The University of Queensland Australia | CREATE CHANGE

Generative Adversarial Networks

Siyu

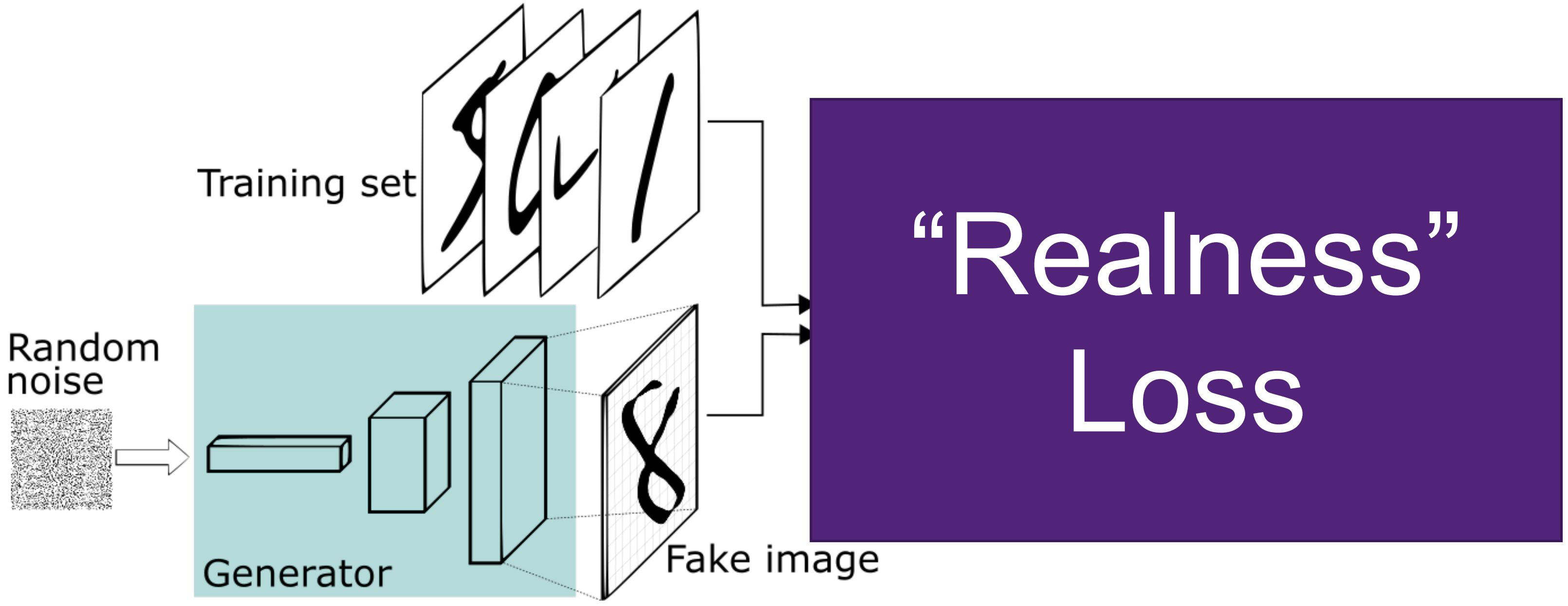
Today's Content

Theory of GANs

TensorFlow Implementation

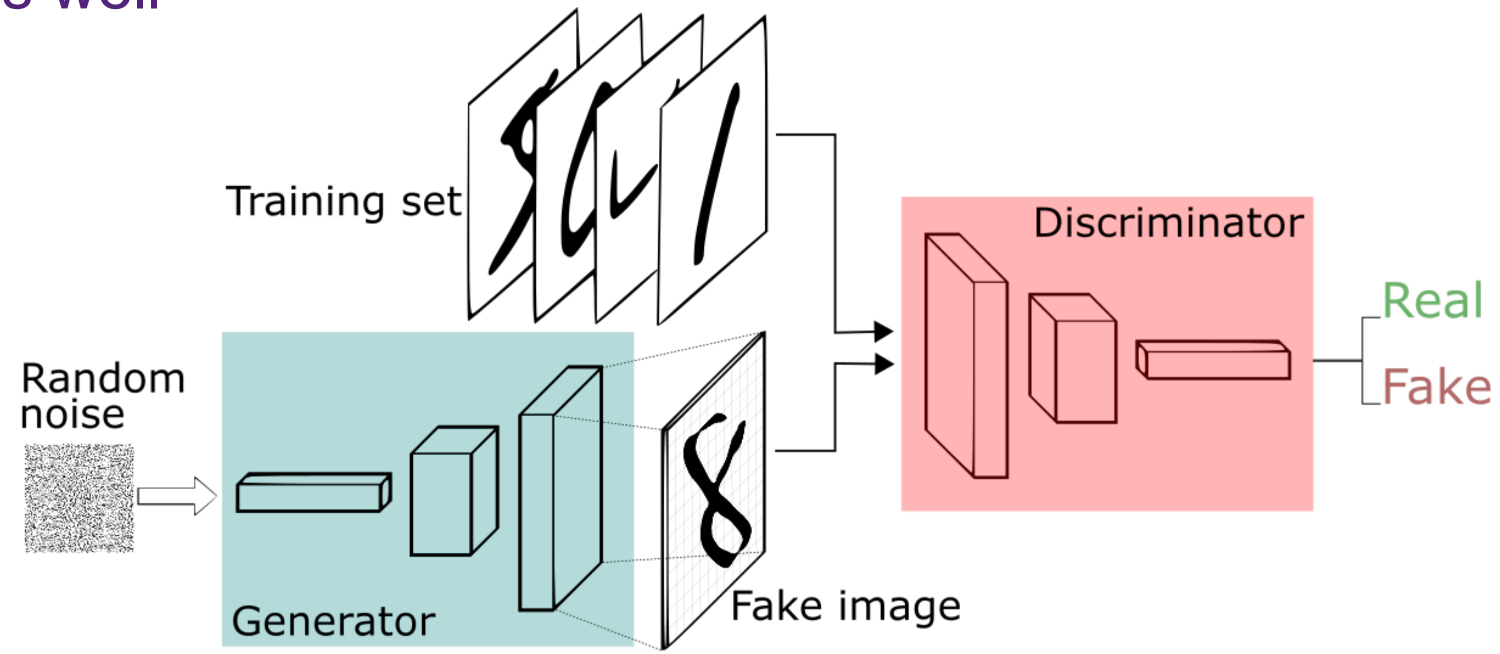
Generating Data

- Generating data had been historically difficult and was thought to be impossible in many cases.
- How do we define **realness**?
- Naïve loss functions like MSE cannot quantify **realness**.



Generative Adversarial Networks (GANs)

- A framework for **generating** data, can be supervised or **unsupervised**.
- Involved at least two models: a **generator** and a **discriminator**.
- Neural nets can be trained to classify things, they can be trained to assess realness as well

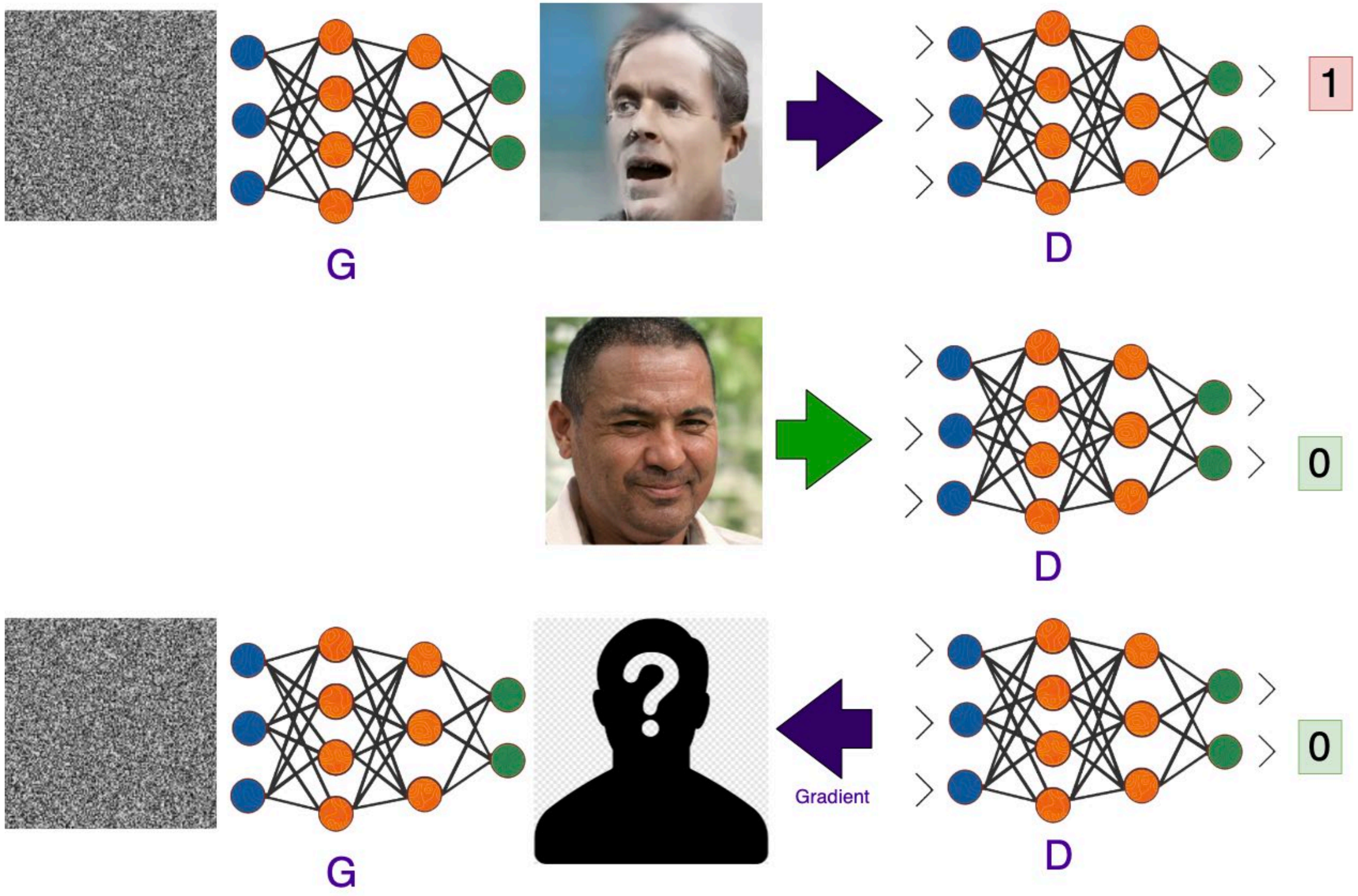


Generative Adversarial Networks – Training Step

Step 1 – Training the **Discriminator** on a fake sample

Step 2 – Training the **Discriminator** on a real sample

Step 3 – Optimise the **Generator** to make the **Discriminator** predict “real”



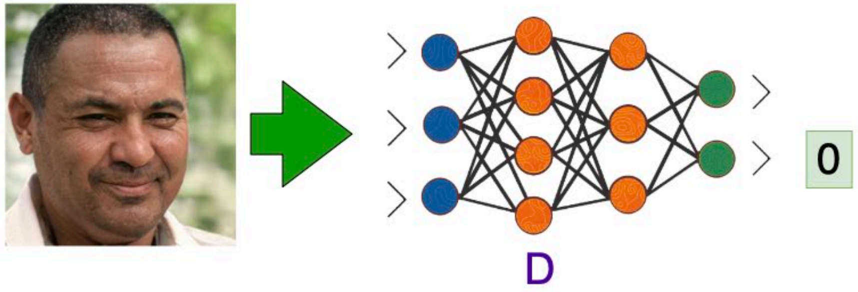
Generative Adversarial Networks – Training Step

Step 1

$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(1, D(fake))$$

Step 2

$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(0, D(real))$$



Step 3

$$\theta_G \leftarrow \theta_G - \alpha \frac{\partial}{\partial \theta_G} \mathcal{L}_{CE}(0, D(G(noise)))$$

The models are trained alternatively until **equilibrium**

Training Using Model.fit()

Step 1

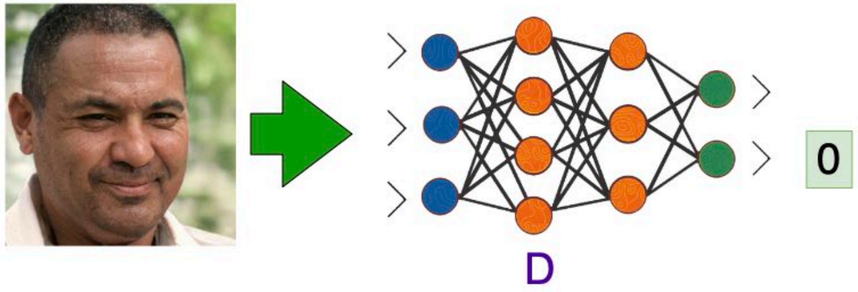
$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(1, D(fake))$$

d.train_on_batch(fake, [1, 1, 1 …])

Step 2

$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(0, D(real))$$

d.train_on_batch(real, [0, 0, 0 …]))



Step 3

$$\theta_G \leftarrow \theta_G - \alpha \frac{\partial}{\partial \theta_G} \mathcal{L}_{CE}(0, D(G(noise)))$$

gd.train_on_batch(noise, [0, 0, 0 …]))

The models are trained alternatively until **equilibrium**

Training Using Model.fit()

Step 1

$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(1, D(fake))$$

d.train_on_batch(fake, 1)

Step 2

$$\theta_D \leftarrow \theta_D - \alpha \frac{\partial}{\partial \theta_D} \mathcal{L}_{CE}(0, D$$

d.train_on_bat

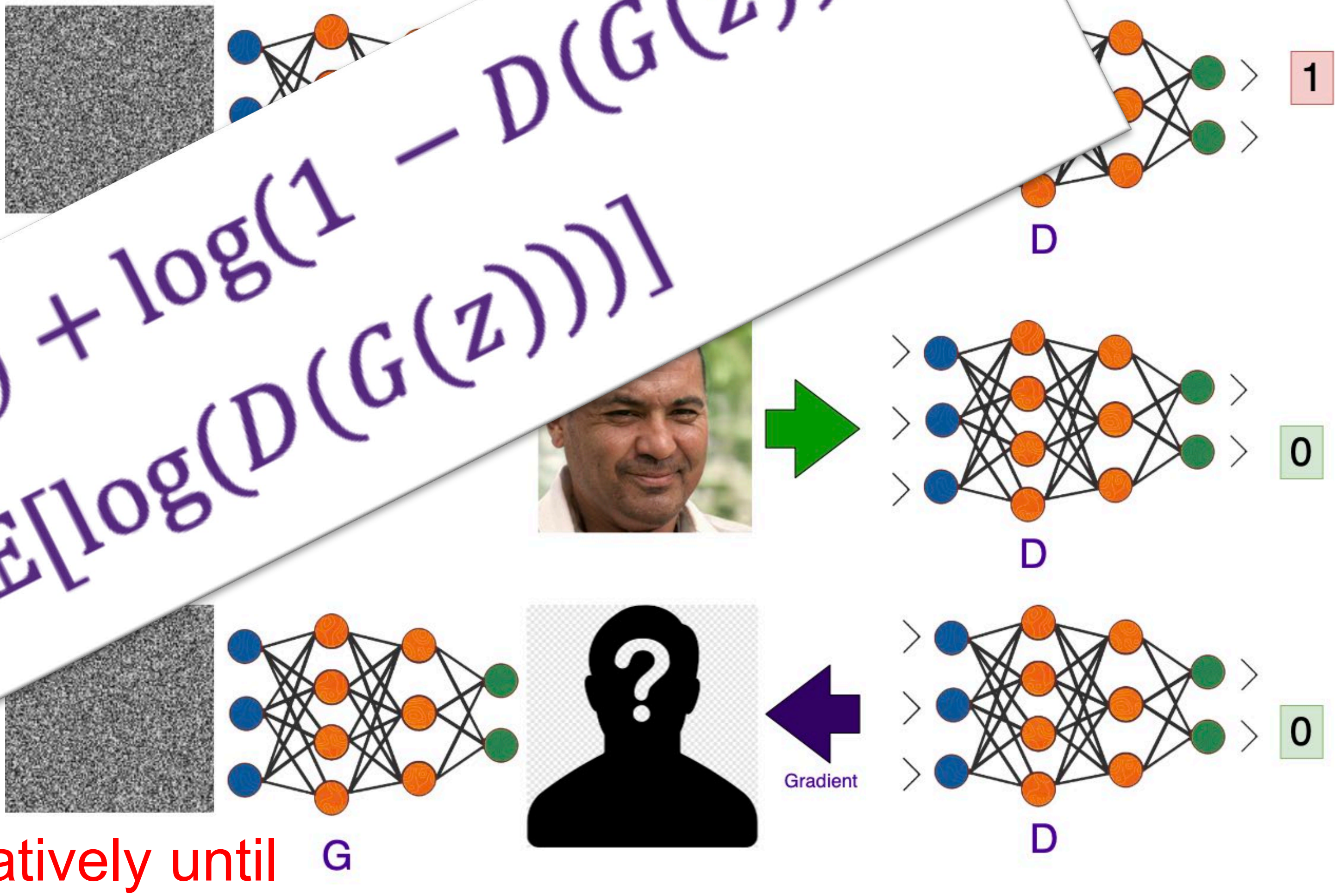
St

$\theta_G \leftarrow$

g_an oise, 0)

$$\nabla_{\theta_d} \mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\nabla_{\theta_g} \mathbb{E}[\log(D(G(z)))]$$



The m rained alternatively until **equilibr**

Generative Adversarial Networks – Implementation

- We had to convert the real loss function to a supervised learning problem

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$
$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

Cross Entropy Losses for model.fit(x, y)

- GAN losses can be very complex. For example, the StarGAN v2 loss:

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x},y}\left[\log D_y(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}[\log(1 - D_{\widetilde{y}}(G(\mathbf{x},\widetilde{\mathbf{s}})))],$$

$$\mathcal{L}_{sty} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}\left[||\widetilde{\mathbf{s}} - E_{\widetilde{y}}(G(\mathbf{x},\widetilde{\mathbf{s}}))||_1\right].$$

$$\mathcal{L}_{ds} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}_1,\mathbf{z}_2}\left[||G(\mathbf{x},\widetilde{\mathbf{s}}_1) - G(\mathbf{x},\widetilde{\mathbf{s}}_2)||_1\right],$$

$$\mathcal{L}_{cyc} = \mathbb{E}_{\mathbf{x},y,\widetilde{y},\mathbf{z}}\left[||\mathbf{x} - G(G(\mathbf{x},\widetilde{\mathbf{s}}),\hat{\mathbf{s}})||_1\right],$$

$$\min_{G,F,E}\max_{D} \quad \mathcal{L}_{adv} + \lambda_{sty}\mathcal{L}_{sty} - \lambda_{ds}\mathcal{L}_{ds} + \lambda_{cyc}\mathcal{L}_{cyc},$$

Generative Adversarial Networks – In plementation

- We had to convert the real loss function t d learning problem

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$
$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

s for

model.fit(x, y)?

- GAN losses can be ve mple, the StarGAN v2 loss:

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x},y}\left[\log\right.$$
$$\mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}[\log (\quad (\mathbf{x},\widetilde{\mathbf{s}})))],$$
$$\mathcal{L}_{sty} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}\left[||\widetilde{\mathbf{s}} - E_{\widetilde{y}}(G(\mathbf{x},\widetilde{\mathbf{s}}))||_1\right].$$
$$\mathcal{L}_{ds} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}_1,\mathbf{z}_2}\left[||G(\mathbf{x},\widetilde{\mathbf{s}}_1) - G(\mathbf{x},\widetilde{\mathbf{s}}_2)||_1\right],$$
$$\mathcal{L}_{cyc} = \mathbb{E}_{\mathbf{x},y,\widetilde{y},\mathbf{z}}\left[||\mathbf{x} - G(G(\mathbf{x},\widetilde{\mathbf{s}}),\hat{\mathbf{s}})||_1\right],$$

$$\min_{G,F,E}\max_{D} \quad \mathcal{L}_{adv} + \lambda_{sty}\mathcal{L}_{sty} - \lambda_{ds}\mathcal{L}_{ds} + \lambda_{cyc}\mathcal{L}_{cyc},$$

Generative Adversarial Networks – Implementation

model.fit(x, y)?

- model.fit is good for **supervised learning**
- model.fit is a **convenience API**, it does not reflect the nature of deep learning
- It is hard to define the **input (x)** and **target (y)** for complex loss functions

Gradient Tape

- Use **tf.GradientTape()** to train keras models.
- Update models based on **optimisation objectives** rather than input (x) and labels (y).
- Many GAN problems have **no clearly-defined labels**.

$$\nabla_{\theta_d} \mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\nabla_{\theta_g} \mathbb{E}[\log(D(G(z)))]$$

Example GAN loss function

- This is exactly how pytorch works

GAN Example Using GradientTape

```
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

Real Loss

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

GAN Example Using GradientTape

```
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

Fake Loss

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

GAN Example Using GradientTape

```python
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d} \mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

Gradient w.r.t Discriminator

$$\nabla_{\theta_g} \mathbb{E}[\log(D(G(z)))]$$

GAN Example Using GradientTape

```python
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\theta_d \longleftarrow \theta_d - \alpha\nabla_{\theta_d}(x, z, G, D)$$

Update the weights of D

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

GAN Example Using GradientTape

```
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

Generator Loss

GAN Example Using GradientTape

```
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

Gradient w.r.t
Generator

GAN Example Using GradientTape

```
for i in range(conf['ITERATIONS']):

    z = tf.random.normal(shape=[batch_size, latent_dim])
    x = get_img_batch()
    x /= 255

    # g and d are tf.keras models
    with tf.GradientTape() as tape:
        d_loss_real = real_loss(d, x)
        d_loss_fake = fake_loss(g, d, z)
        d_loss = d_loss_real + d_loss_fake

    grads = tape.gradient(d_loss, d.trainable_variables)
    d_opt.apply_gradients(zip(grads, d.trainable_variables))

    # train generator
    with tf.GradientTape() as tape:
        g_loss = gen_loss(g, d, z)
    grads = tape.gradient(g_loss, g.trainable_variables)
    g_opt.apply_gradients(zip(grads, g.trainable_variables))

    # print log
    print("STEP=%05d, d_loss=%.5f, g_loss=%.5f" % (
            i,
            np.mean(d_loss.numpy()),
            np.mean(g_loss.numpy())
        )
    )
```

$$\nabla_{\theta_d}\mathbb{E}[\log D(x) + \log(1 - D(G(z)))]$$

$$\nabla_{\theta_g}\mathbb{E}[\log(D(G(z)))]$$

$$\theta_g \leftarrow \theta_g - \alpha\nabla_{\theta_g}(z, G, D)$$

Update the weights of G

StarGANv2 Example Using GradientTape

```
with tf.GradientTape(persistent=True) as tape:
    # 1. adv loss
    st = F[yt](z)
    g_st = G([x, st])
    d_yt = D[yt](g_st)
    d_y = D[y](x)
    ld_fake = fake_loss(d_yt)
    ld_real = real_loss(-d_y)
    ld_adv = tf.reduce_mean(ld_fake + ld_real + r1_gp(x, D[y]))

    # 1. g_adv_loss
    lg_adv = tf.reduce_mean(tf.nn.softplus(-d_yt))

    # 2. style recon loss
    l_sty = l1_loss(st, E[yt](g_st)) * conf['LAMBDA_STY']

    # 3. style diversification loss
    s1 = F[yt](z1)
    s2 = F[yt](z2)
    g_st1 = G([x, s1])
    g_st2 = G([x, s2])
    l_ds = -l1_loss(g_st1, g_st2) * decay.next() # "-" sign to maximise

    # 4. cycle consistency loss
    l_cyc = l1_loss(x, G([g_st, E[y](x)])) * conf['LAMBDA_CYC']

    lg_tot = lg_adv + l_sty + l_cyc + l_ds
```

$$\mathcal{L}_{adv} = \mathbb{E}_{\mathbf{x},y}\left[\log D_y(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}[\log\left(1 - D_{\widetilde{y}}(G(\mathbf{x},\widetilde{\mathbf{s}}))\right)],$$

$$\mathcal{L}_{sty} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}}\left[||\widetilde{\mathbf{s}} - E_{\widetilde{y}}(G(\mathbf{x},\widetilde{\mathbf{s}}))||_1\right].$$

$$\mathcal{L}_{ds} = \mathbb{E}_{\mathbf{x},\widetilde{y},\mathbf{z}_1,\mathbf{z}_2}\left[||G(\mathbf{x},\widetilde{\mathbf{s}}_1) - G(\mathbf{x},\widetilde{\mathbf{s}}_2)||_1\right],$$

$$\mathcal{L}_{cyc} = \mathbb{E}_{\mathbf{x},y,\widetilde{y},\mathbf{z}}\left[||\mathbf{x} - G(G(\mathbf{x},\widetilde{\mathbf{s}}),\hat{\mathbf{s}})||_1\right],$$

Losses are implemented as they are defined in the paper

StarGANv2 Example Using GradientTape

```
def train(model, loss, opt, tape):
    grads = tape.gradient(loss, model.trainable_variables)
    opt.apply_gradients(zip(grads,  model.trainable_variables))

train(D[yt], ld_adv, D_opts[yt], tape)
train(D[y], ld_adv, D_opts[y], tape)

train(G, lg_tot, G_opt, tape)
train(E[yt], lg_tot, E_opts[yt], tape)
train(E[y], lg_tot, E_opts[y], tape)

train(F[yt], lg_tot, F_opts[yt], tape)
```

Updated all the models using the same loss function