# Unpaired Image Translation with CycleGAN

Avinash M S
PES12017002101348
*Department of CSE*
*PES University*
*Bengaluru, India*

Deepika Karanji
PES1201700103
*Department of CSE*
*PES University*
*Bengaluru, India*

S Sriya
PES1201700048
*Department of CSE*
*PES University*
*Bengaluru, India*

***Abstract*— Image to Image translation involves generating a new synthetic version of a given image with some modifications. Obtaining and constructing paired image datasets is expensive and sometimes impossible. This paper implements Cycle GAN to achieve image translation from Apple to Orange and Horse to Zebra and vice versa. [1]**

## I. INTRODUCTION

GANs belong to the set of algorithms named generative models. These algorithms belong to the field of unsupervised learning, a subset of ML which aims to study algorithms that learn the underlying structure of the given data, without specifying a target value.Generative Adversarial Networks are composed of two models:

The first model is called a Generator and it aims to generate new data similar to the expected one. The Generator could be assimilated to a human art forger, which creates fake works of art.The second model is named the Discriminator. This model's goal is to recognize if an input data is '*real'* — belongs to the original dataset — or if it is '*fake'* — generated by a forger. In this scenario, a Discriminator is analogous to the police (or an art expert), which tries to detect artworks as truthful or fraud. Both models are trained such that the generator is updated to better fool the discriminator and the discriminator is updated to better detect generated images.

The CycleGAN is an extension of the GAN architecture that involves the simultaneous training of two generator models and two discriminator models. One generator takes images from the first domain as input and outputs images for the second domain, and the other generator takes images from the second domain as input and generates images for the first domain. Discriminator models are then used to determine how plausible the generated images are and update the generator models accordingly. Image processing is visually appealing and the results can be verified easily by visual inspection. The image translation problem is fascinating, because of the plethora of applications it offers to us.

Having studied GANs, we wanted to explore CycleGANs to achieve the above results.The CycleGAN encourages cycle consistency by adding an additional loss to measure the difference between the generated output of the second generator and the original image, and the reverse. This acts as a regularization of the generator models, guiding the image generation process in the new domain toward image translation.

## II. IMPORTANCE OF CYCLE GAN

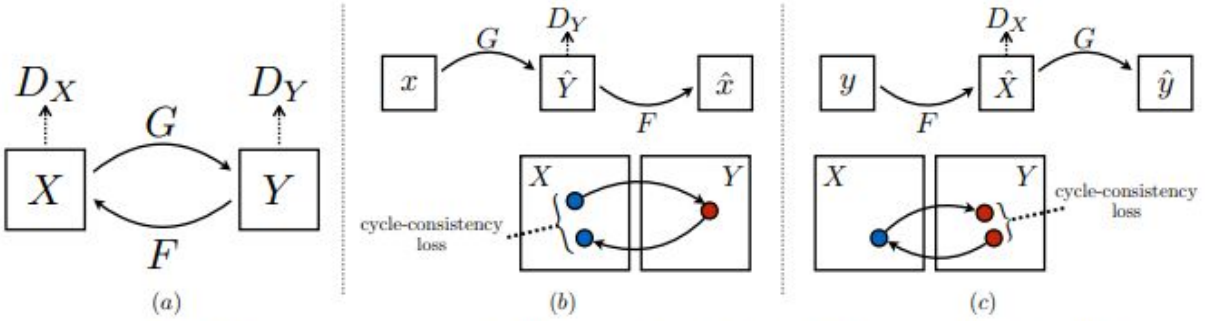
Training a model for image-to-image translation typically requires a large dataset of paired examples.These datasets can be difficult and expensive to prepare, and in some cases impossible, such as photographs of paintings by long dead artists. The CycleGAN is a technique that involves the automatic training of image-to-image translation models without paired examples. The models are trained in an unsupervised manner using a collection of images from the source and target domain that do not need to be related in any way. Other Applications of CycleGANs include:

- Generating a realistic rendering of what a building would look like based on its blueprints.
- Rendering a realistic representation of how a suspect's face would look based on a police sketch.
- Creating an image of how a location would look in each different season.
- Enhancing aspects of photos to make them look more professional.
- Conversion of MRI Scans into CT Scans
- Photo generation from paintings

CycleGAN is extremely usable because it doesn't need paired data. It's often pretty difficult to get a large amount of accurate paired data, so the ability to use unpaired data with high accuracy is good.

## III. SOLUTION ARCHITECTURE

FIG1: CYCLE GAN ARCHITECTURE [1] :




(a) Our model contains two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$, and associated adversarial discriminators $D_Y$ and $D_X$. $D_Y$ encourages $G$ to translate $X$ into outputs indistinguishable from domain $Y$, and vice versa for $D_X$ and $F$. To further regularize the mappings, we introduce two *cycle consistency losses* that capture the intuition that if we translate from one domain to the other and back again we should arrive at where we started: (b) forward cycle-consistency loss: $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$, and (c) backward cycle-consistency loss: $y \rightarrow F(y) \rightarrow G(F(y)) \approx y$

FIG2: THE GENERATOR NETWORK ARCHITECTURE :

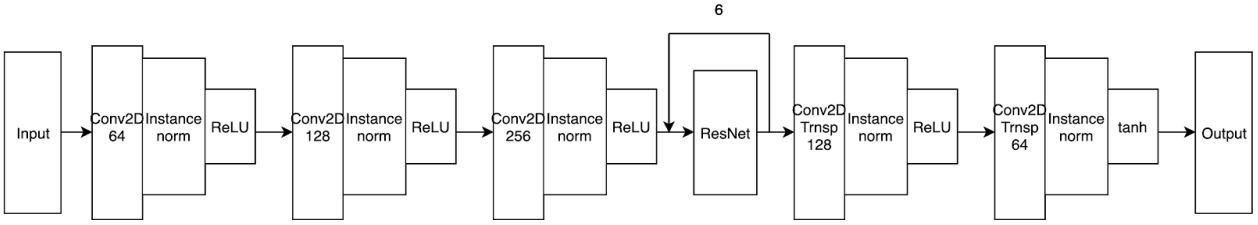
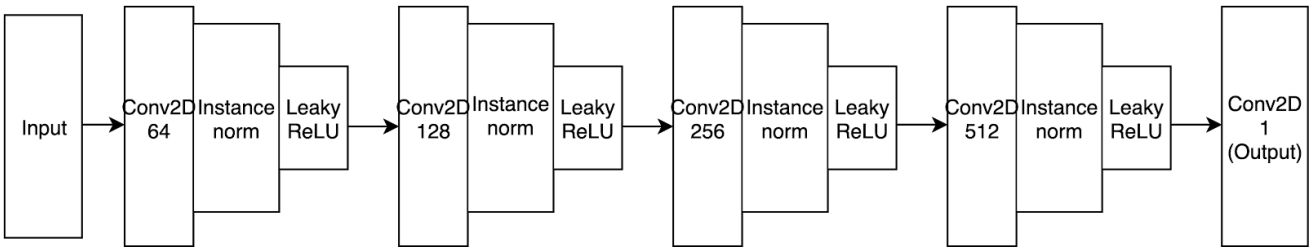



FIG3: THE DISCRIMINATOR NETWORK ARCHITECTURE :




The generator and discriminator models from GAN 1 (Domain X) as follows[2]:

**Generator (G)-**
Input: Takes photos from Domain X (Apple/Horse)
Output: Generates photos of Domain Y (Orange/Zebra).

**Discriminator (Dy)-**
Input: Takes photos from Domain Y (Orange/Zebra) and output generated from F.
Output: Likelihood that the image is from Domain Y.

The generator and discriminator models from GAN 2 (Domain Y) as follows:

**Generator (F)-**
Input: Takes photos from Domain Y (Orange/Zebra).
Output: Generates photos of Domain X (Apple/Horse) .

**Discriminator (Dy)-**
Input: Takes photos from Domain X (Apple/Horse) and output generated from G.
Output: Likelihood that the image is from Domain X.

**Generator Layers:**

- Encoder - 3 Convolution Layers with Relu as the activation function.
  - 7x7 with 1 stride and 32 filters
  - 3x3 with 2 stride and 64 filters
  - 3x3 with 2 stride 128 filters along
- Transformer - 6 resnet layers (residual block) each of which consists of 2 Convolution Layers (3x3 convolution) with 128 filters and 2 strides and relu as the activation function .
- Decoder - 2 deconvolution layers fractionally strided with stride ½ with relu activation function
  - 3x3 with 64 filters
  - 3x3 with 32 filters

  1 conv - tanh 7x7 1 stride 3 filters

**Discriminator Layers:**

- 4 Convolution layers 4x4 with 2 strides each with a leaky relu activation function
  - 64 filters
  - 128 filters
  - 256 filters
  - 512 filters
- 1 last convolution layer 4x4 with 1 stride and 512 filters

## IV. CODE MODULES

**Build.py -** Used to read images from the input directory and then shuffle them in order to guarantee random ordering of images with respect to labels in the TFRecord files. Also used to write data to tfrecords.

**Reader.py -** Converts image into a 4D tensor [batch_size, image_width, image_height, image_depth] after resizing.

**Generator.py -** Builds the Encoder, Transformer and Decoder Layers. We define the layers as described in the architecture

**Discriminator.py -** Builds the Discriminator Layers(2 Convolutional and 1 De-convolution layers)

**Model.py -** Defines the optimizer, discriminator loss, generator loss and cycle consistency loss functions.

**Main.py -** Main module used to construct the tensorflow graph and train the model. Also performs checkpointing of the model for every 20 steps in order to save and backup progress on Google Drive. Training can be resumed from the last saved checkpoint. Training stops when the results are satisfactory.

## V. ALGORITHM

*Algorithm CycleGAN[4]:*

1. Initialise generator G and a discriminator Dx for the Domain X and a generator F and a discriminator Dy for the Domain Y
2. Fake_Y = G(X), Fake_X = F(Y)
3. Calculate Adversarial Generator Loss

   generator_loss = [ D(fake_y) - real_label ]$^2$
4. Calculate Discriminator Loss

   real_error = [ D(y) - real_label ]$^2$
   fake_error = [ D(fake_y) - real_label ]$^2$
   discriminative_loss = (real_error + fake_error)/2
5. Calculate Forward and Backward Cycle Consistency loss

   forward_loss = reduce_mean( | F(G(x)) - x| )
   backward_loss = reduce_mean( | G(F(y)) - y| )
   cycle_loss = lambda1 * forward_loss + lambda2 * backward_loss
6. Calculate overall loss

   overall_loss = generator_loss + cycle_loss
7. Call the optimizer function (Adam Optimizer which takes in loss and variables as the parameters and returns the learning step which is used to update the weights)
8. 1 epoch = 100 steps. Steps get incremented each time the train function is called. Training is stopped when visual results are satisfactory.

## VI. MATH

Two mappings (GANs) G : X -> Y and F : Y -> X exist. The model needs to learn these mappings. F = inverse(G). Corresponding Adversarial discriminators are Dx and Dy as described in the Solution Architecture.

**Adversarial loss:**

$G^*, D_y^* = \arg\min_G \max_{Dy} l(G, D_y)$, where

$$\ell(\theta_f) = \sum_{\substack{n:t_{n1}=1 \\ x_n \sim P(X)}} \log D_X(x_n) + \sum_{\substack{n:t_{n0}=1 \\ y_n \sim P(Y)}} \log 1 - D_X(F(y_n))$$

$F^*, D_x^* = \arg\min_F \max_{Dx} l(G, D_x)$, where

$$\ell(\theta_g) = \sum_{\substack{n:t_{n1}=1 \\ y_n \sim P(Y)}} \log D_Y(y_n) + \sum_{\substack{n:t_{n0}=1 \\ x_n \sim P(X)}} \log 1 - D_Y(G(x_n))$$

i.e, $\mathcal{L}_{adversarial} = \ell_{X \to Y}(G, D_Y) + \ell_{Y \to X}(F, D_X)$

**Cycle Consistency loss:**

Forward Loss = X - F(G(X))

Backward Loss = Y - G(F(Y))

$$\mathcal{L}_{cycle\_con} = \mathbb{E}_{x\sim P(X)}\left[\|x - F(G(x))\|_1\right] + \mathbb{E}_{y\sim P(Y)}\left[\|y - G(F(y))\|_1\right]$$

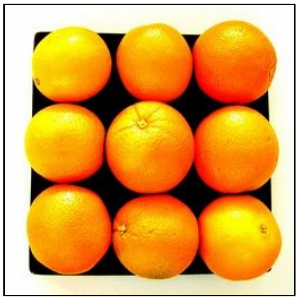
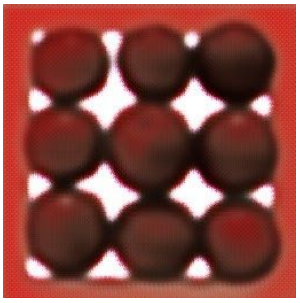
**Overall Loss:**

$$\mathcal{L}_{overall} = \mathcal{L}_{adversarial} + \lambda\,\mathcal{L}_{cycle\_con}$$

## VII. RESULTS

Apple to Orange cycle gan results after 80 epochs:


Input Output

Input Output

Reference [1] mentions that the training time needed to achieve good results is 1000 Epochs where 1 epoch = 10 steps.

For the 80 epochs, we believe that our model has done a good job in attempting to convert apples to oranges and vice versa.

## VIII. CONSTRAINTS, ASSUMPTIONS AND DEPENDENCIES

**Constraints:**

The training time for computation depends on the complexity of the input images and the required output. Apple to Orange took lesser time to achieve better results, while it took more time for Horse to Zebra conversion.

**Assumptions:**

Dataset we choose is clean

The dataset contains separated images of the 2 domains. The pictures aren't mixed.

**Dependencies:**

The dependencies are Tensorflow 1.0.0 Python 3

## IX. FUTURE WORK

Currently, the model training is stopped manually, when the image translation results are satisfactory(subjective). Automatically evaluating the results and stopping the training is an open area of research.

## REFERENCES


[1] https://arxiv.org/pdf/1703.10593.pdf

[2] https://machinelearningmastery.com/what-is-cyclegan/

[3] https://www.youtube.com/watch?v=NyAosnNQv_U&t=553s

[4] https://machinelearningmastery.com/what-is-cyclegan/