Homework 4 submission

ECET 612 — Applied Machine Learning

Murat Isik

June 6, 2022

1 Submitted files

For this assignment, besides this report, the following archives were created:

1.1 SRC Folder

* *" Convolution_Encoder"*: This **function** is a discrete linear time-invariant system. Every output of an encoder can be described by its own transfer function, which is closely related to the generator polynomial.
* *" Convolution_Decoder"*: This **function** can be done in two approaches. One approach is called hard decision decoding which uses Hamming distance as a metric to perform the decoding operation, whereas the soft decision decoding uses Euclidean distance as a metric.
* *" Sequential Model"*: This **function** is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.
* *" History"*: This **function** above has an optional parameter to specify a maximum number of lines to be printed.
* "+ ***"project4.ipyb"***: This **script** initiates all the necessary function calls, and also generates the graphs for the fashion dataset. A total of 58 epochs have been set aside for training. To get the greatest outcomes from a model, it is also vital to keep it from overtraining and from training too deeply. First, it calculates with the information provided by the training and validation. Then it finds the plot of the graphs by using the *different python* functions. Afterward, it creates graph models. It is critical to keep the model from overtraining since this might widen the gap between training and validation loss. The goal of this project is to clean up noisy image and predict the digits from the noisy data.

2 Discussion

This dataset is to be a noisy variation of the MNIST digit set, with the noise coming from an internal sense of style.

Figure 1: Project dataset.

Figure 2: Project noisy dataset.

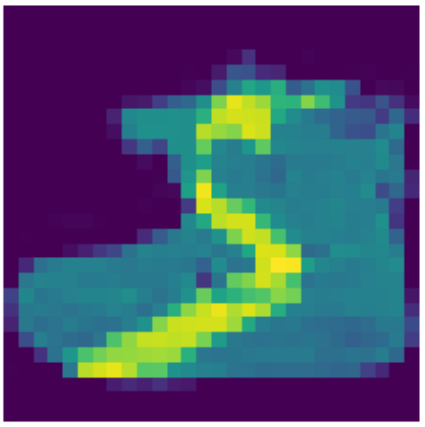

Figure 3: Digit Image.

3 Choosing an Architecture

CNN autoencoder is used in this software. There are two layers with size 12 and 14 filters, as well as one pooling layer of size 2. Finding various results that work has proven to be difficult. I'll describe different architectures I attempted and how the results turned out. To ensure that the reconstructed images can be detected accurately by human eyes, they were plotted (textbook functions). The original noisy image is in the top row, while the denoised images are in the bottom row. I displayed first autoencoder output in Figure 4.

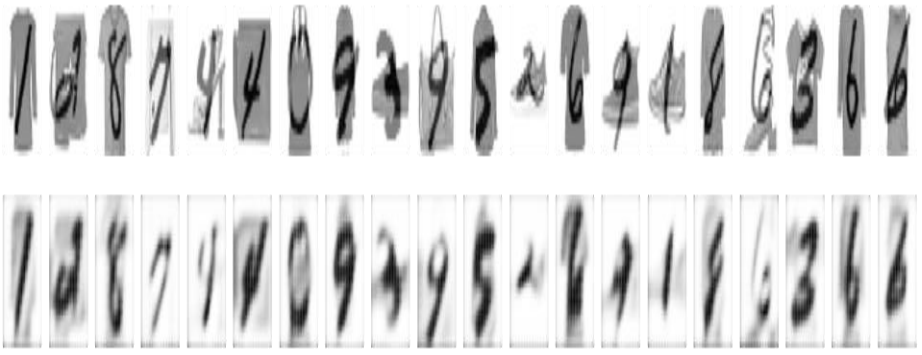

Figure 4: First autoencoder output

```
    from sklearn.model_selection import train_test_split
#X_train, X_test = train_test_split(X_train, test_size=0.2,
                        random_state=42)
  X_train, X_valid= X_train_in[:-5000], X_train_in[-5000:]
                    X_train_set = X_train
                    X_valid_set = X_valid

  y_train, y_valid= y_train_in[:-5000], y_train_in[-5000:]
                    y_train_set = y_train
                    y_valid_set = y_valid
                  print(X_train_set.shape)
                  print(y_train_set.shape)
```

Figure 5: Split dataset code

Datasets were imported and separated into train and valid sets. The next stage for me was to try to reduce the number of parameters and make it easier on the eyes. Setting the training epochs higher initially resulted in a lot better outline, but I still needed to cut the parameter count. The steps I made are listed below, along with pictures of a few recoveries that match to the alterations.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 reshape_2 (Reshape)          (None, 28, 28, 1)         0

 conv2d_2 (Conv2D)            (None, 28, 28, 12)        120

 max_pooling2d_1 (MaxPooling  (None, 14, 14, 12)        0
 2D)

 conv2d_3 (Conv2D)            (None, 14, 14, 14)        1526

=================================================================
Total params: 1,646
Trainable params: 1,646
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Keras Model Sequential

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_3 (Sequential)   (None, 14, 14, 14)        1646

 sequential_4 (Sequential)   (None, 28, 28)            1633

=================================================================
Total params: 3,279
Trainable params: 3,279
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Keras model combined encoder and decoder

The final model has a total of 3279 parameters

3. Results

3.1 Dataset

Data (X_train,y_train) will split into 2 sets;

- **Train set:** The training set receives 80% of the whole data, whereas the validation set receives 20% of the total data. The actual training data contained 60,000 photos inall.

3.2 Performance and Results

I eventually settled on the 20-epoch training time. Although it can occasionally reduce the visibility of already visible numerals, it does a better job of cleaning out the number and making it far more visible in more challenging circumstances like the first image. For a modest number of training parameters, the results are reasonable, with the greatest accuracy percent of 80.83.

```
Epoch 1/20
1719/1719 [==============================] - 20s 4ms/step - loss: 0.6908 - binary_accuracy: 0.5161 - val_loss: 0.6850 - val_binary_accuracy: 0.6022
Epoch 2/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.6756 - binary_accuracy: 0.6870 - val_loss: 0.6622 - val_binary_accuracy: 0.7547
Epoch 3/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.6394 - binary_accuracy: 0.7773 - val_loss: 0.6091 - val_binary_accuracy: 0.7911
Epoch 4/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.5819 - binary_accuracy: 0.7962 - val_loss: 0.5558 - val_binary_accuracy: 0.8019
Epoch 5/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.5402 - binary_accuracy: 0.8071 - val_loss: 0.5211 - val_binary_accuracy: 0.8088
Epoch 6/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.5075 - binary_accuracy: 0.8088 - val_loss: 0.4893 - val_binary_accuracy: 0.8088
Epoch 7/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.4747 - binary_accuracy: 0.8088 - val_loss: 0.4556 - val_binary_accuracy: 0.8088
Epoch 8/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.4386 - binary_accuracy: 0.8088 - val_loss: 0.4179 - val_binary_accuracy: 0.8088
Epoch 9/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3980 - binary_accuracy: 0.8088 - val_loss: 0.3755 - val_binary_accuracy: 0.8088
Epoch 10/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3532 - binary_accuracy: 0.8088 - val_loss: 0.3314 - val_binary_accuracy: 0.8088
Epoch 11/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.3144 - binary_accuracy: 0.8088 - val_loss: 0.3006 - val_binary_accuracy: 0.8088
Epoch 12/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2902 - binary_accuracy: 0.8090 - val_loss: 0.2827 - val_binary_accuracy: 0.8091
Epoch 13/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2757 - binary_accuracy: 0.8092 - val_loss: 0.2710 - val_binary_accuracy: 0.8093
Epoch 14/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2655 - binary_accuracy: 0.8092 - val_loss: 0.2620 - val_binary_accuracy: 0.8091
Epoch 15/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2571 - binary_accuracy: 0.8090 - val_loss: 0.2542 - val_binary_accuracy: 0.8090
Epoch 16/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2497 - binary_accuracy: 0.8089 - val_loss: 0.2474 - val_binary_accuracy: 0.8089
Epoch 17/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2432 - binary_accuracy: 0.8089 - val_loss: 0.2411 - val_binary_accuracy: 0.8087
Epoch 18/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2372 - binary_accuracy: 0.8087 - val_loss: 0.2353 - val_binary_accuracy: 0.8085
Epoch 19/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2317 - binary_accuracy: 0.8085 - val_loss: 0.2300 - val_binary_accuracy: 0.8083
Epoch 20/20
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2266 - binary_accuracy: 0.8083 - val_loss: 0.2250 - val_binary_accuracy: 0.8082
```

Figure 8: Keras Model Sequential

3.3 Outputs

A confusion matrix, given a CNN, illustrates where the model is getting confused, i.e. which classes the model properly predicts and which classes the model erroneously predicts. More training parameters and deeper layers in the autocoder could improve the model's performance. However, in the context of this study, the model performs admirably in reassembling denoised photos. Figure 9, Figure 10, Figure 11 represent the outputs of the proposed system.

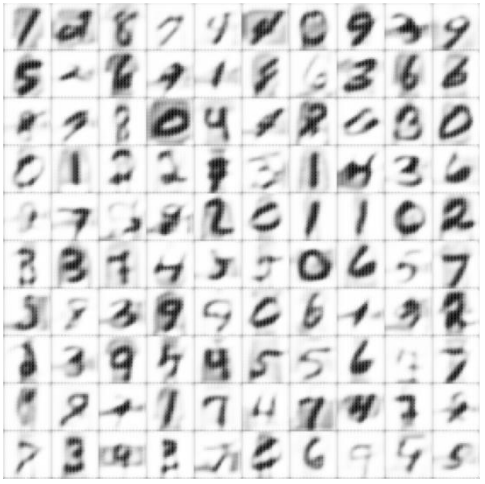

Figure 9: Denoised Image

Figure 10: Example of Y Valid Set

Figure 11: Example of X Valid Set