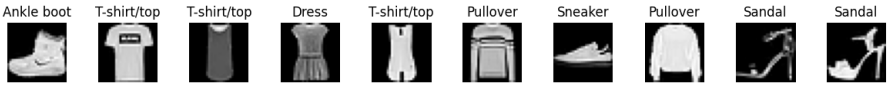
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model


# NO Target requirement for autoencoders (unsupervised learning)
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255. # normalize [0-1]
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

```
class_names=['T-shirt/top','Trouser','Pullover','Dress','Coat','Sandal','Shirt','Sneaker','Bag','Ankle boot']
plt.figure(figsize=(15,1))
for i in range(10):
  plt.subplot(1,10,i+1)
  plt.imshow(x_train[i],cmap='gray')
  plt.title(class_names[y_train[i]])
  plt.axis('off')
```



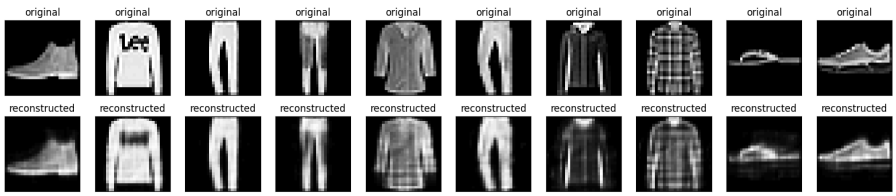

```
class Autoencoder(Model):
  def __init__(self, latent_dim, shape):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.shape = shape
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(tf.math.reduce_prod(shape).numpy(), activation='sigmoid'),
      layers.Reshape(shape)
    ])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded


shape = x_test.shape[1:]
latent_dim = 64
autoencoder = Autoencoder(latent_dim, shape)


autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())


autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0237 - val_loss: 0.0131
Epoch 2/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0116 - val_loss: 0.0107
Epoch 3/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0101 - val_loss: 0.0097
```

```
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0095 - val_loss: 0.0093
Epoch 5/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0092 - val_loss: 0.0091
Epoch 6/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0090 - val_loss: 0.0091
Epoch 7/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0089 - val_loss: 0.0090
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0088 - val_loss: 0.0089
Epoch 9/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0088 - val_loss: 0.0088
Epoch 10/10
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0087 - val_loss: 0.0089
<keras.src.callbacks.History at 0x7aea0241ad40>
```

```
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  plt.imshow(x_test[i])
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)
  plt.imshow(decoded_imgs[i])
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

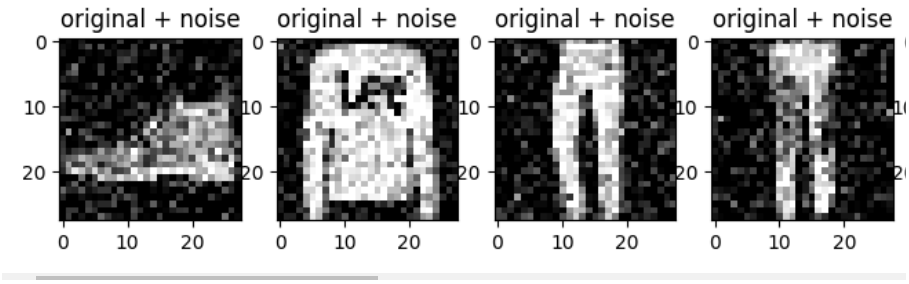



```
############ 2nd example DENOISING
(x_train, _), (x_test, _) = fashion_mnist.load_data()
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]
print(x_train.shape)
```

```
(60000, 28, 28, 1)
```

```
noise_factor = 0.2
x_train_noisy = x_train + noise_factor * tf.random.normal(shape=x_train.shape)
x_test_noisy = x_test + noise_factor * tf.random.normal(shape=x_test.shape)

x_train_noisy = tf.clip_by_value(x_train_noisy, clip_value_min=0., clip_value_max=1.)
x_test_noisy = tf.clip_by_value(x_test_noisy, clip_value_min=0., clip_value_max=1.)
```

```
n = 10
plt.figure(figsize=(20, 2))
for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.title("original + noise")
    plt.imshow(tf.squeeze(x_test_noisy[i]))
    plt.gray()
plt.show()
```

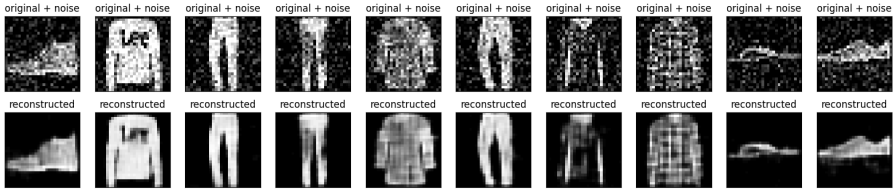



```
class Denoise(Model):
  def __init__(self):
    super(Denoise, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Input(shape=(28, 28, 1)),
      layers.Conv2D(16, (3, 3), activation='relu', padding='same', strides=2),
      layers.Conv2D(8, (3, 3), activation='relu', padding='same', strides=2)])

    self.decoder = tf.keras.Sequential([
      layers.Conv2DTranspose(8, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padding='same'),
      layers.Conv2D(1, kernel_size=(3, 3), activation='sigmoid', padding='same')])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = Denoise()


autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())


autoencoder.fit(x_train_noisy, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.0164 - val_loss: 0.0094
Epoch 2/10
1875/1875 [==============================] - 66s 35ms/step - loss: 0.0087 - val_loss: 0.0081
Epoch 3/10
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0078 - val_loss: 0.0076
Epoch 4/10
1875/1875 [==============================] - 59s 31ms/step - loss: 0.0074 - val_loss: 0.0073
Epoch 5/10
1875/1875 [==============================] - 57s 31ms/step - loss: 0.0072 - val_loss: 0.0072
Epoch 6/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.0070 - val_loss: 0.0071
Epoch 7/10
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0070 - val_loss: 0.0069
Epoch 8/10
1875/1875 [==============================] - 60s 32ms/step - loss: 0.0069 - val_loss: 0.0069
Epoch 9/10
1875/1875 [==============================] - 58s 31ms/step - loss: 0.0068 - val_loss: 0.0068
Epoch 10/10
1875/1875 [==============================] - 61s 32ms/step - loss: 0.0068 - val_loss: 0.0068
<keras.src.callbacks.History at 0x7aea027e5930>
```

```
encoded_imgs = autoencoder.encoder(x_test_noisy).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

```
n = 10
plt.figure(figsize=(20, 4))
for i in range(n):

    # display original + noise
    ax = plt.subplot(2, n, i + 1)
    plt.title("original + noise")
    plt.imshow(tf.squeeze(x_test_noisy[i]))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    bx = plt.subplot(2, n, i + n + 1)
    plt.title("reconstructed")
    plt.imshow(tf.squeeze(decoded_imgs[i]))
    plt.gray()
    bx.get_xaxis().set_visible(False)
    bx.get_yaxis().set_visible(False)
plt.show()
```




```
# Example 3 Anomaly detection
# Download the dataset
dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv', header=None)
raw_data = dataframe.values
dataframe.head()
```

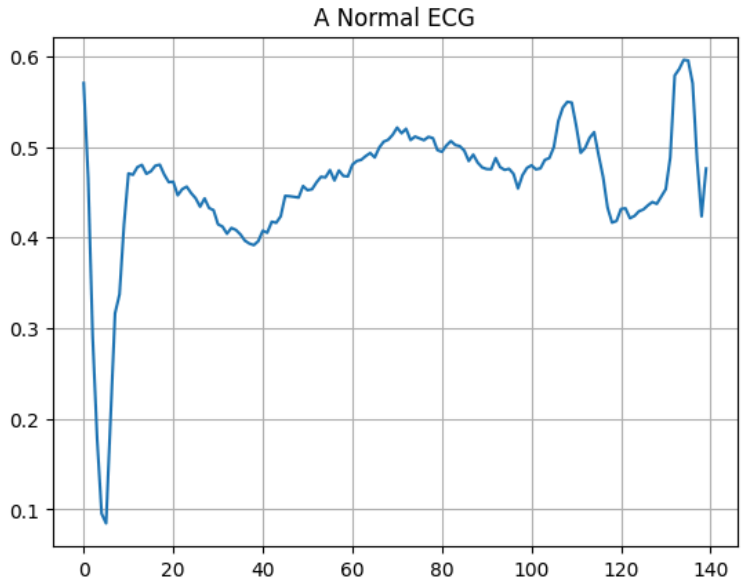
	6	7	8	9	...	131	132	133	134	135
	408	-1.818286	-1.250522	-0.477492	...	0.792168	0.933541	0.796958	0.578621	0.25774(
	126	-0.992258	-0.754680	0.042321	...	0.538356	0.656881	0.787490	0.724046	0.55578
	940	-1.490659	-1.183580	-0.394229	...	0.886073	0.531452	0.311377	-0.021919	-0.71368
	280	-1.671131	-1.333884	-0.965629	...	0.350816	0.499111	0.600345	0.842069	0.95207
	510	-1.783423	-1.594450	-0.753199	...	1.148884	0.958434	1.059025	1.371682	1.27739

```
dataframe.shape
```

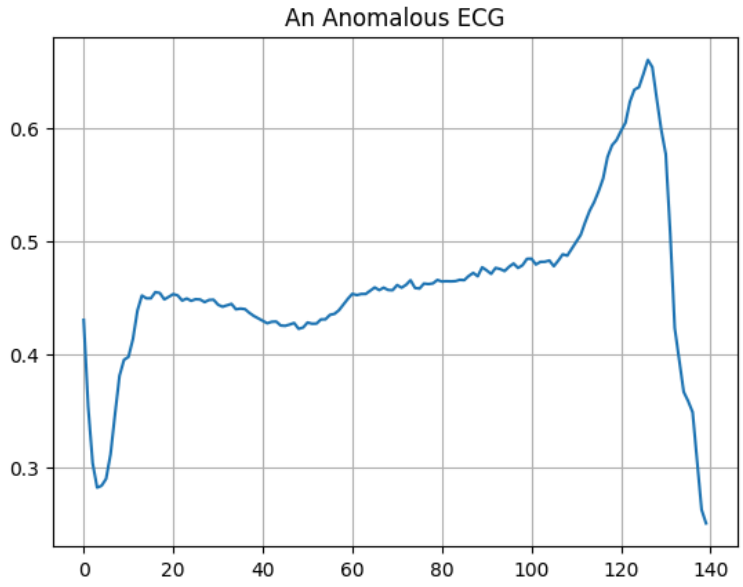
```
(4998, 141)
```

```
# The last element contains the labels
labels = raw_data[:, -1]

# The other data points are the electrocadriogram data
data = raw_data[:, 0:-1]

train_data, test_data, train_labels, test_labels = train_test_split(
    data, labels, test_size=0.2, random_state=21
)


min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)

train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)

train_data = tf.cast(train_data, tf.float32)
test_data = tf.cast(test_data, tf.float32)
```

```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

normal_train_data = train_data[train_labels]
normal_test_data = test_data[test_labels]

anomalous_train_data = train_data[~train_labels]
anomalous_test_data = test_data[~test_labels]


plt.grid()
plt.plot(np.arange(140), normal_train_data[0])
plt.title("A Normal ECG")
plt.show()
```




```
plt.grid()
plt.plot(np.arange(140), anomalous_train_data[0])
plt.title("An Anomalous ECG")
plt.show()
```

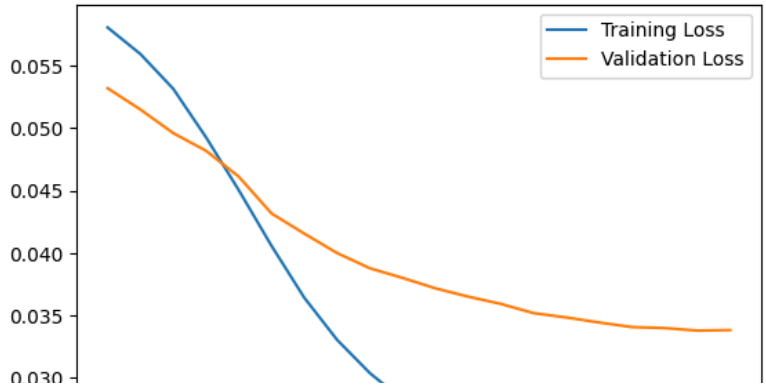


```
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Dense(32, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="relu")])

    self.decoder = tf.keras.Sequential([
      layers.Dense(16, activation="relu"),
      layers.Dense(32, activation="relu"),
      layers.Dense(140, activation="sigmoid")])

  def call(self, x):
    encoded = self.encoder(x)
    decoded = self.decoder(encoded)
    return decoded

autoencoder = AnomalyDetector()


autoencoder.compile(optimizer='adam', loss='mae')


history = autoencoder.fit(normal_train_data, normal_train_data,
          epochs=20,
          batch_size=512,
          validation_data=(test_data, test_data),
          shuffle=True)


```

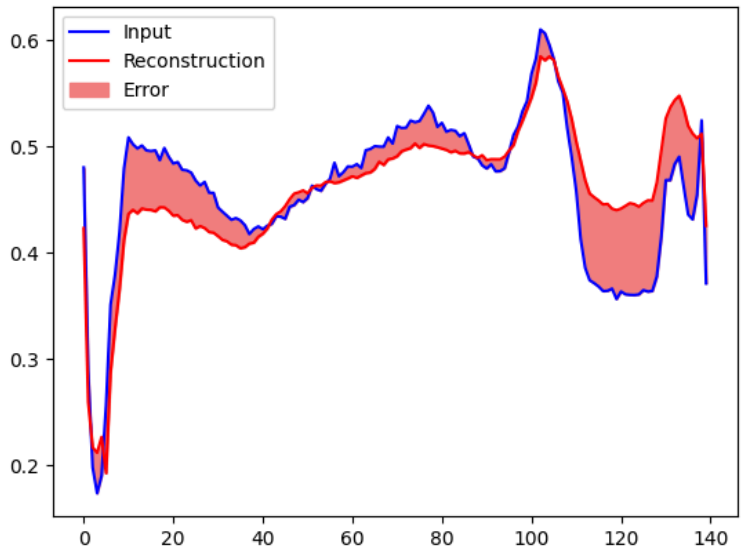
```
Epoch 1/20
5/5 [==============================] - 2s 55ms/step - loss: 0.0581 - val_loss: 0.0532
Epoch 2/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0559 - val_loss: 0.0515
Epoch 3/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0532 - val_loss: 0.0496
Epoch 4/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0493 - val_loss: 0.0482
Epoch 5/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0450 - val_loss: 0.0461
Epoch 6/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0406 - val_loss: 0.0432
Epoch 7/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0364 - val_loss: 0.0415
Epoch 8/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0330 - val_loss: 0.0400
Epoch 9/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0304 - val_loss: 0.0388
Epoch 10/20
5/5 [==============================] - 0s 20ms/step - loss: 0.0282 - val_loss: 0.0380
Epoch 11/20
5/5 [==============================] - 0s 18ms/step - loss: 0.0266 - val_loss: 0.0372
Epoch 12/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0255 - val_loss: 0.0365
Epoch 13/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0246 - val_loss: 0.0359
Epoch 14/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0238 - val_loss: 0.0352
Epoch 15/20
5/5 [==============================] - 0s 18ms/step - loss: 0.0231 - val_loss: 0.0348
Epoch 16/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0224 - val_loss: 0.0344
Epoch 17/20
5/5 [==============================] - 0s 17ms/step - loss: 0.0219 - val_loss: 0.0341
Epoch 18/20
5/5 [==============================] - 0s 13ms/step - loss: 0.0214 - val_loss: 0.0340
Epoch 19/20
5/5 [==============================] - 0s 16ms/step - loss: 0.0211 - val_loss: 0.0338
Epoch 20/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0209 - val_loss: 0.0338
```

```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ae9f83b7cd0>
```




```
encoded_data = autoencoder.encoder(normal_test_data).numpy()
decoded_data = autoencoder.decoder(encoded_data).numpy()

plt.plot(normal_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(140), decoded_data[0], normal_test_data[0], color='lightcoral')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```




```
encoded_data = autoencoder.encoder(anomalous_test_data).numpy()
decoded_data = autoencoder.decoder(encoded_data).numpy()

plt.plot(anomalous_test_data[0], 'b')
plt.plot(decoded_data[0], 'r')
plt.fill_between(np.arange(140), decoded_data[0], anomalous_test_data[0], color='lightcoral')
plt.legend(labels=["Input", "Reconstruction", "Error"])
plt.show()
```