set images.*

5.6 Discussion of the results

The final tests were performed with the original (1800x1500) and the cropped (about 1150x1300) pixel image sizes. The input images were resized in the VGG16 model to 224x224 pixels. In earlier tests in the designing phase, the VGG16 structure was modified to resize the input images to bigger sizes (e.g. 448x448 or 440x500 pixels). These procedures did not give better results, so the original 224x224 resize value was maintained.

All of the tests were performed 10 times to achieve more reliable results. Mean and standard deviations were calculated from the accuracy scores (Tables 4,6,8) and the mean of the other performance metrics (Tables 5,7,9).

When analysing the class-wise images, it is noteworthy that several class 2 brick images resembled class 1 bricks (Figure 40). When inspecting these difficult cases manually, the correct classification could not even be made by the naked eye. These similar datapoints probably had a detrimental influence on the accuracy scores. In general, when comparing class 1 and class 2 brick images, the colour hue difference was the main difference between these classes.

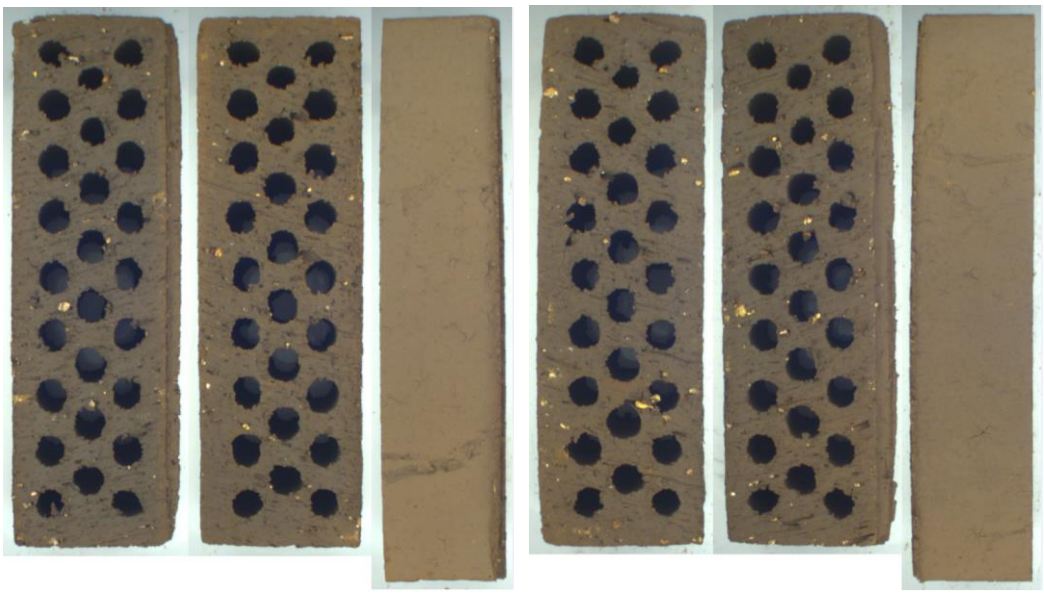

Figure 40. *Class 2 brick on the right, resembling class 1 brick on the left.*

When interpreting the accuracy test results in Tables 4, 6 and 8, no major differences were observed. Class 3 images were the easiest to classify with all test combinations. This can be noticed from the class-wise tests (Tables 4, 6, 8). In all two-class comparisons, class 3 bricks were detected with at least 90 percent accuracy. The best performance, 96 %, was achieved with a class-wise 1-3 comparison and a cropped image size, when all layers were opened for training.

Class 3 images were easily recognised, also with the naked eye. Obvious colour differences with cracks and chips were typical features in class 3 brick images. Additionally, there were more than twice as many images in class 3 as compared to the other classes. This also contributed to better class 3 detection results.

Class-wise 1-2-3 and 1-2 accuracy tests gave approximately the same results within the same test table. This can be noticed in the top two rows in Tables 4, 6 and 8. When comparing these two class-wise tests, about 90 per cent top accuracies were obtained. These were achieved with the tests where all the network layers were opened for training (Tables 4 and 8).

The class-wise 1-2 test achieved the lowest accuracy scores of all the two-class comparisons (Tables 4, 6 and 8). According to the accuracy tests, class 2 images were the hardest to classify correctly. In the accuracy tests both the original image and cropped image sizes with all layers open gave the best results. This leads to the conclusion that it is advisable to open the whole network for training.

The F1 score indicates the harmonic mean of precision and recall. The F1 score can be described as a combination of precision and recall metrics. The precision, recall and F1 score performance metrics are listed in Tables 5, 7 and 9. In all tables, the metrics indicate the highest values for class 3 images.

In all of the tests (Tables 5, 7 and 9), class 1 bricks achieved a better F1 score than class 2. Class 2 brick images achieved the lowest F1 scores in all tests. The performance metric results led to the same kind of conclusion as with the accuracy scores. Class 3 brick images were the easiest to classify, and class 2 images were most likely to be confused with other classes.

To summarise the results above, the original and cropped image sizes with all layers open provided the best results. Overall, the cropped image size tests are assumed to be more reliable when estimating classification performance. This is because of the more detailed input image. An important aspect is that the classification results are likely to be improved by having more data samples of all classes.

6. CONCLUSIONS

The convolutional neural network turned out to be an effective learning model for finding different defects from the brick images. Additionally, the usage of transfer learning gave improved results regarding classification performance. According to the results obtained using two different image size datasets, it can be concluded that the convolutional neural network is suitable for the classification of brick images.

Diverse CNN models from scratch were tested during the development. However, the usage of the VGG16 model with the transfer learning concept gave the best results. The usage of pretrained weights speeded up the training process. The same method is also recommended for use in future design. According to the classification test results, further development of the detection system is advisable, with certain considerations.

In this project, the amount of datapoints available to be exploited with deep learning model was rather low. The main reason for the limited dataset was that there was no proper conveyor installed at the factory facilities. Therefore, gathering the image data was a quite challenging task. Approximately 2400 kg bricks were manually processed three times in the laboratory to collect the whole dataset. Overall, it can be assumed that by having more data samples, the classification test results can be improved.

Concerning data collection in these particular factory conditions, over 10 colour options with several brick sizes and five outer brick surface options are available. Every brick type must be taught once to the detector. It is reasonable to perform this alongside the normal handmade, class-wise separation in production.

The proposed option is to construct a conveyor, equipped with an imaging box, where cameras image the bricks from three directions. By building a solid imaging box around the camera setup, harmful effects caused by the brick dust can be minimised. The following procedure is to combine three images into one in a specific program, as in the thesis work. Frontal LED lighting is preferable as a light source, instead of the dome lighting used in the laboratory conditions.

In the factory setup, at the beginning of the production line, a robot arm would pick each brick one by one and place it onto the conveyor. Only the teaching phase would require manual brick handling by an employee. The operator would choose the correct brick class by pressing the corresponding button on the control panel. Then the operator would feed the brick along the conveyor through the image capturing box. Different class images would be stored in separate directories in the computer used.

After enough images have been stored out of all the classes, the CNN application could be used to detect various brick classes. Three conveyors and two cartesian robots are required after the detection system to manage the brick separation.

The number of collected images would be at first 1000 images per class, and once stored, the brick data can be reused. Additionally, extra images may be added to the data directory later to gain a more comprehensive and efficient dataset.

The input image is resized to 224x224 pixels in the VGG16 model. When the cropped image size is used as input, more detailed defects from the brick surfaces can be represented. It is recommended to use a cropped image size in the actual implementation.

By using the cropped image size, the background effects of the conveyor belts on the learning process can be minimised. When using large background images, brick dust or broken brick pieces may disrupt the learning. Image cropping can be made with the program used in this thesis, or also a CNN-based R-CNN object detection methods can be developed.

A subject for future code design would be to develop the VGG16 model classifier part that was used, by altering its dense layer structures. Other experiments to be conducted could be to test brick classification with other pretrained CNN models available in Keras.

When designing a camera set-up for factory conditions, the spreading of fine-grained brick dust must be considered. Brick dust can easily be propagated on the brick surfaces or on the camera lens. The lighting conditions should remain constant, and also dust propagation on the lights must be prevented. It is recommended to install a pneumatic blower or dust hoover in the system. A line scan camera with an RGB sensor is a valid type for imaging. Resolution and other camera specifications can be chosen after a more detailed set-up design.

According to the examinations described in this thesis, it is possible to exploit a CNN-based classification system for brick quality inspection. More reliable results can be achieved by having additional image data for the classifier. If different class images are very similar even to the naked eye, correct class detection accuracy will be limited. In the image acquisition phase, it is important to provide clearly separable class-wise images for the classifier.

REFERENCES

[1] "The History of Bricks and Brickmaking." Accessed: Nov. 17, 2021. [Online]. Available:https://brickarchitecture.com/about-brick/why-brick/the-history-of-bricks-brickmaking.

[2] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 2nd Editio. Pearson, 2012.

[3] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd editio. Cambridge University Press, 2004.

[4] G. R. Bradski and A. Kaehler, *Learning OpenCV : computer vision with the OpenCV Library*, 1st editio. O'Reilly Media, 2008.

[5] J. Hirvonen, "Computer Vision Measurements for Automated Microrobotic Paper Fiber Studies, Publication 1456," Tampere University of Technology, 2017.

[6] J. Beyerer, F. Puente Leon, and C. Frese, *Machine Vision. Automated Visual Inspection: Theory, Practice and Applications*. Springer, Berlin, 2016.

[7] A. Hornberg, *Handbook of Machine and Computer Vision: The Guide for Developers and Users*, 2nd Editio. Somerset: John Wiley & Sons, Incorporated, 2017.

[8] S. Anand and L. Priya, *A Guide for Machine Vision in Quality Control*, 1st ed. Milton: CRC Press, 2020.

[9] "Imaging Fundamentals | Edmund Optics." Accessed: Jun. 23, 2021. [Online]. Available:https://www.edmundoptics.eu/knowledge-center/application-notes/imaging/6-fundamental-parameters-of-an-imaging-system/.

[10] R. Szeliski, *Computer Vision*, 1st ed. Springer London, 2011.

[11] B. G. Batchelor, *Machine Vision Handbook*, Vol 2. Springer, London, 2012.

[12] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[13] "CIFAR-10 and CIFAR-100 datasets." Accessed: Dec. 21, 2021. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[14] C. M. Bishop, *Pattern recognition and machine learning*, 1st ed. New York: Springer, 2006.

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

[16] Bhartendu, "Linear Regression [Simplest Implementation]," *MATLAB Central File Exchange.*2021,Accessed:Nov.12,2021.[Online].Available: https://www.mathworks.com/matlabcentral/fileexchange/64930-linear-regression-simplest-implementation.

[17] M. Cord and P. Cunningham, *Machine Learning Techniques for Multimedia, Case Studies on Organization and Retrieval*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2008.

[18] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, 2nd ed. New York, NY: Springer New York, 2009.

[19] M. Pietikäinen and O. Silven, *Tekoälyn haasteet: koneoppimisesta ja konenäöstä tunnetekoälyyn*. Oulun yliopisto, 2019.

[20] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. Cambridge: MIT Press, 2010.

[21] "k-means clustering - MATLAB kmeans - MathWorks Nordic." Accessed: Nov. 12, 2021. [Online]. Available: https://se.mathworks.com/help/stats/kmeans.html.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning : An Introduction*, 2nd editio. Cambridge, MA ; MIT Press, 2018.

[23] M. A. Nielsen, *Neural Networks and Deep Learning*, vol. 25. Determination Press San Francisco, CA, 2015, http://neuralnetworksanddeeplearning.com/.

[24] D. Graupe, *Principles of artificial neural networks*, 3rd ed. Singapore: World Scientific Publishing Co. Pte. Ltd, 2013.

[25] M. Swain, "An Approach for IRIS Plant Classification Using Neural Network," *Int. J. Soft Comput.*, vol. 3, no. 1, pp. 79–89, 2012, doi: 10.5121/ijsc.2012.3107.

[26] "CS231n Convolutional Neural Networks for Visual Recognition." Accessed: Nov. 08, 2021. [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[27] K. P. Murphy, *Machine learning: A probabilistic perspective*. Cambridge, MA: MIT Press, 2012.

[28] A. R. Webb and K. D. Copsey, *Statistical pattern recognition*, 3rd ed. John Wiley & Sons, 2011.

[29] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv Prepr. arXiv1811.03378*, 2018.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 25, pp. 1097–1105, 2012.

[31] B. H. Nayef, S. N. H. S. Abdullah, R. Sulaiman, and Z. A. A. Alyasseri, "Optimized leaky ReLU for handwritten Arabic character recognition using convolution neural networks," *Multimed. Tools Appl.*, vol. 81, no. 2, pp. 2065–2094, 2021, doi: 10.1007/s11042-021-11593-6.

[32] J. Y. F. Yam and T. W. S. Chow, "A weight initialization method for improving training speed in feedforward neural network," *Neurocomputing*, vol. 30, no. 1–4, pp. 219–232, 2000.

[33] Nassim Khaled, “Derivative-based Optimization,” *MATLAB Central File Exchange.*2021,Accessed:Nov.12,2021.[Online].Available: https://www.mathworks.com/matlabcentral/fileexchange/27631-derivative-based-optimization.

[34] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv Prepr. arXiv1609.04747*, 2016.

[35] Y. LeCun *et al.*, “Backpropagation applied to handwritten zip code recognition,” *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[37] “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science.” Accessed: Nov. 08, 2021. [Online]. Available:https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[38] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv Prepr. arXiv1511.08458*, 2015.

[39] “How Do Convolutional Layers Work in Deep Learning Neural Networks?” Accessed:Nov.08,2021.[Online].Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[40] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv Prepr. arXiv1409.1556*, 2014.

[41] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015, doi: 10.1007/s11263-015-0816-y.

[42] “VGG16 - Convolutional Network for Classification and Detection.” Accessed: Nov.05,2021.[Online].Available: https://neurohive.io/en/popular-networks/vgg16/.

[43] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010, doi: 10.1109/TKDE.2009.191.

[44] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?,” *arXiv Prepr. arXiv1411.1792*, 2014.

[45] E. Cetinic, T. Lipic, and S. Grgic, “Fine-tuning Convolutional Neural Networks for fine art classification,” *Expert Syst. Appl.*, vol. 114, pp. 107–118, 2018, doi: 10.1016/j.eswa.2018.07.026.

[46] Y. Gao and K. M. Mosalam, “Deep Transfer Learning for Image-Based Structural Damage Recognition,” *Comput. Civ. Infrastruct. Eng.*, vol. 33, no. 9, pp. 748–768, 2018, doi: 10.1111/mice.12363.

[47] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” in *Computer Vision – ECCV 2014, Springer, Cham*, 2014, vol. 8689, no. 1, pp. 818–833, doi: 10.1007/978-3-319-10590-1_53.

[48] V. Shankar, R. Roelofs, H. Mania, A. Fang, B. Recht, and L. Schmidt, “Evaluating machine accuracy on imagenet,” in *International Conference on Machine Learning*, 2020, pp. 8634–8644.

[49] “KerasApplications.”Accessed:Oct.28,2021.[Online].Available: https://keras.io/api/applications/.

[50] “Visualize Features of a Convolutional Neural Network - MATLAB & Simulink - MathWorks Nordic.”Accessed:Dec.01,2021.[Online].Available: https://se.mathworks.com/help/deeplearning/ug/visualize-features-of-a-convolutional-neural-network.html.

[51] J. Lemley, S. Bazrafkan, and P. Corcoran, “Smart Augmentation Learning an Optimal Data Augmentation Strategy,” *IEEE access*, vol. 5, pp. 5858–5869, 2017, doi: 10.1109/ACCESS.2017.2696121.

[52] C. Shorten and T. M. Khoshgoftaar, “A survey on image data augmentation for deep learning,” *J. Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[53] J. Davis and M. Goadrich, “The relationship between Precision-Recall and ROC curves,” in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 233–240.

[54] “A Gentle Introduction to the Fbeta-Measure for Machine Learning.” Accessed: Nov. 03, 2021. [Online]. Available: https://machinelearningmastery.com/fbeta-measure-for-machine-learning/.

[55] X. Ying, “An overview of overfitting and its solutions,” in *Journal of Physics: Conference Series*, 2019, vol. 1168, no. 2, p. 022022 IOP Publishing.

[56] “Overfitting in Machine Learning: What It Is and How to Prevent It.” Accessed: Jan. 04, 2022. [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning.

[57] “GPU support | TensorFlow.” Accessed: Oct. 28, 2021. [Online]. Available: https://www.tensorflow.org/install/gpu.

[58] “Transfer learning and fine-tuning | TensorFlow Core.” Accessed: Oct. 28, 2021. [Online]. Available: https://www.tensorflow.org/tutorials/images/transfer_learning.

[59] M. Buda, A. Maki, and M. A. Mazurowski, “A systematic study of the class imbalance problem in convolutional neural networks,” *Neural Networks*, vol. 106, pp. 249–259, 2018, doi: 10.1016/j.neunet.2018.07.011.

[60] N. V Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” in *Journal of artificial intelligence research*, 2002, vol. 16, pp. 321–357.