IMPROVING OBJECT DETECTION USING SYNTHETIC DATA & CYCLEGAN

A PREPRINT

December 7, 2020

ABSTRACT

In order to train deep learning models, it is important to have a significant amount of data. As data can be hard to come by alternatives have been explored in recent years, an example of this is *domain randomization(DR)*[1]. DR enables the generation of synthetic training data, aiming to reduce overfitting to the synthetic model. This leads to better generalization to real data. This work explores a possible way to improve the quality of such synthetic data by utilizing *unpaired image-to-image translation* in the form of CycleGAN[2]. We assume access to a small amount to real data, as this is needed to train CycleGAN. To evaluate the effect of using CycleGAN, we implement the object detection network Faster R-CNN[3]. Our experiments clearly show that using CycleGAN on our data does ***NOT*** offer any improvement, but rather makes training impossible. More concretely, Faster R-CNN trained on 99.9% synthetic data generated using DR and 0.1% real data achieves 73.4% mAP. When trained on the output of CycleGAN, it achieves 0.0% mAP.

***K*eywords** Convolutional Neural Network · Deep Learning · Domain Randomization · Generative Adversarial Network · Object Detection · Region-based Convolutional Neural Network · Region Proposal Network · Synthetic-to-Real · Unpaired image-to-image translation

1 Introduction

Data is an integral part of any *deep learning(DL)* model, but obtaining enough data can at times be difficult. A potential solution to this issue, in a computer vision setting, is the use of images generated from synthetic 3D models. Such an approach eliminates the time consuming process of manual data collection. Synthetic data has a problem named the *reality gap*, which concerns the gap between synthetic images and real images. Since we need to generalize well on real images rather than synthetic images, it is necessary to bridge the reality gap.

In [1] Tobin *et al.* propose a solution to bridge the reality gap by using *domain randomization(DR)*. This works by randomizing the environments of the synthetic model, thus emulating a lot of variability in the images, see subsection 2.3 for details.

Another method is to "translate" one image into another is by using *image-to-image translation*. An example of this is CycleGAN [2], which focus on the setting of *unpaired* image-to-image translation, see subsubsection 3.2.2 for details. This is especially useful in settings where we do not have one-to-one correspondences between the images we seek to translate from, i.e. source domain, and the images we seek to translate to, i.e. target domain.

In this report, we will perform experiments to determine the effects of using CycleGAN in an object detection setting. Specifically, where the training data is predominantly synthetic, i.e. 99.9% synthetic and 0.1% real. More concretely, we contribute the following;

1. An implementation of an object detection model Faster R-CNN[3], which we will be trained on images consisting of the 99.9% synthetic images generated using DR and the 0.1% real images.
2. An implementation of an object detection model Faster R-CNN[3], which we will be trained on images produced by the unpaired image-to-image translation network CycleGAN. The source domain of CycleGAN will be the 99.9% synthetic images generated using DR and target domain will be the 0.1% real images.

We evaluate these implementations and determine the effect of using CycleGAN to improve the performance of Faster R-CNN. Implementation of CycleGAN can be found in [2].

2 Related Work

2.1 Object Detection

Object detection is important in computer vision as this allows natural feature tracking and doing so using *convolutional neural nets(CNNs)* show good results [4, 3, 5]. The task of object detection is to predict classes for multiple objects in an image as well as the positions of these objects. These positions are indicated by bounding boxes(*bboxs*), which are rectangles encapsulation the different objects.

Multiple ways of performing object detection have been explored in the past few years, two such approaches are regression/classification-based approach, e.g. SSD [4], and Region-based Convolutional Neural Networks (R-CNNs), e.g. Faster R-CNN[3]. Regression/classification-based will in do this in a single forward pass consisting of extracting feature map, for each feature map cell k bounding boxes are predicted and for each of these boxes c class scores are computed.

Faster R-CNN runs in *two stages*; the *first stage* is the Region Proposal Network (RPN), which efficiently generates accurate rectangular region proposals from the feature map generated for an input image. More concretely, the feature map inputted to the RPN will go through a number of convolutions. This means that a $n \times n$ kernel slides across the feature map and for each location, then k rectangular regions of 3 scales and 3 aspects ratio are proposed. Each of these proposals are then mapped to a lower-dimensional vector, e.g. 256-d, after which it is parsed into two separate fully-connected (fc) layers which predict *objectness* score and bounding box, see Figure 1. Note that the objectness score indicate confidence of being an object versus background, i.e. class-agnostic. Furthermore, non-max suppression(NMS) is used in terms of the objectness score to reduce the number of region proposals.

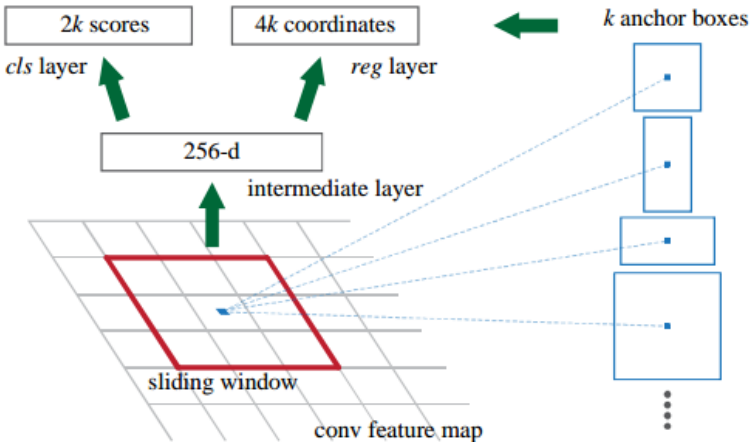

Fig. 1. Region Proposal Network (RPN) [3].

Second stage is to combine the region proposals and the feature maps extracted from the image into several Region of Interest (RoI). These are then parsed into the pooling layer, which reduce dimensionality thus reducing number of parameters. This reduction of parameters mitigate overfitting as well reduce training time since fewer parameters need to be determined. More importantly, pooling is needed as the fc layers need a fixed size input and the RoI might be of different sizes. This is afterwards parsed into the fc layers classification and bounding-box regression is performed.

2.2 Feature Pyramid Network for feature extraction

In [6] He *et al.* propose an extension to Faster R-CNN with some architectural changes. One of these changes are to use a *Feature Pyramid Network(FPN)* for feature extraction instead of a regular CNN. FPNs use a top-down architecture to generate a "pyramid" consisting of the input image on different scales. For each of these a feature maps is constructed and parsed into the RPN to generate region proposals. Each of the region proposals are combined with the feature map with the proper scale to define the RoIs, e.g. large region proposals then small scale feature maps. The use of such a network yields the best results for He *et al.* in terms of accuracy as well as speed.

2.3 Domain Randomization

Domain randomization(DR)[1] is a method for addressing the reality gap, when training on purely synthetic data. The method itself takes root in the idea of constructing training data based on a synthetic object and subjecting this to various uniformly random environments. The environment is changed in different ways such as; number and shapes of distracting (sometimes occluding) objects, color of all objects, lighting as well as textures chosen with different colors, gradients of colors and checkered patterns of two colors. This will create variability in the training data and if this is significant enough then the model being trained will generalize well on real data.

As highlighted in [1] DR is just one of many ways of bridging the reality gap, other examples include *domain adaptation* and *system identification*. The former method is to train model on a source domain and then adapting this domain for instance by re-training the model on the target domain. This aforementioned scenario does however require the presence of real training data to some extent. In system identification the goal is to adjust parameters of the simulation, such that is matches the behaviour of the real data. Such adjustments are however time-consuming and error-prone.

The work of Tobin *et al.* in [1] show that DR indeed yields reasonable results when seeking to solve object detection, i.e. the model generalize well on real data though only trained of the synthetic data.

2.4 Unpaired Image-to-Image Translation

Image-to-image translation is a computer vision problem, concerning itself with learning a mapping $G: X \rightarrow Y$ from a source domain X to a target domain Y. The objective is to make $G(X)$ indistinguishable from Y, i.e. $G(X) \approx Y$. This require capturing characteristics in one image and mapping this to another image. In image-to-image translation there are two main variations, namely *paired* and *unpaired* input-output examples, a.k.a. *supervised* and *unsupervised* settings respectively. Paired image-to-image translation performs better than unpaired image-to-image translation, but available data sets with paired input-output examples are usually relatively small compared to data sets with unpaired input-output examples.

In [2] Zhu *et al.* introduce a *cycle consistency loss* in the construction of the *Generative Adversarial Network(GAN)* CycleGAN, thus imposing that $F(G(X)) \approx X$, where $F: Y \rightarrow X$ as shown in Figure 2. This means that the mappings F and G should be each others inverses, which is imposed using the cycle consistency loss while simultaneously training both mappings. By using such a loss, the space of potential mappings is significantly reduced, which help inputs and outputs being paired in a meaningful way. CycleGAN also use least-squares loss for the *adversarial losses* which describe the indistinguishablility of the mappings, more concretely that $G(X) \approx Y$ and $F(Y) \approx X$. Furthermore, an identity loss can be used to regularize CycleGAN. If an image of the target domain is inputted then the output will be close to the input. If not used CycleGAN can freely change the tint of the input images.

The evaluation by Zhu *et al.* in [2] show that CycleGAN performs reasonably well, especially on tasks involving color and texture changes. It does however struggle on certain data sets. For instance, failures occur when the training data is significantly different to the test data, e.g. trained on source domain of horses with no rider but tested using horses with rider. Furthermore, the performance gap between supervised and unsupervised data sets, vary from small to large in favour of the supervised setting. This leads Zhu *et al.* to highlight the possibility of using a semi-supervised setting which potentially could increase performance of CycleGAN.

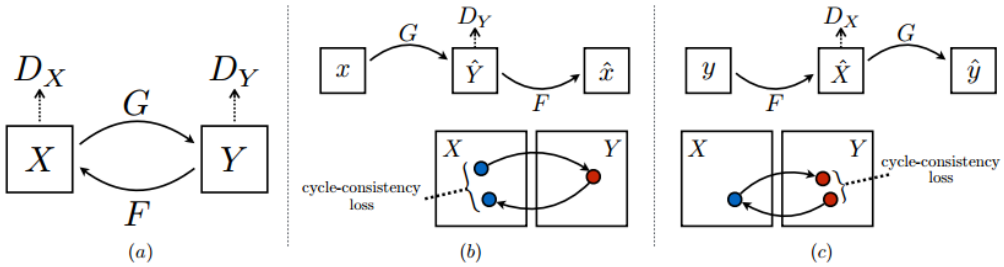

Fig. 2. a) shows the mappings G and F, b) $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ is called *forward cycle consistency loss*, c) $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$ is called *backward cycle consistency loss*. Note that D_X and D_Y distinguish source/target domains and are used in the adversarial loss [2].

2.5 Data augmentation and Transfer learning

Data augmentation In [7] T. He *et al.* experiment with different approaches to boost accuracy of CNNs. They achieve an increase in accuracy using different data augmentations on the training data, e.g. random color hue change, add random noise and flip horizontally. Note that any random augmentation should not be used during the validation.

It is worth noting that DR, as described in subsection 2.3, encapsulate these aforementioned augmentation methods. For instance DR will generate data which has a lot of variability by using random colors and noise.

Transfer learning When training a CNN from scratch, a very large data set is needed. Alternatively one can use *transfer learning*, which enables training of a CNN with a much smaller data set. There are different approaches on how to do this depending on the amount and type of data [8]. *Fine-tuning* a pre-trained CNN is one approach, but this risk overfitting. This is especially the case when the data set used is not large or similar enough, to the data set used in the pre-trained CNN. The choice of which layers and how many should be fine-tuned is dependent on the data, see [8] for details. Alternatively, one can use the pre-trained CNN as a *fixed feature extractor*(encoder), and then simply train a classifier on the extracted features using the data set.

3 Methods

3.1 Data set

The data set we will use consists of synthetic and real images of a drill. To generate the synthetic images we use an image generator provided by Oliver Gyldenberg Hjermitslev from the Alexandra Institute [9]. This will generate the images using DR and output an image and the corresponding segmentation used to compute the bounding box.

The real images and their corresponding segmentation mask originates from the LineMOD data set [10]. The data set also contains the synthetic model, used in the image generator. One discrepancy between our data sets, is that the real images contain multiple objects. The synthetic images however only contain the driller, due to limitations in the image generator.

The overall distribution of the data is as follows; 5000 domain randomized synthetic images and 1188 real images from the LineMod data set. Note that only 50 of the real images will be used for training.

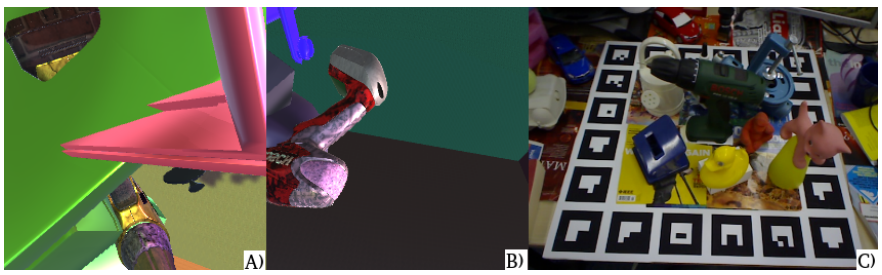

Fig. 3. A) and **B)** are examples of the synthetic images generated using DR, **C)** is example of a real image.

3.2 Implementation

In this section, we describe the implementations details of Faster R-CNN, see subsubsection 3.2.1, and CycleGAN, see subsubsection 3.2.2 as well as the chosen optimization strategies.

3.2.1 Faster R-CNN

The prediction of the bounding box(*bbox*) of the drill will be achieved with an implementation of Faster R-CNN. This will be implemented in Pytorch and trained using Colab. In this section, we present the details of this implementation.

Assumptions Due to the nature of our data and our overall goal, we can make some simplifying choices to the model. As mentioned earlier, the synthetic data holds only one object(the drill) for each image. The real images contain multiple object, of which only one instance is the driller. This allows us to modify the model to output a single bounding box(bbox) for each image. In the case where the model predicts multiple bboxs for a drill, we choose the bbox with the highest confidence score. Furthermore, we restrict the classification aspect of the prediction to be one of the too possible classes $\{drill, background\}$.

In the model we, contrary to [7], only apply random horizontal flip as data augmentation during training. This is due to the fact that the other data augmentations mentioned in [7] are already encapsulated by the use of DR.

Architecture The architecture of our Faster R-CNN implementation is nigh identical to that of [3], though with one exception. The exception is in terms of the backbone utilized for the feature extraction. In [3] the backbone is a regular CNN, such as ResNet50. However, we utilize a FPN using ResNet50, see subsection 2.2, as the backbone, as done in [6]. Such a backbone is expected to perform better, than simply using ResNet50 for the feature extraction. In the backbone each layer consists of consecutive convolutions followed by batch normalization, thus enforcing zero mean and unit variance throughout. The activation function used is *rectified linear unit(ReLU)*. We also note that the images are zero-centered, before going into the network.

Transfer learning Transfer learning plays a significant role when training deep learning models in practice, see subsection 2.5. We will employ transfer learning, by initializing the weights using a pre-trained model. More concretely, the weights of the Faster R-CNN model are pre-trained on the COCO data set[11]. As our data set is of a reasonable size, we expect that fine-tuning the pre-trained model will perform better, than using it as a fixed feature extractor.

Optimization The data used for training our model is highly dependent of experiments in question, see subsection 4.2 for details. It is however common to all experiments that we train on 80% of the data and test on 20% of the data, where the latter is only real images.

We train the model using the multi-task loss function proposed in [3], in which Ren *et al.* use both *log loss* and *smooth L1 loss*. We employ Stochastic Gradient Decent (SGD) with a batch size of 2, learning rate(lr) of 0.005 and weight decay of 0.0005 to penalize complexity. Note that the choice of such small batch size, is mainly due to us seeking to avoid local minima. This is further aided, by using momentum of 0.9. As suggested by T. He *et al.* in [7], we use linear lr warmup. We set lr to 0 and increase linearly towards it's intended value. This is done for the first epoch in our case and helps to avoid early overfitting, by reducing the primacy of early training examples. We also decay the learning rate by a multiplicative factor of 0.1 every third epoch. This type of decay is widely used as highlighted in [7]. Note that we train the model for 10 epochs.

3.2.2 CycleGAN

Our CycleGAN implementation is responsible for learning the mapping from our synthetic source domain, to the real target domain as described earlier. The implementation is done in Pytorch and trained using Google Colab. We will in this section go into the implementational details.

Assumptions The data generator provided by Oliver Gyldenberg Hjermitslev is only able to generate images with one driller. We therefore chose to implement CycleGAN to only learn a mapping of a single object. Furthermore, we do not assume that there is any predefined similarity function between the source and target domain.

Data For CycleGAN to work, it needs images from both domains. We therefore give the model 1100 images from synthetic domain X, and 50 images from target Y, i.e. LineMod. The model was trained on a total of 1040 images, and tested on 110 images. The structure of the data used, is defined as follows:

- *trainA* contains 1000 synthetic images of the drill, generated using DR.
- *testA* contains 100 synthetic images of the drill, taken at random from the entire data set of 5000 synthetic images.
- *testB* contains 40 real images from the LineMod dataset.
- *testA* contains 10 real images from LineMod dataset.

Note that the training/test split for CycleGAN is $90/10$, as this is the split used in [2].

Optimization To optimize the CycleGAN model, we use the Adam solver with a batch size of 1. The reason for this is that it gives the best results in [2]. We use the default $\lambda = 10$, where λ controls the relative importance of the cycle consistency loss. The model was trained with a learning rate of 0.0002 for 15 epochs, and then another 15 epoch with linear weight decay to zero. In total the model ran 30.000 iterations.

Unpaired Image-to-Image Translation Since we assume that there is no predefined similarity function, CycleGAN has to learn to approximate the special characteristics in the image. This is the case since there are no information provided in regards to which synthetic image matches up with given real image. Therefore, we have to use unpaired image-to-image translation even though paired yields the best results [2].

In Figure 4 we see the inputs and the output of running CycleGAN, this output is used to train the Faster R-CNN model, see subsection 4.2 for details.

Fig. 4. A) is the source domain, i.e. synthetic image(using DR), **B**) is the target domain, i.e. real image, **C)** is the learned mapping from source to target domain applied to the synthetic image.

4 Results

To test the performance of our different architectures, we perform a quantitative evaluation using 20% of the data to test on. The test data consist of real images, which are not subject to any random data augmentations as suggested in [7]. The test data consist of the real images while the nature of the training data is dependent on the experiment.

4.1 Evaluation Metrics

In terms of evaluation metrics, we use Intersection over Union (IoU) for the bounding box(*bbox*). This metric will calculate IoU$= \frac{target \cap prediction}{target \cup prediction}$, where $0 \leq IoU \leq 1$ and target is ground truth of the bbox. Then by using IoU we can define a true/negative threshold, e.g. if IoU of predicted-target pair exceeds the defined threshold then classify as true positive (TP). In the case where it is lower than the threshold then classify as false positive (FP). If the ground truth is present in the image and we fail to detect then classify as false negative (FN) [12].

Furthermore, we also utilize the standard classification metrics average precision (AP) and average recall (AR) to evaluate the placed bounding boxes. Note that precision $= \frac{TP}{TP+FP}$ and recall$= \frac{TP}{TP+FN}$. These metrics are used to compute the notion of *mean average precision(mAP)* and *mean average recall(mAR)*, where the mean is of the average precision or recall for IoU thresholds $[0.50, 0.95]$.

4.2 Experiments

To test the effect of using CycleGAN with the object detection model Faster R-CNN, we conduct the following experiments:

1. Fine-tune pre-trained Faster R-CNN, using synthetic(using DR) & real images.
2. Fine-tune pre-trained Faster R-CNN, using the 50 available real images.
3. Use pre-trained Faster R-CNN as fixed feature extractor on synthetic(using DR) & real images.
4. Faster R-CNN trained from scratch, i.e. no transfer learning, using synthetic & real images.
5. Fine-tune pre-trained Faster R-CNN using output of CycleGAN.
6. Use pre-trained Faster R-CNN as fixed feature extractor on output of CycleGAN.
7. Faster R-CNN trained from scratch, i.e. no transfer learning, using output of CycleGAN.

Note that *experiments 1,3,4* test the effects of using different transfer learning strategies to train Faster R-CNN. The data used in these experiments consist of 99.9% synthetic images generated using DR and 0.1% real images. In *experiment 2* we fine-tune Faster R-CNN using only the 0.1% real images, to show the effect of using synthetic image during training. The *experiments 5-7* test the effect of using different transfer learning strategies to train Faster R-CNN on the output of CycleGAN.

In Figure 5, an example prediction of the bounding box is shown. In Figure 6, we see the results of *experiments 1-4*. More concretely, we depict the mAP and mAR on the test data, i.e. only real images, for each epoch during training of Faster R-CNN. From this we see that fine-tuning the weights of the pre-trained model(*experiment 1*) yields the best results, namely $mAP = 0.734$ and $mAR = 0.765$. This is achieved while training on both the synthetic and real images. When only training on the real images(*experiment 2*) we see a significant drop in the performance, more concretely 0.109 for mAP and 0.093 for mAR. In *experiment 4*, where we train without transfer learning, we see a even larger drop in the performance compared to fine-tuning the weights. The drop is 0.145 for mAP and 0.126 for mAR. Lastly, we note that using the pre-trained model as fixed feature extractor(*experiment 3*) yield the worst performance with $mAP = 0.116$ and $mAR = 0.196$, i.e. drop of 0.618 for mAP and 0.569 for mAR.

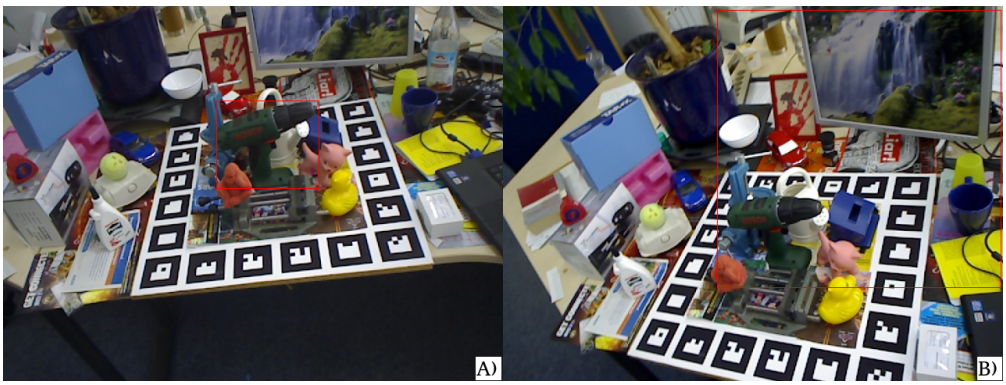

Fig. 5. Predicted bounding box by the model described in *experiment 1*(see **A**)) and in *experiment 5*(see **B**)).

The results of the evaluation of *experiment 5-7* can be seen in Figure 7. Here we see that for all experiments the mAP and mAR remain 0 throughout the training. We see in Figure 5, an example of the predicted bounding box made by the model described in *experiment 5*.

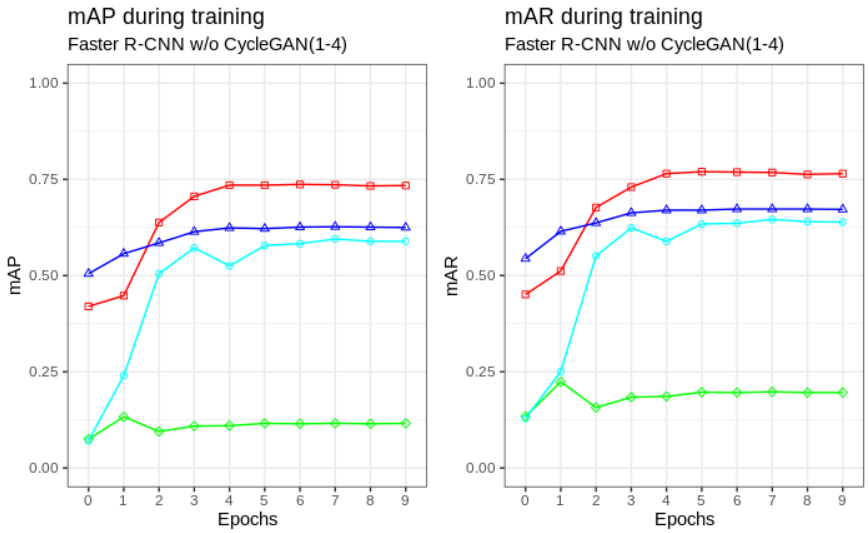

Fig. 6. *mAP & mAR* of each epoch during of Faster R-CNN using following transfer learning strategies and data: **RED**; fine-tuning on synthetic & real images, **BLUE**; fine-tuning on only real images, **GREEN**; fixed feature extractor on synthetic & real images, **CYAN**; no transfer learning on synthetic & real images

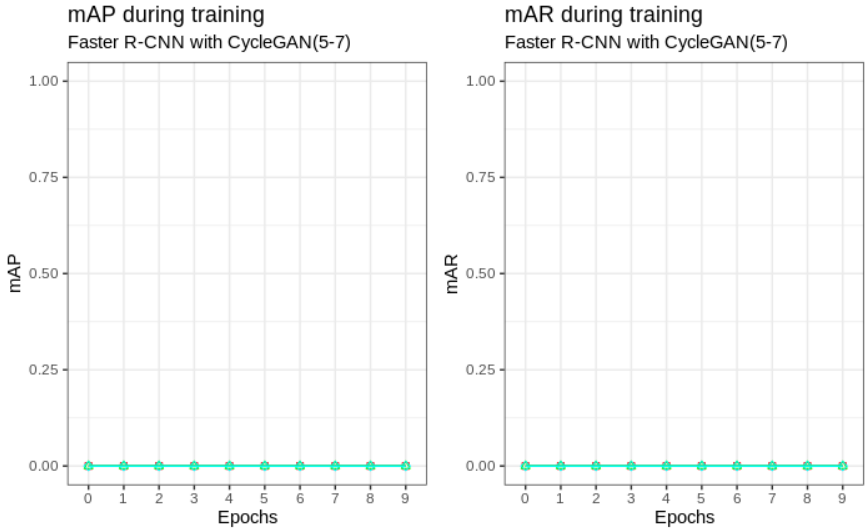

Fig. 7. Training results(*mAP & mAR*) of Faster R-CNN on output of CycleGAN using following transfer learning strategies: **RED**; fine-tuning, **GREEN**; fixed feature extractor, **CYAN**; no transfer learning

5 Discussion

The result of the evaluation clearly shows that using CycleGAN as an out-of-the-box tool for improving the quality of synthetic images, did NOT work when using unpaired input-output images. This can be confidently concluded in terms of our data by looking at the result in Figure 7. The main reason this is seen when looking at Figure 4. Here we see that the output of CycleGAN(see Figure 4**C**)), does not map the target domain in the desired manner. Instead the background is mapped onto the driller, making it impossible for the Faster R-CNN model to learn successfully. As argued by Zhu *et al.* in [2], we could expect better performance if we where to do this with paired input-output images. Alternatively, one can work in a semi-supervised setting, where some paired input-output images exists. This is not feasible in this problem setting, as we only have 50 real images for the target domain. If we where to do work in the semi-supervised, CycleGAN would output 50 images which in the best case would be indistinguishable from the 50 real images. But this approach would at best result in the same performance as training on the 50 real images, i.e. *experiment 2*.

Though CycleGAN cannot aid us in improving the quality of the synthetic images, we do still see positive effect of using the synthetic images. As shown in Figure 6, we clearly see a performance improvement when fine-tuning a pre-trained Faster R-CNN with the synthetic images in addition to the limited amount of real images. This indicates that there is enough training data to avoid overfitting. The ability to generalize well is heavily influenced by the fact that the synthetic images impose a lot of variability. We also note that fine-tuning the model works significantly better than using it as a fixed feature extractor. This is also to be expected as the synthetic images differ quite significantly from the images in the COCO data set.

Lastly, we note that training the Faster R-CNN model from scratch and fine-tuning a pre-trained Faster R-CNN on only real images, yield quite similar results. For the latter, we most likely overfit due to the training images being almost identical to the test images. Overfitting is also a concern in the model trained from scratch, due to the small amount of data. The small amount of data makes it difficult to estimate the parameters of the model which are randomly initialized. It should generalize better on other test images, since the addition of the synthetic images impose a lot variability, due to it being generated using DR.

6 Conclusion & Future work

In this report, we have tested the effects of using CycleGAN to improve the quality of the synthetic images generated using DR. To evaluate this effect, we trained Faster R-CNN on the output of CycleGAN and on the images provided to CycleGAN, specifically 99.9% synthetic images and 0.1% real images. We experimented with various transfer learning strategies, which clearly showed that fine-tuning a pre-trained Faster R-CNN achieved the best results. More concretely, we achieved 73.4% mAP and 76.5% mAR when training on the input to CycleGAN, compared to 0.0% mAP and 0.0% mAR when training on the output of CycleGAN. From this, we conclude that CycleGAN using unpaired input-output images did not to improve the quality of the synthetic images. In fact, it drastically worsened the quality.

We also conclude that fine-tuning a pre-trained Faster R-CNN, on the 99.9% synthetic images and 0.1% real images, achieved greater results compared to only using the real images. More concretely, an increase of 10.9% in mAP and 9.3% in mAR was achieved.

For future work, the effect of using CycleGAN should be explored more rigorously. One idea is to look into using simpler images, e.g. remove the background of the real images used in the target domain.

Furthermore, the effect of fine-tuning different layers and different amounts of layers should be explored. Only fine-tuning deeper layers, that is the layers that find smaller details. might mitigate overfitting further.

7 Acknowledgements

We thank our advisors Henrik Pedersen and João Marcelo Evangelista Belo for guidance throughout the project. Furthermore, we would like thank Oliver Gyldenberg Hjermitslev from the Alexandra Institute for providing the domain randomization image generator.

References

[1] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017.

[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2017.

[3] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[5] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[7] T. He, Z. Zhang, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of tricks for image classification with convolutional neural networks," *CoRR*, vol. abs/1812.01187, 2018.

[8] http://vision.stanford.edu/teaching/cs231n/, "Transfer learning, https://cs231n.github.io/transfer-learning/ (accessed 04-11-2020),"

[9] O. G. Hjermitslev, "Training object detectors with no real data using domain randomization, https://towardsdatascience.com/training-object-detectors-with-no-real-data-using-domain-randomization-1569cb3b8c6 (accessed 22-10-2020),"

[10] S. Hinterstoisser, "Linemod data set, http://campar.in.tum.de/Main/StefanHinterstoisser (accessed 28-10-2020),"

[11] info@cocodataset.org, "Coco data set, https://cocodataset.org/home (accessed 11-11-2020),"

[12] J. Jordan, "Evaluating image segmentation models, https://www.jeremyjordan.me/evaluating-image-segmentation-models/ (accessed 26-10-2020),"