# Learning from Simulated and Unsupervised Images through Adversarial Training

Abhishake Kumar Bojja
V00892833
Computer science
University of Victoria
abojja@uvic.ca

## Abstract

*Synthetic images always helps in places where we are in shortage of real world data, as they are readily available with corresponding annotations. But machine learning models trained on synthetic images gives poor performance on real images as the models learn the artifacts from synthetic images. In this project, I implemented an adversarial network, which adds realistic features to the synthetic images and make them real and natural. These network generated images can be used to train machine learning models to give better performance on real images.*

## 1. Introduction

In supervised learning, a lot of amount of labeled data is required to train machine learning/deep learning models for better accuracies. Collecting massive datasets is a severe problem. In some cases, it is easy to get an enormous amount of data, but it is challenging to annotate them, which requires a lot of human effort. An alternative approach for this scenario is to make use of simulated data, generated from a simulator. The simulated data contains automatically generated labels, which helps the machine learning models to train better. Since the images generated from the simulator is synthetic, it misses the characteristics present in the real world data. So, the models trained on this simulated data will try to learn the artifacts present in them and fail to perform well on actual data. If the synthetic data is made to look realistic, then the models work well on real-world data. But making simulator to generate realistic information is a computationally expensive problem. The goal of this project is to add realism to the synthetic images made from simulator by using unlabeled real data through the process of adversarial training.

Towards this goal, the eye gaze prediction problem is used to do the experiments. In eye gaze prediction problem, the goal is to predict the gaze direction of the eye, i.e., to anticipate where the eyes are pointing in the space, given an image of the eye. To get significant real-world data with perfect annotations is difficult. So the unlabeled MPIIGaze dataset is used to add realism to the synthetic data generated from Unity eyes simulator. The adversarial training is done by using two networks namely Refiner and Discriminator, which are similar to generator and discriminator networks of Generative Adversarial Network (GAN). While adding realism to the synthetic images, the annotations of the simulated data must be preserved, which otherwise makes the data useless.

## 2. Contributions of original paper

The main idea of the work [2] is to make the synthetic images realistic using unlabelled real data through adversarial training. The critical contribution of the paper is the architecture of the Refiner and Discriminator networks used for adversarial training. The refiner network is a fully convolutional neural network build with identity blocks used in the residual network (ResNet), and the discriminator network is also a fully convolutional neural network. The refiner network is trained on synthetic images generated from a simulator and produces refined images as the output. The discriminator network is trained on unlabeled real and refined images alternately. Both the networks are trained alternately by minimizing adversarial loss functions using gradient descent optimization.

Moreover, to preserve the annotations of the synthetic data, a self-regularization loss is added to the adversarial loss of the refiner. The self-regularization loss is an L-1 norm used to reduce the difference between the synthetic and refined images.

So, the training loss functions for both refiner and discriminator are as follows:
The training loss for discriminator is:

$$L_D = -log(1 - D(y)) - log(D(R(x))) \tag{1}$$

where -log(1-D(y)) is making label 0 for real images and -log(D(R(x))) for making label 1 for refined images.
The training loss for the refiner is:

$$L_R = -log(1 - D(R(x))) + \lambda * ||R(x) - x|| \quad (2)$$

where $-log(D(R(x)))$ for making label 0 for refined images and $\lambda * ||R(x) - x||$ is self regularization loss used for preserving annotations. Here lambda is regularization parameter and ||.|| is L-1 norm.

In a generative adversarial network, for a discriminator, the loss is calculated globally. For a synthetic image the label is set to 1, and for a real image, the label is set to 0. So, the discriminator looks at the output image of the discriminator globally and tries to minimize the loss accordingly. Instead of the global loss, this paper suggested using local adversarial loss, which means dividing the entire output of the image into several local patches and the loss is defined for individual patches. The final adversarial loss is the average of all the individual local patch losses. With this modification, they observed the reduction of artifacts in the refined images as now the network is forced to learn realistic features locally. In addition to this, to improve the training of this GAN like network, they suggested adding a part of the history of refined images to the current refined images. Generally, a discriminator is trained on a batch of refined images and real images alternately. To the batch of refined images, half of the batch is replaced by the past refined images. By considering this refined images, they observed that the refiner network achieves realistic images for synthetic ones more fastly.

## 3. Contributions of this project

I implemented the experiments on eye gaze prediction problem in this project. Towards this, I explained my implementation details below.

### 3.1. Data

For this project, we require both synthetic eye dataset and real eye dataset and the authors of the paper used UnityEyes gaze estimation synthetic dataset and MPIIGaze real dataset. I used Eye gaze kaggle dataset [1], which was specifically prepared for reproducing this paper. The dataset contains the following files

#### 3.1.1 $synthetic\_gaze.h5$

This hdf5 file contains 50,000 synthetic eye gaze images simulated from the UnityEyes tool. The size of each image is 35x55 and is available in *uint8* format.

#### 3.1.2 $synthetic\_gaze.csv$

This file contains the labels for the 50,000 synthetic eye gaze images present in synthethic_gaze.h5. The label is a value representing the gaze information.

#### 3.1.3 $real\_gaze.h5$

This hdf5 file contains 34,100 unlabeled real eye gaze images simulated from the UnityEyes tool. The size of each image is 35x55 and is available in *uint8* format.

The range of values of each image available in this dataset is 0 to 255. All the images are preprocessed using keras preprocessing library functions. After this preprocessing stage, each image data is in the range -1.0 to 1.0 and is in *float32* format. To convert entire data into batches, which can be readily fed to the networks for training, I used keras ImageDataGenerator functions to generate batches of images. The parts related to data loading and preprocessing are implemented under $\_build\_data$ function in the code.

### 3.2. Algorithm

#### 3.2.1 Networks

I used the same architecture used by the authors to implement both the refiner and discriminator networks.

*Refiner Network*: The refiner network is a fully convolutional neural network. The input to this network is 35x55 image and is convolved with 64 3x3 filters. This convolved image is then sent to 4 resnet blocks. The resnet block is similar to the identity blocks present in ResNet architecture and contains a conv2d layer, relu layer, a conv2d layer. Both the conv2d layers contain 3x3 filters with 64 feature maps and stride is set to 1. The output of the resnet block is convolved with 1x1 filter, which generates the refined image. This architecture is implemented under $\_build\_refiner$ function.

*Discriminator Network*: The discriminator is also a fully convolutional neural network, which takes a 35x55 image as input. The input image is passed through a series of convolutional and max pooling layers as follows: a) conv2d layer with 96 3x3 filters, stride=2 b) conv2d layer with 64 3x3 filters, stride=2 c) maxpool layer with 3x3 filter, stride=1 d) conv2d layer with 32 3x3 filters, stride=1 e) conv2d layer with 32 1x1 filters f) conv2d layer with 2 1x1 filters. The output at this stage consists of logits, which is passed to a softmax layer to get the final output. This architecture is implemented under $\_build\_discriminator$ function.

#### 3.2.2 Training Loss Functions

I used the same loss functions implemented in the paper. If x is an synthetic image, R(x) is a refined image and D(R(x)) is the output of the discriminator. If y is a real image, then

the discriminator output for real image id D(y). Here R is refiner network and D is a discriminator network.
The training loss for discriminator is:

$$L_D = -log(1 - D(y)) - log(D(R(x))) \quad (3)$$

where -log(1-D(y)) is making label 0 for real images and -log(D(R(x))) for making label 1 for refined images.
The training loss for the refiner is:

$$L_R = -log(D(R(x))) + \lambda * ||R(x) - x|| \quad (4)$$

where $-log(D(R(x)))$ is for making label 0 for refined images and $\lambda * ||R(x) - x||$ is self regularization loss used for preserving annotations. Here lambda is regularization parameter and $||.||$ is L-1 norm. The loss functions are implemented under $\_build\_loss$ function in the code.

#### 3.2.3 Training Optimizers

I used adam optimizer for both refiner and discriminator networks. The optimizer functions are implemented under $\_build\_optim$ function in the code.

#### 3.2.4 Training Phase

*Pre-training*: Before doing the actual training, the refiner is trained for 1000 iterations by minimizing only the self regularization loss and discriminator is trained for 200 iterations by minimizing the complete discriminator adversarial loss. The refiner and discriminator networks are trained alternately by updating refiner parameters 5 times for each discriminator update.

*Actual Training*: In the actual training loop, the refiner and discriminator are trained alternately for 50 epochs. The refiner and discriminator networks are trained on a batch of images at time. Here I used a batch size of 32. For the discriminator, while training on a batch of refined images, half of the batch contains images from the current discriminator and other half contains images from the history buffer. Both the refiner and discriminator are trained using gradient descent optimization alternately. The parameters for the discriminator are updated ten times for each refiner update. The training loop is implemented under $\_build\_train$ function.

### 3.3. Results

I have trained simulated GAN network with different optimizers and hyperparameters. I have found that synthetic images look more real with Adam optimizer, learning rate (lr)= 1e-3, $\beta 1 = 0.5$, regularization parameter $(\lambda) = 0.01$ as hyper parameter settings. The training losses for both discriminator and refiner are shown in figure. 2. The self regularization loss graph is shown in the figure 1

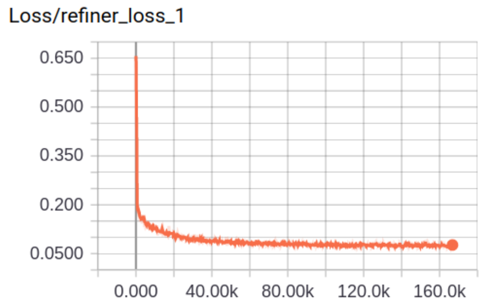

Figure 1. Self regularization loss function graph for refiner

The outputs of the refiner are taken out at intermediate stages in training and are shown in figures 3, 4, 5, 6, 7. In all the figures, left image contains a batch of synthetic eye images and the right image contains the corresponding refined images.

## 4. Discussion and future directions

I observed that during the pre-training phase, as the optimization was done by minimizing only the self-regularization loss for the refiner, the refined image is equivalent to the synthetic image and refiner loss went to zero for the first 1000 iterations of training. Once the actual training started, the refiner loss and discriminator loss increased and started fluctuating. The refiner network learned the distribution of real data and started applying real-world noise to the synthetic images. As the iterations go on increasing, the refined images look more realistic. Interestingly, the refiner network tries to give complete natural image even for cropped synthetic images, which can be observed in fig.. The refined images in the same figure look more natural and realistic. Interestingly, the refiner models eyebrows and hair around eyes even though they are missing in synthetic images.
From this project, I realized that it is hard to train GANs and requires a lot of time in choosing the best hyperparameters that give better results. In future, this project can be extended to make synthetic animated human images to real humans, which can be applied in video games to make them more interesting.

## References

[1] K. Mader. Eye Gaze, Simulated and real datasets of eyes looking in different directions. `https://www.kaggle.com/kmader/simgan-notebook/data`, 2017. [Online; accessed 11-April-2018].

[2] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *CoRR*, abs/1612.07828, 2016.

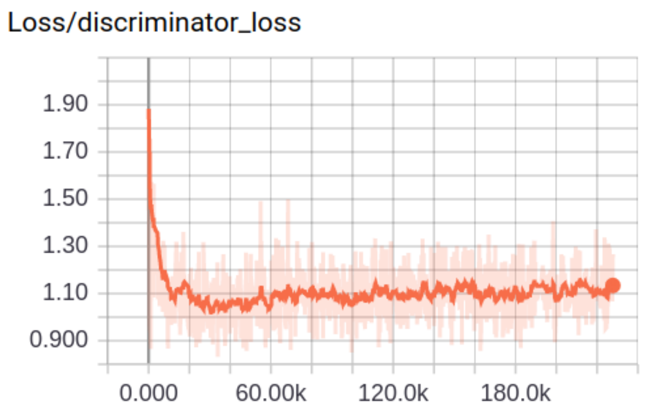

Figure 2. Loss functions for both discriminator and refiner

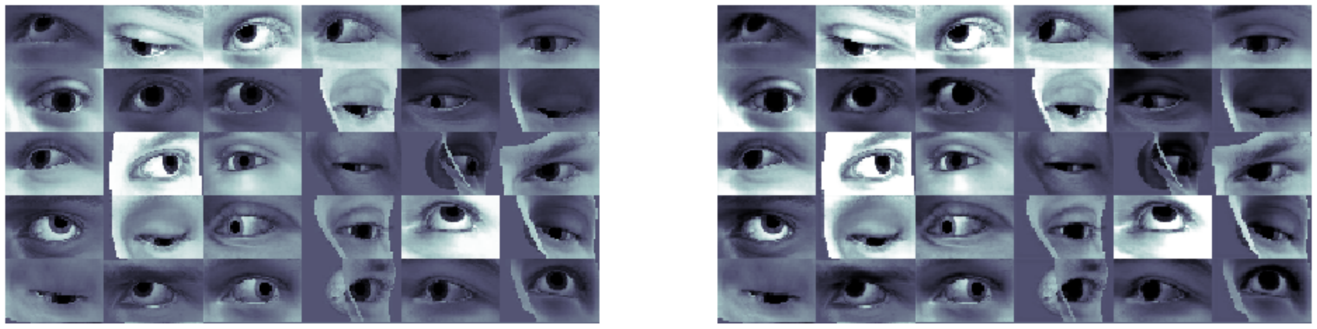

Figure 3. Synthetic images (left), Refined images (right), Hyper parameter settings $lr = 1e-3, \beta 1 = 0.5, \lambda = 1e-2, iteration = 0$

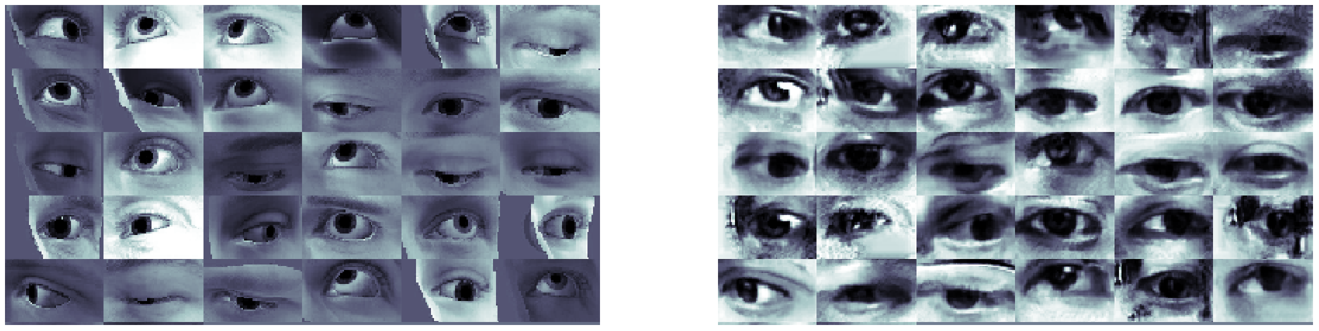

Figure 4. Synthetic images (left), Refined images (right), Hyper parameter settings $lr = 1e{-}3, \beta 1 = 0.5, \lambda = 1e{-}2, iteration = 5,000$

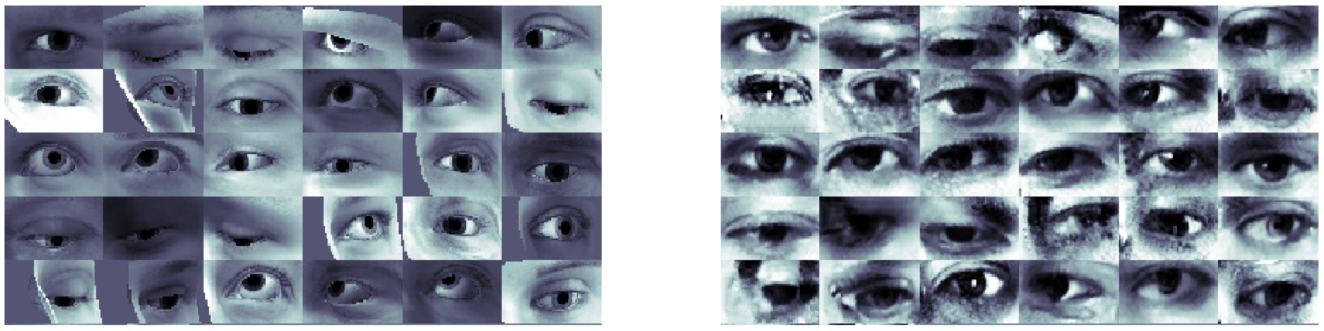

Figure 5. Synthetic images (left), Refined images (right), Hyper parameter settings $lr = 1e{-}3, \beta 1 = 0.5, \lambda = 1e{-}2, iteration = 10,000$

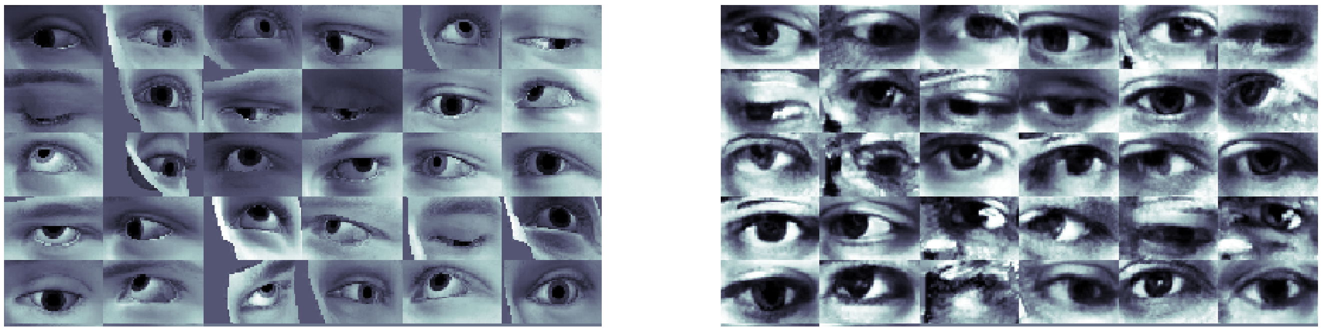

Figure 6. Synthetic images (left), Refined images (right), Hyper parameter settings $lr = 1e{-}3, \beta 1 = 0.5, \lambda = 1e{-}2, iteration = 20,000$

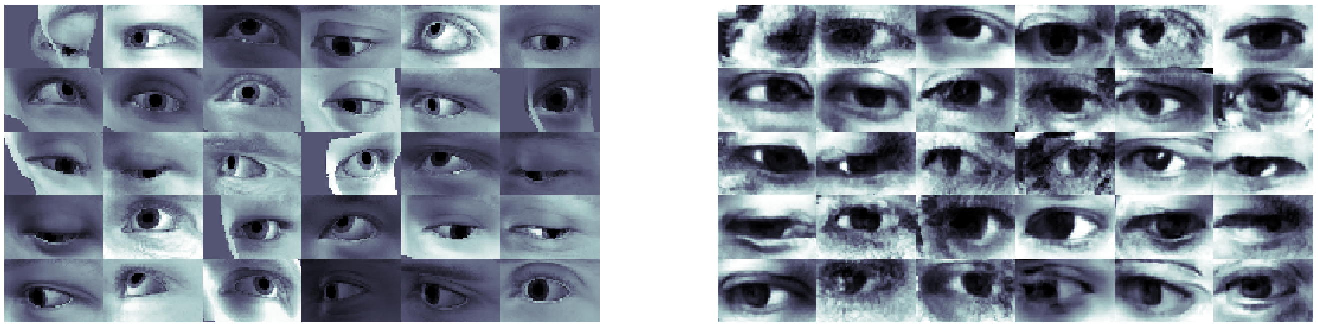

Figure 7. Synthetic images (left), Refined images (right), Hyper parameter settings $lr = 1e{-}3, \beta 1 = 0.5, \lambda = 1e{-}2, iteration = 25,000$