Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

Programming 4 Report

Introduction

I worked on exercise 1 and turning in one python code named “ConvAutoEncoder.py”. The dataset used in the Fashion-MNIST dataset. The code outputs the summary for the CAE model, trains the model, and outputs the first 10 images of the trained model’s prediction and respective original test images. Then, the code trains a denoising CAE, outputs the first 10 images of the trained model’s prediction and respective original test images. The last portion of the code extracts latent features from the encoder subnetwork and outputs a graph for randomly chosen 500 images, and then runs a PCA on the same images and outputs a graph. Because I was experimenting with the model in many ways, I ran the code part by part as I needed. The current code is set up to run the encoder latent feature extraction and plotting along with CAE training and testing, but I have the portion of the code needed commented out in case you want to stop the program at step 2 or step 3 and run step 4 at a separate time by loading up saved model. I should have made the code more modular to make this easier, and that’s something I may work on in my spare time.

Results/Discussion

Step 1: Design your CAE

I experimented with batch normalization, transpose convolutional layers, order of upsampling and convolutional layers, and depth and width of convolutional layers. This was the best model:

```
# Build Model
model = models.Sequential()

# Encoder
model.add(tf.keras.Input(shape=(28, 28, 1)))
model.add(layers.Conv2D(112, 3, activation='relu'))  # (None, 26, 26, 112)
model.add(layers.BatchNormalization())  # (None, 26, 26, 112)
model.add(layers.MaxPooling2D(2))  # (None, 13, 13, 112)
model.add(layers.Conv2D(112, 3, activation='relu'))  # (None, 26, 26, 112)
model.add(layers.BatchNormalization())  # (None, 11, 11, 64)
model.add(layers.MaxPooling2D(2))  # (None, 5, 5, 64)
# Bottleneck
model.add(layers.Flatten())  # Output shape (None, 1600)
model.add(layers.Dense(2, activation='relu'))  # (None, 2)
model.add(layers.BatchNormalization())  # (None, 2)
# Decoder
model.add(layers.Dense(1024, activation='relu'))  # (None, 1600)
model.add(layers.BatchNormalization())  # (None, 1600)
model.add(layers.Reshape((4, 4, 64), input_shape=(2, )))  # (None, 4, 4, 64)
model.add(layers.BatchNormalization())  # (None, 4, 4, 64)
model.add(layers.UpSampling2D(3))  # (None, 12, 12, 64)
model.add(layers.Conv2D(112, 3, activation='relu'))  # (None, 26, 26, 112)
model.add(layers.BatchNormalization())
model.add(layers.UpSampling2D(3))  # (None, 30, 30, 64)
model.add(layers.Conv2D(112, 3, activation='relu'))  # (None, 28, 28, 112)
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(1, 1, activation='sigmoid'))  # (None, 28, 28, 1)
```

Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

Here is the output from the *model.summary()*:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 112)       1120

 batch_normalization (BatchN  (None, 26, 26, 112)      448
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 13, 13, 112)      0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 112)       113008

 batch_normalization_1 (Batc  (None, 11, 11, 112)      448
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 112)        0
 2D)

 flatten (Flatten)           (None, 2800)              0

 dense (Dense)               (None, 2)                 5602

 batch_normalization_2 (Batc  (None, 2)                8
 hNormalization)

 dense_1 (Dense)             (None, 1024)              3072

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 reshape (Reshape)           (None, 4, 4, 64)          0

 batch_normalization_4 (Batc  (None, 4, 4, 64)         256
 hNormalization)

 up_sampling2d (UpSampling2D  (None, 12, 12, 64)       0
 )

 conv2d_2 (Conv2D)           (None, 10, 10, 112)       64624

 batch_normalization_5 (Batc  (None, 10, 10, 112)      448
 hNormalization)

 up_sampling2d_1 (UpSampling  (None, 30, 30, 112)      0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 112)       113008

 batch_normalization_6 (Batc  (None, 28, 28, 112)      448
 hNormalization)

 conv2d_4 (Conv2D)           (None, 28, 28, 1)         113

=================================================================
Total params: 306,699
Trainable params: 303,623
Non-trainable params: 3,076
_________________________________________________________________
```

The model is symmetric, as it contains the same number of convolution layers with the same number of filters in encoder and decoder, and same number of maxpooling layers in encoder as upsampling layers in decoder. The last convolutional layer in the decoder portion is symmetric with the input layer since its only job is to make the output's dimension 28x28x1. All the convolutional layers have 112 filters, as it seems to help with lowering the binary cross entropy loss. Batch normalization after each activation is also included to help with the loss. The current model runs slower than I would like, taking about 3 hours to train 100 epochs with batch size of 100 if it was validating test data at every epoch. The model ran faster without any batch normalization and with smaller filters but produced slightly worse outputs with the model

struggling to learn the top and the bottom of the images. I also tried replacing decoder's convolutional layers with transpose convolutional layers, but that also seemed to have a negative effect on learning for my model. Adding one more depth also made the CAE run worse, likely because a lot of information was lost. For the decoder portion of CAE, upsampling layer is used first, followed by a convolutional layer. I initially had convolutional layer followed by upsampling, but this seemed to cause my model to not learn correctly. I think ultimately ordering of the decoder's convolutional and upsampling layers, changing the number of filters, and changing the decoder's dense layer's width helped the most.

Step 2: Training & Testing the Model

I trained my model for 100 epochs with 100 batch size. Optimizer chosen was adam and the loss measured was binary cross-entropy.

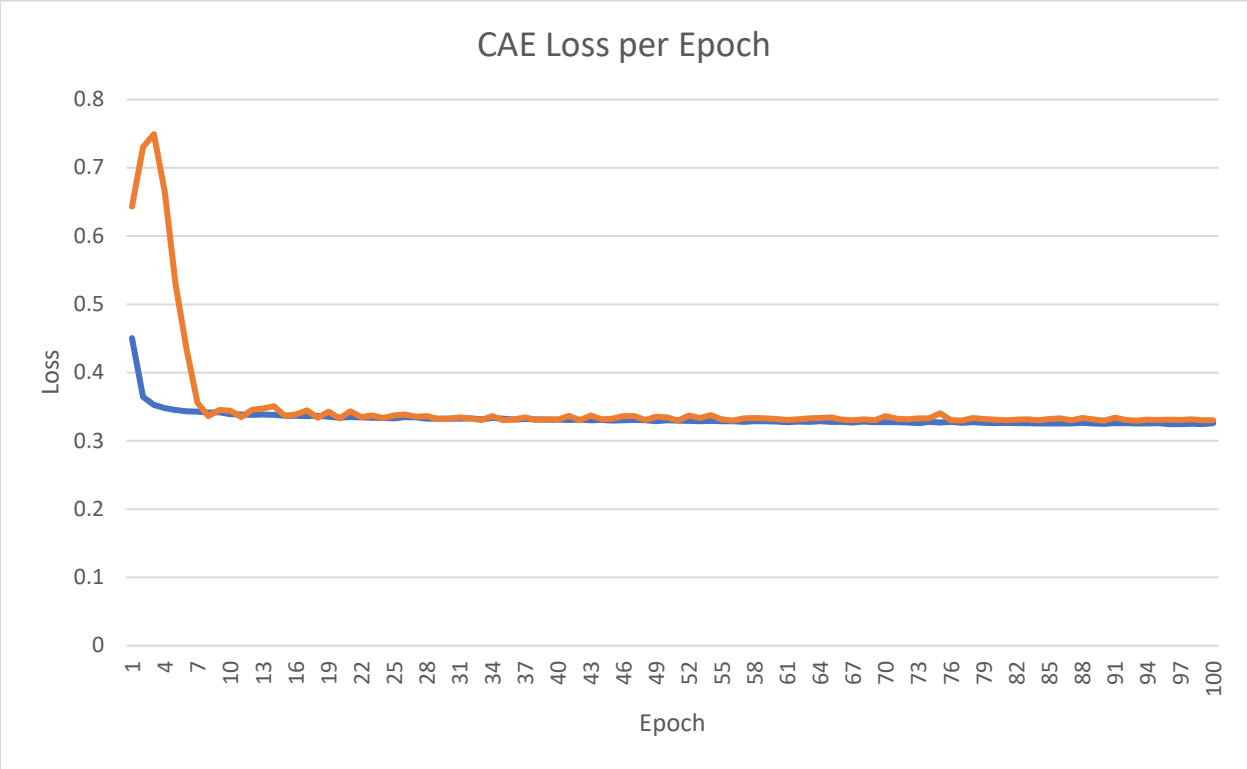
CAE Training/Test Loss per Epoch Result

Epoch	Training Loss	Test Loss	Epoch	Training Loss	Test Loss
1	0.4505	0.6435	**51**	0.3295	0.3296
2	0.3643	0.7304	**52**	0.3294	0.3374
3	0.3527	0.7493	**53**	0.3290	0.3337
4	0.3482	0.6644	**54**	0.3291	0.3379
5	0.3450	0.5278	**55**	0.3290	0.3315
6	0.3434	0.4344	**56**	0.3289	0.3299
7	0.3428	0.3559	**57**	0.3279	0.3329
8	0.3416	0.3365	**58**	0.3290	0.3333
9	0.3420	0.3450	**59**	0.3289	0.3328
10	0.3393	0.3441	**60**	0.3281	0.3321
11	0.3384	0.3348	**61**	0.3276	0.3307
12	0.3382	0.3455	**62**	0.3284	0.3317
13	0.3385	0.3476	**63**	0.3277	0.3328
14	0.3383	0.3510	**64**	0.3286	0.3336
15	0.3369	0.3366	**65**	0.3280	0.3344
16	0.3369	0.3388	**66**	0.3277	0.3311
17	0.3364	0.3449	**67**	0.3268	0.3303
18	0.3368	0.3339	**68**	0.3281	0.3317
19	0.3355	0.3428	**69**	0.3273	0.3303
20	0.3340	0.3329	**70**	0.3273	0.3364
21	0.3351	0.3433	**71**	0.3272	0.3326
22	0.3343	0.3348	**72**	0.3268	0.3315
23	0.3340	0.3374	**73**	0.3262	0.3328
24	0.3335	0.3337	**74**	0.3280	0.3329
25	0.3331	0.3372	**75**	0.3267	0.3407
26	0.3348	0.3388	**76**	0.3277	0.3304
27	0.3344	0.3354	**77**	0.3265	0.3299
28	0.3327	0.3362	**78**	0.3272	0.3336

29	0.3326	0.3322	**79**	0.3263	0.3319
30	0.3325	0.3326	**80**	0.3258	0.3313
31	0.3323	0.3345	**81**	0.3265	0.3304
32	0.3332	0.3323	**82**	0.3261	0.3311
33	0.3314	0.3307	**83**	0.3260	0.3317
34	0.3333	0.3361	**84**	0.3257	0.3303
35	0.3323	0.3302	**85**	0.3254	0.3322
36	0.3311	0.3317	**86**	0.3253	0.3332
37	0.3322	0.3343	**87**	0.3255	0.3301
38	0.3316	0.3307	**88**	0.3265	0.3334
39	0.3309	0.3315	**89**	0.3254	0.3314
40	0.3313	0.3312	**90**	0.3251	0.3296
41	0.3306	0.3369	**91**	0.3261	0.3339
42	0.3311	0.3300	**92**	0.3258	0.3308
43	0.3304	0.3370	**93**	0.3253	0.3296
44	0.3306	0.3316	**94**	0.3253	0.3311
45	0.3298	0.3326	**95**	0.3259	0.3306
46	0.3302	0.3362	**96**	0.3244	0.3310
47	0.3305	0.3361	**97**	0.3248	0.3306
48	0.3301	0.3305	**98**	0.3250	0.3315
49	0.3287	0.3355	**99**	0.3246	0.3303
50	0.3301	0.3345	**100**	0.3259	0.3304

CAE Training/Test Loss per Epoch Graph

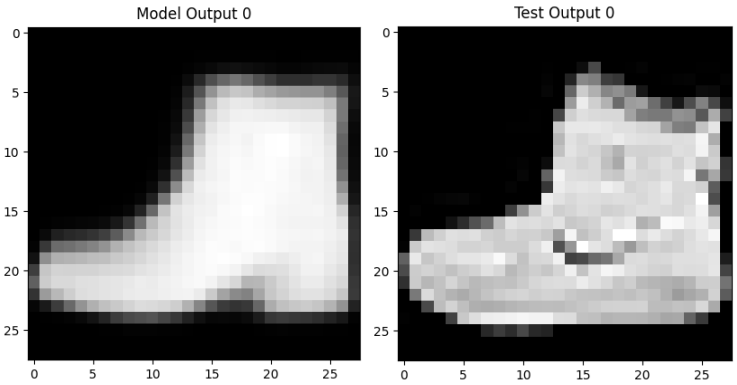
The blue line is the training loss and the red line is the test loss.

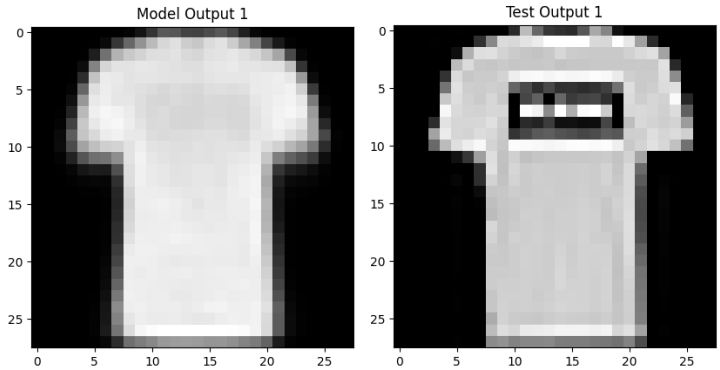


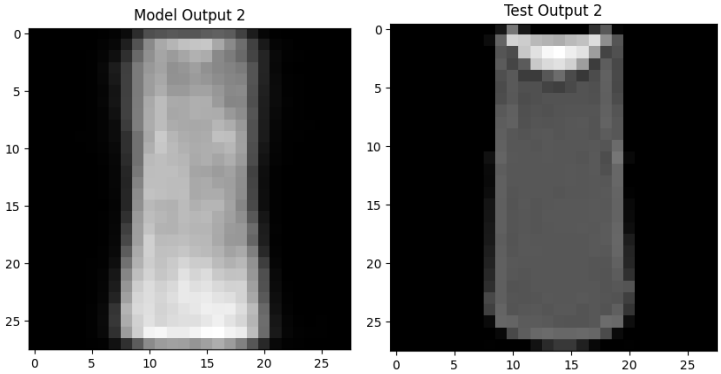
Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

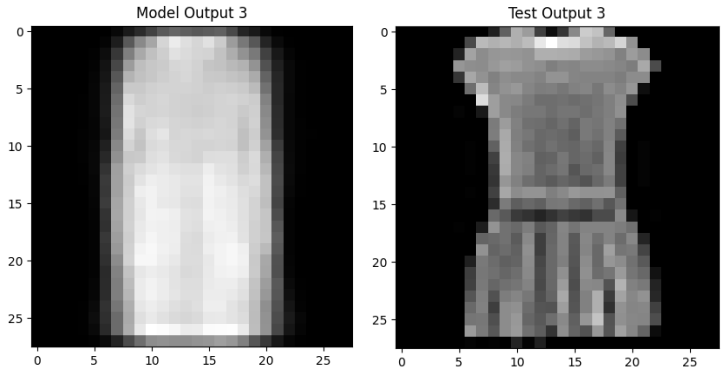
CAE Output and Original Images

I grabbed the first 10 test images and predictions from the trained model for comparison.

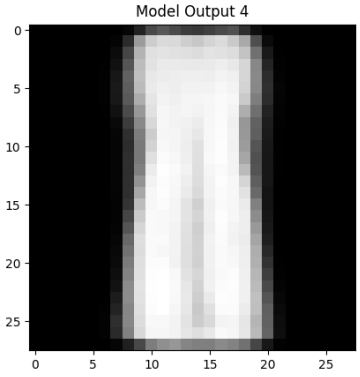


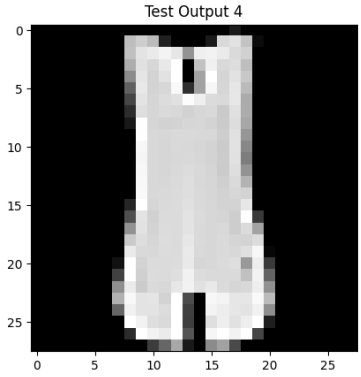


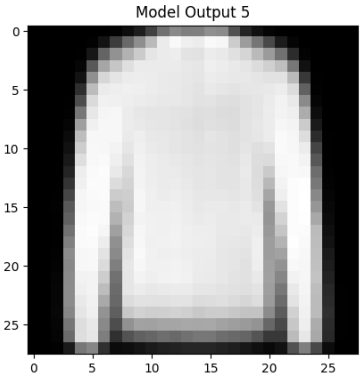


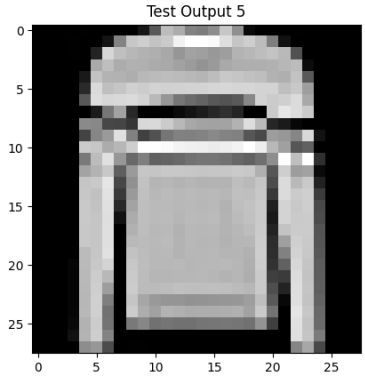


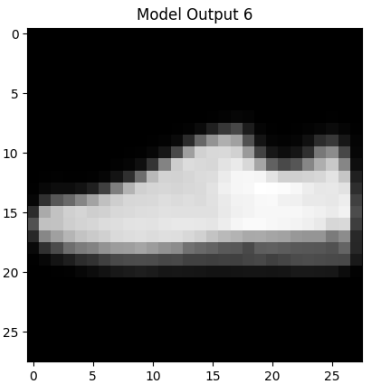
Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

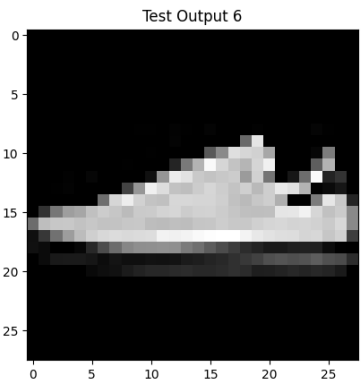


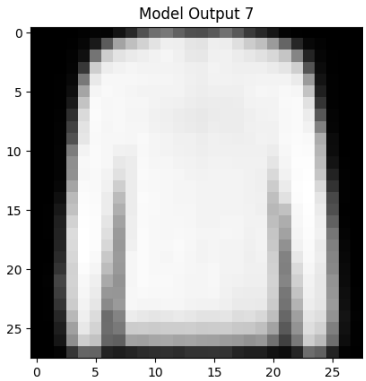


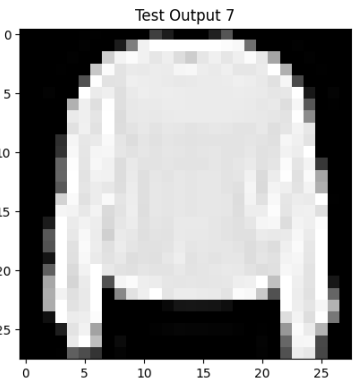




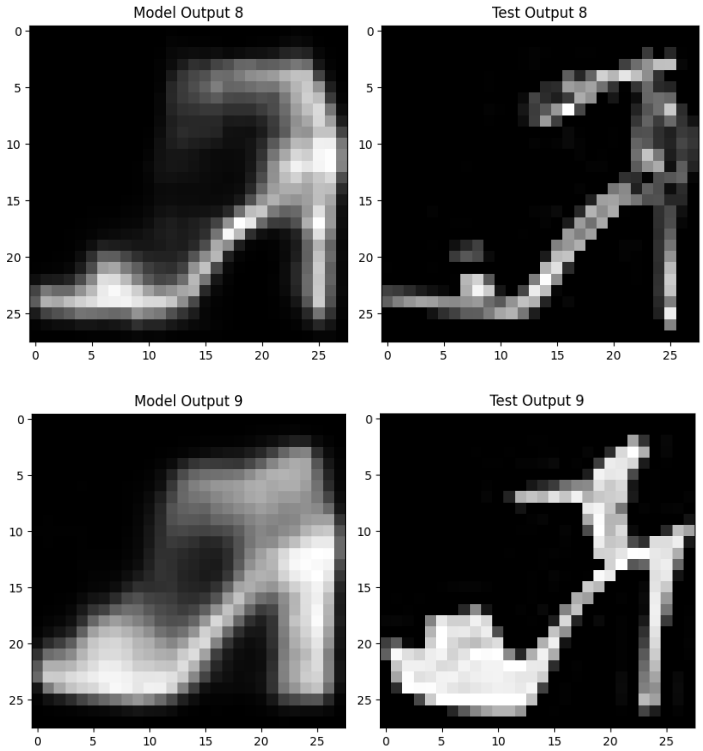








Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023



Step 3: Training a Denoising CAE

I trained my model for 100 epochs with 100 batch size. Optimizer chosen was adam and the loss measured was binary cross-entropy.

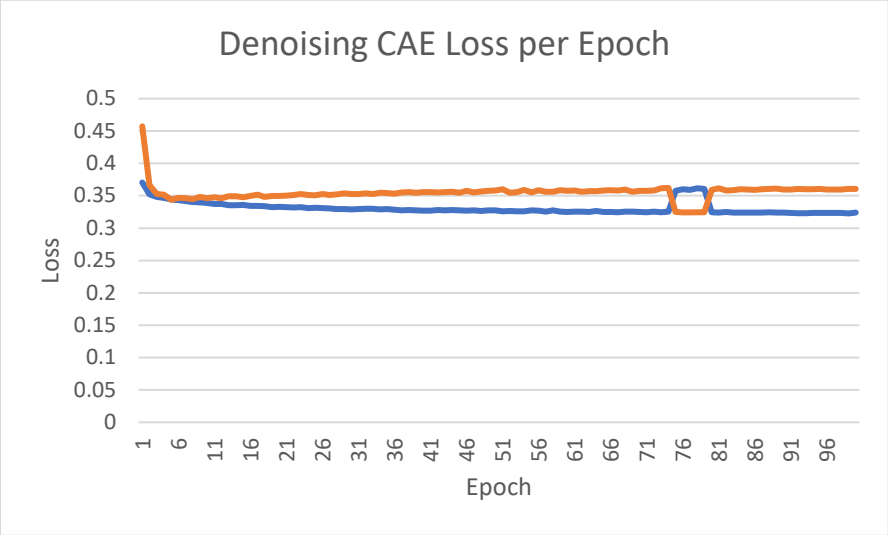
Denoising CAE Training/Test Loss per Epoch Data

Epoch	Training Loss	Test Loss	Epoch	Training Loss	Test Loss
1	0.3705	0.4572	**51**	0.3260	0.3601
2	0.3521	0.3663	**52**	0.3264	0.3546
3	0.3483	0.3526	**53**	0.3260	0.3558
4	0.3468	0.3515	**54**	0.3260	0.3589
5	0.3443	0.3440	**55**	0.3276	0.3550
6	0.3433	0.3468	**56**	0.3271	0.3587
7	0.3416	0.3461	**57**	0.3257	0.3563
8	0.3402	0.3448	**58**	0.3273	0.3560
9	0.3398	0.3482	**59**	0.3257	0.3588
10	0.3389	0.3462	**60**	0.3252	0.3575
11	0.3375	0.3477	**61**	0.3253	0.3580
12	0.3374	0.3464	**62**	0.3253	0.3559
13	0.3356	0.3493	**63**	0.3252	0.3572
14	0.3354	0.3491	**64**	0.3267	0.3571
15	0.3358	0.3477	**65**	0.3248	0.3581
16	0.3345	0.3499	**66**	0.3249	0.3584
17	0.3342	0.3518	**67**	0.3247	0.3580
18	0.3341	0.3480	**68**	0.3255	0.3597
19	0.3323	0.3497	**69**	0.3254	0.3563

20	0.3327	0.3496	**70**	0.3249	0.3577
21	0.3324	0.3503	**71**	0.3246	0.3578
22	0.3319	0.3513	**72**	0.3256	0.3580
23	0.3324	0.3529	**73**	0.3244	0.3616
24	0.3309	0.3513	**74**	0.3255	0.3622
25	0.3313	0.3509	**75**	0.3575	0.3250
26	0.3308	0.3525	**76**	0.3601	0.3243
27	0.3306	0.3514	**77**	0.3590	0.3243
28	0.3296	0.3521	**78**	0.3615	0.3245
29	0.3295	0.3538	**79**	0.3605	0.3244
30	0.3290	0.3529	**80**	0.3243	0.3590
31	0.3295	0.3528	**81**	0.3239	0.3617
32	0.3298	0.3538	**82**	0.3248	0.3582
33	0.3300	0.3528	**83**	0.3241	0.3584
34	0.3287	0.3546	**84**	0.3239	0.3602
35	0.3292	0.3542	**85**	0.3240	0.3596
36	0.3283	0.3532	**86**	0.3240	0.3592
37	0.3277	0.3553	**87**	0.3238	0.3601
38	0.3278	0.3557	**88**	0.3243	0.3604
39	0.3273	0.3547	**89**	0.3238	0.3611
40	0.3272	0.3557	**90**	0.3242	0.3598
41	0.3271	0.3555	**91**	0.3237	0.3596
42	0.3279	0.3552	**92**	0.3232	0.3604
43	0.3276	0.3557	**93**	0.3231	0.3603
44	0.3279	0.3563	**94**	0.3237	0.3602
45	0.3274	0.3545	**95**	0.3236	0.3607
46	0.3272	0.3576	**96**	0.3234	0.3598
47	0.3273	0.3552	**97**	0.3233	0.3594
48	0.3267	0.3565	**98**	0.3235	0.3596
49	0.3273	0.3574	**99**	0.3227	0.3604
50	0.3273	0.3581	**100**	0.3239	0.3605

Denoising CAE Training/Test Loss per Epoch Graph

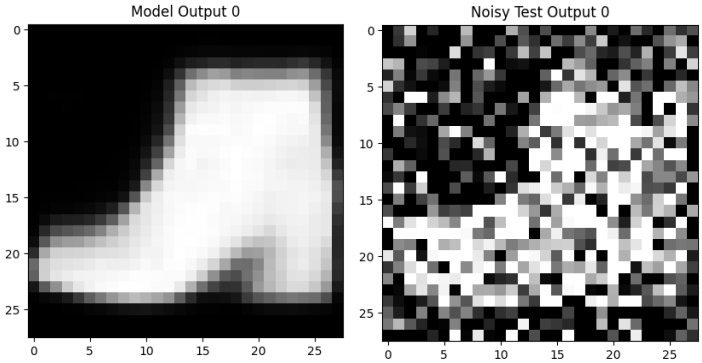
The blue line is the training loss and the red line is the test loss.

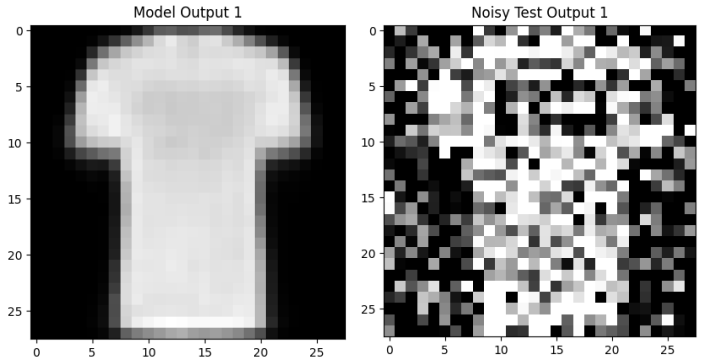


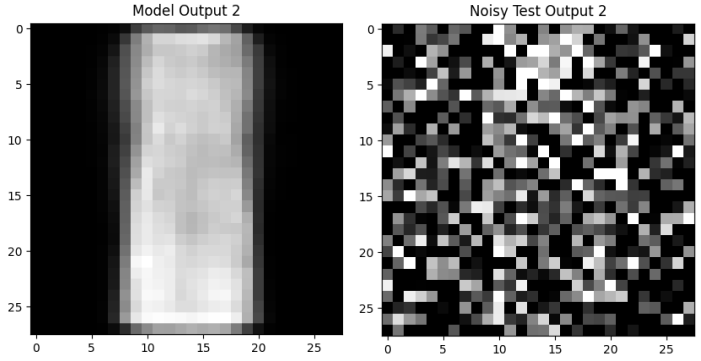
Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

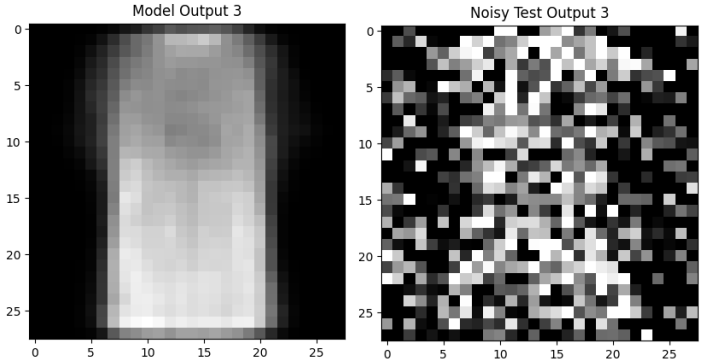
Denoising CAE Output and Original Images

I grabbed the first 10 images for comparison.

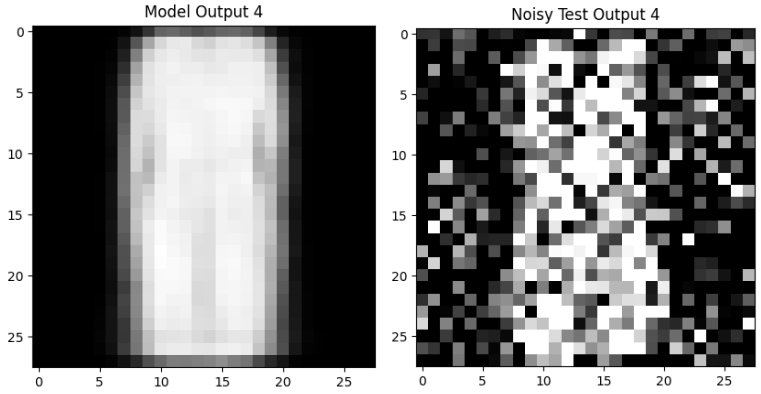


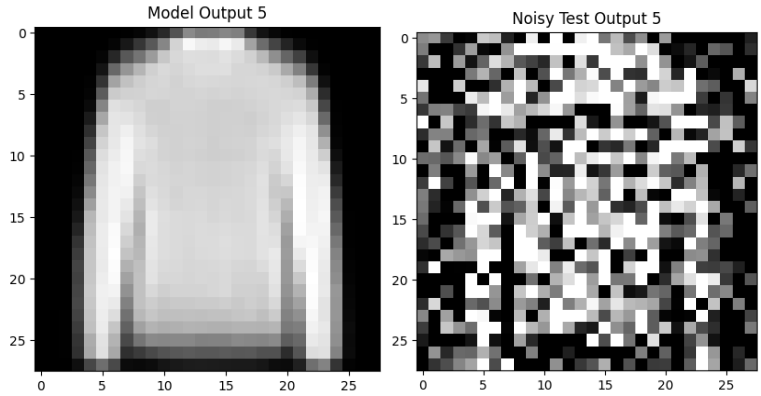


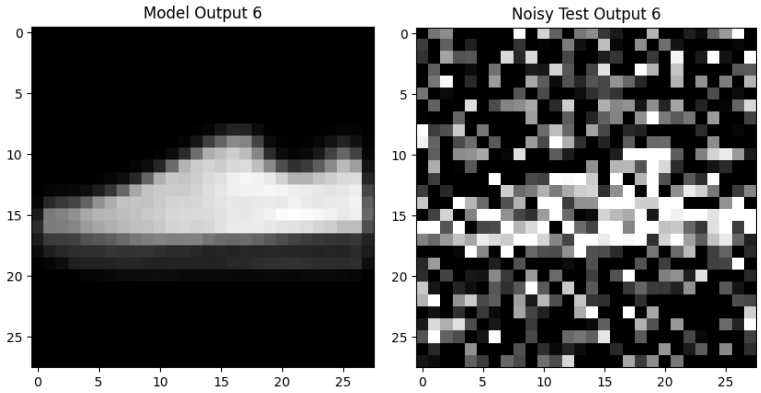


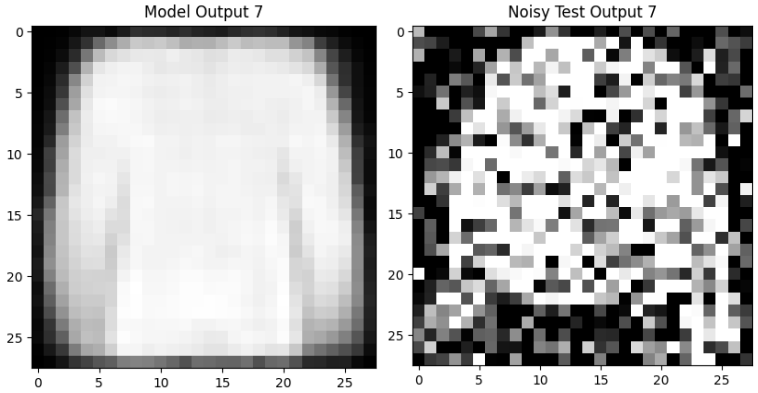


Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

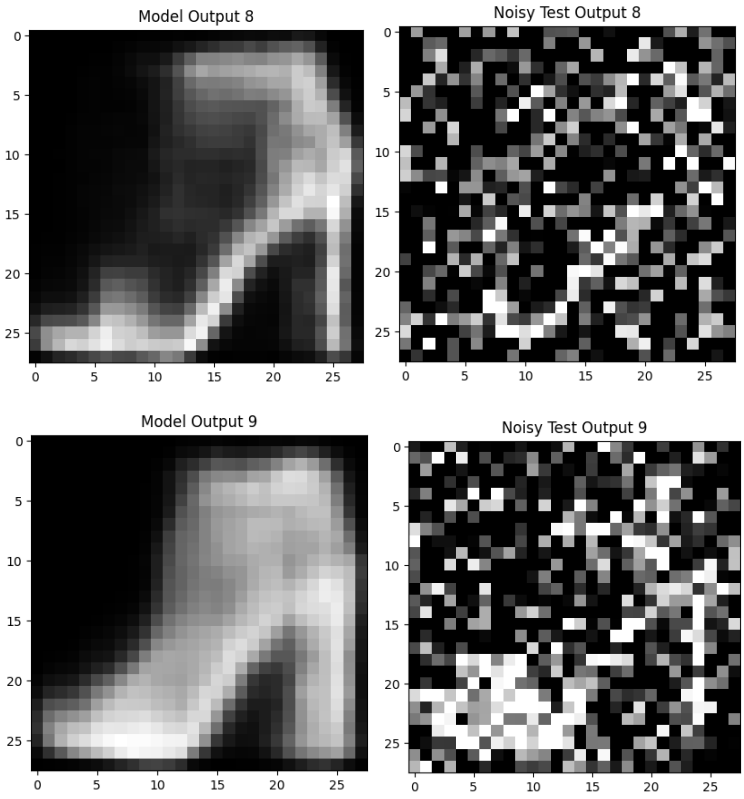








Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

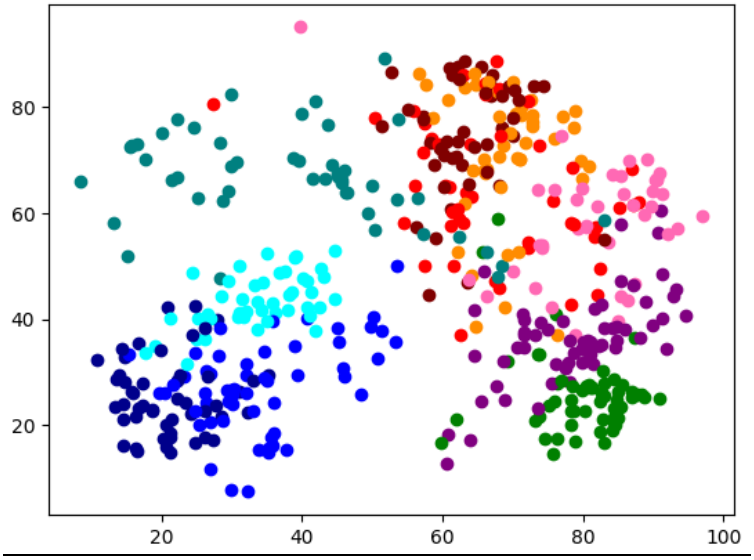


Step 4: Latent Feature Extraction

I selected 500 random images to plot for both CAE and PCA.

CAE

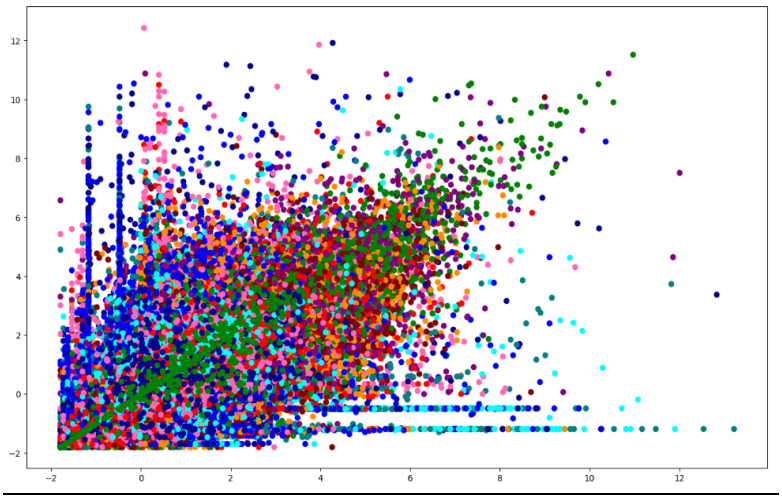
I extracted the latent features after the dense layer in the encoder/bottleneck.



Label Number	Label Name	Color
0	Top	Hot Pink
1	Trouser	Green
2	Pullover	Maroon
3	Dress	Purple
4	Coat	Dark Orange
5	Sandal	Blue
6	Shirt	Red
7	Sneaker	Dark Blue
8	Bag	Teal
9	Ankle Boot	Cyan

Cera Oh
CS510 CV and DL
Summer 2023
8/10/2023

The latent features groups each label into mostly one area, with related articles of clothing such as ankle boot, sneaker, and sandal being closer together than different articles, such as a top or a bag. The graph shows that the encoder has correctly extracted latent features that are useful in identifying the article of clothing depicted in the images. I also tried extracting the latent features after the last convolutional layer in the encoder, so there were 4x4x64 points to plot per image.

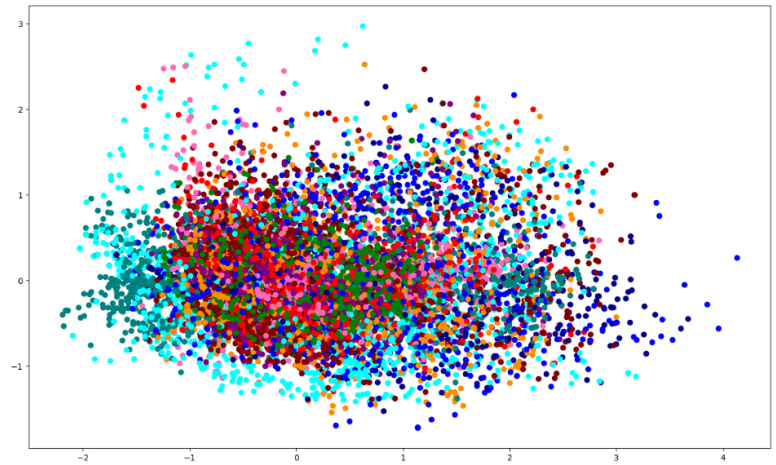


Label Number	Label Name	Color
0	Top	Hot Pink
1	Trouser	Green
2	Pullover	Maroon
3	Dress	Purple
4	Coat	Dark Orange
5	Sandal	Blue
6	Shirt	Red
7	Sneaker	Dark Blue
8	Bag	Teal
9	Ankle Boot	Cyan

There seems to be concentrated zones for each article of clothing, but with a lot of overlap, and it isn't as clear the separation between the articles of clothing as was the feature extraction from the dense layer. However, this looks more similar to what I have for PCA below.

PCA

I plotted all the principal components returned by the PCA.



Label Number	Label Name	Color
0	Top	Hot Pink
1	Trouser	Green
2	Pullover	Maroon
3	Dress	Purple
4	Coat	Dark Orange
5	Sandal	Blue
6	Shirt	Red
7	Sneaker	Dark Blue
8	Bag	Teal
9	Ankle Boot	Cyan

The PCA plot shows some concentration of colors in areas for articles of clothing, but a lot of the points are dispersed and mixed together, similar to what the latent feature graph for before the dense layer for CAE looked like. The dense layer feature extraction for CAE showed class division more clearly. However, you can still tell that some articles of clothing's features are

more closely related than another in both PCA and CAE layers. For example, tops and shirts are more closely concentrated near each other than the dots for ankle boot. Therefore, there are distinct features that are extracted and used to differentiate between different articles of clothing, and the model and PCA are more likely to confuse between closely related images, such as sneakers and ankle boot than between more unrelated images, such as ankle boot and dress.

Conclusion

The latent features extracted from dense layer of the CAE encoder is good at extracting features for each class, and graph from it is easier for seeing the relationship between classes than the graph from PCA. Denoising CAE does a good job reconstructing images even when random Gaussian noises are added to the images, though test loss was worse in general than training loss for denoising CAE than CAE. Test loss and training loss ended up being similar for CAE, but test loss had more oscillation. It was interesting to work with the model to try to improve the loss. Because ultimately there is information loss at the bottleneck, the reconstructed images had ~0.33 loss and was never exactly like the original images.