# Synthetic Data Project

John Crossman
(Dated: September 21, 2024)

In this project, I train convolutional neural networks (CNNs) on synthetic data generated by a variational autoencoder (VAE) for the task of handwritten digit recognition. I look at how different mixtures of real and synthetic training data affect the performance of these CNNs. Ultimately, I found that adding synthetic data generally did lead to a performance improvement, though this improvement was not as pronounced as just training on redundant data.

## I. INTRODUCTION

The MNIST dataset contains a total of 70,000 28x28 pixel images of handwritten digits [1]. 60,000 of these images constitute the training set, while 10,000 constitute the test set. Fig. 1 displays 4 random example images from the MNIST training set. In 1998, LeCun, Y. et al. introduced a convolutional neural network (CNN) architecture called LeNet, which is able to achieve a test set classification accuracy greater than 98% [2].

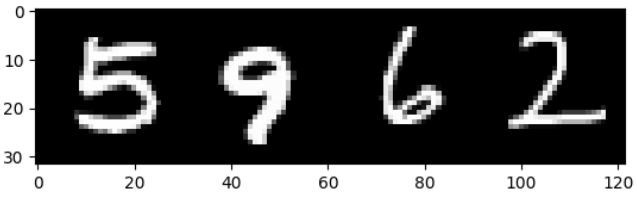


FIG. 1. Four random MNIST image samples. The numbers are 5, 9, 6, and 2.

A variational autoencoder (VAE) is a type of generative model that outputs samples similar to what it has been trained on. For example, if trained on MNIST samples, it should generate images that look like handwritten digits, but ideally are not just blatant copies of what is already in the dataset. It does this by mapping the input data to a learned latent Gaussian distribution and sampling from it. The loss function it is trained with is composed of two parts:

$$\mathcal{L}_{VAE} = \mathcal{L}_{\text{recons}} + \mathcal{L}_{KL}, \tag{1}$$

where $\mathcal{L}_{\text{recons}}$ is the reconstruction loss and $\mathcal{L}_{KL}$ is the Kullback-Leibler (KL) divergence loss. The reconstruction loss helps ensure that the generated images closely match what the model is being trained on, whereas the KL loss helps ensure latent features are learned to prevent overfitting and help with generating new samples.

A $\beta$ variational autoencoder ($\beta$-VAE) is a type of VAE wherein the loss is instead given by

$$\mathcal{L}_{VAE} = \mathcal{L}_{\text{recons}} + \beta\mathcal{L}_{KL}, \tag{2}$$

where a $\beta$ parameter has been introduced to influence the weighting of the KL loss relative to the reconstruction loss. If $\beta = 1$, the original VAE is recovered. However, if $\beta < 1$, then more weighting is given to the reconstruction loss, and thus the VAE will generate sharper images at the cost of a more entangled latent space. If $\beta > 1$, then more weighting is given to the KL loss, and so the VAE will generate blurrier images, as more emphasis is put on learning general latent features. This leads to a more compressed learned representation, in line with the rate distortion curve [3].

A conditional VAE (CVAE) is an extension of a normal VAE, wherein additional information is passed in to influence generation. For example, for MNIST generation, the desired number may be passed in to generate an image of that specific number.

In this project, I use a $\beta$-CVAE to generate synthetic samples from the MNIST data set. I then train CNNs based on LeNet's architecture with varying mixtures of synthetic and real MNIST samples and compare their accuracy in digit recognition on the real MNIST test set.

## II. RESULTS

The $\beta$-CVAE was trained on all 60,000 images in the MNIST training set for 50 epochs with a batch size of 100. The Adam optimizer was used with a learning rate of $\alpha_{VAE} = 0.0003$. $\beta$ was fixed at $\beta = 1.0$. The latent dimension of the model was 20. The encoder uses a series of Conv2d layers followed by ReLU activation functions, while the decoder uses a series of ConvTranspose2d layers, also with ReLU activation functions interspersed. More details on the model architecture can be found in the code. Fig. 2 shows the training loss of the $\beta$-CVAE versus batch.

After training, the model was then used to generate 60,000 new synthetic MNIST samples. Specifically, 6,000 images of each digit were created. Fig. 3 shows some of these synthetic images. The model clearly works well.

The CNNs used were based on the LeNet architecture [2]. They were all trained for 5 epochs with a batch size of 20. The Adam optimizer was used with a learning rate of $\alpha_{CNN} = 0.001$. Fig. 4 shows the training loss versus batch for a CNN trained on all of the original MNIST training data, while Fig. 5 shows the training loss versus batch for a CNN trained on all of the synthetic data. The CNN trained on the original data achieved a test accuracy of 98.23%, while the CNN trained on the synthetic data achieved a test accuracy of 96.40%.

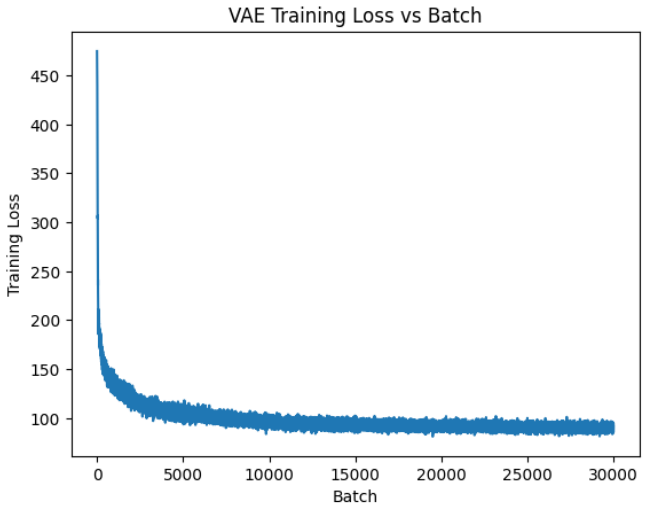


FIG. 2. $\beta$-CVAE training loss versus batch.

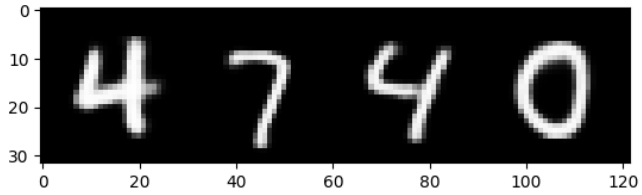


FIG. 3. Four randomly generated images from the trained $\beta$-CVAE. The prompted labels were 4, 7, 4, and 0. Clearly, the model generates convincing images of the requested digit.

After training the $\beta$-CVAE and confirming that a CNN trained on synthetic data generated by it could achieve high classification accuracy, different mixtures of real and synthetic training data were tested. In all scenarios, the same training parameters were used as before; only the dataset trained on varied.

First, CNNs were trained on varying percentages of the real MNIST training set, ranging from 5% to 100%. 20 different percentages were tested. For each percentage, 10 random slices of the training set of that percentage size were taken and a CNN was trained for each one of these 10 random same-sized slices. The test classification accuracy of these 10 CNNs was then averaged. The result of this is graphed in Fig. 6. This was also done for the synthetic data set, as depicted in Fig. 7. As expected, test classification accuracy increases with dataset size in both scenarios.

After that, mixing both real and synthetic data was tested. For this trial, the total number of training images was fixed at 60,000, with varying percentages of these images being from the original MNIST dataset. 19 different percentages, ranging from 5% to 95%, were tested. For each percentage, $p$, a random group of $p*60,000$ real MNIST images were used, while $(1-p)*60,000$ synthetic MNIST images were used. For each of these $p$ percentages, 10 CNNs were trained on 10 different random slices of $p*60,000$ real images and $(1-p)*60,000$ synthetic images. The test accuracy of these CNNs was then averaged. This is plotted in Fig. 8. The red line in the figure is the average test accuracy for ten CNNs trained on all real data, while the green line is the average test accuracy for ten CNNs trained on all synthetic data. There is a clear trend in increasing test accuracy as more of the total data set is composed of the real, original MNIST images.

Finally, augmenting the data was tested. For this trial, all of the real MNIST images were used in the training set, with varying percentages of the synthetic dataset being added as well. 20 different percentages, ranging from

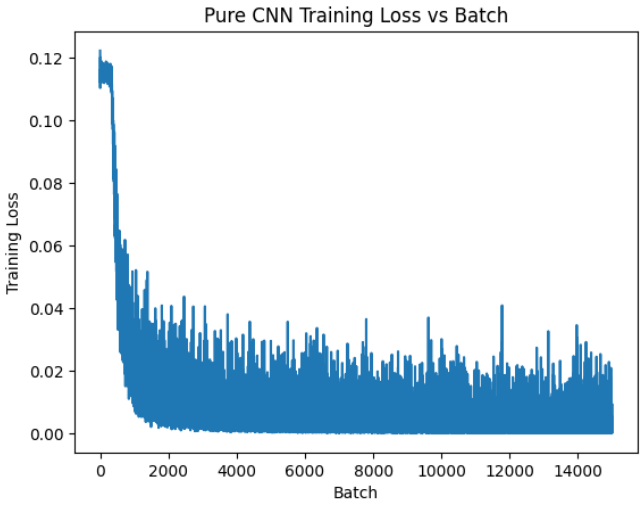


FIG. 4. Training loss versus batch for a CNN trained on all 60,000 of the original MNIST training images. This CNN achieved a test accuracy of 98.23%.

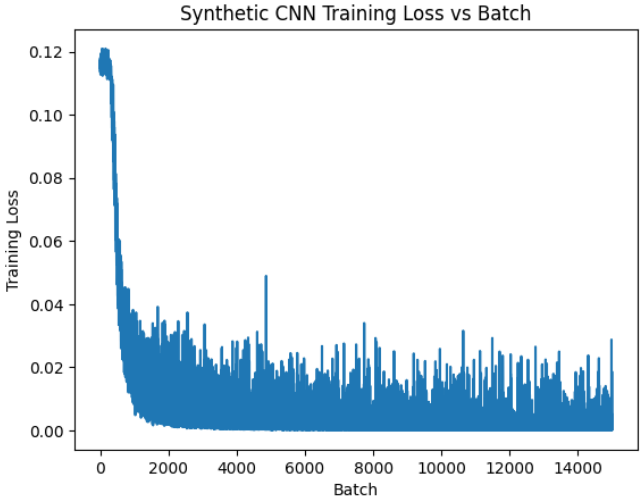


FIG. 5. Training loss versus batch for a CNN trained on all 60,000 of the synthetic images. This CNN achieved a test accuracy of 96.4%.

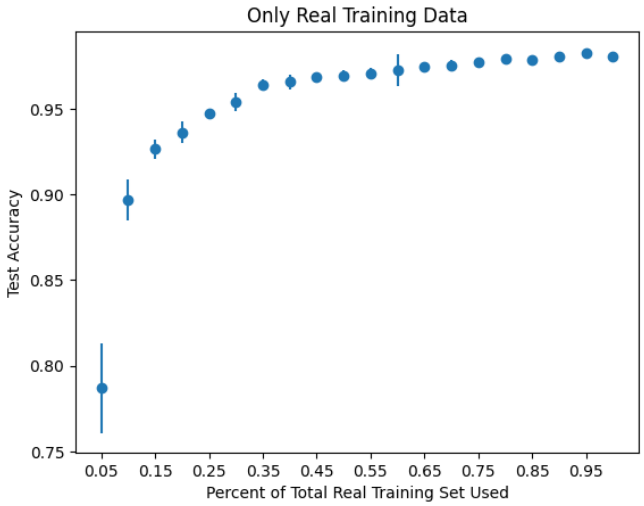


FIG. 6. CNN test accuracy versus percentage of the total real MNIST dataset used in training. Each point was computed using an average of 10 CNNs, each trained on a random slice of the same percentage size of the full dataset.

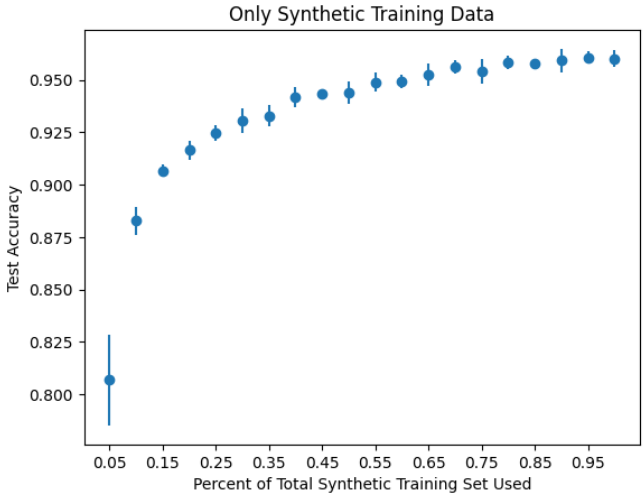


FIG. 7. CNN test accuracy versus percentage of the total synthetic dataset used in training. Each point was computed using an average of 10 CNNs, each trained on a random slice of the same percentage size of the full dataset.

5% to 100% were tested. For each percentage, 10 CNNs were trained on different random groups of the synthetic images added to the training set. The test classification accuracy of these 10 CNNs was then averaged. The resulting plot is shown in Fig. 9. Augmenting the full real MNIST dataset with itself was also tested, and was done in the same fashion as with the synthetic data (this is the same as adding redundant data). The resulting plot is shown in Fig. 10. The red line in the figures is the average test accuracy for ten CNNs trained on all real data, while the green line is the average test accuracy for ten CNNs trained on all synthetic data. There is a clear

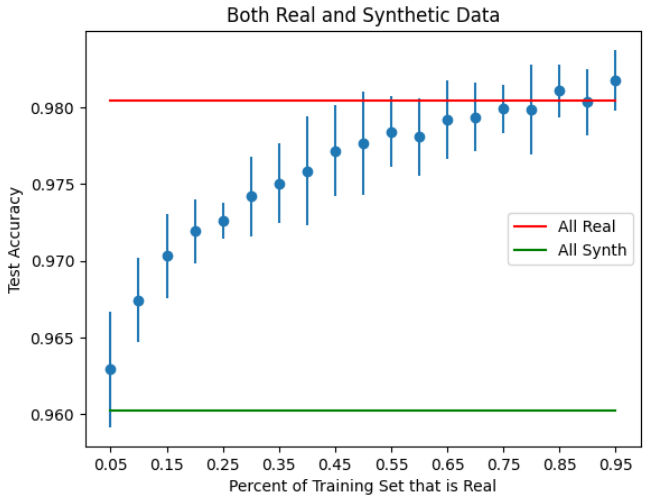


FIG. 8. CNN test accuracy versus percentage of real data used in the training set. Each point was computed using an average of 10 CNNs, each trained on a random dataset of 60,000 total images, $p$ percent of which were from the original MNIST training set, and $(1-p)$ percent of which were from the synthetic dataset.

trend in increasing test accuracy with both synthetic and redundant data added to the training set, though the redundant data clearly performs better than the synthetic data.

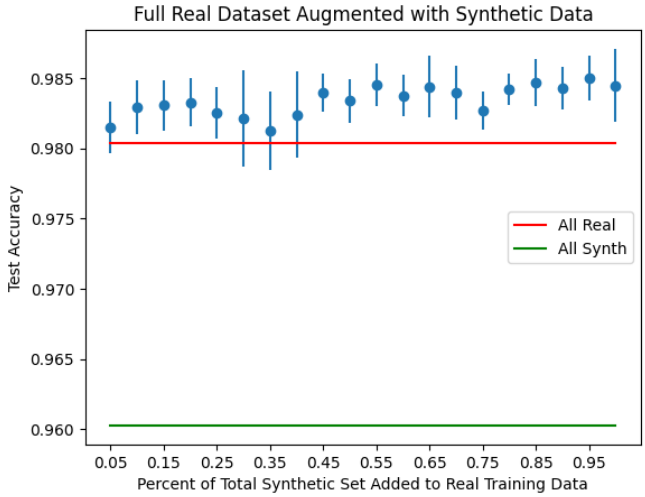


FIG. 9. CNN test accuracy versus percentage of synthetic data added to the training set. Each point was computed using an average of 10 CNNs, each trained with a different random slice of the same percentage size of the synthetic dataset added to the 60,000 image original real dataset.

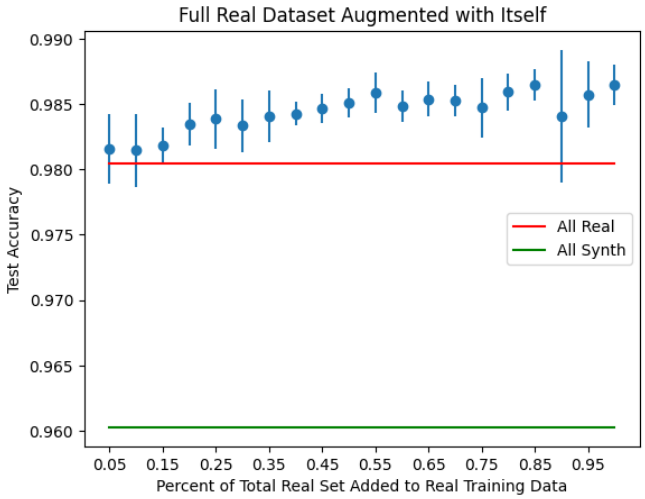


FIG. 10. CNN test accuracy versus percentage of real data added to the training set. Each point was computed using an average of 10 CNNs, each trained with a different random slice of the same percentage size of the real dataset added to the 60,000 image original real dataset.

## III. DISCUSSION

In conclusion, while augmenting the original dataset with synthetic data did in fact lead to a performance improvement, training on redundant data led to a greater improvement.

### A. Future Work

In the future, I plan to try generating synthetic data using different generative models, including generative adversarial neural networks and diffusion models, and comparing each model's efficacy at generating useful training data using the same methods here. I also think that a more thorough investigation of the $\beta$ parameter of the $\beta$-CVAE used in this work is warranted, as I had only fixed it to $\beta = 1.0$.

It might also be worth investigating using multiple generative models, i.e., one for each digit, to generate the data.

[1] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine **29**, 141 (2012).
[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE **86**, 2278 (1998).
[3] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics* (MIT Press, 2023).