Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

Programming 5 Report

Introduction

I worked on exercise 2 and turning in two python codes named "GAN.py" and "ConvGAN.py". The dataset used is the Fashion-MNIST dataset. The "GAN.py" code outputs the summary for the generator, the discriminator, and the combined GAN model for step 1, trains the models by first training the discriminator with unfrozen weights for an epoch and then training the GAN model's generator with the discriminator's weights frozen for an epoch. Both are trained 100 epochs each. The code also saves the first 3 images from the GAN model's generator per 10 epochs and the last epoch 99. The "ConvGAN.py" code outputs the summary for the CNN generator, the CNN discriminator, and the combined CNN GAN model for step 3, trains the models by first training the discriminator with unfrozen weights for an epoch and then training the GAN model's generator with the discriminator's weights frozen for an epoch. Both are trained 100 epochs each. The code saves the first 3 images from the step 3 GAN model's generator per 10 epochs and the last epoch 99.

Results

Step 1: Design the GAN

I experimented with my GAN model's layer orders, the number of dense layers, width of the dense layers, and settled on what's shown below, though my GAN model does not seem to produce a fake image that looks like it could be one of the real images, it does come close. I also experimented with batch size but didn't see too much improvement, so I stuck with 100. I think my generator is weaker than the discriminator, but I could not figure out how to balance them well. I used batch normalization between dense layers and leaky RELU as suggested.

The Generator

```
# Generator
gen_model = models.Sequential()
gen_model.add(tf.keras.Input(shape=(100,)))
gen_model.add(layers.Dense(256))
gen_model.add(layers.LeakyReLU())
gen_model.add(layers.BatchNormalization())
gen_model.add(layers.Dense(512))
gen_model.add(layers.LeakyReLU())
gen_model.add(layers.BatchNormalization())
gen_model.add(layers.Dense(1024))
gen_model.add(layers.LeakyReLU())
gen_model.add(layers.BatchNormalization())
gen_model.add(layers.Dense(784))  # (None, 784)
gen_model.add(layers.LeakyReLU())  # (None, 28, 28, 1)
gen_model.add(layers.Reshape((28, 28, 1), input_shape=(2, )))
```

Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               25856

 leaky_re_lu (LeakyReLU)     (None, 256)               0

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dense_1 (Dense)             (None, 512)               131584

 leaky_re_lu_1 (LeakyReLU)   (None, 512)               0

 batch_normalization_1 (Batc  (None, 512)              2048
 hNormalization)

 dense_2 (Dense)             (None, 1024)              525312

 leaky_re_lu_2 (LeakyReLU)   (None, 1024)              0

 batch_normalization_2 (Batc  (None, 1024)             4096
 hNormalization)

 dense_3 (Dense)             (None, 784)               803600

 leaky_re_lu_3 (LeakyReLU)   (None, 784)               0

 reshape (Reshape)           (None, 28, 28, 1)         0

=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
```

The Discriminator

```python
# Discriminator
disc_model = models.Sequential()
disc_model.add(tf.keras.Input(shape=(28, 28, 1)))
disc_model.add(layers.Flatten())  # (None, 784)
disc_model.add(layers.Dense(1024))
disc_model.add(layers.LeakyReLU())
disc_model.add(layers.BatchNormalization())
disc_model.add(layers.Dense(512))
disc_model.add(layers.LeakyReLU())
disc_model.add(layers.BatchNormalization())
disc_model.add(layers.Dense(256))
disc_model.add(layers.LeakyReLU())
disc_model.add(layers.BatchNormalization())
disc_model.add(layers.Dense(1, activation='sigmoid'))  # (None, 1)
```

Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense_4 (Dense)             (None, 1024)              803840

 leaky_re_lu_4 (LeakyReLU)   (None, 1024)              0

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 dense_5 (Dense)             (None, 512)               524800

 leaky_re_lu_5 (LeakyReLU)   (None, 512)               0

 batch_normalization_4 (Batc  (None, 512)              2048
 hNormalization)

 dense_6 (Dense)             (None, 256)               131328

 leaky_re_lu_6 (LeakyReLU)   (None, 256)               0

 batch_normalization_5 (Batc  (None, 256)              1024
 hNormalization)

 dense_7 (Dense)             (None, 1)                 257

=================================================================
Total params: 1,467,393
Trainable params: 1,463,809
Non-trainable params: 3,584
_________________________________________________________________
```

<u>The GAN Model</u>

```
# Full GAN
disc_model.trainable = False
GAN_model = models.Sequential([tf.keras.layers.Input(
    shape=(100,)), gen_model, disc_model])
GAN_model.build(input_shape=(100,))
GAN_model.summary()
```

```
_________________________________________________________________
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 28, 28, 1)         1493520

 sequential_1 (Sequential)   (None, 1)                 1467393

=================================================================
Total params: 2,960,913
Trainable params: 1,489,936
Non-trainable params: 1,470,977
```

Step 2: Training GAN model and Discriminator model

Data was preprocessed by multiplying the image values by 1/255. I trained my model for 100 epochs with 100 batch size. Optimizer was adam and the loss measured was binary cross-entropy. I coded the training loop following the guide in https://keras.io/guides/writing_a_training_loop_from_scratch/

Training Loss per Epoch

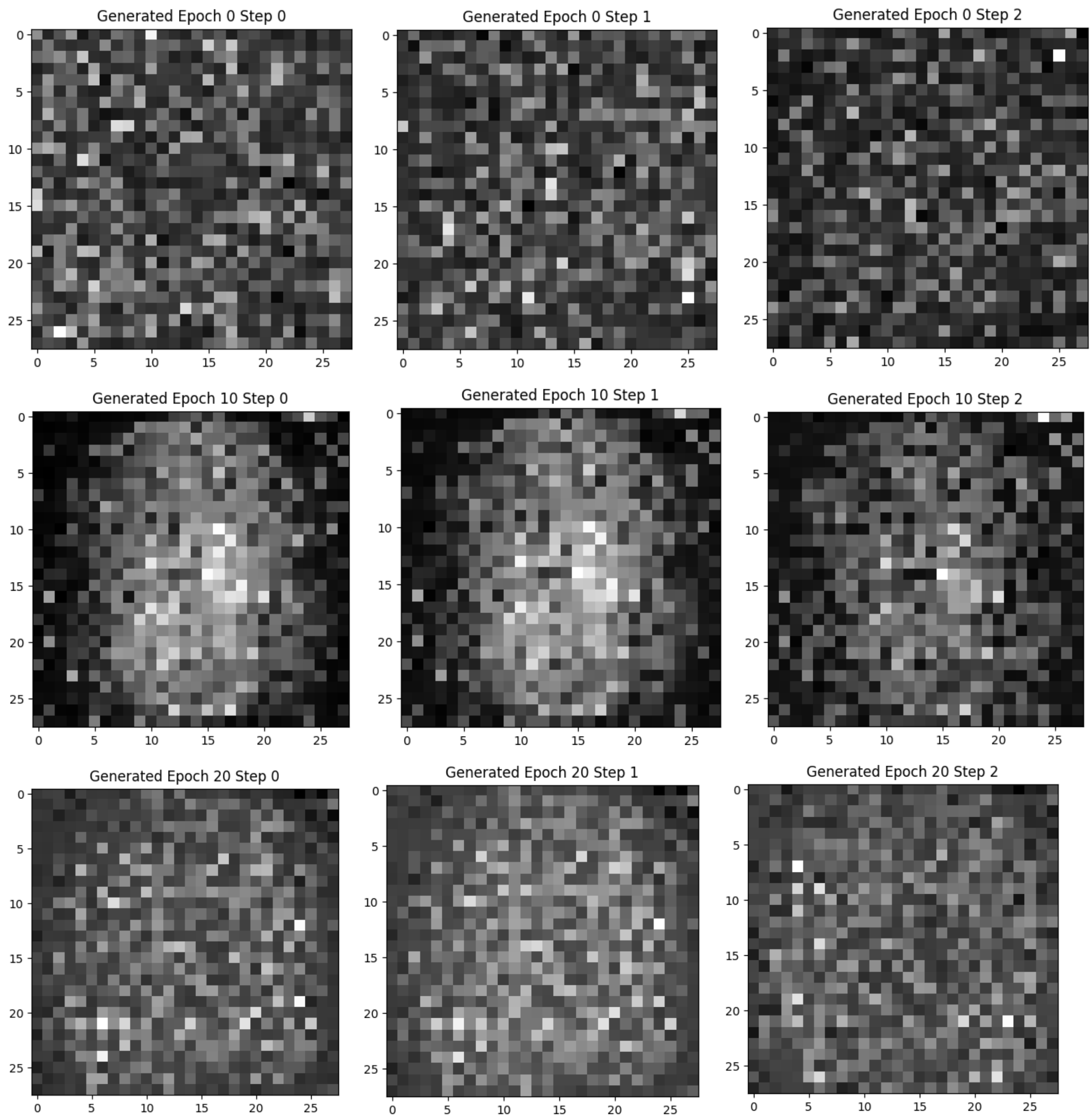
Epoch	Discriminator Training Loss	GAN Training Loss	Epoch	Discriminator Training Loss	GAN Training Loss
0	3.107323565121334e-10	1.8828283487344882e-20	**50**	0.0584297813475132	0.0009730348247103393
1	6.628647042816738e-06	0.0	**51**	0.0027646152302622795	0.0021094121038913727
2	1.7890829334503422e-13	0.0	**52**	0.0011166115291416645	0.0005461795371957123
3	1.2568125779005338e-25	0.0	**53**	0.002532252110540867	9.268229769077152e-05
4	1.181110646396125e-19	0.0	**54**	0.0027956985868513584	0.00019525884999893606
5	1.5007686689514291e-21	0.0	**55**	0.00580389192327857	0.0002568997151684016
6	1.313190178819923e-08	0.0	**56**	0.0020593523513525724	1.0518755516386591e-05
7	6.885981774730923e-38	0.0	**57**	0.0004420839250087738	0.00017423676035832614
8	3.319027896964144e-08	0.0	**58**	9.997850611398462e-06	0.00038750431849621236
9	0.00014608855417463928	0.0	**59**	4.398263627081178e-06	2.378181918061273e-08
10	1.76285345787619e-06	1.3902207612991333	**60**	9.690074512036517e-05	8.186914055841044e-05
11	2.6132571292691864e-05	6.445011635491937e-09	**61**	4.909489143756218e-05	2.6243464162689634e-05
12	6.386033055605367e-05	0.0001176200748886913	**62**	0.0003471345407888293	0.0007622029515914619
13	0.00033392079058103263	0.3596891164779663	**63**	0.0012816556263715029	0.000492788094561547
14	0.0013279863633215427	0.2481156885623932	**64**	0.0030268938280642033	0.0005286442465148866
15	0.019973058253526688	0.040013059973716736	**65**	0.047363437712192535	3.6267261748434976e-05
16	0.05885313078761101	0.4276794195175171	**66**	1.629531107028015e-05	5.95403544139117e-05
17	0.01747729629278183	0.1900664120912552	**67**	1.4044260296941502e-06	3.817094210156746e-12
18	0.0020217604469507933	0.047411952167749405	**68**	1.0396170324611376e-07	4.421699117962419e-11
19	0.009358644485473633	0.14184927940368652	**69**	2.8612439564312808e-05	2.594969373603817e-06
20	0.0202618557959795	0.0028923461213707924	**70**	3.4176098324678605e-06	8.584412825030086e-08

21	0.02353852055966854	0.034965310245752335	**71**	0.00013184588169679046	3.396354486540076e-08
22	0.025270387530326843	0.021904923021793365	**72**	7.323890258703614e-06	1.7422651581000537e-05
23	0.007923358120024204	0.11100969463586807	**73**	0.00394035829231143	0.0014235982671380043
24	0.05595920607447624	0.21468229591846466	**74**	0.0001477643963880837	0.0019803501199930906
25	0.025974364951252937	0.004148258361965418	**75**	0.0038281597662717104	6.568321259692311e-05
26	0.012430590577423573	0.007364942226558924	**76**	0.0394989512860775	0.010636317543685436
27	0.019361548125743866	0.009204610250890255	**77**	0.002626393921673298	0.005469164811074734
28	0.010745820589363575	0.12214437127113342	**78**	0.015969334170222282	0.001841038465499878
29	0.022217316552996635	0.04506676644086838	**79**	0.04545379802584648	0.006780379917472601
30	0.010732552967965603	0.01830469816923415	**80**	0.004218677990138531	0.016124464571475983
31	0.037731196731328964	0.028597110882401466	**81**	0.007464053574949503	0.006814803462475538
32	0.022078871726989746	0.019603684544563293	**82**	0.016049819067120552	0.007481928449124098
33	0.037839341908693314	0.02506358176469803	**83**	0.006777258589863777	0.0041136485524475574
34	0.018960848450660706	0.027037739753723145	**84**	0.005471150856465101	0.0003059450536966324
35	0.018887551501393318	0.021262560039758682	**85**	0.004278677515685558	0.004491932690143585
36	0.013619081117212772	0.009250054135918617	**86**	0.011253374628722668	0.0021203511860221624
37	0.013600916601717472	0.017269015312194824	**87**	0.006726287771016359	0.011302080005407333
38	0.013729634694755077	0.012675553560256958	**88**	0.003075928660109639	0.006646629422903061
39	0.013807655312120914	0.0019850856624543667	**89**	0.03071611188352108	0.0012072158278897405
40	0.010182742960751057	0.013674041256308556	**90**	0.003150765784084797	0.006561307702213526
41	0.007732328958809376	0.004690696485340595	**91**	0.017119912430644035	0.00942391436547041
42	0.006054171826690435	0.005900361575186253	**92**	0.003734018886461854	0.0020642979070544243
43	0.006444049999117851	0.0023717170115560293	**93**	0.006089173723012209	0.0013895189622417092
44	0.0034913208801299334	0.024584012106060982	**94**	0.01940811611711979	0.007762295193970203
45	0.005997875705361366	0.009161565452814102	**95**	0.002697514835745096	0.0040717110969126225

46	0.003262854181230068	0.0026606740429997444	**96**	0.009124013595283031	0.008223935961723328
47	0.0026590460911393166	0.0012421151623129845	**97**	0.021852925419807434	0.00833314098417759
48	0.007605062332004309	0.002356533892452717	**98**	0.035771340131759644	0.011003077030181885
49	0.005776549689471722	0.0004685789463110268	**99**	0.0167892724275589	0.003909039776772261

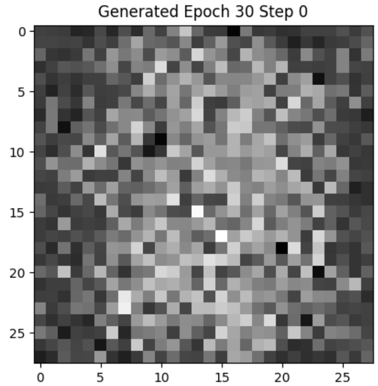
<u>Synthetic Images</u>

I grabbed the first 3 synthetic images every 10th epoch right before training:

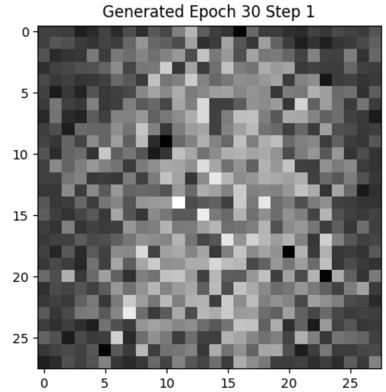


Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

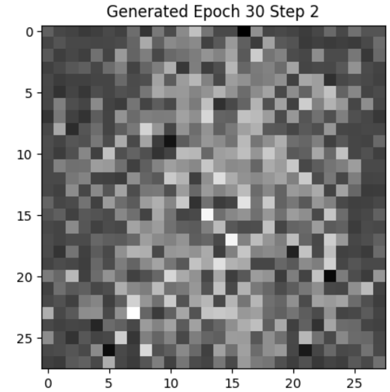
Generated Epoch 30 Step 0



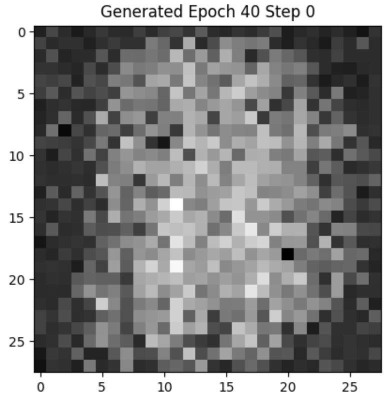
Generated Epoch 30 Step 1



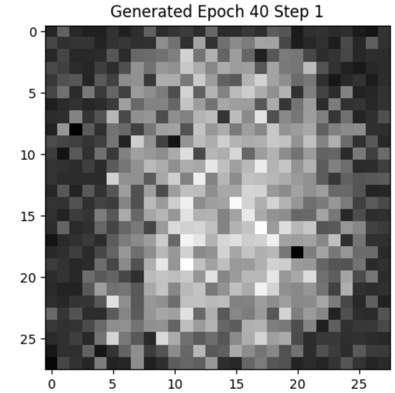
Generated Epoch 30 Step 2



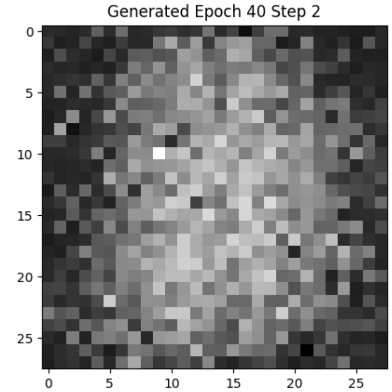
Generated Epoch 40 Step 0



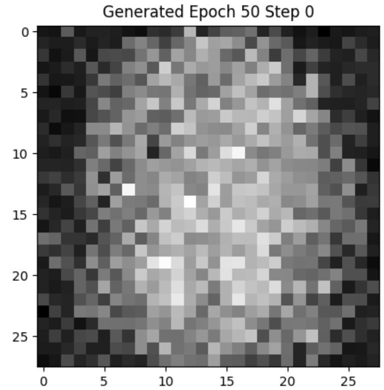
Generated Epoch 40 Step 1



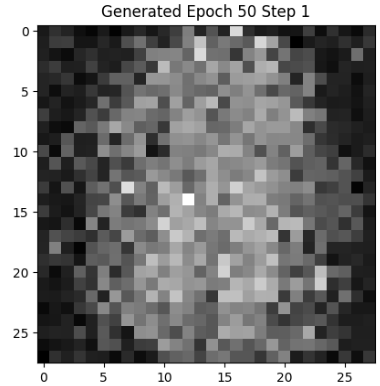
Generated Epoch 40 Step 2



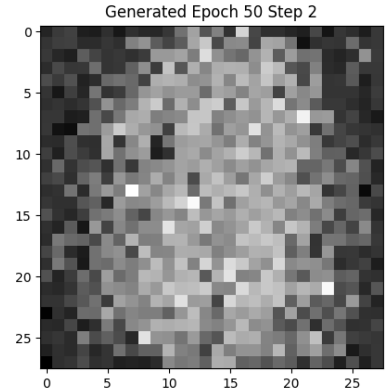
Generated Epoch 50 Step 0



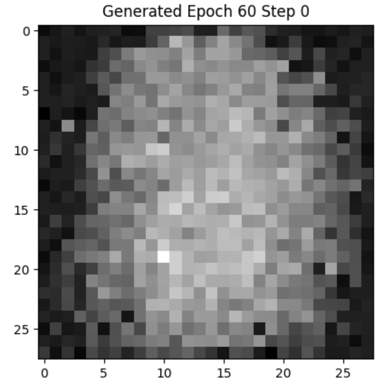
Generated Epoch 50 Step 1



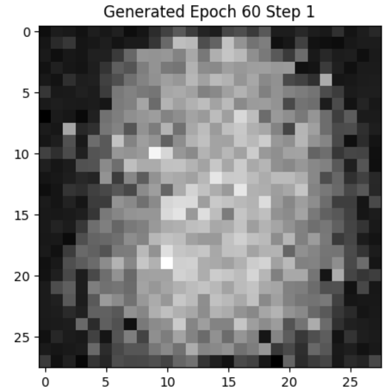
Generated Epoch 50 Step 2



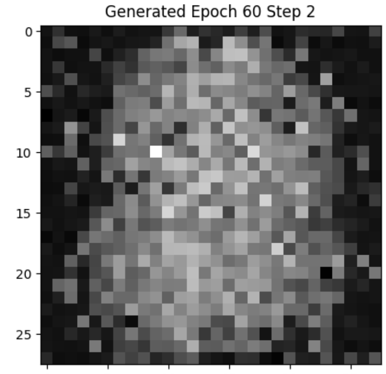
Generated Epoch 60 Step 0



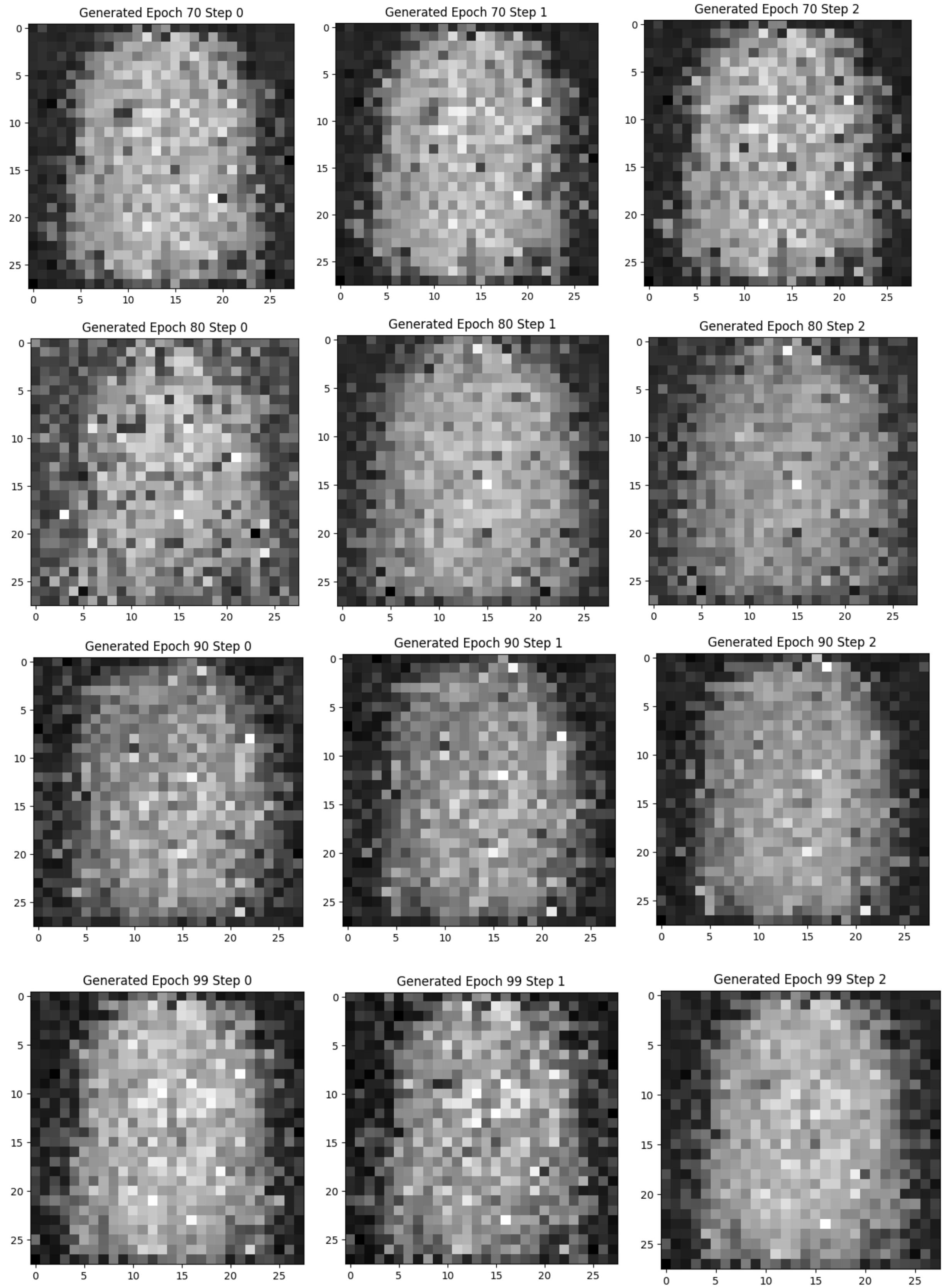
Generated Epoch 60 Step 1



Generated Epoch 60 Step 2



Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023



Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

Step 3: Conv GAN

I experimented with layer depth and width as well for CNN GAN model's generator and discriminator. I used batch normalization and Leaky RELU. I tried RELU and it did not work well. I tried to do stride 2 and filter size 5 as suggested for all the layers but had a hard time making the generator produce a 28x28x1 output that way, so I changed to filter size 2 with stride 2. None of what I tried made my CNN GAN model as good as the GAN model from step 1 and the below is the best model I had from the experiments:

<u>The CNN Generator</u>

```
cnn_gen_model = models.Sequential()
cnn_gen_model.add(tf.keras.Input(shape=(6272,)))
cnn_gen_model.add(layers.Reshape(
    (7, 7, 128), input_shape=(6272, )))  # 28 * 28 * 8
cnn_gen_model.add(layers.Conv2DTranspose(56, 2, strides=2))
cnn_gen_model.add(layers.LeakyReLU())
cnn_gen_model.add(layers.BatchNormalization())  # (None, 14, 14, 56)
cnn_gen_model.add(layers.Conv2DTranspose(112, 2, strides=2))
cnn_gen_model.add(layers.LeakyReLU())
cnn_gen_model.add(layers.BatchNormalization())  # (None, 28, 28, 112)
cnn_gen_model.add(layers.Conv2D(1, 1))  # (None, 28, 28, 1)
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 reshape (Reshape)           (None, 7, 7, 128)         0

 conv2d_transpose (Conv2DTra  (None, 14, 14, 56)       28728
 nspose)

 leaky_re_lu (LeakyReLU)     (None, 14, 14, 56)        0

 batch_normalization (BatchN  (None, 14, 14, 56)       224
 ormalization)

 conv2d_transpose_1 (Conv2DT  (None, 28, 28, 112)      25200
 ranspose)

 leaky_re_lu_1 (LeakyReLU)   (None, 28, 28, 112)       0

 batch_normalization_1 (Batc  (None, 28, 28, 112)      448
 hNormalization)

 conv2d (Conv2D)             (None, 28, 28, 1)         113

=================================================================
Total params: 54,713
Trainable params: 54,377
Non-trainable params: 336
```

<u>The CNN Discriminator</u>

```
cnn_disc_model = models.Sequential()
cnn_disc_model.add(tf.keras.Input(shape=(28, 28, 1)))
cnn_disc_model.add(layers.Conv2D(32, 5, strides=2))
cnn_disc_model.add(layers.LeakyReLU())
cnn_disc_model.add(layers.BatchNormalization())
cnn_disc_model.add(layers.Conv2D(64, 5, strides=2))
cnn_disc_model.add(layers.LeakyReLU())
cnn_disc_model.add(layers.BatchNormalization())
cnn_disc_model.add(layers.Flatten())
cnn_disc_model.add(layers.Dense(1, activation='sigmoid'))  # (None, 1)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 12, 12, 32)        832

 leaky_re_lu_2 (LeakyReLU)   (None, 12, 12, 32)        0

 batch_normalization_2 (Batc  (None, 12, 12, 32)       128
 hNormalization)

 conv2d_2 (Conv2D)           (None, 4, 4, 64)          51264

 leaky_re_lu_3 (LeakyReLU)   (None, 4, 4, 64)          0

 batch_normalization_3 (Batc  (None, 4, 4, 64)         256
 hNormalization)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 1)                 1025

=================================================================
Total params: 53,505
Trainable params: 53,313
Non-trainable params: 192
```

The CNN GAN Model

```
# Full GAN
cnn_disc_model.trainable = False
cnn_GAN_model = models.Sequential([tf.keras.layers.Input(
    shape=(6272,)), cnn_gen_model, cnn_disc_model])
cnn_GAN_model.build(input_shape=(6272,))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 28, 28, 1)         54713

 sequential_1 (Sequential)   (None, 1)                 53505

=================================================================
Total params: 108,218
Trainable params: 54,377
Non-trainable params: 53,841
```

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

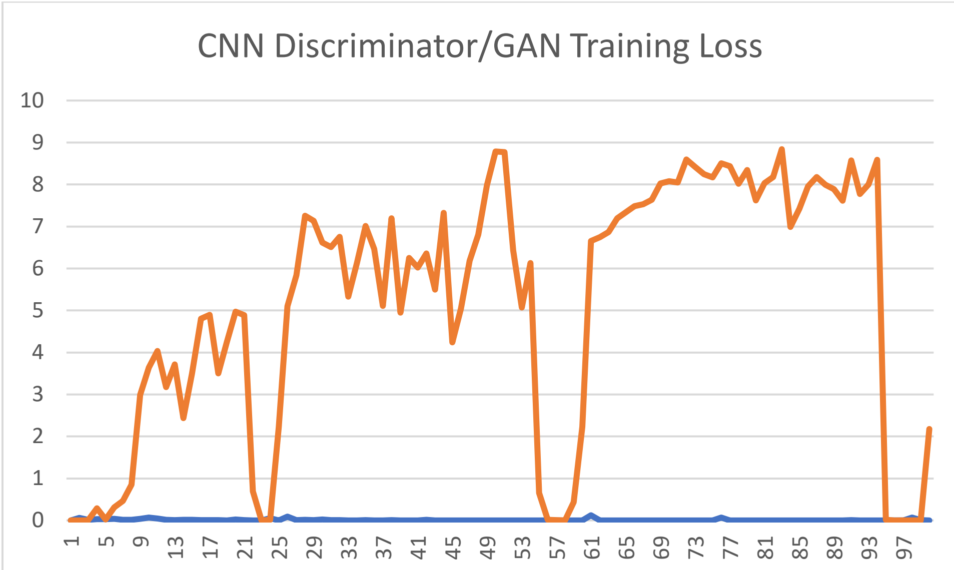
CNN Training Loss Per Epoch

Epoch	Training Loss	Test Loss	Epoch	Training Loss	Test Loss
0	0.0001791047106962651	1.2611647005344524e-34	**50**	0.0005514330696314573	8.770377159118652
1	0.05957120284438133	0.0015854808734729886	**51**	0.001434166100807488	6.432137966156006
2	0.010034257546067238	4.5194802567039005e-08	**52**	4.7547848225804046e-05	5.066755771636963
3	0.030130553990602493	0.2879081666469574	**53**	0.0002733312430791557	6.130360126495361

4	0.02610974945127964	0.020778097212314606	**54**	0.00012528635852504522	0.65610671043396
5	0.033010635524988174	0.31226906180381775	**55**	1.7217266758962069e-06	0.012183445505797863
6	0.0104494858533144	0.46299535036087036	**56**	4.478170376387425e-05	0.007874768227338791
7	0.015244405716657639	0.8511728048324585	**57**	0.0008844331023283303	0.0005371880251914263
8	0.0347551628947258	2.988057851791382	**58**	4.3432210077298805e-05	0.4318878948688507
9	0.06737023591995239	3.6387710571289062	**59**	6.217388727236539e-05	2.226496696472168
10	0.04552295804023743	4.034547328948975	**60**	0.12197347730398178	6.655332088470459
11	0.01012037880718708	3.177661657333374	**61**	0.00066091661574319	6.745500564575195
12	0.005894658155739307	3.7149665355682373	**62**	0.0007597299409098923	6.868861198425293
13	0.01004200242459774	2.4360475540161133	**63**	0.00044388126116245985	7.193081855773926
14	0.012006807141005993	3.505023241043091	**64**	0.0006670180591754615	7.340569019317627
15	0.0034727829042822123	4.801883697509766	**65**	0.000524028495419770	7.480850696563721
16	0.0030860528349876404	4.894974231719971	**66**	0.0004922249936498702	7.533853530883789
17	0.007208674680441618	3.50071382522583	**67**	0.0003795555967371911	7.639799118041992
18	0.001297667738981545	4.2584004402160645	**68**	0.0003163463552482426	8.026603698730469
19	0.02375160902738571	4.9730000495910645	**69**	0.00047755689593032	8.075765609741211
20	0.0025050409603863955	4.887083530426025	**70**	0.000378977885702624	8.051383018493652
21	0.0006188740371726453	0.6921811103820801	**71**	0.0002622345054987818	8.595427513122559
22	0.00023532958584837615	0.0003986225347034633	**72**	0.00038689016946591437	8.42041015625
23	0.0507756806910038	0.014116782695055008	**73**	0.00026363975484855473	8.243671417236328
24	7.433368591591716e-05	2.2276875972747803	**74**	0.00027741401572711766	8.171669006347656
25	0.09195014834403992	5.100812911987305	**75**	0.0702829509973526	8.50546646118164
26	0.004182533826678991	5.842406749725342	**76**	0.00018474162789061666	8.433987617492676
27	0.016741693019866943	7.258368968963623	**77**	0.00021110702073201537	8.01400375366211
28	0.003255910240113735	7.130593776702881	**78**	0.00023042417888063937	8.346099853515625

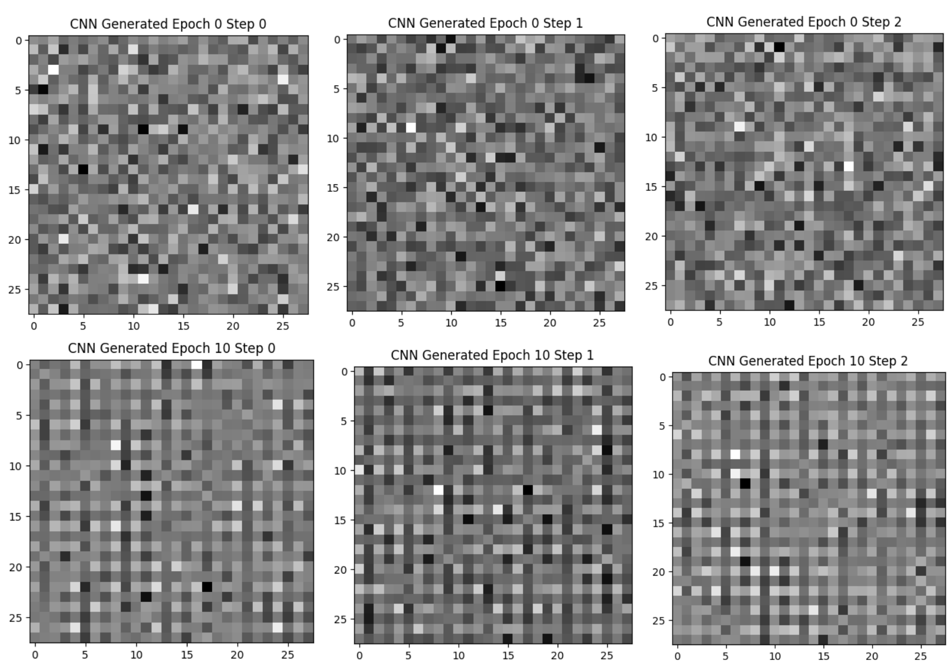
29	0.020350737497210503	6.613577365875244	**79**	0.00012125368812121451	7.620498180389404
30	0.0026462769601494074	6.50969123840332	**80**	0.0001418890169588849	8.030329704284668
31	0.0034328096080571413	6.755902290344238	**81**	0.0002293815341545269	8.174145698547363
32	0.0007909965352155268	5.331682205200195	**82**	0.00010832848056452349	8.840995788574219
33	0.001001332071609795	6.130487442016602	**83**	0.0001802610349841416	6.990046501159668
34	0.0024883649311959743	7.010079383850098	**84**	6.738227239111438e-05	7.423491954803467
35	0.0022047697566449642	6.4646992683410645	**85**	9.671755105955526e-05	7.959722995758057
36	0.0005545821622945368	5.107856273651123	**86**	0.0002011724136536941	8.179006576538086
37	0.006070754025131464	7.191289901733398	**87**	0.00016036634042393416	7.992383003234863
38	0.0004963439423590899	4.945166110992432	**88**	0.00017344091611448675	7.885098934173584
39	0.0007118553621694446	6.25137186050415	**89**	0.00011105283920187503	7.610461235046387
40	0.0005598401767201722	6.020808696746826	**90**	0.00275879236869514	8.571656227111816
41	0.013507477939128876	6.355642318725586	**91**	7.348756480496377e-05	7.771444320678711
42	0.00024395556829404086	5.49357795715332	**92**	0.0004482402582652867	8.002350807189941
43	0.0017429712461307645	7.3229475021362305	**93**	8.444989362033084e-05	8.586740493774414
44	0.00015546039503533393	4.242682933807373	**94**	0.0003593981673475355	0.01759181171655655
45	8.874354534782469e-05	5.051324367523193	**95**	2.7691252069139694e-11	7.074774657667149e-06
46	0.000597742444369942	6.1808953285217285	**96**	2.1056118839624105e-06	0.0
47	0.0005582966841757298	6.804682731628418	**97**	0.06361004710197449	1.995489927811289e-12
48	0.0004354499978944659	7.979450702667236	**98**	0.009731325320899487	8.20274042179122e-20
49	0.0011792771983891726	8.789504051208496	**99**	0.0008047168958000839	2.173821449279785

Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023



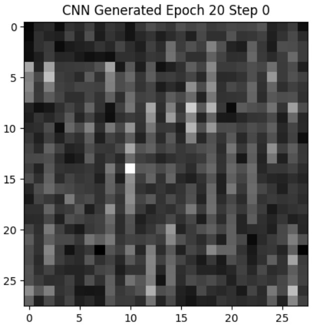
I plotted the loss to see what was happening, and it seems that my generator was not learning properly. I would have spent more time figuring out how to fix the issue, but I was short on time. There does seem to be a jump in loss for the discriminator (blue) when the loss for the CNN GAN model (orange) deeps down to its level, so I think there may be still competition between the two, but the discriminator seems to be just way better than the generator. The changes in the images below suggest that the generator is learning something but not quite what we wanted it to learn.

<u>CNN Synthetic Images</u>

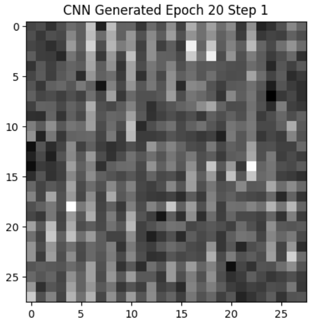


Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

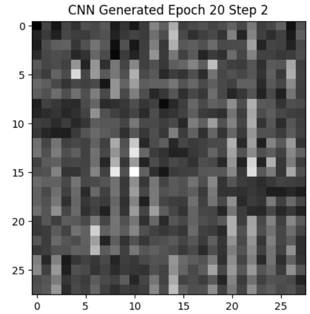
CNN Generated Epoch 20 Step 0



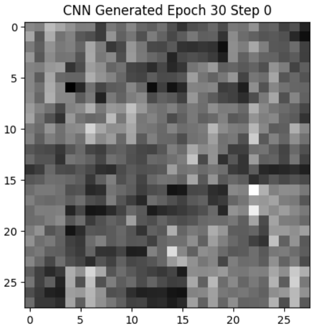
CNN Generated Epoch 20 Step 1



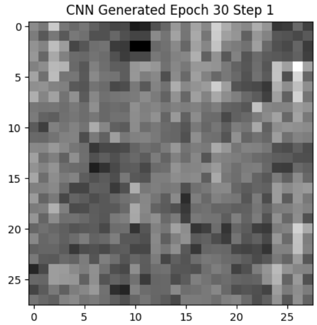
CNN Generated Epoch 20 Step 2



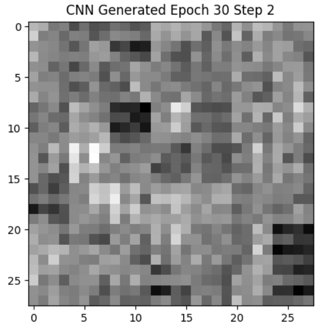
CNN Generated Epoch 30 Step 0



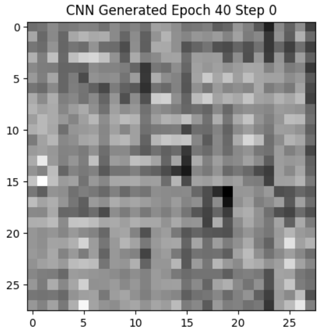
CNN Generated Epoch 30 Step 1



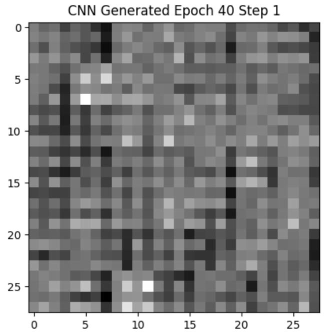
CNN Generated Epoch 30 Step 2



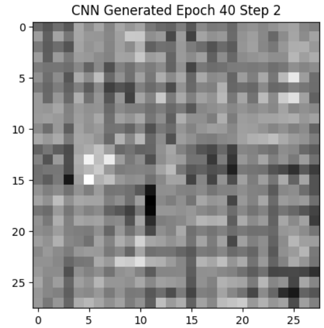
CNN Generated Epoch 40 Step 0



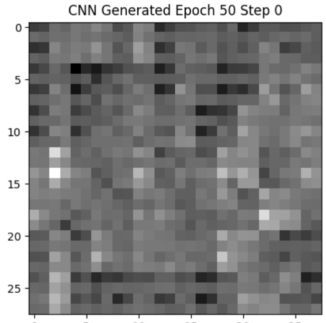
CNN Generated Epoch 40 Step 1



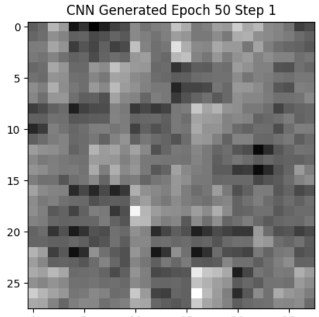
CNN Generated Epoch 40 Step 2



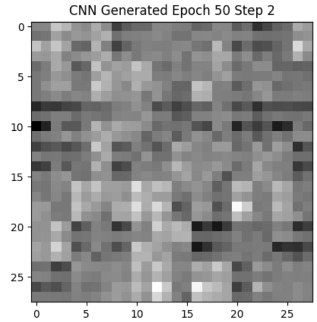
CNN Generated Epoch 50 Step 0



CNN Generated Epoch 50 Step 1

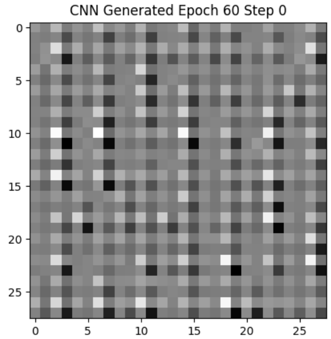


CNN Generated Epoch 50 Step 2

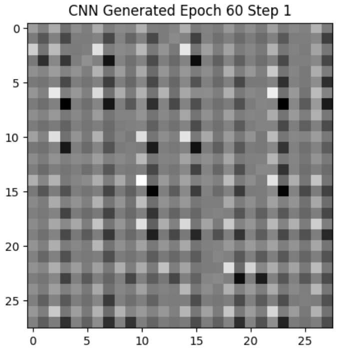


Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023

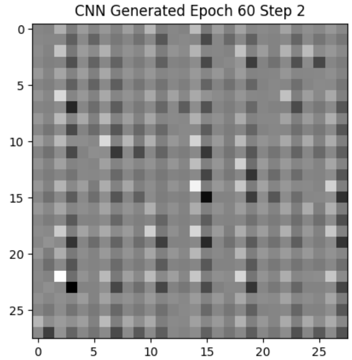
CNN Generated Epoch 60 Step 0



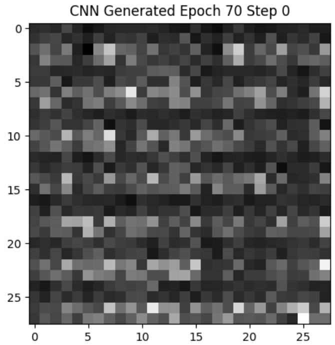
CNN Generated Epoch 60 Step 1



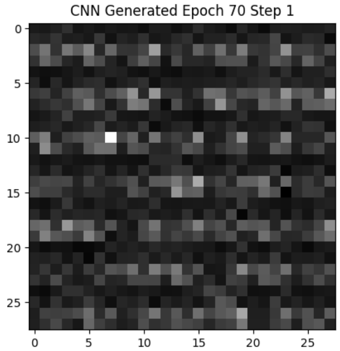
CNN Generated Epoch 60 Step 2



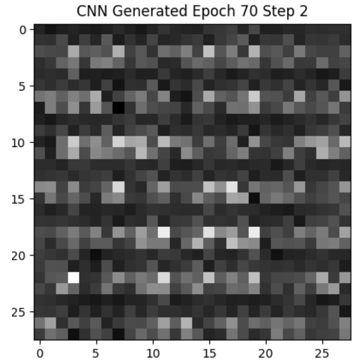
CNN Generated Epoch 70 Step 0



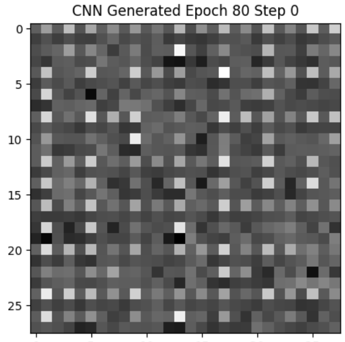
CNN Generated Epoch 70 Step 1



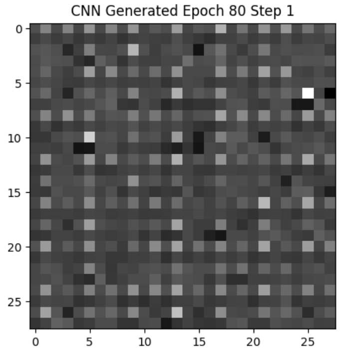
CNN Generated Epoch 70 Step 2



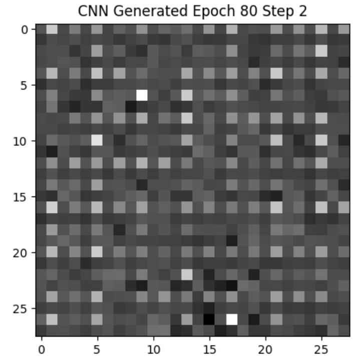
CNN Generated Epoch 80 Step 0



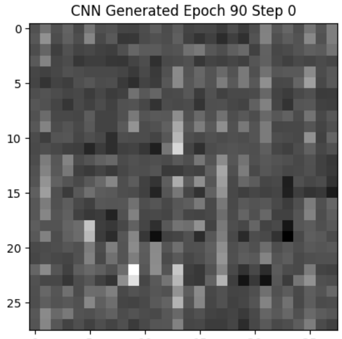
CNN Generated Epoch 80 Step 1



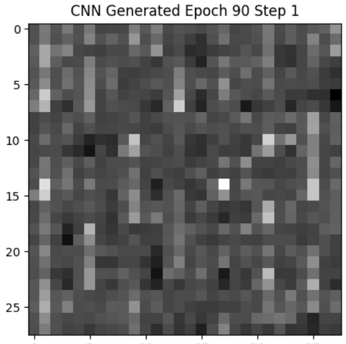
CNN Generated Epoch 80 Step 2



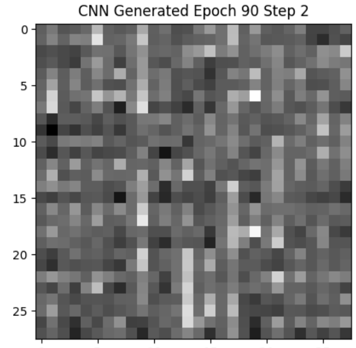
CNN Generated Epoch 90 Step 0



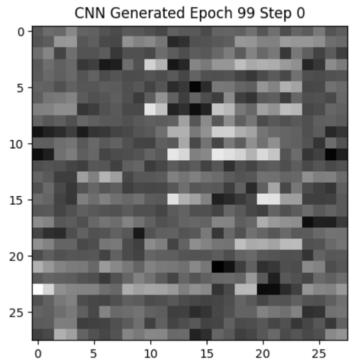
CNN Generated Epoch 90 Step 1

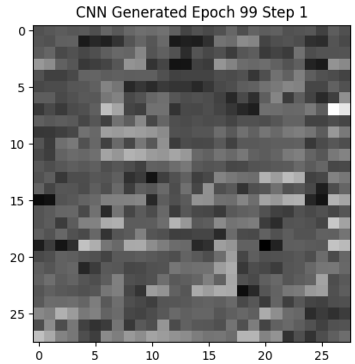


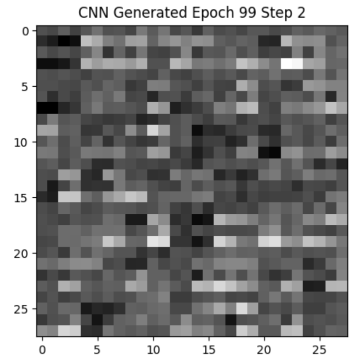
CNN Generated Epoch 90 Step 2



Cera Oh
CS510 CV and DL
Summer 2023
8/17/2023







Conclusion

The generator of the GAN model has learned that fashion items are lighter color and in the center area of the 28x28 square, but it still produced outputs with a lot of noise. I suppose the last epoch synthetic images pulled can be though of as a shirt or sweater with a lot of noise. Generally, the generator seems to improve little by little with each epoch. The loss values fluctuate for the discriminator and GAN, which shows that the generator and the discriminator were in competition. I am not sure why I have GAN loss of 0 for epochs 1-9 for GAN in the beginning. It could be that the discriminator was really bad at spotting fake images and thought everything was real. The GAN model worked as expected. The CNN GAN model though, did not work as I expected and I think I etiher have architectural issues with the generator, or an overly good discrminator in comparison to the generator, or some kind of training issues with the generator. The generator was learning something, as it was producing different looking images per epoch, but it seems to be missing the point that the image is supposed to have a lighter colored object in the middle and the edges should be black.

References:

https://keras.io/guides/writing_a_training_loop_from_scratch/
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/