**Assignment 4: Reproducing Weakly Supervised Clustering by Exploiting Unique Class Count**

**Question**

Generate results for MNIST dataset for ucc and clustering as in Table 1 in the paper.

Table

Description automatically generated

Plot all your results for analysis and explain them.

Try:

1. Make different feature extractors and distribution regression
2. Use full MNIST data set 60,000 for training, also use subset of MNIST for training and analyse the overfitting effects
   1. 60,000 for training
   2. 20,000 for training (same number of examples per class)
   3. 5,000 for training (same number of examples per class)
   4. 500 for training (same number of examples per class)
3. Devise ways to improve this algorithm.

**Answer**

The code of this work and the trained models are available at

* <http://bit.ly/uniqueclasscount> and
* <https://github.com/COMP6248-Reproducability-Challenge/UCC-Classifier/>

The UCC models have several variants, depending on the weight α of two loss functions as shown below, and whether the autoencoder is included in the model. Below is the loss function where the left is the *ucc* loss calculated by the cross-entropy error, the right is the mean square error (MSE). By modifying the weights of cross-entropy and MSE errors, we can check the performance of each component of the model.

Diagram

Description automatically generated with medium confidence

| Model | AE loss  (Cross entropy) | UCC loss  (MSE) | Bag number |
| --- | --- | --- | --- |
|  | Yes | Yes | 1-4 |
|  | Yes | Yes | 2-4 |
|  | No | Yes | 1-4 |
|  | No | Yes | 2-4 |

Here I trained the models with 2000 epochs, which provides us with insights into the training difficulty and time cost. Below are my Colab Notebook links:

| Model | Colab Notebook link |
| --- | --- |
|  | <https://colab.research.google.com/drive/1HZw5sFDCLuoSRaLAXMLUZhsey_Ob27mV?usp=sharing> |
|  | <https://colab.research.google.com/drive/1DMiCXlVLzleUm8FL36gXqRdStjUHH8Y0?usp=sharing> |
|  | <https://colab.research.google.com/drive/1lkpv3ZN-2wRi2ZKp3SbYJ5YKW2lCZR20?usp=sharing> |
|  | <https://colab.research.google.com/drive/1tzgYVpA2I2YeAD2rNPGa1X-RuhQJJwum?usp=sharing> |

| Dataset size | Colab Notebook link |
| --- | --- |
| 60000 | <https://colab.research.google.com/drive/1HZw5sFDCLuoSRaLAXMLUZhsey_Ob27mV?usp=sharing> |
| 20000 | <https://colab.research.google.com/drive/1opoWz30I1wSLCsij7XhgWhAVyTMJYZ_C?usp=sharing> |
| 5000 | <https://colab.research.google.com/drive/1ts0mkQwlDHA40QhZtxwbybli06Xd-OPB?usp=sharing> |
| 500 | <https://colab.research.google.com/drive/12lUO268vuXtxdfu4-wfobvdaujNlOJNN?usp=sharing> |

In our experiments, no matter how large the dataset is, the training is similarly slow. It takes around 25-30 seconds to run each epoch. Reducing dataset size only slightly decreases the training time per epoch within the range. Thus, there is little possibility to run a full 60000 epochs as described in the paper, since Google Colab has 12-hour runtime limit.

| Model | ucc  accuracy | Clustering accuracy |
| --- | --- | --- |
|  | 0.949 | 0.941 |
|  | 0.937 | 0.931 |
|  | 0.960 | 0.947 |
|  | 0.931 | 0.899 |

With the expansion of dataset, the generalisation ability of the model seems to increase as indicated by reduced loss values, yet the variance of the training loss increase with the epoch number and is larger in larger datasets, as indicated in the plots below and decreased validation accuracy.

| Dataset size = 20000 | Dataset size = 5000 |
| --- | --- |
|  |  |

As for KDE, Appendix C.6 was added as per reviewers’ opinion[[1]](#footnote-0). In the paper, a replacement of averaging layers was used to show the importance of KDE. But I didn’t do this due to a lack of time. Below is the result from the paper.

Table

Description automatically generated

**Devise methods to improve the model**

Firstly, according to the paper, the feature extractor and KDE parameters are determined according to model performance. The layer number of the convolutional architecture was determined as soon as the authors acquired good results.

Similarly, the kernel function of KDE could also be tuned, where there are a number of options that have not yet been tested, such as uniform, triangular, biweight, triweight, Epanechnikov, normal, and others.[[2]](#footnote-1) The original paper only shows that KDE is better than averaging but does not show that the current option is optimal.

Additionally, other layer structures could also be tested in future experiments for optimisation to include more optimal algorithms. As per our experiences in the last assignment, including data augmentation in the training may improve the training results.

1. <https://openreview.net/forum?id=B1xIj3VYvr> [↑](#footnote-ref-0)
2. <https://en.wikipedia.org/wiki/Kernel_density_estimation> [↑](#footnote-ref-1)