University *of* Dayton

ECE 595 Homework #4
7/27/2021
Christopher Couture Del Valle

Introduction:

Students were assigned to understand utilizing a denoising auto-encoder for cleaning noisy documents and generating new samples with generative models.

Methodology:

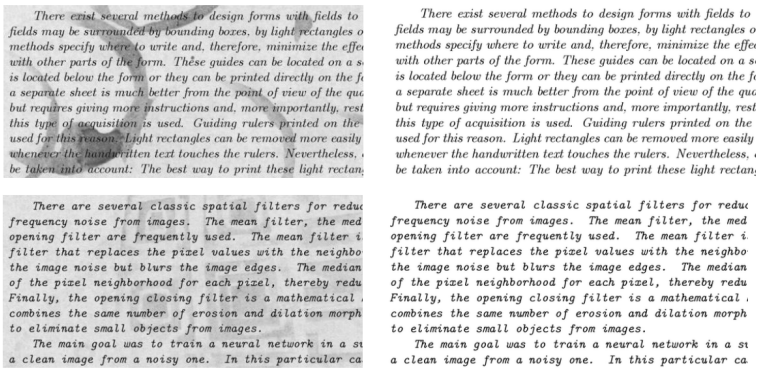

Figure 1: Example images from document denoising dataset: noisy image on the left and clear images on the right.

Task 1 was to implement and evaluate an Auto-Encoder (AE) model for denoising the document files. The database was provided on Isidore including training and testing samples and images are shown in Figure 1. After training the model successfully, qualitative and quantitative analysis was provided on testing samples.

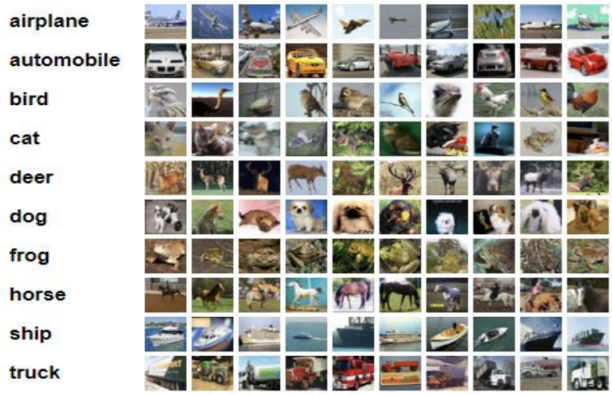

Figure 2: Cifar-10 Dataset

Task 2 was to implement generative models: The Convolutional Variational Auto-Encoder (VAE) and Deep Convolutional Generative Adversarial Network (DCGAN). The models were trained and tested on CIFAR 10 dataset and the example images were shown in Figure 2. A qualitative comparison was made between the two models.

Deep Learning Architecture:

Autoencoder

_____The autoencoder was created utilizing an input layer, 3 encoded layers, 3 decoded layers and an output layer. The input layer consisted of the input image size shape which was then fed to the first layer of the encoder. Each encoded layer utilized the previous and consisted of 3 Conv2d and 3 max pooling. The latent view would then pass off to the decoded layers which utilized Conv2d and upsampling 2d to produce a final output layer.

```
# input layer
input_layer = Input(shape=(258, 540,1))

# encoding architecture
encoded_layer1 = Conv2D(128, (3, 3), activation='relu', padding='same')(input_layer)
encoded_layer1 = MaxPool2D( (2, 2), padding='same')(encoded_layer1)
encoded_layer2 = Conv2D(64, (3, 3), activation='relu', padding='same')(encoded_layer1)
encoded_layer2 = MaxPool2D( (2, 2), padding='same')(encoded_layer2)
encoded_layer3 = Conv2D(32, (3, 3), activation='relu', padding='same')(encoded_layer2)
latent_view    = MaxPool2D( (2, 2), padding='same')(encoded_layer3)
```

```
# decoding architecture
decoded_layer1 = Conv2D(32, (3, 3), activation='relu', padding='same')(latent_view)
decoded_layer1 = UpSampling2D((2, 2))(decoded_layer1)
decoded_layer2 = Conv2D(64, (3, 3), activation='relu', padding='same')(decoded_layer1)
decoded_layer2 = UpSampling2D((2, 2))(decoded_layer2)
decoded_layer3 = Conv2D(128, (3, 3), activation='relu')(decoded_layer2)
decoded_layer3 = UpSampling2D((2, 2))(decoded_layer3)
output_layer   = Conv2D(1, (3, 3), padding='same')(decoded_layer3)
```

```
# compile the model
model = Model(input_layer, output_layer)
model.compile(optimizer='adam', loss='mse')
```

Figure 3: Autoencoder Model Python Syntax

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 260, 540, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 260, 540, 128)     3584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 130, 270, 128)     0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 130, 270, 64)      73792
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 65, 135, 64)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 65, 135, 32)       18464
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 33, 68, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 33, 68, 32)        9248
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 66, 136, 32)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 66, 136, 64)       18496
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 132, 272, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 130, 270, 128)     73856
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 260, 540, 128)     0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 260, 540, 1)       1153
=================================================================
Total params: 198,593
Trainable params: 198,593
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: Autoencoder Model Architecture

VAE

_____The variational autoencoder, a generative model was created in two parts similar to the ae. The decoder was created with the decoder taking the latent distribution sample as input. The original decoder layer outputted the total pixel size of the input image which was in this case 32x32x3 = 3072.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_decoder (InputLayer)   [(None, 2)]               0
_________________________________________________________________
intermediate_decoder (Dense) (None, 512)               1536
_________________________________________________________________
original_decoder (Dense)     (None, 3072)              1575936
=================================================================
Total params: 1,577,472
Trainable params: 1,577,472
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Decoder Model architecture

The vae was then compiled utilizing the previously created decoder model utilizing the input image size and y value. The model was then compiled utilizing RMSprop as the optimizer.

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input (InputLayer)              [(None, 3072)]       0
__________________________________________________________________________________________________
intermediate_encoder (Dense)    (None, 512)          1573376     input[0][0]
__________________________________________________________________________________________________
latent_encoder (Dense)          (None, 2)            1026        intermediate_encoder[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 2)            6           latent_encoder[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 2)            6           latent_encoder[0][0]
__________________________________________________________________________________________________
lambda (Lambda)                 (None, 2)            0           dense[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
model (Functional)              (None, 3072)         1577472     lambda[0][0]
__________________________________________________________________________________________________
custom_variational_layer (Custo (None, 3072)         0           input[0][0]
                                                                 model[0][0]
==================================================================================================
Total params: 3,151,886
Trainable params: 3,151,886
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 6: VAE Model architecture

DCGAN

_____The DCGAN, another generative model, utilized a discriminator and generator. The job of the generator is to spawn 'fake' images that look like the training images. The job of the discriminator is to look at an image and output whether or not it is a real training image or a fake image from the generator.

```
def build_generator(self):

    model = Sequential()

    model.add(Dense(256, input_dim=self.latent_dim))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(1024))
    model.add(LeakyReLU(alpha=0.2))
    model.add(BatchNormalization(momentum=0.8))
    model.add(Dense(np.prod(self.img_shape), activation='tanh'))
    model.add(Reshape(self.img_shape))

    model.summary()

    noise = Input(shape=(self.latent_dim,))
    img = model(noise)

    return Model(noise, img)

def build_discriminator(self):

    model = Sequential()

    model.add(Flatten(input_shape=self.img_shape))
    model.add(Dense(512))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(256))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.summary()

    img = Input(shape=self.img_shape)
    validity = model(img)
    return Model(img, validity)
```

Figure 7: DCGAN Generator and Discriminator Model Syntax

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 3072)              0
_________________________________________________________________
dense (Dense)                (None, 512)               1573376
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 1,704,961
Trainable params: 1,704,961
Non-trainable params: 0
```

Figure 8: DCGAN Discriminator Model

```
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
dense_6 (Dense)              (None, 3072)              3148800
_________________________________________________________________
reshape (Reshape)            (None, 32, 32, 3)         0
=================================================================
Total params: 3,838,720
Trainable params: 3,835,136
Non-trainable params: 3,584
```

Figure 9: DCGAN Generator Model

Results + Discussion & Comparison:

Autoencoder

The autoencoder did not initially perform that well due to several factors. Initially the incorrect image size was being extrapolated giving outputting some bad training values. This issue was addressed by extrapolating a 250,540 image and epochs were progressively increased. High RAM usage was addressed by lowering the batch size.

```
Epoch 1/25
9/9 [==============================] - 52s 554ms/step - loss: 0.1712 - val_loss: 0.0779
Epoch 2/25
9/9 [==============================] - 4s 485ms/step - loss: 0.0819 - val_loss: 0.0870
Epoch 3/25
9/9 [==============================] - 4s 488ms/step - loss: 0.0682 - val_loss: 0.0756
Epoch 4/25
9/9 [==============================] - 4s 490ms/step - loss: 0.0612 - val_loss: 0.0794
Epoch 5/25
9/9 [==============================] - 4s 495ms/step - loss: 0.0583 - val_loss: 0.0863
Epoch 6/25
9/9 [==============================] - 4s 511ms/step - loss: 0.0571 - val_loss: 0.0809
Epoch 7/25
9/9 [==============================] - 4s 501ms/step - loss: 0.0556 - val_loss: 0.0818
Epoch 8/25
9/9 [==============================] - 4s 514ms/step - loss: 0.0541 - val_loss: 0.0812
Epoch 9/25
9/9 [==============================] - 5s 520ms/step - loss: 0.0552 - val_loss: 0.0890
Epoch 10/25
9/9 [==============================] - 4s 516ms/step - loss: 0.0555 - val_loss: 0.0793
Epoch 11/25
9/9 [==============================] - 4s 502ms/step - loss: 0.0542 - val_loss: 0.0802
Epoch 12/25
9/9 [==============================] - 4s 500ms/step - loss: 0.0527 - val_loss: 0.0837
Epoch 13/25
9/9 [==============================] - 4s 495ms/step - loss: 0.0525 - val_loss: 0.0805
```

Figure 10: Autoencoder Training Output 25 epochs

After 25 epochs, the loss had only decreased to .05 with validation loss being around .079. I determined this was the case compared to the lecture examples due to the images being much larger so I had to increase the number of epochs.

Figure 11: Autoencoder Prediction Output 25 epochs

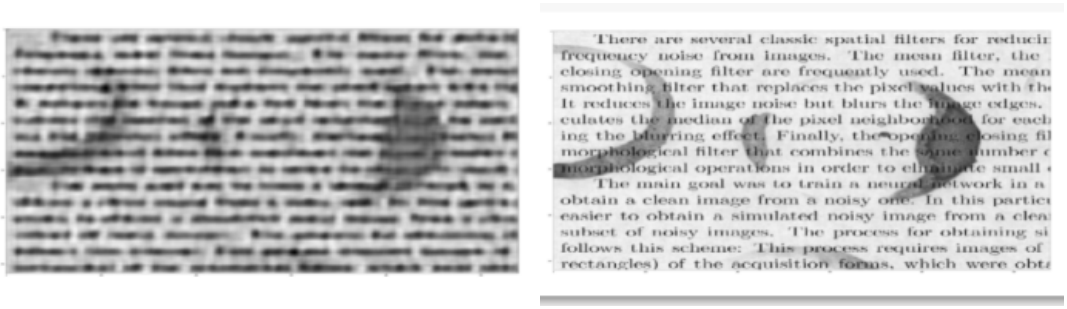

Figure 12: Autoencoder Pred Image (L), Autoencoder Pred Image (R),

_____The prediction output was nowhere near as clean as I would have liked. The image was left blurry while the only main improvement was creases were less visible and the coffee stain had been lightened.

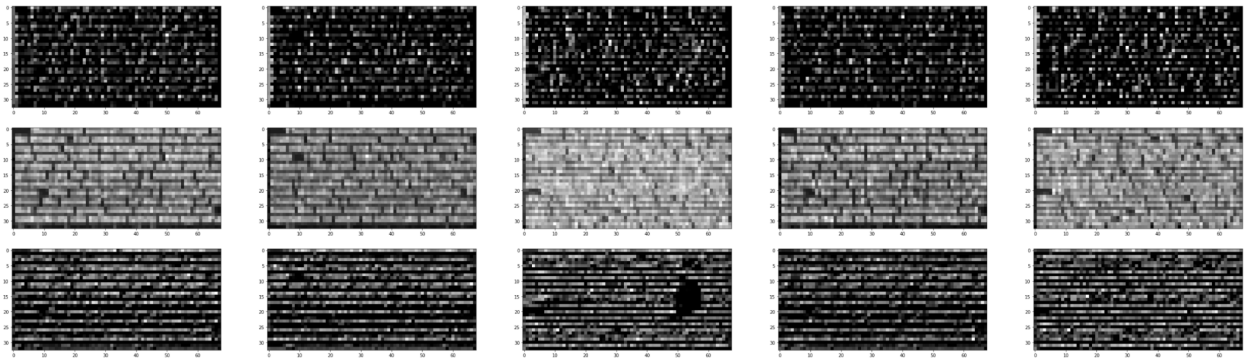

Figure 13: Autoencoder Quality Output 25 epochs

_____I could tell the model was on the right track as it was identifying some lines of text and in particular noticed the coffee stain. This gave me the confidence to progressively tweak the model.

```
Epoch 91/100
9/9 [==============================] - 4s 501ms/step - loss: 0.0283 - val_loss: 0.0279
Epoch 92/100
9/9 [==============================] - 5s 519ms/step - loss: 0.0280 - val_loss: 0.0278
Epoch 93/100
9/9 [==============================] - 4s 504ms/step - loss: 0.0279 - val_loss: 0.0276
Epoch 94/100
9/9 [==============================] - 4s 516ms/step - loss: 0.0274 - val_loss: 0.0279
Epoch 95/100
9/9 [==============================] - 4s 503ms/step - loss: 0.0275 - val_loss: 0.0275
Epoch 96/100
9/9 [==============================] - 4s 507ms/step - loss: 0.0271 - val_loss: 0.0275
Epoch 97/100
9/9 [==============================] - 4s 504ms/step - loss: 0.0272 - val_loss: 0.0272
Epoch 98/100
9/9 [==============================] - 4s 505ms/step - loss: 0.0277 - val_loss: 0.0272
Epoch 99/100
9/9 [==============================] - 4s 502ms/step - loss: 0.0266 - val_loss: 0.0271
Epoch 100/100
9/9 [==============================] - 4s 515ms/step - loss: 0.0272 - val_loss: 0.0269
```

Figure 14: Autoencoder Training Output 100 epochs

Upon increasing the number of epochs I immediately noticed an immediate improvement in loss which dipped all the way down to .027. This indicated to me my hunch about image size requiring a larger run was correct.

Figure 15: Autoencoder Prediction Output 25 epochs

Upon increasing the number of epochs I immediately noticed an improvement in predicted image quality. The coffee stains were close to being completely removed in some places while the wrinkles completely gone from other images.

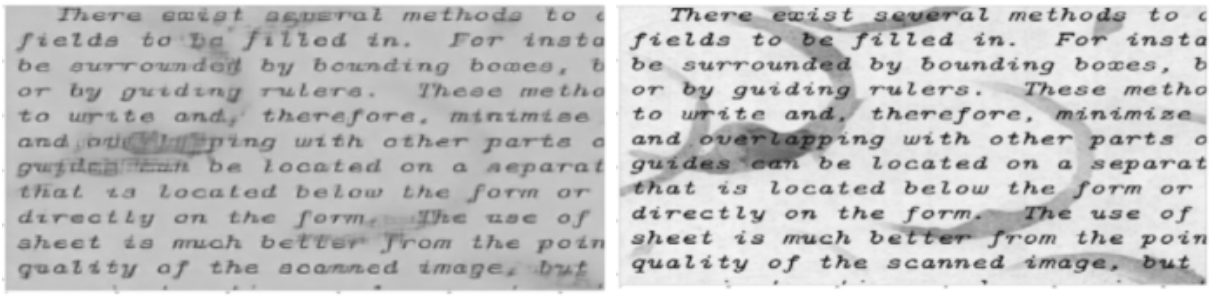

Figure 16: Autoencoder Pred Image (L), Autoencoder Pred Image (R),

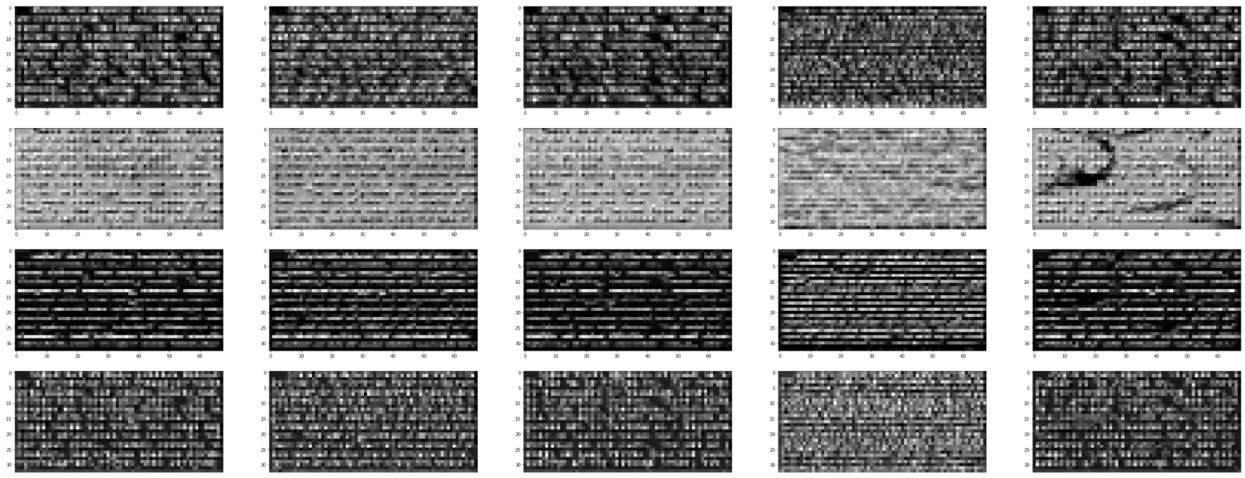

Figure 17: Autoencoder Quality Output 100 epochs

In terms of image quality metrics, the model was able to distinguish characters significantly better while also identifying creases, spills and other blemishes.

VAE

_____The VAE model did not perform that well similarly to the example given in lecture. Whereas the lecture model outputted a loss of .49, my model outputted a loss of 0.69. This was due to my implementation utilizing the cifar 10 dataset which utilizes three color channels. This makes the images more difficult to run through the model. The generative output at the end was also lacking in quality but was nonetheless generated properly.

```
Epoch 10/20
9375/9375 [==============================] - 46s 5ms/step - loss: 0.6915 - val_loss: 0.6919
Epoch 11/20
9375/9375 [==============================] - 49s 5ms/step - loss: 0.6916 - val_loss: 0.6919
Epoch 12/20
9375/9375 [==============================] - 49s 5ms/step - loss: 0.6916 - val_loss: 0.6919
Epoch 13/20
9375/9375 [==============================] - 48s 5ms/step - loss: 0.6915 - val_loss: 0.6918
Epoch 14/20
9375/9375 [==============================] - 48s 5ms/step - loss: 0.6916 - val_loss: 0.6918
Epoch 15/20
9375/9375 [==============================] - 46s 5ms/step - loss: 0.6915 - val_loss: 0.6919
Epoch 16/20
9375/9375 [==============================] - 46s 5ms/step - loss: 0.6915 - val_loss: 0.6919
Epoch 17/20
9375/9375 [==============================] - 47s 5ms/step - loss: 0.6915 - val_loss: 0.6919
Epoch 18/20
9375/9375 [==============================] - 49s 5ms/step - loss: 0.6916 - val_loss: 0.6919
Epoch 19/20
9375/9375 [==============================] - 49s 5ms/step - loss: 0.6916 - val_loss: 0.6919
Epoch 20/20
9375/9375 [==============================] - 49s 5ms/step - loss: 0.6915 - val_loss: 0.6919
<keras.callbacks.History at 0x7f8ba04b91d0>
```

Figure 18: VAE training output

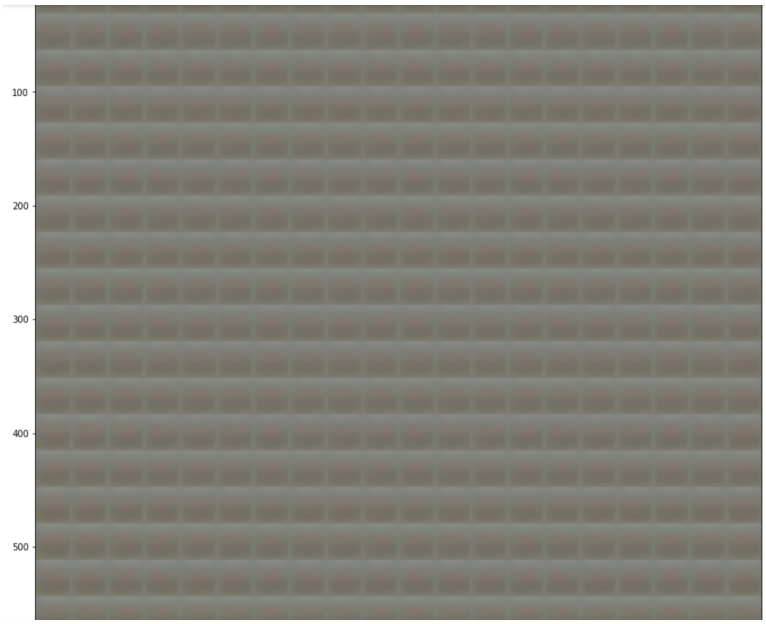

Figure 19: VAE Generative Output

DCGAN

_____In comparison to VAE, the DCGAN yielded some rather solid results. Initially the generator had the discriminator fooled. With an accuracy of 50% this indicates the discriminator is only right half the time. As the epochs progressed and the generator images became more realistic, the Discriminator loss decreased, while the accuracy and generator loss increased. This indicated the DCGAN was functioning properly!

```
786 [D loss: 0.725293, acc.: 42.19%] [G loss: 0.802963]
787 [D loss: 0.719020, acc.: 39.06%] [G loss: 0.806783]
788 [D loss: 0.726040, acc.: 37.50%] [G loss: 0.804946]
789 [D loss: 0.701920, acc.: 50.00%] [G loss: 0.827112]
790 [D loss: 0.736028, acc.: 37.50%] [G loss: 0.796697]
791 [D loss: 0.747874, acc.: 32.81%] [G loss: 0.778332]
792 [D loss: 0.727529, acc.: 40.62%] [G loss: 0.804175]
793 [D loss: 0.706207, acc.: 48.44%] [G loss: 0.825914]
794 [D loss: 0.737578, acc.: 50.00%] [G loss: 0.832175]
795 [D loss: 0.724798, acc.: 46.88%] [G loss: 0.817256]
796 [D loss: 0.733033, acc.: 42.19%] [G loss: 0.821423]
797 [D loss: 0.712584, acc.: 51.56%] [G loss: 0.805397]
798 [D loss: 0.723849, acc.: 50.00%] [G loss: 0.791525]
799 [D loss: 0.724990, acc.: 45.31%] [G loss: 0.781141]
800 [D loss: 0.709014, acc.: 54.69%] [G loss: 0.772218]
```

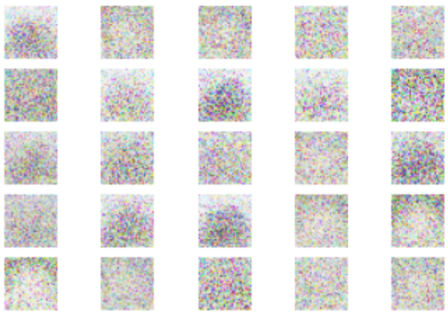

Figure 20: DCGAN Training Output and Generated Images 800 epochs

```
2385 [D loss: 0.697102, acc.: 51.56%] [G loss: 0.771726]
2386 [D loss: 0.712711, acc.: 34.38%] [G loss: 0.844759]
2387 [D loss: 0.738876, acc.: 42.19%] [G loss: 0.770505]
2388 [D loss: 0.730289, acc.: 29.69%] [G loss: 0.747046]
2389 [D loss: 0.705783, acc.: 32.81%] [G loss: 0.775418]
2390 [D loss: 0.710887, acc.: 42.19%] [G loss: 0.757735]
2391 [D loss: 0.710191, acc.: 48.44%] [G loss: 0.757998]
2392 [D loss: 0.707184, acc.: 54.69%] [G loss: 0.747911]
2393 [D loss: 0.713165, acc.: 29.69%] [G loss: 0.768743]
2394 [D loss: 0.699943, acc.: 54.69%] [G loss: 0.762696]
2395 [D loss: 0.699267, acc.: 54.69%] [G loss: 0.770742]
2396 [D loss: 0.710907, acc.: 43.75%] [G loss: 0.766701]
2397 [D loss: 0.700451, acc.: 48.44%] [G loss: 0.757077]
2398 [D loss: 0.709386, acc.: 42.19%] [G loss: 0.760474]
2399 [D loss: 0.699813, acc.: 53.12%] [G loss: 0.807126]
2400 [D loss: 0.725718, acc.: 35.94%] [G loss: 0.765838]
```

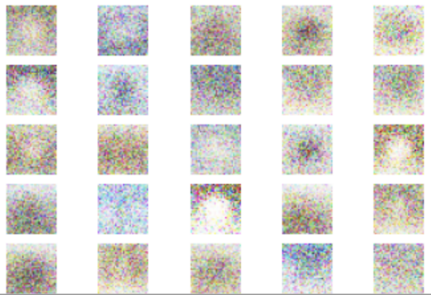

Figure 21: DCGAN Training Output and Generated Images 2400 epochs

```
9188 [D loss: 0.303323, acc.: 92.19%] [G loss: 2.704818]
9189 [D loss: 0.487477, acc.: 75.00%] [G loss: 1.681684]
9190 [D loss: 0.299526, acc.: 90.62%] [G loss: 2.741573]
9191 [D loss: 0.488861, acc.: 75.00%] [G loss: 2.628856]
9192 [D loss: 0.336703, acc.: 85.94%] [G loss: 2.931231]
9193 [D loss: 0.417487, acc.: 76.56%] [G loss: 1.667081]
9194 [D loss: 0.415616, acc.: 82.81%] [G loss: 1.815463]
9195 [D loss: 0.371422, acc.: 84.38%] [G loss: 2.357869]
9196 [D loss: 0.363875, acc.: 90.62%] [G loss: 2.928938]
9197 [D loss: 0.509812, acc.: 75.00%] [G loss: 1.586779]
9198 [D loss: 0.499059, acc.: 76.56%] [G loss: 1.821058]
9199 [D loss: 0.469546, acc.: 76.56%] [G loss: 1.738559]
9200 [D loss: 0.383822, acc.: 90.62%] [G loss: 2.150250]
```

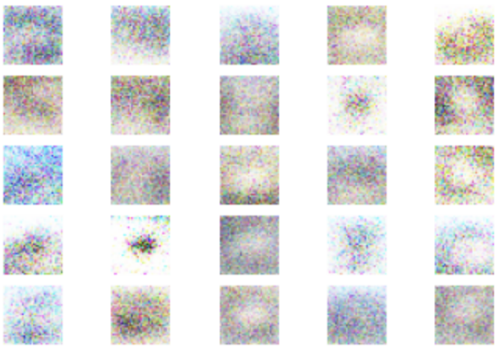

Figure 22: DCGAN Training Output and Generated Images 9200 epochs

Conclusion:

Homework 4 was another astounding success with images being mostly denoised and the generative DCGAN and VAE models being properly run, compiled and compared.