Copyright Notice

What are AutoEncoders?

- Neural networks capable of learning dense representations of input data without supervision
  - Training data is not labelled
- Useful for dimensionality reduction and for visualization
- Can be used to generate new data that resembles input data
- In practice they
  - Copy input to output
  - They learn efficient ways to represent data

What are AutoEncoders?

- Neural networks capable of learning dense representations of input data without supervision
  - Training data is not labelled
- Useful for dimensionality reduction and for visualization
- Can be used to generate new data that resembles input data
- In practice they
  - Copy input to output
  - They learn efficient ways to represent data

What are AutoEncoders?

- Neural networks capable of learning dense representations of input data without supervision
  - Training data is not labelled
- Useful for dimensionality reduction and for visualization
- Can be used to generate new data that resembles input data
- In practice they
  - Copy input to output
  - They learn efficient ways to represent data

What are AutoEncoders?

- Neural networks capable of learning dense representations of input data without supervision
  - Training data is not labelled
- Useful for dimensionality reduction and for visualization
- Can be used to generate new data that resembles input data
- In practice they
  - Copy input to output
  - They learn efficient ways to represent data

What are AutoEncoders?

- Neural networks capable of learning dense representations of input data without supervision
  - Training data is not labelled
- Useful for dimensionality reduction and for visualization
- Can be used to generate new data that resembles input data
- In practice they
  - Copy input to output
  - They learn efficient ways to represent data

83 12 21 42 99 18 51

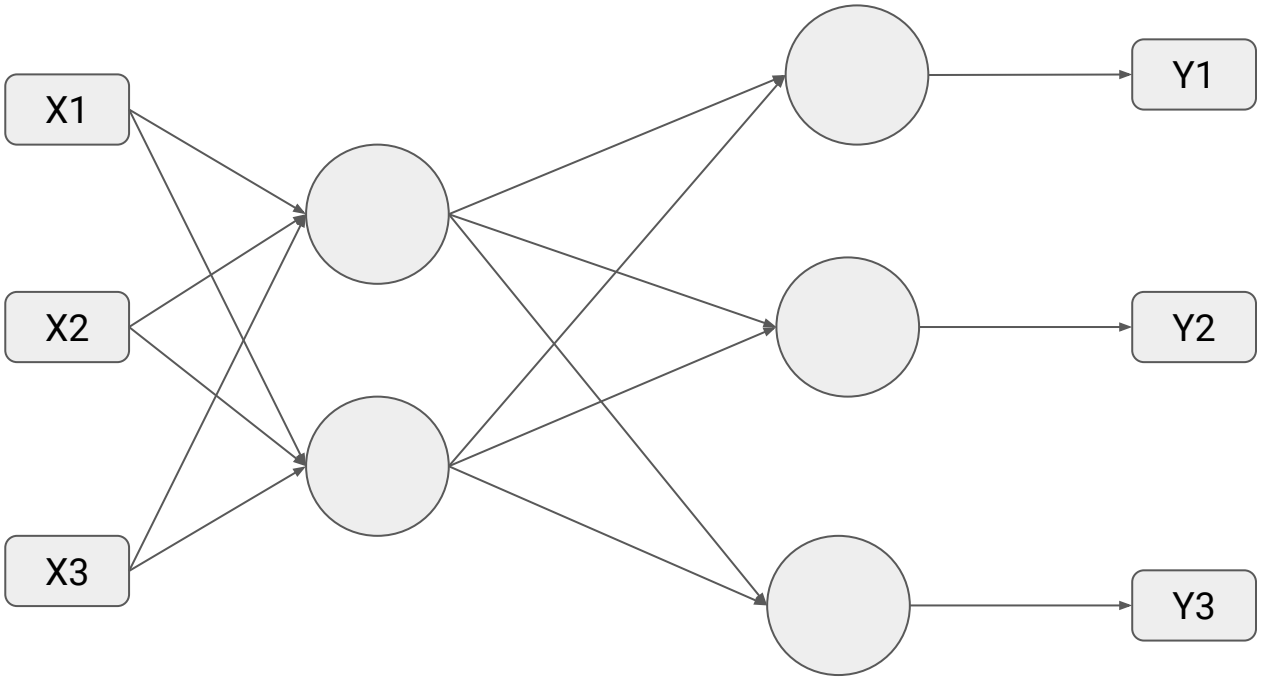
4 9 16 25 36 49 64 81 100 121 144 169

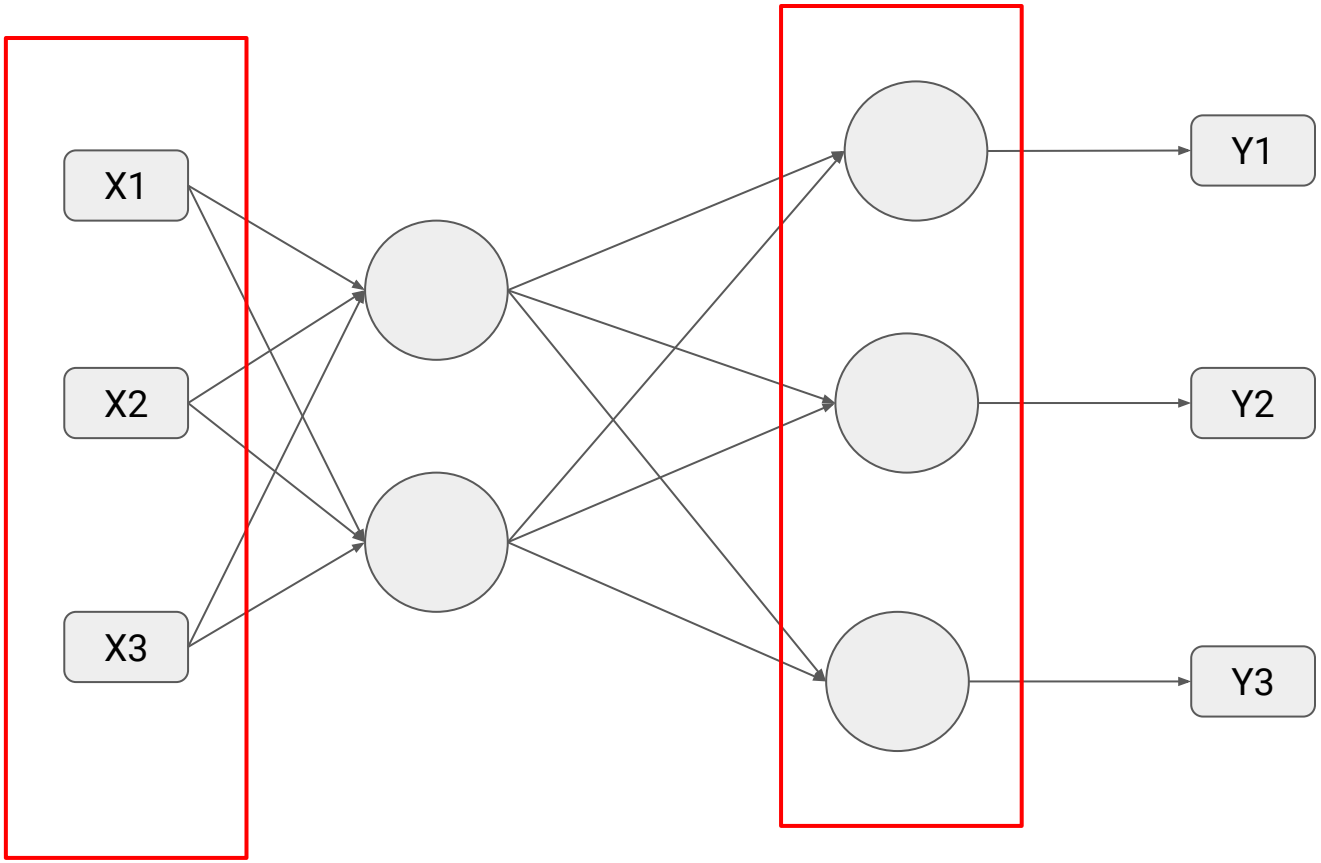
83 12 21 42 99 18 51

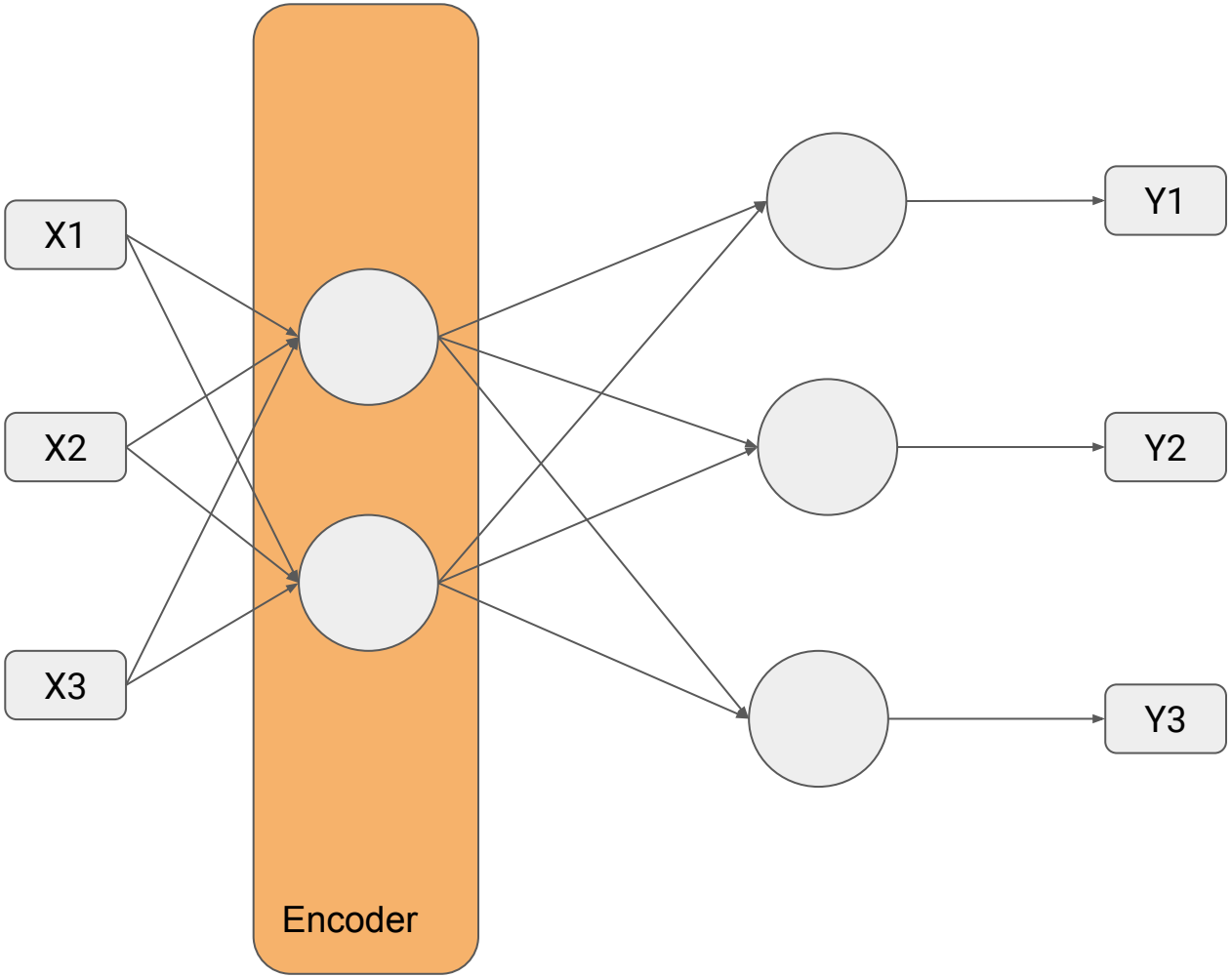
4 9 16 25 36 49 64 81 100 121 144 169

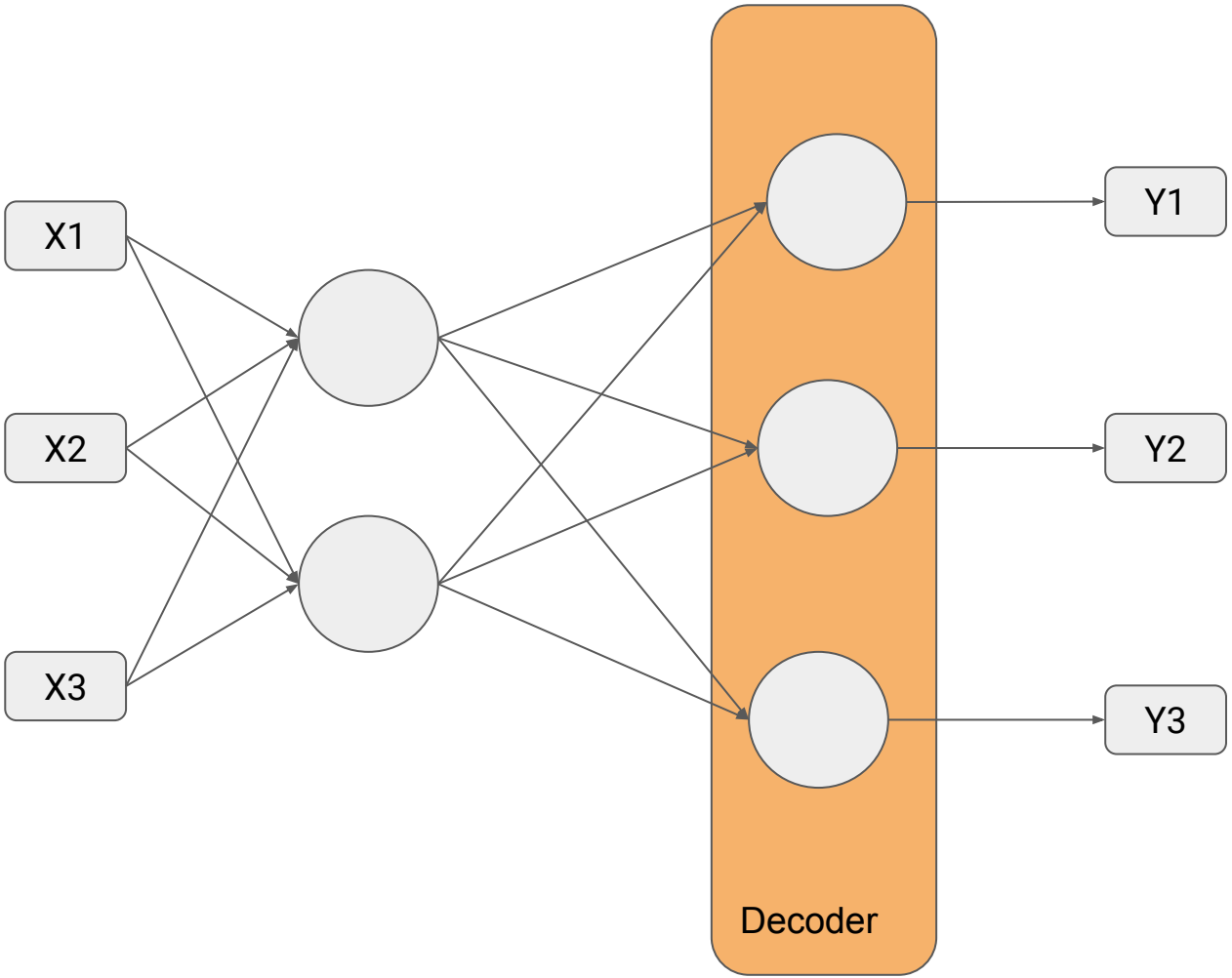
83 12 21 42 99 18 51

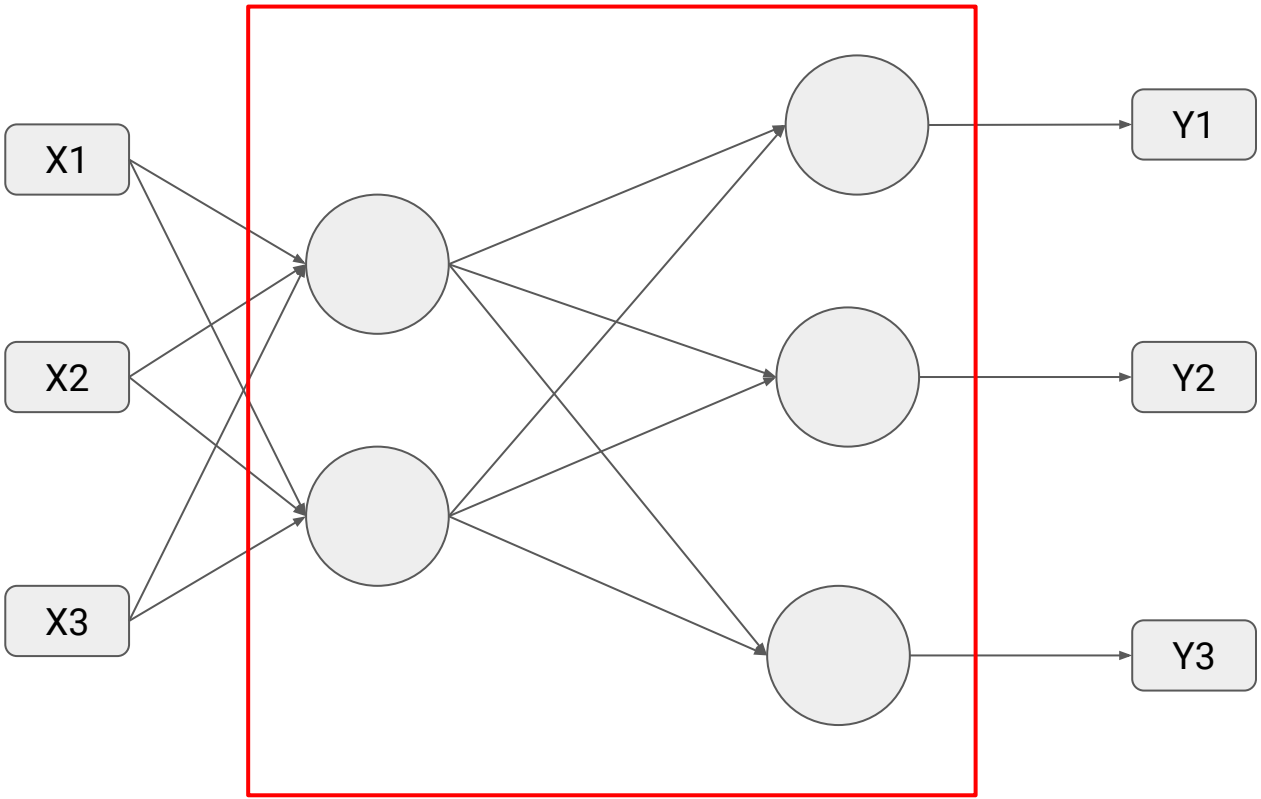
4 9 16 25 36 49 64 81 100 121 144 169

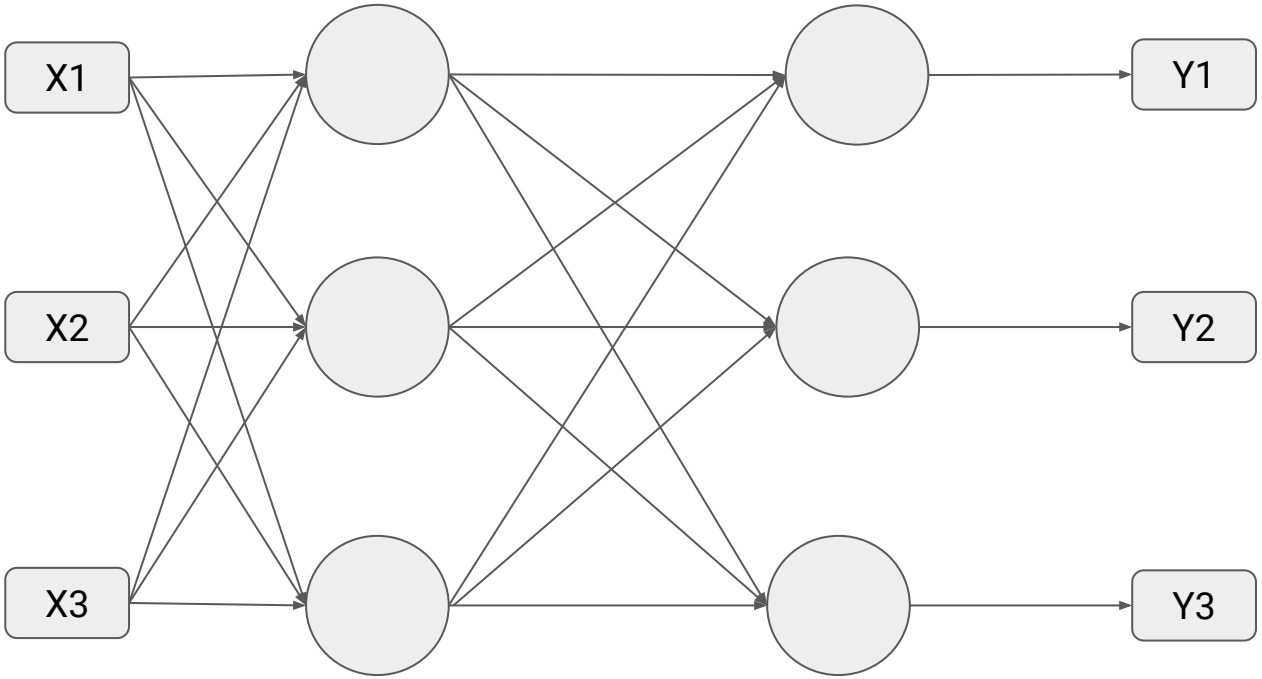

X1
X2
X3
Y1
Y2
Y3

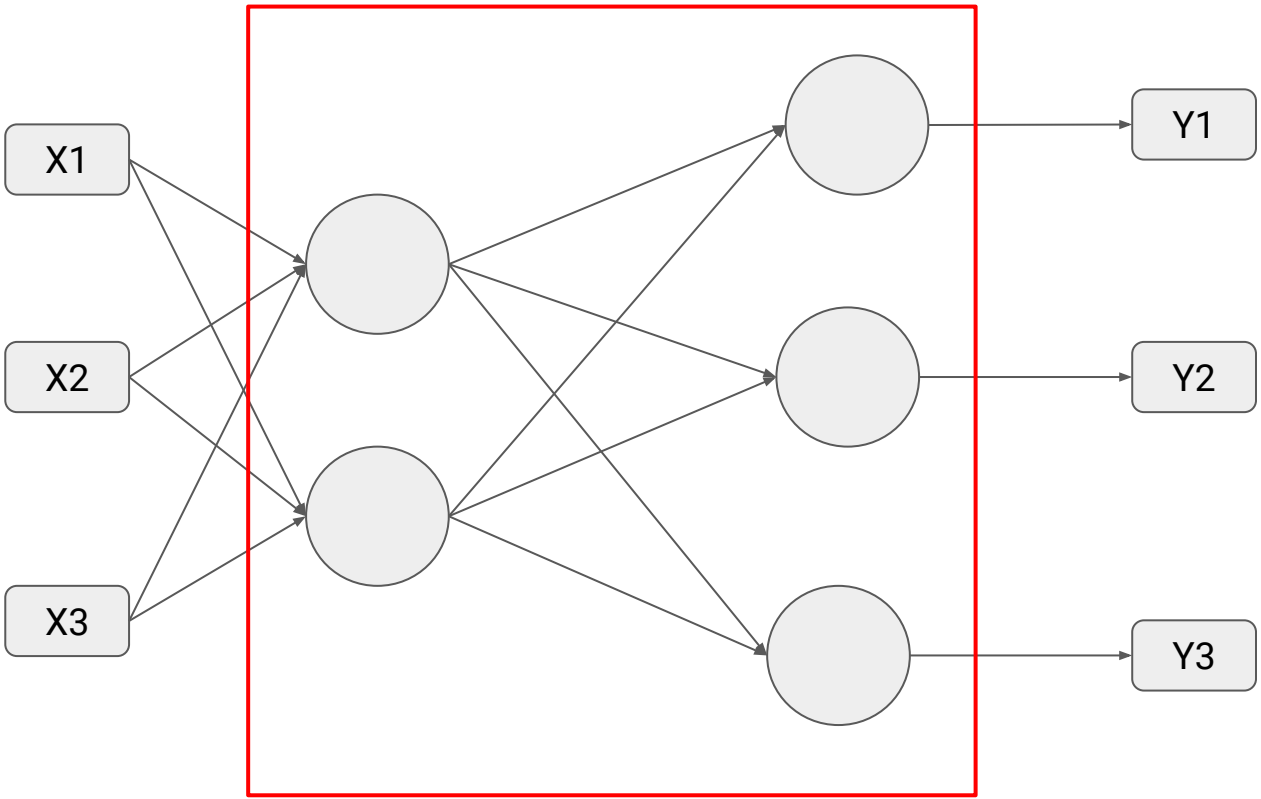

X1
X2
X3
Y1
Y2
Y3

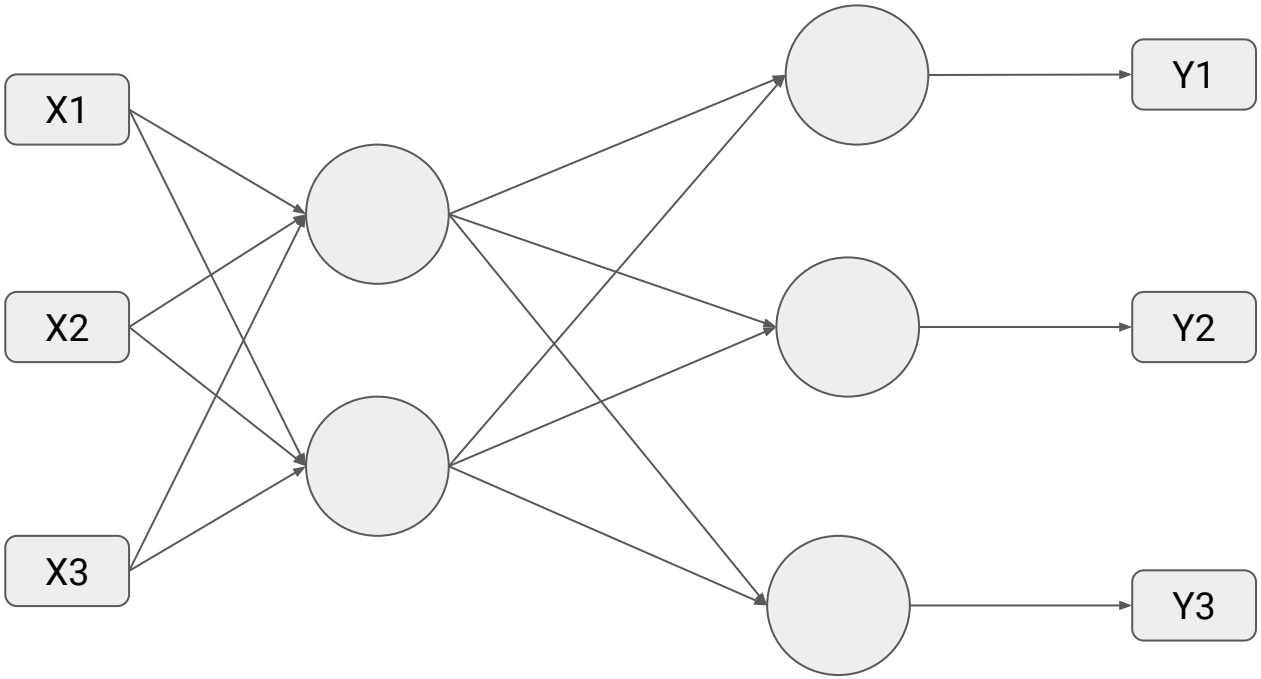

X1
X2
X3
Encoder
Y1
Y2
Y3

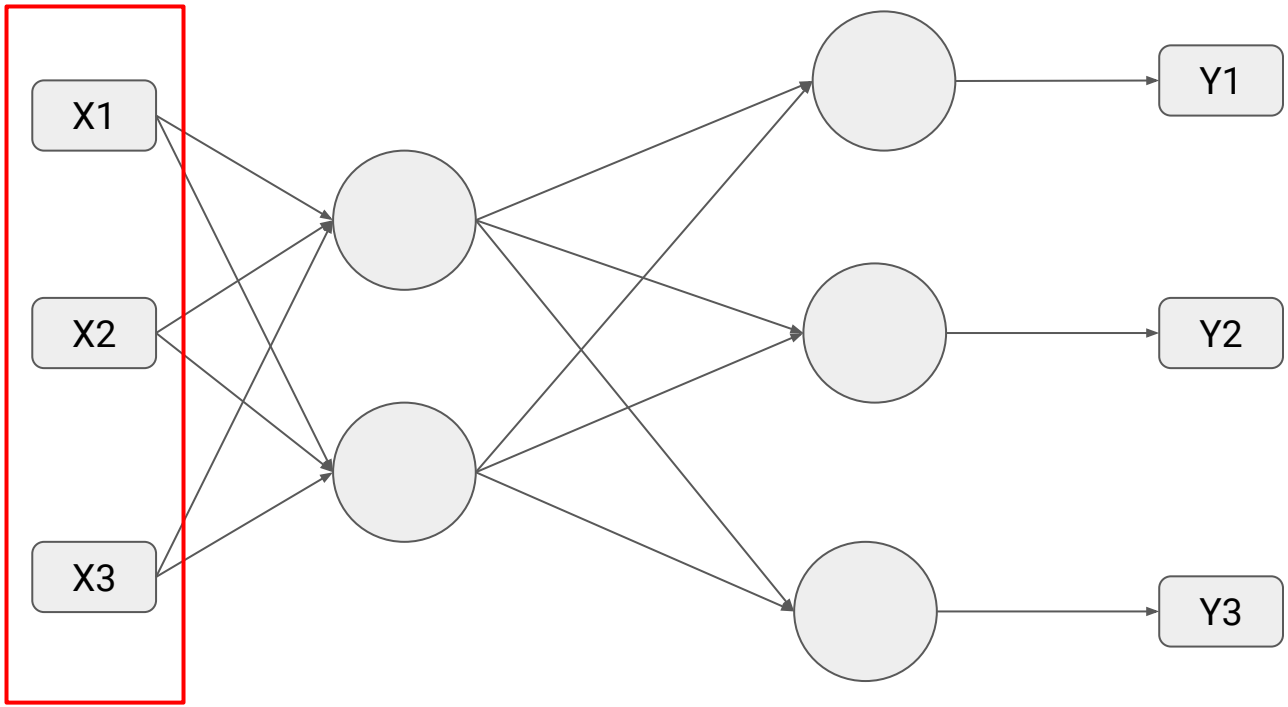

X1
X2
X3
Y1
Y2
Y3
Decoder

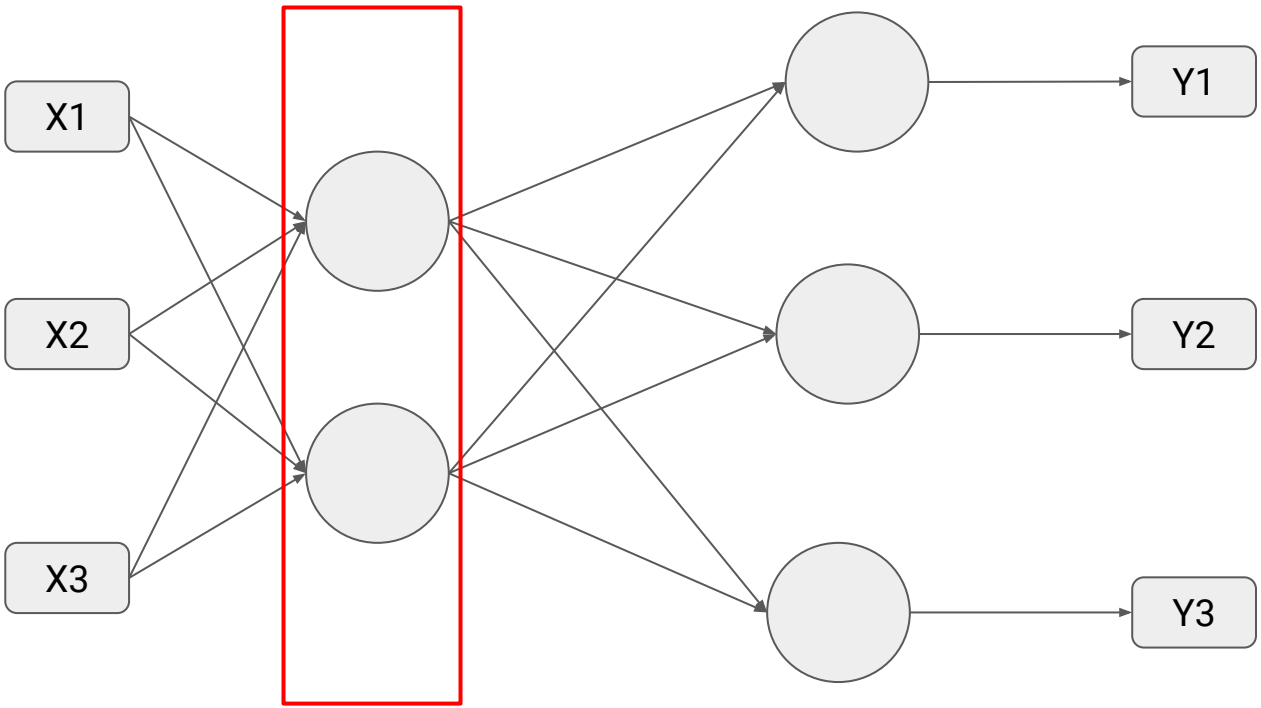

X1
X2
X3
Y1
Y2
Y3

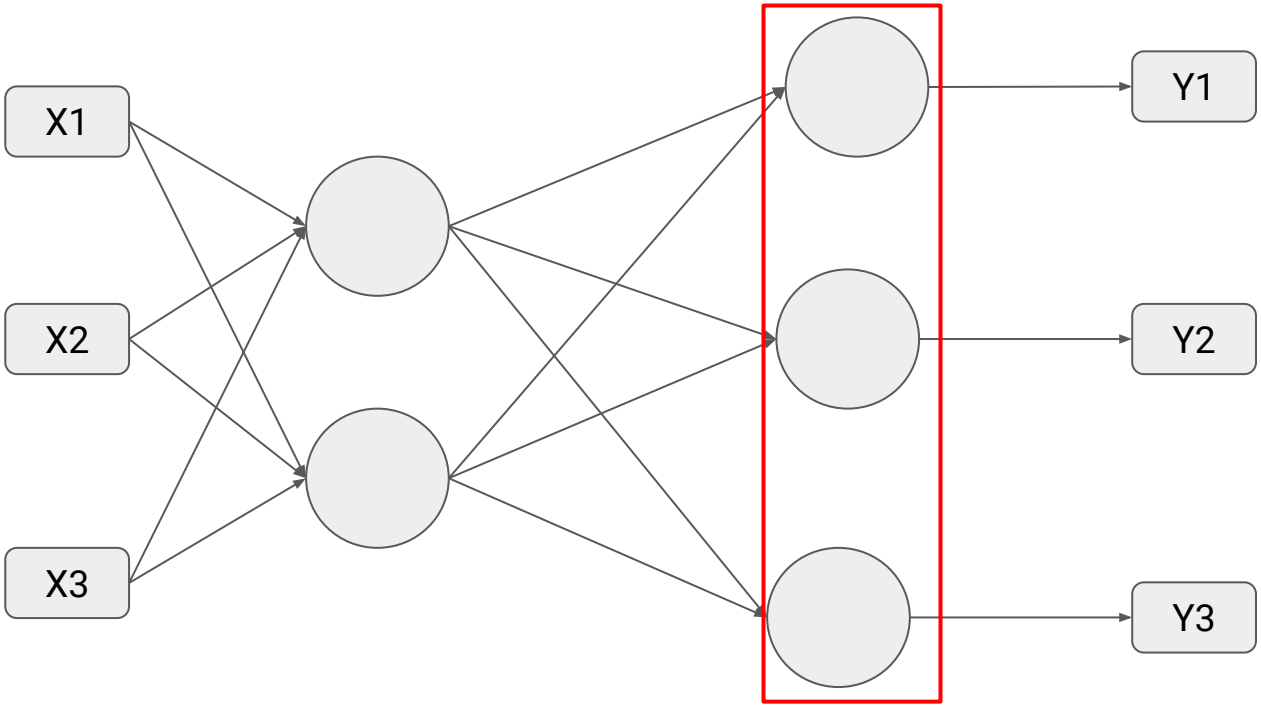

X1
X2
X3
Y1
Y2
Y3

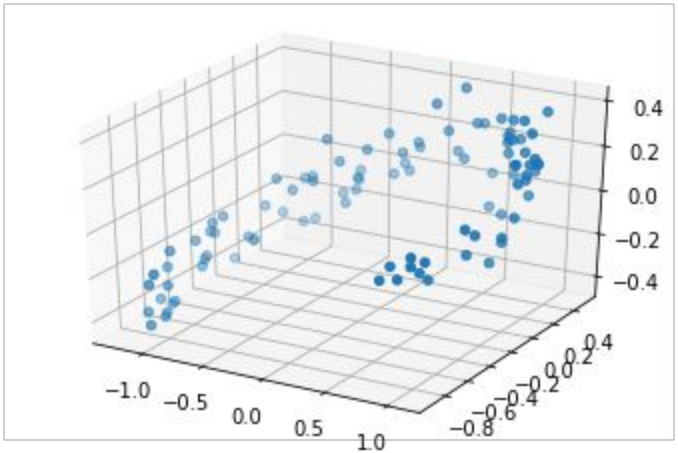

X1
X2
X3
Y1
Y2
Y3

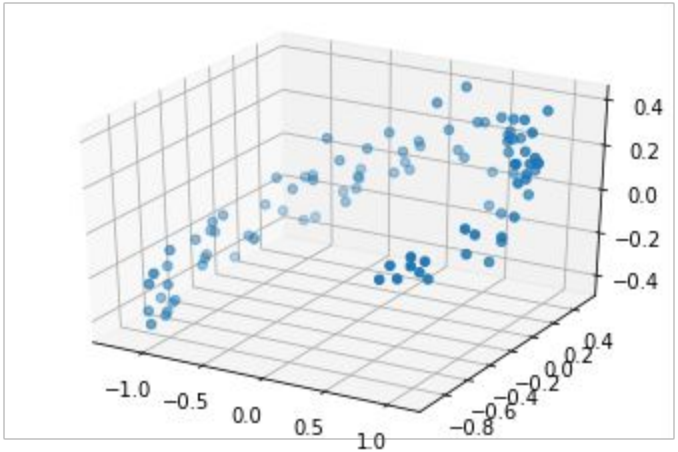

X1
X2
X3
Y1
Y2
Y3

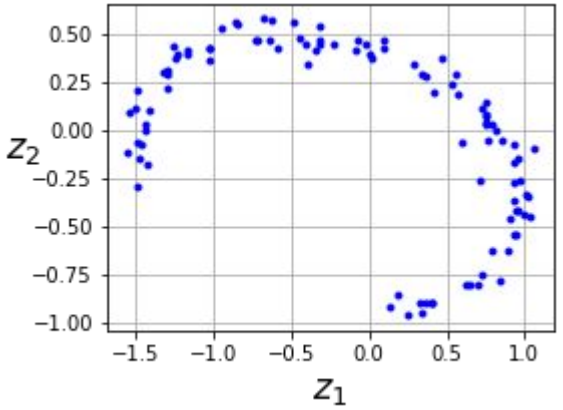

X1
X2
X3
Y1
Y2
Y3

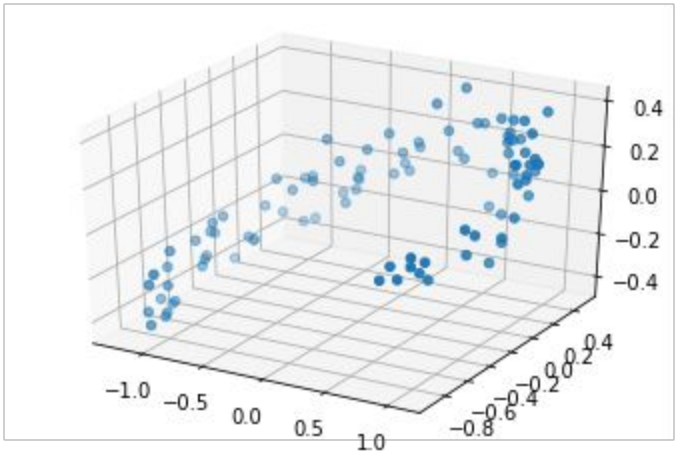

X1
X2
X3
Y1
Y2
Y3

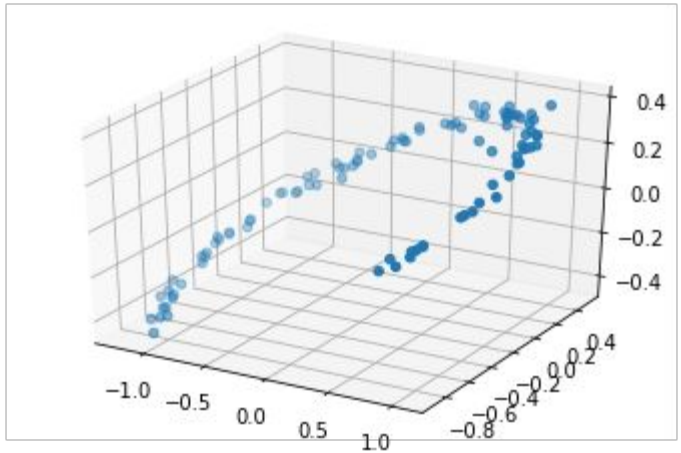

X1
X2
X3
Y1
Y2
Y3


0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

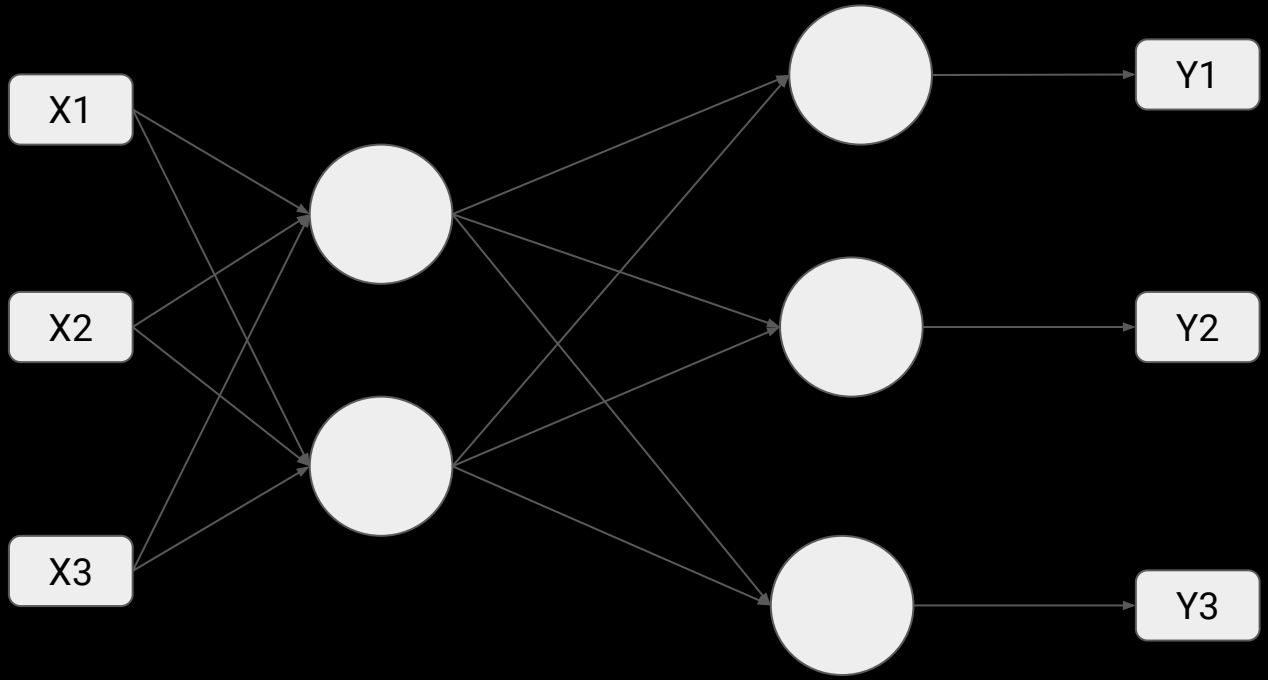

0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8


0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
z_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
z_1

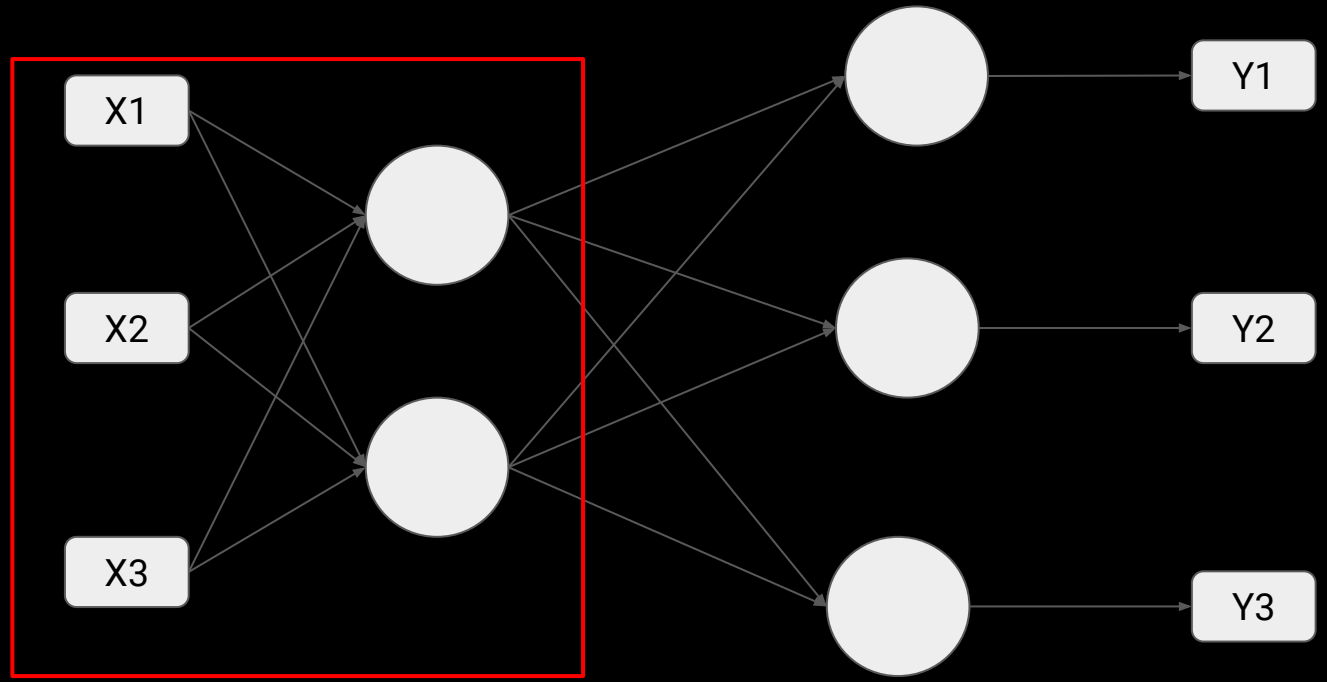

0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8


0.4
0.2
0.0
−0.2
−0.4
−1.0
−0.5
0.0
0.5
1.0
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

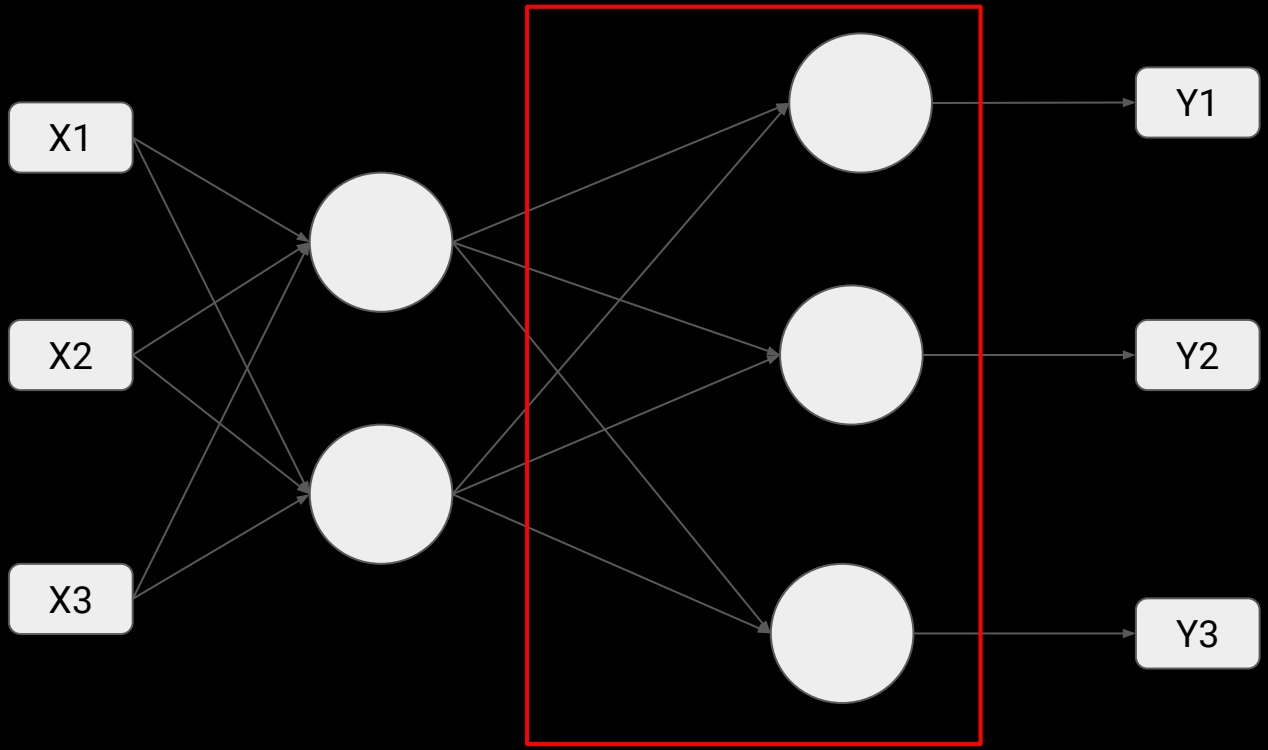
```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])

decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])

autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1.5))
```



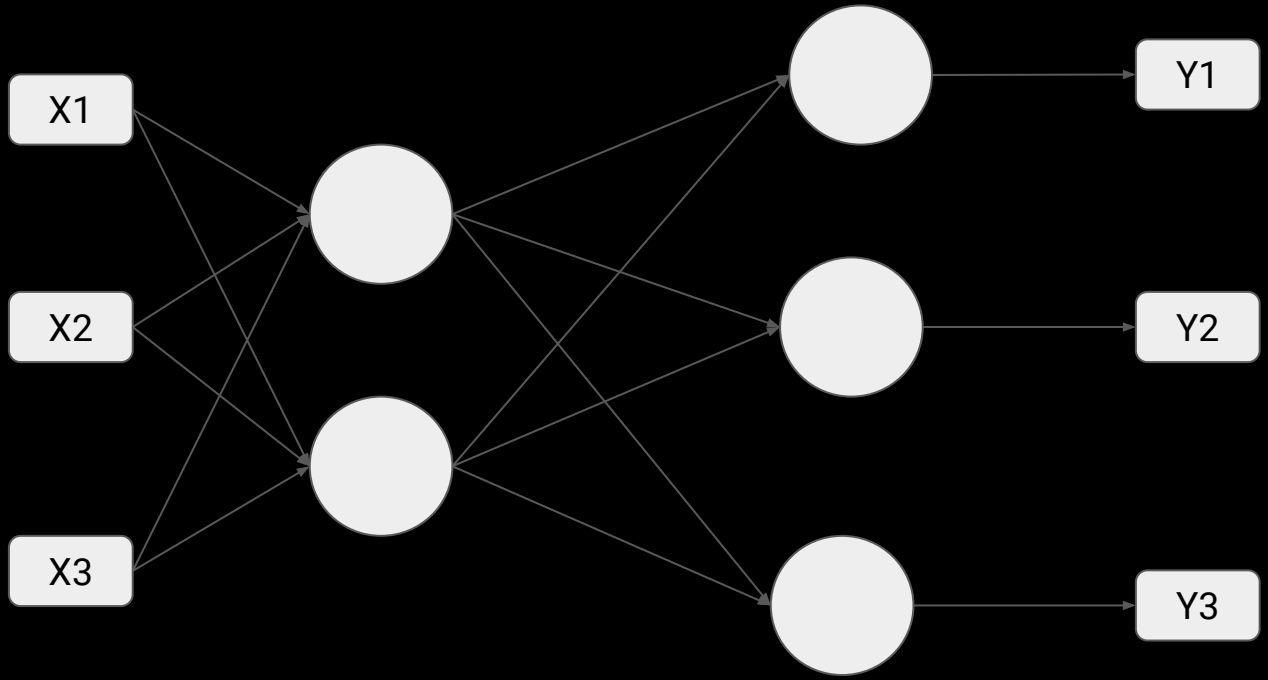
```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])

decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])

autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1.5))
```



```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])
decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])
autoencoder = keras.models.Sequential([encoder, decoder])
autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1.5))
```



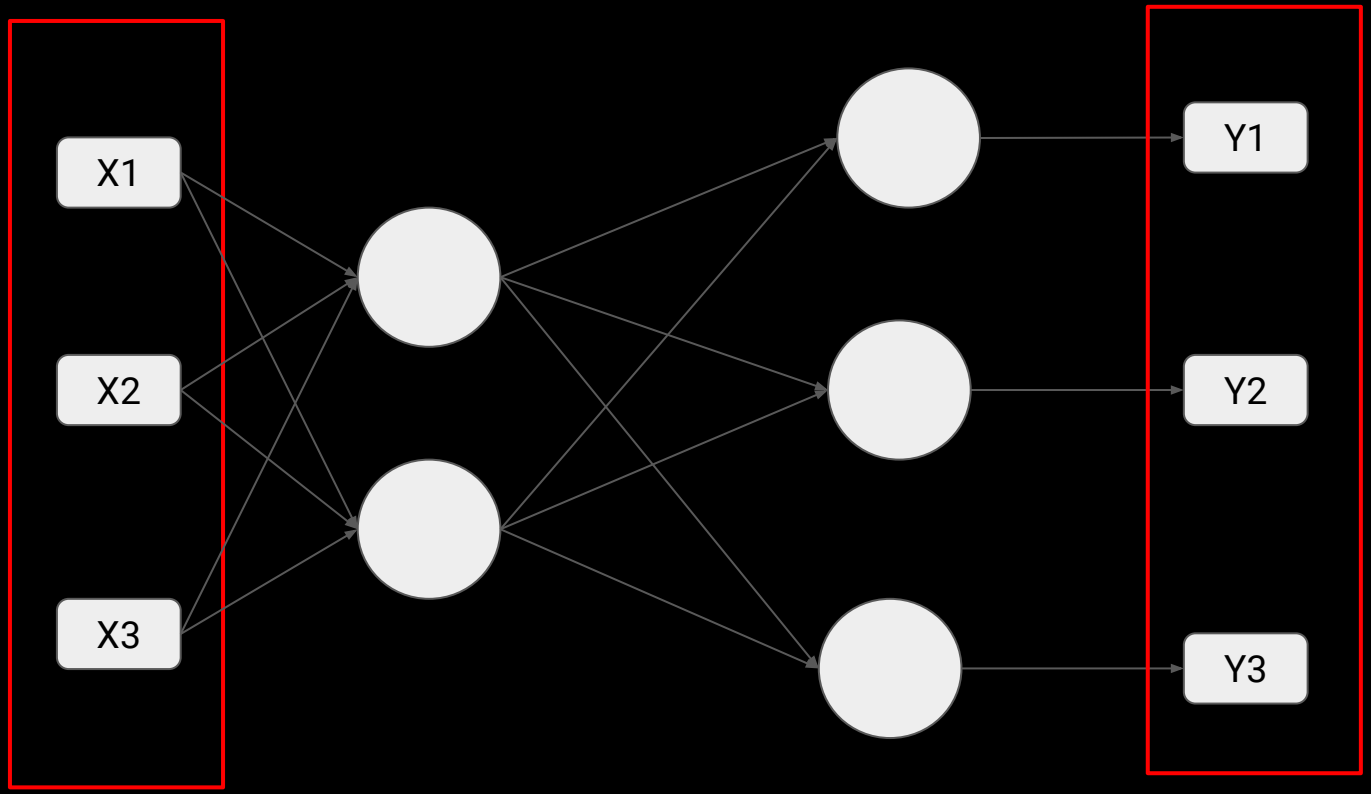
```
encoder = keras.models.Sequential([keras.layers.Dense(2, input_shape=[3])])

decoder = keras.models.Sequential([keras.layers.Dense(3, input_shape=[2])])

autoencoder = keras.models.Sequential([encoder, decoder])

autoencoder.compile(loss="mse", optimizer=keras.optimizers.SGD(lr=1.5))
```



```
history = autoencoder.fit(X_train, X_train, epochs=200)
```

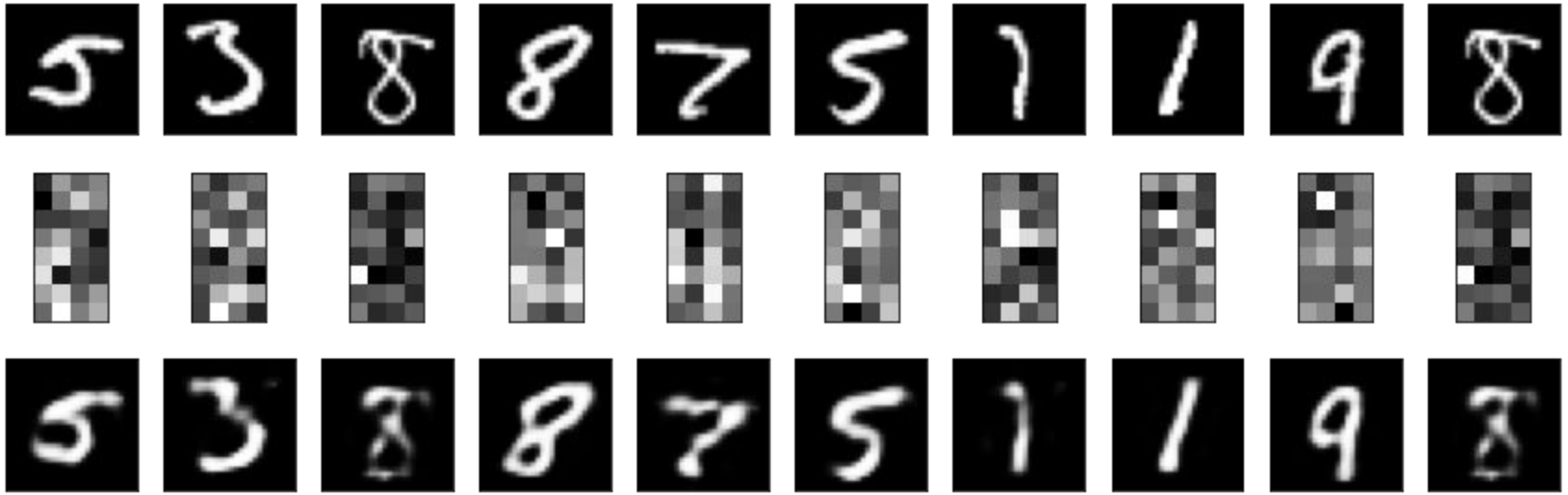
```
history = autoencoder.fit(X_train, X_train, epochs=200)
```

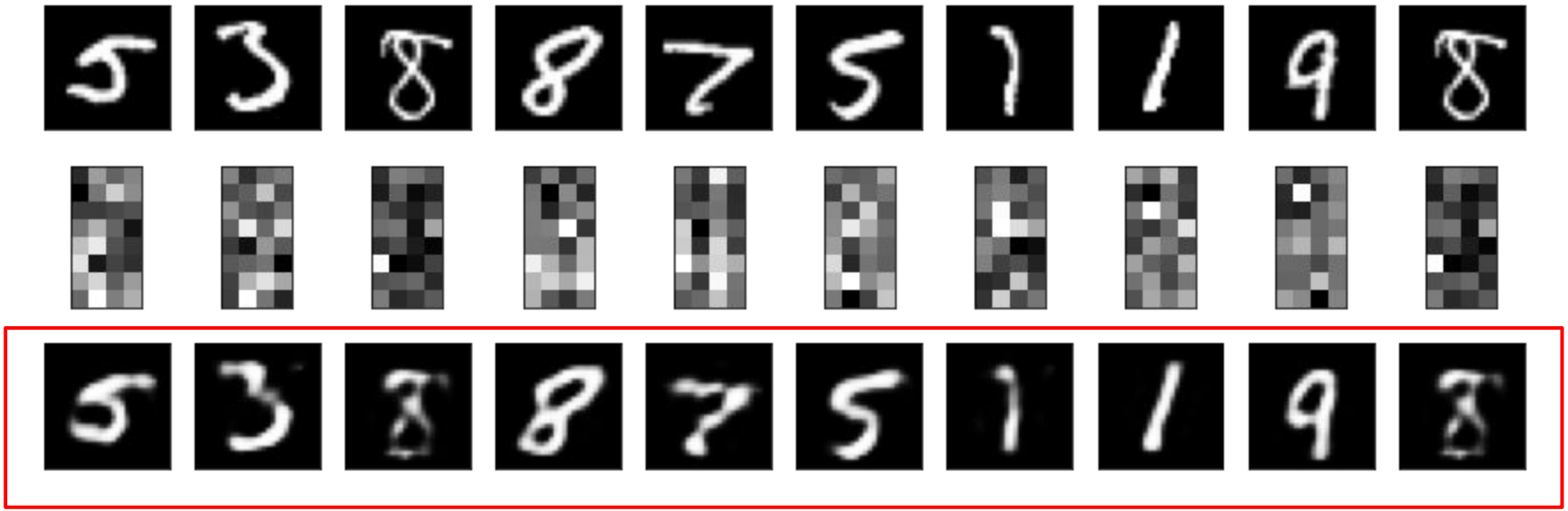
```
history = autoencoder.fit(X_train, X_train, epochs=200)
```



```
codings = encoder.predict(data)
```

```
decodings = decoder.predict(codings)
```





```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

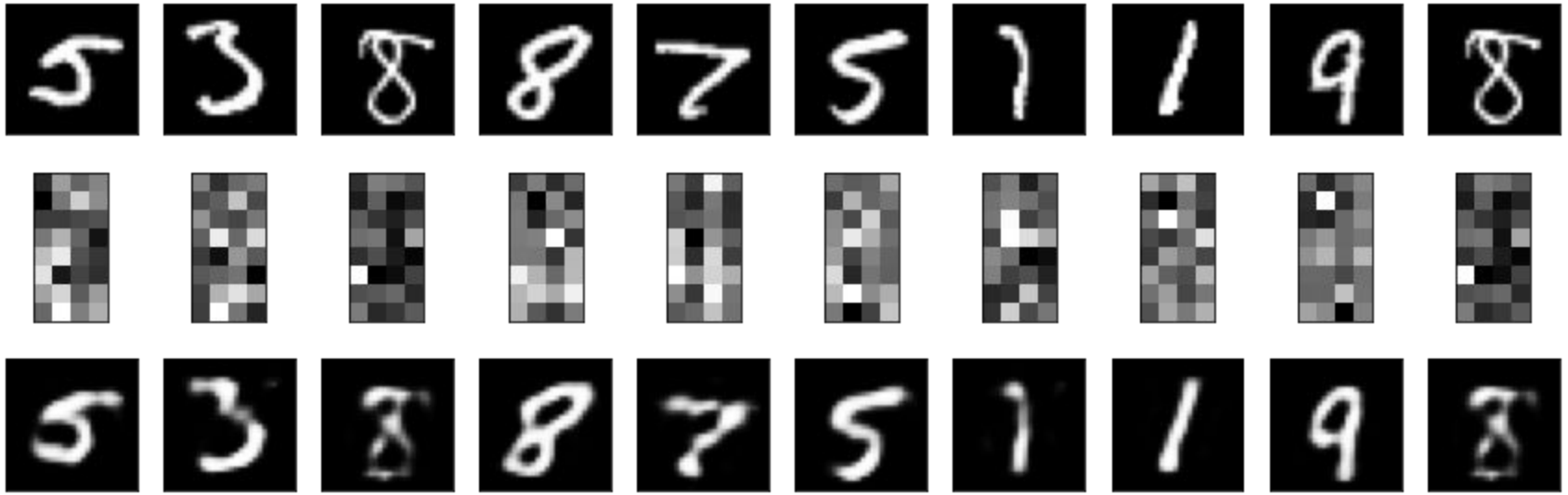
```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

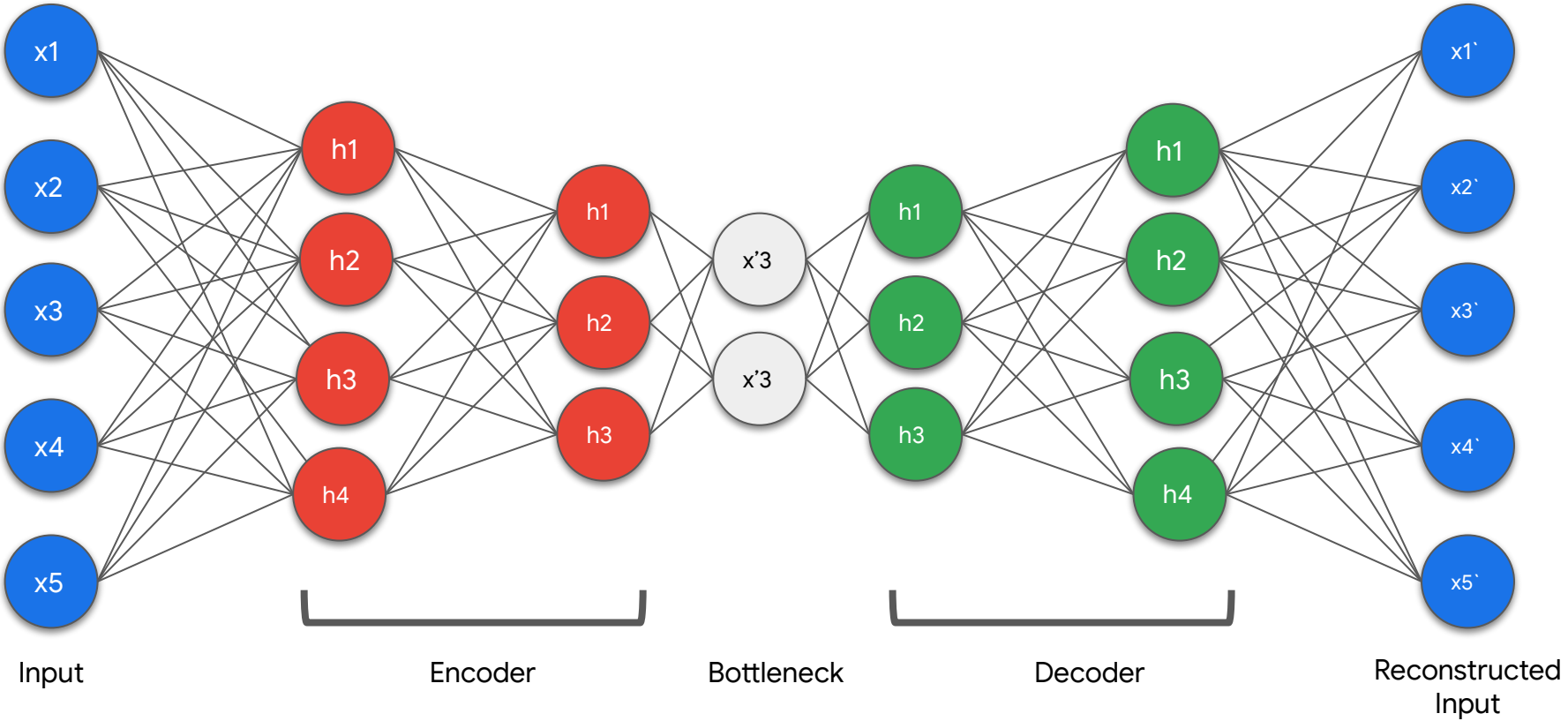
```
inputs =  tf.keras.layers.Input(shape=(784,))

def simple_autoencoder():
    encoder = tf.keras.layers.Dense(units=32, activation='relu')(inputs)
    decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(encoder)
    return encoder, decoder

encoder_output, decoder_output = simple_autoencoder()

encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_output)

autoencoder_model = tf.keras.Model(inputs=inputs, outputs=decoder_output)
```

```
autoencoder_model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss='binary_crossentropy')
```

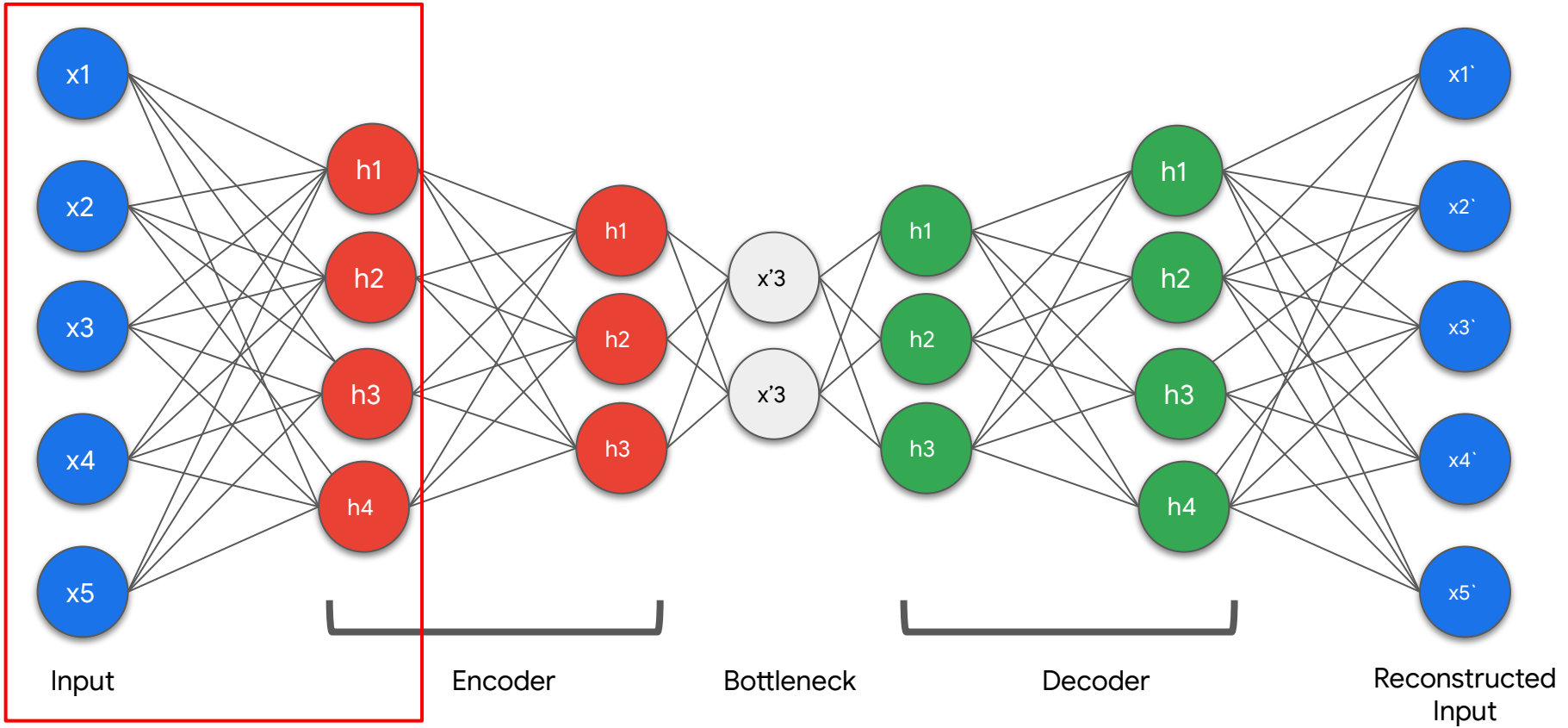
```
autoencoder_model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss='binary_crossentropy')
```



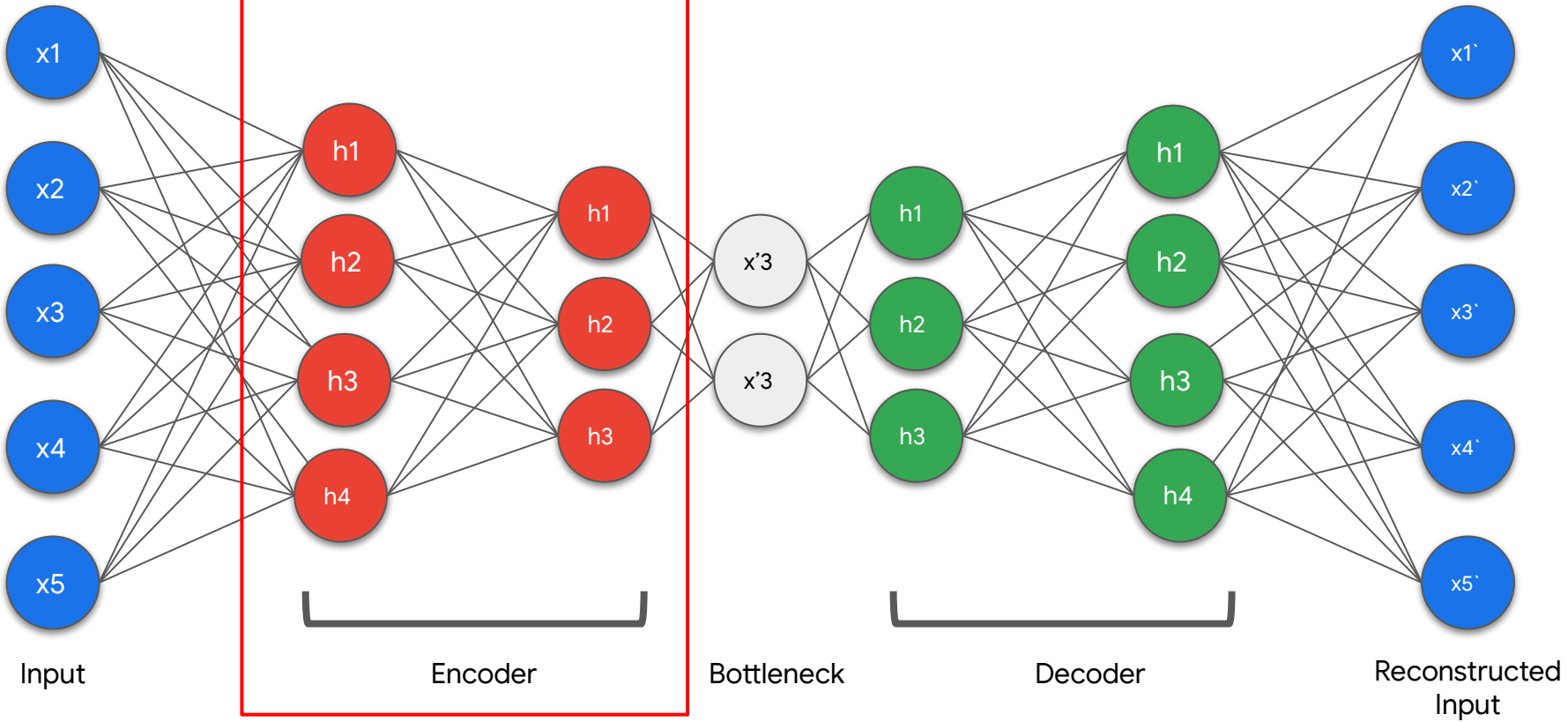
Stacked Auto-Encoders



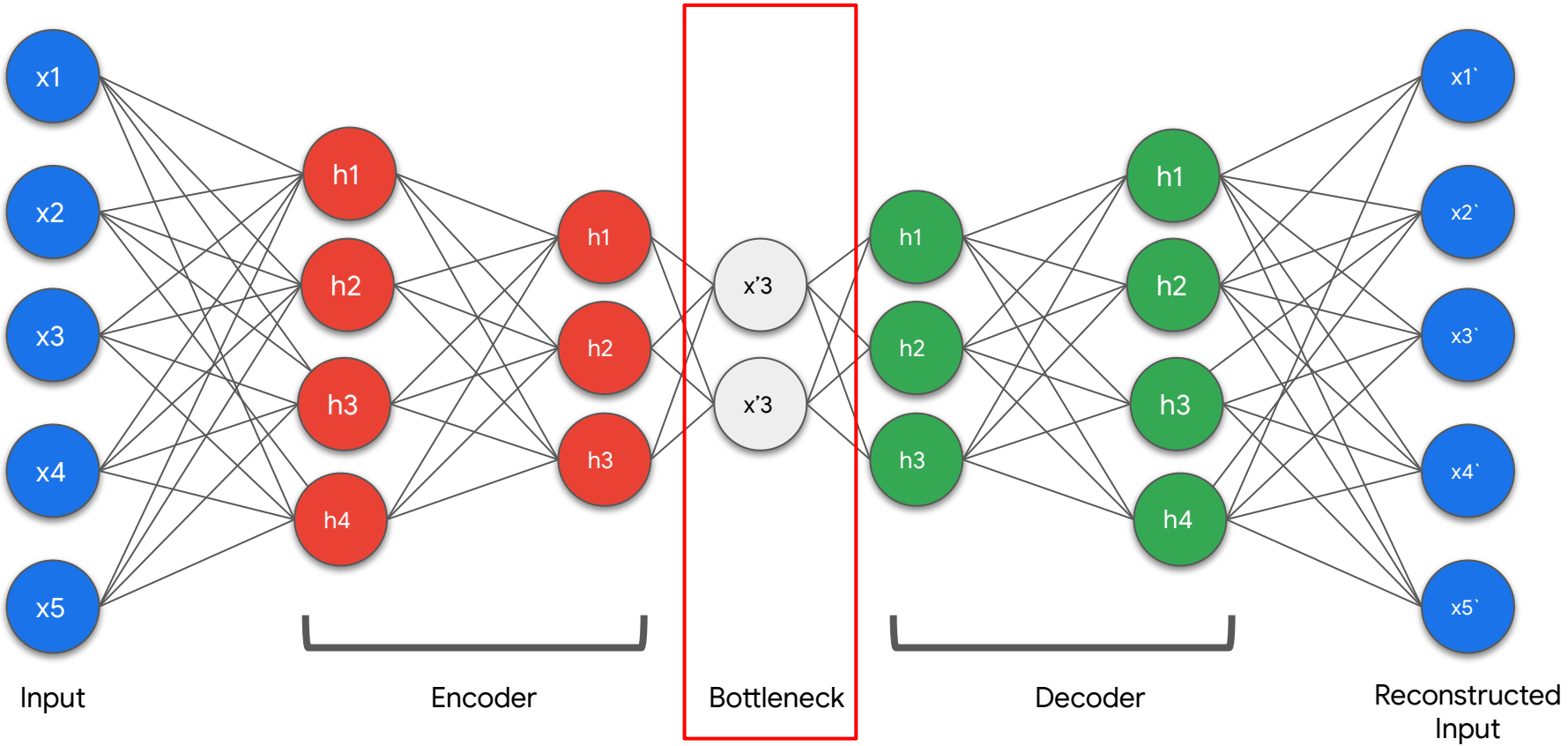
Stacked Auto-Encoders



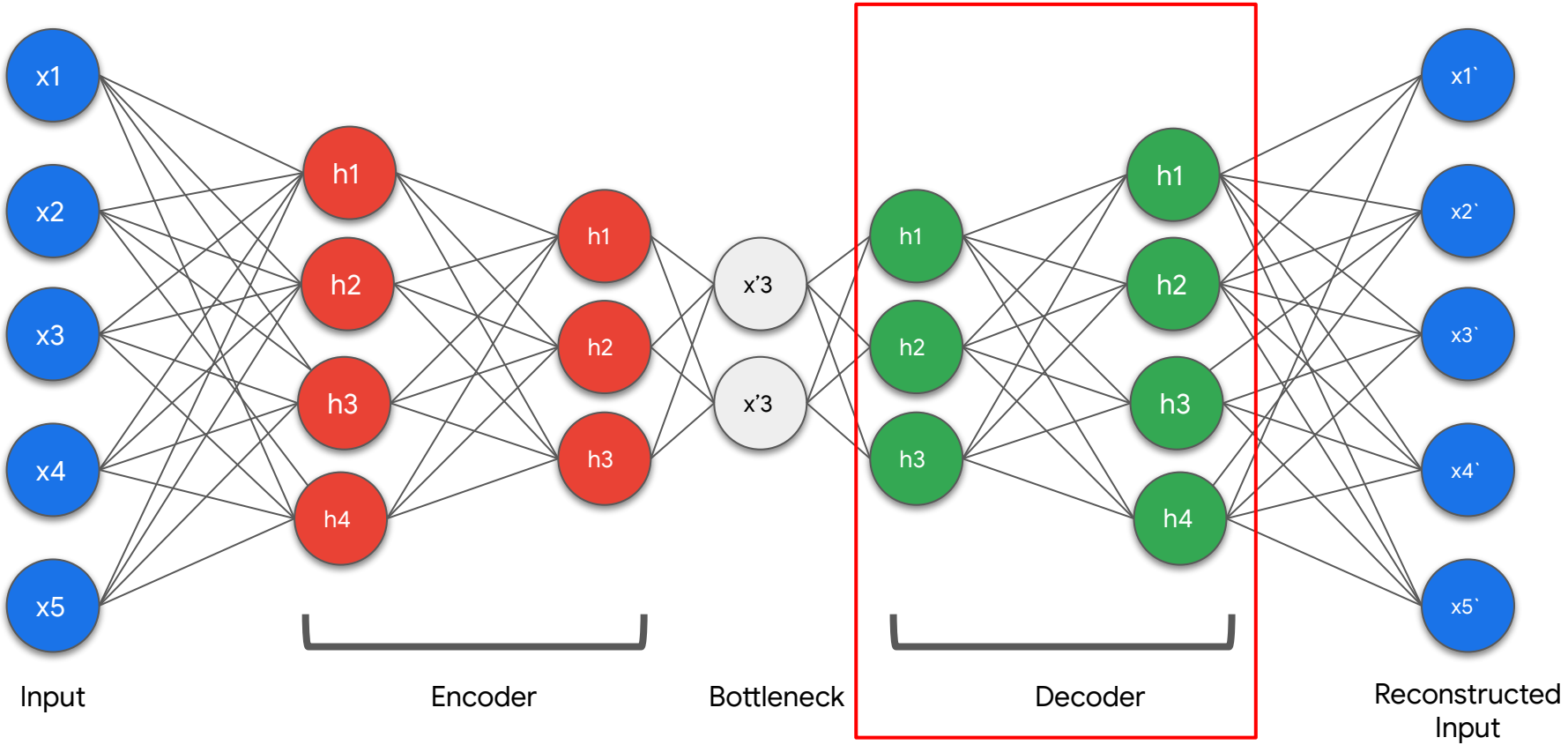
Stacked Auto-Encoders



Stacked Auto-Encoders



Stacked Auto-Encoders

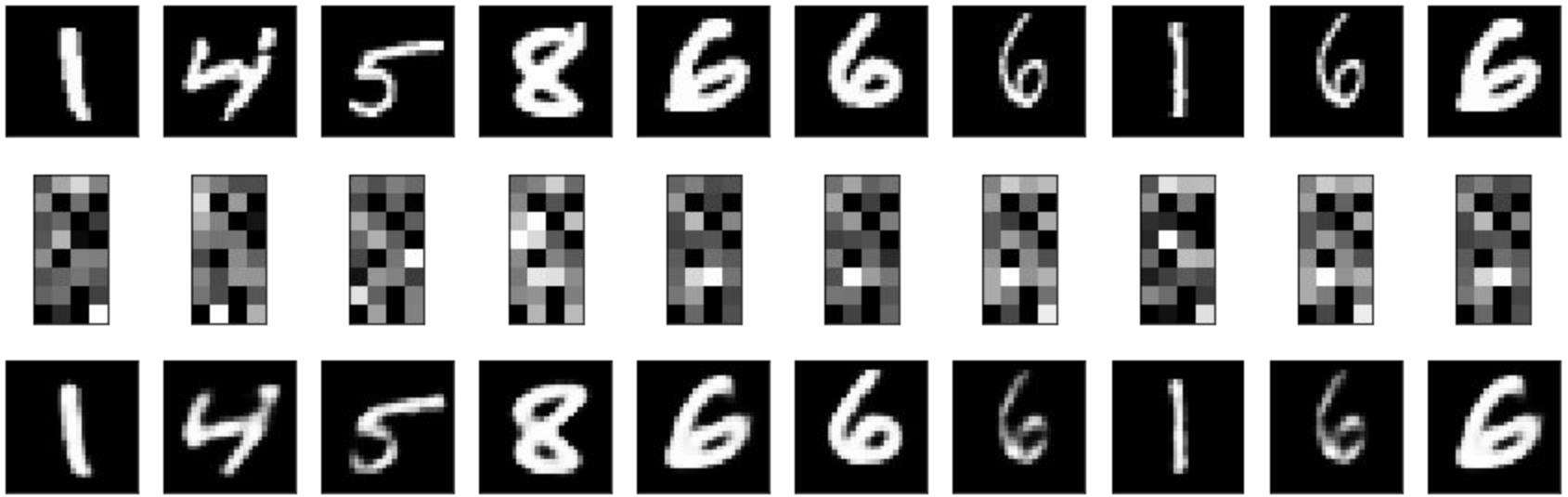


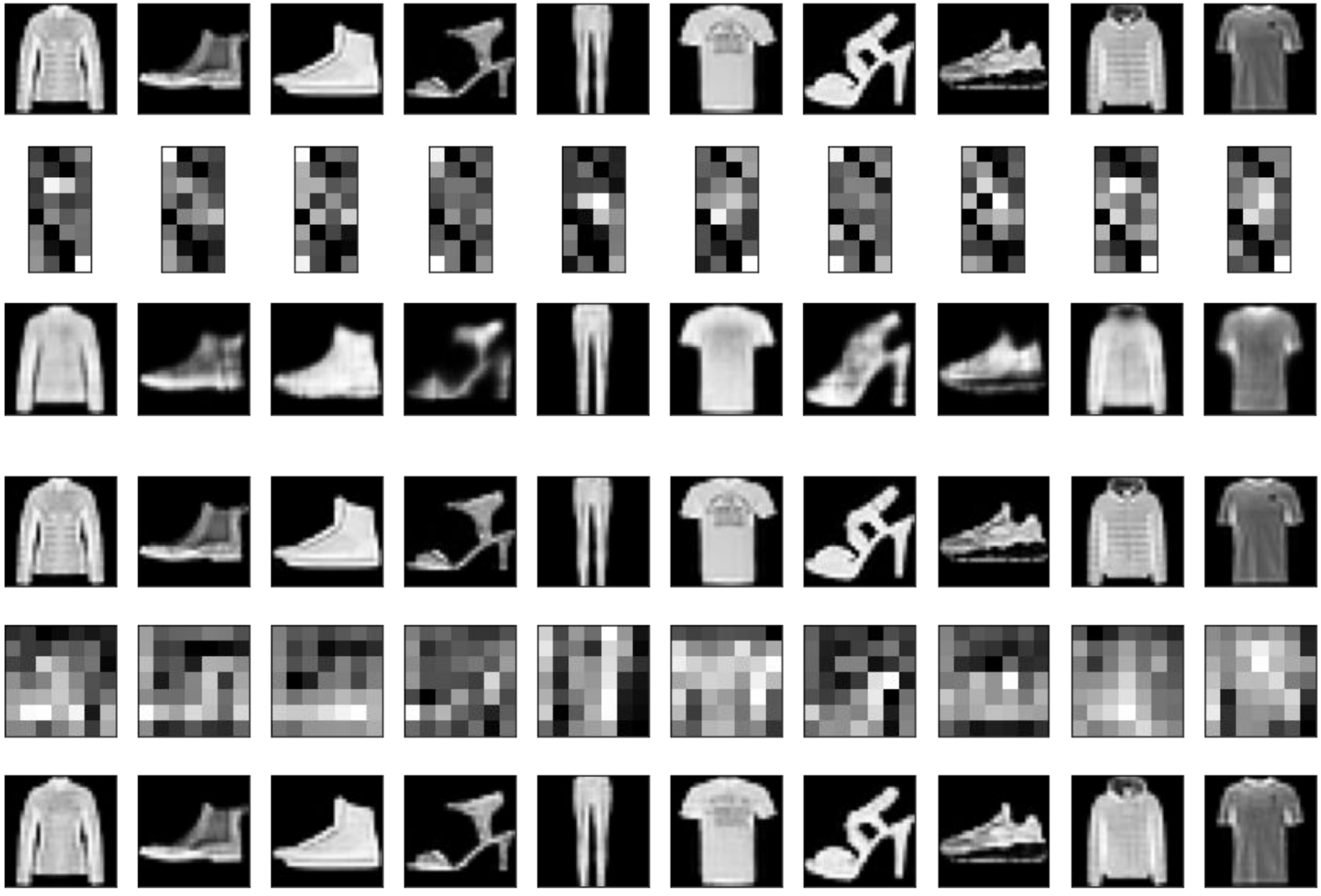
```
inputs =  tf.keras.layers.Input(shape=(784,))

def deep_autoencoder():
 encoder = tf.keras.layers.Dense(units=128, activation='relu')(inputs)
 encoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 encoder = tf.keras.layers.Dense(units=32, activation='relu')(encoder)

 decoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 decoder = tf.keras.layers.Dense(units=128, activation='relu')(decoder)
 decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(decoder)

 return encoder, decoder

deep_encoder_output, deep_autoencoder_output = deep_autoencoder()

deep_encoder_model = tf.keras.Model(inputs=inputs, outputs=deep_encoder_output)
deep_autoencoder_model = tf.keras.Model(inputs=inputs, outputs=deep_autoencoder_output)
```

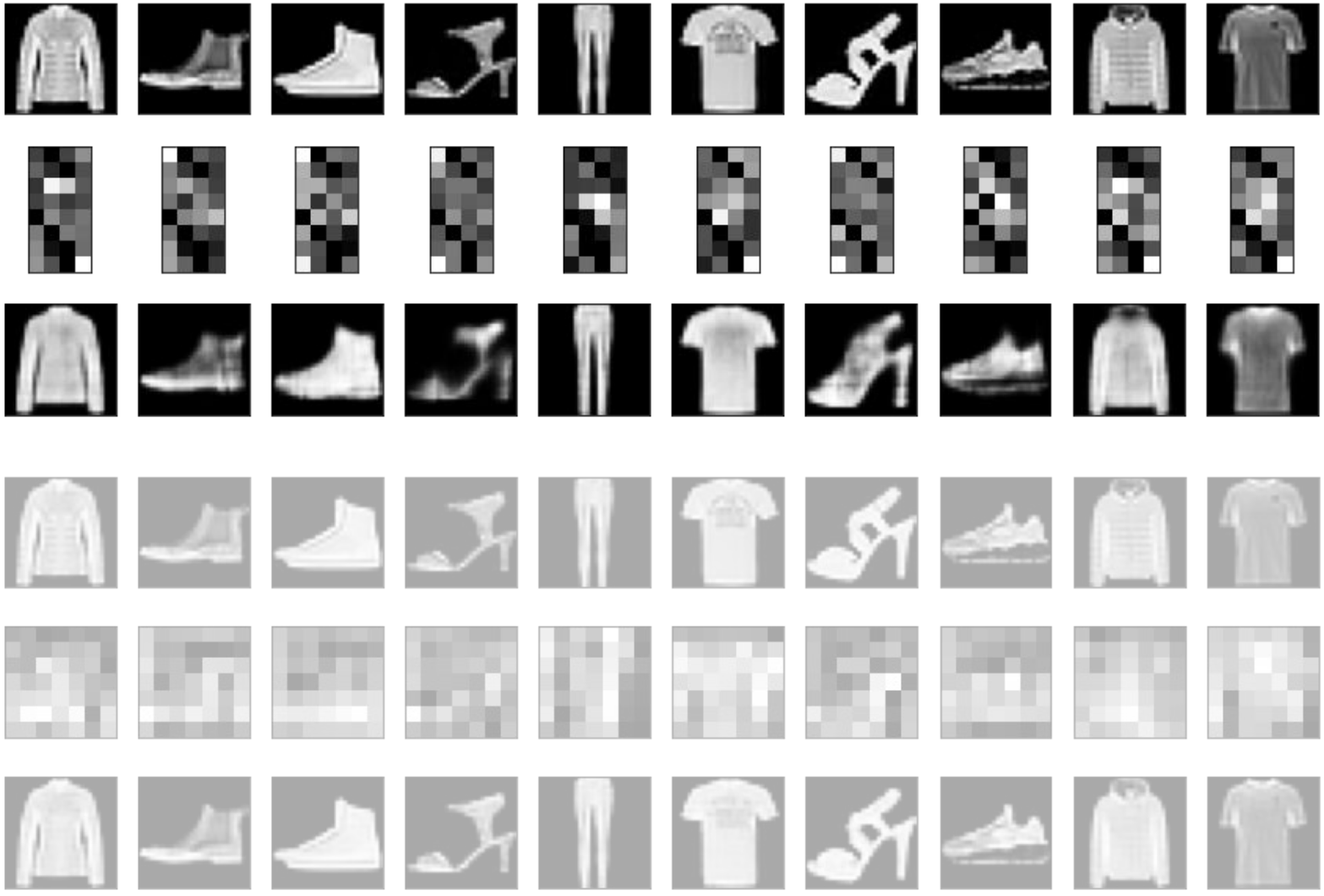
```
def deep_autoencoder():
 encoder = tf.keras.layers.Dense(units=128, activation='relu')(inputs)
 encoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 encoder = tf.keras.layers.Dense(units=32, activation='relu')(encoder)

 decoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 decoder = tf.keras.layers.Dense(units=128, activation='relu')(decoder)
 decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(decoder)

 return encoder, decoder

deep_encoder_output, deep_autoencoder_output = deep_autoencoder()

deep_encoder_model = tf.keras.Model(inputs=inputs, outputs=deep_encoder_output)
deep_autoencoder_model = tf.keras.Model(inputs=inputs, outputs=deep_autoencoder_output)
```

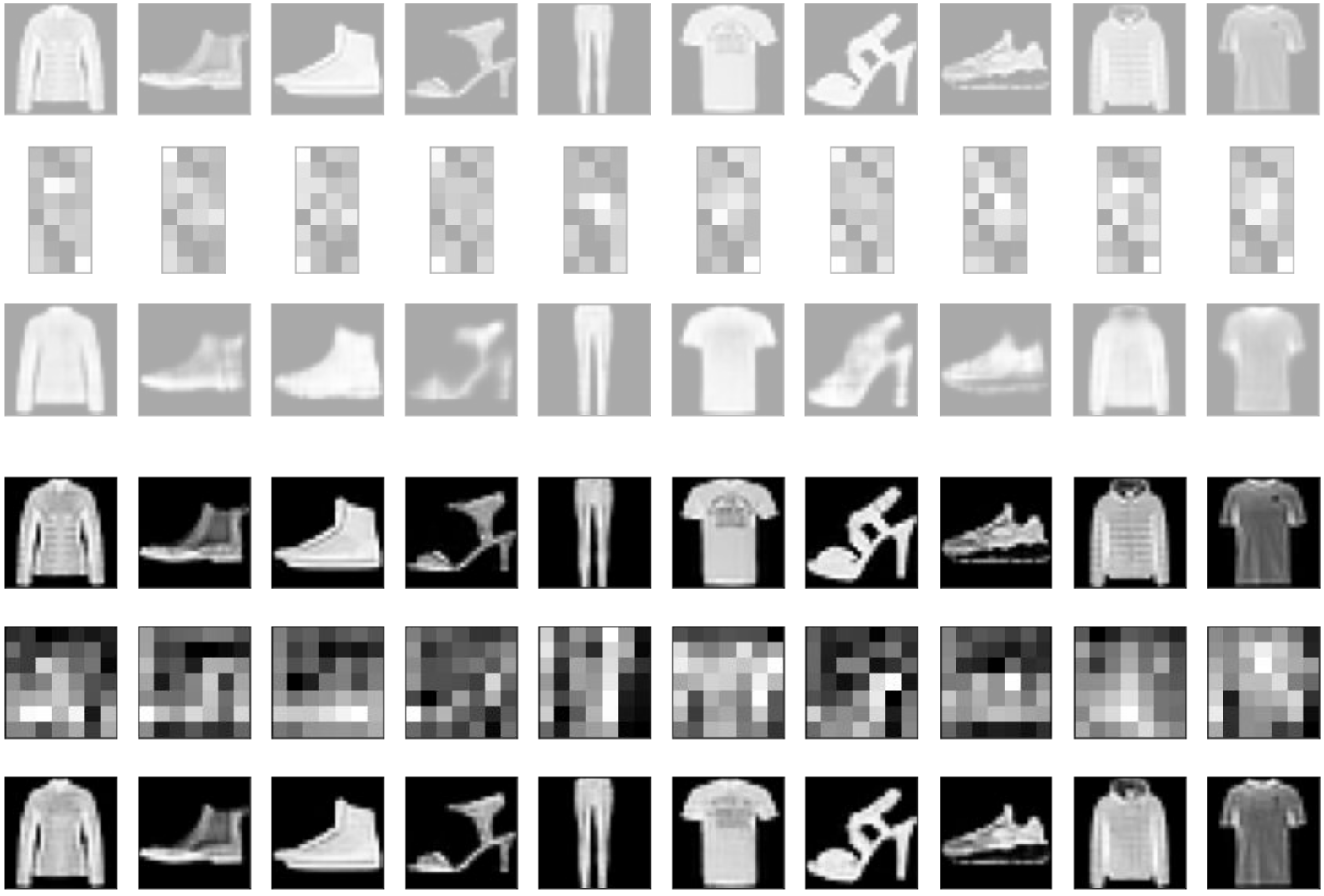
```
def deep_autoencoder():
 encoder = tf.keras.layers.Dense(units=128, activation='relu')(inputs)
 encoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 encoder = tf.keras.layers.Dense(units=32, activation='relu')(encoder)

 decoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 decoder = tf.keras.layers.Dense(units=128, activation='relu')(decoder)
 decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(decoder)

 return encoder, decoder

deep_encoder_output, deep_autoencoder_output = deep_autoencoder()

deep_encoder_model = tf.keras.Model(inputs=inputs, outputs=deep_encoder_output)
deep_autoencoder_model = tf.keras.Model(inputs=inputs, outputs=deep_autoencoder_output)
```

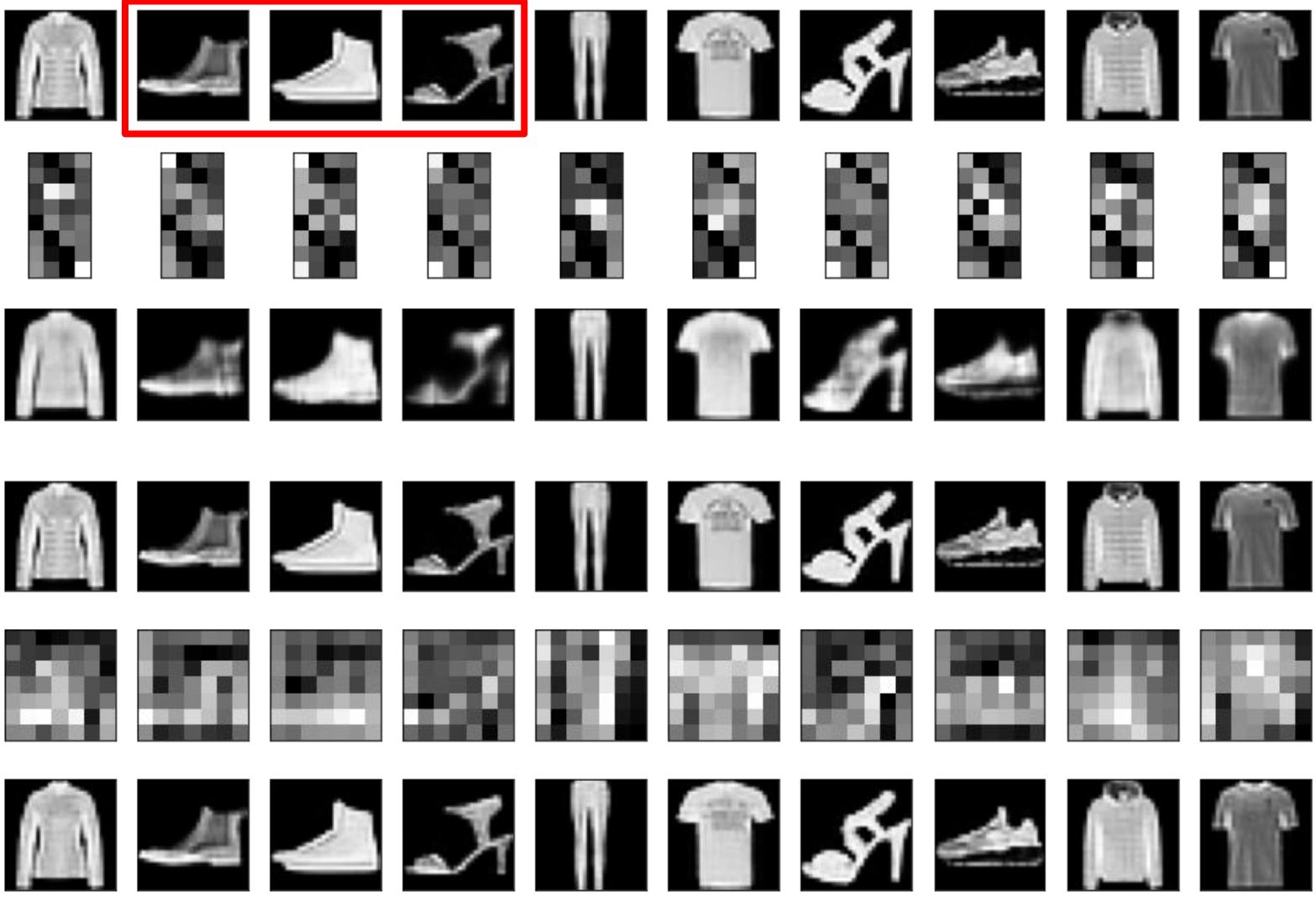
```
def deep_autoencoder():
 encoder = tf.keras.layers.Dense(units=128, activation='relu')(inputs)
 encoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 encoder = tf.keras.layers.Dense(units=32, activation='relu')(encoder)

 decoder = tf.keras.layers.Dense(units=64, activation='relu')(encoder)
 decoder = tf.keras.layers.Dense(units=128, activation='relu')(decoder)
 decoder = tf.keras.layers.Dense(units=784, activation='sigmoid')(decoder)

 return encoder, decoder

deep_encoder_output, deep_autoencoder_output = deep_autoencoder()

deep_encoder_model = tf.keras.Model(inputs=inputs, outputs=deep_encoder_output)
deep_autoencoder_model = tf.keras.Model(inputs=inputs, outputs=deep_autoencoder_output)
```

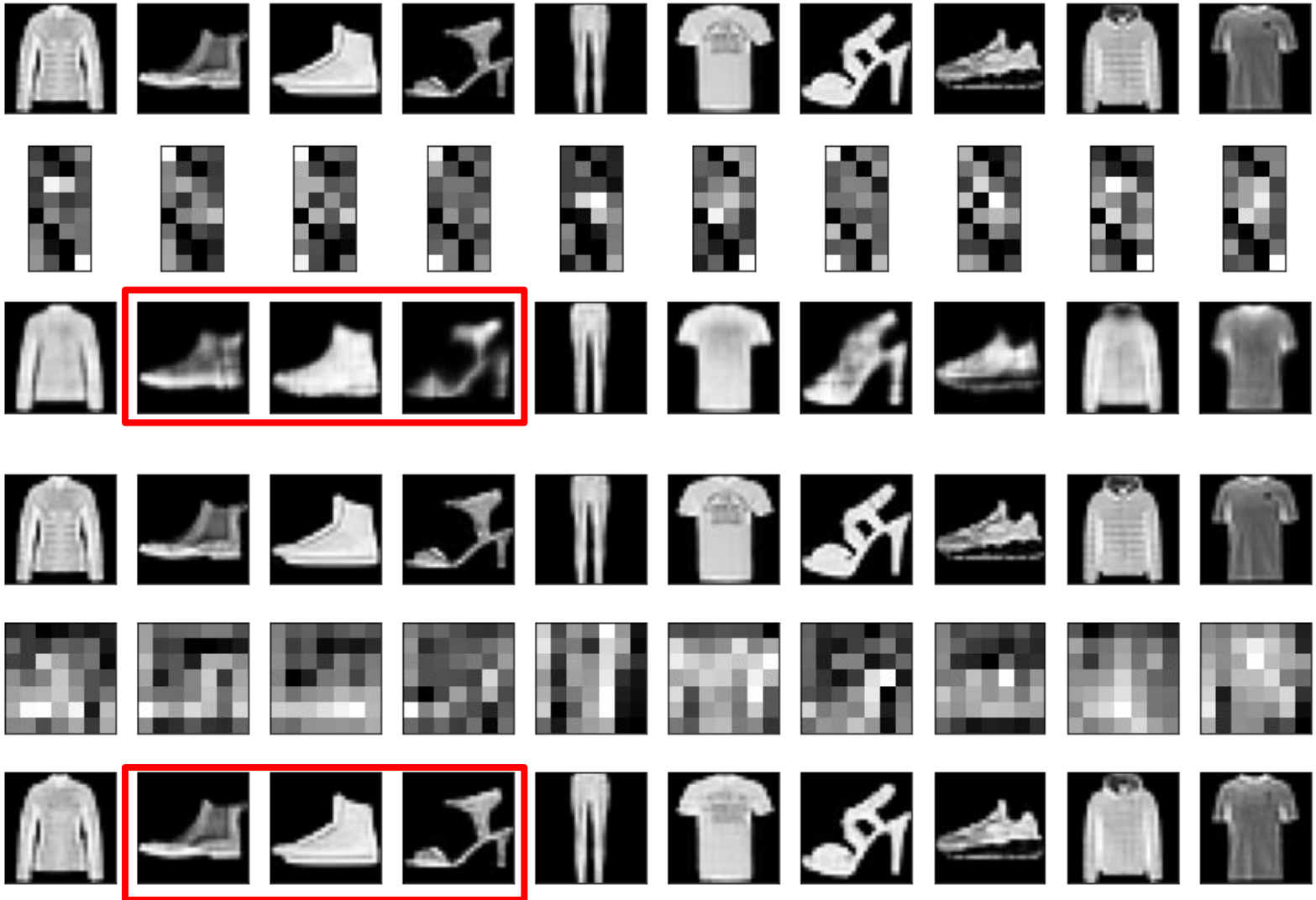


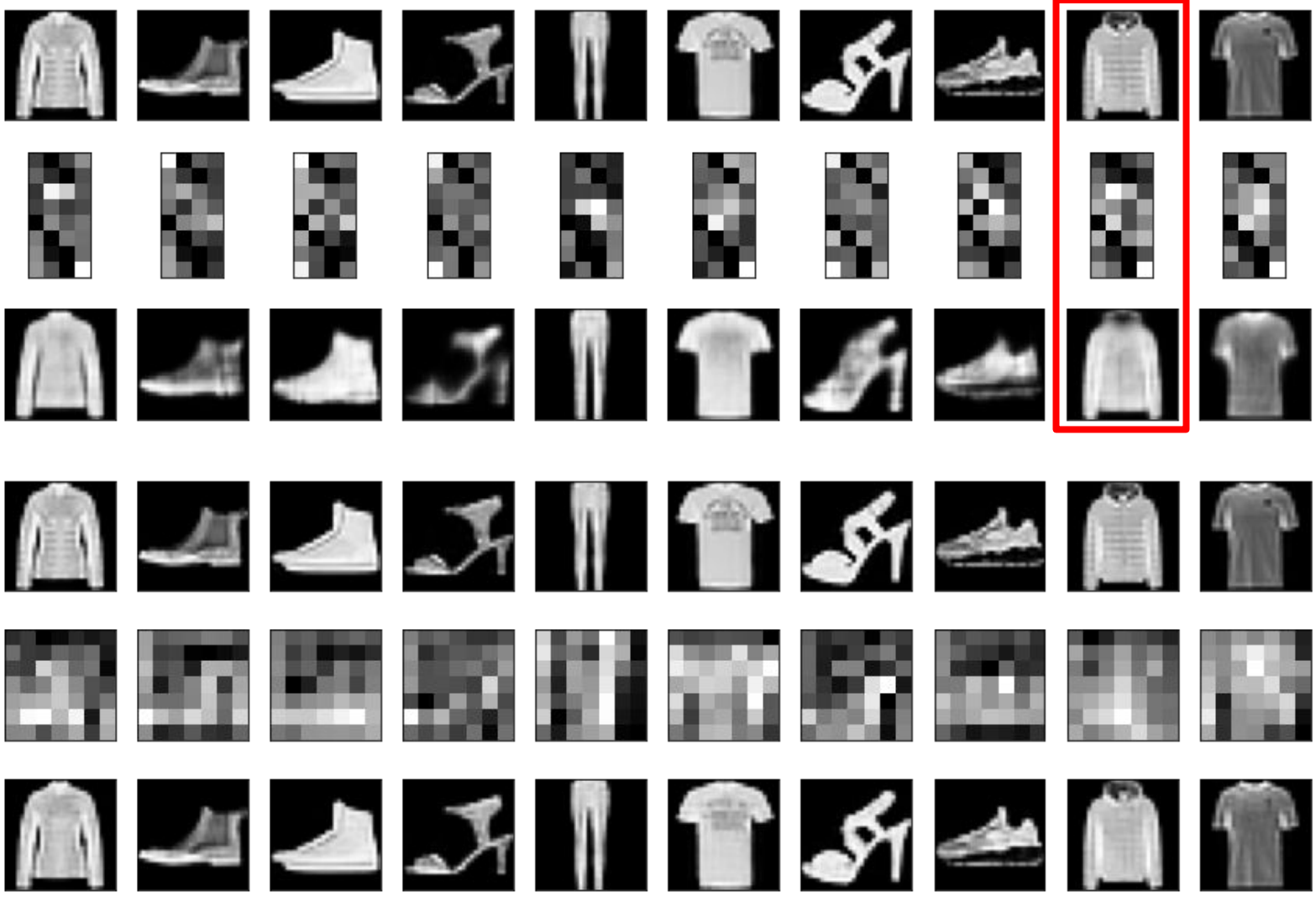


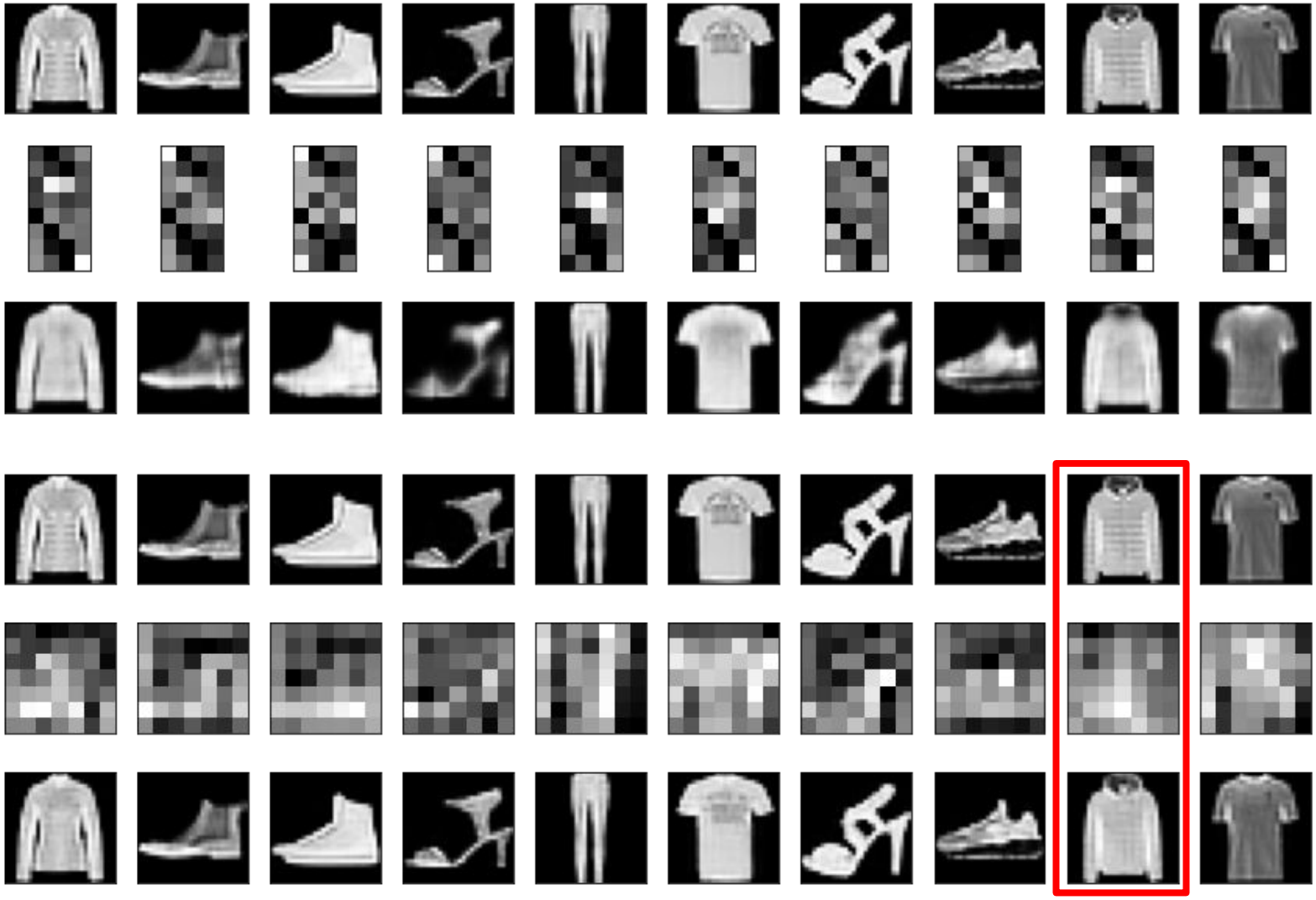




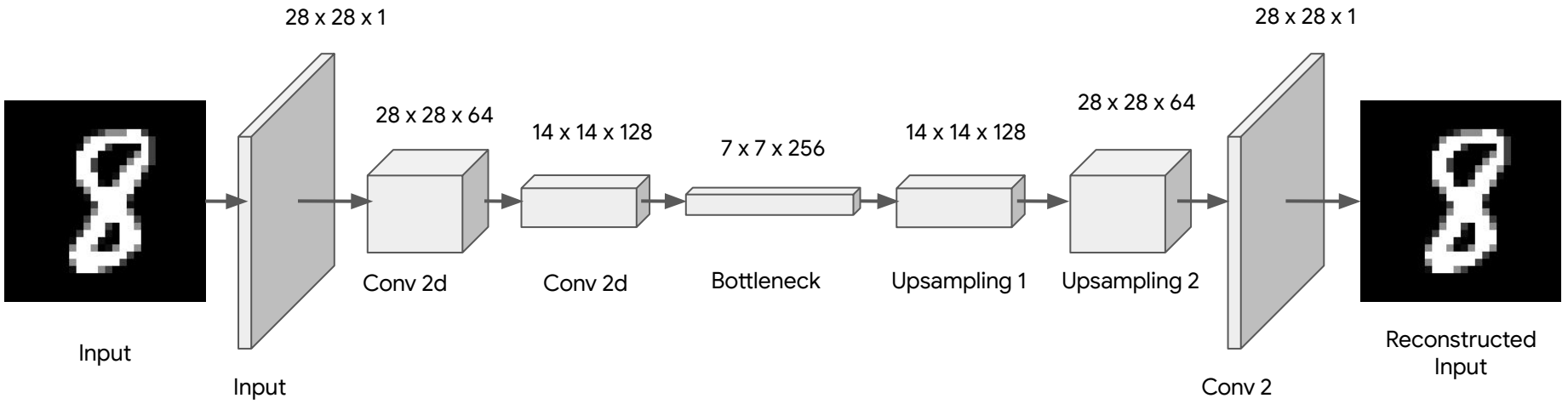




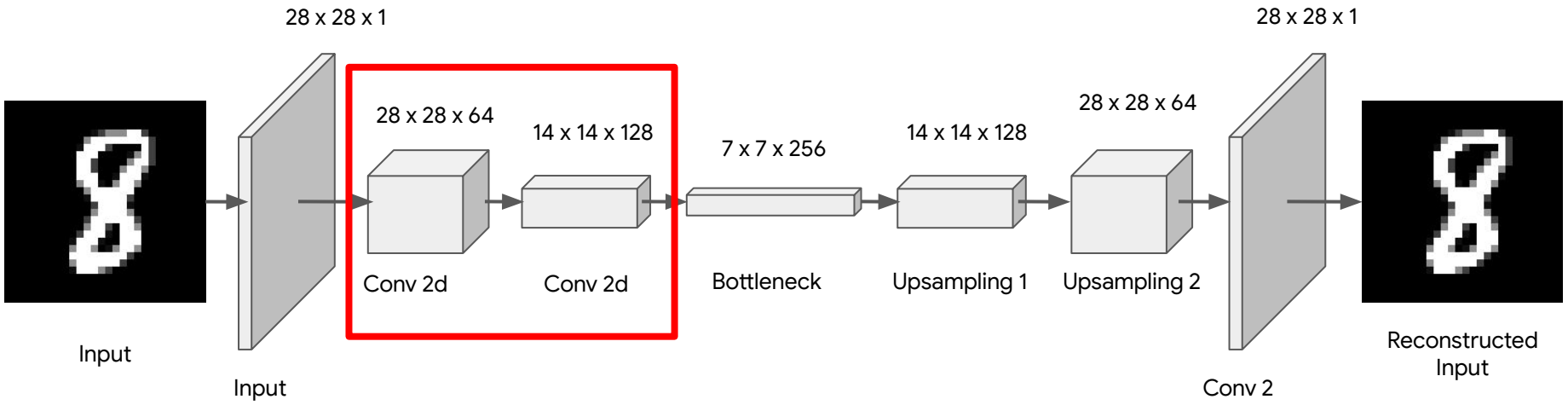




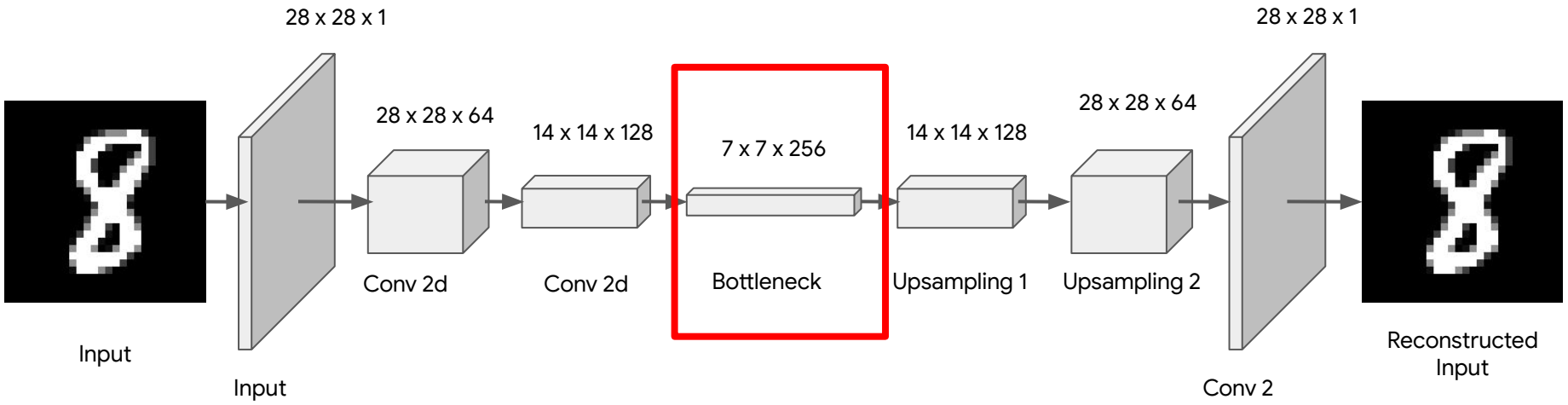
Convolutional Auto-Encoders



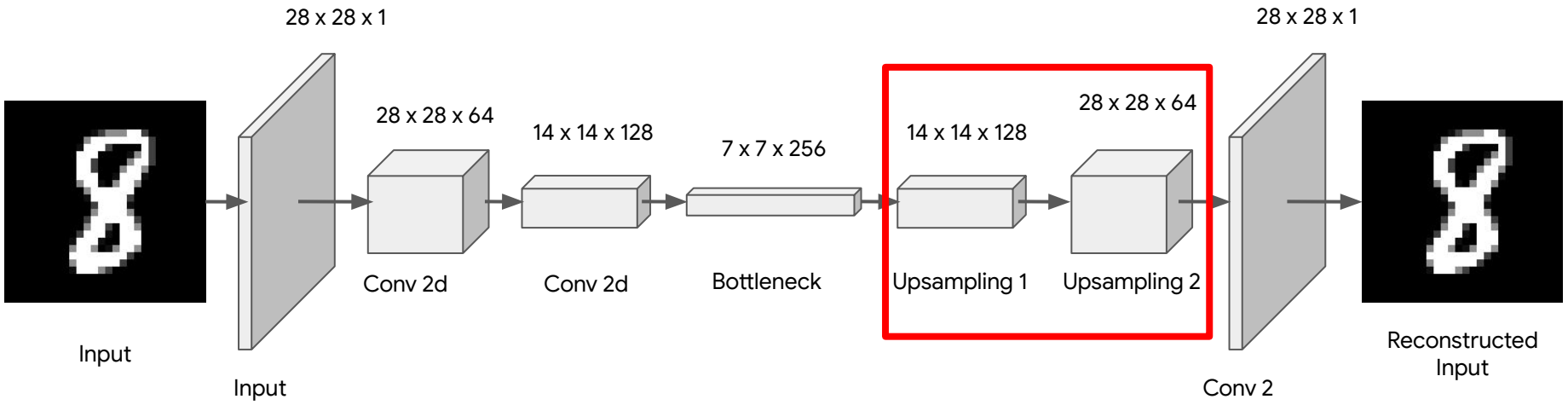
Convolutional Auto-Encoders



Convolutional Auto-Encoders



Convolutional Auto-Encoders



```
def encoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)

  max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(max_pool_1)

  max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)

  return max_pool_2
```

```
def encoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)

  max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(max_pool_1)

  max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)

  return max_pool_2
```

```
def encoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)

  max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(max_pool_1)

  max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)

  return max_pool_2
```

```
def encoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)

  max_pool_1 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(max_pool_1)

  max_pool_2 = tf.keras.layers.MaxPooling2D(pool_size=(2,2))(conv_2)

  return max_pool_2
```

```
def bottle_neck(inputs):
  bottle_neck = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3),
                                       activation='relu', padding='same')(inputs)



  encoder_visualization = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                                 activation='sigmoid',
                                                 padding='same')(bottle_neck)

  return bottle_neck, encoder_visualization
```

```
def bottle_neck(inputs):
  bottle_neck = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3),
                                       activation='relu', padding='same')(inputs)


  encoder_visualization = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                                 activation='sigmoid',
                                                 padding='same')(bottle_neck)

  return bottle_neck, encoder_visualization
```

```
def bottle_neck(inputs):
  bottle_neck = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3),
                                       activation='relu', padding='same')(inputs)


  encoder_visualization = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                                 activation='sigmoid',
                                                 padding='same')(bottle_neck)

  return bottle_neck, encoder_visualization
```

```
def bottle_neck(inputs):
  bottle_neck = tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3),
                                       activation='relu', padding='same')(inputs)



  encoder_visualization = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                                 activation='sigmoid',
                                                 padding='same')(bottle_neck)

  return bottle_neck, encoder_visualization
```

```
def decoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)
  up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(up_sample_1)
  up_sample_2 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_2)

  conv_3 = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                  activation='sigmoid',
                                  padding='same')(up_sample_2)

  return conv_3
```

```
def decoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)
  up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(up_sample_1)
  up_sample_2 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_2)

  conv_3 = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                  activation='sigmoid',
                                  padding='same')(up_sample_2)

  return conv_3
```

```
def decoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)
  up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(up_sample_1)
  up_sample_2 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_2)

  conv_3 = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                  activation='sigmoid',
                                  padding='same')(up_sample_2)

  return conv_3
```

```
def decoder(inputs):
  conv_1 = tf.keras.layers.Conv2D(filters=128, kernel_size=(3,3),
                                  activation='relu', padding='same')(inputs)
  up_sample_1 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_1)

  conv_2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3,3),
                                  activation='relu', padding='same')(up_sample_1)
  up_sample_2 = tf.keras.layers.UpSampling2D(size=(2,2))(conv_2)

  conv_3 = tf.keras.layers.Conv2D(filters=1, kernel_size=(3,3),
                                  activation='sigmoid',
                                  padding='same')(up_sample_2)

  return conv_3
```

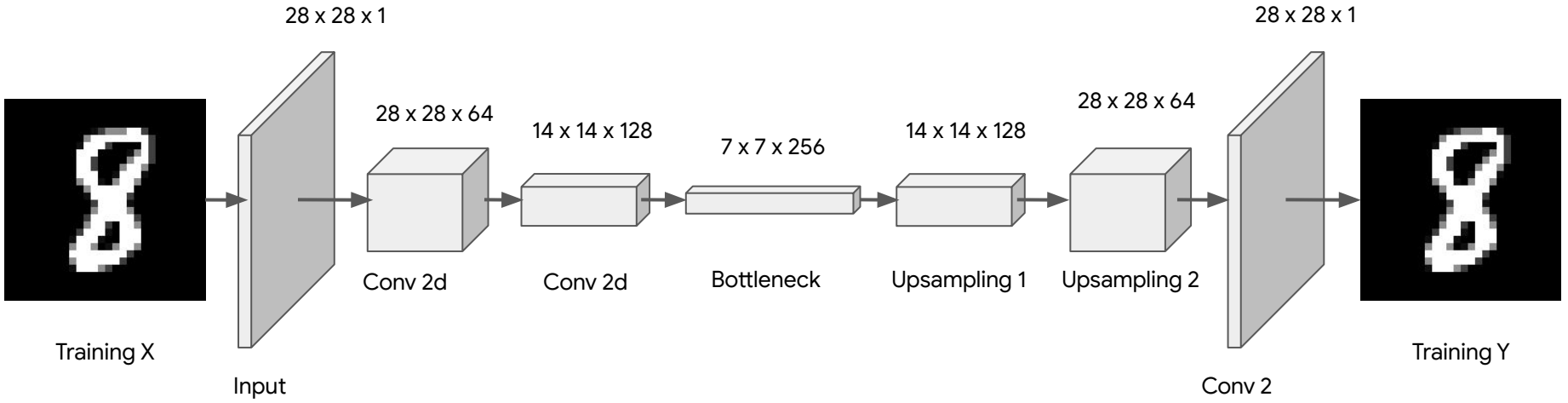
```
def convolutional_auto_encoder():
  inputs = tf.keras.layers.Input(shape=(28, 28, 1,))

  encoder_output = encoder(inputs)

  bottleneck_output, encoder_visualization = bottle_neck(encoder_output)

  decoder_output = decoder(bottleneck_output)

  model = tf.keras.Model(inputs =inputs, outputs=decoder_output)
  encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_visualization)

  return model, encoder_model
```

```
def convolutional_auto_encoder():
  inputs = tf.keras.layers.Input(shape=(28, 28, 1,))

  encoder_output = encoder(inputs)

  bottleneck_output, encoder_visualization = bottle_neck(encoder_output)

  decoder_output = decoder(bottleneck_output)

  model = tf.keras.Model(inputs =inputs, outputs=decoder_output)
  encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_visualization)

  return model, encoder_model
```

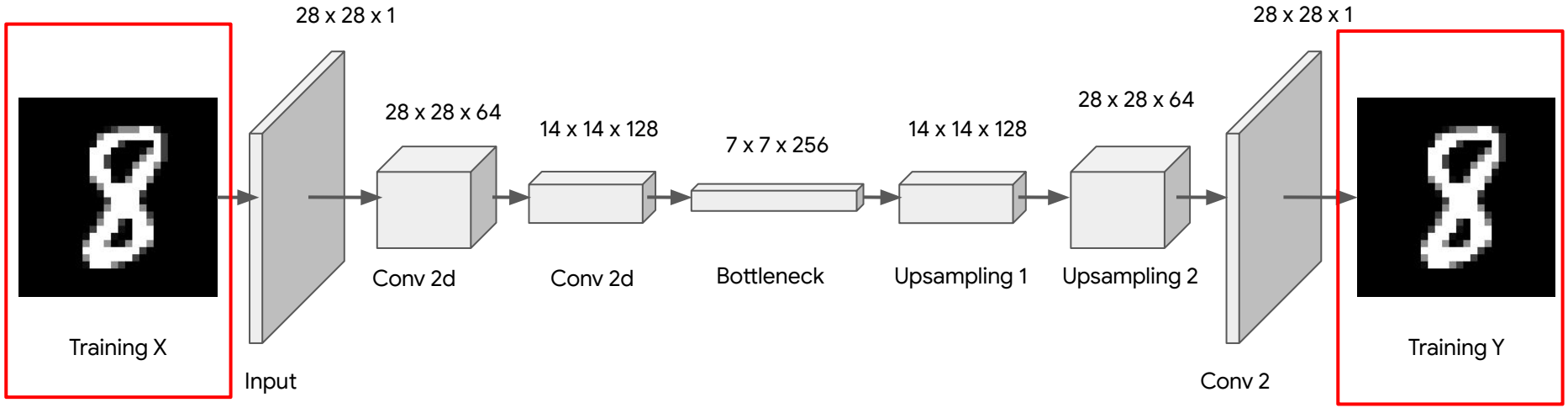
```
def convolutional_auto_encoder():
  inputs = tf.keras.layers.Input(shape=(28, 28, 1,))

  encoder_output = encoder(inputs)

  bottleneck_output, encoder_visualization = bottle_neck(encoder_output)

  decoder_output = decoder(bottleneck_output)

  model = tf.keras.Model(inputs =inputs, outputs=decoder_output)
  encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_visualization)

  return model, encoder_model
```

```
def convolutional_auto_encoder():
  inputs = tf.keras.layers.Input(shape=(28, 28, 1,))

  encoder_output = encoder(inputs)

  bottleneck_output, encoder_visualization = bottle_neck(encoder_output)

  decoder_output = decoder(bottleneck_output)

  model = tf.keras.Model(inputs =inputs, outputs=decoder_output)
  encoder_model = tf.keras.Model(inputs=inputs, outputs=encoder_visualization)

  return model, encoder_model
```

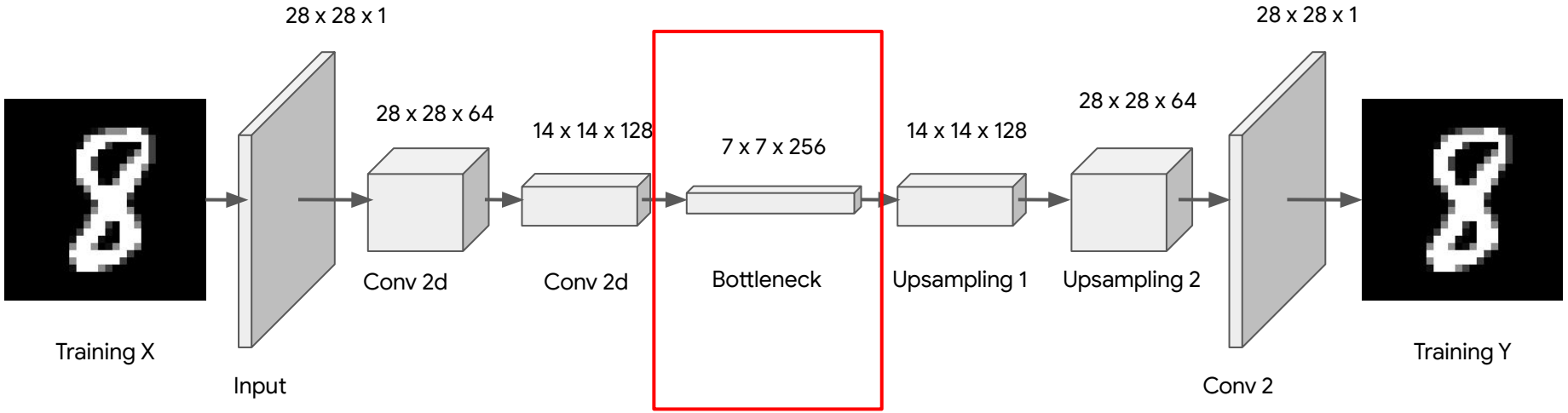
Convolutional Auto-Encoders



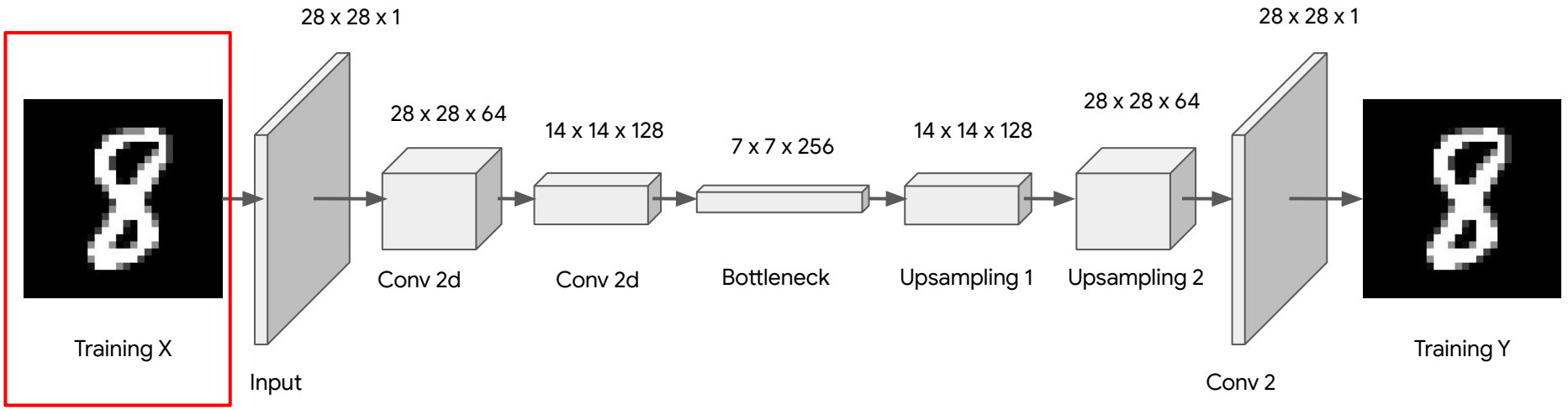
Convolutional Auto-Encoders



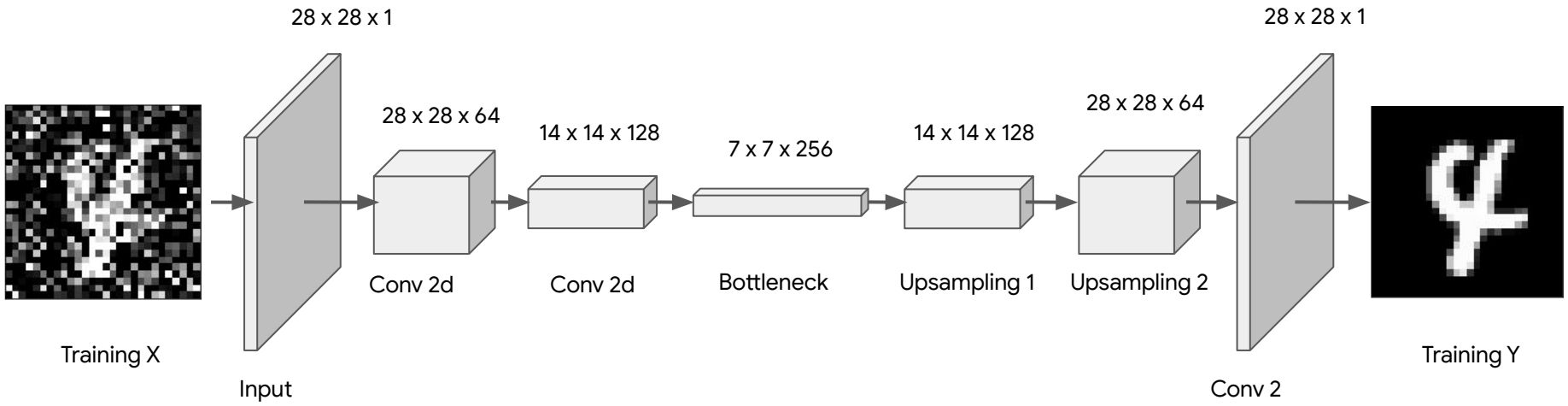
Convolutional Auto-Encoders

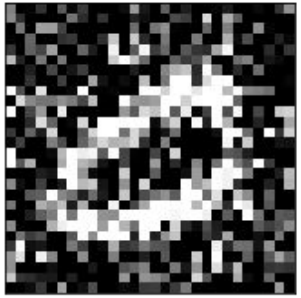
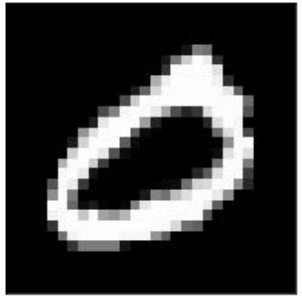
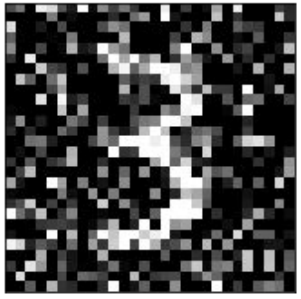


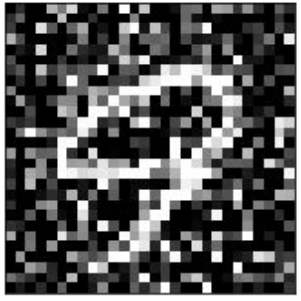
Convolutional Auto-Encoders

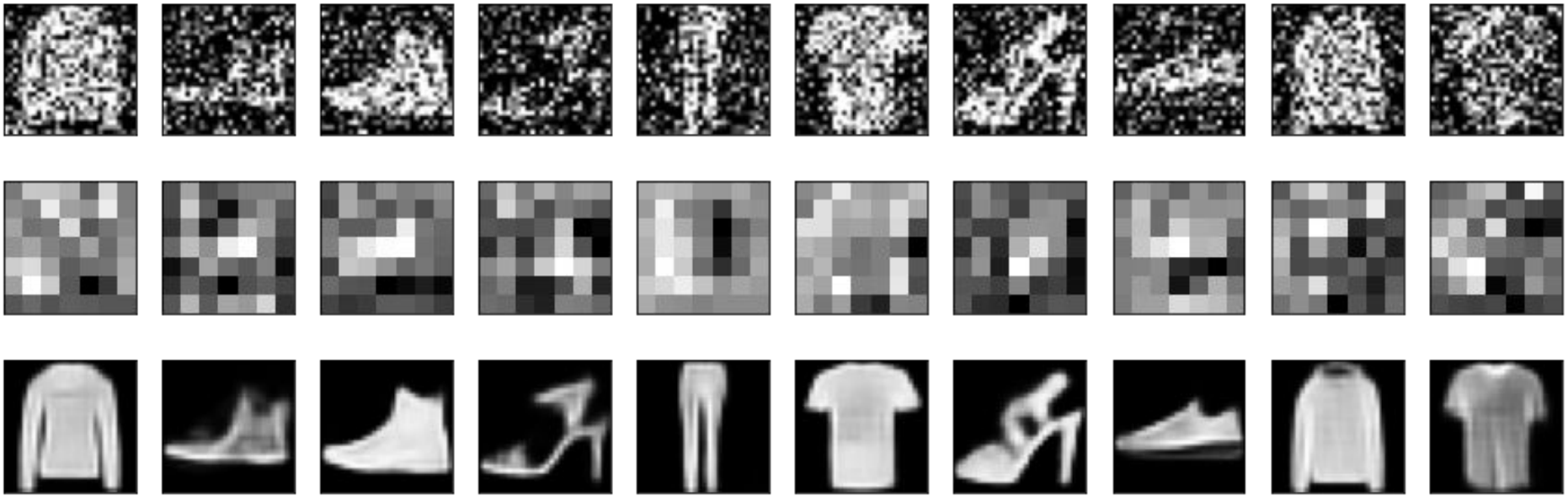


Convolutional Auto-Encoders





```
def map_image_with_noise(image, label):
  noise_factor = 0.5
  image = tf.cast(image, dtype=tf.float32)
  image = image / 255.0

  factor = noise_factor * tf.random.normal(shape=image.shape)
  image_noisy = image + factor
  image_noisy = tf.clip_by_value(image_noisy, 0.0, 1.0)

  return image_noisy, image
```

```
def map_image_with_noise(image, label):
  noise_factor = 0.5
  image = tf.cast(image, dtype=tf.float32)
  image = image / 255.0

  factor = noise_factor * tf.random.normal(shape=image.shape)
  image_noisy = image + factor
  image_noisy = tf.clip_by_value(image_noisy, 0.0, 1.0)

  return image_noisy, image
```

```
def map_image_with_noise(image, label):
  noise_factor = 0.5
  image = tf.cast(image, dtype=tf.float32)
  image = image / 255.0

  factor = noise_factor * tf.random.normal(shape=image.shape)
  image_noisy = image + factor
  image_noisy = tf.clip_by_value(image_noisy, 0.0, 1.0)

  return image_noisy, image
```

```
def map_image_with_noise(image, label):
  noise_factor = 0.5
  image = tf.cast(image, dtype=tf.float32)
  image = image / 255.0

  factor = noise_factor * tf.random.normal(shape=image.shape)
  image_noisy = image + factor
  image_noisy = tf.clip_by_value(image_noisy, 0.0, 1.0)

  return image_noisy, image
```