Hands on with Generative Adversarial Networks (GANs): Making some pretty pictures

Nabeel Seedat

SUMMARY

1. Intuition behind GANs
2. How GANs work
3. Code Hands-on
 - GAN for making new pictures of clothes
 - Tips & Tricks to make GANs work
4. Where to from here?

http://tinyurl.com/pygotham-GANS

FACE GENERATION


2014


2015

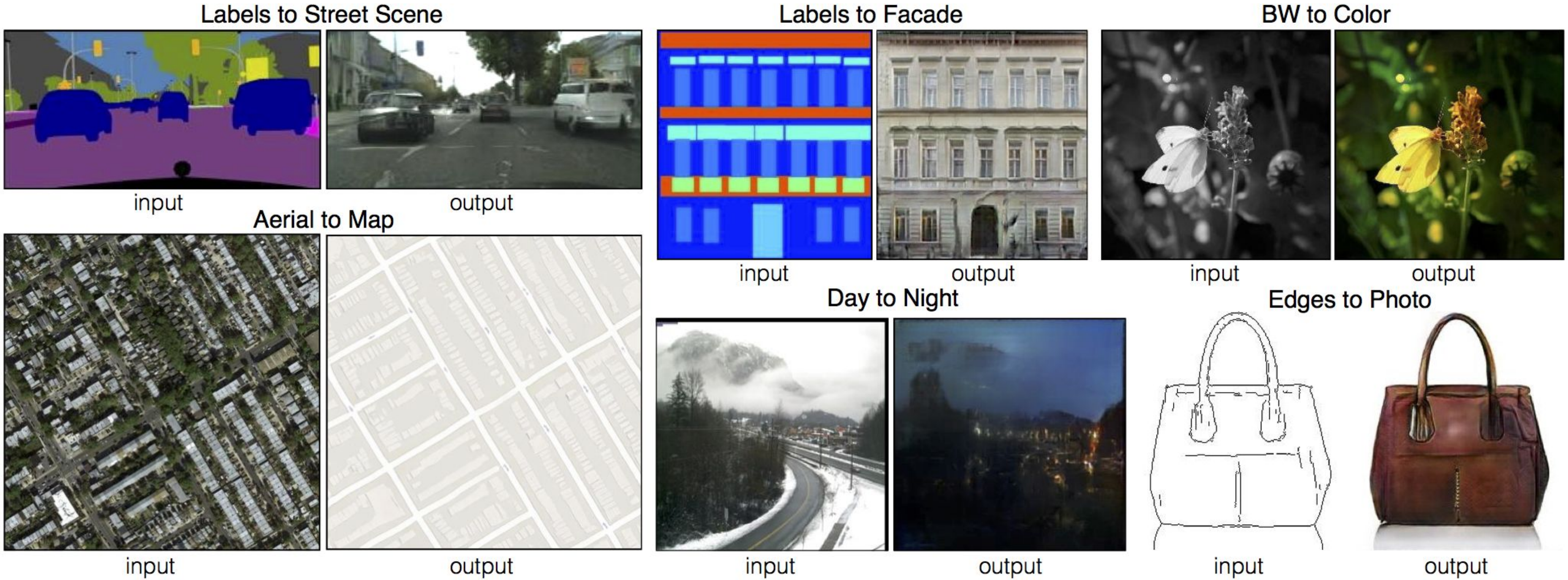

2016


2017

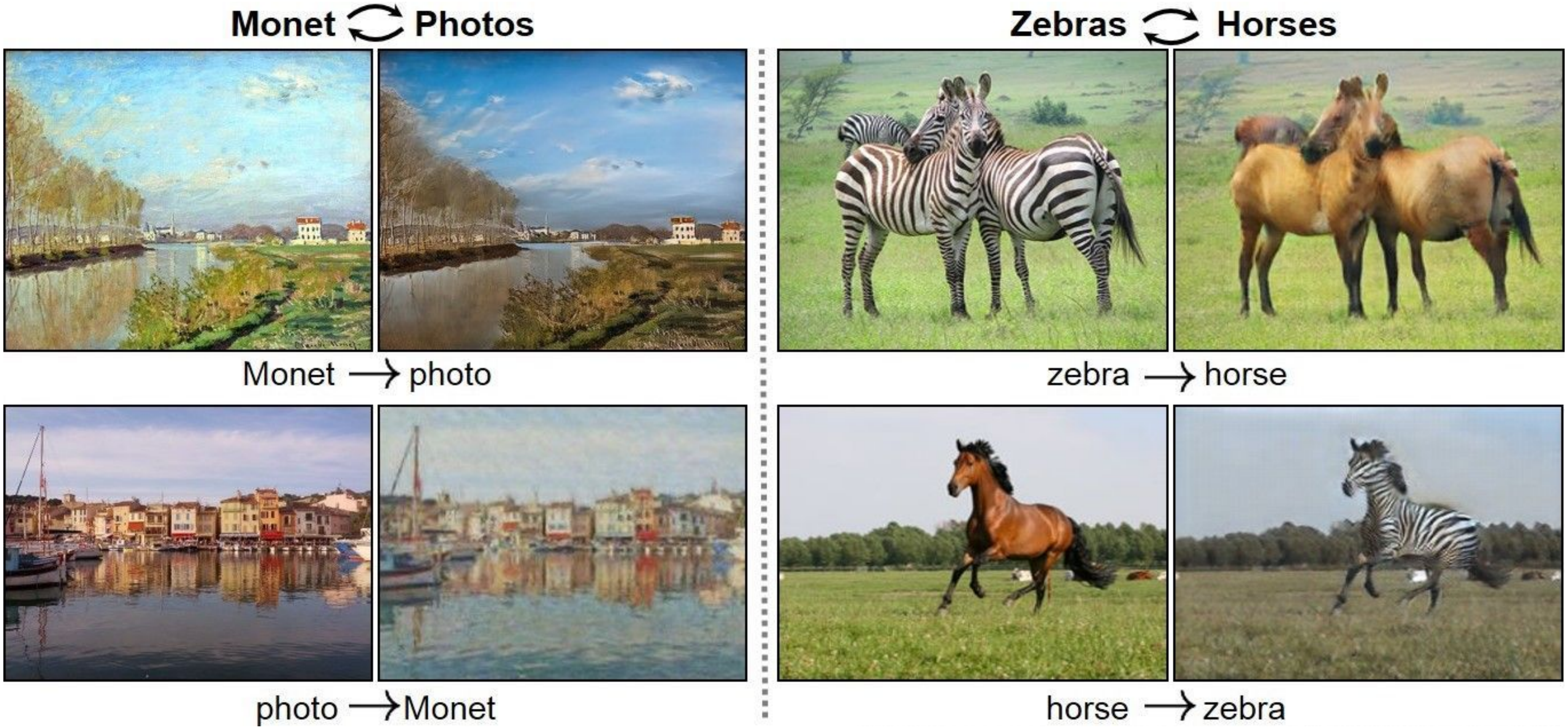

2018

[Goodfellow]

IMAGE-TO-IMAGE GENERATION



[CGAN (Isola et al)]



CycleGAN (Xu et al)

Text-to-Image Generation

AttnGAN (Xu et al)

Some Deepfakes too

Derpfakes

Generative Adversarial Networks (GANs) make this possible!

Goal: Generate new data!

- **Generative**

  Learn a generative model
- **Adversarial**

  Model trained in an adversarial setting
- **Networks**

  Use Deep Neural Networks

Generative?

- **Most deep learning is discriminative**
  Given image (x) - predict label (y)
  ρ(y|x)
- **Generative - don't care about labels**
  Learn how the data (x) was generated = ρ(x)
  If we know ρ(x), we can generate new images!

- *Deeper look in the notebook:*
  BUT note unsupervised is not always generative (e.g. K-Means clustering)

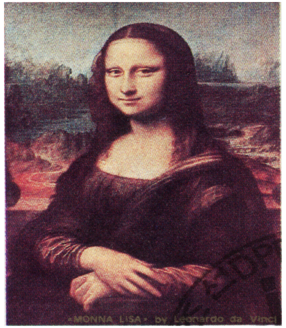
Adversarial Intuition

Generator

Discriminator

FAKE

REAL



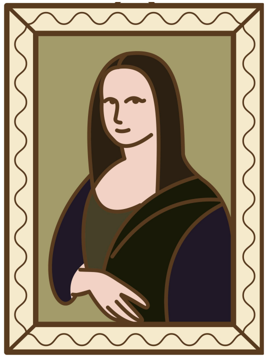
Real Data

Image Sources: Shutterstock

Intuition behind GANs

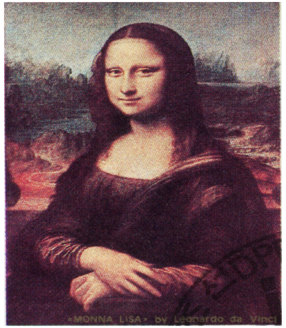
Generator

Discriminator



FAKE

REAL



Real Data

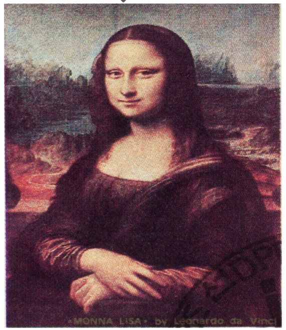
Image Sources: Shutterstock

Intuition behind GANs

Generator

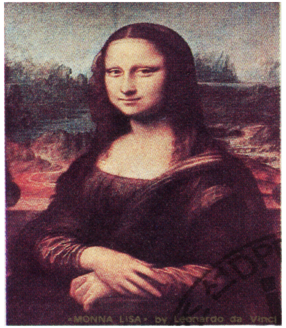
Discriminator

SUCCESS!
DISCRIMINATOR CAN'T TELL THE DIFFERENCE



REAL

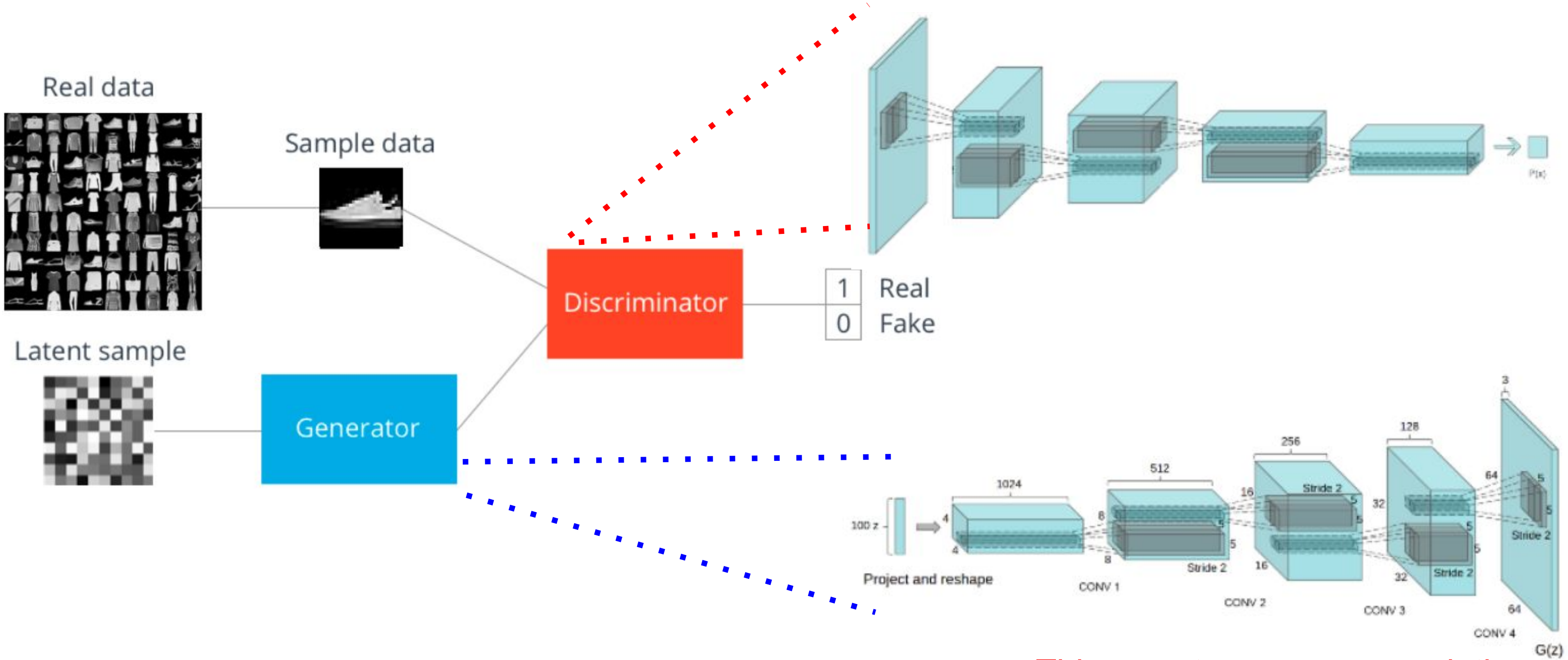
REAL



Real Data

Image Sources: Shutterstock

How GANs work



This uses transpose convolution

[DCGAN: Radford et al](#)

LOSS FUNCTION - MINIMAX

Generator tries to minimize the reward of the discriminator (by fooling it)

Discriminator tries to maximize reward (predicting correctly

$$\min_G \max_D V(D, G)$$

$$V(D,G) = \boxed{\mathbb{E}_{x\sim p_{data}(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]}$$

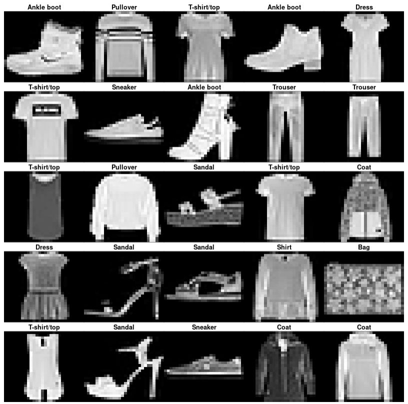
Log probability of D predicting the real data is actually real

Log probability of D predicting the data from G is NOT real

Generator plays no role so during generator training we don't need this

This has problems of vanishing gradients - BUT let's ignore it & chat at the end

GAN Hands On!



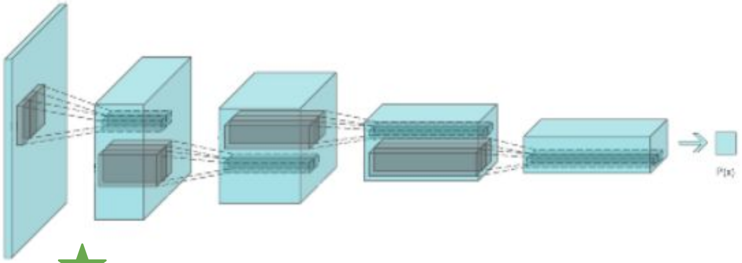
- Code to follow along can be found on Github
- It’s also a self-contained tutorial in a notebook linked to Google Colab (with a free GPU!)

Github link:
http://tinyurl.com/pygotham-GANS

Data Pre-processing

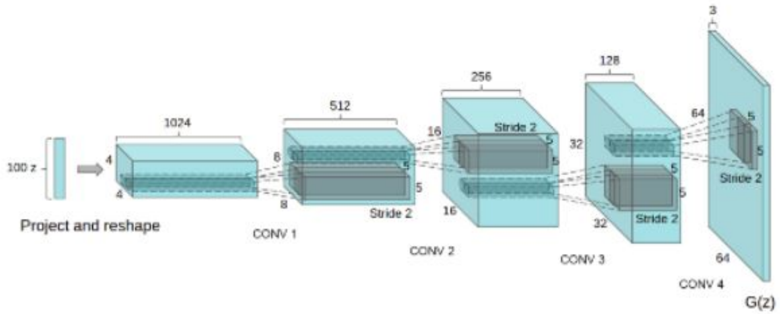
```
images = np.expand_dims(train, axis=3) # add a channel dim
images = images.astype('float32') # convert to float32
images = (images - 127.5) / 127.5 # normalize the images
batch_size = 256
dataset =
tf.data.Dataset.from_tensor_slices(images).shuffle(batch_size*10).batch(batch_size)
```

Discriminator



```python
def discriminator():

  discriminator = tf.keras.Sequential(name="discriminator")

  discriminator.add(tf.keras.layers.Conv2D(32,kernel_size=3,strides=2,padding="same",input_shape=(28,28,1)))

  discriminator.add(tf.keras.layers.LeakyReLU(alpha=0.2))
  discriminator.add(tf.keras.layers.Dropout(0.25))
  discriminator.add(tf.keras.layers.Conv2D(64,kernel_size=3,strides=2,padding="same"))
  discriminator.add(tf.keras.layers.BatchNormalization(momentum=0.8))

  discriminator.add(tf.keras.layers.LeakyReLU(alpha=0.2))
  discriminator.add(tf.keras.layers.Dropout(0.25))
  discriminator.add(tf.keras.layers.Conv2D(128,kernel_size=3, strides=2,padding="same"))
  discriminator.add(tf.keras.layers.BatchNormalization(momentum=0.8))

  discriminator.add(tf.keras.layers.LeakyReLU(alpha=0.2))
  discriminator.add(tf.keras.layers.Dropout(0.25))
  discriminator.add(tf.keras.layers.Conv2D(256,kernel_size=3,strides=1,padding="same"))
  discriminator.add(tf.keras.layers.BatchNormalization(momentum=0.8))

  discriminator.add(tf.keras.layers.LeakyReLU(alpha=0.2))
  discriminator.add(tf.keras.layers.Dropout(0.25))
  discriminator.add(tf.keras.layers.Flatten())
  discriminator.add(tf.keras.layers.Dense(1, activation='sigmoid'))

  return discriminator
```

Generator




```
def generator():
  generator = tf.keras.Sequential(name="generator")


  generator.add(tf.keras.layers.Dense(7 * 7 * 128,activation="relu",input_dim=100))
  generator.add(tf.keras.layers.Reshape([7, 7, 128]))

  generator.add(tf.keras.layers.UpSampling2D())
  generator.add(tf.keras.layers.Conv2D(128,kernel_size=3,padding="same"))
  generator.add(tf.keras.layers.BatchNormalization(momentum=0.8))
  generator.add(tf.keras.layers.Activation("relu"))

  generator.add(tf.keras.layers.UpSampling2D())
  generator.add(tf.keras.layers.Conv2D(64,kernel_size=3,padding="same"))
  generator.add(tf.keras.layers.BatchNormalization(momentum=0.8))
  generator.add(tf.keras.layers.Activation("relu"))

  generator.add(tf.keras.layers.Conv2D(1,kernel_size=3,padding="same"))
  generator.add(tf.keras.layers.Activation("tanh"))

  return generator
```

Transpose convolution

Transpose convolution

This outputs data= [-1,1]

Loss Functions

```
# define the loss function for the discriminator
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=False)

def discriminator_loss(real_images, fake_images):

    real_labels=tf.ones_like(real_images)
    real_loss = cross_entropy(real_labels, real_images)

    fake_labels=tf.zeros_like(fake_images)
    fake_loss = cross_entropy(fake_labels, fake_images)

    total_loss = real_loss + fake_loss
    return total_loss
```

Loss: real images

Loss: fake images

Discriminator Loss

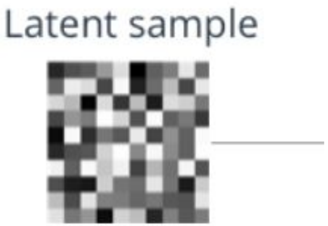
```
def generator_loss(fake_images):
    fake_labels=tf.zeros_like(fake_images)
    return cross_entropy(fake_labels, fake_images)
```

Generator Loss

GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])
```

1. Generate random noise



GAN Training

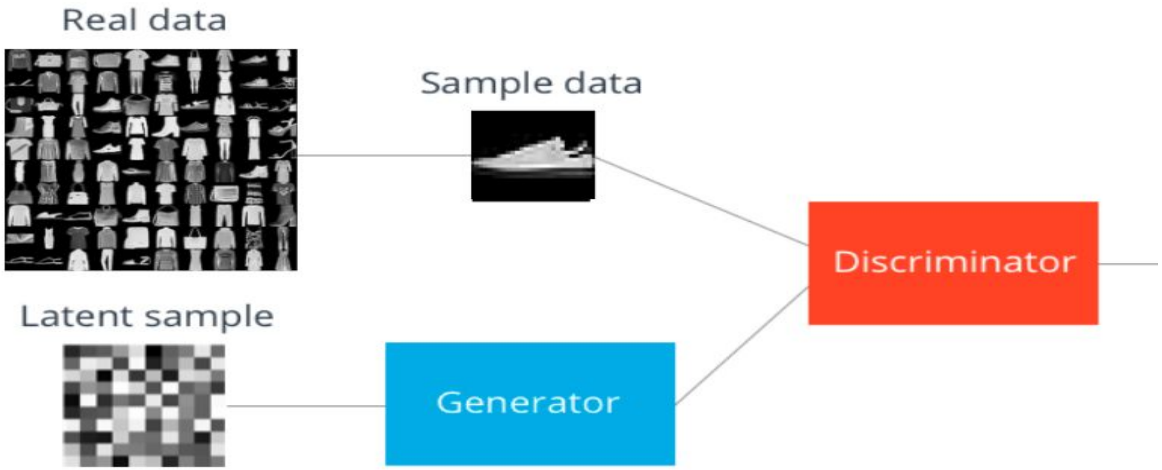
```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)
```

2. Generate fake image using noise

GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_images = discriminator(images, training=True)
        fake_images = discriminator(generated_images, training=True)
```

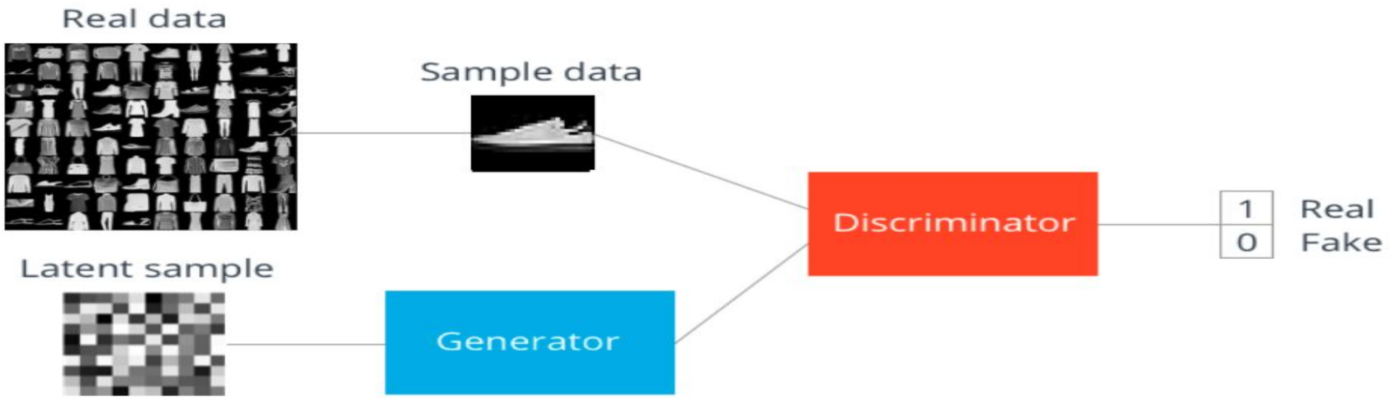
3. Pass real & fake images through the discriminator



GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_images = discriminator(images, training=True)
        fake_images = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_images)
        disc_loss = discriminator_loss(real_images, fake_images)
```

4. Compute Losses for both



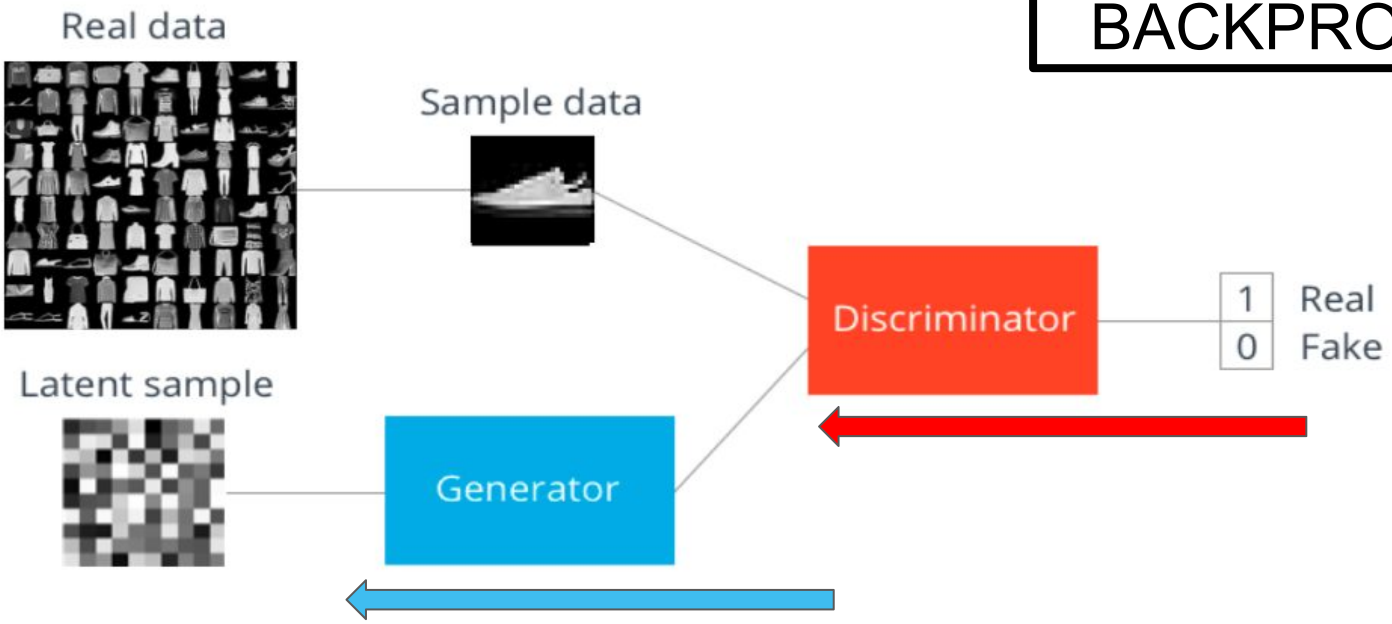
GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        generated_images = generator(noise, training=True)

        real_images = discriminator(images, training=True)
        fake_images = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_images)
        disc_loss = discriminator_loss(real_images, fake_images)

    gen_loss_mean(gen_loss)
    disc_loss_mean(disc_loss)
```

5. Get avg loss for the epoch

GAN Training

BACKPROP!



GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
      generated_images = generator(noise, training=True)

      real_images = discriminator(images, training=True)
      fake_images = discriminator(generated_images, training=True)

      gen_loss = generator_loss(fake_images)
      disc_loss = discriminator_loss(real_images, fake_images)

    gen_loss_mean(gen_loss)
    disc_loss_mean(disc_loss)

    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
```

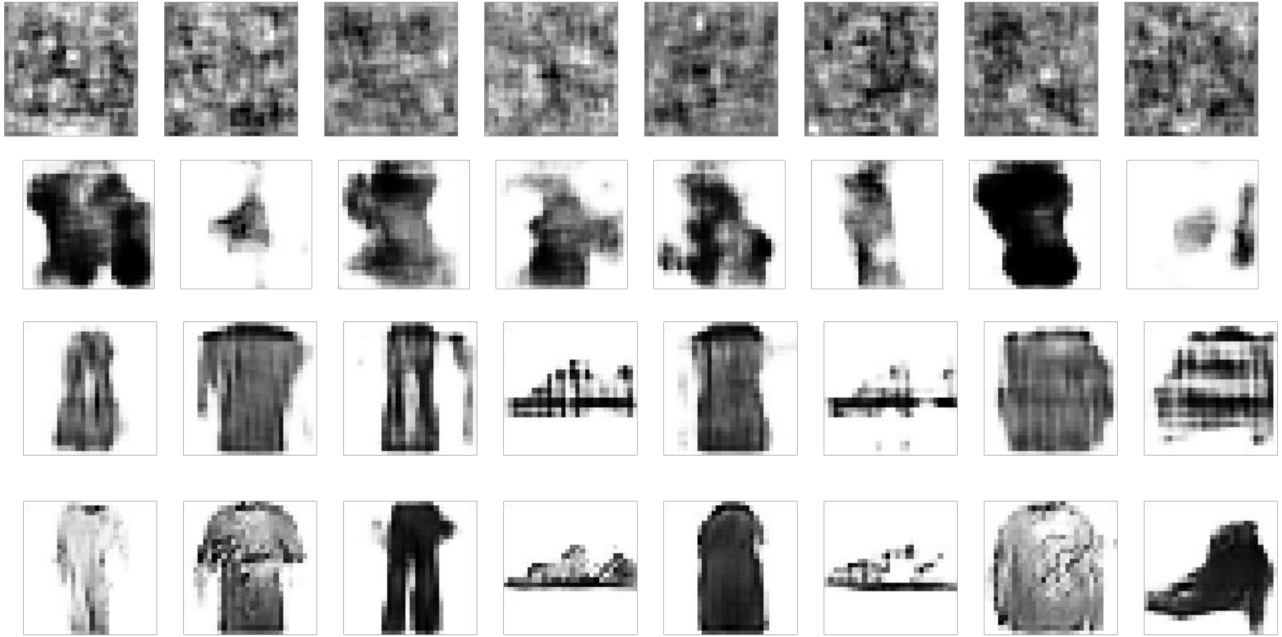
6. Compute gradient.
(Autodiff for backprop)

GAN Training

```
@tf.function
def train_step(images):
    noise = tf.random.normal([batch_size, 100])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
       generated_images = generator(noise, training=True)

       real_images = discriminator(images, training=True)
       fake_images = discriminator(generated_images, training=True)

       gen_loss = generator_loss(fake_images)
       disc_loss = discriminator_loss(real_images, fake_images)

    gen_loss_mean(gen_loss)
    disc_loss_mean(disc_loss)


    gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)

    generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
discriminator.trainable_variables))
```

7. Apply optimizer. Gradient step

RESULTS



Where to from here?

- Deeper problems + solutions
- Beyond the vanilla type GAN for more fun examples

Deeper Problems and Solutions

- ***Not converging!:*** Discriminator is too good so generator fails

Solution: Add noise to discriminator input (Arjovsky & Bottou)

- ***Vanishing gradients:*** Discriminator is too good so generator fails

Solution: Use Wasserstein Loss (link)

- ***Mode collapse:*** Generator produces the same examples

Solution: Wasserstein Loss or Unrolled GAN (Metz et al)

Fun Examples - GAN Zoo

- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

https://github.com/hindupuravinash/the-gan-zoo

THANK YOU!

Questions?