Comparison of CNN for the detection of small ojects based on the example of components on an assembly table

Jonas Hansert[1], Madlon Pécaut[2], and Thomas Schlegel

[1] Institute of Ubiquitous Mobility Systems, Karlsruhe University of Applied Sciences
jonas.hansert@hs-karlsruhe.de
[2] Institute of Ubiquitous Mobility Systems, Karlsruhe University of Applied Sciences
madlon.pecaut@googlemail.com

Abstract. This paper presents the results of a comparison of deep neural networks for detection of small objects typical for manual manufacturing tasks. We created a set of training, validation and evaluation data and selected four state of the art deep neural networks for object detection. We trained them with the same number of epochs, 200 epochs per network architecture and compared the training time, accuracy and prediction time on evaluation data. Additional we compared the neural networks on thirty images of three very small and similar components.

Keywords: We would like to encourage you to list your keywords within the abstract section

1 Use Case and Requirements

Many tasks which only a few years ago had to be performed by humans can now be performed by robots or will be performed by robots in the near future. Nevertheless, there are some tasks in assembly processes which cannot be automated in the next few years. This applies especially to workpieces that are only produced in very small series or tasks that require a lot of tact and sensitivity, such as inserting small screws into a thread or assembling small components.

In conversations with companies we have found out that a big problem for the workers is learning new production processes. This is currently done with instructions and by supervisors. But this requires a lot of time. This effort can be significantly reduced by modern systems, which accompany workers in the learning process. Such intelligent systems require not only instructions that describe the target status and the individual work steps that lead to it, but also information on the current status at the assembly workstation. One way to obtain this information is to install cameras above the assembly workstation and use image recognition to calculate where an object is located at any given moment.

The individual parts, often very small compared to the work surface, must be reliably detected. We have trained and tested several deep neural networks for this purpose.

We have developed an assembly workstation where work instructions can be projected directly onto the work surface using a projector. At a distance, 21 containers for components are arranged in three rows, slightly offset to the rear, one above the other. These

containers can also be illuminated by the projector. Thus a very flexible Pick-by-Light system can be implemented. In order for the system behind it to automatically switch to the next work step and, in the event of errors, to point them out and provide support in correcting them, it is helpful to be able to identify the individual components on the work surface.

Fig. 1. Our assembly Workstation with a projector, a camera and 21 containers for the components.

We use a RealSense depth camera for this purpose, from which, however, we are currently only using the colour image. The camera is mounted in a central position at a height of about two meters above the work surface. Thus the camera image includes the complete working surface as well as the 21 containers and a small area next to the working surface.

The objects to be detected are components of a kit for the construction of various toy cars. The kit contains 25 components in total. Some of the components vary considerably from each other, but some others are very similar to each other. Since it is the same with real components of a production, the choice of the kit seemed appropriate for the purposes of this project.

2 Related Work

Object detection, one of the most fundamental and challenging problems in computer vision, seeks to local object instances from a large number of predefined categories in natural images.

Until the beginning of 2000, a similar approach was mostly used in object detection. Keypoints in one or more images of a category were searched for automatically. At these points a feature vector was generated. During the recognition process, keypoints in the image were again searched, the corresponding feature vectors were generated and compared with the stored feature vectors. After a certain threshold an object was assigned to the category. One of the first approaches based on machine learning was published by Viola and Jones in 2001 [1]. They still selected features, in their case they were selected by using a Haar basis function [2] and then using a variant of AdaBoost [3].

Starting in 2012 with the publication of AlexNet by Krizhevsky et al. [4], deep neural networks became more and more the standard in object detection tasks. They used a convolutional neural network which has 60 million parameters in five convolutional layers, some of them are followed by max-pooling layers, three fully-connected layers and a final softmax layer. They won the ImageNet LSVRC-2012 competition with a error rate almost half as high as the second best.

Inception-v2 is mostly identical to Inception-v3 by Szegedy et al. [5]. It is based on Inception-v1 [6]. All Inception architectures are composed of dense modules. Instead of stacking convolutional layers, they stack modules or blocks, within which are convolutional layers. For Inception-v2 they redesigned the architecture of Inception-v1 to avoid representational bottlenecks and have more efficient computations by using factorisation methods. They are the first using batch normalisation in object detection tasks.

In previous architectures the most significant difference has been the increasing number of layers. But with the network depth increasing, accuracy gets saturated and then degrades rapidly. Kaiming et al. [7] addressed this problem with ResNet using skip connections, while building deeper models.

In 2017 Howard et al. presented MobileNet architecture [8]. MobileNet was developed for efficient work on mobile devices with less computational power and is very fast. They used depthwise convolutional layers for a extremely efficient network architecture.

One year later Sandler et al. [9] published a second version of MobileNet. Besides some minor adjustments, a bottleneck was added in the convolutional layers, which further reduced the dimensions of the convolutional layers. Thus a further increase in speed could be achieved.

In addition to the neural network architectures presented so far, there are also different methods to detect in which area of the image the object is located. The two most frequently used are described briefly below. To bypass the problem of selecting a huge number of regions, Girshick et al. [10] proposed a method where they use selective search by the features of the base CNN to extract just 2000 regions proposals from the image. Liu et al. [11] introduced the Single Shot Multibox Detector (SSD). They added some extra feature layers behind the base model for detection of default boxes in different scales

and aspect ratios. At prediction time, the network generates scores for the presence of each object in each default box. Then it produces adjustments to the box to better match the object shape.

There is just one publication over the past few years which gives an survey of generic object detection methods. Liu et al. [12] compared 18 common object detection architectures for generic object detection. There are many other comparisons of specific object detection tasks. For example pedestrian detection [13], face detection [14] and text detection [15].

3 Training Dataset

The project is based on the methodology of supervised learning. Thereby the models are trained using a training dataset consisting of many samples. Each sample within the training dataset is tagged with a so called label (also called annotation). The label provides the model with information about the desired output for this sample. During training, the output generated by the model is then compared to the desired output (labels) and the error is determined. This error on the one hand gives information about the current performance of the model and, on the other hand it is used for further mathematical computations to adjust the model's parameters, so that the model's performance improves.

For the training of neural networks in the field of computer vision the following rule of thumb applies: The larger and more diverse the training dataset, the higher the accuracy that can be achieved by the trained model. If you have too little data and/or run it through the model too often, this can lead to so-called overfitting. Overfitting means that instead of learning an abstract concept that can be applied to a variety of data, the model basically memorizes the individual samples [16, 17]. If you train neural networks for the purpose of this project from scratch, it is quite possible that you will need more than 100,000 different images - depending on the accuracy that the model should finally be able to achieve. However, the methodology of the so-called Transfer Learning offers the possibility to transfer results of neural networks, which have already been trained for a specific task, completely or partially to a new task and thus to save time and resources [18]. For this reason, we also applied transfer learning methods within the project.

The training dataset was created manually: A tripod, a mobile phone camera (10 megapixel format 3104 x 3104) and an Apeman Action Cam (20 megapixel format 5120x3840) were used to take 97 images for each of the 25 classes. This corresponds to 2,425 images in total (actually 100 images were taken per class, but only 97 were suitable for use as training data). All images were documented and sorted into close-ups (distance between camera and object less than or equal to 30 cm) and standards (distance between camera and object more than 30 cm). This procedure should ensure the traceability and controllability of the data set. In total, the training data set contains approx. 25% close-ups and approx. 75% standards, each taken on different backgrounds and under different lighting conditions (see Fig. 2). The LabelImg tool was used for the labelling of the data. With the help of this tool, bounding boxes, whose coordinates are stored in either YOLO or Pascval VOC format, can be marked in the images [19].

For the training of the neural networks the created dataset was finally divided into:

Fig. 2. An excerpt from the training data set - images of the component 32_064

- **Training Data (90% of all labelled images):** Images that are used for the training of the models and that pass through the models multiple times during the training.
- **Test Data (10% of all labelled images):** Images that are used for later testing or validation of the training results. In contrast to the images used as training data, the model is presented these images for the first time after training. The goal of this approach, which is common in Deep Learning, is to see how well the neural network recognizes objects in images, that it has never seen before, after the training. Thus it is possible to make a statement about the accuracy and to be able to meet any further training needs that may arise.

4 Implementation

The training of deep neural networks is very demanding on resources due to the large number of computations. Therefore, it is essential to use hardware with adequate performance. Since the computations that run for each node in the graph can be highly parallelized, the use of a powerful Graphical Processing Unit (GPU) is particularly suitable. A GPU with its several hundred computing cores has a clear advantage over a current CPU with four to eight cores when processing parallel computing tasks [20]. These are the outline parameters of the project computer in use:

- **Operating System (OS):** Ubuntu 18.04.2 LTS
- **GPU:** GeForce ®GTX 1080 Ti (11 GB GDDR5X-Memory, data transfer speed 11 Gbit/s)

4.1 Selected Models

For the intended comparison the Tensorflow Object Detection API was used. Tensorflow Object Detection API is an open source framework based on TensorFlow, which among other things provides implementations of pre-trained object detection models for transfer learning [21, 22]. The API was chosen because of its good and easy to understand documentation and its variety of pre-trained object detection models. For the comparison the following models were selected:

- **ssd_mobilenet_v1_coco:**[11, 23, 24]
- **ssd_mobilenet_v2_coco:**[11, 25, 26]
- **faster_rcnn_inception_v2_coco:**[27–29]
- **rfcn_resnet101_coco:**[30–32]

To ensure comparability of the networks, all of the selected pre-trained models were trained on the COCO dataset [33]. Fundamentally, the algorithms based on CNN models can be grouped into two main categories: region-based algorithms and one-stage algorithms [34].

While both SSD models can be categorized as one-stage algorithms, Faster R-CNN and R-FCN fall into the category of region-based algorithms. One-stage algorithms predict both - the fields (or the bounding boxes) and the class of the contained objects - simultaneously. They are generally considered extremely fast, but are known for their trade-off between accuracy and real-time processing speed. Region-based algorithms consist of two parts: A special region proposal method and a classifier. Instead of splitting the image into many small areas and then working with a large number of areas like conventional CNN would proceed, the region-based algorithm first proposes a set of regions of interest (ROI) in the image and checks whether one of these fields contains an object. If an object is contained, the classifier classifies it [34]. Region-based algorithms are generally considered as accurate, but also as slow. Since, according to our requirements, both accuracy and speed are important, it seemed reasonable to compare models of both categories.

4.2 Training Configuration

Besides the collection of pre-trained models for object detection, the Tensorflow Object Detection API also offers corresponding configuration files for the training of each model. Since these configurations have already shown to be successful, these files were used as a basis for own configurations. The configuration files contain information about the training parameters, such as the number of steps to be performed during training, the image resizer to be used, the number of samples processed as a batch before the model parameters are updated (batch size) and the number of classes which can be detected.

To make the study of the different networks as comparable as possible, the training of all networks was configured in such a way that the number of images fed into the network simultaneously (batch size) was kept as small as possible. Since the configurations of some models did not allow batch sizes larger than one, but other models did not allow batch sizes smaller than two, no general value for all models could be defined for this parameter. During training, each of the training images should be passed through the net 200 times (corresponds to 200 epochs). The number of steps was therefore adjusted accordingly, depending on the batch size. If a fixed shape resizer was used in the base configurations, two different dimensions of resizing (default: 300x300 pixels and custom: 512x512 pixels) were selected for the training. Table 1 gives an overview of the training configurations used for the training of the different models.

Table 1. Overview of different training runs, configurations and durations

Model	Batch Size	Steps	Epochs	Image Resizer		Total Loss	Training Duration
				keep_aspect_ratio_resizer min_dimension / max_dimension	fixed_shape_resizer height / width		
ssd_mobilenet_v1_coco	2	217 500	200	-	512 / 512	5.759	11h 51m 45s
ssd_mobilenet_v1_coco	2	217 500	200	-	300 / 300	5.889	09h 45m 45s
ssd_mobilenet_v2_coco	2	217 500	200	-	512 / 512	3.289	12h 23m 49s
ssd_mobilenet_v2_coco	2	217 500	200	-	300 / 300	3.516	09h 41m 47s
faster_rcnn_inception_v2_coco	1	435 000	200	600 / 1024	-	0.066	14h 35m 46s
rfcn_resnet101_coco	1	435 000	200	600 / 1024	-	0.031	26h 39m 27s

5 Evaluation

In this section we will first look at the training, before we then focus on evaluating the quality of the results and the speed of the selected convolutional neural networks.

5.1 Training

When evaluating the training results, we first considered the duration that the neural networks require for 200 epochs (see Fig. 3). It becomes clear that especially the two Region Based Object Detectors (Faster R-CNN Inception V2 and RFCN Resnet101) took significantly longer than the Single Shot Object Detectors (SSD Mobilenet V1 and SSD Mobilenet V2). In addition, the Single Shot Object Detectors clearly show that the size of the input data also has a decisive effect on the training duration: While SSD Mobilenet V2 with an input data size of 300x300 pixels took the shortest time for the training with 9 hours 41 minutes and 47 seconds, the same neural network with an input data size of 512x512 pixels took almost three hours more for the training, but is still far below the time required by RFCN Resnet101 for 200 epochs of training.

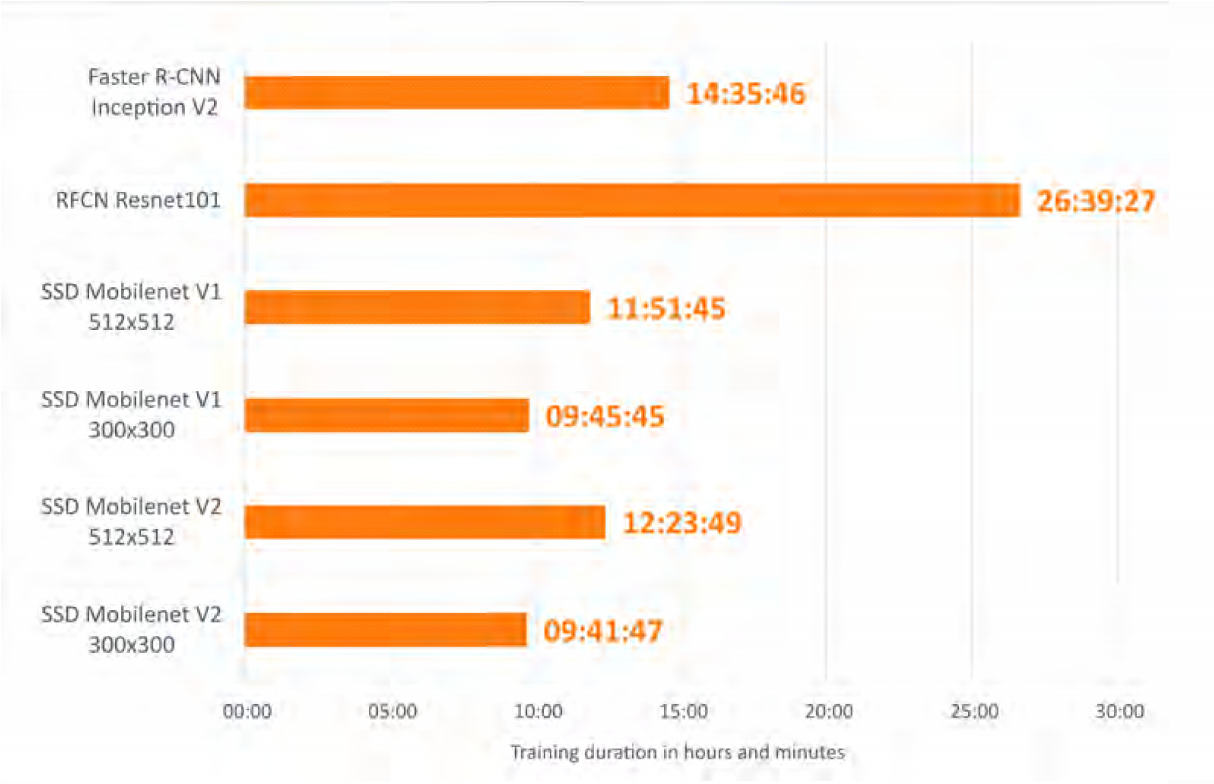

Fig. 3. The training duration for 200 epochs and two different input image sizes.

5.2 Quality of detection

The next point in which we compared the different networks was accuracy (see Fig. 4). We focused on seeing which of the nets were correct in their detections and how often (absolute values), and we also wanted to see what proportion of the total detections were

correct (relative values). The latter seemed to us to make sense especially because some of the nets showed more than three detections for a single object. The probability that the correct classification will be found for the same object with more than one detection is of course higher in this case than if only one detection per object is made. With regard to the later use at the assembly table, however, it does not help us if the neural net provides several possible interpretations for the classification of a component.

Figure 4 shows that, in this comparison, the two Region Based Object Detectors generally perform significantly better than the Single Shot Object Detectors - both in terms of the correct detections and their share of the total detections. It is also noticeable that for the Single Shot Object Detectors, the size of the input data also appears to have an effect on the comparison point on the result. However, there is a clear difference to the previous comparison of the required training durations: While the training duration increased uniformly with increasing size of the images with the Single Shot Object Detectors, such a uniform observation cannot be made with the accuracy, concerning the relation to the input data sizes. While SSD Mobilenet V2 achieves good results with an input data size of 512x512 pixels, SSD Mobilenet V1 delivers the worst result of this comparison for the same input data size (regarding the number of correct detections as well as their share of the total detections). With an input data size of 300x300 pixels, however, the result improves with SSD Mobilenet V1, while the change to a smaller input data size has a deteriorating effect on the result with SSD Mobilenet V2. The best result of this comparison - judging by the absolute values - was achieved by Faster R-CNN Inception V2. However, in terms of the proportion of correct detections in the total detections, the Region Based Object Detector is two percentage points behind RFCN Resnet 101, also a Region Based Object Detector.

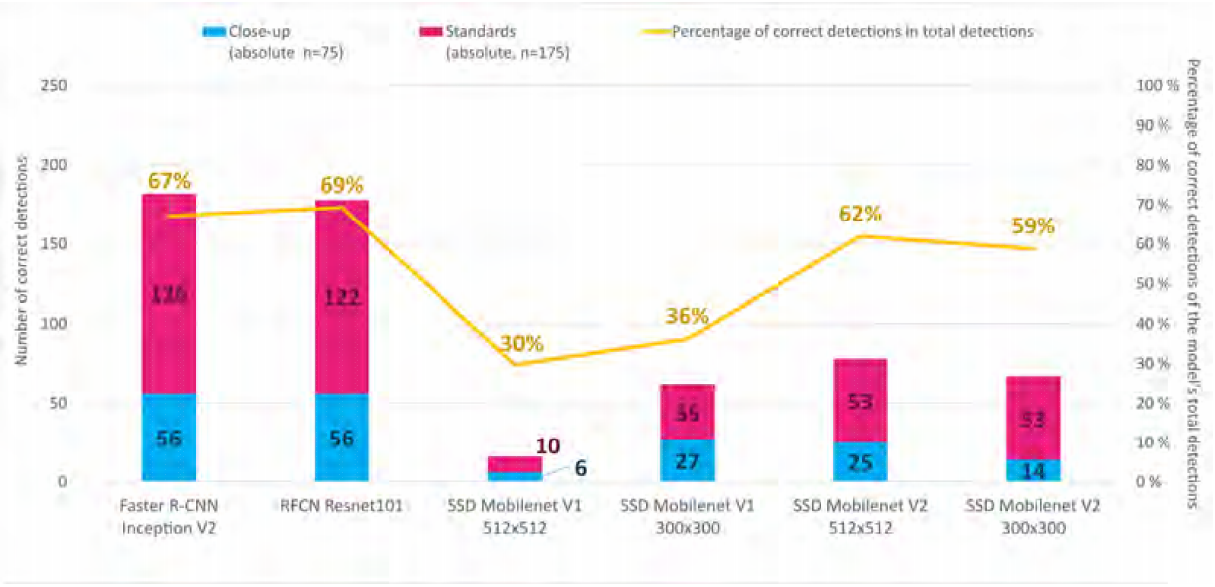

Fig. 4. The accuracy of the CNNs we are looking at.

We were particularly interested in how neural networks would react to particularly similar, small objects. Therefore, we decided to investigate the behavior of neural networks within the comparison using an example to illustrate the behavior of the three very similar objects. Figure 5 shows the selected components for the experiment.

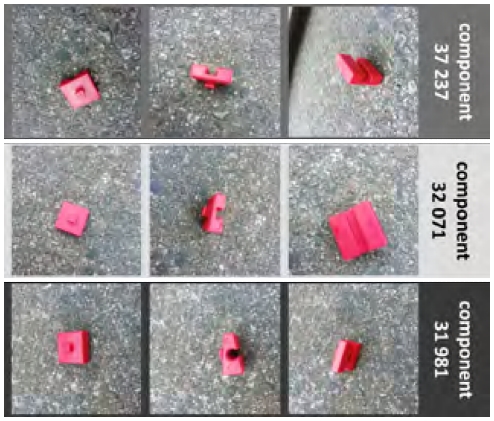

Fig. 5. Very similar objects, which we particularly looked at.

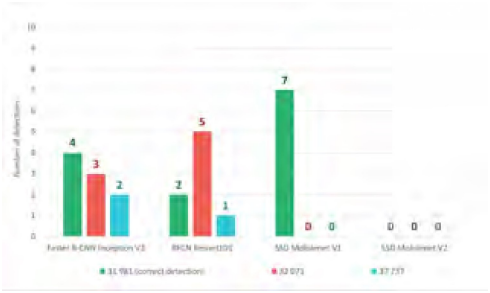

Fig. 6. Overview of the behaviour of different neural networks when detecting parts with strong similarity using the example of detections in images of part 31 981.

For each of these three components we examined how often it was correctly detected and classified by the compared neural networks and how often the network misclassified it with which of the similar components. The first and the second component was detected in nearly all cases by both region based approaches. The classification by Inception-v2 and Resnet-101 failed in about one third of images. The SSD networks detected the object in just one of twenty cases but Mobilenet classified this correct.

It has been surprising, that the results for the third component looks very different to the others (see Fig. 6). SSD Mobilenet V1 correctly identified the component in seven of ten images and did not produce any detections that could be interpreted as misclassifications with one of the similar components. SSD Mobilenet V2 did not detect any of the three components, as in the two previous investigations. The results of the two region based object detectors are rather moderate. Faster R-CNN Inception V2 has detected the correct component in four of ten images, but still five misclassifications with the other two components. RFCN Resnet101 has caused many misclassifications with the other two components. Only two of ten images were correctly detected but it had six misclassifications with the similar components.

An other important aspect of the study is the speed, or rather the speed at which the neural networks can detect objects, especially with regard to later use at the assembly table. For the comparison of the speeds on the one hand the data of the GitHub repository of the TensorFlow Object Detection API for the individual neural nets were used, on the other hand the actual speeds of the neural nets within this project were measured. It becomes clear that the speeds measured in the project are clearly below the achievable speeds that are mentioned in the GitHub repository of the TensorFlow Object-Detection API. On the other hand, the differences between the speeds of the Region Based Object Detectors and the Single Shot Object Detectors in the project are far less drastic than expected.

6 Conclusions

We have created a training dataset with small, partly very similar components. With this we have trained four common deep neural networks. In addition to the training times, we examined the accuracy and the recognition time with general evaluation data.

In addition, we examined the results for ten images each of three very similar and small components.

None of the networks we trained produced suitable results for our scenario. Nevertheless, we were able to gain some important insights from the results. At the moment, the runtime is not yet suitable for our scenario, but it is also not far from the minimum requirements, so that these can easily be achieved with smaller optimizations and better hardware. It was also important to realize that there are no serious runtime differences between the different network architectures.

The two region based approaches delivered significantly better results than the SSD approaches. However, the results of the detection of the third small component suggest that Mobilenet in combination with a faster R-CNN could possibly deliver even better results. Longer training and training data better adapted to the intended use could also significantly improve the results of the object detectors.

References

1. Viola, P., Jones, M.: 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. Volume 1. (Dec 2001) I–I
2. Papageorgiou, C., Oren, M., Poggio, T.: General framework for object detection. Volume 6:. (02 1998) 555 – 562
3. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: Proceedings of the Second European Conference on Computational Learning Theory. EuroCOLT '95, Berlin, Heidelberg, Springer-Verlag (1995) 23–37
4. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. Neural Information Processing Systems **25** (01 2012)
5. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision (2015)
6. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions (2014)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)
8. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications (2017)
9. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks (2018)
10. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation (2013)
11. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In Leibe, B., Matas, J., Sebe, N., Welling, M., eds.: Computer Vision – ECCV 2016. Volume 9905 of Lecture Notes in Computer Science. Springer International Publishing, Cham (2016) 21–37
12. Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., Pietikäinen, M.: Deep learning for generic object detection: A survey. International Journal of Computer Vision **128**(2) (2020) 261–318
13. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: An evaluation of the state of the art. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(4) (April 2012) 743–761
14. Zafeiriou, S., Zhang, C., Zhang, Z.: A survey on face detection in the wild: past, present and future. Computer Vision and Image Understanding **138** (04 2015)

15. Ye, Q., Doermann, D.: Text detection and recognition in imagery: A survey. IEEE Transactions on Pattern Analysis and Machine Intelligence **37**(7) (July 2015) 1480–1500
16. Nakatsu, R.T.: Information visualizations used to avoid the problem of overfitting in supervised machine learning. In Nah, F.F.H., Tan, C.H., eds.: HCI in Business, Government and Organizations. Supporting Business, Cham, Springer International Publishing (2017) 373–385
17. Provost, F., Fawcett, T.: Data science for business: What you need to know about data mining and data-analytic thinking. 1st ed. edn. O'Reilly Media, Sebastopol, CA (2013)
18. Yabuki, N., Nishimura, N., Fukuda, T.: Automatic object detection from digital images by deep learning with transfer learning. In Smith, I.F.C., Domer, B., eds.: Advanced Computing Strategies for Engineering, Cham, Springer International Publishing (2018) 3–15
19. Tzutalin: Labelimg: Git code (2015)
20. Paine, T., Jin, H., Yang, J., Lin, Z., Huang, T.: Gpu asynchronous stochastic gradient descent to speed up neural network training
21. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors
22. Martín, A., Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, Xiaoqiang Zheng: Tensorflow: Large-scale machine learning on heterogeneous systems (2015)
23. TensorFlow: Tensorflow object detection api: ssd_mobilenet_v1_coco. online
24. TensorFlow: Tensorflow object detection api: ssd_mobilenet_v1_coco.config. online (2018)
25. TensorFlow: Tensorflow object detection api: ssd_mobilenet_v2_coco. online
26. TensorFlow: Tensorflow object detection api: ssd_mobilenet_v2_coco.config. online (2018)
27. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett, eds.: Advances in Neural Information Processing Systems 28. Curran Associates, Inc (2015) 91–99
28. TensorFlow: Tensorflow object detection api: faster_rcnn_inception_v2_coco. online
29. TensorFlow: Tensorflow object detection api: faster_rcnn_inception_v2_coco.config. online (2018)
30. Dai, J., Li, Y., He, K., Sun, J.: R-fcn: Object detection via region-based fully convolutional networks
31. TensorFlow: Tensorflow object detection api: rfcn_resnet101_coco. online
32. TensorFlow: Tensorflow object detection api: rfcn_resnet101_coco.config. online (2018)
33. Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context
34. Zhai, M., Liu, J., Zhang, W., Liu, C., Li, W., Cao, Y.: Multi-scale feature fusion single shot object detector based on densenet. In Yu, H., Liu, J., Liu, L., Ju, Z., Liu, Y., Zhou, D., eds.: Intelligent Robotics and Applications, Cham, Springer International Publishing (2019) 450–460