



Unpaired Image-to Image Translation using Cycle-Consistent Adversarial Networks

Deep Learning Project Update 1

By
Priya & Meghana
12/05/2022

Table of Contents

Project Description.....	3
Dataset.....	3
Method.....	4
Network Architecture.....	4
Model from the Paper.....	4
Generator.....	4
Discriminator.....	4
Our Implementation - Mini Model.....	5
Network Training.....	6
Model parameters.....	6
Experiments.....	7
Training results.....	8
Banana to Cucumber training.....	9
Experiment 1 Training log.....	9
Experiment 4 Training log.....	10
Experiment 5 Training log.....	10
Cucumber to Banana training.....	12
Experiment 2 Training log.....	12
Experiment 3 Training log.....	12
Training Log.....	13
Model Testing.....	13
Results Files.....	14
References.....	16



Project Description

Image-to-image translation is a class of vision and graphics problems where the goal is to learn the mapping between an input image and an output image using a training set of aligned image pairs. In this project, we shall learn an approach for learning to translate an image from a source domain A to a target domain B in the absence of paired examples.

Reference Paper : <https://arxiv.org/pdf/1703.10593v7.pdf>

Dataset

Since we need unpaired images for training, we choose the following pairs in the form of input and target. The dataset is collected using a mobile phone camera. We collected images from various images with various backgrounds.

	Domain A	Domain B
Label	Banana	Cucumber
Number of Train Images	55	55
Number of Test Images	8	8
Reference Image		

Dataset constraints

1. We create two directories to host images from domain A /path/to/data/trainA and from domain B /path/to/data/trainB
2. No annotations are needed for this data
3. The images from Domain A not paired with Domain B, hence the number of images from Domain A are not equal to the images in Domain B.
4. No labels are needed for the train or test images
5. The dataset is unpaired, segregated into 2 folders
 1. For Training
 1. Train A – Train B

2. For Testing

1. Test A – Test B

6. In this project we collected only a single pair of images Domain A is banana and Domain B is cucumber.

Method

Network Architecture

Model from the Paper

In this project adopts the architecture for our generative networks from Johnson et al. who have shown impressive results for neural style transfer and super resolution. This network contains three convolutions, several residual blocks, two fractionally-strides convolutions with stride 1/2, and one convolution that maps features to RGB. We use 6 blocks for 128×128 images and 9 blocks for 256×256 and higher-resolution training images. For the discriminator networks we use 70×70 PatchGANs [22, 30, 29], which aim to classify whether 70×70 overlapping image patches are real or fake. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and can work on arbitrarily sized images in a fully convolutional fashion.

Generator

The current implementation has Our current implementation provides two types of generators:

U-Net: [unet_128] (for 128×128 input images) and [unet_256] (for 256×256 input images).

Resnet-based generator: [resnet_6 blocks] (with 6 Resnet blocks) and [resnet_9 blocks] (with 9 Resnet blocks) Resnet-based generator consists of several Resnet blocks between a few downsampling/upsampling operations. A resnet block is a conv block with skip connections. We construct a conv block with build_conv_block function, and implement skip connections in forward function.

We adapt Torch code from Justin Johnson's neural style transfer project (<https://github.com/jcjohnson/fast-neural-style>).

Discriminator

Our current implementation provides three types of discriminators:

[basic]: 'PatchGAN' classifier described in the original pix2pix paper. It can classify whether 70×70 overlapping patches are real or fake. Such a patch-level discriminator architecture has fewer parameters than a full-image discriminator and can work on arbitrarily-sized images in a fully convolutional fashion

[n_layers]: With this mode, you can specify the number of conv layers in the discriminator with the parameter <n_layers_D> (default=3 as used in [basic] (PatchGAN).)

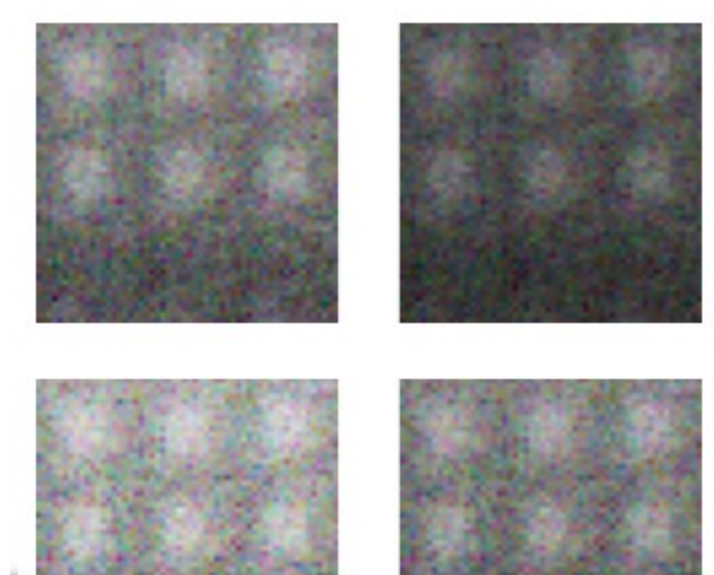
[pixel]: 1x1 PixelGAN discriminator can classify whether a pixel is real or not. It encourages greater color diversity but has no effect on spatial statistics. The discriminator has been initialized by <init_net>. It uses Leaky RELU for non-linearity.

Our Implementation - Mini Model

We designed a mini model with vanilla GAN architecture. The model has generator and discriminator architecture is as shown on the image below.

Mini Model Generator	Mini Model Discriminator
<pre>class Generator(nn.Module): def __init__(self,): super().__init__() self.layers = nn.Sequential(# First upsampling nn.Linear(NOISE_DIMENSION, 128, bias=False), nn.BatchNorm1d(128, 0.8), nn.LeakyReLU(0.25), # Second upsampling nn.Linear(128, 256, bias=False), nn.BatchNorm1d(256, 0.8), nn.LeakyReLU(0.25), # Third upsampling nn.Linear(256, 512, bias=False), nn.BatchNorm1d(512, 0.8), nn.LeakyReLU(0.25), # Final upsampling nn.Linear(512, GENERATOR_OUTPUT_IMAGE_SHAPE, bias=False), nn.Tanh()) def forward(self, x): return self.layers(x)</pre>	<pre>class Discriminator(nn.Module): def __init__(self): super().__init__() self.layers = nn.Sequential(nn.Linear(GENERATOR_OUTPUT_IMAGE_SHAPE, 1024), nn.LeakyReLU(0.25), nn.Linear(1024, 512), nn.LeakyReLU(0.25), nn.Linear(512, 256), nn.LeakyReLU(0.25), nn.Linear(256, 1), nn.Sigmoid()) def forward(self, x): return self.layers(x)</pre>
<pre>generator Generator((layers): Sequential((0): Linear(in_features=3136, out_features=128, bias=False) (1): BatchNorm1d(128, eps=0.8, momentum=0.1, affine=True, track_running_stats=True) (2): LeakyReLU(negative_slope=0.25) (3): Linear(in_features=128, out_features=256, bias=False) (4): BatchNorm1d(256, eps=0.8, momentum=0.1, affine=True, track_running_stats=True) (5): LeakyReLU(negative_slope=0.25) (6): Linear(in_features=256, out_features=512, bias=False) (7): BatchNorm1d(512, eps=0.8, momentum=0.1, affine=True, track_running_stats=True) (8): LeakyReLU(negative_slope=0.25) (9): Linear(in_features=512, out_features=9408, bias=False) (10): Tanh()))</pre>	<pre>discriminator Discriminator((layers): Sequential((0): Linear(in_features=9408, out_features=1024, bias=True) (1): LeakyReLU(negative_slope=0.25) (2): Linear(in_features=1024, out_features=512, bias=True) (3): LeakyReLU(negative_slope=0.25) (4): Linear(in_features=512, out_features=256, bias=True) (5): LeakyReLU(negative_slope=0.25) (6): Linear(in_features=256, out_features=1, bias=True) (7): Sigmoid()))</pre>

We trained the dataset in the mini vanillaGAN such that, the real images are bananas and fake images are cucumbers. We trained the model for 50 epochs. The sample images of generated images after training are below



Network Training

CycleGAN learns to map input A to Input B. CycleGAN is quite memory-intensive as four networks (two generators and two discriminators) need to be loaded on one GPU, so a large image cannot be entirely loaded. We used GPU training and batch-size 4. We were unable to process the training if the batch-size is more than 4 as there is no room for memory after loading the models.

To view training results and loss plots, run the following command in other terminal

python3 -m visdom.server and click the URL <http://localhost:8097>

To log training progress and test images to W&B dashboard, set the `--use_wandb` flag with train and test script

Train the model:

python3 train.py --dataroot ./datasets/banana2cucumber/ --name banana2cucumber3 --model cycle_gan --batch_size 4 --display_freq 10 --use_wandb --n_epochs 50

The trained model results are saved to “**checkpoints/banana2cucumber**”

Model parameters

```
----- Networks initialized -----  
[Network G_A] Total number of parameters : 11.378 M  
[Network G_B] Total number of parameters : 11.378 M  
[Network D_A] Total number of parameters : 2.765 M  
[Network D_B] Total number of parameters : 2.765 M  
-----
```

Experiments

Experiment ID	Hyper Parameters	Evaluation Metrics	
1	batch_size: 4 beta1 : 0.5 lr: 0.0002 n_epochs: 10 n_epochs_decay: 2 Training direction: AtoB init_type : Normal lr_policy : Linear	D_A	0.21
		D_B	0.21
		G_A	0.36
		G_B	0.41
		cycle_A	1.41
		cycle_B	1.29
		idt_A	0.6
		idt_B	0.68
2	batch_size: 4 beta1 : 0.5 lr: 0.0002 n_epochs: 10 n_epochs_decay: 2 Training direction: BtoA init_type : Normal lr_policy : Linear	D_A	0.21
		D_B	0.22
		G_A	0.43
		G_B	0.32
		cycle_A	1.14
		cycle_B	1.97
		idt_A	0.96
		idt_B	0.54
3	batch_size: 4 beta1 : 0.5 lr: 0.0001 n_epochs: 10 n_epochs_decay: 2 Training direction: BtoA init_type : xavier lr_policy : plateau	D_A	0.24
		D_B	0.2
		G_A	0.29
		G_B	0.29
		cycle_A	1.65
		cycle_B	2.96
		idt_A	1.36
		idt_B	0.77
4	batch_size: 4 beta1 : 0.5 lr: 0.0001 n_epochs: 10 n_epochs_decay: 2 Training direction: AtoB	D_A	0.25
		D_B	0.18
		G_A	0.27
		G_B	0.33

	init_type : xavier lr_policy : plateau	cycle_A	2.68
		cycle_B	1.7
		idt_A	0.84
		idt_B	1.3
5	batch_size: 4 beta1 : 0.5 lr: 0.0002 n_epochs: 10 n_epochs_decay: 2 Training direction: AtoB init_type : Normal lr_policy : Linear gan_mode: vanilla	D_A	0.65
		D_B	0.54
		G_A	1.13
		G_B	0.96
		cycle_A	2.19
		cycle_B	1.47
		idt_A	0.7
		idt_B	0.98
6	batch_size: 8 beta1 : 0.5 lr: 0.0002 n_epochs: 10 n_epochs_decay: 2 Training direction: BtoA init_type : Normal lr_policy : Linear gan_mode: vanilla	Batch size 8 gave out of memory exception	

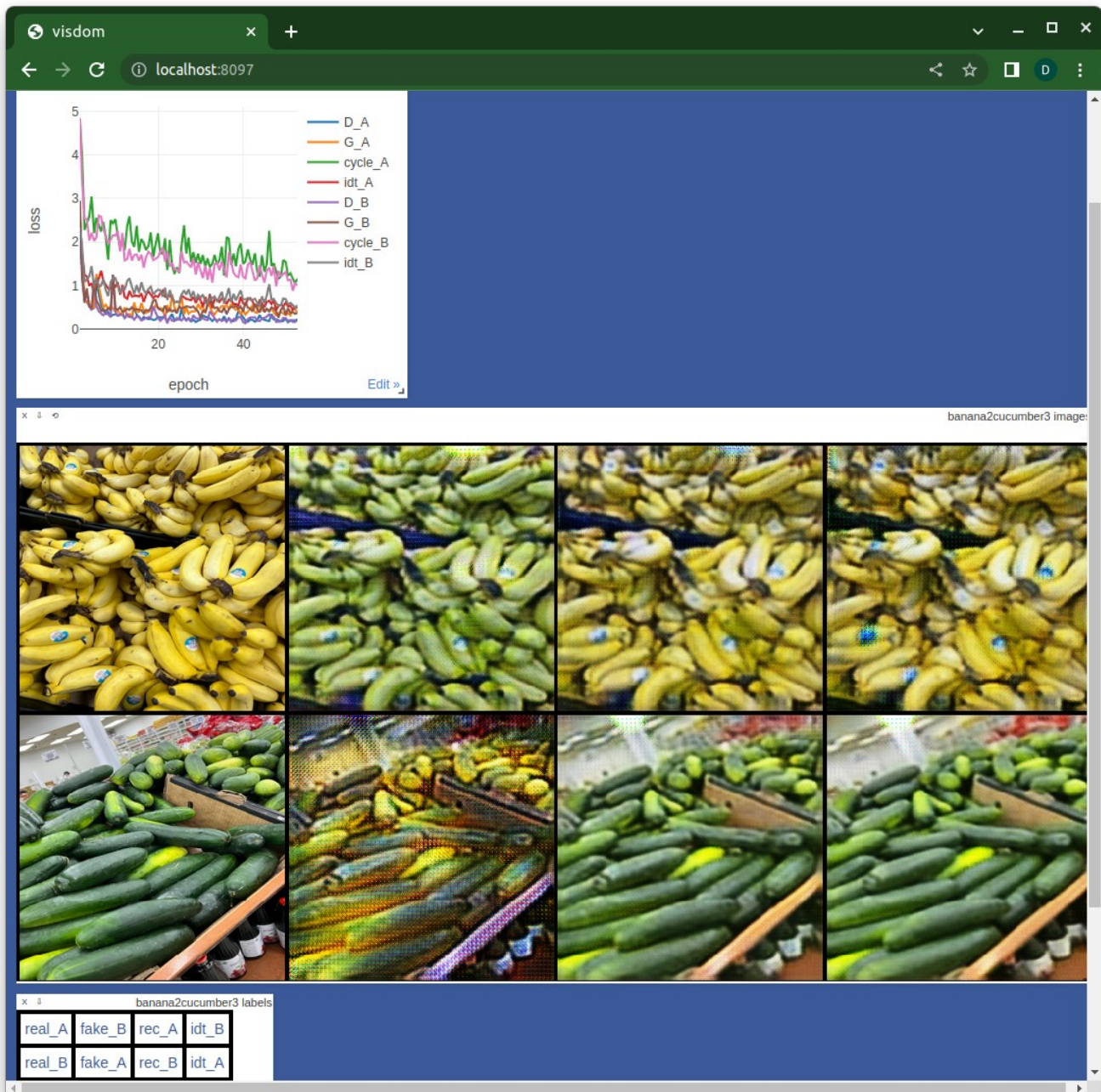
Training results

The training results are captured in wandb

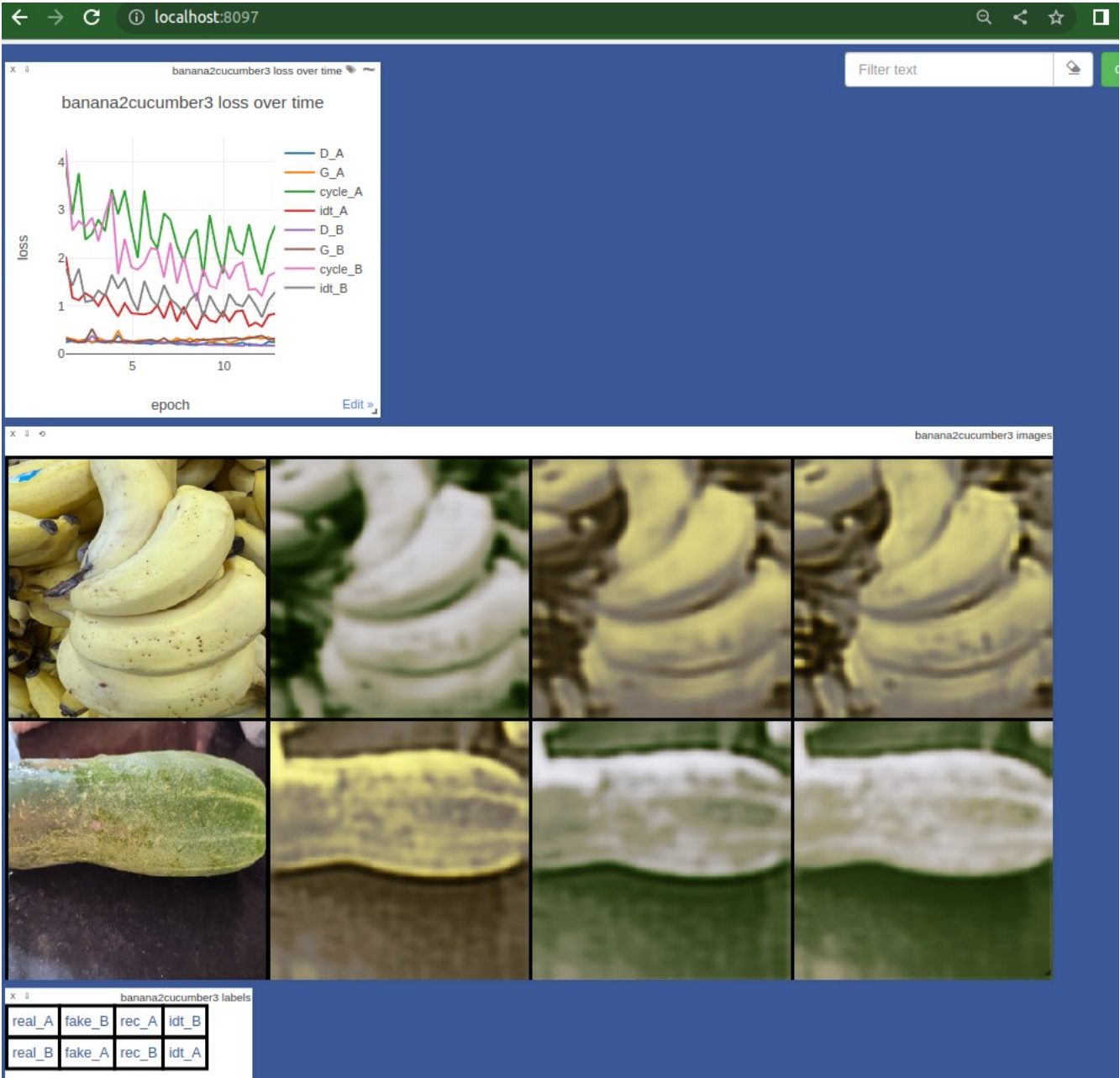
The training results of the model are visualized in the visdom browser as shown in the following figure

Banana to Cucumber training

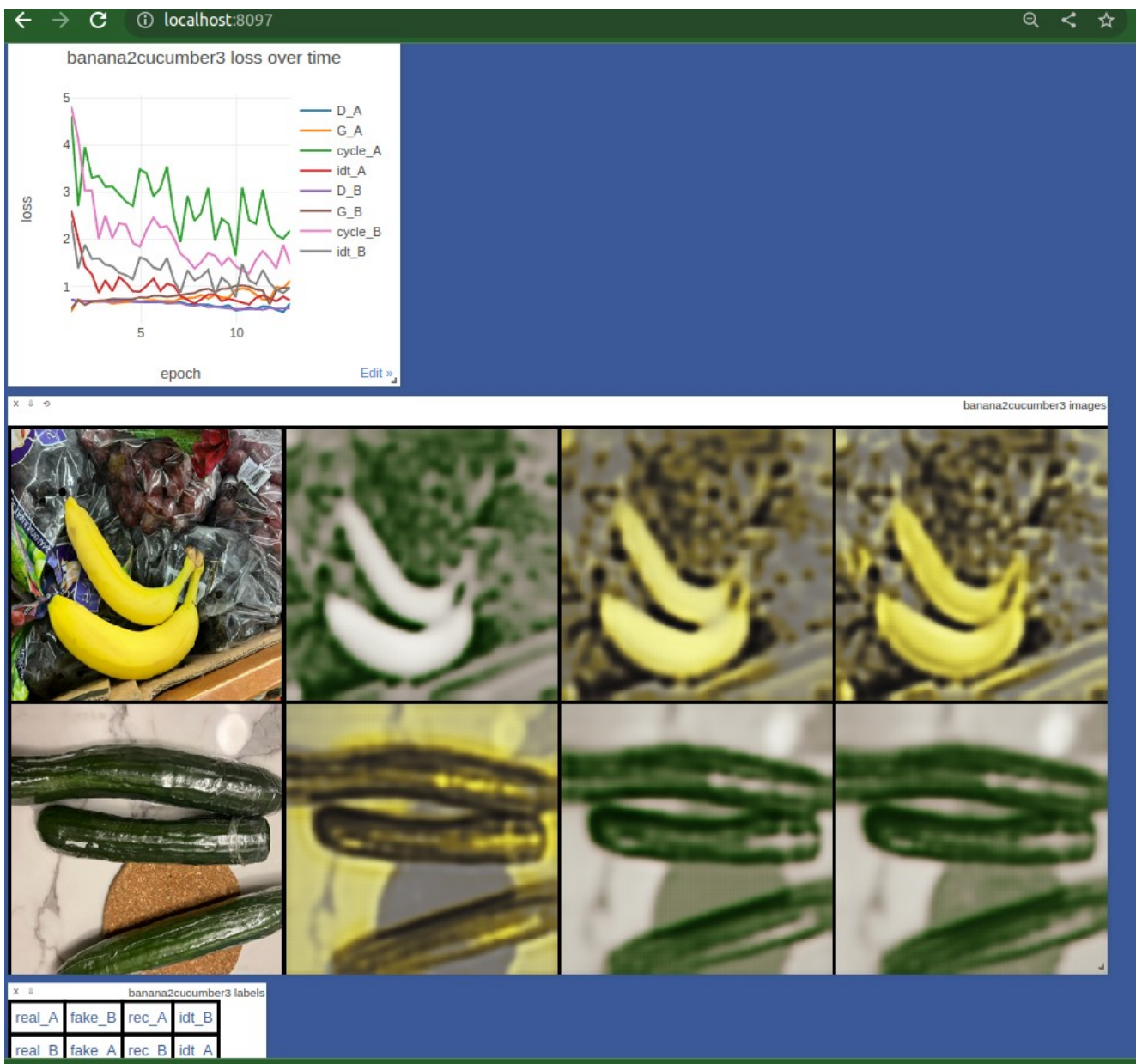
Experiment 1 Training log



Experiment 4 Training log

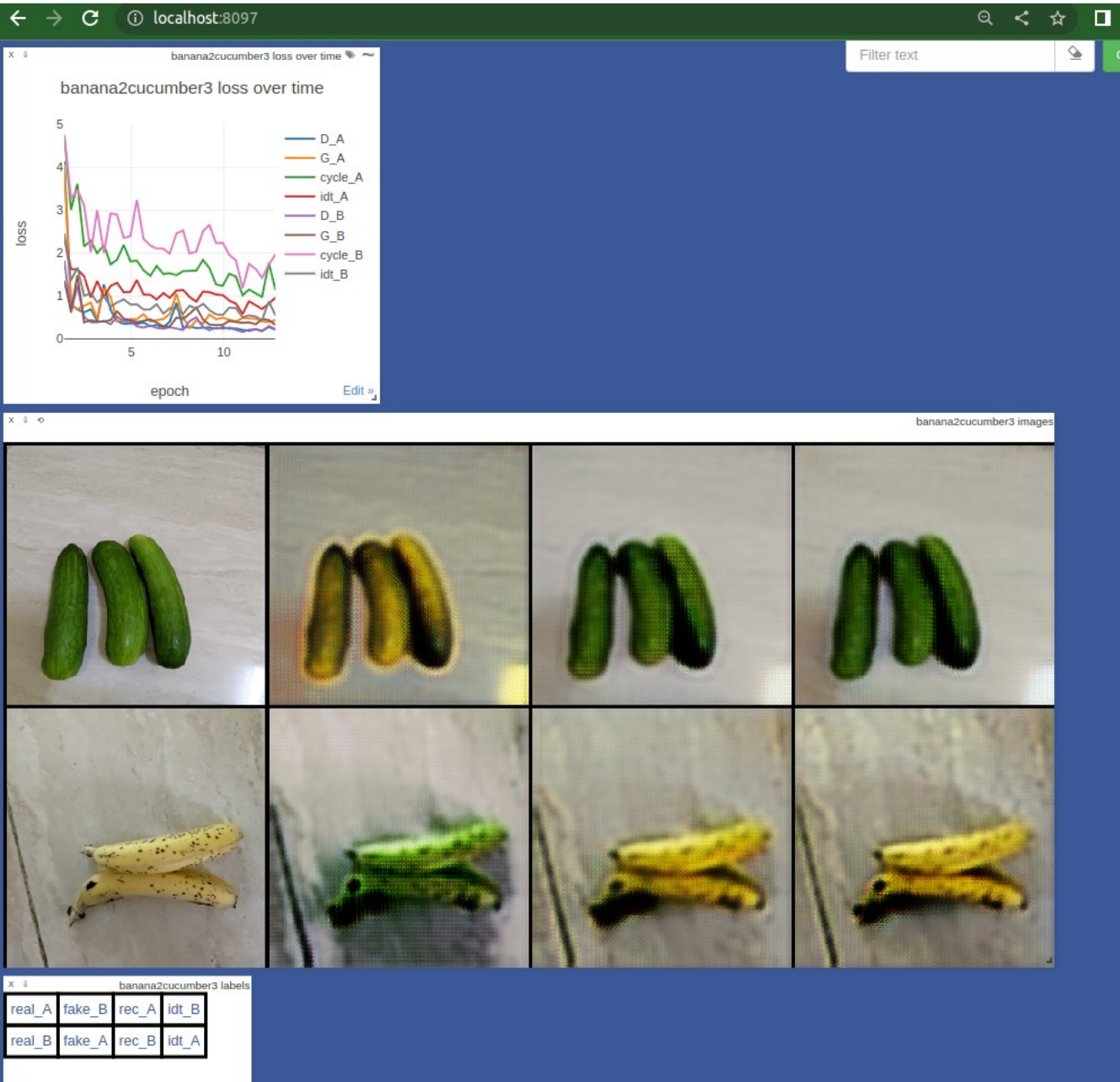


Experiment 5 Training log

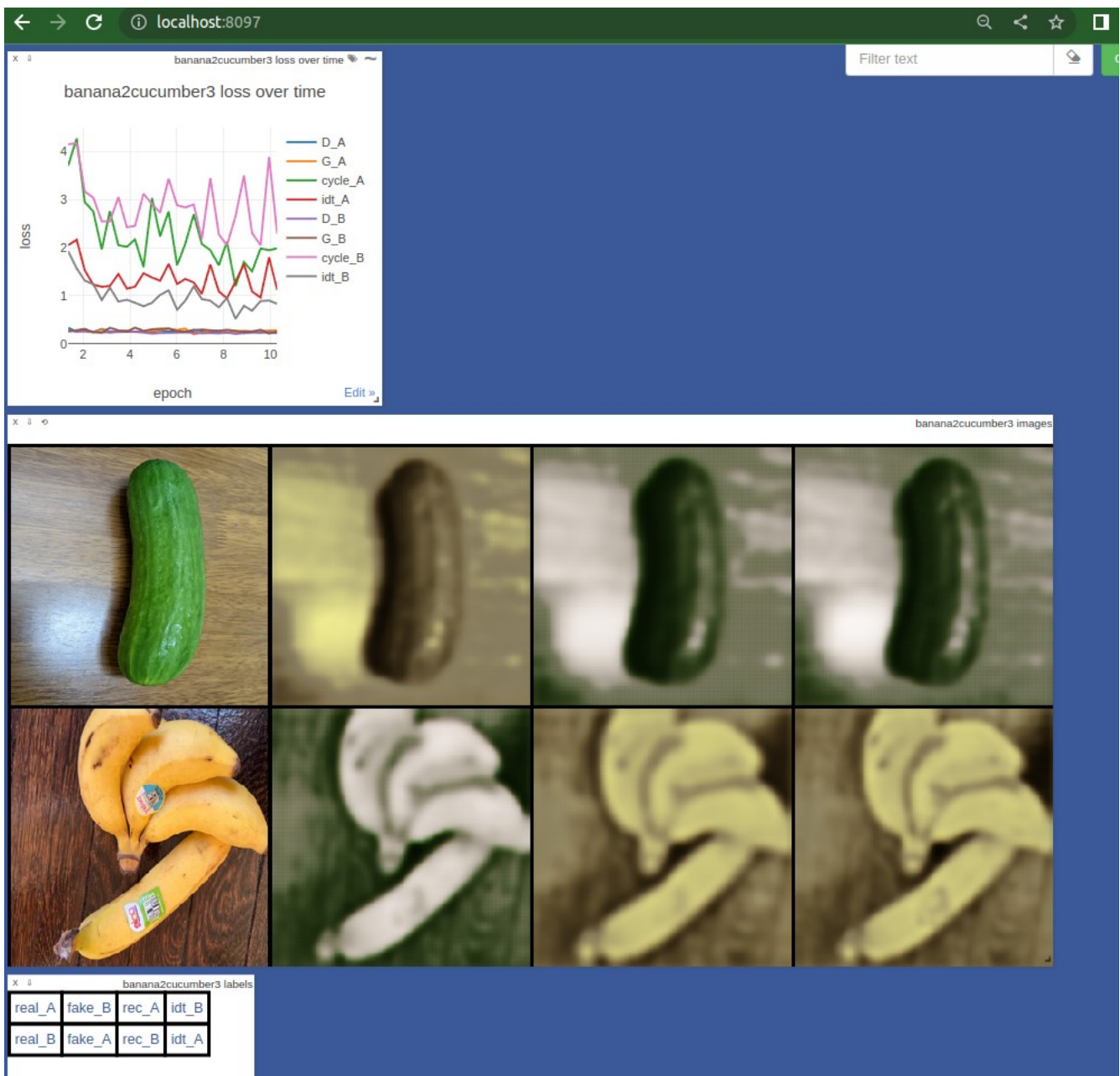


Cucumber to Banana training

Experiment 2 Training log



Experiment 3 Training log



Training Log

Note : The trained model weights are uploaded in google drive :

<https://drive.google.com/file/d/1X6TCDgzpEgmH9wn3F1dQ4CwmYVJkWMWN/view?usp=sharing>

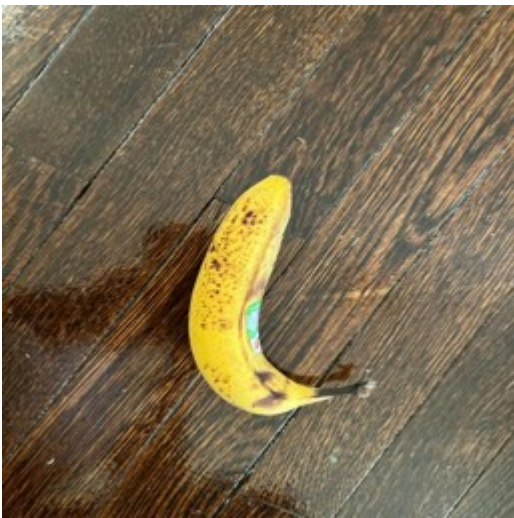

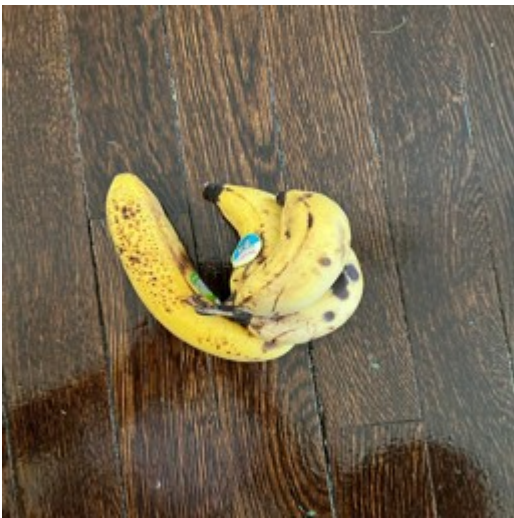

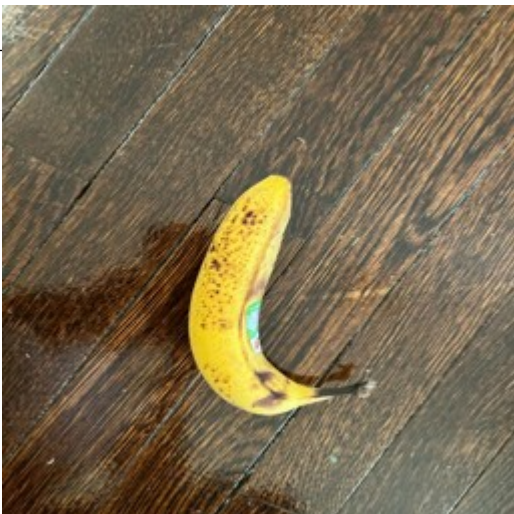

Model Testing

The model is tested on test images testA and testB


```
python3 test.py --dataroot ./datasets/banana2cucumber --name banana2cucumber3 --model cycle_gan
```

Results Files

We are unable to produce compelling results as the model is trained from the scratch using our dataset for only 50 epochs. The results are be saved in “/banana2cucumber3/test_latest/images/”

Test Images Domain A	Generated Images for Domain B
	
	
	



Test Images Domain B



Generated Images for Domain A



The results are viewed in “results/banana2cucumber3/test_latest/index.html”

References

1. <https://arxiv.org/pdf/1703.10593v7.pdf>
2. <https://www.kaggle.com/datasets/suyashdamle/cyclegan>
3. <https://www.kaggle.com/datasets/vikramtiwari/pix2pix-dataset>
4. <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>
5. <https://drive.google.com/file/d/1X6TCDgzpEgmH9wn3F1dQ4CwmYVJkWMWN/view?usp=sharing>