Exercise

Using an autoencoder to pre-train an image classifier: split the MNIST data to training and test sets. 1) Train a deep denoising autoencoder on the full training set. Check that the autoencoder works well (i.e. check if the images are fairly well reconstructed), and visualize the low-level features. Visualize images that most activate each neuron in the coding layer. 2) Build a classification deep neural network, by reusing the lower layers of the autoencoder. Train it using only 10% of the training set. How does the performance of this classifier compare to the same classifier trained on the full training set?

Approach:

- Developed Encoder-Decoder model
- Trained the model with MNIST data
    - For this, we don't need labelled data as we only want to project the higher dimensional image to non-linear lower-dimensional data
- And the error matrix we can use can be MSE
- Lower level filters are visualized
    - conv2D_1
    - conv2D_2
- CNN based supervised model is trained with using the Autoencoder model's early layers
    - Using only 10% of the training dataset
- The same model was trained with the full dataset
- The accuracy and other evaluation matrix are compared

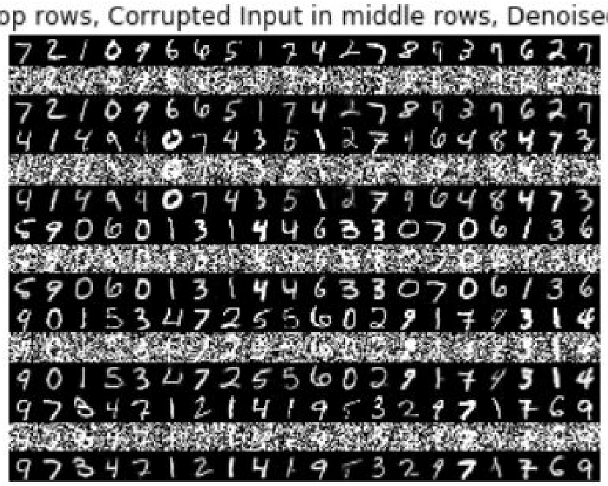
Results:

- Structure of Encoder-Decoder model:

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
latent_vector (Dense)        (None, 32)                100384
=================================================================
Total params: 119,200
Trainable params: 119,200
Non-trainable params: 0
_________________________________________________________________
Model: "decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3136)              103488
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 28, 28, 1)         289
_________________________________________________________________
decoder_output (Activation)  (None, 28, 28, 1)         0
=================================================================
Total params: 159,169
Trainable params: 159,169
Non-trainable params: 0
```

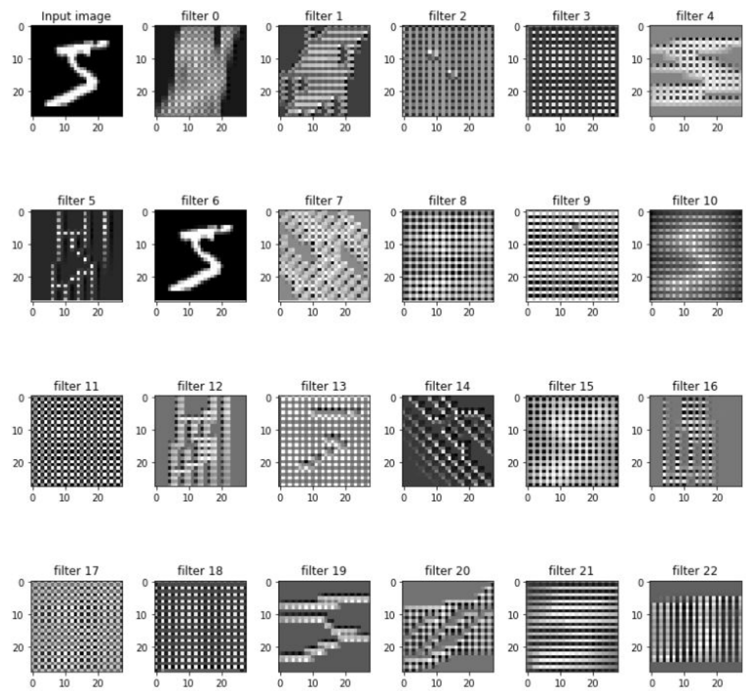
Denoising on “noisy” test images:

Original images in top rows, Corrupted Input in middle rows, Denoised Input in third rows

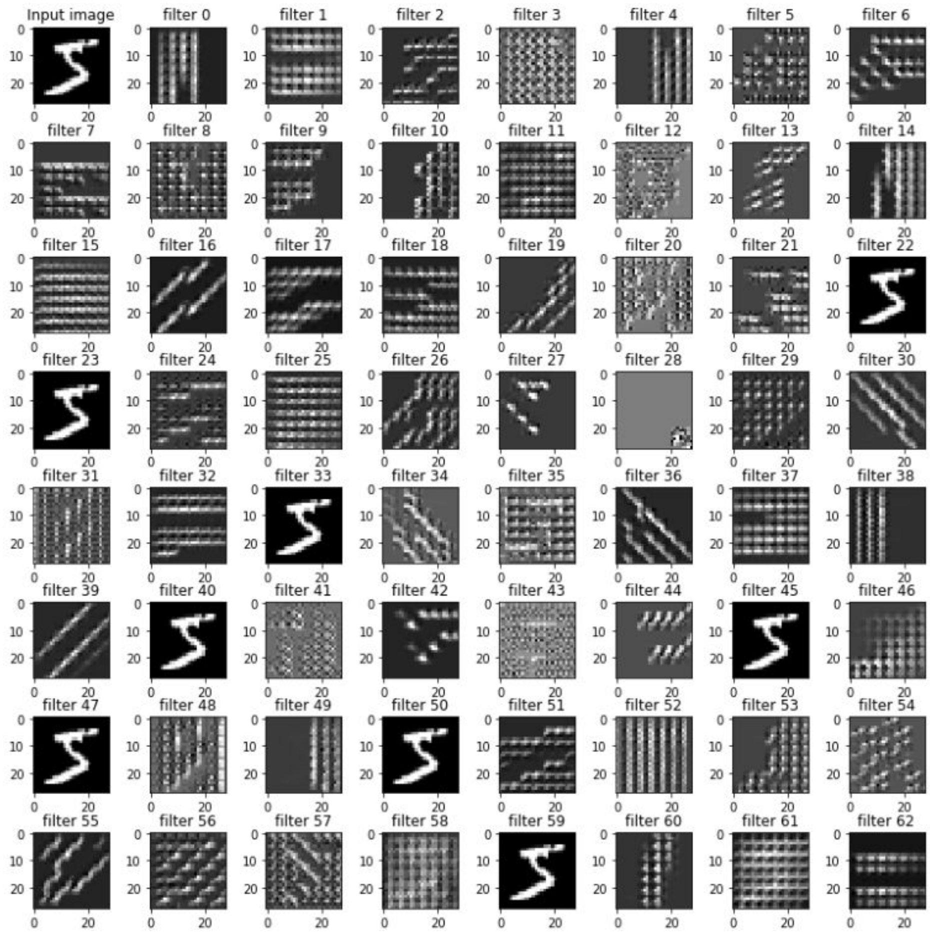


- Visualize images that most activate each neuron in the encoding layer

Conv2D_1 output:



Conv2D_2 output:



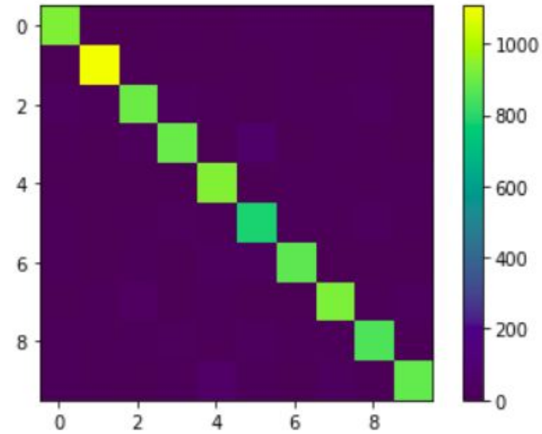
Part 2 :

Using the Encoder model as a lower level feature detector and train the model:

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder (Model)              (None, 32)                119200
_________________________________________________________________
dense_5 (Dense)              (None, 16)                528
_________________________________________________________________
dense_6 (Dense)              (None, 16)                272
_________________________________________________________________
dense_7 (Dense)              (None, 10)                170
=================================================================
Total params: 120,170
Trainable params: 970
Non-trainable params: 119,200
```

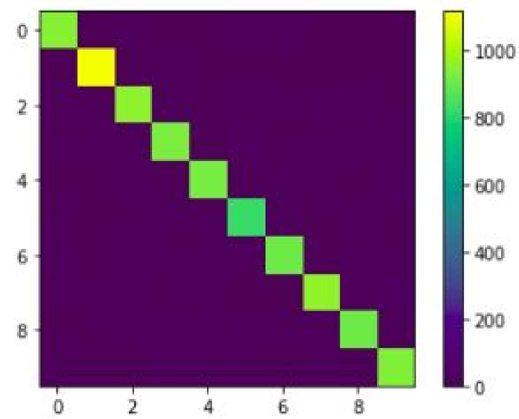
Result with 10% of data:

	precision	recall	f1-score	support
0	0.92	0.95	0.93	980
1	0.97	0.97	0.97	1135
2	0.91	0.87	0.89	1032
3	0.92	0.89	0.90	1010
4	0.88	0.95	0.92	982
5	0.85	0.88	0.86	892
6	0.93	0.92	0.92	958
7	0.91	0.90	0.91	1028
8	0.90	0.88	0.89	974
9	0.93	0.88	0.90	1009
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000



Result with 100% of data:

	precision	recall	f1-score	support
0	0.96	0.97	0.96	980
1	0.97	0.99	0.98	1135
2	0.93	0.93	0.93	1032
3	0.95	0.92	0.94	1010
4	0.96	0.94	0.95	982
5	0.92	0.93	0.93	892
6	0.94	0.95	0.94	958
7	0.95	0.94	0.94	1028
8	0.93	0.93	0.93	974
9	0.92	0.93	0.93	1009
accuracy			0.94	10000
macro avg	0.94	0.94	0.94	10000
weighted avg	0.94	0.94	0.94	10000



Conclusion:

- We can say that just by using the initial layers of AE, the model is able to fit the dataset with really very fewer data without losing accuracy.
- AE is able to denoise the images very effectively
  - The images contain salt and pepper noise the extent to which humans are not able to recognize the digit, yet AE performs well.