Cycle GAN

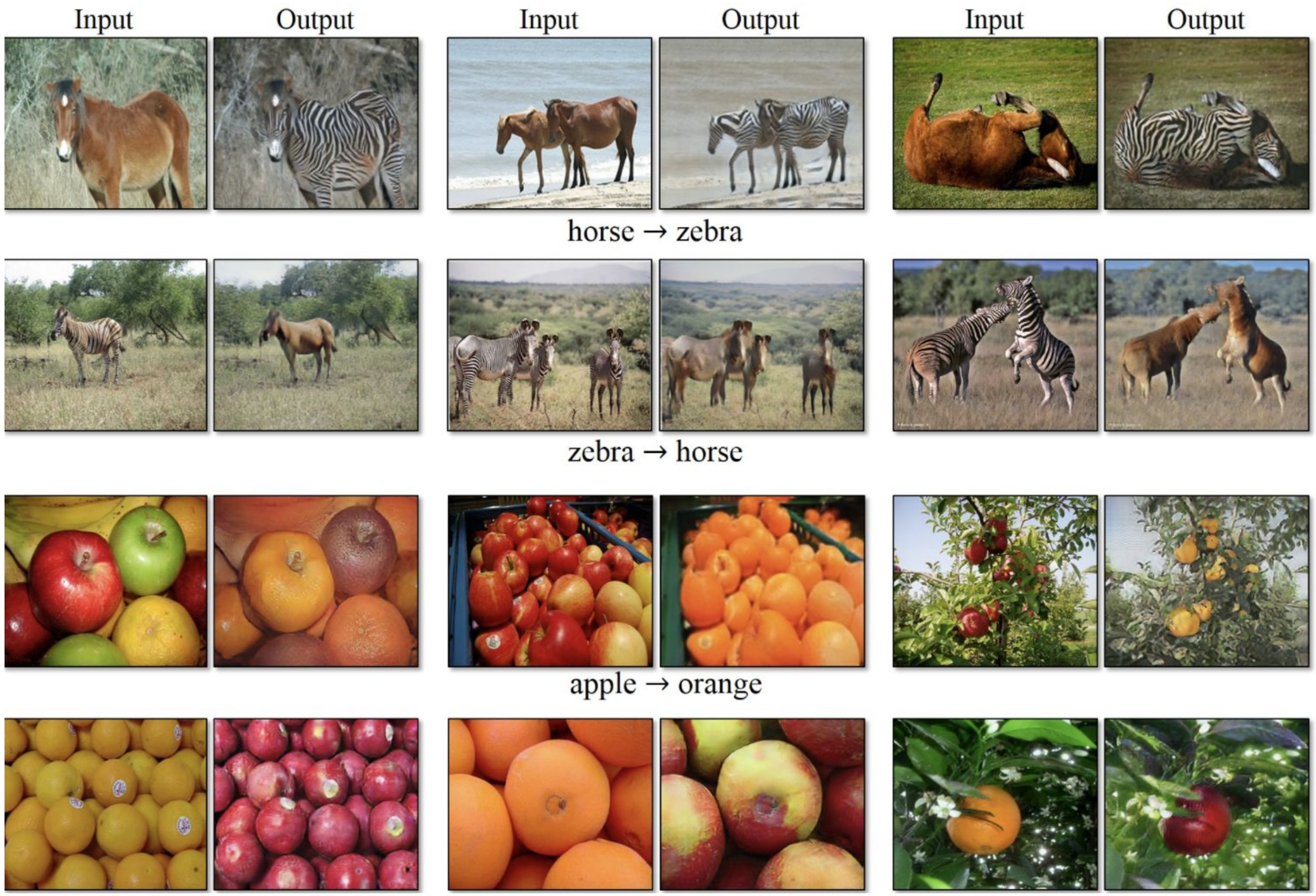
Presenter: Akilesh B

Introduction

- Cross domain image transfer.

- Recent methods such as Pix2Pix depend on the availability of training examples where the same data is available in both domains.

- Main advantage: no one-to-one mapping required between images in the input and target domain.


Input
Output
Input
Output
Input
Output
horse → zebra
zebra → horse
apple → orange

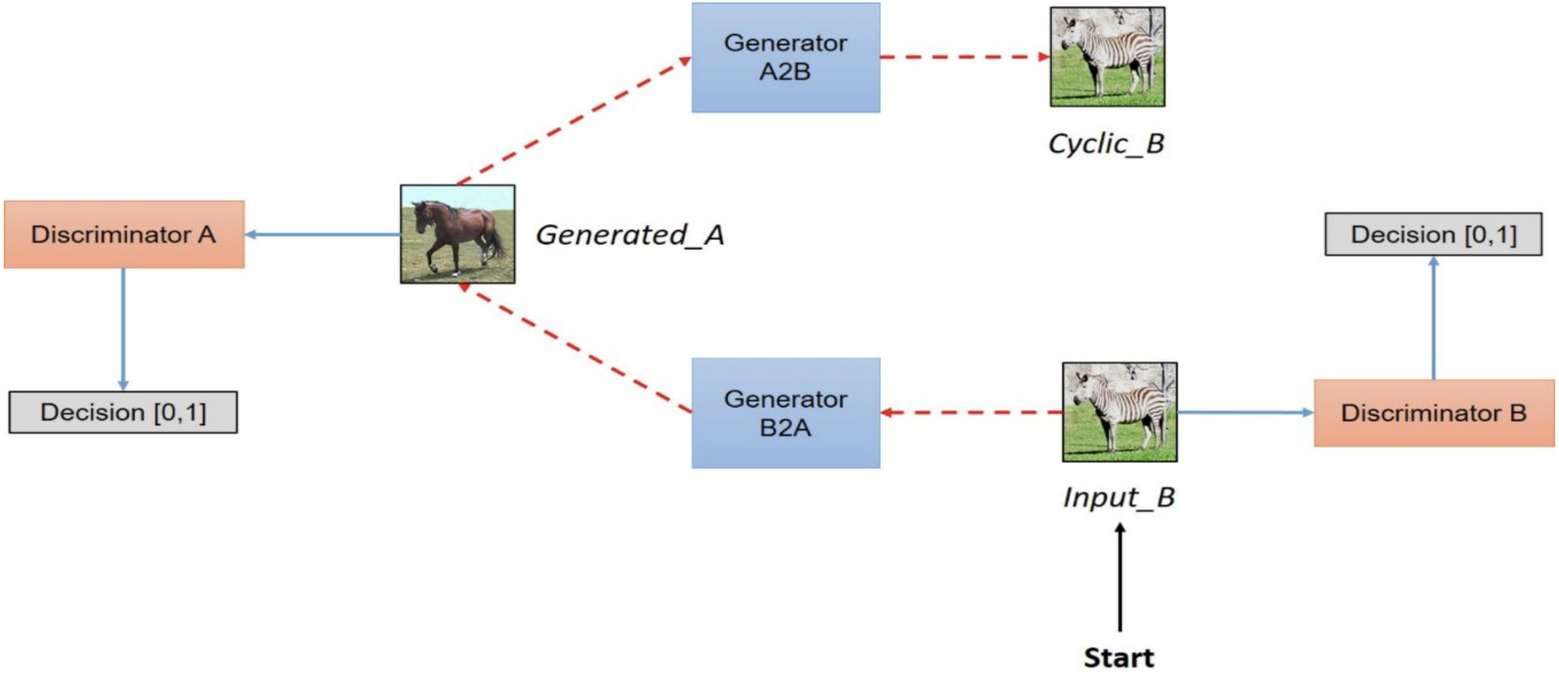
Cyclic consistency

- Two-step transformation of source domain image - first by trying to map it to target domain and then back to the original image.

- Mapping the image to target domain is done using a generator network and the quality of this generated image is improved by pitching the generator against a discriminator.

- G : X -> Y and F : Y -> X, F(G(X)) ~ X (and vice-versa).

Architecture



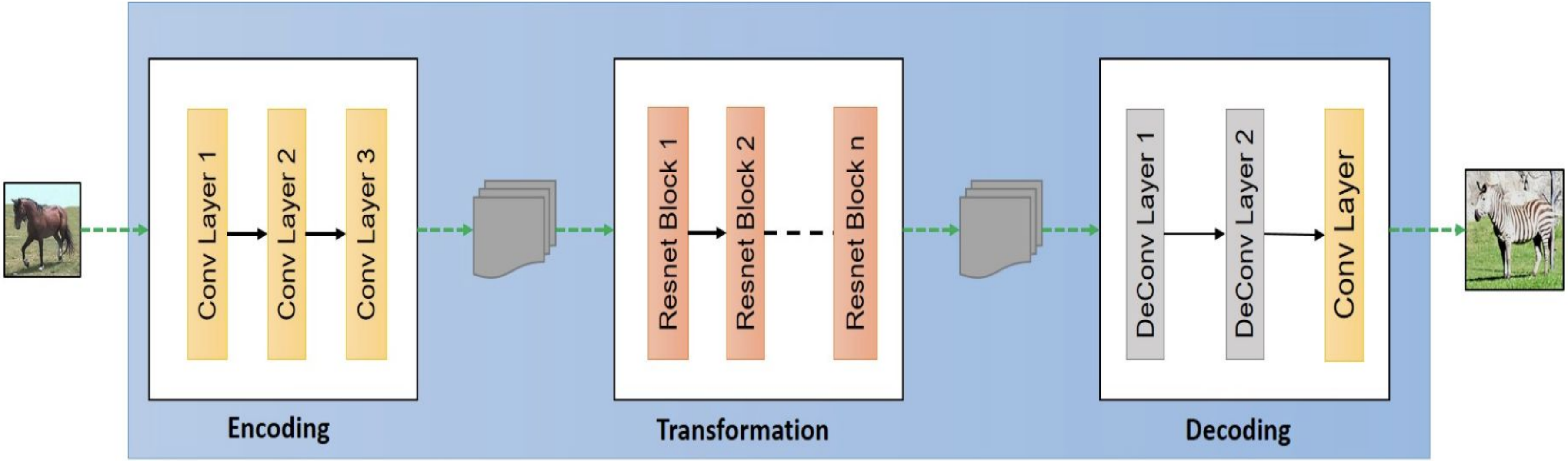
Explanation

- The model works by taking an input image from domain D_A which is fed to our first generator $Generator_{A \rightarrow B}$ whose job is to transform a given image from domain D_A to an image in target domain D_B.

- This new generated image is then fed to another generator $Generator_{B \rightarrow A}$ which converts it back into an image, $Cyclic_A$, from our original domain D_A.

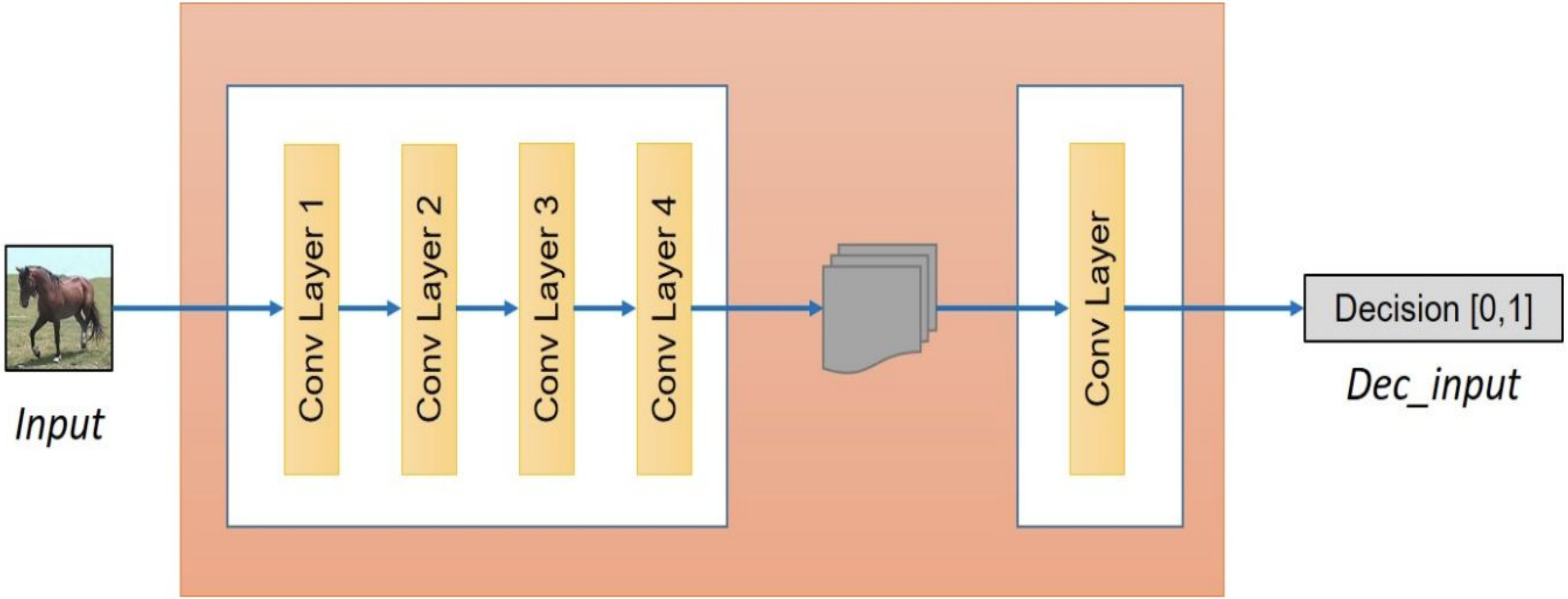
- This output image must be close to original input image to define a meaningful mapping that is absent in unpaired dataset.

Generator



Conv Layer 1
Conv Layer 2
Conv Layer 3
Encoding
Resnet Block 1
Resnet Block 2
Resnet Block n
Transformation
DeConv Layer 1
DeConv Layer 2
Conv Layer
Decoding

Discriminator



Model

```
gen_B = build_generator(input_A, name="generator_AtoB")
gen_A = build_generator(input_B, name="generator_BtoA")
dec_A = build_discriminator(input_A, name="discriminator_A")
dec_B = build_discriminator(input_B, name="discriminator_B")

dec_gen_A = build_discriminator(gen_A, "discriminator_A")
dec_gen_B = build_discriminator(gen_B, "discriminator_B")
cyc_A = build_generator(gen_B, "generator_BtoA")
cyc_B = build_generator(gen_A, "generator_AtoB")
```

Discriminator loss

Component 1

```
D_A_loss_1 = tf.reduce_mean(tf.squared_difference(dec_A,1))
D_B_loss_1 = tf.reduce_mean(tf.squared_difference(dec_B,1))
```

Component 2

```
D_A_loss_2 = tf.reduce_mean(tf.square(dec_gen_A))
D_B_loss_2 = tf.reduce_mean(tf.square(dec_gen_B))
```

Overall

```
D_A_loss = (D_A_loss_1 + D_A_loss_2)/2
D_B_loss = (D_B_loss_1 + D_B_loss_2)/2
```

Generator loss

Component 1

```
disc_loss_A = tf.reduce_mean(tf.squared_difference(dec_gen_A,1))
```

```
disc_loss_B = tf.reduce_mean(tf.squared_difference(dec_gen_B,1))
```

Component 2

```
cyc_loss = tf.reduce_mean(tf.abs(input_A-cyc_A)) + tf.reduce_mean(tf.abs(input_B-cyc_B))
```

Overall

```
g_loss_A = disc_loss_B + 10*cyc_loss
```

```
g_loss_B = disc_loss_A + 10*cyc_loss
```

Misc points

- Least Square GAN is used instead of original log likelihood (link).
- Discriminator uses a Patch GAN (random 70x70 crops) and scores are averaged. (I haven't used this!)