Faster R-CNN as an Application for Object Detection of Scattered LEGO Pieces

Hiba Abderrazik[1] , **Jan van Gemert**[1] , **Attila Lengyel**[1]
[1]TU Delft
habderrazik@student.tudelft.nl, {j.c.vangemert, a.lengyel}@tudelft.nl

Abstract

The benchmarks for the accuracy of the best performing object detectors to date are usually based on homogeneous datasets, including objects such as vehicles, people, animals and foods. This excludes a whole set of scenarios containing small, cluttered and rotated objects. This paper selects a state-of-the-art object detection model, Faster R-CNN, and investigates its performance on several custom datasets of scattered LEGO pieces. We discover that the model reaches a high F_1 score on data with images containing up to 13 bricks and that data manipulation, such as cropping, can further improve this performance. Furthermore, we evaluate how this model can be optimized to perform better on a more complex dataset, showing that tweaking the Faster R-CNN RPN layer results in a higher F_1 score for images containing up to 50 bricks. In conclusion, tweaking the RPN layer allows the Faster R-CNN model to reach high performance on datasets containing cluttered images of small LEGO bricks. All data is on the TU Delft server and all code is available at *https://gitlab.com/lego-project-group/faster_rcnn_lego.git*.

Keywords: Computer Vision, Object Detection, LEGO recognition

1 Introduction

Object detection methods have reached high accuracies [17] and are widely applied to different areas of work, but their performance is still poor when applied to crowded images with similar objects [11]. An instance of this could be an image of a pile of LEGO pieces. This paper is partially motivated by the struggles that many LEGO builders face, like wading through piles of LEGO pieces for hours just to find a missing piece, or worse, discovering the piece was not even there to begin with. Essentially, this can be mapped to a problem of object detection, i.e. identifying and localizing a LEGO piece in a pile of different pieces. By solving this problem of distinguishing between objects of similar colors, shapes and sizes to localize a specific piece, we can create a more robust framework for object detection that can be applied to other fields as well. For example, in manufacturing assembly lines where manual checking is still necessary to ensure that all pieces are present, or finding someone in a cluttered scene of people [19].

Current research on the use of object detection models for automated LEGO recognition is limited. The most notable project that uses deep learning to recognize individual pieces of LEGO is the "LEGO sorter" built by Daniel West. This sorter makes use of ResNet-50, a Convolutional Neural Network (CNN) method for image classification [21]. Since no work is done on localizing LEGO pieces in a scene of multiple pieces there is no baseline for the performance of object detection methods on similar data.

However, extensive work on the topic of object detection is present. A few years ago Regional Convolutional Neural Networks (R-CNNs) were introduced as an improvement to object detection methods using CNNs. R-CNN relies on the assumption that only one single object of interest will dominate in a given region. It combines CNN with the principle of region proposals [7]. This is a two-stage process where the first step is the generation of a set of possible object locations and the second step classifies each location as a fore- or background class using a CNN [16]. This framework currently dominates when looking at the COCO benchmark [17]. Most state-of-the-art object detectors rely on R-CNN. *Faster R-CNN* combines the training of Regional Proposal Networks (RPNs) and Fast R-CNN [6] to share convolutional features, resulting in a low-cost region proposal computation [23]. *Mask R-CNN* extends Faster R-CNN by creating masks of regions of interest, allowing for alignment between input and output images [8]. *Feature Pyramid Networks (FPN)* can also be used by different R-CNN methods, introducing a multi-scale pyramidal hierarchy to increase the robustness of object detection on different scales [15]. Lastly, different from the aforementioned two-stage models, *YOLO (You Only Look Once)* is the state-of-the-art in single-stage object detection, using a single CNN to both localize and classify data [22].

This paper aims to measure and optimize the performance of a state-of-the-art object detection model on scenes

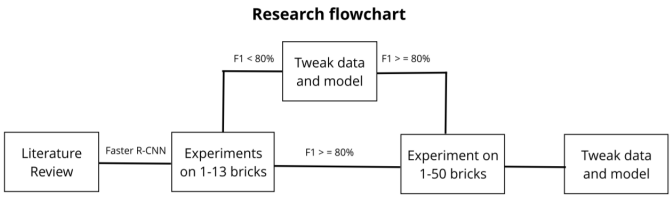

Figure 1: Simple flowchart depicting the research structure.

with multiple LEGO pieces. The approach is two-fold: firstly, a literature review will be conducted to analyze different object detection methods, their features and how they will potentially perform on the LEGO data. Secondly, an experiment will be performed to measure and optimize the performance of the aforementioned method, using the PyTorch frameworks for object detection [1]. The following questions are investigated:

- Which state of the art model in object detection is most suitable to be applied to scattered LEGO scenes?
- What parameters of the data should be tweaked to improve its performance on LEGO scenes?
- What parameters of the model should be tweaked to improve its performance on LEGO scenes?

The main contributions of this research are a literature review selecting an object detection method to apply to a new LEGO dataset, a performance analysis of Faster R-CNN on this dataset and optimizing the model for cluttered scenes of small LEGO bricks (Figure 1).

The related work discusses and compares the state-of-the-art object detection techniques and relates them to the application specific to this paper, namely LEGO scenes and the Faster R-CNN model and architecture are explained. Subsequently the experimental setup is discussed. This section specifies the methods of data collection, annotation and preprocessing, explains the module, hyperparameters and optimization technique used for the model implementation, and discusses evaluation methods of the experiment. Furthermore, the reproducibility and ethics of the experiment are reflected upon, followed by a display and discussion of the results and the conclusion. This includes future work to be done in testing and improving the performance of object detection methods on LEGO scenes.

2 Related Work

To decide which model will give a desirable speed vs. accuracy trade-off for the LEGO dataset, the following sections give a brief overview of the timeline of object detection history, which can roughly be divided in three groups.

2.1 Classic Object Detectors

Most early-day object detectors make use of the sliding window paradigm, meaning that every region in an image is considered as relevant and possibly containing an object. Furthermore, there a few machine learning-based (or pre-deep learning) approaches for calculating representative features, which is step two of the object detection pipeline (Figure 2).

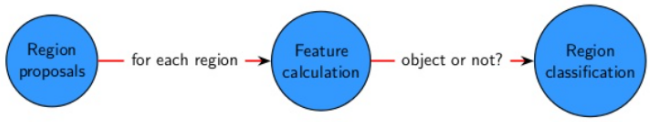

Figure 2: The general object detection pipeline, *retrieved from* [5]

A noteworthy early implementation of object detection is the Viola-Jones object detection framework using HAAR features [27]. Dalal and Triggs were able to significantly increase the performance of pedestrian detection by using HOG (Histogram of Gradients) [3]. Additionally by taking a dynamic programming approach to implement a cascade classifier, object detectors reached a higher performance on more object classes [4]. These methods are slow and are not widely implemented anymore in present-day computer vision tasks. However, soon a leap was taken towards a more efficient and accurate method of object detection, using deep learning-based approaches [7]. This meant that typically CNN's could be used to facilitate end-to-end object detection, without having to specifically define features.

2.2 Two-stage Object Detectors

Two-stage detectors split up the detection process in two parts. As introduced with selective search [26], the first stage identifies candidate object locations while filtering out as many irrelevant regions as possible. This means that the model does not need to consider every single part of the image in a sliding window manner anymore. The second stage then classifies those regions into foreground or background classes. The second stage saw a big improvement in accuracy with the introduction of R-CNN. Leading methods in this domain of two-stage detectors include Fast R-CNN, Faster R-CNN and Mask R-CNN, each one an extension of its predecessor. Though a massive speed-up was achieved from the aforementioned classic methods, these two separate stages require the image to be "looked" at twice by the algorithm. One-stage detectors, described next, only require a sample to be "looked" at once by the algorithm, thus resulting in an even bigger speed-up.

2.3 One-stage Object Detectors

SSD, YOLO and RetinaNet are prominent one-stage detectors [16; 18; 22]. These methods are significantly faster than Faster R-CNN for example, but score worse in terms of accuracy. Furthermore, this work shows that two-stage detectors such as Faster R-CNN can achieve similar speeds by reducing the image resolution and number of region proposals [9]. There are scenarios where one could wish to trade accuracy for speed, but considering the new LEGO dataset proposed in this paper and the complex attributes of this dataset, the experiments will focus more on achievable accuracy rather than speed.

In conclusion, related work shows that both classic and one-stage object detectors respectively have a lower efficiency and accuracy than desirable for the LEGO datasets. Mask R-CNN, though an extension of and reaching a higher accuracy than Faster R-CNN, is more costly to implement because it requires and returns additional information, namely object masks, that are not relevant to the first iteration of this project. Therefore, Faster R-CNN is the ideal trade-off when considering speed, accuracy, accessability and understandability.

3 Faster R-CNN

To better understand Faster R-CNN in the context of the following experiments, a short description will be given of the working of the model.

As the name suggests, Faster R-CNN is the successor of Fast R-CNN (Figure 3).

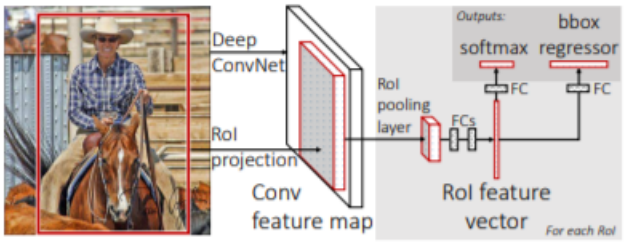

Figure 3: Fast R-CNN architecture [6].

Fast R-CNN uses a single CNN to extract features for the entire image and also creates a set of RoIs (Region of Interest) using selective search. The RoI layer is a special case of the Spacial Pyramid Pooling (SPP) layer with only one pyramid level. This pooling layer extracts fixed-length feature maps from the features in every RoI. The Fully Connected (FC) layer then produces two outputs, using softmax probability to classify the object and a regressor to localize the object.

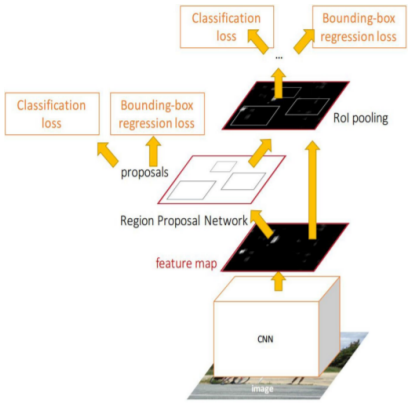

Figure 4: Faster R-CNN architecture [23].

The main improvement made by Faster R-CNN is that the expensive selective search is replaced by a Region Proposal Network (RPN) (Figure 4). RPNs make use of so-called ”anchor boxes” of different aspect ratios and sizes to go over the image and create region proposals. The RPN shares convolutional features with a CNN object detector, performing binary classification to classify whether the region in an anchor box contains an object or no object. This method creates much less region proposals and with a higher accuracy.

Although Faster R-CNN reaches top accuracy on the COCO benchmark, the COCO dataset used to evaluate the performance of object detectors has different properties than scenes of scattered LEGO pieces [17]. Therefore, a high performance on the COCO dataset might not be translatable to an equally high or even sufficient performance on the LEGO dataset. Faster R-CNN does not address challenges introduced by small objects, cluttered arrangement and arbitrary orientations. There are models that have modified the Faster R-CNN framework to be suitable for small object detection, however these are not tested well enough to be considered reliable and to be recognized as ”state-of-the-art”. The first experiments will start by comparing the performance of a state-of-the-art method as recognized by the COCO benchmark. Based on the evaluation of the performance of Faster R-CNN, future experiments can be designed to test the performance of modified Faster R-CNN models, models built on the Faster R-CNN model (such as Mask R-CNN) and even one-stage detectors such as YOLO or RetinaNet.

4 Experimental Evaluation of Faster R-CNN

The experiments designed to evaluate the performance of Faster R-CNN on new datasets have several different phases. The following sections provide insight into the dataset generation, data preprocessing, the model implementation and how the performance was evaluated.

4.1 Data collection and annotation

To test the performance of object detection models on a new dataset, said dataset first needs to be created. A set of parameters was set up in order to have more control over the data, namely lighting, camera angle, background, quality and number of bricks in an image. Furthermore, we can distinguish between three different types of data. The ”real” dataset, consisting of manually taken pictures of LEGO bricks, the synthetic dataset, consisting of rendered images containing 3D models of LEGO bricks and the so called ”cut and paste” dataset, where pictures of individual bricks are cut out and photoshopped over a background. These different methods of data collection vary in efficiency and cost but give different levels of control over the aforementioned parameters. The time and cost constraints of this research did not allow for robust and generous collection of a real dataset. One of the disadvantages of speeding up data collection was a repetitive dataset, essentially choosing a large, angle-varied dataset over smaller, brick-varied dataset (Figure 16).

Starting the experiment on synthetic data (rendered and photoshopped) was more efficient with getting initial results,

Figure 5: Three separate images from the real dataset, showing the same bricks from several angles.

since real data generation took longer to be ready, but it can also aid in testing and verifying the results on real data. More details about the conditions and methods of data collection can be found in *Dataset Generation Methods for Multi-Label Images of LEGO Bricks* [13].

The data annotation is again done using different methods, as described in *Evaluating Methods for Improving Crowdsourced Annotations of Images Containing Lego Bricks.* [20]. An annotations contains the name of the image and for each object in this image its brick id (i.e. the label) and the (x_1, y_1, x_2, y_2) coordinates representing its bounding box.

4.2 Data preprocessing

There is not a lot of data preprocessing or augmentation described in the original paper. Faster R-CNN is originally trained and tested on images resized to 500×375, so all LEGO images are first resized to 500×375 before being fed to the detector. Resizing the data also allows for a more efficient training and testing process. Furthermore, the PyTorch object detection model ensures that the data is normalized and horizontally flipped for a more robust detector [1]. For rendered data the bricks became extremely small due to a far camera angle combined with the resizing, so for this dataset the images were also cropped down to the bricks.

4.3 Faster R-CNN implementation

For the evaluation of an "out-of-the-box" Faster R-CNN model, as many hyperparameters as possible are set to match the originally published values. An SGD optimizer is used from the *torch.optim* package to reduce the loss. The loss is essentially a penalty for a bad prediction, which has a different value for different model parameters. The loss function for Faster R-CNN combines both the losses from classification (cls) and regression (reg) by comparing each predicted classification label to its ground truth for each anchor, and the same for each predicted bounding box. The function is defined as

$$L(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{1}$$

,where p_i, p_i^*, t_i and t_i^* are the predicted probability for anchor i being an object, the ground truth label, the predicted bounding box coordinates and the ground truth bounding box coordinates respectively. N_{cls}, N_{box} and λ are normalizing terms and a balancing parameter preset to 256, 2400 and 10, in line with the original paper.

Suppose we have a model parameter weight w_1, Stochastic Gradient Descent (SGD) computes a gradient at a specific point of the loss curve over a single image from the dataset, which can point us in the right direction for an optimal value for w_1. The size of the steps taken in that direction are depicted by the learning rate (lr). Since SGD only uses a batch size of one it is noisy, but also efficient. The lr was set to 0.005 with a step size of 3, meaning that the lr is decreased by $10\times$ every three epochs. The momentum and weight decay are set to 0.9 and 0.0005, again, matching the parameter settings in the original Faster R-CNN model [23].

The *torchvision.models.detection.fasterrcnn_resnet50_fpn()* module is used for executing this experiment. This module implements a Faster R-CNN model with ResNet-50-FPN as the backbone, an image classifier using 50 layers. ResNet (Residual Network) is similar to a plain CNN, but provides shortcuts between layers. Essentially, if the output to a certain layer matches the input of a couple layers further ResNet skips over these to reduce the training cost but also increase accuracy [14]. The FPN is a feature extractor which creates feature maps at different scales and feeds them to the RPN to create anchor boxes. The FPN makes the model more robust against scale variation [10].

In training mode, the model expects two inputs, a list of images in the format [number of color channels, height, width] and a list of "targets", containing the information for each image that we want to train on, i.e. bounding boxes depicting the location of each object and labels indicating the class of each object. During training the classification and regression losses are returned (Equation 1). In evaluation mode the model takes an image as an input and outputs a prediction for the image, containing the predicted bounding boxes and labels for each object in the image as well as a confidence score for each prediction.

Additionally, instead of training the torchvision Faster R-CNN model from scratch, we finetune the model from the pretrained version, trained on the COCO dataset. By using transfer learning one enables their custom model to inherit knowledge from being trained on a large dataset (Figure 6). This is especially beneficial for training and testing on small custom datasets.

4.4 Scaling up and optimization

To test the performance of an existing object detection model on a new dataset, especially one as complex as in the LEGO use case, one must start with a simplified version of this dataset, containing up to 13 bricks in this case. However, considering that the use case is a situation where one is looking for a specific brick in a pile of LEGO pieces, we also performed the same experiment on a small dataset consisting of 1,191 rendered images containing 50 bricks per image.

As well as looking at how changing the dataset to be more complex influences the performance results, it is

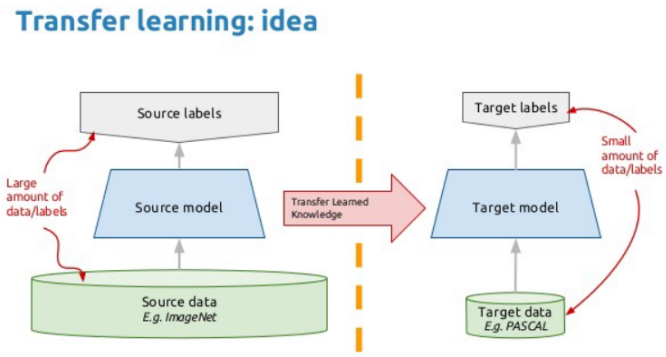

Figure 6: The concept of transfer learning [12].

also interesting to see how changing the model itself can change the results. Since the biggest difference between the LEGO datasets and COCO dataset is the number and size of objects in the images, which is the challenge addressed in this research, tweaking the anchor box generator in the RPN layer is expected to have the largest impact on the performance of the Faster R-CNN model. The original paper uses the aspect ratios 2:1, 1:1 and 1:2, which correspond to the shapes of the LEGO objects in the images, but uses the sizes $(128^2, 256^2, 512^2)$ pixels (Figure 7). The objects in the

anchor	128^2, 2:1	128^2, 1:1	128^2, 1:2	256^2, 2:1	256^2, 1:1	256^2, 1:2	512^2, 2:1	512^2, 1:1	512^2, 1:2
proposal	188×111	113×114	70×92	416×229	261×284	174×332	768×437	499×501	355×715

Figure 7: Anchor box sizes and aspect ratios set in the original model [23].

LEGO datasets can have areas down to 4^2 pixels, so if many of these are cluttered in a small space they all fall within one anchor box and RoI. By tweaking these parameters to include smaller anchor boxes, the performance of Faster R-CNN might improve on datasets with small objects.

4.5 Method of evaluation

For each dataset a train-val-test set approach will be used, where the model will be trained on 80% of the data, validated on 10% of the data and tested on the remaining 10% of the data. By looking at train and validation learning curves we can see if the model is overfitting. Since the objectives of object detection are localization and classification, both of these tasks need to be taken into account when evaluating the model. A common metric for evaluating the localization accuracy is the Jaccard index, or Intersection over Union (IoU) [25]. When considering two bounding boxes P (prediction) and G (ground truth), the IoU can be determined with the formula $IoU = \frac{|P \cap G|}{|P \cup G|}$, indicating how much of the area of P overlaps with the area of G.
Evaluating the performance of the Faster R-CNN model on the given datasets is done by looking at the following metrics.

- Precision $P = \frac{TP}{TP+FP}$
- Recall $R = \frac{TP}{TP+FN}$
- F_1 score $= 2 * \frac{P*R}{P+R}$

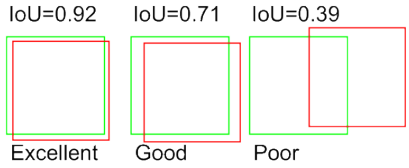

Figure 8: Using the IoU metric to evaluate localization [24].

TP (True Positives) indicates the number of bricks successfully detected by the model. FP (False Positives) indicates the number of non-brick objects that are falsely detected as bricks and FN (False Negative) indicates the number of bricks in the ground truth that the algorithm did not detect. A prediction is only considered a TP if the IoU of the bounding boxes is > 0.5 (localization) and the predicted label for the object corresponds to the ground truth label (classification). The precision score then tells us how the model performed relative to how many object it detected in total (decreases if FP increases) and recall indicates how the model performed relative to the total amount of object it should have recognized. The F_1 score is the harmonic mean of the previous two metrics.

The benefit of displaying both the precision and recall separately instead of only the harmonic mean of the two is that it allows one to determine if the model performs better in one than the other. The F_1 score is useful for seeing the overall performance of the model in one glance.

Predictions are first filtered on their confidence scores. Since the model predicts many bounding boxes for each brick (Figure 9), the chance that one of them also has the right label is higher, which can give a false indication of the performance of the model. To filter out these coincidental results, predictions are only taken into account if they have a confidence score > 0.5, i.e. if the model is over 50% sure that it has the correct prediction (Figure 10).

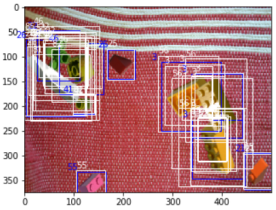

Figure 9: Before thresholding the confidence scores on cut and paste data.

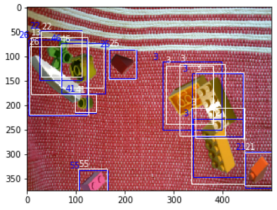

Figure 10: After thresholding the confidence scores on cut and paste data.

5 Responsible Research

Throughout this project consistent effort was put in to ensure the ethical integrity from the first experiment design all the way through to processing the results of the experiment. The following two sections address and reflect upon certain concerns that came up during the project.

5.1 Ethical concerns

Data collection and annotation has been an ethically ambiguous topic for a while. The data collection for this project was done by a group of students (including myself) voluntarily taking, photoshopping and rendering images to create a dataset that was sizeable enough for a first iteration of the experiment. These images needed to be annotated however. A quarter of these images was annotated by the aforementioned group of students, but due to heavy time constraints the rest of the annotations were crowdsourced using Amazon Mechanical Turk (MTurk). As discussed by Oltmans [20], determining a minimum wage is hard, since it is unknown where the anonymous workers live and if this is their main source of income. Furthermore, the resources for this project were scarce. MTurk workers do choose their "tasks" however, which means that workers could volunteer to perform our task and our annotations were obtained on a voluntary basis.

5.2 Experimental reproducibility

The model used for this experiment is available as a TorchVision library and finetuning of the model was done according to the Mask R-CNN finetuning tutorial by TorchVision [2]. Furthermore, both the code and datasets used in this paper have been made public, as well as specifications for the conditions under which the experiments were performed, enabling the reproduction of these experiments and results.

6 Results and discussion

The experiments were performed on the HPC Cluster, using two CPUs with 9GB memory and a GPU with CUDA 10.0 and CUDNN 7.4 to report the results. During training, a batch size of 10 with 5 workers is used, with a batch size of 5 with 2 workers when validating and testing. The model is trained for 20 epochs, since the F_1 score did not increase with more than 1.5% when training longer and training for longer than 20 epochs violated the time constraints of the Cluster. I.e. 20 epochs gave the best speed vs. accuracy trade-off.

Main experiments

The first experiments on the datasets are rather promising for the cut and paste and real data, but the synthetic data has a relatively low F_1 score (Table 1).

	Data size	Precision	Recall	F1 Score
Rendered	5,000	58.69%	59.82%	59.25%
Rendered-Cropped	5,054	91.94%	96.14%	93.99%
Cut & Paste	10,000	89.38%	96.19%	92.66%
Real	3,062	81.99%	79.33%	80.64%

Table 1: Evaluation of Faster R-CNN on different datasets containing up to and including 13 bricks after training for 20 epochs

How can we improve the performance of Faster R-CNN on rendered data?
The renders were made with a randomized camera position to simulate as many real-life data situations as possible, which resulted in some zoomed out images which, after down-sampling, contained bricks that were barely recognizable, even to the human eye (Figure 11). There was a lot of empty background space in these cases as well, so cropping down to the bricks resulted in a more recognizable image (Figure 12) and resulted in a higher F_1 score.

Figure 11: A rendered image after down-sampling.

Figure 12: A rendered image after cropping down to the bricks.

The significant increase in performance could be a product of overfitting on the test set, but the rendered data actually has a higher degree of variety in camera positions, brick compositions, lighting and backgrounds than the real data, so this is unlikely.

How does Faster R-CNN perform on the real data?
The real dataset is of significantly lesser quality than the synthetic datasets, both in the uniqueness of the images and the amount of available images. This dataset is therefore prone to overfitting, which is slightly reflected in the gap between training and validation losses in figure 13.

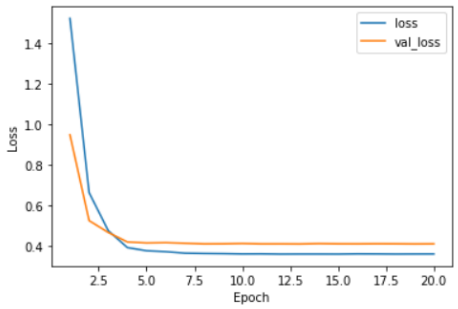

Figure 13: The train and validation learning curves for real data.

The gap is small however, and the recall for real data is lower than expected considering the anticipated overfitting. Part of this might be due to the fact that object detection not only classifies but also localizes bricks. Although the real images are shot in groups of three with the same bricks from different angles (Figure 16) and it would be easy to overfit with the classification, the bricks are in a different spot in every image.

How does Faster R-CNN perform on the cut and paste data?
The cut and paste dataset is significantly larger than the other ones, which partially explains the high F_1 score, but about half of the cut and paste data contains samples that are "easier" than the real data we initially wanted to test

the model on. In some cases the bricks are significantly larger than in the real data, which allows for variety in the dataset on one hand but on the other hand agrees with the pretrained anchor boxes that are used to extract RoIs better than smaller bricks (Figures 14, 16). Furthermore, perfectly photoshopping a brick onto a background is nearly impossible, so in some cases there is a white "glow" around the brick that distinguishes the brick from the background. This visible difference between brick and background can also be caused by inconsistencies in lighting between the background and bricks. Bricks in these images are easier to localize and classify, which can cause skewed results. It is possible to create cut and paste data that is more similar to the real data however, as shown in figures 15 and 16.

Figure 14: An image from the cut and paste dataset, showing the inconsistency with an image from the real dataset.

Figure 15: An image from the cut and paste dataset, showing the similarity with an image from the real dataset.

Figure 16: An image from the real dataset.

Scaling up

	Data size	Precision	Recall	F1 Score
Rendered-50 bricks	1,191	73.64%	45.70%	56.40%
Rendered-50 bricks-new RPN	1,191	81.73%	64.82%	72.30%

Table 2: Evaluation of Faster R-CNN on a rendered datasets containing up to and including 50 bricks per image after training for 20 epochs

How can we improve the performance of Faster R-CNN on rendered data with up to and including 50 bricks?
When training and testing on renders containing up to 50 bricks we see a decent precision of 73.64%, but a low recall of 45.70% (Table 2). This indicates that the model makes less high-confidence predictions on this dataset, lowering the amount of FP, thus increasing the precision, but at the same time lowering the amount of TP, thus decreasing the recall (Figure 17). When lowering the confidence score threshold to 0.25, significantly more correct results are shown (Figure 18).

When increasing the number of bricks in an image, the image becomes more cluttered than before when there were a maximum of 13 bricks in an image. Since the anchor generator in the RPN is pretrained on less cluttered samples with larger samples, some bricks can end up in the same anchor box in this case. After tweaking the RPN layer a significant improvement is achieved. Modifying the RPN layer to accommodate for a larger number of small and cluttered bricks by having appropriately sized anchor boxes resulted in a 15.9% increase of the F_1 score. The recall went from 45.70% to 64.82%.

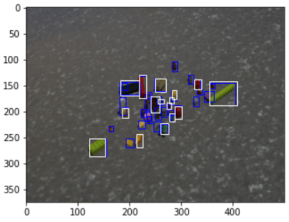

Figure 17: Thresholding confidence scores at 0.5.

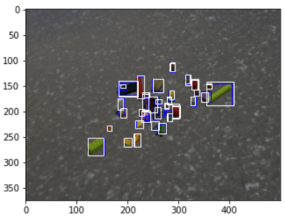

Figure 18: Thresholding confidence scores at 0.25.

This indicates that the previous datasets with less bricks per image were too simple for the initial use case. The bricks in those images were large and uncluttered enough for the Faster R-CNN model to reach high performance while being optimized for the COCO dataset.

7 Conclusions and Future Work

In this work we measured the performance of a state-of-the-art object detection model on a custom dataset of scattered LEGO pieces. This was done by acknowledging which object detection model is most appropriate to test on the new dataset and what parameters of the data and model should be tweaked to improve the performance. Conducting a literature review showed that Faster R-CNN gives the ideal accuracy vs. speed trade-off for the dataset and resources available during this research.

In conclusion, the cropped and rendered, cut and paste, and real data reach an average F_1 score of 90%, with real data being the smallest and worst performing set. This high performance with the original Faster R-CNN model is due to inconsistencies in the dataset, as shown by easier cut and paste images skewing the performance, and limited dataset sizes. We can clearly see the effects of data manipulation on these datasets, where a cropping of images to be more visible had a significant impact on the performance. When looking at samples in a dataset with 50 bricks, we notice a significant drop in the F_1 score. Modifying the RPN layer to accommodate for a larger number of small and cluttered bricks, by having appropriately sized anchor boxes resulted in a 15.9% increase, indicating that the aforementioned datasets were oversimplifications of the real use case.

This study can be extended in a few different ways. Firstly, one can tweak the parameters of Faster R-CNN in this experiment. Trying a less noisy optimizer function, like Adam, and different lr combinations can improve the performance of the model. Secondly, one can perform the same experiment with different object detection models, like Mask R-CNN, YOLO or RetinaNet, to see how they compare to the baseline performance of Faster R-CNN. Thirdly, one can perform the same experiment on a more complex dataset, containing images with 50+ bricks, more occlusion, etc.

In all of these situations it holds that it is necessary to

create better datasets, both in terms of quality (variety and realisticness) and quantity. Applying more complex data augmentation for training, like noise, rotation and scaling, will increase the robustness of the system.

References

[1] Torchvision models.

[2] Torchvision object detection finetuning tutorial.

[3] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR05)*, page 886–893, 2005.

[4] Pedro F. Felzenszwalb, Ross B. Girshick, and David Mcallester. Cascade object detection with deformable part models. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, page 2241–2248, 2010.

[5] Preferred Networks Follow. A brief history of object detection / tommi kerola, Sep 2019.

[6] Ross Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, page 1440–1448, 2015.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[8] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017.

[9] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and et al. Speed/accuracy trade-offs for modern convolutional object detectors. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[10] Jonathan Hui. Understanding feature pyramid networks for object detection (fpn), Mar 2018.

[11] Jonathan Hui. Object detection: speed and accuracy comparison (faster r-cnn, r-fcn, ssd, fpn, retinanet and yolov3, Mar 2019.

[12] Integrate.ai. Transfer learning explained, Aug 2018.

[13] Berend Kam. Dataset generation methods for multi-label images of lego bricks. 2020.

[14] Sihan Li, Jiantao Jiao, Yanjun Han, and Tsachy Weissman. Demystifying resnet, 2016.

[15] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 936–944, 2017.

[16] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, page 2980–2988, 2017.

[17] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science*, page 740–755, 2014.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, page 21–37, 2016.

[19] Anurag Mittal and Larry S. Davis. M2tracker: A multi-view approach to segmenting and tracking people in a cluttered scene. *International Journal of Computer Vision*, 51(3):189–203, 2003.

[20] Rembrandt Oltmans. Evaluating methods for improving crowdsourced annotations of images containing lego bricks. 2020.

[21] Katyanna Quach. You looking for an ai project? you love lego? look no further than this reg reader's machine-learning lego sorter, Dec 2019.

[22] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, page 7263–7271, 2017.

[23] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[24] Adrian Rosebrock. Intersection over union (iou) for object detection, Nov 2016.

[25] Stephanie. Jaccard index / similarity coefficient, Aug 2019.

[26] J. R. R. Uijlings, K. E. A. Van De Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, Feb 2013.

[27] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, page 511–518, 2001.