Generating Camouflage using Generative Adversarial Networks

ALEX RUNDLE, University of Exeter, UK

This report covers the design, implementation and critical evaluation of the application of Generative Adversarial Networks to Camouflage. Camouflage intends to form a cognitive illusion to hide an object within a background or scene, and as such the ability to generate effective camouflage is highly sought after. To this end, this project intends to explore the effectiveness of four types of Generative Adversarial Network (GAN) when applied to camouflage generation. The previously undertaken Literature Review, project specification and evaluation criteria are first summarised, followed by a review of the design process and a high-level system overview. The implementation of each Generative Adversarial Network is covered in detail, followed by a comprehensive analysis of experiment results including analysis of loss values over the course of training.

Evaluation of two Image Similarity metrics suggests that the original GAN and Deep Convolutional GAN produce images that are more similar to the original dataset than that of the Wasserstein GAN. However, human observational tests show that the Wasserstein GAN outperforms all other implementations and that it's generated camouflage can successfully blend with the background to the point that a human observer cannot detect it with certainty. Future avenues of research include the use of Image Inpainting to better blend camouflage with background images, as well as exploring generation when applied to other, more complex datasets.

I certify that all material in this dissertation which is not my own work has been identified and correctly credited:

Signed: Alex Rundle

1 INTRODUCTION

Camouflage is based on the idea of forming a cognitive illusion in which an object projects the scene behind it in order to reduce it's own saliency (how prominent it is against the background) [1]. Given this, camouflage is often used in military applications to effectively hide infantry and vehicles from the enemy by blending in with the landscape around them and, as such, the ability to generate effective camouflage is of great interest to researchers.

Furthermore, the Generative Adversarial Network is well-researched and is shown to provide excellent image generation capabilities when compared with previous generation techniques [2]. Combining the knowledge of implicit density generative models, which do not directly estimate data distributions but instead define stochastic procedures for the generation of data [3], with the knowledge of adversarial training techniques from game theory, the Generative Adversarial Network is able to produce high quality images which better match the given data distribution.

Consequently, this report details a project in which several Generative Adversarial Networks are implemented and trained, with the ultimate goal of generating camouflage for a Forest image dataset.

This project aims to explore the effectiveness of multiple Generative Adversarial Network techniques when applied to the generation of camouflage, as well as the evaluation measures that exist to help evaluate the efficacy of said camouflage. This paper consists of a comprehensive report, starting with a brief summary of a prior Literature Review and followed by the specification for this project, under which I have worked. Following this, the Design section covers the technologies that will be used over the course of the project, the dataset to be used for training and testing and a high-level system overview. Next, the Development section explains the actions taken at each stage in development and the functions of each aspect of the system, starting with processing of the dataset and subsequently explaining the implementation of each technique. Further from this, presented are the process and results of training, explaining and analyzing the losses and visual results of each implementation. Analysis of both image similarity measures and the human observation test follows this, alongside evaluation of the project as a whole and the extent to which each functional and non-functional requirements has been met. Finally, this paper concludes with closing remarks and research avenues that could further expand and follow this project.

2 SUMMARY OF LITERATURE REVIEW

Firstly, I presented the areas of Generative and Adversarial techniques respectively. Generative techniques are supervised learning techniques focused on the sampling of new data formed from estimations of a data model [3]. Adversarial techniques, on the other hand, use knowledge from game theory to create a set of networks which compete against one another during the training process [4].

Following this, I presented how Goodfellow et al. [2] combine knowledge of Generative and Adversarial techniques to formulate the original Generative Adversarial Network (hereinafter GAN). The basis of their model is the Nash equilibrium where, in an n-player non-cooperative game, each player's strategy is optimal against the other players and that no player can increase their own payoff, even when changing their own strategy [5]. The GAN consists of a 2-player non-cooperative game, in which the generator network $\mathcal{G}$ aims to correctly learn the distribution of a given dataset, whilst the discriminator $\mathcal{D}$ aims to correctly distinguish real data x from generated data $\mathcal{G}(z)$ [2]; both networks are represented

by multi-layer perceptrons. The generator $\mathcal{G}$ maps a noise vector z to an image x and the discriminator $\mathcal{D}$ maps an image x to a binary classifier [2].

During the training of the GAN, $\mathcal{D}$ is trained to maximise the probability of assigning the correct class labels to both the real and generated data samples. Simultaneously, $\mathcal{G}$ is trained to minimise the probability that $\mathcal{D}$ correctly labels the generated data $\mathcal{G}(z)$. The value function seen in Equation (1), is a measure of how correctly the real data and generated data is classified:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{x \sim p_{data}(x)} [\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - \mathcal{D}(\mathcal{G}(z))] \tag{1}$$

I further presented the ways in which researchers expand and improve upon the original GAN formulation based on the understanding that the network is unstable to train; the GAN is prone to mode collapse (in which $\mathcal{G}$ generates a limited set of outputs [6], [7]) and suffers from the vanishing gradient problem [8].

Such improvements include the Deep Convolution GAN (DCGAN), in which Radford et al. [9] model a new variant of the GAN based on the structure of the Convolutional Neural Network (CNN) [10]. Using structural changes such as the use of global average pooling [11], Batch Normalization [12] and the Rectified Linear Units (ReLU) activation function [13], they find modelling capabilities to be of a higher resolution but fail to fully address the instability of the original GAN [2], [9].

Another further improvement was found in the Wasserstein GAN (WGAN), in which Arjovsky et al. [8] identify that the value function is the source of much instability and thus define a newer, more stable value function. They propose the Earth-Mover (EM) distance (the optimal plan to move one mass x to another mass y) and show that it is not prone to mode collapse and does not experience vanishing gradients, making the WGAN more stable to train [8]. Shown to be more stable in training than the original GAN [2], [8], the Wasserstein GAN still contains an instability caused by the use of weight clipping [14]. Gulrajani et al. [14] propose the WGAN with Gradient Penalty (WGAN-GP), which removes weight clipping in favour of a direct constraint to the gradient norm of the discriminator's output. Comparing the results of the WGAN-GP to the WGAN and DCGAN, they show that the WGAN-GP converges significantly faster than the WGAN. Although converging slower than the DCGAN, the convergence of the WGAN-GP is more stable than the DCGAN [14].

2.1 Project Specification

2.1.1 Functional Requirements. The functional requirements, as originally presented, specify the Scene Classification dataset as the main source of data for training and testing; this remains unchanged. I had originally specified a requirement to use the dataset as provided (using the 150x150 size images for training) to generate camouflage to fit a specific size and shape of 'hole' in an image (a blank subsection). Furthermore, I had stated that another similar project, the CamoGAN [15], should be modified to use my chosen dataset. Moreover, I had stated that the CamoGAN should be expanded into using the EM distance [8] and further again into using the Gradient Penalty [14]. The original functional requirements can be seen below:

- The Scene Classification Dataset [16] must be used to train and test the GANs
- All GANs must be trained to generate camouflage for a 32x32 central hole
- If above requirement is met, the GANs should be trained to generate camouflage for holes of arbitrary shape, size or location

- A basic GAN based on Goodfellow et al.'s method [2] must be implemented
- The CamoGAN [15] must be modified to use the Scene Classification dataset [16]
- Time permitting, the CamoGAN should be modified to use the EM distance [8]
- If above requirement is met, the modified CamoGAN should further use the Gradient Penalty from Gulrajani et al. [14]

Whilst working on this project, I had found that some of these requirements needed modification to better fit the scope of the project. Firstly, I made the decision to instead train each GAN on, at minimum, 64x64 resized versions of the Scene Classification dataset (where each image was originally 150x150 [16]); each GAN would also produce images at a minimum size of 64x64. I made this decision as much existing literature surrounds the use of image sizes belonging to the power of two set i.e. 64x64, 128x128, 256x256, as well as the hyper-parameters being well researched. Moreover, whilst working with the CamoGAN [15] I found that it was too different in structure and operation to justify using it as a basis for my own work; thus I deemed it not fit for the scope of this project and, instead, opted to build my own DCGAN for the Scene Classification dataset [16]. By building my own DCGAN, I was able to create a structure more fitting for this project, alongside it being much more open to extension into the WGAN.

Upon determining these key modifications, the functional requirements to which the project adhered to are as follows:

- All GANs must be trained on and produce images at a minimum size of 64x64.
- A GAN based on Goodfellow et al.'s method [2] must be implemented
- A DCGAN based on Radford et al.'s method [9] must be implemented
- A WGAN based on Arjovsky et al.'s method [8] must be implemented
- Time permitting, the result of the above requirement should use the Gradient Penalty from Gulrajani et al. [14]

2.1.2 Non-Functional Requirements. I had also defined a small number of non-functional requirements which, in short, are as follows:

- Train the GANs for a maximum of 10,000 epochs or 24 hours; whichever choice is the shortest
- Create a user-friendly user interface (UI) for a human observation test

The non-functional requirement for training existed simply to limit the length to which I was training the various networks. Additionally, the non-functional requirement to create a user-friendly UI was chosen to reduce the risk of testing errors where test participants might be confused by an overly complex, hard-to-use test software.

2.1.3 Evaluation Criteria. Furthermore, I determined the following evaluation criteria which did not need any further modification over the course of the project:

- Track the loss metric from both networks and graph with respect to the number of epochs
- Carry out a user observation test in which a minimum of 5 participants must visually identify whether an image contains a camouflaged patch

Whilst training the networks, I had decided that the two evaluation criteria that had been determined at the beginning of the project were not entirely sufficient and that I should also specify some evaluation criteria to determine the similarity between the generated camouflage and a given background image

using some common image quality assessment techniques, as shown in Lin et al. [17]. These evaluation criteria are as follows:

- The Mean Square Error (MSE) should be calculated between an image with a camouflage patch and the original image
- The Universal Image Quality Index (UIQI) modification should be calculated between an image with camouflage patch and the original image [18]

3 DESIGN

In this section, I cover the design process followed over this project as well as the initial high-level system architecture (seen in Section 3.3) of the project.

3.1 The Scene Classification Dataset

As already mentioned in Section 2.1.1, I will be using the Scene Classification Dataset [16] to train and test the models built. As stated prior, the images are originally a width of 150 pixels and a height of 150 pixels, which does not fit into the powers of 2 set (e.g. sizes 64x64, 128x128, 256x256) that much literature has already researched well.

3.2 Technologies

Key to any research project is the correct choice of technologies, of which I discuss in the following section. I will be using the Python programming language (specifically version 3.9) due to its simplicity and readability for users, as well as its lightweight implementation. Furthermore, there exists a vast number of libraries best suited for Machine Learning which provide excellent tools for building, training and testing various models and networks. All code will be written in the PyCharm IDE. Firstly, I will be using the NumPy library [19] to extend the mathematical functionalities of Python, as well as to allow for better structural representation of images. Secondly, I will use the scikit-image library [20] to facilitate image processing during the test and evaluation stages of this project, as well as using the Matplotlib library [21] to facilitate the creation and saving of data and image plots. Finally, I will use Pytorch [22], a high-level library which wraps around the Tensorflow machine learning interface [23], to build and train the Generative Adversarial Network and variants. Alongside the use of Pytorch to build the networks, Torchvision [24] will be used to facilitate the loading, transformation and subsequent formatting of generated data back into images.

All programming, running and testing of the GANs will be carried out on a machine with an Intel i9-9900K processor, 8GB NVIDIA GeForce RTX 2070 Super graphics card and 16GB of memory. Due to the strength of the graphics card in this machine, I will leverage the Compute Unified Device Architecture (CUDA) [25] to perform the training on the graphics card itself.

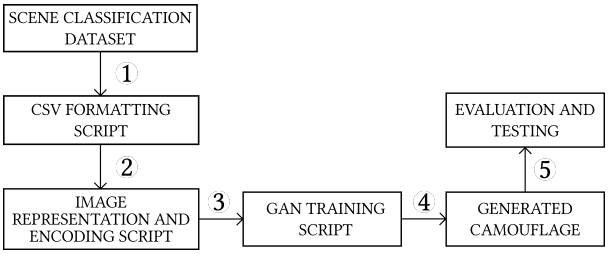


Fig. 1. Abstract, high-level design of the system environment. The GAN Training Script represents any of the GAN variants implemented in this project.

3.3 High-level System Architecture

Important to the development of this project was defining an overall high-level view of the system itself and how data flows through it. This overall structure can be seen in Figure 1, with the result of each stage labelled from 1 through 5.

(1) The Scene Classification Dataset [16] CSV, containing details for each image's class, is read by a CSV formatting script and split into separate folders, one for each class.
(2) Once correctly formatted, the dataset is loaded from each class folder into a Pytorch [22] `DataLoader` object, resized to shape $(3, 64, 64)$, converted into a Tensor and is mean-normalized. A single class is chosen at this stage to be used as training and test data.
(3) The new `DataLoader` is supplied to one of various GAN training scripts in which the hyper-parameters (e.g. Learning Rate, Batch Size or Number of Epochs) and the two networks are instantiated.
(4) During training, a selection of 32 images are outputted to a results folder for each epoch. The Generator and Discriminator losses are also outputted to a log file for each epoch. Once outputted, a 32x32 central subsection is extracted from each generated image and placed over the exact same area on a real image.
(5) Now that the camouflage has been generated, it is passed to evaluation and testing scripts. Selected camouflage is used as part of user observation tests, as well as for calculating the Mean Square Error and Universal Image Quality Index.

4 DEVELOPMENT

The following section describes the complete life-cycle of development over the course of this project. Each variation of the GAN is explained in detail including refined designs, discussion of structure and the training of each model. Development can be split into five independent sections; the first section covers the development of scripts which handle the formatting and encoding of the dataset for use with the implemented GAN techniques. Moreover, the subsequent four sections each cover an implemented GAN technique, starting with the development of the discriminator and following with the development of the generator. Finally, these sections each end with details about how training is structured.

4.1 Dataset Formatting and Encoding

A logical first step in this project was to properly format the Scene Classification dataset for use with Pytorch machine learning models. As detailed in Section 3.3, I first developed a script (`csv_handling.py`) which split each image into separate directories based on the classes listed in the given CSV file (`train.csv`).

Once formatted into class-specific directories, the images are ready to be imported into a Pytorch `DataLoader`. Firstly, the images are mean normalised, with each pixel's colour value being replaced with the Z-score, and transformed into tensors. The `DataLoader` used allows for effective splitting of training and test data, as well as facilitating iteration over large datasets through the use of batches.

4.2 Original Generative Adversarial Network

I started by prototyping and implementing the original Generative Adversarial Network technique proposed in [2], as it is the most simple of the techniques proposed in the functional requirements (Section 2.1.1).

As seen in Figure 2, input data images of shape (64, 64, 3) (i.e. 64 pixel width, 64 pixel height and 3 colour channels) are flattened to shape (12288, 1, 1) and fed into a Linear transformation layer of 64 neurons. The result of this then passes through a LeakyReLU activation function [26], [27], chosen for it's suitability for higher resolution modelling [9]. Subsequently, these results are passed into another linear transformation layer consisting of a single node intended to produce a singular output; thus, a Sigmoid activation function is used to ensure that the output of this network falls within the range of $[0, 1]$.

It can be seen in Figure 3 that the structure of the Generator is not dissimilar to that of the Discriminator in Figure 2; a noise vector of length 100 is supplied to a linear transformation layer, consisting of 256 neurons. As with the Discriminator, LeakyReLU [26], [27] is used again, followed by another linear transformation layer, with 12288 neurons to ensure the output of the network is a flattened 64x64x3 image. Finally, the Tanh activation function is used to ensure the values of the flattened image fall within the range $[-1, 1]$.

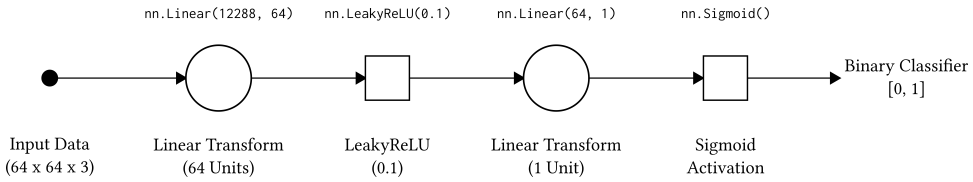


Fig. 2. Structure of the Discriminator Network. Shown below each component is the component's name and technical details. Above each component is the relevant Python statement.

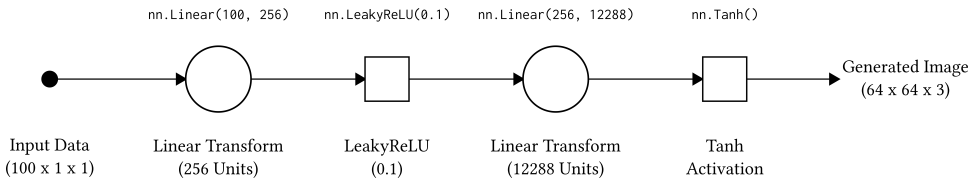


Fig. 3. Structure of the Generator Network. Shown below each component is the component's name and technical details. Above each component is the relevant Python statement.

For both networks, an alpha value of 0.1 is used for the LeakyReLU activation functions, which is the size of the small, non-zero gradient produced when the unit would be saturated and not active (i.e. for negative values) [26].

4.2.1 Training the Generator and Discriminator. Both networks are trained using the Adam optimiser [28], with learning rate of 3×10^{-4}. To train the discriminator, both a real and fake sample are passed through the network with the Binary Cross Entropy (BCE) loss calculated for each respectively. The loss for the discriminator is then calculated as seen in Equation (2); subsequently, the stored gradients are set to None (using `Optimizer.zero_grad()`) and a backwards pass is carried out to calculate the gradient of the discriminator (using `Criterion.backward()`), this gradient is then back-propagated using `Optimizer.step()`.

$$\frac{BCE(D_{real}) + BCE(D_{fake})}{2} \tag{2}$$

To train the generator, a fake sample is passed to the discriminator and the loss for the generator is calculated using the BCE loss criterion. Fixed noise is passed to the Generator to ensure that the outputted images can be tracked over each epoch. 32 images are outputted in an image grid, alongside the losses for both networks in a text file.

4.3 Extension to the Deep Convolutional GAN

Following the implementation of Goodfellow et al.'s GAN technique [2], I next implemented Radford et al.'s Deep Convolutional GAN (DCGAN) [9]. Similar to the original GAN, the DCGAN implementation is split between two scripts: `scene_dcgan.py` implements the Discriminator and Generator networks and `scene_dcgan_train.py` implements the methods for training the DCGAN.

4.3.1 Implementation of the Discriminator. For the DCGAN, and all further implementations, I began using reusable blocks of layers, with each block (`Discriminator()._block()`) made up of a 2-Dimensional (2D) Convolution, a 2D Batch Normalization [12] and a LeakyReLU activation (using an alpha value of 0.2). This would allow me to create a network of an arbitrary number of layers with little repetition of code. As seen in Figure 4 an image, real or fake, is supplied to the network with the shape $(3, 64, 64)$. The very first and very last layer of the DCGAN differ from all other layers as they do not utilise the `_block()` method. Instead, the first layer performs a 2D Convolution followed by a LeakyReLU activation. Per Radford et al., use of Batch Normalization in the input layer of the Discriminator causes sample oscillation and model instability [9] and is omitted. The resulting feature map from the first 2D convolution has the shape $(64, 32, 32)$.

Batch Normalization is used in each block to normalize the inputs to each layer of the network and is shown to provide more stability in training and is critical in the prevention of mode collapse [2]. After three block layers, the feature map has the shape of $(512, 4, 4)$ and is at the point where a final 2D convolution can flatten the feature map to a single $(1, 1, 1)$ feature map, where it is passed through a Sigmoid activation function to produce a single output value within the range $[0, 1]$. To classify the output values, the Binary Cross-Entropy (BCE) loss is calculated for the output, with the target for real images being values of 1 and the target for fake images being values of 0.

4.3.2 Implementation of the Generator. Similar to the Discriminator, layer blocks (`_block()`) are utilised and are comprised of a 2D transposed convolution, 2D Batch Normalization and a ReLU activation function; unlike the Discriminator, the blocks can be used for all layers except the output layer, which is again a cause of instability if subject to Batch Normalization [9].

A 100-value vector of random noise is supplied to the Generator and passed through a layer block, producing a feature map of shape $(1024, 4, 4)$. As seen in Figure 4.3.2, this layer block is repeated a total of four times resulting in a feature map of shape $(128, 32, 32)$. At this point in the Generator, the feature map is ready to be transformed into the shape and format of an RGB-colour image and is passed through a final transposed 2D convolution, resulting in the shape $(3, 64, 64)$. Finally, passing through a Tanh activation function ensures that each pixel value falls within the range $[-1, 1]$.

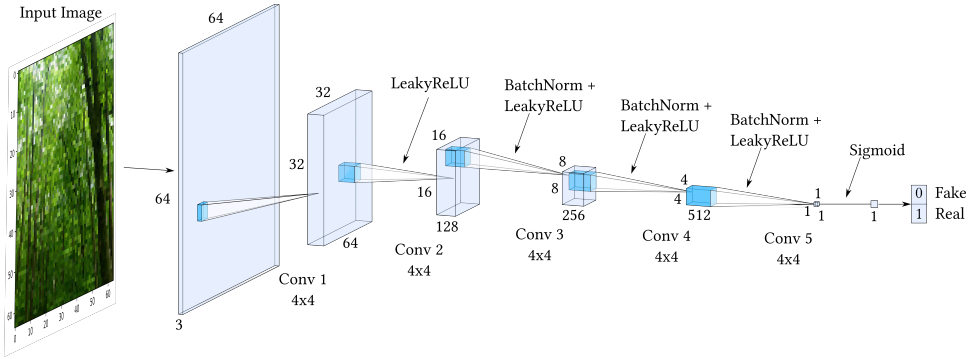


Fig. 4. The architecture of the Discriminator for the implemented DCGAN. Each Convolution uses a stride of 2 and kernel size of (4,4). The final Sigmoid activation ensures the output is within the range [0, 1], with the BCE loss calculated from this output value

4.3.3 Training the DCGAN. Training of the DCGAN is carried out in the script `scene_dcgan_train.py` and uses a similar structure to the GAN training script, `scene_gan_train.py`, seen in section 4.2.1. This script begins by setting the hyper-parameters needed for training and imports the Forest class from the Scene Dataset [16] following the process detailed in section 4.1.

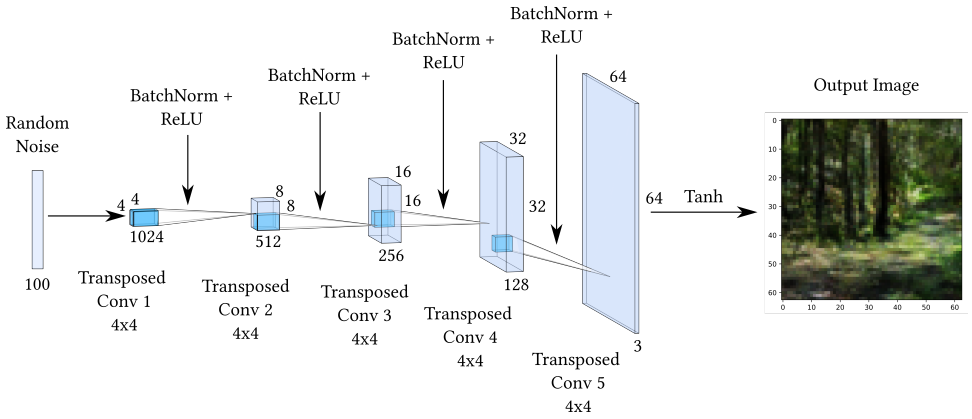


Fig. 5. The architecture of the Generator for the implemented DCGAN. Each Transposed Convolution uses a stride of 2 and kernel size of (4,4). The final Tanh ensures that each pixel value is within the range [-1, 1].

The DCGAN uses a Convolutional Neural Network (CNN) in-place of the Multi-Layer Perceptron (MLP) used by the original GAN [9]; due to this, the weights of the network must also be initialised. Next, the Adam optimiser is initialised for both networks using the set learning rate and set betas $\beta_1 = 0.5$ and $\beta_2 = 0.999$. During the main training loop, both real and fake images are supplied to the discriminator, with the BCE Loss being calculated for the results of both the real and fake images, using Equation (1).

Training the generator follows the same path as the Original GAN, with a fake sample being passed to the discriminator and the generator BCE loss calculated from that result. At the start of every epoch, the generator and discriminator losses are both outputted to a log file and a grid of 32 images generated from a sample of fixed noise are outputted.

4.4 The Wasserstein GAN

The Wasserstein GAN (WGAN) is implemented following Arjovsky et al.'s method [8], in which the objective function of a DCGAN is replaced with the Earth-Mover (EM) distance.

4.4.1 Implementation of the Discriminator. The discriminator for the WGAN is, for the most part, unchanged from the DCGAN discriminator, so much of the DCGAN discriminator can be used for the WGAN. Owing to the new objective value function, the output of the discriminator does not need to fall within any specific range and thus the main difference between this and the DCGAN discriminator is the omission of the Sigmoid activation function in the output layer.

As seen in Figure 6, the input image is supplied with the shape $(3, 64, 64)$ and undergoes a single convolution, producing a feature map of shape $(64, 32, 32)$, and is subject to a LeakyReLU activation function. Next, the same layer blocks consisting of a 4×4 kernel convolution, a 2D Batch Normalization and a LeakyReLU activation are repeated three times, producing a single loss value. Unlike the DCGAN, the discriminator loss does not utilise the Binary Cross-Entropy loss criterion, instead taking the direct output of the discriminator for both real and fake images. The discriminator loss, $L^{(\mathcal{D})}$, is calculated as seen in Equation (3), where $\overline{\mathcal{D}(x)}$ is the mean loss for real images and $\overline{\mathcal{D}(\mathcal{G}(z))}$ is the mean loss for

fake images.

$$L^{(\mathcal{D})} = -(\overline{\mathcal{D}(x)} - \overline{\mathcal{D}(\mathcal{G}(z))}) \tag{3}$$

No changes are made to the generator, as the differences between the DCGAN implementation and the WGAN exist only within the Discriminator and training process itself.

4.4.2 Training the WGAN. Once both the discriminator and generator were implemented, developing the training script was the next focus. Unlike the GAN and DCGAN, the WGAN does not utilise the Binary Cross-Entropy loss when calculating the Discriminator loss, thus meaning the previous training scripts would not be completely reusable as in the past; in fact, only the area of script pertaining to dataset importation was reused.

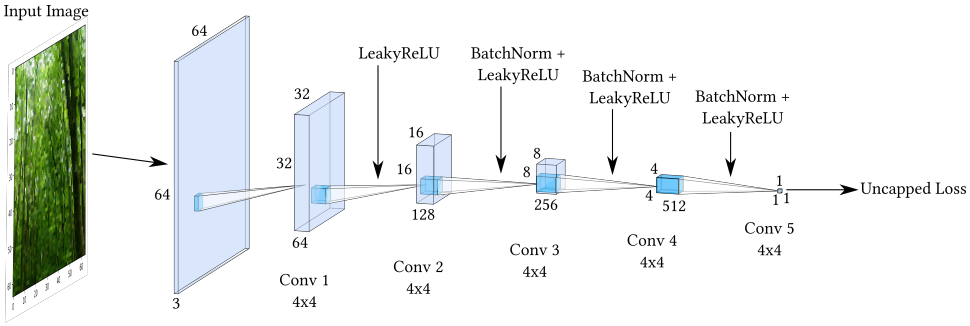


Fig. 6. The architecture of the Discriminator for the implemented WGAN. Each Convolution uses a stride of 2 and kernel size of (4, 4). There is no activation function in the output layer to ensure that the output is uncapped.

A weight clip parameter is employed to provide the range within which the discriminator's parameters are clamped. This use of weight clipping was proposed by Arjovsky et al. [8] to regularise the weights of the WGAN and enforce the Lipschitz constraint required to train successfully. Unlike the GAN and DCGAN (sections 4.2.1 and 4.3.3 respectively), the WGAN does not use the Adam optimiser; instead, the RMSProp optimiser is used in line with Arjovsky et al.'s paper [8], who find it better suited for training the WGAN and find it outperforms the Adam optimiser.

For each epoch of training, the discriminator is trained multiple times every time the generator is trained once i.e. for a Critic Iteration value of five, the discriminator is trained five times every time the generator is trained once. Within the discriminator training loop, a sample of both real and fake images are supplied to the discriminator for which two individual loss values are produced. Equation (3) is used to calculate the overall loss for the discriminator, and the process introduced in Section 4.2.1 is followed to train the model itself. Finally, every parameter in the discriminator is clipped within the range of the Weight Clip hyper-parameter i.e. within the range $[-x, x]$, where x is the Weight Clip hyper-parameter.

To train the generator, a fake image is passed to the discriminator, with the result being used in the calculation of the generator loss seen in Equation (4).

$$L^{(\mathcal{D})} = -\overline{\mathcal{D}(\mathcal{G}(z))} \tag{4}$$

Again, the model training process is followed and, at the start of each epoch, a selection of 32 images are generated from some fixed noise and saved as a grid of images; alongside these images, the loss values of both the generator and discriminator are saved into a text (`.txt`) file for later interpretation.

4.5 Addition of the Gradient Penalty

As detailed in the functional requirements for this project (section 2.1.1), I had created an optional functional requirement, being: "Time permitting, the result of the above [WGAN] requirement should use the Gradient Penalty from Gulrajani et al. [14]". On finishing the WGAN implementation, I had found that I would have a sufficient amount of time to implement the WGAN with Gradient Penalty (WGAN-GP).

The implementation of the WGAN-GP does not require any structural changes to the generator, so the script `scene_wgan.py` remains largely unchanged, although it is copied into another script, `scene_wgan_gp.py`, for the sake of encapsulation and readability. For the structure of the WGAN-GP's generator, refer to Section 4.3.2 and Figure 5. However, to properly implement the gradient penalty, Batch Normalization should not be used within the discriminator; hence, these are replaced with Instance Normalizations. The structure of the discriminator is as seen in Figure 7.

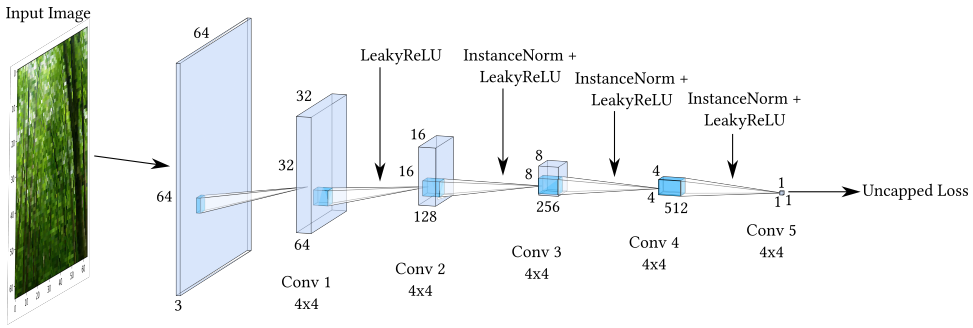


Fig. 7. The architecture of the Discriminator for the implemented WGAN-GP. Each Convolution uses a stride of 2 and kernel size of (4, 4). InstanceNorm is used in-place of BatchNorm.

4.5.1 Training the WGAN-GP. Training of the WGAN-GP is handled by the script `scene_wgan_gp_train.py` and uses an additional utility function, `gradient_penalty()`, from the script `scene_util.py`. As Gulrajani et al. [14] had proposed in their paper, the weight clipping used by the WGAN to regularize model parameters was still a source of instability and thus needed to be replaced by a more stable alternative. This was found in the Gradient Penalty, which directly constrains the gradient norm of the output of the discriminator [8], [14].

The Gradient Penalty Lambda, which controls the significance of the Gradient Penalty when calculating the discriminator loss, is set as a hyper-parameter in place of the Weight Clip parameter previously. The discriminator loss, $L^{(\mathcal{D})}$, is calculated as seen in Equation (5).

$$L^{(\mathcal{D})} = -(\overline{\mathcal{D}(x)} - \overline{\mathcal{D}(\mathcal{G}(z))}) + \lambda(\|\nabla_{\hat{x}}\mathcal{D}(\hat{x})\|_2 - 1)^2 \tag{5}$$

where λ is the Gradient Penalty Lambda, $\overline{\mathcal{D}(x)}$ is the mean loss for real images and $\overline{\mathcal{D}(\mathcal{G}(z))}$ is the mean loss for fake images. Furthermore, $\hat{x}$ is an interpolated image between real data $x \approx p_{data}$, generated data $\tilde{x} \approx p_g$ and a random number $\varepsilon \approx U[0, 1]$, seen in Equation (6).

$$\hat{x} = \varepsilon x + (1 - \varepsilon)\tilde{x} \tag{6}$$

Moreover, the Adam optimiser is used here in place of RMSProp. As with the WGAN, the discriminator is trained multiple times per epoch, in which both real and fake images are passed and two individual losses are calculated. The gradient penalty is also calculated based on this set of real and fake images and is used in the calculation of the overall discriminator loss, seen in Equation (5). As with all other previous implementations, the same process is used for training of the models themselves, seen in Section 4.2.1, and, for each epoch, a grid of 32 generated images and the losses for both generator and discriminator are saved.

5 EXPERIMENTAL RESULTS

In this section, I discuss the process of creating the generated camouflage, as well as the results seen from training each GAN over the course of this project.

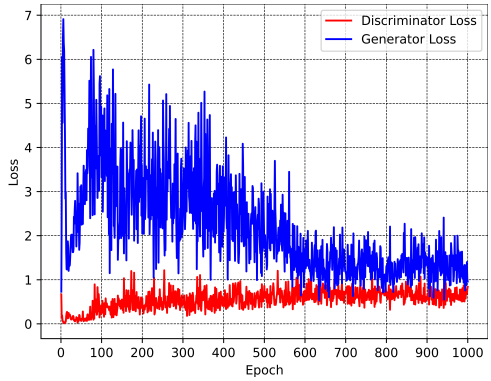
For each training run, only the hyper-parameter controlling number of epochs (`NUM_EPOCHS`) was changed. The training runs were as follows: 10 epochs, 25 epochs, 50 epochs, 100 epochs, 250 epochs, 500 epochs and 1000 epochs. I had intended to further extend this training routine to allow for a larger number of epochs in the case that these would be insufficient; after subsequent training runs, I found that neither the 10,000 epoch limit nor the 24 hour limit ever needed to be imposed and would in fact be too long of a time to train the models for.

Each loss graph seen in Sections 5.1 through 5.4 is representative of the generator loss and combined discriminator loss (i.e. the loss for the discriminator owing to Equations (2) or (3), rather than for real and fake images independently) and are captured over the course of the 1000 epoch training session.

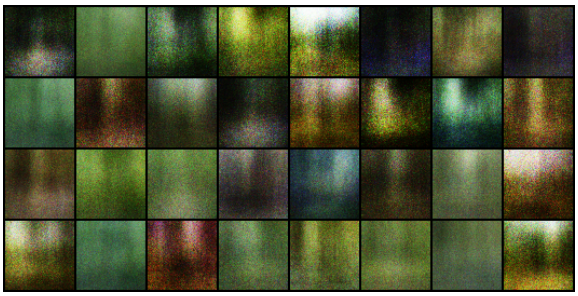
5.1 Original GAN Results

Although the exact loss values generated by the GAN are not indicative of success by themselves, the behaviour of the losses over time can tell one whether a GAN is stable and whether it has encountered any failure modes. Seen in Figure 8a, one can see that the generator loss starts initially very high and, over the course of training, trends downwards, demonstrating the improvements in generation the model finds over 1000 epochs. It can also be seen further that the generator loss oscillates largely until around 600 epochs of training and continues to both oscillate less and overall decrease less, meaning the model is beginning to converge. The discriminator loss starts very low and begins to rise until 200 epochs where, for the remaining 800 epochs, it oscillates between a loss value of 0.5 and 1.0. As mentioned in Goodfellow et al. [2], the optimally trained discriminator produces a loss value of 0.5 and, as such, I would expect a near-optimal discriminator to maintain a loss value within the range $[0.5, 1.0]$.

Although the loss values suggest a stable GAN, the images produced over the course of training do not suggest successful training. The images produced until 500 epochs of training has passed are largely still noise and only after 500 epochs of training do they start to take on their own colour and structural details; even still, after 1000 epochs the images are still very noisy and lack any discernible structures that could be attributed to any forest, as seen in Figure 8b.



(a) Discriminator and generator loss against epochs

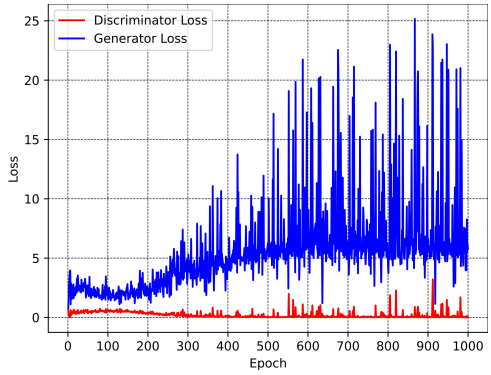


(b) Generated images at 1000 epochs

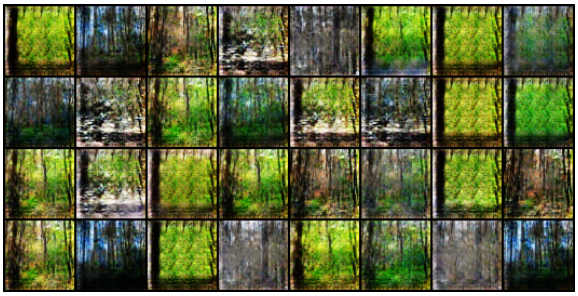
Fig. 8. GAN loss graph and generated images at 1000 epochs

5.2 DCGAN Results

The loss values for the DCGAN, seen in Figure 9a, initially demonstrates stable training. Again, like in Section 5.1, the generator loss starts high and decreases at the same time as the discriminator loss rises to a stable state, demonstrating both convergence and stable training of the generator. This occurs until around 200 epochs, where the generator loss makes a sharp increase and begins to wildly oscillate. At the same time as this, the discriminator loss decreases until it has a constant value of zero. These two behaviours are demonstrative of mode collapse, in which the generator learns to produce a limited set of images from the latent noise z [6], [7], meaning that until 200 epochs the DCGAN was stably training and that after 200 epochs, the DCGAN was falling into mode collapse.



(a) Discriminator and generator loss against epochs



(b) Generated images at 1000 epochs

Fig. 9. DCGAN loss graph and generated images at 1000 epochs

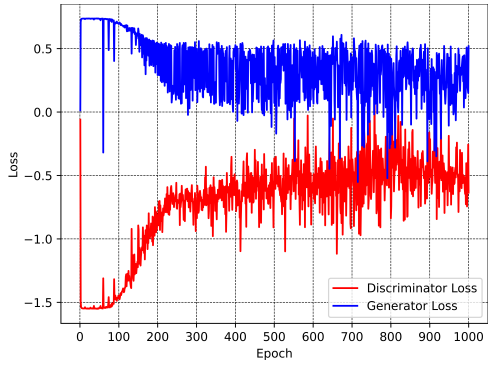
Furthermore, the full impact of mode collapse can be seen in Figure 9b, where many of the generated images are near-identical to each other; at this point in training the DCGAN is oscillating between multiple modes of generation, resulting in the repetition of four images instead of the generation of 32 unique images.

5.3 Wasserstein GAN Results

With the changes made to the objective function for the WGAN and the use of the Wasserstein Distance, the shape of the loss graph produced by the WGAN is more representative of what exactly is happening within the training process, in particular where the discriminator is concerned; the behaviour of the generator loss by itself is not representative of improvements in training and does not tell one much about the behaviour of the GAN during training, unless studied alongside the behaviour of the discriminator loss.

In Figure 10a, it can be seen that the generator loss starts at a high positive figure (approximately 0.73) whilst the discriminator starts at a low negative number (just under -1.5) and rises whilst the generator loss falls. As the discriminator loss represents the negative Wasserstein distance between the generator's learned distribution and the distribution of the real dataset, this behaviour is indicative of stable learning. Furthermore, it it can be seen that at 500 epochs, the discriminator loss oscillates within the range $[-1.0, 0.0]$; the same appears to occur with the generator loss, oscillating within the range $[-0.5, 0.5]$. This reduction in change in loss is evidence of convergence in learning, where the model sees less dramatic changes over time and instead sees the two networks oscillate between states as they learn.

Visually, images generated by this WGAN only improve over time as the models converge (although the improvements seen over each epoch become less noticeable as convergence slows) and show no evidence of mode collapse, as evidenced by the images generated at Epoch 1000 in Figure 10b. On comparison with Figure 9b, one can see that the images produced by the WGAN do not experience mode collapse and, in fact, appear to be more detailed than the resulting images from the DCGAN.



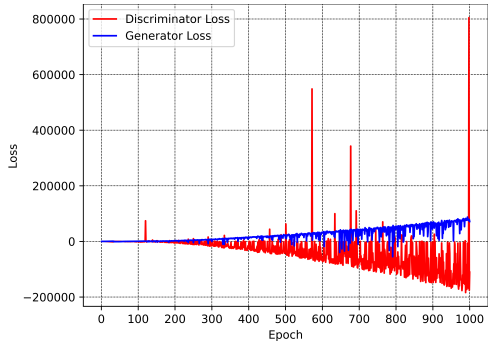
(a) Discriminator and generator loss against epochs

(b) Generated images at 1000 epochs

Fig. 10. WGAN loss graph and generated images at 1000 epochs

5.4 Gradient Penalty Results

The addition of the Gradient Penalty in Section 4.5 is made with the intention of both increasing stability during training and increasing the quality of images generated. However, the loss graph in Figure 11a suggests that instabilities in training remain, with both the generator and discriminator losses diverging where they should instead be converging. Furthermore, a failure to converge indicates a failure in training and that neither the generator, nor the discriminator, are learning successfully.



(a) Discriminator and generator loss against epochs

(b) Generated images at 1000 epochs

Fig. 11. WGAN-GP loss graph and generated images at 1000 epochs

Failure to converge often results in failure to learn the distribution of the real dataset and, as such, results in failure to generate images of sufficient detail. However, this appears to not be the case with the images generated by this implementation of the WGAN-GP. Looking at Figure 11b, after 1000 epochs of training the images produced by the generator are of the same quality as the images produced by the original WGAN (Figure 10b). This suggests that the divergent nature of the WGAN-GP loss values bear little effect on the actual images produced as a result of training. Furthermore, the oscillation of the discriminator loss returning to a value of zero might suggest that the discriminator has converged sufficiently well enough to enable learning in the generator. It is also possible that an unknown artifact of my own implementation is causing the discriminator to attempt to improve when, in reality, it cannot improve any further and hence causes an oscillatory nature between 0 and an ever-decreasing minimum loss value.

6 ANALYSIS AND EVALUATION

6.1 Image Similarity Measures

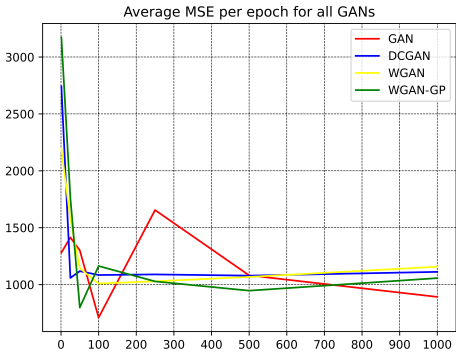
As mentioned previously in the Project Specification (Section 2.1.3), I have chosen both qualitative and quantitative methods for evaluating the extent of success in training the various GAN implementations. In this section I cover the quantitative measures of success using the Mean Square Error (MSE) and Universal Image Quality Index (UIQI), based on the findings of Lin et al. [17].

Both similarity measures are calculated between an image with camouflage and the same image without camouflage; for the sake of this paper, the same set of background images has been used for

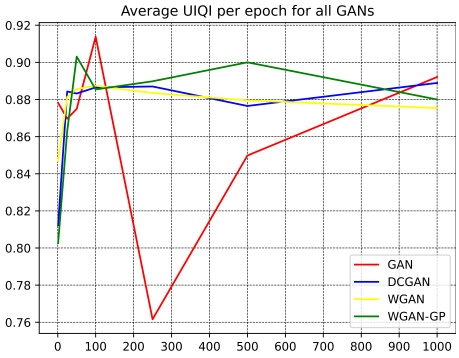
each implementation. The lower the MSE value, the more similar an image is and vice versa; alongside this, the UIQI always falls within the range $[-1, 1]$, with the most similar images tending towards 1.0 and the most dissimilar tending towards -1.0.

For each GAN type, a selection of 32 generated images were chosen at epochs 1, 10, 25, 50, 100, 250, 500 and 1000 and made into camouflage images; both the MSE and UIQI were calculated for each image at these epochs and for each GAN implementation, thus giving me approximately one thousand values for both MSE and UIQI. From these values, the mean MSE and mean UIQI values were calculated; the two graphs, Figure 12a and Figure 12b, are plots of the average MSE per epoch and average UIQI per epoch respectively.

Starting with the Mean Square Error (MSE), one can see that all four GAN implementations start with very high MSE values and after 100 epochs have reduced to very low MSE values; furthermore, the original GAN sees the lowest average MSE value of all implementations at 100 epochs of training, meaning the images produced by the GAN after 100 epochs could be considered the most similar to the original dataset. After 250 epochs the DCGAN plateaus, likely due to the mode collapse seen at this point, whereas the GAN spikes at a very high value, demonstrating the high instability during training. Both the WGAN and WGAN-GP see a very small increase in average MSE after 500 epochs, suggesting that the images generated after this point are less similar than the ones generated at 250 epochs. One can see that, at the end of training, the GAN has the lowest average MSE, which suggests that it produces camouflage that is the most similar in colour to the original dataset. The WGAN-GP is considered second most similar, the DCGAN third and the WGAN placing last with the highest average MSE at the end of training.



(a) A graph showing the Average Mean Square Error value for each GAN implementation at each epoch



(b) A graph showing the Average Universal Image Quality Index value for each GAN implementation at each epoch

Fig. 12. Graphs of Average MSE and Average UIQI per epoch

Additional plotting of the average Universal Image Quality Index values for each implementation over specific intervals in training reveals similar results. The DCGAN and WGAN-GP start with UIQI values close to 0.80 and the WGAN starts with a UIQI value close to 0.85; the original GAN starts with a much higher UIQI and sees a sharp rise to the highest UIQI value seen over all implementations at just over 0.91. The WGAN proceeds to plateau which might signify a convergence of image quality.

After 1000 epochs, the GAN is still considered able to produce camouflage that is the most similar to the real dataset, with the DCGAN narrowly behind it, WGAN-GP in third and, once again, the WGAN producing the least similar camouflage.

These values take a new perspective when compared with the images produced by each respective implementation after 1000 epochs. One might expect the WGAN or WGAN-GP to take the top spot in both of these measures, as visually they seem the most similar to the background dataset. However, the measures used here do not take into account the structure of the background image nor the context and instead use only the raw pixel data (i.e. Red-Green-Blue levels and Luminance) to calculate similarity. This means that although some generated camouflages might look similar to the background to humans based on the structures generated within, they might not match the colours and luminance levels used by the MSE and UIQI measures causing lower similarity levels than expected.

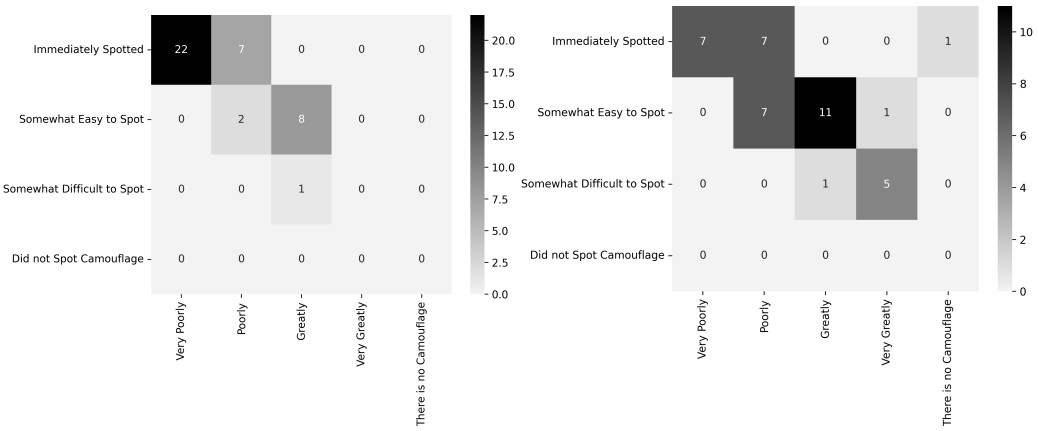
6.2 Observational Test

The other evaluation measure mentioned in Section 2.1.3 was to run a human observation test, in which five participants were asked questions about fifty individual images. Each section of the survey focused on one image, in which the participants were asked how well they could spot the camouflage, ranging from being able to immediately spot the camouflage to not being able to spot camouflage at all. Furthermore, participants were asked how well they thought the camouflage blended in with the background (qualitatively measuring saliency based on human opinion), ranging from 'Very Poorly' to 'Very Greatly', alongside another option for when the participant thought that there was no camouflage. By asking all five participants two questions about fifty different images, I have a comprehensive set of five hundred datapoints from which evaluation can be made.

Although only providing qualitative data, which is not easily interpreted statistically, this survey does allow me to evaluate the opinions of people who, unlike myself, have not spent any time looking at and creating the camouflage and, thus, are not affected by confirmation bias. Moreover, by calculating the number of so-called incorrect responses (i.e. responses where the participant thought that there was camouflage where there was not, or vice versa), I can evaluate the efficacy of camouflage in regards to human observation.

Seen in Figures 13 and 14, the responses were separated by the source of each image and compiled into heatmaps. Furthermore, each individual figure on the heatmap represents the number of individual responses, regardless of image or participant, that pertain to those classes.

Starting with the GAN in Figure 13a, one can see that participants considered a majority of the generated camouflages to be very poor in quality and thus immediately spotted them; in fact, only 9 of the 40 responses considered the generated camouflages to be of a good quality, with exactly one participant finding a single GAN generated camouflage to be difficult to spot and the remaining 31 responses being considered poor in quality. Overall, although the average MSE and UIQI graphs (Figure 12 in Section 6.1) suggest that the GAN produces the most similar images based on raw pixel values, when presented to a human they find that the generated images are not sufficient as camouflage and are, as a result, considered very ineffective.

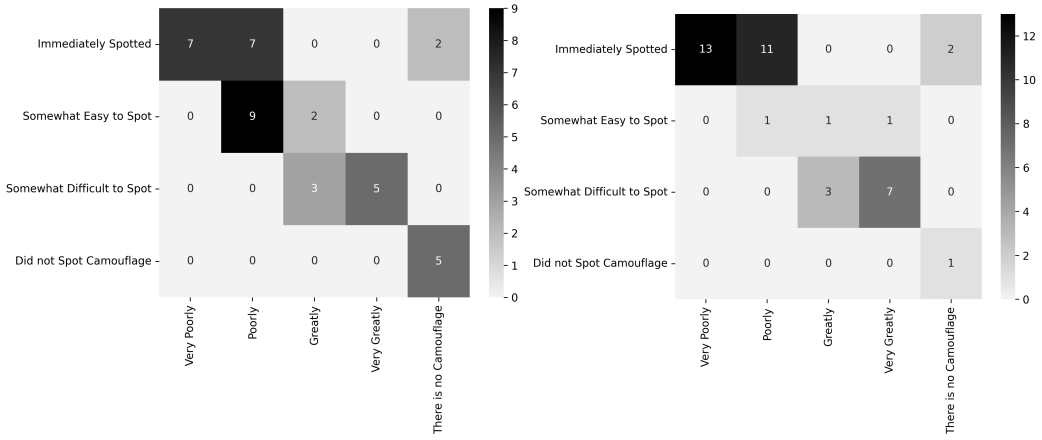


(a) A response heatmap for the GAN (b) A response heatmap for the DCGAN

Fig. 13. Response Heatmaps

Compared to the GAN, the DCGAN in Figure 13b performs much better when presented to human participants. A much smaller majority of 21 responses felt that the generated images were of poor quality, whereas 18 responses considered the generated images to be of a great quality (although 11 of those responses still easily spotted the camouflage). Exactly one response had said that an image contained no camouflage, yet also spotted the camouflage immediately; this is likely an anomalous response in which a participant had seen such a poor example of camouflage that they did not consider it to be camouflage at all. In this case, I should have explicitly stated that the name camouflage is used independently of the sample's effectiveness and is an all-encompassing term. Furthermore, it appears that the DCGAN images are both considered similar to the real images by the similarity metrics, whilst also being generally considered as good camouflage by humans.

Similarly, the WGAN (Figure 14a) sees a slight majority of 23 responses considered as poor quality camouflage where 14 were immediately spotted and 9 were easily spotted. Contrarily, 10 responses considered the WGAN camouflage to be great camouflage. The most important aspect of the responses to the WGAN, which differentiates it from all other implementations, is that 5 responses misidentified the camouflage and in fact stated that there was no camouflage at all. These responses mean that the WGAN has the ability to produce camouflage that is indiscernible from the real dataset to the untrained eye. Although a modified version of the WGAN, the WGAN-GP in Figure 14b performs worse than the WGAN with 25 responses stating that the camouflage produced was of poor quality and over half of the responses stating that the participant saw the camouflage immediately. In fact, only one participant responded on a single occasion that they did not spot any camouflage, down from 5 with the original WGAN. This suggests that the camouflage generated by the WGAN-GP is not as effective as that of the WGAN when presented to human participants, whereas the WGAN-GP is capable of producing images of better colour similarity.



(a) A response heatmap for the WGAN (b) A response heatmap for the WGAN-GP

Fig. 14. Response Heatmaps for the WGAN and WGAN-GP

6.3 Project Evaluation

Overall, I believe that this project has been successful in the aims set out at the beginning, with all four mandatory functional requirements and the single optional stretch functional requirement being met (Sections 2.1.1 and 2.1.2); additionally, all four evaluation criteria from Section 2.1.3 are used. However, the two non-functional requirements were not met to the fullest extent as the maximum limit of 10,000 epochs or 24 hours of training time was never reached during any of the training regimes (Section 4). Moreover, I decided upon using an existing survey service instead of creating my own as, due to extended time spent developing the original GAN, I could not create the survey software with complete confidence that the results would not be affected by issues imposed by such a short time limit. Multiple GAN prototypes had been created and tested, but failed to train; issues such as the failure to train can normally be accounted for by modifying hyper-parameters, but these prototypes did not respond to this strategy and failed to train at all.

Furthermore, once all four implementations were finished, there was not extra time to extend training to arbitrary size images or for camouflage using different datasets. I would not consider this a failure of this project as these were purely optional objectives in the case that I found myself with additional time.

Given this, I believe that this project has been successful in the overall aim of generating camouflage using Generative Adversarial Networks; four individual implementations of the GAN technique have been trained, tested and evaluated and demonstrate the fact that the Generative Adversarial Network is capable of producing effective camouflage. Although the original GAN is shown to produce more similar images in regards to image and colour similarity (Section 6.1), the human observation tests show that the WGAN produces the most amount of successful camouflage samples (where the participant struggles to or does not at all see the camouflage). Moreover, I believe that the human observation test is more representative of the effectiveness of camouflage and thus think that the WGAN has produced the best example of camouflage of the four techniques.

7 CONCLUSION

The aim of this project was to generate Camouflage using Generative Adversarial Networks, exploring the effectiveness of various techniques when applied to this problem. In this paper, I have discussed the background knowledge and fundamental bases of the Generative Adversarial Network, as well as the set of functional and non-functional requirements that this project was to meet. All functional components of the project (being the four GAN techniques) were delivered on time and to a satisfactory standard; all were tested to a high standard and extensively trained to ensure that results were of as high a quality as possible.

Moreover, the design stage of this project covered the main technologies used, the Scene Classification Dataset [16] and a high-level overview of the architecture of the system was presented and explained in detail; this architecture was followed throughout development to ensure that each stage of the project ran on time and successfully.

Each stage of development was covered in detail, starting with formatting and encoding of the dataset and further expanding into the implementation of each GAN technique. Even given the delay caused by the development of the original GAN, the other three implementations were delivered on time with ample time for implementation of the WGAN-GP.

The evaluation criteria stated in Section 2.1.3 were all used and implemented successfully, providing great insight into both the successes of the WGAN and WGAN-GP, as well as the failures of the GAN and DCGAN. As seen in Section 6, the image similarity measures suggest that the GAN and DCGAN produce images of similar raw pixel values, whereas the human observation tests suggest that the WGAN and WGAN-GP both produce camouflage that is more successful at blending with the background. Furthermore, the DCGAN is shown to experience mode collapse quite significantly towards the end of training and the original GAN is shown to produce very low quality camouflage.

7.1 Further Research

This project shows that the generation of camouflage is certainly possible with the use of Generative Adversarial Networks, but there are many avenues research could expand into.

Firstly, one might explore the generative capabilities of the GANs when applied to different datasets, or perhaps by using one of the many other subsets of the Scene Classification Dataset [16]. Datasets which contain more man-made structures could propose an interesting challenge, as the variety of colours in man-made settings is usually much higher than that of a forest or natural landscape.

One might also extend the generation of camouflage to include further advanced GAN techniques, such as the use of Image Inpainting techniques. Image Inpainting is the task of modifying an image in an undetectable manner, often with the goal of restoring missing sections of an image. One example of Image Inpainting is the Context Encoder, which uses the $\mathcal{L}_2$ loss and the adversarial loss seen in Equation (1) to predict missing regions of an image [29].

In summary, this project has explored and successfully shown that a variety of Generative Adversarial Network techniques can be used to generate camouflage, varying in effectiveness. Camouflage generation is an area in which the GAN could benefit and is something I look forward to see improve, whether in my own work or in others.

REFERENCES

[1] M. A. Hogervorst, A. Toet, and P. Jacobs, "Design and evaluation of (urban) camouflage," in *Infrared Imaging Systems: Design, Analysis, Modeling, and Testing XXI*, International Society for Optics and Photonics, vol. 7662, 2010, p. 766 205.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[3] S. Mohamed and B. Lakshminarayanan, "Learning in implicit generative models," *arXiv preprint arXiv:1610.03483*, 2017.

[4] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F.-Y. Wang, "Generative adversarial networks: Introduction and outlook," *IEEE/CAA Journal of Automatica Sinica*, vol. 4, no. 4, pp. 588–598, 2017.

[5] J. Nash, "Non-cooperative games," *Annals of mathematics*, pp. 286–295, 1951.

[6] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[7] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[8] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, PMLR, 2017, pp. 214–223.

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[10] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[11] A. Mordvintsev, C. Olah, and M. Tyka, "Inceptionism: Going deeper into neural networks," 2015.

[12] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[13] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International conference on machine learning*, PMLR, 2010.

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[15] L. Talas, J. G. Fennell, K. Kjernsmo, I. C. Cuthill, N. E. Scott-Samuel, and R. J. Baddeley, "Camogan: Evolving optimum camouflage with generative adversarial networks," *Methods in Ecology and Evolution*, vol. 11, no. 2, pp. 240–247, 2020.

[16] N. Bharathi. "Scene classification dataset." (2018), [Online]. Available: https://www.kaggle.com/nitishabharathi/scene-classification (visited on 11/22/2021).

[17] C. J. Lin, C.-C. Chang, and B.-S. Liu, "Developing and evaluating a target-background similarity metric for camouflage detection," *PLoS One*, vol. 9, no. 2, e87310, 2014.

[18] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE signal processing letters*, vol. 9, no. 3, pp. 81–84, 2002.

[19] C. R. Harris, K. J. Millman, S. J. Van Der Walt, *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[20] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, *et al.*, "Scikit-image: Image processing in python," *PeerJ*, vol. 2, e453, 2014.

[21] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[22] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[23] M. Abadi, A. Agarwal, P. Barham, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[24] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM international conference on Multimedia*, 2010, pp. 1485–1488.

[25] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?" *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[26] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, Citeseer, vol. 30, 2013, p. 3.

[27] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[29] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2536–2544.