Finding Hidden Meaning:

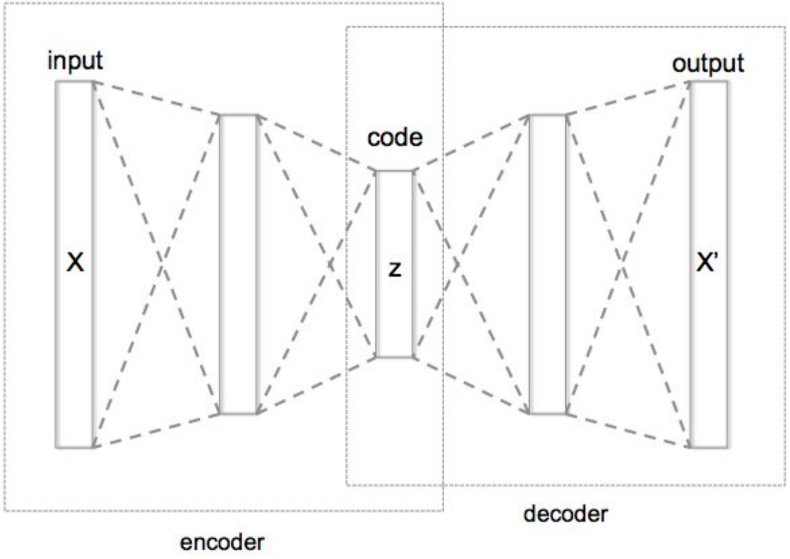
Using Autoencoders to Better Understand Student Design Work

Alexander Boucher,
Amixadai Hernandez,
Ana Forrister,
Brooke Carlson,
Melos Shtaloja

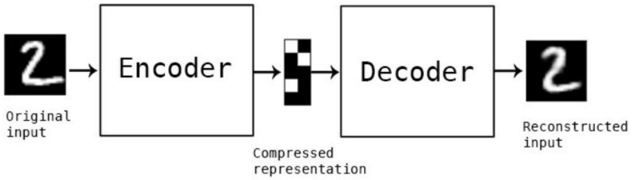
Introduction

An autoencoder is an **unsupervised learning** technique for **neural networks** that learns efficient data representations (encoding) by training the network to ignore signal "noise."

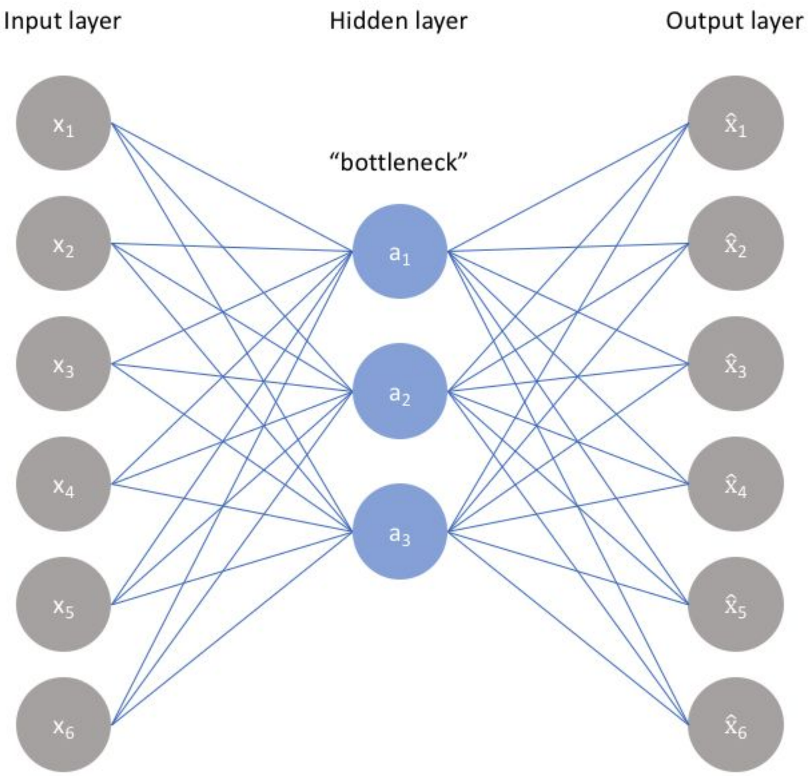
It learns how to reconstruct the data back from the reduced encoded representation to a representation that is as close to the original input as possible.



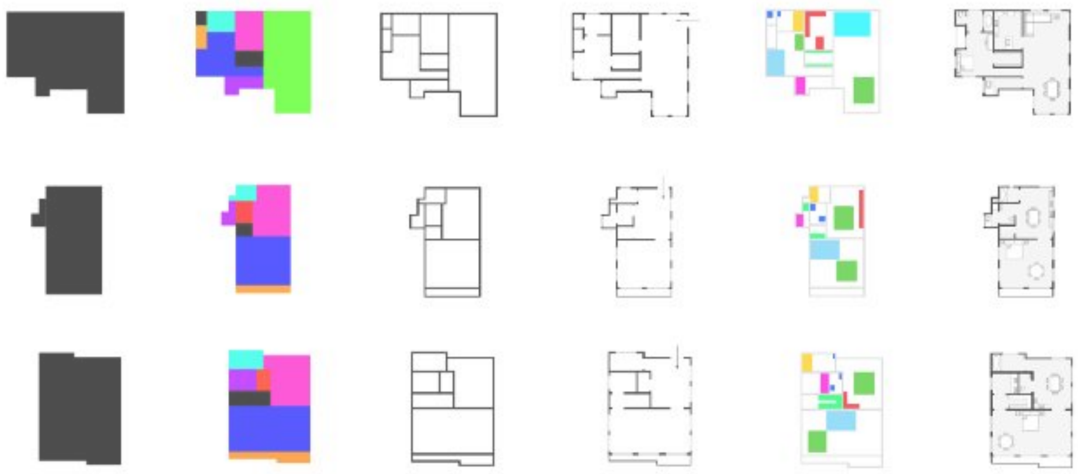
Schematic structure of an "deep" autoencoder with 3 layers

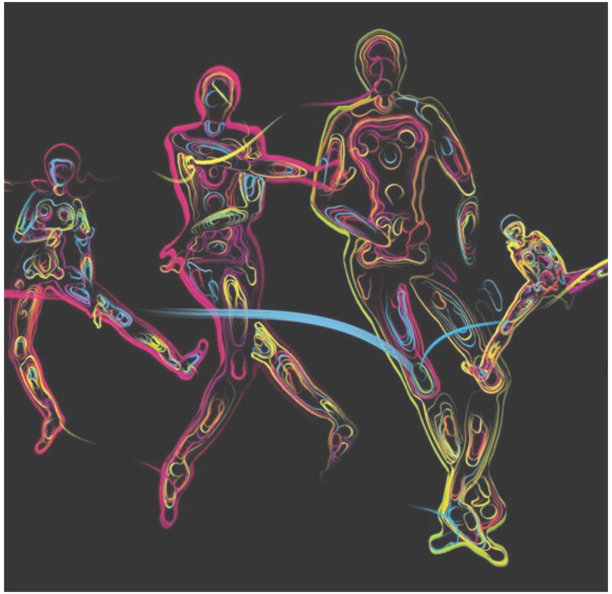


Introduction



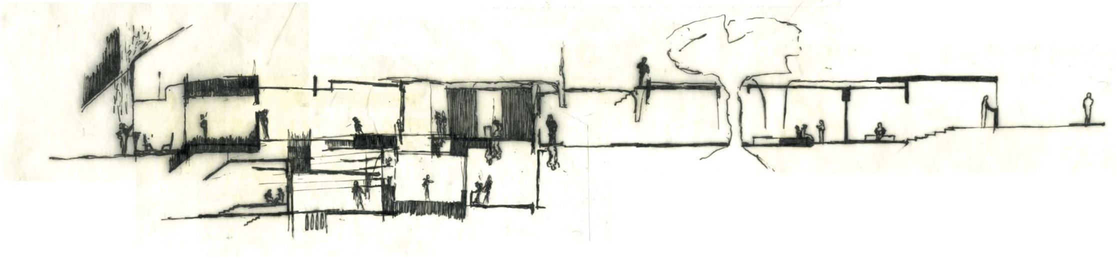
Precedents

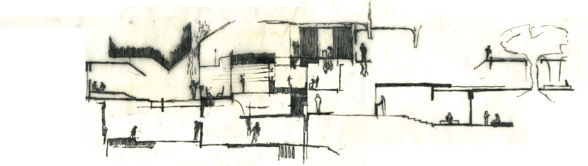


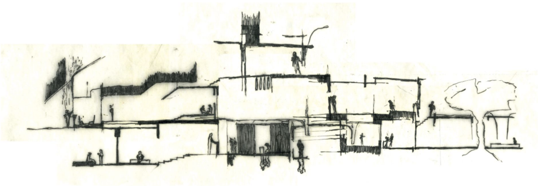


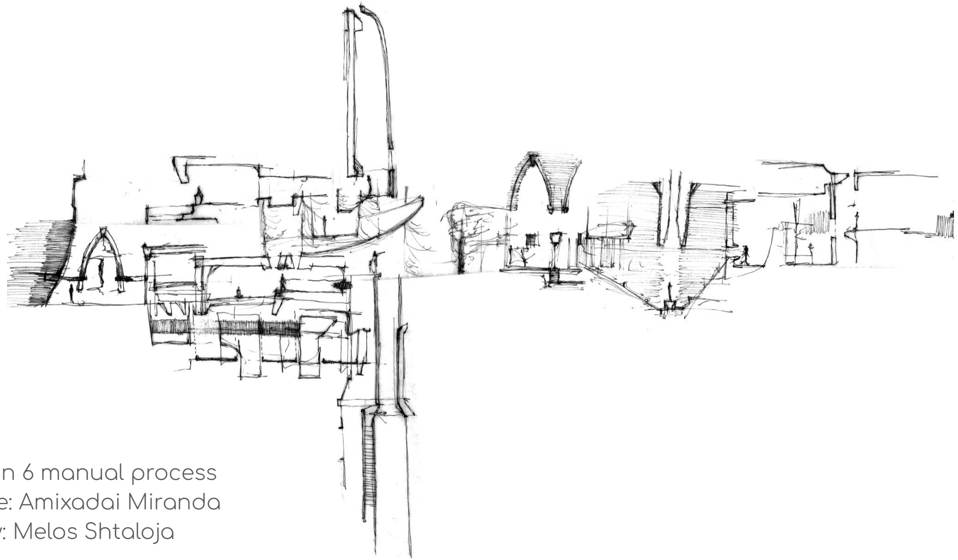
Left: Stanislas Chaillou, Harvard GSD (2019)

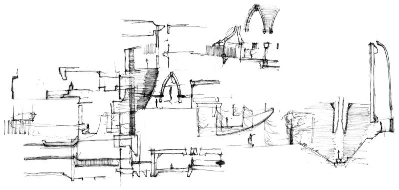
Right: The Dancing Salesman, Problem (2011)

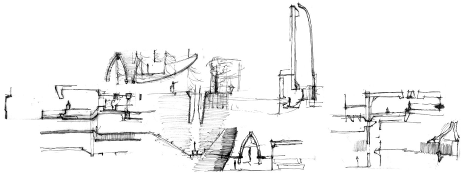












Design 6 manual process
Above: Amixadai Miranda
Below: Melos Shtaloja

What underlying information can be discovered in the UF SoA Archives by using an autoencoder?

What are the most recurrent or common features in undergraduate design work?

What are the strengths and weakness of current teaching methodologies?

Goals

- Create a stylistic analysis of the teaching methodology at UF School of Architecture
- Embody the collection of student work from SOA with a conceptual composition that reflects the most common approaches to design and design representation
- Discuss considerations for the implementation of the results of a conceptual composition created with autoencoders into the curriculum as a tool for inspiring design

Methodology

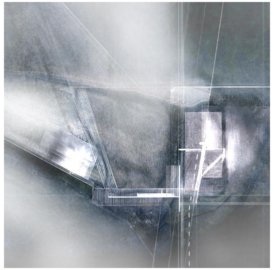
1. Find and select data from UF School of Architecture Design Studio archives

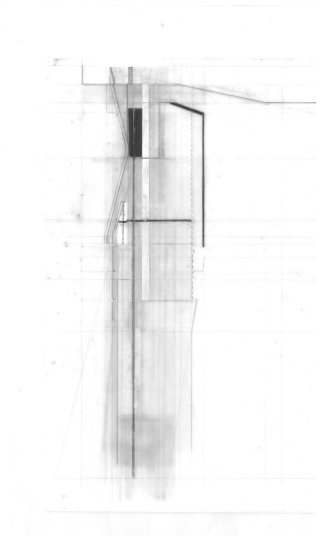
2. Have the selected images go through image partition to expand dataset

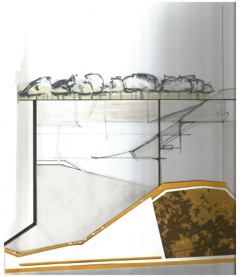
3. Use the dataset to train the autoencoder

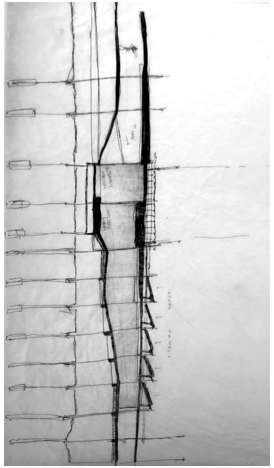
4. Form collages with the generated images to be used as part of the early design process

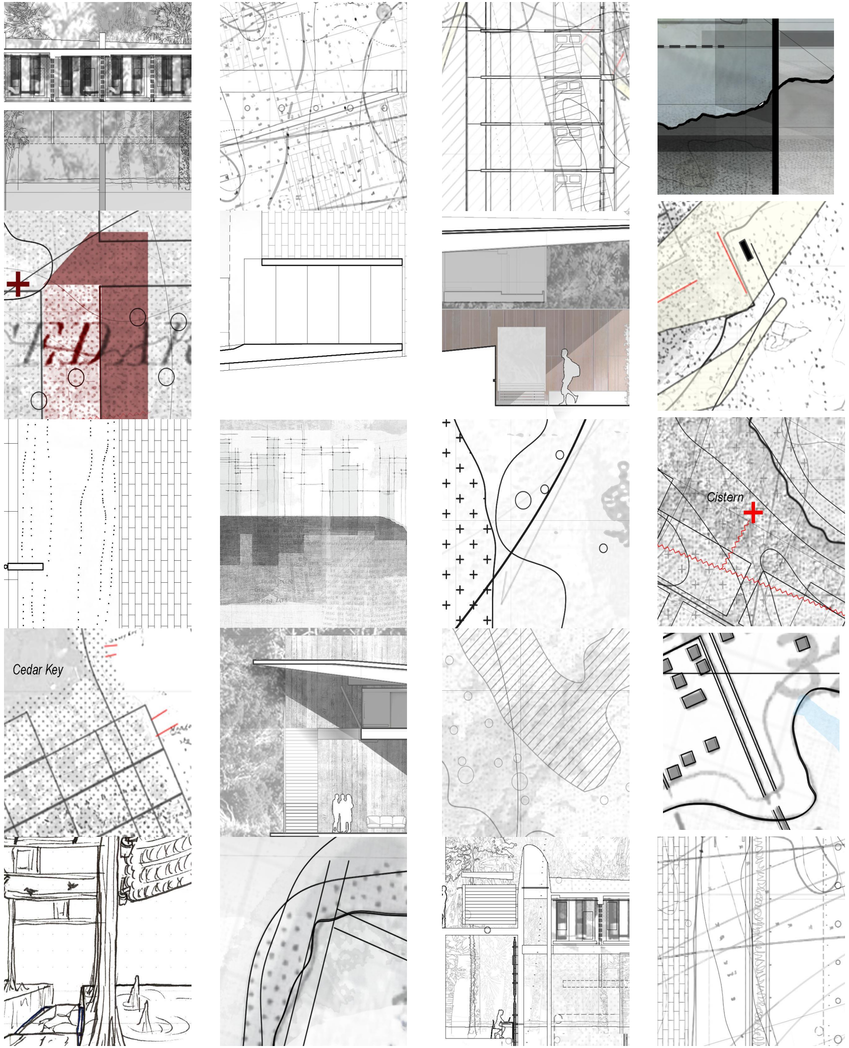
Step 1: Archival Work











Step 2: Data Partitioning

Step 3: Training

```
from glob import glob
import cv2
import random
from google.colab.patches import cv2_imshow

files = glob('/content/gdrive/MyDrive/D5Sorted/PartProcess/*.jpg')
data = []
for filename in files:
  img = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
  #img = cv2.imread(filename)
  resized = cv2.resize(img, (n,n), interpolation = cv2.INTER_AREA)# downscale
  data.append(resized)

data = np.asarray(data)
random.shuffle(data)

x_train = data[:int(0.9*len(data))]
x_test = data[int(0.9*len(data)):]

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

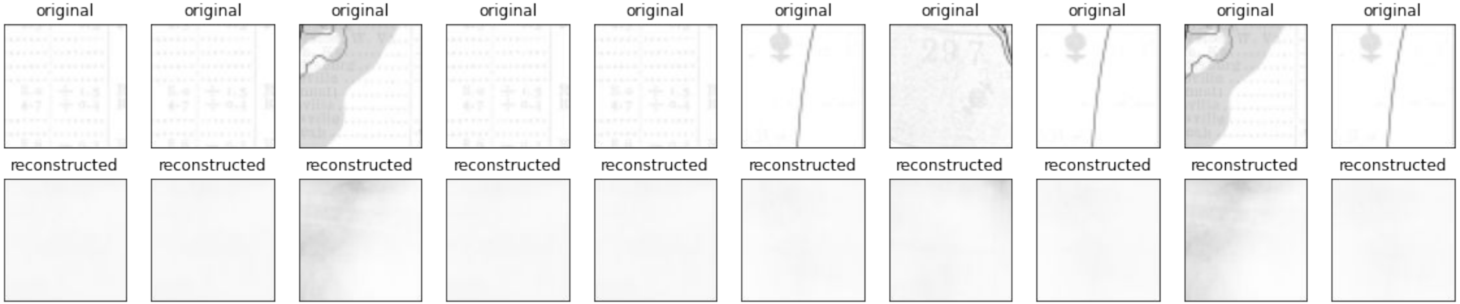
```
(5724, 112, 112)
(636, 112, 112)
```

```
[7] autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

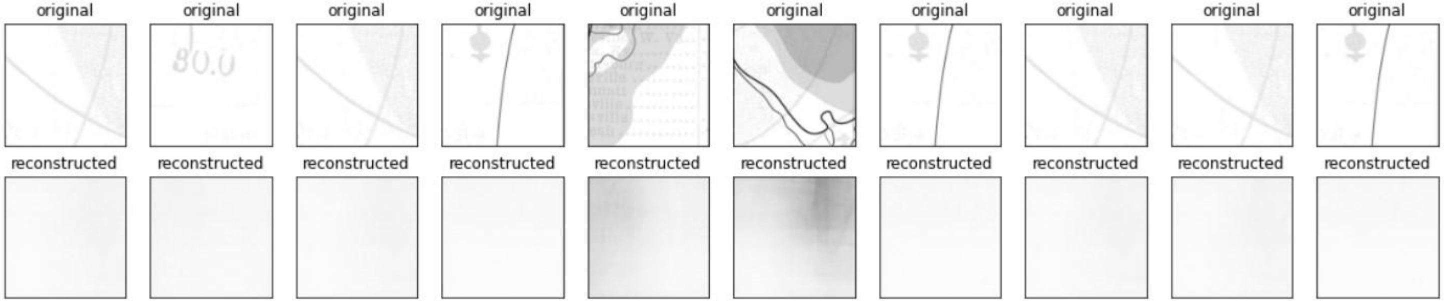
Train the model using `x_train` as both the input and the target. The `encoder` will learn to compress the dataset from 784 dimensions to the latent space, and the `decoder` will learn to reconstruct the original images. .

```
autoencoder.fit(x_train, x_train,
                epochs=4000,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```python
# plot both original and generated

n = m = 5
plt.figure(figsize=(20, 4))
for i in range(n):
  # display original
  ax = plt.subplot(2, n, i + 1)
  #img1 = cv2.cvtColor(x_test[n], cv2.COLOR_GRAY2RGB)
  cv2.imwrite(f'x/original/{i}.jpg', img1)
  plt.imshow(img1)
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, n, i + 1 + n)

  img2 = cv2.cvtColor(generated_imgs[i], cv2.COLOR_GRAY2RGB)
  cv2.imwrite(f'x/target/{i}.jpg', img2)
  plt.imshow(img2)
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

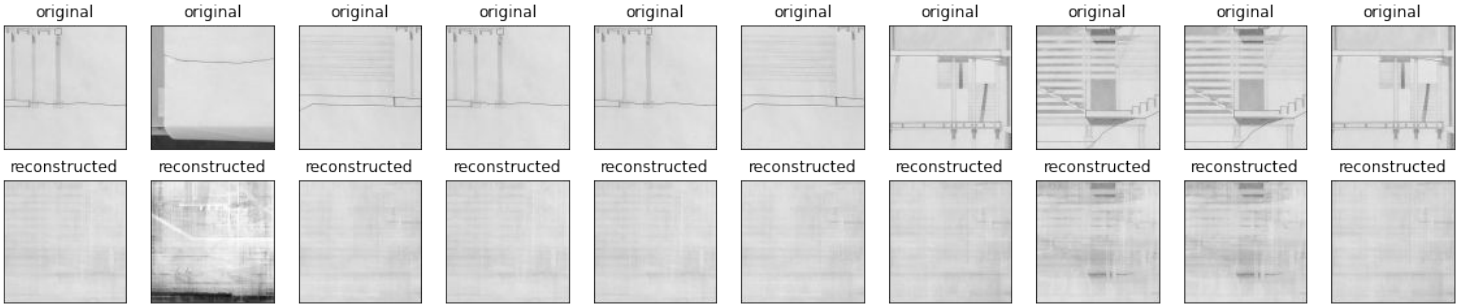


Mu = 1
sigma = 2

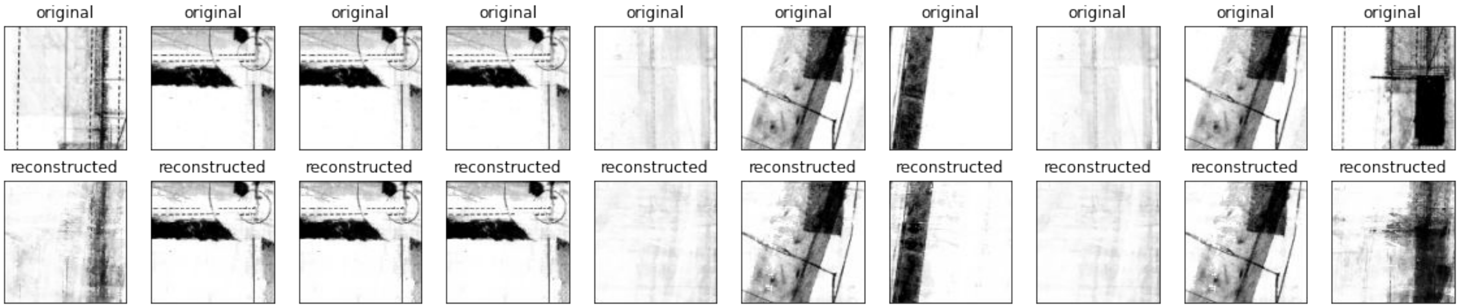


Mu = 1
sigma = 2

Test with smaller dataset



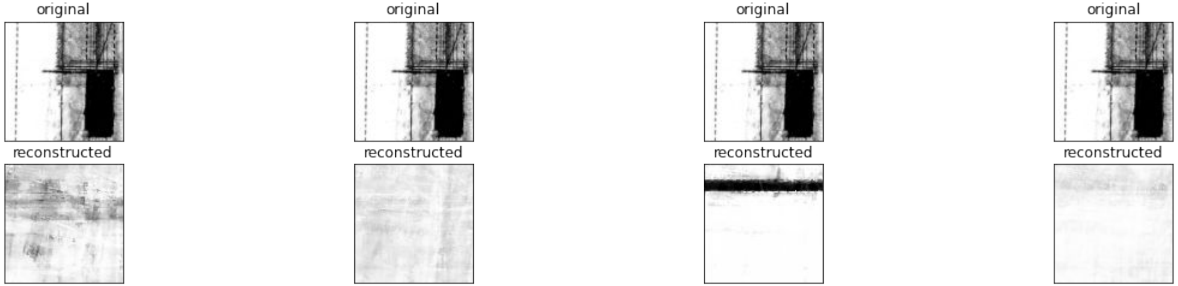
Mu = 1
sigma = 2



Mu = 1
sigma = 4

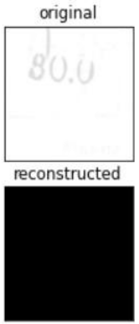
Results

Mu = 1
sigma = 2

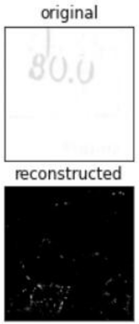


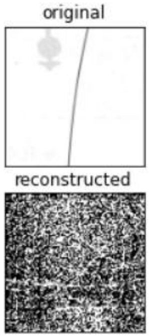
Mu = 1
sigma = 4

Results

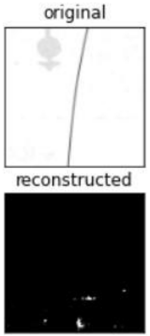


Mu = 1
sigma = 2





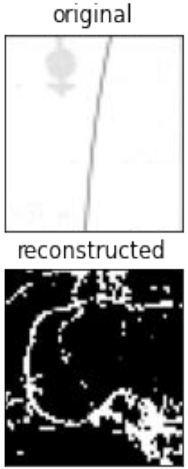
Mu = 1
sigma = 2



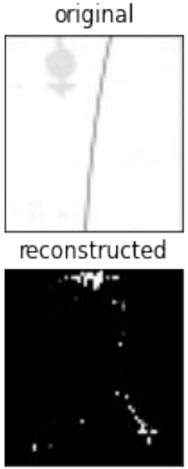
```python
import numpy as np
mylatents = []
mu = 12 # you can play with this
sigma = 6 # you can play with this
m = len(encoded_imgs)
for i in range(m):
  mylatents.append(encoded_imgs[i] + np.random.lognormal(mu, sigma, latent_dim)) # you can play with this

mylatents = np.asarray(mylatents)

generated_imgs = autoencoder.decoder(mylatents).numpy()
```

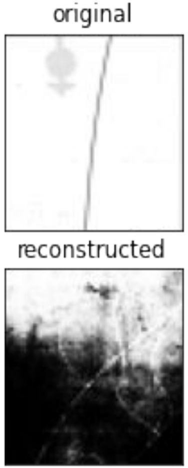
```python
encoded_imgs_train = autoencoder.encoder(x_train[0:10]).numpy()
decoded_imgs_train = autoencoder.decoder(encoded_imgs_train).numpy()

r = 10
plt.figure(figsize=(20, 4))
for i in range(r):
  # display original
  ax = plt.subplot(2, r, i + 1)
  img1 = cv2.cvtColor(x_train[i], cv2.COLOR_GRAY2RGB)
  plt.imshow(img1)
  plt.title("original")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, r, i + 1 + r)
  img2 = cv2.cvtColor(decoded_imgs_train[i], cv2.COLOR_GRAY2RGB)
  plt.imshow(img2)
  plt.title("reconstructed")
  plt.gray()
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```

Test 1

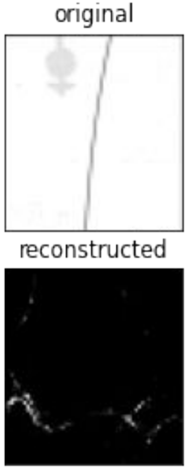


Mu = 0
sigma = 1

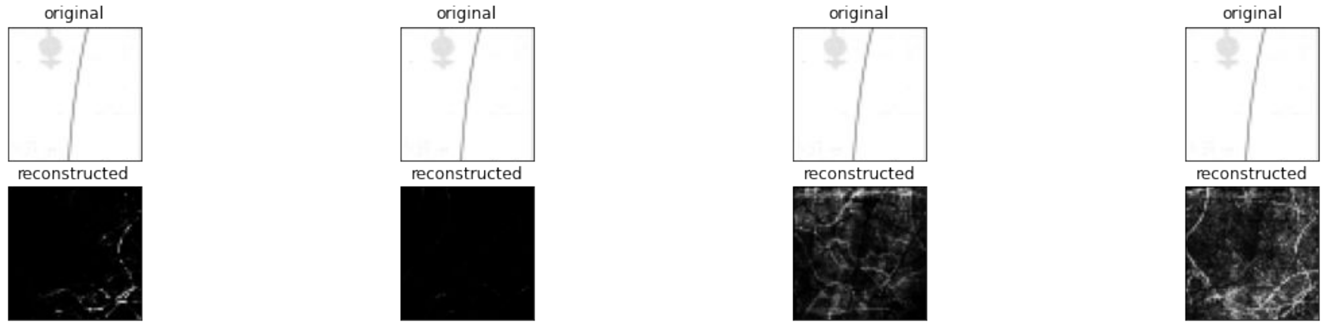




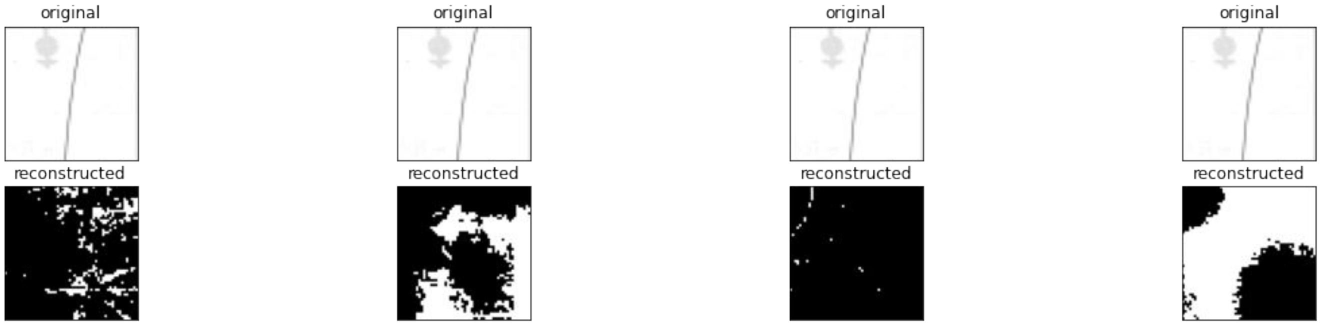
Mu = 1
sigma = 2



Test 2

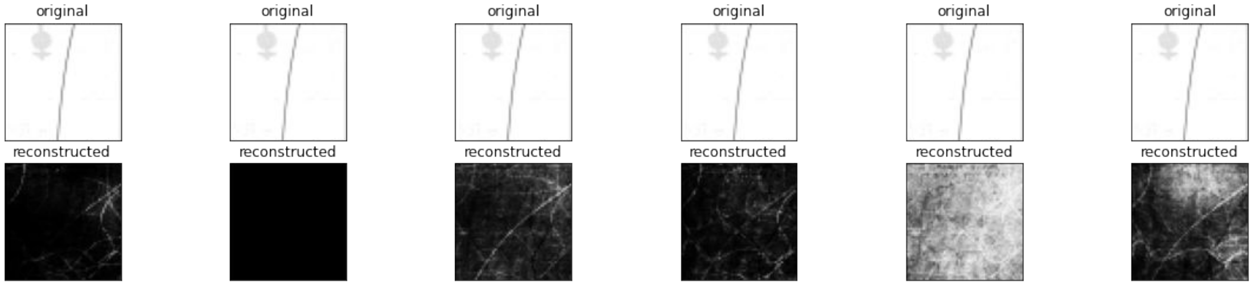


Mu = 0
sigma = 0.5

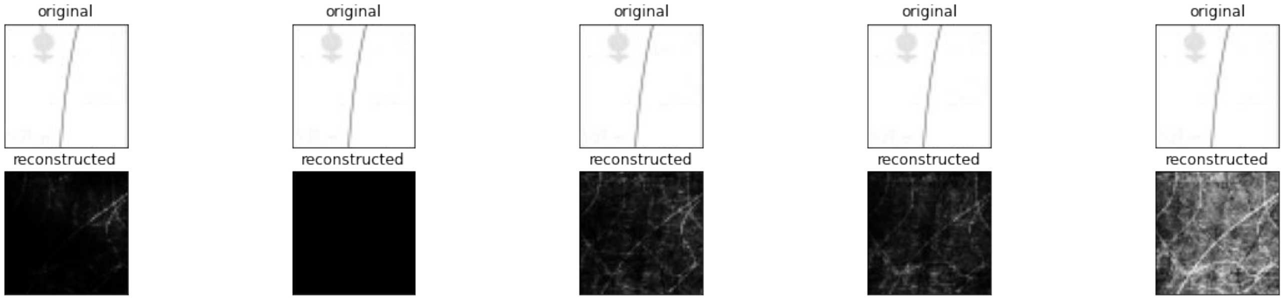


Mu = 1
sigma = 1

Test 3



Mu = 2
sigma = 2



Mu = 0.25
sigma = 0.25

Conclusion

While the output images lacked the detail and usefulness we were hoping for, the resulting images were beginning to pick up on linework and boundary conditions. This means that the autoencoder and sampling process was working, albeit in a limited manner.

Conclusion

The outputs were not new generations of student work because the AE did not yet fully identify the characteristics that define this work. In order to gain better results, we need to increase the number of training images and specify a higher number of epochs.

Future developments

- Use vector lines from CAD as dataset or output as opposed to pixels
- Expand dataset including more years, studio levels or image types
- Explore StyleGAN into generating drawing style applications to a target image

References

Badr, W. (2019). *Auto-Encoder: What is It? And What is It Used For?*. Towards Data Science. Retrieved from https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

Bandyopadhyay, H. (2022). *An Introduction to Autoencoders: Everything You Need to Know*. V7 Labs. Retrieved from https://www.v7labs.com/blog/autoencoders-guide

Chaillou, S. (2019). *AI & Architecture. An Experimental Perspective*. Towards Data Science. Retrieved from https://towardsdatascience.com/ai-architecture-f9d78c6958e0

Lopez de Mantaras, R. (2017). *Artificial Intelligence and the Arts: Toward Computational Creativity*. BBVA OpenMind. Retrieved from https://www.bbvaopenmind.com/en/articles/artificial-intelligence-and-the-arts-toward-computational-creativity/