1. ROC analysis

   Loading the files and then determining the positive and negative values from the data. Then determining if the detector output is true based on a threshold value. We will use all the detector output values as a threshold to get all the possible different ROC curve values for the data. This gives us the ROC curve in figure 1.

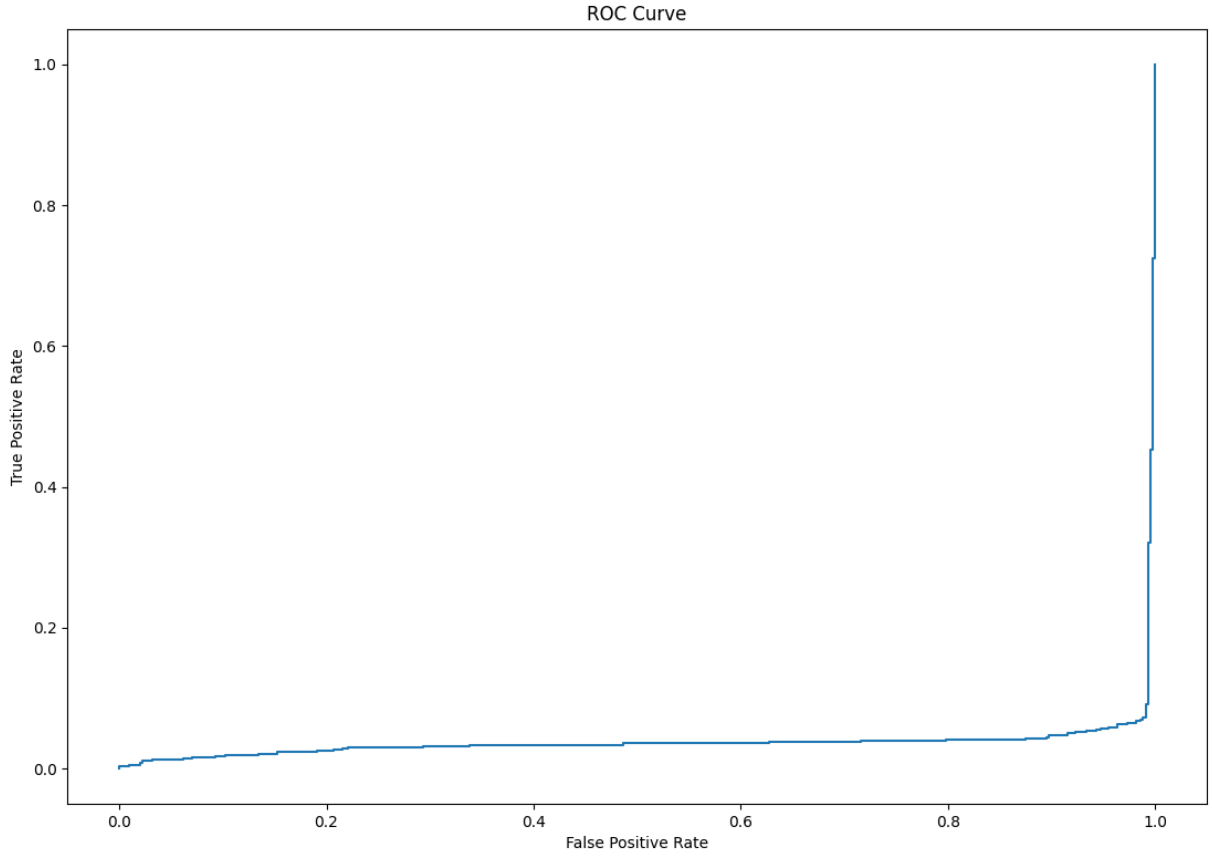

Figure 1: ROC curve from the data

2. Noisy MNIST Fashion classification

   We load the MNIST Fashion data and add some noise to the training and testing images.

   We then define the CNN classifier model for the data and train it with clean images. The model configuration is shown in figure 2.

```
# Defining a CNN model

# declare input shape
input = tf.keras.Input(shape=(28,28,1))
# Block 1 (convolution)
conv1 = tf.keras.layers.Conv2D(32, 3, strides=1, activation="relu")(input)
#x = tf.keras.layers.MaxPooling2D(3)(x)
#x = tf.keras.layers.BatchNormalization()(x)
# Block 2 (convolution 2)
conv2 = tf.keras.layers.Conv2D(64, 3, strides=1, activation="relu")(conv1)
# Block 3 (full connected9)
fc = tf.keras.layers.Flatten()(conv2)
fc = tf.keras.layers.Dense(10)(fc)
# Finally, we add a classification layer.
output = tf.keras.layers.Dense(10, activation="softmax")(fc)
# bind all
cnn_model = tf.keras.Model(input, output)
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
cnn_model.compile(loss=loss_fn, optimizer="adam", metrics=["accuracy"])
cnn_model.summary()
```

Figure 2: CNN classifier model

We can then evaluate how good the model is for the clean and noisy test images. The accuracy for the clean test images is a fairly good 90% and for the noisy test images 44%. The evaluation and resulting accuracies are presented in figure 3.

```
test_loss, test_acc = cnn_model.evaluate(test_images,  test_labels, verbose=2)
print('\nTest accuracy for clean images:', test_acc)

test_loss, test_acc = cnn_model.evaluate(test_images_noisy,  test_labels,
                                         verbose=2)
print('\nTest accuracy for noisy images:', test_acc)
```

```
313/313 - 1s - loss: 0.4335 - accuracy: 0.9066

Test accuracy for clean images: 0.9065999984741211
313/313 - 1s - loss: 4.4777 - accuracy: 0.4453

Test accuracy for noisy images: 0.44530001282691956
```

Figure 3: Model evaluation and classification results

Defining a CNN autoencoder model to denoise the noisy images. The autoencoder is trained with the noisy images so it learns to denoise these. The model configuration is shown in Figure 4

```
class Denoise(Model):

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

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

autoencoder.fit(train_images_noisy, train_images,
                epochs=10,
                shuffle=True,
                validation_data=(test_images_noisy, test_images))
```

Figure 4: CNN autoencoder configuration and training

The results of the autoencoder are presented in figure 5.

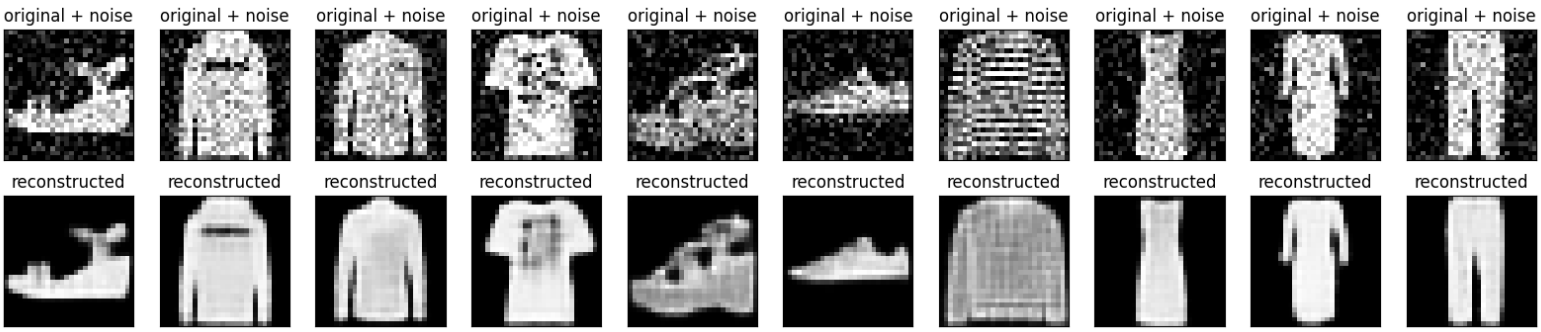

Figure 5: Noisy images and the denoised images done with the autoencoder

The classification accuracy with the CNN classification model presented in figure 2 for the images denoised with the autoencoder is around 82% which is way better than for the noisy images. The model evaluation and results are presented in figure 6.

```
denoised_test_images = autoencoder.call(test_images_noisy)

test_loss, test_acc = cnn_model.evaluate(denoised_test_images,  test_labels,
                                         verbose=2)
print('\nTest accuracy for denoised images:', test_acc)
```

```
313/313 - 1s - loss: 0.6525 - accuracy: 0.8239

Test accuracy for denoised images: 0.8238999843597412
```

Figure 6: Model evaluation and classification results

Finally training the original CNN model with the noisy images results in classification accuracy of around 85% which is better than for the denoised images. So if the goal is strictly for classification and the model architecture is defined this way it seems that the autoencoder doesn't offer much as it performs worse than just straight training with the noisy images. If the goal is to present the images to people however the results are good. The model training, evaluation and results are presented in figure 7.

```
history = cnn_model.fit(train_images_noisy, train_labels, epochs=10)

test_loss, test_acc = cnn_model.evaluate(test_images_noisy,  test_labels,
                                         verbose=2)
print('\nTest accuracy with noisy trained model for noisy images:', test_acc)
```

```
313/313 - 1s - loss: 1.2837 - accuracy: 0.8515

Test accuracy with noisy trained model for noisy images: 0.8514999747276306
```

Figure 7: Model training, evaluation and classification results