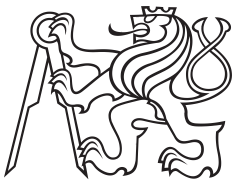
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Automatic Control of the Number and Positions of Weld Studs at Škoda Auto

Erik Pásztor

Supervisor: Ing. Martin Macaš, Ph.D.
Field of study: Cybernetics and Robotics
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Pásztor Erik** Personal ID number: **483595**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Automatic Control of the Number and Positions of Weld Studs at Škoda Auto

Bachelor's thesis title in Czech:

Automatická kontrola počtu a pozic navařovaných šroubů ve Škoda Auto

Guidelines:

The objectives of the thesis are:
1. Propose and implement the system for detection of weld studs from an image of a car part The output should be the position of the weld studs. The detection can be performed using common methods of image processing or using pattern recognition. For training and evaluation use the provided data. If needed, measure additional needed data or generate synthetic data by a proper rotation and shift of the studs in a real image.
2. Implement multiple detection approaches (at least 2) and compare them from the accuracy and time requirements points of view. For each method, compare also different parameter settings.
3. Propose and implement a suitable visualization of the detection results – positions of the detected studs, misplaced studs and missing studs.
4. Integrate the above implementations into a simple GUI application. Use an arbitrary programming language.

Bibliography / sources:

[1] Wu, Bin, Fang Zhang, and Ting Xue - Monocular-vision-based method for online measurement of pose parameters of weld stud. - Tianjin, China, 2014
[2] Miranda, Larnier, Herbulot, Devy - UAV-based Inspection of Airplane Exterior Screws with Computer Vision. - Prague, Czech Republic, 2019

Name and workplace of bachelor's thesis supervisor:

Ing. Martin Macaš, Ph.D., Cognitive Neurosciences, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **11.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until: **30.09.2022**

Ing. Martin Macaš, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

Foremost, I would like to thank my supervisor Ing. Martin Macaš, Ph.D. for his support and valuable insights during the writing of this thesis.

This work was supported by Škoda Auto. I would like to express gratitude to my supervisor Pavel Kocek and everyone from the company who provided the necessary data and information about the given issue.

Special thanks goes to my brother and my parents whose help, encouragement and motivation made this work a pleasant endeavor.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 10. May 2021

Signature:

Abstract

The objective of this thesis was to find a detector which would most reliably find the locations of weld studs in photographs of car parts and create a simple system for comparing these locations to required one. This would lower the time required when manually inspecting parts in an industry setting.

We proposed and described the properties of classic image processing methods and the cascade classifier with Haar-like features. We proposed a classifier ensemble by adding a support vector machine binary classifier to the cascade classifier. Training and testing was done on a personally collected and annotated set of data. Because of the small size of our data set, we did not explore the use of convolutional neural networks.

We judged the performance of the classic methods to be insufficient for real applications. The detector which combined the cascade classifier and support vector machines achieved scores of 8.54% for false detection rate and 72.82% for true positive rate, which is comparable to other works using more complicated detection methods.

We consider the proposed detector to be applicable as a solution to the given task if it was supplemented with an inspection by a human worker to confirm the detection results.

Keywords: computer vision, object detection, cascade classifier, support vector machines, weld studs

Supervisor: Ing. Martin Macaš, Ph.D.

Abstrakt

Cieľom tejto práce bolo nájsť detektor, ktorý by bol schopný čo najspoľahlivejšie určiť polohy navarovaných šróbov na fotografiách častí auta a vytvoriť jednoduchý systém, ktorý tieto polohy porovná s požadovanými. Toto by skrátilo čas potrebný na manuálnu kontrolu častí v priemyselnom prostredí.

Navrhli a porovnali sme vlastnosti klasických metód spracovania obrazu a kaskádneho klasifikátoru s príznakmi typu Haar. Navrhli sme súbor klasifikátorov pridaním binárneho klasifikátoru založenom na metóde podporných vektorov ku kaskádnemu klasifikátoru. Trénovanie aj testovanie bolo prevedené na osobne nazbieraných a anotovaných dátach. Kvôli malému množstvu dát sme neskúmali použitie konvolučných neurónových sietí.

Výkon klasických metód sme vyhodnotili ako nedostatočný na reálne použitie. Detektor kombinujúci kaskádny klasifikátor a metódu podporných vektorov dosiahol hodnoty 8.54% pre mieru falošnej detekcie a 72.82% pre mieru skutočne pozitívnych detekcií, čo je porovnateľné s inými prácami využívajúcimi zložitejšie detekčné metódy.

Navrhnutý detektor považujeme za aplikovateľný ako riešenie daného problému, ak by bol doplnený o kontrolu ľudským pracovníkom, ktorý by potvrdil výsledky detekcie.

Kľúčové slová: počítačové videnie, detekovanie objektov, kaskádny klasifikátor, podporné vektory, navarované šróby

Preklad názvu: Automatická kontrola počtu a pozíc navarovaných šróbov v Škoda Auto

Contents

Figures

Tables

Chapter 1

Introduction

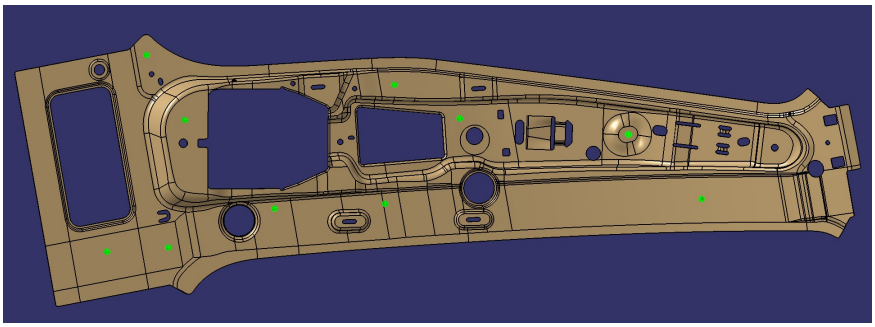
1.1 Motivation and Task Description

Automating manual processes in the automotive industry is important, as computer programs and robots can often work faster and more accurately than human workers and thus they are free for other, more complicated tasks.

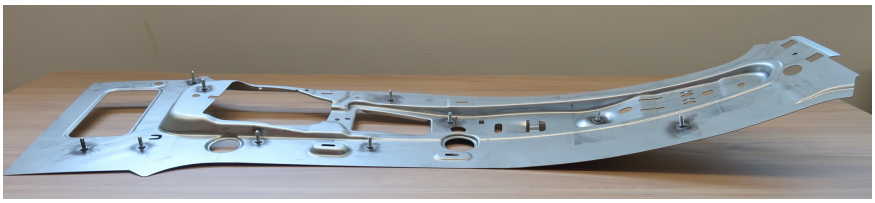
In the automotive industry, during the production of cars, one of the fundamental parts is the car platform. Studs are welded onto the platform and they are used to attach other parts (the biggest being the chassis). They are also welded onto other parts of the car body. In the production line, this process has been automated and is automatically and quickly performed by robots producing hundreds of cars every day. On the other hand, during the development stage, when only a few cars or chassis need to be put together, it would be too expensive to use robots and so this is done manually by human workers. They use stud welding machines to put the welds in place, according to a 3D model. When considering that there are usually several hundreds of studs located on a platform, it can be long and complicated to ensure that all of them are attached safely and in the correct position. An example of the studs is depicted in figure 1.1 and figure 1.2 shows how they look when welded onto a metal part. In subfigures 1.2b and 1.2d, which show a 3D model of the part, the studs are colored green.

Figure 1.1: Photograph of studs

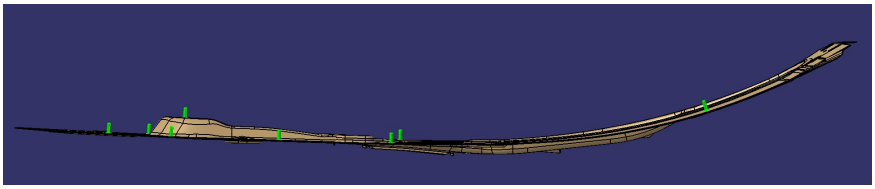
(a) : Photograph from top



(b) : Rendered view from top



(c) : Photograph from side



(d) : Rendered view from side

Figure 1.2: Example of a car part fitted with studs

The goal of this work is to design and implement a robust detector which could locate all studs in an photo of a car platform (or a smaller metal part) and could later be used for checking whether all required studs are present and positioned correctly by comparing the photograph to a predefined model. Currently, the process of visually checking the studs takes several hours and is regularly done by several workers. Contrarily, taking a small amount of photographs does not take a long time and a computer detector can generally run in a matter of minutes and the following comparison between found and required positions is done almost instantaneously, even for large amounts of data. So in conclusion, a well-designed program can reduce the needed time from hours to as little as half an hour and requires less manpower, probably

only one operator.

With the rise in computing power and new algorithms in the last decades, computer vision projects dealing with problems such as this one are ordinarily designed with a machine learning model at their core. These programs involve a high number of convolutional layers with weights which are fine-tuned by a computer with the assistance of a programmer by training on considerable data sets consisting of thousands of samples of the wanted object. Due to a lack of data, we were forced to design a program with a small data set of images of car parts, each containing approximately ten studs.

Because of this, we first tried to implement a method relying on traditional computer vision techniques. In general, they work by applying a set of hand-tuned rules to an image and deciding whether it meets the criteria. As a rule, this approach is less robust and reliable then more modern ones, but can be precise enough for some tasks and requires very little training data. When this approach failed to achieve satisfactory results, we created a machine learning model called Cascade classifier that is similar to neural networks but is simpler and can be trained on smaller data sets.

1.2 Related Work

In [1], the authors developed a method for measuring the position and orientation of a weld stud. Based on a mathematical model, a calibration method was developed and an optimal observation condition was introduced as the constraint in the measurement process to enhance the location precision. They used simple computer vision methods and their method achieved sufficient accuracy and speed.

In [2], a team in France used a drone-mounted camera to inspect screws on the exterior of airplanes. They proposed a convolutional neural network to locate the screws in a video and methods to create a model of screw locations from given data. Then the defined screws and the found ones are matched together using graph algorithms and computer vision methods are applied to evaluate the state of each visible screw and detect missing and loose ones. They successfully demonstrated this system as a proof of concept.

Generally, older computer vision techniques tend to perform well enough in controlled environments with simple goals. This was proven in [3] whose authors used thresholds in HSV color space, a procedure called Connected component labeling and filtering based on the statistics of the items' size to count a group of objects on a photograph. Their method was tested on images of pharmaceutical tablet blisters and boxes filled with bottles, where the goal was to find tablets and bottle caps, respectively, based on their round shape and color contrast with the background. In both cases, the resulting accuracy was almost 100%, only objects occluded by other faulty ones were not counted properly.

The paper [4] dealt with the problem of programming a robot that could assist a human worker with disassembling electric vehicle batteries. The task involved locating screws in a video feed from the robot's camera. The authors used a cascade classifier, similar to our approach. They described sample photographs collected for the training process and their difficulties in achieving satisfactory results.

In a recent work [5], the authors used a camera with auxiliary lighting to take photographs of a car part with attached studs. Their task was to compare the location of every stud on the part to its nominal position and determine if it is positioned correctly but they did not consider missing studs. The proposed solution consisted of taking calibrating images (of a checker board attached above the car part), taking photographs of the part from several angles and cutting out sub-windows based on the expected locations of studs. For each stud, they used the different views to generate a new image in a format called normal map. From this, they were able to determine a stud's 2D coordinates by employing their own convolutional neural network (CNN), which they named "Normal map regression network". These coordinates were then mapped to 3D positions and compared to nominal values. With the best parameter configuration, their method did not make almost any false negative predictions - situations where a correctly placed stud is marked as incorrect. But on the other hand, there were more false positive predictions than true positives. This means that many incorrectly positioned studs were marked as correct.

Other works have dealt with similar problems with conditions different to ours. Most of these either used CNNs or their tasks involved simplifications that enabled the use of simpler methods. [6] achieved precision of over 80% in locating and classifying defects of metal surfaces by employing a CNN and a follow-up multi-class classifier. Part of the work done in [7] was designing a CNN to locate discarded nails and screws in building construction sites. Despite difficult conditions, the authors achieved a detection rate of almost 90%. The team in [8] used a camera mounted on a robot arm to automate the detection of studs in defined locations. They described a simple method using a source of structured light (mounted near the camera) to detected studs with the robot looking at each given location individually. This way, almost 100% accuracy was achieved. [9] and [10] both dealt with finding railway bolts with a camera mounted on the bottom of a train. With unchanging bolt appearance and camera viewing angle, both works achieved near perfect results. [9] employed a simple template matching algorithm, whereas [10] explored the use of CNNs.

Chapter 2

Methods

We use the programming language Python 3.8 and its computer vision package OpenCV 4.4 (available at `http://opencv.org/`), in which most of the following algorithms are implemented. OpenCV is an open source software library including both classic and state-of-the-art computer vision and machine learning algorithms. In section 2.3, we use a machine learning module called scikit-learn (from `https://scikit-learn.org/stable/`) and scikit-image (from `https://scikit-image.org/`) to implement a learning algorithm. The input of our program's main component - a detector - is a photo of a metal car part with weld studs and it outputs the locations of the detected studs in the form of bounding boxes. We implemented two algorithms as a solution to this task. One is an older, classical approach of computer vision and the second is a more modern technique which can be looked upon as an artificial neural network and is trained on a set of data. This is then improved by adding a second, simpler algorithm that also learns on data.

2.1 Hand-tuned Detector

Firstly, we will describe a detector which combines a series of preprocessing steps with a technique that tries to locate objects that meet certain requirements, for example, size and color, which are hand-tuned. Figure 2.1 shows a flowchart of the proposed method.

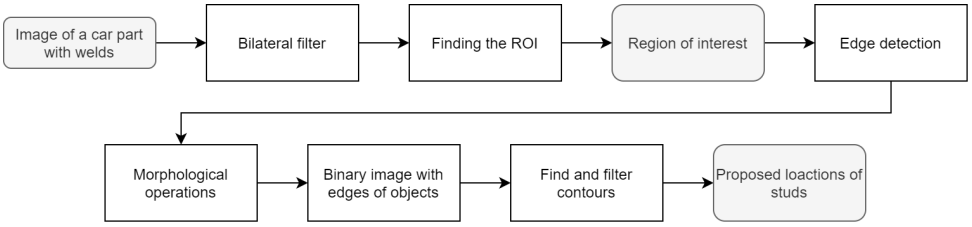


Figure 2.1: Flowchart of the hand-tuned detector

2.1.1 Bilateral Filter

Filtering out noise in a photo is important when trying to find objects in the photo as the noise can decrease a detector's performance. The advantage of bilateral filter is that while reducing Gaussian noise, it has less impact on the edges, and it keeps them sharp. Typical Gaussian filter treats all pixels in an image equally and computes a new value for a pixel as a convolution (or average) of its neighbouring pixels weighted by coefficients with Gaussian distribution.

Bilateral filter [11] combines two Gaussian filters. One with weights as described before, which is a function of space. The second one has coefficients in the convolution's kernel equal to zero in places where the value of the image is similar to the value of the central pixel - it is a function of intensity. This means that it computes the average of smaller patches of the input image which have approximately the same colour. Usually parameters are set, which define how each of the filters contributes to the output.

For a noisy image Y the equation expressing the output of the bilateral filter is:

$$X[k] = \frac{\sum_{n=-N}^{N} W[k,n]Y[k-n]}{\sum_{n=-N}^{N} W[k,n]} \tag{2.1}$$

which is a normalized weighted average of a neighborhood of size $2N+1$ pixels around the k-th input pixel. Weights $W[k,n]$ are computed by multiplying two factors:

$$\begin{aligned} W_S[k,n] &= exp\{-\frac{d(k,k-n)^2}{2\sigma_S^2}\} = exp\{-\frac{n^2}{2\sigma_S^2}\} \\ W_R[k,n] &= exp\{-\frac{d(Y[k],Y[k-n])^2}{2\sigma_R^2}\} = exp\{-\frac{(Y[k]-Y[k-n])^2}{2\sigma_R^2}\} \end{aligned} \tag{2.2}$$

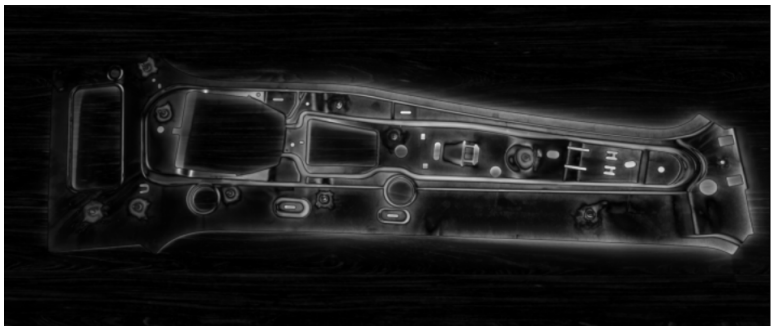
where d denotes Euclidian distance between two points or two intensities.

2.1.2 Saliency, Contours and Region of Interest

We apply an algorithm to detect visually salient features in an image, proposed in [12]. It outputs a grayscale image called saliency map where higher values signify locations of change in the original image. Input and output of this algorithm is shown in figure 2.2.

Next, we use the morphological operation dilation followed by erosion, which closes up small holes in the image and keeps the width of lines the same. Morphological operations work by applying a kernel of a given size to every pixel in an image and determining a new value for this central pixel based on the values around it, much in the same way as the convolution. For erosion, the new value of a pixel will be calculated as a local minimum of the pixels in its neighbourhood. This serves to remove unwanted small patches in the image, which can be a product of poor photograph quality or some

(a) : Original photo



(b) : Saliency map

Figure 2.2: Output of saliency algorithm

of the previous methods. However, it also makes all edges thinner, so large kernels should not be used. For dilation, a pixel's new value is determined as a local maximum of its neighbourhood. It is used to make objects more pronounced and fill up small holes inside them. The downside is that it can close the gaps between objects that are supposed to be separated. One step of erosion and dilation, respectively, can be expressed as:

$$E[x, y] = min_{(x',y')}O[x + x', y + y'] \tag{2.3}$$

$$D[x, y] = max_{(x',y')}O[x + x', y + y'] \tag{2.4}$$

where O denotes the original, E the eroded, and D the dilated image, x and y are coordinates of the central pixel and x' and y' coordinates of the kernel - they would, for example, range from -1 to 1 for a kernel of size 3. We used kernels of size 5. Lower dimensions proved to be ineffective in closing the gaps in photographs with given resolutions and larger kernels produced results where components that were clearly not meant to be connected were made into one continuous patch.

Finally, we use the contour-finding algorithm from [13]. This looks for continuous lines in image pixels with similar colour close to each other. We use it to find the edge that encloses the largest area in the grayscale image, which we suppose is the car part, and determine its bounding rectangle. We use it to cut out a section of the original photo. This is our region of interest (ROI), which we work with in the following parts. An example of a found ROI is shown in figure 2.3.

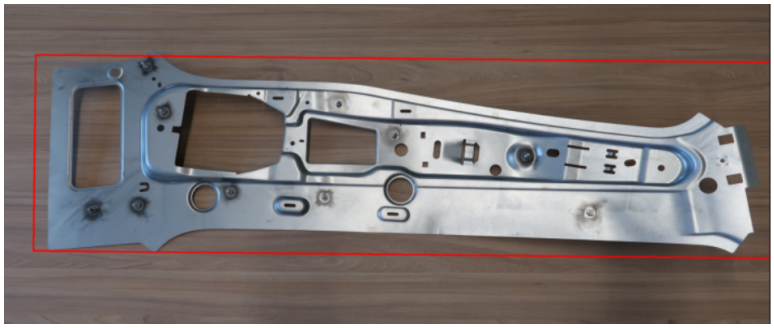


Figure 2.3: Marked ROI

2.1.3 Canny Edge Detector

We apply the Canny edge detector [14] to a grayscale image. It computes the gradients of intensity in the horizontal and vertical direction for each pixel in the image. This is done by computing convolutions with two 3x3 kernels, expressed as:

$$G[x,y] = \sum_{x'=-1}^{1} \sum_{y'=-1}^{1} K[x',y']O[x+x',y+y'] \tag{2.5}$$

where G represents the gradient image, x and y are the image coordinates, x' and y' are kernel coordinates, and O is the original image. Kernels K are given by the matrices:

$$H = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad V = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2.6}$$

H is the kernel for horizontal gradient and V for vertical. From these two values, it determines the magnitude of the overall gradient (the square root of sum of squared partial gradients) and its angle (arctangent of partial gradients). By reasoning that the edges are perpendicular to the gradient (and filtering based on the gradient's magnitude), we find the edges.

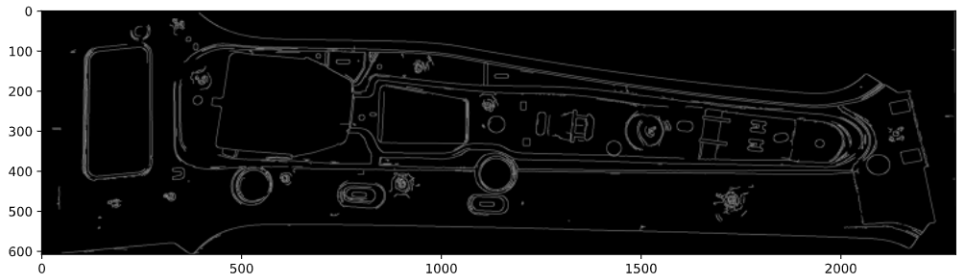
This algorithm takes two threshold values as input: $minVal$ and $maxVal$. Any pixels with an intensity gradient greater than $maxVal$ are sure to be edges and those below $minVal$ are sure to be non-edges. Pixels with gradient between these two thresholds are classified based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are discarded.

For each input image, we determine the thresholds according to the equations

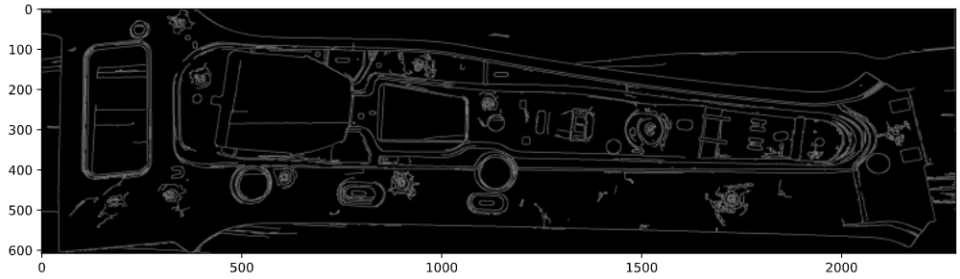
$$\begin{aligned} minVal &= max(0, (1-\sigma) \times v) \\ maxVal &= min(255, (1+\sigma) \times v) \end{aligned} \tag{2.7}$$

where v denotes the mean value of intensity in the input grayscale image and σ is a parameter controlling the width of the area between the thresholds. The higher its value, the more edges the algorithm finds. By default, we use $\sigma = 0.33$. Using the mean intensity value for determining the thresholds eliminates the need to manually adjust the thresholds when processing images with different lighting.

Figure 2.4a shows the output of the Canny detector applied to the ROI with the value $\sigma = 0.33$, 2.4b shows the output for $minVal = 10$ and $maxVal = 100$, which is too wide. When comparing these two images, we can see that 2.4a contains marginally less noise then when using wide thresholds but still retains important objects.



(a) : With minVal and maxVal set by σ



(b) : With minVal and maxVal set too far apart

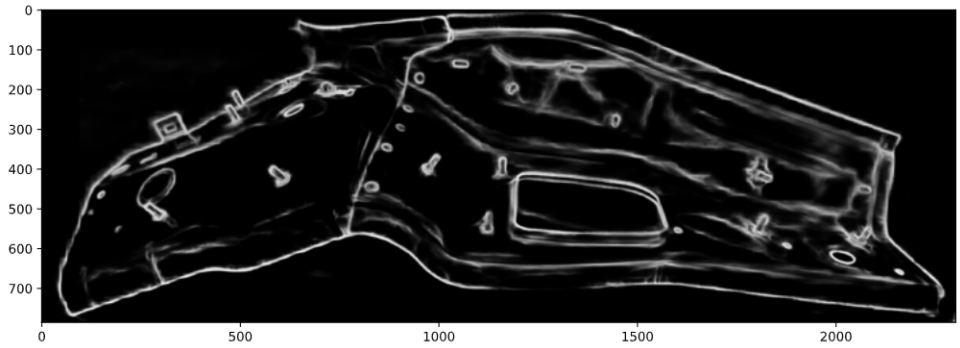
Figure 2.4: Output of Canny detector

2.1.4 Holistically-nested Edge Detection

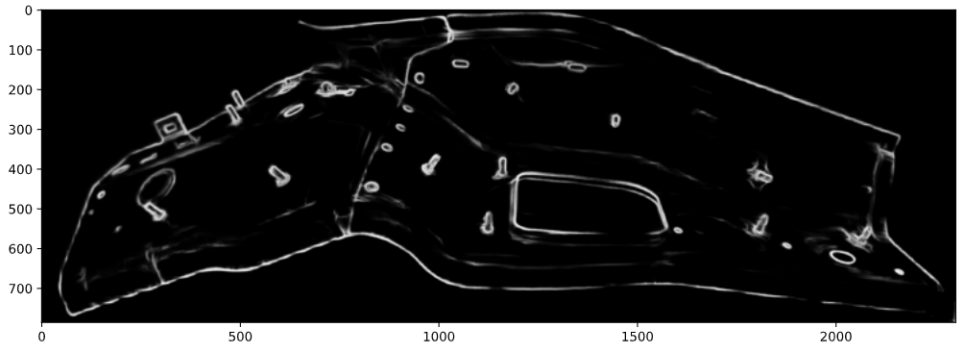
Holistically-nested edge detection (HED) is a learning-based end-to-end edge detection system that uses a convolutional neural network to find edges in RGB images created in [15]. HED makes use of the side outputs of intermediate layers. The output of earlier layers is called side output and the output of all 5 convolutional layers is fused to generate the final predictions. Since the feature maps generated at each layer are of different sizes, it is effectively looking at the image at different scales. Due to the lack of data, we were not able to retrain the network to our specific task, so we used the model that was trained as part of the original work. It is available from [16].

HED works on RGB images and takes a mean value for each colour channel

of an image as an input parameter. The model that we use was pretrained with mean equal to 104, 116 and 112 for R, G, and B channel, respectively. When applying this network to an image, the result's quality depends on how closely the mean values used as input match those that the network was trained with and how well they represent the colours in the image. Using the original produced images that contained noise but edges were clearly visible. Achieving a perfect prediction does not seem possible without retraining the network, but we found that using mean values of 130, 120 and 75 removed almost all noise and kept most of the edges. In section 2.1.5, we used a combination of these two outputs to get a better approximation. Figure 2.5 shows the output of HED applied to an image with the original and our values of mean colour. It can be seen that the second one contains less noise and this is therefore considered as a better result.



(a) : Output of HED with the original mean values



(b) : Output of HED with mean values [130, 120, 75]

Figure 2.5: Different outputs of HED

2.1.5 Finding the Studs - Contours

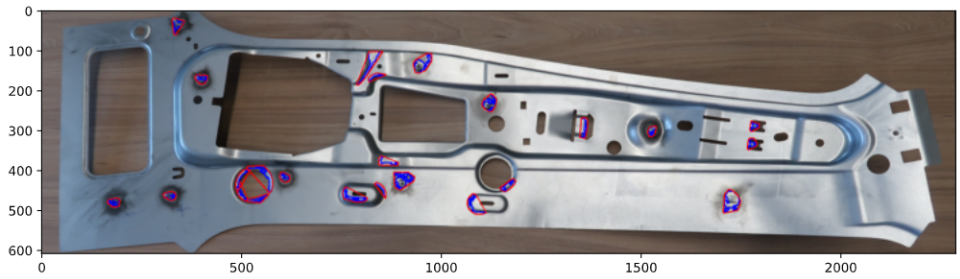
We apply one of the edge detectors to ROI image and after using morphological filters, we find the contours. We can filter the contours by different criteria, such as length, dimensions of the bounding rectangle or shape of convex hull.

Examples of filtered contours and their convex hulls are displayed in figure 2.6. The contours in 2.6a were acquired by using the bilateral filter with

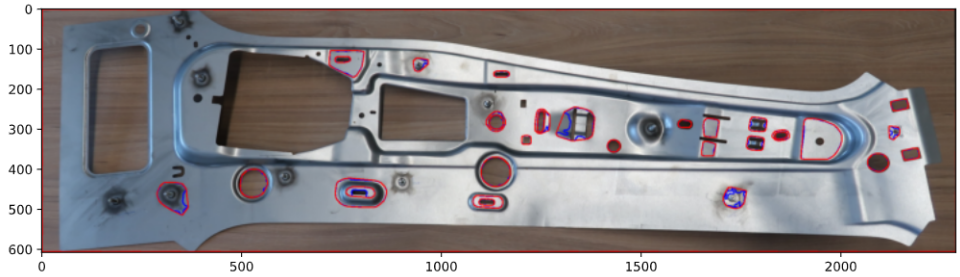
parameters $N = 5$, $\sigma_S = 50$, $\sigma_R = 50$, applying the Canny detector with $\sigma = 0.33$ followed by the morphological operations of dilation and erosion with kernel size 3x3. To get the ones in 2.6b, we used filter with the same parameters, HED algorithm with mean values 130, 120 and 75 for R,G and B channels and two passes of morphological erosion and one dilation. In both cases, the found contours were filtered by the following criteria:

$$
\begin{aligned}
500 <= w \times h <= 5000 \\
0.25 <= \frac{h}{w} <= 4 \\
\frac{w \times h}{a} <= 4 \\
hl >= 400 \\
\frac{a}{h} > 0.2
\end{aligned}
$$

where w and h are width and height of bounding rectangle, a is area of contour and hl is area of convex hull. These were the parameters which performed best on the tuning set and were used for evaluating the detector in chapter 3.



(a) : Canny detector



(b) : HED

Figure 2.6: Output of the contour finding algorithm

The output of this procedure is a list of bounding rectangles for the detected studs .

2.2 Cascade Classifier

Secondly, we describe a detector that is more similar to modern techniques because it is trained on a set of data but requires fewer images to learn and perform well. Figure 2.7 shows the flowchart of the different stages of working with this method.

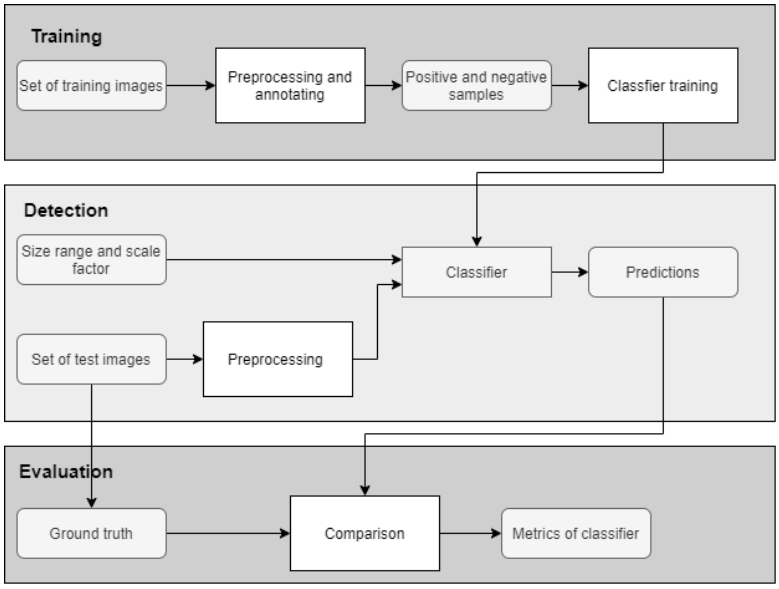


Figure 2.7: Flowchart of the cascade classifier

2.2.1 Basic Characteristics

Methods called classifiers are, in general, used to assign a class to an input observation. However, they can act as object detectors when classifying one object class against a background. One such classifier was described in [17] and [18] which introduced a technique called the cascade classifier. It is a machine learning based approach where a function is trained from a lot of positive and negative samples. However, the number of such training examples is often not big enough for training a modern neural network. Similarly to neural networks, the basic operation of this technique is convolution of a target image with many different kernels (the convolution is expressed in 2.5). The cascade classifier uses simpler kernels,figure 2.8 shows three of the basic kernels. Using convolution with these kernels, the values of Haar-like features are computed from an image and compared to previously learned thresholds. One such feature is considered to be a weak classifier, so more of them are needed to get a more accurate prediction. The overall classifier considers the values of all its weak classifiers when examining an image. More complicated kernels are created as all the possible combinations of two, three, or four rectangles in a window of given size, which was originally 24x24

pixels and can be set before training the classifier. Furthermore, kernels can be formed by rotating these combinations by multiples of 45°. Black parts are mathematically represented by positive numbers and white ones by -1. Particular positive values, which are integers, are set during training, when these kernels are combined into stages to find those classifiers that produce the clearest distinction on the training data.

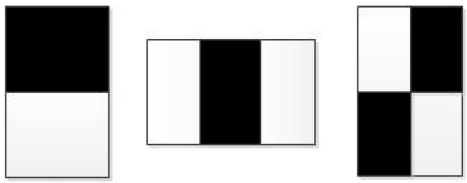


Figure 2.8: Basic kernels for computing Haar features

The authors created several improvements to make this algorithm run faster. First of these is an image representation called integral image - every pixel in this representation is the sum of pixels to the left and above of it in the original image. This can be calculated in one pass of an image by using two recurrent equations:

$$\begin{aligned} S[x,y] &= S[x,y-1] + O[x,y] \\ I[x,y] &= I[x-1,y] + S[x,y] \end{aligned} \tag{2.8}$$

where S denotes a cumulative row sum, O the original image, and I its integral representation. This greatly reduces the number of operations needed during convolutions. For example, when using the left feature from figure 2.8 with size 4x4, we can simply find the sum of the two rectangles under the kernel with the use of six points from the integral image and four subtractions, and then subtract these two numbers instead of using fifteen operations on the original image. Their second improvement is discarding those combinations of kernels that perform poorly. This is done by training the detector in stages (a stage is combination of several kernels) and setting an increasing limit to their precision to select the right kernels. The AdaBoost algorithm (first described in [19]) is used to optimally select and group kernels (or weak classifiers) into stages and so create a strong classifier. The final concept was applying the learned kernels in a cascade manner similar to training when performing classification. It means that if an image's region does not achieve a good score in the first stage, this region is not considered in future stages. It improves performance by not focusing on patches of the image where an object is unlikely to be. This, of course, requires creating these stages during training. The original work had more than six thousand features divided into thirty-eight stages, each containing an increasing number of features, starting with one, ten, and twenty-five in the first stages.

2.2.2 Training Data Requirements

A cascade classifier is trained on a set of images that contain the target objects - positive samples and a set where the object does not occur - negative samples. If these images contain multiple colour channels, they are converted to grayscale format when the training is initialized. All positive samples should have the same ratio of side lengths and contain as little background as possible. We chose to use square samples as it was the best fit for most of the studs in our data set. The negative samples can have varied sizes and they should contain background specific to the task. Before training, the kernel size needs to be specified. This should be set to the biggest size of the target object we expect to find in the test data and should keep to the same ratio as the positive samples. The need for keeping the ratio comes from the fact that the positive samples are scaled to the size of the sample window and this would cause image distortion for samples that originally had different ratio of sides. Kernel size also determines the size of the sliding window in the convolution and the number of different features that can be created.

As seen from the description, this approach employs convolutions similarly to modern neural nets, but with simpler kernels and without so much additional data manipulation that those nets have. Theoretically, this results in less accurate predictions but also in faster training and detection times and, most importantly in our case, smaller training set size requirements. Authors of [4] used this method to detect screws in a video feed of car batteries with a training data set containing over 2000 positive images and over 3000 negative ones. However, they were not successful, their model detected only about fifty percent of screws when testing. On the other hand, the authors of [20] used a pre-trained model, which is available as part of the OpenCV library, to detect human faces in photographs. They achieved near-perfect detection rate. Training data set for this model is not available, so we do not know how many images it contained. However, we had not collected nearly as many images as even in the first mentioned work, so we did not expect such good results.

From the available photographs, we created a data set split into the required positive and negative samples. The OpenCV implementation of the training algorithm takes as input a set of negative images with a text file with names of the corresponding images and a set of images with a text file which contains the names of these images, the number of positive samples located in each image and position of the samples. A utility program then converts the text files into .vec format, which contains the positive samples cut out from the original images and scaled to the kernel size. This format is then used in training.

As part of this format conversion, OpenCV offers the ability to create more positive samples by applying a perspective transformation (like skew and rotation) on a given sample, but we decided against using this because theses transformations did not seem plausible to occur in real photographs.

This tool simply places a given positive sample on a selection of background images and rotates or skews them randomly. It does not manipulate the original background of the positive sample, so when the target image does not take up the whole rectangular sample, it creates a distinctive change of image properties in the new samples. Instead of this, we created new training images as described in section 2.2.4.

2.2.3 Histogram Equalization

When dealing with images that have different lighting conditions or when parts of one image are illuminated improperly, histogram equalization can be used to normalize them, therefore increasing the quality. The goal of this operation is to take an image whose histogram has high spikes and manipulate its values in such a way that the histogram becomes more evenly spaced. When viewing the image, it should improve its contrast. In our case, this is useful because it makes the studs more distinguishable in poorly illuminated parts of photographs. It should also produce images with similar characteristics from photographs taken with various lighting. This does not work well when applied to an RGB image as it changes the distribution of each colour separately. It can also be beneficial to apply a filter to an image ahead of the equalization. Thus, we applied the bilateral filter from section 2.1.1 to images and converted them to grayscale format before applying this technique.

The basic version of this algorithm works by scaling the input image using the cumulative distribution function of its histogram. This can be expressed through equations [21]:

$$h[i] = \sum_{x=1}^{N} \sum_{y=1}^{M} \begin{cases} 1, & \text{if } I[x,y] = i \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

$$CDF[j] = \sum_{i=1}^{j} h[i] \tag{2.10}$$

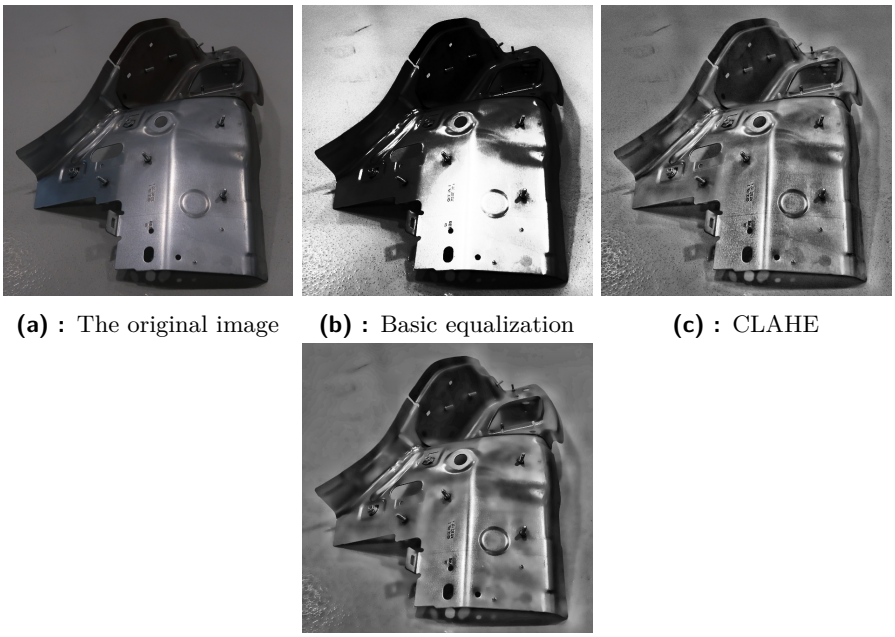
$$O[x,y] = \frac{CDF[I[x,y]] - CDF_{min}}{N \times M - CDF_{min}} \times 255 \tag{2.11}$$

where h is the histogram, I and O are the input and output images with size $M \times N$, CDF is the histogram's cumulative sum distribution and CDF_{min} is the the minimal nonzero value of this distribution.

This implementation does not generate usable outputs in most real-world situations, because it looks at the input as a whole and can produce areas that are too bright to compensate for areas that were originally too dark. Contrast limited adaptive histogram equalization (CLAHE), introduced in [22], improves upon this, by applying the same principle as before on smaller patches of the input image. To prevent amplification of noise, it computes a

histogram for each image patch and clips the values that are above a given limit and uniformly distributes the excess pixels into other histogram values before starting the equalization. In the OpenCV implementation, the patch size can be set as p_s and the limit as c_l. Throughout the work, we used values $p_s = (30 \times 30)$ and $c_l = 5$.

Figure 2.9 shows a dark image and compares different versions of the histogram equalization applied to it after converting it to grayscale format. It is clear that using figure 2.9b in further processing would be impractical, but figures 2.9c and 2.9d are an improvement over the original image. Then figure 2.10 shows the histograms of the original grayscale image and two variants of equalization (without the filtered image). The intended effect of flattening the spike described previously can be seen between the red and green lines.



(a) : The original image

(b) : Basic equalization

(c) : CLAHE

(d) : Bilateral filter and CLAHE

Figure 2.9: Histogram equalization

2.2.4 Data Set Preparation

First, we downscaled most of the photographs by a factor of two to four to speed up training - the original resolution was almost 18 MP in some cases. Since this detector can be sensitive to object rotation (it would detect the object only with rotations that have been seen in training), we extended our training data set by rotating each image by either 90° or −90°, which doubled the amount of available images. We rotated half of the images one way and half the other.

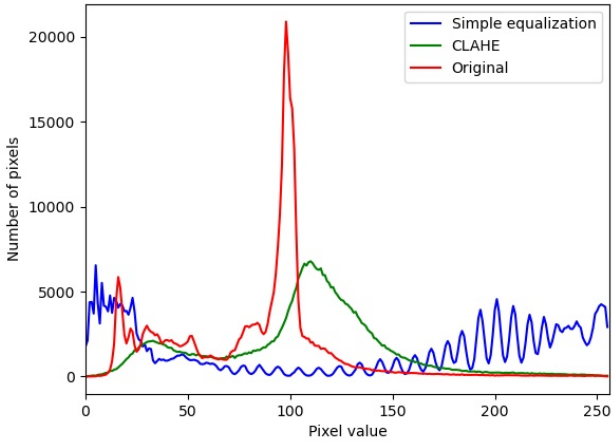


Figure 2.10: Histograms of images

To compare the detector's performance with different data formats, we created two copies of the training data set. We kept one set without any alterations and used the bilateral filter described in section 2.1.1 (with parameters $N = 12$, $\sigma_S = 25$, $\sigma_R = 25$) on the others. Then we converted one of them to grayscale format and applied the CLAHE algorithm from section 2.2.3 (with parameters $p_s = 30 \times 30$ and $c_l = 5$) to it. For the second copy, we applied the HED algorithm from section 2.1.4, using mean colour value [130, 120, 75] because, as can be seen in figure 2.5, it retained more information about studs. This meant we had the original RGB image set (which were converted to grayscale format when training), a set of preprocessed grayscale images, and a set of binary images of edges.

We had created annotations for the original data before and we used the same ones for the different versions of the training data. These were all squares for the reasons explained earlier. Then we cut out sub-images without studs for the negative samples. Image 2.11 shows how we divided one of the images, green squares signify positive samples, and red rectangles are negative samples. We repeated this for all three versions of the training data set.

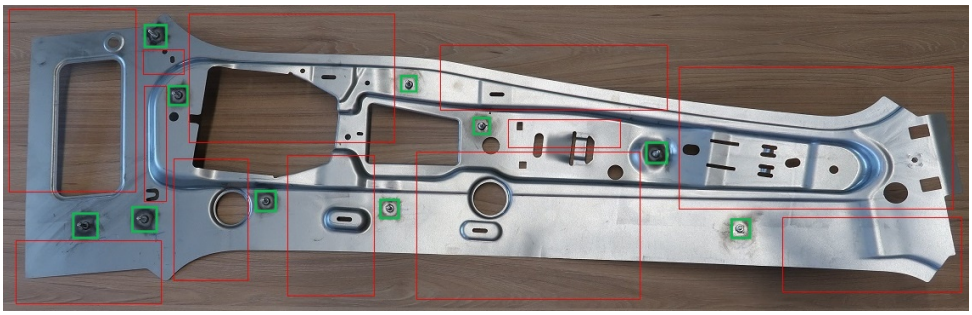


Figure 2.11: Example image split into positive and negative samples

2.2.5 Training Parameters

The OpenCV implementation of the cascade classifier takes several parameters for training (apart from the training data):

- Number of positive and negative samples to use during each stage of training.
- Number of cascade stages to be trained.
- Value limiting how precise should the detector attempt to become on the training data. The lower this is, the better the classifier will be fitted during training and setting it correctly can prevent overtraining. Precision increases drastically when a small set of positive samples is used.
- Width and height of positive training samples and the (this specifies the kernel size).
- The desired minimal hit rate (correct predictions divided by all predictions) and maximal false alarm rate (incorrect predictions divided by all predictions) of each stage. During a stage, these values usually start at 1 and decrease with training. When either of them reach their limit, training of the stage stops.
- The maximum number of features in each stage. A stage keeps learning and adding features up until this number unless it reaches one of the above limits.
- The maximum depth of search-trees in the AdaBoost algorithm.
- The specific variant of AdaBoost to use (Discrete AdaBoost, Real AdaBoost, LogitBoost, Gentle AdaBoost).
- What types of features (described in section 2.2.1) should be explored can be only the basic upright ones (depicted on figure 2.8) or all their rotated versions.

During training, positive samples and random cut-out parts of negative samples are shown to the classifier and it decides if there is an object or not. Hit rate and false alarm rate are updated based on the correctness of this prediction. Based on the estimation in [23], these values scale exponentially with the number of stages trained. When a positive sample is identified as negative, the sample is discarded from training in future stages and is replaced by another one so that the number of positive training samples is constant as set in the beginning. After each stage, the program goes over the negative samples with the already learned stages and computes the percentage of those marked as positive. If this is below the accuracy limit, it means that the model has overtrained and the training stops. Using more positive

samples results in a slower decrease of this metric. Moreover, increasing the search-tree depth generally improves the robustness of the classifier during training as it can search more combinations of features when it achieves an unsatisfactory score in a stage. This can consequently lead to overfitting and significantly lengthens the time required to train the classifier. Similarly, setting a larger kernel size results in more features, therefore possibly training a better classifier, but it takes longer to train. With these systems combined and with the restricted training data set, we need to experiment with different volumes of positive data to be used.

2.2.6 Detection

We can use the trained cascade classifier to predict the locations of studs in test images. The detection function in OpenCV creates downscaled versions of the input image and applies the detector to each one. The downscaling percentage and the lower limit of image size are taken as inputs. This limit, together with the third input, creates a boundary for how large the detected object can be. This compensates for the fact that the classifier has a fixed size of the trained kernels and can only detect objects of approximately the same size. Downsclaing is performed in order to find objects that are originally larger than the trained window. Resizing the image by a factor larger than one before applying the classifiers also proved to help in detecting smaller objects. The range of object sizes for the classifiers to consider was determined based on the sizes of bounding boxes in the training set, the limits were rounded to 10 and 40 px. These sizes were prescaled by the same factor as images before detection. When the classifier has searched all versions of the image, it filters the found bounding boxes. This is done based on the last input parameter, which specifies how many neighbouring boxes should a detection have to be considered positive, similar to non-maximum suppression. After this, the detector outputs bounding boxes of areas, where the target objects are located.

Apart from simple detectors, we implemented one that combined three trained cascade classifiers, each trained with different parameters, in such a way that increased prediction quality while keeping the required time relatively low. This is done by applying a classifier to the whole modified input image, and then using the other two classifiers to the areas marked by the first one. If both of these produce positive classification, an area is proclaimed to contain a stud. If only one identifies a stud in the area, it is considered uncertain. Figure 2.12 shows a flowchart of this ensembled detector. If the input image is larger than the training images (all of those were smaller than 1.11 MP), its dimensions are first scaled down. The ratio is calculated from the original $width$ and $height$ as $r = \sqrt{\frac{1.11 \times 10^6}{width \times height}}$. This makes it more probable that the studs will have sizes similar to those in the training set and saves time depending on the original size. It is then preprocessed in the same way as the training images of the first classifier

and passed to the classifier which determines areas with potential studs. The areas are then enlarged by 5 px on all sides to compensate for imprecise bounding boxes and cut out from the images preprocessed for the next two classifiers which produce the final decisions for each area. When classifying whether an area contains a stud, the upper limit of a detected object's size in these secondary classifiers is the same as the area's. The lower limit is one eighth of the upper one.

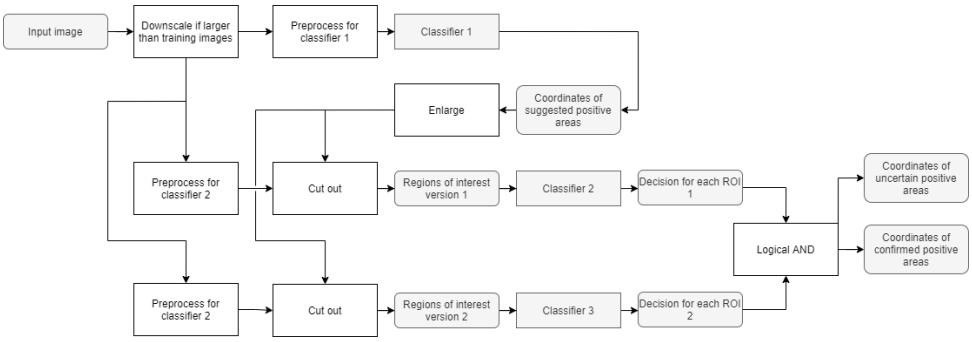


Figure 2.12: Flowchart of the detection process using an ensemble of three cascade classifiers with different parameters

Section 3.2.4 contains specific values used for the parameters in both training and detection that were not stated in this section.

Using another type of classifier instead of the second and third cascade classifiers in this configuration seems like a good option to explore. We examine this possibility in section 2.3.

2.3 Combining with Support Vector Machines

Performance of the ensemble of three cascade classifiers (depicted in figure 2.12) was significantly corrupted by a high number of false positive predictions - empty areas that were marked as if they contained a stud. This motivated us to put another type of classifier in place of the two secondary cascade classifiers. This should, in theory, correct some of the faults that went unnoticed before because two classifiers of the same kind probably focus on similar features and produce similar errors. We decided to implement support vector machines (SVM) machine learning model as a binary classifier as it is relatively simple and easier to train then a general detector. A binary classifier takes an input image and assigns one of two predefined values to it.

Figure 2.13 shows the flowchart of the complete detector with this alteration. The process makes use of a cascade classifier that has already been trained (the same one in both training and detection stages).

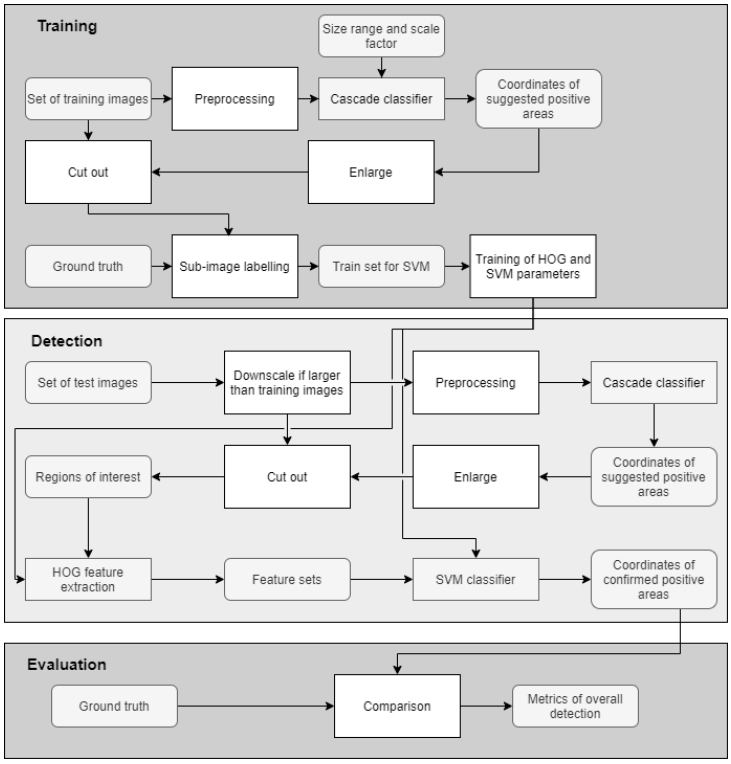


Figure 2.13: Flowchart of the cascade classifier supplemented by SVM

2.3.1 Description

SVM method was first described in [24]. It was designed as a binary classifier to learn from a given training data divided into two and then classify other images as one of the categories. The component that is being learned is a hyperplane (or rather its mathematical representation) that best separates the two groups of data. Hyperplane is a geometric expression, meaning a subspace whose dimension is one less than that of its ambient space - a line in 2D, a plane in 3D, etc. They can be represented by affine equations in the form: $a_1x_1 + a_2x_2 + a_{n-1}x_{n-1} = b$ for n-dimensional space (where a_i and b are the hyperplane's parameters and at least one of a_i must be different to 0).

In the most simplistic form, this problem could be seen as finding a line separating two sets of points (two classes) in a 2D plane. If there is such a line, we can define one point from each set that is closest to the line. Then we can measure the distance between an arbitrary line and its corresponding closest points and find the line that maximizes this criterion while still dividing the classes. The two points are called support vectors, because, when a line is set, removing either of them leads to changing the value of the criterion and potentially altering the best line.

If the points are not linearly separable, as is the case in most problems, the authors use a non-linear function (called kernel) that maps the input vectors into a high-dimensional space where they can be separated by a hyperplane.

This can also involve warping of the original Euclidian space - for example, a circle can be seen as a linear object when viewed through polar coordinates. They also implemented the idea that the hyperplane does not have to precisely divide two sets of vectors, if it is not possible without transforming them into a space whose dimension is too high. In such a case, a soft margin hyperplane is introduced. This is found by minimizing the number of training errors (vectors that are separated from their corresponding set) while maximizing the distance between the hyperplane and its support vectors.

The training and testing data can generally be of any type. However, if the images are directly used as inputs, the classifier would treat each pixel as an individual feature of the input data and this typically produces poor results. It also means that for an image with dimension 25x25 and three colour channels (this is the average size of sub-images in section 2.2.6 that we eventually want to classify), the classifier would work with 1875 features, which is a high number and could lead to slow training and overfitting. Therefore, a set of features needs to be extracted from each image that better describes its properties. It is similar to what kernels did in the cascade classifier in section 2.2.1. In the following section, we describe a method that we implemented to do this for SVM.

2.3.2 Feature extraction

The authors of [25] introduced a method called histogram of oriented gradients (HOG), which they used (in combination with the SVM algorithm) to detect pedestrians in images.

The first step of this technique is preprocessing. To end with a feature vector of a manageable size, the input needs to have a certain size. How much scaling is needed depends on other parameters and is usually a subject of training together with a classifier. The authors then suggest using gamma correction (also known as power law compression) on the input image and they showed that it can slightly improve the output quality. For each colour channel, this process normalizes each colour channel by dividing its contents by the maximum possible pixel value (usually 255), then applying the square root operation to them and multiplying by the maximum value. This reduces the effects of illumination variations in an image. The following steps can work with both colour and grayscale images, but the algorithm was shown to perform better on RGB images.

Next, the horizontal and vertical gradients are computed for each pixel and from these, we get the magnitude and direction of the gradient (again, for each pixel). This is done in the same way as for the Canny detector in section 2.1.3, but only gradient angles in the range $0° - 180°$ are allowed (others are

shifted by 180°) and simpler kernels are used:

$$H = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix} \quad V = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{2.12}$$

For multi-channel images, only the gradient with the largest magnitude and its corresponding direction are taken from each pixel.

Then the image is divided into cells, each with size p in pixels (the size is one of the input parameters). This means that for a cell of size 8x8 px, we will have $8 \times 8 \times 2 = 128$ values. A histogram is created, its horizontal axis contains n angles equally spaced between 0° and 180°, where n is an input parameter. The vertical axis contains sums of magnitudes of those gradients that have the corresponding directions. If a gradient's angle falls between two values, its magnitude is proportionally divided between them.

In the final step, blocks of cells are created and histograms for cells inside a block are concatenated. So where we had n values for each cell, we now have a vector of $m \times n$ values for each block (m is the number of cells in a block taken as input) as a vector. Blocks are created in a sliding window way with step of one cell. Every vector v is normalized in the usual way:

$$\begin{aligned} L &= \sqrt{\sum_{i=1}^{m \times n} v_i^2} \\ v_i' &= \frac{v_i}{L} \end{aligned} \tag{2.13}$$

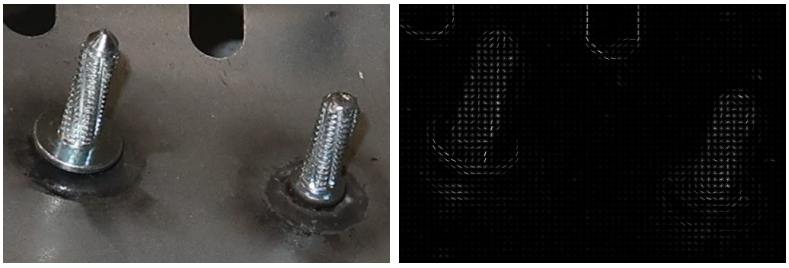
Then the vectors v' are concatenated to form the output feature vector. Its length can be calculated as

$$\begin{aligned} H_b &= m_x \times m_y \times n \\ B &= \lfloor (\frac{X}{p_x} - 1) \rfloor \times \lfloor (\frac{Y}{p_y} - 1) \rfloor \\ F &= H_b \times B \end{aligned} \tag{2.14}$$

where H_b is the size of a histogram in each block, X and Y are image dimensions, $\frac{X}{p_x}$ and $\frac{Y}{p_y}$ are numbers of cells in horizontal and vertical directions and B is the number of created blocks. The feature vector can be viewed as an image. Figure 2.14 shows a photo of studs and its corresponding HOG representation computed with the default scikit-image parameters ($p = (8 \times 8)$, $m = (3 \times 3)$, $n = 9$) and the gamma correction applied.

2.3.3 Training

When scikit-learn is used to train SVM, several parameters are set for the training algorithm. C is a regularization parameter, the strength of used regularization is inversely proportional to C. Regularization means that a



(a) : Input **(b) :** Output

Figure 2.14: Example of HOG

penalty is added to the training error of each explored set of SVM coefficients. In this case, the penalty is the sum of squares of the coefficients (this is called L2 regularization). For larger values of C, a smaller margin (explained in section 2.3.1) will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy.

The second parameter, *kernel*, indicates the type of function that we want to use when mapping the data to higher-dimensional spaces. This can be set to a linear, polynomial, sigmoid, or radial basis function (RBF) - this can produce complex elliptical shapes with multiple folds from linear lines in the Euclidian space.

Parameter *gamma* is the last relevant parameter. It defines the weight assigned to each individual sample in training, again the weight is inversely proportional to *gamma*. Low value means that each sample will influence the coefficients of the learned function strongly and this can lead to problematic training, as the coefficients would change drastically at each training step.

We used grid search optimization (also implemented in scikit-learn) to find the best performing combinations of parameters for HOG feature extraction and SVM training. This type of optimization takes as input possible values for each desired parameter, trains a classifier on a given training data set with each possible combination, and compares the results of classification on a validation subset of the given training data which was not used for training. To achieve more accurate metrics, k-fold cross-validation is used. This means that the given data is divided into k sets and a classifier is trained k times, each time with a different validation subset and a train set of the remaining $k-1$ subsets and the average score is taken. The parameter k is an input parameter of the optimizer method (we used $k = 3$ for training). It can compare the results based on multiple criteria, we chose accuracy - which is the most general criterion available. The detailed values of these parameters are listed in section 3.2.5. We left the other unmentioned parameters of each algorithm here with their default values.

Chapter 3

Achieved Results

3.1 Data Set

We collected 46 photographs with 584 studs and divided them into a training set of 38 images with 483 studs and a test set of eight images with 101 studs. Figures 3.1 and 3.2 show examples of images in the training and test data sets.

Data collection was done in the development and construction department of Škoda Auto factory. We picked metal parts that were suitable for our task. These contained both flat parts and bends and had elements that could look similar to studs in a photograph, for example, circular holes or metal fasteners made for installing screws. Then we used a stud welding machine to attach the studs. We selected most of the locations so that the studs would be clearly visible. Some of them were close to bends in the material and this resulted in occluded studs when viewed from certain angles. Occlusion means that another object prevents us from seeing the stud completely and only a part (usually the threaded top part) of it is visible. As we were not permitted to use a camera ourselves, we specified the desired viewing angles and the company's personnel took the photographs. We included several views per part, for example, some where the studs were easily discernible, as well as those with occlusions or studs close to unwanted objects like holes. There were studs photographed from both top and side perspectives. On different occasions, we were not able to replicate constant lighting conditions and backgrounds, which resulted in differences in the illumination of some of the images. Image sizes ranged from 3.8 MP to 14 MP depending on the camera that was used. Before working with them, we downscaled the images to approximately 1 MP.

Before creating the detectors, we created annotations marking the locations of studs in all images. We used the built-in annotation tool from OpenCV and kept the shape of the annotations as a square.

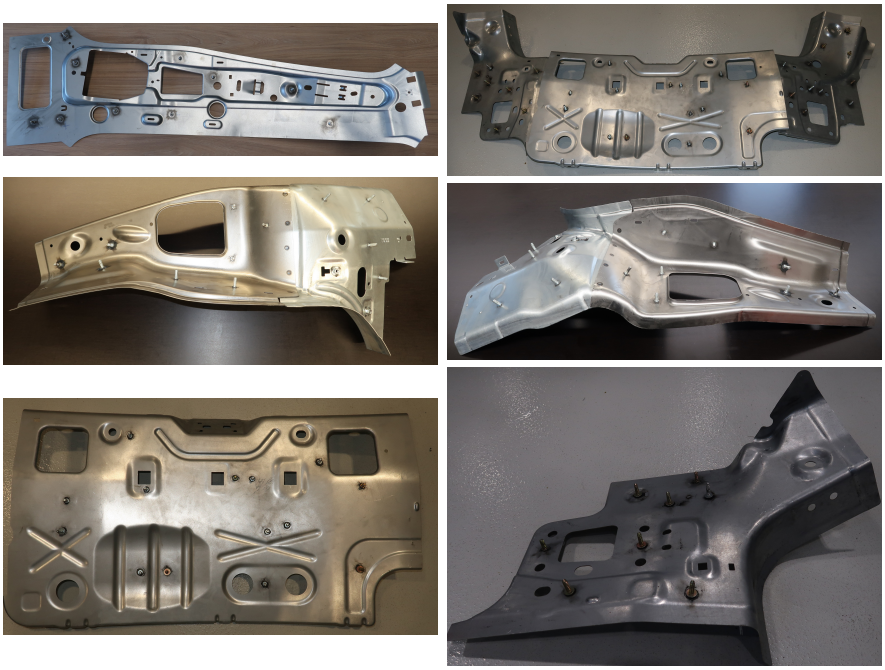


Figure 3.1: Examples of images from the training data set

3.2 Training and Test Results

During testing, a laptop with an i5 2.3 GHz processor and 8 GB of RAM was used. As the methods were not implemented for use on GPU, we only used the CPU.

3.2.1 Metrics

Metrics used for evaluating the detectors are true positive count (TP), which is the number of weld studs which the detector correctly identified, false positive count (FP), which is the number of undetected studs and false negative count (FN), which is the number of objects incorrectly classified as studs. They are calculated by comparing the locations of the studs produced by the detector with the manually annotated locations of the input image. From these, the true positive rate (TPR) and false negative rate (FNR) are calculated as $TPR = \frac{TP}{TP+FN}$ and $FNR = \frac{FN}{TP+FN}$.

We employ confusion matrices to better visualize these numeric results. The structure of these is shown in table 3.1. Since our detectors do not produce negative predictions, we cannot compute the true negative count (TP).

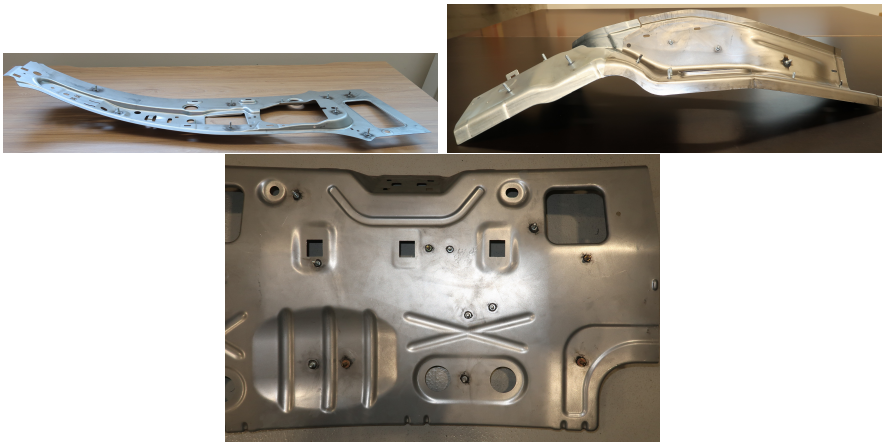


Figure 3.2: Examples of images from the test data set

		Target	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

Table 3.1: Arrangement of confusion matrices

3.2.2 Evaluation of Detectors

To evaluate the performance of a detector, we compare the manually annotated bounding boxes and those found by the detector (rectangles for the hand-tuned versions, squares otherwise). We use a metric called intersection over union (IOU) which is the ratio of two rectangle's overlap and their union. For every ground truth box, we compute an IOU score for each detection box, finding the IOU score of the best fitting detection or 0 when there is no detection box overlapping with the ground truth box. If this best score is above 0.33 or above 0.2 with the detection box situated entirely inside the ground truth box, we proclaim the stud as found and the prediction as a true positive. Otherwise, it is counted in FP. If a predicted bounding box contains two or more studs, TP is increased only by one stud and the rest of the studs are included into FN. If there is another prediction overlapping the first one, this increases TP and therefore decreases FN. If there are multiple prediction boxes for one stud, only one of them is considered correct, the others are considered as FP.

3.2.3 Hand-tuned Detector

The hand-tuned detector was tuned on a subset of training data. This subset contained images where the studs were more prominent and took up more space than in other images, for example one of the photographs in this data set was the top left one in figure 3.1, but the bottom right was not included.

Classifier	Canny	HED
TP	28	10
FP	57	95
FN	75	93
TPR [%]	27.18	9.71
FNR [%]	72.82	90.29
IOU_{found} [%]	31.84	13.14
IOU_{all} [%]	15.09	5.22
Time [ms]	198	3196

Table 3.2: Detection results with the hand-tuned detector

Canny		Target		HED		Target	
		1	0			1	0
Pred	1	28	57	Pred	1	10	95
	0	75	-		0	93	-

Table 3.3: Confusion matrices of the hand-tuned detector's versions

This was done to make the detection simpler and more effective as it is easier to clearly differentiate between a stud and background when the stud is prominent. Tuning the parameters on the whole training data set would be impossible because of the differences between images. The best found parameters were listed in section 2.1.5.

Table 3.2 shows the results achieved on the test data set with the two versions of hand-tuned detector and table 3.3 shows their confusion matrices. Time represents the average time taken to process an image.

When comparing the two versions, the one with Canny detector achieved slightly better results, mainly lower FP. From the high numbers of FP (places incorrectly marked as studs) and FN (missed studs), it is obvious that this type of detector does not perform well in real-world situations. The environment, in which it is applied, would need to be more controlled - without changes to conditions such as lighting or material colour between photographs. The detector makes false predictions mainly around the edges where a colour similar to a stud occurs. Most of the undetected studs were located inside a part of the image, which was brighter or darker than the average brightness.

This makes the hand-crafted detector not applicable as a solution to the given problem, as it would need to be re-tuned very often. Because of this limitation, we were motivated to attempt to use machine learning for training a predictor. Results of this are summarised in the following sections.

3.2.4 Cascade Classifier

In this section, we train several versions of cascade classifier on our data and try to find the best parameters for it. We also consider a method combining

several cascade classifiers in an ensemble to aid with the detection.

Training

As described in section 2.2.4, we created two additional versions of the original set and created positive and negative samples before training the classifier. For this classifier, negative samples are individual sub-images that do not contain studs and are listed in a .txt file. Positive samples are stored differently - an auxiliary script (provided by OpenCV) takes the whole training images and a text file specifying ground truth bounding boxes and outputs a .vec file that contains information about the positive sub-images. The negative .txt file and the positive .vec file are inputs of the training script, which, internally, fetches the specified negative samples. After this, we had three training sets, each containing 966 positive samples of studs, all keeping the ratio of side lengths 1:1, and 503 negative samples of various sizes. For comparison, the size of the positive set was 965 kB and the size of the negative set was 3.17 MB for the original images. To find the best performing version of the classifier, we set some of the training parameters listed in section 2.2.5 as fixed and tuned the others.

We used the precision limit of the training set to 10^{-5} and 100 for the maximum number of features in a stage. For each stage, 780 positive samples, which is approximately 80% of the available amount, were supplied. We enabled all possible variations of kernels which was, for a 35x35 kernel, over 10^6. In all cases, the Gentle AdaBoost optimization [26] was used. Table 3.4 shows the parameters set when training each version, some of the attributes of the learned classifiers, and the detection settings (explained in section 2.2.6) that produced the best results when being tuned on a subset of the original training data containing ten images. The Images element expresses the version of data set used to train a classifier - Original for grayscale images, HED for edge images formed by the HED algorithm and Refined for grayscale images with filtering and histogram equalization.

It can be seen that the training is slower when using images from HED compared to other types of images and the resulting classifier contains more kernels. In many stages, the number of kernels reaches the set limit. This was probably caused by the algorithm's inability to find kernels that would produce clear distinctions between studs and background in the edge images. Decreasing the kernel size had the effect of shortening the training time, because there were fewer possible kernels to consider. Specifying a higher depth of the search tree resulted in much slower training in version F but not in E. However, as these versions consisted of approximately half the number of learned kernels compared to their counterparts with a depth equal to 1, it can be argued that they were able to find a combination of kernels that performed better on the training data. And having fewer kernels means that they would be able to perform detection in less time.

Based on the tuning results, we selected three classifiers to combine into the

Classifier	A	B	C	D	E	F
Images	Original	HED	Refined	Original	Original	Refined
Kernel size [px]	35x35	35x35	35x35	20x20	20x20	35x35
False alarm$_{max}$ [%]	50	50	50	25	25	30
Hit rate$_{min}$ [%]	99.5	99.5	99.5	99.5	99.8	99.8
Depth	1	1	1	1	3	3
Stages	10	10	10	10	15	15
Kernels	159	720	186	291	104	100
Training time [h]	1.33	2.5	1.5	0.33	0.33	5.33
Prescale [%]	10	10	10	-25	-10	25
Scale factor [%]	1	5	1	0.5	0.5	1
Neighbours	25	2	30	25	10	15

Table 3.4: Characteristics of cascade classifier versions

Sub-classifier	D	E	F
Prescale [%]	100	100	150
Scale factor [%]	0.5	0.5	0.5
Neighbours	50	5	5

Table 3.5: Parameters of the ensemble of cascade classifiers

ensemble version (named G). Because it achieved the lowest FN (missed the lowest number of studs), we chose version D to work as the primary detector. Then we picked versions E and F to confirm D's predictions, because they had significantly lower FP and better TPR than versions A,B or C. We changed the detection parameters used in G to produce more candidates from D and more accurate classifications from E and F. This increased the detection time, but it was not critical, because the secondary classifiers work only on small parts of the original images. Table 3.5 shows the modified detection parameters.

Testing

Table 3.6 shows the results achieved by the different trained versions on the test set. IOU$_{all}$ denotes the average achieved IOU score over all the ground truth boxes, including those that were not found (in that case, IOU was zero). These are excluded from the IOU$_{found}$ metric. Time shows the average detection time. Confusion matrices are shown in table 3.7.

When considering the results of the first three versions, version B, which was trained on HED images, performed worse. Its TPR score was close to the performance of the hand-crafted detectors described in section 3.2.3 and the FP was much higher, although it is more robust to changes in the input images, whereas the hand-tuned versions can perform well only on invariable data. The high size of kernels and their number combined with

Classifier	A	B	C	D	E	F	G
TP	77	34	52	77	72	78	92
FP	340	565	509	74	34	84	95
FN	26	69	51	26	31	25	11
TPR [%]	74.76	33.01	50.49	74.76	69.9	75.73	89.32
FNR [%]	25.24	66.99	49.51	25.24	30.1	24.27	10.68
IOU_{found} [%]	44.91	29.04	36.11	51.76	57.68	50.77	57.11
IOU_{all} [%]	41.91	22.21	29.98	40.6	41.32	40.92	53.55
Time [ms]	5283	11723	20015	1326	703	759	12468

Table 3.6: Detection results with the cascade classifier

A		Target	
		1	0
Pred	1	77	340
	0	26	-

B		Target	
		1	0
Pred	1	34	565
	0	69	-

C		Target	
		1	0
Pred	1	52	509
	0	51	-

D		Target	
		1	0
Pred	1	77	74
	0	26	-

E		Target	
		1	0
Pred	1	72	34
	0	31	-

F		Target	
		1	0
Pred	1	78	84
	0	25	-

G		Target	
		1	0
Pred	1	92	95
	0	11	-

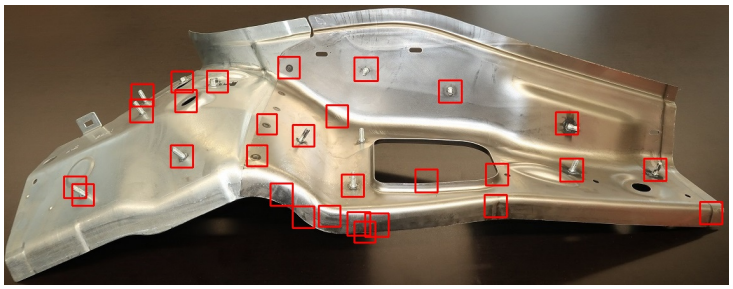
Table 3.7: Confusion matrices of the hand-tuned detector's versions

the needed preprocessing also meant that this version was slow compared to the others. Versions C and A performed better but still had high FP and slow detection. That is why we would not consider them to be suitable for real-world applications. From the substantial drop in FP between versions A and D, it is apparent that setting a lower desired maximum false alarm during training has a direct effect on the FP metric during detection. Increasing the search-tree depth and the desired minimum hit rate in version E lowered FP even further at the cost of a worse TP. From the result of version F, we can conclude that using a higher kernel size, stricter desired parameters, and preprocessed training images produces a more accurate detection. Version G achieved the highest TPR but still retained many false positive detections. Its detection time was also considerably higher compared to versions D, E and F but we think this is an acceptable trade-off for accuracy. It also should be noted that the detection time does not rise with a test image's size, but rather with the number of areas in the image that look similar to studs. This is caused by the cascading arrangement of the classifier and dismissing of unpromising areas.

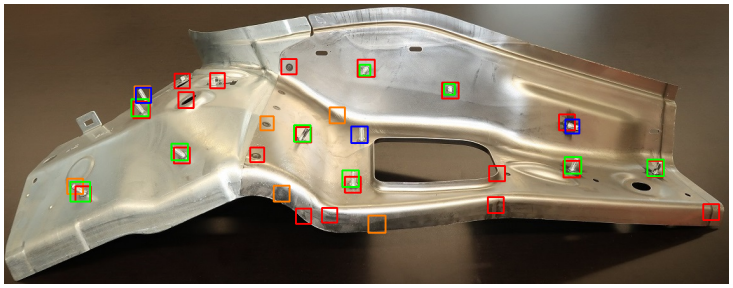
Figure 3.3 shows the output of version G. Enlarged areas marked by the primary classifier are shown in subfigure 3.3a. Here, it can be seen that most studs were identified, apart from one in the centre of the image. There are also many false positives, but the overall area for the secondary classifiers is greatly reduced. In subfigure 3.3b, areas indicated as positive are coloured

red. Orange squares mark areas that were confirmed by only one of the two secondary classifiers. If a stud is identified correctly, its ground truth bounding box is displayed in green, otherwise it is blue. The false positive predictions are are mostly circular objects or areas that have a similar colour pattern to studs. It is evident that the detector recognizes studs from both top and side views.

Regarding the accuracy of individual predictions, the classifier achieved an average IOU score equal to 60% and 52% in the first and second image in 3.3b, respectively. This could be seen as low, but, based on the images, we consider the predictions to overlap precisely enough with the ground truth boxes.



(a) : Output of the primary classifier



(b) : Output of the secondary classifiers

Figure 3.3: Detections made by version G of cascade classifier

From these results, we can conclude that this type of detector performs sufficiently well. It reached almost 90% TPR although it still retained a considerable number of false positives. For quality control applications like ours, this can function as a satisfactory first stage of inspection, which is followed by a human examination on a much smaller scale than would otherwise be needed. However, the performance can be better if we remove the false positives. We attempted to do this in section 3.2.5 by designing a better way to make binary classification in the secondary part of version G.

3.2.5 Support Vector Machines

Binary classification is not usually done by a detector that can find multiple objects in a given image (like the cascade classifier), but by an algorithm that

SVM version	A	B	C
Image size [px]	30	36	60
Cell size [px]	10x10	12x12	6x6
Angles	9	9	16
C	10	10	10
$Gamma$	10^{-2}	10^{-2}	10^{-4}
Features	144	144	5184

Table 3.8: Best found HOG and SVM training parameters

simply assigns one of the two classes. One such algorithm is called support vector machine and it is easy to train and use. In this section, we try to create the most suitable SVM for our task.

Training

To create a training set for this method, we applied the primary part of detector G (column D in table 3.5) from the previous section to the original training set and saved the marked areas as new images. This supplied us with over 1700 samples, from which more than 1200 were negative (not containing studs).

For grid search optimization, we explored the desired preprocessing size of images (explained in section 2.3.2), parameters n (angles) and p (cell size) of the HOG algorithm and C and $gamma$ for SVM training. m (number of cells in each block for HOG) was set to 2x2. Although other versions of block histogram normalization are available, we used the L2 version - this was used in the work that originally implemented this method [25] and is explained by equation 2.13. Moreover, input image normalization by gamma correction was applied. We used the 3-fold cross-validation method for the optimizer. The optimizer searched every available combination of parameters from the supplied values and found the ones that resulted in the best performing learned classifiers. Optimizer parameters and those SVM parameters that have not been mentioned were left with their default values, as set in the scikit-learn implementation.

We selected the three best classifiers, their learning parameters and numbers of features (extracted by HOG) are listed in table 3.8. Even though we included linear kernels in the search parameters, classifiers with RBF kernels always achieved better results, so we did not consider the linear ones further.

Testing

For testing, we applied the primary part of G cascade classifier to the test set and saved the results as in the previous section. This set involved 282 samples, 100 of which were positive. We classified these using each of the

SVM A		Target	
		1	0
Pred	1	75	10
	0	25	172

SVM B		Target	
		1	0
Pred	1	66	5
	0	34	177

SVM C		Target	
		1	0
Pred	1	74	8
	0	26	174

Cascade		Target	
		1	0
Pred	1	93	94
	0	7	88

Table 3.9: Confusion matrices of proposed binary classifiers tested on sub-images from the train set

SVM version	A	B	C
TP	73	65	75
FP	12	6	7
FN	30	38	28
TPR [%]	70.87	63.11	72.82
FNR [%]	29.13	36.89	27.18
IOU_{found} [%]	54.47	55.93	55.9
IOU_{all} [%]	40.25	36.5	42.69
Time [ms]	10542	10598	11019

Table 3.10: Detection results with the cascade classifier combined with SVM on the test set

three SVM classifiers and compared their scores with the one achieved by the original secondary part of G cascade classifier. These results are shown as confusion matrices in table 3.9.

Then we used every SVM version as the secondary classifier after the primary part of G cascade classifier and measured the relevant metrics of the overall detection. This is summarised in tables 3.10 and 3.11. A slight improvement can be seen by increasing the image size and the number of cells (and thus having a feature vector that is more than 36 times longer) from version A to C. And because of the low increase in detection time, we consider version C to be better.

All combinations of cascade classifiers and SVM classifiers performed better than the original cascade classifiers. Although TPR is marginally lower, most of the false positive detections were removed and this outweighs the reduction in TPR, in our opinion. Figure 3.4 shows output of the cascade classifier combined with SVM version C. Unlike the previous version G of cascade classifier, this method employs only one classifier in its second part, so there are no uncertain predictions. Red boxes show the predictions made by the detector, the correctly found ground truth boxes are depicted in green color, and those that were not found are shown in blue.

SVM A		Target	
		1	0
Pred	1	73	12
	0	30	-

SVM B		Target	
		1	0
Pred	1	65	6
	0	38	-

SVM C		Target	
		1	0
Pred	1	75	7
	0	28	-

Table 3.11: Confusion matrices of the cascade classifier combined with SVM applied to the test set

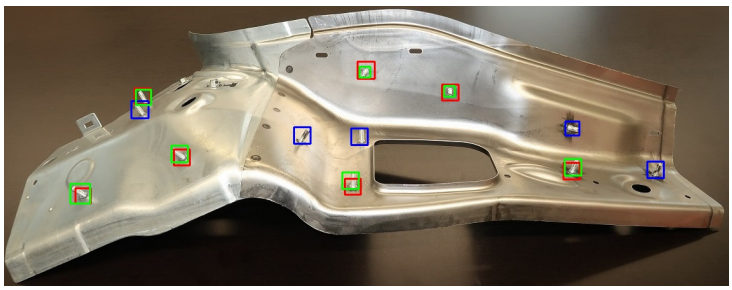


Figure 3.4: Detections made by the cascade classifier and SVM version C

3.3 Comparison of the Proposed Methods and Baseline

The method employing a hand-tuned detector does not perform well enough to be worth further effort. In practice, the input images would need to have consistent lighting, colour palette, and stud dimensions. As this was not true for images in the test set, the method worked adequately on some of them, but the overall metrics were lowered by its poor performance on the rest of the data. Neither is this consistency possible to maintain in real-world applications, because it would probably take more time to set up the images properly than inspect a car part visually.

On the other hand, the cascade classifier, mainly version G, achieved better results throughout the test data set. This method correctly found most of the studs but retained a high number of false positive detections. We consider the requirements for image resolution and format and the detection time to be acceptable in the given real-world scenario.

When we used SVM classifier as the secondary classifier with the cascade classifier G, we achieved substantially better results. The false positive detections were reduced to a minimum and an acceptable level of TPR was maintained. Moreover, the detection time was slightly lower when using SVM. This version has the same image requirements as the previous one. This makes it better suited for real-world applications.

Figure 3.5 shows a comparison of all proposed algorithm versions evaluated on the test set in the form of a graph. The horizontal axis contains the true positive rate ($TPR = \frac{TP}{TP+FN} \times 100\%$) - this represents the percentage of correctly found studs. The vertical axis shows false discovery rate ($FDR =$

$\frac{FP}{TP+FP} \times 100\%$) - this is the percentage of false positive detections from all detections that a detector made. Usually, this graph is used to compare a method's different settings, but we can use it to compare our different proposed algorithms. Points of the graph represent the algorithm versions. From definitions of the used metrics, we can rank the classifiers based on their distance to the upper left corner of the graph - we are looking for the algorithm with the smallest distance. The distance is given by the equation: $D = \sqrt{(100 - TPR)^2 + FDR^2}$. The work [5] can be seen as a baseline when considering our results. They used the precision and recall metrics to evaluate their model, which makes it easy to include in our comparison, because $TPR = recall$ and $precision = 100 - FDR$. Their best-performing version achieved scores 98% and 20.8% for recall and precision, respectively. From

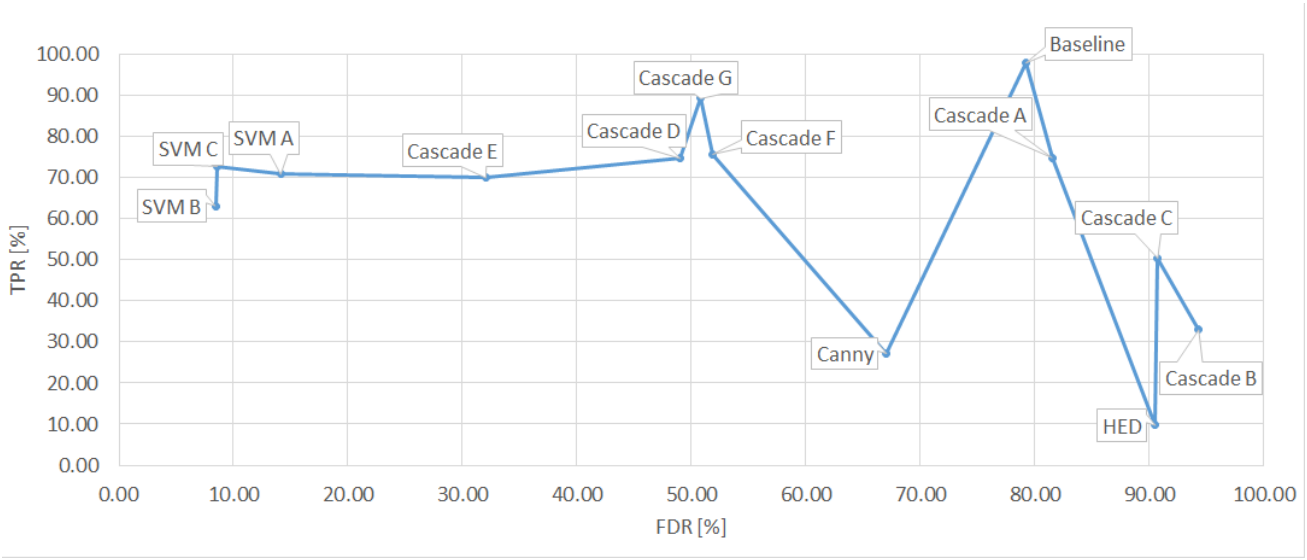


Figure 3.5: Graphical comparison of classifiers

the distance metric, we conclude that the best performing detectors are the three combinations of the cascade classifier and SVM. Although some of the other methods have higher TPR (they detected more studs), their FDR is considerably higher (they made too many false positive predictions). Hand-tuned detectors and cascade classifiers with simpler training A,B, and C perform poorly. Cascade version E achieves results comparable to the best methods, but we consider the cascade + SVM C version to be better as a solution to our given task because of the reasons explained in the previous paragraphs. When comparing this version to the baseline, we can see that although we achieved substantially lower TPR (72.82% compared to 98%), our FDR was also much lower than theirs (8.54% compared to 79.2%). This was caused by the fact that their model had a high FP score, which was acceptable for their task of measuring the deviation of studs from their defined positions, because they would not look for studs which were placed unnecessarily and therefore could reliably filter out false positives. In our case, we focus on detection all studs and not so much on the correct positions.

Having different tasks and solution set-ups means that we cannot directly compare the results. However, when we consider that the baseline included a CNN and was trained on a larger data set, it serves as a strong indication that our algorithm performed well enough compared to other research methods.

As seen in figure 3.4, the detector had the most problems with finding studs whose appearance was not commonly represented in the train set, viewed from neither the top nor the side perspective but from approximately 45° angle. Its performance is strongly dependent on the correct function of the primary cascade classifier - if this does not mark a stud, it is not even considered by the SVM classifier. On the other hand, it is difficult to find a suitable balance between the number of correctly marked studs and the number of predictions when tuning the parameters of the cascade classifier. Average detection time of less than 15 seconds is acceptable for our task. Both detection quality and time could probably be improved by training both parts of the detector on a larger data set, which may be a motivation for further work.

3.4 User Interface and Impact on the Inspection Process

We designed a simple graphical user interface (GUI) application. It lets the user select an image of a 3D model, on which the studs have previously been marked in green (this is a simple operation in any modeling software), and a photograph of a car part that corresponds to the model and that was taken from the same view. When the user starts the detection, green studs in the model image are found with a simple colour-seeking script and a detector is applied to the other image. When this is finished, the bounding boxes obtained from the images are compared and the results are marked in the photograph together with a simple legend and statistics shown below them. The user can then inspect the results and visually evaluate whether, for example, the areas marked as uncertain contain studs or not.

Figure 3.6 shows how this application looks after a detection is complete. For this example, we modified locations of the studs from the model to better represent the locations in the photograph, because we did not have a precise model available. Version G of the cascade classifier was used for detection, if we used one of the versions with SVM, the uncertain statistic would be removed.

We approximated how much time our method would take compared to a simple visual control of several car parts like the ones in our data set or one larger, based on the experiences of workers who perform this task. Figure 3.7 shows how much time could be saved by using our program. This process is divided into several parts, the first of which is the creation of screenshots of the 3D model. Then, photographs of the real part would need to be taken and ideally preprocessed (cropped, rotated, etc.) to match the shape as precisely

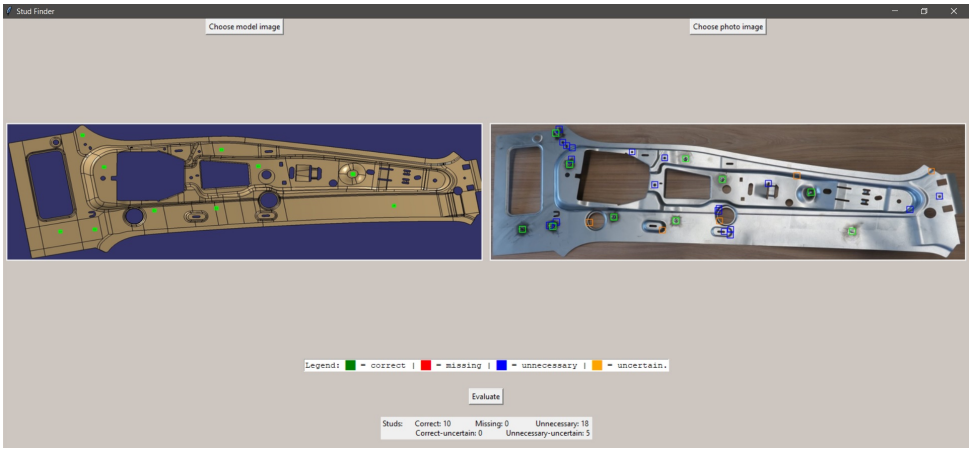


Figure 3.6: GUI application

as possible to the screenshots. This is necessary to help with the comparison of positions. After detection, the user would inspect areas marked as incorrect at locations specified by the bounding boxes as in figure 3.6.

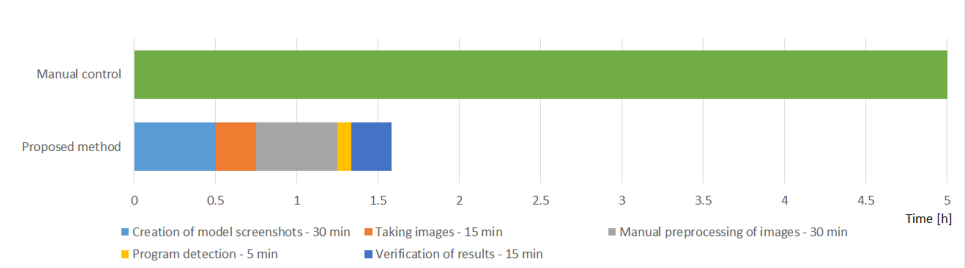


Figure 3.7: Comparison of time needed for manual control and our method

Chapter 4

Conclusion and Future Work

4.1 Discussion and Conclusion

In this thesis we described, implemented and measured the performance of several object detection methods and their different parameters for detecting weld studs on metal car parts. We implemented a detector which used image processing techniques to find objects in an image and a set of simple predefined rules, which were tuned on the training data set, to detect studs from those objects. We considered this approach because works like [3] achieved almost perfect results while using similar hand-tuned detectors. This is most likely caused by the non-uniformity of our data which makes it difficult to define a precise set of rules that covers all the possible variations of studs but rejects other objects. Conditions (lighting, color) would need to be controlled and constant throughout the images for this method to function reliably.

Other works, for example [2], dealing with tasks similar to ours achieved very good detection quality while using CNNs. We could not experiment with such methods because our data set was not large enough to train them. Instead, we implemented an algorithm called a cascade classifier. We were able to achieve high true positive count, but the problem of too many false positive detections prevailed. We added another two different versions of this detector to the detection process after the main cascade classifier which decided if the original predictions were correct. This, however, did not improve our results. This approach proved to be similarly ineffective in [4], where the authors employed the cascade classifier to make binary predictions on whether images contained a screw. They constructed a complete FPR/TPR curve from differently trained versions of the classifier and if we translate the scores from the cascade confusion matrix in table 3.9 to other metrics, we get 52% FPR and 93% TPR. At the same FPR, they achieved approximately 50% TPR, so our method could be considered better. But we cannot directly compare the results, as this work is several years old and the detector was not the main part of it. We greatly improved upon this system by training a simple SVM classifier which was able to reduce FPR to approximately 4%

(again based on table 3.9) at the cost of lowering TPR to 74%. At this FPR, [4] achieved TPR under 10%.

The performance of our best method, which is an ensemble of a cascade classifier and an SVM, was comparable to a model from [5] with conditions most similar to ours. We achieved a FDR/TPR score of 8.54%/72.82% on the test set, compared to their 79.2%/98%. Because their work focused on measuring deviations from the desired locations, they could reject those predictions that were not supposed to contain studs and a high number of false positives did not concern them. We needed to detect every stud independent on the expected locations and so this compensation of predictions was not available to us. Still, we think our method performs well enough, considering the limited amount of data available and the fact that they used a CNN when detecting the studs. Test set contained 103 studs in 8 images and the proposed method made 75 true positive predictions (correctly identified studs), 28 false negative ones (undetected studs) and 7 false positives (areas not containing studs but marked as positive).

The ideal FDR/TPR score would, of course, be 0%/100%, otherwise the inspection looses its purpose. We feel certain that a better detection model could be created with additional data and training of a better primary detector. Even with the achieved performance, we think the proposed detector's performance is good enough to serve as a part of a solution for the given real-world problem. It could serve for the initial control of studs, with a human worker inspecting the results in the GUI and determining if the marked areas' content (both marked as positive and negative) corresponds to predictions. This way it could save a lot of time and the needed accuracy would be maintained.

4.2 Future Work

The most constructive work to be done is probably obtaining a larger data set, replacing the used cascade classifier with a more modern CNN and determining if it provides an improvement in performance.

Apart from this, auxiliary programs should be created to help with creating images from 3D models, preprocessing of photographs and the GUI should be improved to be able to compare multiple views of a car part with corresponding photographs instead of only one. This would take more work from the user and make the whole process more repeatable without the factor of human error.

Another worthwhile line of work would be implementing the tested methods in a faster programming language (like C++) with optimized code. This could reduce detection time and make it possible to use more complicated algorithms.

Appendix A

Contents of CD

- Scripts and Data
  - Cascade Training
    - hed-neg, orig-neg, ref-neg - Files containing negative samples.
    - vec_txt - Folder with positive samples and negative .txt files.
    - opencv_traincascade.exe - Script for training.
  - SVM Training
    - train_images
      - pos, neg
    - train.py
  - Trained Detectors
    - Cascade A - B
    - SVM A - B.sav
  - Documentation for Code and Data.pdf
  - GUI Images, Test Images, Train Images - This contains only the original set of data, without rotations.
  - functions.py, GUI.py, Test Detectors.py

Appendix B

Bibliography

[1] B. Wu, F. Zhang, and T. Xue, "Monocular-vision-based method for online measurement of pose parameters of weld stud," *Measurement*, vol. 61, pp. 263–269, 2015.

[2] J. Miranda, S. Larnier, A. Herbulot, and M. Devy, "Uav-based inspection of airplane exterior screws with computer vision," in *14h International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.*, 2019.

[3] A. A. Khule, M. S. Nagmode, and R. D. Komati, "Automated object counting for visual inspection applications," in *2015 International Conference on Information Processing (ICIP)*, pp. 801–806, 2015.

[4] K. Wegener, W. H. Chen, F. Dietrich, K. Dröder, and S. Kara, "Robot assisted disassembly for the recycling of electric vehicle batteries," *Procedia CIRP*, vol. 29, pp. 716–721, 2015. The 22nd CIRP Conference on Life Cycle Engineering.

[5] H. Liu, Y. Yan, K. Song, H. Chen, and H. Yu, "Efficient optical measurement of welding studs with normal maps and convolutional neural network," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–14, 2021.

[6] Y. He, K. Song, Q. Meng, and Y. Yan, "An end-to-end steel surface defect detection approach via fusing multiple hierarchical features," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 4, pp. 1493–1504, 2019.

[7] Z. Wang, H. Li, and X. Zhang, "Construction waste recycling robot for nails and screws: Computer vision technology and neural network approach," *Automation in Construction*, vol. 97, pp. 220–228, 2019.

[8] L. Geng, J. Wang, W. Wang, and Z. Xiao, "Welding studs detection based on line structured light," in *2017 International Conference on Optical Instruments and Technology: Optoelectronic Measurement Technology*

and Systems, vol. 10621, p. 106210Z, International Society for Optics and Photonics, 2018.

[9] Y. Dou, Y. Huang, Q. Li, and S. Luo, "A fast template matching-based algorithm for railway bolts detection," *International Journal of Machine Learning and Cybernetics*, vol. 5, no. 6, pp. 835–844, 2014.

[10] F. Marino, A. Distante, P. L. Mazzeo, and E. Stella, "A real-time visual inspection system for railway maintenance: automatic hexagonal-headed bolts detection," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 3, pp. 418–428, 2007.

[11] M. Elad, "On the origin of the bilateral filter and ways to improve it," *IEEE Transactions on image processing*, vol. 11, no. 10, pp. 1141–1151, 2002.

[12] S. Montabone and A. Soto, "Human detection using a mobile platform and novel features derived from a visual saliency mechanism," *Image and Vision Computing*, vol. 28, no. 3, pp. 391–402, 2010.

[13] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[14] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[15] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 1395–1403, 2015.

[16] S. Xie, "Holistically-nested edge detection," 2015. `https://github.com/s9xie/hed`, visited 2021-02-01.

[17] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, 2001.

[18] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004.

[19] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[20] L. Cuimei, Q. Zhiliang, J. Nan, and W. Jianhua, "Human face detection algorithm via haar cascade classifier combined with three additional classifiers," in *2017 13th IEEE International Conference on Electronic Measurement Instruments (ICEMI)*, pp. 483–487, 2017.

[21] G. L. Team, "What is histogram equalization and how it works?," 2020. http://www.mygreatlearning.com/blog/histogram-equalization-explained/, visited 2021-04-16.

[22] K. Zuiderveld, "Contrast limited adaptive histogram equalization," *Graphics gems*, pp. 474–485, 1994.

[23] "Cascade classifier training." https://docs.opencv.org/2.4/doc/user_guide/ug_traincascade.html, visited 2021-04-18.

[24] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[25] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, pp. 886–893, Ieee, 2005.

[26] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *Annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.