Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) Applied to MNIST datasets

A Major Qualifying Project report:

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the degree of

Bachelor of Science

By

Jacob Wilusz

Date: August 9 2024

Approved:
Professor George Heineman, Major Advisor

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects.

Table of Contents

Abstract

Two of the earliest forms of generative machine learning algorithms are Generative Adversarial Networks (GAN) and Variational Autoencoders (VAE). Despite being developed at approximately the same time, their functionality and structure are markedly different. This project aims to compare these two networks when trained on two different datasets and analyze the differences in their output and runtime in reference to their structure.

Introduction

Generative Artificial Intelligence (AI) is an incredibly volatile subject in the current technological sphere, with constant debates over its potential uses and whether or not such uses are ethically or morally correct. Due to technological innovations, multiple different types of generative AI have been developed, with their own strengths and weaknesses. One of the first major types of generative AI networks developed was the Generative Adversarial Network, or GAN, which is used for image generation (Goodfellow *et al.*, 2014). While there have been numerous different offshoots of GAN models, the core structure of the network remains largely unchanged. Shortly thereafter, a different type of generative AI network, the Variational Autoencoder, or VAE, was developed (Bernstein, 2023). It had a similar basic framework to a GAN, but functioned completely differently from its predecessor. One of the major goals of this project is to examine both a Generative Adversarial Network and a Variational Autoencoder to gauge their effectiveness when compared with one another.

To do this, this project aims to focus on two domains of interest well-known in the Machine Learning (ML) literature. The first dataset of interest is the Modified National Institute of Standards and Technology dataset, or MNIST dataset (LeCun *et al*., 1998), which is composed of images of handwritten digits from zero (0) through nine (9), and developed in 1994 by National Institute of Standards and Technology (NIST), from which the dataset gets its name. However, much like the GAN, the MNIST database has seen improvement over time, and the NIST developed and released an extended dataset in 2017, fittingly called the Extended Modified National Institute of Standards and Technology dataset, or EMNIST (Cohen *et al*., 2017). In addition to the 70,000 images contained in the original dataset, the EMNIST dataset contains 814,255 images including both handwritten letters (in both uppercase and lowercase) as well as handwritten digits. The other major goal of this project is to train both networks mentioned above on both datasets to both attempt to replicate the results of earlier studies and to compare their effectiveness on different datasets.

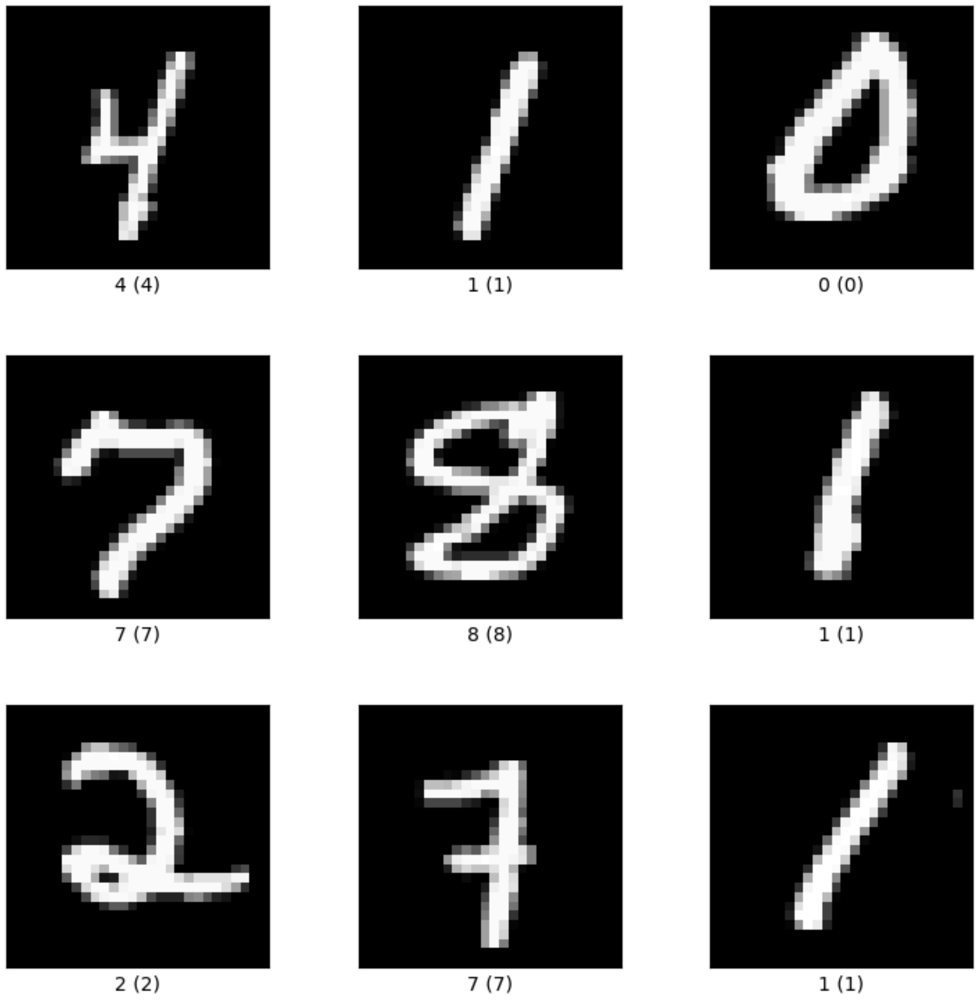


Figure 1 - Examples of images from the MNIST dataset.

The second goal is to demonstrate how to retrain the model using additional data, which is included in an enhanced EMNIST data set that contains uppercase handwritten letters. Retraining the models is necessary to include the extra training data included in the EMNIST dataset, as without it, the models will not be able to parse or generate data outside the original MNIST dataset. Both GANs and VAEs will be retrained and we will report the results of the updated models on both the original validation set as well as additional ones. This means you will validate on the original validation set that contains just digits as well as an enhanced validation set that also includes uppercase letters. In order to reduce the time to train models, it was decided to use the EMNIST-balanced subset of the dataset, which only contains 47 unique letters and digits instead of the full 62, as part of this second effort.

	MNIST Dataset	EMNIST Dataset
GAN Network	GAN on MNIST	GAN on EMNIST
VAE Network	VAE on MNIST	VAE on EMNIST

Table 1: Comparisons for the project

The structure of this project is shown in **Table 1**. There are four quadrants in this comparison.

We will report our success in replicating the success rate in applying GANs on the MNIST handwritten digit set. This is depicted in the upper left quadrant of the figure.

We will report our success in applying Variational Encoders on the same data set, and compare these results against the GAN. This is depicted in the lower left quadrant of the figure.

We will retrain the GAN by including the additional EMNIST handwritten uppercase letters. We will report the success in replicating the results on the handwritten validation set that still only includes handwritten digits. But then we will also report success on this newly retrained model on a validation set that includes a mix of digits and upper case letters. This is depicted in the upper right quadrant of the figure.

We will retrain the VAE by including the additional EMNIST handwritten uppercase letters. We will report the success in replicating the results on the handwritten validation set that still only includes handwritten digits. But then we will also report success on this newly retrained model on a validation set that includes a mix of digits and upper case letters. This is depicted in the lower right quadrant of the figure.

This structured experiment is designed to see the impact on both GANs and VAEs when retraining models to include additional data.

Background

For this project, we will be using the Neural Network structures of GANs and VAEs, and the datasets of MNIST and EMNIST. In this section, we will examine each of these components of the project in detail.

Domain Data Sets

For this project the domain of interest is handwritten digits and letters. One of the most commonly used data sets is MNIST. This data set contains 60,000 training images of handwritten numerals, each of which is a 28x28 grayscale image. There are an additional 10,000 images in the data set used for testing kept separately, resulting in a total of 70,000 images.

In addition, there is also the Extended MNIST dataset, also commonly referred to as the EMNIST dataset. Developed in 2017, this dataset expands on the original dataset by adding images of handwritten letters in addition to numbers, including a total of over 800,000 images of handwritten characters. While the MNIST dataset was, and largely still is used as the basis for artificial intelligence algorithm training and categorization, the EMNIST dataset not only expands upon the size of the original MNIST dataset, but also includes new training and testing data and categories for alphabetical characters in addition to numerals. In addition, the EMNIST dataset contains multiple formats of data ranging from a copy of the original MNIST dataset to datasets that include only alphabetical characters, only numerals, and other combinations of categories.

Machine Learning

Machine Learning, commonly abbreviated as ML, is the general term for studying and developing programs or algorithms that can improve their precision and output over time, or to perform tasks without explicit instructions. While Machine Learning and Artificial Intelligence are generally seen as interchangeable terms, it is more accurate that AI is a subsection of Machine Learning.

Neural Networks/Supervised Learning

A neural network is a computational structure that appears to mimic some of the internal structures found in human brains. The goal of the structure is to define an output given inputs. In the neural network in **Figure 2**, all information flows in one direction, that is forward.

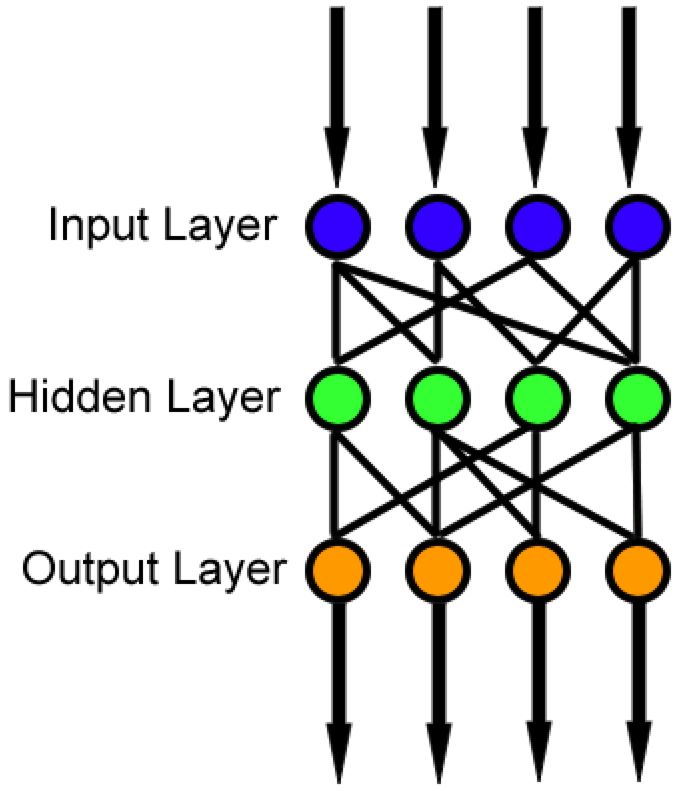


Figure 2: Basic structure of a Neural Network

A supervised learning algorithm constructs a mathematical model of a set of training data that contains the input data and there is descriptive information about the desired output. Using a standardized iterative approach, a supervised learning algorithm "learns" a function, **f(data$_i$)** that can be used to predict the output given new input data.

To give a human example, given **Table 2** below, one can infer the desired output for the new data input. The training set is contained in the first two rows. Based on this information, one could infer that the function **$f(x) = x^2$** and the answer would be 49. However it could also be **$f(x) = 6x - 5$** and the answer would then be 37.

Input	Output
1	1
5	25
7	???

Table 2: Sample Input and Output data for an unknown function, f(x)

Supervised learning algorithms are effective because they consume a very large data set of input information, for which all outputs are known. When more data is available, the accuracy of the learned function improves.

Because of this, algorithms can be trained to have a variety of functions, such as autocomplete functions common in word processor programs or on a cell phone, or image and text generation programs that have become more widely used in the current era. And as these networks and their functions improve, their accuracy and precision improve with them. For instance, older generation generative AI (Cao *et al*., 2023) pales in comparison to programs like OpenAI's ChatGPT.

Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a feed-forward neural network that learns features by itself via filter optimization (IBM, 2017). It is currently one of the most used neural network structures today, thanks in large part due to its ease of application to image classification and generation tasks thanks to its structure.

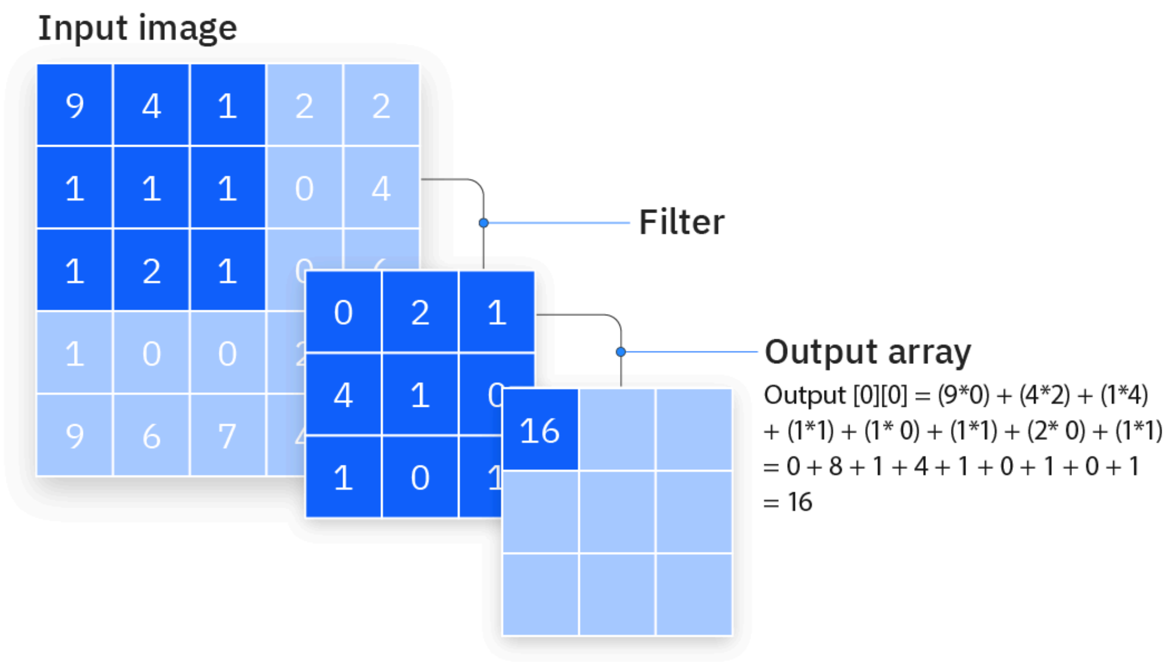


Figure 3: Example of a CNN, from IBM, 2017

CNNs, or Convolutional Neural Networks, serve as the basis for many of the early and some of the more modern AI algorithms. They serve as one of the core concepts of Generative Adversarial Networks, or GANs, by using filter optimization on the generator's output and the discriminator's judgment criteria. A CNN contains convolution layers, or layers that use the process of convolution to process inputs into outputs. A convolution is defined by using a convolution window of specified size to read data, which a filter is then applied to, and then mapped to an output. The filters can adjust their own values based on the output of the layer, hence they are critically important for machine learning algorithms as they can improve their outputs over time.

CNNs are typically divided into three types of layers: a convolution layer, where most colvolutions performed by the network are done, a pooling layer, which accumulates and compiles the results and values used in the convolution layer, and the fully-connected layer, which collects and generates the final output of the network. A typical CNN is comprised of several convolution and pooling layers and one fully-connected layer to produce the output. Part of what makes CNNs so versatile is their ability to be custom-built and custom-trained for many different tasks, and it allowed them to become one of the cornerstones of machine learning networks.

Generative Adversarial Networks (GANs)

A General Adversarial Network, or GAN, is one of the cornerstones of generative AI networks (Goodfellow *et al*., 2014). First conceptualized and developed around 2012, a GAN works by using two neural networks, a generator and a discriminator, and pitting them against each other, as shown in **Figure 4**. The generator, which is shown images, text, or data from a training set, learns to generate new data with the same statistics as the training set.

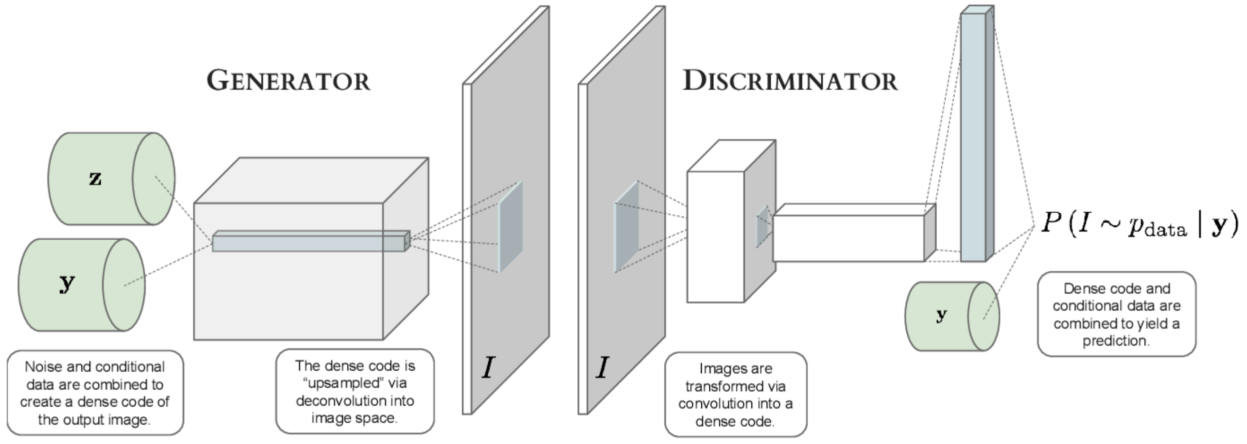

GENERATOR
DISCRIMINATOR
z
y
I
I
y
$P(I \sim p_{\text{data}} \mid \mathbf{y})$
Noise and conditional data are combined to create a dense code of the output image.
The dense code is "upsampled" via deconvolution into image space.
Images are transformed via convolution into a dense code.
Dense code and conditional data are combined to yield a prediction.

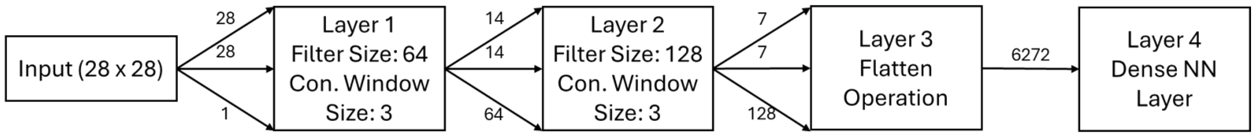
Figure 4: A diagram displaying the structure of both networks involved in the GAN.

In most machine learning applications, the goal is to optimize a function, but in GAN the generator aims to "fool" the discriminator into believing that the generated output is legitimate.

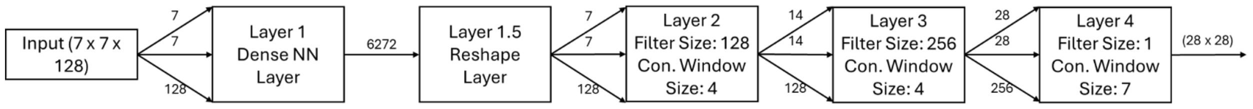
Once the generator can consistently create output that the discriminator can classify as legitimate, the discriminator is removed from the system, and the generator provides its output to the user. GANs were one of the first major generative AI networks, and while they had mixed results at their inception, as they have improved and innovated over time their results have become more and more promising.

GAN First Attempt

TensorFlow (TensorFlow, 2024b) provides functions that are used to train over the data (and read it) and produces a minimal structure of a final potential model, which is completed by Keras (Keras Team, 2024a).



The GAN is set up with two networks - the discriminator and the generator. The discriminator's general structure is shown above, with the input of the 28x28 MNIST image fed into the discriminator, and then going through 4 layers before providing an output.



For the first working setup of the GAN, the discriminator and the generator were set up in a similar fashion. For the discriminator, a total of four layers were used: two convolutional layers, a flatten/funnel layer, and a dense NN layer. The way the convolutional layers work is by using a number of filters of a certain size to scan for features in input, and then adding each result from the input and the filter into a map or a matrix for the final output. For the first setup, there were two convolutional layers used, with a total of 64 and 128 filters in the respective layers, and both used a filter size of a 3x3 matrix. The output from those layers was then put into the flatten/funnel layer, in which reshapes the multi-dimensional output from the two convolution windows into a one-dimensional array/vector, which is then fed to the final layer, the dense NN layer. The dense layer processes the input vector produced by the above layers, and uses it to produce the final output with its own activation function.

For the generator, it is built in a similar fashion, although it could be said that it is built as the opposite of the discriminator, starting with a dense NN layer, then followed by a reshape layer and two transposed convolution layers. For the generator, the dense layer processes the input layer and creates a one-dimensional array/vector, which is then processed by the reshape layer into a multi-dimensional output as seen in the earlier layers of the discriminator. Then, these outputs are processed by two transposed convolutional layers, which performs the functions of the discriminator's convolutional layers in reverse, by taking the feature map, funneling through a filter and constructing a possible input. These two layers reshape and resize the provided multi-dimensional inputs before feeding them to a final standard convolution layer with one filter, which processes the inputs into a standard 28x28 image comparable to the MNIST/EMNIST dataset.

Variational Autoencoders (VAEs)

A Variational Autoencoder, or VAE, works by compressing data down to a smaller value via one encoding algorithm, and then uses another to decompress it to a similar, if not the same, value as the original.

When using these in generative models, VAEs will essentially create a sort of 2D plane that contains compressed data points for all the data given in its training set. Then, when the user requests something from that data set, it will pick out one or more points that should, probabilistically, produce that output, and decompresses the points into the proper output data.

While VAEs and GANs may have similar functionality with generating output, their methods in generating that output are fundamentally different. A GAN will start with countless outputs based on the training data, and will slowly filter out and refine them with the help of the discriminator, whereas a VAE picks a most likely output and builds from around it, expanding its list of possible outputs.

A VAE works by minimizing any potential loss/miscategorization of inputs and outputs, or in other words, making sure that the chance it produces the correct data point/output is as close to 100% as possible.

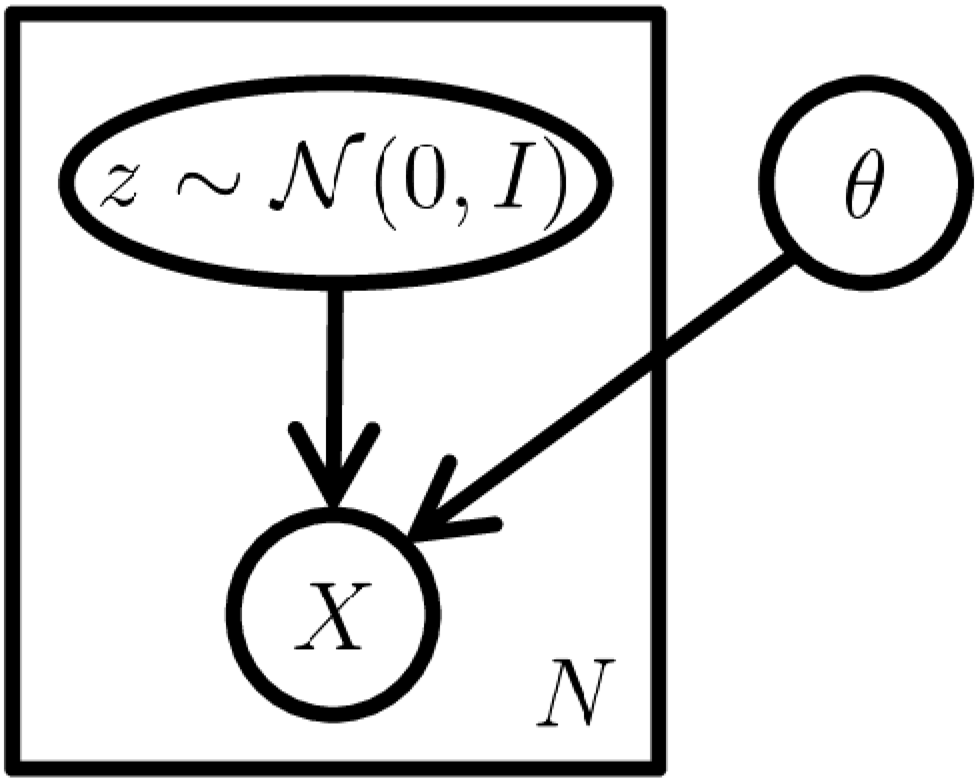
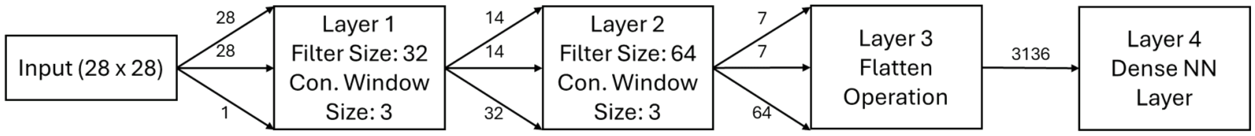


Figure 5: General encoding process of a VAE

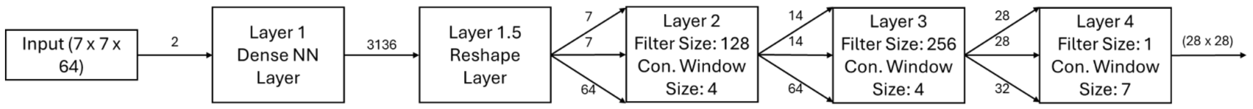
Figure 5 outlines the general encoding process of a VAE, of which the goal is to maximize the chosen probability distribution (in this case, z), with the given parameter theta. The goal of the encoder is to create a probability distribution such that the chance that the given data is encoded to specific values is maximized, and the goal of the decoder is to create a similar distribution such that the chance encoded values are decoded to specific data is maximized. In other words, the goal of a VAE is to create two parameterized probability distributions in which loss between the two is minimized.

First VAE structure attempt

For my first attempt at the VAE, I used a simple structure much like the GAN, using only about four layers for the encoder and decoder networks.



For the encoder, I had a simple structure composed of four layers: two convolutional layers, followed by a flatten/funnel layer, and then a dense NN layer to produce the output. For the first two layers, they contained 32 and 64 3x3 filter matrices respectively, which provided a multi-dimensional output to be reshaped into a one-dimensional vector by the flatten layer. Finally, the one-dimensional array was processed by the dense NN layer, providing a basis for the normalized distribution of encoded data.



For the decoder, I had a five-layer structure built as the opposite of the encoder, starting with a dense NN layer to process the encoded data, followed by a reshape layer to transform it into a multi-dimensional output, and followed by three transposed convolutional layers to process and convert the data into a standardized 28x28 image.

Methodology

Using Tensorflow and Keras

In order to create the networks to be used in this project, there were two major data packages used, Keras and Tensorflow. The purpose of the Keras data package was to provide the framework for the layers of the networks, allowing them to actually function as needed for this project, while the Tensorflow package was used to not only provide the structure for the network, but also to provide the datasets used in this project, namely through the tf_datasets package (Tensorflow, 2023).

Using both of these data packages allowed for the creation of both a functional GAN and VAE network, which were built using two online tutorials from the Keras website as a basis.

The GAN’s general structure was based off of a tutorial that involved the generation of images of faces from a local storage dataset (Keras Team, 2023). This tutorial had to be modified extensively for it to fit our purposes, mainly in restructuring the code to pull from datasets provided by the tfds_datasets package. However, its foundation for the GAN it created was largely unchanged, and thus the final product operated very similarly to the tutorial, and did not affect the runtime of the model.

For the VAE, the tutorial that was used was originally using the MNIST dataset, so it was remarkably easy to make minor changes for the code to fit the needs of the project. However, much like the GAN tutorial, the VAE tutorial code had to be modified in order to make use of the `tfds_datasets` package, but this did not affect the functionality or runtime of the model.

Setting up GAN

In order to set up the GAN, the first step was to use the original tutorial code and modify it for our purposes in using it on our datasets. In the tutorial provided by Keras, the GAN was run on a database in local storage which contained a collection of face images for training and testing. However, since we would be training and testing on the MNIST/EMNIST datasets from first Keras, then the tfds_datasets packages, it would be necessary to modify the code provided by the tutorial. After successfully altering and checking the code to ensure it still functioned as needed, the first task was running the GAN on the MNIST dataset.
GAN and VAE tutorials are written by the same people, but have different styles of coding.

In the first implementation, each epoch required 360 seconds to compute and the results are not as accurate. When running the GAN, it seemed to work well enough, however, it ended up taking 3 hours to train, requiring 6 minutes per epoch of training. After comparing it to examples of GANs and other network types, I adjusted the number of filters for the layers, changing the numbers from 64 and 128 to 32 and 64, which reduced the training time of the network drastically, now only needing 6 minutes to train with roughly 30 seconds per epoch of training.

After realizing that my current method of reading in data from the datasets would not be effective for reading and training the EMNIST data, I reworked my currently existing code to use the tfds_datasets package, which includes both MNIST and EMNIST. However, as a result, I also needed to rework the way my code functioned for the baseline MNIST to be compatible with my new tfds_datasets methods and structure.

Setting up VAE

The tutorial code (Keras Team, 2024b) used the MNIST data set and TensorFlow and Keras. The first step was to modify the code to work with EMNIST data set. This was not hard to do because Keras already supported the MNIST data set. Running the modified code generates a visual representation of the encoded data for the MNIST digits

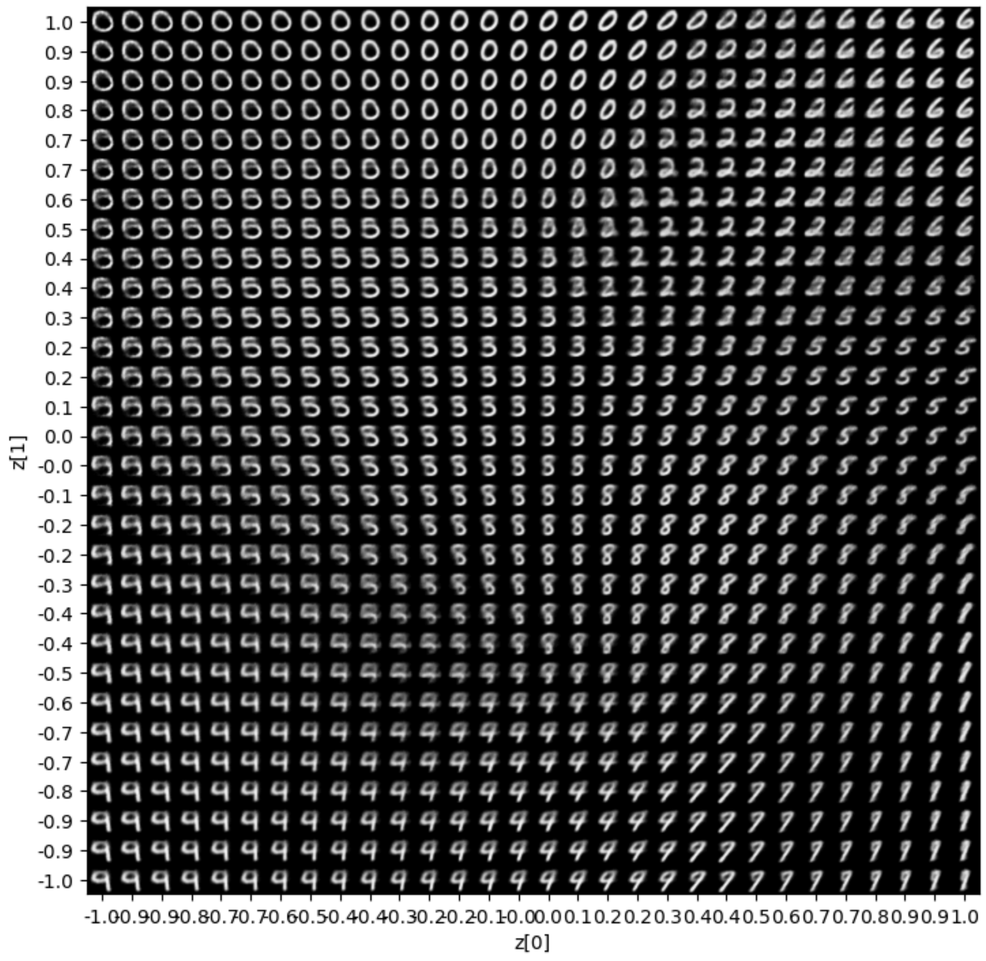


Figure 6: Visual representation of encoded data

In **Figure 6**, you can see all ten digits at various locations within the two-dimensional plane. There is smooth variation as you move between neighboring images, and so in the left edge of the image looking vertically down the image you can see the digit “0” smoothly transform into the digit “9”. The grid represents the output of the VAE at unique points in a 2D plane.

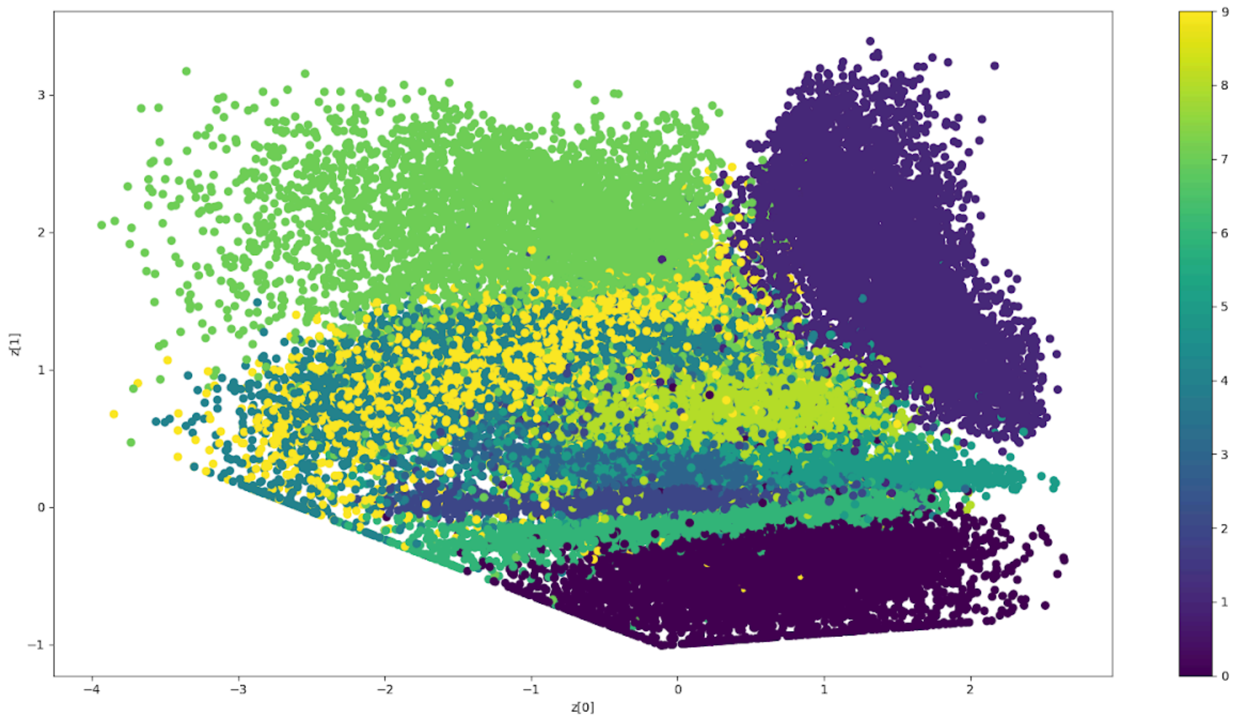


Figure 7: Sample spatial distribution of a VAE trained on MNIST.

When working to develop this code for the VAE, I initially had problems with the process of correlating the textual digit you wanted the VAE to generate with the image output provided by the algorithm. Since a VAE generates output based on probability rather than discrimination and categorization like a GAN, it was hard to directly associate the generated images with categories to pull from.

After getting the GAN and the VAE to function correctly for the MNIST dataset, I attempted testing the trained networks from the MNIST dataset on the EMNIST dataset for input. However, I encountered two distinct problems: First, the EMNIST dataset contains the MNIST dataset as a subset, meaning that the categories in MNIST are all present in EMNIST, but not the other way around. This results in not being able to fully convert categories from EMNIST to MNIST, and with my first implementation having no ways for the GAN to distinguish categories, it became very difficult to manage. The second problem was that while the setups through Keras and TensorFlow allowed trained models to predict their output, the corresponding input to predict from, particularly the GAN, is difficult to format and properly feed into the network. As a result, I had a lot of trouble properly formatting and differentiating the training and testing datasets for both networks.

Another major problem for testing false positives arose from realizing that the function to generate outputs from the GAN and VAE operated independently from the database they were trained on, as well as their inability to categorize inputs shown to them. In other words, if a network was trained on MNIST, then inputting a test image from EMNIST would not affect the output of the network than if I used a test image from MNIST. And since this is true for the GANs and the VAE trained on MNIST, as well as the fact the MNIST dataset is a subset of the EMNIST dataset, the only possibility for a false positive would be from a VAE trained on EMNIST. As a result, I started working on training the VAE and GAN on the EMNIST data set.

When initially loading the EMNIST data set, I couldn’t find a dedicated dataset loaded into Keras to pull from. As a result, I attempted to use the database downloaded from the internet onto my local storage to use the EMNIST database. However, after running into too many complications with importing EMNIST from my local storage, I instead turned to using the tensorflow_datasets package, which includes both MNIST and EMNIST. After reformatting my current code, I was able to successfully run EMNIST, but since EMNIST is markedly larger than the MNIST dataset, containing over 800,000 images as opposed to MNIST's 70,000, I specifically ended up using the "balanced" subset of EMNIST, which contains approximately 131,600 images.

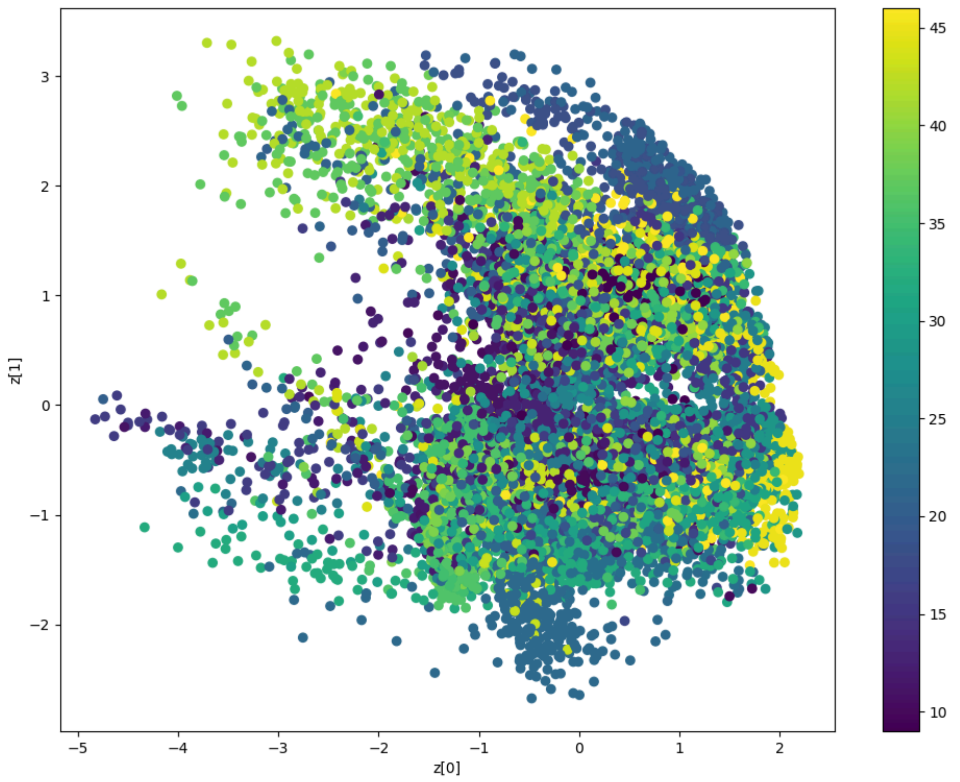


Figure 8: Sample spatial distribution of only the letters in the EMNIST dataset when run through a VAE.

In addition, in order to test the quality of my code and debug it, I included a function to display only certain digits and characters when plotting a sample distribution, as seen in the figure below.

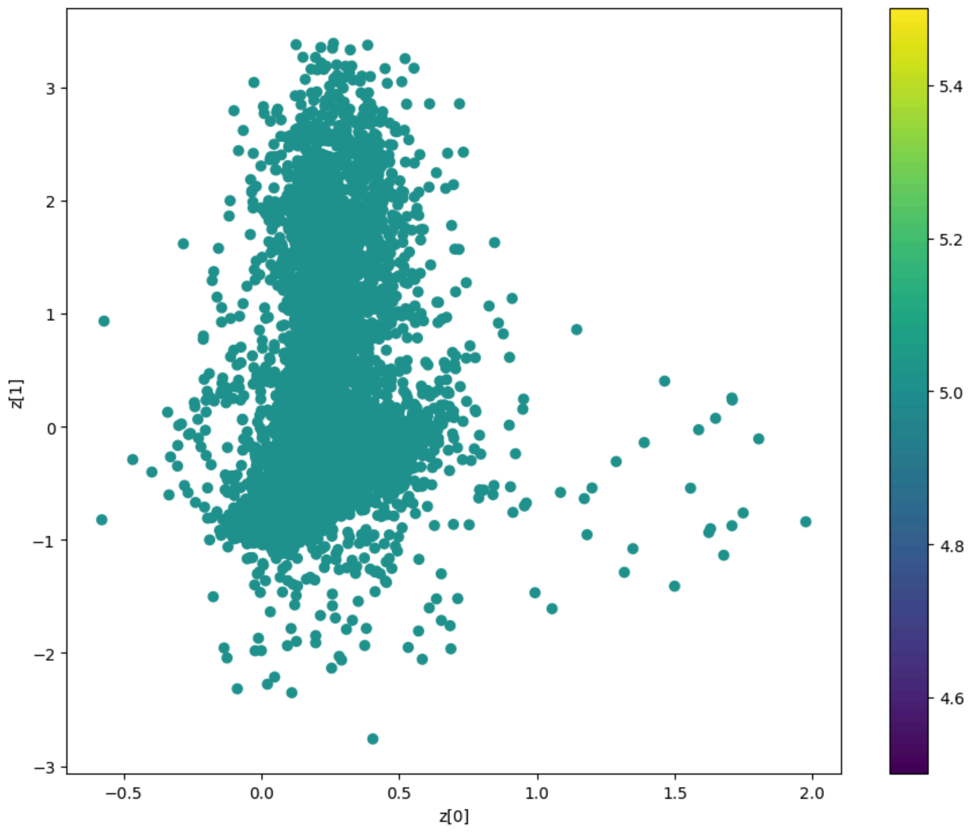


Figure 9: Values that align with "5" as can be seen on the Y axis.

Design

For the final setup of the networks, I used Keras, Tensorflow, and the tfds_datasets packages to provide the structure of the networks and datasets. As an open-source machine learning platform, Tensorflow provided the foundation and framework of the two networks that were created, and provided built-in methods for image training and image generation. Once the foundation and framework were in place, Keras was used to create the individual layers of the networks and facilitate interaction between the user and network. With the functions and code provided by the Keras package, each layer of the network could be fully built and trained. Lastly, the tfds_datasets package was used to provide the datasets trained and tested on. While the Keras package does contain an MNIST dataset that was initially used, for the sake of convenience, the tfds_datasets package was used instead, as it contained both MNIST and EMNIST datasets.

Both the GAN and the VAE were developed with similar structure, having 4-5 layers for each component of the network with the same window size, filter size, and number of filters. While the general structure of the network and its layers remained very similar to the original tutorials, only containing ~5 layers and stacking convolution layers before funneling them into a dense NN layer, the final structure changed the setup of the filters in both their size and number per layer.

For the datasets used, both MNIST and EMNIST datasets were accessed using the tfds_datasets package, and the subset of EMNIST used was the EMNIST_Balanced subset. For the MNIST dataset, both networks were trained on 60,000 training images and tested on 10,000 testing images consisting of digits from 0-9, while for the EMNIST dataset, both networks were trained on 112,800 images and tested on 18,800, including all digits from 0-9 and all letters, although due to problems in recognition due to similarities between uppercase and lowercase, both versions of the letters C, I, J, K, L, M, O, P, S, U, V, W, X, Y, and Z have been merged into one class. Therefore, the EMNIST_Balanced dataset contains 47 classes, with 10 digits, 26 uppercase (and merged lowercase) letters, and 11 lowercase letters as separate classes, specifically a, b, d, e, f, g, h, n, q, r and t (Cohen *et al*., 2017).

Evaluation

To evaluate the effectiveness of these algorithms and the datasets, we will compare them and their results against each other and previously found results from earlier research.

GAN trained on MNIST

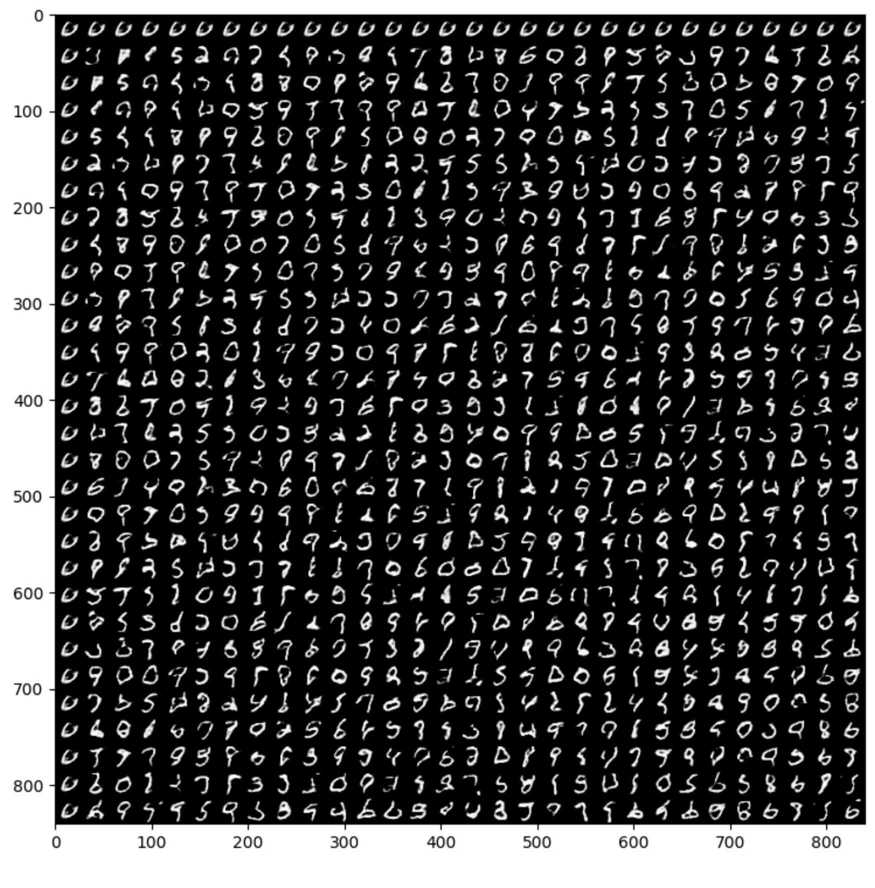


Figure 10: Randomly Generated output from the GAN for MNIST.

Figure 10 shows some randomly generated outputs from the GAN. It is worth noting that at least 2 of every digit between 0 and 9 are visible on this plot.

GAN trained on EMNIST

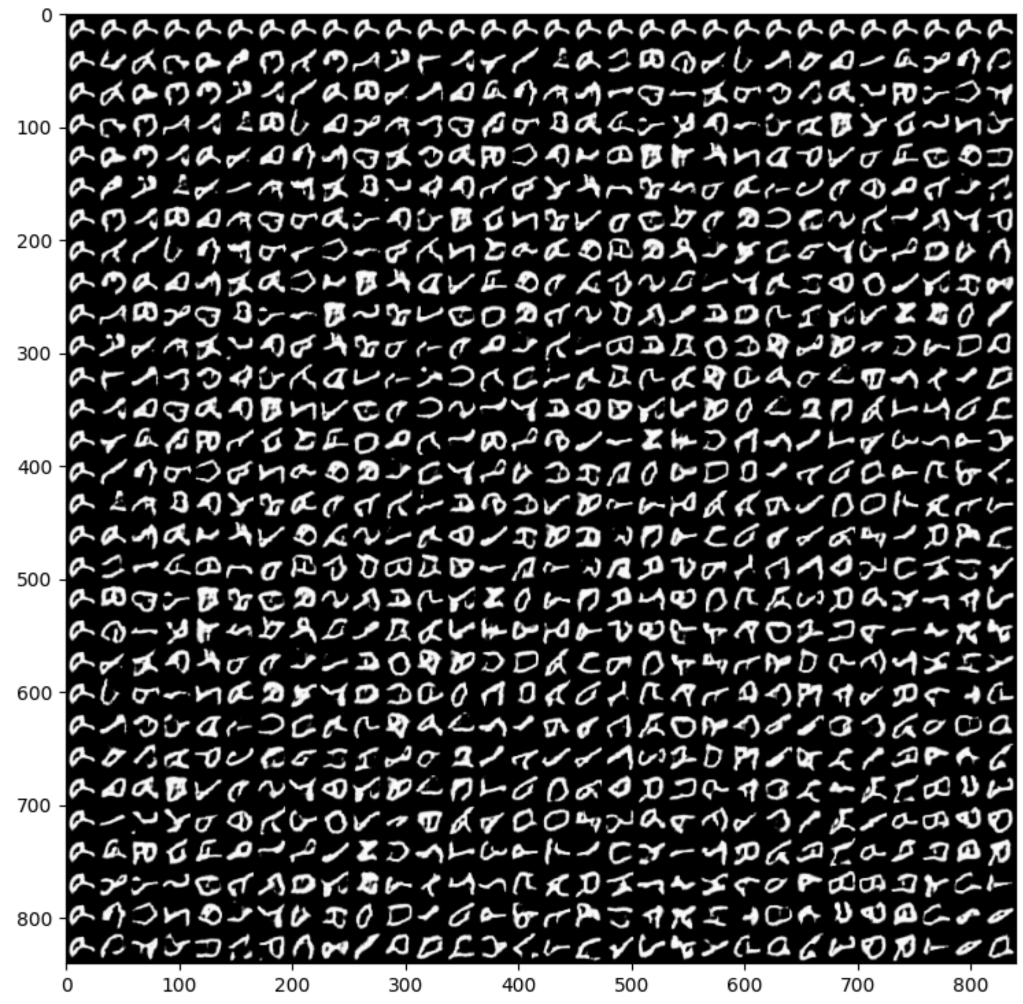


Figure 11: Randomly Generated output from the GAN for EMNIST

Note: the EMNIST dataset is set up such that training images are flipped horizontally and rotated 90 degrees counterclockwise. As such, the results output by the GAN presented as they would be normally seen, would look like this:

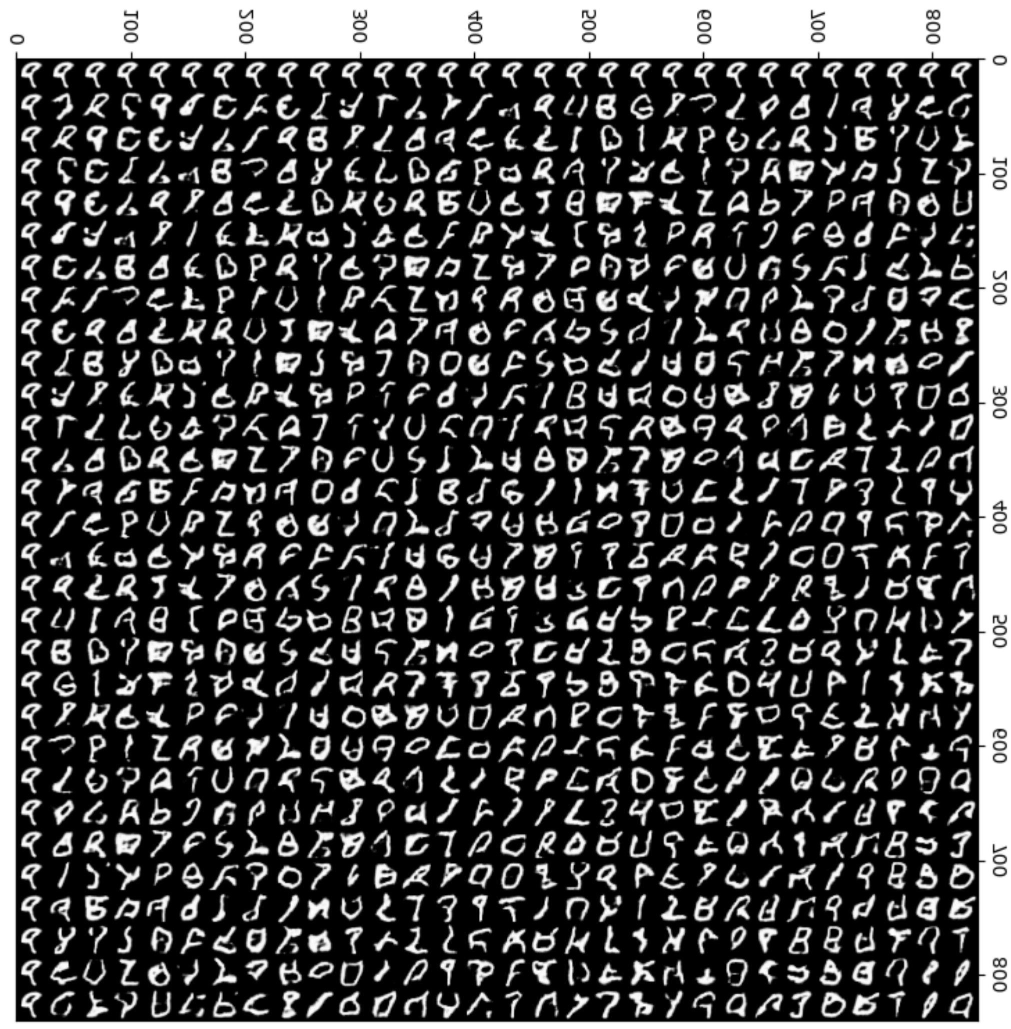


Figure 12: Rotated and Inverted images from **Figure 11**.

GAN output trained on EMNIST, reformatted. This plot shows several random images generated from the GAN trained on the EMNIST_Balanced dataset.

VAE trained on MNIST

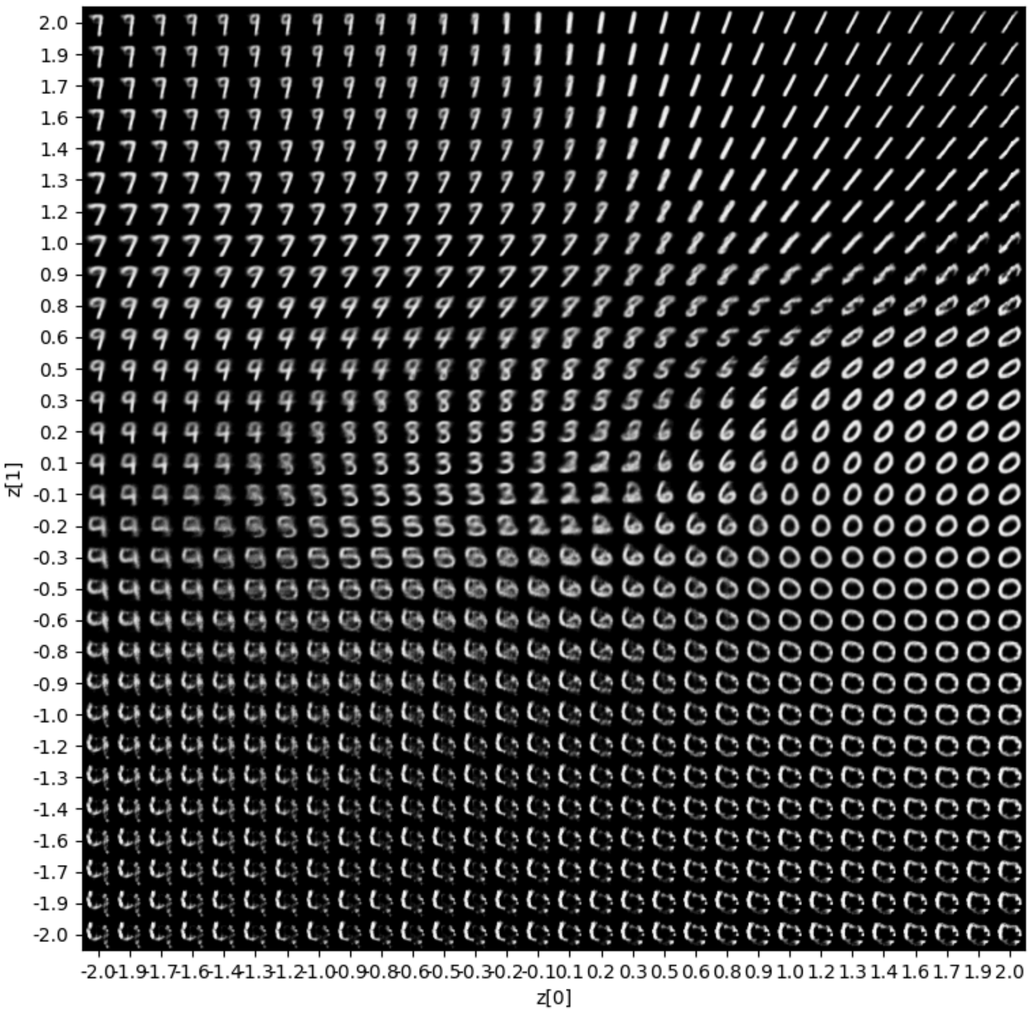


Figure 13: VAE's normalized distribution on the 2D plane.

Figure 13 shows the outputs of the VAE's normalized distribution on the 2D plane. As shown above, all digits in the MNIST database are shown.

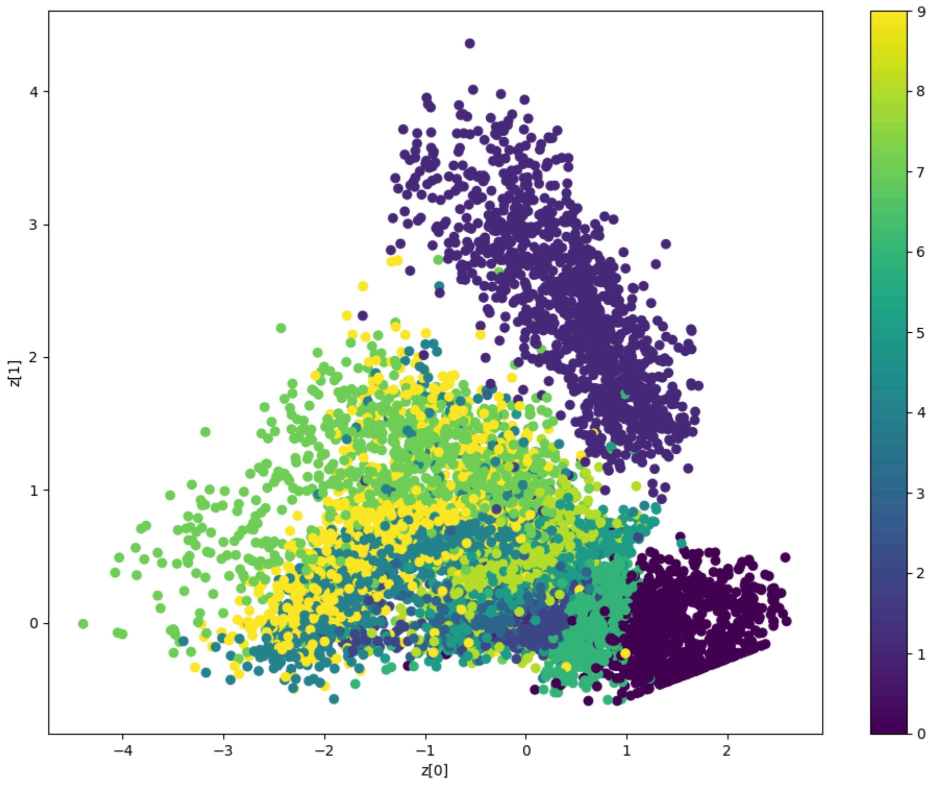


Figure 14: VAE's distribution of digits from VAE on MNIST

This plot shows the distribution of digits generated from the VAE trained on MNIST. The plot of the digits above is a visual representation of a section of this plot.

VAE trained on EMNIST

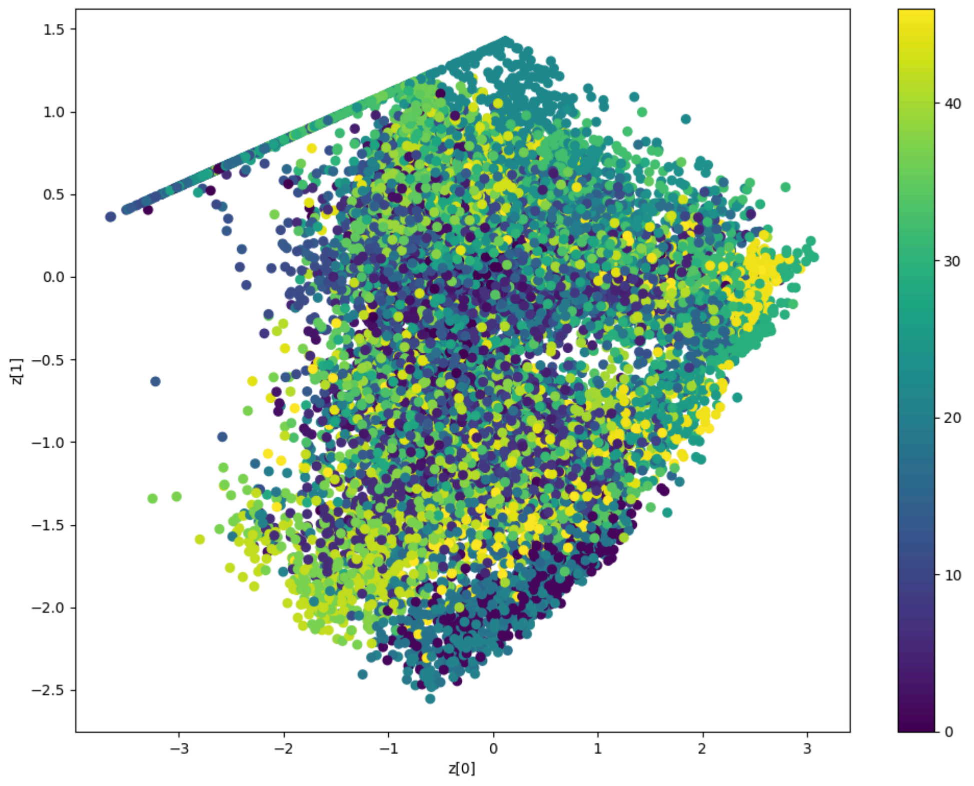


Figure 15 VAE's distribution of digits from VAE on EMNIST

This plot shows the distribution of digits and letters generated from the VAE trained on EMNIST. When compared to the MNIST plot, the data is much more centrally concentrated, meaning that the network considers most characters in the dataset to be similar to each other.

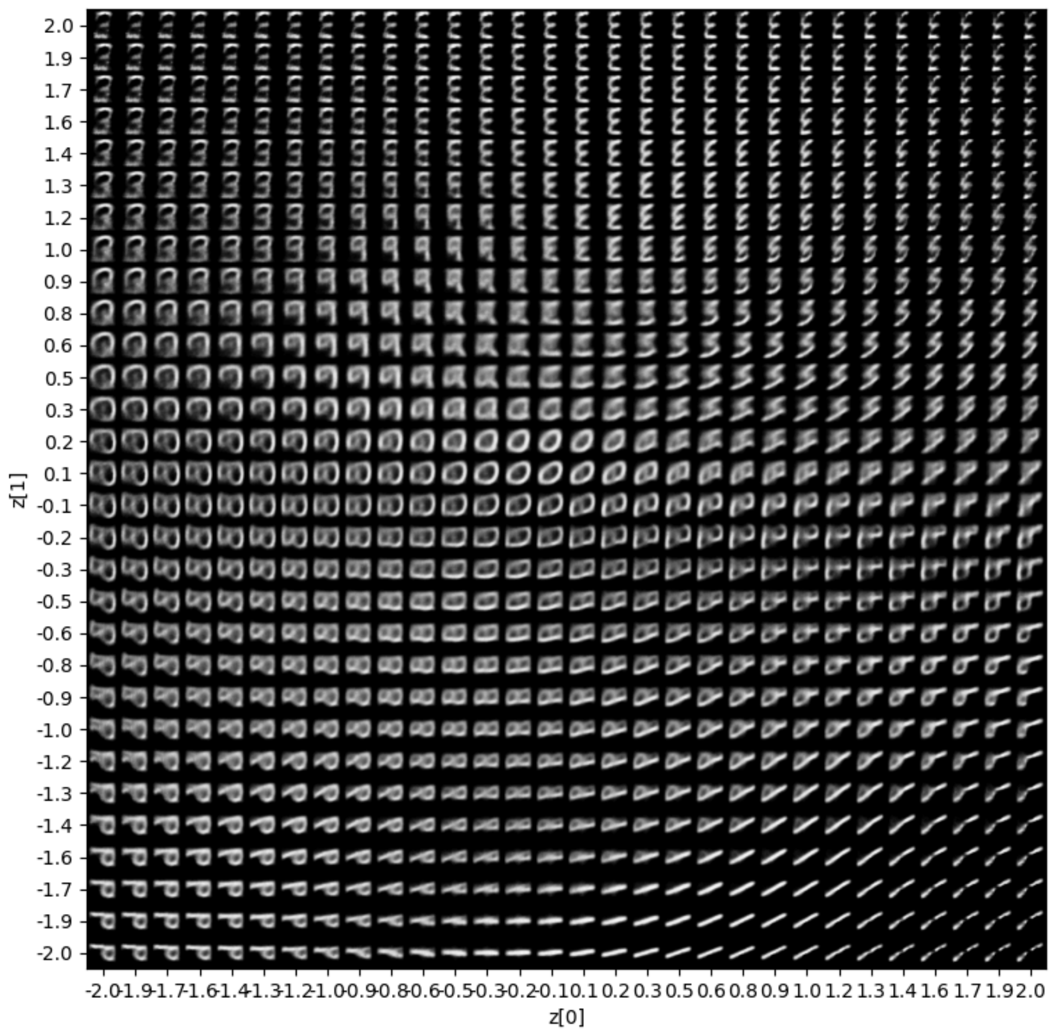


Figure 16 Visualization of central cluster from **Figure 15**.

This plot is a more visual representation of the central cluster shown in the above plot. However, since the EMNIST dataset is formatted to be rotated 90 degrees and horizontally inverted, the letters and digits should be more humanly recognizable in the below image:

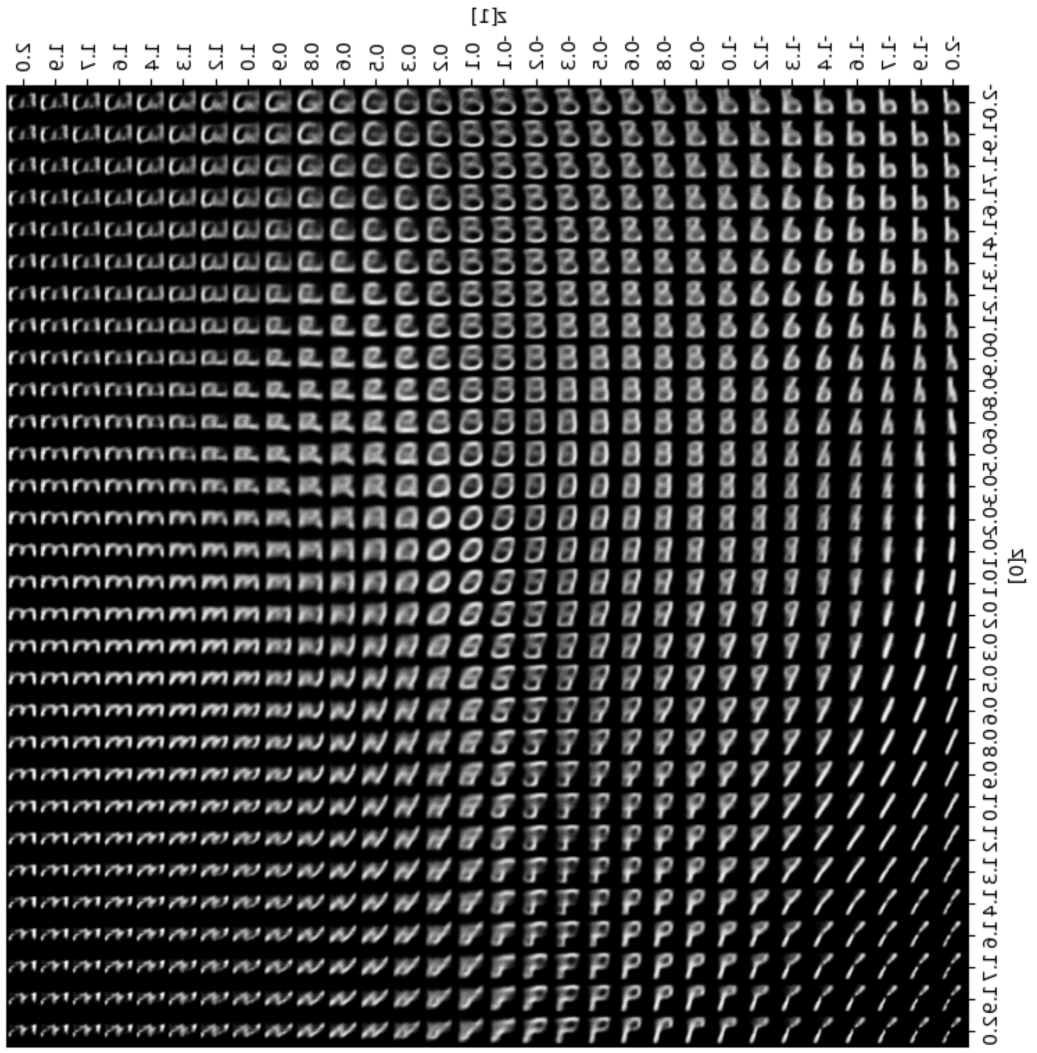


Figure 17. Rotation and Inversion of Figure 16 for readability.

Conclusions

In regards to the goals of this project, we were able to successfully replicate the generation of images from a neural network trained on the MNIST dataset, as well as the

EMNIST dataset. In comparison to other networks that were examined in the research process for this project (Goodfellow *et al*., 2014, Cao *et al*., 2023), the networks that were built were comparatively smaller, only comprised of about five layers, whereas the ones used in previous studies used more filters and layers in their construction. However, it can be argued that the outputs provided by the networks constructed in this project were comparable to the sample outputs provided by previous research, specifically the initial results of the GAN developed by Goodfellow. In addition, it is worth noting that while the VAE's results when trained on the MNIST dataset were exceptionally clear, most of the outputs of the VAE trained on EMNIST are more indiscernible, likely due to the difference in the amount of classifications present in the EMNIST dataset when compared to the MNIST dataset.

When comparing runtime for both algorithms, it is worth noting that the VAE was consistently faster in its training than the GAN on both datasets, with the GAN taking 30 to 35 seconds per epoch of training, while the VAE took 25 to 30 seconds per epoch of training. This is likely because the process of normalization for the encoder and decoder in the VAE is a much faster mathematical process than the training of the generator and discriminator in the GAN. Therefore, when compared to the generated results of both models, the GAN appears to perform better with a larger amount of categories to classify, like the EMNIST dataset despite taking longer to train, and the VAE appears to perform better on smaller amounts of categories to classify, namely the MNIST dataset. In addition, it is worth noting that running these models on my machine locally may have caused an increase in runtime. Since my machine's storage and processing power is limited, a more specialized machine may have been able to produce more efficient run times.

For possible future work, it is worth exploring how the number of layers and filters in a convolutional neural network can affect the quality of their output and their runtime when training. When first testing the models, a mismatch in the number of filters caused a massive increase in runtime. Although the training and adjustment of the larger number of filters increased the runtime of the model, it is worth exploring whether they have a noticeable effect on the output of the models.

References

Bernstein, M. N. (2023, March 14). *Variational autoencoders*. Variational Autoencoders - Matthew N. Bernstein. https://mbernste.github.io/posts/vae/#:~:text=Introduction,down%20to%20their%20intrinsic%20dimensionality

Bond-Taylor, S., Leach, A., Long, Y., & Willcocks, C. G. (2022). Deep generative modeling: A comparative review of vaes, Gans, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7327–7347. https://doi.org/10.1109/tpami.2021.3116668

Cao, Y., Li, S., Liu, Y., Yan, Z., Dai, Y., Yu, P. S., & Sun, L. (2023, March 7). *A comprehensive survey of AI-generated content (AIGC): A history of generative AI from gan to chatgpt*. arXiv.org. https://doi.org/10.48550/arXiv.2303.04226

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017, March 1). *EMNIST: An extension of MNIST to handwritten letters*. arXiv.org. https://arxiv.org/abs/1702.05373

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets - neurips. NeurIPS Proceedings.

https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

LeCun, Y., Cortes, C., & Burges, C. (1998). *The mnist database*. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. https://yann.lecun.com/exdb/mnist/

MNIST :   tensorflow datasets. TensorFlow. (2024a). https://www.tensorflow.org/datasets/catalog/mnist

Module: TF :   tensorflow V2.16.1. TensorFlow. (2024b). https://www.tensorflow.org/api_docs/python/tf

Team, K. (2023). *Keras documentation: DCGAN to generate face images*. https://keras.io/examples/generative/dcgan_overriding_train_step/

Team, K. (2024a). *Keras Documentation: Keras 3 API documentation*. https://keras.io/api/

Team, K. (2024b). *Keras documentation: Variational Autoencoder*. https://keras.io/examples/generative/vae/

Tensorflow datasets. TensorFlow. (2023). https://www.tensorflow.org/datasets/overview

What are convolutional neural networks?. IBM - What are convolutional neural networks? (2021, October 6). https://www.ibm.com/topics/convolutional-neural-networks