An autoencoder is a special type of neural network that is trained to copy its input to its output. For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image. An autoencoder learns to compress the data while minimizing the reconstruction error.

To learn more about autoencoders, please consider reading chapter 14 from Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.

```
#Import TensorFlow and other libraries
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
```

Load the dataset

To start, you will train the basic autoencoder using the Fashion MNIST dataset. Each image in this dataset is 28x28 pixels.

```
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [================================] - 0s 0us/step
40960/29515 [========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [===============================================================================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
(60000, 28, 28)
(10000, 28, 28)
```

First example: Basic autoencoder Define an autoencoder with two Dense layers: an encoder, which compresses the images into a 64 dimensional latent vector, and a decoder, that reconstructs the original image from the latent space.

```
latent_dim = 64

class Autoencoder(Model):
  def __init__(self, latent_dim):
    super(Autoencoder, self).__init__()
    self.latent_dim = latent_dim
    self.encoder = tf.keras.Sequential([
      layers.Flatten(),
      layers.Dense(latent_dim, activation='relu'),
    ])
    self.decoder = tf.keras.Sequential([
      layers.Dense(784, activation='sigmoid'),
      layers.Reshape((28, 28))
    ])

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

autoencoder = Autoencoder(latent_dim)
```

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

Train the model using x_train as both the input and the target. The encoder will learn to compress the dataset from 784 dimensions to the latent space, and the decoder will learn to reconstruct the original images. .

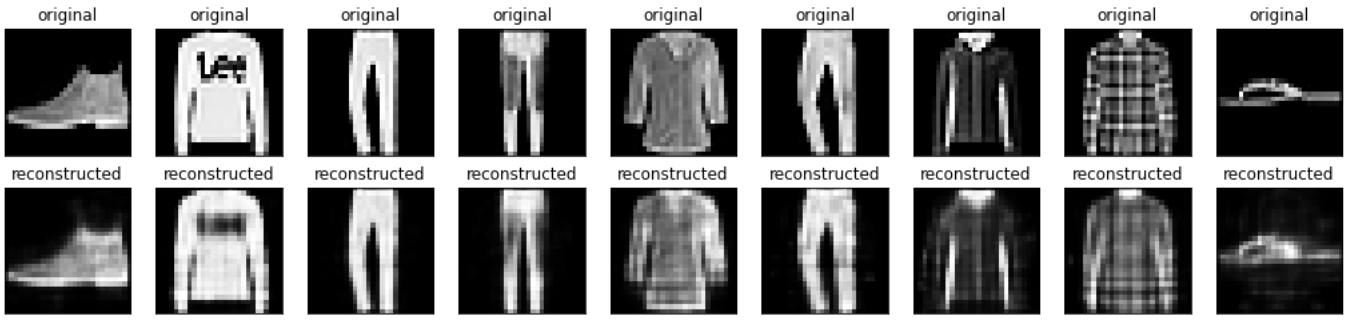
```
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0234 - val_loss: 0.0130
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0115 - val_loss: 0.0105
Epoch 3/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0100 - val_loss: 0.0097
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0094 - val_loss: 0.0093
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0091 - val_loss: 0.0091
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0090 - val_loss: 0.0090
Epoch 7/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0089 - val_loss: 0.0089
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0088 - val_loss: 0.0089
Epoch 9/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0088 - val_loss: 0.0088
Epoch 10/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0087 - val_loss: 0.0088
<keras.callbacks.History at 0x7f3abfd29490>
```

Now that the model is trained, let's test it by encoding and decoding images from the test set.

```
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()
```



Second example: Anomaly detection

Overview

In this example, you will train an autoencoder to detect anomalies on the ECG5000 dataset. This dataset contains 5,000 Electrocardiograms, each with 140 data points. You will use a simplified version of the dataset, where each example has been labeled either `0` (corresponding to an abnormal rhythm), or `1` (corresponding to a normal rhythm). You are interested in identifying the abnormal rhythms.

Note: This is a labeled dataset, so you could phrase this as a supervised learning problem. The goal of this example is to illustrate anomaly detection concepts you can apply to larger datasets, where you do not have labels available (for example, if you had many thousands of normal rhythms, and only a small number of abnormal rhythms).

How will you detect anomalies using an autoencoder? Recall that an autoencoder is trained to minimize reconstruction error. You will train an autoencoder on the normal rhythms only, then use it to reconstruct all the data. Our hypothesis is that the abnormal rhythms will have higher reconstruction error. You will then classify a rhythm as an anomaly if the reconstruction error surpasses a fixed threshold.

```
#Load ECG data
# Download the dataset
dataframe = pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv', header=None)
raw_data = dataframe.values
dataframe.tail()
```

	0	1	2	3	4	5	6	7	8	9	...	131	132
4993	0.608558	-0.335651	-0.990948	-1.784153	-2.626145	-2.957065	-2.931897	-2.664816	-2.090137	-1.461841	...	1.757705	2.291923
4994	-2.060402	-2.860116	-3.405074	-3.748719	-3.513561	-3.006545	-2.234850	-1.593270	-1.075279	-0.976047	...	1.388947	2.079675
4995	-1.122969	-2.252925	-2.867628	-3.358605	-3.167849	-2.638360	-1.664162	-0.935655	-0.866953	-0.645363	...	-0.472419	-1.310147
4996	-0.547705	-1.889545	-2.839779	-3.457912	-3.929149	-3.966026	-3.492560	-2.695270	-1.849691	-1.374321	...	1.258419	1.907530
4997	-1.351779	-2.209006	-2.520225	-3.061475	-3.065141	-3.030739	-2.622720	-2.044092	-1.295874	-0.733839	...	-1.512234	-2.076075

5 rows × 141 columns

```
# 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
)
```

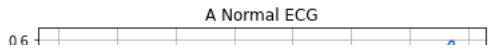
```
#Normalize the data to [0,1].

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)
```

You will train the autoencoder using only the normal rhythms, which are labeled in this dataset as 1. Separate the normal rhythms from the abnormal rhythms.

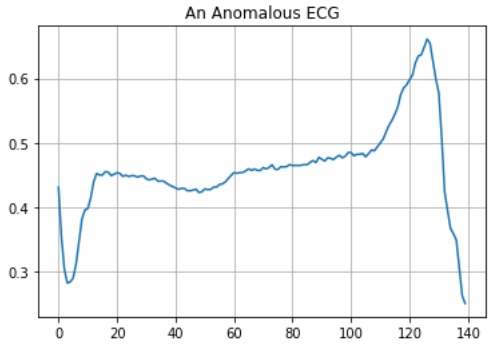
```
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)

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]
```

Plot a normal ECG.

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




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

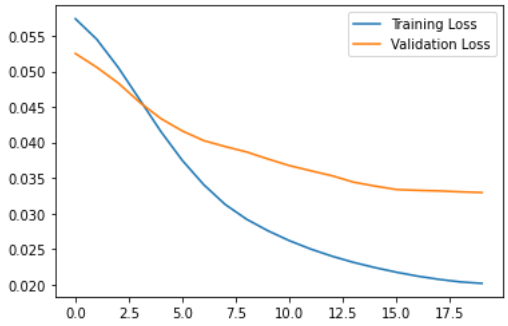



```
#Build the model
class AnomalyDetector(Model):
  def __init__(self):
    super(AnomalyDetector, self).__init__()
    self.encoder = tf.keras.Sequential([
      layers.Dense(32, activation="relu"),
      layers.Dense(16, activation="relu"),
      layers.Dense(8, activation="relu")])

    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')


#Notice that the autoencoder is trained using only the normal ECGs, but is evaluated using the full test set.


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 [==============================] - 1s 39ms/step - loss: 0.0574 - val_loss: 0.0525
     Epoch 2/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0545 - val_loss: 0.0506
     Epoch 3/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0506 - val_loss: 0.0484
     Epoch 4/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0460 - val_loss: 0.0457
     Epoch 5/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0415 - val_loss: 0.0433
     Epoch 6/20
     5/5 [==============================] - 0s 11ms/step - loss: 0.0375 - val_loss: 0.0416
     Epoch 7/20
     5/5 [==============================] - 0s 11ms/step - loss: 0.0340 - val_loss: 0.0403
     Epoch 8/20
     5/5 [==============================] - 0s 11ms/step - loss: 0.0313 - val_loss: 0.0394
     Epoch 9/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0292 - val_loss: 0.0387
     Epoch 10/20
     5/5 [==============================] - 0s 14ms/step - loss: 0.0276 - val_loss: 0.0377
     Epoch 11/20
     5/5 [==============================] - 0s 12ms/step - loss: 0.0262 - val_loss: 0.0367
     Epoch 12/20
     5/5 [==============================] - 0s 11ms/step - loss: 0.0250 - val_loss: 0.0360
     Epoch 13/20
     5/5 [==============================] - 0s 10ms/step - loss: 0.0240 - val_loss: 0.0353
     Epoch 14/20
```

```
5/5 [==============================] - 0s 10ms/step - loss: 0.0231 - val_loss: 0.0344
Epoch 15/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0224 - val_loss: 0.0339
Epoch 16/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0217 - val_loss: 0.0334
Epoch 17/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0212 - val_loss: 0.0333
Epoch 18/20
5/5 [==============================] - 0s 11ms/step - loss: 0.0207 - val_loss: 0.0332
Epoch 19/20
5/5 [==============================] - 0s 12ms/step - loss: 0.0204 - val_loss: 0.0330
Epoch 20/20
5/5 [==============================] - 0s 10ms/step - loss: 0.0202 - val_loss: 0.0329
```

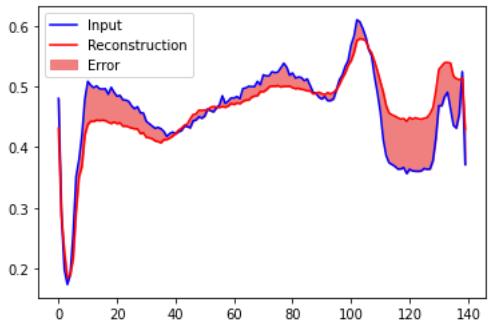
```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f3abd296dd0>
```



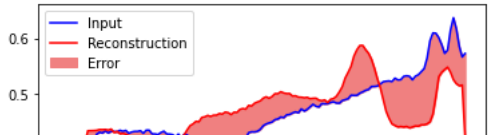
You will soon classify an ECG as anomalous if the reconstruction error is greater than one standard deviation from the normal training examples. First, let's plot a normal ECG from the training set, the reconstruction after it's encoded and decoded by the autoencoder, and the reconstruction error.

```
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()
```



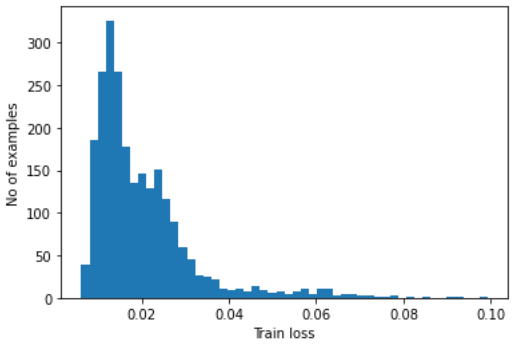
Create a similar plot, this time for an anomalous test example.

```
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()
```



Detect anomalies by calculating whether the reconstruction loss is greater than a fixed threshold. In this tutorial, you will calculate the mean average error for normal examples from the training set, then classify future examples as anomalous if the reconstruction error is higher than one standard deviation from the training set.

```
#Plot the reconstruction error on normal ECGs from the training set
reconstructions = autoencoder.predict(normal_train_data)
train_loss = tf.keras.losses.mae(reconstructions, normal_train_data)
plt.hist(train_loss[None,:], bins=50)
plt.xlabel("Train loss")
plt.ylabel("No of examples")
plt.show()
```



Choose a threshold value that is one standard deviations above the mean.

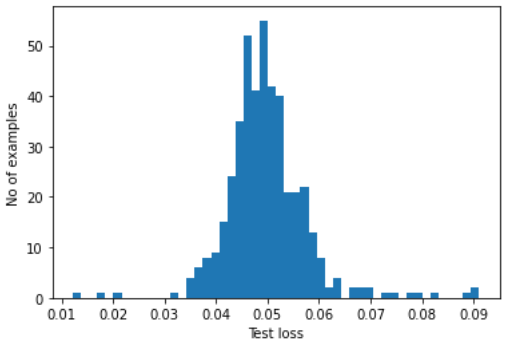
```
threshold = np.mean(train_loss) + np.std(train_loss)
print("Threshold: ", threshold)
```

```
Threshold:  0.03208865
```

Note: There are other strategies you could use to select a threshold value above which test examples should be classified as anomalous, the correct approach will depend on your dataset.

If you examine the reconstruction error for the anomalous examples in the test set, you'll notice most have greater reconstruction error than the threshold. By varing the threshold, you can adjust the precision and recall of your classifier.

```
reconstructions = autoencoder.predict(anomalous_test_data)
test_loss = tf.keras.losses.mae(reconstructions, anomalous_test_data)

plt.hist(test_loss[None, :], bins=50)
plt.xlabel("Test loss")
plt.ylabel("No of examples")
plt.show()
```



Classify an ECG as an anomaly if the reconstruction error is greater than the threshold.

```
def predict(model, data, threshold):
  reconstructions = model(data)
  loss = tf.keras.losses.mae(reconstructions, data)
  return tf.math.less(loss, threshold)
```

```
def print_stats(predictions, labels):
  print("Accuracy = {}".format(accuracy_score(labels, predictions)))
  print("Precision = {}".format(precision_score(labels, predictions)))
  print("Recall = {}".format(recall_score(labels, predictions)))


preds = predict(autoencoder, train_data, threshold)
print_stats(preds, train_labels)
```

```
Accuracy = 0.9354677338669335
Precision = 0.9892873777363763
Recall = 0.9003815175922001
```

```
preds = predict(autoencoder, test_data, threshold)
print_stats(preds, test_labels)
```

```
Accuracy = 0.945
Precision = 0.9922027290448343
Recall = 0.9089285714285714
```

+ Code + Text