DATA.ML.200 Pattern Recognition and Machine Learning

Exercise Set 4: Detector evaluation and autoencoder denoising

Miska Romppainen
H274426

April 2023

1 ROC analysis

In the first task the goal was to write a code that calculates and plots the ROC curve of true positive rate (recall) vs false positive rate with 1-precision, from the "detector_groundtruth.dat" and "detector_output.dat" files in the moodle. ROC stands for "Receiver Operating Characteristic". According to wikipedia, ROC analysis "is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The method was originally developed for operators of military radar receivers starting in 1941, which led to its name."

A ROC analysis can be done to "detector_groundtruth.dat" and "detector_output.dat". To get the true positive rate (a.k.a recall), we calculate the recall $tp/(tp + fn)$, where tp is the true positives, and fn is the false negatives. And using the function roc_curve from the library *sklearn.metrics*, we get the false positive rate (a.k.a 1-precision). Plotting these together we get the figure 1.

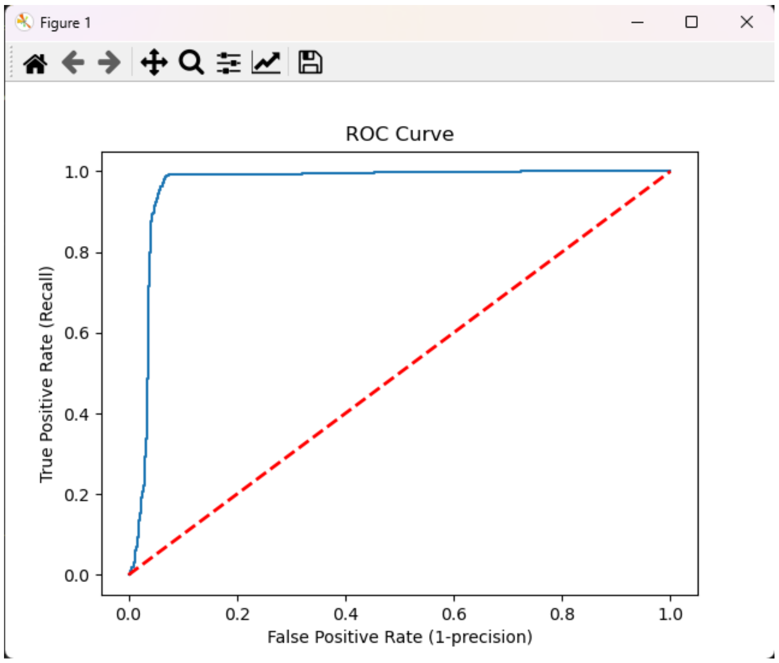


Figure 1: ROC curve

2 Noisy MNIST Fashion classification

In the task 2 our goal was to create an auto-encoder that trains the neural network to encode and decode noisy images and to report different values from our network.

First we need to load the data and to noise it using the given example.

```python
# load data
(train_images, train_labels), (test_images, test_labels) = keras.datasets.fashion_mnist.load_data()

# normalize data
train_images = train_images / 255.0
test_images = test_images / 255.0

# add noise to images
noise_factor = 0.2
train_images_noisy = train_images + noise_factor * tf.random.normal(shape=train_images.shape)
test_images_noisy = test_images + noise_factor * tf.random.normal(shape=test_images.shape)

# clip values to (0,1) range
train_images_noisy = tf.clip_by_value(train_images_noisy, clip_value_min=0., clip_value_max=1.)
test_images_noisy = tf.clip_by_value(test_images_noisy, clip_value_min=0., clip_value_max=1.)
```

Listing 1: Loading the data and adding noise

After adding noise, we can define the CNN model, compile it and train it using the loaded train images and labels.

```python
# The CNN classifier model for the MNIST Fashion
CNNmodel = tf.keras.Sequential([
tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)),
tf.keras.layers.MaxPooling2D((2,2)),
tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
tf.keras.layers.MaxPooling2D((2,2)),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(10)
])

# Compile the model
CNNmodel.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
```

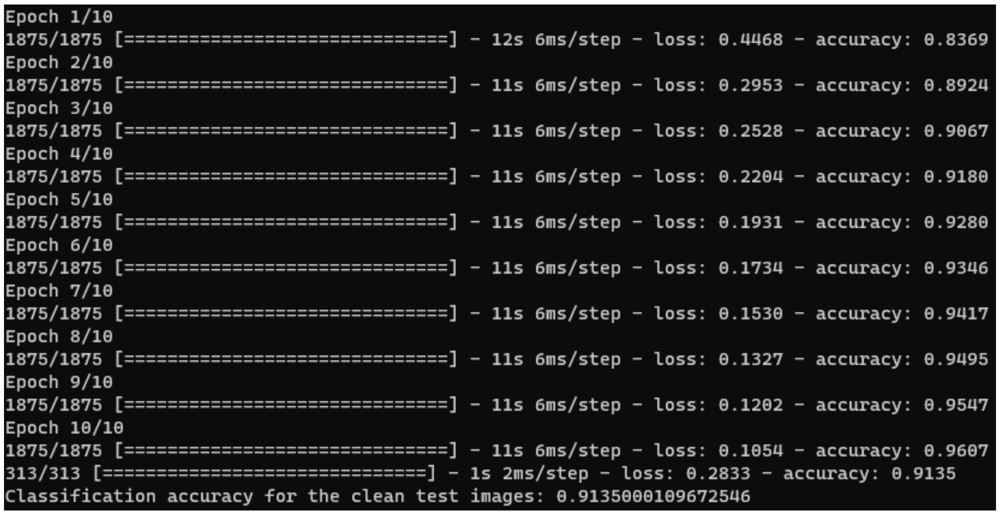
Listing 2: Defining the CNN

After compiling the CNN, we can train the model using the clean images and the noisy images, and check the networks accuracy evaluating the output to the test images.

```python
# Evaluate the model with clean test images
test_loss, test_acc = CNNmodel.evaluate(test_images, test_labels)
print('Classification accuracy for the clean test images:',
    test_acc)
```

Listing 3: Training the model and printing the accuracy for clean images

For clean images we get the output in image 2, which is around 91%.



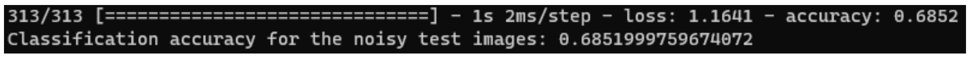
```
Epoch 1/10
1875/1875 [==============================] - 12s 6ms/step - loss: 0.4468 - accuracy: 0.8369
Epoch 2/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2953 - accuracy: 0.8924
Epoch 3/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2528 - accuracy: 0.9067
Epoch 4/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2204 - accuracy: 0.9180
Epoch 5/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1931 - accuracy: 0.9280
Epoch 6/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1734 - accuracy: 0.9346
Epoch 7/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1530 - accuracy: 0.9417
Epoch 8/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1327 - accuracy: 0.9495
Epoch 9/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1202 - accuracy: 0.9547
Epoch 10/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1054 - accuracy: 0.9607
313/313 [==============================] - 1s 2ms/step - loss: 0.2833 - accuracy: 0.9135
Classification accuracy for the clean test images: 0.9135000109672546
```

Figure 2: Classification for clean images

```python
# Evaluate the model with noisy test images
test_loss , test_acc = CNNmodel.evaluate(test_images_noisy ,
    test_labels)
print('Classification accuracy for the noisy test images:',
    test_acc)
```

Listing 4: Training the model and printing the accuracy for noisy images

For noisy images we get the output in image 3, which is around 69%.



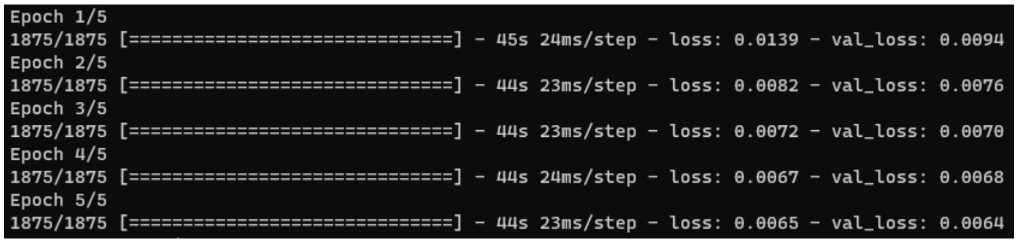
```
313/313 [==============================] - 1s 2ms/step - loss: 1.1641 - accuracy: 0.6852
Classification accuracy for the noisy test images: 0.6851999759674072
```

Figure 3: Classification for noisy images

After checking the accuracy of clean and noisy images we can get started defining a CNN auto-encoder model to denoise the noisy images and train it using the noisy training images.

```
 # Autoencoder for denoising
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(32, (3, 3), activation='relu',
    padding='same'),
            tf.keras.layers.MaxPooling2D((2, 2), padding='same'),
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
    padding='same'),
            tf.keras.layers.MaxPooling2D((2, 2), padding='same')
        ])
        self.decoder = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
    padding='same'),
            tf.keras.layers.UpSampling2D((2, 2)),
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu',
    padding='same'),
            tf.keras.layers.UpSampling2D((2, 2)),
            tf.keras.layers.Conv2D(1, (3, 3), activation='sigmoid',
     padding='same')
        ])

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

autoencoder = Denoise()

# Compile and train the autoencoder model using the noisy images as
     both input and target
autoencoder.compile(optimizer='adam', loss=tf.keras.losses.
    MeanSquaredError())

history = autoencoder.fit(train_images_noisy, train_images,
                    # epochs 5-10 should be enough
                    epochs=5,
                    shuffle=True,
                    validation_data=(test_images_noisy, test_images
    ))
```

Listing 5: Auto-encoder for denoising



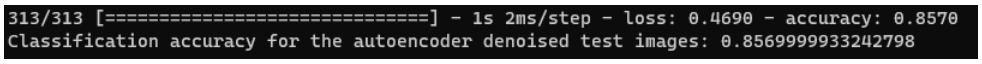
```
Epoch 1/5
1875/1875 [==============================] - 45s 24ms/step - loss: 0.0139 - val_loss: 0.0094
Epoch 2/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0082 - val_loss: 0.0076
Epoch 3/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0072 - val_loss: 0.0070
Epoch 4/5
1875/1875 [==============================] - 44s 24ms/step - loss: 0.0067 - val_loss: 0.0068
Epoch 5/5
1875/1875 [==============================] - 44s 23ms/step - loss: 0.0065 - val_loss: 0.0064
```

Figure 4: Training for 5 epochs

After training the network we can check the summary of the encoding and decoding networks - to see how they behave. After which we can also plot the loss, from which we get the output presented in the figure 5, the accuracy is around 86%. From the plotting we get the loss plot output in the figure 6.

```
# Check the summary to see what they do
autoencoder.encoder.summary()
autoencoder.decoder.summary()

# Plot the Loss
plt.plot(history.history['loss'])
plt.show()

# Use the autoencoder to denoise the noisy test images
denoised_test_images = autoencoder.predict(test_images_noisy)
# Evaluate the classifier model with denoised test images
test_loss, test_acc = CNNmodel.evaluate(denoised_test_images,
    test_labels)
print('Classification accuracy for the autoencoder denoised test
    images:', test_acc)
```

Listing 6: Evaluate the auto-encoder



```
313/313 [==============================] - 1s 2ms/step - loss: 0.4690 - accuracy: 0.8570
Classification accuracy for the autoencoder denoised test images: 0.8569999933242798
```

Figure 5: Classification for Auto-encoder denoised images

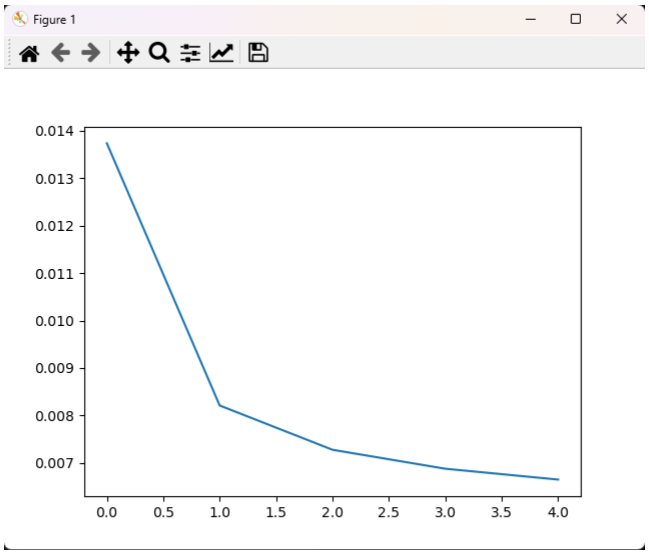


Figure 6: Loss Plot of the auto-encoder

We can also print out the noised images next to the reconstructed images from the auto-encoder, in order to see how the auto-encoder works, this output can be seen in the figure 7.

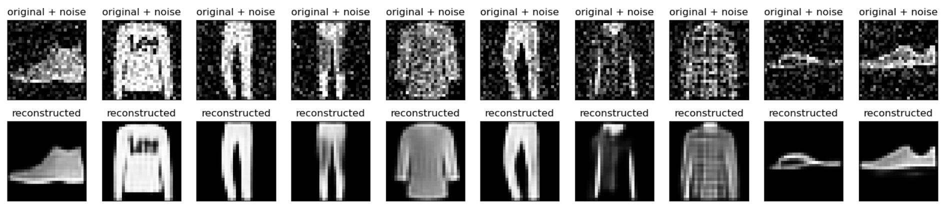
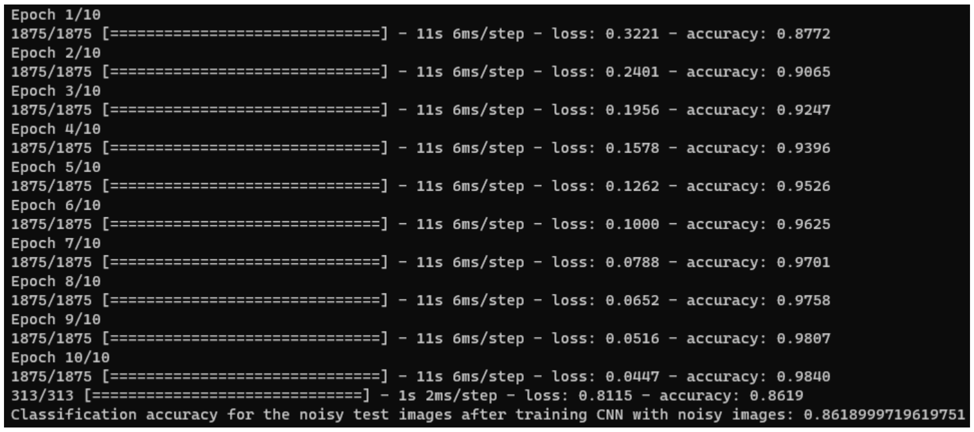


Figure 7: Noisy and reconstructed images

After we train the original CNN with the noisy training images, we get the accuracy of 86%, this can be seen in the figure 8.



```
Epoch 1/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.3221 - accuracy: 0.8772
Epoch 2/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.2401 - accuracy: 0.9065
Epoch 3/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1956 - accuracy: 0.9247
Epoch 4/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1578 - accuracy: 0.9396
Epoch 5/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1262 - accuracy: 0.9526
Epoch 6/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.1000 - accuracy: 0.9625
Epoch 7/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0788 - accuracy: 0.9701
Epoch 8/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0652 - accuracy: 0.9758
Epoch 9/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0516 - accuracy: 0.9807
Epoch 10/10
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0447 - accuracy: 0.9840
313/313 [==============================] - 1s 2ms/step - loss: 0.8115 - accuracy: 0.8619
Classification accuracy for the noisy test images after training CNN with noisy images: 0.8618999719619751
```

Figure 8: Accuracy of noisy test images after training with noisy training images

We can see that training with the noisy images and the auto-encoder output roughly the same accuracy of 86%, even though the auto-encoder has been trained only for 5 epochs and the noisy images trained for 10 epochs.