

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see

<https://creativecommons.org/licenses/by-sa/2.0/legalcode>

[Browse](#) > [Data Science](#) > [Machine Learning](#)

This course is part of the **Generative Adversarial Networks (GANs) Specialization**

Build Basic Generative Adversarial Networks (GANs)

★★★★★ 4.7 222 ratings



Sharon Zhou [+2 more instructors](#)

Enroll for Free

Starts Oct 23

Try for Free: Enroll to start your 7-day full access free trial

Financial aid available

11,610 already enrolled

Offered By



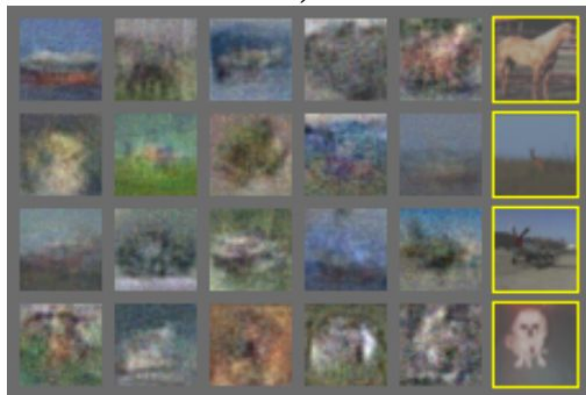
deeplearning.ai



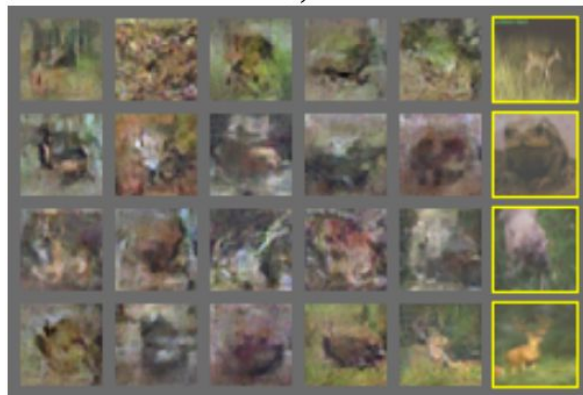
a)



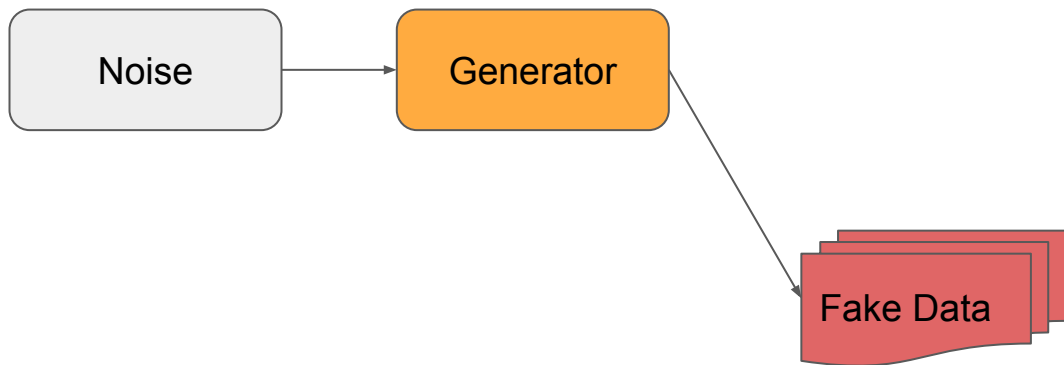
b)

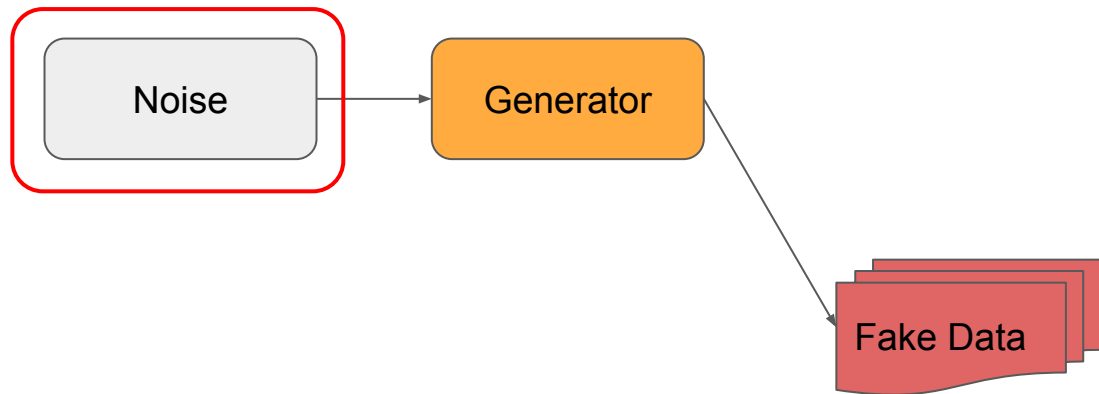


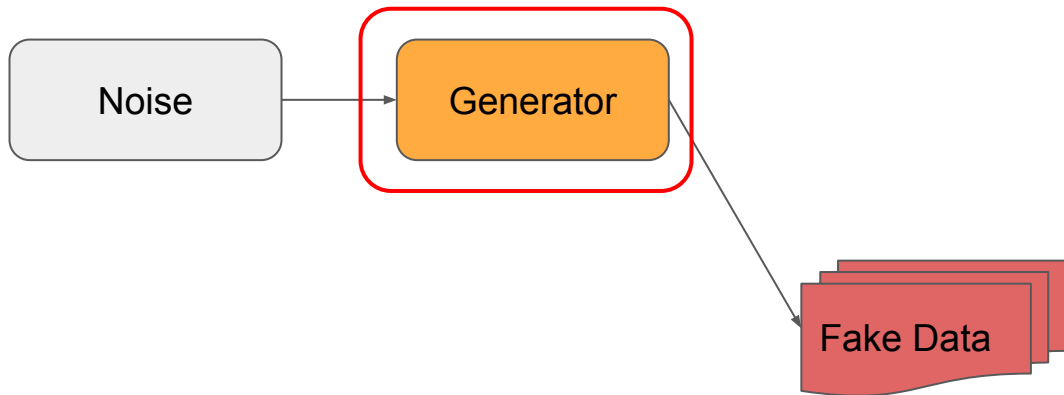
c)

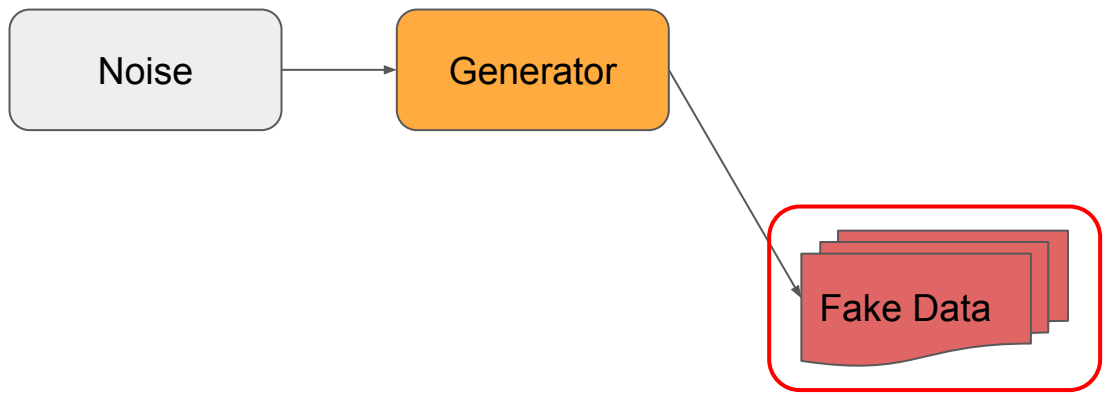


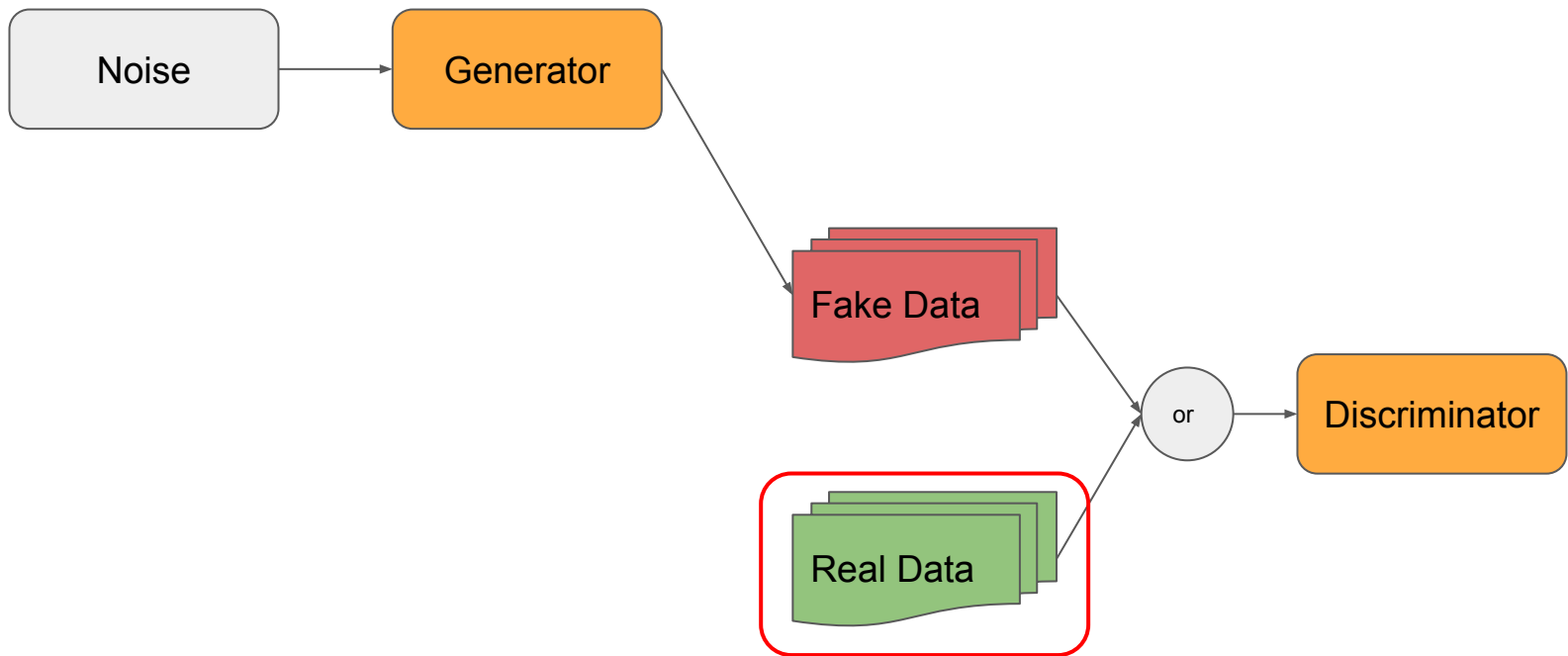
d)

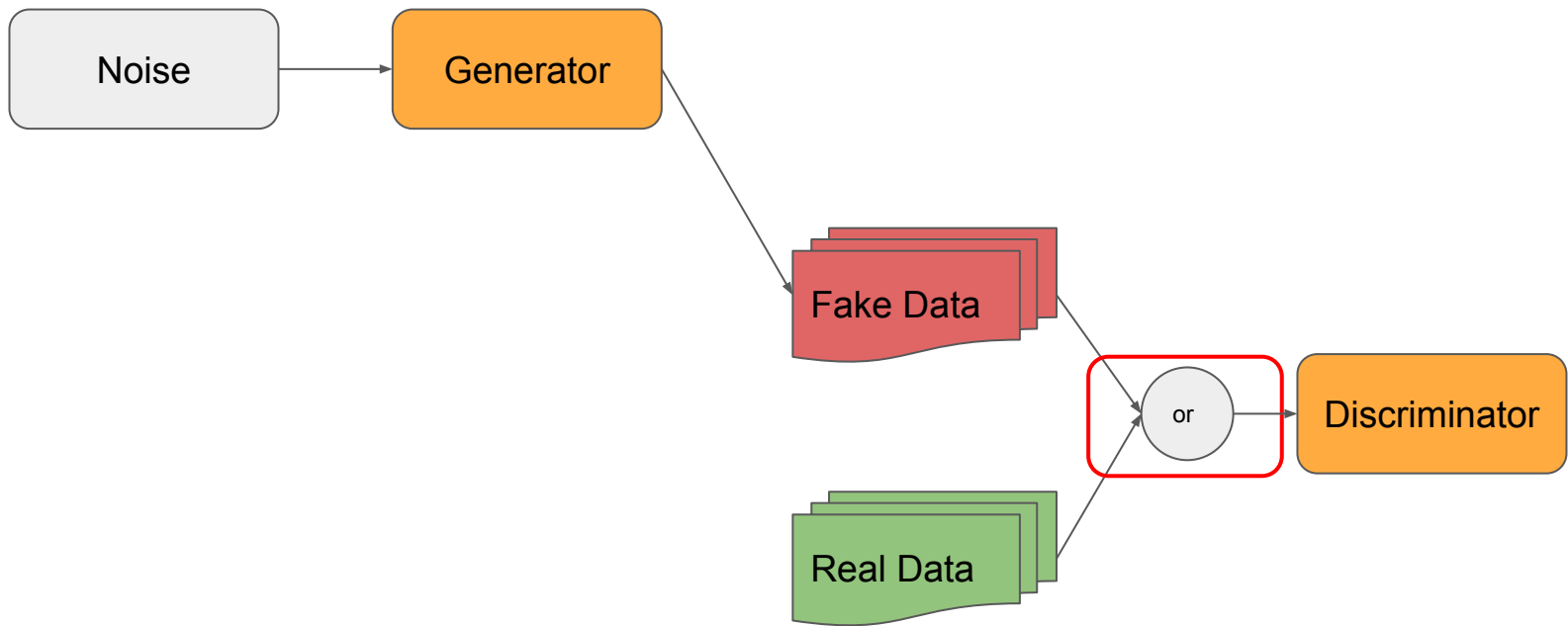


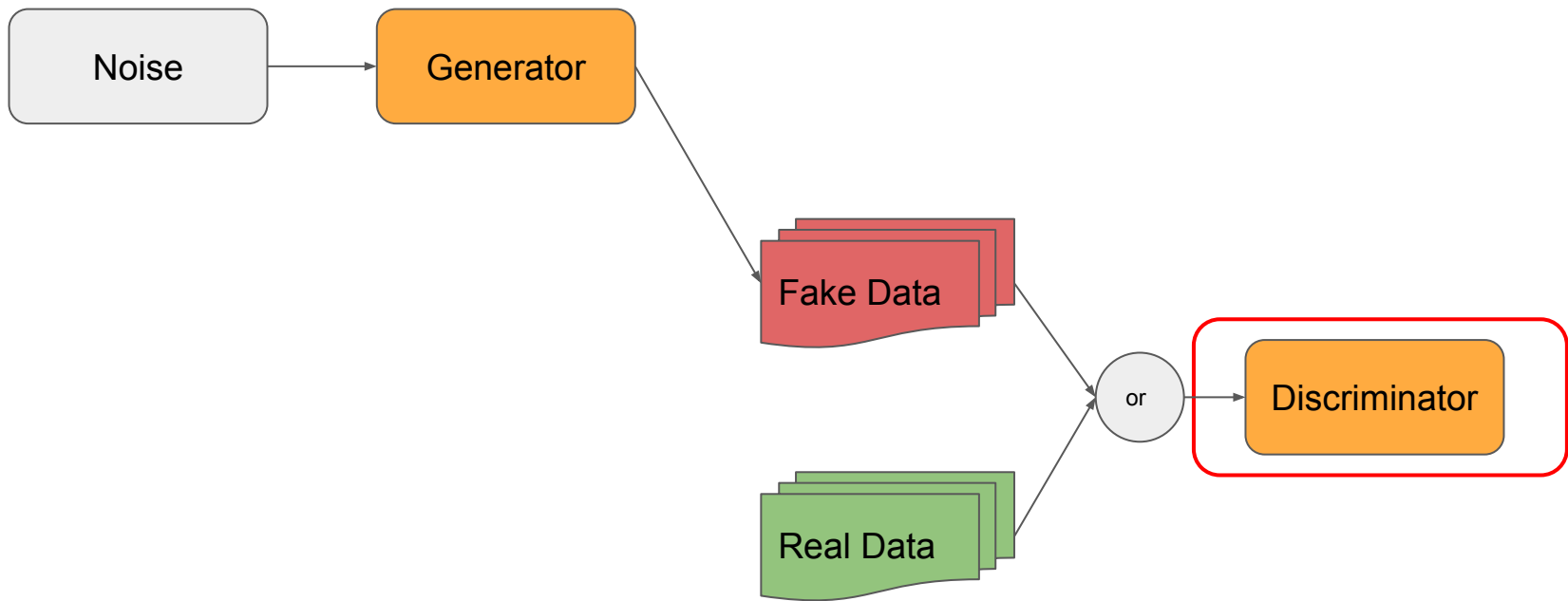


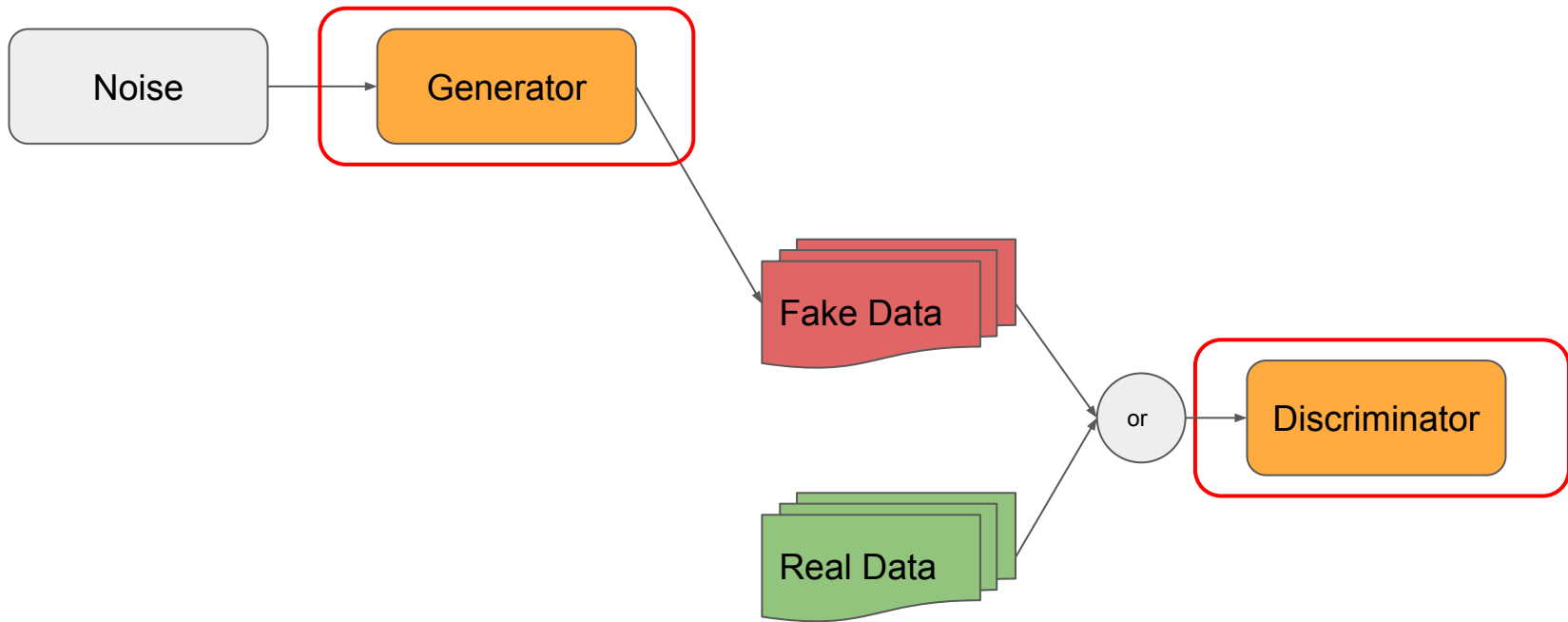


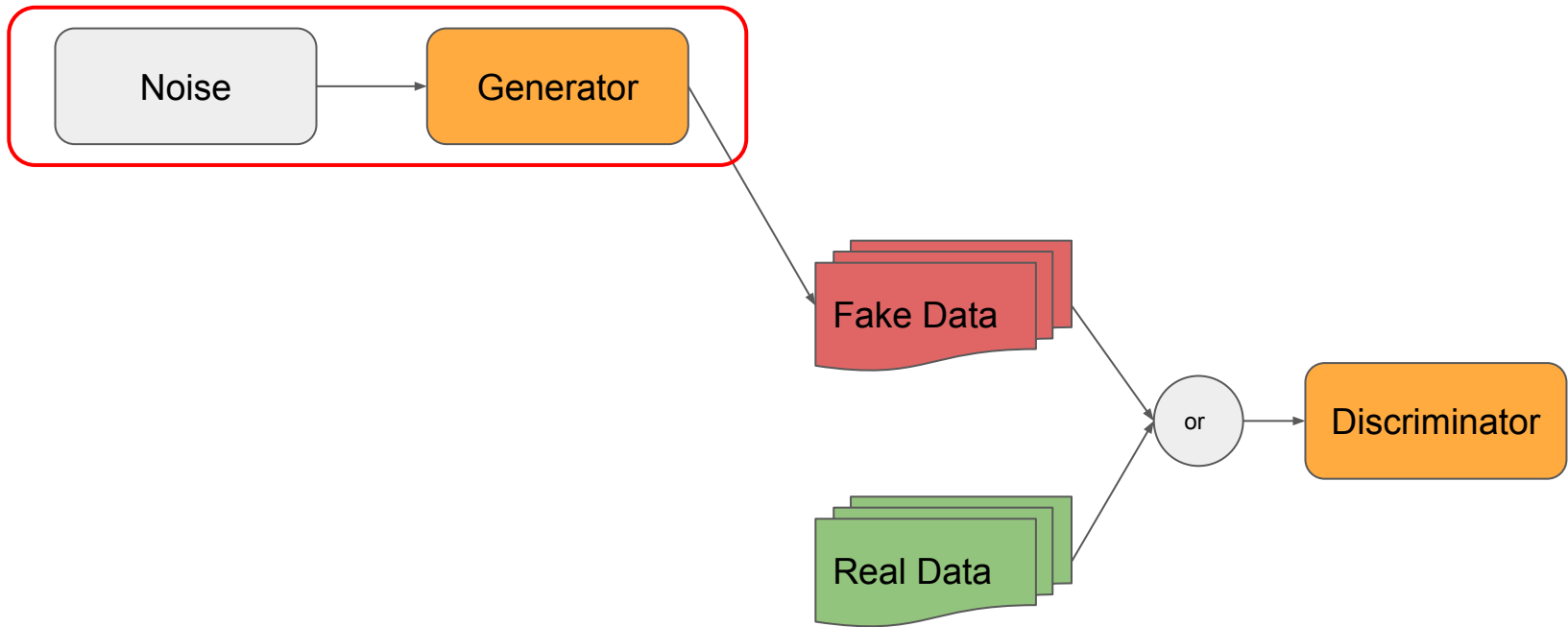


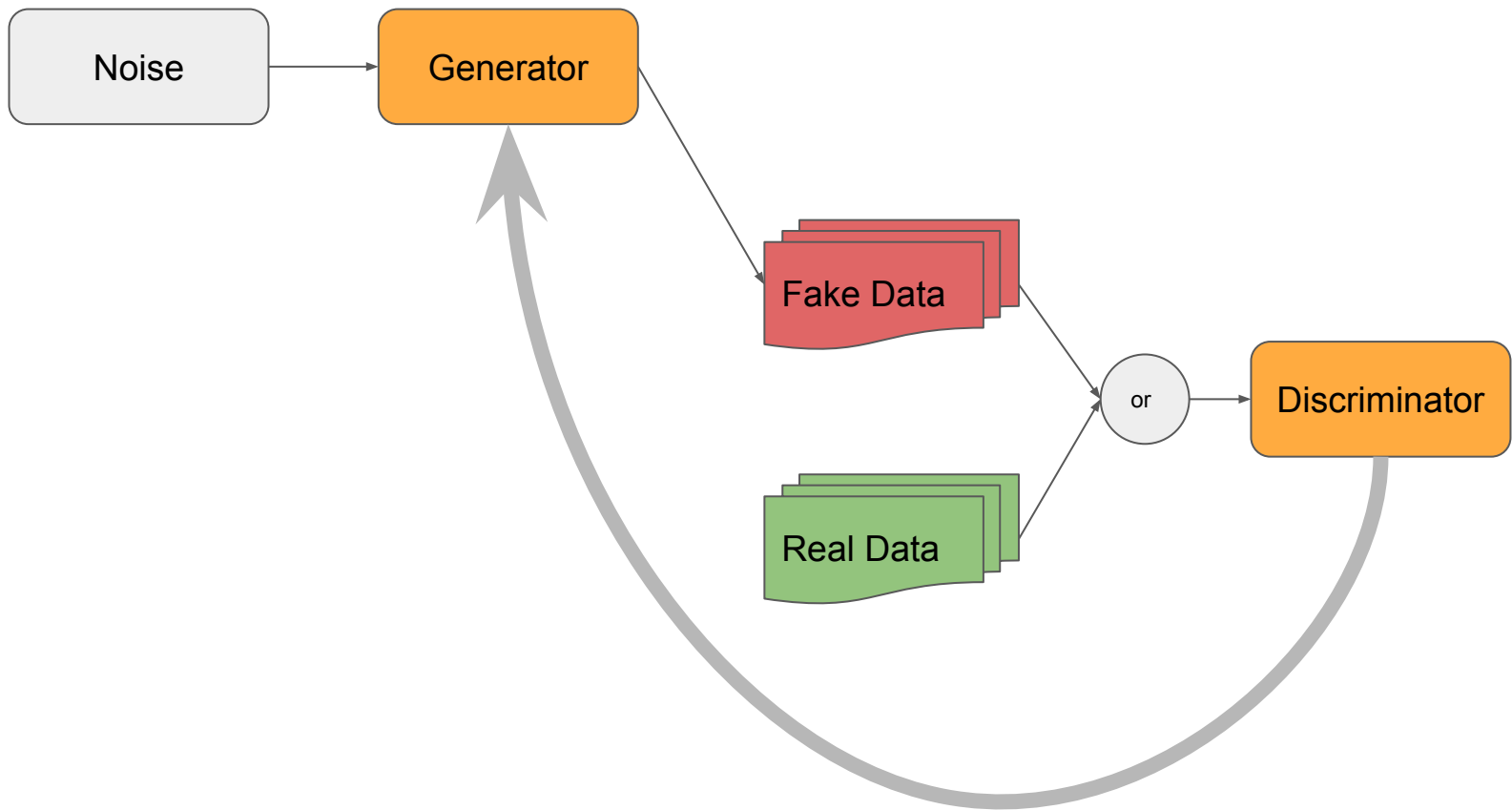


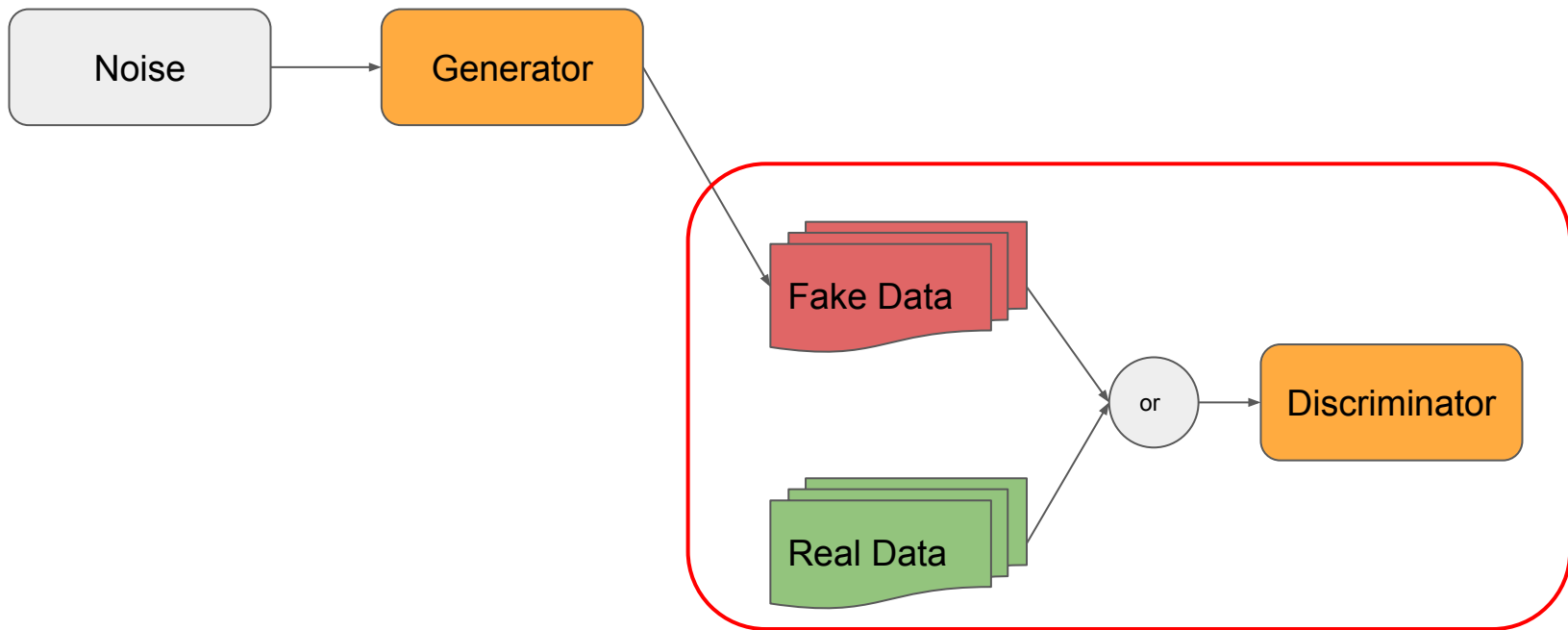




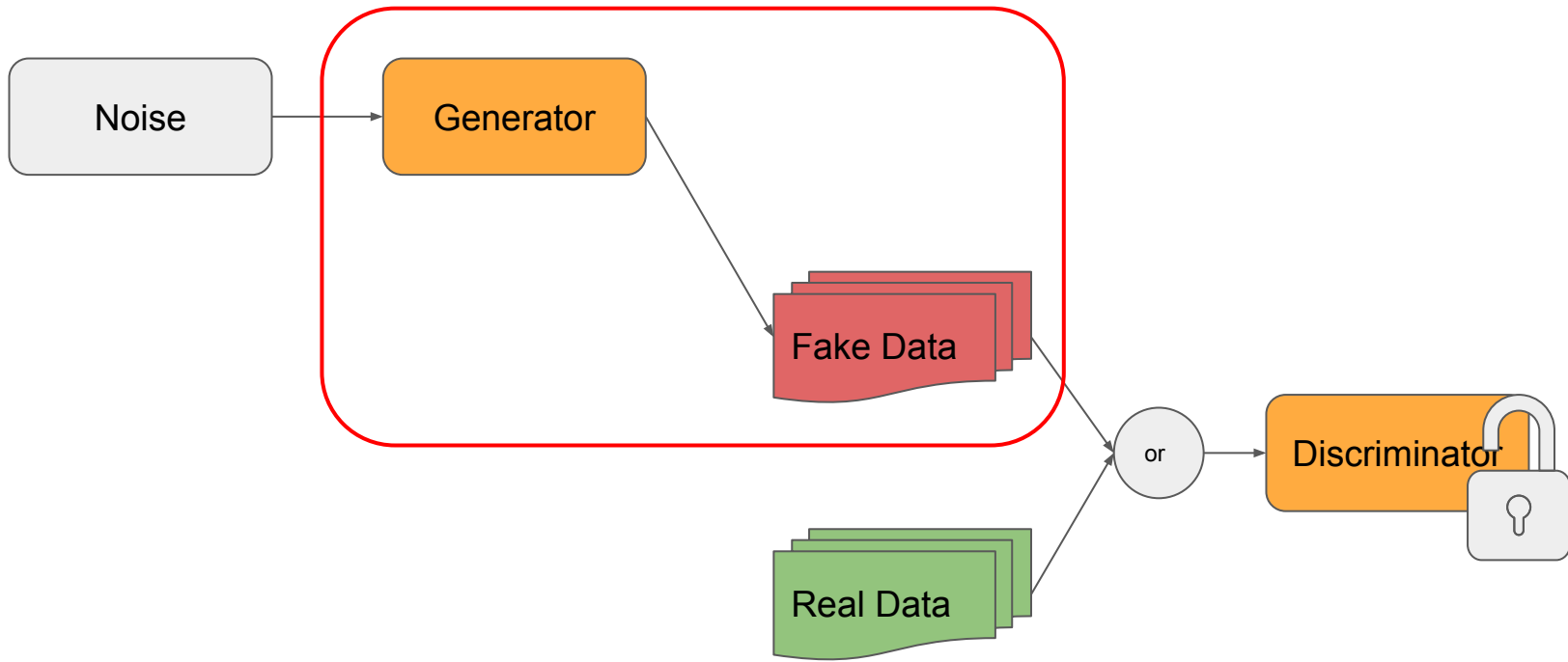




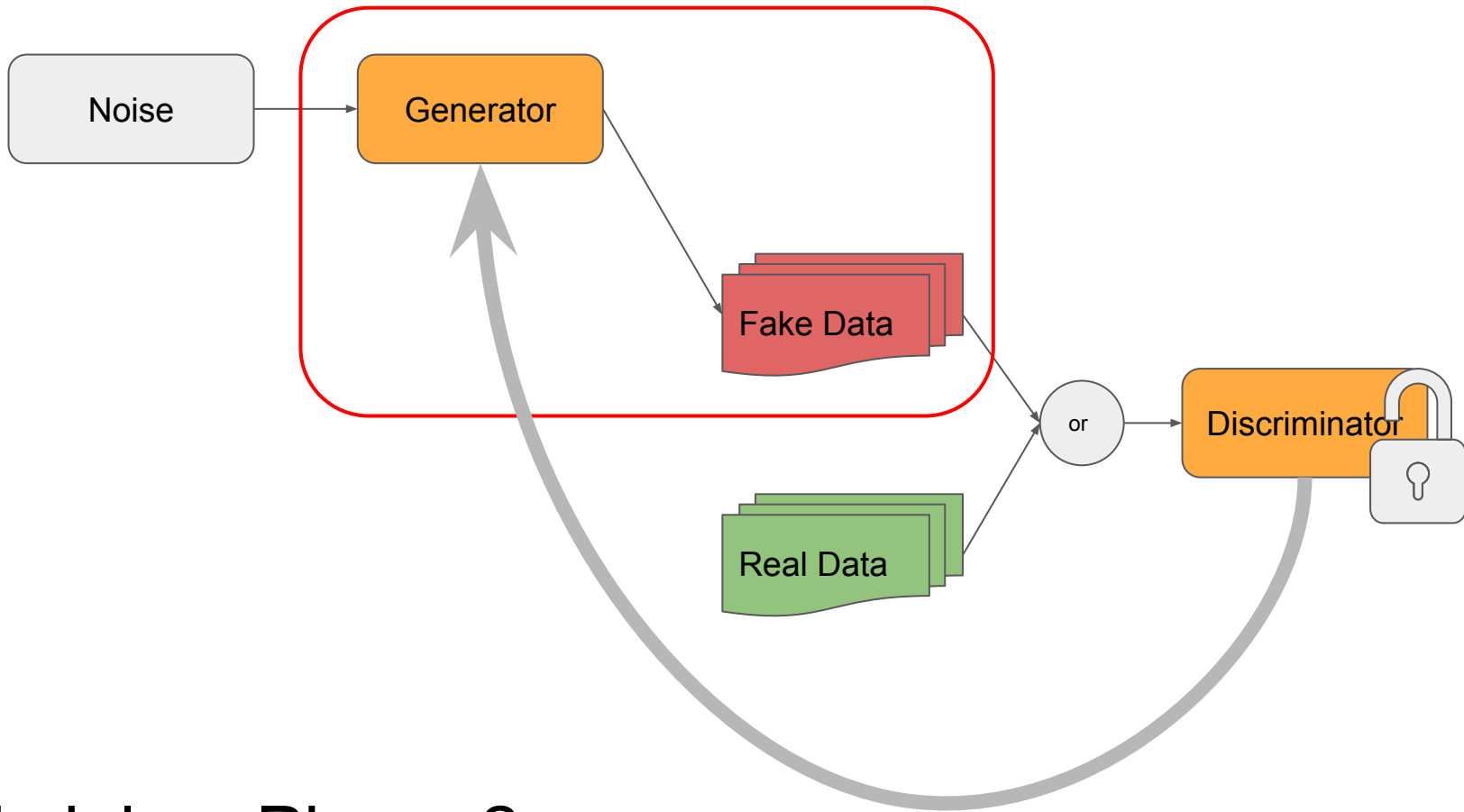




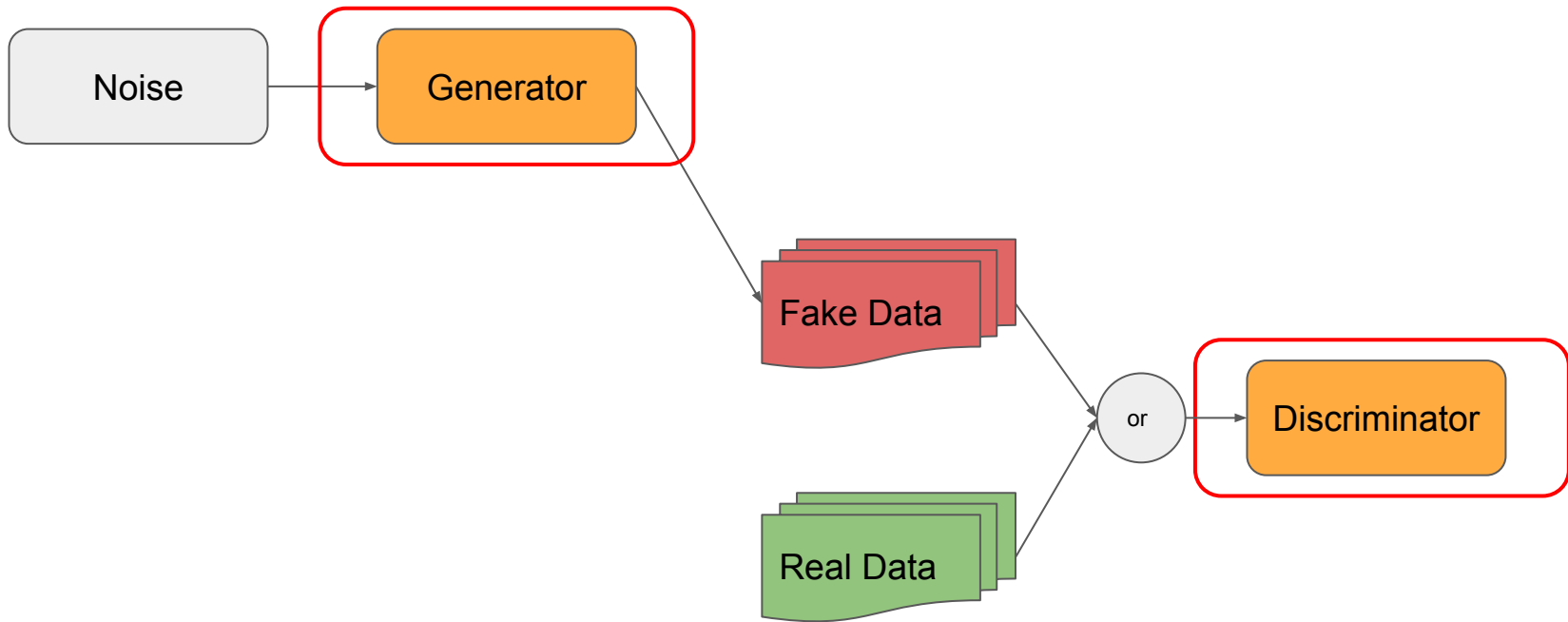
Training: Phase 1



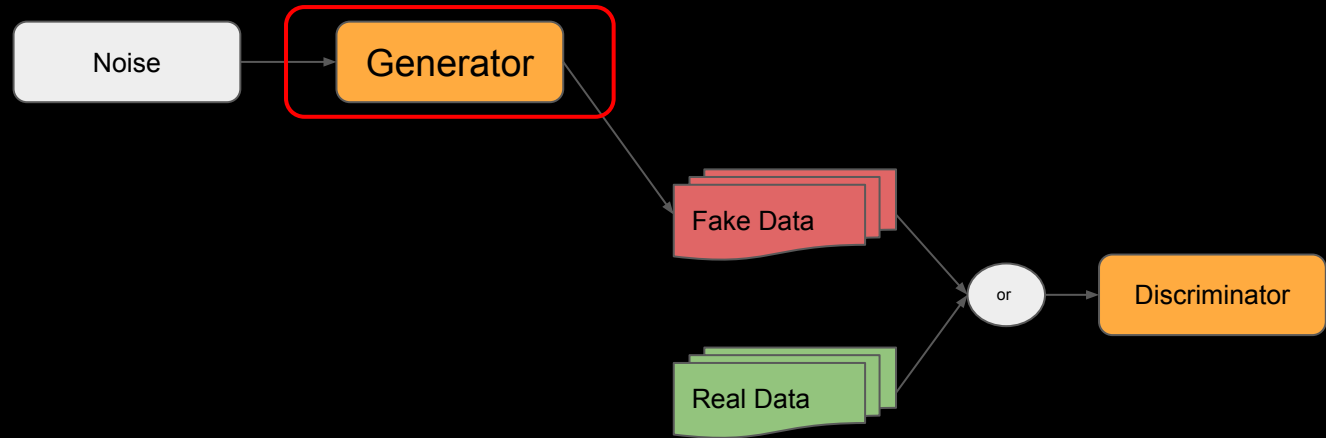
Training: Phase 2



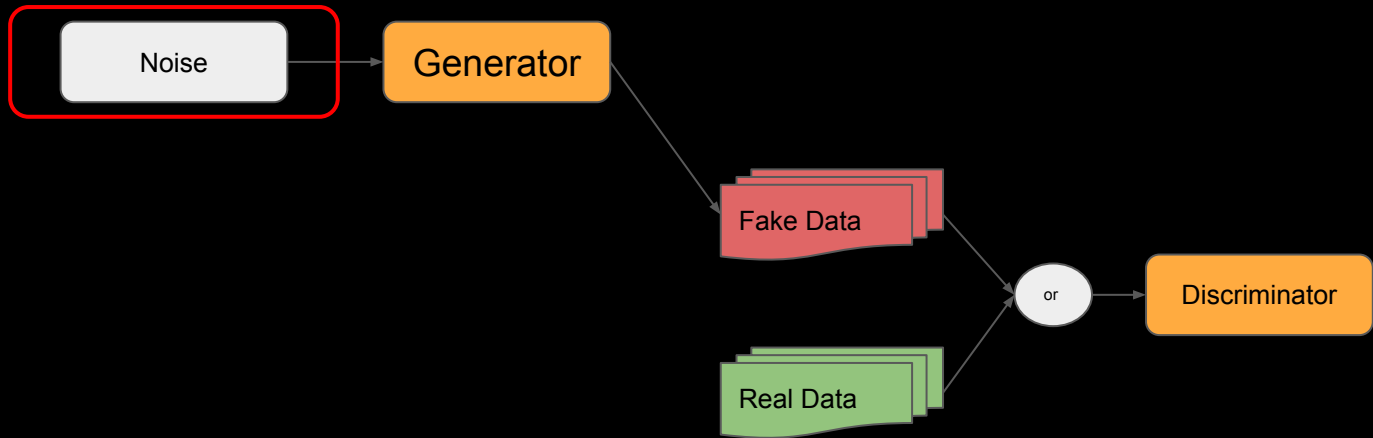
Training: Phase 2



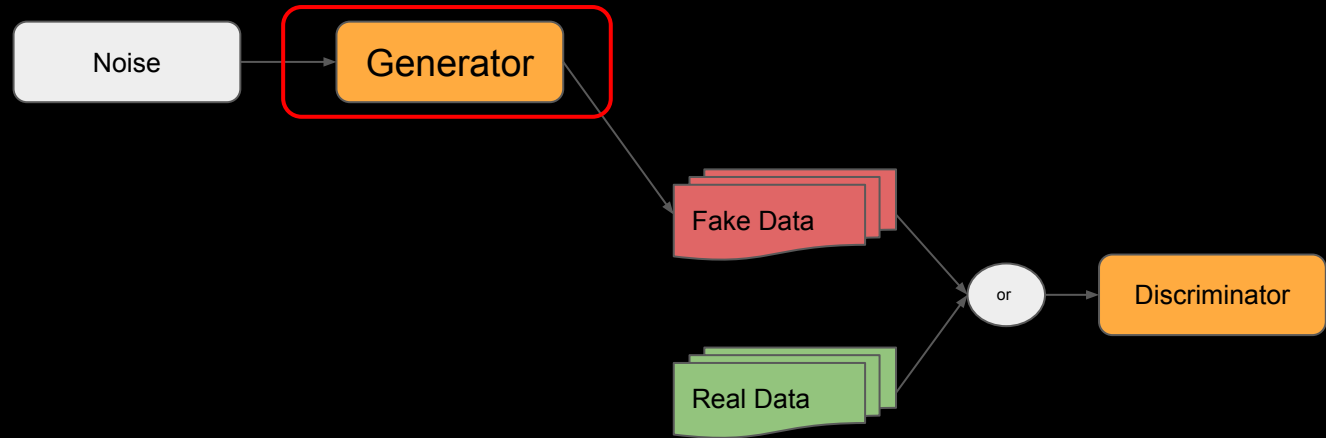
```
generator = keras.models.Sequential([  
    keras.layers.Dense(64, activation="selu",  
                        input_shape=[random_normal_dimensions]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(28 * 28, activation="sigmoid"),  
    keras.layers.Reshape([28, 28])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



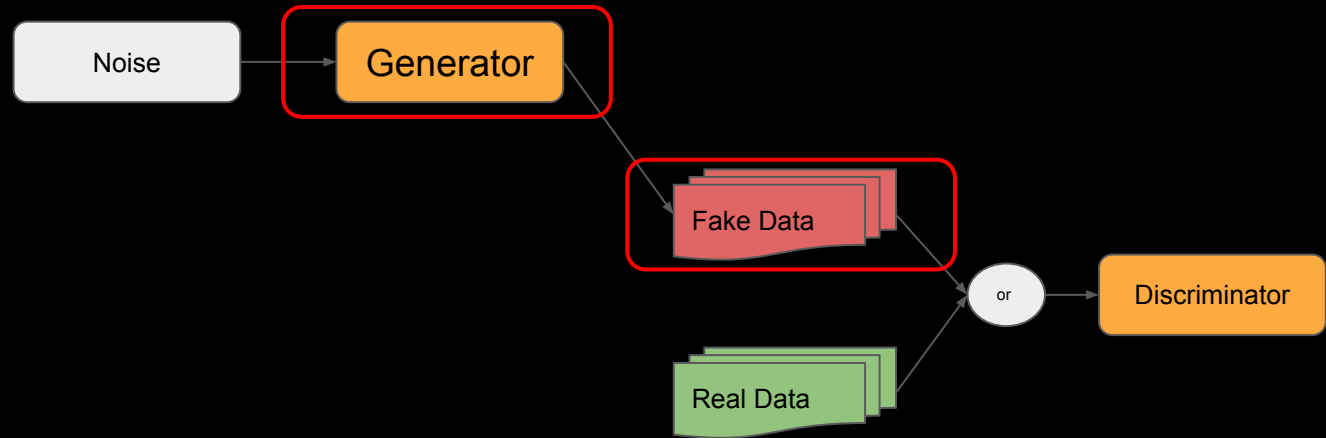
```
generator = keras.models.Sequential([  
    keras.layers.Dense(64, activation="selu",  
                        input_shape=[random_normal_dimensions]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(28 * 28, activation="sigmoid"),  
    keras.layers.Reshape([28, 28])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



```
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])  
])
```

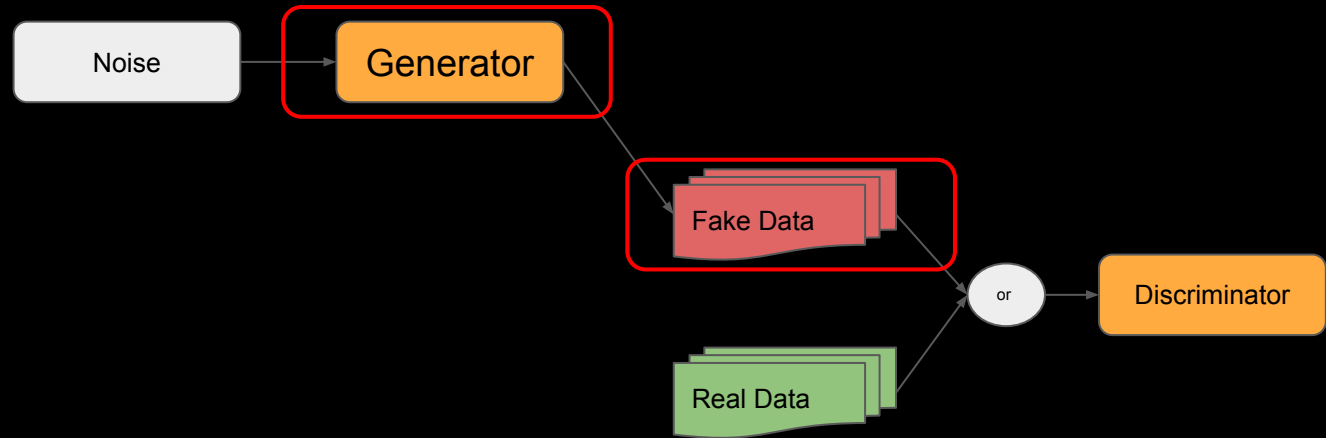


```
generator = keras.models.Sequential([  
    keras.layers.Dense(64, activation="selu",  
                        input_shape=[random_normal_dimensions]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(28 * 28, activation="sigmoid"),  
    keras.layers.Reshape([28, 28])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

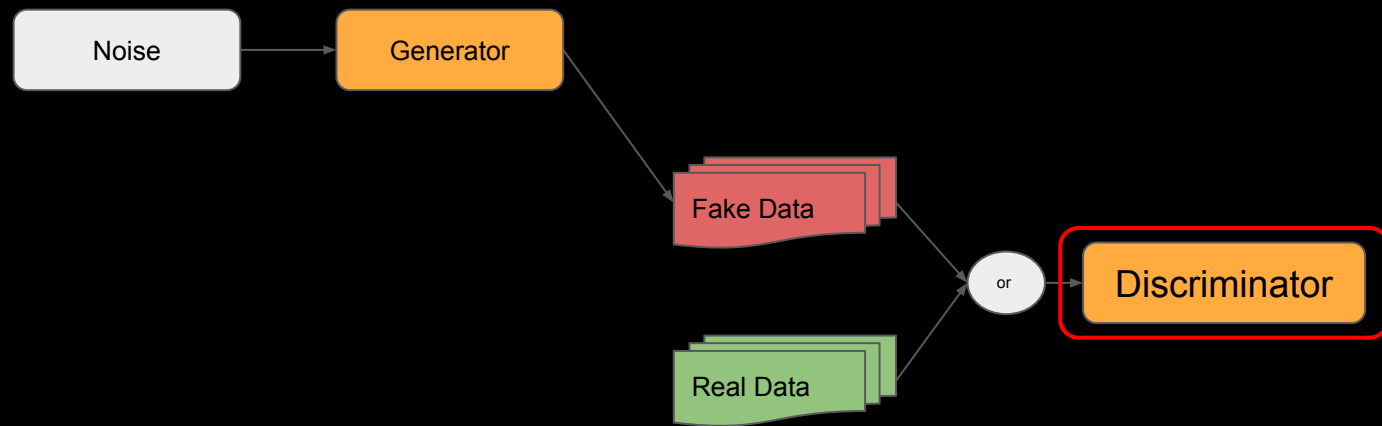


```
generator = keras.models.Sequential([  
    keras.layers.Dense(64, activation="selu",  
                        input_shape=[random_normal_dimensions]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(28 * 28, activation="sigmoid"),  
    keras.layers.Reshape([28, 28])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

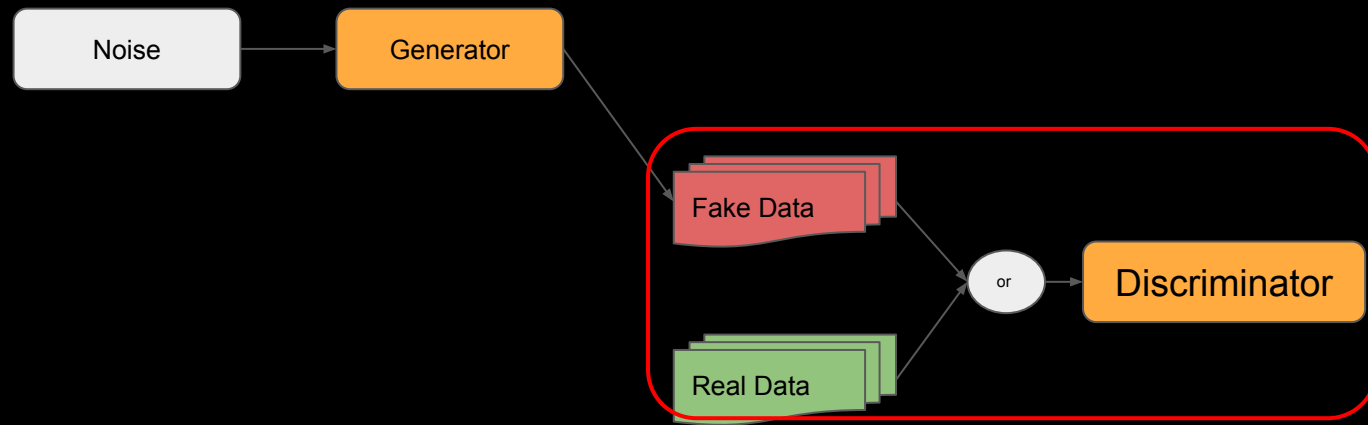
<https://arxiv.org/abs/1706.02515>



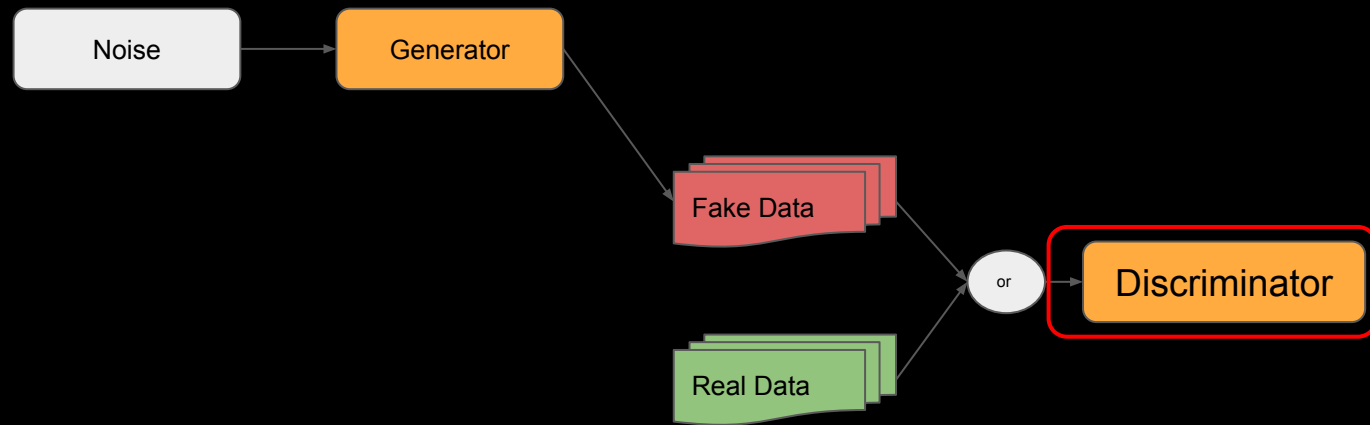
```
discriminator = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(64, activation="selu"),  
    keras.layers.Dense(1, activation="sigmoid")  
])
```



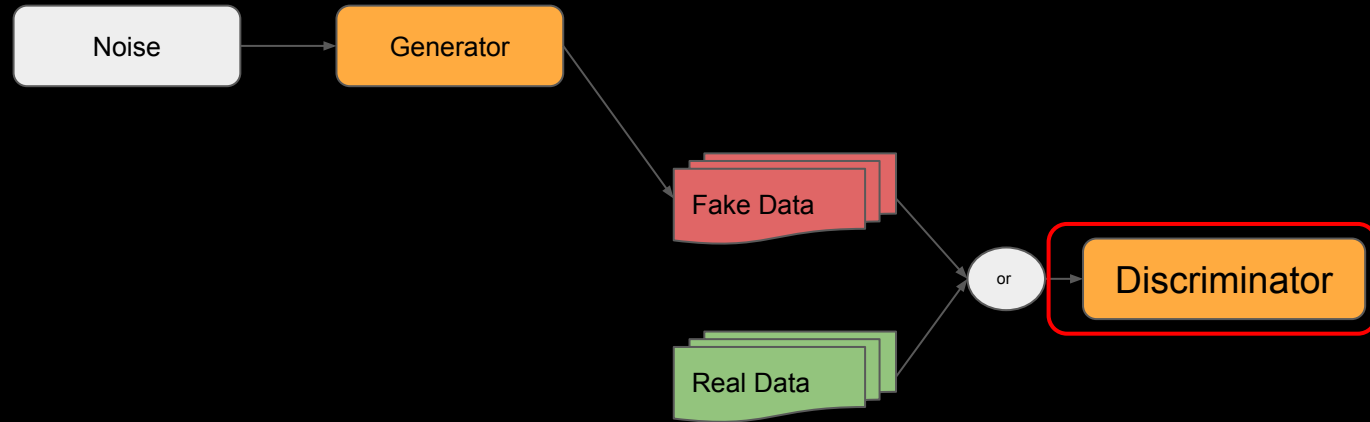
```
discriminator = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(64, activation="selu"),  
    keras.layers.Dense(1, activation="sigmoid")  
])
```



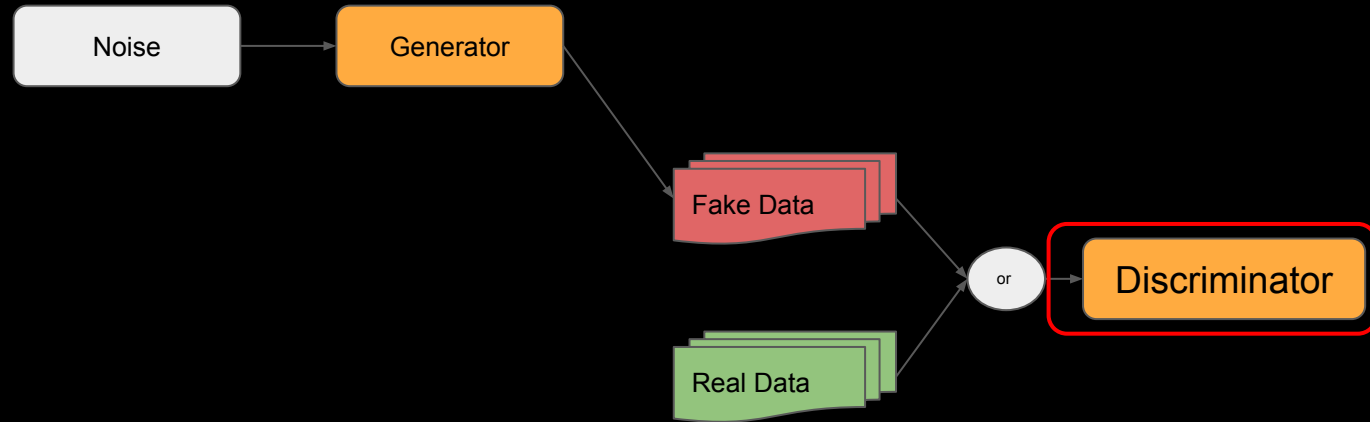

```
discriminator = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(64, activation="selu"),  
    keras.layers.Dense(1, activation="sigmoid")  
])
```



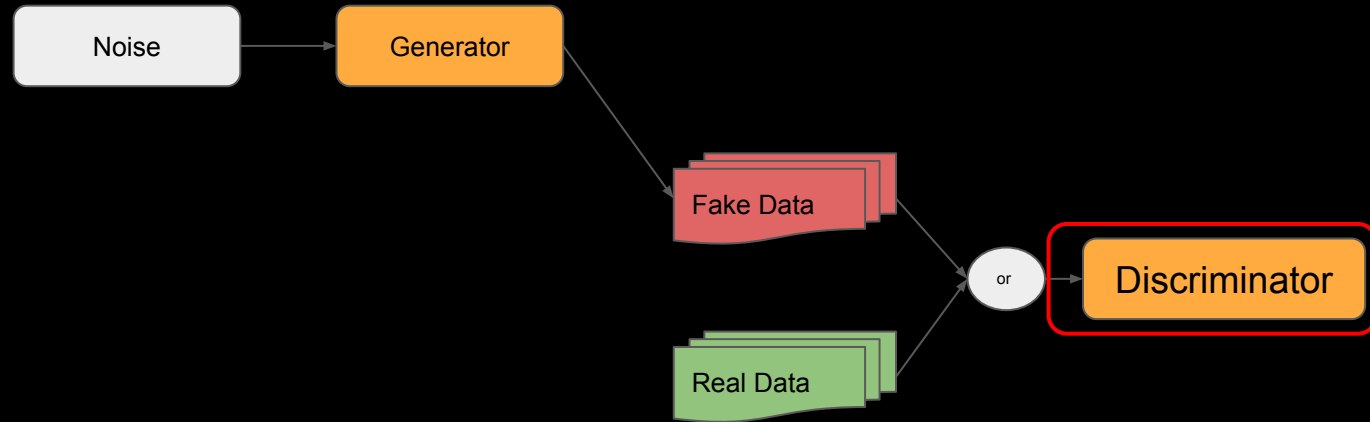
```
discriminator = keras.models.Sequential([  
    keras.layers.Flatten(input_shape=[28, 28]),  
    keras.layers.Dense(128, activation="selu"),  
    keras.layers.Dense(64, activation="selu"),  
    keras.layers.Dense(1, activation="sigmoid")  
])
```



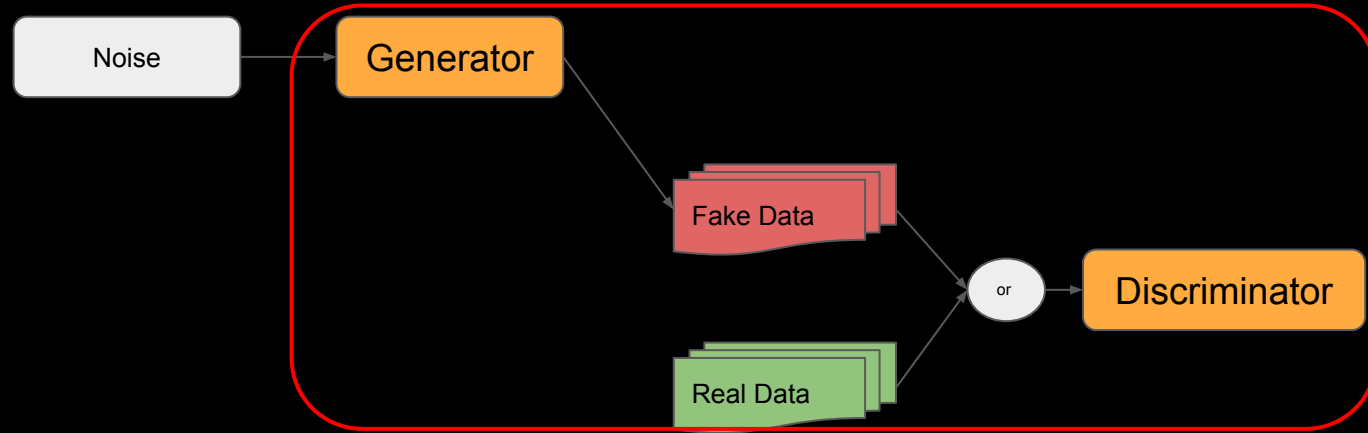
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")  
gan = keras.models.Sequential([generator, discriminator])  
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



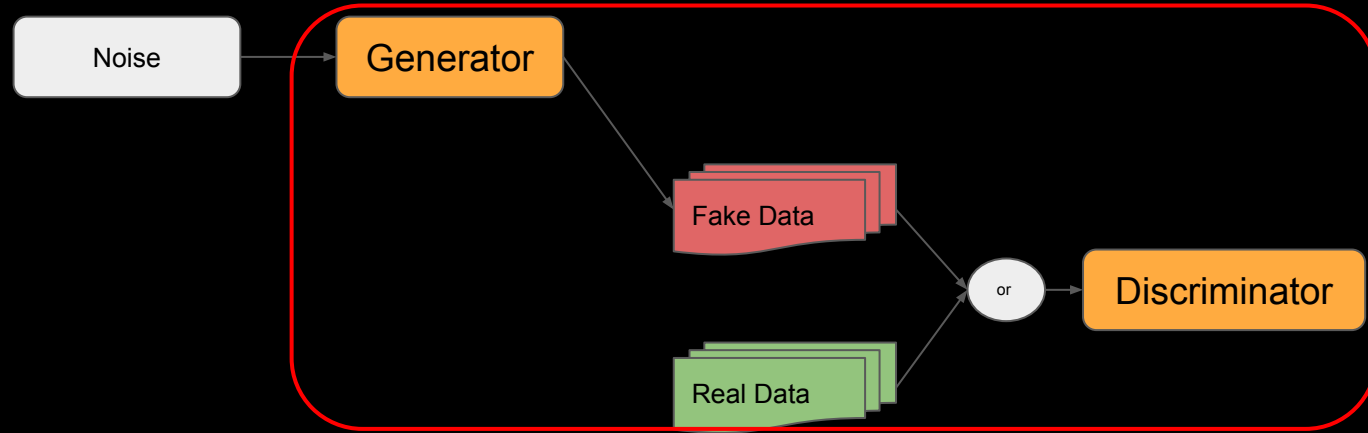
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")  
gan = keras.models.Sequential([generator, discriminator])  
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```

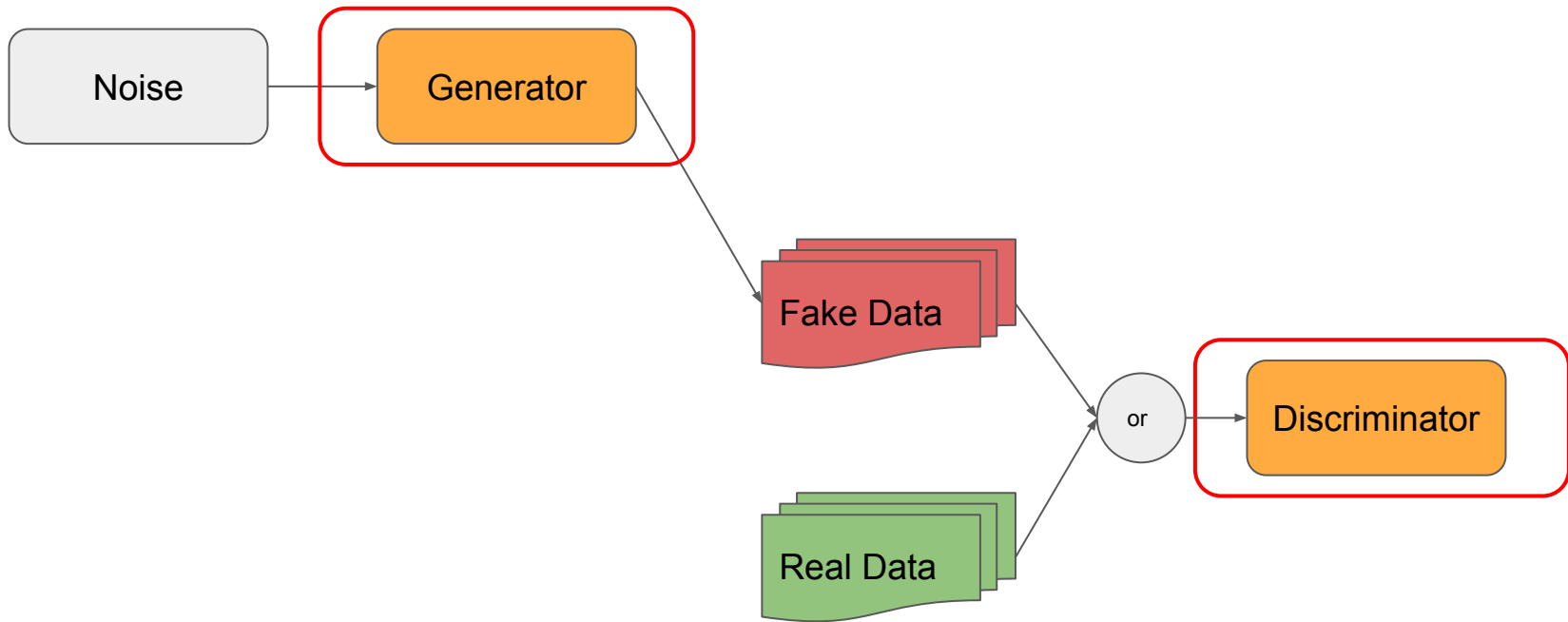


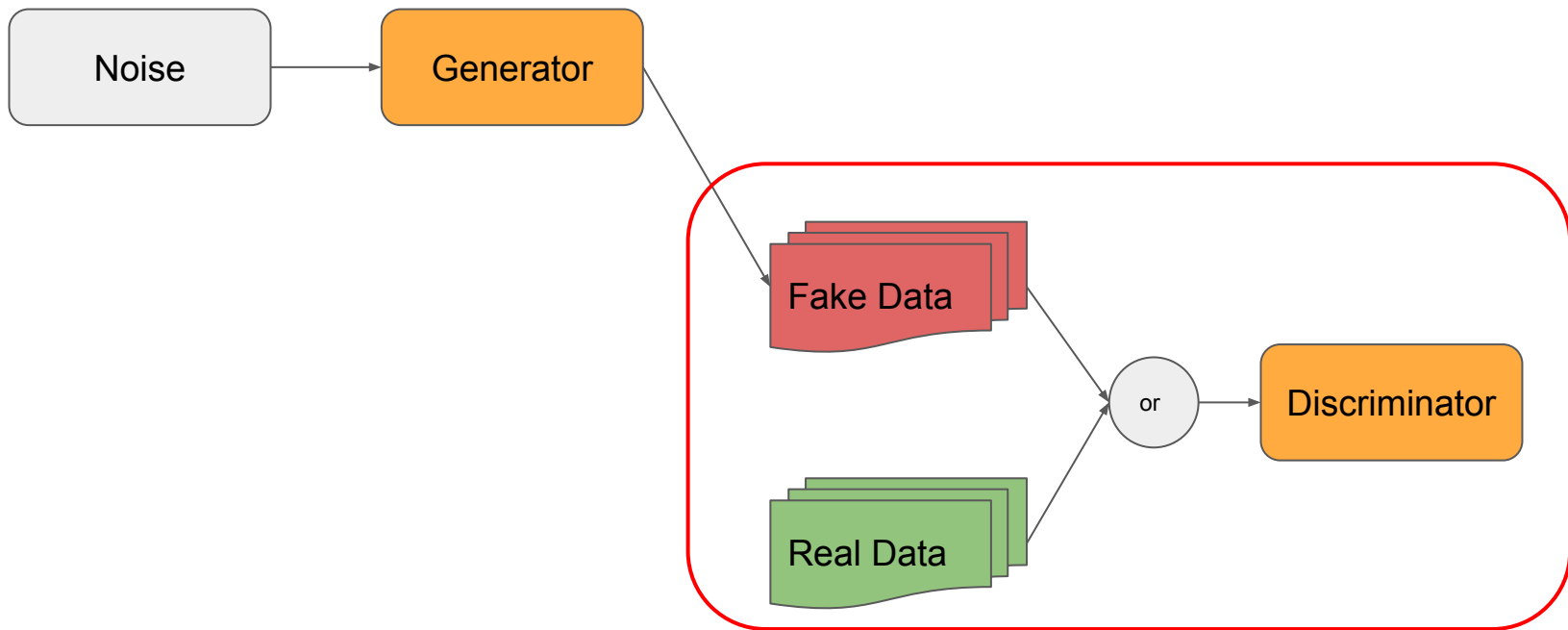
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")  
gan = keras.models.Sequential([generator, discriminator])  
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



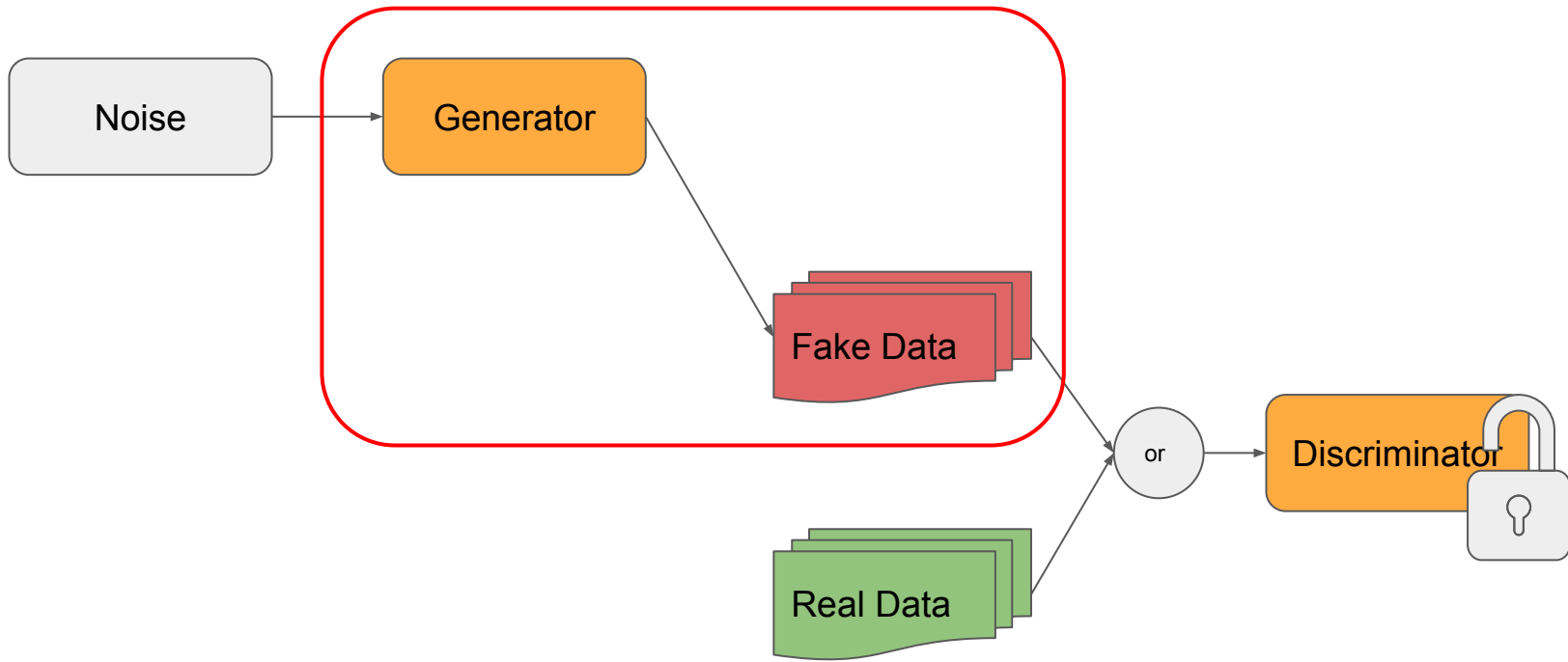
```
discriminator.compile(loss="binary_crossentropy", optimizer="rmsprop")
gan = keras.models.Sequential([generator, discriminator])
gan.compile(loss="binary_crossentropy", optimizer="rmsprop")
```



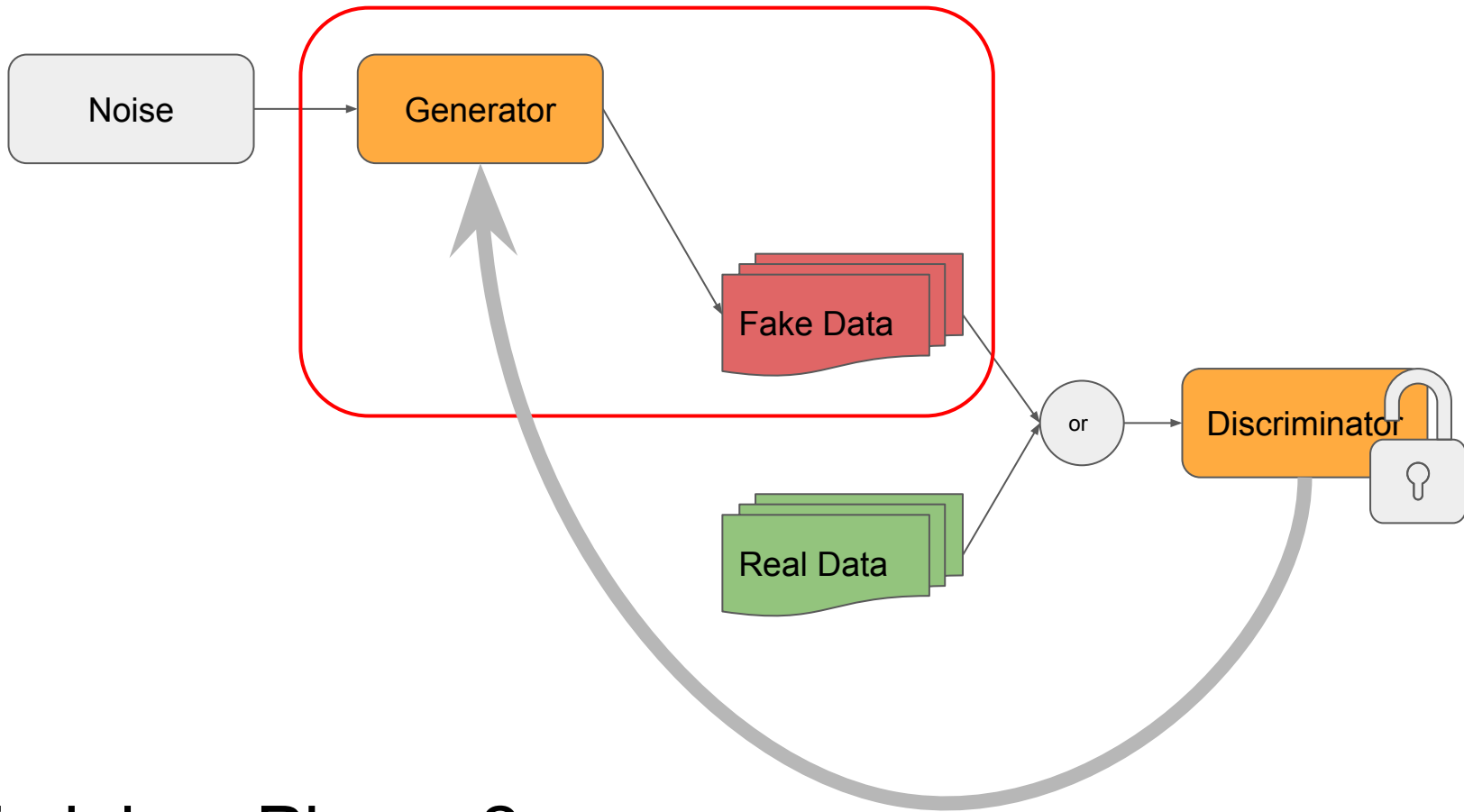




Training: Phase 1



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)
```

3	P	3	P	3	1	1	0	2	0	3	7	1	9	2	9
2	9	9	P	2	1	9	7	3	2	7	3	P	9	9	2
0	0	9	2	P	9	2	2	2	P	1	9	P	0	2	1
1	1	0	1	P	2	9	2	2	0	0	1	2	3	9	9
0	9	9	3	2	2	2	0	9	9	2	3	9	0	9	0
2	9	1	2	P	9	2	2	2	3	9	P	9	2	2	0
1	2	2	9	0	2	9	2	9	1	1	2	2	0	9	9
9	9	0	0	2	2	1	P	P	9	3	1	9	3	9	9

2	3	4	2	8	0	7	2	4	3	4	9	8	3	3	4
7	0	9	0	4	8	3	3	4	4	9	8	5	9	8	
3	8	8	8	7	8	6	4	8	9	8	3	4	3	3	9
8	9	8	3	8	7	6	3	8	8	9	8	7	7	8	4
8	5	6	5	7	9	8	5	8	7	6	6	0	4	3	4
7	2	5	7	6	3	6	7	3	7	9	4	4	4	7	3
4	8	4	4	8	5	3	9	7	4	3	8	7	7	4	8
4	4	3	2	8	6	7	9	3	4	4	4	7	9	4	9

0	2	1	9	7	2	7	8	9	1	1	9	9	1	4	7
1	9	2	4	4	9	1	5	1	9	1	9	4	3	3	4
1	7	7	8	7	5	9	4	4	7	1	7	7	8	1	8
7	7	7	3	4	1	7	4	1	1	3	5	5	3	3	6
7	7	7	7	3	7	5	3	7	1	7	9	4	1	1	1
7	1	1	5	1	6	7	7	3	1	7	9	1	7	9	8
9	2	1	1	1	1	1	7	1	9	9	1	1	9	1	6
7	4	2	1	1	1	1	1	9	7	7	1	3	4	9	7

9	8	7	1	9	1	1	7	1	1	1	7	1	1	1
1	1	1	1	9	1	1	7	9	7	1	1	1	1	9
1	1	1	1	9	1	1	1	1	1	1	7	1	1	7
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	9	8	1	1	9	1	1	1	1	1	1	1	1
1	4	1	1	1	7	1	1	1	7	7	8	1	1	1
1	1	1	1	1	1	1	1	7	1	1	1	1	1	1
1	1	1	7	1	1	1	1	7	1	1	1	1	1	1

9	8	7	1	9	1	1	7	1	1	1	7	1	1	1
1	1	1	1	9	1	1	7	9	7	1	1	1	1	9
1	1	1	1	9	1	1	1	1	1	1	7	1	1	7
4	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	9	8	1	1	9	1	1	1	1	1	1	1	1
1	4	1	1	1	7	1	1	1	7	7	8	1	1	1
1	1	1	1	1	1	1	1	7	1	1	1	1	1	1
1	1	1	7	1	1	1	1	7	1	1	1	1	1	1



<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.

[illegible]


```
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),

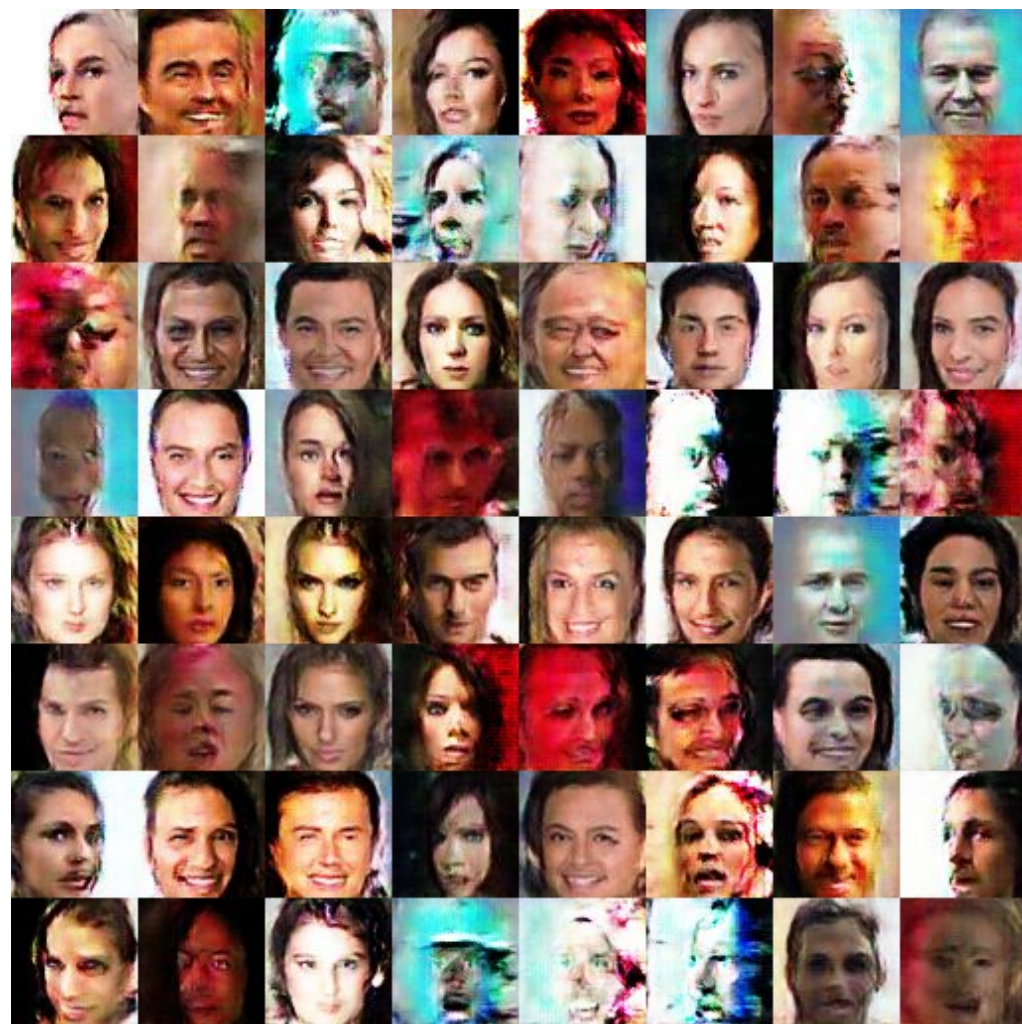
    keras.layers.Conv2D(128, kernel_size=5, strides=2, padding="SAME",
        activation=keras.layers.LeakyReLU(0.2)),
    keras.layers.Dropout(0.4),

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

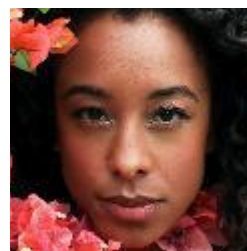
https://www.tensorflow.org/api_docs/python/tf/keras/layers/LeakyReLU

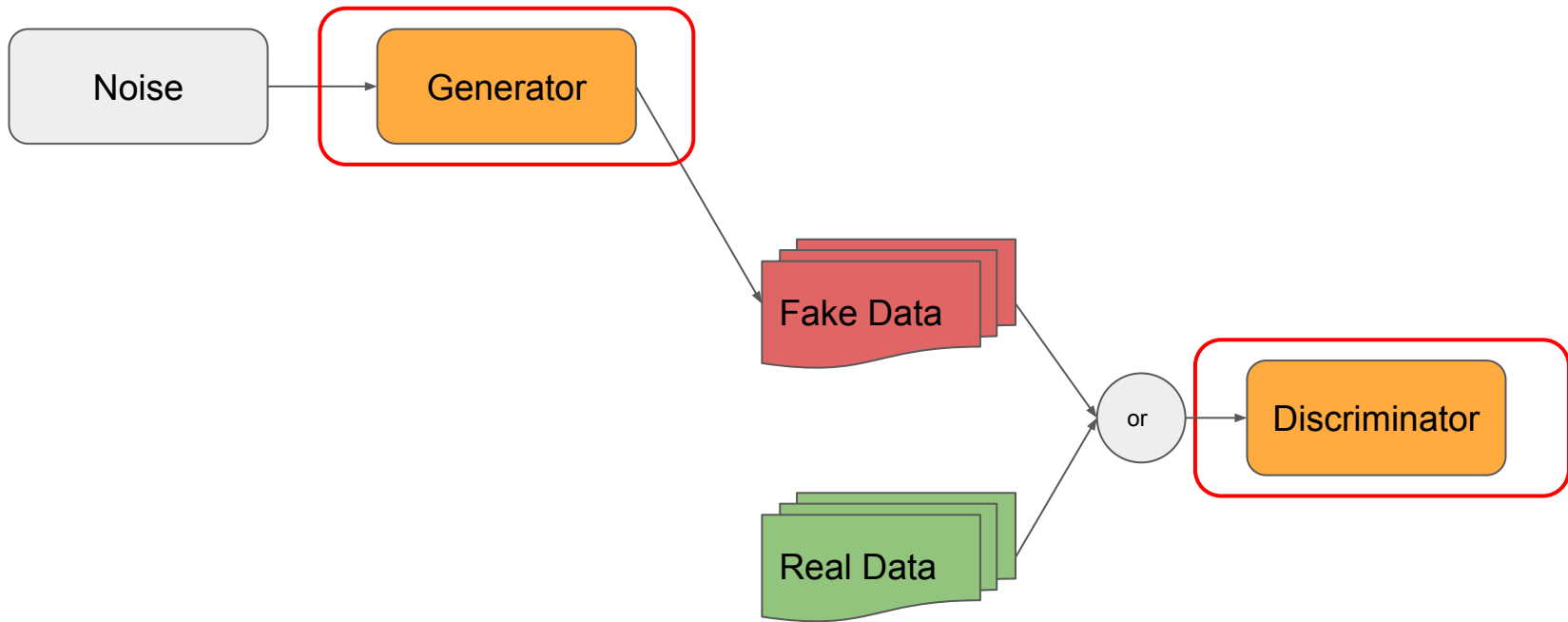














Conv2DTranspose

A diagram showing three stacked rectangular blocks representing layers in a neural network. The top block is yellow and labeled 'Conv2DTranspose'. The middle block is blue and labeled 'Batch Normalization'. The bottom block is teal and labeled 'Relu'. All blocks have a thin black border and are centered horizontally.

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

231	87
3	200

1

Strides = 2,2

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

1

Strides = 2,2

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

1

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

Strides = 2,2



1



231	0
0	0

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

Strides = 2,2



1



231	0	87	0
0	0	0	0

Strides = 2,2

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



1



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

Strides = 2,2

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



1



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



A diagram showing three stacked rectangular blocks representing neural network layers. The top block is yellow and labeled 'Conv2DTranspose', the middle block is blue and labeled 'Batch Normalization', and the bottom block is teal and labeled 'Relu'. A red rectangular border encloses the top yellow block.

Conv2DTranspose

Batch Normalization

Relu



A diagram showing three stacked rectangular blocks representing neural network layers. The top block is yellow and labeled 'Conv2DTranspose'. The middle block is blue and labeled 'Batch Normalization', and it is enclosed within a red rectangular border. The bottom block is teal and labeled '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

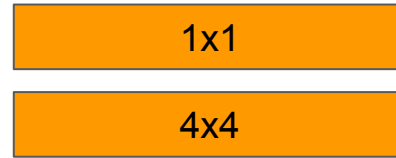
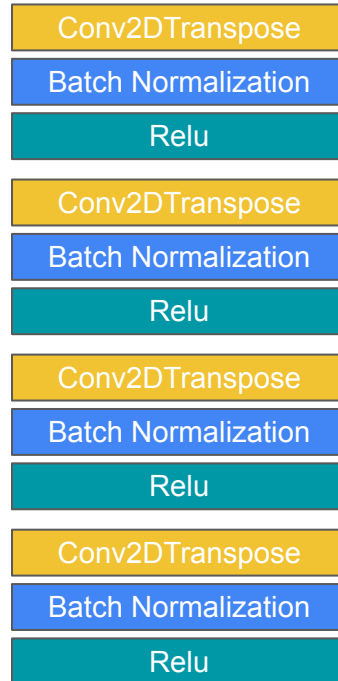
Batch Normalization

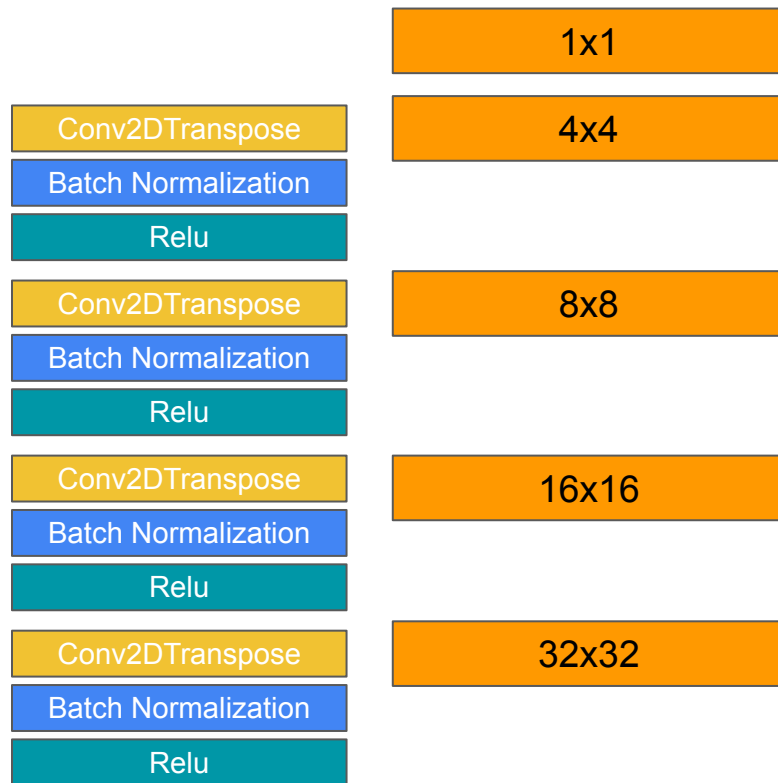
Relu

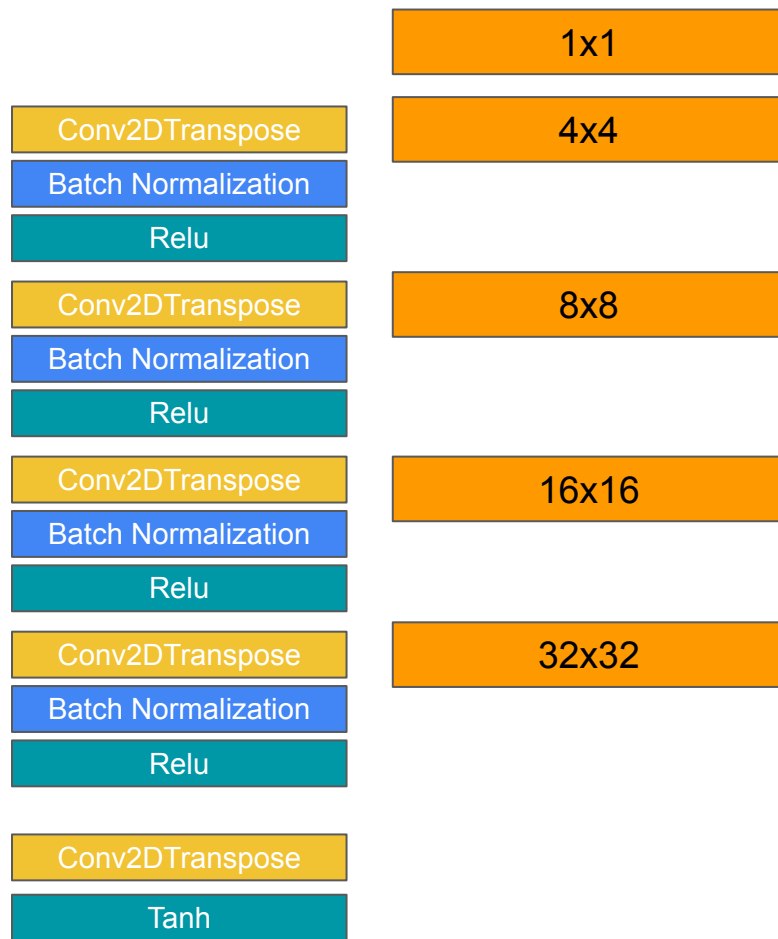
Conv2DTranspose

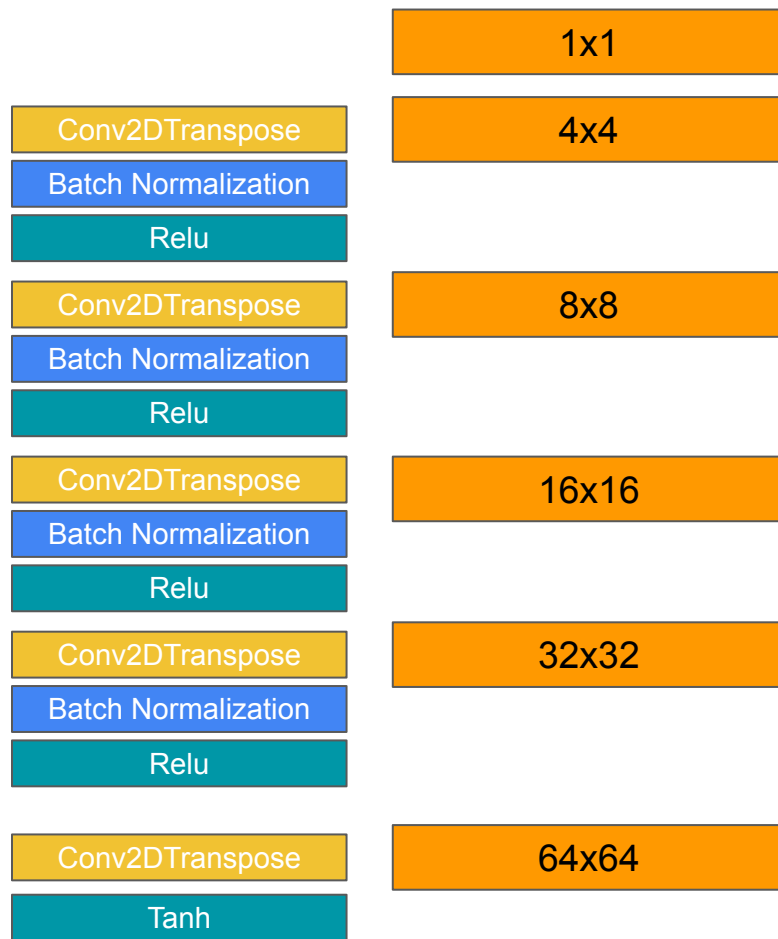
Batch Normalization

Relu









```
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)
```

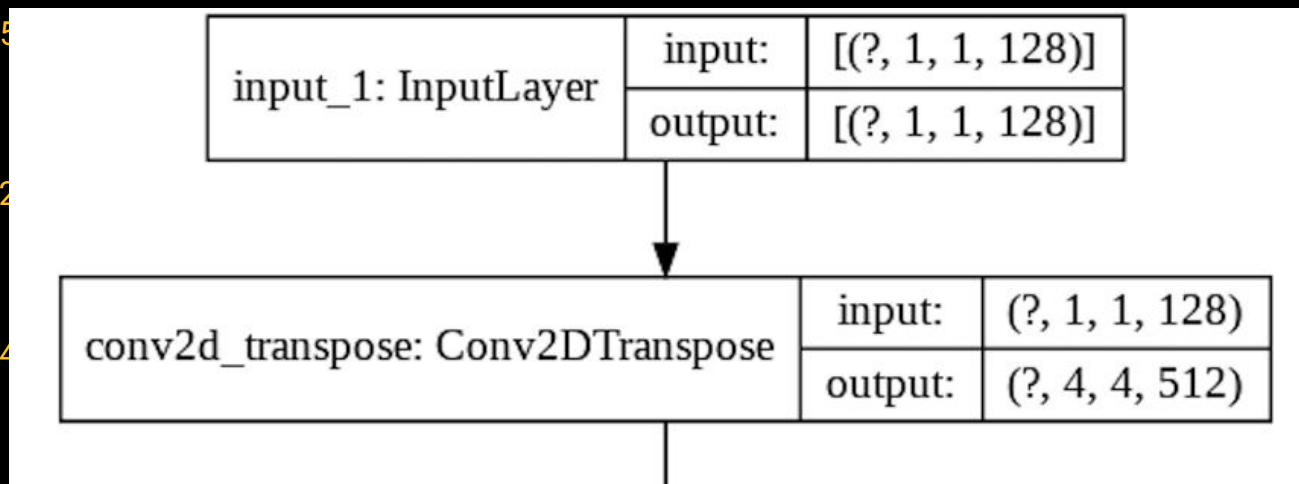
```
x = layers.Conv2DTranspose(512, 4, strides=1, padding='valid', use_bias=False)(x)  
x = BatchNormalization()(x)  
x = layers.ReLU()(x)
```

```
x = layers.Conv2DTranspose(256, 4, strides=1, padding='valid', use_bias=False)(x)  
x = BatchNormalization()(x)  
x = layers.ReLU()(x)
```

```
x = layers.Conv2DTranspose(128, 4, strides=1, padding='valid', use_bias=False)(x)  
x = BatchNormalization()(x)  
x = layers.ReLU()(x)
```

```
x = layers.Conv2DTranspose(64, 4, strides=1, padding='valid', 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(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)
```

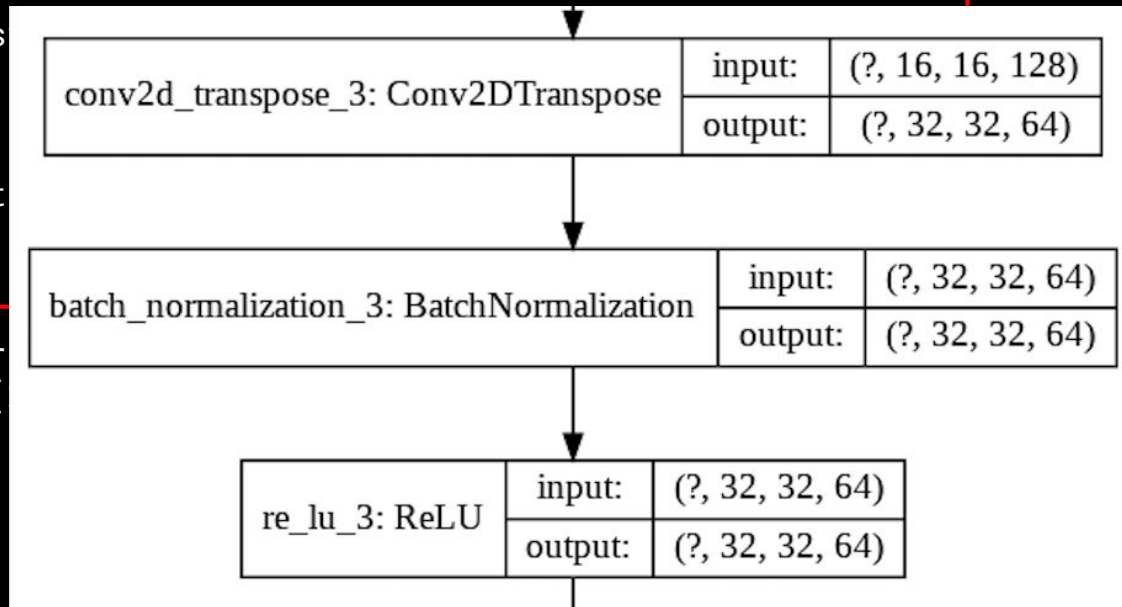
```
x = layers.Conv2DTranspose(64, 4, st
```

```
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)
```

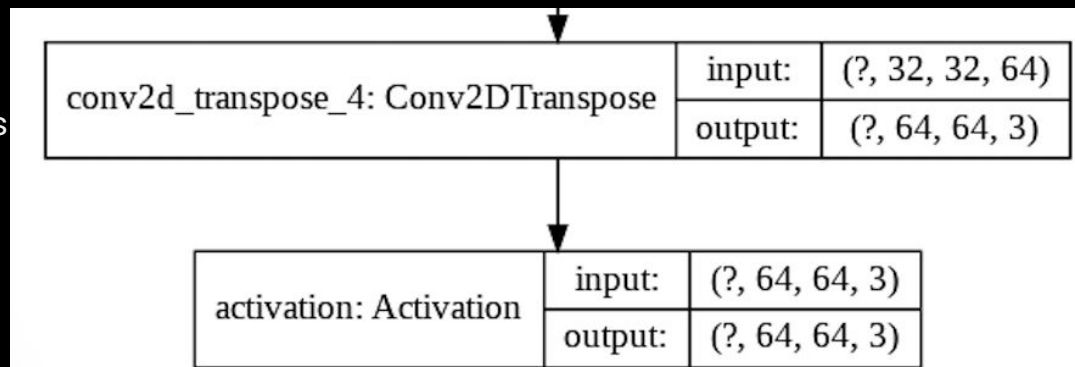
```
x = layers.ReLU()(x)
```

```
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=2, padding='same', use_bias=False)(x)  
outputs = layers.Activation('tanh')(x)
```





Conv2D

The diagram consists of three stacked rectangular blocks. The top block is yellow and labeled 'Conv2D'. The middle block is blue and labeled 'Batch Normalization'. The bottom block is teal and labeled 'Leaky Relu'. All blocks have a thin black border and are centered horizontally.

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

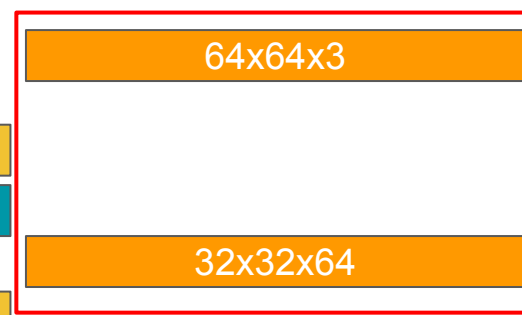
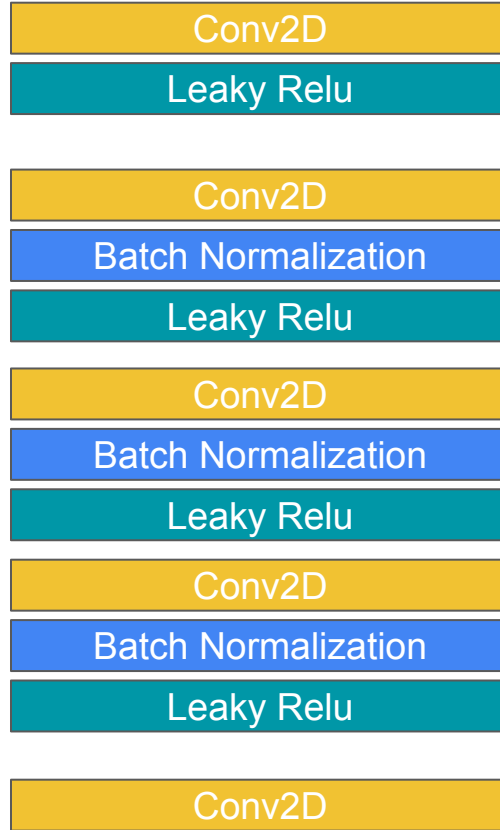
Leaky Relu

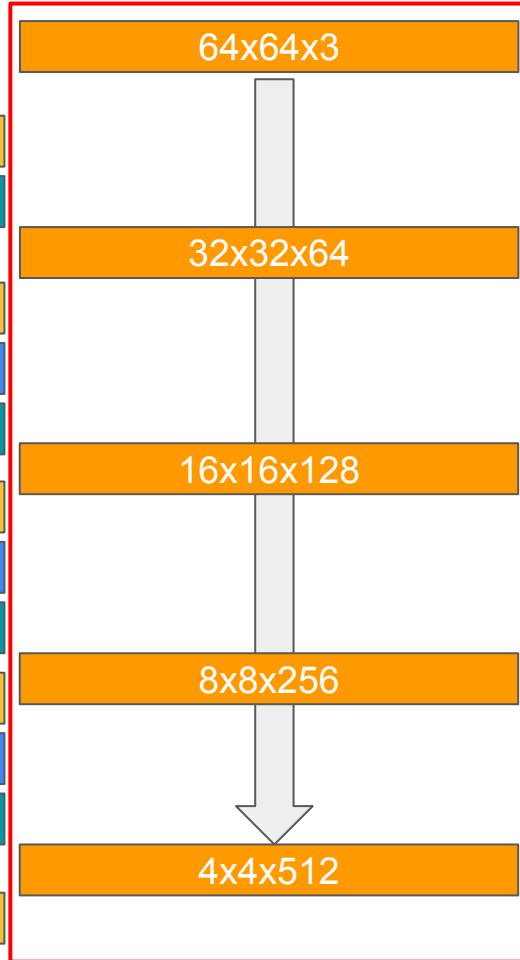
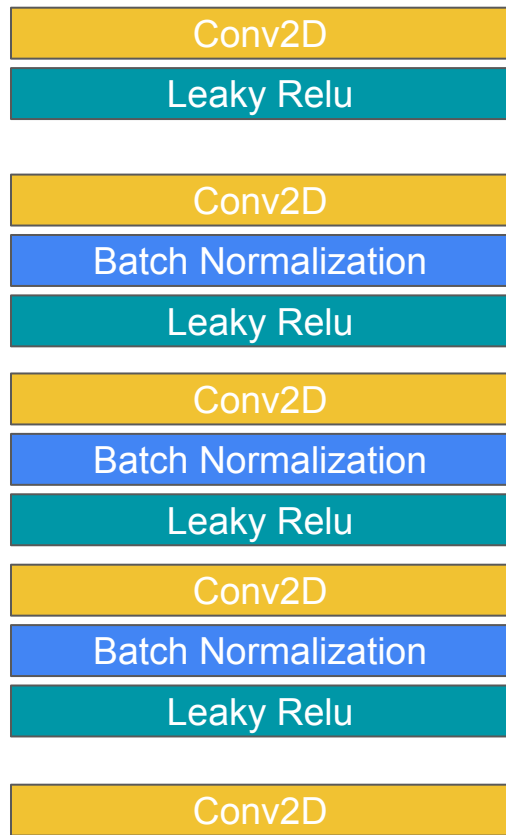
Conv2D

Batch Normalization

Leaky Relu

Conv2D





Conv2D

Leaky Relu

Conv2D

Batch Normalization

Leaky Relu

Conv2D

Batch Normalization

Leaky Relu

Conv2D

Batch Normalization

Leaky Relu

Conv2D

4x4x512

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)
```