CS 4/510: Computer Vision & Deep Learning, Summer 2022

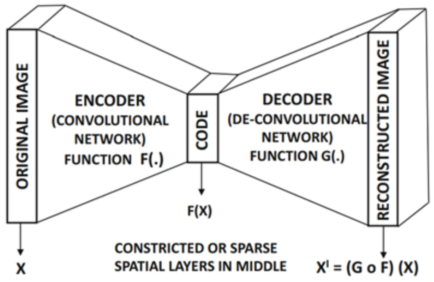
Programming Assignment #4

A. Rhodes

Note: This assignment is **due by Sunday, 8/7 at 10:00pm**; turn in the assignment by email to our TA.

Note that there are (2) exercises listed below. **Complete one of the exercises of your choosing** (please don't do both).

Exercise #1: Convolutional Autoencoder


ORIGINAL IMAGE
ENCODER (CONVOLUTIONAL NETWORK) FUNCTION F(.)
CODE
DECODER (DE-CONVOLUTIONAL NETWORK) FUNCTION G(.)
RECONSTRUCTED IMAGE
X
F(X)
CONSTRICTED OR SPARSE SPATIAL LAYERS IN MIDDLE
X' = (G o F) (X)

As described in lecture (see "Visualization" lecture series), in this exercise you will design and train a convolutional autoencoder (CAE) using the Fashion-MNIST dataset (example shown below): https://www.tensorflow.org/datasets/catalog/fashion_mnist; in Keras:

```
tf.keras.datasets.fashion_mnist.load_data()
```

Pre-process the data by dividing each image by 255.

Step 1: Design your CAE

For the basic CAE architecture, please use the basic template:

Encoder: Conv2D ⟶ Maxpooling ⟶ Conv2D ⟶ Maxpooling, etc.

Decoder: Conv2D ⟶ Upsampling2D ⟶ Conv2D ⟶ Upsampling, etc.

Please use a high-level DL library to construct your CAE, e.g., Keras/TF, etc. I recommend using RELU activation for all layers, with the exception of the last layer of the decoder sub-network, which should use a sigmoid activation. You are welcome to experiment with the width of each layer (i.e., number of conv filters), and the total number of layers (include at least three in each of the encoder/decoder sub-networks), but please architect your encoder and decoders **so that they are symmetric**. For example, if the first layer of your encoder has 32 filters, the last layer of your decoder should likewise have 32 filters. The encoder and decoder should have the same number of layers. **Include a model summary of your CAE in your assignment write-up**.

If you want to experiment with batch normalization, dropout, and other techniques we have discussed in class, feel free to do this – please include any relevant details about these design choices in your write-up.

Please read the following comments carefully:

***Note**: Design your CAE so that the last layer of your encoder is followed by a flatten operation. The middle layer of your model, i.e., the "**bottleneck layer**" (see the layer labeled "CODE" in the CAE diagram above) should be a **dense layer with two output neurons** (yes, only two) whose input is the flattened output of the encoder. In this way, the output of the encoder subnetwork is a 2D array.

The decoder portion of the CAE takes as input this 2D array; I recommend using a dense layer as the first layer of the encoder subnetwork (just as the previously described dense layer was the last layer of your encoder). Following the first dense layer in your decoder, you will need to **reshape the output** of this dense layer into a 3D tensor so that it can be passed successfully to a conv2D layer (corresponding with the last conv2D layer from your encoder).

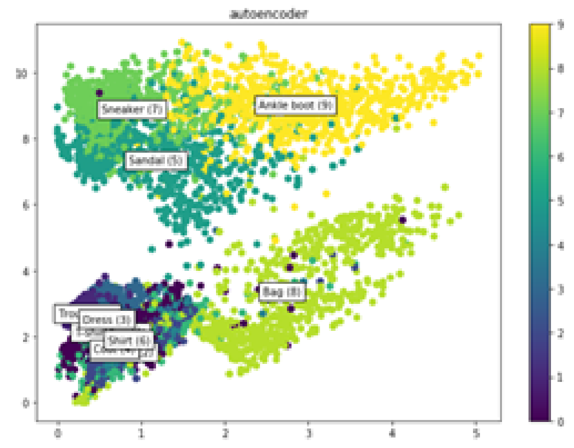
For example, after the bottleneck layer we have a 2D vector. Suppose that the first layer of the encoder network is a Dense(2,4) layer (2 inputs, 4 outputs); you should then reshape this vector to, say, dimension (2,2,1) (still has four values), so that it can be passed to a conv2D layer, as the conv2D function will expect a 3D tensor.*

Part of the challenge of this exercise rests in designing the CAE architecture in a symmetric fashion (encoder/decoder have same number of layers and same width, i.e. number of corresponding filters); finally, you need to design the CAE so that the input and output are exactly the same dimension (otherwise we aren't reconstructing the input appropriately). To this end, for debugging purposes, I recommend checking/confirming the dimension of the output of each layer of your CAE as you pass an image though it – feel free to use basic debug stop conditions as needed.

Step 2: When you have your CAE architected successfully, train your model using a standard optimizer (Adam, etc.), using **binary cross-entropy loss**; make sure that your loss is computed with respect to the input image and the reconstructed output of your CAE. **Include a graph of your training/test loss per epoch with your assignment write-up.** Please include several examples of test images and their corresponding reconstruction, generated by your model.

Step 3: Train a denoising CAE. Using your same CAE architecture from before, randomly add Gaussian noise, drawn from a standard normal distribution: $N(0,1)$ to your input and test images. Retrain your CAE using the noisy training data (but with the non-noisy images as the ground-truth for reconstruction). As in the previous step, use binary cross-entropy loss and **include a graph of your training/test loss per epoch**. Please include several examples of test images and their corresponding reconstruction, generated by your model.

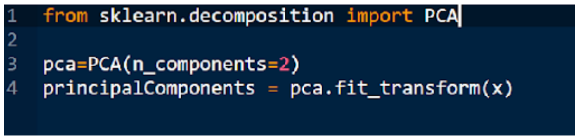
Step 4: Pass each test image through your trained (non-denoising) CAE and extract the 2D latent feature from your encoder sub-network (if you don't want to run all of the test data, it is fine to use 500 or 1000 randomly sampled test datapoints). Plot all of these 2D points in graph, and color code them so that the color of each point corresponds with the ground-truth class label. Plot this 2D projection learned by your CAE to get a qualitative visualization of the dimensionality reduction process learned by your CAE. Your result should look something like the image below (results will vary depending on CAE architecture):



Include this 2D plot in your write-up; provide some insights in your write-up reflecting on this visualization.

Finally, compare the CAE-derived 2D projection of the Fashion-MNIST dataset to a standard dimensionality reduction algorithm: PCA. Please don't implement PCA from scratch; use a built-in library function.

I personally recommend using the **scikit learn** library (https://scikit-learn.org). Write a for-loop over either the entire test dataset or a reasonably sized sample of 500 or 1000 test datapoints; for each test image **x**, simply execute:



```python
from sklearn.decomposition import PCA

pca=PCA(n_components=2)
principalComponents = pca.fit_transform(x)
```

Which will yield the PCA-derived 2D compressed version of the test image **x**. As you did for the CAE-derived 2D projections, plot the PCA-derived 2D projection. Include this 2D plot in your write-up; provide some additional insights in your write-up reflecting on this visualization compared with the CAE-derived plot.

Report: Your report should include a short description of your experiments, along with the plots and discussion paragraphs requested above and any other relevant information to help shed light on your approach and results.

Here is what you need to turn in:

- Your report.
- Readable code.

Report

- Firstly the dataset is loaded and is divided by 255 to normalize as input.

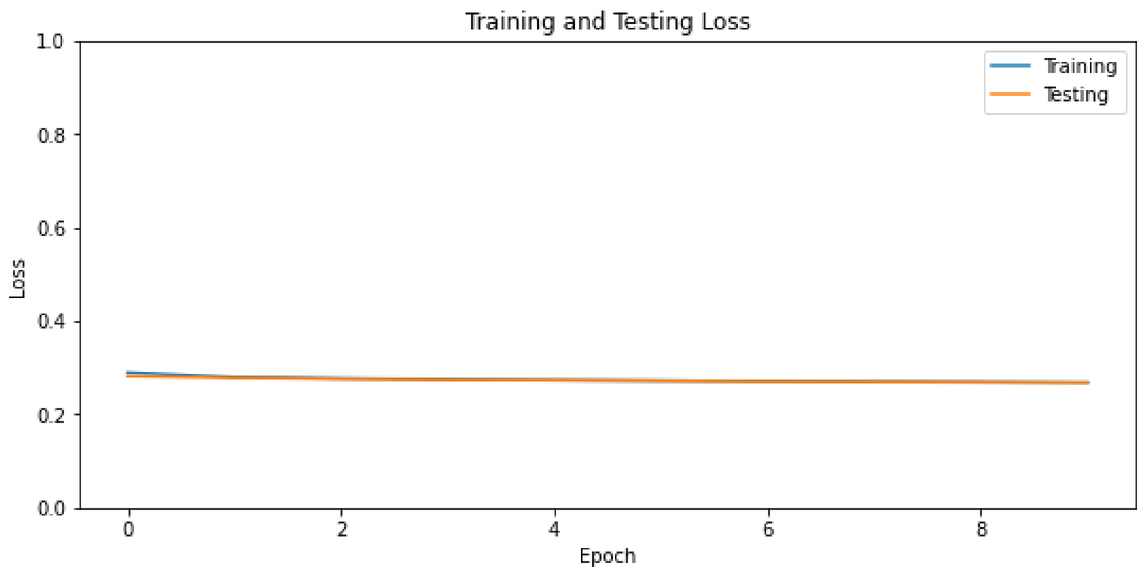
Step - 1

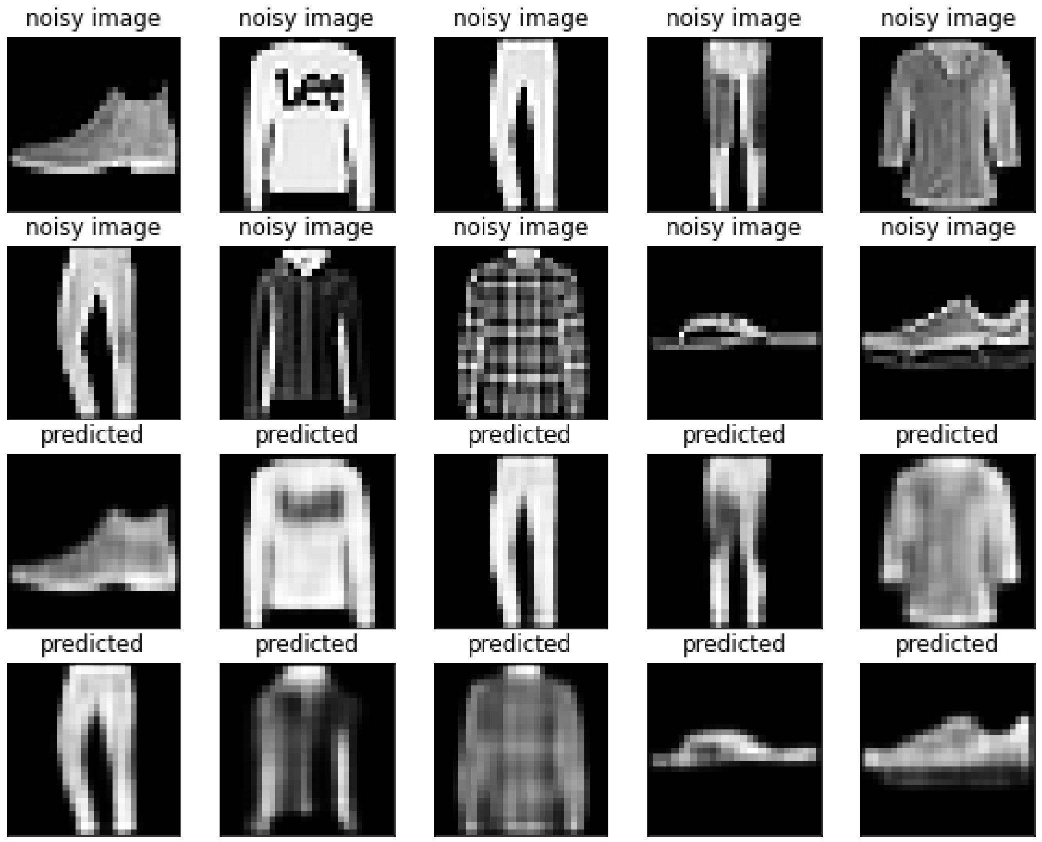
- The input size of MNIST image dataset was 28 x 28 so, the convolutional layers are 28 x 28. After max pooling the image will be reduced by half so, max pooling is 14 x 14. Furthermore, the convolution and reduction using max pooling is done. As per the described architecture, a 32 layer system is maintained which is the example size described in the document. This is a low pixel quality dataset with only 784 input pixels so, the 32 layer system is decided.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 28, 28, 1)]       0

 conv2d_5 (Conv2D)           (None, 28, 28, 32)        320
 max_pooling2d_2 (MaxPooling  (None, 14, 14, 32)       0
 2D)

 conv2d_6 (Conv2D)          (None, 14, 14, 32)       9248

 max_pooling2d_3 (MaxPooling  (None, 7, 7, 32)        0
 2D)

 conv2d_7 (Conv2D)          (None, 7, 7, 32)         9248

 up_sampling2d_2 (UpSampling  (None, 14, 14, 32)       0
 2D)

 conv2d_8 (Conv2D)          (None, 14, 14, 32)       9248

 up_sampling2d_3 (UpSampling  (None, 28, 28, 32)       0
 2D)

 conv2d_9 (Conv2D)          (None, 28, 28, 1)        289

=================================================================
==
Total params: 28,353
Trainable params: 28,353
Non-trainable params: 0
```

Step - 2

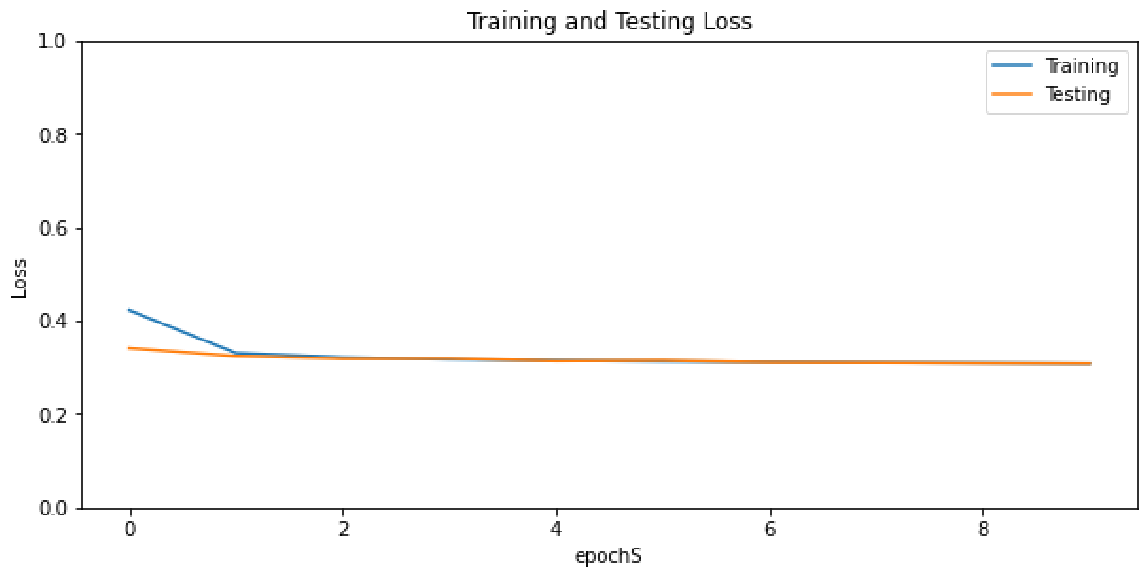
- In this step, we processed prediction data with comparison of training data and trained the model. This created the given graph of loss where loss seems flat because the changes are happening in small portions. Moreover, the test image with their given predicted image is also provided.

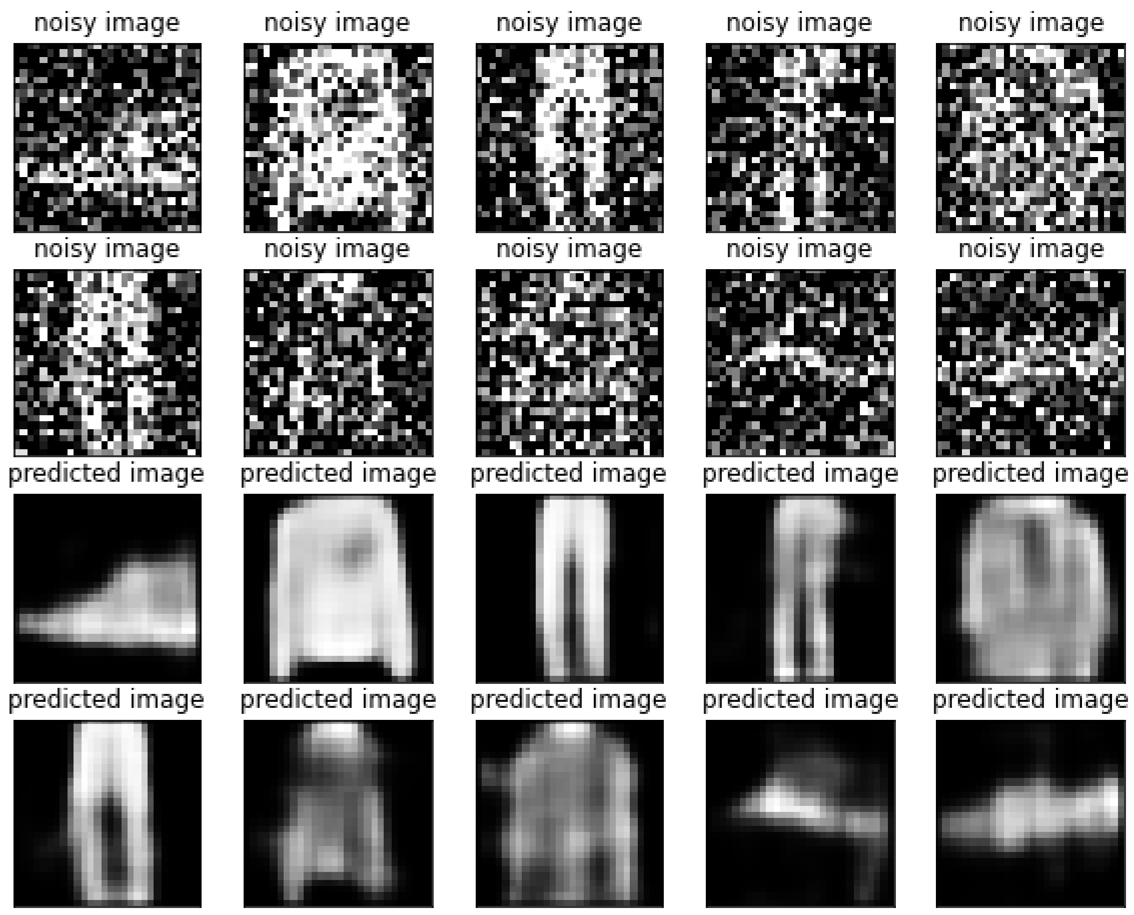




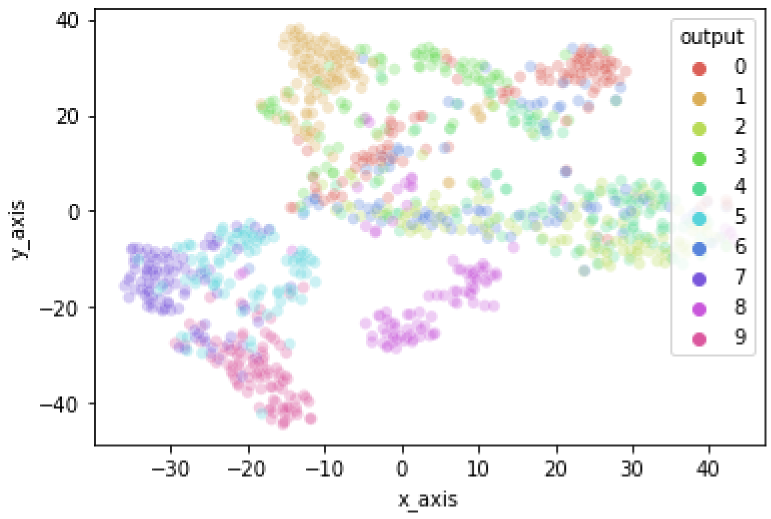
Step - 3

- In this step, we processed data with comparison of noisy data and trained the model. This created the given graph of loss where loss seems flat because the changes are happening in small portions. Moreover, the test image with their given predicted image is also provided.





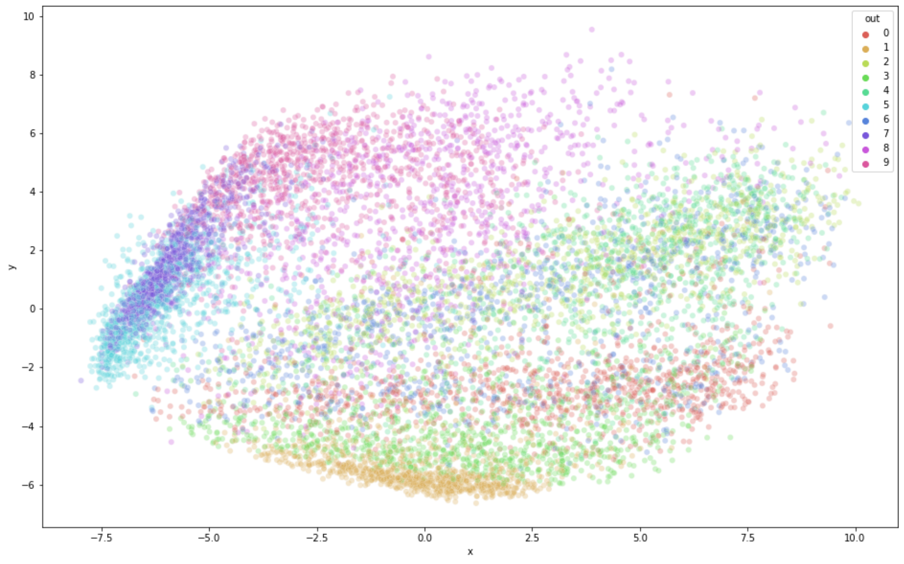
Step - 4



- In this step, predicted data is transformed into two dimensional data so that we can plot it onto the 2D graph. For this process, T sne algorithm is used from Skit Learn. This algorithm takes a lot of time to process, so the first 1000 rows of test images are processed for transformation. Here, 0 to 9 present the categories of the Fashion Mnist Dataset.

['T-Shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Step - 5



- In this step, PCA dimensionality reduction is used on the test images to classify the given 10 categories into 2 dimensional components. this two dimensional data is then represented with the graphs.
- In comparison of the both graphs, we can see that convolutional auto encoder is displaying the predicted images from noisy images distribution with the tsne transformation and PCA is using the test images. In both the images, we can see the degree of separability according to the categories; however, CAE one is more clear and distinctive. CAE is a more regular description of the categories.This graph shows that CAE model is better than PCA decomposition.