Domain Adaptation of Unreal Images for Image Classification

Johan Thornström
Abstract

Deep learning has been intensively researched in computer vision tasks like image classification. Collecting and labeling images that these neural networks are trained on is labor-intensive, which is why alternative methods of collecting images are of interest. Virtual environments allow rendering images and automatic labeling, which could speed up the process of generating training data and reduce costs.

This thesis studies the problem of transfer learning in image classification when the classifier has been trained on rendered images using a game engine and tested on real images. The goal is to render images using a game engine to create a classifier that can separate images depicting people wearing civilian clothing or camouflage. The thesis also studies how domain adaptation techniques using generative adversarial networks could be used to improve the performance of the classifier. Experiments show that it is possible to generate images that can be used for training a classifier capable of separating the two classes. However, the experiments with domain adaptation were unsuccessful. It is instead recommended to improve the quality of the rendered images in terms of features used in the target domain to achieve better results.
Acknowledgments

I want to thank my supervisors David Gustafsson and Erik Valldor, at the Swedish Defence Research Agency, for their continuous support and thoughtful discussions. Furthermore, I would like to thank my examiner Per-Erik Forssén and supervisor Gustav Häger for their feedback and patience throughout this thesis.
Contents

1 Introduction .................................................. 1
  1.1 Motivation .................................................. 1
  1.2 Aim ......................................................... 2
  1.3 Methodology and research questions ...................... 2
  1.4 Delimitations ............................................... 3
  1.5 Thesis outline ............................................. 3

2 Theory & Related Work ...................................... 5
  2.1 Unreal Engine 4 ............................................. 5
    2.1.1 UnrealCV .................................................. 6
  2.2 Generative Adversarial Networks ......................... 6
    2.2.1 Training Generative Adversarial Networks ............ 7
      2.2.1.1 Loss functions ...................................... 8
      2.2.1.2 Mode collapse ...................................... 8
      2.2.1.3 Feature matching ................................... 9
      2.2.1.4 One-sided label smoothing ......................... 9
    2.2.2 Conditional Generative Adversarial Networks ........ 9
  2.3 Domain Adaptation ......................................... 10

3 Method ...................................................... 13
  3.1 Rendering images using Unreal Engine 4 ................. 13
  3.2 CycleGAN .................................................. 14
    3.2.1 Network architectures ................................ 16
    3.2.2 Training ............................................... 17
  3.3 Datasets .................................................. 18
  3.4 Object Classifier ......................................... 19

4 Results .................................................... 21
  4.0.1 Datasets ................................................ 21
  4.0.2 Images from Unreal Engine ............................ 22

5 Discussion .................................................. 25
  5.0.1 Results ................................................ 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0.1.1 Rendered Images</td>
<td>25</td>
</tr>
<tr>
<td>5.0.1.2 Domain Adaptation</td>
<td>26</td>
</tr>
<tr>
<td>5.0.2 Methodology</td>
<td>26</td>
</tr>
<tr>
<td>5.0.2.1 Rendered Images</td>
<td>26</td>
</tr>
<tr>
<td>5.0.2.2 Domain Adaptation</td>
<td>27</td>
</tr>
<tr>
<td>5.0.3 Future work</td>
<td>28</td>
</tr>
</tbody>
</table>

6 Conclusion

Bibliography
Introduction

With the progress made in computer graphics in recent years, it has emerged as an alternative to train image classifier using rendered images. Because modern deep learning approaches for image classification require large amounts of data, generating the needed data could potentially avoid expensive tasks such as data annotation.

This thesis studies the problem of how well an image classifier performs depending on the type of dataset that has been used for the training phase. Specifically, this thesis compares how an object classifier tasked at classifying real images performs depending on if the provided dataset consists of rendered images or real images. The thesis will also evaluate if it is possible to improve the performance of an image classifier that is trained on rendered images by either adding real images to the dataset or using Generative Adversarial Networks (GANs) to domain adapt the images towards real images.

The following chapter contains an overview of the problems this thesis will study and a motivation for why it is of interest, and lastly, formulate a set of research questions which this thesis aims to answer.

1.1 Motivation

With a combination of more powerful graphics processing units and Convolutional Neural Networks, it is now possible to train neural networks that outperform human-level performance on tasks such as image classification on datasets like ImageNet[9][26]. There has been rapid development within the field of computer vision, and new ways of applying CNN’s to different tasks are discovered each year. If it is possible to construct a sufficiently good image classifier that has been trained on rendered images, then it might become easier to construct
datasets of objects that are hard to acquire. We could see even faster development in tasks related to computer vision than before.

With the help of game engines, it is possible to render images with annotations at pixel-level [23]. The current generation of game engines can produce realistic images of different types of objects and environments. Previously, when training object classifiers, the performance of these classifiers would drop if the images used for training were of rendered images instead of real images [25].

Adding noise to rendered images has long been used as a method of improving the generalization capabilities of an image classifier, techniques such as motion blur or random noise were commonly used methods in an effort of making the rendered images exhibit the complexity of real images [25]. In 2014 a new way of generating annotated data using an adversarial network had demonstrated it is possible to generate data without providing parametric specifications of a probability distribution function [5]. With this method, it might be possible to filter the simulated images to potentially make them realistic enough to be used as a substitute for real images.

1.2 Aim

A sufficiently large dataset is needed in order to train a neural network for image classification that is capable of generalizing well on unseen data. Most of the publicly available datasets that are used for image classification were constructed by manually annotating each sample [12]. Constructing these datasets is both time-consuming and expensive to manually annotate every image, which makes methods for automatic annotation of images of interest. With the introduction of virtual environments such as game engines, it is possible to annotate every pixel in a frame automatically [23]. Virtual environments open up the possibility of constructing datasets of rendered images. If these images can be used to train an image classifier, then it would be possible for a company to easily acquire and construct a dataset for training their classifiers for their specific needs and to a lower cost.

1.3 Methodology and research questions

This thesis will evaluate the performance of an image classifier and how the performance will be affected depending on the type of images used for the training phase. The image classifier determines if an image is a person wearing civilian or military clothing. A dataset of images containing real people was provided for this thesis by the Swedish Defence Research Agency (FOI). The datasets of rendered images will be rendered in the Unreal Engine 4 game engine. The evaluation process will determine how different characteristics of a dataset could affect the performance. The following research questions will be answered in this thesis:
1. By training an image classifier on images rendered in Unreal Engine 4, what performance can be achieved on real images?

2. How will the size of a dataset affect the accuracy of a CNN?

3. How will the performance of the image classifier change if the dataset has been domain adapted towards real images?

4. What performance can be achieved when combining rendered and real images?

1.4 Delimitations

Because of time restrictions, this thesis will be limited by a number of factors. The images will be rendered in Unreal Engine 4.16.3, together with UnrealCV 0.3.10. The 3D-models that are used for the rendered images are provided by FOI. Domain adaptation is a broad field with multiple different approaches, and this thesis will only consider methods using GANs. A dataset containing real images was provided by FOI. The thesis was conducted during a 20 week period, and the models were trained on a computer equipped with a GeForce GTX 1080 Ti graphics card.

1.5 Thesis outline

The theory used when answering the research questions can be found in chapter 2. A more thorough explanation of the methodology and evaluation can be found in chapter 3. Chapter 4 presents the results and in chapter 5 the results are discussed in detail. The conclusion for the thesis can be found in chapter 6.
Theory & Related Work

This chapter covers the theory and related work of the areas necessary for answering the research questions. Because one of the tasks of the thesis is to generate a dataset using a game engine, section 2.1 contains theory related to the virtual environments that are used. In section 2.2 has the theory and related work for Generative Adversarial Networks, how they are trained, and common problems that occur when training this type of model. Section 2.3 presents how domain adaptation can be used in combination with GANs. This thesis assumes the reader has previous knowledge of concepts related to the area of computer vision and deep learning. This thesis assumes that the reader has previous knowledge of basic deep learning concepts. If the reader wants a summary of deep learning concepts and its application on computer vision, then there are good sources online [6].

2.1 Unreal Engine 4

Unreal Engine 4 (UE4) is a game engine that is widely used among game developers today. The engine comes with a complete suite of development tools such as Blueprints, which allows rapid development without writing a single line of code. Blueprints are a type of node-based visual scripting system if the developer does not want to implement changes to their virtual world through code. UE4 is source-available that allows the developers with knowledge in C++ a high degree of freedom at modifying the source code. UE4 has a marketplace that gives the users of the engine a platform at sharing or selling development tools, environments, 3D-models, textures, and more.
2.1.1 UnrealCV

UnrealCV is an open-source project for UE4 created by Qiu et al. [21] that extends UE4 by providing a set of in-engine commands to interact with the virtual world. The UnrealCV plugin consists of two components, the server and the client, as can be seen in figure 2.1. The server is embedded into a game during compilation and makes use of the API of Unreal Engine to access information of the virtual world. While it is possible to compute things such as ground truth from the internal data from the engine, it is not an available feature in UE4. UnrealCV has a feature called object masking, where all objects in a scene have a designated color. In figure 3.2 (a) show how a virtual environment in UE4 could be seen and figure 3.2 (b) visualizes how the object masking works for the objects seen on the screen. Together with the vast amounts of available resources from the Marketplace and Internet, it is possible to build virtual environments of various types of settings. The quality of the freely available textures varies, from low polygon objects to high-resolution photogrammetry objects. The UnrealCV plugin has a set of client commands to interact with the Unreal Engine Environment. Through the client, different types of images can be rendered in the scene, including depth, ground truth, normal mapping, and RGB. The camera trajectory can be changed through client requests to the server, and with a set of built-in functions, it is possible to get object information of every pixel in the image. The set of built-in functions also include making segmentation masks around objects in the scene and commands through the Unreal Engine terminal.

![Figure 2.1: An overview of the architecture for UnrealCV. The UnrealCV Server is embedded in the running Unreal Engine environment as a plugin. The UnrealCV client is then able to communicate with the running environment through the Unreal Engine Server](image)

2.2 Generative Adversarial Networks

In 2014 Goodfellow et al. [5] presented a framework for estimating generative models. With an adversarial process, two models are alternatingly trained. One of the models is called the generator $G$ that generates synthetic data, which is then sent to the other model. That model is called the discriminator $D$, and it estimates the probability of its input coming from the training data distribution or the $G$ distribution. With a minimax procedure, $G$ tries to maximize the probability of $D$ making a mistake. An analogy used by the authors of the paper is that you can imagine the two models as police and counterfeiter. Where $G$ could
2.2 Generative Adversarial Networks

be seen as the counterfeiter with the goal is to produce a fake currency that is indistinguishable from real currency. While $D$ is the police with the goal to be able to differentiate between real and fake currency. As they both improve their respective methods, this process continues until the counterfeiter is capable of producing a fake currency that is indistinguishable from real currency.

This adversarial framework explored the case when the generator $G$ produces samples by passing random noise through a multilayer perceptron. A multilayer perceptron is a class of feedforward neural networks. The discriminator $D$ is also a multilayer perceptron which input is the size of the samples that $G$ generates and outputs a single scalar. In order to learn the generator distribution $P_g$ over data $x$, they define a prior for the input noise variable $P_z(z)$. Then they represent a mapping to data space as $G(z; \theta_g)$ where $G$ is a differentiable function with parameters $\theta_g$. Also defined is the second differentiable function $D(x; \theta_d)$, where $D(x)$ represents the probability of $x$ coming from the training data distribution rather than $P_g$. Then they are both trained simultaneously where $D$ maximizes the probability of making the correct classification for samples from both data distributions while $G$ minimizes $\log(1 - D(G(z)))$. The procedure can be seen in the Algorithm 2.1.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - G(D(z)))]. \quad (2.1)$$

2.2.1 Training Generative Adversarial Networks

Training GANs requires finding a Nash equilibrium in a two-player non-cooperative game with continuous, high dimensional parameters. Most GANs are usually trained using gradient descent techniques. As the generator and discriminator
both want to lower their cost functions, it might cause the algorithm to be incapable of converging [7]. The cost function for the generator is \( J^{(G)}(\theta^{(D)}, \theta^{(G)}) \) and for the discriminator is \( J^{(D)}(\theta^{(D)}, \theta^{(G)}) \). The Nash equilibrium is a point \((\theta^{(D)}, \theta^{(G)})\) that occurs when \( J^{(D)} \) and is at minimum with respect to \( \theta^{(D)} \) and \( J^{(G)} \) is at minimum with respect to \( \theta^{(G)} \). Using gradient descent-based methods to find a solution to this problem seems like an intuitive choice. A problem when the cost functions are non-convex is that a modification to \( \theta^{(G)} \) that would reduce \( J^{(G)} \) could also increase \( J^{(D)} \) and the other way around.

2.2.1.1 Loss functions

As the generator’s goal is to generate samples which distribution is indistinguishable from the training data distribution. A problem that can occur is that when measuring the distance between the generated distribution and the training data distribution, is that their gradients might point in almost random directions if there is not enough overlap between the two distributions [1]. Arjovsky et al. [2] introduced an alternative way of training adversarial networks by introducing a different loss function that they argue has better theoretical properties. They argue that there can be training difficulties when the GAN minimizes a divergent loss function that could potentially not be continuous with respect to the parameters of the generator. The loss function Arjovsky et al. [2] propose to minimize instead is the Earth-Movers (also known as Wasserstein-1) distance \( W(P, Q) \) which is the measure of the distance between the two distributions \( P \) and \( Q \). Where \( \Pi(P, Q) \) is the set of all distributions \( \gamma(x_p, x_Q) \) over \( P \) and \( Q \). The loss function for the Earth-Movers distance can be seen in Algorithm 2.2. An important change they make when training is that the discriminator (called the critic in that work) is updated at a higher rate than the generator. Which results in a more computationally demanding version of the original GAN.

\[
W(P, Q) = \inf_{\gamma \in \Pi(P, Q)} \mathbb{E}_{(x_p, x_Q) \sim \gamma} \{\|x_p - x_Q\|\}
\]  

(2.2)

2.2.1.2 Mode collapse

Another problem that might happen when training GANs is known as mode collapse. It happens when the generator parameter settings collapse and always emits to the same points. When this occurs, the gradient of the discriminator point in a similar direction for multiple points. Because the discriminator processes each sample independently, and there is no mechanism to encourage outputs of the generator to have variance. Then the gradient of the discriminator pushes the single point emitted from the generator around and never converges to a distribution with the right amount of entropy. Radford et al. [22] introduced the deep convolutional generative adversarial network (DCGANs) in which they argued that batch normalization could improve the optimization of a neural network. While batch normalization helps with stabilizing the training of a GAN, it changes the objective of the discriminator. Instead of mapping a single input to a single output, it matches an entire batch of inputs to a batch of outputs [27].
There have been multiple suggestions for different approaches to batch normalization. Such as virtual batch normalization, which was presented in a paper by Salimans et al. [27], where instead of each sample is normalized once at the start of training based on the statistics collected on a reference batch. Another suggestion made in [27] is to prevent mode collapse was minibatch discrimination that adds another input to the discriminator, which is the distance between the sample and the other samples in the minibatch. Karras et al. [15] had impressive results with generating images with high resolution, they suggest that constraining signal magnitudes and competition avoids the same problems batch normalization face but without learnable parameters. This is done by calculating the standard deviation for each feature in every spatial location over the minibatch. With the average value of these features, they can concatenate the value overall spatial locations and over the minibatch yielding one additional constant feature map that is inserted in the discriminator. Gulrajani et al. [8] suggested removing batch normalization from the discriminator and instead recommend using layer normalization as a replacement [4].

### 2.2.1.3 Feature matching

Salimans et al. [27] introduced more ways of stabilizing the training phase for GANs, such as feature matching. The way it counteracts instability of GANs is by introducing another objective for the generator that prevents it from overtraining on the discriminator. Instead of only maximizing the output of the discriminator, it also has to generate data that matches the statistics of the real data. The generator has to match the expected feature activation of an intermediate layer of the discriminator. This is intuitive because, as the discriminator is trained, it learns the features that separate the generated data and real data.

### 2.2.1.4 One-sided label smoothing

Another problem that can occur during adversarial training is that the discriminator uses a few features for its classification boundary. It could result in the generator getting greedy, seeking to exploit those features. One-sided label smoothing changes the target for the discriminator to 0.9 instead of 1. Sønderby et al. [29] and Arjovsky et al. [1] explored this idea by adding noise to the samples before feeding them into the discriminator. Sønderby et al. [29] argues that bringing the two distributions closer to each other prevents the discriminator from learning boundaries that completely separates them.

### 2.2.2 Conditional Generative Adversarial Networks

A problem with unconditioned generative models is that there is no way of controlling the data that is generated. Mirza et al. [20] presented a way of extending generative adversarial networks to a conditional model by adding additional input $y$ to the generator and discriminator. This additional input could be anything such as class labels, text, or images. This allows the model to generate data that is not only similar to the data distribution but is also from a specific class from...
the data distribution. This is done by combining the input noise $P_z(z)$ and $y$ in
the generator as a hidden representation. Then in the discriminator, $x$ and $y$ is
the input for the discriminative function, which can be seen in Algorithm 2.3.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x \mid y)] + \mathbb{E}_{z \sim p_z(z)} [1 - G(D(z \mid y))] \quad (2.3)$$

### 2.3 Domain Adaptation

Domain adaptation is the task of learning and minimizing the domain shift that
happens when the model is trained on a dataset and then is tested on data from
a different unseen environment. Without learning this shift in the domain, a
common problem that can occur is known as dataset bias. It happens when a
model trained on a specific dataset might not generalize well if it is then tested
on a different dataset [31]. Previous research shows that even slight differences
in the domains can significantly hurt the model’s performance [32].

Isola et al. [14] presented a general-purpose model for translating an image
from one domain to another that is known as pix2pix. This model requires match-
ing pairs of data from two domains and is capable of colorizing images, construct-
ing images from label maps, among other tasks. The principle of the model is that
the discriminator is exposed to pairs of images and then tries to separate images
from the training data distribution from the generated data distribution. The
generator generates its pair by being provided with an image from one of the
training data domains and generates an image of the other domain. They tested
two different methods for adding noise to the generator. One of the methods for
adding noise was by injecting a noise vector with the normal distribution that is
transformed to layer in the generator, and the other method used dropout. None
of those methods seemed to be working as the generated output of low variance.
They also tested different architectures for the generator and had promising re-
sults with the U-Net architecture [24].

The method presented by Isola et al. [14] has a restriction in which it requires
matching pairs of labeled data from two domains. Zhu et al. [35] introduced the
Cycle-Consistent Generative Adversarial Network (CycleGAN) with a simple idea.
Instead of the loss function trying to reduce the distance between the distribu-
tions of $G(x) = \hat{x}$ and the data distribution, by introducing a cycle-consistency
loss it reduces the distance of $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ as can be seen in equation
2.4.

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)} [\|F(G(y)) - y\|_1] \quad (2.4)$$

A similar approach has also been proposed by Kim et al. [16] with the Disco-
GAN model. The goal is to learn mapping functions between two distributions
that work in both directions. One of the main problems with regular GAN ap-
proaches to domain adaptation is that there is no restriction on the generator
for manipulation of the structure in the original image. By adding a type of
cycle-consistent loss, function forces the generator to avoid making structural changes to the image so it can reconstruct the original image. This type of cycle-consistency makes it possible to transfer an image from one domain to another, i.e., transforming a zebra to a horse without having matching pairs. However, it is not possible to ensure that the semantic information from the original image is preserved when translated to the other domain. Because while the cycle-consistency preserves the structural aspects of an image, it does not necessarily keep the color composition of the original image when domain translated. The loss function also takes into account the identity loss in order to preserve the color composition in the images [35][30]. The idea of cycle-consistency enables style transfers from one domain to another. This area is further researched by He et al. [11] as their method for color transfer is spatial variant and globally coherent. Instead of mapping the distances between two distributions, the method compares an image with one or multiple references.

Hoffman et al. [13] presented a method for preserving semantic information in the images known as Cycle-Consistent Adversarial Domain Adaptation (CyCADA). Because having only a cycle-consistent loss function does not prevent the semantic information in the original image to change. It is common for a generator in a CycleGAN framework to e.g. changing the color of the ground and turn the asphalt into the grass. To prevent it from happening, they enforce semantic consistency by adding a pre-trained classifier on the target domain, which punishes semantic changes when translating from one domain to another.

Shrivastava et al. [28] explored the idea of refining rendered images to make them more realistic. With the recent progress made in the areas of both graphics and generative methods. Is it possible to improve the realism of rendered images using unlabeled real data? They developed a method called Simulated + Unsupervised learning (S+U-learning) that uses adversarial training with some similarities to a GAN. With the help of a sizeable unlabeled dataset, it is possible to improve a dataset of simulated images and preserve the annotation. They extended the adversarial loss of a GAN by adding a self-regularization loss that minimizes per-pixel differences. The idea is to preserve information from the rendered images such as gaze direction in eyes or joint position in hand gesture estimation. They trained a CNN to predict gaze direction using the MPIIGaze and UnityEyes datasets [34][33]. Their model outperformed the state-of-the-art models by refining the images from the same dataset.

Atapour-Abarghouei and Breckon [3] tried to solve the problem of domain shift that occurs when monocular depth estimators are trained using only rendered data and are used on real-world data. Their solution is using domain adaptation to shift the rendered data distribution towards real-world data distribution. Their approach is two models trained simultaneously where one model is a depth estimator trained on rendered data. The second model tries to reduce the domain divergence between the rendered data distribution and the real image data distribution, and this is done by introducing a cycle-consistent loss function similar to previous research done in image-to-image translations [35].
This chapter describes the methods used for finding the answers to the research questions. Section 3.1 describes the methods that will be used for rendering images and section 3.2 explains how the rendered images will be domain adapted. Then in section 3.3 describes the datasets will be constructed and section 3.4 presents the method used for evaluating the datasets.

### 3.1 Rendering images using Unreal Engine 4

UnrealCV provides a set of functions that interact with the virtual world. However, it is limited when it comes to manipulating objects within the virtual world, such as moving objects or changing textures. Each virtual world has a Blueprint in which the developer can create functions that interact with the world. These functions can then be connected to something called custom event within UE4, which is callable through the terminal. This allows the construction of an API that extends the UnrealCV-plugin capabilities of interacting with the virtual world, which can be seen in figure 3.1.

FOI provided a 3D-model of a human and a couple of virtual environments that will be used to render the images that will be used to train the object classifiers. The 3D-model has a set of textures of 21 male and 21 female civilians with different animations of various tasks such as walking or running. There are also 47 textures of males with military clothing with animations such as firing a rifle or kneeling. In the Blueprint of the provided virtual worlds, functions for switching the textures and animations for the 3D-model will be implemented. And it should be possible to change the position or rotation of the 3D-model. In UE4, there are data tables where the information can be saved about what textures to use or what position or rotation an object should be in. These data tables allow
Figure 3.1: An overview of how the UnrealCV-client can interact with a running environment running in Unreal Engine. The UnrealCV Client can directly interact with Unreal Engine by making camera adjustments. However, it cannot directly interact with objects in the environment and must give commands through the Blueprints for the running environment.

UnrealCV to manipulate the virtual world and know what type of texture the 3D-model is using. With this information, UnrealCV can, for each scene, save an image and calculate bounding boxes of the object, then the image can be cropped and store it with a corresponding label. As an example, if the objective was to save an image of the 3D-model in figure 3.2(a), then it would look like figure 3.2(c).

Previous attempts at constructing rendered datasets have shown that the images rendered should be similar to the real images, not in terms of image quality but terms of features used during the detector training [25]. As the goal of the thesis is to evaluate different object classifiers on if an image contains human with military or civilian clothing, the images should contain similar features as the real images such as environments or poses. The virtual environments that will be used were found on the Unreal Marketplace and contained mountainous landscapes and urban areas. The 3D-model will be placed on an arbitrary spot in the environment, and four different arbitrary camera angles will be selected. The UnrealCV client will then iterate through different combinations of animations and textures for that scene. The 3D-model has five animations without weapons, four with animations with pistols, and five with a rifle. When it performs e.g. an animation with a rifle, then one will be placed in its hand. When the client iterates through the different camera angles, for every military texture, select one random animation without weapons, one random animation where it uses a pistol, and two random when it uses a rifle. When the model has a civilian texture, then it iterates through all the five animations without weapons. The images will be cut at the bounding boxes of the 3D-model and saved, the label for the image will be saved in a JSON-file.

3.2 CycleGAN

CycleGAN is a type of GAN that has been able to map the domain shift between two domains and produce visually compelling results [35]. CycleGAN achieves this by having a cycle-consistent loss function that constrains the generator from
Figure 3.2: Method pipeline for how a scene in Unreal Engine 4 can be cropped using UnrealCV. (a) An image is saved that selected scene with an object of interest. (b) With object masking, a bounding box for the object can be calculated and saved to a JSON-file. (c) With the image from (a) and the bounding box from (b) a new image can be created that only contains the object of interest.
Figure 3.3: An overview of the mapping functions of a CycleGAN. In (a) shows the two mapping functions $G : A \rightarrow B$ and $F : B \rightarrow A$ and their associated discriminators $D_B$ and $D_A$. $D_B$ encourages $G$ to translate $A$ into something that is indistinguishable from domain $B$ as can be seen in (b). (c) The same idea is applied but $D_A$ encourages $F$ to turn $B$ into something indistinguishable from $A$.

making structural changes to an image when it translates to the other domain. As the rendered images that will be rendered in UE4 should have the same structural properties as a real image, using a GAN with a cycle-consistent loss function would be ideal for answering the research questions. CycleGAN achieves this image-to-image translation between two domains by having two GANs as one of the networks maps the domain shift between domain $A \rightarrow B$. The other network map the domain shift from $B \rightarrow A$. In this case, one of the generators will take an image from the rendered image dataset as input and generate an image. One of the discriminators will evaluate whether the images it receives as comes from an image generated by the first generator or if it comes from the real image dataset. The second generator and discriminator will do the same thing but with the datasets flipped. The generator takes an image from the real image dataset as input and generates a new image. The second discriminator will take images from this generator and images from the rendered image dataset and try to differentiate between them. An overview of the CycleGAN architecture can be seen in figure 3.3.

3.2.1 Network architectures

One of the common problems with deep neural networks is training difficulties, such as the vanishing gradient problem. One of the common approaches that have had promising results is the residual network by He et al. [10]. By adding "shortcut connections" which are connections that are skipping layers, the problems with vanishing gradients when using backpropagation are reduced. These residual networks usually have their skipping layers divided into residual blocks, which can be seen in figure 3.4. The generators will use a residual network that can be seen in Table 3.1. The discriminators will use a PatchGAN architecture [14][18] and it can be seen in table 3.2.
3.2 CycleGAN

During training, the loss function that the CycleGAN minimizes consists of two adversarial loss functions, as proposed by Goodfellow et al. [5]. Equation 2.1 is the loss function, and it minimizes the mapping function for $G: X \rightarrow Y$ and its discriminator $D_y$ as well as the mapping function for $F: Y \rightarrow X$ and the $D_x$ discriminator. The generator $G$ tries to generate images $G(x)$ that is similar to the domain of $Y$ or in this case, generate images that are similar to the domain of real images. While the generator $F(y)$ tries to generate images that are similar to the domain of the images rendered in UE4. The two mapping functions combined becomes the equation 2.4 and the total loss function becomes equation 3.1.

<table>
<thead>
<tr>
<th>Block</th>
<th>Filter</th>
<th>Filter size</th>
<th>Stride</th>
<th>Padding</th>
<th>Inst. Norm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv+LeakyReLU</td>
<td>64</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>64</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>9x Residual Blocks</td>
<td>128</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>128</td>
<td>3</td>
<td>0.5</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>64</td>
<td>3</td>
<td>0.5</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+TanH</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

*Table 3.1:* An overview of the CycleGAN generators architecture used in this thesis. The different show the configurations and dimensions used for all the convolutional layers.
Table 3.2: An overview of the PatchGAN discriminators architecture used in the CycleGAN model.

<table>
<thead>
<tr>
<th>Block</th>
<th>Filter</th>
<th>Filter size</th>
<th>Stride</th>
<th>Inst. Norm.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv+LeakyReLU</td>
<td>64</td>
<td>4</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>128</td>
<td>4</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>256</td>
<td>4</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv+LeakyReLU</td>
<td>512</td>
<td>4</td>
<td>2</td>
<td>Yes</td>
</tr>
<tr>
<td>Conv</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3.3: The PyTorch parameters used in the CycleGAN implementation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda$</td>
<td>10</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.002</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>0.5</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>0.999</td>
</tr>
<tr>
<td>Batch size</td>
<td>1</td>
</tr>
</tbody>
</table>

$L(G, F, X, Y) = L_{GAN}(G, D_y, X, Y) + L_{GAN}(F, D_x, Y, X) + \lambda L_{cyc}(G, F)$

The model will be trained for 200 epochs, and the networks will be trained to minimize the least square error. During training, the weights will be updated using Adam optimizer [17]. Compared to regular stochastic gradient descent, which has a single learning rate, Adam optimizer allows for an adaptive learning rate. Adam optimizer can do this by regulating the decay of the learning rate. This decay will start after 100 epochs, and the parameters that Adam optimizer uses can be seen in Table 3.3.

### 3.3 Datasets

The thesis will evaluate the performance of an object classifier that has been trained on different datasets. Four different datasets will be constructed to answer research questions 1 and 2. One dataset will use a third of the generated images in UE4 using the method that is presented in section 3.1. Then two datasets will use the remaining two-thirds of the images, one of them will use real images as validation data, and the other will use the remaining third of the generated images as validation data. Then a fourth dataset that uses all the generated images from UE4 that dataset 1 and 2 uses.

The real images that are used for this thesis come from two sources. The images of people wearing civilian clothing come from the COCO dataset that has
been cropped from the provided object segmentation that the dataset provides [19]. Half the dataset contains people wearing camouflage, and it has been taken from public sources such as Twitter. Regarding research question 3, how the performance will be affected by domain adapting the images generated in UE4 will require another dataset to be constructed. The domain adapted images generated with the method presented in section 3.2 will be used as training data while the real images will be used for validation and testing. For the last research question, one dataset that combines UE4 images and real images in the training data will be made. Finally, a dataset using only real images for training, validation, and the test will be used as a control.

### 3.4 Object Classifier

The datasets will be evaluated using the same architecture for the object classifier. The selected model is a residual learning network presented by He et al. [10]. It works by adding a shortcut connection to a CNN where the input from one layer is added to a layer further down the CNN. Each of these skips is called a residual block and can be seen in figure 3.4. The images that are fed through the CNN have been padded and then scaled to a 300x300 pixel size. The model that will be used is the default PyTorch ResNet50 model that is trained for 200 epochs with a learning rate of 0.001. After every epoch, the model is saved and tested on the validation data. The model with the highest accuracy on the validation data will be selected for the final evaluation. This process is repeated for every dataset, and the best models for every dataset are then tested on the test data, and the results are presented in chapter 4.
In this chapter, the results of the experiments done during this thesis are presented. No metrics for the CycleGAN are presented, only samples from the final generator.

### 4.0.1 Datasets

In table 4.1 shows the final datasets that were constructed using the methods presented in chapter 3 and the accuracy they achieved on the test images. The first four datasets that are only using rendered images have similar test accuracy. Dataset 6 that only uses domain adapted images for training data achieved the lowest accuracy. The best results were achieved by dataset 5 and 7 that had real images in the training data. Figure 4.1 shows a graph of dataset 2 and 3 for their first 100 epochs.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset 1</td>
<td>5063 Rendered</td>
<td>6126 Real</td>
<td>2044 Real</td>
<td>0.74609</td>
</tr>
<tr>
<td>Dataset 2</td>
<td>10125 Rendered</td>
<td>5063 Rendered</td>
<td>2044 Real</td>
<td>0.76174</td>
</tr>
<tr>
<td>Dataset 3</td>
<td>10125 Rendered</td>
<td>6126 Real</td>
<td>2044 Real</td>
<td>0.77544</td>
</tr>
<tr>
<td>Dataset 4</td>
<td>15188 Rendered</td>
<td>6126 Real</td>
<td>2044 Real</td>
<td>0.77348</td>
</tr>
<tr>
<td>Dataset 5</td>
<td>5063 Rendered &amp; 4084 Real</td>
<td>2042 Real</td>
<td>2044 Real</td>
<td>0.90705</td>
</tr>
<tr>
<td>Dataset 6</td>
<td>6599 Domain</td>
<td>6126 Real</td>
<td>2044 Real</td>
<td>0.66977</td>
</tr>
<tr>
<td>Dataset 7</td>
<td>4084 Real</td>
<td>2042 Real</td>
<td>2044 Real</td>
<td><strong>0.91438</strong></td>
</tr>
</tbody>
</table>

*Table 4.1: Accuracy for the datasets used to train the Object Classifier. The columns describe the type and amount of images used for the training, validation, and testing phase.*
Figure 4.1: Validation accuracy of dataset 2(red) and 3(blue) throughout 100 epochs.

4.0.2 Images from Unreal Engine

This section presents the resulting images from the rendered dataset, and the domain adapted dataset. Figure 4.2 shows images of the model of a person with military textures taken in the Unreal Engine and the final domain adaptation of those images. As can be seen in the images, in most cases, the camouflage pattern becomes less distinct. When the model of a person is equipped with a weapon, the bounding box includes the weapon, which in some angles, can make the model only cover a small area of the image. Figure 4.3 shows the images of the civilian textures and their corresponding domain adaptation.
Figure 4.2: The images contain six pairs that show the final domain adaptation of the model with military clothing. The left image is from the Unreal Engine 4 dataset and the right image is the same image domain adapted towards real images.
Figure 4.3: Six pairs of domain adaptation of the civilian models. The left image is from Unreal Engine 4 and the right image is from the final domain adaptation.
In this chapter, the methodology presented in chapter 3 and the results from chapter 4 of the thesis are discussed. Furthermore, a discussion about possible areas that future work could expand on.

5.0.1 Results

One of the assumptions that were made before this thesis was conducted was that it would require a large dataset of real images to create a model that would be capable of generalizing well enough to be capable of separating camouflage clothing from regular clothes. However, as can be seen in table 4.1, dataset seven that only used 8142 real images, was sufficient for creating a model that quickly could achieve accuracy over 90%.

5.0.1.1 Rendered Images

A few samples from the generated images are shown in figure 4.2 and figure 4.3, show similar features to what can be expected in a real image. As the results in table 4.1 indicate, the difference in accuracy for the datasets that only use rendered images are marginal. The ResNet50-model quickly converged with the provided datasets, which could mean that the generated images did not have a high degree of variation. That claim could also be drawn from the graph presented in figure 4.1. Dataset 2 has a set of rendered images for validation, while dataset 3 has real images as validation data. The model was faster to converge when using rendered images for validation data compared to real images.

Previous attempts at constructing rendered datasets have shown that the images rendered should be similar to the real images, not in terms of image quality but terms of features used during the detector training[25]. In this case, the
number of features that are used to separate civilian clothing from camouflage is insufficient as the datasets using only generated images had an accuracy of 77%. Dataset 5 that used a combination of real and rendered images had a slightly worse performance than only using real images, but the rendered images are likely the reason the model performed worse than just using real images.

5.0.1.2 Domain Adaptation

The results of the domain adaptation were surprising as it was significantly worse than the rendered images. One of the theories of why is that while CycleGAN does not make any structural changes to the images, it does make changes to colors and patterns of materials. The camouflage patterns change, as can be seen in the images from figure 4.2 and figure 4.3. Instead of having multiple types of camouflage patterns on the clothing, they are instead domain adapted towards a single one. The domain adapted images could improve object detection as previous research that uses domain adaptation for object detection might indicate. However, in this case of binary object classification with these types of labels, it does not improve the results.

As artifacts are still visible most of the resulting images, the loss function for the CycleGAN could be improved. Shrivastava et al. [28] had something called the realism loss function, which enforces the generator to not make structural changes from the original rendered image. This is achieved by discouraging per-pixel changes by minimizing a mapping from image space to a feature space, where the feature map could be mean of color channels, image derivatives, or a neural network.

5.0.2 Methodology

One of the main hurdles for this thesis was the time constraint on doing the actual implementation. As the thesis had multiple steps to complete. First to create a method for rendering images using Unreal Engine 4 and domain adapt those images. Finally evaluate the performance of an image classifier that had been trained on the different datasets. As the main focus of the thesis was to make a pipeline for generating images, limited time was spent on tuning parameters of CycleGAN and the object classifier, which means the results could have been improved with additional tuning.

5.0.2.1 Rendered Images

A shortcut that was taken to speed up the process of rendering images was to avoid using animations and also use fixed poses. Avoiding animations was done as a considerable amount of time was spent on making sure the object masking worked properly when the 3D-model interacted in the different environments. It is hard to define what separates a bad dataset from a decent one. The world is complex, and taking pictures generates quite a bit of noise, such as the quality of the camera, light intensity, poses. The images generated through UE4 for these
datasets were static in several aspects. The camera angles were fixed, and therefore multiple images in terms of background were very similar. Fixed camera angles was a way to ensure that the 3D-model always was placed right in front of the camera. The main reason the camera was fixed was to save time by rendering the environment to ensure that the camouflage patterns on the 3D-model were visible. The intensity and direction of lighting remained constant as well for every virtual environment due to the significant time it takes for the engine to rebuild the lighting.

The 3D-models themselves had a shortcoming in terms of customizability as there was no feasible way of changing the textures of the model. As a result, the clothing and hair combinations remained constant for all the different textures. The image classifier will then have difficulties if it gets an image as input with a person wearing a different colored shirt, or an image of a posing soldier with a different type of camouflage pattern. The results indicate that there is not enough variance in the images like the large gap in performance when real images are present in the training data or not. None of the 47 textures with camouflage were of a female who would cause female features to be correlated with civilian clothing.

As can be seen in the images from figure 4.2 and figure 4.3, is that the bounding boxes take the weapons into account, which in some cases causes the image to have the 3D-model only covering a small area of the image. While a weapon might be more frequent in an image of someone wearing camouflage, it does not indicate what type of clothing a person has in a real image. An object classifier should separate the person and weapon instead of reducing the area that the object does not cover. The graph that can be seen in figure 4.1 presents the results that compare the performance of the same set of training images but have different sets of images for validation. As can be seen in the graph, the model quickly converged when using rendered images as validation data, which indicates that the dataset does not have a high degree of variation, which results in a poor model. When using the real images as validation data, the model took a long time to converge and, not surprisingly, surpassed the rendered validation dataset. Relatively shorter converging time indicates that there is room for improvement in constructing a method for rendering images with a higher degree of variation.

5.0.2.2 Domain Adaptation

As previously mentioned, there was limited time spent on tuning parameters, and thus, the results of the first iteration of using CycleGAN was the one that was used for the final testing. In hindsight, when knowing that the domain adapted images performed worse than not using them at all, a different approach at domain adaptation would be recommended. Instead of using residual blocks, other attempts at domain adaptation use a U-Net architecture, which was introduced by Ronneberger et al. [24]. However, with the images generated in this thesis, it is likely that a different architecture or parameter tuning would not improve the results until a dataset with high variance is used.
5.0.3 Future work

One area that could be further researched is how diversity of the rendered images affect the performance of an image classifier. One could compare two datasets where one of the datasets has larger variation in terms of features used compared to the other dataset. There are multiple ways of creating a dataset with a higher degree of variation compared to this thesis. Such as instead of using a fixed camera positions and 3d-model poses, one could implement a tracking shot that pans around the 3d-model and takes images at different intervals while does an animation. Not only does this render images from more angles, but the background become less static. The 3d-model could also move through the environment and the camera follows it. The background in the images would then change as the 3d-model moves from one point to another. Both of these options are available in UE4. Also introducing different types of noise, lighting or if the object is partially hidden, could create a classification model that is better at generalizing. One thing that is hard to replicate is that images that are taken with a camera change a lot in terms of photo quality. Therefore methods for introducing noise in terms of image quality might be of interest.

As for domain adaptation for the task of separating images of civilian and military clothing, there might be improvements in the quality of the results if the two labels are separated. Camouflage patterns are distinct, and many of the domain adapted images lost their specific patterns as a result, which can be seen in figure 4.2 and figure 4.3. If the domain adaptation is instead separated between the two labels, then a model could be created that is good at preserving patterns on the clothing. Another way it could be improved is if a different loss function is introduced that ensures that the pattern and color histogram remains the same. Testing different types of architectures could provide interesting results for image-to-image translation like the U-Net architecture [24].
This thesis has studied the computer vision problem of object classification using rendered images. The classification has been done in the domain of determining if people are wearing either camouflage or civilian clothing, and what accuracy could be achieved if the classification model is tested on real images. The thesis also studied how Generative Adversarial Networks can be applied to perform domain adaptation for this specific task. The results show that it is possible to create datasets of rendered images with Unreal Engine 4 that can train an image classifier capable of separating images portraying people wearing either civilian and military clothing. Experiments using CycleGAN for domain adaptation of rendered images towards real images were unsuccessful. Further research is proposed to areas such as evaluating how datasets containing a higher degree of variation in terms of camera angles and animation could affect performance on an object classifier.


