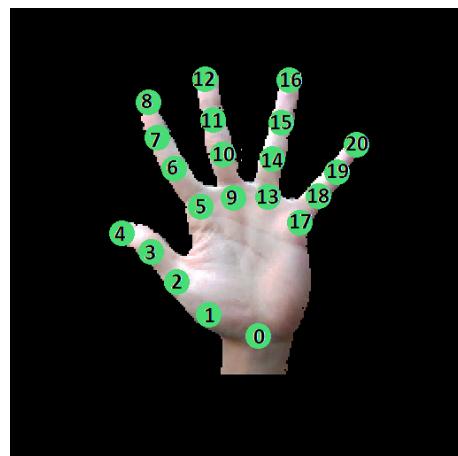


Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2018

---



Project Title: **Depth to Colour Translation for 3D Hand Pose Estimation From Monocular RGB With Generative Adversarial Networks**

Student: **Aaron Low Weng Soon**

CID: **00920886**

Course: **EEE4**

Project Supervisor: **Dr. Tae-Kyun Kim**

Second Marker: **Dr. Krystian Mikolajczyk**

## Abstract

With the recent development of deep learning techniques, 3D hand pose estimation has experienced great improvements. Recent methods require the collection of large experimental datasets which is tedious and difficult especially in the annotation of ground-truth 3D joint positions. Various methods have been devised which has resulted in many available datasets with most of the comprehensive datasets only catering to systems that operates on depth information. In this paper we propose a system that takes advantage of these available datasets to create a similar dataset for systems that operates on RGB information. Our method employs generated adversarial networks (GAN) which learns a mapping from existing depth-based images to RGB-based images. Specifically, we attempt to convert the benchmark BigHand dataset which is currently the most comprehensive dataset for 3D hand pose estimation with depth images. We experiment with various GAN image translation systems and analyse the quality of images. We also compare our generated dataset with existing datasets on a baseline hand pose estimator and demonstrate that our generated dataset can produce comparable and even superior results.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>3</b>
2.1	Related Works . . . . .	3
<b>3</b>	<b>Depth-RGB Dataset Generation</b>	<b>6</b>
3.1	Image acquisition . . . . .	6
3.2	Preprocessing . . . . .	6
3.3	Depth-RGB Dataset . . . . .	8
3.3.1	Analysis Dataset . . . . .	8
3.3.2	Comprehensive Dataset . . . . .	9
3.4	Implementation . . . . .	10
<b>4</b>	<b>Generative Adversarial Network</b>	<b>10</b>
4.1	Baseline GAN (pix2pix) . . . . .	10
4.1.1	Objective function . . . . .	11
4.1.2	Network architecture . . . . .	11
4.2	Wasserstein Loss . . . . .	13
4.3	Laplacian Pyramid . . . . .	14
4.4	Implementation . . . . .	16

<b>5 Hand Pose Estimation</b>	<b>17</b>
5.1 Network Architecture . . . . .	17
5.2 Implementation . . . . .	18
<b>6 GAN Evaluation</b>	<b>19</b>
6.1 Training Analysis . . . . .	20
6.1.1 Baseline GAN . . . . .	20
6.1.2 Wasserstein Loss . . . . .	21
6.1.3 Laplacian Pyramid . . . . .	22
6.2 Evaluation of Generated Images . . . . .	25
6.2.1 Baseline Generated Images . . . . .	25
6.2.2 Learning Rate . . . . .	28
6.2.3 Discriminator Patch Size . . . . .	29
6.2.4 GAN System . . . . .	31
6.3 Evaluation on Comprehensive Dataset . . . . .	32
6.3.1 Training Analysis . . . . .	33
6.3.2 Evaluation of Generated Images . . . . .	35
<b>7 Hand Pose Estimation Evaluation</b>	<b>37</b>
7.1 Results . . . . .	38

<b>8 Discussion and Conclusion</b>	<b>41</b>
<b>9 Future Work</b>	<b>42</b>
<b>10 Acknowledgements</b>	<b>43</b>
<b>Appendices</b>	<b>50</b>
<b>A Segmented hand mask</b>	<b>50</b>
<b>B Baseline GAN</b>	<b>50</b>
B.1 Network architecture . . . . .	50
B.1.1 Generator architecture . . . . .	51
B.1.2 Discriminator architecture . . . . .	51
B.1.3 Settings . . . . .	51
B.2 Optimisation . . . . .	52
B.2.1 Settings . . . . .	52
B.3 Training Analysis . . . . .	53
B.3.1 Training Samples . . . . .	53
B.3.2 Epoch 100 . . . . .	53
B.4 Learning Rate . . . . .	54
B.4.1 0.00002 . . . . .	54

B.4.2	0.002 . . . . .	55
B.5	Patch Size . . . . .	56
B.5.1	32x32 . . . . .	56
B.5.2	286x286 . . . . .	57
B.6	Comprehensive Dataset . . . . .	58
B.6.1	1025 . . . . .	58
<b>C</b>	<b>Laplacian Pyramid</b>	<b>59</b>
C.1	Generated Images . . . . .	59
C.2	Training Analysis . . . . .	60
C.2.1	h1 . . . . .	60
C.2.2	h0 . . . . .	61
<b>D</b>	<b>Hand Pose Estimation</b>	<b>62</b>
D.1	Network Architecture . . . . .	62
D.1.1	Settings . . . . .	62
D.2	Optimisation . . . . .	63
D.2.1	Settings . . . . .	63
D.3	Joint positions . . . . .	64
D.4	Training Analysis . . . . .	65

D.5 Average Joint Error . . . . .	66
-----------------------------------	----

# 1 Introduction

Articulated hand pose estimation is a challenging problem in computer vision with many applications in the areas of human computer interaction, robotics, health care, and sign language recognition. Inferring the pose of a human hand is difficult due to articulated objects having many degrees of freedom, self-occlusion and self-similar parts.

Recently, an increasing amount of research has been carried out in the field of articulated hand pose estimation which has led to significant progress and the proposals of numerous systems [1, 2, 3, 4]. While there are various methodologies available, the current state-of-the-art systems employ deep learning and convolutional neural networks (CNN) [5]. A majority of these systems allow for marker-free hand pose estimation with real-time hand tracking. Most of these systems are completely reliant on depth information for detecting hand features and learning joint positions. As a result, these systems require a depth camera to be implemented which limits its practical appeal. Depth cameras are sensitive to environment and are susceptible to interference. They are also not as widely available as RGB cameras which can be found in most smart phones and laptops.

In this project we focus on the problem of inferring 3D hand pose from single RGB images. To further increase the practicality of our solution, we also restrict ourselves to monocular views using single RGB images. Multiview setups [6, 7] have been shown to handle the problem of self-occlusion well but monocular setups are preferable as they reduce setup overheads and are more readily applicable.

Most recent methods which rely on deep learning techniques benefit from training on larger datasets. A significant bottleneck in hand pose estimation research comes from obtaining high quality, comprehensive datasets consisting of a variety of poses, viewpoints, shapes and sizes. Obtaining large sets of input (depth hand image) and ground truth (3D joint annotation) pairs is not a trivial task. The main difficulty stems from the annotation of 3D hand joint positions. Annotating the joint positions manually is very time consuming and could lead to inaccurate data [8]. Data-capturing gloves, on the other hand, can increase the accuracy of ground truth data significantly but input depth information becomes noisy and corrupted [9]. Other methods

which combine semi-automatic inferencing of the joint locations with some manual refinement were shown to be successful but still proved to be time consuming and resulted in limited datasets [3, 1]. The BigHand [10] dataset uses compact external magnetic sensors and is currently the largest and most comprehensive set. Most of these existing datasets are developed specifically for systems which take depth images as input [3, 1, 10, 11].

The task of obtaining datasets which take RGB images as input comes with additional difficulties. Obtaining accurate 3D hand joint annotation for RGB images is difficult as any external sensor would corrupt the RGB images even more so than when used for obtaining depth images. RGB hand image datasets do exist but collection methods have been restricted to manually annotating ground truth joint positions [12, 13] or synthetically creating models [14, 15]. Synthetic data may have accurate ground truths but often do not generalise well to real world images. The NYU [16] dataset contains RGB-3D joint annotation pairs but is limited in size and pose variation. The images also only provide colour for pixels with valid depth data. The GANerated [17] dataset generates realistic hand images from synthetic hand images using generative adversarial networks (GAN) [18]. So far, these datasets are still limited in size and pose variation compared to depth-based alternatives.

The aim of this project is to leverage the large-scale BigHand [10] benchmark dataset which is a million-scale depth image dataset that contains a wide variety of poses to create a hand pose estimation dataset for RGB images. The original RGB images from the BigHand dataset are not sufficient as they are corrupted with noise (as the external magnetic sensors are present in the images).

We experiment with various methods to generate RGB images from depth images provided by the BigHand dataset using GANs as inspired by *Mueller et al.* [17]. Our goal is to train a GAN system that would allow high resolution mapping of RGB images from depth images. We demonstrate our networks ability to generate RGB hand images of reasonable quality. Using the generated dataset, we evaluate quantitatively by training a baseline hand pose estimator [3]. We show that our generated dataset can achieve state-of-the-art performance compared to existing benchmarks such as the BigHand [10] and SynthHands [19] datasets. Our intention is to create a comprehensive dataset and pave the way for further exploration in hand pose estimation with

RGB images. Code for the project is available at <https://github.com/aaronlws95/gan-depth-to-rgb>.

## 2 Background

### 2.1 Related Works

**Hand Pose Estimation** In the past few years, numerous methodologies have been proposed to tackle the challenge of estimating the pose of an articulated object. As the main focus of recent techniques, we consider markerless camera based methods on monocular images. *Oikonomidis et al.* [2] proposes a technique based on Particle Swarm Optimisation (PSO). *Tang et al.* [1] uses a latent regression forest and recursively divides the input into two cohesive sub-regions until each sub-region contains only one skeletal joint. *Tompson et al.* [3] uses a CNN for detection of key hand points in 2D on a multi-resolution image pyramid followed by post refinement into hand pose with inverse kinematics optimisation. *Zhou et al.* [20] showed that a forward kinematics process, that is, mapping from the joint angles to joint locations of an articulated hand pose could be used in training the CNN. This removes the need of post-processing and directly regresses the 3D joint coordinates with a CNN. *Oberweger et al.* [21] does similarly, but instead enforces a prior on the 3D pose and predict the parameters of the pose in a lower dimensional space. Hybrid methods that use a discriminative (learning based) method to obtain the initial stage of a generative (model based) method [3, 4, 22] have been known to show the best performance on benchmark datasets. The current paradigm in hand pose estimation are methods which are entirely reliant on using depth information. However, efforts have been made to further research in hand pose estimation from RGB images [14, 17, 6, 7]. Of these, *Zimmermann et al.* [14] and *Mueller et al.* [17] propose systems that estimate 3D hand pose in a monocular setting.

**Convolutional Neural Networks** Convolutional Neural Networks (CNN) are a class of feed-forward artificial neural networks inspired by biological processes present in the visual cortex. CNNs have shown to be useful in solving problems associated with computer vision such as object classification, face detection, image recognition [5, 23, 24] and now in hand pose estimation [4, 3]. Compared to standard artificial neural networks, CNNs have a 3-dimensional

neuron arrangement, local connectivity and shared weights which allows it to have better generalisation on vision problems. To date, numerous modifications have been applied to the original CNN implementation. *Krizhevsky et al.* [25] uses deep convolutional networks for image classification. *He et al.* [26] proposes ResNet, which applies residual learning to ease the training of deep convolutional networks. *Ronneberger et al.* [27] proposes U-net, a system that uses data augmentation and reduces the need for large training datasets. In terms of hand pose estimation, *Tompson et al.* [3] uses a multi-resolution image pyramid with three parallel convolutional network layers to obtain heat-maps of key hand joints. *Ye et al.* [4] integrates cascaded and hierarchical regression into a CNN framework and enforces kinematic constraints on hand pose estimation.

**Hand Pose Datasets** Dataset collection has proven to be an important aspect of hand pose estimation. It has been observed that for current state-of-the-art methodologies that employ CNNs, their performance scale well with the size of datasets used and the variation of poses present in the dataset. Besides that, good benchmark datasets are also necessary for effective comparison between hand pose estimation algorithms. Most of these datasets consist of depth hand images together with their corresponding ground truth joint annotations. However, 3D joint annotations are difficult to obtain with manual methods such as obtained by the MSRA14 [8] dataset. This is mainly because it is too tedious to annotate and also prone to human error. On the other hand, using an external sensor such as data-gloves would provide accurate joint annotations but as a result it would also slightly corrupt the depth information [9]. Some methods use a combination of semi-automatic inferencing of joint locations with some manual refinement such as the MSRA15 [11], NYU [3], and ICVL [1] datasets. The MSRC [22] dataset is a synthetic benchmark. Although more refined and lacking in noise, synthetic methods sometimes produces kinematically implausible and unnatural hand poses which do not generalise well with real world images. More recently, the Handnet [28] dataset uses less intrusive external magnetic sensors to obtain accurate fingertip locations. These magnetic sensors are small and compact and do not affect depth information to a large extent. The BigHand [10] dataset follows a similar approach but provides 3D joint annotation for the full hand pose and currently proves to be the most extensive dataset. In terms of hand pose systems that attempt to obtain 3D joint annotations using RGB images, there is relatively fewer benchmark datasets that provide RGB-ground truth pairs. Available RGB image datasets include the Stereo Hand Pose Tracking

Benchmark [12], Dexter [13] and EgoDexter [19] which have manually annotated 3D joint positions. There are also synthetic datasets such as SynthHands [19] and Z&B [14]. Recently, the "GANerated" [17] dataset uses GANs to generate a realistic hand dataset from synthetic images.

**Generative Adversarial Networks** Generative Adversarial Networks (GAN) [18] are deep learning systems that can train a model that is capable of generating complex data. Two neural network systems are used: a discriminator and a generator. The generator is trained to generate fake data and the discriminator is trained to classify input as either fake or real. This allows GANs to learn a loss function in addition to learning a function mapping the given input and output. GANs have been shown to be successful in a variety of applications such as synthetic data generation [29], image super-resolution [30] and human pose estimation [31]. Ever since its conception, various modifications and improvements have been made to the GAN architecture. *Radford et al.* [32] proposes Deep Convolutional GAN (DCGAN) as a stable architecture for training GANs. *Arjovsky et al.* [33] introduces the current state-of-the-art Wasserstein GAN (WGAN) which further improves training stability of learning. *Mirza et al.* [34] condition GANs on labels and open the doorway to generate specified inputs which resulted in Conditional GANs (CGANS). This has been extended to images [35] and text [36].

**GAN Image-to-Image Translation** A known successful application of GANs is image-to-image translation. *Isola et al.* [35] successfully uses CGans training on paired images. This method was shown to be generalisable to several tasks such as semantic labels to photo, map to aerial photo, sketch to photo, black-and-white to colour photo and vice versa. *Zhu et al.* [37] (CycleGAN) and *Kim et al.* [38] (DiscoGAN) build on this but consider an implementation with unpaired images. CycleGANs [37] use a single loss consistency whereas DiscoGANs [38] use two reconstruction losses. *Mueller et al.* [17] implements a modified CycleGAN (GeoConGAN) to generate real hand images from synthetic hand images which uses a geometric consistency loss to ensure that images produced maintain hand pose during image translation.

### 3 Depth-RGB Dataset Generation

#### 3.1 Image acquisition

An important resource for this project would be a dataset that we could use to train and validate our GAN system. Existing datasets such as the NYU [3] dataset do not include egocentric hand poses which is prevalent in the BigHand [10] dataset. Besides that, the EgoDexter [19] dataset contain hand images occluded by objects. Thus, it was preferable to collect our own dataset.

In order to do this we developed our own system that allowed us to easily construct the dataset. This was implemented with the use of an Intel RealSense F200 camera which is capable of capturing both RGB and depth images. An example of a depth-RGB image pair is shown in Fig. 1. Observe that parts of the arm that should not be included in the final image has also been captured.

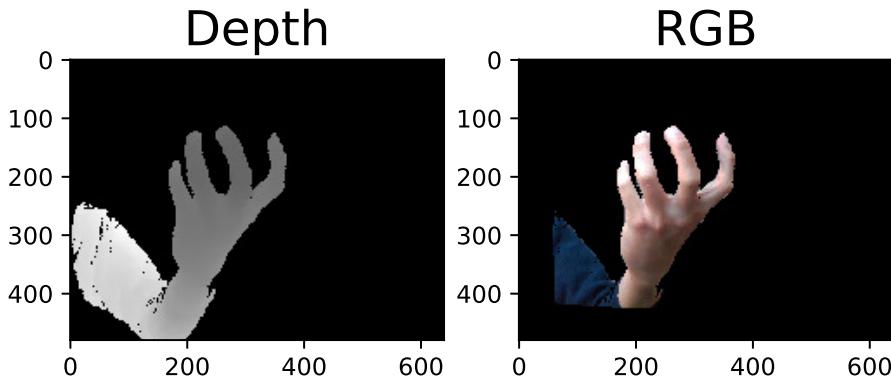


Figure 1: Example of depth and RGB image obtained from the Intel RealSense F200.

#### 3.2 Preprocessing

As our acquired images are not suitable, we require the images to be preprocessed before we can use them effectively for our GAN. As only images of the hand is required, we remove any outliers from the final image. In order to do this, we employ a hand classifier which allows us to create a

mask of the hand using the depth image as input. The hand classifier we use is developed based on a U-net [27] architecture and detects hand images from depth input. From analysis of the system, the segmented hand masks created were not satisfactory (see Appendix A). To rectify this issue, instead of using the mask to extract the segment the hand from the image, we use the mask as a reference point to crop the image. We then remove the outliers by thresholding, that is, we remove parts of the image that exceed a certain depth or colour range.

For the depth image, we carry out further preprocessing by normalising the image using the following equation:

$$\hat{Depth} = \frac{(Depth - \overline{Depth})}{hand\_span} \quad (1)$$

where  $\overline{Depth}$  is the mean depth value from the image and  $hand\_span$  is set to be the upper threshold of the span of the average human hand in millimetres which is about 200mm [39]. Normalisation is carried out to induce depth invariance in the depth images.

The preprocessing pipeline is shown in Fig. 2.

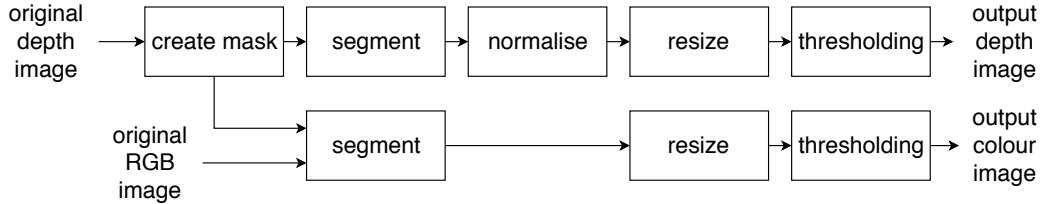


Figure 2: Preprocessing pipeline for Dataset Generation

Examples of preprocessing images are shown in Fig. 3.

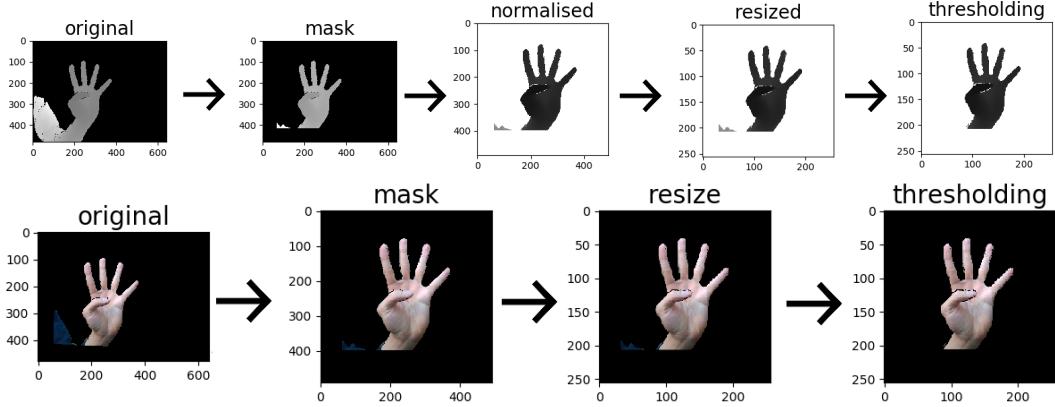


Figure 3: Example of preprocessing images. Top: Depth image. Bottom: RGB image.

### 3.3 Depth-RGB Dataset

#### 3.3.1 Analysis Dataset

Initially, we acquire a smaller, less extensive dataset to analyse the effectiveness of the GAN image translation systems on converting the depth hand images to RGB hand images. To reduce training time and complexity, we generate a dataset relatively smaller than the intended dataset to be used in converting the BigHand 2.2M [10] dataset and we do not use as many viewpoints (no egocentric poses), articulations and poses. We also restrict hand shape and skin colour, by acquiring data from 1 subject (male). We ensure our testing set contains both poses contained in the training set and new unseen poses. Examples of RGB-depth pairs in the initial dataset used for analysis are shown in Fig. 4.

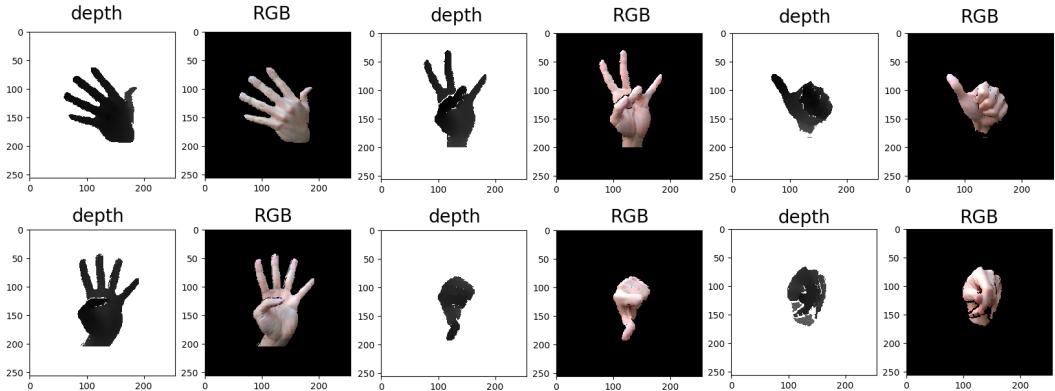


Figure 4: Top: Example testing set RGB-depth hand pairs. Bottom: Example training set RGB-depth hand pairs.

### 3.3.2 Comprehensive Dataset

Our final trained GAN should be able to map a variety of viewpoint, articulation and poses which are present in the BigHand dataset. This means that our final dataset for training and testing the GAN should have similar variety.

The BigHand dataset consists of three main sections: (1) Schemed poses: covers all the articulations that a human hand can adopt. (2) Random poses: the subject freely explores the pose space. (3) Egocentric poses: The hand is viewed from a first person perspective while covering the 32 extremal poses with each finger either maximally bent or straight similar to that studied by *Wu et al.* [40] with random movements. We attempt to match this variety in our GAN dataset. To reduce training times, we do not attempt to match the size of the BigHand dataset. We also see in later sections that our GAN system does not require as many samples in the dataset to learn a mapping. In fact, in certain cases we found that training on larger datasets results in lesser performance and lower quality generated images.

We do ensure our dataset covers all  $2^5 = 32$  extremal hand poses for both the palm side and back side of the hand with a constant hand pose angle. Furthermore, we cover the same extremal poses for both sides from an egocentric perspective.

### 3.4 Implementation

To obtain the depth and RGB hand image pairs we use an Intel RealSense F200 camera which is composed of an RGB camera, an infrared detector and an infrared sensor. It's functionality is programmed using the Intel RealSense SDK [41] in C++. The developed program allows us to capture multiple images in quick succession.

As the RGB camera and infrared detector are located in different positions on the camera, the depth and RGB images acquired are misaligned. Alignment is necessary for the images to be used effectively as training data for our GAN system. To ensure alignment, we use the function `CreateColorImageMappedToDepth` which provides us with registered colour images that only provide colour information for pixels with valid depth data. This introduces some corruption to the images. However, we find that overall the quality of hand images is suitable in practice.

The program for preprocessing the images is developed in Python. When normalising the depth image, we set  $hand\_span = 300$ . We also resize the images to 256 x 256 pixels.

## 4 Generative Adversarial Network

### 4.1 Baseline GAN (pix2pix)

We investigate current state-of-the-art in image-to-image translation using generative adversarial networks(GANs) as a baseline. We use the image-to-image translation GAN architecture as proposed by *Isola et al.* [35] which is implemented in the `pix2pix` [42] software provided at <https://github.com/phillipi/pix2pix> which have been shown to successfully carry out domain transfer for a wide range of general applications.

Using this architecture as a baseline, we attempt to improve its performance by experimenting with two approaches: (1) Applying Wasserstein GAN [33] loss which has shown to be the state-of-the-art in terms of stable GAN training. (2) Applying Laplacian Pyramid decomposition and reconstruction as inspired by *LAPGAN* proposed by *Denton et al.* [43].

#### 4.1.1 Objective function

The objective function is given as:

$$L_{p2p}(G, D) = L_{cGAN}(G, D) + \lambda L_1(G). \quad (2)$$

$L_{cGAN}(G, D)$  represents the objective function for a conditional GAN [34] given as:

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x|y)] + \mathbb{E}_{x,z}[\log(1 - D(x|G(x, z)))]. \quad (3)$$

$L_1(G)$  represents the L1 loss function given as:

$$L_1(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1], \quad (4)$$

where  $\lambda$  is a constant that we define.

In GAN systems, the generator attempts to minimize the objective function whereas the discriminator attempts to maximize the objective function. This is expressed as:

$$\min_G \max_D L_{p2p}(G, D) = L_{cGAN}(G, D) + \lambda L_1(G) \quad (5)$$

The L1 term is introduced as it has been found that introducing a conventional norm loss function would help the network learn particular structures in the image [44]. In this case, the L1 term is known to produce a blurred result [45]. This allows it to capture low frequency information from the input and enforce it onto the output.

#### 4.1.2 Network architecture

As the L1 term in the objective function enforces low frequency structure in the output, the discriminator is designed to learn the high frequency structure in the output. *Isola et al.* [35]

introduce a *PatchGAN* that penalises structure at the scale of patches. That is, the discriminator checks each  $N \times N$  patch, classifies it as real or fake and averages all responses.

The architecture of the discriminator network is shown in Fig. 5. Patch size corresponds to the receptive field [46] of the network. The input image is processed by passing through multiple convolutional layers. After each convolutional layer (except the first), normalization is carried out via a batch norm [47] layer. Batch normalisation is introduced to optimise network training by increasing convergence rates, reducing parameter sensitivity and providing some regularisation. Additionally, the output of each layer is processed by a Leaky Rectified Linear Unit (Leaky ReLU) [48] layer. ReLU layers are included to introduce nonlinearity and result in sparse outputs. This has been shown to improve training convergence and performance when carrying out discrimination tasks [25]. Leaky ReLUs act similarly but in contrast to ReLUs, leaky ReLUs prevent the gradient from reaching 0. The ReLU function is given as:

$$ReLU(x) = \max(x, 0) \quad (6)$$

and the Leaky ReLU function is given as:

$$Leaky\ ReLU(x) = \max(x, \alpha x) \quad (7)$$

where  $\alpha$  is set as a small positive constant with  $|\alpha| < 1$ .

The result is then mapped to a 1 dimensional output followed by a Sigmoid function.

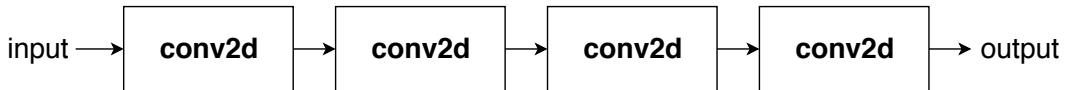


Figure 5: Discriminator network architecture for *pix2pix* GAN.

The generator is modeled as an encoder-decoder network [49] with skip connections similar to a "U-Net" [27]. The skip connections ensure additional information to carry to the decoder section of the network. This prevents loss of information through the encoder-decoder bottleneck. This

is especially important for image translation problems as the input and output share many low level details. The skip connections connect the  $i$ -th layer with the  $n - 1$ -th layer.

The architecture for the generator is displayed in Fig. 6. The encoder section of the architecture (displayed in blue) consists of convolutional layers which downsample by a factor of two. The decoder section, on the other hand, consists of deconvolutional layers which upsample by a factor of two. Similar to the discriminator, after each layer besides the first, a batch norm layer is introduced. After each layer the output is processed by a Leaky ReLU function. Finally, the output is processed by a  $\tanh$  function.

To ensure non-deterministic results we are required to add a noise term,  $z$  to the input alongside  $x$ . *Isola et al.* [35] found that adding Gaussian noise to the input was not effective. Instead, in the architecture, noise is administered through a dropout layer which is included in the decoder segment after every deconvolutional layer. Note that even with this only minor stochasticity was observed.

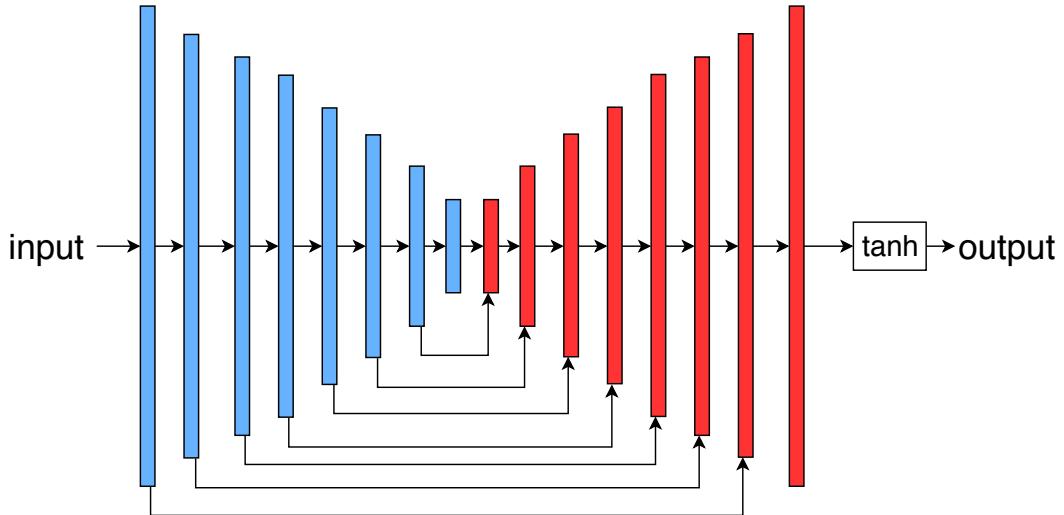


Figure 6: Generator network architecture for *pix2pix* GAN.

## 4.2 Wasserstein Loss

In this section we investigate running the Baseline GAN architecture using an objective function inspired by the Wasserstein GAN (WGAN) [33]. While it has shown success and promise, GANs

are still a relatively new design of deep learning network. Currently, there are no reliable methods for ensuring the stability of training GANs. There are also no reliable conditions or methods that have been shown to guarantee convergence in training GANs. WGANs have been designed to approach this issue and have been shown to improve the stability of the original GAN algorithm.

The initial GAN [18] algorithm can be interpreted as minimising Jensen-Shannon divergence:

$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q\|\frac{p+q}{2}) \quad (8)$$

where  $D_{KL}$  is the Kullback–Leibler divergence:

$$D_{KL}(p\|q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx \quad (9)$$

These divergences are essentially measuring the similarities between two probability distributions. The idea of WGANs is to attempt to improve this metric. WGANs update the loss function to include the Wasserstein distance or Earth Mover's distance:

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \quad (10)$$

The improved version of WGANs [50] have been shown to improve convergence speed and is more robust to hyperparameter tuning than the original. Our approach is to implement the Wasserstein loss algorithm as proposed by the improved WGAN with the baseline GAN architecture and attempt to stabilize the training in hopes of improving generated image quality.

### 4.3 Laplacian Pyramid

We also investigate a Laplacian Pyramid approach to generating output images inspired by LAPGANs [43]. This method involves decomposing the original image into smaller, simpler images. The process can be reversed to reconstruct the original image.

With this, instead of learning a mapping from depth to the target RGB image, we train the network to learn a mapping from depth to Laplacian Pyramid coefficients,  $[h_1, h_2, \dots, h_n]$  and the final downsampled image  $I_n - 1$ . This is motivated by the idea that the baseline GAN should be able to, more easily, learn a mapping as the output is a less complex image. The final output results can then be used to reconstruct the original target image. Laplacian Pyramid decomposition and reconstruction with a hand example is visualised in Fig. 7.

Thus, we would have to train a GAN for each Laplacian Pyramid coefficient and the final downsampled image in order to obtain a generated reconstructed image.

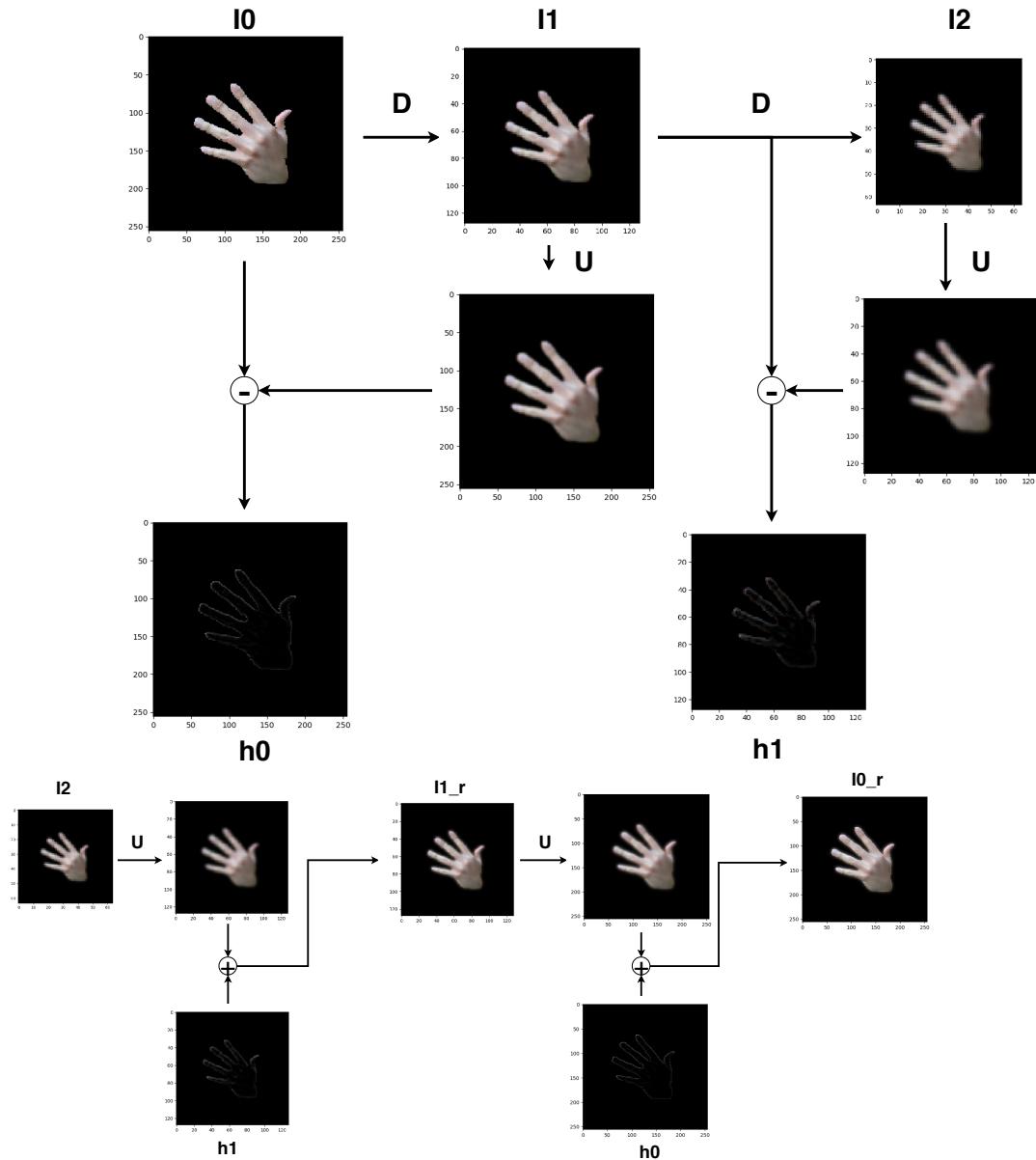


Figure 7: Top: Laplacian Pyramid representation. Bottom: Laplacian Pyramid reconstruction. D represents blurring followed by downsampling by a factor of 2. U represents upsampling by a factor of 2 followed by blurring.

#### 4.4 Implementation

The software used is a TensorFlow [51] implementation [52] of pix2pix based on DCGAN-tensorflow [53] which is modified to suit our specifications.

Similar to the `pix2pix` method, we use minibatch stochastic gradient descent with ADAM [54]. During the generation of images, the generator network is tuned under similar conditions. The specifications of our network architecture and training system is given in Appendix B.

In this implementation, the generator network is updated twice per update of the discriminator network to avoid the fast convergence of the discriminator network. This differs from the original `pix2pix` method which divides the discriminator’s objective function by 2 to slow down the rate at which the discriminator learns relative to the generator.

Before training, we also randomise the images by resizing the initial input training images to 286 x 286 and then randomly cropping back to 256 x 256. This adds extra noise to each image which helps our network generalise better.

## 5 Hand Pose Estimation

The ultimate purpose of our generated images is to create a dataset that can improve training for hand pose estimation tasks. Thus, the most suitable method to assess our generated dataset would be to train a hand pose estimation system using the dataset and compare with existing benchmark datasets.

### 5.1 Network Architecture

To carry out hand pose estimation, we follow a similar architecture to that proposed by *Tompson et al.* [3]. The architecture consists of two network stages: 1) Convolutional. 2) Dense. The system, when trained, estimates 21 3D hand joint positions from input hand images.

The initial convolutional stage consists of 3 parallel convolutional networks. The purpose of this stage is to detect features from the input image. Each layer processes the image at a different resolution. Specifically, we process the original image and the image downsampled by a factor of 2 and the image downsampled twice by a factor of 2. This assists the system in distinguishing

between global features i.e. contours and local features i.e fingernails. The convolutional layers include a max-pooling module proposed by *Nagi et al.* [55] which carries out down-sampling. This has been shown to effectively reduce the dimensions and complexity of the data at the cost of spatial precision. A rectified linear unit (ReLU) activation layer is also included after each layer.

The output of the first stage is concatenated, batch normalised and flattened before being sent to the second stage. The dense or fully connected layers takes the processed hand features and learns a mapping to the 21 3D hand joint positions. Unlike the original architecture proposed by *Tompson et al.* [3], we include a dropout layer after each dense layer. The dropout layers help to improve generalisation of the system and prevents overfitting. An overview of the architecture can be seen in Fig. 8.

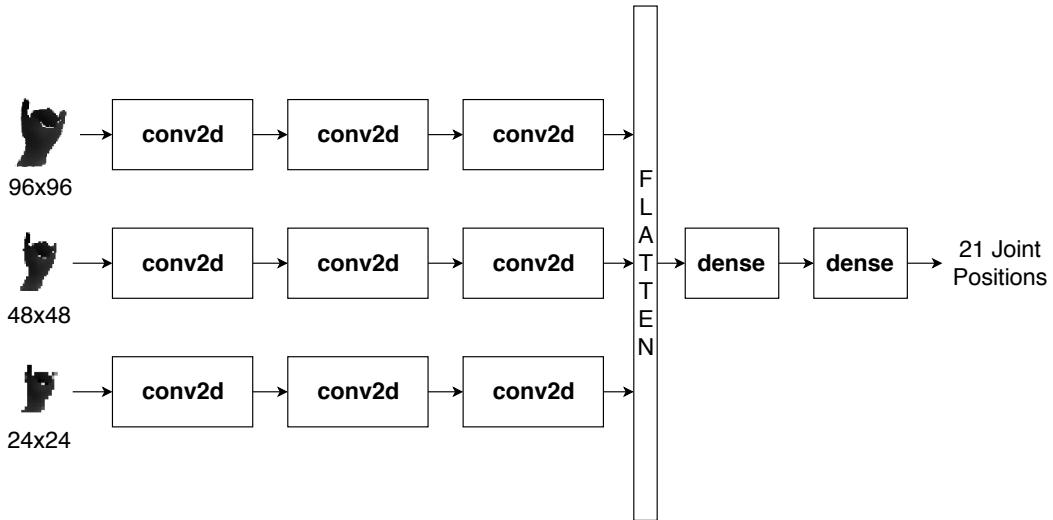


Figure 8: Network architecture for hand pose estimation.

## 5.2 Implementation

The hand pose estimator network is implemented in Keras [56] and Tensorflow. To carry out training, we use minibatch stochastic gradient descent with ADAM. Mean squared error is used as the error measure during training. Detailed specifications of our network architecture and training system is given in Appendix D.

## 6 GAN Evaluation

Evaluation of synthesised image quality is an open and difficult problem [57]. Per-pixel error is a traditional metric we can use to measure the similarity of synthesised image with ground-truth target. *Isola et al.* [35] do not employ this as per-pixel error is only meaningful when treating the output space as unstructured when each pixel is considered independent of the other. This contradicts with conditional GANs which learn structured losses which penalize joint configuration of the output.

As we have restricted our problem to primarily dealing with image mapping, we would like our output image to match the ground truth or target image as much as possible. Thus, we apply per-pixel mean squared error as a metric for quantizing the efficacy of our system. The mean squared error is given as:

$$\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{m \times n}$$

$$\mathbf{C} = \mathbf{A} - \mathbf{B}$$

with entries  $c_{i,j}$  where  $0 \leq i \leq m$ ,  $0 \leq j \leq n$

$$MSE = \frac{1}{(mn)} \sum_{i,j} (c_{i,j})^2 \tag{11}$$

The final mean squared error is the sum of the mean squared error for each colour channel of the image i.e red, green and blue.

Using this metric alone is not sufficient. It is also necessary to pair this together with human judgment as the error measure would only provide us with a rough understanding of the quality of generated image. It does not do well in quantifying the feature detail generated in the output. Ultimately, we would prefer generated images that would be good enough for humans to classify as true which are not strictly exact mappings of the original RGB image.

For our experiments, we run our network training on a smaller analysis dataset as given in Section. 3.3.1. We train our system using 400 training pairs with batch size 1 for 200 epochs and test on 400 testing pairs. This amounts to a total of 80000 steps.

## 6.1 Training Analysis

### 6.1.1 Baseline GAN

In this section we evaluate using the optimal settings as suggested by *Isola et al.* [35] as a guideline (summarised in Appendix. B). From analysing the loss functions given in Fig. 16, we see that discriminator loss values oscillate around (0.5, 2) whereas generator loss values oscillate around (1, 4). A known issue with training losses obtained from GANs are that they can be unintuitive as the discriminator and generator are constantly trying to maximise and minimise the other. The main result that we can extract from the loss function graphs is that training is relatively stable.

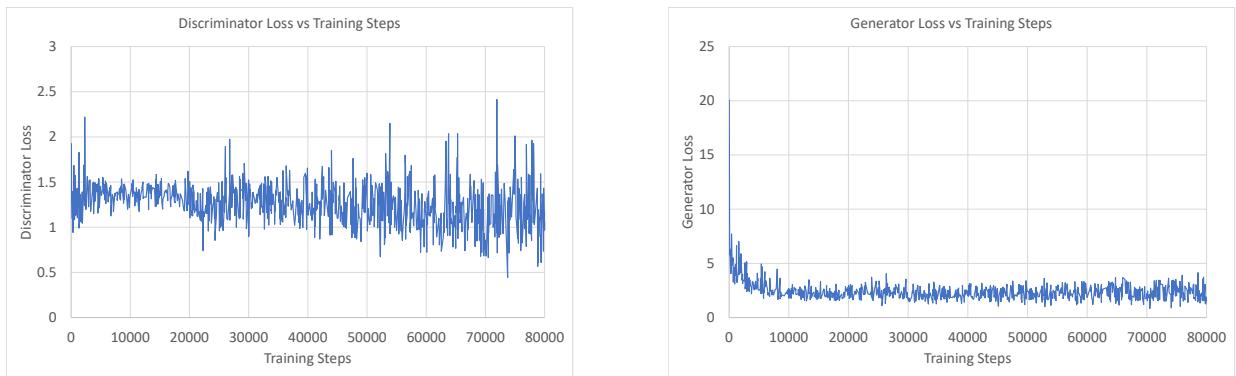


Figure 9: Loss graphs from 70x70 Baseline GAN with learning rate of 0.0002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

To ensure our system is training well, we also sample images throughout the training. On earlier samples we observe noticeable improvement in the quality of our samples. The noticeable improvement gradually decreases as training carries on. A trained model is saved at epoch = 100 and images are generated to compare with generated images from the trained model at epoch = 200. We observe improved results from training the system to epoch = 200. Results and samples can be seen in Appendix. B.3.

Finally, we also analyse the probability distributions of the discriminator given ground truth

outputs and when given generated outputs. Ideally, we would like the distribution to peak at 0.5 which means that the discriminator cannot distinguish between generated and ground truth outputs and randomly classifies it with a probability of 50%. The probability distributions are given in Fig. 10.

Notice that at 200 epochs, our distributions have not converged to 0.5. The distribution given ground truth  $D(x|y)$  peaks at around 0.55 whereas the distribution given generated output  $D(x|G(x, z))$  peaks at around 0.35. This implies that the discriminator finds it more difficult to label ground truth images as real or fake but it can more easily classify generated output images.

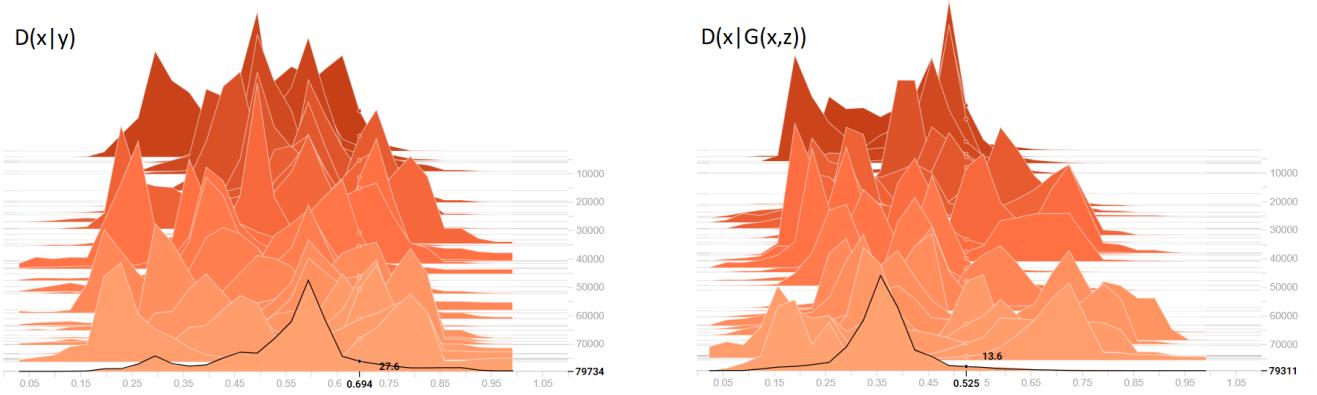


Figure 10: Moving probability distributions from 70x70 Baseline GAN with learning rate of 0.0002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x, z))$ .

### 6.1.2 Wasserstein Loss

We run the training again but change the loss objective to the Wasserstein Loss and follow the training algorithm as proposed by *Guljarani et al.* [50] which is explained in Section. 4.2.

Observing the training loss and distribution given in Fig. 11, we can see that the discriminator loss starts off unstable but eventually converges to oscillate around fixed values. The generator loss shows unstable oscillations until about epoch 50000 where it begins to show a decreasing trend. Note that the stabilisation of the generator loss graph occurs around the same epoch as when the discriminator loss stabilises. From analysing the probability distributions given in Fig. 12, we see that the discriminator tends to classify the image as real regardless of it being the

ground-truth or the generated image.

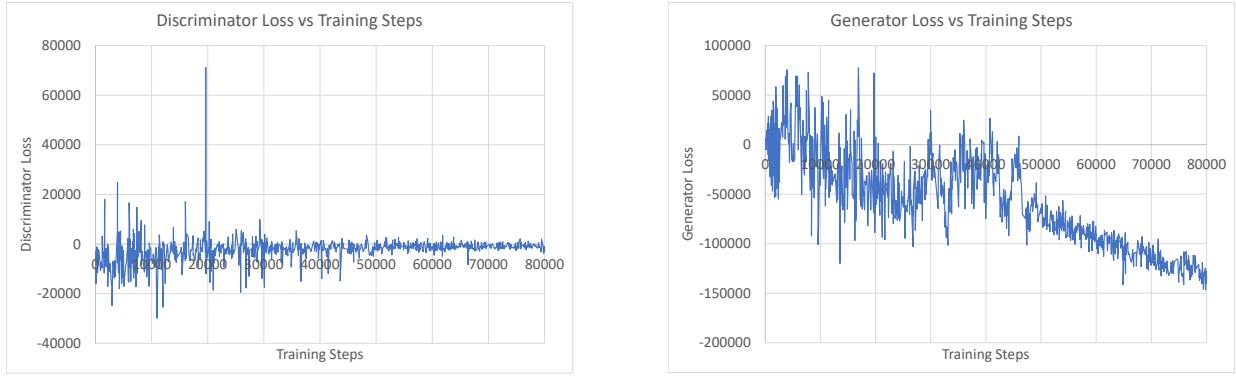


Figure 11: Loss graphs from 70x70 GAN with Wasserstein Loss and learning rate of 0.0002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

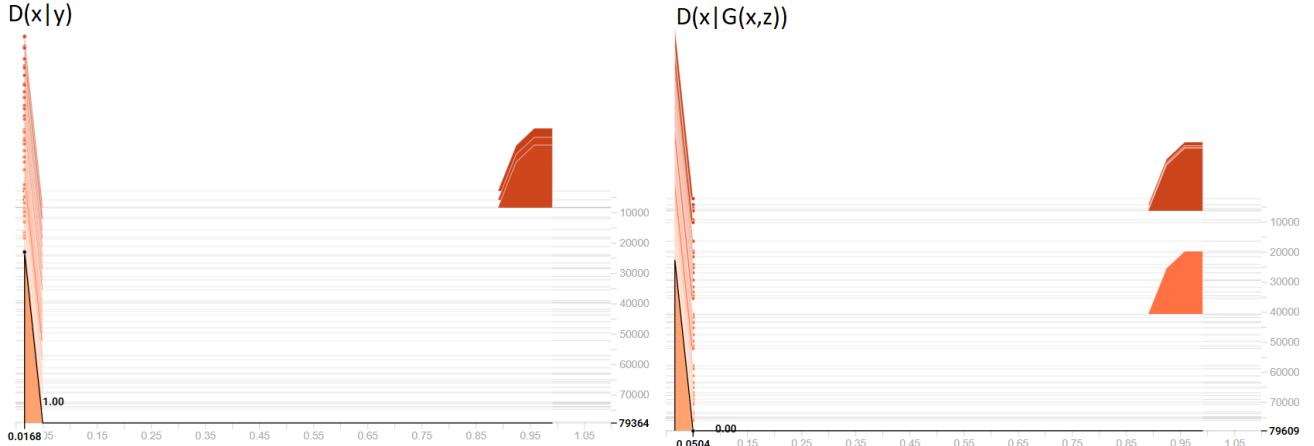


Figure 12: Moving probability distributions from 70x70 GAN with Wasserstein Loss and learning rate of 0.0002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

### 6.1.3 Laplacian Pyramid

For the Laplacian Pyramid GAN, we train 3 different networks to generate the Laplacian coefficient images ( $h1, h0$ ) and the target image downsampled twice by a factor of two ( $i2$ ) as explained in Section. 4.3.

We observe stable and similar training trends for all 3 networks. From the training loss of the *i2* network given in Fig. 13, the generator and discriminator loss distribution both converge and become stable with minimal oscillation at an early stage during the training. We believe this to be attributed to the fact that the target images are simpler with less resolution so the network has information to learn.

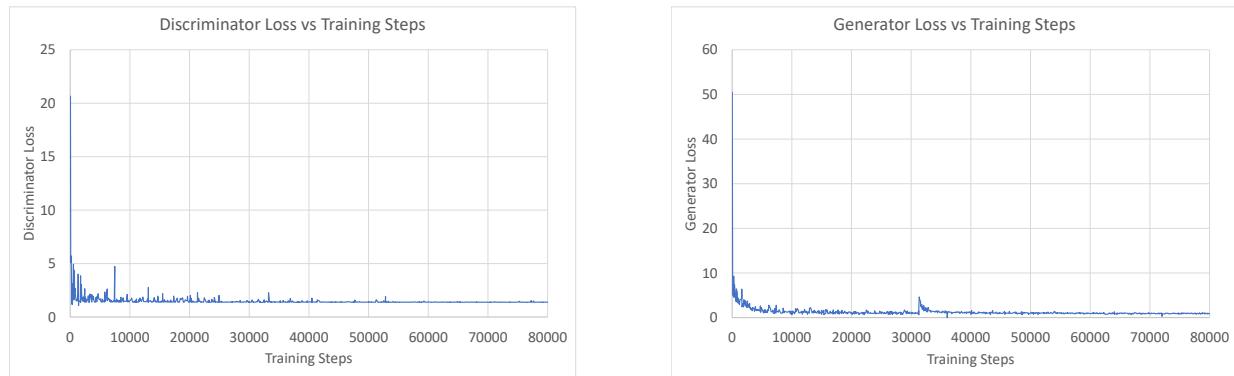


Figure 13: Loss graphs from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid downsampled target image *i2*. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

The probability distribution for the *i2* network as shown in Fig. 14 shows good convergence with the final distribution peaking at around 0.5. The discriminator cannot distinguish between ground-truth and generated images and so the generator cannot improve any further.

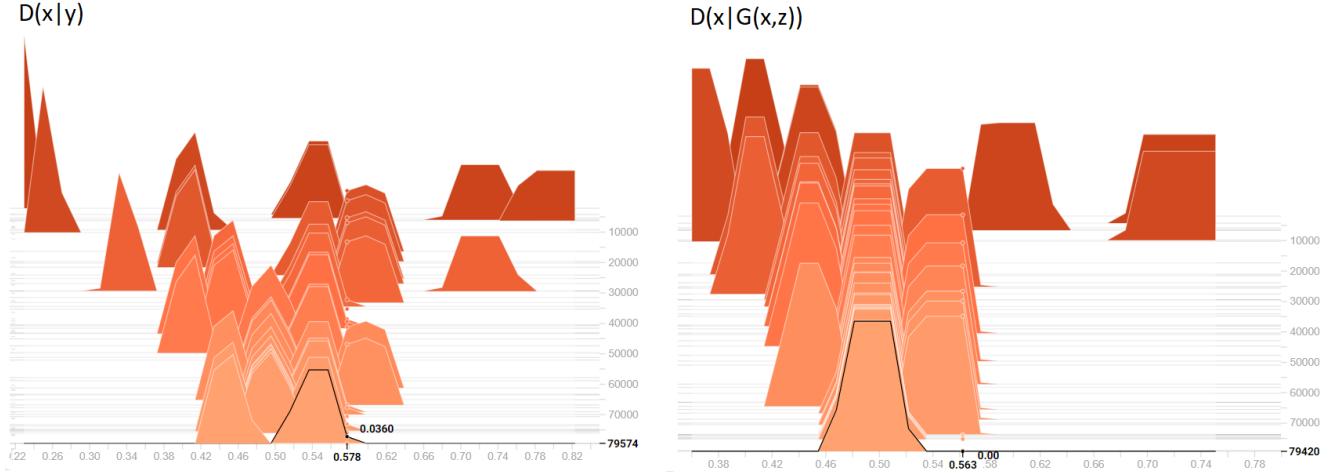


Figure 14: Moving probability distributions from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid downsampled target image i2. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x, z))$ .

Training results on the Laplacian coefficients,  $h1$  and  $h2$  show similar trends and can be viewed in Appendix. C.2.

## 6.2 Evaluation of Generated Images

### 6.2.1 Baseline Generated Images

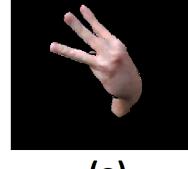
Set Quality \	Train				Test			
	mse=187.80	mse=206.76	mse=287.66	mse=303.59				
<b>Bad</b>					(a)	(b)	(i)	(j)
					mse=212.16	mse=238.01	BigHand	BigHand
<b>Good</b>					(e)	(f)	(m)	(n)
					mse=133.68	mse=147.22	mse=148.53	BigHand

Figure 15: Examples of generated RGB output from baseline GAN patch size  $70 \times 70$  with learning rate of 0.0002 for 200 epochs

Examples of generated images are displayed in Fig. 15. From Examples (a) and (h), we see that the network can distinguish between individual fingers well. When the fingers are placed together there is ambiguity in the generation which can be seen in Examples (b) and (i). This is also shown in Example (a) where the thumb overlaps with the palm. From Example (g), we see that generated images with overlapping features can be distinguished visually with reasonable quality.

From the testing set, in Example (m), we see that the generator performs well on unseen images with poses present in the training set. Based on Examples (o) and (n), we also observe that the generator deals well with images of similar poses with a slight angular rotation. Example (j) displays an example of a generated unseen hand pose. Qualitatively, the hand pose can be distinguished easily but the generator is not able to generate the hand features very well. Generating images from the BigHand [10] dataset (Example (k),(l),(p)), we see that the network generally does not perform well on poses not present in the training dataset (as we've observed in Example (j)). More prominent artifacts and distortion is present on the generated image from different viewpoints such as in Example (l). On poses present in the training dataset, we observe qualitatively better results such as in Example (p).

Train			Test				
depth	output	target	mse	depth	output	target	mse
			147.22				235.41
			197.11				184.35
			195.24				285.23
			238.01				339.96
			135.11				110.47
			92.142				123.63

Figure 16: Examples of generated RGB output from baseline GAN patch size  $70 \times 70$  with learning rate of 0.0002 for 200 epochs. Left: Examples generated from training data. Right: Examples generated from testing data. "mse" represents mean-squared error.

Overall, we observe that the default values do a good job in mapping the overall colour structure onto the depth image. The hand is easily recognisable and key joint positions are immediately noticeable. However, lower frequency information and finer hand detail such as palm creases and fingernails are not always generated with excellent resolution. Examples with input and target information are shown in Fig. 16.

### 6.2.2 Learning Rate

We train multiple baseline GAN systems with similar training settings (patch size 70 x 70) while varying the learning rate. No anomalies were observed during training. Training details and analysis is provided in Appendix. B.4.

We observed that images produced from a learning rate of 0.00002 generally do not distinguish individual fingers well especially in images with the fingers in close proximity of each other. Details in palm and hand creases are also not as extensive.

In comparison, images produced using a learning rate of 0.0002 have finer detail and resolution. Intricate hand detail such as creases, knuckles and fingers are more immediately obvious.

Finally, images produced with a learning rate of 0.002 has similar visual quality compared to the 0.0002 results in terms of hand detail. Training and testing error have also been further improved. However, testing error has improved by a lesser extent relative to the training error improvement. Examples of generated images given different learning rates can be seen in Fig. 17. From the mean squared errors displayed in Table. 1, we see that the training and testing error improves as learning rate decreases.

Learning rate	MSE (Train)	MSE (Test)
0.00002	206.9301	260.4526
0.0002	162.7022	209.1948
0.002	136.0324	200.5820

Table 1: Mean squared error of image datasets generated from training and testing images while varying learning rate.

Train				Test					
depth	0.00002	0.0002	0.002	target	depth	0.00002	0.0002	0.002	target
	167.64	147.22	117.00			325.32	235.41	236.98	
	170.12	197.11	132.26			204.06	184.35	609.45	
	278.05	195.24	153.95			386.45	285.23	355.77	
	249.72	238.01	165.98			460.57	339.96	365.22	
	143.19	135.11	116.51			162.24	110.47	99.18	
	132.92	92.14	70.84			192.07	123.63	133.90	

Figure 17: Examples of RGB output from baseline GAN patch size  $70 \times 70$  with varying learning rates.

### 6.2.3 Discriminator Patch Size

We analyse the effects of varying discriminator patch sizes on the quality of generated output images. Similar to experiments on learning rate, we train multiple baseline GAN systems with similar training settings (learning rate = 0.0002) while varying the discriminator patch size. Training analysis is provided in Appendix. B.5.

Overall, we see similar to the results obtained by *Isola et al.* [35] when experimenting with different patch sizes. Examples of generated images are given in Fig. 18. We observed that images generated using the  $32 \times 32$  discriminator is able to colourise the image successfully but

fails to capture detail in the hand. Instead we observe artifacts where the network attempts to recreate the detail. Results improve when increasing the patch size to  $70 \times 70$  and  $286 \times 286$ . We observe that generated images from the  $286 \times 286$  patch do not appear to have any significant visual improvement. Our observations correspond with the mean squared error values tabulated in Table. 2. The  $70 \times 70$  patch architecture displays the best results with  $286 \times 286$  error values being only slightly worse and  $32 \times 32$  error values being significantly worse.

Patch Size	MSE (Train)	MSE (Test)
32x32	268.5717	303.4199
70x70	162.7022	209.1948
286x286	178.7796	226.2236

Table 2: Mean squared error of image datasets generated from training and testing images while varying patch sizes.

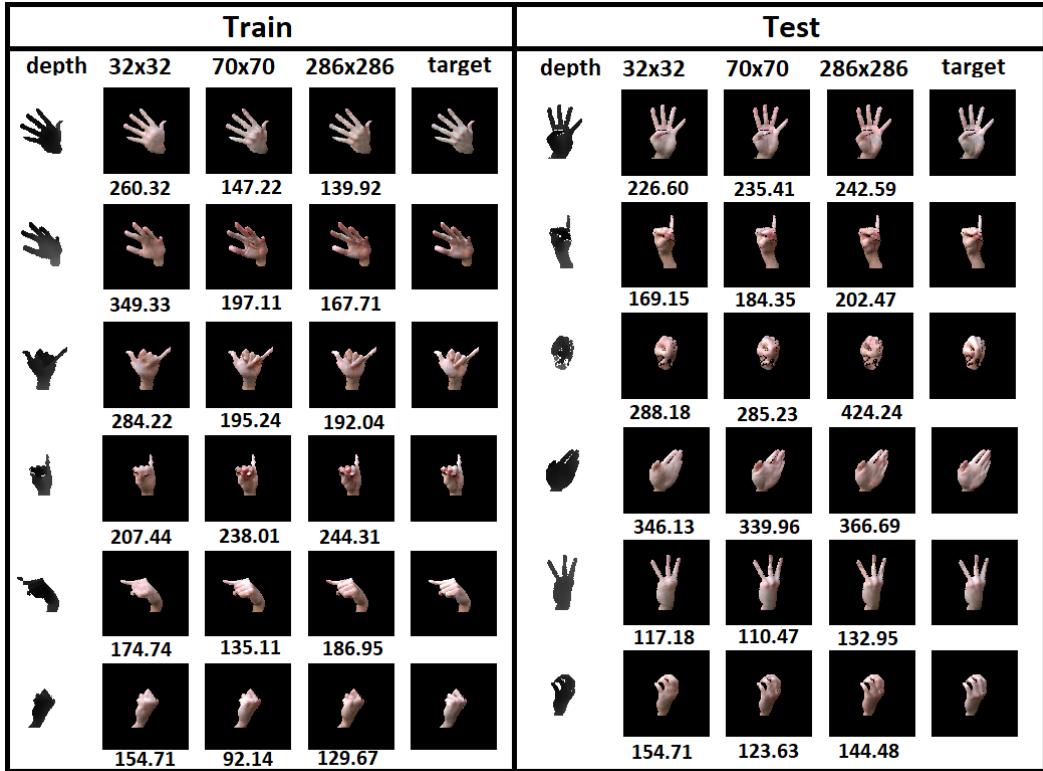


Figure 18: Examples of RGB output from baseline GAN with learning rate 0.0002 with varying patch sizes.

#### 6.2.4 GAN System

Visually, the Wasserstein Loss GAN results show good colour mapping and detail representation. The overall mean square error is similar to the baseline GAN with patch size 70x70 and learning rate equal to 0.0002. The most notable difference between the generated images from the Wasserstein Loss GAN and the baseline GAN is the prominent square patch artifact discoloration present in a majority of the generated images.

The Laplacian Pyramid GAN system shows the worst quantitative result by far. The generated reconstructed images have managed to map the colour correctly onto the images but there is clear lack of detail in hand creases and finger definition. We believe this to be due to the blurring of images when upsampling and downsampling images. This combined with the further loss of information in generating images has resulted in the smooth features in the reconstructed images. We see the best visual quality with the baseline system with patch size 70x70 and learning rate equal to 0.002. Generated image examples can be observed in Fig. 19.

A comparison of the overall mean square error values are displayed in Table. 3. The Laplacian Pyramid errors are significantly worse while the Wasserstein Loss implementation shows comparable results to the baseline system with patch size 70 x 70 and a learning rate of 0.0002. Corresponding to our visual analysis, the best error results are given by the baseline system with patch size 70 x 70 and a learning rate of 0.002.

GAN system	MSE (Train)	MSE (Test)
Baseline (best)	136.0324	200.5820
Wasserstein Loss	164.9672	214.1693
Laplacian Pyramid	1160.3557	1175.2614

Table 3: Mean squared error of image datasets generated from training and testing images from GAN network trained on default settings.

Train					Test				
depth	lap_pyr	baseline	wgan	target	depth	lap_pyr	baseline	wgan	target
									
	1780.49	117.00	156.16			1601.54	236.98	231.70	
						1376.52	609.45	193.50	
	1310.80	132.26	207.56						
									
	1135.55	153.95	191.20						
						883.72	355.77	291.24	
	586.30	165.98	218.57						
						891.92	365.22	296.84	
	836.06	116.51	140.64						
						1218.74	99.18	117.83	
	621.07	70.84	98.12						
						1080.68	133.90	140.39	

Figure 19: Examples of RGB outputs from different GAN systems.

### 6.3 Evaluation on Comprehensive Dataset

From our analysis in Section. 6.2.4 we use the baseline GAN with patch size 70x70 and a learning rate of 0.002 to generate images trained on our comprehensive dataset as given in Section. 3.3.2.

### 6.3.1 Training Analysis

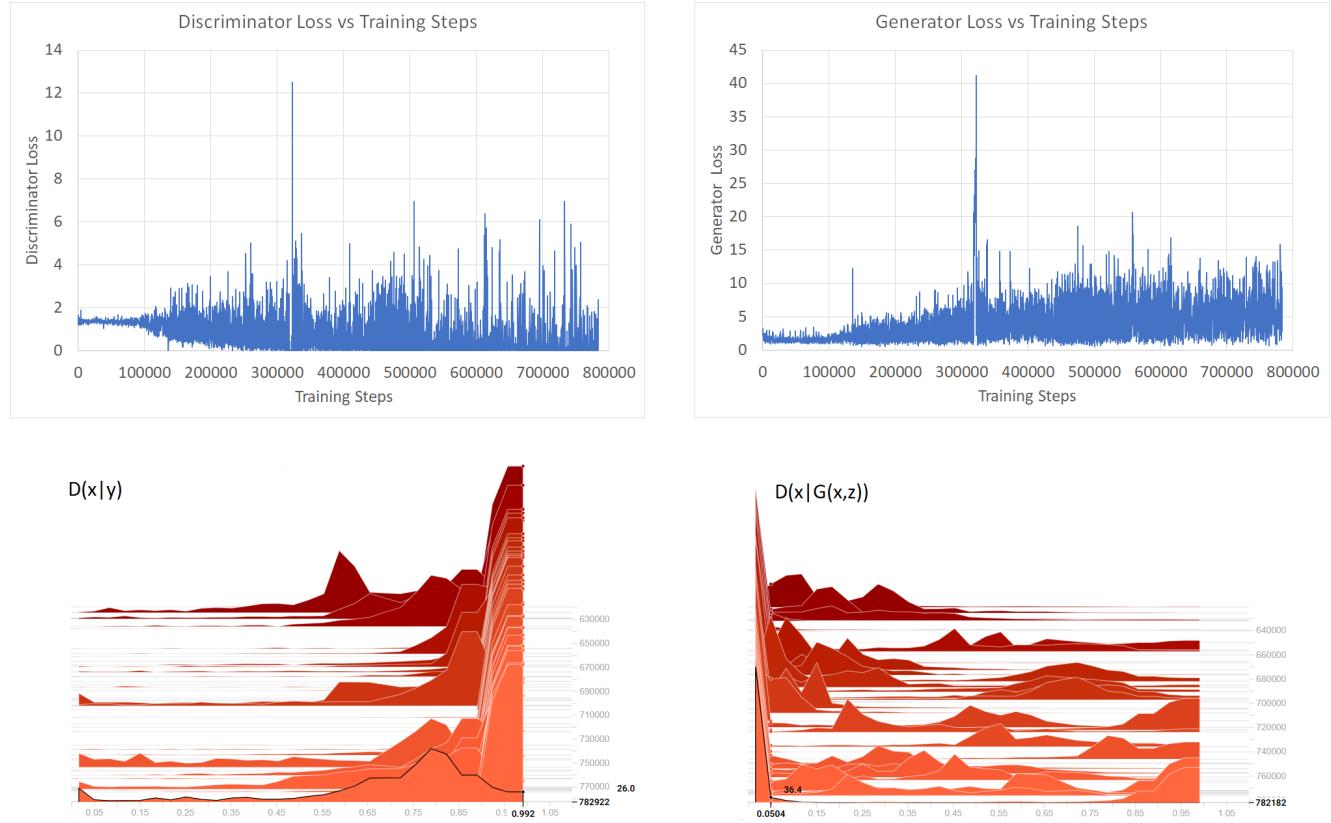


Figure 20: Training graphs from 70x70 Baseline GAN with learning rate of 0.002 trained on comprehensive dataset with 3918 training pairs. Top: Discriminator and generator loss graph. Bottom: Moving probability distribution of  $D(x|y)$  and  $D(x|G(x,z))$ .

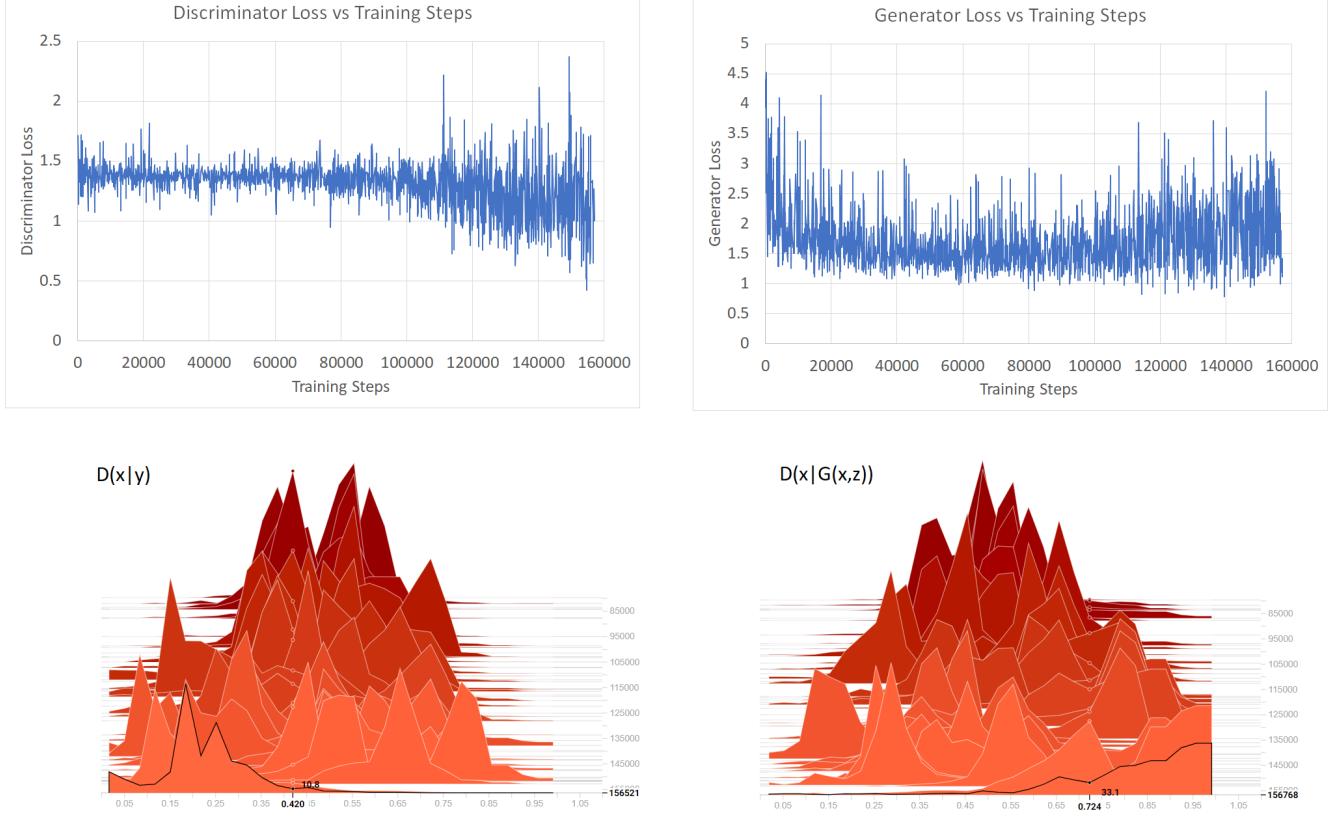


Figure 21: Training graphs from 70x70 Baseline GAN with learning rate of 0.002 trained on comprehensive dataset with 785 training pairs. Top: Discriminator and generator loss graph. Bottom: Moving probability distribution of  $D(x|y)$  and  $D(x|G(x,z))$ .

Fig. 20 and 21 shows the loss graphs and moving probability distributions from training the comprehensive dataset on different sized datasets. Loss graph and moving probability distribution for training the dataset with size 1025 is given in Appendix. B.6.1.

During training, we observe instabilities when dealing with larger sized datasets. When trained on 3918 input images, the discriminator loss as training carries out increasingly becomes unstable and the loss decreases by a large amount. At the same time, the generator loss gradually increases. This implies the network finds it too easy to distinguish between ground-truth and generated images. This can clearly be seen in the moving probability distribution. Early stopping did not yield great results as the quality of the generated images were not improved.

Training on 1025 and 785 input images, training carries out relatively stabler with less

oscillations. We do notice higher levels of oscillations and an increasing generator loss curve at later stages of the training but to a lesser extent than with the 3918 dataset.

Difficulty in training large datasets could be attributed to the design of the network architecture. Besides that, we notice that the generator struggles with mapping colour due to the non-consistent illumination of the target images. Normalisation of target image luminosity should be carried out when processing future datasets.

### 6.3.2 Evaluation of Generated Images

Analysing the generated images quantitatively, as expected, the 785 dataset has the best results among the datasets. This corresponds with the qualitative analysis of the generated images. The mean squared errors are displayed in Table. 4. Prior to testing, the testing set was divided into one consisting of known poses and one consisting of unknown poses.

Examples of generated images are given in Fig. 22. From the 3918 dataset we observe inconsistent colour and illumination. We believe this to be due to the variety of illuminations present in the larger dataset. We also see minimal hand detail and more smooth blur with patch artifacts in the images.

The 1025 dataset has less issue with colour and illumination as compared to the 3918 dataset. Hand features are also slightly more noticeable but smoothness and blur artifacts are still prominent.

The 785 dataset shows greatly improved results. Patch sized artifacts can still be seen but generated hand features are displayed more clearly. We attribute blurriness in some images due to the illumination and resolution of the target image.

Dataset Size	MSE (Train)	MSE (Test-Seen)	MSE (Test-Unseen)
785	179.5398	181.2764	361.5778
1025	251.0001	305.4645	521.02234
3918	530.8364	534.7366	700.7367

Table 4: Mean squared error of image datasets generated from training and testing images for known poses and unknown poses from GAN network trained on patch size 70x70 and a learning rate of 0.002 with the comprehensive dataset.

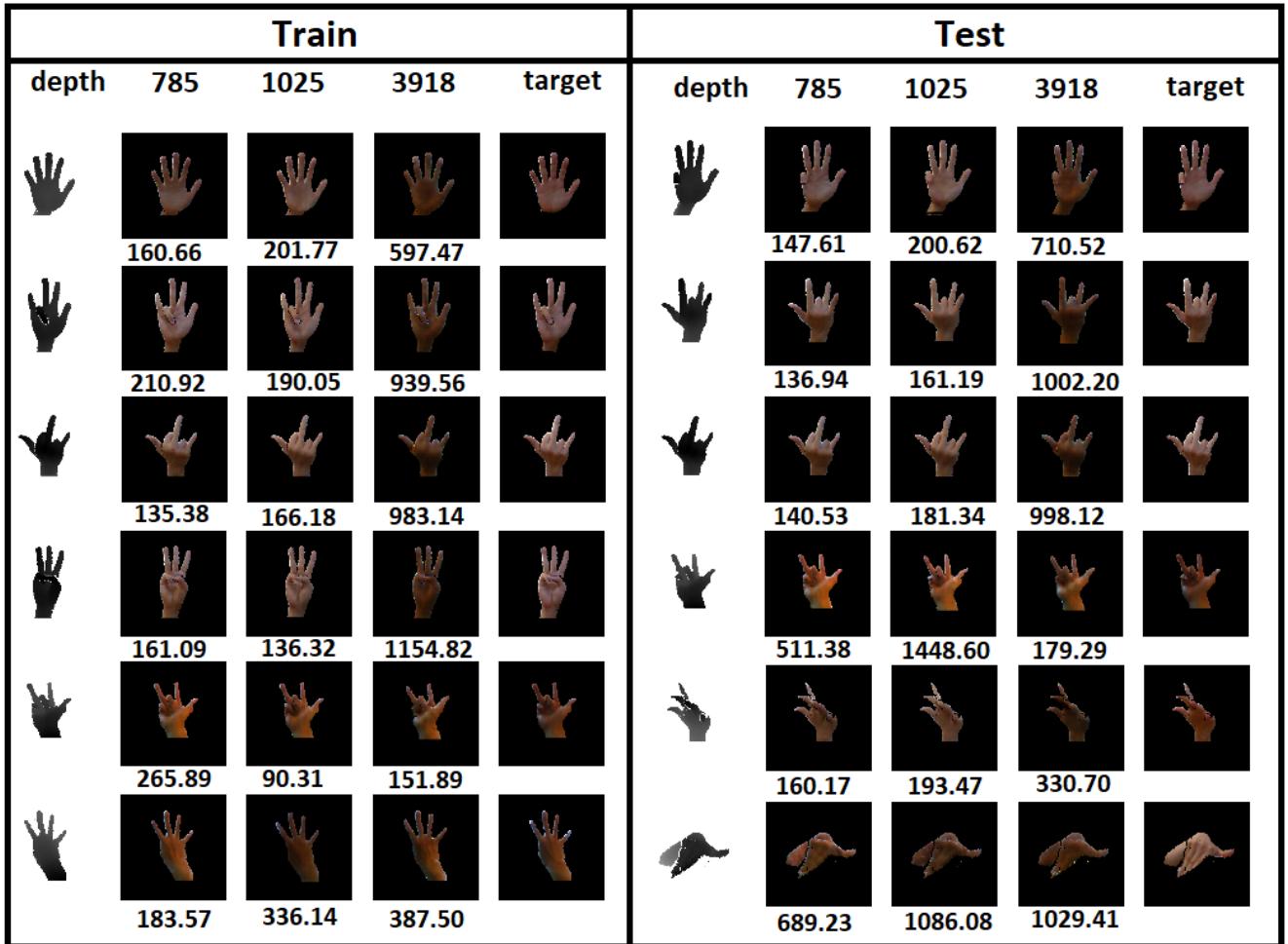


Figure 22: Examples of RGB outputs from different sized datasets trained on the comprehensive dataset.

## 7 Hand Pose Estimation Evaluation

We train a hand pose estimation system as given in Section. 5. The network is trained using existing datasets and also generated datasets using our GAN system for depth to RGB translation for comparison. Based on our results in Section. 6, we choose to use the baseline GAN with *learning\_rate* = 0.002. We also train our GAN using the comprehensive dataset as given in Section. 3.3.2. Training for 1000 epochs and is carried out until the loss converges and does not decrease any further. Loss graphs are given in Appendix. D.4.

We calculate the accuracy of our predicted 21 3D hand joints as the mean Euclidean distance between the predicted,  $\hat{\mathbf{Y}}$ , and ground-truth,  $\mathbf{Y}$ , 3D hand joints given as:

$$3D \text{ Joint Error} = \frac{1}{N} \sum_{i=0}^{N-1} \sqrt{(\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2} \quad (12)$$

$$\mathbf{Y} \in \mathbb{R}^{21 \times 3}$$

$$\hat{\mathbf{Y}} \in \mathbb{R}^{21 \times 3}$$

$N$  is the total number of samples

The total error is given as the mean of the 3D joint error.

We experiment on the following datasets:

**BigHand: Vassileios** A subset of the BigHand [10] dataset consisting of one male human subject. We use 13557 samples in training and testing.

**Generated RGB Bighand: Vassileios** A subset of the BigHand [10] dataset consisting of one male human subject translated to RGB images using our GAN system. We use 13557 samples in training and testing matching that of the original BigHand dataset.

**SynthHands RGB: Male, No Object** A subset of the SynthHands [15] dataset consisting of one male human subject with RGB images. We use 23315 samples in training and 22105 in testing.

We also train using a dataset consisting of a combination of the generated RGB BigHand set and the original depth BigHand set. The testing set consists of a combination of known poses (present in training set) and unknown poses. Only subsets of each dataset are used to reduce the time taken to train a model using the hand pose estimation network.

## 7.1 Results

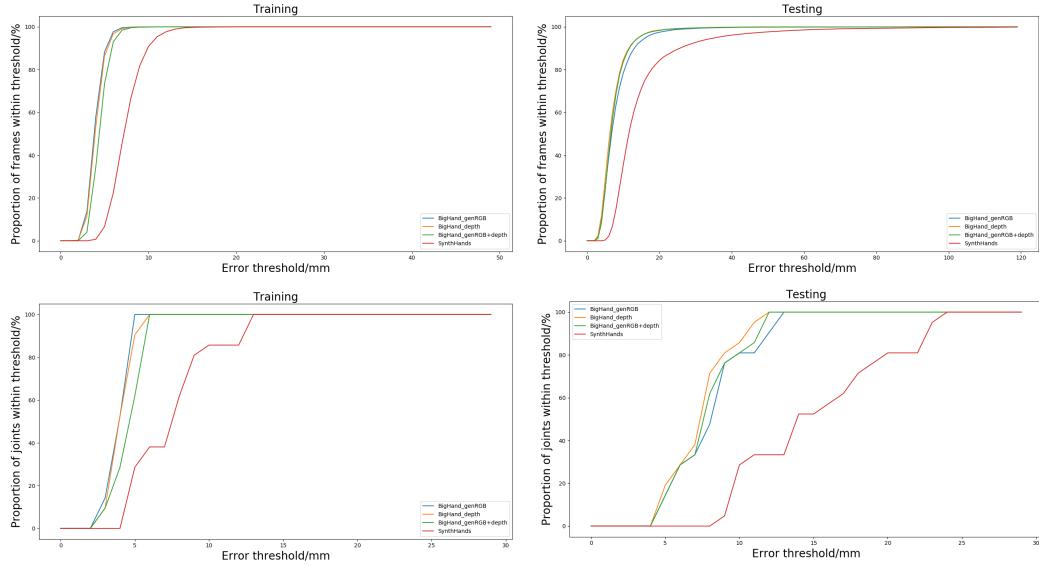


Figure 23: Top: Proportion of image frames with mean joint error within a given error threshold against error threshold. Bottom: Proportion of joints with error within a given error threshold against error threshold.

To analyse our results, we calculate the mean 3D joint error from the estimation of all images in the dataset. We calculate the mean total error as well as mean fingertip error and mean palm error. Furthermore, we plot the proportion of image frames and hand joints within a given error threshold against error threshold given in Fig. 23. The hand is labelled with indices as given in Appendix. D.3. Fingertip positions are given by joint indices [4,8,12,16,20]. Palm positions are given by joint indices [0,3,5,9,13,17]. Our results are tabulated in Table. 5.

	BH:Depth		BH:GenRGB		BH:GenRGB+Depth		SynthHands	
Mean Error/mm	Train	Test	Train	Test	Train	Test	Train	Test
Total	4.0323	7.3114	3.9311	8.0246	4.4747	7.5470	7.4483	15.0619
Fingertip	4.9748	10.3070	4.9121	11.4669	5.5848	10.6034	11.1144	21.1511
Palm	3.0703	4.8155	2.9484	5.1607	3.3495	4.9886	4.5772	9.4238

Table 5: Mean joint hand pose estimation errors for varying datasets. "BH" represents the BigHand dataset.

We see comparable mean errors when evaluating on the BigHand dataset. Even without depth information, our BigHand generated RGB hand images can provide similar error results to the BigHand depth images. We also see that training on both the generated RGB images and depth images do not result in improved results. Comparing results from RGB only datasets, our BigHand generated RGB achieves significantly better results than the SynthHands dataset. This can also clearly be seen in Fig. 23. We suspect this could be due to the synthetic nature of the SynthHands RGB images which lack any detailed hand features such as creases.

	BH:Depth		BH:GenRGB		BH:GenRGB+Depth		SynthHands	
Mean Error/mm	Train	Test	Train	Test	Train	Test	Train	Test
Thumb	4.2097	6.8657	4.0753	7.4067	4.6132	7.0997	6.5033	12.2718
Index	4.1524	7.4186	4.1509	8.392	4.8046	7.8374	8.2019	15.6545
Middle	4.0781	7.6406	3.9813	8.3532	4.4722	7.8533	8.0857	15.9951
Ring	3.9098	7.6242	3.7682	8.2897	4.2526	7.7609	8.1840	17.1325
Pinky	4.0234	7.5916	3.8767	8.3482	4.4865	7.7790	6.9166	15.6995

Table 6: Mean joint hand pose estimation errors for varying datasets for specific fingers. "BH" represents the BigHand dataset.

We also see that the network finds it more difficult to estimate fingertips than palms. From Table. 6, the network finds equal difficulty in inferring the 3D joint position of individual fingers. Finger indices are given as: Thumb [1,2,3,4], Index [5,6,7,8], Middle [9,10,11,12], Ring [13,14,15,16] and Pinky [17,18,19,20]. Mean error on each individual joint is given in Appendix.

D.5. From Table. 7, as expected, we see that generally, the network finds more difficulty in inferring the depth coordinate compared to the  $x$  and  $y$  coordinate especially in the testing set.

	BH:Depth		BH:GenRGB		BH:GenRGB+Depth		SynthHands	
Mean Error/mm	Train	Test	Train	Test	Train	Test	Train	Test
x	1.9064	3.2283	1.8370	3.3904	2.0200	3.2749	3.9624	7.9921
y	2.0233	3.6686	2.000	3.9062	2.3002	3.851	3.6586	6.2921
depth	2.1200	4.0462	2.0583	4.6735	2.3852	4.1688	3.5486	8.4687

Table 7: Mean joint hand pose estimation errors for varying datasets for each coordinate direction. "BH" represents the BigHand dataset.

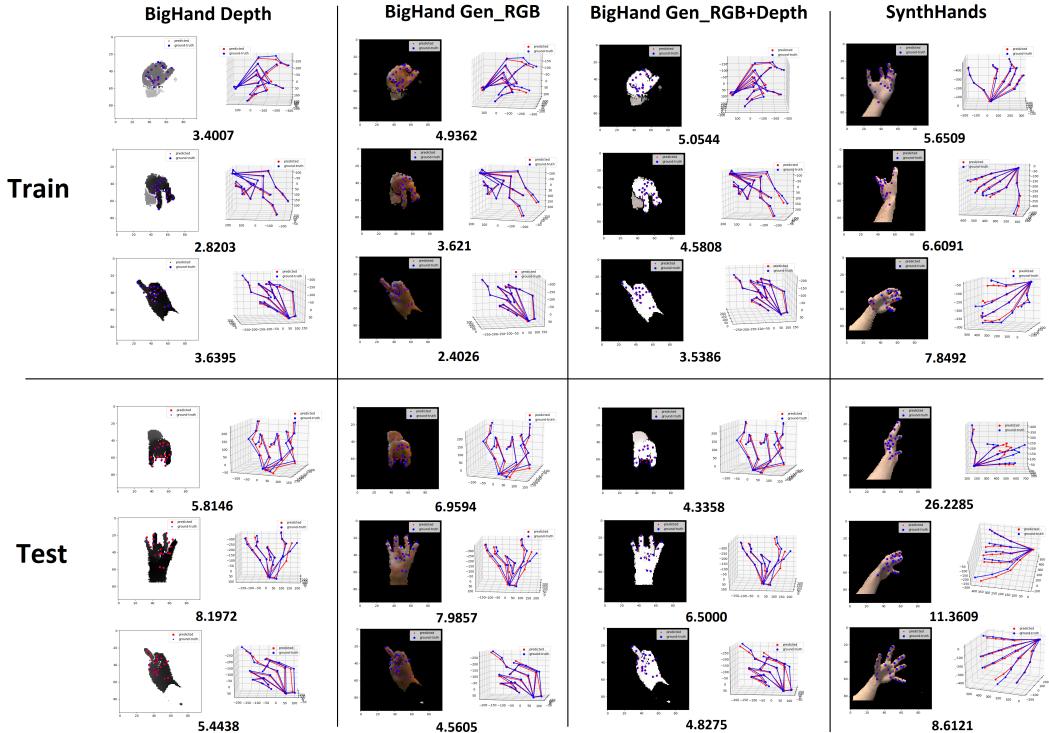


Figure 24: Examples of input images with predicted (red) and ground-truth (blue) hand annotations and corresponding 3D hand skeleton representation. Below each example pair is the corresponding mean joint error.

Individual examples with corresponding error visualisation is given in Fig. 24. Analysing individual samples from the BigHand dataset, error results on the generated RGB dataset shows

generally higher error values on hand poses which suffer from self-occlusion which are worse than the error values obtained from the depth dataset. We believe that the network finds it easier to infer occluded joint positions given depth information than RGB information. This could also be due to the low resolution quality of the image where the network could find ambiguity in hand features and shading. Conversely, we find that the network infers un-occluded joint positions easier on RGB information than depth information i.e. visible fingers that aren't bent. We did not find any strong evidence that training a network with combining both generated RGB images and depth images resulted in improved joint estimation.

As expected, overall testing joint errors are worse than training joint errors. However, we found comparing training and testing errors for similar hand pose images that testing errors are substantially worst. Still, these testing errors are noticeably better than testing errors of images with unseen hand poses.

Results from the SynthHands dataset follows similar trends and displays generic results: 1) Images with occlusions showed relatively worse results compared to those without. 2) Testing errors are higher than training errors. 3) Known poses have better results compared to unknown poses in the testing set. Overall, individual examples showed strictly worse results when compared to results from the BigHand dataset.

## 8 Discussion and Conclusion

Hand pose estimation has seen significant progress recently with the introduction of deep learning techniques which is heavily reliant on high quality, comprehensive datasets which include a wide variety of poses and viewpoints. Development on datasets based on depth information has seen more progress compared to datasets based on RGB information.

We propose a method to leverage the comprehensive BigHand dataset to generate a similarly comprehensive dataset using generative adversarial networks (GAN). We design this network based on the current state-of-the-art in image translation with GANs and train it to translate depth images into RGB images using our own collected dataset. The network is evaluated on

multiple parameters and architecture designs based on current GAN research by analysing the generated images quantitatively and qualitatively. From this we show that our network can reasonably generate realistic RGB hand images using depth image.

Using the best GAN architecture, we generate a dataset for hand pose estimation from the BigHand dataset. We evaluate our dataset by training a baseline hand pose estimation network. The results obtained by our generated RGB dataset are compared with the original depth dataset and also on synthetically generated RGB hand images from the SynthHands dataset. We show that our generated RGB dataset yields comparable results to the original depth dataset. We also show that it yields significantly better results than the synthetically generated RGB dataset. We hope our contribution to generate a comprehensive RGB dataset from the BigHand dataset will lead to further research in hand pose estimation with machine learning methodologies based on RGB images.

## 9 Future Work

Further refinement and fine tuning is still necessary on our GAN architecture and also the corresponding training dataset. From Section. 6.3 we see that our GAN has issues when training on large datasets. We also see that image RGB illumination should be accounted for when acquiring the training dataset. We propose designing a method for normalising image luminosity during the preprocessing stages.

During our experiments, we also only consider 1 subject with 1 hand size and skin colour with only a subset of hand pose orientations. In future experiments, to capture all aspects of the BigHand dataset we require more subjects with a wider variety of hand sizes. To match the BigHand dataset specifically, we would require 10 subjects (7 male, 3 female). A wide variety of skin colours are also necessary when training the hand pose estimator as we would like our model to be as robust as possible. There is concern that a GAN trained on this many variables could confuse it and cause it to merge the features together when generating new images. One possibility to rectify this is to create individual specific GANs for each variable i.e. 1 GAN per hand size or viewpoint.

In terms of hand pose estimation, further analysis is required on large datasets. To reduce training time, we only train our hand pose estimator on only a subset of images within the BigHand and SynthHands dataset. We would like to carry out our evaluation on ideally the entire dataset. We would also like to train our datasets using more current state-of-the-art hand pose estimation networks. It is also important that we carry out the evaluation of our hand pose estimation models on real hand images, ideally in a real time practical situation. After successfully generating an RGB hand pose estimation dataset from the BigHand dataset, an interesting direction would be to look at generating datasets for hand pose estimation on hands suffering from occlusion (holding an object) or two hands in one image.

## 10 Acknowledgements

I would like to thank my supervisor, Dr. Tae-Kyun Kim for considering me for this project and the Imperial Computer Vision & Learning Lab for providing me with the support and resources to carry out this project. Specifically, I would like to express gratitude to Qi Ye for guiding me throughout this project and helpful discussions. Besides that, I would also like to thank Imperial College London for providing me with a conducive environment to carry out this project. Finally, I would like to thank my friends and family for being there for me.

## References

- [1] D. Tang, H. Jin Chang, A. Tejani, and T.-K. Kim, “Latent regression forest: Structured estimation of 3d articulated hand posture,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3786–3793.
- [2] I. Oikonomidis, N. Kyriazis, and A. A. Argyros, “Efficient model-based 3d tracking of hand articulations using kinect.” in *BmVC*, vol. 1, no. 2, 2011, p. 3.
- [3] J. Tompson, M. Stein, Y. Lecun, and K. Perlin, “Real-time continuous pose recovery of human hands using convolutional networks,” *ACM Transactions on Graphics (ToG)*, vol. 33, no. 5, p. 169, 2014.
- [4] Q. Ye, S. Yuan, and T.-K. Kim, “Spatial attention deep net with partial pso for hierarchical hybrid hand pose estimation,” in *European Conference on Computer Vision*. Springer, 2016, pp. 346–361.
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [6] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, “Hand keypoint detection in single images using multiview bootstrapping,” *arXiv preprint arXiv:1704.07809*, 2017.
- [7] F. Gomez-Donoso, S. Orts-Escalano, and M. Cazorla, “Large-scale multiview 3d hand pose dataset,” *arXiv preprint arXiv:1707.03742*, 2017.
- [8] C. Qian, X. Sun, Y. Wei, X. Tang, and J. Sun, “Realtime and robust hand tracking from depth,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1106–1113.
- [9] C. Xu, A. Nanjappa, X. Zhang, and L. Cheng, “Estimate hand poses efficiently from single depth images,” *International Journal of Computer Vision*, vol. 116, no. 1, pp. 21–45, 2016.
- [10] S. Yuan, Q. Ye, B. Stenger, S. Jain, and T.-K. Kim, “Bighand2. 2m benchmark: Hand pose dataset and state of the art analysis,” *arXiv preprint arXiv:1704.02612*, 2017.

- [11] S. Sridhar, A. Oulasvirta, and C. Theobalt, “Interactive markerless articulated hand motion tracking using rgb and depth data,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2456–2463.
- [12] J. Zhang, J. Jiao, M. Chen, L. Qu, X. Xu, and Q. Yang, “3d hand pose tracking and estimation using stereo matching,” *arXiv preprint arXiv:1610.07214*, 2016.
- [13] S. Sridhar, F. Mueller, M. Zollhöfer, D. Casas, A. Oulasvirta, and C. Theobalt, “Real-time joint tracking of a hand manipulating an object from rgb-d input,” in *European Conference on Computer Vision*. Springer, 2016, pp. 294–310.
- [14] C. Zimmermann and T. Brox, “Learning to estimate 3d hand pose from single rgb images,” *arXiv preprint arXiv:1705.01389*, 2017.
- [15] F. Mueller, D. Mehta, O. Sotnychenko, S. Sridhar, D. Casas, and C. Theobalt, “Real-time hand tracking under occlusion from an egocentric rgb-d sensor,” in *Proceedings of International Conference on Computer Vision (ICCV)*, 2017. [Online]. Available: <http://handtracker.mpi-inf.mpg.de/projects/OccludedHands/>
- [16] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, “Real-time human pose recognition in parts from single depth images,” *Communications of the ACM*, vol. 56, no. 1, pp. 116–124, 2013.
- [17] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, and C. Theobalt, “Generated hands for real-time 3d hand tracking from monocular rgb,” *arXiv preprint arXiv:1712.01057*, 2017.
- [18] 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*, 2014, pp. 2672–2680.
- [19] F. Mueller, D. Mehta, O. Sotnychenko, S. Sridhar, D. Casas, and C. Theobalt, “Real-time hand tracking under occlusion from an egocentric rgb-d sensor,” *arXiv preprint arXiv:1704.02201*, 2017.
- [20] X. Zhou, Q. Wan, W. Zhang, X. Xue, and Y. Wei, “Model-based deep hand pose estimation,” *arXiv preprint arXiv:1606.06854*, 2016.

- [21] M. Oberweger, P. Wohlhart, and V. Lepetit, “Hands deep in deep learning for hand pose estimation,” *arXiv preprint arXiv:1502.06807*, 2015.
- [22] T. Sharp, C. Keskin, D. Robertson, J. Taylor, J. Shotton, D. Kim, C. Rhemann, I. Leichter, A. Vinnikov, Y. Wei *et al.*, “Accurate, robust, and flexible real-time hand tracking,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2015, pp. 3633–3642.
- [23] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, “Learning hierarchical features for scene labeling,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2013.
- [24] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, “Face recognition: A convolutional neural-network approach,” *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.
- [25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [27] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.
- [28] A. Wetzler, R. Slossberg, and R. Kimmel, “Rule of thumb: Deep derotation for improved fingertip detection,” *arXiv preprint arXiv:1507.05726*, 2015.
- [29] 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.

- [30] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” *arXiv preprint arXiv:1609.04802*, 2016.
- [31] L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, “Pose guided person image generation,” *arXiv preprint arXiv:1705.09368*, 2017.
- [32] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.
- [33] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein gan,” *arXiv preprint arXiv:1701.07875*, 2017.
- [34] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” *arXiv preprint arXiv:1411.1784*, 2014.
- [35] 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.
- [36] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” *arXiv preprint arXiv:1605.05396*, 2016.
- [37] 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.
- [38] T. Kim, M. Cha, H. Kim, J. Lee, and J. Kim, “Learning to discover cross-domain relations with generative adversarial networks,” *arXiv preprint arXiv:1703.05192*, 2017.
- [39] “What is my hand size?” [Online]. Available:  
<https://choosehandsafety.com/choosing-hand-tools/hand-tool-size>
- [40] Y. Wu, J. Y. Lin, and T. S. Huang, “Capturing natural hand articulation,” in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 426–432.
- [41] “Intel® realsense™ sdk for windows (discontinued),” Aug 2017. [Online]. Available:  
<https://software.intel.com/en-us/realsense-sdk-windows-eol>

- [42] P. Isola, “pix2pix,” <https://github.com/phillipi/pix2pix>, 2017.
- [43] E. L. Denton, S. Chintala, R. Fergus *et al.*, “Deep generative image models using a laplacian pyramid of adversarial networks,” in *Advances in neural information processing systems*, 2015, pp. 1486–1494.
- [44] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context encoders: Feature learning by inpainting,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2536–2544.
- [45] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, “Autoencoding beyond pixels using a learned similarity metric,” *arXiv preprint arXiv:1512.09300*, 2015.
- [46] D. H. T. Hien, “Guide to receptive field arithmetic for convolutional neural networks,” 2017. [Online]. Available: <https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807>
- [47] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International Conference on Machine Learning*, 2015, pp. 448–456.
- [48] A. L. Maas, A. Y. Hannun, and A. Y. Ng, “Rectifier nonlinearities improve neural network acoustic models.”
- [49] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *science*, vol. 313, no. 5786, pp. 504–507, 2006.
- [50] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of wasserstein gans,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5769–5779.
- [51] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.
- [52] yenchenlin, “pix2pix-tensorflow,” <https://github.com/yenchenlin/pix2pix-tensorflow>, 2017.

- [53] carpedm20, “Dcgan-tensorflow,”  
<https://github.com/carpedm20/DCGAN-tensorflow/blob/master/model.py>, 2017.
- [54] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [55] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, “Max-pooling convolutional neural networks for vision-based hand gesture recognition,” in *Signal and Image Processing Applications (ICSIPA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 342–347.
- [56] F. Chollet *et al.*, “Keras,” <https://keras.io>, 2015.
- [57] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

# Appendices

## A Segmented hand mask

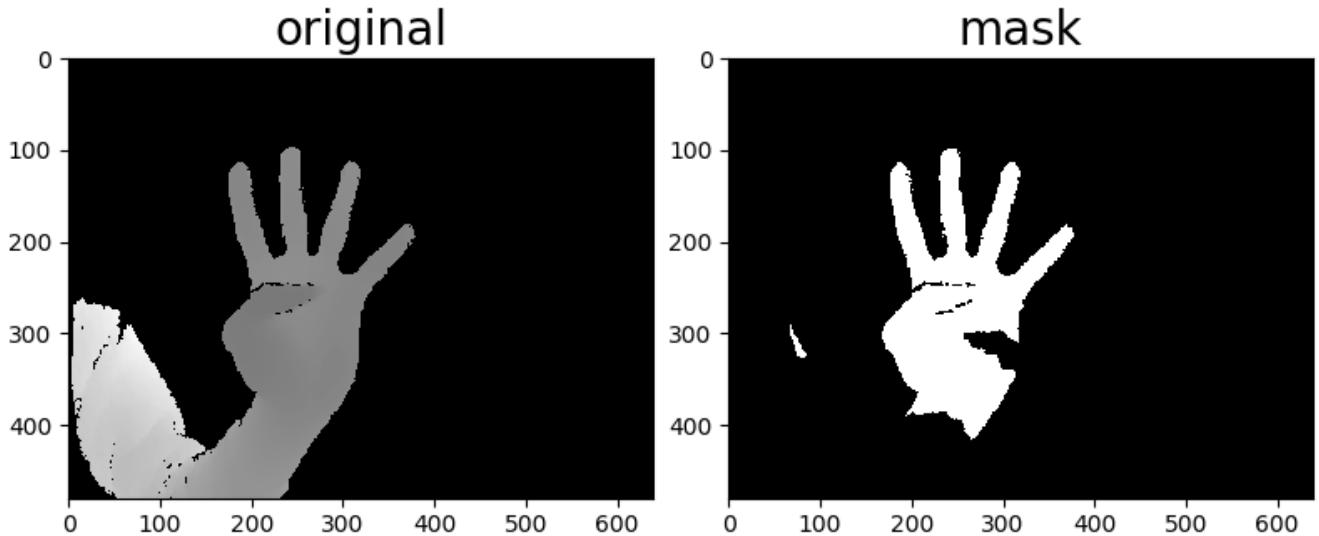


Figure 25: Segmented hand during preprocessing stage of dataset generation using hand classifier

## B Baseline GAN

### B.1 Network architecture

The network architecture is based off that implemented by the `pix2pix` software which is adapted from DCGANs [32].

The base convolutional unit is denoted as  $C_k$  where  $k$  represents the number of filters. This is a convolutional layer with a batch norm and ReLU layer applied. Similarly,  $DC_k$  is a deconvolutional layer with a batch norm, ReLU and dropout layer applied.

### B.1.1 Generator architecture

The encoder is given as:

$$C_{64} - C_{128} - C_{256} - C_{512} - C_{512} - C_{512} - C_{512}$$

The decoder is given as:

$$DC_{512} - DC_{512} - DC_{512} - DC_{512} - DC_{256} - DC_{128} - DC_{64}$$

Differing from earlier, the first convolutional layer,  $C_{64}$  does not include a batch norm layer and the ReLUs in the decoder are not leaky with  $\alpha = 0$ .

### B.1.2 Discriminator architecture

The  $70 \times 70$  discriminator is given as:

$$C_{64} - C_{128} - C_{256} - C_{512}$$

The  $286 \times 286$  discriminator is given as:

$$C_{64} - C_{128} - C_{256} - C_{512} - C_{512} - C_{512}$$

Differing from earlier, the first convolutional layer,  $C_{64}$  does not include a batch norm layer and the final convolutional layer is applied to a 1 dimensional output with  $stride = 1$

### B.1.3 Settings

#### Convolutional/Deconvolutional Layer

Weights are initialised from a Gaussian distribution with  $\mu = 0$  and  $\sigma^2 = 0.02$ .

- $kernel\_size = 4 \times 4$

- $stride = 2$
- $padding = SAME$

## Batch Norm

- $\epsilon = 0.00001$
- $momentum = 0.9$

## Leaky ReLU

- $\alpha = 0.00001$

## Dropout

- $keep\_prob = 0.5$

## B.2 Optimisation

### B.2.1 Settings

#### ADAM

- $learning\_rate = 0.0002$
- $\beta_1 = 0.5$
- $\beta_2 = 0.999$

## B.3 Training Analysis

### B.3.1 Training Samples

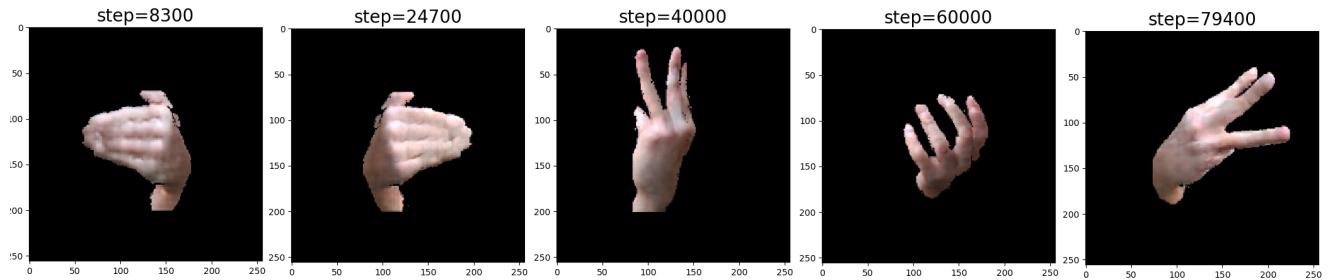


Figure 26: Examples of generated samples taken at different steps during the training

### B.3.2 Epoch 100

This section contains results obtained from training a baseline GAN with default settings using the initial analysis dataset given in Section. 3.3.1 for 100 epochs.

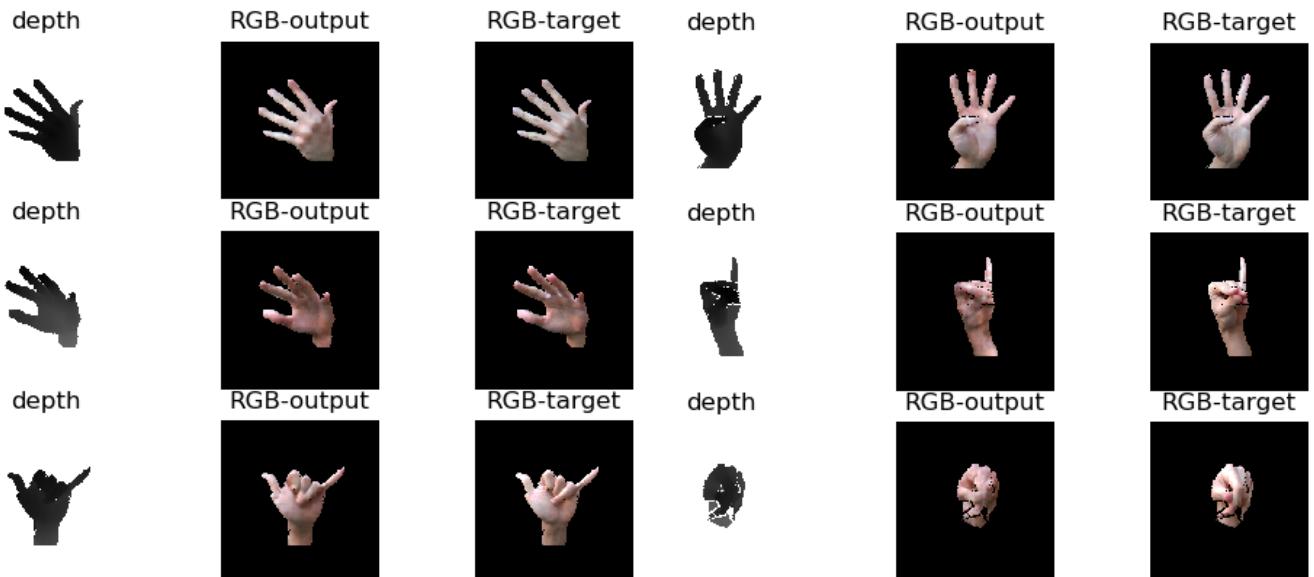


Figure 27: Examples of generated RGB output from baseline GAN patch size  $70 \times 70$  with learning rate of 0.0002 for 100 epochs. Left: Examples generated from training data. Right: Examples generated from testing data.

Epoch	Train	Test
100	185.5800	238.1461
200	162.7022	209.1948

Table 8: Mean squared error of image datasets generated from training and testing images while varying epoch

## B.4 Learning Rate

### B.4.1 0.00002

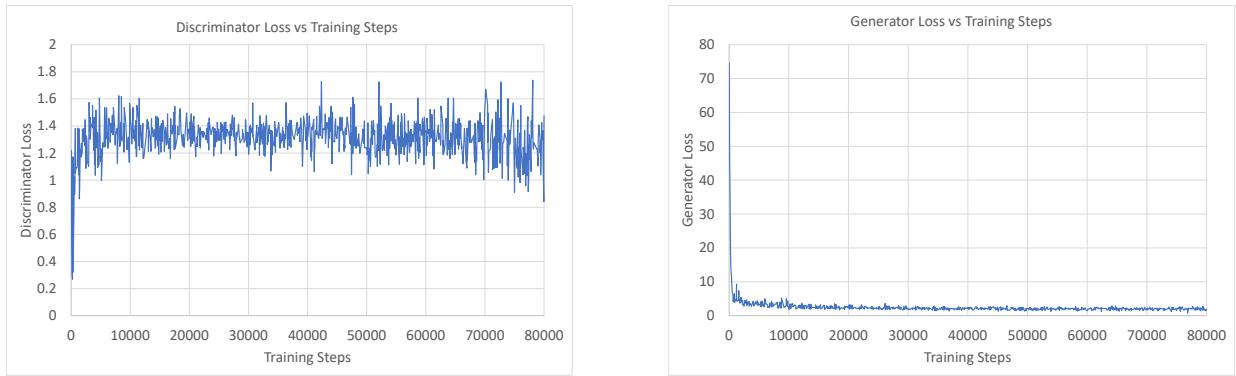


Figure 28: Loss graphs from 70x70 Baseline GAN with learning rate of 0.00002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

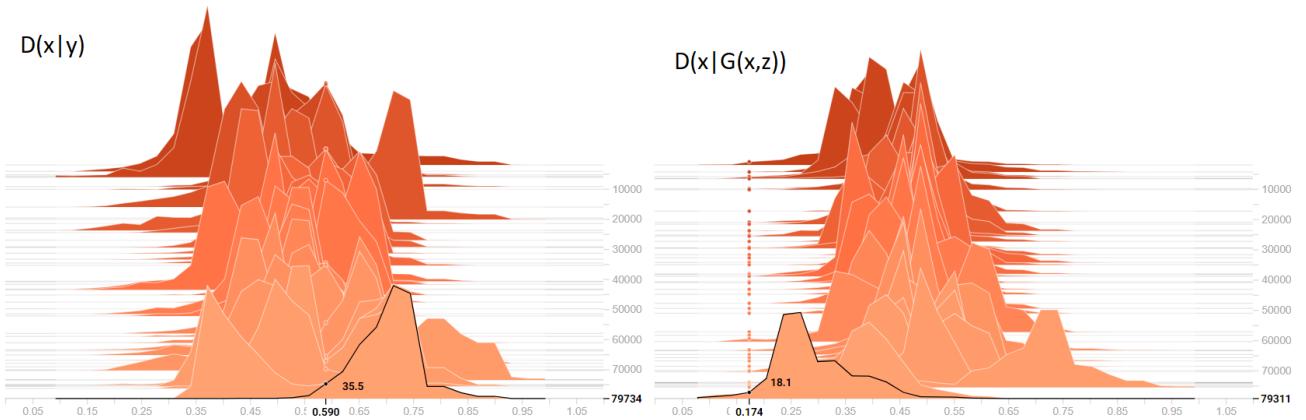


Figure 29: Loss graphs from 70x70 Baseline GAN with learning rate of 0.00002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

### B.4.2 0.002

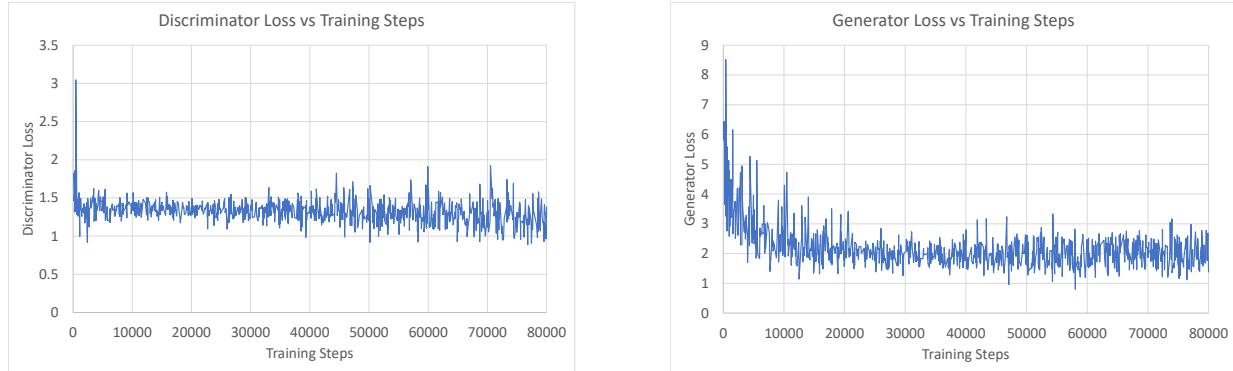


Figure 30: Loss graphs from 70x70 Baseline GAN with learning rate of 0.002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

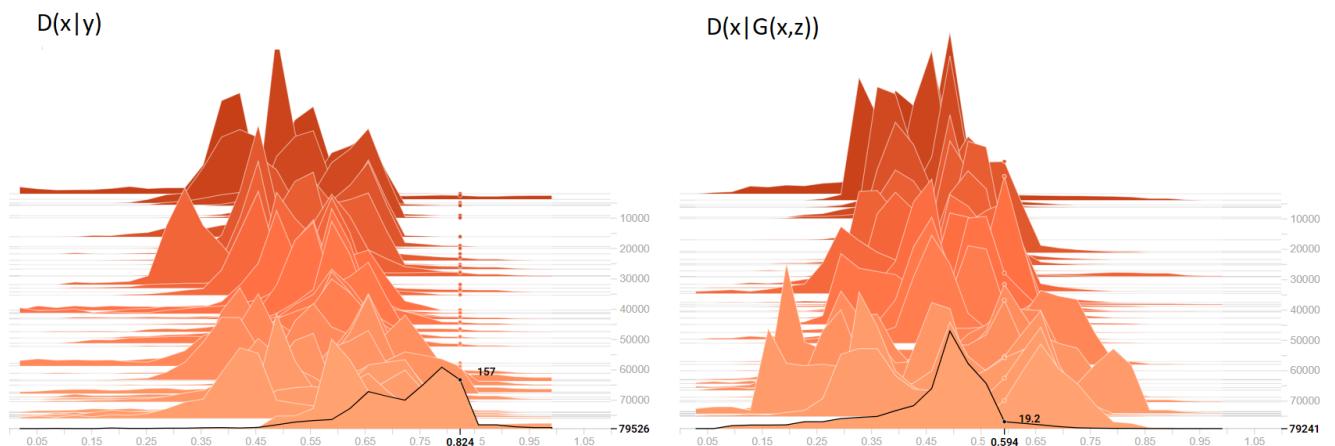


Figure 31: Moving probability distributions from 70x70 Baseline GAN with learning rate of 0.002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

## B.5 Patch Size

### B.5.1 32x32

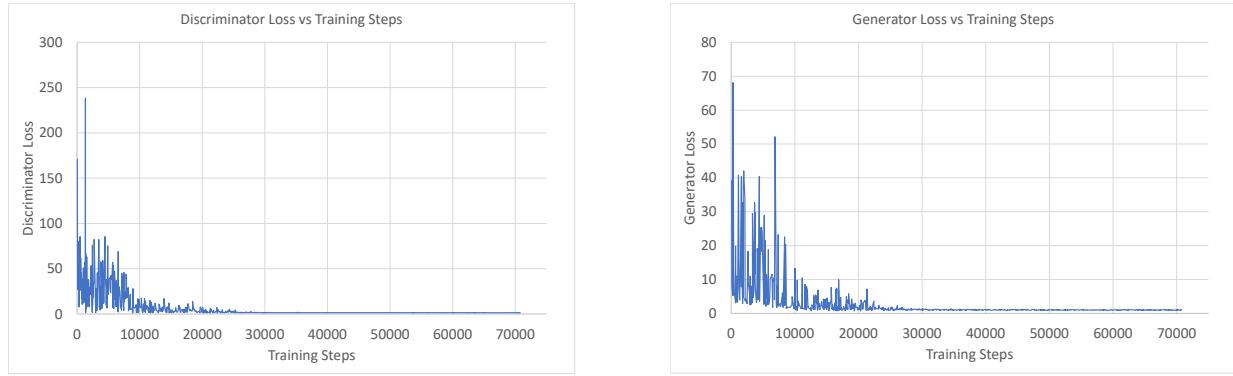


Figure 32: Loss graphs from 32x32 Baseline GAN with learning rate of 0.0002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

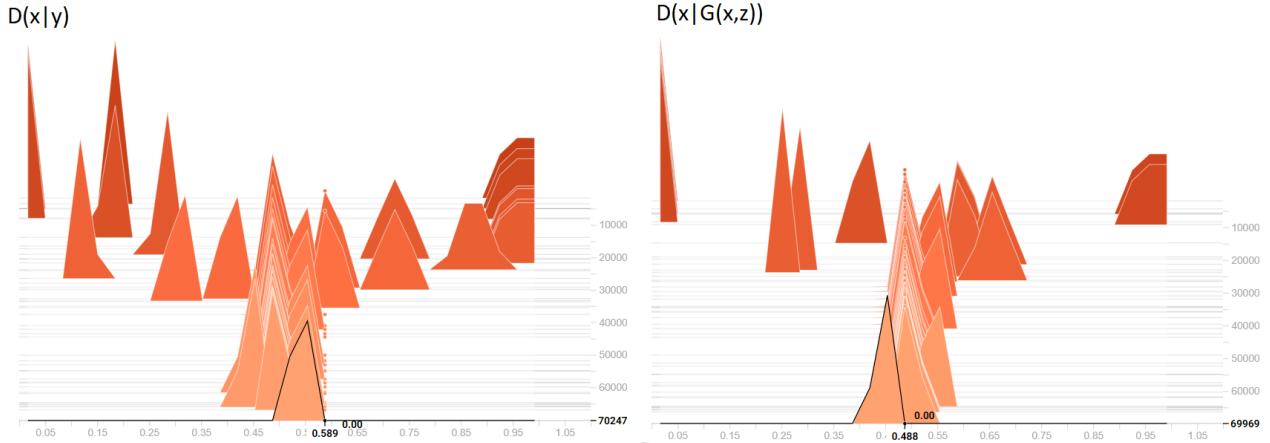


Figure 33: Moving probability distributions from 32x32 Baseline GAN with learning rate of 0.0002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x, z))$ .

### B.5.2 286x286

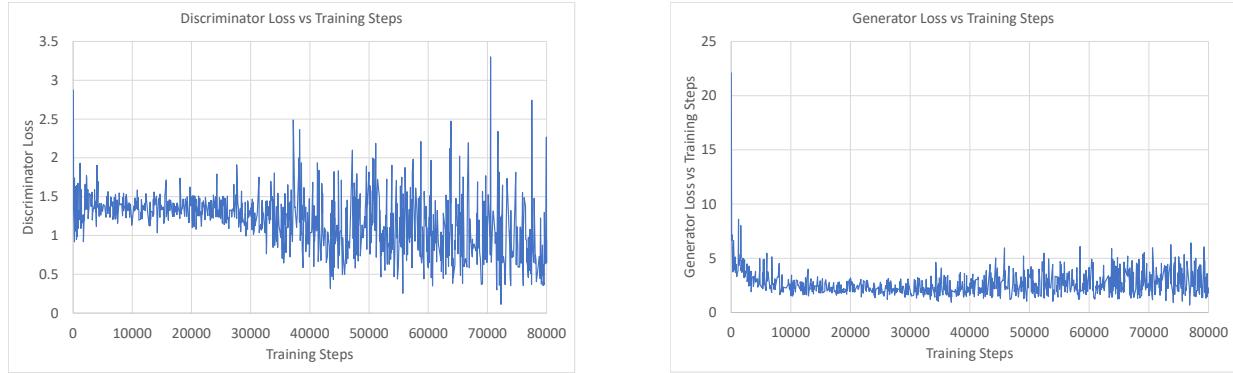


Figure 34: Loss graphs from 286x286 Baseline GAN with learning rate of 0.0002. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

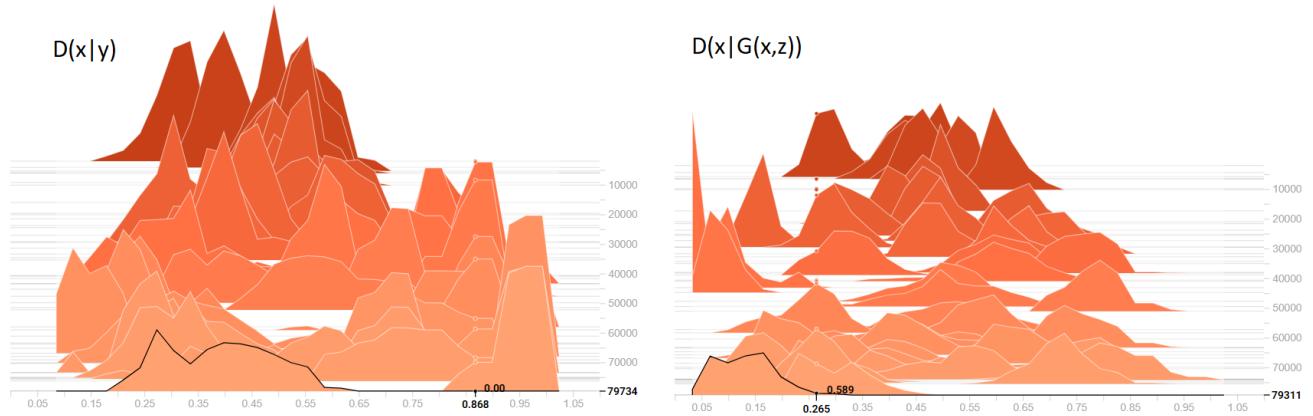


Figure 35: Moving probability distributions from 286x286 Baseline GAN with learning rate of 0.0002. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

## B.6 Comprehensive Dataset

### B.6.1 1025

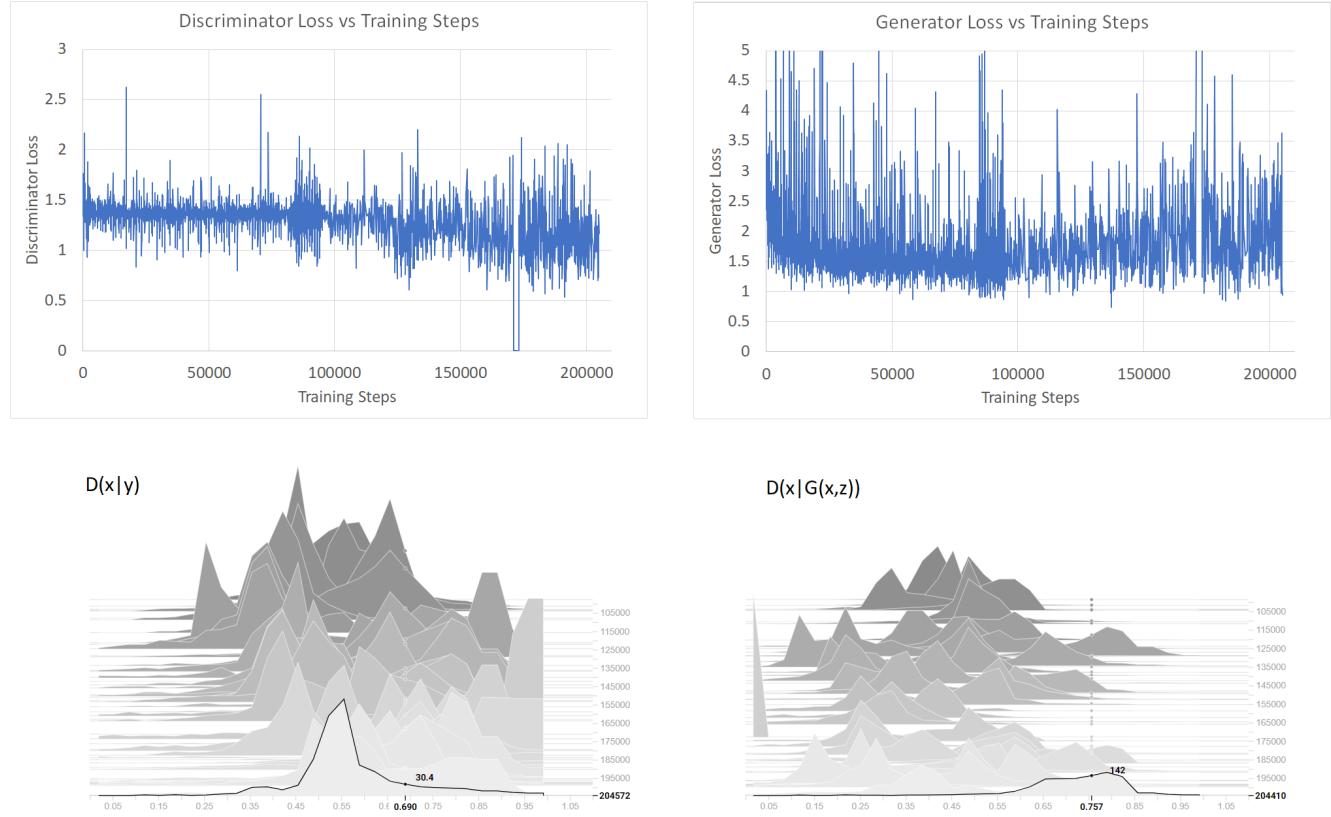


Figure 36: Training graphs from 70x70 Baseline GAN with learning rate of 0.002 trained on comprehensive dataset with 1025 training pairs. Top: Discriminator and generator loss graphs. Bottom: Moving probability distribution of  $D(x|y)$  and  $D(x|G(x,z))$ .

## C Laplacian Pyramid

### C.1 Generated Images

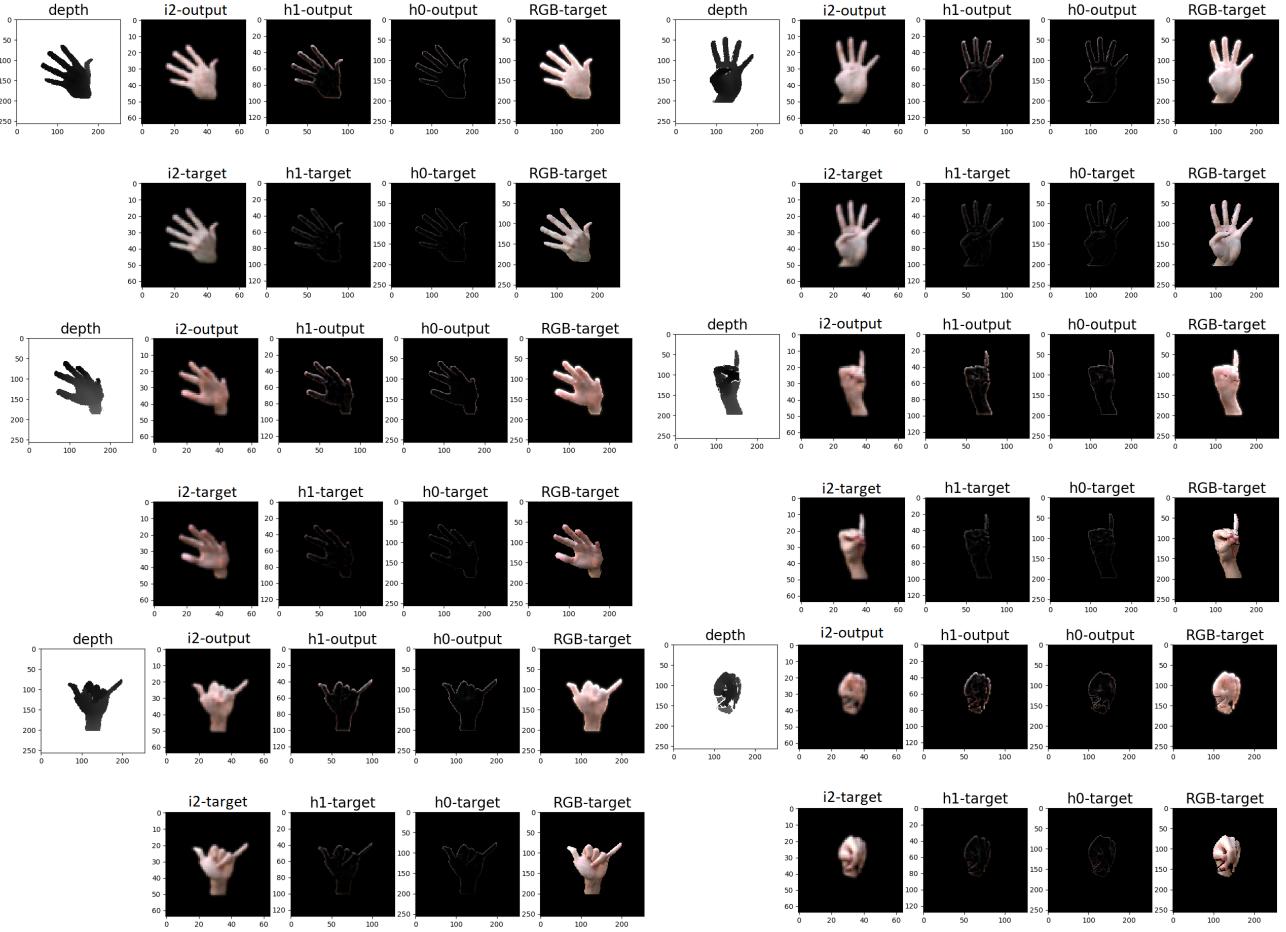


Figure 37: Examples of generated output from 70x70 GAN with Laplacian Pyramid and learning rate of 0.0002 showing each individual generated output. Right: Images generated from training samples. Left: Images generated from testing samples.

## C.2 Training Analysis

### C.2.1 h1

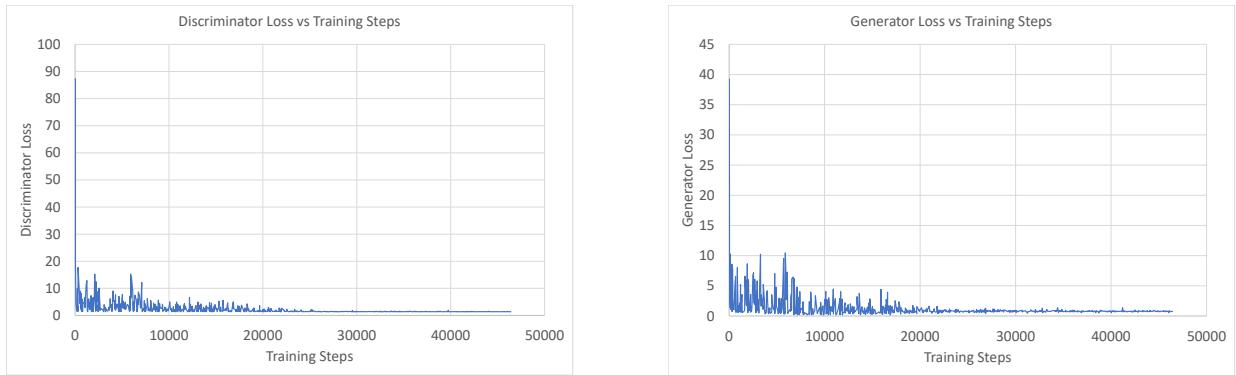


Figure 38: Loss graphs from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid coefficient h1. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

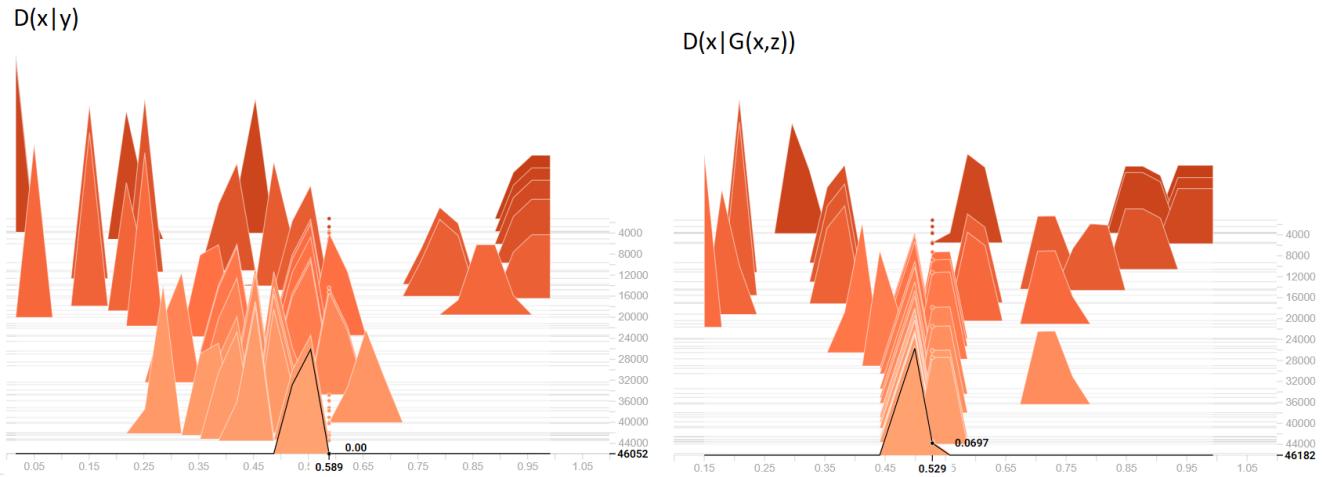


Figure 39: Moving probability distributions from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid coefficient h1. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

### C.2.2 h0

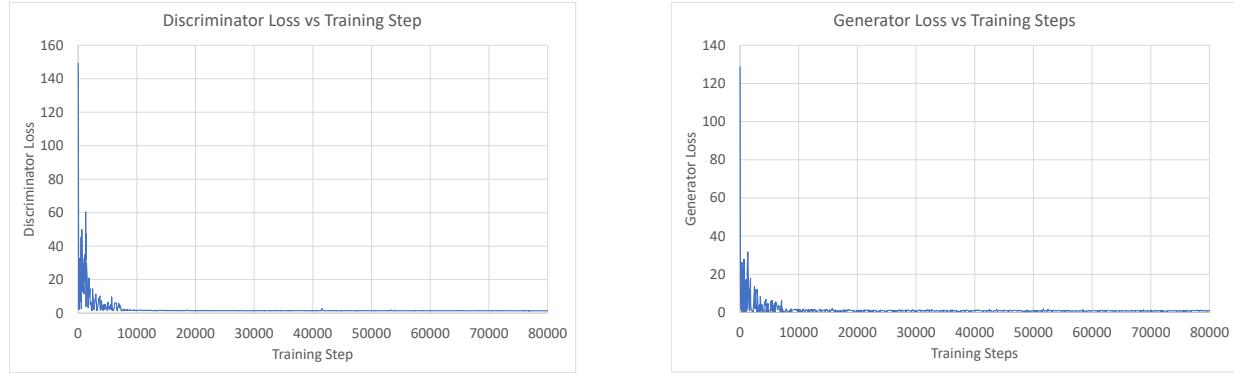


Figure 40: Loss graphs from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid coefficient h0. Left: Discriminator loss against training steps. Right: Generator loss against training steps.

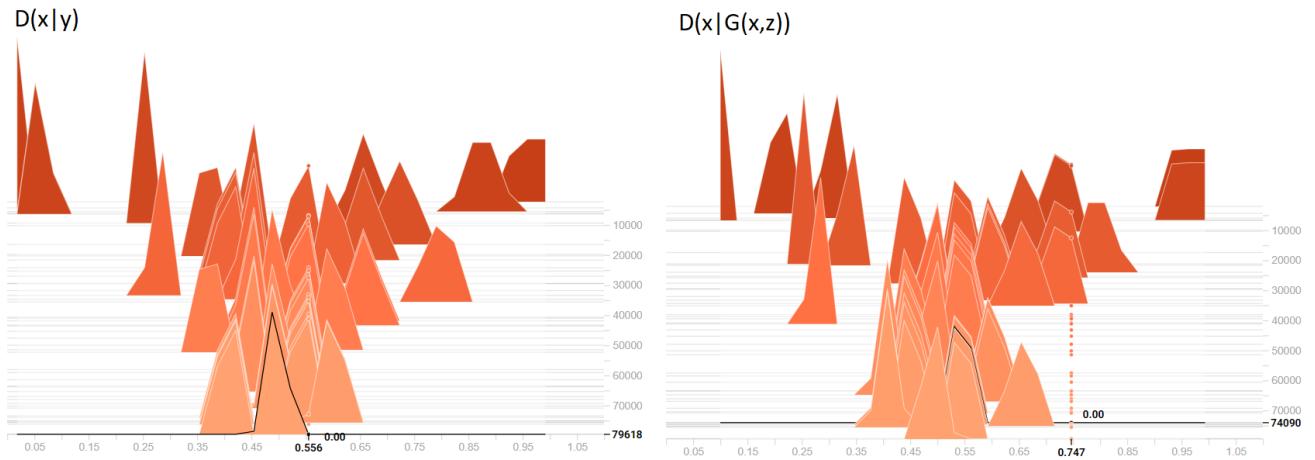


Figure 41: Moving probability distributions from 70x70 Baseline GAN with learning rate of 0.0002 and Laplacian Pyramid coefficient h0. Left: Moving probability distribution of  $D(x|y)$ . Right: Moving probability distribution of  $D(x|G(x,z))$ .

## D Hand Pose Estimation

The base convolutional unit is denoted as  $C_{i,j,k}$  where  $i$  represents the number of filters,  $j$  represents the kernel size,  $j \times j$  and  $k$  represents the size of the max-pooling layer,  $k \times k$ . This is a convolutional layer with a and ReLU layer applied.

### D.1 Network Architecture

96x96 image network:  $C_{32,5,4} - C_{64,3,2} - C_{96,3,2}$

48x48 image network:  $C_{32,5,2} - C_{64,3,2} - C_{96,3,2}$

24x24 image network:  $C_{32,5,2} - C_{64,5,1} - C_{96,3,1}$

#### D.1.1 Settings

##### Convolutional Layer

- $stride = 1$
- $padding = VALID$

##### Batch Norm

- $\epsilon = 0.001$
- $momentum = 0.99$

##### Dropout

- $keep\_prob = 0.7$

## D.2 Optimisation

### D.2.1 Settings

#### ADAM

- $\text{learning\_rate} = 0.0001$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$

### D.3 Joint positions

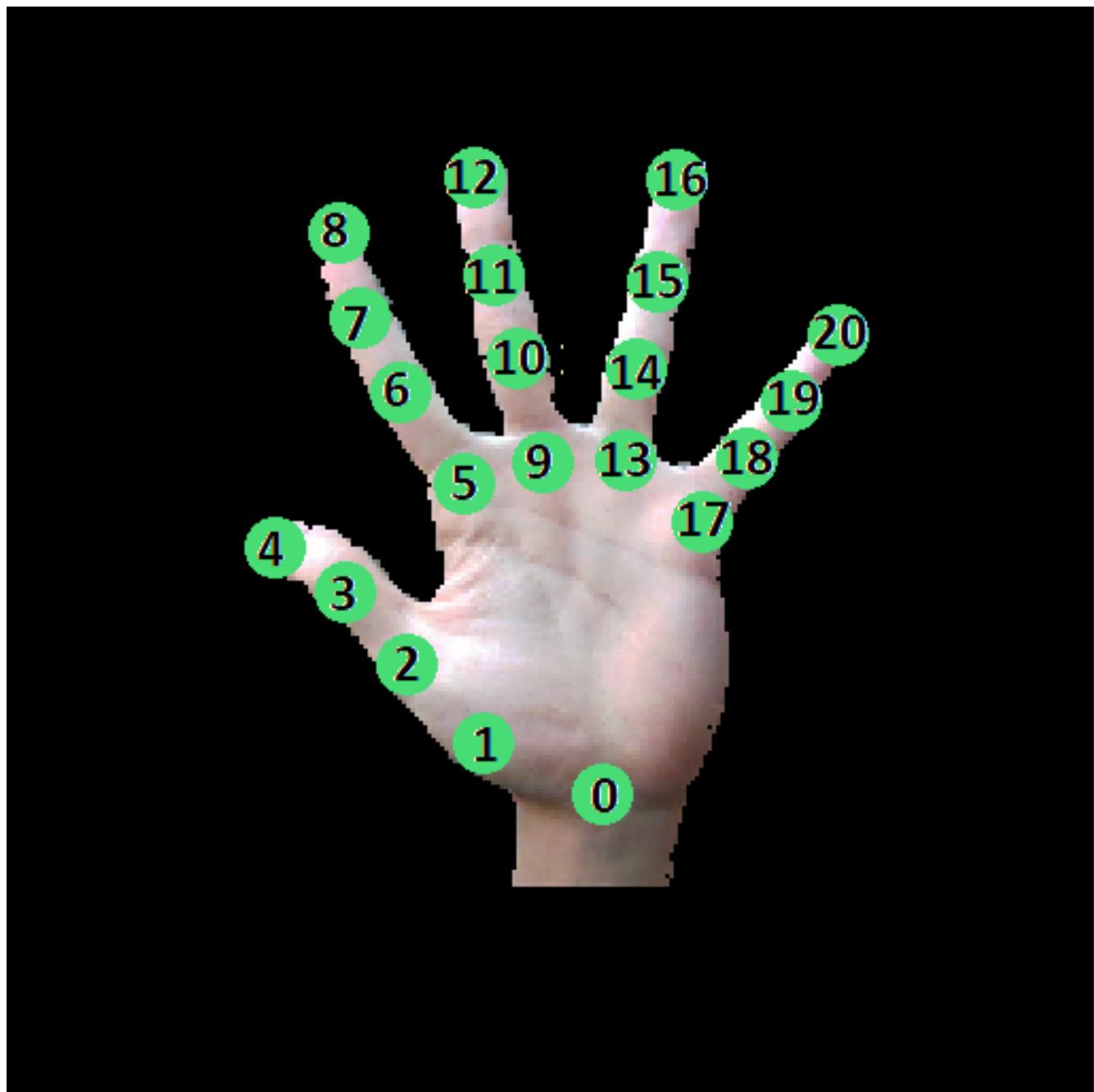


Figure 42: Labeled joint positions

## D.4 Training Analysis

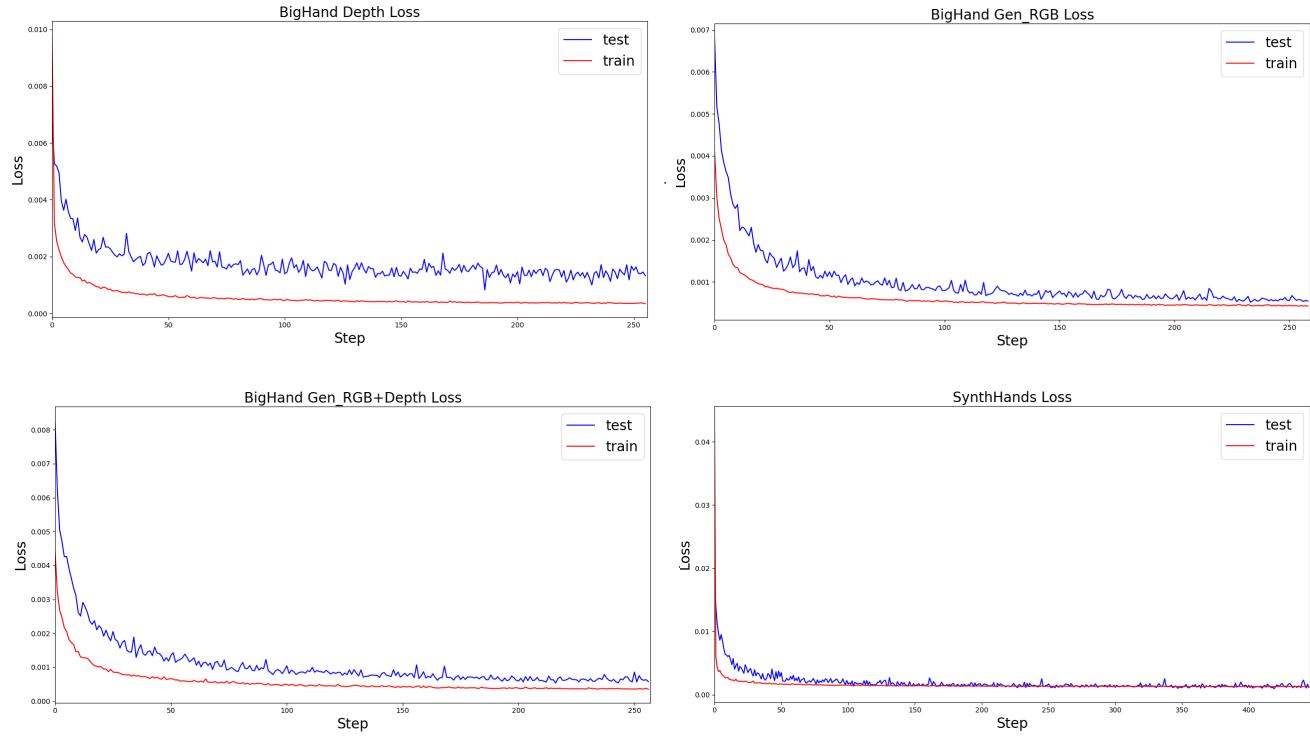


Figure 43: Training and testing loss against steps during hand pose estimation network training with varying datasets.

## D.5 Average Joint Error

	BH:Depth		BH:GenRGB		BH:GenRGB+Depth		SynthHands	
Index	Train	Test	Train	Test	Train	Test	Train	Test
0	3.1856	4.9766	3.1438	5.3568	3.4522	5.1662	4.8484	9.2859
1	3.4619	5.3759	3.4084	5.7988	3.7712	5.5974	4.3849	8.8728
2	4.7443	7.1321	4.4513	7.4687	5.0842	7.2848	5.5795	10.7419
3	3.6341	6.2056	3.5423	6.8420	4.0098	6.4598	7.1088	13.4071
4	4.9985	8.7492	4.8992	9.5173	5.5876	9.0568	8.9398	16.0656
5	3.3743	5.2369	3.3056	5.6844	3.7552	5.4723	4.8216	9.8161
6	4.8374	7.4779	4.7598	8.1837	5.4418	7.8317	7.1548	13.5100
7	3.6034	7.1942	3.6230	8.3633	4.3137	7.6562	8.8129	17.2357
8	4.7944	9.7654	4.9152	11.3368	5.7077	10.3893	12.0183	22.0563
9	2.6894	4.3744	2.5113	4.6692	2.9858	4.5397	4.2275	9.3342
10	4.8500	7.7479	4.7070	8.2663	5.3931	8.0494	7.0800	13.9817
11	3.7770	7.5923	3.7543	8.4916	4.0945	7.7842	8.8823	18.5493
12	4.9960	10.8480	4.9528	11.9859	5.4155	11.0401	12.1532	22.115
13	2.6145	4.2109	2.4175	4.4618	2.8443	4.3323	4.3282	9.4203
14	4.2654	7.0591	4.1790	7.6044	4.6950	7.2661	7.0298	15.8854
15	3.7435	8.0162	3.6112	8.8127	4.0630	8.1076	8.7524	19.9073
16	5.0156	11.2106	4.8650	12.2798	5.4082	11.3375	12.6254	23.3169
17	3.0958	4.7183	2.9038	4.9929	3.2884	4.8238	4.8528	9.8134
18	4.2043	6.8165	4.1195	7.3927	4.6122	7.0125	5.6272	13.3674
19	3.7241	7.8695	3.5552	8.7928	4.2401	8.0867	7.3514	17.4156
20	5.0695	10.9620	4.9284	12.2145	5.8051	11.1931	9.8352	22.2017

Table 9: Average joint hand pose estimation errors for varying datasets on individual joint indices as given in Appendix. D.3.