# City-GAN: a Conditional GAN Application

by Anas Hamed

# About the Paper

- City-GAN: Learning architectural styles using a custom Conditional GAN architecture[1]
- Maximilian Bachl & Daniel C. Ferreira
- Published 3 July 2019
- https://github.com/muxamilian/city-gan

# City-GAN Overview

Aims to

- Capture architectural features of certain cities

by

- Training on pictures of facades
- Generating new ones in a specific style

# Why this paper was chosen

- Builds upon DC-GANs[2] and Conditional GANs[3]
- Outlines the limitations and shortcomings of such methods
- Showcases recent applications and developments
- Introduces new ideas

# The dataset

- Google Street View images
- Four cities: Amsterdam, Manhattan, Paris, Vienna
- 1000 Images per city were chosen such that
  - Facades are fully visible
  - No anomalous structures (no construction sites or occlusions)

# Examples of anomalous samples (not used)


# Examples of valid samples

Amsterdam


Manhattan

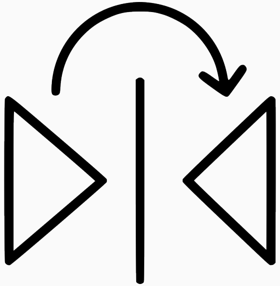

Paris

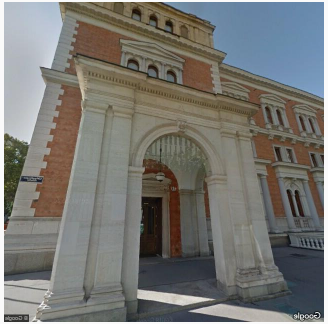

Vienna

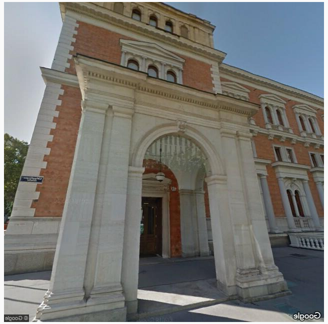
# Preprocessing & Augmentation

640x640 pixels

Stochastic flip





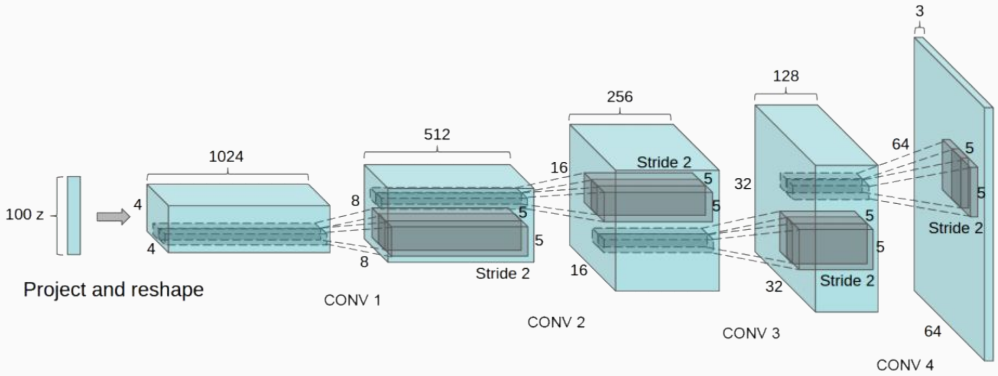


Random crop

- 64x64
- 128x128
- 256x256

# First Approach: Standard GAN

- Deep Convolutional GAN[2]
- No distinguishing factors between different cities
- Discriminator: continuously convolute input image
- Generator: continuously deconvolute noise vector



# DC-GAN: Results

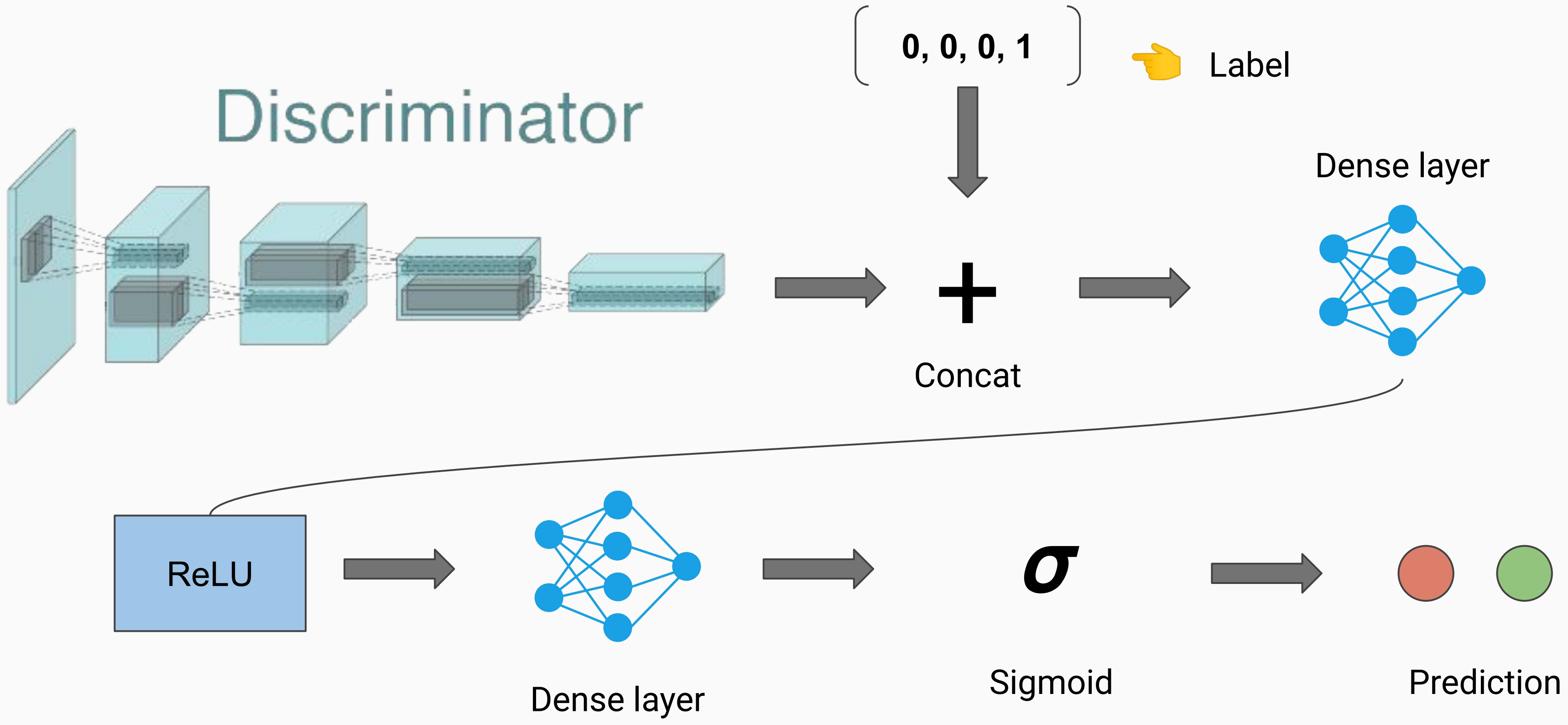
- Mode collapse
- Images too blurry
- Facades have no specific style
- Architecture does not adequately represent features

# DC-GAN: Results

# Second Approach: Conditional GAN

- Motive: allow Gan to focus on one style at a time
- Generator
  - Same architecture as DC-GAN
  - Concatenate label with latent variable
- Discriminator
  - Concatenate labels with output
  - Add dense layers

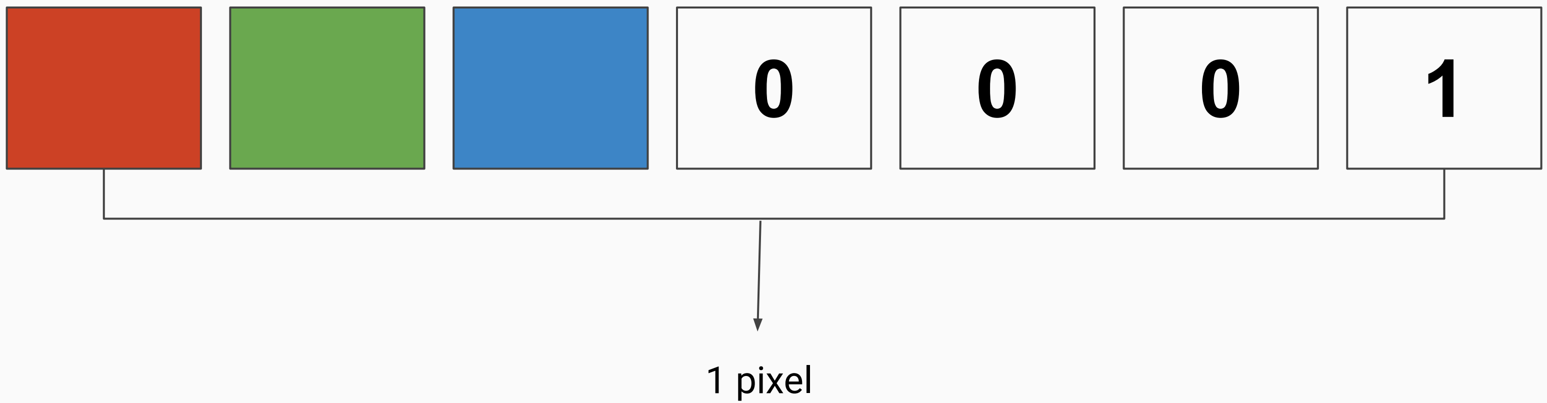
# C-GAN Discriminator Architecture



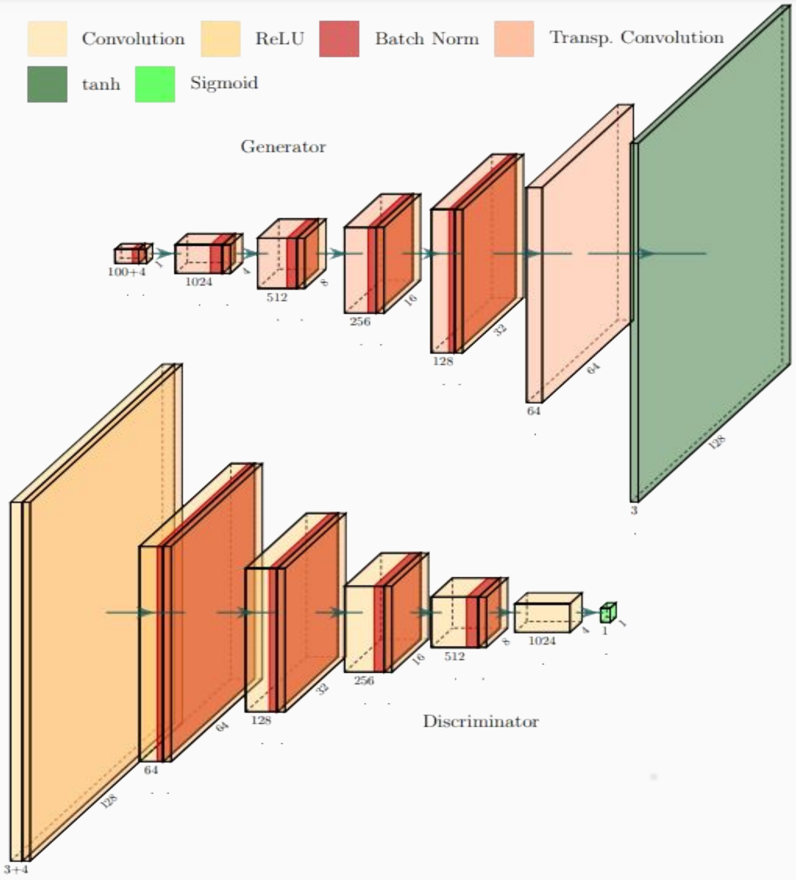
# C-GAN: Results

# City-GAN approach

- Generator: concatenate label with latent variable
- Discriminator: concatenate label with each pixel



# City-GAN Architecture



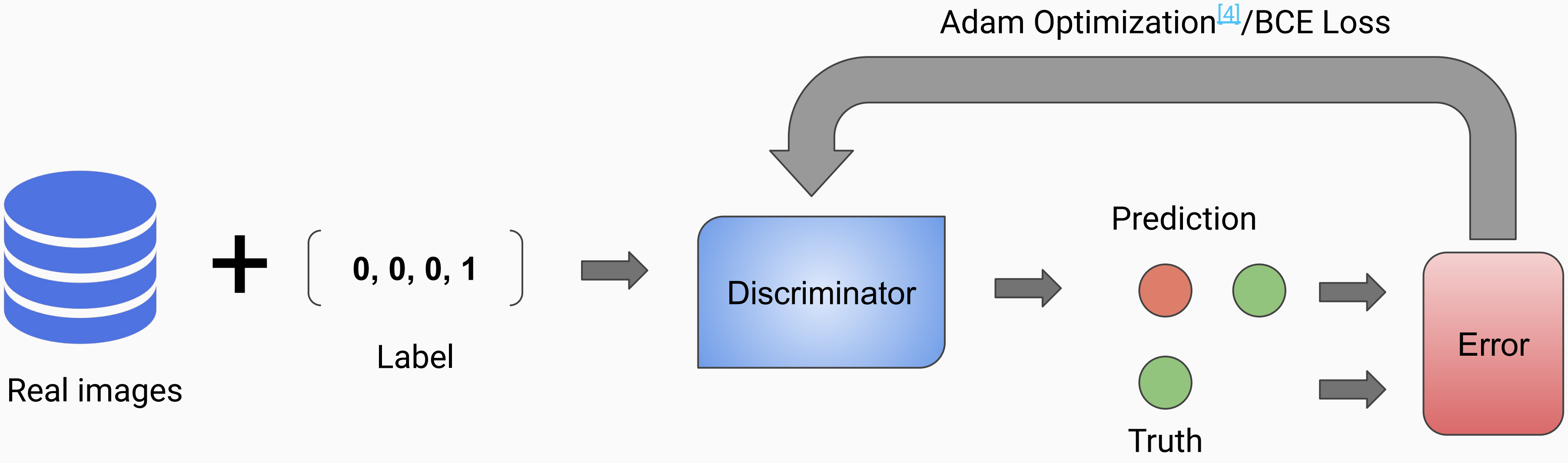
# City-GAN: Results

# Training

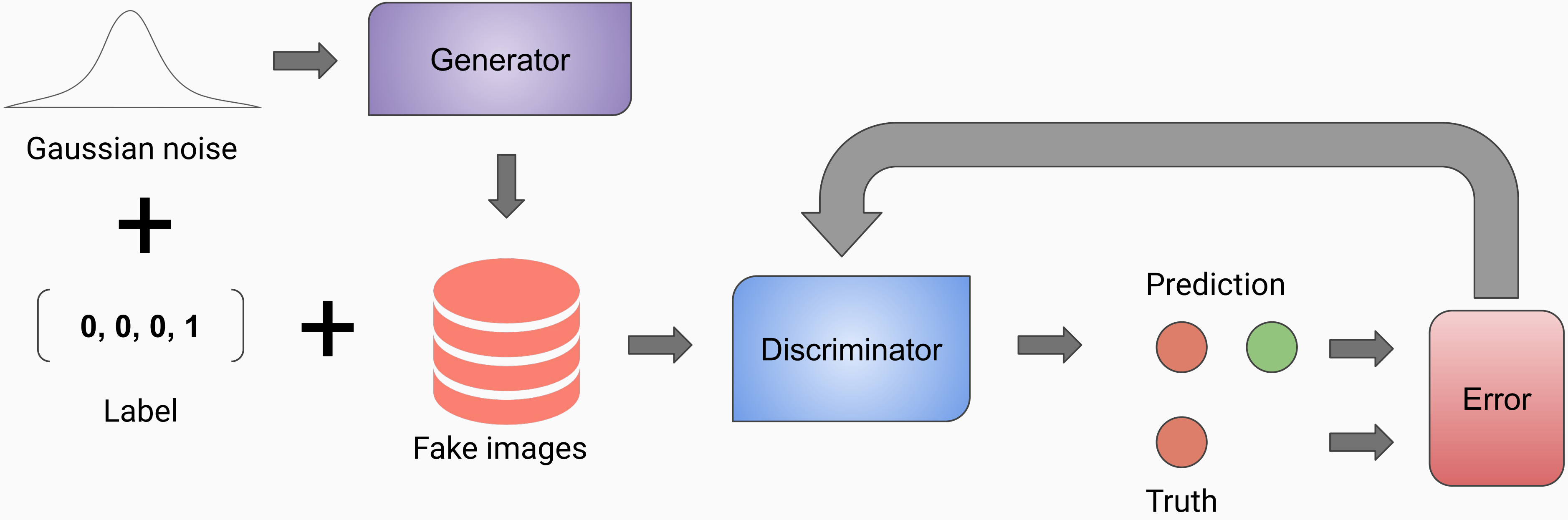
- Discriminator
    - Real images (1 batch)
    - Fake images (1 batch)

- Generator
    - Produce images, adjust weights based on discriminator feedback

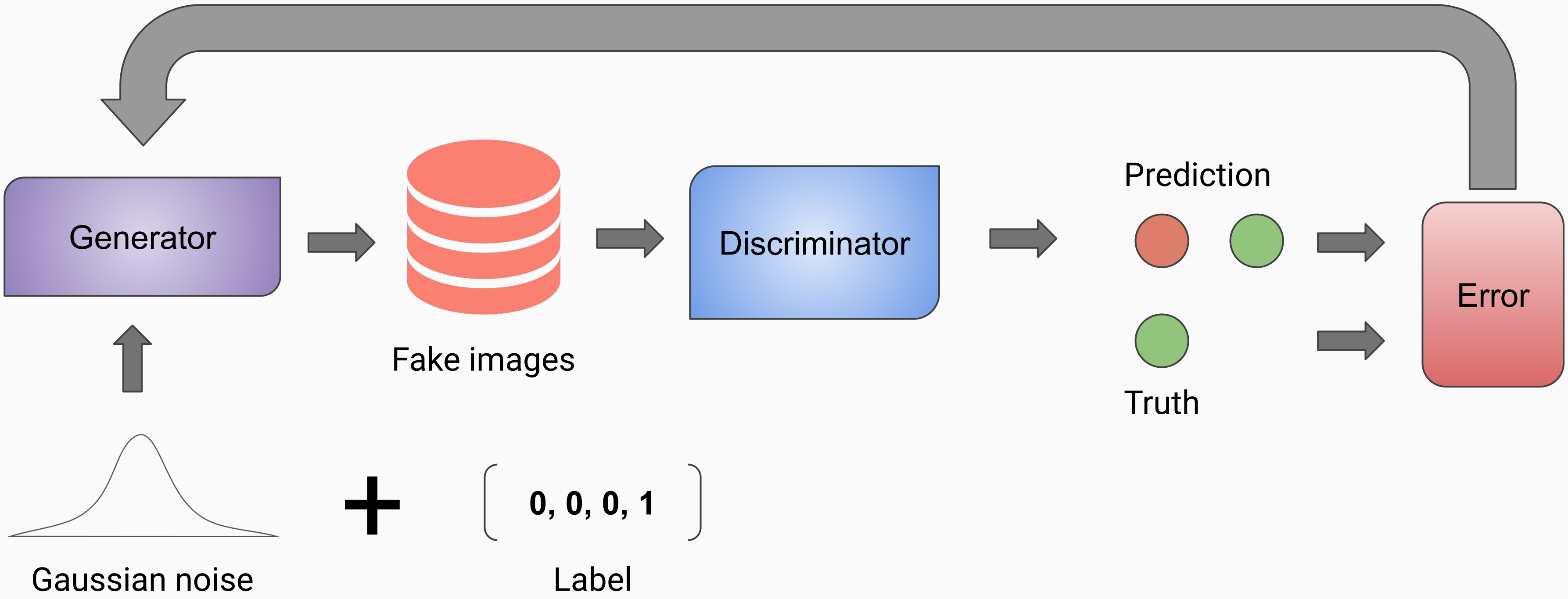
# Discriminator Training: Phase 1



# Discriminator Training: Phase 2



# Generator Training



# GAN Training

```
for batch in batches:
    #### DISCRIMINATOR TRAINING: PHASE 1 ####
    batch_with_labels = concat_with_each_pixel(class_one_hot, batch)
    train_discriminator(batch_with_labels, ground_truth=np.ones((batch_size, 1))

    #### DISCRIMINATOR TRAINING: PHASE 2 ####
    noise = torch.randn(batch_size, latent_dim) # Gaussian noise
    noise_with_labels = np.concatenate((class_one_hot, noise), axis=1) # Concat class with each latent vector
    fake_images = generator.generate(input=noise_with_labels, size=batch_size)
    fake_images_with_labels = concat_with_each_pixel(class_one_hot, fake_images)
    train_discriminator(fake_images_with_labels, ground_truth=np.zeros((batch_size, 1)))

    #### GENERATOR TRAINING ####
    fake_preds = discriminator.evaluate(fake_images_with_labels)
    loss_function = torch.nn.BCELoss()  # Binary Cross Entropy loss
    error = loss_function(output=fake_preds, ground_truth=np.ones((batch_size, 1)))
    gradient_generator = error.backward()  # Computes the gradient
    # Adam optimizer
    optimizer = torch.optim.Adam(generator, error, gradient_generator, lr=0.0002, betas=(0.5, 0.999))
    optimizer.step()

def train_discriminator(images, ground_truth):
    image_preds = discriminator.evaluate(images)
    loss_function = torch.nn.BCELoss()  # Binary Cross Entropy loss
    error = loss_function(output=image_preds, ground_truth=ground_truth)
    gradient_discriminator = error.backward()  # Computes the gradient
    # Adam optimizer
    optimizer = torch.optim.Adam(discriminator, error, gradient_discriminator, lr=0.0002, betas=(0.5, 0.999))
    optimizer.step()
```

# Generator Layers

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(104, 1024, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)

    (3): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (5): ReLU(inplace=True)

    (6): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (8): ReLU(inplace=True)

    (9): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (11): ReLU(inplace=True)

    (12): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (14): ReLU(inplace=True)

    (15): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (16): Tanh()
  )
)
```

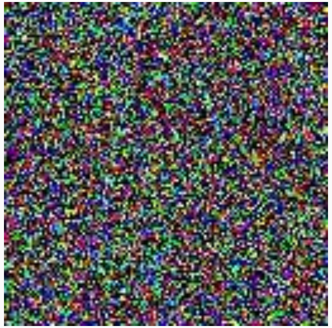
# Discriminator Layers

```
Discriminator(
  (main): Sequential(
    (0): Conv2d(7, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(negative_slope=0.2, inplace=True)

    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)

    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)

    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): LeakyReLU(negative_slope=0.2, inplace=True)

    (11): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (12): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (13): LeakyReLU(negative_slope=0.2, inplace=True)

    (14): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (15): Sigmoid()
  )
  (end): Sequential()
)
```

# Effect of Image Size

Varying levels of success were observed for different image sizes:

- **64x64:** best results
- **128x128:** some artifacts but generally acceptable
- **256x256:** no satisfactory results



# Additional Dataset

- Larger, uncurated dataset
- Stanford university
- Washington DC, Las Vegas
- Same quality as curated dataset

# Interpolation

Amsterdam → Florence

Amsterdam → D.C.

Amsterdam → Manhattan

Europe → U.S.

# Label Weights

- Sign and magnitude control feature presence

| Amsterdam | Florence | DC | Manhattan |
|---|---|---|---|
| **1** | **0** | **0** | **0** |
| **1/2** | **1/2** | **0** | **0** |
| **0** | **0** | **1/2** | **1/2** |
| **1** | **0** | **0** | **-1** |

Amsterdam

Europe

U.S.

Amsterdam - Manhattan

# Label Weights

- Amsterdam - Manhattan: (1, 0, 0, 0)-(0, 0, 0, 1)=(1, 0, 0, -1)

- Europe ➜ U.S.: (1/2, 1/2, 0, 0) ➜ (0, 0, 1/2, 1/2)

# Wrap-up

- DC-GAN not suited to the task
- Conditional GANs did not produce satisfactory results
- Adding labels to images produced better results
- Larger image sizes led to non-convergence

- Thoughts/questions?

# References

[1] Maximilian Bachl and Daniel Ferreira. 2019. City-GAN: Learning architectural styles using a custom Conditional GAN architecture. arXiv:1907.05280 [cs.CV] (Jul. 2019). https://arxiv.org/abs/1907.05280 arXiv:1907.05280.

[2] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs] (Nov. 2015). http://arxiv.org/abs/1511.06434 arXiv: 1511.06434.

[3] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. arXiv:1411.1784 [cs, stat] (Nov. 2014). http://arxiv.org/abs/1411.1784 arXiv: 1411.1784.

[4] Diederik Kingma and Jimmy Lei Ba. 2017. Adam: a method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] (Jan. 2017). https://arxiv.org/abs/1412.6980 arXiv:1412.6980v9.

Thanks!