Enhancing Classification with Autoencoders: Noise Handling and Discriminative Learning

Xiangsheng Chai
University of Florida
Gainesville, Florida, USA
xchai@ufl.edu

Shania Shakri
University of Florida
Gainesville, Florida, USA
shaniashakri@ufl.edu

***Abstract*—We investigate the use of Autoencoders for dimensionality reduction in the KMNIST digits dataset to enhance classification. We compare this approach against Multi Layer Perceptrons (MLP) for classification performance. Additionally, we introduce an information theory regularizer to guide the Autoencoder in learning 3D latent representations. Further, we explore the impact of Gaussian Model Priors on unsupervised clustering within the latent space. This exploration aims to unveil Autoencoders' potential for improved dimensionality reduction and assess Gaussian priors' influence on unsupervised clustering.**

I. Introduction

In the field of machine learning, managing high-dimensional data efficiently is a key challenge. High-dimensional datasets can lead to complex models that are hard to train and prone to overfitting. Dimensionality reduction techniques are used to address these issues by transforming the data into a simpler form that retains the most critical information for making predictions. This process not only makes the computation less demanding but also often improves the performance of learning algorithms.

Autoencoders, a type of neural network, are one such technique used for dimensionality reduction. They work by encoding data to a lower-dimensional space and then reconstructing it back to the original space. The goal is to learn representations that capture the most important aspects of the data.

In this project, we focus on the K-MNIST dataset, which consists of 70,000 images of handwritten Japanese characters, each 28x28 pixels in size. The dataset is challenging due to its high dimensionality and the subtleties of the characters. We investigate how different types of autoencoders can compress this data and how well they can then be used to classify the characters.

Our main contributions are exploring how to enhance the feature discrimination in the latent space created by the autoencoders and how this affects the classification performance. By comparing different approaches to regularizing the autoencoder's learning process, we aim to find the most effective strategy for this particular dataset.

[0]*In Part 1, Xiangsheng Chai designed the autoencoder and selected hyperparameters. Shania Shakri was responsible for the correntropy part and denoising. For Part 2, both members collaborated equally, with Xiangsheng Chai particularly focusing on the penalty function's parameter selection.*

II. Methodology

A. *Stacked Autoencoder (SAE) Design and Training*

The objective here is to devise an effective Stacked Autoencoder and identify the optimal hyperparameters for achieving peak performance in classification using derived features. The SAE design and training we implemented encompass three core phases:

Initial Exploration This phase involves initial experimentation to understand the hyperparameter space and define baselines for the SAE architecture.

Architecture Overview:

The SAE architecture comprises five hidden layers with specific unit sizes:

1) Input layer: Flattened (dimensions: 28×28)
2) Five hidden layers: 800-400-200-100-XXX (neurons for the bottleneck layer)
3) Activation function: ReLU for all hidden layers

The initial SAE model was trained with:

1) Adam optimizer
2) MSE/Cross-entropy loss function
3) Batch normalization is done between each dense layer
4) 10 epochs to establish a preliminary understanding of the model's behavior

Hyperparameter Investigation: Building upon the initial insights, this phase was focused on a systematic and comprehensive exploration of the hyperparameter landscape to refine the Stacked Autoencoder (SAE) architecture. We engaged in a multi-tiered approach to optimize key facets:

1) Bottleneck sizes: The strategy employed for selecting the best bottleneck size involves iterative experimentation across different intervals. Initially, the performance across discrete points: 10, 40, 70, and 100 for bottleneck size is explored. Through observation, an interval is identified between 10 and 40 to be most promising due to its relatively lower loss within this range. Further narrowing focus, a refined exploration within this range (15, 20, 25, 30, 35, 40) highlighted bottleneck size 40 as having significantly lower Mean Squared Error (MSE), indicating its suitability for out classification task. However, we continued to refine our search, leading to explore the range between 40 and 70. Here, varying

bottleneck sizes (40, 45, 50, 55, 60, 65, 70) helped identify a more optimal size. We sought a balance—a size small enough to maintain computational efficiency while ensuring the loss remains under 0.15. From the results obtained, a bottleneck size of 60 emerged as the optimal choice, meeting both criteria effectively.

2) Learning rate: Similarly, the determination of the optimal learning rate followed a methodical approach of iteratively testing various values. Initially, five points (0.05, 0.01, 0.005, 0.001, 0.0001) are examined, assessing their impact on the model's learning efficiency. Observations from the initial tests reveal that a learning rate of 0.01 is too large, while 0.0001 requires more time to converge. Consequently, the exploration narrows down to the interval between 0.001 and 0.0001. Further experiments are conducted within this refined range, including values like 0.001, 0.0008, 0.0006, 0.0005, and 0.0003. The learning curve plots for each learning rate iteration illustrate their performance in terms of train and validation loss. Based on these results and to optimize model training, a learning rate of 0.0008 was selected for the model, alongside a bottleneck size of 60, as they demonstrated better convergence and computational efficiency.
3) Kernel size for Correntropy: The experimentation encompassed testing different kernel sizes, including 0.05, 0.2, 0.4, 0.6, 0.8, 1, 2, and 5 initially. Subsequently, another set of kernel sizes—0.8, 0.9, 1, 1.1, 1.2, and 1.3—were examined. These kernel sizes were explored while implementing the correntropy cost function to ascertain their efficacy in reducing noise from the output.

Refinement and Optimization: In the final stage of refining the Stacked Autoencoder (SAE), we employed a streamlined set of hyperparameters derived from our prior experiments. These insights led us to focus on specific adjustments: first, maintaining a bottleneck layer size of around 200 neurons, which proved to be most effective. Additionally, we fine-tuned essential parameters like the learning rate and kernel size to enhance the SAE's ability to handle noisy data. To test its capacity to reduce noise, we increased the model's depth slightly and extended the training period to roughly 80 epochs. This prolonged training facilitated better learning within the deeper segments of the model.

B. Impulsive Noise Addition and Correntropy in Stacked Autoencoders (SAE)

In this project, the inclusion of impulsive noise within input images is done as a way to assess the resilience of Stacked Autoencoders (SAE) in real-world scenarios. It is employed as an augmentation technique. We develop a noise simulation technique to simulate random pixel corruption, where pixels are randomly turned either completely black (salt) or white (pepper) as seen in Figure 1. This utilizes the probability-based approach, adjusting pixel values within the image tensor governed by a tunable parameter [2]. The aim is to enhance the robustness of the Stacked Autoencoders (SAE) model by training it on both clean and noisy data representations.

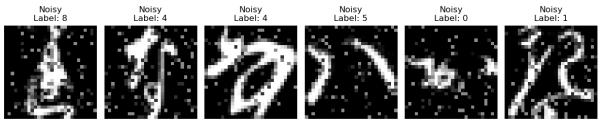

Fig. 1: Impulsive Noise Addition to Data

Integration of Correntropy in Stacked Autoencoders (SAE): In parallel with impulsive noise addition, the training of Stacked Autoencoders (SAE) was structured around the integration of correntropy, a robust similarity measure. This methodology incorporated correntropy as a guiding principle, shaping the cost function during training. We achieved this through the implementation of the Correntropy Loss function, defined as follows:

$$\text{CorrentropyLoss} = 1 - \frac{1}{N}\sum_{i=1}^{N}\exp\left(-\frac{(x_i - y_i)^2}{2\sigma^2}\right) \quad (1)$$

In this formula, x_i and y_i represent the elements of the predicted and actual value tensors, N is the total number of elements, and σ is the standard deviation of the Gaussian kernel. This loss function effectively measures the similarity between predicted and actual values using the Gaussian kernel, thereby mitigating the impact of impulsive noise and outliers within the dataset [4].

C. Penalty Function for Discrimination Enhancement

To enhance the discrimination capabilities of the latent space in our Stacked Autoencoder, we introduced a penalty function based on the Cauchy-Schwarz Divergence (CSD). The CSD is an information theoretic measure that assesses the divergence between two distributions, encouraging the latent representations to align with a Gaussian prior.

1) Regularization Approach: We applied the CSD as a regularizer in the Autoencoder's loss function. This divergence measure penalizes the latent representations that deviate from a pre-defined Gaussian distribution, effectively shaping the latent space to be more discriminative. Given two distribution $f(x)$ and $g(x)$, the Cauchy-Schwarz divergence $D_{cs}(f, g)$ is showed as formula(1).

$$D_{cs}(f, g) = -\log \frac{\left(\int f(x)g(x)dx\right)^2}{\int f^2(x)dx \int g^2(x)dx} \quad (2)$$

2) Experimentation: We experimented with various regularization strengths, denoted by λ, to observe the effect on the latent space distribution. The regularization strengths tested were λ = [0, 0.01, 0.1, 0.5, 1, 2, 10], with the understanding that a higher λ would enforce a stronger penalty for deviation from the Gaussian prior. In addition, we also experiment with different batch size values, include [5, 10, 20, 50, 100, 500, 1000]. To assess the impact, we visualized the latent space using a 3D

scatter plot, where each point represents an encoded sample colored by its class label.

3) Comparison: After the experiment with all kinds of hyper-parameters, we used same bottleneck size as in first part and made a comparison between SAE classifier with and without penalty function. Confusion matrices were used to make a comprehensive comparison. Then we will use bottleneck size 10 retrain the model the see the performance again.

III. Experimental Results

A. *Comparison of SAE+Classifier and MLP*

The comprehensive exploration of the hyperparameter space, including bottleneck size and learning rate, is essential in optimizing the Stacked Autoencoder (SAE) model. The choice of a bottleneck size of 60 and a learning rate of 0.0008, as depicted in Figure 2 and Figure 3, strikes a balance between minimizing reconstruction loss and maintaining computational efficiency. The larger bottleneck size captures more nuanced features of the data, improving the model's ability to learn complex patterns, while the selected learning rate ensures steady and effective convergence during training.

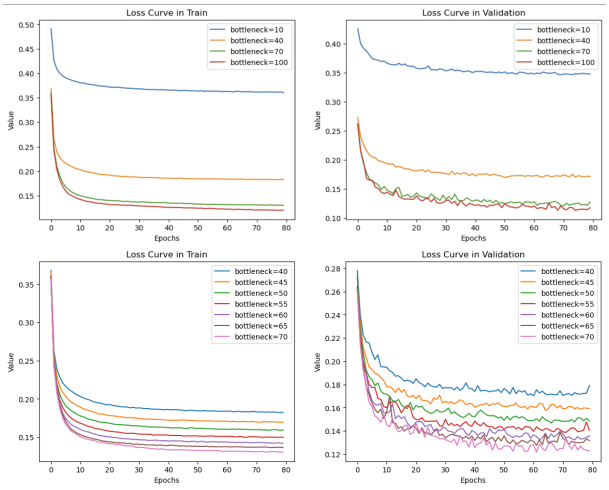

Fig. 2: Effect of Bottleneck Size

In project 1, we introduced 2 hidden layers' MLP classifier with an input layer that flattens the 28x28 pixel images into a contiguous array of 784 elements. The test accuracy is 92.99%, see confusion matrix in Fig.4.

TABLE I: Comparison of Two Classifiers

Classifier	Accuracy	Time per epoch (s)	Number of parameters
MLP	92.99%	10.9	537,354
MLP with SAE	93.11%	7.6	213,010

Now we introduce SAE which reduces the dimension of input images from 784 to 60. We built a simple two hidden layer MLP classifier with batch normalization, and the accuracy of classifying is 93.11%. The accuracy of two classifier is close, but MLP use 784 dimensions of features as input and SAE plus

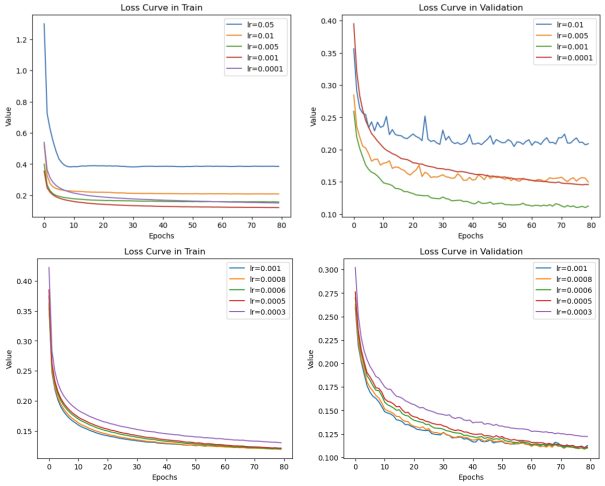

Fig. 3: Effect of Learning Rate

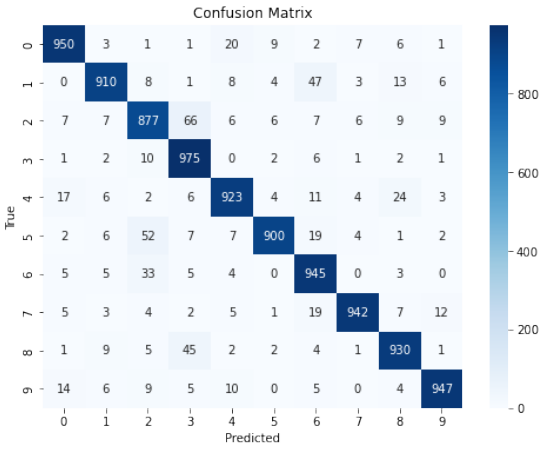

Fig. 4: Confusion Matrix: MLP

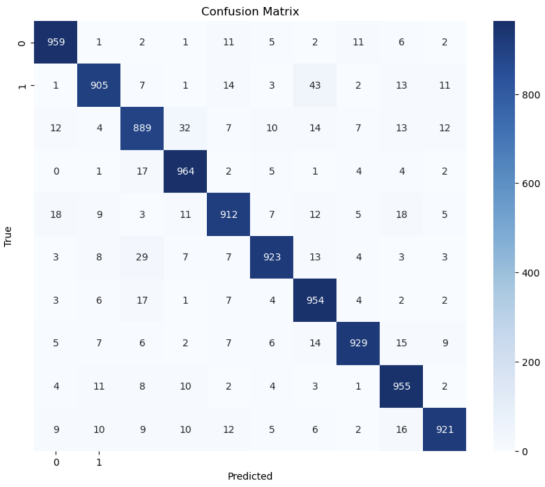

Fig. 5: Confusion Matrix: MLP with encoder

MLP only use 60 dimensions of features, which means the second classifier use less parameters and save more resource

on computation. From Table I, it illustrates the priority of classifier with dimension reduction with auto-encoder. It saves lots of resource on computational cost. The time cost on per epoch is based on computation in NVIDIA GeForce RTX 3060 Ti GPU with batch size equal to 100.

B. Impact of Noise Addition on Latent Representations

The robustness of a dimensionality reduction model is not solely characterized by its accuracy but also by its ability to maintain performance under perturbed conditions, such as the presence of noise in the data. To evaluate the resilience of our Stacked Autoencoder (SAE), we introduced controlled noise to the KMNIST dataset, simulating real-world data corruption scenarios.

1) Denoising with MSE: In the training phase utilizing Mean Squared Error (MSE) loss, the model underwent 80 epochs of training. The initial MSE loss recorded was approximately 0.425, which indicates the difference between the noisy input and the clean target. Over the course of training, this loss value showed a steady decrease, settling around 0.231 by the 80th epoch, as detailed in Table II. This steady decline in loss suggests that the autoencoder was gradually learning to filter out the noise and reconstruct the digit images more accurately. However, MSE is known to be sensitive to outliers, which could potentially lead to overfitting if the noise includes such anomalies. The sensitivity to outliers might cause the autoencoder to adjust excessively to the noise present in the training data, potentially impeding the model's generalization capability when exposed to new, unseen data. The reconstructed image is shown in Figure 6.

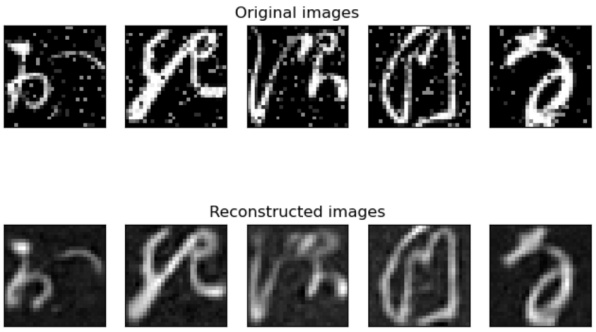

Fig. 6: Reconstructed Images with MSE Optimization

2) Denoising with Correntropy: For the correntropy-based loss, the training was carried out over 50 epochs. The correntropy loss started higher at approximately 0.458, yet showed a quicker convergence, ending up at around 0.244 by the 50th epoch. The advantage of using correntropy lies in its resilience to outliers due to its incorporation of a Gaussian kernel, which focuses on the local similarities between the reconstructed output and the clean target. This characteristic allows the autoencoder to be less affected by outliers, hence avoiding the overtraining issue that might occur with MSE. The faster convergence and the lower loss after 50 epochs as compared to MSE after 80 epochs suggest that correntropy is potentially more effective at generalizing the denoising process. This robustness to noise leads to a latent space that preserves the discriminative features necessary for the classification task, ensuring the model's resilience even when trained with noisy data. The reconstructed image is shown in Figure 7.

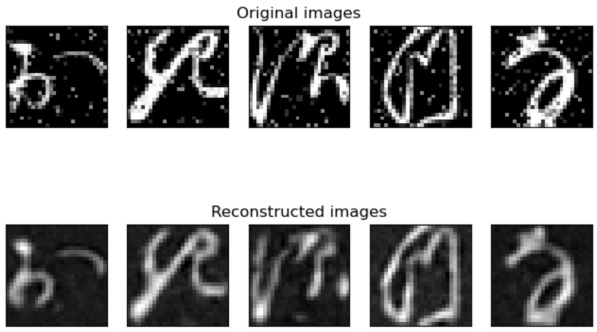

Fig. 7: Reconstructed Images with Correntropy Optimization

TABLE II: Impact of Noise on SAE Model Performance

Training Condition	Initial Loss	Final Loss	Number of Epochs
Train with MSE	0.425%	0.231	80
Train with correntropy-based loss	0.458%	0.244	50

3) Kernel Size Experimentation: The experimentation process for selecting the kernel size of 1.1 involved an iterative approach. Initially, we tested a broad range of kernel sizes (including 0.6, 0.8, 1, 2, and 5) to understand their impact on the SAE's ability to handle noise. The kernel size of 1 showed promising results in terms of loss convergence and generalization capabilities without overfitting, as seen in the visual comparison of train and validation loss curves, Figure 8. We decided to conduct a more focused set of experiments

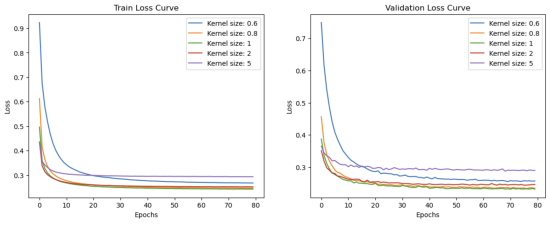

Fig. 8: Loss Curves for Varied Kernel Sizes

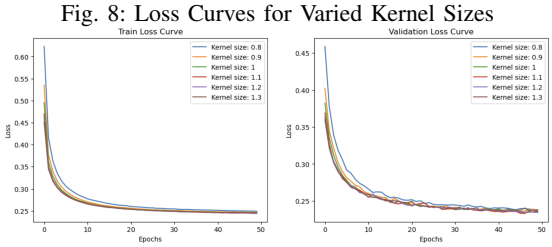

Fig. 9: Loss Curves for Fine-tuned Kernel Sizes

around the promising kernel size of 1 to fine-tune the selection, with sizes like 0.8, 0.9, 1, 1.1, 1.2, and 1.3 being evaluated, see Figure 9. The kernel size of 1.1 resulted as the optimal choice, balancing the trade-off between loss convergence speed and

the autoencoder's ability to denoise the input data effectively. The decision for 1.1 as the best kernel size was reinforced by the lowest validation loss achieved during training, indicating its superior performance for the given SAE architecture and dataset.

C. *Latent Space Regularization with Gaussian Prior*

To encourage spread in the 3d latent space, we enforce a Gaussian prior with zero mean and variance one $N(\mu, \sigma)$ on it. We integrated class-specific constellation targets into the latent space to enhance discrimination, ensuring codes are optimally positioned for classification while adhering to the Gaussian distribution. We maintained consistency with our optimization strategy, using the Adam optimizer with a learning rate of 10^{-5}. The model was subjected to a total of 60 epochs to ensure adequate learning without overfitting. Gaussian distribution imposes a spread on the space centered around a mean. We apply Cauchy-Schwarz divergency to measure the distance between latent code distribution and imposed prior. CSD shows how much one vector can project onto another.

1) Effect of batch size (K): Firstly, we tried to figure out the best batch size. We fixed the regularization parameter $\lambda = 0.1$ at first and changed the batch size from [5, 10, 20, 50, 100, 500, 1000]. The 3D space visualizations are shown in Figure 10.

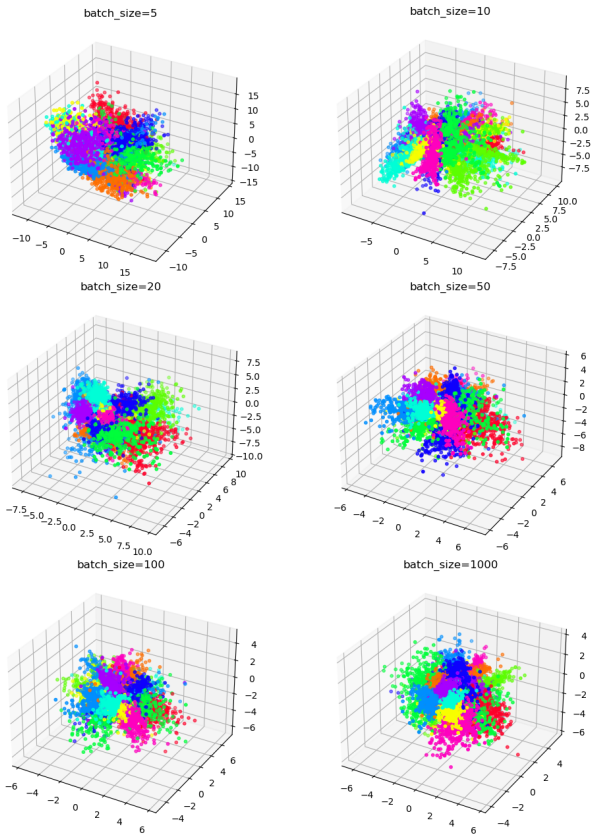

Fig. 10: 3D latent space for different batch size

From the figure, 5 batch size has divergent structure and 1000 batch size is more impact. We can see as the batch size going larger, whole data points are going to impact in a "smaller" ball and the outlier of dataset are more closed to zero. It means bigger batch size will force auto-encoder to focus on learning to establish the latent code distribution into Gaussian distribution rather than improve the MSE loss in reconstrucrtion. For these reason, batch size in 100 is a good choice.

2) Effect of regularization parameter (λ): After finding the optimal batch size, we fixed it in 100 and changed the regularization parameter λ in [0.01, 0.1, 0.5, 0.1, 2, 5, 10] to optimize it.

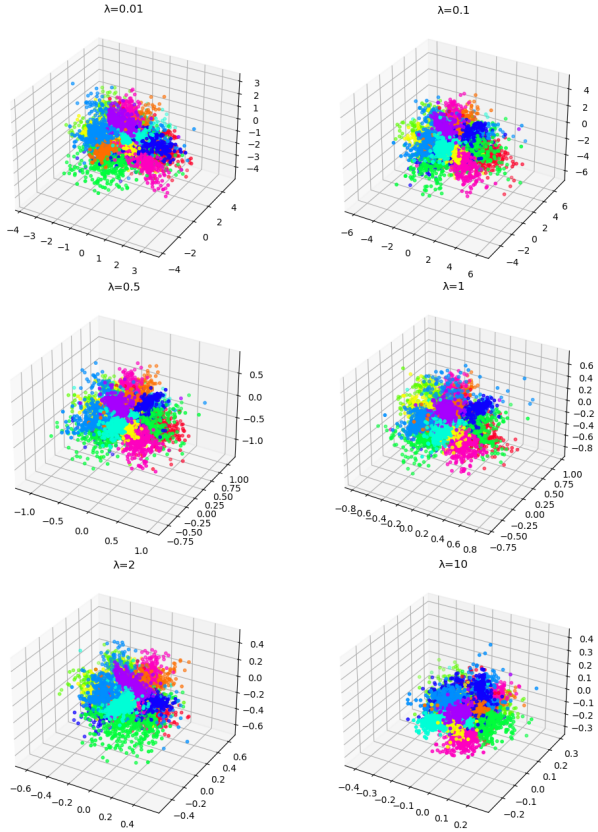

Fig. 11: 3D latent space for different λ

From Figure 11, regularization parameter λ played the same role as batch size. With smaller value, latent code space has more divergent structure and less like Guassian distribution, and with larger value, the latent code space is more like a "ball". λ apply a direct control on CSD divergence between latent code distribution and imposed prior in penalty function. When λ is small, MSE will dominate the penalty function to learn more to reduce the reconstruction difference. Finally, we use the optimal batch size and regularization parameter to train the model with euclidean distance. Compared to euclidean distance, CSD made the latent code structure more close to Gaussian distribution.

3) Comparison: With the comparison of confusion matrix between the classifier with and without regularization, we can

see it has improvement on correctly classifying almost all class except class 2, and it has huge improvement on classifying class 1 and 4, more than 30 samples are correctly assigned. In training time, classifier with regularization use 7.4 seconds per round in average compared to classifier without regularization's 7.6 seconds. Then, we changed the bottleneck size

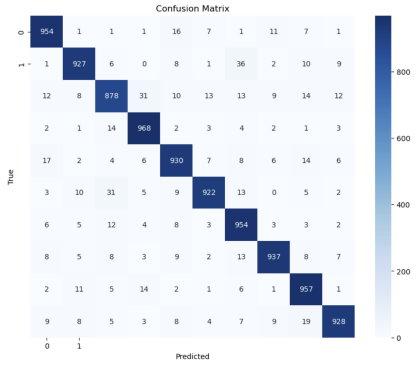

Fig. 12: Confusion Matrix: SAE + MLP with regularization

from 60 to 10, and retrained the auto-encoder and classifier model with and without regularization. The confusion matrix are shown as Figure 13 and Figure 14.

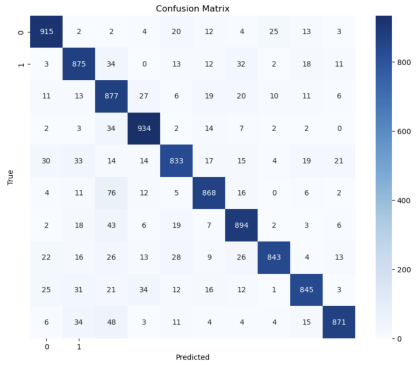

Fig. 13: Confusion Matrix: SAE + MLP in 10 latent space

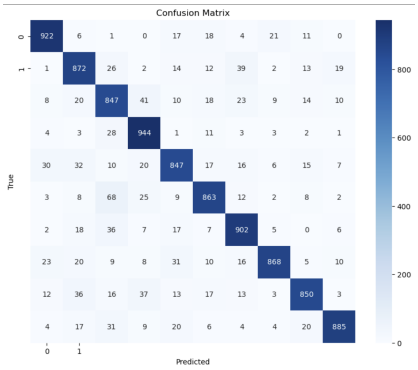

Fig. 14: Confusion Matrix: SAE + MLP in 10 latent space with regularization

D. Quantification of Computation Time

1) Environmental specification: We implemented the project on a system powered by an NVIDIA GeForce RTX 3060 Ti GPU. The software framework utilized was PyTorch, under Python 3.x environment.

2) Training time of Stacked Encoders (SAE): The training process of the SAE was conducted over 60 epochs. On an average, each epoch required approximately 7.5 seconds to complete, leading to a total training duration of around 450 seconds, or 7.5 minutes. This time span covered the full forward and backward propagation cycles for the SAE across the Kuzushiji-MNIST dataset.

3) Classifier Training Time: Following to the SAE training, the Multi-Layer Perceptron (MLP) Classifier was trained, also over 60 epochs. The classifier exhibited a similar performance in terms of time efficiency, averaging 7.6 seconds per epoch. Therefore, the cumulative training time for the classifier was about 456 seconds, equivalently 7.6 minutes.

IV. Discussions

Expectations: In the field of dimensionality reduction and unsupervised learning, especially in image classification tasks, it is generally anticipated that the integration of a penalty function within an autoencoder's architecture would significantly increase its discriminative power. By imposing a structured distribution in the latent space, penalty functions, like Cauchy-Schwarz Divergence (CSD), facilitate better class separation and noise robustness. We expected that the addition of CSD would result in more distinct and well-separated clusters in the latent space, leading to improved classification accuracy and noise handling capabilities compared to a standard autoencoder without such regularization.

Results: As a result of incorporating the CSD penalty function into the autoencoder architecture, the model's ability to distinguish between different characters improved significantly. This was clearly evidenced by the enhanced learning curves and confusion matrices. Particularly when Correntropy was used as a loss function instead of Mean Squared Error (MSE), the model demonstrated a considerable improvement in handling noise. As a result of the penalty function, the model maintained a Gaussian-like distribution in the latent space, which is essential for effective pattern recognition in unsupervised learning. In addition, we found that hyperparameter tuning, particularly the regularization strength λ and batch size, was important for optimizing latent space distributions and model performance.

TABLE III: Head on Comparison of Performance Across Models

ML Model	Bottleneck Size	Accuracy
2 hidden layers MLP	Not Applicable	[92.99%]
SAE + 2 hidden layers MLP	60	[93.11%]
Classifier without Regularizer	10	[87.55%]
Classifier with Regularizer	60	[93.55%]
Classifier with Regularizer and Bottleneck Size 10	10	[88.00%]

V. Conclusion

In this project, we used stacked auto-encoder to reduce the dimension of KMINST dataset from 784 to 60 and applied a MLP classifier to make the classification. As compared to regular MLP classifier and SAE plus MLP classifier, the latter used less computational cost in almost the same accuracy after dimension reduction. Also, in the reconstruction task, we show that correntropy, which is robust against outliers and able to handle non-linear relationships, can be used to denoise data more effectively.

In the second phase of the project, we explored the enhancement of classification performance by designing a penalty function that aims at enhancing discriminative capacity of the latent space. As a result of systematic experimentation and hyperparameter tuning, we identified optimal settings for this function, resulting in improved classification accuracy. Additionally, the study contributes to the broader field of neural network optimization and its application in real-world situations by integrating dimensionality reduction and advanced denoising techniques.

References

[1] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," arXiv preprint arXiv:1812.01718, 2018.

[2] Rusak, Evgenia et al. "A Simple Way to Make Neural Networks Robust Against Diverse Image Corruptions." European Conference on Computer Vision (2020).

[3] https://www.kaggle.com/code/weka511/autoencoder-implementation-in-pytorch

[4] Santana, E., Emigh, M.S., & Príncipe, J.C. (2016). Information Theoretic-Learning auto-encoder. 2016 International Joint Conference on Neural Networks (IJCNN), 3296-3301.

Appendix

```
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
           Linear-1                 [-1, 800]         628,000
      BatchNorm1d-2                 [-1, 800]           1,600
           Linear-3                 [-1, 200]         160,200
      BatchNorm1d-4                 [-1, 200]             400
           Linear-5                  [-1, 10]           2,010
================================================================
Total params: 792,210
Trainable params: 792,210
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 3.02
Estimated Total Size (MB): 3.04
----------------------------------------------------------------
```

Fig. 15: Structure of Encoder

```
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
           Linear-1                 [-1, 200]           2,200
      BatchNorm1d-2                 [-1, 200]             400
           Linear-3                 [-1, 800]         160,800
      BatchNorm1d-4                 [-1, 800]           1,600
           Linear-5                 [-1, 784]         627,984
================================================================
Total params: 792,984
Trainable params: 792,984
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.02
Params size (MB): 3.02
Estimated Total Size (MB): 3.05
----------------------------------------------------------------
```

Fig. 16: Structure of Decoder

```
----------------------------------------------------------------
       Layer (type)               Output Shape         Param #
================================================================
           Linear-1                 [-1, 800]         628,000
      BatchNorm1d-2                 [-1, 800]           1,600
           Linear-3                 [-1, 200]         160,200
      BatchNorm1d-4                 [-1, 200]             400
           Linear-5                  [-1, 60]          12,060
          Encoder-6                  [-1, 60]               0
           Linear-7                 [-1, 800]          48,800
      BatchNorm1d-8                 [-1, 800]           1,600
           Linear-9                 [-1, 200]         160,200
     BatchNorm1d-10                 [-1, 200]             400
          Linear-11                  [-1, 10]           2,010
    MLPClassifier-12                 [-1, 10]               0
================================================================
Total params: 1,015,270
Trainable params: 1,015,270
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.03
Params size (MB): 3.87
Estimated Total Size (MB): 3.91
----------------------------------------------------------------
```

Fig. 17: Structure of Autoencoder