Autoencoders performance on KMNIST Classification Problem and the effect of ITL-regularization on unsupervised classification

This work is presented as a project for EEL6814 – Deep Learning Course at the University of Florida

Group Members
Gijung Lee
Mayar Shahin

Autoencoders performance on KMNIST Classification Problem and the effect of ITL-regularization on unsupervised classification

Gijung Lee
Electrical and Computer Engineering department
University of Florida
lee.gijung@ufl.edu

Mayar Shahin
Physics department
University of Florida
mayar.shahin@ufl.edu

Abstract—**We investigate the use of Autoencoders for reducing the dimensionality of KMNIST digits data set to improve classification performance. We compare the classification performance to the use of Convolutional Neural Networks (CNNs). We further introduce an information theory regularizer on the autoencoder. We force the autoencoder to learn latents on a 3d swiss roll prior and decode the images. Lastly, we introduce a Gaussian and a Gaussian Mixture Model Prior to investigate its effect on unsupervised clustering of the latent space.** This work is presented as the final project for **EEL6814 – Deep Learning Course.**

I. Introduction

A. *Dimensionality reduction*

The age of abundance of high resolution high dimensional data calls for a method to deal with it. Learning on high dimensional data can be very computationally taxing and can lead to overfitting of training data. Dimensionality reduction methods aim to mitigate these problems by preserving the most relvant information in the data needed to learn accurate predictive models. It helps in reducing redundant features by extracting a lower dimensional representation of the data that is more essential and more interpretable. It can also be used in visualization if the lower dimensional representation is 2 or 3 dimensional.

B. *Autoencoder (AE) with a bottleneck as an architecture for dimensionality reduction*

An autencoder is a type of artificial neural network network that aims at learning lower dimensional latent codings from data. Two networks are simalteneously trained; the encoder and decoder. The encoder transforms the data onto latent codings while the decoder tries to reconstruct the data from the codings. The loss function is given by the reconstruction error between the input of the encoder and the output of the decoder: $\mathcal{L} = |x - \hat{x}|$. By learning to cipher and decipher the data, the autoencoder learns the essential part of the image. The latent codes are 'denoised' versions of the input data that for example facilitates classification [8].

AEs have variants that are tailored for different tasks. Regualrized AEs and variational AEs are good examples.

II. Methods

A. *Data Preparation*

We download the publicly available data set **Kuzushiji-MNIST** (KMNIST) [1]. With advancement in deep learning and how well current classifiers work on the MNIST data set, more complicated data set is need to benchmark performance. KMNIST is used as a replacement for the MNIST dataset (28x28 grayscale, 70,000 images). It consists of 10 classes each representing 10 characters of Hiragana, a Japanese writing system, see Fig. 1. The data set is divided into 60000 training samples and 10000 testing samples. We set aside 10% of the training set for model validation as the learning is done.

B. *Stacked Autoencoders*

We design an autoencoder network, which projects data to a subspace to obtain features that can be then used for classification. We use a stacked autoencoder has the architecture 800-400-XXX as the encoder and 400-800 as the decoder, where XXX is the latent space dimensions. Each layer is a fully connected layer. We use tanh activation function for the encoder hidden layers and linear activation for the latent space. As for the decoder we use ReLU activation and sigmoid for the final layer to produce pixel values between 0 and 1.

Figure 1: The 10 classes of KMNIST, with the first column showing each character's modern hiragana counterpart. [1]

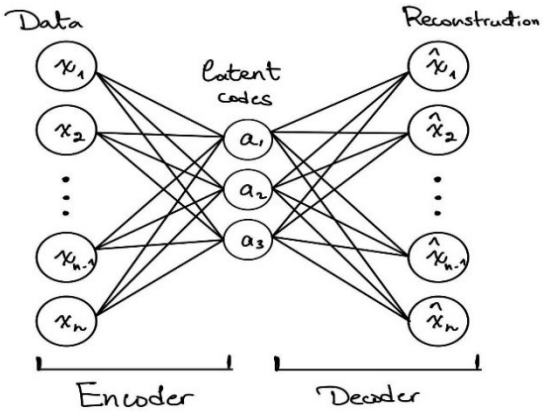


Figure 2: A standard autoencoder architecture

C. *MLP Classifier*

We use a fully connected multilayer perceptron with 3 hidden layers with units [1000-500-500] with relu activations and softmax output for classification. We use categorical cross-entropy as the training loss.

D. *Information Theoretic Learning (ITL) regularization*

Having an autoencoder that is penalized only on the reconstruction loss can lead to biasing the network to learn only mapping data points to very specific points on the latent space, i.e. the spatial structure of the space doesn't have meaning. This bias leads to two main problems: first, learning a discontinuous latent space where if you sample a neighboring point to an encoded data point and pass it through the decoder you do not get a meaningful representation (a problem solved by variational autoencoders). Second, the distances between points do not necessarily mean similarity, which can make you lose the advantage of clustering. To solve this issue, we are imposing an Information Theoretic Learning (ITL) regularizer that computes the Cauchy-Schwarz divergence (CSD) of the latent code with respect to a prior distribution. This divergence is added to the lost function with some multiplier λ.

$$\mathcal{L} = |x - \widehat{x}| + \lambda\, CSD(q(z|x) \,||\, p(z)),$$

where $q(z|x)$ is the encoder and $p(z)$ is an imposed prior distribution on the latent space.

a) Probability density estimation using prazen window

We use Prazen Window method to estimate the probability density functions. The estimation is done by centering a Gaussian Kernel $K_\sigma(x - x_i)$ with size σ at each data point x_i and summing the Gaussians to estimate the PDF. The kernel size σ is an added hyperparameter that we optimize for. The more data points you include in your batch, the smaller the kernel size you need.

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N} K_\sigma(x - x_i).$$

b) Renyi's 2nd order entropy and crossentropy estimation

An important information theoretic learning metric is Renyi's Entropy of a probability distribution $p(x)$. The second order entropy is given by $H_2(X) = -\log \int p^2(x)dx$ and can be estimated using the prazen window method to be

$$\hat{H}_2(X) = -\log \frac{1}{N^2}\sum_j \sum_i K_{\sigma\sqrt{2}}(x_j - x_i) \equiv -\log V(X)\,.$$

Similarly, the cross-entropy between two distributions can be estimated as

$$\hat{H}_2(X,Y) = -\log \frac{1}{N_X N_Y}\sum_j \sum_i K_{\sigma\sqrt{2}}(x_i - y_j) \equiv -\log V(X,Y)\,,$$

where $V_2(X)$ is the information potential and $V_2(X,Y)$ is the cross-information potential. In terms of the information potentials, the Cauchy Schwartz divergence is given by

$$CSD(p_X || p_Y) = \log \frac{V(X)V(Y)}{V^2(X,Y)}.$$

By minimizing the CSD, we enforce the model to maximize the entropy of the latent code distribution which helps in spreading out the latents while at the same time minimizes the cross-entropy between the latent code and the prior distributions which makes latent codes fit the prior distribution better.

c) Unsupervised clustering of the latent codes using K-means

The k-means algorithm is an iterative technique that attempts to cluster data points into K separate non-overlapping subgroups, with each data point belonging to just one of these groups. It aims to make intra-cluster data points as comparable as feasible while maintaining clusters as distinct (far) as possible. It distributes data points to clusters so that the sum of the squared distances between them and the cluster's center is as little as possible. Within a cluster, the less variance there is, the more similar the data points are.

We use k-means clustering on the latent codes to predict which data points belong to which classes. To match the clustering labels to the original labels, we evaluate the cluster-class pairings that maximizes the classification accuracy.

III. RESULTS

A. *Classification results on the latent codes generated by a Stacked Autoencoder (SAE)*

a) 30 dimensional latent codes are optimal for classification using a Multilayer Perceptron (MLP)

The idea is that by limiting the bottleneck size, we are reducing noise in the data and learning only the features essential for classification. Even though the autoencoder reconstruction loss will always be lower when you add more dimensionality into the bottleneck, the classification accuracy on the latent codes will eventually get worse since the network is overfitting the data.

We experiment with different sizes and stop increasing the sizes when classification accuracy decreases, see table 1.

10	20	30	35	40	50
87.18%	90.97%	92.54%	92.41%	91.81%	91.81%

Table 1: Classification accuracy using different number of bottleneck units

b) Classification using the 30 dimensional latent codes outperforms classification using the full image dimensionality

Now that we have established a proper number of latent codes (30), we can improve on the model predictions by adding batch normalization and 25% dropout on the nodes in the MLP classifier. This addition increases accuracy to 93.25%, see confusion matrix in Fig. 3.

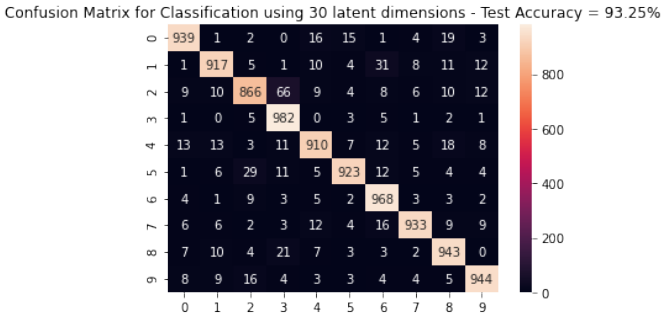


Figure 3

c) Comparison with CNN classification performance on the same dataset

Although the MLP classifier on the latent codes outperforms the MLP classifier on the full dimensional image (91.5%), the classification accuracy does not compete with the best performing CNN with 3 hidden layers we trained in project 1 (96.5%). CNN is still better at capturing image details because it does not involve flattening of the

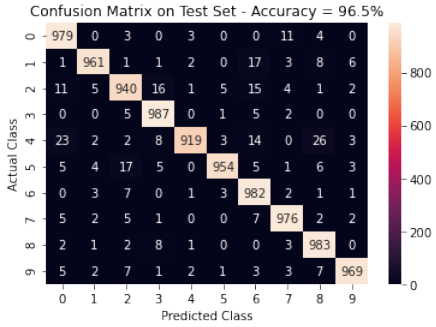


Figure 4: Confusion matrix for CNN classification on the full image

B. Enforcing a 3D swiss roll prior on the latent codes

In order to enforce the minimization of the Cauchy-Schwarz divergence while maintaining low reconstruction error, we need to tune for the regularization parameter λ and the kernel size σ.

For this experiment, we used a 3d swiss roll distribution for the prior. Throughout the paper, we use Adam optimizer with learning rate 10^{-3} and mini-batches of size 1000 to avoid extensive hyperparameters search. We chose a 1000 as a tradeoff between fast learning and preserving the underlying structure of the data in each batch. We used early-stopping based on the best validation loss to avoid overfitting.

a) Effect of kernel size σ

First, we tried to find the effect of the kernel size σ. In this experiment, we fix λ = 10. We could notice that with higher kernel size, the latents failed to be close to the prior distribution.

As seen in Figure 2, the higher kernel size makes the CSD divergence between input data distribution and the prior distribution to be smaller. With a fixed λ, this smaller order of magnitude in the CSD causes the MSE loss to dominate which leads to the model mainly learning reconstruction with little regard to the prior. This inverse relation between kernel size and CSD can be viewed from the information potential equation. For these reasons, in figure 1 graphs (kernel size = 100, 1000), the model fails to learn the swiss roll distribution and the latent codes form a shape closer to a sphere.

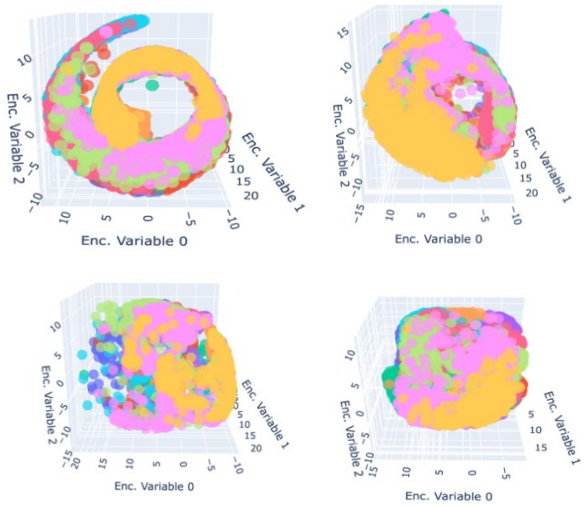
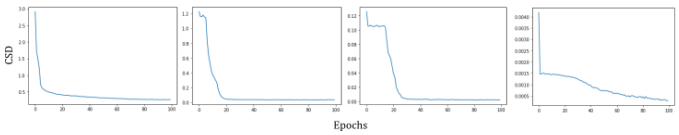


Figure 5. 3dimensional latent space from top left to bottom right σ = 1, σ = 10, σ = 100, σ = 1000 on fixed λ = 10.



b) **Effect of regularization parameter λ**

To find a best λ value, we fixed the kernel size $\sigma = 1$ based on the first experiment. The higher λ spreads out the latent codes on the prior didtribution removing the natural clustering in the dataset. It somehow over-enforces the prior on all the data classes. We achieve best clustering with $\lambda = 0.1$. In figure 8, we implement a linear walk along both directions of the 3d swissroll manifold. The walk shows smooth transitions between digits in the data set.

C. Enforcing a Gaussian prior on the latent codes

To encourage spread in the 3d latent space, we enforce a Gaussian prior on it, $N(\mu, \sigma)$. Gaussians impose a spread on the space centered around a mean. In order to facilitate clustering, we consider a Gaussian Mixture Model with 10 different Gaussians, same (corresponding to the 10 classes). We first run the models with the same hyperparameters from the swiss-roll experiment. We further tuned to find the optimal kernel size $\sigma = 100$ and λ=0.1. We evaluate the clustering performance by applying applied k-means clustering algorithm (see methods). The results are reported in Table 2. Although adding a structure improved clustering accuracy as expected, the gaussian mixture prior did not perform better than a simple Gaussian, but not significantly. Further experiments are needed. Our preliminary interpretation is that some digits in KMNIST look very similar so they are closer in the latent space and forcing the dispersion in the GMM leads to a worse latent coding. Figure 9 shows the 2d projections of the latent codes for the no prior AE, the simple gaussian prior and the GMM. Figures 10, 11 and 12 show confusion matrices for different priors. Confusion matrices and accuracy are calculated after mapping the labels produces by k-means clustering to labels that maximize accuracy.

No prior	Simple Gaussian N(0,1)	Gaussian Mixture Model (10)	Swiss roll
40%	45.19%	44.24%	42%

Table 2: Kmeans clustering classification (unsupervised) accuracy on different regularizers

D. Computational cost of adding a regularization

The CSD involves extra computation which was very slow to compute on a CPU. We used Colab GPU: Tesla P100.

Time	AE (no regularizer)	AE + CSD
Second	83	61

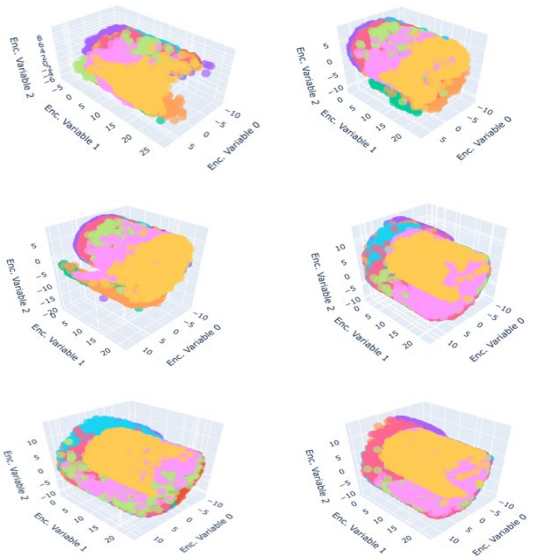


Figure 7:3D latent space for $\lambda = 0.01, 0.05, 0.1, 1, 10, 10$ (from top left to bottom right)

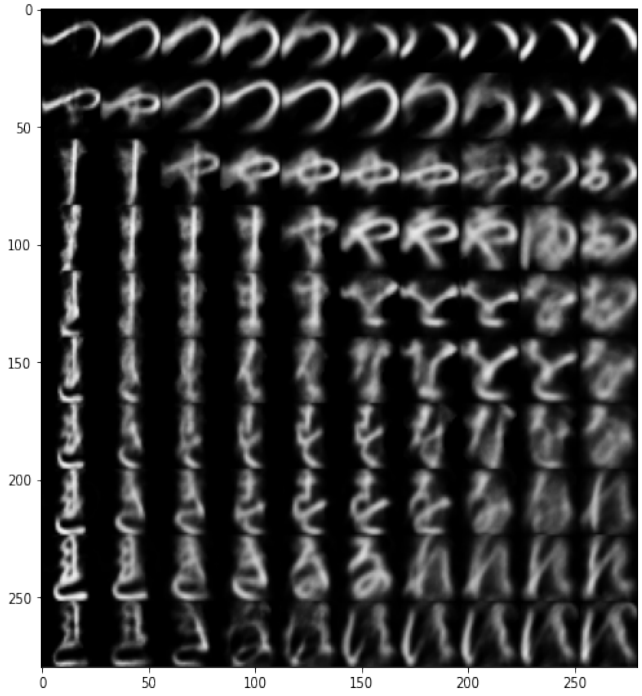


Figure 8: Samples from a linear walk over the swiss-roll. Right to left is going along the linear direction of the manifold. Top to bottom is going along the rotating direction of the manifold.

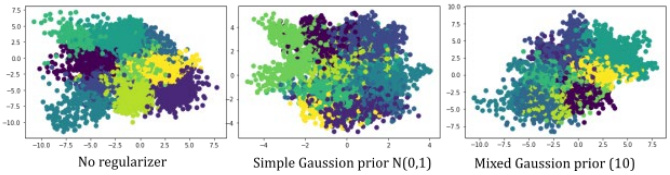


Figure 9: 2D projections of the 3D latent space for different priors. Colors given by k-means clustering (unsupervised).

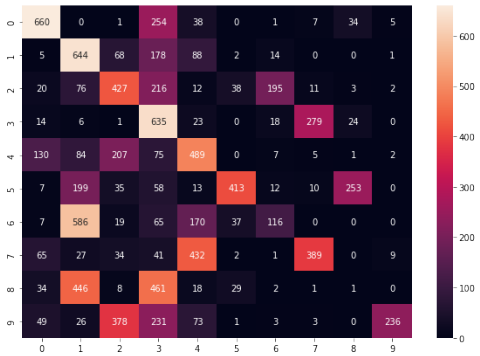


Figure 10 Confusion matrix for k-means clustering (unsupervised) without a prior $\lambda = 0$, accuracy 40%

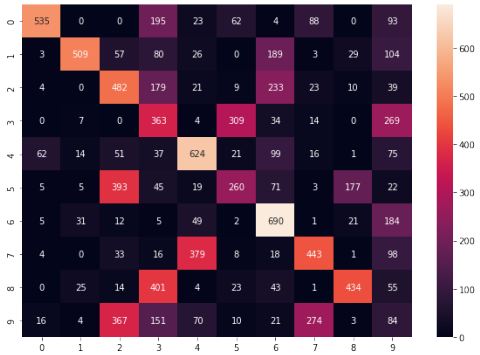


Figure 11: Confusion matrix for k-means clustering (unsupervised) for a simple Gaussian prior N(0,1), accuracy 45.19%

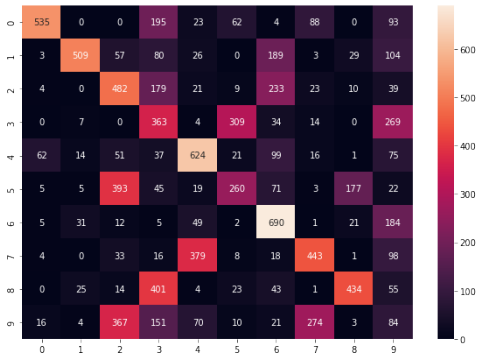


Figure 12: Confusion matrix for k-means clustering (unsupervised) for Mixed Gaussian, accuracy 44.24%

REFERENCES

[1] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*.

[2] J. Principe, N. Euliano and W. Lefebvre, Neural and adaptive systems. New York: Wiley, 2000.

[3] "Understanding of Convolutional Neural Network (CNN) — Deep Learning", Medium, 2021. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed: 24- Oct- 2021].

[4] "GitHub - rois-codh/kmnist: Repository for Kuzushiji-MNIST, Kuzushiji-49, and Kuzushiji-Kanji", GitHub, 2021. [Online]. Available: https://github.com/rois-codh/kmnist. [Accessed: 22- Oct- 2021].

[5] A. Agarap, "Deep Learning using Rectified Linear Units (ReLU)", 2019.

[6] D. Liu, "A Practical Guide to ReLU", Medium, 2021. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. [Accessed: 27- Oct- 2021].

[7] K. Bajaj, D. Singh and M. Ansari, "Autoencoders Based Deep Learner for Image Denoising", *Procedia Computer Science*, vol. 171, pp. 1535-1541, 2020. Available: 10.1016/j.procs.2020.04.164.

[8] E. Santana, M. Emigh and J. C. Principe, "Information Theoretic-Learning auto-encoder," *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 3296-3301, doi: 10.1109/IJCNN.2016.7727620.