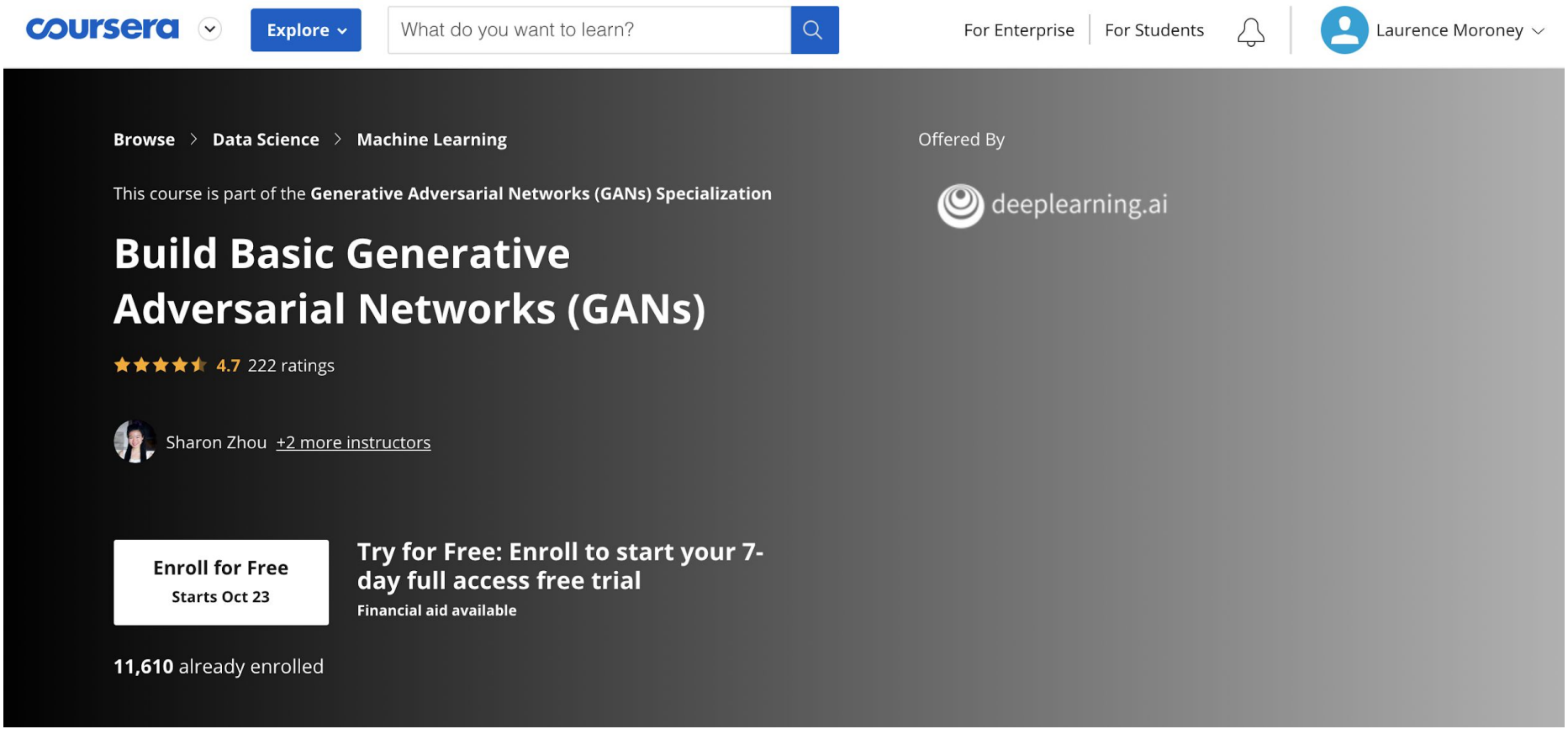
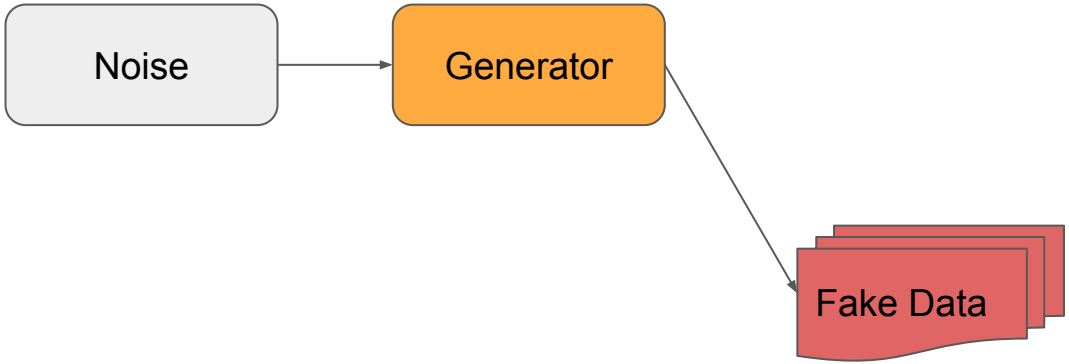
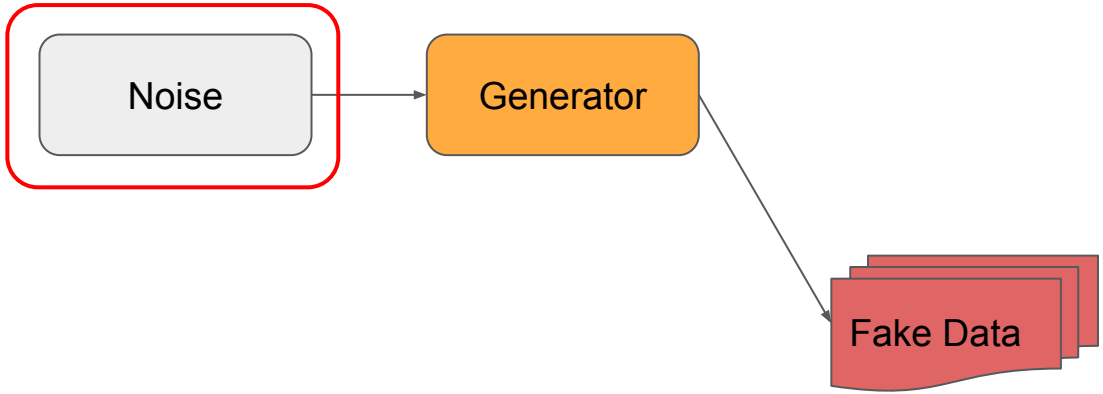
Copyright Notice

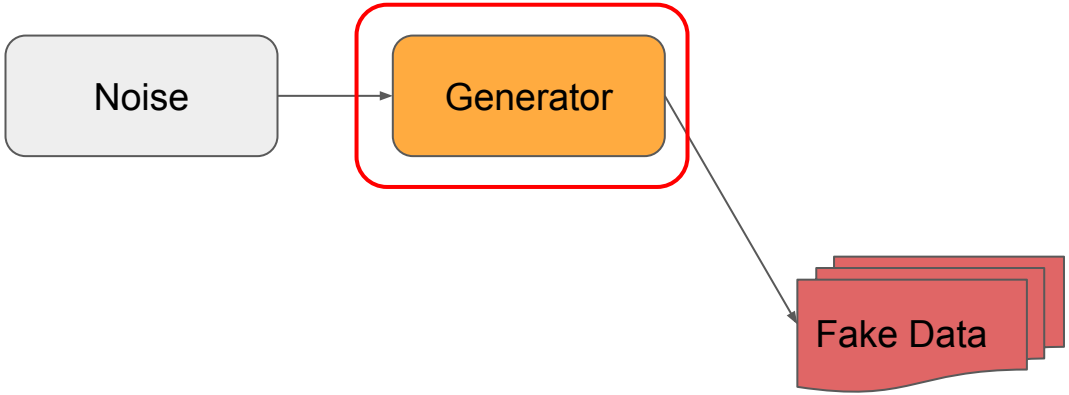
https://www.deeplearning.ai/generative-adversarial-networks-specialization

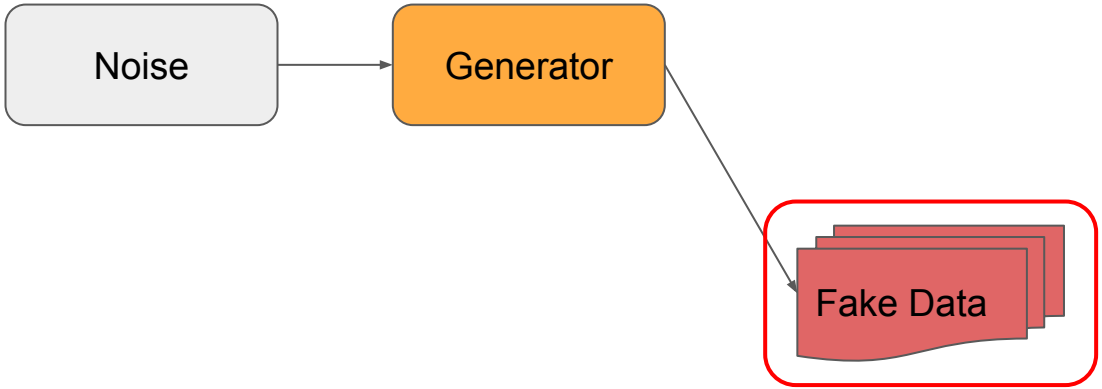


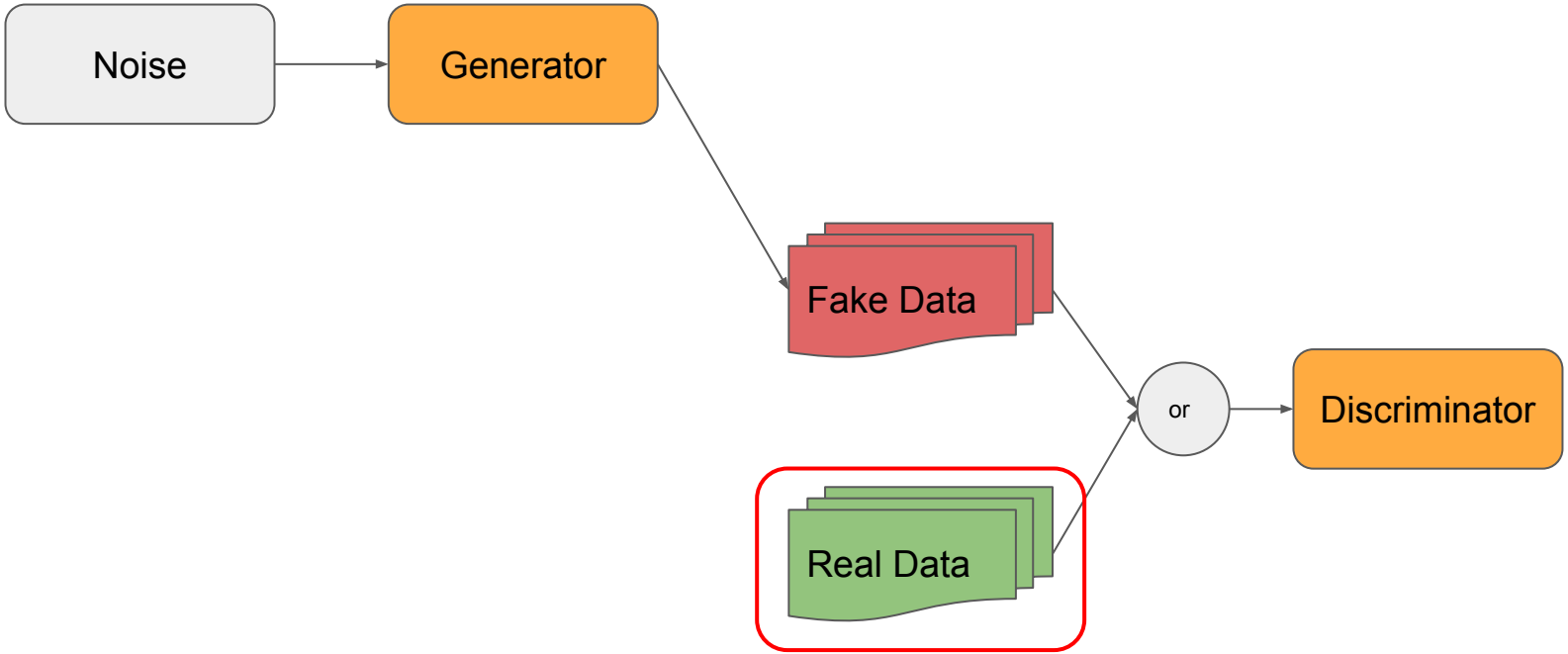
https://arxiv.org/abs/1406.2661

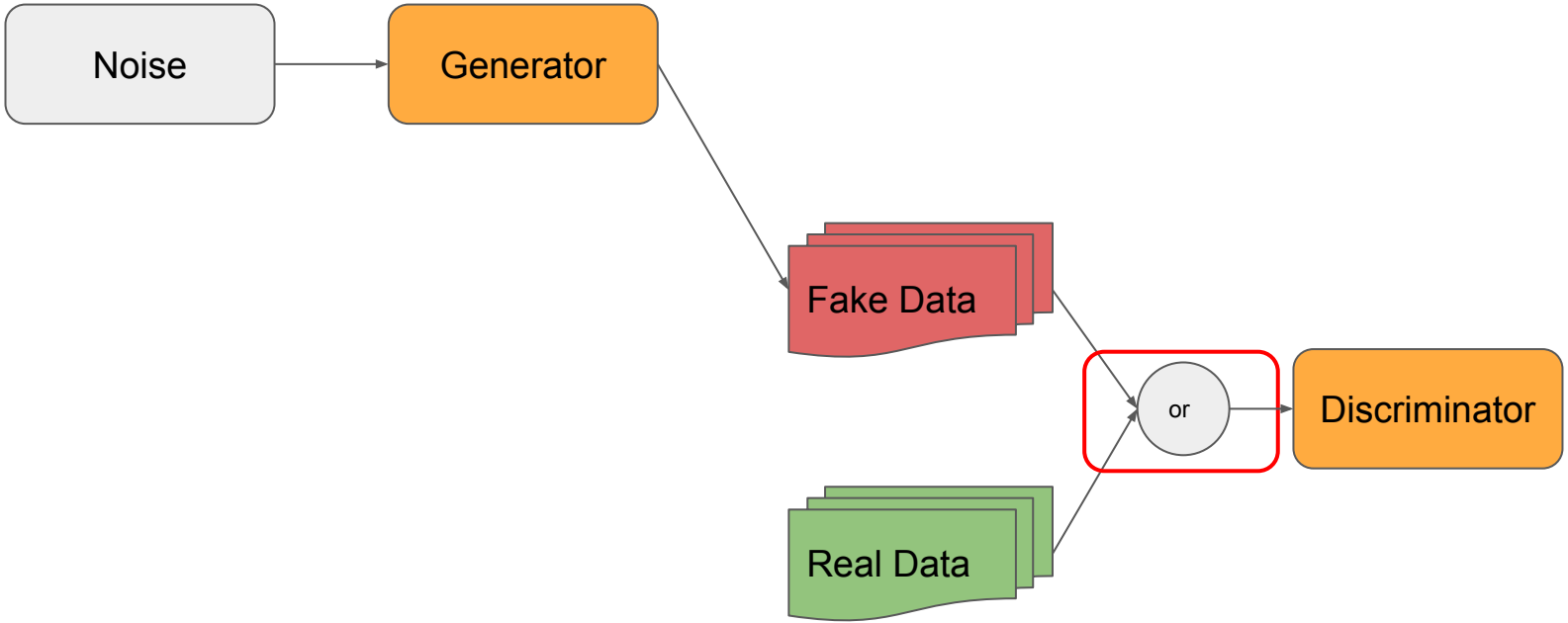

Noise
Generator
Fake Data

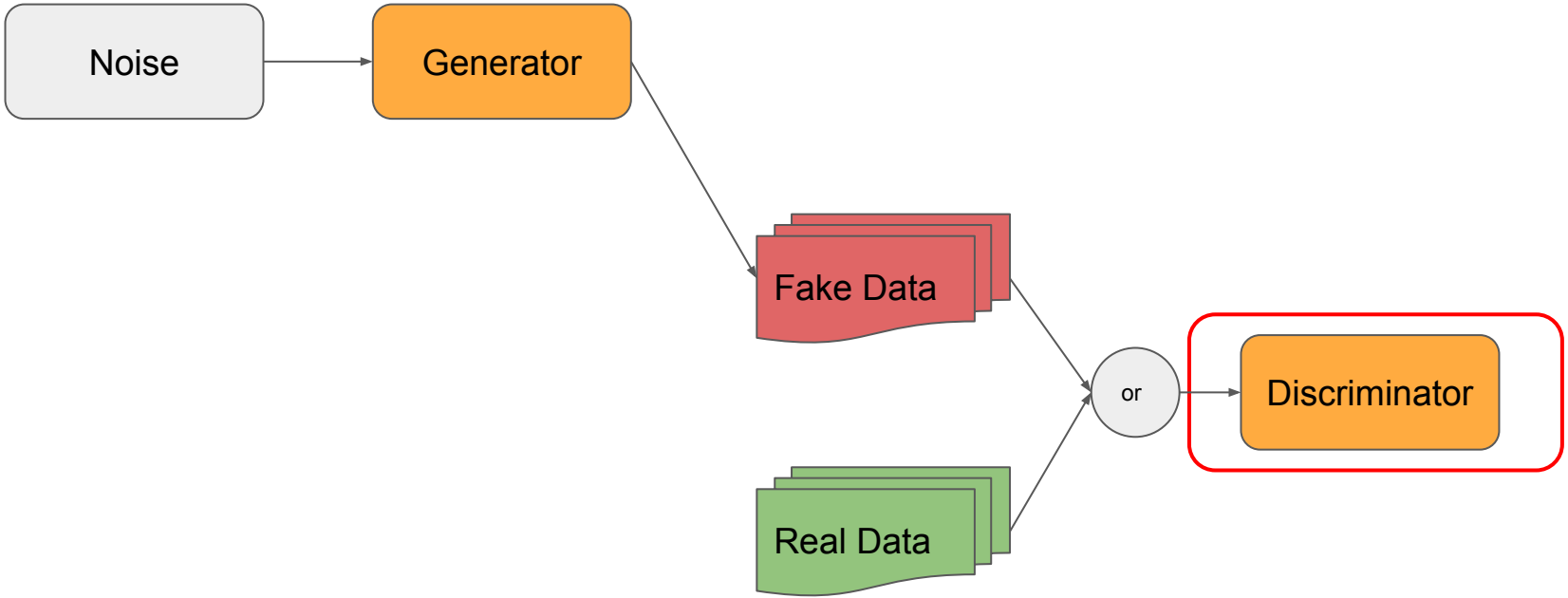

Noise
Generator
Fake Data

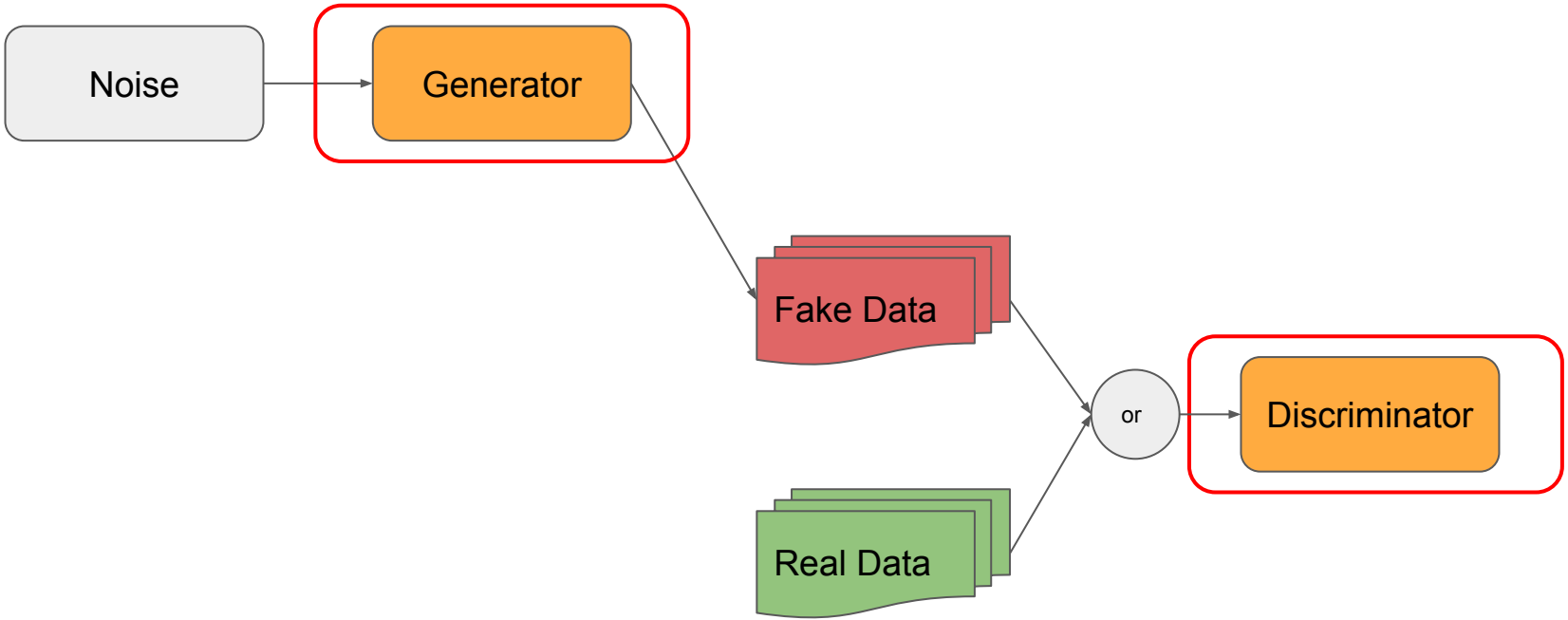

Noise
Generator
Fake Data

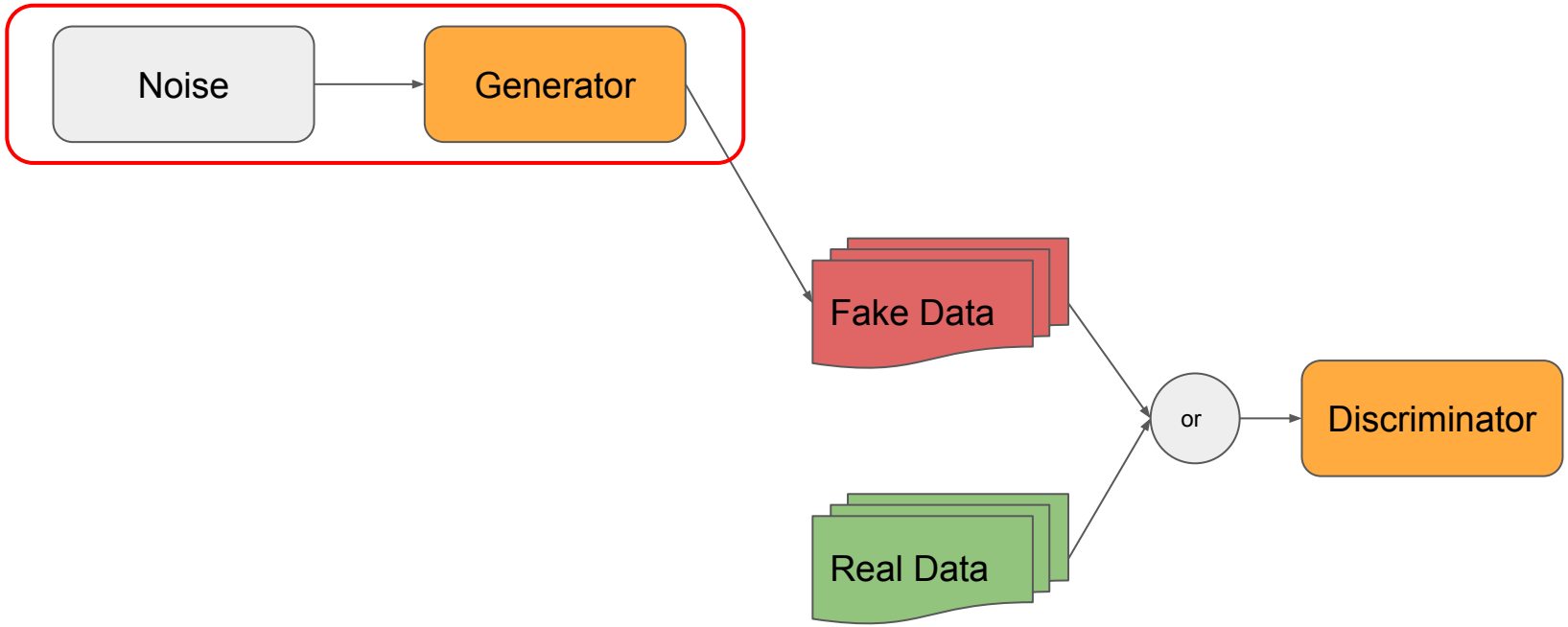

Noise
Generator
Fake Data

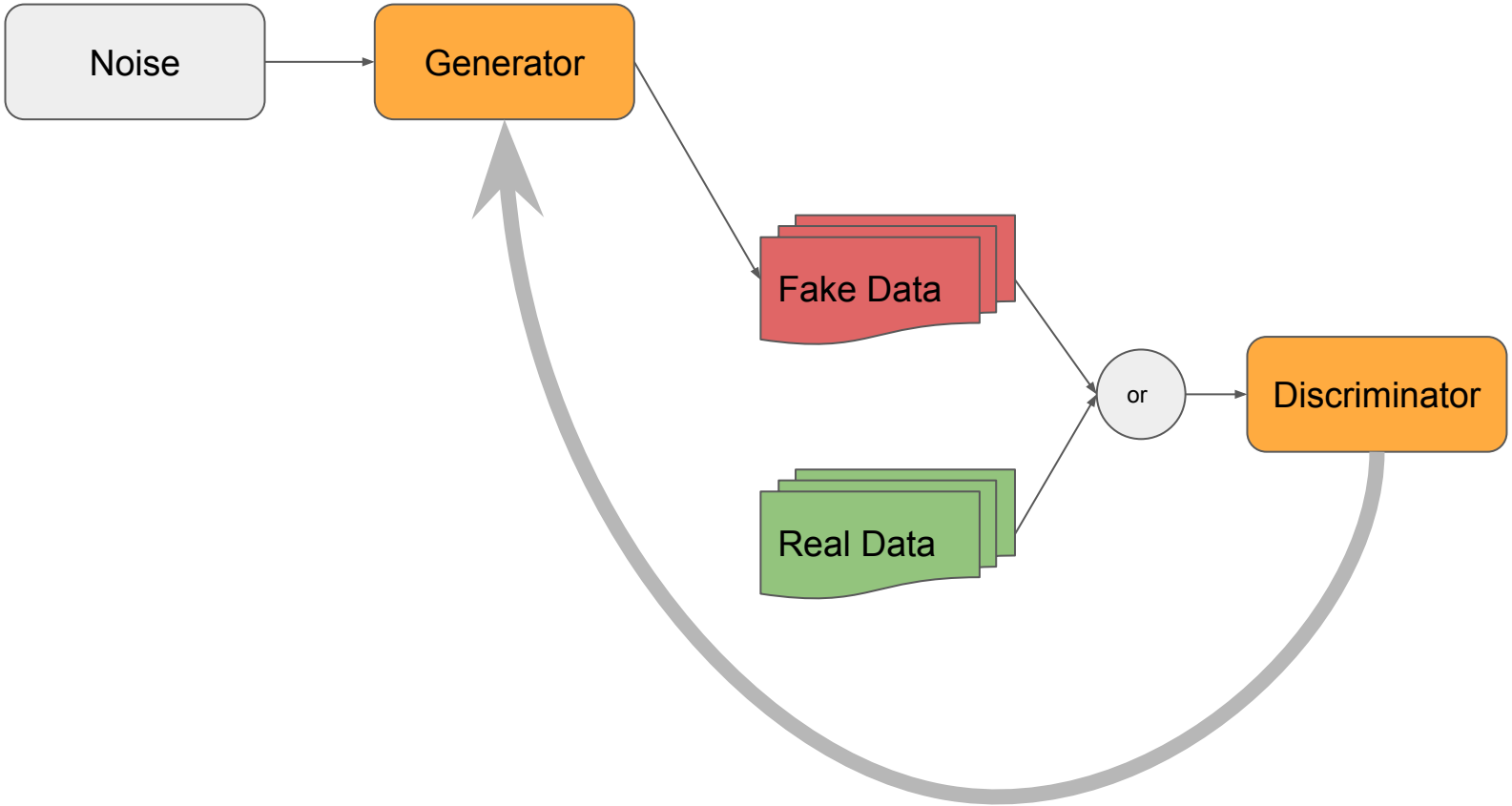

Noise
Generator
Fake Data
or
Discriminator
Real Data

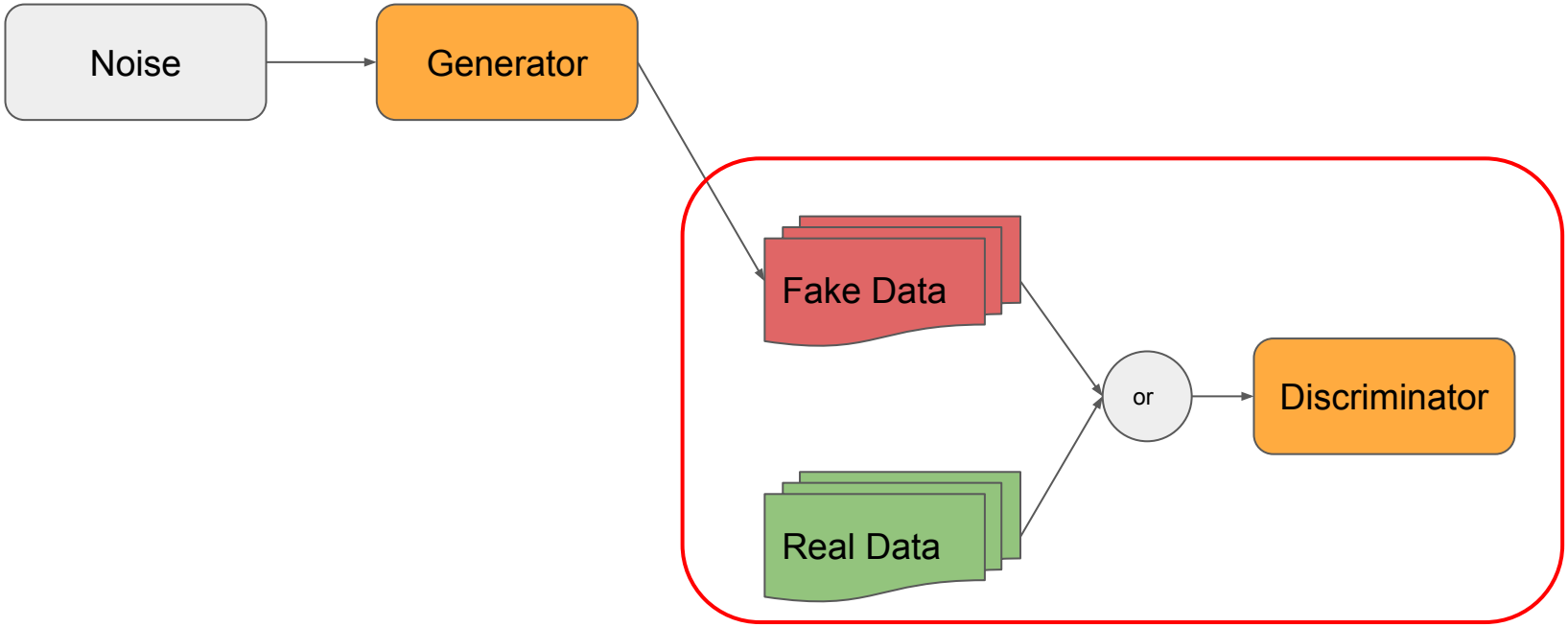

Noise
Generator
Fake Data
or
Discriminator
Real Data


Noise
Generator
Fake Data
Real Data
or
Discriminator

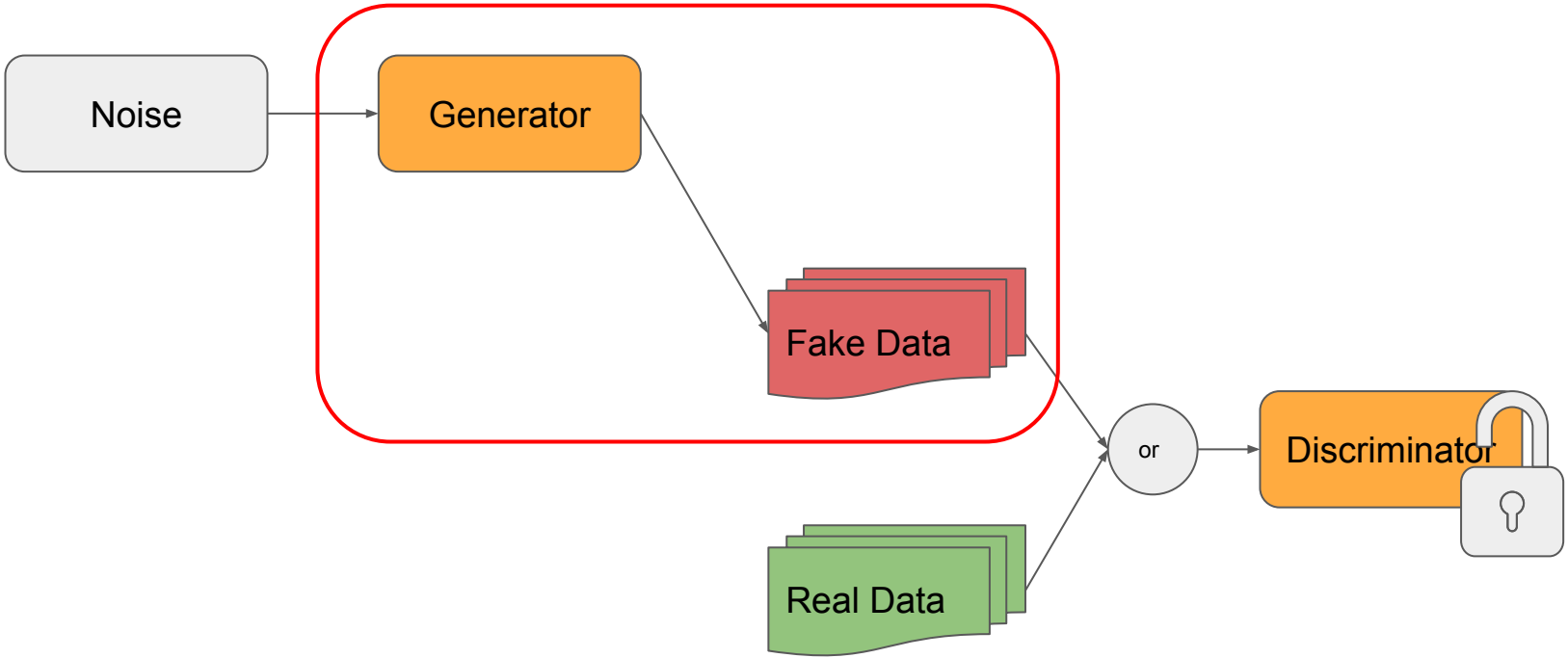

Noise
Generator
Fake Data
Real Data
or
Discriminator


Noise
Generator
Fake Data
Real Data
or
Discriminator

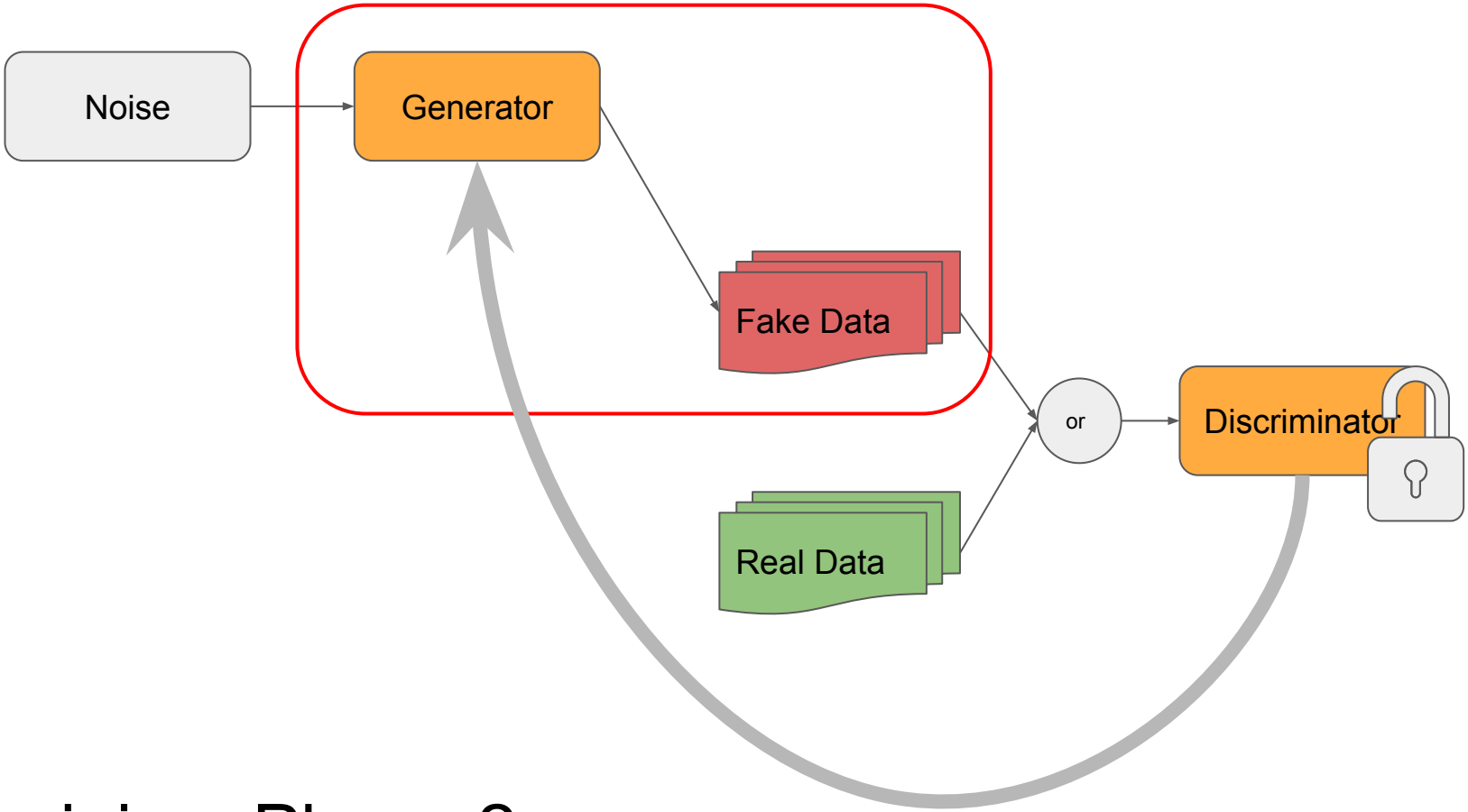

Noise
Generator
Fake Data
Real Data
or
Discriminator



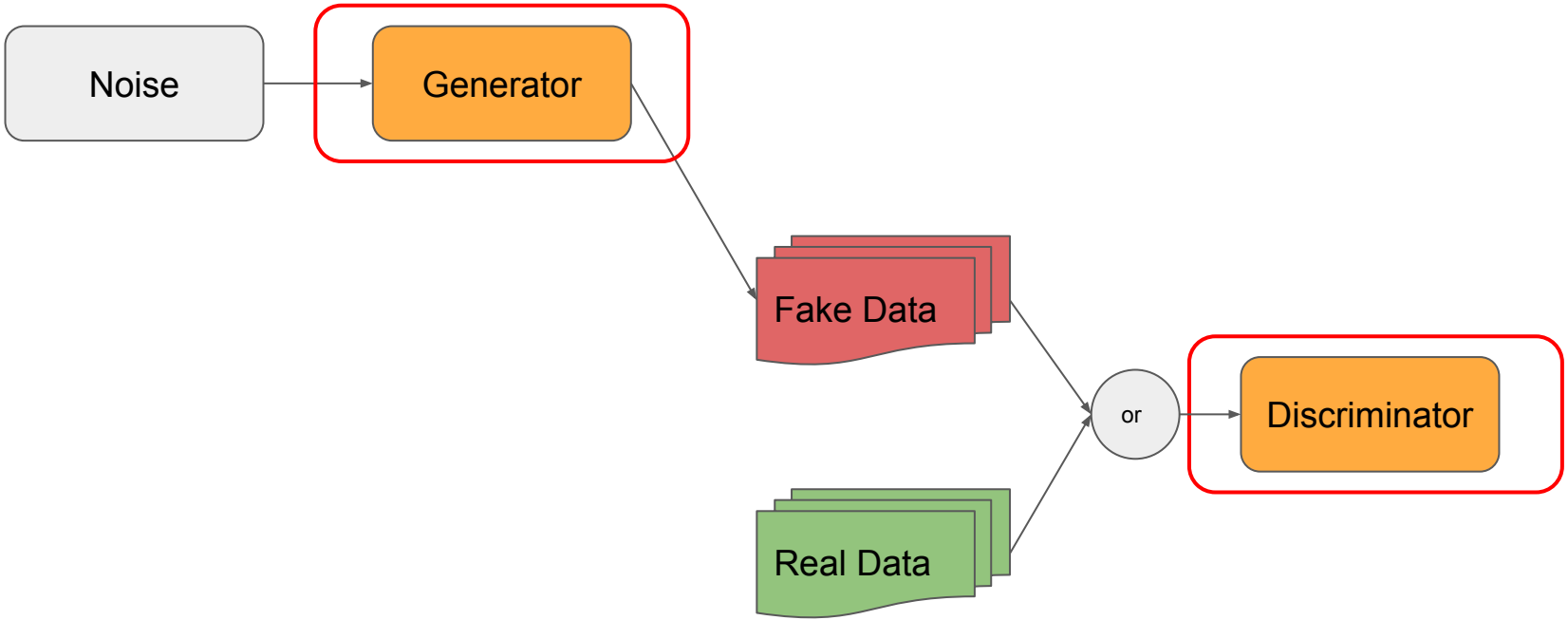
Training: Phase 1



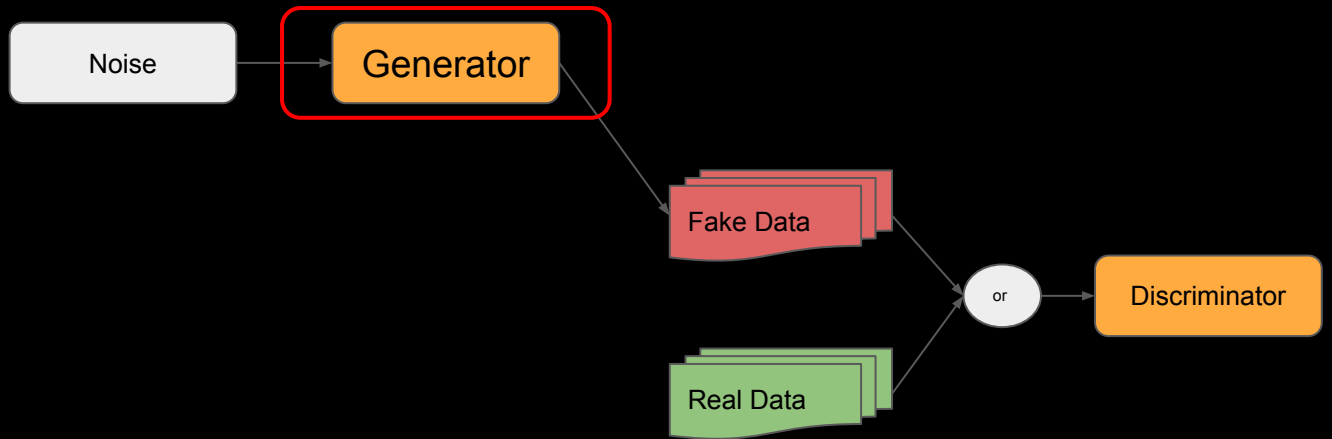
Training: Phase 2



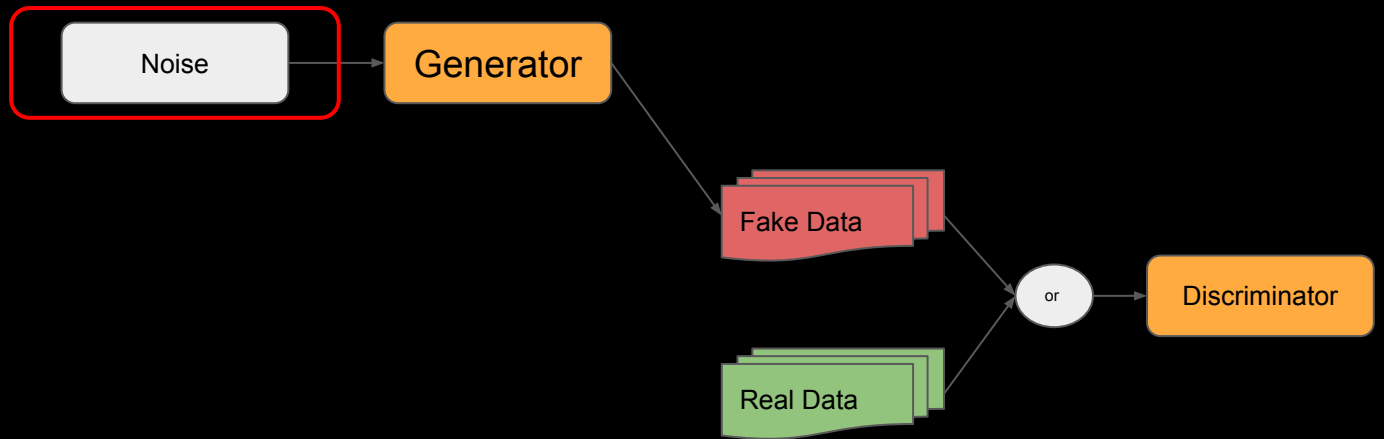
Training: Phase 2


Noise
Generator
Fake Data
Real Data
or
Discriminator

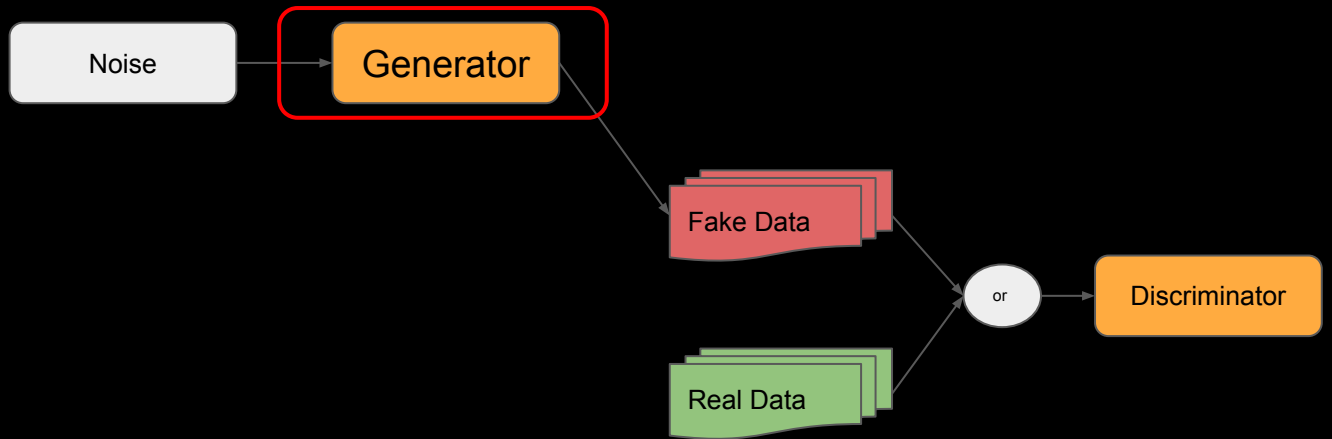
```
generator = keras.models.Sequential([
    keras.layers.Dense(64, activation="selu",
                       input_shape=[random_normal_dimensions]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```



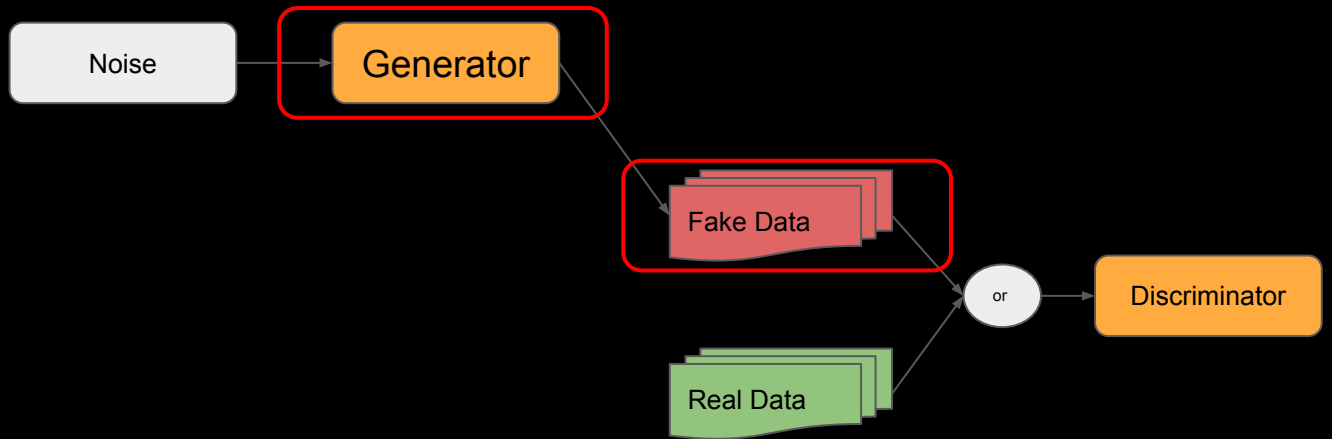
```
generator = keras.models.Sequential([
    keras.layers.Dense(64, activation="selu",
                       input_shape=[random_normal_dimensions]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```



```
generator = keras.models.Sequential([
    keras.layers.Dense(64, activation="selu",
                       input_shape=[random_normal_dimensions]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

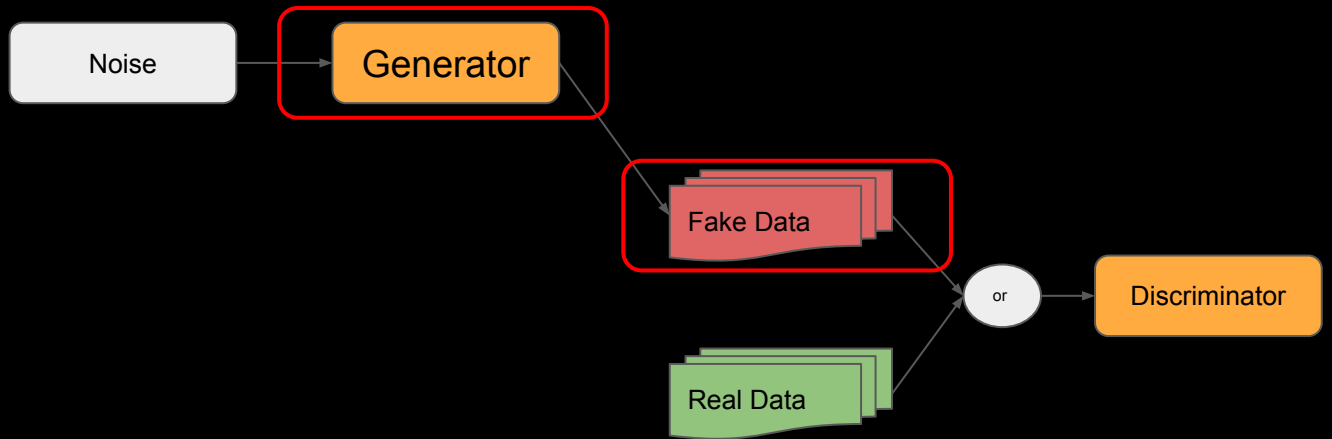


```
generator = keras.models.Sequential([
    keras.layers.Dense(64, activation="selu",
                       input_shape=[random_normal_dimensions]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

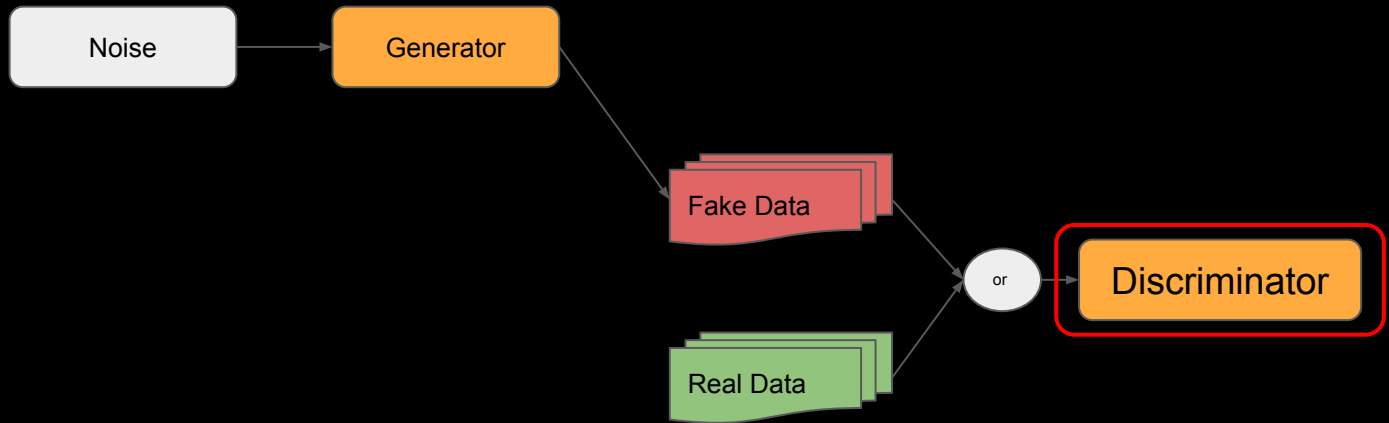


```
generator = keras.models.Sequential([
    keras.layers.Dense(64, activation="selu",
                       input_shape=[random_normal_dimensions]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(28 * 28, activation="sigmoid"),
    keras.layers.Reshape([28, 28])
])
```

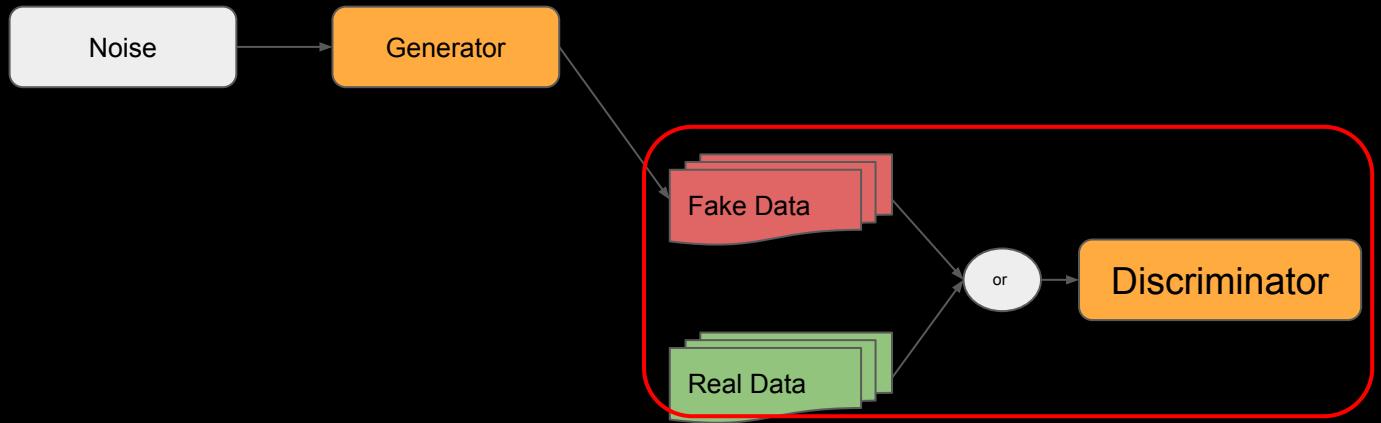
https://arxiv.org/abs/1706.02515



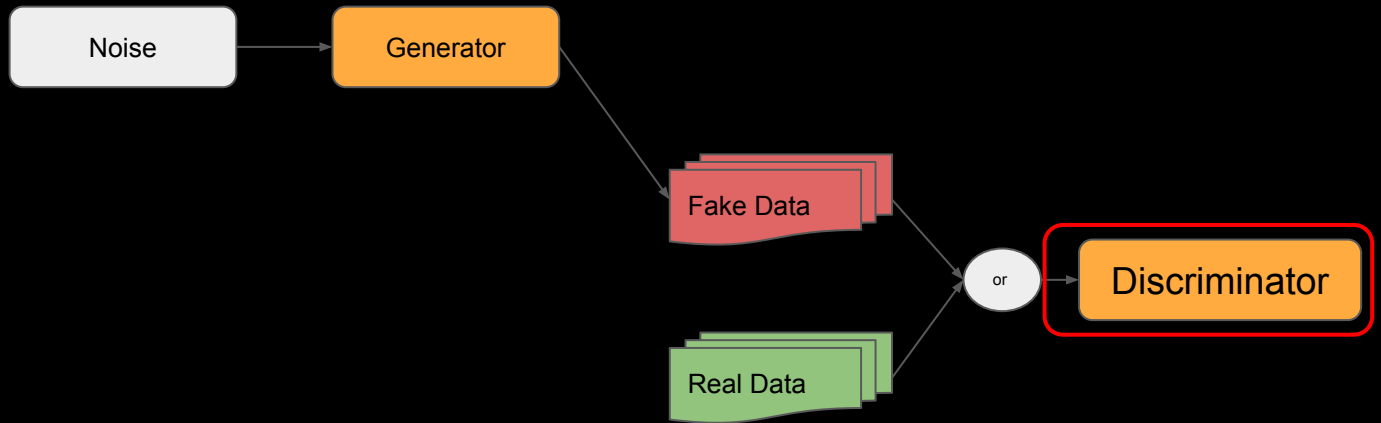
```
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
```



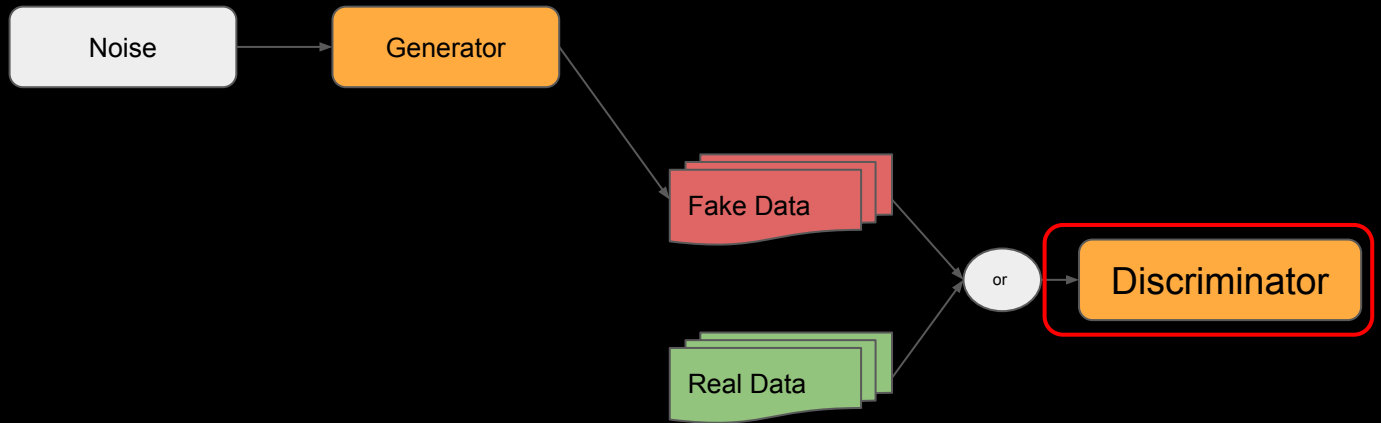
```
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
```



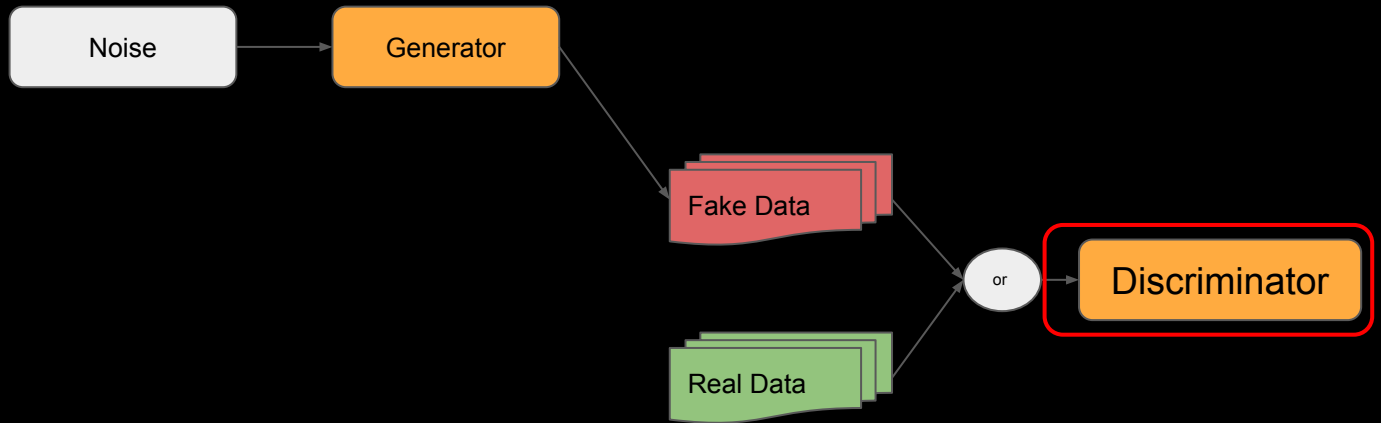
```
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
```



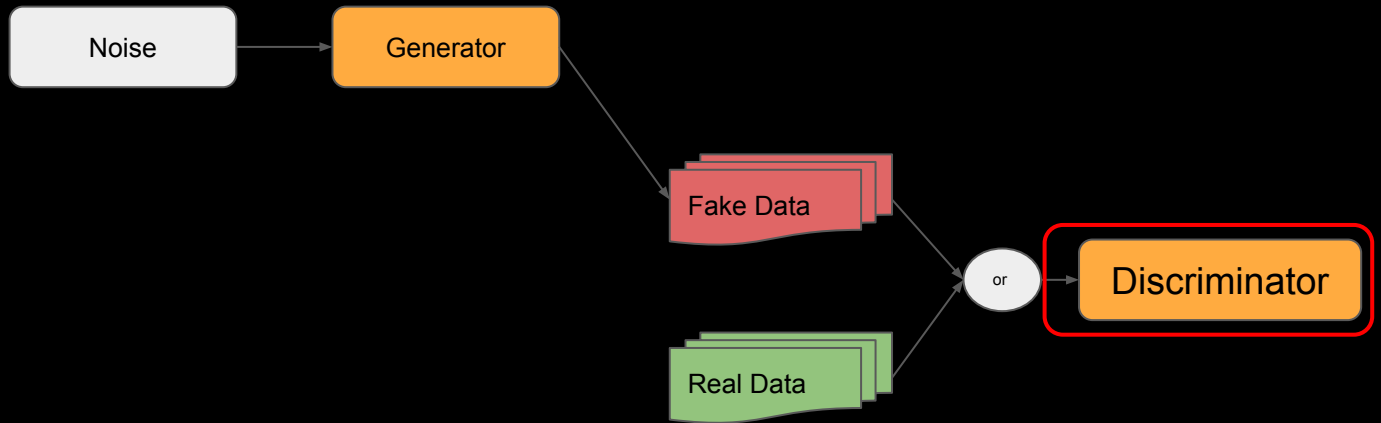
```
discriminator = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(128, activation="selu"),
    keras.layers.Dense(64, activation="selu"),
    keras.layers.Dense(1, activation="sigmoid")
])
```



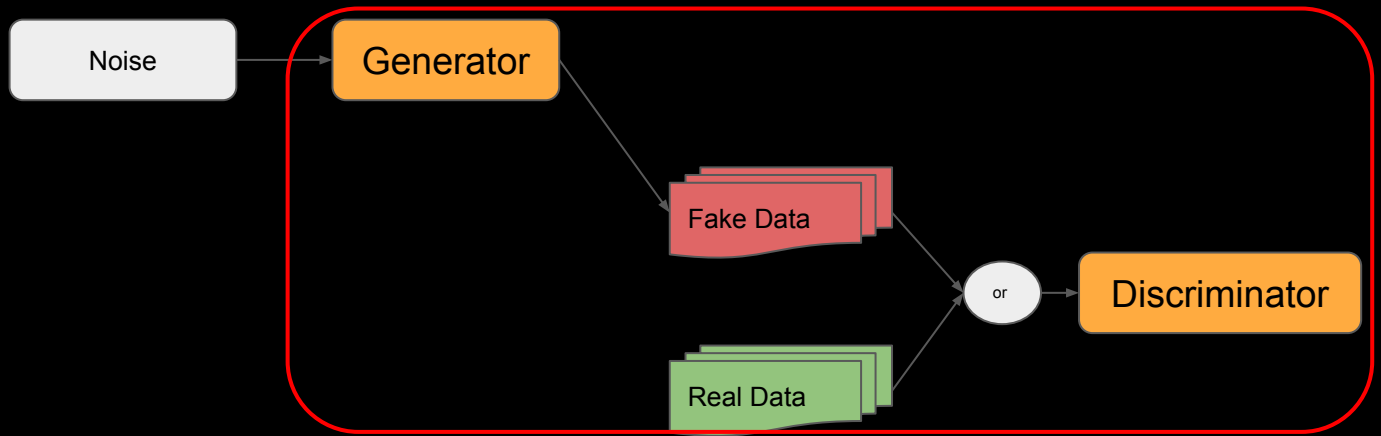
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
gan = keras.models.Sequential([generator, discriminator])
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



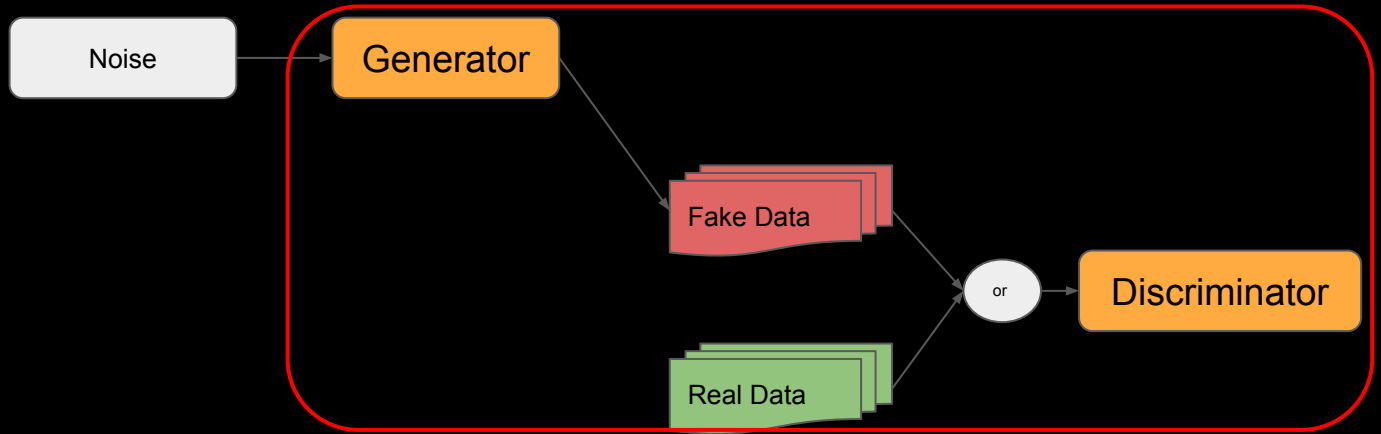
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
gan = keras.models.Sequential([generator, discriminator])
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

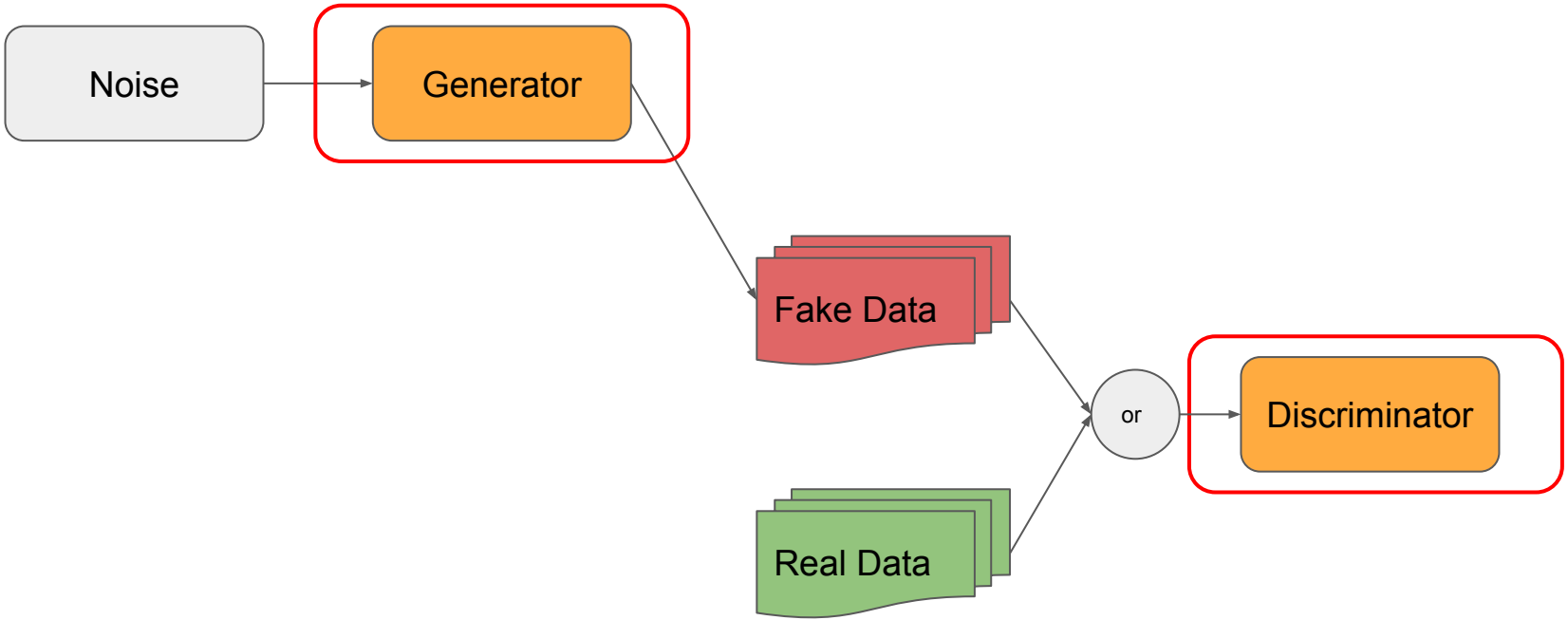


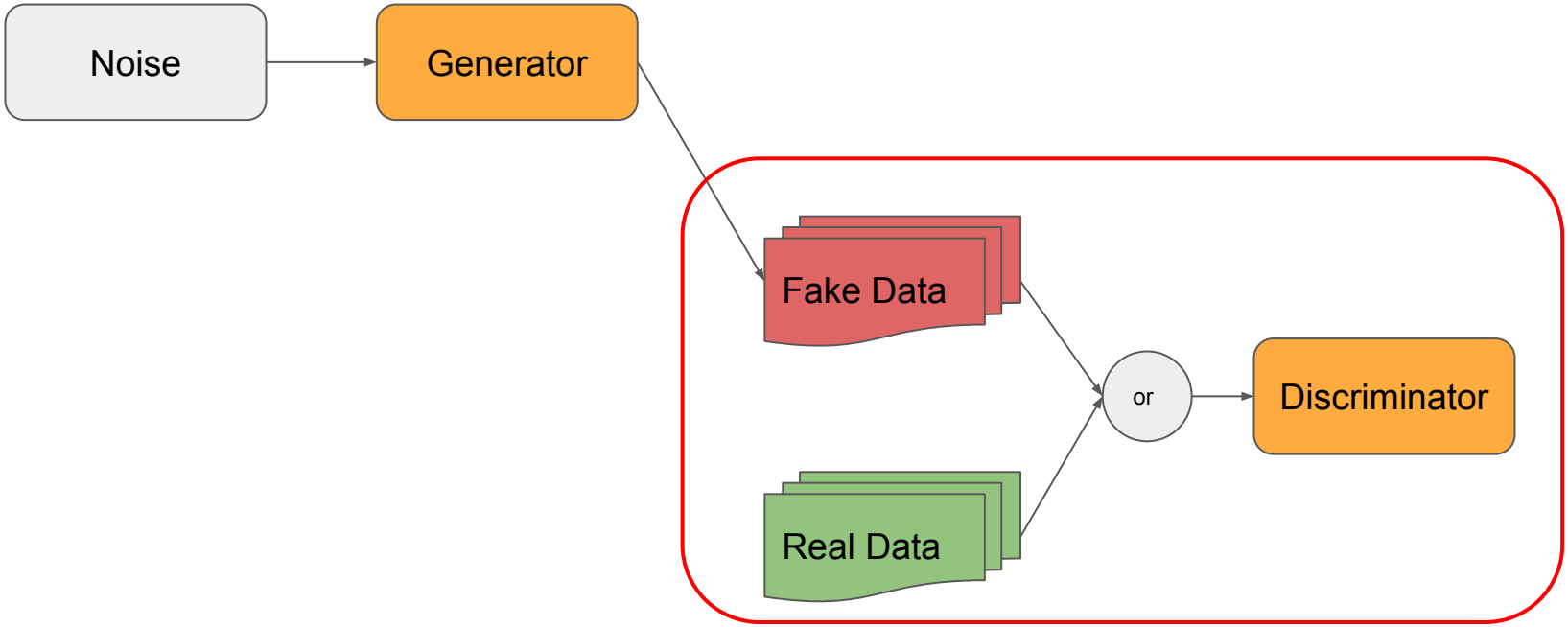
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
gan = keras.models.Sequential([generator, discriminator])
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



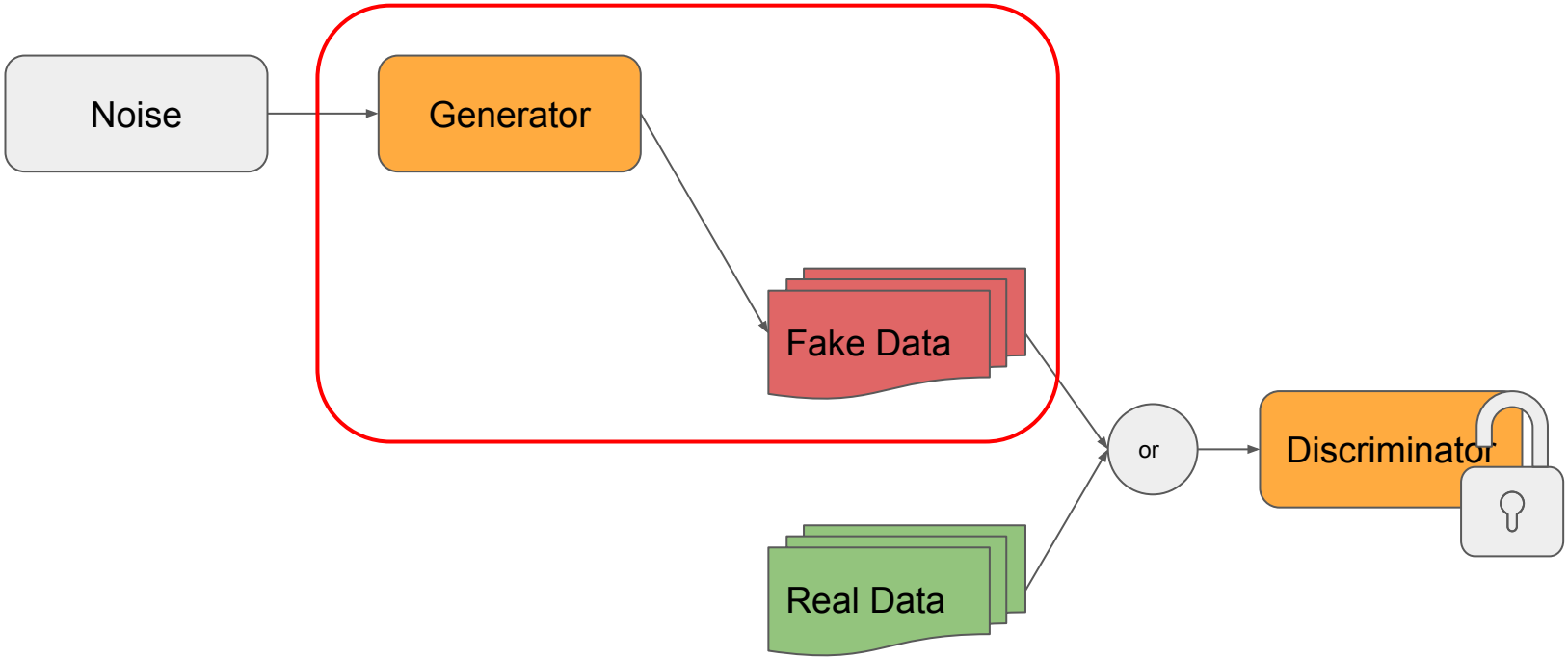
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
gan = keras.models.Sequential([generator, discriminator])
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



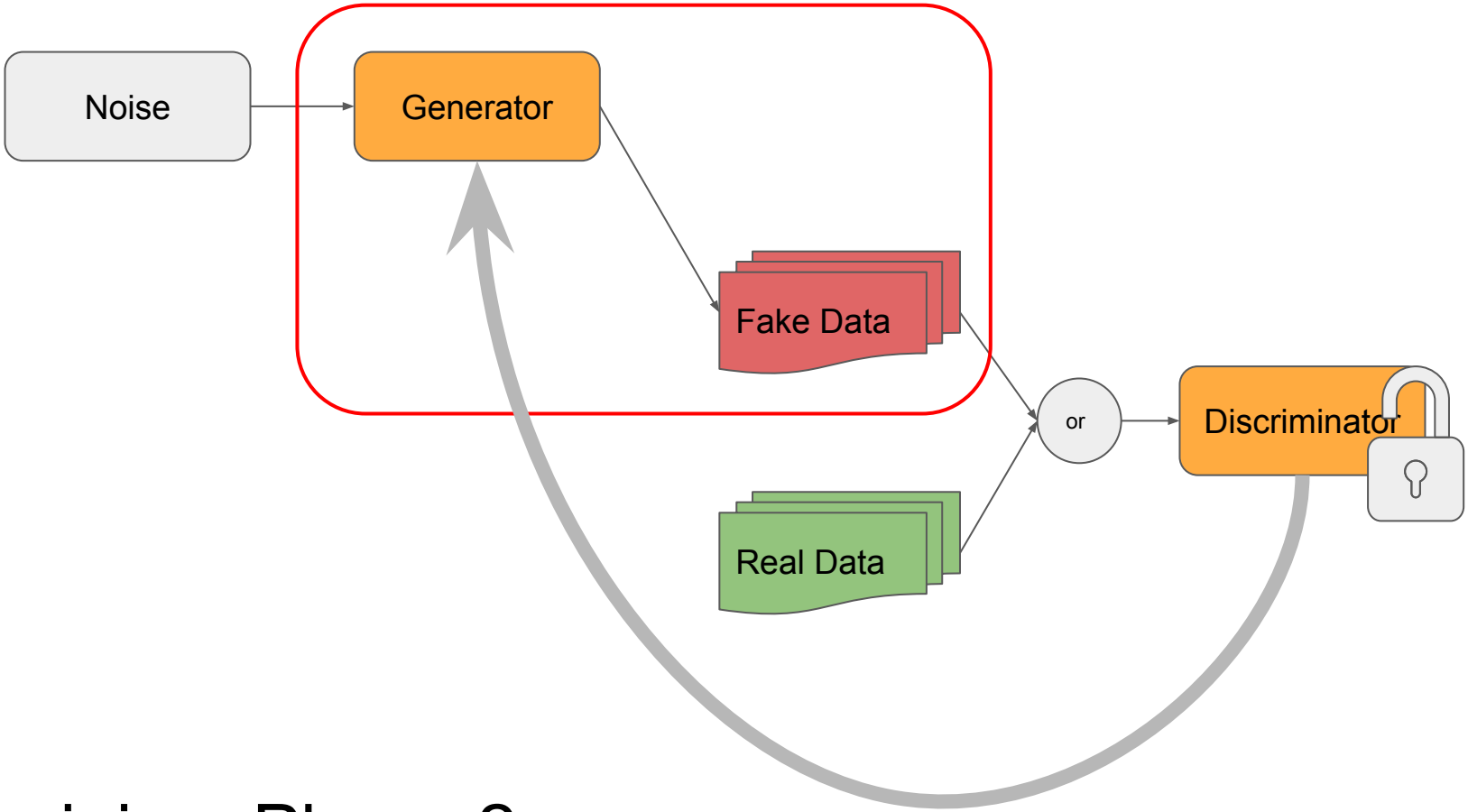

Noise
Generator
Fake Data
Real Data
or
Discriminator



Training: Phase 1



Training: Phase 2



Training: Phase 2

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

```
for epoch in range(n_epochs):
  for real_images in dataset:
    noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
    fake_images = generator(noise)
    mixed_images = tf.concat([fake_images, real_images], axis=0)
    discriminator_labels = tf.constant([[0.]] * batch_size +
                                       [[1.]] * batch_size)
    discriminator.trainable = True
    discriminator.train_on_batch(mixed_images, discriminator_labels)
```

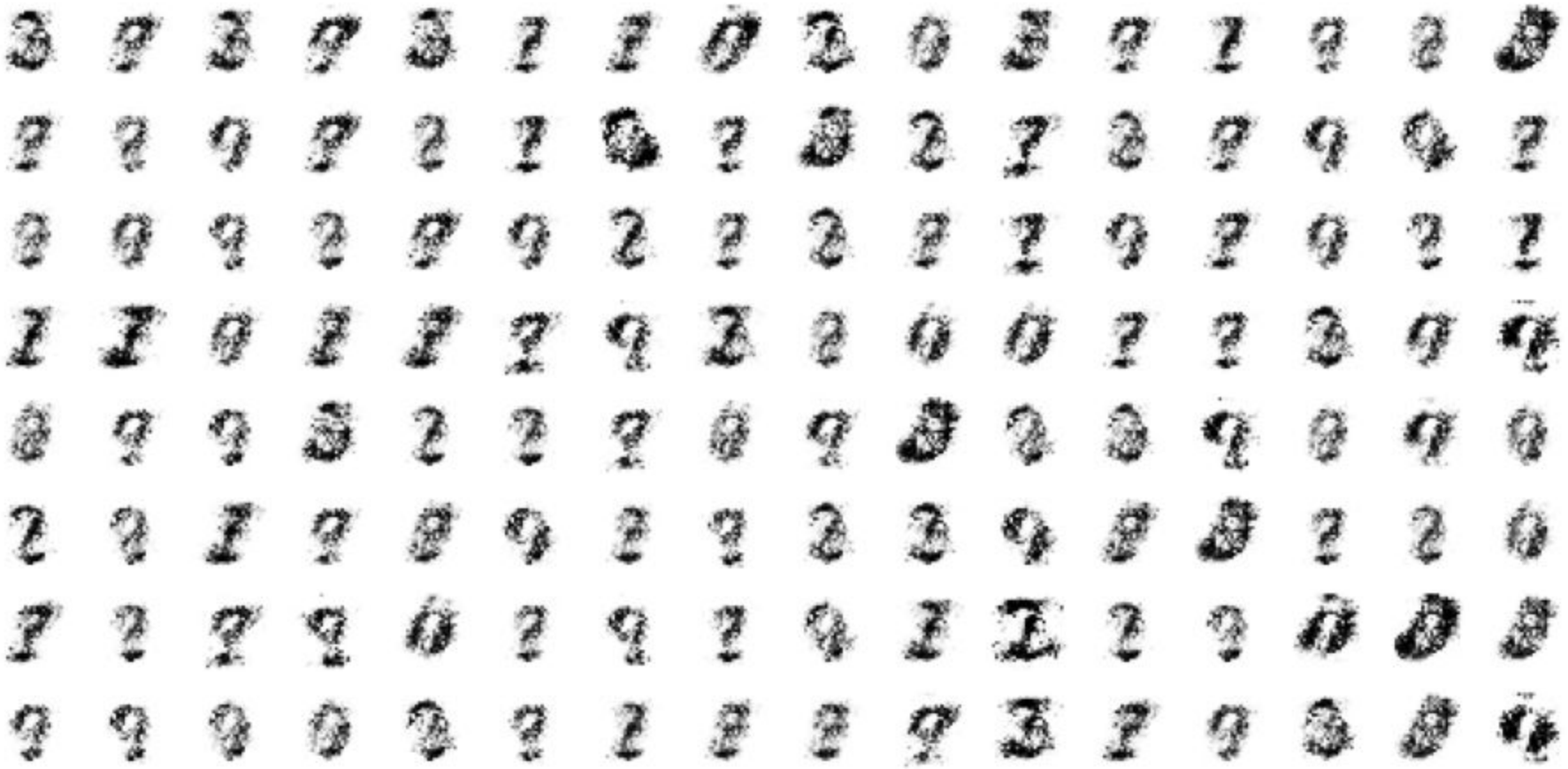
```
# Train the generator - PHASE 2
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
generator_labels = tf.constant([[1.]] * batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, generator_labels)
```

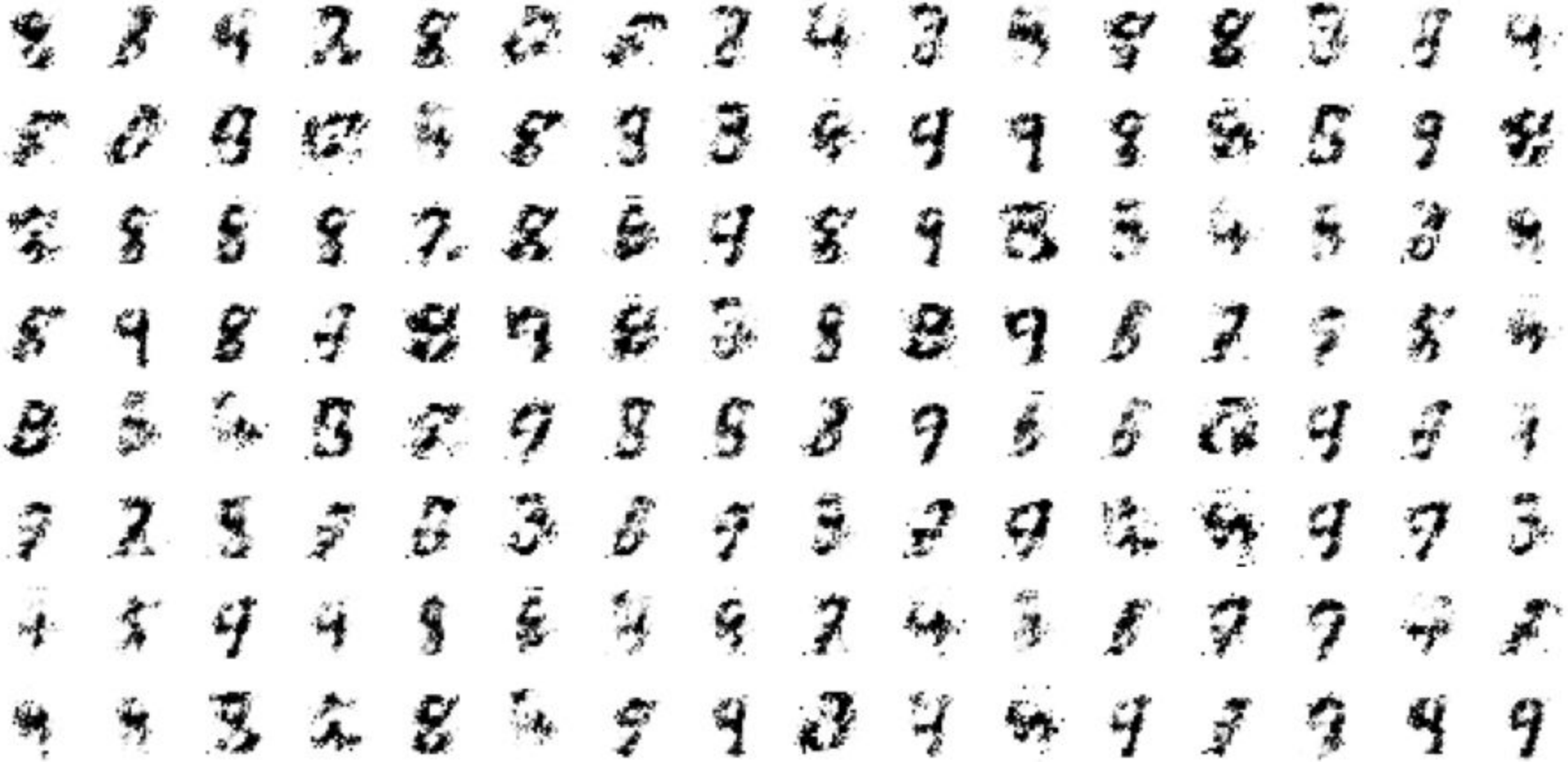
```
# Train the generator - PHASE 2
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
generator_labels = tf.constant([[1.]] * batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, generator_labels)
```

```
# Train the generator - PHASE 2
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
generator_labels = tf.constant([[1.]] * batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, generator_labels)
```

```
# Train the generator - PHASE 2
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
generator_labels = tf.constant([[1.]] * batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, generator_labels)
```

```
# Train the generator - PHASE 2
noise = tf.random.normal(shape=[batch_size, random_normal_dimensions])
generator_labels = tf.constant([[1.]] * batch_size)
discriminator.trainable = False
gan.train_on_batch(noise, generator_labels)
```





https://arxiv.org/pdf/1511.06434.pdf

UNSUPERVISED REPRESENTATION LEARNING WITH DEEP CONVOLUTIONAL GENERATIVE ADVERSARIAL NETWORKS

Alec Radford & Luke Metz
indico Research
Boston, MA
{alec,luke}@indico.io

Soumith Chintala
Facebook AI Research
New York, NY
soumith@fb.com

ABSTRACT

In recent years, supervised learning with convolutional networks (CNNs) has seen huge adoption in computer vision applications. Comparatively, unsupervised learning with CNNs has received less attention. In this work we hope to help bridge the gap between the success of CNNs for supervised learning and unsupervised learning. We introduce a class of CNNs called deep convolutional generative adversarial networks (DCGANs), that have certain architectural constraints, and demonstrate that they are a strong candidate for unsupervised learning. Training on various image datasets, we show convincing evidence that our deep convolutional adversarial pair learns a hierarchy of representations from object parts to scenes in both the generator and discriminator. Additionally, we use the learned features for novel tasks - demonstrating their applicability as general image representations.

```
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.BatchNormalization(),


    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
```

```
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
```

```
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.BatchNormalization(),


    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
```

```
generator = keras.models.Sequential([
    keras.layers.Dense(7 * 7 * 128, input_shape=[codings_size]),
    keras.layers.Reshape([7, 7, 128]),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2DTranspose(64, kernel_size=5, strides=2,
                                 padding="SAME", activation="selu"),
    keras.layers.BatchNormalization(),


    keras.layers.Conv2DTranspose(1, kernel_size=5, strides=2, padding="SAME",
                                 activation="tanh"),
])
```

```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),

    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),

    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
```

```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),

    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),

    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
```

```
discriminator = keras.models.Sequential([
    keras.layers.Conv2D(64, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2),
                        input_shape=[28, 28, 1]),
    keras.layers.Dropout(0.4),

    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
                        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),

    keras.layers.Flatten(),
    keras.layers.Dense(1, activation="sigmoid")
])
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU



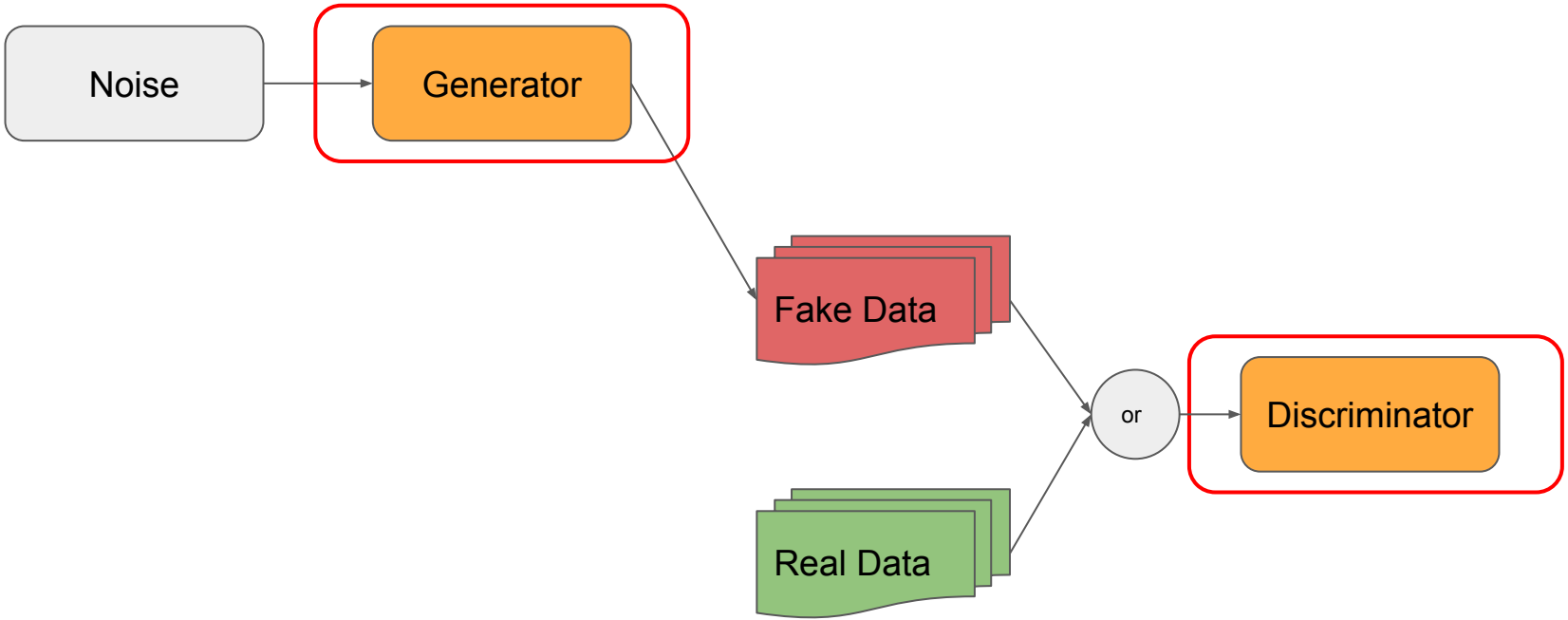


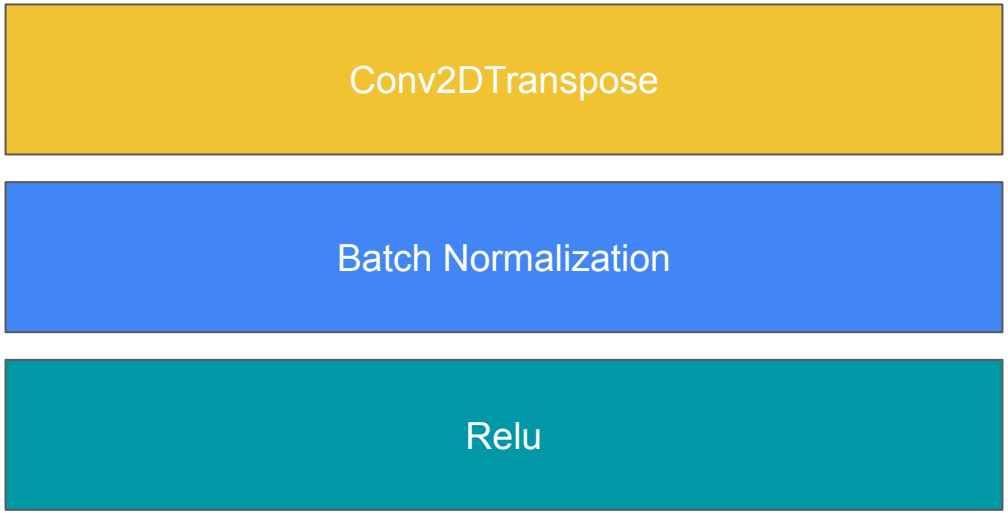


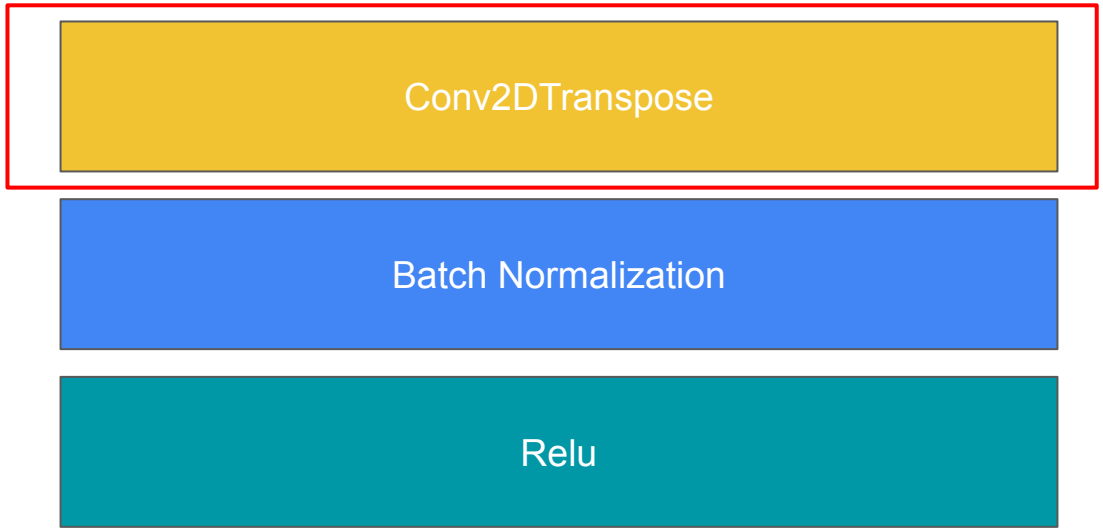


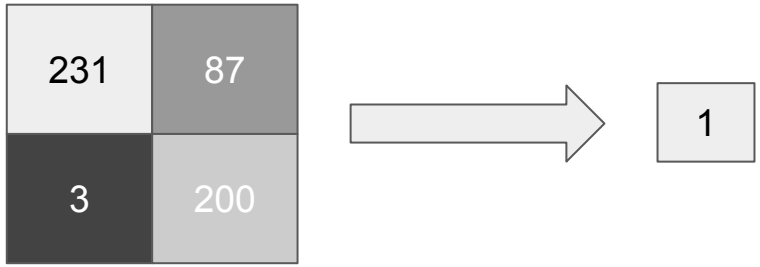


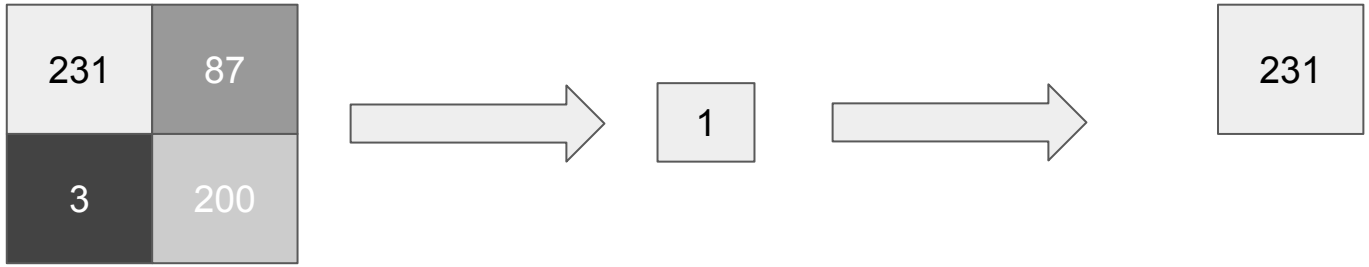


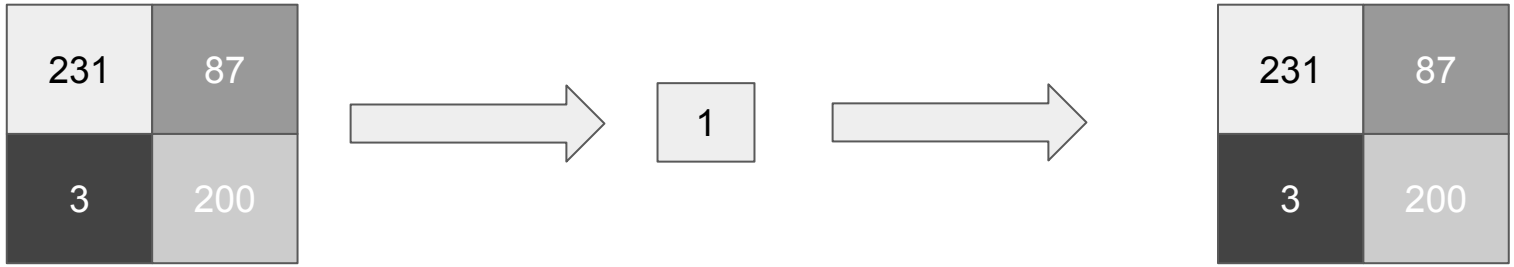


Noise
Generator
Fake Data
Real Data
or
Discriminator


Conv2DTranspose
Batch Normalization
Relu

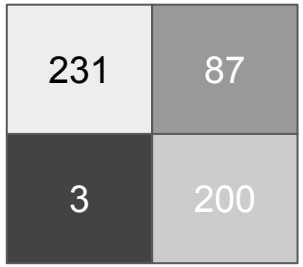

Conv2DTranspose
Batch Normalization
Relu


231
87
3
200
1


231
87
3
200
1
231


231
87
3
200
1
231
87
3
200

Strides = 2,2

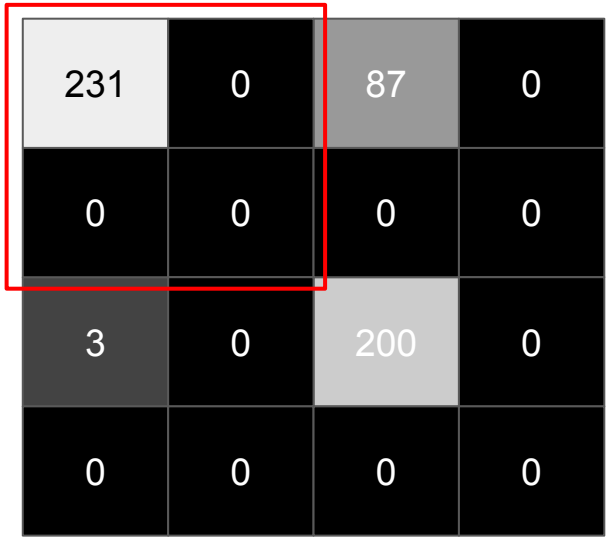


Strides = 2,2

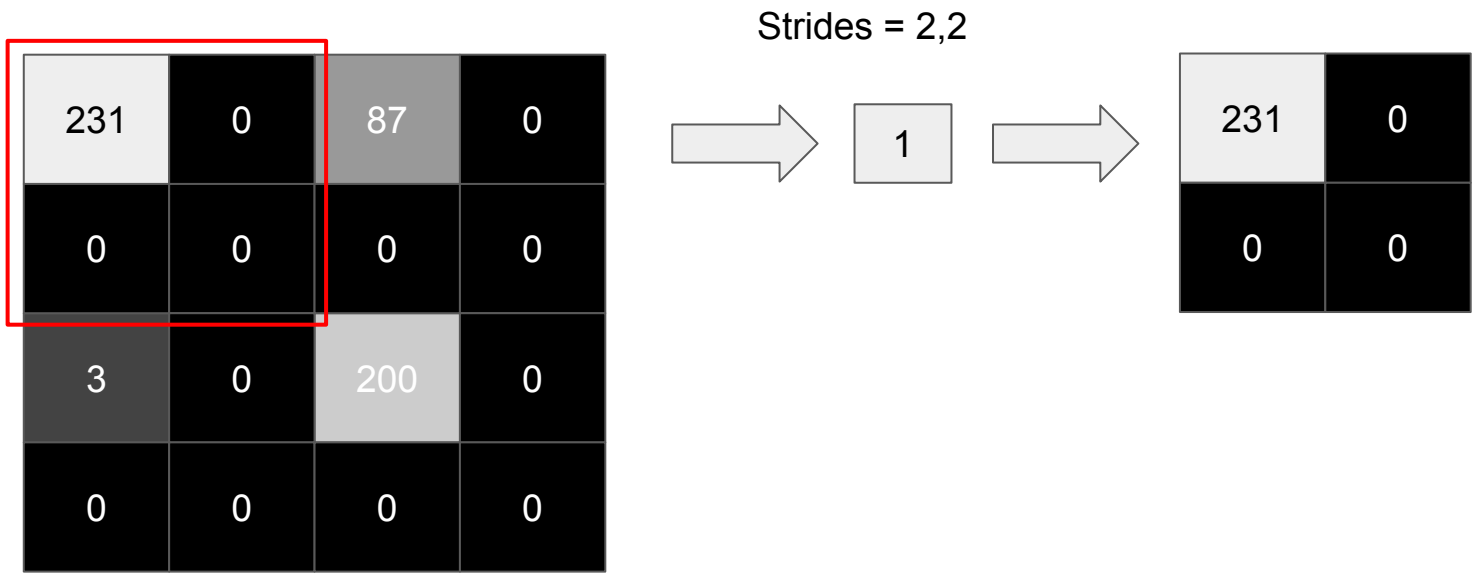
231	0	87	0
0	0	0	0
3	0	200	0
0	0	0	0

1

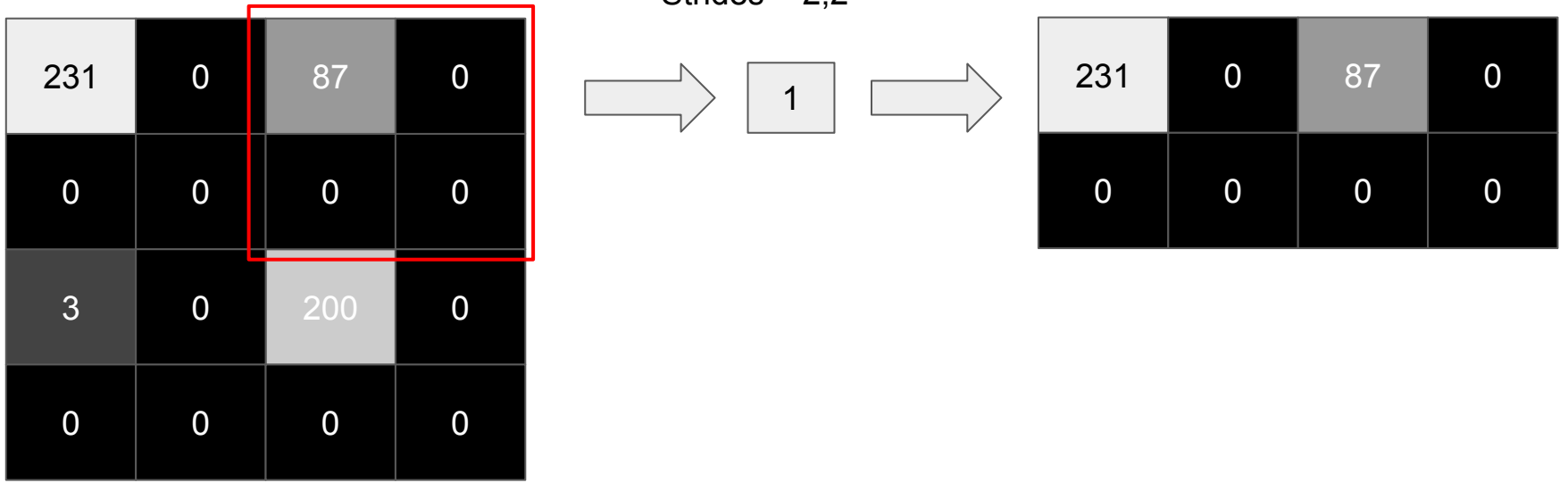
Strides = 2,2



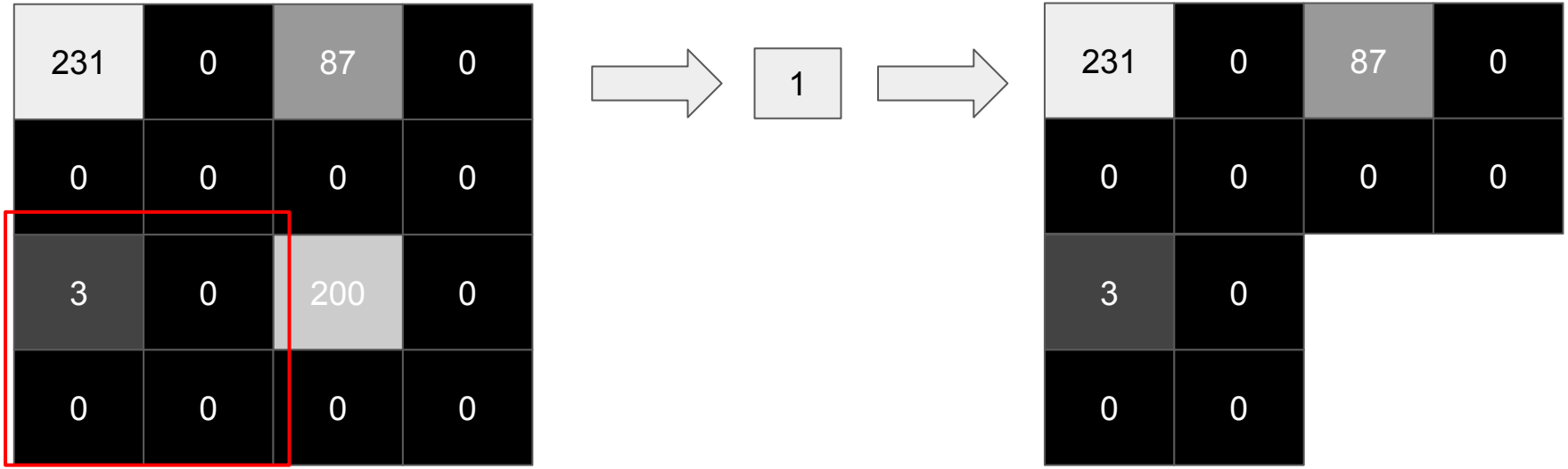
1


Strides = 2,2
231
0
87
0
0
0
0
0
3
0
200
0
0
0
0
0
1
231
0
0
0

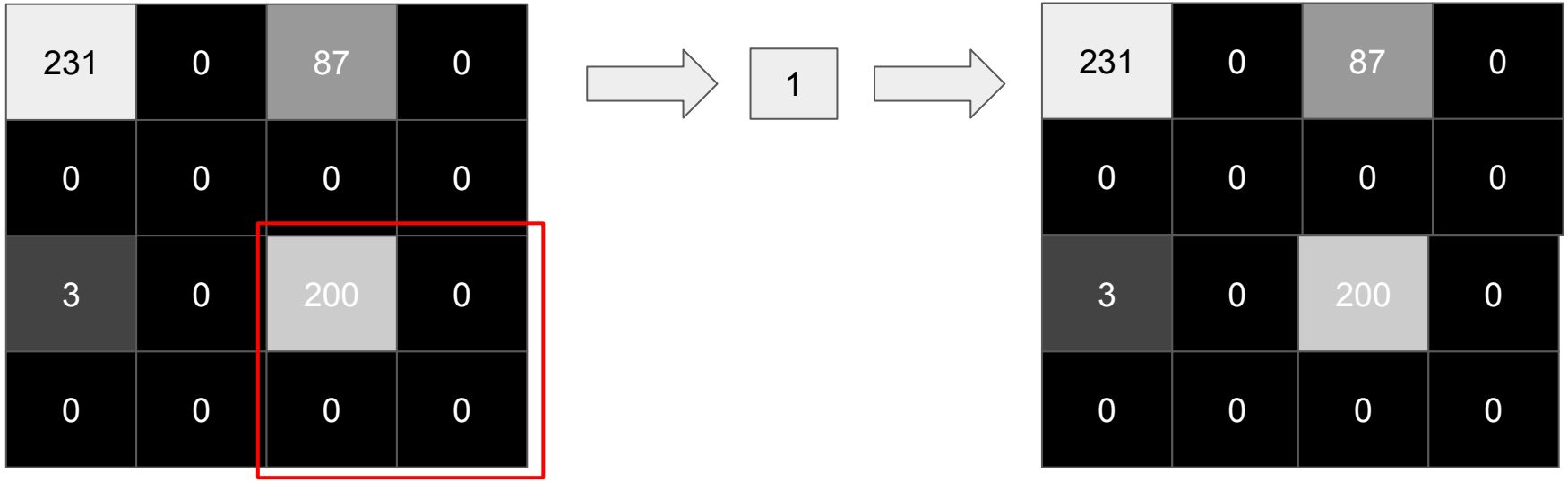
Strides = 2,2

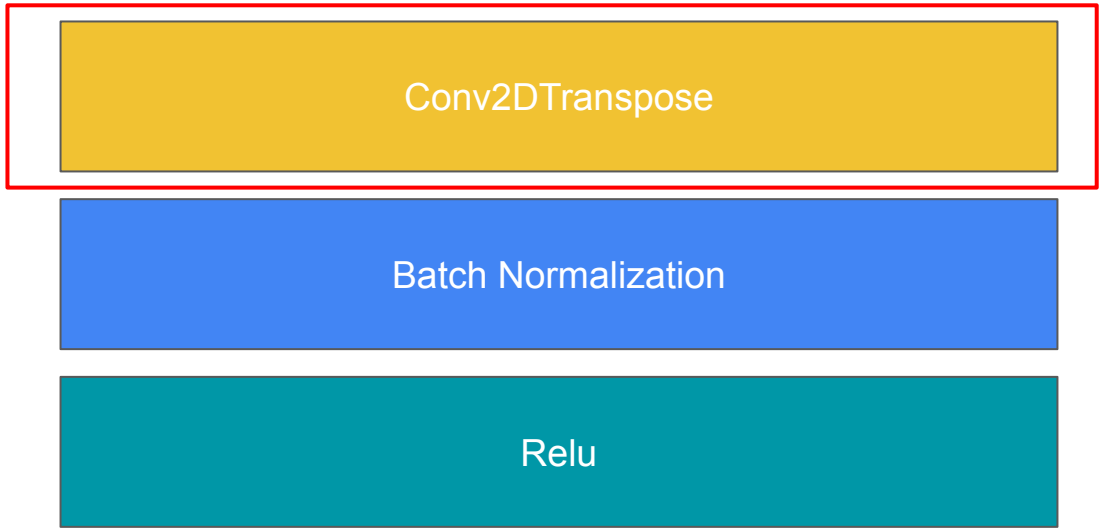


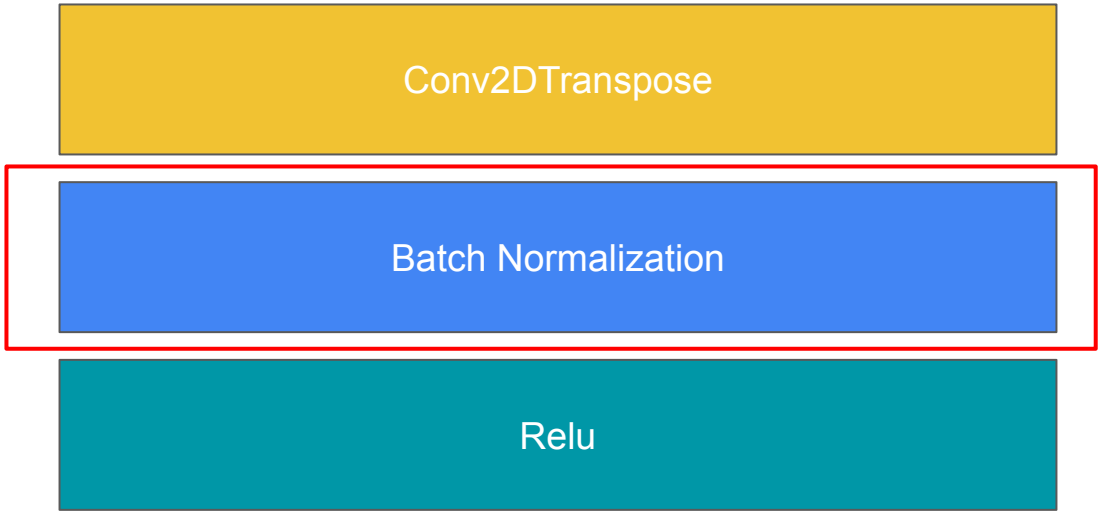
Strides = 2,2

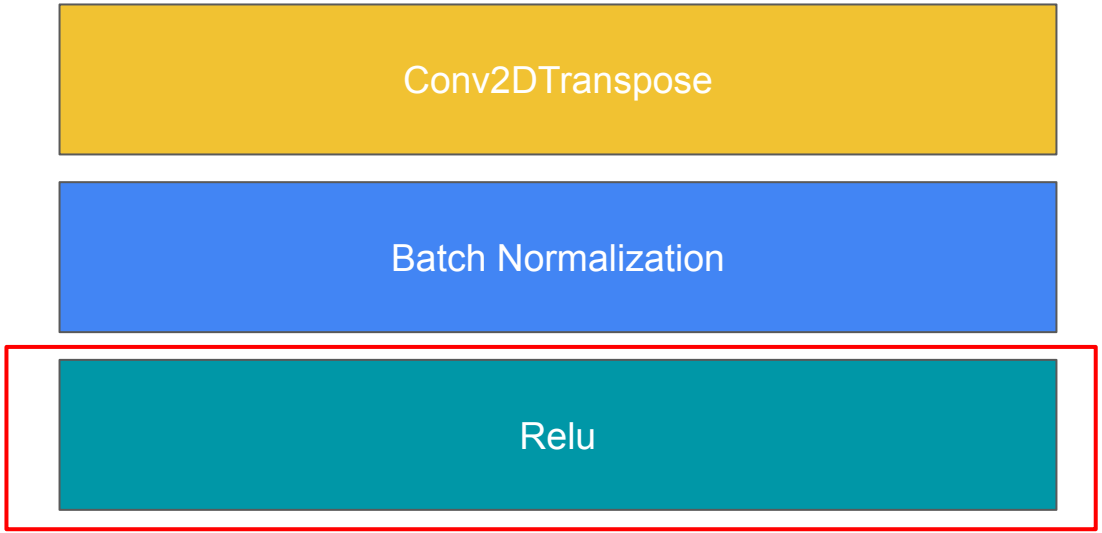


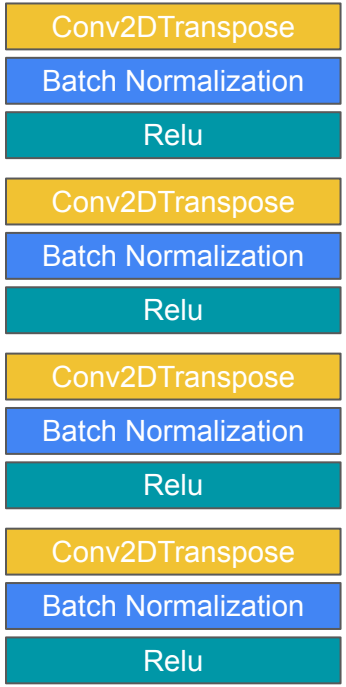
Strides = 2,2

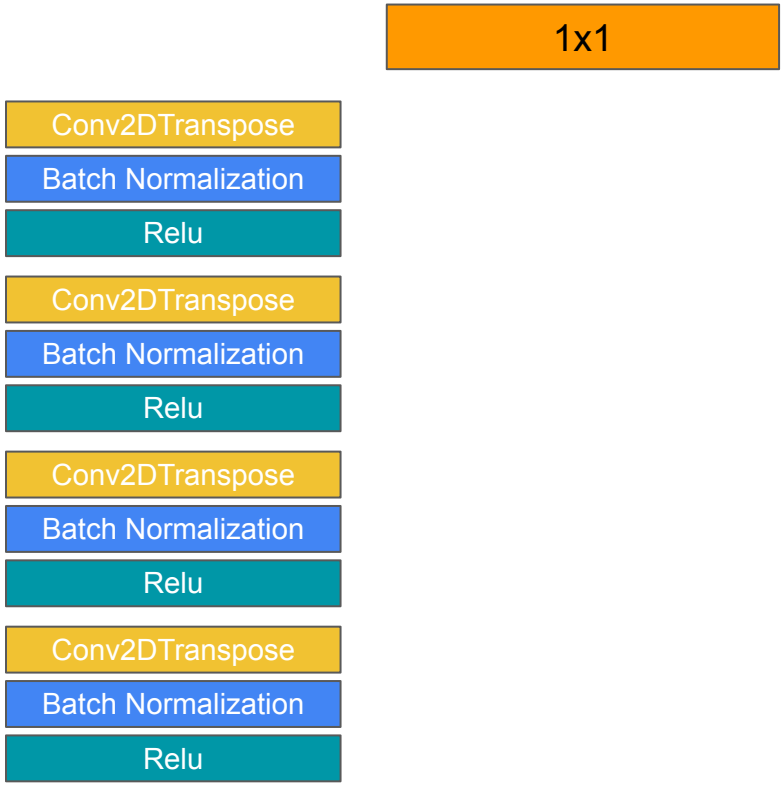


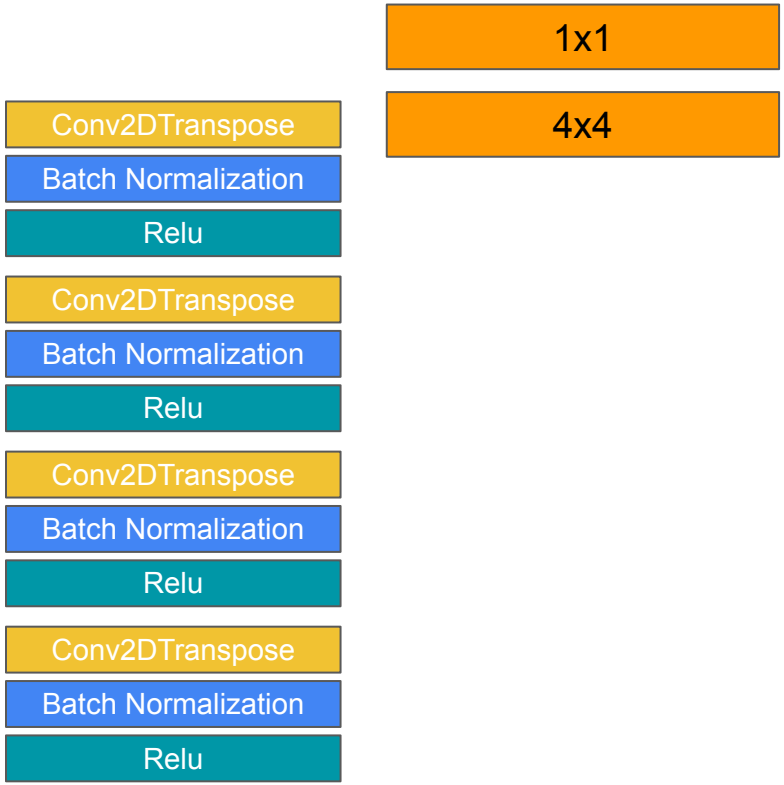

Conv2DTranspose
Batch Normalization
Relu

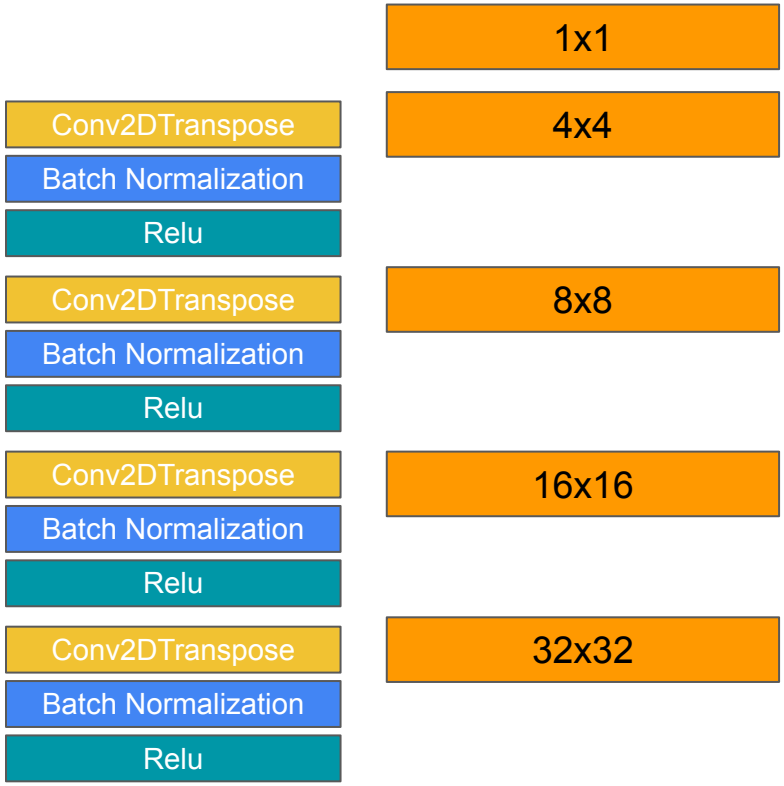

Conv2DTranspose
Batch Normalization
Relu

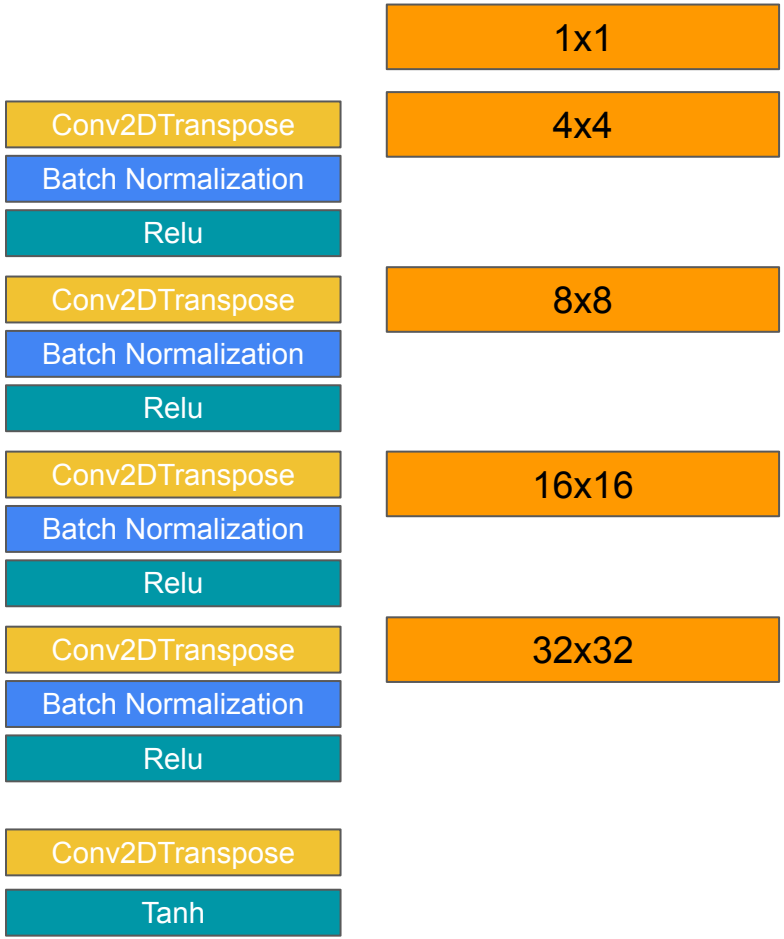

Conv2DTranspose
Batch Normalization
Relu

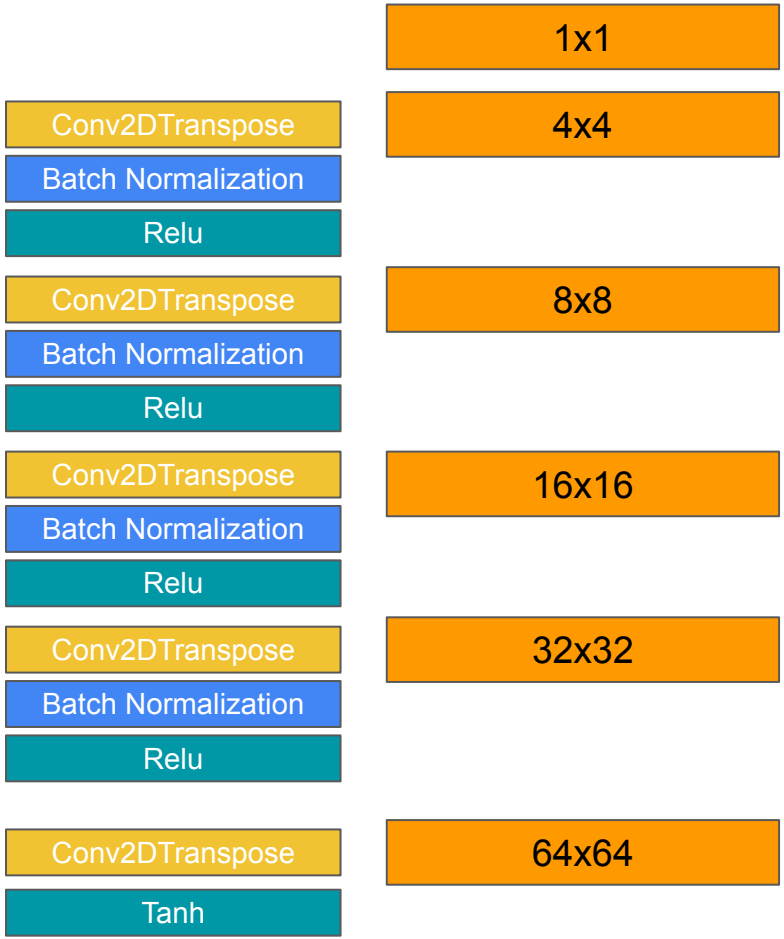

Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu


1x1
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu


1x1
4x4
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu
Conv2DTranspose
Batch Normalization
Relu


1x1
Conv2DTranspose
Batch Normalization
Relu
4x4
Conv2DTranspose
Batch Normalization
Relu
8x8
Conv2DTranspose
Batch Normalization
Relu
16x16
Conv2DTranspose
Batch Normalization
Relu
32x32

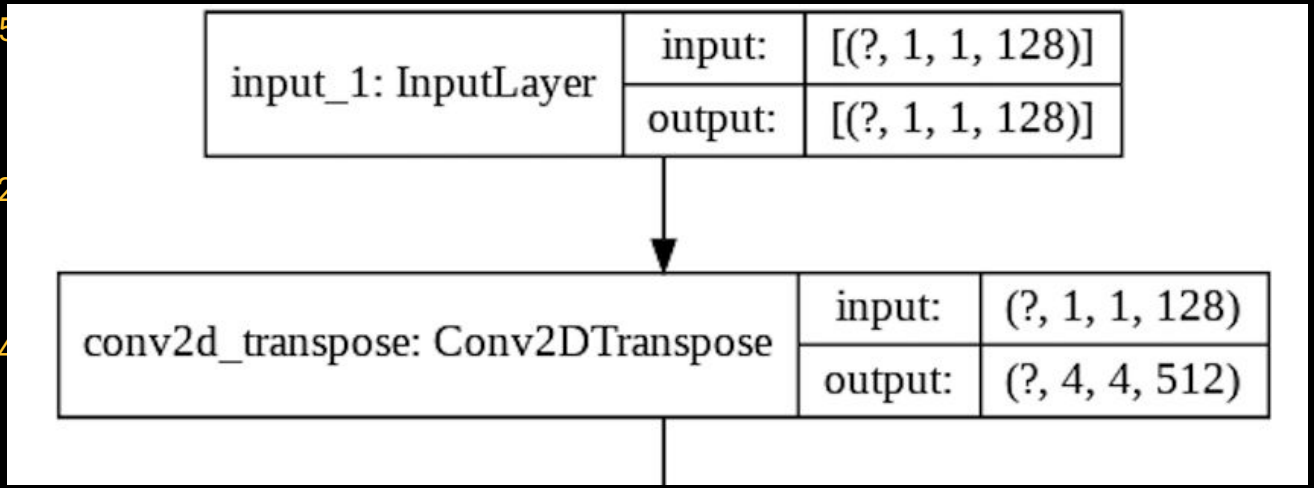

1x1
Conv2DTranspose
Batch Normalization
Relu
4x4
Conv2DTranspose
Batch Normalization
Relu
8x8
Conv2DTranspose
Batch Normalization
Relu
16x16
Conv2DTranspose
Batch Normalization
Relu
32x32
Conv2DTranspose
Tanh


1x1
Conv2DTranspose
Batch Normalization
Relu
4x4
Conv2DTranspose
Batch Normalization
Relu
8x8
Conv2DTranspose
Batch Normalization
Relu
16x16
Conv2DTranspose
Batch Normalization
Relu
32x32
Conv2DTranspose
Tanh
64x64

```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides=2, padding='same')(x)
outputs = layers.Activation('tanh')(x)
```

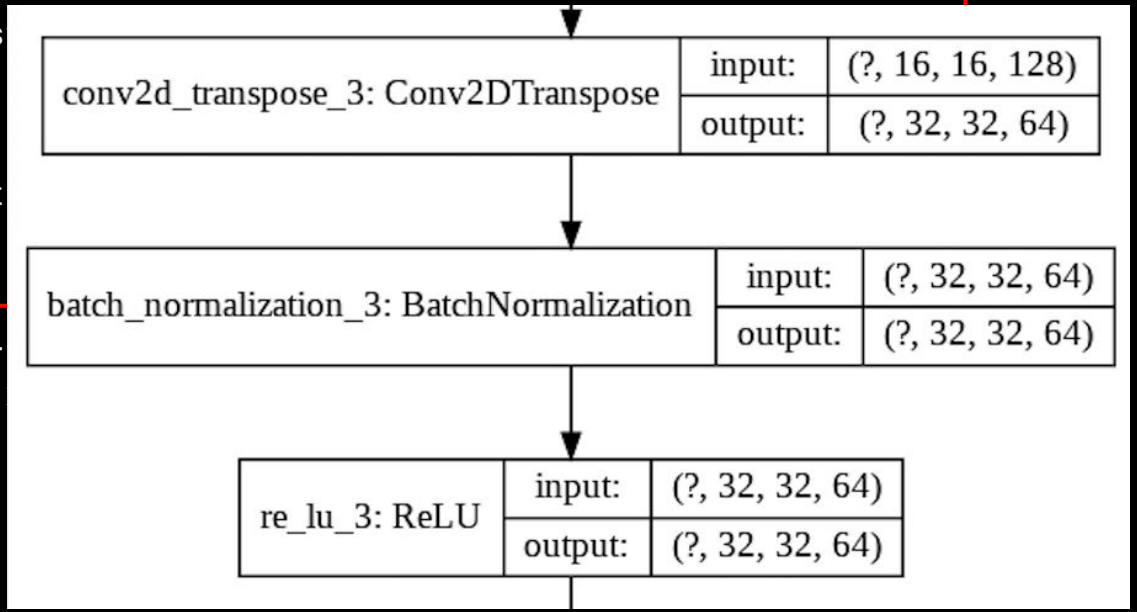
```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides=2, padding='same')(x)
outputs = layers.Activation('tanh')(x)
```

```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(25
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(12
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides=2, padding='same')(x)
outputs = layers.Activation('tanh')(x)
```



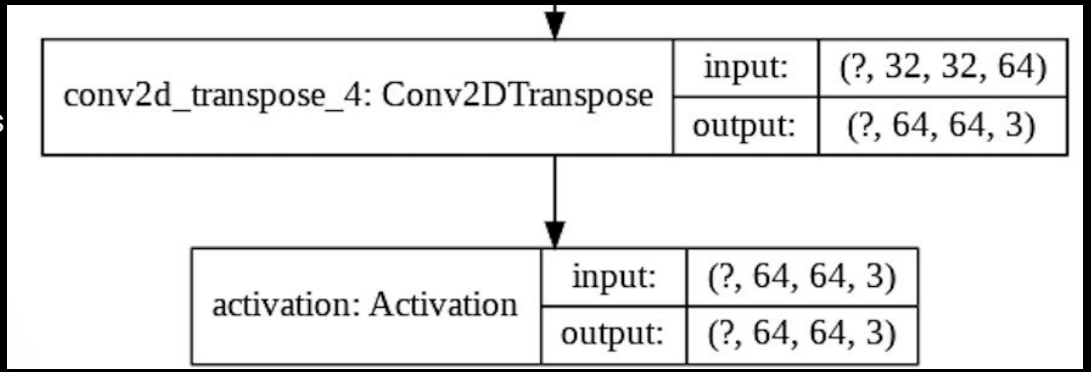
```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides=2, padding='same')(x)
outputs = layers.Activation('tanh')(x)
```

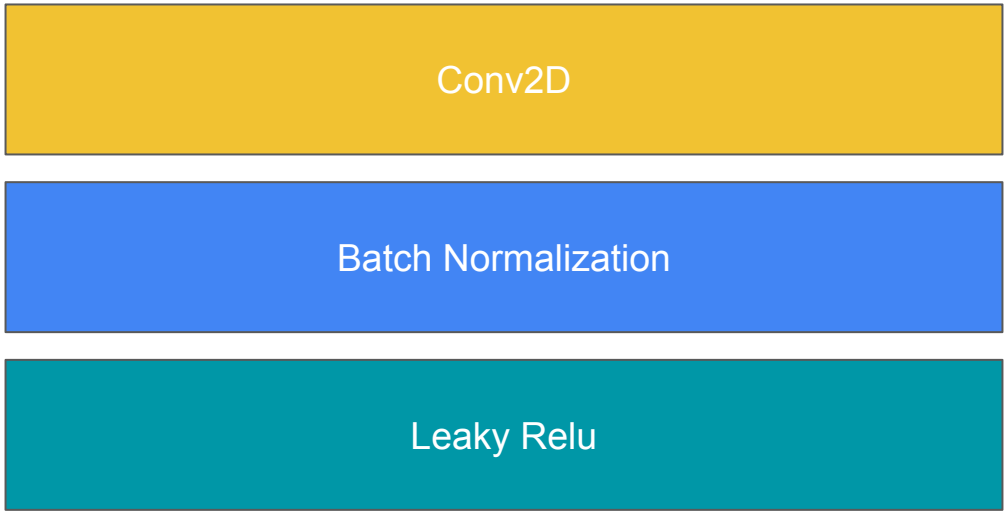
```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, s
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, st
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, str
outputs = layers.Activation('tanh')(
```

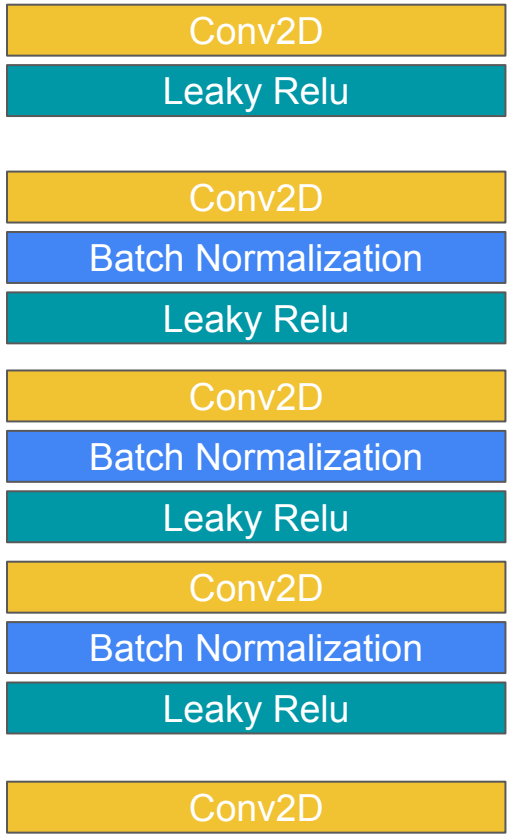


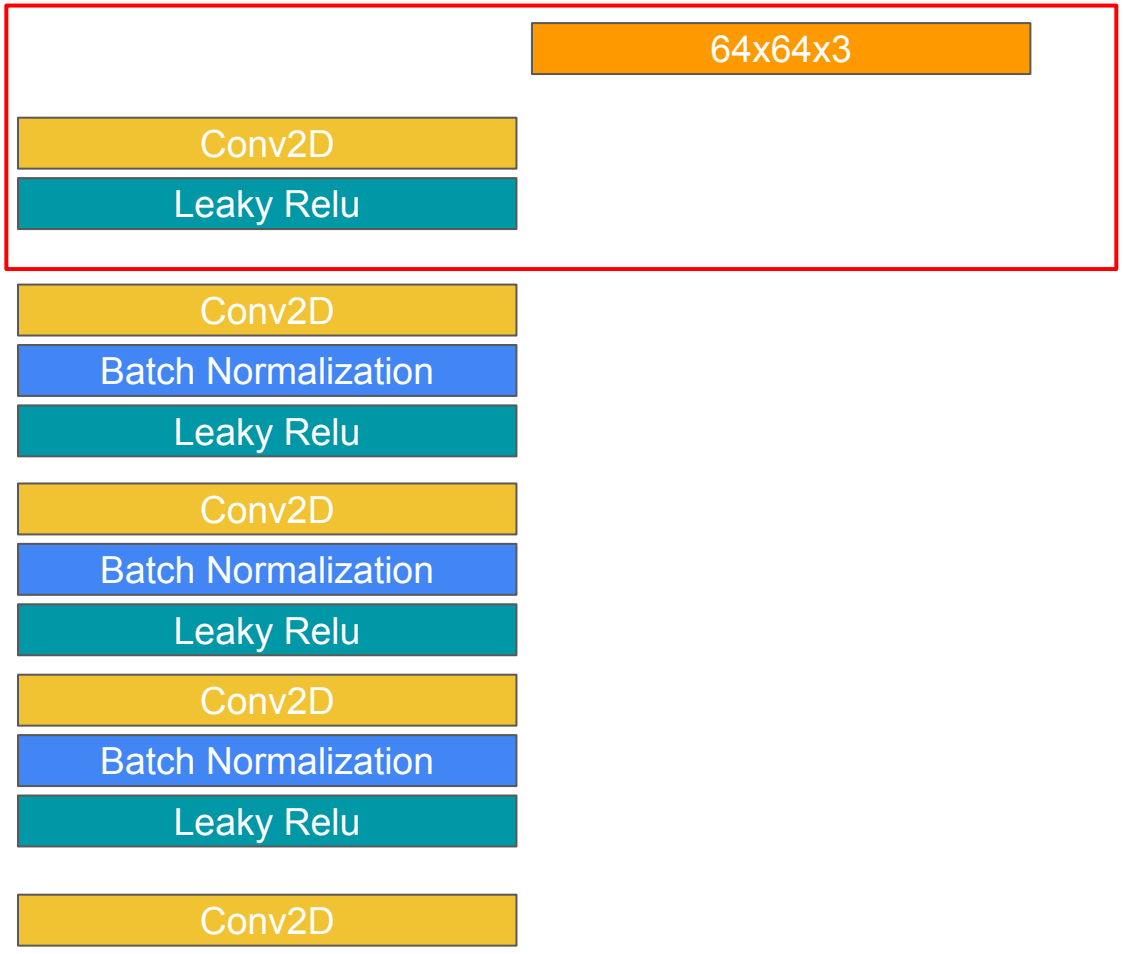
```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides=2, padding='same')(x)
outputs = layers.Activation('tanh')(x)
```

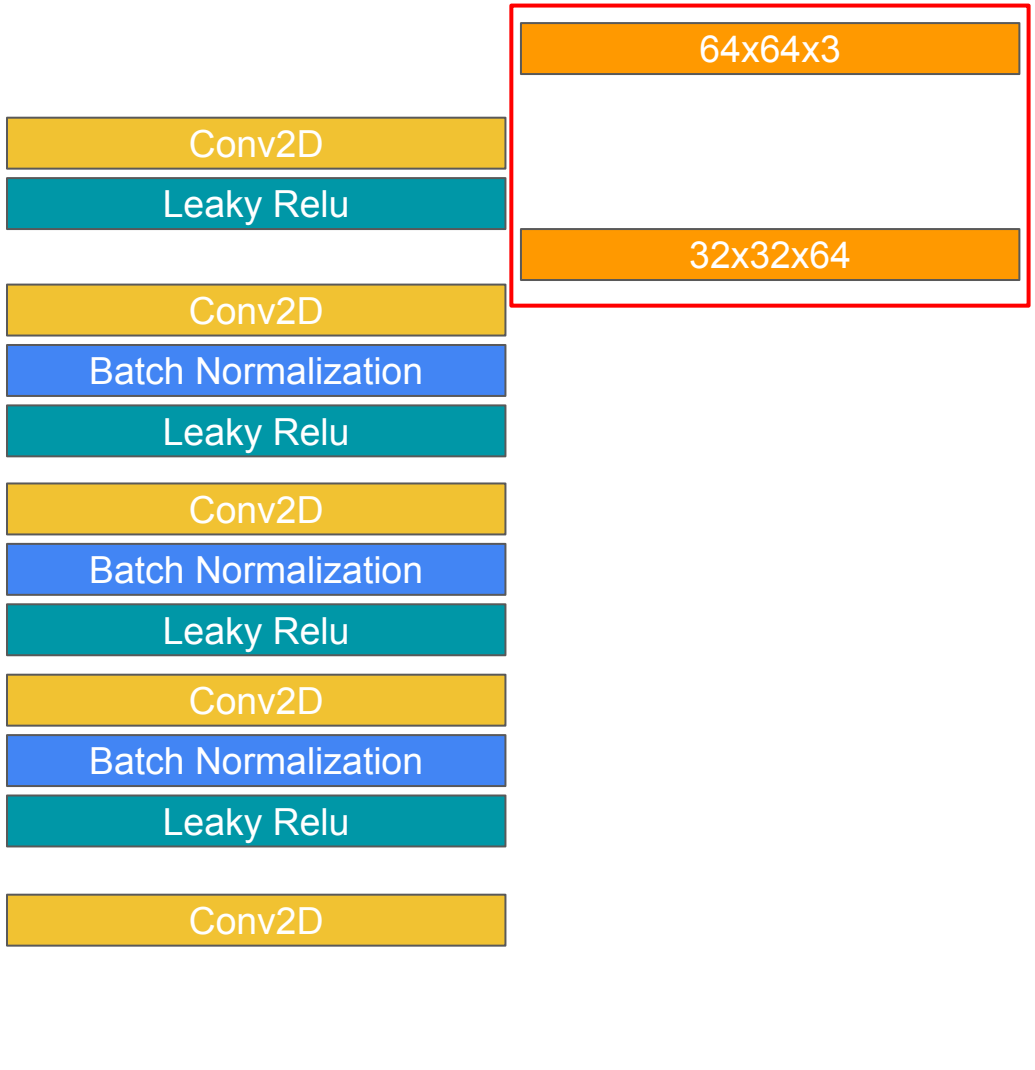
```
x = inputs = tf.keras.Input(shape=input_shape)

x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(256, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(128, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(64, 4, strides=2, padding='same', use_bias=False)(x)
x = BatchNormalization()(x)
x = layers.ReLU()(x)

x = layers.Conv2DTranspose(3, 4, strides
outputs = layers.Activation('tanh')(x)
```

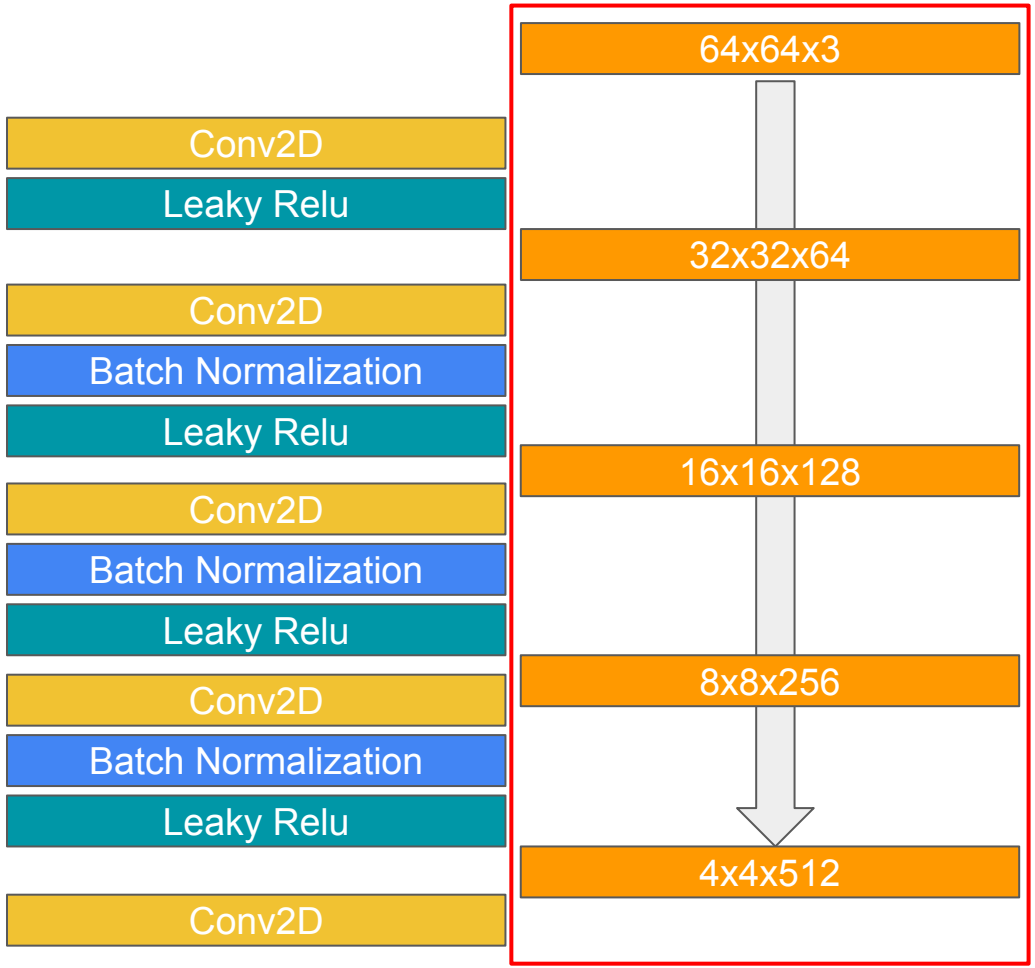


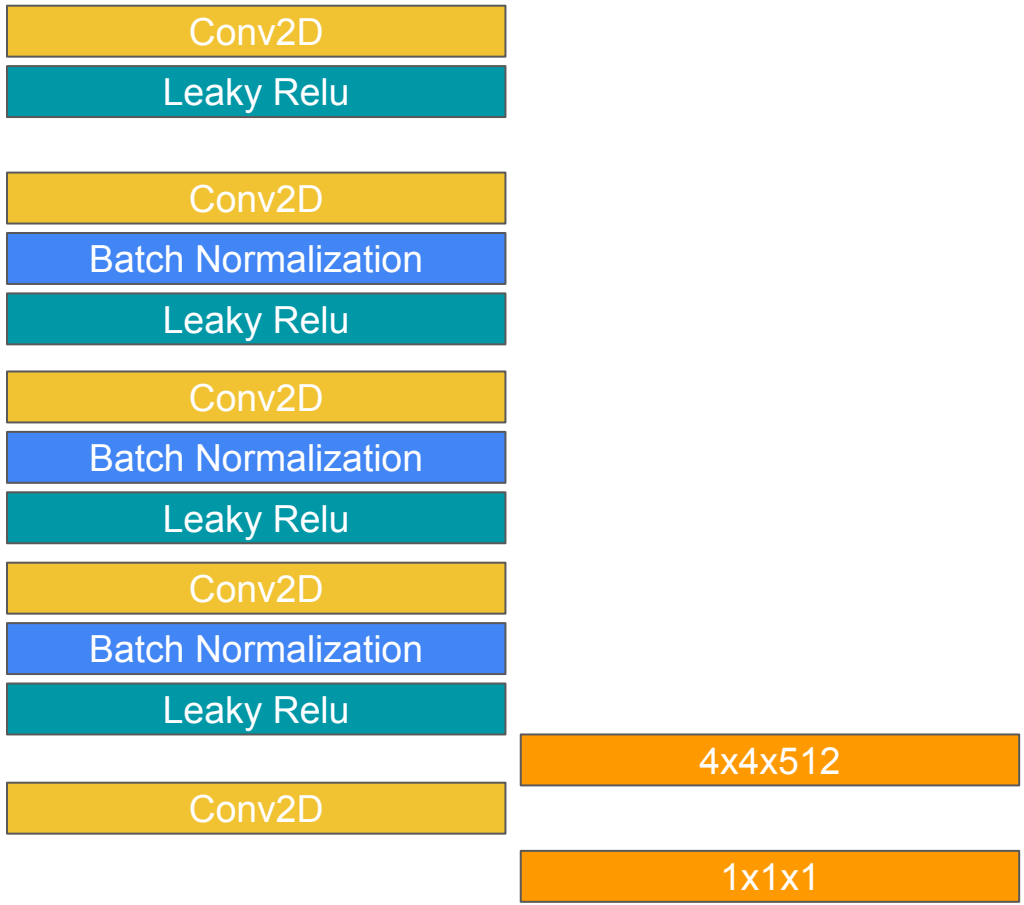

Conv2D
Batch Normalization
Leaky Relu


Conv2D
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D


64x64x3
Conv2D
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D


64x64x3
Conv2D
Leaky Relu
32x32x64
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D


64x64x3
Conv2D
Leaky Relu
32x32x64
Conv2D
Batch Normalization
Leaky Relu
16x16x128
Conv2D
Batch Normalization
Leaky Relu
8x8x256
Conv2D
Batch Normalization
Leaky Relu
4x4x512
Conv2D


Conv2D
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
Conv2D
Batch Normalization
Leaky Relu
4x4x512
Conv2D
1x1x1

```
x = inputs = tf.keras.Input(shape=input_shape)
x = layers.Conv2D(64, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(128, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(512, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

outputs = layers.Conv2D(1, 4, strides=1, padding='valid')(x)
```

```
x = inputs = tf.keras.Input(shape=input_shape)
x = layers.Conv2D(64, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(128, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(512, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

outputs = layers.Conv2D(1, 4, strides=1, padding='valid')(x)
```

```
x = inputs = tf.keras.Input(shape=input_shape)
x = layers.Conv2D(64, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(128, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(512, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

outputs = layers.Conv2D(1, 4, strides=1, padding='valid')(x)
```

```
x = inputs = tf.keras.Input(shape=input_shape)
x = layers.Conv2D(64, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(128, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(256, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

x = layers.Conv2D(512, 4, strides=2, padding='same', use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.LeakyReLU(alpha=0.2)(x)

outputs = layers.Conv2D(1, 4, strides=1, padding='valid')(x)
```