Imperial College London

Department of Computing

Transfer Learning for Robot Control with Generative Adversarial Networks

Author:
Jonathan Heng

Supervisor:
Dr. Edward Johns

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 2017

Abstract

Training a robot in reality is more resource intensive as compared to training it in a simulator. Annotations are easily available in simulators, data can be generated much more quickly, and training speeds are faster. Hence, it would be ideal to be able to use simulated data to train a robot and deploy it successfully in the real world. However, even while physics simulators are improving and able to render increasingly realistic images, there still exists a difference between real data and simulated data. This difference is also known as the reality gap.

The reality gap is a classic transfer learning problem, where the source domain X is simulated data and the target domain Y is real world data. To bridge this gap, we explore the idea of using image translation to create source-target domain pairs X-Y. The idea of image translation is to transfer the styles of a particular domain to another and various related works have been done with use of generative adversarial networks, where the aim is to learn a mapping $G : X \rightarrow Y$. This paper presents a novel approach of training generative adversarial networks to generate a paired target domain $G(X)$ to a source domain image X by minimizing the task loss of the generated target image on the same task as its paired source image.

We then evaluate methodologies under the condition of limited source-target pairs to train a robot arm to reach for a cube in the target domain. With the use of an image translation model to generate a large amount X-$G(X)$ pairs, we find significant performance improvements ranging between 20-50% compared to just using limited X-Y pairs in various source-target domain pairings.

Acknowledgments

I would like to acknowledge my supervisor, Dr. Edward Johns, for all his time and patience in providing valuable advice on the thesis as well as guidance on future pathways. I am also grateful to my family and friends for all the support throughout the years. Finally, I would like to give special thanks to my parents, who gave me the wonderful opportunity to be here.

Contents

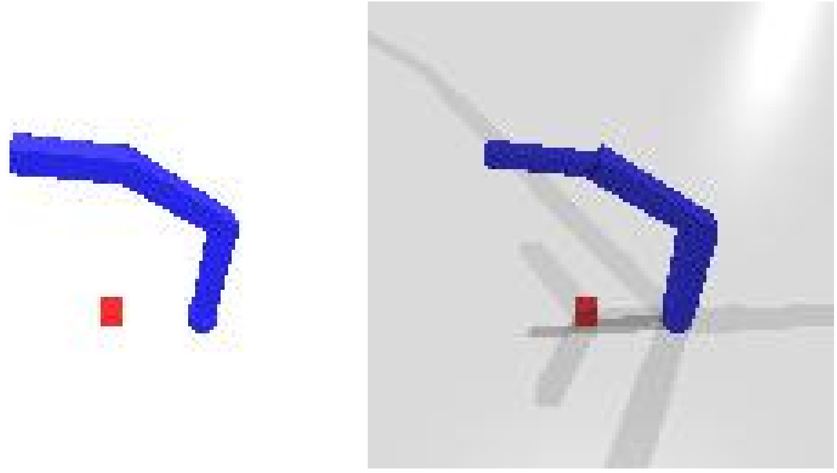
Chapter 1

Introduction

1.1 Motivation

One of the key hallmarks of human intelligence is the capacity to adapt to new situations based on similar past experiences. For example, a tennis player is likely to be able to pick up other racket sports much more easily than a person who has no experience in racket sports. The ability to reuse knowledge learnt from solving one task on a separate but related task is present in all humans. In other words, being able to transfer knowledge between different task domains is a highly desirable trait of intelligence. However, it remains extremely difficult to come up with a general algorithm to allow a computer to effectively reuse the learnt knowledge across a range of different yet similar tasks. The objective of transfer learning is to effectively reuse past knowledge for new tasks. The potential benefits are far-reaching and this paper aims to study this topic in the context of training robots in simulation and transferring the knowledge over to a robot in the real world.

The ideal goal is the capability to train a robot completely in simulation and successfully deploy it in the real world. The desire to achieve transfer learning in this particular domain is due to the numerous obstacles faced when training a robot in the real world. Firstly, there is the issue of safety. The robot can potentially go out of control during the training stage, resulting in damage to surrounding properties, itself, or even humans. Secondly, large amounts of time have to be put in by humans to supervise the robot and create relevant training data. Imagine the case of training a robot to pick up a cube. To create training data of images of objects on a table, a human has to manually set up the scene. When training the robot, a human needs to present to supervise it due to safety concerns and to reset the scene for the task. Furthermore, simulation environments often come with the capability of providing annotations to the training data (e.g. joint states, object orientation/pose etc.). These labels are expensive to obtain in reality. On the other hand, these issues are circumvented in a simulated environment. There are no safety concerns, simulation time can be sped up during training, and large amounts of annotated training data can be generated programmatically. Hence, if a robot is able to effectively use the knowledge learnt in a simulated environment in the real world, training in the real world can be reduced dramatically or even rendered completely unnecessary. This problem of transferring knowledge learned in a simulated world to the real world is one that transfer learning aims to solve. More generally, this can be seen as a problem of transferring knowledge from a source domain to a related target domain.

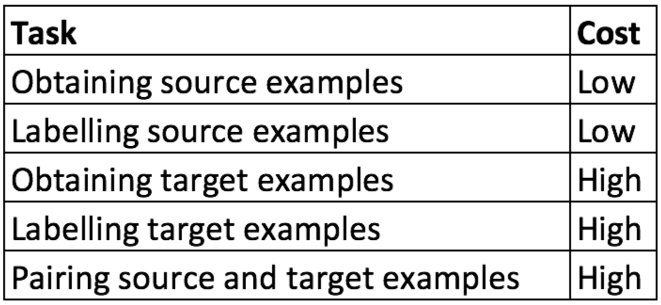


(a) Source domain (b) Target domain

Figure 1.1: A sample source-target domain pairing created in simulation. The source domain is a simple setup of a 3DOF arm and a cube, while the target domain adds lighting effects, slight colour changes, and slight shape deformations to simulate typical differences between simulation and reality.

This project studies the transfer of knowledge from a source to a target domain in a simulated setting. Figure 1.1 shows a possible source-target domain pairing that can be created in simulation. Since the project is motivated by the problem of training a robot in a simulated environment on a task and successfully performing the task in reality, the source and target domains will be designed with the respective characteristics found in simulation and reality.

1.2 Objectives



Task	Cost
Obtaining source examples	Low
Labelling source examples	Low
Obtaining target examples	High
Labelling target examples	High
Pairing source and target examples	High

Figure 1.2: Cost of tasks for obtaining relevant training data given a source-target domain pairing of simulated world-real world

Given a source-target domain pairing of simulated world-real world, the costs of obtaining relevant training data are described in Figure 1.2. It is desirable to minimize the need of high cost tasks and this project aims to study the transfer of knowledge from a source domain to a target domain with these characteristics. Reducing the dependency on target data will be an aim of the project. The specific problem for testing and evaluating of transfer learning methodologies is to control a 3-degree-of-freedom (3DOF) robot arm to move towards a cube. This will be formulated as a regression problem, where joint velocities will be predicted using an image input. However, it remains difficult for any algorithm to learn a regression task in a fully unsupervised fashion. On the other hand, semi-supervised learning combines unsupervised and supervised methods, which suits our aim of reducing the dependence on target domain data. For training and evaluation of methodologies, source-target domain pairs will be created in simulation. This can be done by creating applying a transformation to the source domain image directly, or adjusting the original source domain scene to create a target domain scene. The objectives can be summarized as follows:

- **Setting up the simulated environment:** The simulator of choice for this project is V-REP [1]. The simulator will act as a training data generator and an environment for the testing of models.

- **Control a 3DOF robot arm in the simulated environment:** Train a neural network that is able to control the robot using the images generated from the simulator. Design performance metrics to evaluate the performance of various models.

- **Transfer knowledge between different simulated environments:** Evaluate transfer learning methodologies from a source domain in simulation to another target domain in simulation. Design training methodologies that reduces the dependence of target domain training examples that are costly to obtain.

1.3 Contributions

The contributions of this paper are as follows:

- a novel dataset for the study of transfer learning for robot control which is easily configurable to simulate various characteristics between simulation and reality
- motivate the use of image translation using generative adversarial networks (GANs) to create image pairs between source and target domains in order to easily generate more target domain data from source domain data
- analyze methodologies of supplementing limited target domain data with ample source domain data to improve performances in target domain
- a novel method in training GANs for image translation by minimizing the task loss of generated target domain images

1.4 Outline

This section gives an outline of this paper with a brief summary of each chapter.

- **Chapter 2** covers background research and related work.
- **Chapter 3** details the process behind generating relevant training data for the problem at hand, including the use of the simulator V-REP and processing techniques on the training data.
- **Chapter 4** discusses training a neural network to perform regression for robot control directly from images. The evaluation set-up and metrics are described. Training and test results in conditions with full target domain data and limited target domain data are evaluated.
- **Chapter 5** explores the use of generative adversarial networks (GAN) for image translation purposes. The use of CycleGAN from [2] and self-proposed variations of GAN are tested and evaluated.
- **Chapter 6** combines the models from Chapters 4 and 5 to create a transfer learning process. Performance gains from transfer learning on several source-target domain pairings akin to that of simulation-reality are evaluated.
- **Chapter 7** concludes the work with a summary and points out possible future directions of research.

Chapter 2

Background

2.1 Introduction to transfer learning

We now consider the below definitions following from a survey on transfer learning [3].

Definition 2.1 (Transfer Learning)**.** Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, *transfer learning* aims to help improve the learning of the target predictive function $f_T(\cdot)$ in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$, or $T_S \neq T_T$.

Definition 2.2 (Domain)**.** A domain is a pair $D = \{X, P(X)\}$ where X is a feature space and $P(X)$ is a marginal probability distribution.

Definition 2.3 (Task)**.** A task is a pair $T = \{Y, P(Y|X)\}$ where Y is a label space and $P(Y|X)$ is an objective predictive function.

Particular to this project, we will be dealing with the case of transferring knowledge for controlling a robot in a simulated environment to the real world. This is characterized as a case of different source and target domains and different source and target tasks. The source and target domains share the same feature space, X, which are pixel values in an image, but do not share the marginal distribution, $P(X)$. Similarly, the source and target tasks share the same label space, Y, which are the joint velocities, but do not share the same predictive function, $P(Y|X)$.

According to the survey [3], transfer learning problems can be further categorized into three distinct settings: *inductive, transductive,* and *unsupervised*. In the *inductive* setting, source and target labels are available. In the *transductive* setting, source labels are available while target labels are unavailable. For the scope of this project, both *inductive* and *transductive* settings are most relevant and will be investigated in greater detail.

2.2 Neural networks

In recent years, the applications of neural networks have grown explosively. Neural networks achieved state-of-the-art performances in various domains including visual recognition [4], video game playing [5], and natural language processing [6]. Neural networks have proven to be a powerful learning model that is able to learn a wide range of tasks.

Works on transfer learning have been done with the use of neural networks as the model. In [7], a neural network is trained in simulation and fine-tuned with a small number of real-world images to perform a planar reaching task. This was done by setting up a simulated environment to train a robot arm in and approaching it as a reinforcement learning problem. The robot arm's state includes its end-effector position and the joint configuration. Control of robot is done by taking an action from a finite set of actions, which is quantified by a combination of decrease or increase in the joint angles or leaving them unchanged. Instead of directly obtaining the Q-values from the neural network trained on raw pixel inputs as in [5], the authors separated the neural network into two distinct components, one for perception and the other for control. This allowed the networks to be trained in parallel. The perception network will learn to predict the state of the robot arm from raw input pixels while the control network will learn to predict the Q-values from the state of the robot arm as input. Furthermore, instead of utilizing the ϵ-Greedy method for policy search, which is a form of random exploration, a kinematics-based controller is introduced to guide the policy search. Experiments show that training a prediction network from scratch in simulation and fine-tuning a network with a good ratio of images (found to be 75% real and 25% simulated images in the paper) is able to outperform a network trained from scratch using only real-world images. This work shows that a neural network trained in simulation can provide an effective starting point to further fine-tune with a much smaller amount of real-world data. One possible limitation of the above work is lack of generality of the chosen search policy. Since it is designed specifically for a kinematics problem, it is not widely applicable to other problems.

Another interesting approach is to design a neural network architecture that can utilize past knowledge learnt. [8] tackles the transfer learning problem through innovative construction of an architecture, termed progressive networks, that grows as new tasks are added. A network column is first trained for a particular task. Instead of re-weighting the neural network when being trained on a new task, old weights are frozen and additional network columns are added in parallel to the previous network columns. The old network columns are linked to the new network columns to enable the reuse of previously learnt features. Freezing the weights would also mean that it ensures previously learnt features are not lost when being trained for a different task or on a separate data source. The new network columns are then free to learn new features and combine them with previously learnt features. Experiments compared the stability and performances of fine-tuning versus progressive network approaches and found progressive networks to be more robust and better performing. A possible downside to progressive networks is the need to expand the network. While the paper explores branching out to a couple of new tasks successfully, if there are a large number of new tasks, computational costs will rise significantly.

Neural networks have also proven to be a highly effective feature extractors. In [9], features learnt from a convolutional neural network trained on ImageNet are used to train a SVM to perform classification tasks in different datasets and achieves leading results. This shows the reusability and generality of the features that are learnt by neural networks. In [10], extensive experiments are done to investigate the degree of specificity of features learnt in neural networks. It is found that later layers learn more specific features and more interestingly, transferring part of a network learnt in a separate domain leads to an overall better performance than just training in a single domain. From these results, transfer learning when applied appropriately can even be used to give a boost to performance.

2.3 Robot control systems

Traditional approaches to controlling a robot [11] relies on designing multiple modules such as perception, planning, and motor control that processes input sensor signals and outputs a value to actuators. However, such approaches tend to require a significant amount of manually designed components to obtain a low-dimensional feature representation of the information at each step. Ideally, we would like to avoid using hand-crafted features as information tends to be lost. Intuitively, a model that is able to directly utilize input information and output an action can possibly perform better since all information is built and retained by the model.

In [12], the authors present a method of training robot control policies that predicts an action directly from visual observations by using deep convolutional neural networks. This falls in line with the ideology of having a single end-to-end model that trains all modules of the robot control system jointly. While the overall methodology covered in the [12] is largely reinforcement learning centric, it shares similarities with this paper in that supervised learning samples are used to guide the policy search. In Chapter 4, we will cover the use of a physics simulator to provide supervised learning samples of a trajectory. The difference is that the problem in this paper is formulated as a regression task instead of a reinforcement learning problem.

2.4 Distance between distributions

In the case of different source and target domains, either $X_S \neq X_T$ or $P(X_S) \neq P(X_T)$. Consider the latter case. Clearly it is not a trivial task to train a model to recognize data generated from separate probability distributions. One approach is to incorporate statistical measures of distances between probabilities during the training of the model. This was done in [13] by minimizing the Maximum Mean Discrepancy (MMD) as part of the loss function when training a neural network to perform object recognition and detection tasks. In this case, the statistical distance that the neural network is trying to minimize is between the features extracted from the source images and target images. The authors were also utilizing a two stream architecture, with one network being trained on the source domain and the other being trained on the target domain. This loss function encourages both networks to learn to recognize a similar set of features. In other words, these learnt features can be seen as domain invariant since they are effective in both domains.

The MMD has many properties that make it a good choice for a probability metric and we shall consider its empirical estimate as taken from [14].

Definition 2.4 (Maximum Mean Discrepancy). Let p and q be distributions defined on a domain Z. Let the observations $X := \{x_1, ..., x_m\}$ and $Y := \{y_1, ..., y_n\}$ be drawn independently and identically distributed (i.i.d.) from p and q respectively. Then we define the maximum mean discrepancy's empirical estimate as

$$MMD[X,Y] = \left[\frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} k(x_i, x_j) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j) + \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k(y_i, y_j) \right]^{\frac{1}{2}} \quad (2.1)$$

where, $k(\cdot, \cdot)$ is a kernel function. Ideal choices for the kernels are the Gaussian and Laplace kernels for their universal property as further discussed in [15].

Computation time of the above empirical estimate costs $O((m+n)^2)$ time. More efficient approximations of the MMD can be found in [16]. There are other probability metrics that exists and can be considered, such as Kullback-Leibler divergence, Kolmogorov metric, and χ^2 distance. A comprehensive study of these metrics can be found in [17].

2.5 Domain shift

One of the major problems that transfer learning attempts to solve is domain shift. For example, a robot arm can be trained to pick up a red cube. However, the learned model will often break down when the environment changes such as when the colour of the cube is different or when the lighting effects are different. A simple 2-dimensional dataset can be used to illustrate this effect. In Figure 2.1, domain 1 is generated by a 2-dimensional Gaussian variable with mean 0 and variance 1. Domain 2 is generated by the datapoints in domain 1 with a shift in the y-axis by a value of 10. In this case, the x-dimension represents a domain invariant feature. Imagine trying to learn a function for these datapoints. Being able to identify the x-dimension in this case would be much more helpful in solving the problem since it provides the same information for both domains, whereas the y-dimension is not helpful for learning a single function mapping for both domains simultaneously. Identifying this feature that describes data from separate domains is far from a trivial task and we will discuss some works in the next paragraphs. Other terms in literature describing a similar phenomenon include domain adaptation and covariate shift.

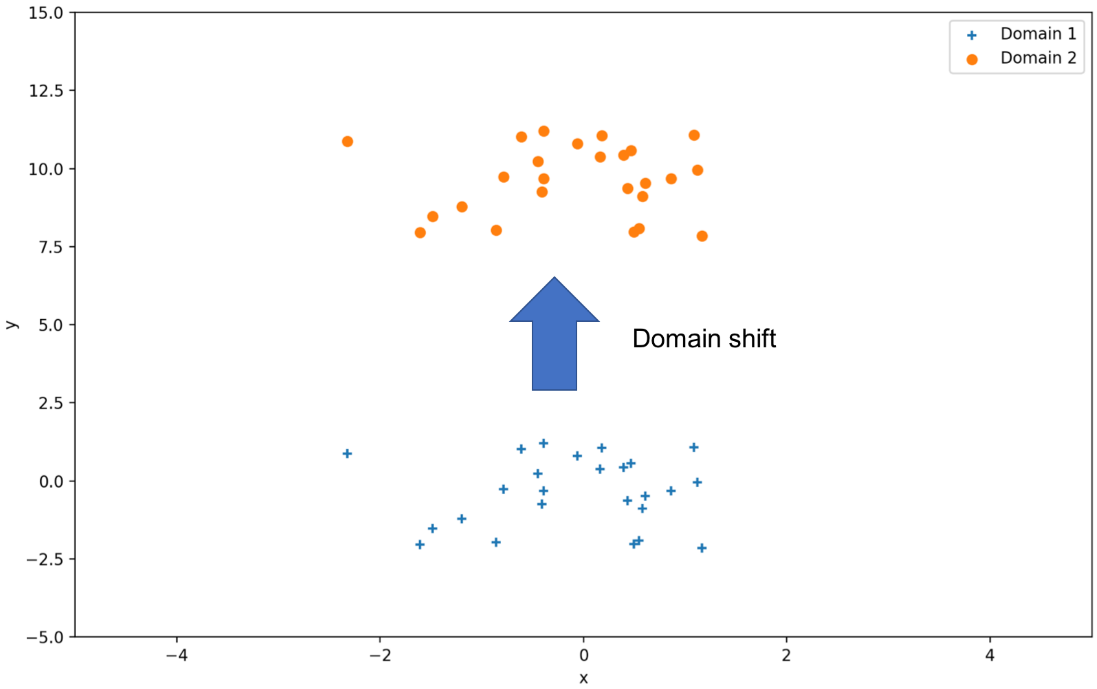


Figure 2.1: 2-dimensional visualization of domain shift

One method of approaching the problem of discrepancies between the source and target data distributions is to learn a feature representation which is unable to distinguish between the two domains. This is known as domain confusion. In [18], the negative cross entropy loss is added to the loss function. In addition to training the neural networks for the task, a domain classifier is trained to recognize which domain the image originates from. This loss function is minimized when the domain classifier is maximally confused. Hence, training the model with this loss function will encourage the model to learn a feature representation that maps the data from different domains into a similar feature space.

A different take on bridging the gap between the simulated environment and the real world is through domain randomization [19]. This approach attempts to introduce a large amount of variability in the simulated environment such that the real world appears to be just another variation. In this paper, the aim is to train an object detector on simulated images and test its effectiveness when giving a scene from the real world. Aspects of the simulated environment that were randomized include the number and shape of distractor objects present, position and texture of objects, camera position and orientation, lighting, and type and amount of noise added. Even without the need of any real-world training data, the object detector was able to perform well. A possible explanation for the effectiveness of the method is that the neural network model learned a set of features that are invariant to cope with large amount of variability in training data.

2.6 Domain alignment

A recurring problem in machine learning is the difficulty in obtaining annotated real world data. Ideally, we would like a model to be completely trained on easily generated synthetic data and perform well when thrown into the real world. Unfortunately, this is far from a trivial task. One way to overcome the lack of annotated real world data is to have some form of automatic annotation. In [18], a weak pairwise constraint method was proposed. This method attempts to pair annotated synthetic data with an unannotated real world counterpart by finding a nearest neighbouring source image for each target image. The distance between images can be measured by the Euclidean distance between the set of features extracted by a convolutional neural network. This distance can then be added into the loss function as a pairwise loss.

Both the domain confusion loss and pairwise loss were evaluated in [18]. It is found that pairing unlabelled real world data with simulated data is able to outperform just having real world labelled data.

2.7 Style transfer between image domains

The visual world is often characterized by style. What is stylistically coherent in reality might look completely out of place in a cartoon world. Humans are able to understand and find correspondence between similar object across visually distinct domains. For example, while a real car and a cartoon car may possess distinct visual styles, humans easily understand that these two objects share the same semantic meaning. Recent works [20, 21] presented methods of transferring the visual styles and characteristics from an image domain to other images.

Generative adversarial networks, or GANs in short, are a class of neural networks that estimates a generative model through training a generative model and a discriminative model in an adversarial process [22]. It is shown to be able to generate MNIST-like handwritten images from a Gaussian input. This shows the viability of utilizing GANs as a method of generating images from a desired domain. This idea has been furthered in works [2, 23] which used generative adversarial networks to generate source-target domain image pairs. These model uses a source domain image as input and obtains a paired target domain image as output. The CycleGAN model in [2] is of particular interest since the training process does not require explicit paired source-target domain examples. This also removes the need for labelled target examples since labels can be transferred from the respective source examples. The CycleGAN model will be further investigated in Chapter 5.

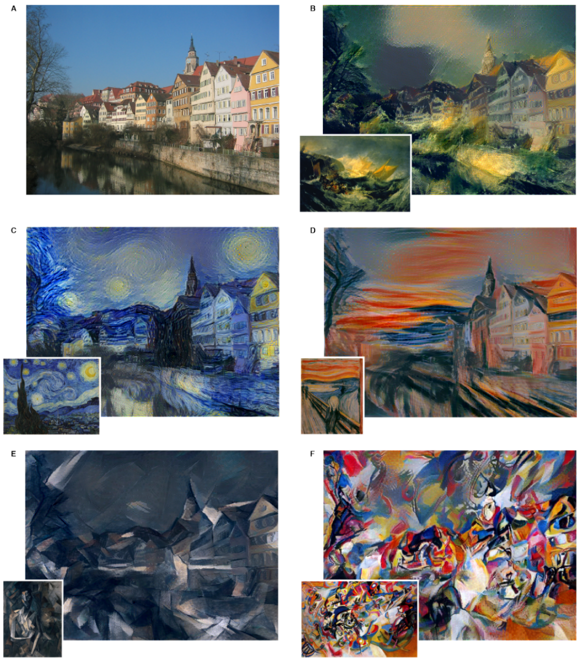


Figure 2.2: An image visualized in the styles of famous works of various artists. Top left image (A) represents the original image, while the other images represent the output after transferring the style of the associated artist's work. Source: [20]

Chapter 3

Generating Training Data

The problem of interest here is to predict joint velocities for a 3DOF arm from an input image. This can be framed as a regression problem and the corresponding training dataset are images annotated with corresponding joint velocities. In this chapter, we cover the use of a simulator to create a scene of a 3DOF robot arm and a cube and utilize it to generate training data.

3.1 Data sources for transfer learning

In other studies of transfer learning between a simulated environment and the real world, custom training data sets are created. In [19], a physics simulator is used to generate various images of objects on a table and object positions can be obtained within the simulator. This is then used to train on image detector and tested on real world images with no additional training. In another work [18], synthetic images are created using the Gazebo simulator and pose annotations can be obtained through the simulator.

3.2 V-REP

For the purposes of this project, a custom training dataset will be created. Training and testing will be done completely in simulation. Target domains are created to simulate typical differences between a simulated source domain and a real target domain. A simulation environment that is able to create a training dataset of images with corresponding joint velocities has to be chosen. Virtual Robot Experimentation Platform (V-REP) [1] is a robot simulator that fulfils the requirements of the project. It possesses an inverse kinematics module that is able to calculate a list of joint states, where following this list of joint states is equivalent to moving a point on a robot arm to a target position along a straight path. This list of joint states can be converted into corresponding joint velocities as labels. The platform also possesses a remote API in various languages including Python.

3.3 Simulation scene set-up

This section will describe the various components that are in the V-REP scene for the 3DOF arm. References to specific objects in the scene will be marked with quotations, e.g. '3dof-arm'. A screenshot of the simulation scene can be found in Figure 3.1. For further details on how to properly set up a scene, V-REP provides their own tutorials on its website which are much more comprehensive.

Firstly, a custom model of the arm is created. 4 simples shapes are needed to create the arm: a cylinder 'base', followed by 3 cuboid 'links'. These shapes can then be linked together with 'joints'. Next, a simple red 'cube' can be added to the scene.

To obtain images of the arm moving towards the cube, the inverse kinematics module in V-REP can be used. This creates the behaviour of moving a designated tip to target in a straight line. A tip-to-target pair has to be created. A dummy object, 'target', was created and placed under the 'cube' in the scene hierarchy. Similarly, a dummy object, 'tip', was added at the end of the arm model, 'link3'. At this point, a tip-to-target relationship can be assigned between this pair of dummy objects, as indicated by the red arrows linking them under the scene hierarchy. For the inverse kinematics to start working, an inverse kinematics group has to be added under the inverse kinematics calculation module.

Vision sensors which act as cameras are also available. They can be added and positioned at various angles to capture different viewpoints of the scene. As seen in the Figure 3.1, there are 4 vision sensors that indicate 4 different viewpoints of the arm and the cube, namely front, side, top, and diagonal. The resolution of the images can be controlled. Images in this paper are of 128x128x3 in dimension.

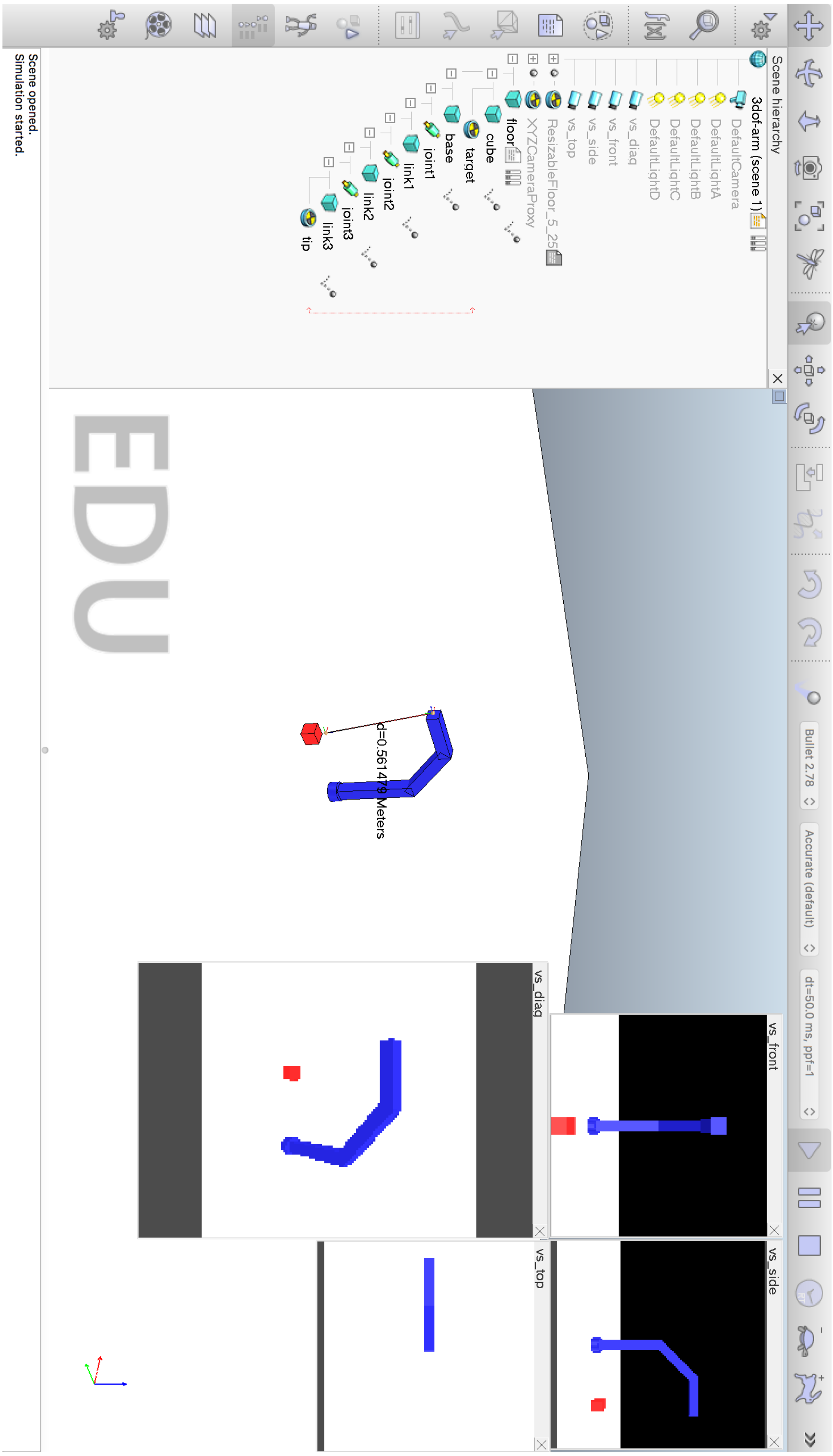
3.4 V-REP remote API

V-REP provides a remote API where other various languages can be used to communicate with V-REP's simulation environment. The choice of language for this project is Python. Help with setting up the remote API can be found in on V-REP's tutorial website[1].

V-REP provides a large range of functions that enables communication for various actions in the simulated environment. In general, objects in the simulated environment have handles and there are functions that allow the user the obtain handles of the objects. Different objects have different properties which can be controlled. For example, images from vision sensors can be obtained, joints have states that can be obtained and changed, and objects have positions that can be obtained and changed. A detailed list of the remote API functions can be found on the website as well[2]. For

[1] http://www.coppeliarobotics.com/helpFiles/en/remoteApiOverview.htm

[2] http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionListAlphabetical.htm



Scene opened.
Simulation started.

Figure 3.1: V-REP scene of 3DOF arm and a red cube

more advanced functions that are not available in this list, the simxCallScriptFunction provides the flexibility of allowing the user to implement any kind of remote API function.

By utilizing these remote API functions, the simulated scene can be prepared and manipulated completely in Python. For the training dataset, variability in the initial cube positions and initial joint positions are desired. After some initial testing of the performances of trained neural networks on various datasets, it is found that creating a dataset in a grid-like manner (e.g. initial cube x position ranges from [0, 1] and taking 5 points between this range will include the points [0, 0.25, 0.5, 0.75, 1]) and mixing it with uniform random samples lead to the best results.

3.5 Training images

Using the remote API, the inverse kinematics module is able to provide a list of joint states. This list of joint states can be followed iteratively to create the behaviour of the robot arm moving towards a cube. The size of this list can be specified and this size corresponds to the number of steps of moving the robot arm from the initial position to the final position. At each step of following this list, the image can be obtained in Python through the remote API. The joint states are also obtained and recorded on a text file. Each image is then exported to be saved on disk. The choice of export format is JPEG. Even though it is not a lossless compression, it is highly space efficient and extremely high fidelity images are not necessary to train the robot arm successfully.

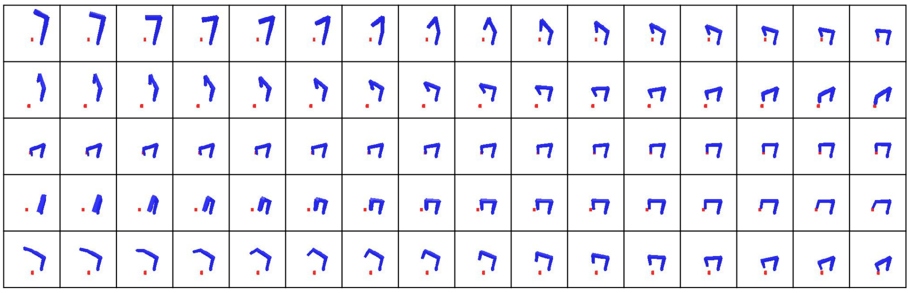


Figure 3.2: Training images. Each row represents a set of images comprised of 16 steps. From left to right, the robot arm gradually moves towards the cube.

3.6 Labelling images

Since the objective is to perform control of the robot as a regression problem, the joint states have to be transformed to joint velocities. To appropriately achieve the effect of moving the arm at each step, the joint velocities label should be the difference between the next joint state and the current joint state. For the final joint state, since there is no next state, the joint velocities labels will be set to 0.

$$jointVelocity[i] = jointState[i+1] - jointState[i] \text{ if } i \neq finalStep$$
$$jointVelocity[i] = [0, ..., 0] \text{ otherwise}$$

Joint velocity labels are further processed by normalizing and multiplying by a damping factor. Normalization is done by the dividing the joint velocities with the absolute norm. The damping factor used is the distance between the tip and target position, which can be obtained at each step through the remote API. The motivation of using a damping factor is to encode knowledge of distance in the labels and create the effect of taking bigger steps while the arm is further from the cube and slowing down as it approaches the cube.

3.7 Domain transformation

To create distinct target domains, transformations can be done to the original image. Simple transformations include gamma correction and multiplying the colour channels with a factor. An example of a more complex transformation is to put an image through a edge detection algorithm such as canny edges and create an edge-version of the original image. Other ways include directly adjust the colours, shapes, and sizes of the objects in the scene and generating new images.

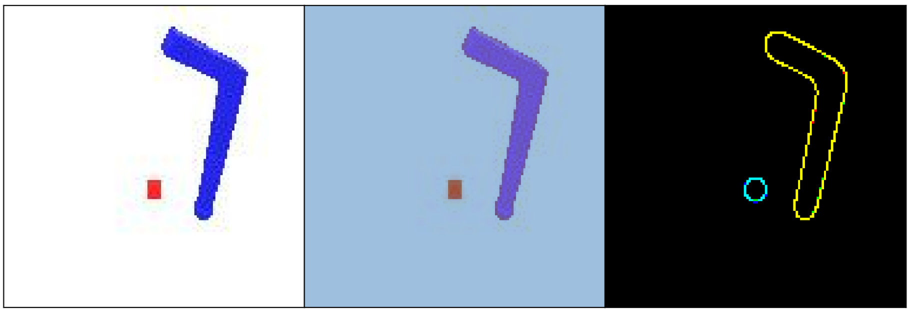


Figure 3.3: An image of 3DOF arm in various domains. **Left**: Original. **Center**: Tinted image by multiplying the colour channels by [0.25, 0.5, 0.75] respectively after preprocessing to values between a range of [-1, 1]. **Right**: Image created by putting each colour channel through a canny edge detector and recombining the colour channels.

Chapter 4

Control of Robot Arm from Images

Convolutional neural networks have proven to be highly effective models in image recognition and other image related tasks and related works such as [7, 19] showed the viability of this model for application in transfer learning. Hence, convolutional neural networks will be trained to predict joint velocities given an image of a robot arm.

Under the transfer learning problem, we would like to study the performances of neural networks in both source and target domains under different modes of training. Firstly, the fully supervised scenario where all labels and pairings are assumed to be available will be studied. This will set the performance benchmarks. Then, we will study methods of training that lead to better performances under the scenario of target domain limited data.

4.1 Evaluation set-up and metrics

An appropriate evaluation setting has to be designed to test the effectiveness of the trained neural networks. The same V-REP scene used for generating training data will be used to evaluate the performances of the neural networks. During test time, at every time step an image will be obtained from the vision sensor in the scene and passed through the neural network. Then, the predicted joint velocities will be applied to the robot arm. Each episode starts with a random cube position and joint position and will be played over 100 time steps.

The following metrics will be used to evaluate the performance:

- **Distance**: Both minimum distance and final distance from the tip to target point will be recorded. Since we want the arm to slow down and stop near the object, the final distance is a better metric of performance since the arm can reach a good minimum distance through random movements and eventually move away from the cube. The overall mean and standard deviation of these distances played over multiple episodes are recorded.

- **Success rate**: A distance threshold will be used to decide which episodes are considered successful. If the minimum/final distance for an episode falls below this threshold, that episode is considered successful.

To ensure fairness when evaluating different models, the same test set will be used. The test set ensures that every episode is loaded from an identical list of starting joint and cube states and is composed of 100 episodes.

4.2 Fully supervised learning

In the fully supervised learning scenario, we assume that all labels are available for both source and target domains and all examples are paired.

4.2.1 Models

- **Source only**: Train on source domain images only.

- **Target only**: Train on target domain images only.

- **Mixed**: Train on both source and target domain images.

- **Separate feature extractor**: Train separate feature extractors on each domain and use a shared regressor head. Target and source domain images are paired and a L2 loss between the feature representations of the source and target domains is minimized.

The first 3 models share the same neural network architecture and a figure of the neural network architecture can be found in Figure 4.1. This is a relatively straight forward convolutional neural network architecture. Given a source domain X with images x_i and labels ϕ_i^x, the neural network is trained to learn a mapping function $f_X : X \to \phi^X$. The training process involves minimizing the following cost function:

$$L_{task}(X, \phi^X) = \frac{1}{N} \sum_{i=1}^{N} (f_X(x_i) - \phi_i^x)^2 \tag{4.1}$$

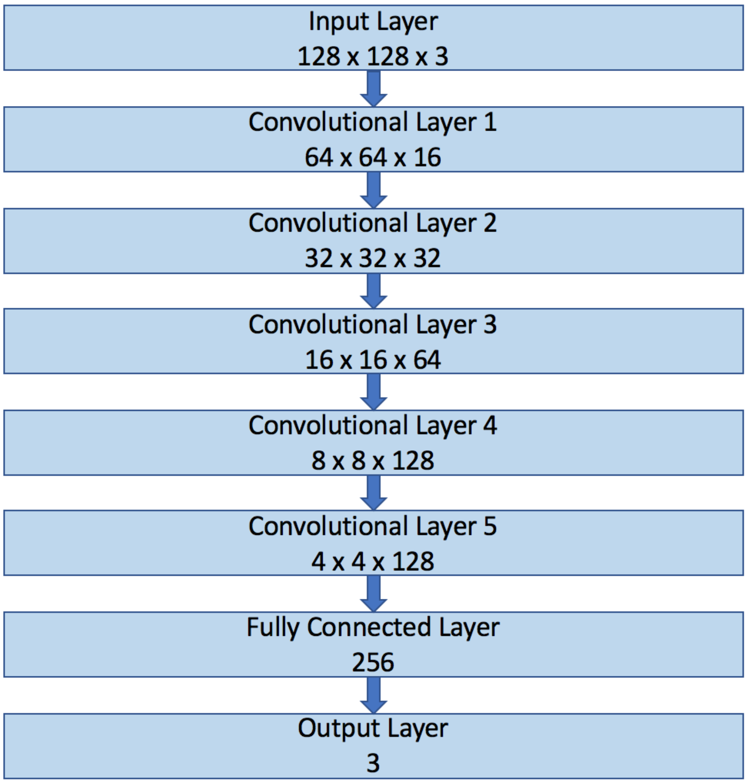


Figure 4.1: Neural network architecture for the first 3 models. Each convolutional layer uses filter of 3x3 size and stride of 2.

Having separate feature extractors is inspired by the work in [13] and a figure of the neural network architecture can be found in Figure 4.2. Neural networks can be seen as a method of extracting image features as discussed in [10]. By treating neural networks as feature extractors and adding a loss that penalizes the Euclidean distance between the feature layers, it encourages the separate feature extractors for distinct domains to map pairs of images to the same feature space. The layer before the output layer is chosen as the feature layer in this implementation.

Given a source domain X with images x_i, labels ϕ_i^x and feature layer representation θ_i^x, a target domain Y with images y_i, labels ϕ_i^y, and feature layer representation θ_i^y, the loss between the feature layers can be expressed as follows:

$$L_\theta(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(\theta_i^x - \theta_i^y)^2 \tag{4.2}$$

Overall, the neural network is trained to minimize the following cost function:

$$\begin{aligned} L_{sum}(X,\phi^X,Y,\phi^Y) &= \frac{1}{N}\sum_{i=1}^{N}[(f_X(x_i) - \phi_i^x)^2 + (f_Y(y_i) - \phi_i^y)^2 + (\theta_i^x - \theta_i^y)^2] \\ &= L_{task}(X,\phi^X) + L_{task}(Y,\phi^Y) + \lambda_\theta L_\theta(X,Y) \end{aligned} \tag{4.3}$$

where λ_θ is a constant that controls the importance of the loss between feature layers. It is set to a value of 1 for experiments in this paper.

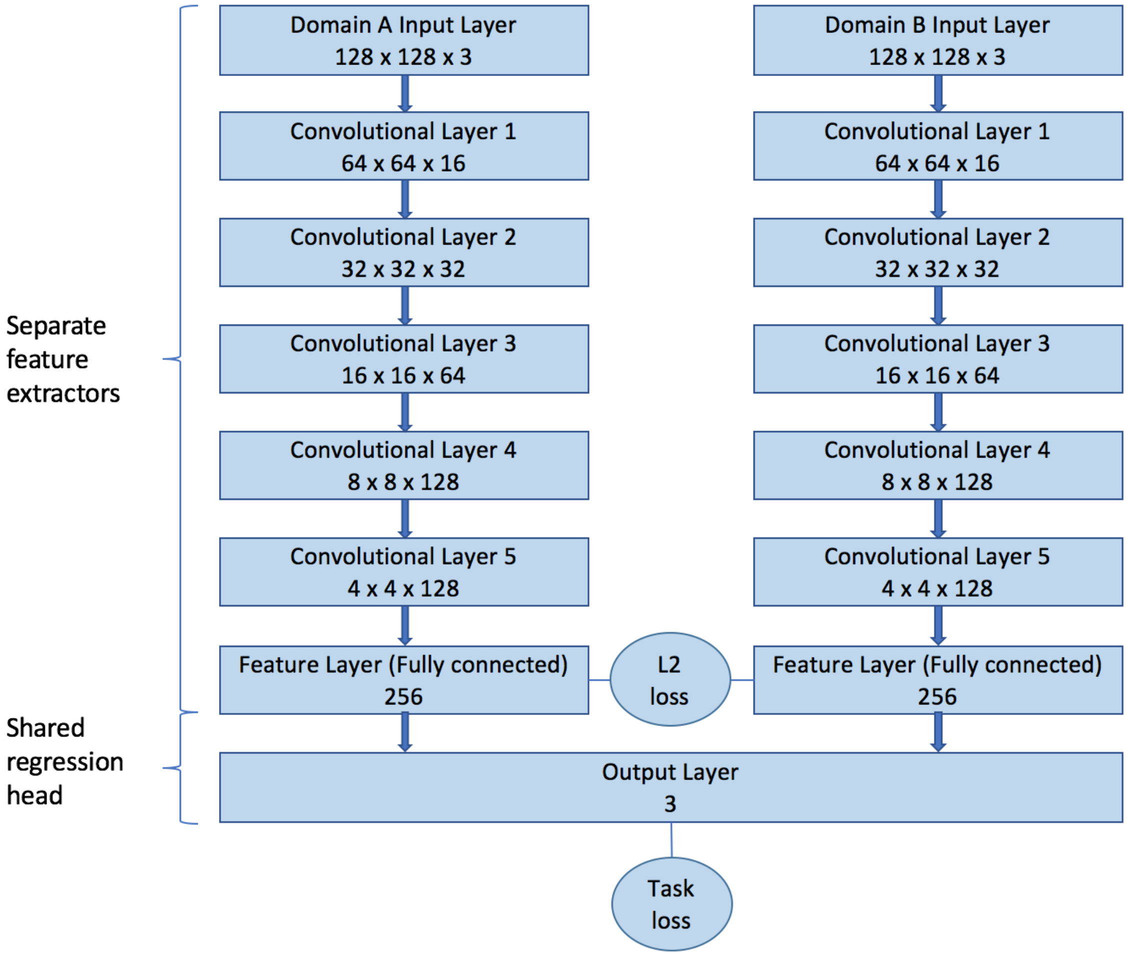


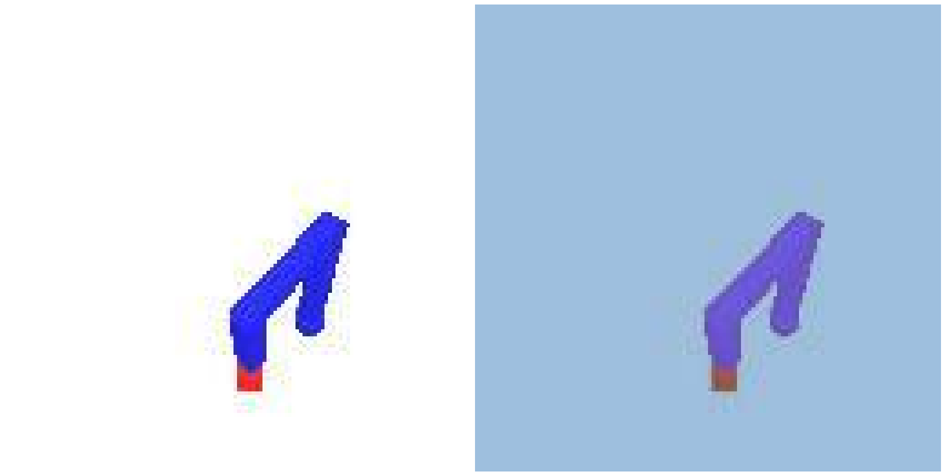
Figure 4.2: Neural network architecture for method separate feature extractors. Input images for separate domains are paired.

4.2.2 Results

These models are trained with the source-target domain pair as seen in Figure 4.3. The target domain image is created by multiplying a constant value of [0.25, 0.5, 0.75] to the RGB channels of the source domain image respectively. The training dataset consists of a total of 22656 images. This is created by saving images of 1416 episodes, 16 steps each episode, of the arm moving towards the cube using the inverse kinematics module in V-REP as described in Chapter 3.

The source only and target only models are trained for 15000 training steps with a batch size of 16 images each. The mixed and separate feature extractor models are trained for 15000 training steps with a batch size of 16 images each for both source and target domains. The images are shuffled before being fed into the neural network. The choice of optimizer is the Adam optimizer with a learning rate of 0.001.

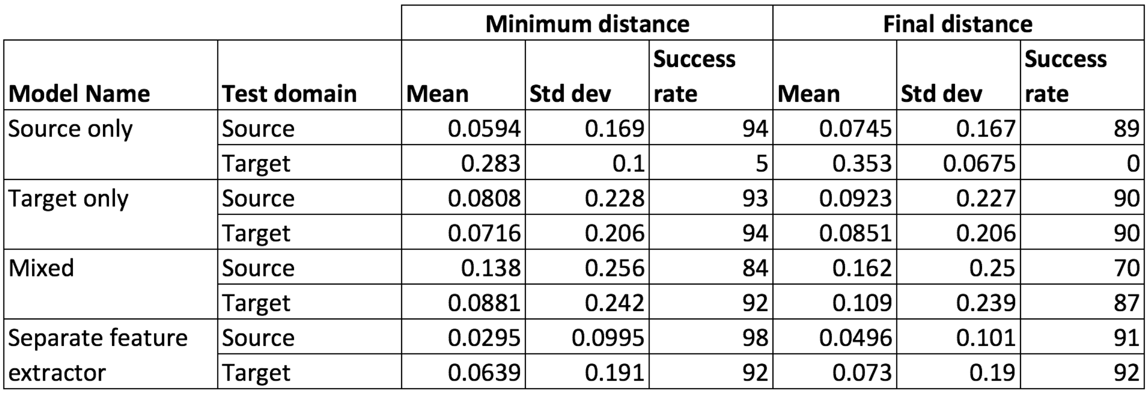
Each of the 4 models are tested in both the source and target domains and the full results can be found in Figure 4.4. An episode is considered a success if the minimum/final distance falls below the chosen threshold value of 0.1.



(a) Source domain

(b) Target domain

Figure 4.3: Sample image from source and target domain for training and testing.



		Minimum distance			Final distance		
Model Name	Test domain	Mean	Std dev	Success rate	Mean	Std dev	Success rate
Source only	Source	0.0594	0.169	94	0.0745	0.167	89
	Target	0.283	0.1	5	0.353	0.0675	0
Target only	Source	0.0808	0.228	93	0.0923	0.227	90
	Target	0.0716	0.206	94	0.0851	0.206	90
Mixed	Source	0.138	0.256	84	0.162	0.25	70
	Target	0.0881	0.242	92	0.109	0.239	87
Separate feature extractor	Source	0.0295	0.0995	98	0.0496	0.101	91
	Target	0.0639	0.191	92	0.073	0.19	92

Figure 4.4: Results of fully supervised models tested on both source and target domains.

As expected, the "Source only" model performs reasonably well in the source domain with a final distance success rate of 89% and fails completely in the target domain with a 0% final distance success rate.

On the other hand, the "Target only" model surprisingly works on both source and target domains with final distance success rates of 90% in both domains. This is highly likely due to the way the target domain was created, which was to multiply the RGB channels of the source domain images with [0.25, 0.5, 0.75] respectively. A simple visual relation is clearly not sufficient since the "Source only" model does not work well in the target domain. It is a coincidence that the chosen transformation led to good performance in the source domain despite training on target domain images only.

The "Mixed" model is trained on both target and source domain images. It fails to perform up to par in the respective domains which the source only and target only models are trained on. Both final distance success rates of 70% in the source domain compared to the 89% of the source only model and 87% in the target domain compared to 90% of target only model are lower. This suggests that training a single neural network to perform a similar task across different domains leads to worse performance, even if the target domain is very similar to the source domain visually and numerically. This is due to the neural network being unable to learn a representation that works for both domains.

The "Separate feature extractor" model trains separate neural networks to extract a similar set of features θ and share a regression layer on top of the extracted feature space. This model outperforms all other models with a final distance success rate of 91% and 92% on both source and target domains respectively. Learning a shared representation across different domains for a similar task improved the overall performance. While these tests are done with complete knowledge of pairs and labels in both domains, it suggests the potential of supplementing training for a target domain task (e.g. in reality) with rich and easily obtainable source domain data (e.g. from a simulator). Another plausible reason for the improved performance is that the training images are saved in jpg, which degrades the quality of the image being saved. Hence, the image passed into the neural network during test time and training time is different. Having the L2 feature layer loss trains the neural network to become more resistant to noisy images.

Overall, the "Separate feature extractor" model works the best and further tests and further studies will be done on variations of this model in the following section.

4.3 Learning with limited data

Since obtaining pairings between real world examples and simulated environment examples and labelling real world examples are expensive, we want to minimize the amount of such work. Ideally, the training can be done in a completely unsupervised manner. However, this remains extremely difficult. A more common approach is to provide a small amount of labels in the target domain. Another approach is semi-supervised learning, where unlabelled target domain data can be utilized. This section will study various models and their effectiveness under the stresses of limited target data.

4.3.1 Models

Each model described in the following will be trained on varying amounts of target domain examples being available. The target only model uses the architecture as shown in Figure 4.1. The other models that train on both source and target domain examples use the architecture as shown in Figure 4.2. Some of the models described here take inspiration from the work in [13], which utilizes the maximum mean discrepancy (MMD) loss and initializing the neural network for the target domain with pretrained weights from a neural network trained in the source domain. The cost function for the MMD can be found in Section 2.4.

- **Target**: Train on target domain images only. All examples are labelled. This model serves as the benchmark for comparison to other models.
- **Paired + L2**: Train from scratch on paired source-target examples and unpaired source examples. All examples are labelled. Paired examples are trained on both task loss and the L2 loss between feature layer. Unpaired source examples are trained on task loss only.
- **Paired + L2 + MMD**: Train from scratch on paired source-target examples and unpaired source and target examples. Unpaired target examples are unlabelled and all other examples are labelled. Paired examples are trained on both task loss and the L2 loss between feature layer. Unpaired source and target examples are trained on task loss and MMD loss between the feature layer.

- **Pre-train + Target**: Pre-train the feature extractor for the source domain and initialize the feature extractor for the target domain with the pre-trained weights. All examples are labelled. Finetune on target examples only. Target examples are trained on task loss only.

- **Pre-train + Paired + L2**: Pre-train the feature extractor for the source domain and initialize the feature extractor for the target domain with the pre-trained weights. All examples are labelled. Finetune on paired source-target examples and unpaired source examples. Paired examples are trained on both task loss and the L2 loss between feature layer. Unpaired source examples are trained on task loss only.

- **Pre-train + Paired + L2 + MMD**: Pre-train the feature extractor for the source domain and initialize the feature extractor for the target domain with the pre-trained weights. Unpaired target examples are unlabelled and all other examples are labelled. Finetune on paired source-target examples and unpaired source examples. Paired examples are trained on both task loss and the L2 loss between feature layer. Unpaired source and target examples are trained on task loss and MMD loss between the feature layer.

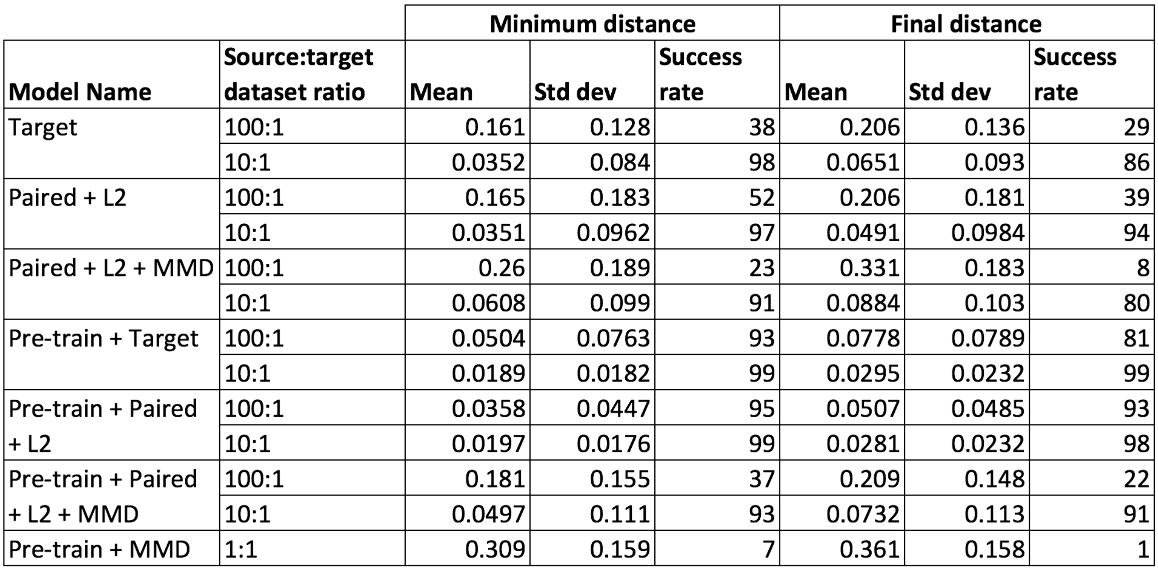
- **Pre-train + MMD**: Pre-train the feature extractor for the source domain and initialize the feature extractor for the target domain with the pre-trained weights. All target examples are unlabelled and all source examples are labelled. Finetune on unpaired source and target examples only. Unpaired source and target examples are trained with MMD loss between the feature layer. Source examples are trained on task loss.

4.3.2 Results

Two different source-target domain pairings will be tested in this section. The first will be a visually similar source-target domain pairing while the latter will have a more significant visual transformation from source to target domain.

The first source-target domain pairing is shown in Figure 4.3. The same source dataset used in Section 4.2.2 is used here as well. The source:target dataset ratios tested are 100:1 and 10:1, with the exception of the unsupervised “Pre-train + MMD” model, which was tested with a source:target dataset ratio of 1:1.

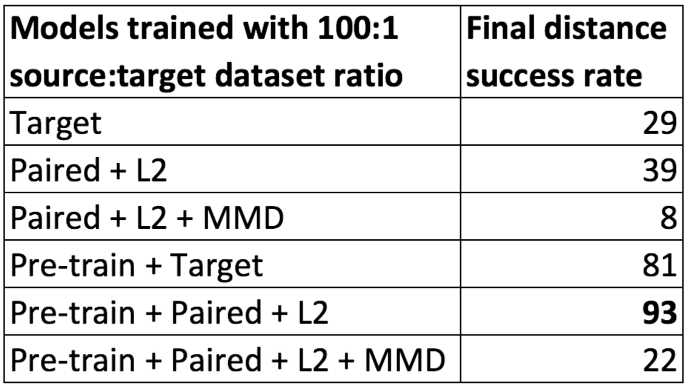
The pre-trained model used is trained for 15000 steps with a batch size of 16 on source domain images only. All models are trained or further trained for 15000 steps with a batch size of 16. The Adam optimizer with a learning rate of 0.001 is used here. Full results can be found in Figure 4.5. A summary of the results for models trained with a source:target dataset ratio of 100:1 can be found in Figure 4.6.



		Minimum distance			Final distance		
Model Name	**Source:target dataset ratio**	**Mean**	**Std dev**	**Success rate**	**Mean**	**Std dev**	**Success rate**
Target	100:1	0.161	0.128	38	0.206	0.136	29
	10:1	0.0352	0.084	98	0.0651	0.093	86
Paired + L2	100:1	0.165	0.183	52	0.206	0.181	39
	10:1	0.0351	0.0962	97	0.0491	0.0984	94
Paired + L2 + MMD	100:1	0.26	0.189	23	0.331	0.183	8
	10:1	0.0608	0.099	91	0.0884	0.103	80
Pre-train + Target	100:1	0.0504	0.0763	93	0.0778	0.0789	81
	10:1	0.0189	0.0182	99	0.0295	0.0232	99
Pre-train + Paired + L2	100:1	0.0358	0.0447	95	0.0507	0.0485	93
	10:1	0.0197	0.0176	99	0.0281	0.0232	98
Pre-train + Paired + L2 + MMD	100:1	0.181	0.155	37	0.209	0.148	22
	10:1	0.0497	0.111	93	0.0732	0.113	91
Pre-train + MMD	1:1	0.309	0.159	7	0.361	0.158	1

Figure 4.5: Results of semi-supervised models tested on test domains with varying amounts of source:target dataset ratio.

Generally, the models performed pretty well when the source:target dataset ratio is 10:1. Final distance success rates range from 80% to 99% with little loss in performance in most cases and surprisingly improvements in some. For example, the “Pre-train + Target” model where 10:1 source:target dataset pairs is used to fine-tune the pre-trained model led to a final distance success rate of 99%, outperforming the results found in the fully supervised scenarios where all source:target dataset pairs are available. A reason is that the dataset is noisy and the randomly sampled source-target pairs were better for performance in the given test set. Also likely is that the target domain is easier in general for the neural network to learn in.



Models trained with 100:1 source:target dataset ratio	**Final distance success rate**
Target	29
Paired + L2	39
Paired + L2 + MMD	8
Pre-train + Target	81
Pre-train + Paired + L2	**93**
Pre-train + Paired + L2 + MMD	22

Figure 4.6: Summary of semi-supervised models tested on test domains with 100:1 source:target dataset ratio.

Incorporating the maximum mean discrepancy (MMD) loss consistently led to worse results. From the results summary in Figure 4.6, training with the MMD loss leads to much worse results than just training the neural network with only the target domain images and on the task loss only. This contradicts the results of [13], where the addition of MMD loss during training led to improved results. Furthermore, in the completely unsupervised test of the “Pre-train + MMD” model where a similar target dataset to the source dataset is provided but unpaired, the neural network fails to learn well. While the MMD loss was not expected to work work the unsupervised setting, it remains unclear why the MMD loss fails to perform well even in the semi-supervised setting but this problem will be left open for future works.

The more prevalent and interesting problem at hand is to test the effectiveness of the models at bridging the lack of target domain data. The effect becomes more obvious when the available target domain data is further reduced to a ratio of 100:1. Initializing the target domain feature extractor with pre-trained weights led to consistently better results. The improvements are especially significant when there is lesser target domain data available. For example, when the source:target dataset ratio is 10:1, the “Paired + L2” model achieved a final distance success rate of 94% compared to the 98% of the “Pre-train + Paired + L2 model”, an improvement of 4%. When the source:target dataset ratio is 100:1, the “Paired + L2” model only achieved a final distance success rate of 39% compared to the 93% of the “Pre-train + Paired + L2” model, a significantly larger improvement of 54%.

Also consistent with the results in the fully supervised learning section is that training with pairs and L2 loss outperforms just finetuning with target examples only. To summarize, training with paired-examples and initializing with pre-trained weights from the source domain leads to better overall performances when the source and target domains are similar.

Intuitively, the less striking the change from source to target domain, the less the pre-trained neural network on the source domain needs to change to adapt to the target domain. The question arises whether pre-training is beneficial for visually dissimilar domains. In the above experiments, the chosen target domain happens to be both visually and numerically similar. On the other hand, we would also like to test the effects of these models on an extreme form of source to target domain transformation. The chosen target domain is created through a putting each colour channel of the source domain image through a Canny edge detector and recombining the channels. A sample of the source-target domain pair can be seen in Figure 4.7. The models are trained with a source:target dataset ratio of 100:1.

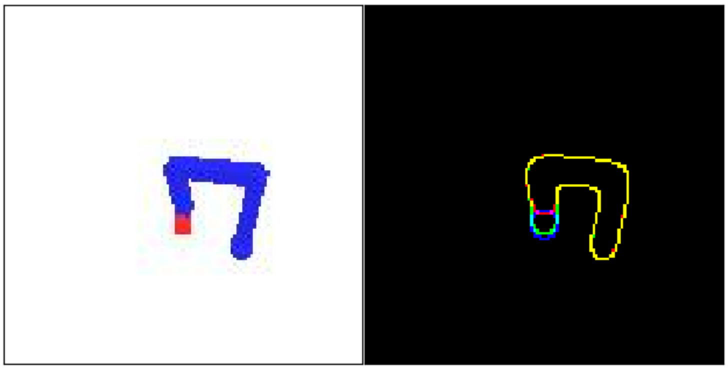
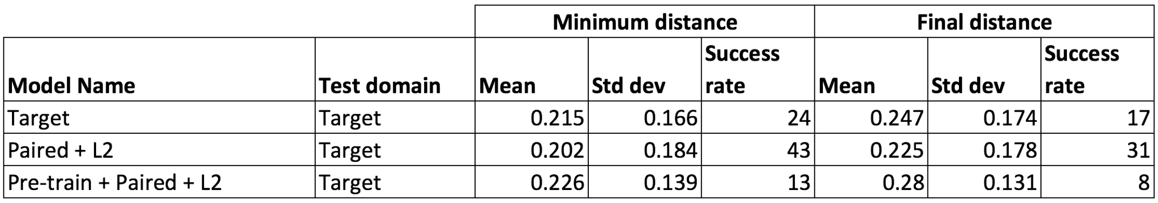


Figure 4.7: Sample of the source and target domain image. Left: source domain image, right: target domain image.



		Minimum distance			Final distance		
Model Name	Test domain	Mean	Std dev	Success rate	Mean	Std dev	Success rate
Target	Target	0.215	0.166	24	0.247	0.174	17
Paired + L2	Target	0.202	0.184	43	0.225	0.178	31
Pre-train + Paired + L2	Target	0.226	0.139	13	0.28	0.131	8

Figure 4.8: Results of selected semi-supervised models on the edge target domain. The source:target dataset ratio for these models are 100:1.

From the results, we find that the effect of pre-training has a negative effect when the source and target domains are highly dissimilar. In fact, initializing with pre-trained weights led to worse results than training a neural network from scratch, with a final distance success rate of 8% compared to 17%. Training with pairs and the L2 loss between feature layers still led to improved performances even when the domains are visually dissimilar. In the context of utilizing simulators as a source domain to generate training images, this problem is not significant since we would not expect such a large visual discrepancy between the source and target domains. The simulator-reality domains should hold a relationship more alike the first set of source-target domains used for testing, where pre-training led to much improved results.

Chapter 5

Image Translation and Pairing across Domains

Being able to automate the task of pairing of images between separate domains is highly valuable. For example, if there is an algorithm that can transform a simulated image to a corresponding real world image, a large amount of realistic training data with annotations can be created. Pairing simulated images with real images is highly expensive task, hence there is a large interest in wanting to automate pairing and translation between domains. A successful pairing between two different image domains maintains the semantics across domains. This concept is illustrated in Figure 5.1.

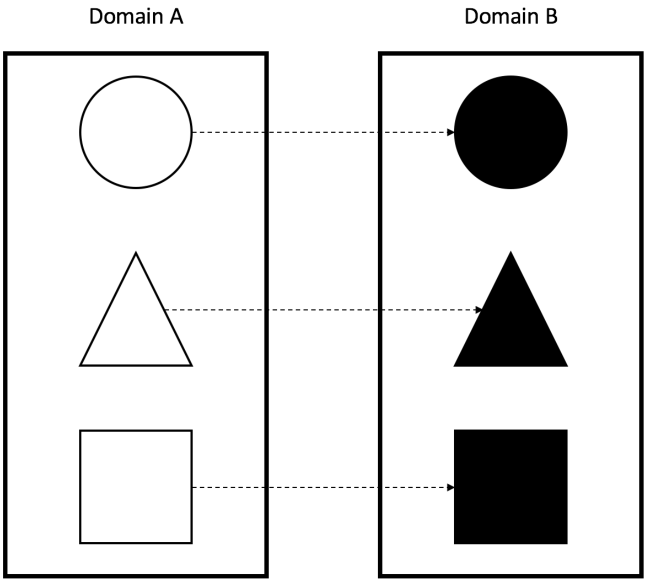


Figure 5.1: Pairing of images between different visual domains.

Generative adversarial networks have been used to generate pictures of a particular domain from random vectors. Figure 5.2 depicts this process. However, for our application, the input is an image from the source domain and the output is an image from the target domain. Furthermore, we want the source and target domain images to be semantically related. The works in [23, 2] present methods to train GANs to translate a source domain image to a corresponding target domain image. The model in [2] is of particular interest since the training process it is an unsupervised method and fits in well with the motivation of this paper. This model is termed CycleGAN by the authors and the application to training a robotic task will be discussed in greater detail in the later sections. Other methods include incorporating a loss function [24] to guide the training of the generator.

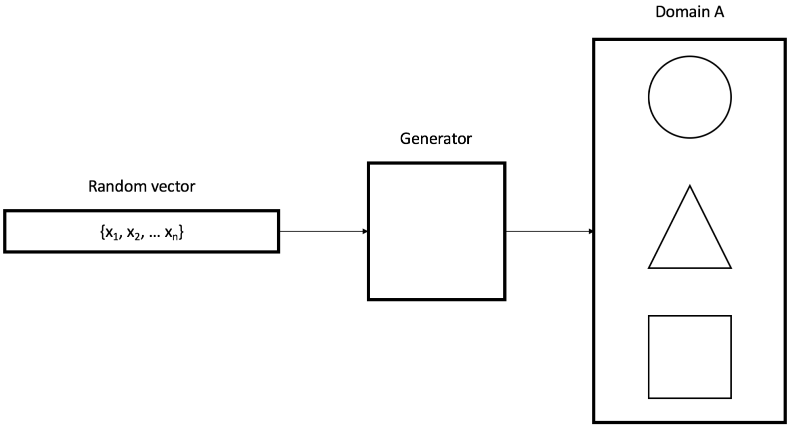


Figure 5.2: GAN as a generative model for images

This chapter will focus mainly on the study of GANs for image translation and pairing across domains as seen in Figure 5.3. A novel model that utilizes the task loss of the generated target domain image to guide the GAN training process will be proposed and evaluated in Section 5.4.

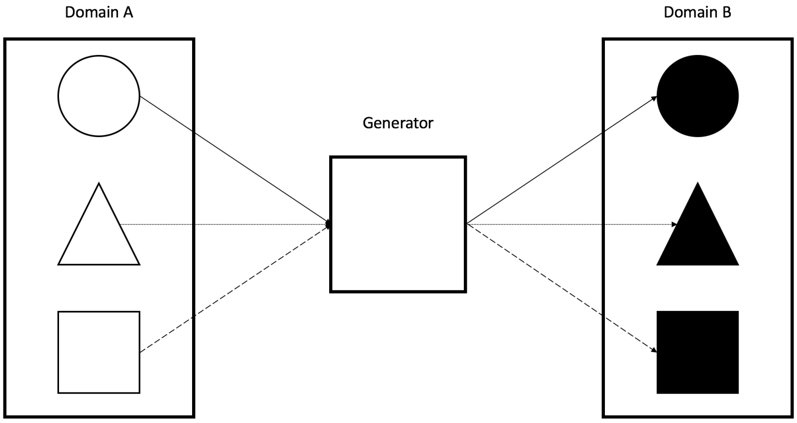


Figure 5.3: Adapting GANs to achieve translation and pairing across domains

5.1 Training details and architectures

One flaw of using generative adversarial networks for image translation is the instability of the training process. This section covers general training details and tips that help to stabilize the training process.

In [24], the authors proposed training the discriminator with a history of generated images. A similar process is implemented for this project. The history of generate images is created as a first-in first-out (FIFO) queue with a limited capacity. At each step where the generator creates new images, they are added to the queue. When the number of images in the queue exceeds the capacity, the oldest images are discarded. For each mini-batch training step for the discriminator, the new batch of generated images is mixed with a randomly sampled batch of generated images from the queue. The ratio of most recent batch of generated images to randomly sampled batch of generated images can be varied, but the default ratio used in experiments is 1:3.

Other ways of stabilizing the training process utilizes additional loss functions to guide the generator. In [2], the authors propose a cycle consistency loss which reduces the space of possible mapping functions, hence encouraging the generators to learn a mapping from a source input to a related target output. In [24], the authors propose a feature transform loss by mapping the generated output and input to a similar feature space and penalizing the L1 loss between the two feature vectors. However, in the experiments of the paper, the authors use an identity map, which implicitly assumes a visual relation between the source and target domain, which may not necessarily always be the case.

We also found that the training of generative adversarial networks is highly sensitive to initialization. A common problem faced by training generative adversarial networks is being stuck at poor local minima/maxima early in the training process and usually re-initializing the training process helps.

The choice of discriminator and generator architectures in this paper follows those described in the appendix CycleGAN paper [2]. Since we are dealing with 128x128 images, we use the 6 block generator architecture as described in the paper.

5.2 Typical GAN for image translation

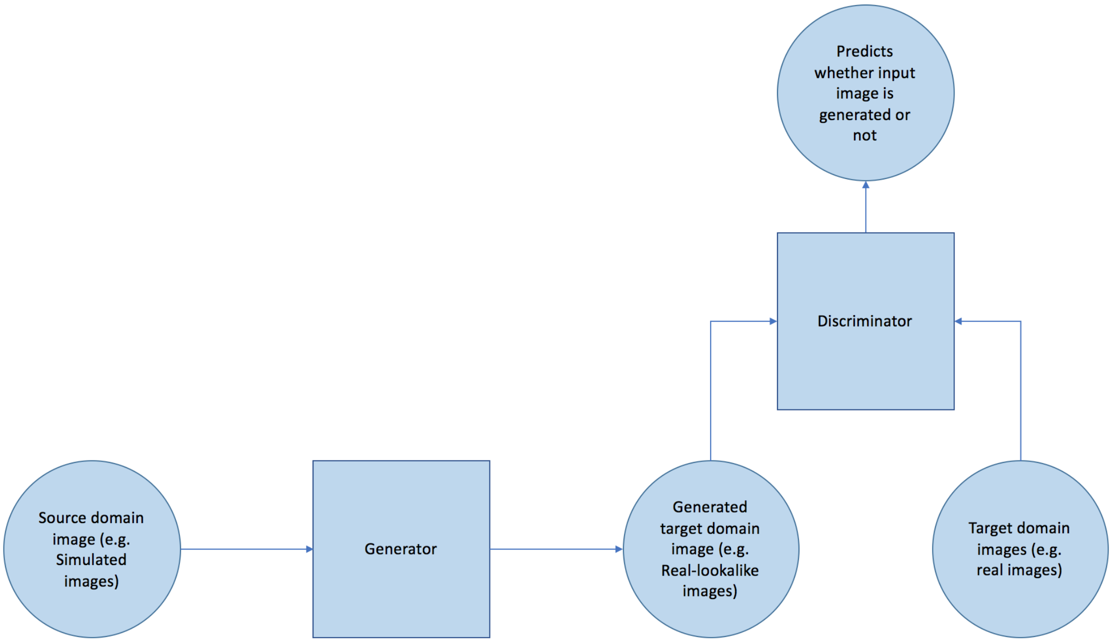

Predicts whether input image is generated or not
Discriminator
Source domain image (e.g. Simulated images)
Generator
Generated target domain image (e.g. Real-lookalike images)
Target domain images (e.g. real images)

Figure 5.4: Typical GAN architecture for image translation

Given a source domain X and a target domain Y, we want train the generator to learn a mapping function $G : X \to Y$ and a discriminator D_Y to differentiate between generated target domain images $G(x)$ and real target domain images y. The objective function can be expressed as:

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim Y}[log(D_Y(y))] + \mathbb{E}_{x\sim X}[log(1 - D_Y(G(x)))] \tag{5.1}$$

The adversarial training leads to a minimax game where the discriminator will try to maximize the above loss function while the generator will try to minimize it. The actual implementation differs from the above equation in that the negative log likelihood objective is replaced with a least square loss. This follows the implementation details from [2], where using the least square loss is reported to be more stable during training and generates higher quality images. The least squares version of the objective function is:

$$L_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y\sim Y}[D_Y(y)^2] + \mathbb{E}_{x\sim X}[(1 - D_Y(G(x)))^2] \tag{5.2}$$

With this basic architecture, the generator will learn to transform an input source image to a target domain image. However, this will likely not lead to a pairing between domains since there is no function in the objective that encourages this relation. This property will be explored and tested in Section 5.4.

5.3 CycleGAN

As discussed in Section 2.7 and in the introduction of this chapter, CycleGAN is a method of training generative adversarial networks to perform image-to-image translation without the need of source-target domain pairs x-y developed by the authors in [2].

5.3.1 Model details

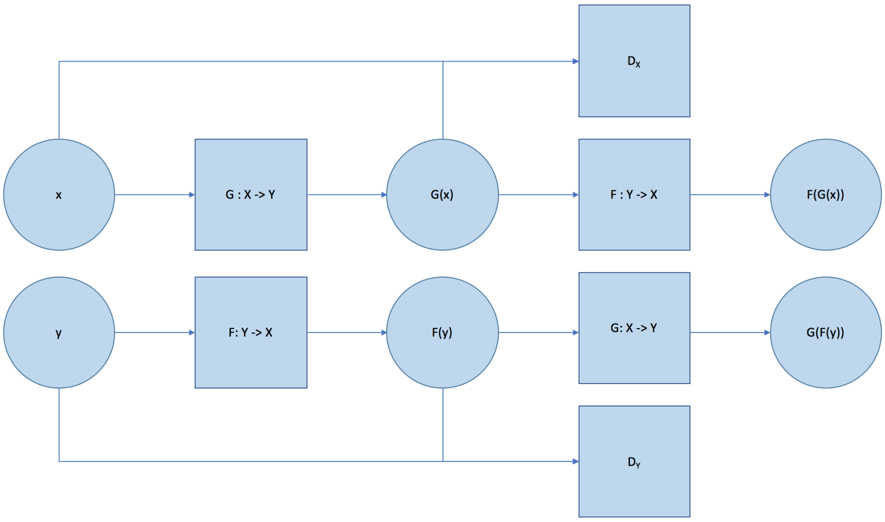


Figure 5.5: CycleGAN architecture

A figure of the CycleGAN architecture can be found in Figure 5.5. Instead of simply learning a mapping function $G : X \rightarrow Y$ only, the inverse mapping function $F : Y \rightarrow X$ is learnt as well. Likewise, a separate discriminator D_Y which learns to differentiate between images from domain y and the generated images $F(y)$ is necessary to train the inverse mapping function.

An additional loss is added to the objective function to utilize the inverse mapping function. This loss is termed a cycle consistency loss. It penalizes the L1 loss between the original image and the reconstructed image after being put through the generator and the inverse generator. This is to encourage the generators to learn to map to a related output in a different domain. The cycle consistency loss and full objective can be formulated as follows:

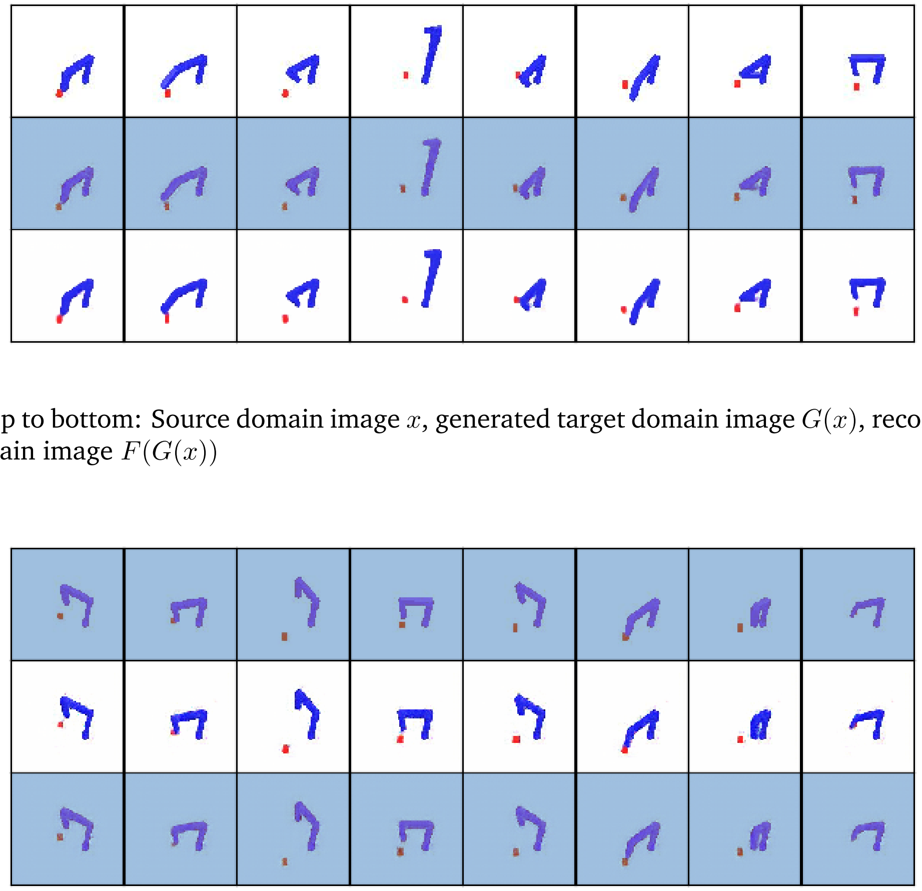
$$L_{cyc}(G,F) = \mathbb{E}_{x \sim X}[||F(G(x)) - x||_1] + \mathbb{E}_{y \sim Y}[||G(F(y)) - y||_1] \tag{5.3}$$

$$\begin{aligned} L_{CycleGAN}(G,F,D_X,D_Y,X,Y) = & \; L_{GAN}(G,D_Y,X,Y) \\ & + L_{GAN}(F,D_X,Y,X) \\ & + \lambda L_{cyc}(G,F) \end{aligned} \tag{5.4}$$

5.3.2 Results

The CycleGAN is trained with a learning rate of 0.0002 for 30000 training steps. Two different source-target domain pairings are tested. Each pairing has 22656 images in the source domain and 8000 images in the target domain. The experiments in the original paper are trained for much longer and due to lack of computational resources and time, experiments in this paper are run over a relatively much shorter period of time.

The first source-target domain pairing that was tested is shown in Figure 5.6. For this simple transformation, the pairings are quite successful and generalized well.

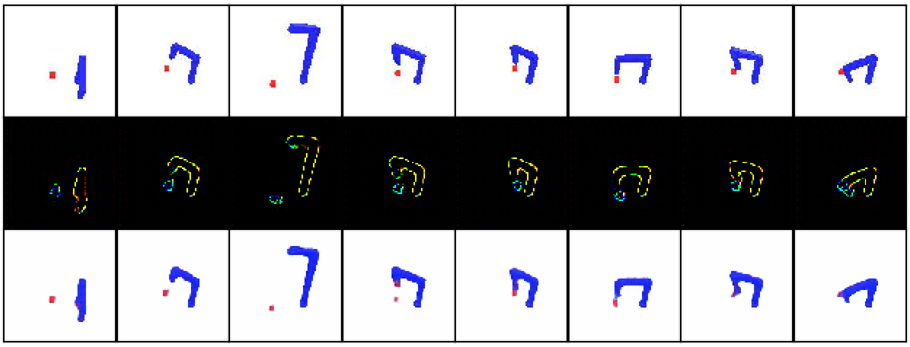


(a) From top to bottom: Source domain image x, generated target domain image $G(x)$, reconstructed source domain image $F(G(x))$

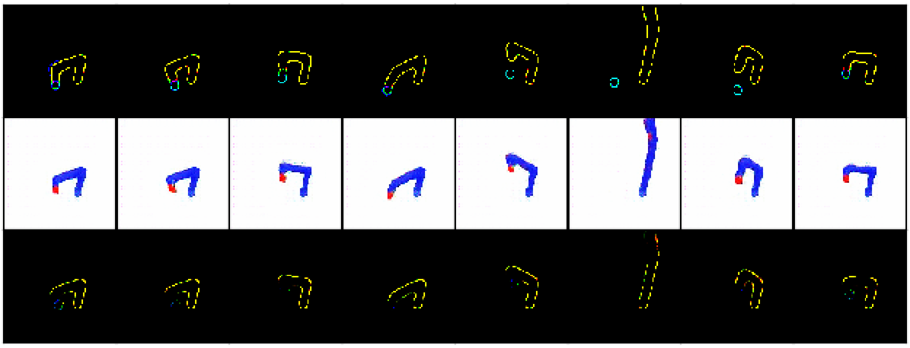
(b) From top to bottom: Target domain image y, generate source domain image $F(y)$, reconstructed target domain image $G(F(y))$

Figure 5.6: Sample images from CycleGAN after 30000 training steps. Target domain created by multiplying [0.25, 0.5, 0.75] to the RGB channels of the source domain image respectively.

Further testing is done on a more difficult source-target domain pairing. The source domain remains the same while the target domain is created by processing the original image with a Canny edge detection algorithm. Each colour channel of the source domain image is put through a Canny edge detection algorithm and recombined to form the target domain image. Samples during the training process can be found in Figures 5.7 and 5.8.

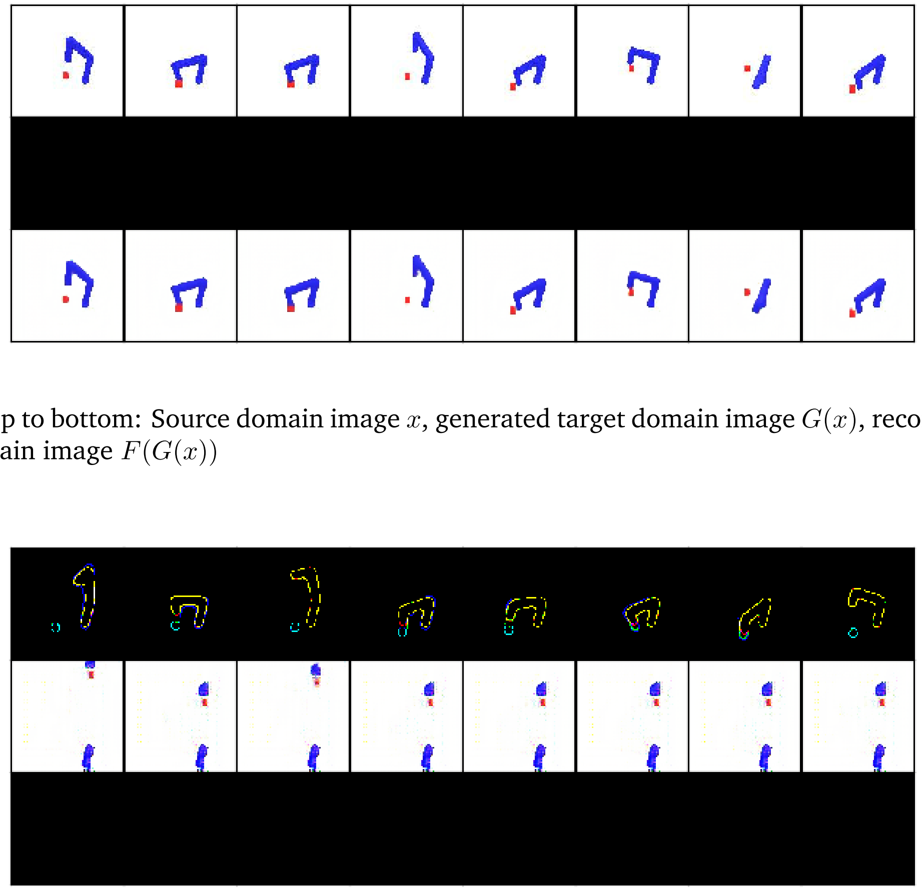


(a) From top to bottom: Source domain image x, generated target domain image $G(x)$, reconstructed source domain image $F(G(x))$



(b) From top to bottom: Target domain image y, generate source domain image $F(y)$, reconstructed target domain image $G(F(y))$

Figure 5.7: Sample images from CycleGAN after 8000 training steps. Target domain created by putting each colour channel of the original image through a Canny edge detection algorithm with a sigma value of 4 and recombining the channels afterwards.



(a) From top to bottom: Source domain image x, generated target domain image $G(x)$, reconstructed source domain image $F(G(x))$

(b) From top to bottom: Target domain image y, generate source domain image $F(y)$, reconstructed target domain image $G(F(y))$

Figure 5.8: Sample images from CycleGAN after 30000 training steps. Target domain created by putting each colour channel of the original image through a Canny edge detection algorithm with a sigma value of 4 and recombining the channels afterwards.

Interestingly, at training step 8000, a reasonably good image translation was achieved. Upon further training, the image translation diverged from the desired pairings. While the image translation at this point is far from perfect, other experiments were ran and a better quality image translation for this source-target domain pairing was never found. A possible explanation is that this particular source-target domain translation was difficult to learn. From the sample images at step 8000, it definitely shows the potential and possibility for it. However, achieving a stable training process towards such an image translation remains rather difficult with this approach.

Another point to note about the CycleGAN is that while the cycle consistency loss encourage a pairing between source domain image x and reconstructed source domain image $F(G(x))$, it does not necessarily ensure a pairing between x and generated target domain image $G(x)$ and between $G(x)$ and $F(G(x))$. This is best seen in Figure 5.8a, where a very strong pairing between x and $F(G(x))$ can be seen while the pairings to $G(x)$ is with a visually completely black image. It is highly likely that there are very small values in this black image that allows this pairing to exist and it shows that the cycle consistency loss is unable to ensure a great and consistent pairing will be found. A similar problem holds for the target domain translations from y to $F(y)$ and $F(y)$ to $G(F(y))$.

As stated in the start of this section, these experiments were ran over 30000 training steps only whereas those in the original paper were ran over a much longer period of time. It is entirely possible that further training might lead to better results, but this will not be further tested in this paper due to lack of computational resources and time. Also worth noting that the failure to achieve a good image translation in the latter domain pairing can also be due to limitations of the generator and discriminator architectures.

5.4 TaskGAN

In the scenario where there are labels for some amount of data in the target domain, we are able to train a neural network to perform a regression task with that label. For example, in Section 4.3, we show that a small amount of labels in the target domain can be supplemented by rich source domain data to achieve a good performance. The neural network that is trained for the regression contains knowledge about how a target domain image should look like given a label. Intuitively, this means that it is also possible for this neural network to guide the generator in creating an image in the target domain that is paired with the source domain image given a label of the source domain image. Given a data rich source domain that has annotations, it is possible to backpropagate gradients from the task loss on the generated target domain image to the generator. The architecture for this model is similar that of the typical GAN model described in Section 5.2. The key configuration is training the generator with the task loss of the generated image.

5.4.1 Model details

The name TaskGAN comes from the addition of a trained neural network on a task in the target domain and using the task loss of the generated target domain image to guide the training of the generator.

Given a target domain Y with images y_i and labels ϕ_i^y, we can train a neural network to learn a mapping function $f_Y : Y \to \phi^Y$. Furthermore, as shown in Chapter 4, supplementing training with the data from a rich source domain X with images x_i and labels ϕ_i^x can lead to the neural network learning a more effective mapping function f_Y. We want the labels across domains to be transferrable in order to create a pairing effect when training. In other words, for each x_i-y_i pair, the labels ϕ_i^x-ϕ_i^y should be identical. This will allow the source domain label ϕ_i^x to be used as the target domain label ϕ_i^y and the task loss of the generated target domain image can be computed with $(f_Y(G(x_i)) - \phi_i^x)^2$. Then, the TaskGAN can be trained without the need of any explicit x-y pairs with the following objective function:

$$\begin{aligned} L_{TaskGAN}(G, D_Y) = &\; \mathbb{E}_{y\sim Y}[D_Y(y)^2] \\ &+ \mathbb{E}_{x\sim X}[(1 - D_Y(G(x)))^2] \\ &+ \lambda_\phi \mathbb{E}_{x\sim X}[(f_Y(G(x)) - \phi^x)^2] \end{aligned} \tag{5.5}$$

where λ_ϕ corresponds to the weight of the task loss of the generated image $G(x)$. The weight used in implementation is 0.01. We find that larger values tend to not work as well. Through adversarial training, we aim to solve the following minimax game:

$$G^* = \arg\min_G \max_{D_Y} L_{TaskGAN}(G, D_Y) \tag{5.6}$$

To further break it down, at each training step, the generator and discriminator are trained to minimize the following functions respectively:

$$L_G(G, D_Y, X, \phi^X) = \mathbb{E}_{x\sim X}[(1 - D_Y(G(x)))^2] + \mathbb{E}_{x\sim X}[(f_Y(G(x)) - \phi^x)^2] \tag{5.7}$$

$$L_{D_Y}(G, D_Y, X, Y) = \mathbb{E}_{y\sim Y}[(1 - D_Y(y))^2] + \mathbb{E}_{x\sim X}[D_Y(G(x))^2] \tag{5.8}$$

Given a minibatch of size N, equations 5.7 and 5.8 can be rewritten in the following manner:

$$L_G(G, D_Y, X, \phi^X) = \frac{1}{N}\sum_{i=1}^{N}[(1 - D_Y(G(x_i)))^2 + (f_Y(G(x_i)) - \phi_i^x)^2] \tag{5.9}$$

$$L_{D_Y}(G, D_Y, X, Y) = \frac{1}{N}\sum_{i,j=1}^{N}[(1 - D_Y(y_j))^2 + (D_Y(G(x_i)))^2] \tag{5.10}$$

At this point, it is also worth pointing out that in earlier sections, the mapping function f_Y was trained with some x-y pairs provided while the above TaskGAN model does not use explicit pairs during training. It is possible to train the TaskGAN with explicit pairs with a small modification. An additional L1 loss between the generated image $G(x)$ and target domain image y can be added and used to train the generator. Equation 5.9 can be updated to the following when training with paired examples:

$$L_G(G, D_Y, X, Y, \phi^X) = \frac{1}{N}\sum_{i=1}^{N}[(1 - D_Y(G(x_i)))^2 + (f_Y(G(x_i)) - \phi_i^x)^2 + ||y_i - G(x_i)||_1] \tag{5.11}$$

Another method of training with paired examples is described in [23], where a discriminator is trained to distinguish between real x_i-y_i pairs and fake x_i-$G(x_i)$ pairs. A reason for not wanting to using x-y pairs in the training on TaskGAN is motivated by the fact that the mapping function f_Y can be trained without pairs at all. If there is a target domain Y dataset that is already well annotated with labels that can be replicated in the source domain, a neural network can be trained solely on that dataset and used for the TaskGAN. Clearly, having some pairs during training would help the learning process and we will evaluate the quality of the images generated using a TaskGAN trained with and without pairs in the next section.

5.4.2 Results

For testing of the TaskGAN, we create several source to target transformations of varying difficulties. We choose transformations that are typical of differences between simulators and reality. The target transformation is created by editing elements V-REP scene. The mechanics of the robot arm are not modified (i.e. the lengths of the links, the relative positions of tip and target). Other attributes such as shape, colour, and width are changed. This is to test the effectiveness of transferring knowledge for the same task over to different visual domains.

All datasets are created with 800 paired images between the source-target domain made of 50 episodes of 16 steps each. This is used in conjunction with the source domain dataset with 22656 images to train a neural network to learn the mapping function f_Y. The neural network is initialized with pre-trained weights and trained over 15000 steps with a learning rate of 0.001 with the task loss and L2 loss between feature layers. Image translation models (i.e. the GAN models) are trained for 20000 steps with a learning rate of 0.0002.

(a) Source domain

(b) Target domain

Figure 5.9: First source to target transformation. Links width are changed, with the bottom link being wider, the middle link remaining the same, and last link being thinner. Colours are slightly changed to a lighter shade.

The first transformation changes the widths of the links and the colours of the cube and robot arm slightly. A sample source-target domain image pair can be found in Figure 5.9. Some samples of the images generated by the TaskGAN can be found in Figure 5.10.

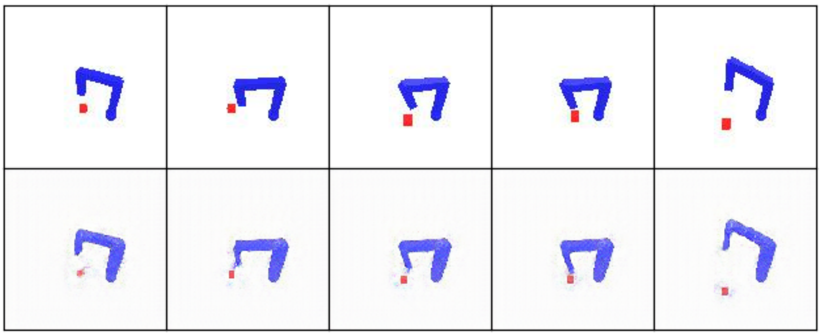


Figure 5.10: Samples from TaskGAN model for first source to target transformation. Top row represents samples from source domain. Bottom row represents the generated target domain images from the source domain samples.

We find that the quality of generated images are of high quality. Generally, pose of the arm is well matched and the position of the cube is correct. The sizes of the links are also transformed to better represent the target domain, with a thicker link at the bottom and a thinner final link. The cube is also correctly being resized to a smaller one. Colours are also lightened to match the target domain. Overall, using the TaskGAN for this transformation is rather successful.

Next, we want to try a more difficult source to target transformation. A sample of the second source to target transformation can be found in Figure 5.11.

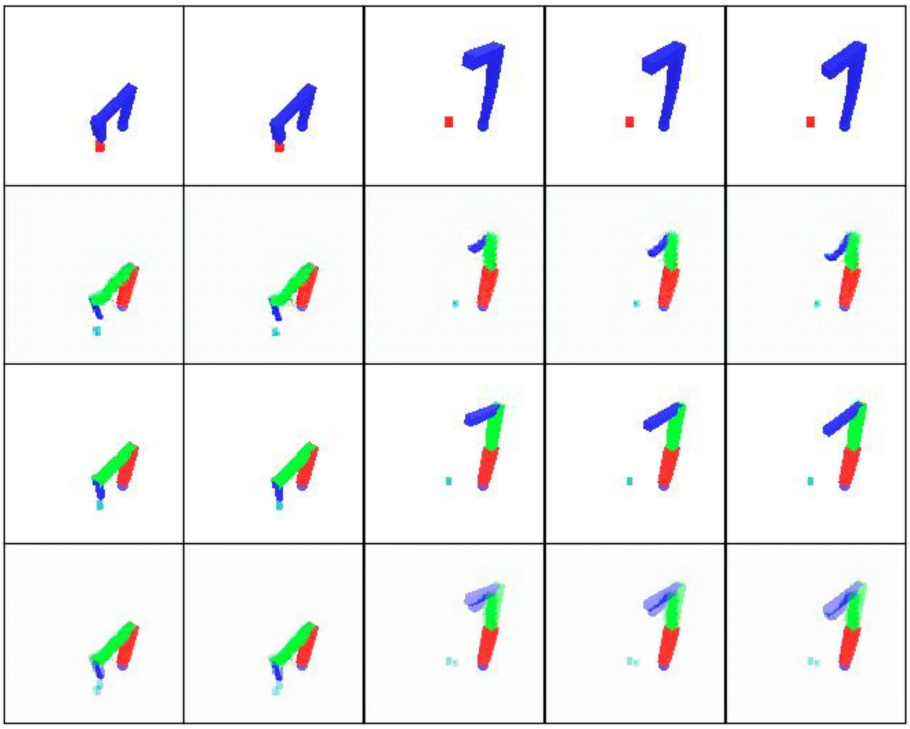
(a) Source domain

(b) Target domain

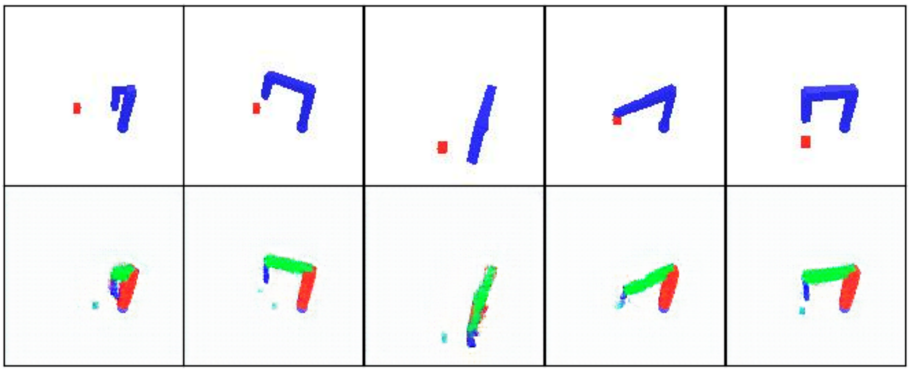
Figure 5.11: Second source to target transformation. In addition to the size changes of the links and cube, the second transformation involves more colour changes, with different colours for each link and inverting the colour of the cube.

For the second source to target transformation, we will evaluate 3 different models: the typical GAN for image translation, TaskGAN trained without paired examples, and TaskGAN trained with paired examples. The respective samples can be found in Figures 5.12, 5.13, 5.14.

As expected, the models' quality of translated images can be ranked in the following ascending order: typical GAN, TaskGAN trained without paired examples, TaskGAN trained with paired examples. From the figures, we can see that the generated images by the typical GAN captures the qualities of the target domain, but consistently distorts the information such as position of the cube and pose of the robot arm. Both TaskGAN models create better translations, with translations that correctly retain arm pose and cube position information. From the paired samples, the TaskGAN trained without paired samples has very slight errors as seen from the superimposed images while the TaskGAN trained with paired samples are of higher quality. For both TaskGANs, we can occasional large errors in the random unpaired samples where the cube disappears after translation. This might be due to the TaskGANs generalizing poorly since small amount of target domain samples (800 in this case) is used to train both f_Y and GAN. Ways to improve it include using more x-y pairs in the training of the f_Y or having more unpaired y examples to train the discriminator.

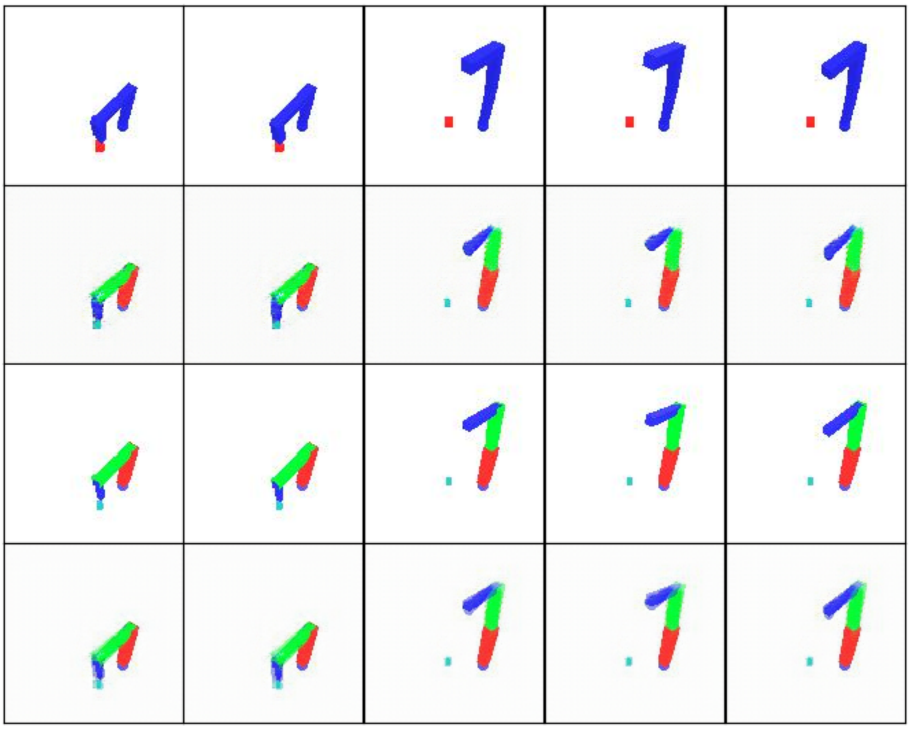


(a) Paired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$, paired target domain images y, superimposed image from $G(x)$ and y.

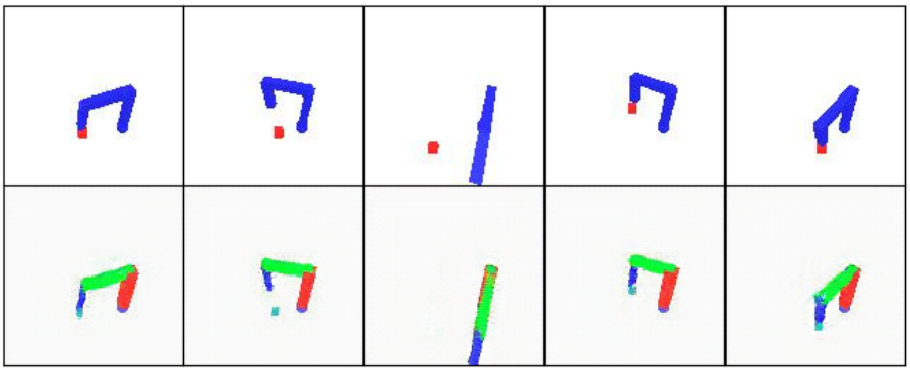


(b) Random unpaired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$.

Figure 5.12: Samples for second source to target transformation using typical GAN.

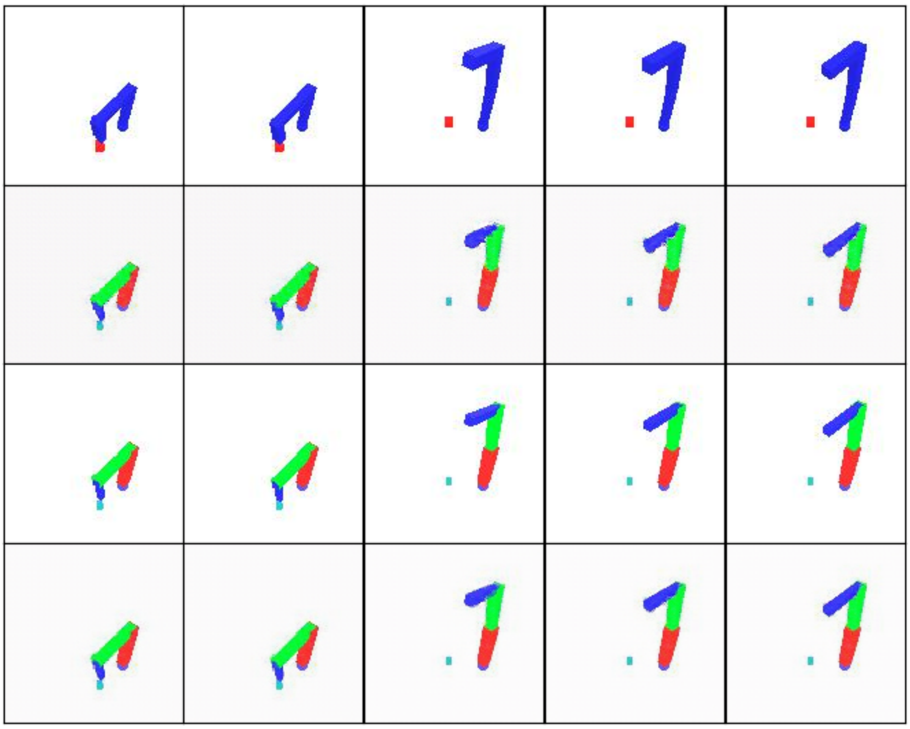


(a) Paired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$, paired target domain images y, superimposed image from $G(x)$ and y.

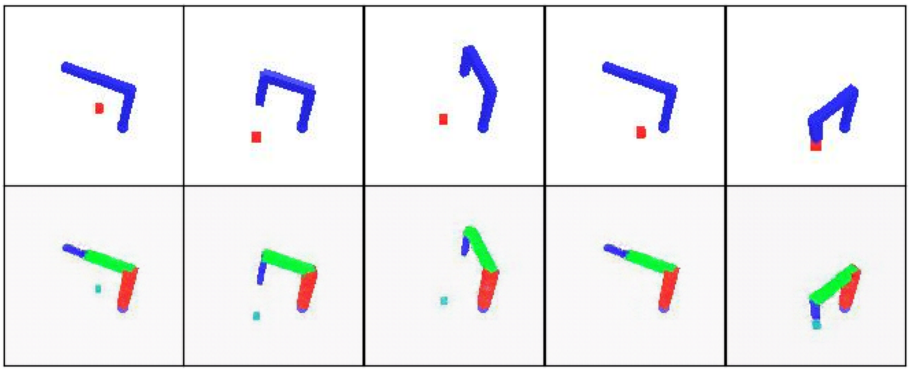


(b) Random unpaired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$.

Figure 5.13: Samples for second source to target transformation using TaskGAN trained without paired examples.

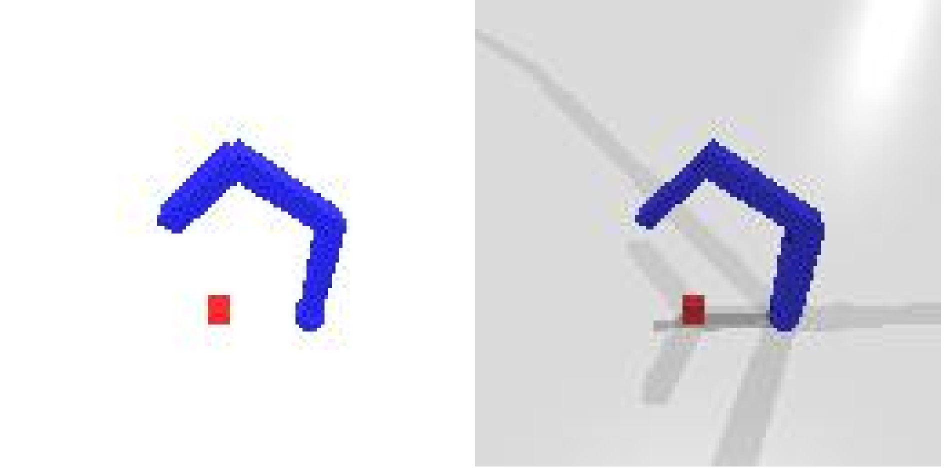


(a) Paired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$, paired target domain images y, superimposed image from $G(x)$ and y.



(b) Random unpaired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$.

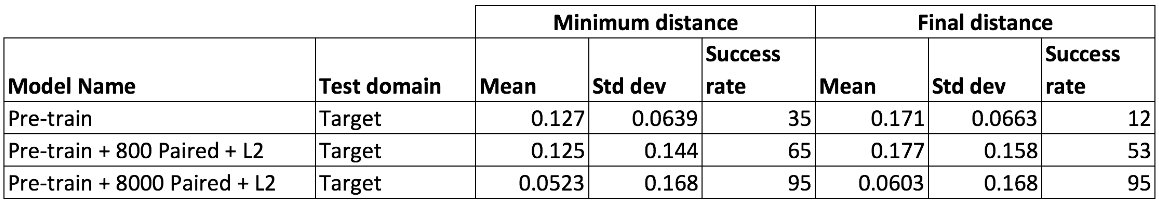
Figure 5.14: Samples for second source to target transformation using TaskGAN trained with paired examples.



(a) Source domain (b) Target domain

Figure 5.15: Third source to target transformation. Cube size remains the same while width of links are changed in a way similar to previous transformations. Lighting effects are added and all colours including background are slightly changed.

For the third source to target transformation, we want to introduce realistic effects as well as a different background. We chose to do this through adding light effects and changing the background colour slightly. In the previous tests, we used 800 paired examples to train the neural network f_Y. However, we found that it was insufficient in guiding the generator in this particular transformation. We trained another f_Y with 8000 paired examples to ensure good performance.



		Minimum distance			Final distance		
Model Name	Test domain	Mean	Std dev	Success rate	Mean	Std dev	Success rate
Pre-train	Target	0.127	0.0639	35	0.171	0.0663	12
Pre-train + 800 Paired + L2	Target	0.125	0.144	65	0.177	0.158	53
Pre-train + 8000 Paired + L2	Target	0.0523	0.168	95	0.0603	0.168	95

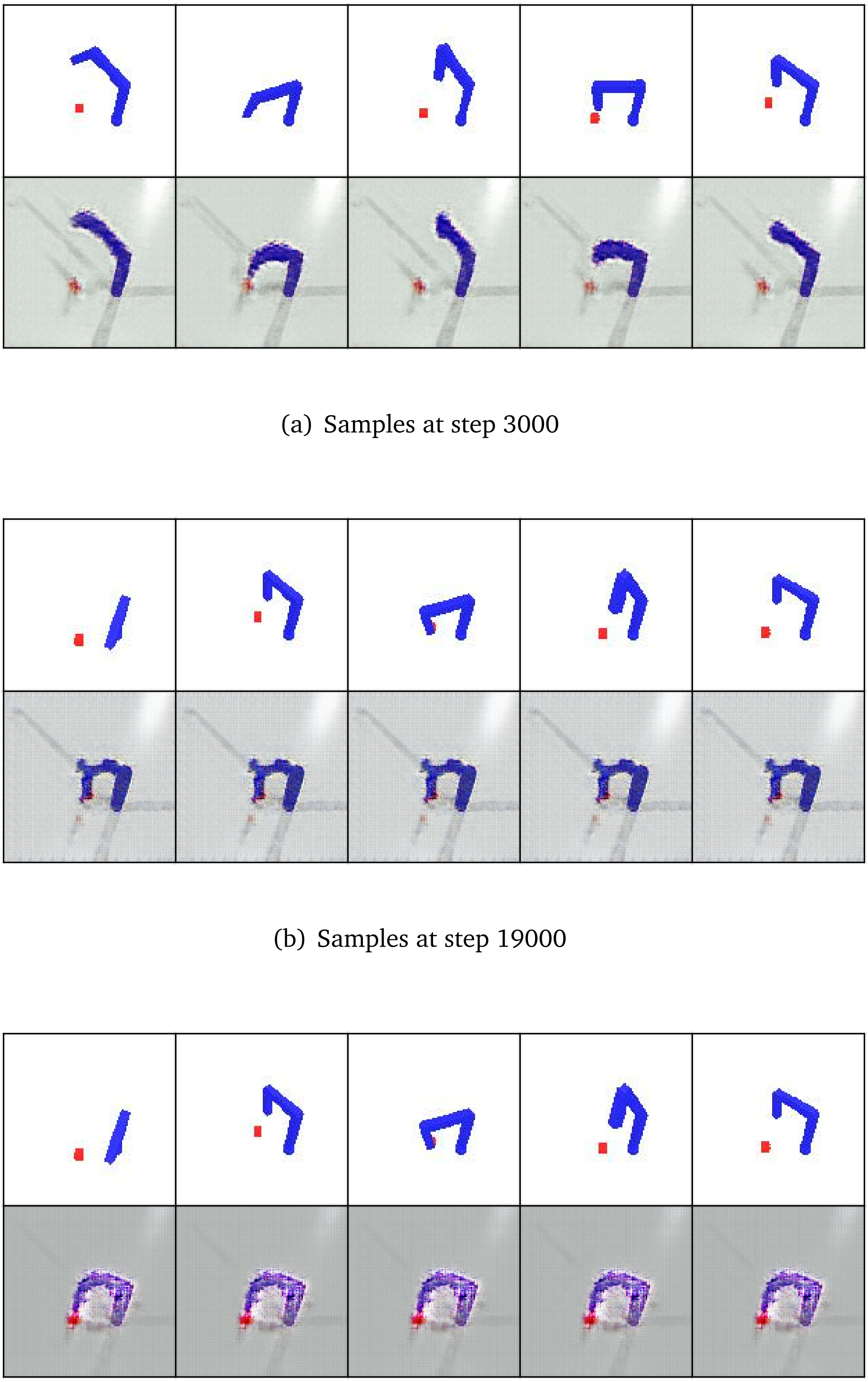
Figure 5.16: Performances of various models in the third target domain.

The results can be found in Figure 5.16. Training with 8000 paired examples led to a final distance success rate of 95% compared to the 53% when trained with 800 paired examples. Intuitively, the better the performance of f_Y, the more effective it will be in guiding the generator to learn to generate paired examples. 3 different models will be evaluated: the typical GAN for image translation, TaskGAN with f_Y trained on 800 paired examples, and TaskGAN with f_Y trained on 8000 paired examples. To be clear, the TaskGAN itself is trained without paired examples. The respective samples can be found in Figures 5.17, 5.18, 5.19.

For the typical GAN, we find that with more training, the generator learns to map any given input source domain image to a similar target domain image. Since the only loss given to the generator is to create a target domain lookalike image, there is no incentive to create a related pairing between source and target domains. This tends to encourage this behaviour of mapping all source domain images to a highly similar target domain image as seen in Figure 5.17b and c. It is worth noting that at earlier training steps, it is still able to generate a target domain image that somewhat matches the pose of the source domain image as seen in Figure 5.17a. It fails to match the position of the cube and treats the cube as part of the background.

For the TaskGAN with f_Y trained with 800 paired examples, we find that the general translation quality is quite poor. While it still manages the match the pose of the arm, the arm itself is of low quality. It manages to recreate the general background of the target domain, but treats the cube as part of the background. In every translated target domain image, the cube is at the exact same position, disregarding the position of the cube in the source domain. This makes it a very poor image translation model for generating target domain data. Reasons for the difficulty of training a good image translation model are the complex background changes, the weak predicting strength of f_Y, and a combination of both factors. As a point to note, the f_Y used for the successful image translation in the second source to target domain translation had a final distance success rate of 43% in that target domain, making it even weaker than the current f_Y used in this example. This leads us to believe the increased complexity of the source to target transformation played a larger role in affecting the image translation quality.

For the TaskGAN with f_Y trained with 8000 paired examples, we find that the generated target domain images are of good quality. Most aspects of the target domain are reconstructed, with shadows of the cube and the arm. The shadows of the arm are not perfect as it is treated as part of the background and always reconstructed in the same manner. However, the shadow are not significantly important to the task, so this is an acceptable flaw. While some shapes of the arms are slightly distorted, both arm pose and position of cube are accurately reconstructed, making these translated images a highly desirable source of training data in the target domain.

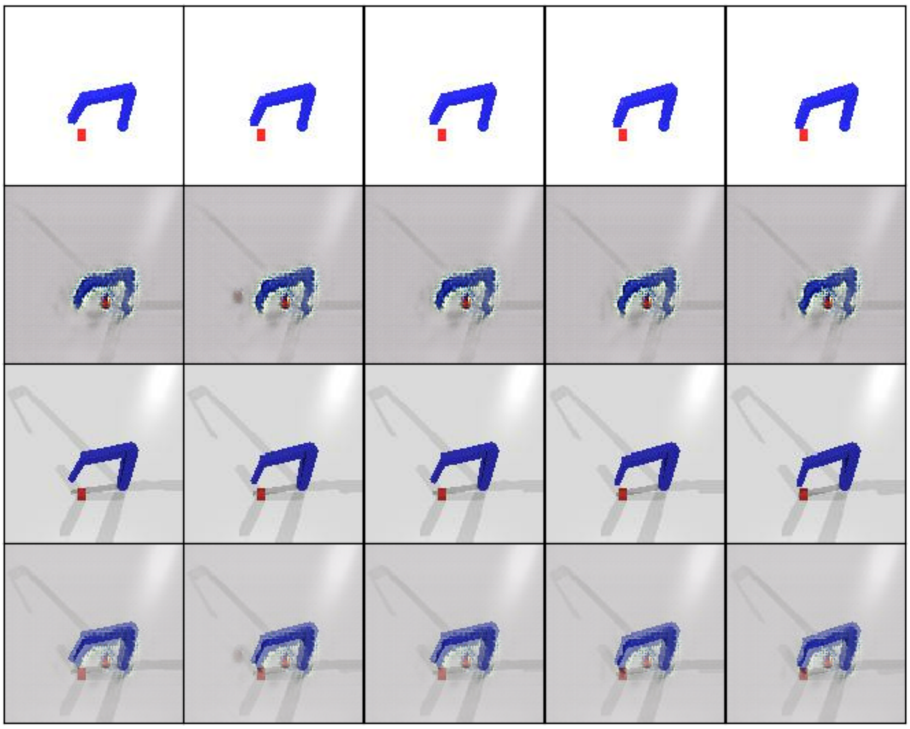


(a) Samples at step 3000

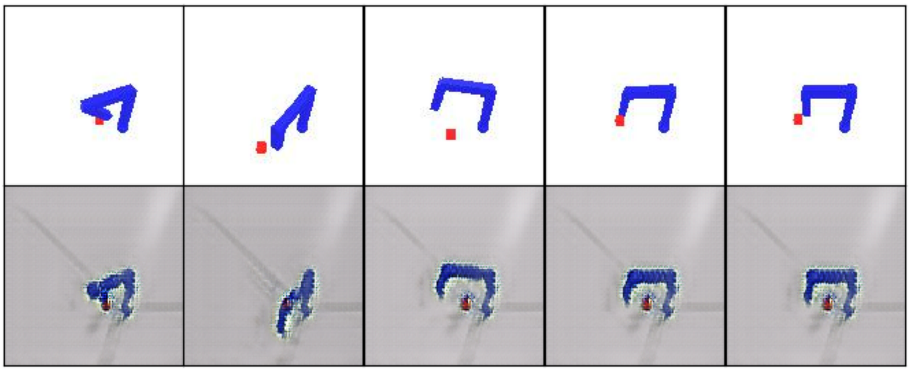
(b) Samples at step 19000

(c) Samples at step 20000

Figure 5.17: Samples at various training steps for the third source to target transformation using a typical GAN for image translation. For each set of samples, the top row represents the source domain images x and the bottom row represents the generated target domain images $G(x)$.

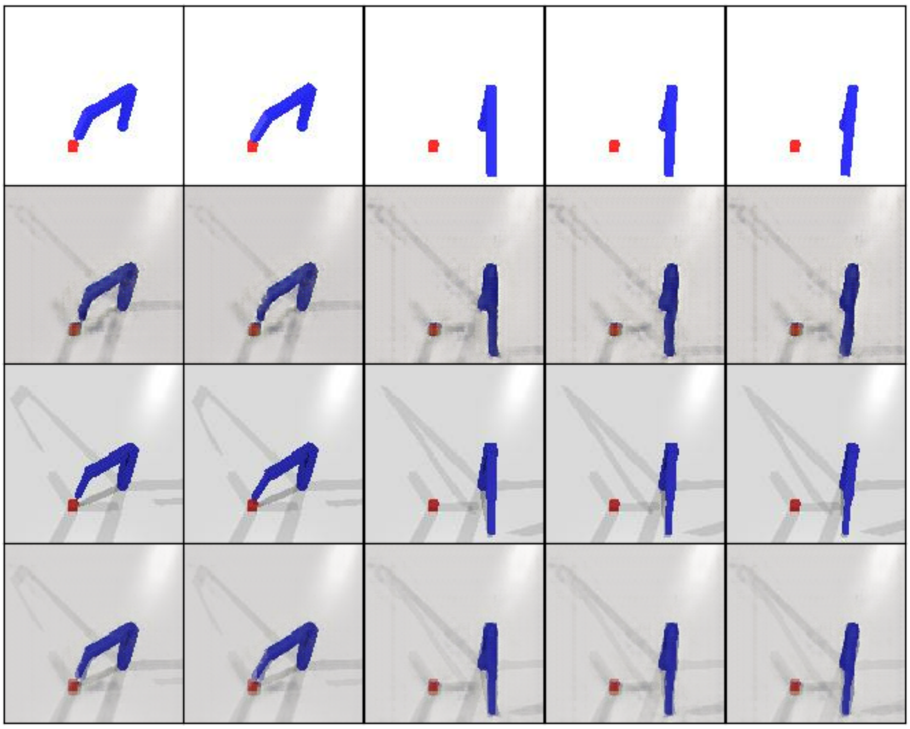


(a) Paired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$, paired target domain images y, superimposed image from $G(x)$ and y.

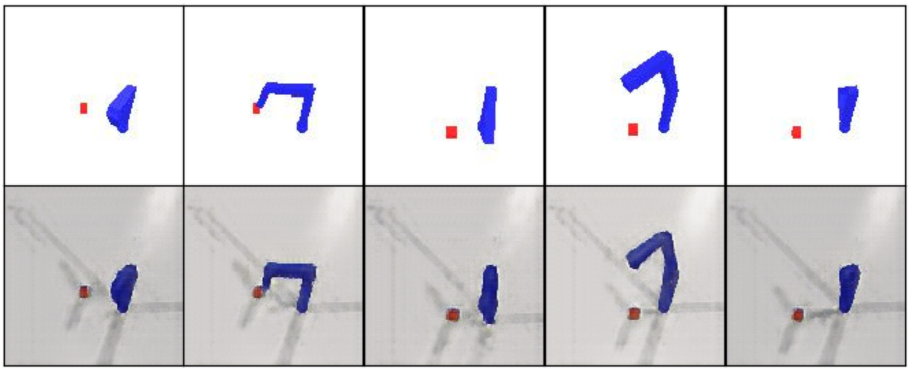


(b) Random unpaired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$.

Figure 5.18: Samples for third source to target transformation using TaskGAN with f_Y trained with 800 paired examples.

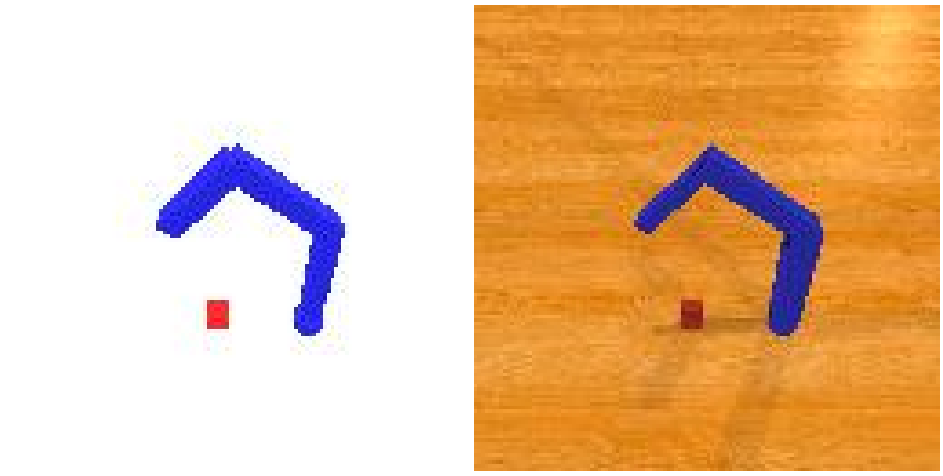


(a) Paired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$, paired target domain images y, superimposed image from $G(x)$ and y.



(b) Random unpaired samples. From top to bottom: Source domain images x, generated target domain images $G(x)$.

Figure 5.19: Samples for third source to target transformation using TaskGAN with f_Y trained with 8000 paired examples.



(a) Source domain

(b) Target domain

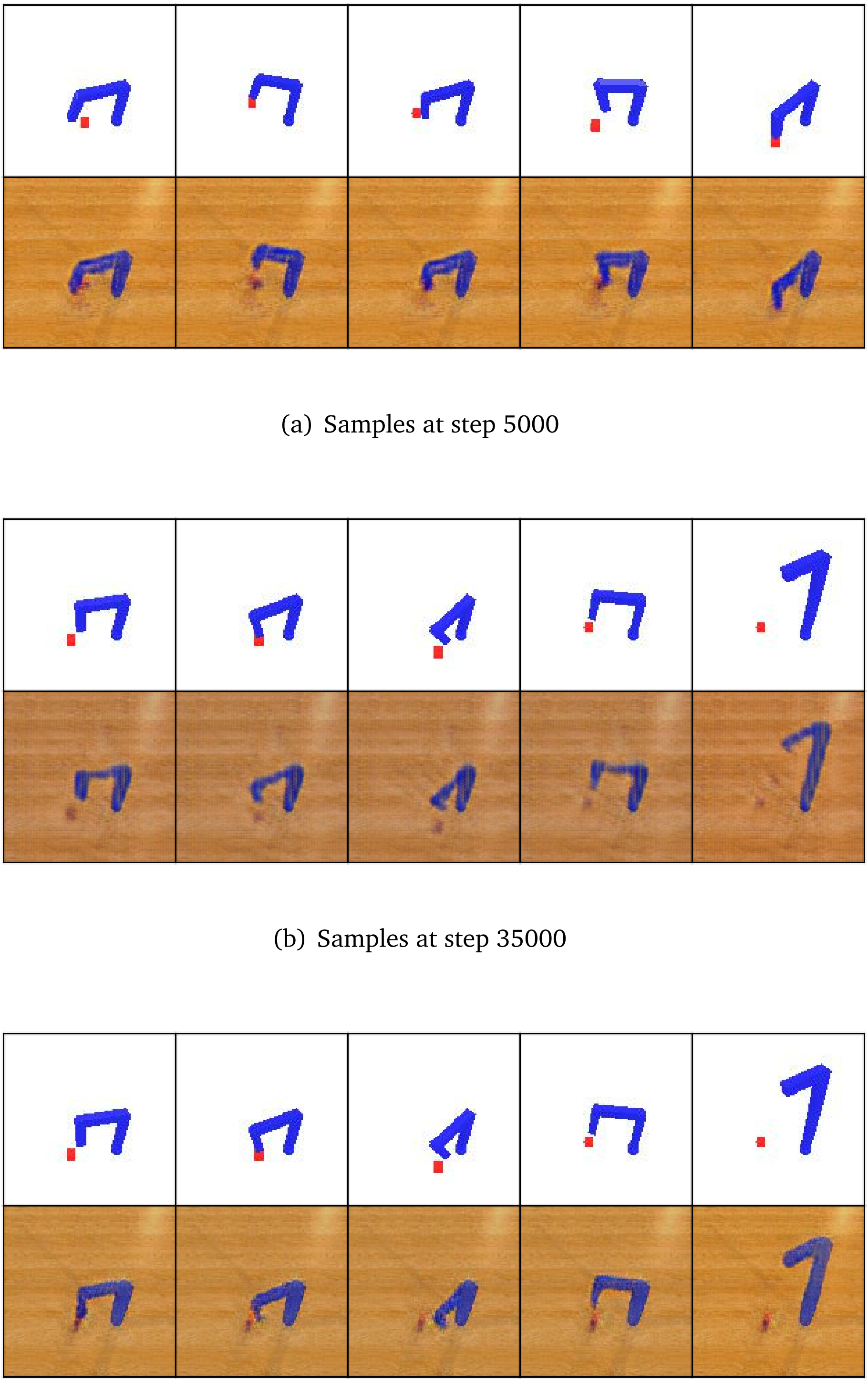
Figure 5.20: Fourth source to target transformation. Cube size remains the same while width of links are changed in a way similar to previous transformations. Lighting effects are added and the background is changed to a wood-like texture.

For the fourth source to target transformation, we introduce a more significant background change by adding a wood-like texture. The f_Y used to train the TaskGAN is trained on 8000 paired examples and had a 91% final distance success rate when tested in the target domain. Samples at various training steps can be found in Figure 5.21.

At step 5000, we find that the generator has learnt to recreate the background and match the pose of the arm. However, the generator ignores the position of the cube in the source domain and recreates it at the same position in the generated target domain image.

At step 35000, we find that arm pose and cube position are generally well matched. However, the quality of the image translation is poor. The generated background is noisy and of poor quality. The generated arm is noisy and does not match the smoother colours that are generated by translations in previous experiments. The cube, while matching in position, has a highly faded appearance.

Upon further training at step 40000, we find that the overall quality of the generated image improves. However, it once again fails to match the position of the cube and generates it at the same exact position in all generated target domain images. It seems that the generator trades off being able to match the object information in the source domain to creating a higher quality target domain characteristics. It is unclear whether further training can lead to better results, but due to lack of computational resources, this will not be attempted in this paper. Overall, the image translation is not considered successful for this source-target domain pairing. However, it shows promise and tweaks in various aspects such as the generator and discriminator architecture might lead to much better results.



(a) Samples at step 5000

(b) Samples at step 35000

(c) Samples at step 40000

Figure 5.21: Samples at various training steps for the fourth source to target transformation using a TaskGAN trained with 8000 paired examples. For each set of samples, the top row represents the source domain images x and the bottom row represents the generated target domain images $G(x)$.

In summary, the simpler the source-target domain transformation, the easier it is for the GAN to learn it. Utilizing the task loss of even a weakly trained f_Y tends to lead to much better results than just using a typical GAN losses. Target domains with complicated and noisy backgrounds tend to be hard to re-create without losing other information and need a well trained f_Y for better results. For best results, source and target domains are recommended to have clean and non-noisy backgrounds, similar colours, shapes and sizes. For application purposes, we would tend towards such a scenario where the source domain in a simulator is created to be visually similar to the target domain in reality.

Chapter 6

Training with Generated Source-Target Pairs

After discussing in depth the use of GANs for image translation in Chapter 5, we will investigate the use of these translated images to train neural networks for a robotic task in this chapter. Clearly, if there is a successful image translation from a source domain to a target domain, the labels from the source domain can be used for the paired generated target domain image. We will adapt the methods discussed in Chapter 4 to utilize the translated target domain images.

Section 6.1 covers a completely unsupervised method of performing image translation and training on a regression task. Section 6.2 covers methods of performing image translation with limited source-target domain pairs and training on a regression task. It will also include a more comprehensive study on the effects of utilizing image translation models for transfer learning on source to target domain transformations of various difficulties.

6.1 Training with CycleGAN images

This section describes a few methodologies to utilize the generated images from CycleGAN to train on a robotic task. Since the CycleGAN model can be trained in an unsupervised fashion, source domain images are assumed to possess labels and target domain images are unlabelled. This is to simulate the source-target domain pairing between simulator-reality and the aim to not require expensive annotations in reality.

6.1.1 Models

The models revolve heavily around the concept of extracting a feature vector from an image using a neural network. Figure 6.1 is a summary of the earlier discussion in Chapter 4 of the feature extractor model.

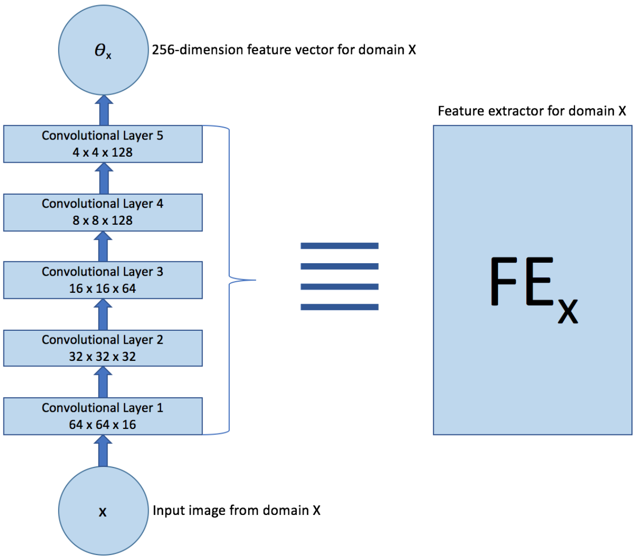
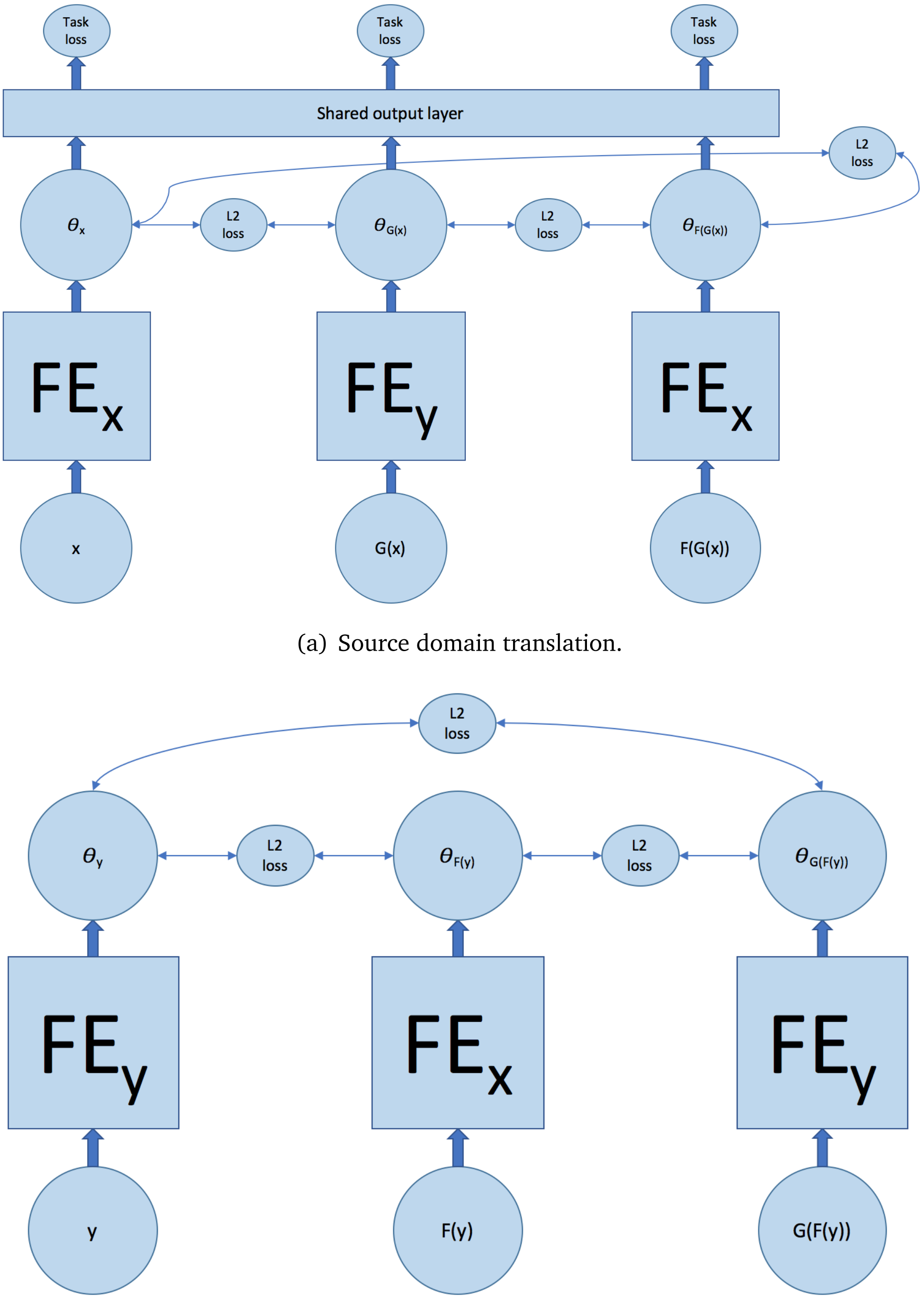


Figure 6.1: Feature extractor example. Images of input size 128x128x3 are mapped to a 256-dimension feature space.

- **Source domain translation**: Source domain images x, generated target domain image, $G(x)$, and reconstructed source domain image $F(G(x))$ are trained on the task with labels of x. L2 loss between feature layers are minimized. Figure 6.2a depicts the architecture.

- **Source + target domain translations**: In addition to training on the source domain translations, the L2 loss between the target domain translations are minimized. Specifically, target domain images y, generated source domain image $F(y)$, and reconstructed target domain image $G(F(y)$ have their corresponding feature layer L2 losses minimized. Figure 6.2b depicts the extensions to the architecture.

- **Transferring of labels from source to target domain**: All source domain images x are mapped to a feature space θ_x. Any nearest neighbours algorithm can be chosen to construct a tree from this feature space. The ball tree algorithm was chosen for this implementation. For each target domain image y, its corresponding feature space vector θ_y is used to query the tree. The nearest neighbour in the feature space will transfer its label to the target domain images y. This idea is inspired by the work in [18], where a similar idea of mapping source and target domain images to a shared feature space and transferring of labels to the nearest neighbour is done.



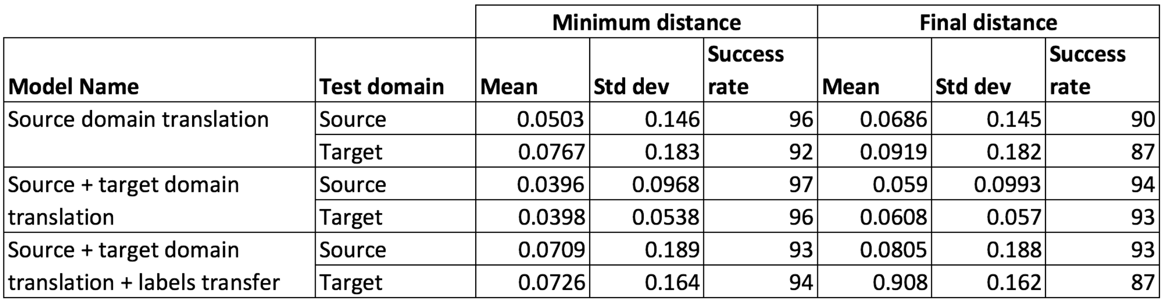
(a) Source domain translation.

(b) Target domain translation.

Figure 6.2: Training architectures for translated images with CycleGAN

6.1.2 Results

The source-target domain pairing used can be found earlier in Figure 5.6. All generated and reconstructed images are pre-computed and saved on disk in jpg format for quicker training. Models are trained for 10000 trainings steps of batch size 16 for each type of image. For the transfer of labels, the “Source + target domain translation” model is used to construct and find the nearest neighbours in the feature space. That model is further fine-tuned using the newly labelled target domain examples with the target domain translation.



		Minimum distance			Final distance		
Model Name	**Test domain**	**Mean**	**Std dev**	**Success rate**	**Mean**	**Std dev**	**Success rate**
Source domain translation	Source	0.0503	0.146	96	0.0686	0.145	90
	Target	0.0767	0.183	92	0.0919	0.182	87
Source + target domain translation	Source	0.0396	0.0968	97	0.059	0.0993	94
	Target	0.0398	0.0538	96	0.0608	0.057	93
Source + target domain translation + labels transfer	Source	0.0709	0.189	93	0.0805	0.188	93
	Target	0.0726	0.164	94	0.908	0.162	87

Figure 6.3: Results of training on translated images of CycleGAN.

Training on the source domain translation only leads to a final distance success rate of 90% and 87% in the source and target domains respectively. With the addition of minimizing the L2 loss on the target domain translation, both source and target domain performances can be improved to a final distance success rate of 94% and 93% respectively. Both generated and reconstructed images can be considered as noisy versions of the originals, but by training on such noisy images, it encourages the neural networks to identify domain invariant features which are essential to the task, leading to an overall improved performance.

The motivation for introducing labels transfer is to utilize the true target domain images for training on the task loss since they are of high quality and it is a waste to not use them for training. As seen in Figure 6.4, the majority of nearest neighbour pairings are of good quality. However, training the source + target domain translation model with the newly transferred labels did not lead to improved results. Final distance success rates fell slightly in the source domain from 94% to 93%, while performance in the target domain fell from 93% to 87%. This could be attributed to the fact that the pairings were not perfect and that the translated pairings were of greater quality. However, these results should not lead to the conclusion of this approach being unviable. In this source-target pairing, the translations were relatively easy to learn and of high quality. However, in other more complex scenarios, high quality image translation models might not be learnt and supplementing the training with labels transfer might lead to performance improvements in those scenarios.

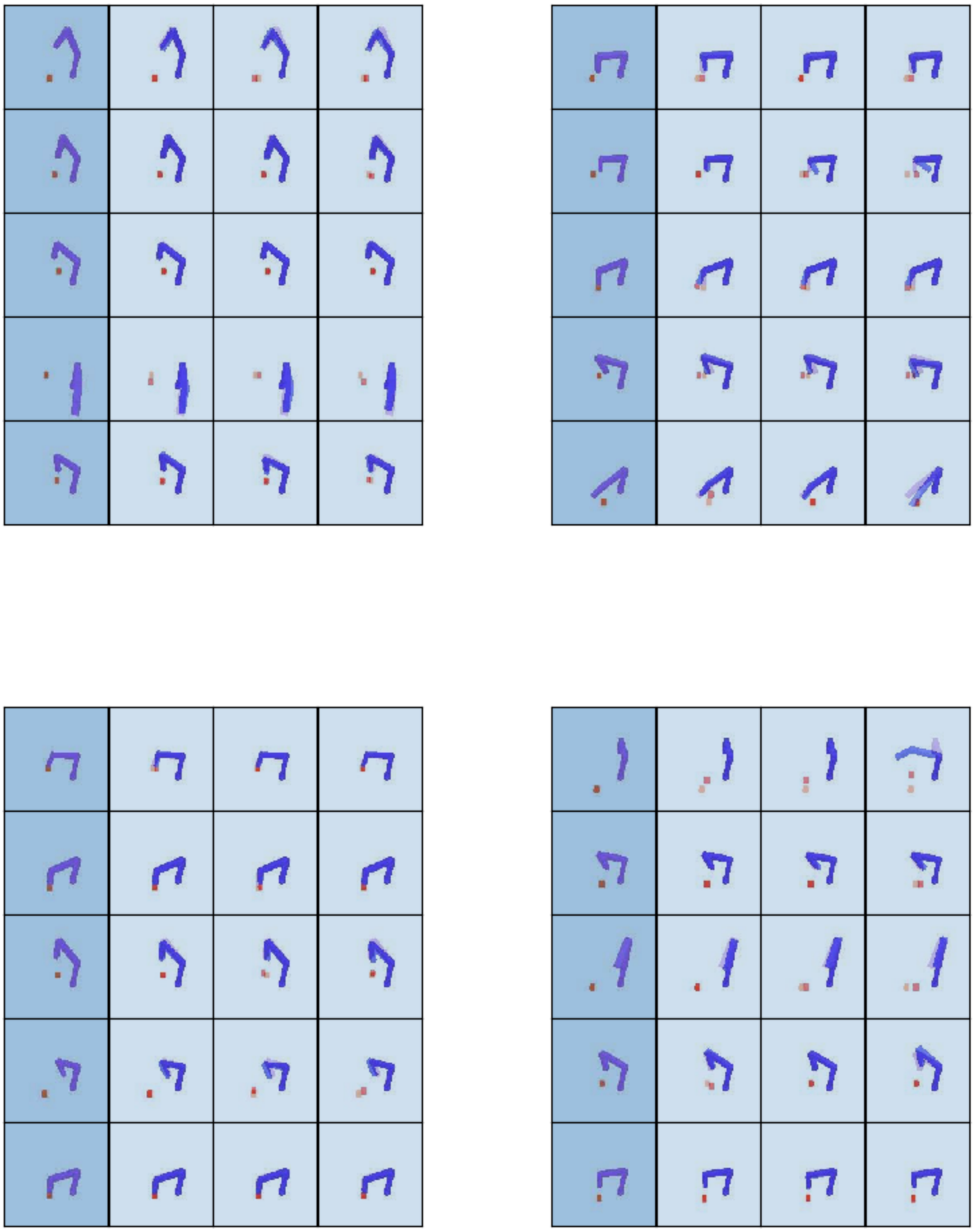


Figure 6.4: Nearest neighbour samples. Each set of samples represents 5 rows of randomly chosen target domain images and its 3 nearest neighbour source domain images superimposed onto itself. Left to right for each row: Original target domain image, superimposed image with nearest neighbour, superimposed image with 2nd nearest neighbour, superimposed image with 3rd nearest neighbour.

6.2 Training with TaskGAN images

Under the scenario where a small number of paired source-target domain examples are present, we can utilize these pairs to train a TaskGAN image translation model as described in Chapter 5. This section describes approaches of using some combination of unpaired source domain images x, paired source-target domain images x-y, and paired source to generated domain images x-$G(x)$ to improve performances in the target domain.

6.2.1 Model details

There are 4 models being evaluated in the following section. The first model is trained on source domain images only while the rest are trained in a manner similar to the 2 separate feature extractor model with L2 loss between feature layers and pre-training as described in Chapter 4. Models are trained for 15000 steps with a learning rate of 0.001 and batch size of 16. More details on the 4 different methods are as follows:

- **Source**: This model is trained on the source domain images x only. This serves as a benchmark against the performances of other models with transfer learning.

- **Source + Paired**: This model is trained on unpaired source domain images x and paired source-target domain images x-y. The aim is also to use only a small number of x-y pairs and evaluate the performance improvements under such conditions.

- **Source + Translated**: This model is trained on unpaired source domain images x and paired source to generated target domain images x-$G(x)$. Since source domain images and labels are assumed to be easily obtainable, we allow the use of a larger amount of x-$G(x)$ pairs and test the potential benefits of such an approach.

- **Source + Paired + Translated**: This model is trained on unpaired source domain images x, paired source-target domain images x-y, and paired source to generated target domain images x-$G(x)$. In a similar vein of thought to the above methodologies, we limit x-y pairs to a small amount and allow a larger amount of x-$G(x)$ pairs. This model determines the effectiveness of training with a combination of a small number of true paired examples and a large number of generated paired examples.

6.2.2 Results

In this section, an evaluation of performances from 4 different methods on various target domains will be done. In particular, two sets of experiments will be done to study the performance changes under different conditions. The first set of experiments studies target domains with a wider range of changes while the second set of experiments investigates the effect of varying only the cube size in the target domain.

Figure 6.5 shows the 3 target domains for the first set of experiments where the models are trained and tested on. The changes from the source domain are detailed as follows:

- **Target A**: Width links are modified with the base link becoming thicker and the tip link becoming thinner. Colours of cube and arm are slightly changed. Cube size is slightly reduced.
- **Target B**: Width links are modified with the base link becoming thicker and the tip link becoming thinner. Colours of cube, arm, and background are slightly changed. Lighting effects are added to simulate realism.
- **Target C**: Width links are modified with the base link becoming thicker and the tip link becoming thinner. Colours are changed more significantly, with the base link becoming red, the middle link becoming green, the tip link having a slight colour change, and the cube's colour is inverted from red to cyan.

The unpaired source domain x dataset used for this set of experiment contains 22656 images, the small x-y pairs dataset contains 800 pairs of images, and the large x-$G(x)$ pairs dataset contains 8000 pairs of images. The f_Y mapping function for the TaskGANs for Target A and C were trained with the small x-y pairs dataset of 800 pairs of images. It was more difficult to achieve a success TaskGAN model for Target B and the f_Y mapping function for the TaskGAN for Target B was trained with a larger x-y pairs dataset of 8000 pairs of images. The TaskGAN training itself was done without the use of the pairs (i.e. without the L1 loss). Further details for the training of TaskGANs and image translation samples can be found in Chapter 5.

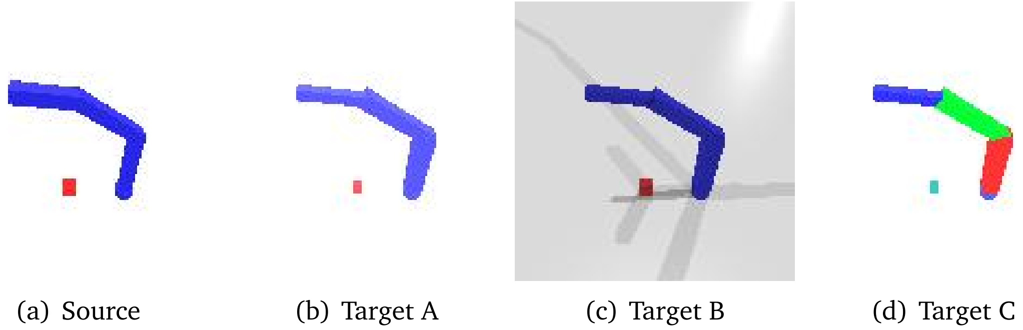
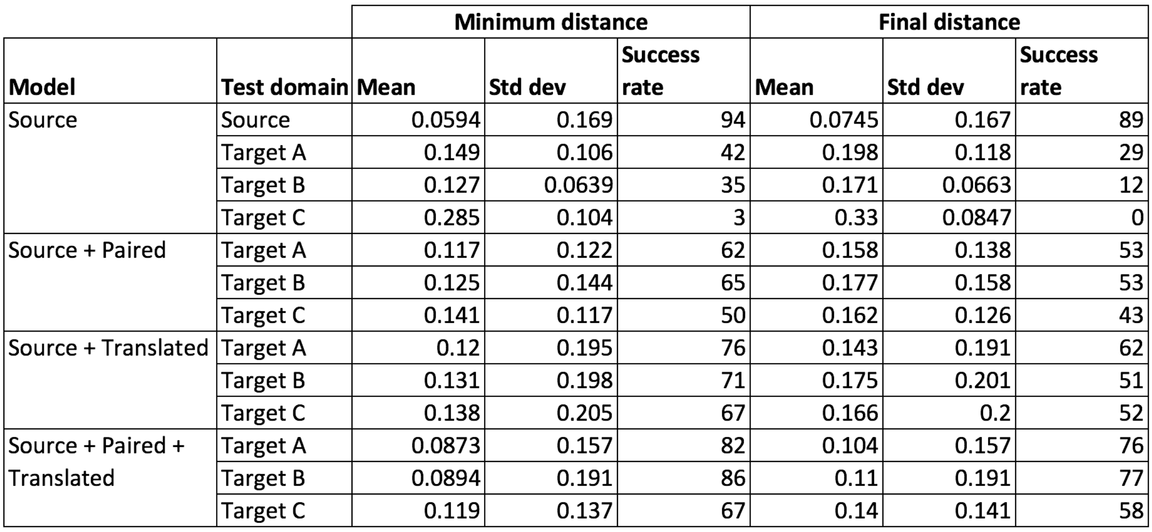

(a) Source (b) Target A (c) Target B (d) Target C

Figure 6.5: Sample image from the source domain and the various target domains being tested on.



		Minimum distance			Final distance		
Model	**Test domain**	**Mean**	**Std dev**	**Success rate**	**Mean**	**Std dev**	**Success rate**
Source	Source	0.0594	0.169	94	0.0745	0.167	89
	Target A	0.149	0.106	42	0.198	0.118	29
	Target B	0.127	0.0639	35	0.171	0.0663	12
	Target C	0.285	0.104	3	0.33	0.0847	0
Source + Paired	Target A	0.117	0.122	62	0.158	0.138	53
	Target B	0.125	0.144	65	0.177	0.158	53
	Target C	0.141	0.117	50	0.162	0.126	43
Source + Translated	Target A	0.12	0.195	76	0.143	0.191	62
	Target B	0.131	0.198	71	0.175	0.201	51
	Target C	0.138	0.205	67	0.166	0.2	52
Source + Paired + Translated	Target A	0.0873	0.157	82	0.104	0.157	76
	Target B	0.0894	0.191	86	0.11	0.191	77
	Target C	0.119	0.137	67	0.14	0.141	58

Figure 6.6: Full results of models trained and tested under different conditions.

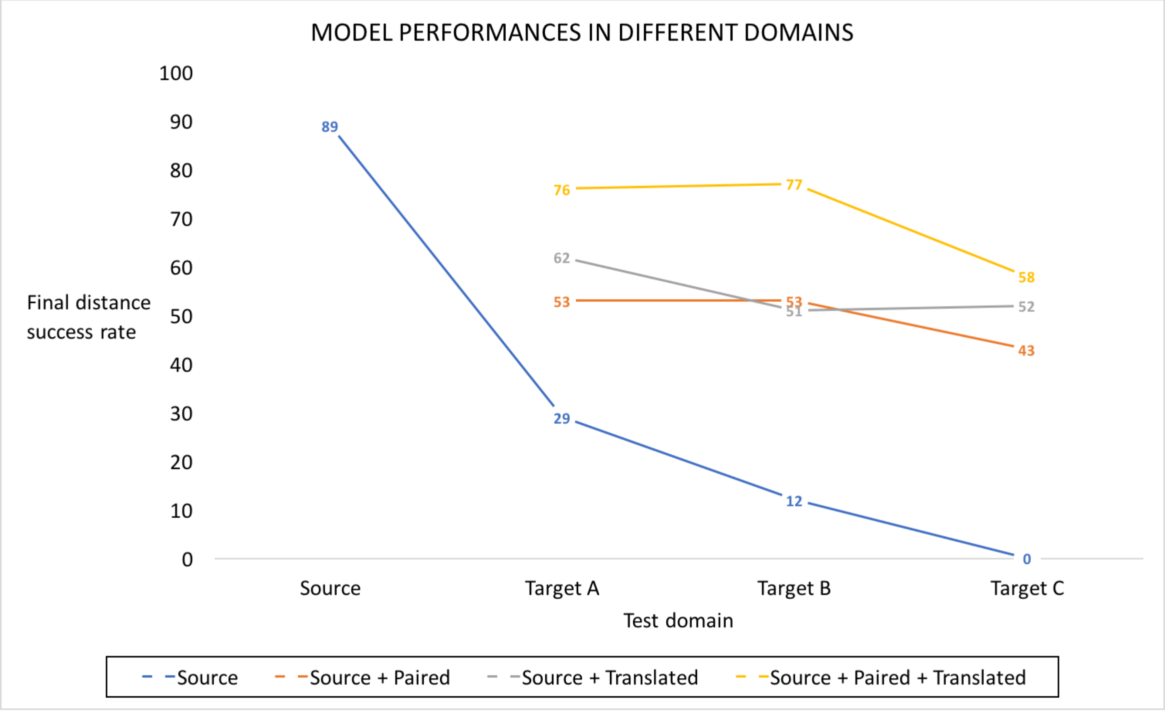


Figure 6.7: Final distance success rates of models trained and tested under different conditions.

The full results can be found in Figure 6.6 and a visual summary in the form of a graph can be found in Figure 6.7. While it is difficult to quantify directly the gap between source-target transformations, the performance of the "Source" in the various target domains can be used as a form of approximation. From this perspective, Target A represents the simplest or smallest source-target domain gap with the source model achieving final distance success rate of 29% while Target C is the most complex or largest source-target domain gap as the “Source” model completely fails with a final distance success rate of 0%.

While we are able to approximately rank the difficulty of the transformations in the ascending order of Target domains A, B, and C, we find that training a TaskGAN for Target B to be the most difficult to achieve. Complex background changes much more difficult for an image translation model to achieve. As described earlier, it is important to note that the TaskGANs for Target B uses a large number x-y pairs in its training (8000 pairs to be exact, compared to the 800 pairs for the other TaskGANs) in order to achieve a reasonable translation. It is possible that the number of pairs can be reduced from 8000, but that was not attempted due to the lack of computational resources. It also goes against the motivation of this section which attempts to build a model to perform in the target domain from a small number of x-y pairs. However, we included it as a proof of concept to show the possible impact if an image translation was successfully trained. To reiterate, the TaskGANs for Target A and Target C were trained with a small number of pairs (800), and the TaskGAN for Target B was trained with a large number of pairs (8000).

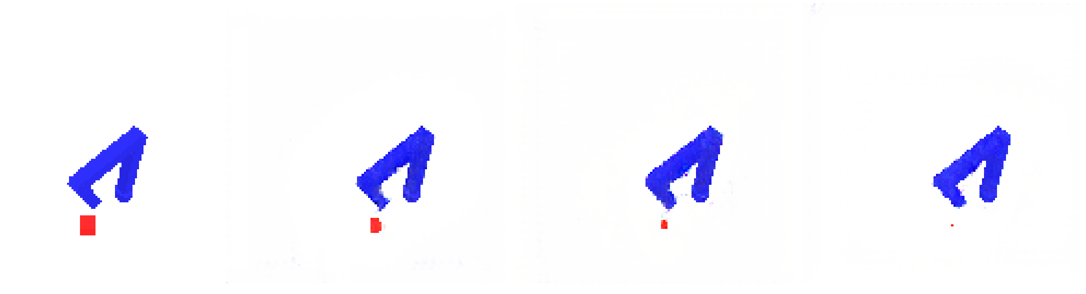
In general, we find the most performance gains from the “Source + Paired + Translated” method of training with both a small amount x-y pairs and a large amount of x-$G(x)$ pairs. The other two methods of “Source + Paired” and “Source + Translated” had performance gains over the “Source” method, which suggests that both types of image pairs are helpful. In test domains Target A and C, we find that having large number of translated x-$G(x)$ pairs worked better than a small number of x-y pairs, while the opposite was true in test domain Target B. A plausible reason is the quality of the translated images $G(x)$. Target B involved the reconstruction of a complex background with shadows and lighting and these generated target domain images tend to be noisy compared to the true samples y. Hence, generated target domain images $G(x)$ in Target B are noisier and less effective in improving performance when used alone.

However, the noisy reconstruction $G(x)$ are not useless. By combining both x-y and x-$G(x)$ pairs in the “Source + Paired + Translated” method in domain Target B, a respectable final distance success rate of 77% could still be achieved. By training on both the noisy images and the true images, the neural network learns to ignore the noise and the large number of $G(x)$ becomes a valuable source of information.

In the second set of experiments, we would like to test how the models will perform when varying only a single factor of the source domain. This is to test how performance changes on target domains with small incremental changes from the source domain. The variable chosen is the cube length. The original cube length in the source domain is 7, and target domains with cube lengths of 5, 3, and 1 will be tested on.

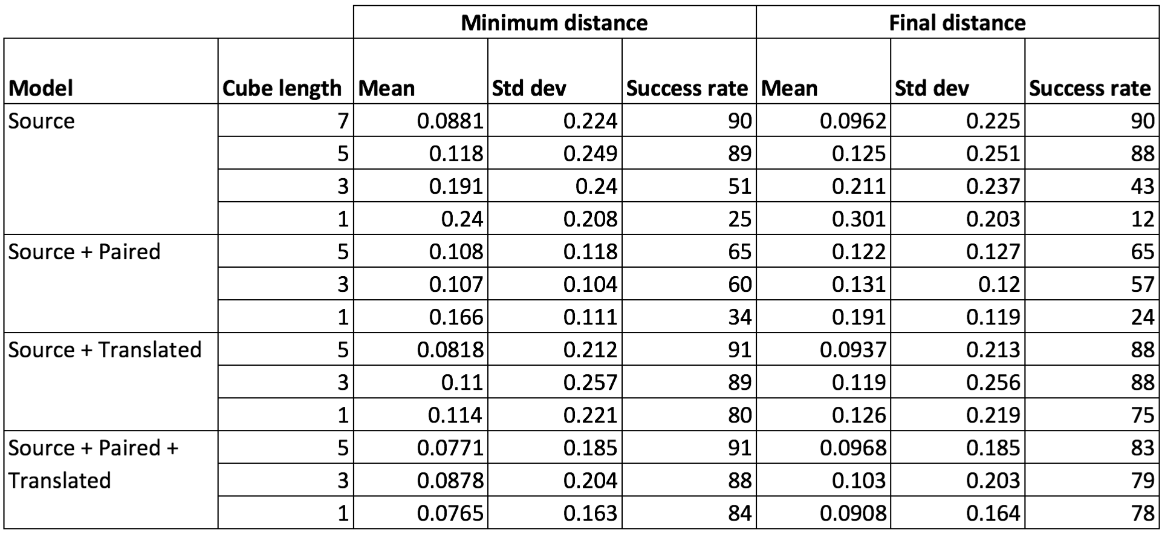
In all previous experiments and tests, the images have been saved in jpg format. However, this led to some loss in information which affects the image quality much more significantly when the cube size is small. Hence, some configurations to the datasets were made for this set of experiments and all images were saved in png format. The unpaired source domain x dataset used here contains 8000 images, the small x-y pairs dataset contains 800 pairs of images, and the large x-$G(x)$ uses the same unpaired source domain x dataset, which makes 8000 pairs of images. We train TaskGANs, G_{c5}, G_{c3}, G_{c1}, to map from the source domain to the respective target domains with a mapping function f_Y trained using the "Source + Paired" method and L1 loss. Samples of source-target pairs can be found in Figure 6.8 and samples of source to generated target pairs can be found in Figure 6.9. Generally, we find that image translation quality is good.

Figure 6.8: Sample image from the source domain and the target domains with decreasing cube length. Cube lengths from left to right: 7, 5, 3, 1.



(a) Source x (b) $G_{c5}(x)$ (c) $G_{c3}(x)$ (d) $G_{c1}(x)$

Figure 6.9: Sample image from the source domain and generated target domain images. From left to right: Source domain image, generated target domain images with cube length 5, 3, and 1.



		Minimum distance			Final distance		
Model	Cube length	Mean	Std dev	Success rate	Mean	Std dev	Success rate
Source	7	0.0881	0.224	90	0.0962	0.225	90
	5	0.118	0.249	89	0.125	0.251	88
	3	0.191	0.24	51	0.211	0.237	43
	1	0.24	0.208	25	0.301	0.203	12
Source + Paired	5	0.108	0.118	65	0.122	0.127	65
	3	0.107	0.104	60	0.131	0.12	57
	1	0.166	0.111	34	0.191	0.119	24
Source + Translated	5	0.0818	0.212	91	0.0937	0.213	88
	3	0.11	0.257	89	0.119	0.256	88
	1	0.114	0.221	80	0.126	0.219	75
Source + Paired + Translated	5	0.0771	0.185	91	0.0968	0.185	83
	3	0.0878	0.204	88	0.103	0.203	79
	1	0.0765	0.163	84	0.0908	0.164	78

Figure 6.10: Full results of models trained and tested under varying cube lengths.

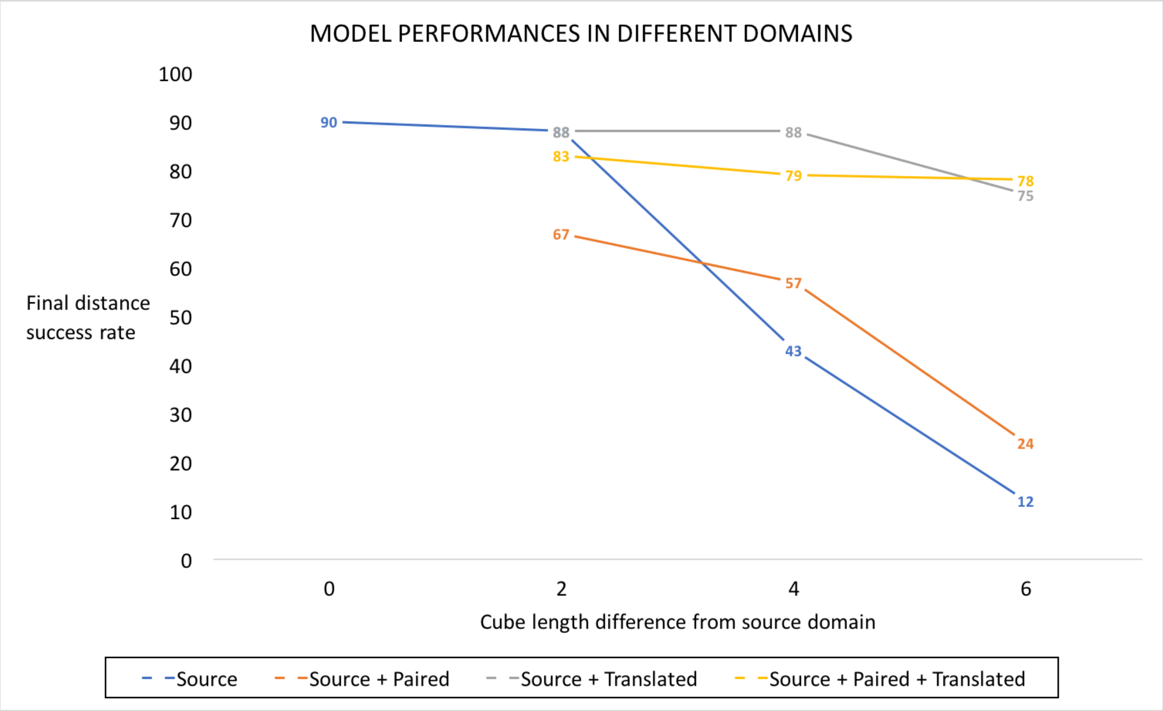


Figure 6.11: Final distance success rates of models trained and tested under varying cube lengths.

Full results can be found in Figure 6.10 and a summary of the performances in the form of a graph can be found in Figure 6.11. Interestingly, the results are not fully consistent with what was found in the first set of experiments. In the first set of experiments where target domains were much more different, the “Source + Paired + Translated” method outperformed all other methods. However, in these target domains which are arguably much more similar to the source domain, the “Source + Paired + Translated” method does not always work best. There are a few possible reasons as to why the “Source + Translated” methodology performs better in the cases where the cube length difference is 2 and 4.

Firstly, the “Source + Translated” methodology performs better in the cases where the cube length difference is 2 and 4 is the overfitting caused by the small number of x-y pairs. The decrease in performance of the “Source” model is relatively lesser compared to the previous set of experiments, which can be interpreted as these two target domains being highly similar to the source domain. Hence, training with a small number of pairs can lead to overfitting and a decrease in performance. This idea is further reinforced by the performance of the “Source + Paired” model when the cube length difference from source domain is 2. In that scenario, the “Source + Paired” model had a final distance success rate of 67%, worse than the “Source” model which had a final distance success rate of 88%. Furthermore, the quality of translated images were good and just using the translated images as training for a target domain task was sufficient in producing good performances of 88% in both cases.

As the difference from the source domain increases, the results once again became consistent to what was found in the first set of experiments. Training with some x-y pairs with the “Source + Paired” method led to slightly improved results, training with a larger number of x-$G(x)$ pairs with the “Source + Translated” method led to better results, and training with both types of pairs with the “Source + Paired + Translated” method led to the best results. The overfitting argument does not hold here since the target domain is far enough from the source domain that the small number of x-y pairs proved to be valuable training data as opposed to causing overfitting.

Overall, we find that if the source and target domains are highly similar, training with the small amount of x-y pairs tend to be detrimental to the overall performance. Performance of the “Source + Translated” method of using x-$G(x)$ pairs tends to decrease as the source to target domain gap increases. At larger source to target domain gaps, training with the small x-y pairs has a positive effect as it helps the neural network learn to ignore the noise in generated target domain images $G(x)$. For most realistic applications, it is highly likely that the gap between simulated data and real data falls under the category of being a large source to target domain gap, where the “Source + Paired + Translated” model will work best.

Chapter 7

Conclusions and Future Work

In this paper, we presented a method of training generative adversarial networks for image translation and pairing across domains, TaskGAN. This requires the training of a neural network f_Y on a task in the target domain Y and we do it by using some number of source-target domain pairs x-y. We showed the viability of using this model to generate target domain images $G(x)$ that recreates target domain characteristics such as background, shape deformations, and colour changes while retaining source domain information such as arm pose and cube position. Results show good image translation and pairing quality for simple to moderately complex source to target domain transformations. The method's potential in applications where simulated source domain images can be enhanced to look more like target domain images and used as training data was demonstrated in various target domains as well. We also introduced a novel dataset that is easily configurable for the training and testing of robot control for transfer learning purposes.

A limitation of TaskGAN is that complicated transformations can be unsuccessful even with a well trained f_Y with many x-y pairs. There are a few areas that can be worked on to improve this. Firstly, improvements in generator and discriminator architectures could lead to higher quality of translations. Also, the f_Y mapping trained in this paper was highly dependent on providing it some x-y pairs. Pairing and manual annotations are expensive and we want to reduce the dependency on x-y pairs or even remove the need of it. While we investigated some unsupervised and semi-supervised methods in this paper, it was not helpful to performance. It would be interesting to see future work that focuses on these branches of methods to reduce the dependence on paired data between source and target domains.

Another point worth noting is that the GANs trained in this paper are trained over a much shorter period of time compared to other similar works due to the lack of computational resources. It is still yet to be seen if extended training periods can lead to significantly better results in the more complicated transformations. Also, training and tests were completely carried out in simulated environments. Further experiments that utilize a real robot for target domain data generating and testing would be highly interesting.

Bibliography

[1] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 1321–1326, IEEE, 2013.

[2] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *arXiv preprint arXiv:1703.10593*, 2017.

[3] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[7] F. Zhang, J. Leitner, B. Upcroft, and P. Corke, "Vision-based reaching using modular deep networks: from simulation to the real world," *arXiv preprint arXiv:1610.06781*, 2016.

[8] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," *arXiv preprint arXiv:1610.04286*, 2016.

[9] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 806–813, 2014.

[10] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

[11] R. Brooks, "A robust layered control system for a mobile robot," *IEEE journal on robotics and automation*, vol. 2, no. 1, pp. 14–23, 1986.

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[13] A. Rozantsev, M. Salzmann, and P. Fua, "Beyond sharing weights for deep domain adaptation," *arXiv preprint arXiv:1603.06432*, 2016.

[14] A. Gretton, K. M. Borgwardt, M. Rasch, B. Schölkopf, A. J. Smola, *et al.*, "A kernel method for the two-sample-problem," *Advances in neural information processing systems*, vol. 19, p. 513, 2007.

[15] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *Journal of machine learning research*, vol. 2, no. Nov, pp. 67–93, 2001.

[16] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, no. Mar, pp. 723–773, 2012.

[17] A. L. Gibbs and F. E. Su, "On choosing and bounding probability metrics," *International statistical review*, vol. 70, no. 3, pp. 419–435, 2002.

[18] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," in *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[19] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *arXiv preprint arXiv:1703.06907*, 2017.

[20] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[21] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2414–2423, 2016.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[23] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint arXiv:1611.07004*, 2016.

[24] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," *arXiv preprint arXiv:1612.07828*, 2016.